



DEVELOPMENT OF RICH INTERNET APPLICATION FOR OFFICE MANAGEMENT SYSTEM

A Dissertation

Submitted In Partial Fulfilment Of
The Requirements For The Degree Of

MASTER OF SCIENCE

In

NETWORK CENTERED COMPUTING,
EBUSINESS, SP06

in the

FACULTY OF SCIENCE
The UNIVERSITY OF READING

by

Omer Dawelbeit, BSc (Hons)

sip05oid

2nd June 2008

Supervisor Ms. Eve-Marie Larsen

Acknowledgement

I would like to thank my supervisor Eve Marie for all her assistance during the project. Also my special thanks go to Nia Alexandrov for her support over the past two years made achieving this master degree possible. I would like to also thank Nellie Round for her help.

I am also grateful to my parents Ibrahim and Amna and their continuous support over the years. This support has given me the energy to carry on and complete this dissertation. Special thanks also go to my brother and sisters.

I would like to thank my wife Amna for her continuous support, patience and constructive feedback in regards to this dissertation. The smiles of my two daughters Laila and Sarah also made the long hours spent on this project bliss.

My appreciation goes to John Rossall my work colleague in Thoughtbreak for kindly offering technical advice and sharing knowledge.

Special thanks also go to Paul Alexander for proof reading this report and providing feedback on the application.

Many thanks to Amna Ahmed, Omer Abu-Bakar, Haleem Abu-Gusiessa and Mohammed Al Haj for their contributions to the survey and for their feedback on the application.

Abstract

This project is concerned with the design and implementation of a cost effective Rich Internet Application for an office management system that can be used to manage the staff in a small business. This dissertation outlines the shortcomings of using traditional Web design methodologies to design functionality-oriented applications. To overcome these shortcomings the project has devised a comprehensive design and implementation methodology to implement a new generation of Web applications called Rich Internet Applications. This design methodology is largely based on two pillars, one is the traditional and established software design principles outlined in the literature such as Bennett et al. [4], Stone. et al [34] and Shneiderman [32] and the other is the new concept of Web 2.0 [24].

The project uses the Unified Software Development Process, relational database theory and the user interface principles to design the application, and then devises a methodology to implement a dynamic, rich user interface. The project implements the application using the Java programming language and other Web technologies and Open Source frameworks such as JavaScript, HTML, CSS, Hibernate, Spring and Struts 2. This report provides the design, implementation and evaluation of the application and clearly demonstrates that the developed Rich Internet Application has delivered better usability, interactivity and performance compared to traditional Web applications.

Table of Contents

Acknowledgement.....	2
Abstract.....	3
Table of Contents.....	4
List of Tables.....	8
List of Figures.....	9
Abbreviations	12
1 Introduction.....	13
1.1 Motivations behind the project.....	13
1.2 Project Aim.....	13
1.3 Project Objectives.....	14
1.4 Organisation of this Dissertation.....	14
2 Technology Background.....	15
2.1 Why Web applications?.....	15
2.2 The Concepts of Web 2.0.....	16
2.2.1 Rich Internet Applications.....	18
2.3 The Web as an OO user interface.....	19
2.4 The use of a dynamic rich user interface.....	21
2.4.1 The use of rich visual widgets:.....	21
2.4.2 Breaking the page model using AJAX.....	23
2.5 The use of lightweight frameworks and established modelling techniques.....	25
2.6 Usability Requirements.....	26
2.6.1 User Interface (UI) Design Principles.....	26
2.6.2 Web pages design principles.....	27
2.6.3 Design rules for the OfficeMA.....	28
2.7 Accessibility.....	29
2.8 Summary.....	30
3 Project Requirements.....	32
3.1 HR Requirements of Small Businesses.....	32
3.1.1 Obstacles facing small businesses.....	33
3.1.2 What is needed and why?.....	33
3.1.3 Alternative applications.....	34
3.2 Software used for the Project.....	36
3.2.1 Java Web Components.....	36
4 Design Methodology.....	37
4.1 Project Lifecycle.....	37
4.2 The Software Development Process.....	38
4.2.1 Requirement Capture and Modelling.....	39
4.2.2 Requirement Analysis.....	40
4.2.3 Class design (Detailed design).....	42
4.2.4 User Interface Design.....	42
4.2.5 Database Design.....	43
4.2.6 Construction, testing and implementation.....	45
5 Preliminary System Design.....	46

5.1 Requirements Gathering.....	46
5.2 Initial System Architecture.....	46
5.3 Requirement Capture and Modelling.....	47
5.3.1 Prototyping the User Interface.....	47
5.3.2 Staff Management Requirements.....	48
5.3.3 Expenses Management Requirements.....	50
5.3.4 Authentication and Authorisation Requirements.....	53
5.3.5 System Settings Management.....	54
5.3.6 Time Booking Requirements.....	56
5.3.7 Holiday Management Requirements.....	58
5.3.8 Task Management Requirements.....	60
5.4 Requirement Analysis.....	61
6 Implementation Strategy.....	62
6.1 System Architecture.....	62
6.2 Database layer.....	63
6.2.1 Choosing a DMBS.....	63
6.2.2 Database Implementation.....	63
6.3 Business Logic Layer.....	64
6.3.1 POJO Architectural Pattern.....	64
6.3.2 Spring Framework and Dependency Injection.....	66
6.3.3 Domain model classes.....	67
6.3.4 Object to relational mapping framework.....	67
6.3.5 Coding practices.....	72
6.3.6 Unit testing the domain model classes.....	72
6.4 Presentation Layer.....	74
6.4.1 Presentation Layer logic and data formatters.....	75
6.4.2 Graphical user interface.....	77
6.5 Overall System.....	82
7 Detailed Software Design	83
7.1 Detailed Class Design and Implementation.....	84
7.1.1 Design and architectural patterns.....	84
7.1.2 Enumerated types.....	85
7.1.3 Dependency Injection using Spring.....	86
7.1.4 The use of Exceptions.....	87
7.1.5 Generics and Parameterized Classes.....	87
7.2 Application Packages.....	88
7.2.1 Company package:.....	89
7.2.2 Expenses package:.....	91
7.2.3 Holiday package:.....	95
7.2.4 Staff package:.....	98
7.2.5 Task package:.....	105
7.2.6 Testing the domain model.....	107
7.3 User Interface Design and Implementation.....	108
7.3.1 Application action classes	109
7.3.2 User Interface Design.....	112
7.3.3 User interface controller.....	116

7.3.4 User interface modelling.....	117
7.4 Database Design and Implementation.....	126
7.4.1 Establishing requirements.....	126
7.4.2 Data Analysis.....	126
7.4.3 Entity Relationship Model.....	128
7.4.4 Normalisation.....	131
7.4.5 Relational Database Model.....	134
7.4.6 Physical Database Model.....	134
7.5 Caching, Pooling and Transactions Support.....	137
7.5.1 Caching.....	137
7.5.2 Connection Pooling.....	137
7.5.3 Transactions.....	138
7.6 Security.....	139
7.6.1 Insecure Communications.....	140
7.6.2 Session Hijacking.....	141
7.6.3 JavaScript Hijacking.....	141
7.6.4 JavaScript Tampering.....	141
7.6.5 SQL Injection, Remote file inclusion and Cross-site scripting.....	142
7.7 Deployment.....	143
7.8 System Testing.....	144
8 Evaluation.....	145
8.1 Satisfaction of business requirements.....	145
8.2 Accessibility.....	145
8.3 Usability.....	146
8.3.1 Visibility, Affordance and Consistency.....	146
8.3.2 Closure, Tolerance and Feedback.....	147
8.3.3 Performance and Data Refresh.....	147
8.4 Evaluation Summary.....	147
9 Conclusions.....	148
9.1 Project Achievements.....	148
9.2 Project Issues.....	149
9.3 Contributions of this Dissertation.....	149
9.3.1 Problems with adapting functional-oriented UI as content-oriented Web UI.....	149
9.3.2 Utilising the Web as a functional user interface.....	150
9.4 Suggestions for Future Work.....	152
9.4.1 Usability and Accessibility of RIA.....	153
9.4.2 Performance of RIA.....	153
9.4.3 Enhancements to the Office Management Application.....	153
10 References.....	154
10.1 Books and Articles.....	154
10.2 Web references	157
11 Appendices.....	161
11.1 Appendix A – Office Management Application’s Modules Survey.....	162
11.2 Appendix B – Documents Sampling.....	163
11.2.1 Sample holiday control spreadsheet.....	163

11.3 Appendix C – Use Case Models.....	164
11.3.1 Add Staff Use Case.....	164
11.3.2 Find Staff Use Case.....	169
11.3.3 View Personal Details / Edit Personal Details.....	172
11.3.4 View brief / complete staff details.....	175
11.3.5 Edit staff details.....	177
11.3.6 Find Expenses.....	180
11.3.7 View, approve, reject and pay Expenses.....	185
11.3.8 Edit Expenses Use Case.....	188
11.3.9 Add New Expenses Use Case.....	191
11.3.10 Login use case.....	196
11.3.11 View/Update system settings.....	198
11.3.12 Update my settings.....	202
11.3.13 View/Add Timesheet.....	204
11.3.14 View timesheet summary.....	207
11.3.15 View Holiday Details use case.....	210
11.3.16 View Holiday Calendar use case.....	213
11.3.17 View / Update Tasks use case.....	215
11.4 Appendix D – Requirement Analysis Models.....	217
11.4.1 Staff management communication diagrams.....	217
11.4.2 Staff management analysis class diagram.....	219
11.4.3 Authentication and Authorisation communication diagram.....	219
11.4.4 Authentication and Authorisation sequence diagram.....	220
11.4.5 Authentication and Authorisation analysis class diagram.....	220
11.4.6 Expenses management communication diagrams.....	221
11.4.7 Expenses management analysis class diagram.....	224
11.4.8 Expenses state diagram.....	225
11.4.9 Holiday management communication diagrams.....	226
11.4.10 Holiday management analysis class diagram.....	228
11.5 Appendix E – Relational Database Model.....	229
11.6 Appendix F – Physical database schema.....	236
11.7 Appendix G – Sample ORM SQL queries.....	247
11.8 Appendix H – Software CD-ROM Contents.....	253
11.9 Appendix I - Software used for the project.....	254
11.10 Appendix J – PostgreSQL database utilities.....	255
11.11 Appendix K – Jude UML CASE tool.....	256
11.12 Appendix L – Google code project.....	257
11.13 Appendix N – Debugging JavaScript and Browser tools.....	259
11.13.1 Firefox Firebug.....	259
11.13.2 Firefox Web Developer Toolbar.....	259
11.13.3 Microsoft Script Debugger for IE.....	260
11.13.4 Microsoft IE Developer Toolbar.....	260
11.14 Appendix O - Project Schedule.....	261
11.14.1 Project Gantt Chart.....	262

List of Tables

Table 1 – A comparison of sample widgets used in desktop and Rich Internet applications.....	22
Table 2 – Similarities between desktop applications and Web capabilities.....	30
Table 3 – Alternative software applications and their issues.....	35
Table 4 – Requirements summary for staff management.....	48
Table 5 – Use cases summary for staff management.....	49
Table 6 – Requirements summary for expenses management.....	50
Table 7 – Use cases summary for expenses management.....	52
Table 8 – Requirements summary for authentication and authorisation.....	53
Table 9 – Use cases summary for authentication and authorisation.....	54
Table 10 – Requirements summary for system settings management.....	54
Table 11 – Use cases summary for system settings.....	56
Table 12 – Requirements summary for time booking.....	56
Table 13 – Use cases summary for time booking.....	57
Table 14 – Requirements summary for holiday management.....	58
Table 15 – Use cases summary for holiday management.....	59
Table 16 – Requirements summary for task management.....	60
Table 17 – Use cases summary for task management.....	61
Table 18 – Some of the features of PostgreSQL vs. MySQL [20].....	63
Table 19 – Crow's feet notation used in the ER- modelling.....	64
Table 20 – ER-Model and Relational Mode to JPA and Hibernate mappings.....	70
Table 21 – Data types for the various models.....	135
Table 22 – Project schedule and deadlines.....	261

List of Figures

Figure 1. Gartner Hype Cycle for 2006 technologies [27].....	17
Figure 2. Comparison between Standalone and Rich Internet Applications.....	18
Figure 3.. The duality of the Web [19].....	19
Figure 4. Traditional server based MVC Web applications [21].....	24
Figure 5. RIA MVC Web applications [21].....	24
Figure 6. Traditional waterfall lifecycle model.....	37
Figure 7. Activities that lead to software deployment, adapted from Grand [21]..	39
Figure 8. Sample Model of Database Development [23].....	44
Figure 9. Initial package architecture for OfficeMA.....	47
Figure 10. Use cases for staff management.....	49
Figure 11. Expenses management use cases.....	51
Figure 12. Login use case diagram.....	53
Figure 13. System settings management use cases diagram.....	55
Figure 14. Time booking use cases diagram.....	57
Figure 15. Holiday management use cases.....	59
Figure 16. Task management use cases.....	60
Figure 17. Layered architecture for a typical Web application.....	62
Figure 18. Part of detailed class model showing entities, a repository and a service.....	65
Figure 19. Business logic layer showing domain model and ORM layer.....	68
Figure 20. JUnit tests executed from within the Eclipse IDE.....	73
Figure 21. Office Management Application Presentation layer.....	75
Figure 22. Using JavaScript eval function on a JSON string to create a JavaScript Object.....	76
Figure 23. View Staff Details dialogue widget.....	79
Figure 24. JavaScript classes declarations in Dojo.....	80
Figure 25. Boundary class diagram for ViewStaff widget.....	80
Figure 26. Sequence diagram for the ViewStaff widget.....	81
Figure 27. OfficeMA candidate technologies. Adapted from Richardson [30].....	82
Figure 28. Package structure for OfficeMA detailed classes.....	84
Figure 29. Data Access Object Class Diagram [30].....	85
Figure 30. Spring beans schematic for OfficeMA classes.....	86
Figure 31. Generic repository super class and interface for CRUD operation....	88
Figure 32. WorkStream, Project and repository classes.....	90
Figure 33. Grade and GradeRepository classes.....	90
Figure 34. Spring beans schematic for ExpensesManagementServiceImpl.....	92
Figure 35. Methods defined by the ExpensesManagementServiceImpl.....	92
Figure 36. Expenses classes and dependencies (Generic repository classes omitted for clarity).....	94
Figure 37. Spring beans schematic for HolidaysManagementServiceImpl.....	95
Figure 38. Methods defined by the HolidaysManagementServiceImpl.....	96
Figure 39. Holidays classes and dependencies (Generic repository classes omitted for clarity).....	97

Figure 40. Role classes and dependencies (Generic repository classes omitted for clarity).....	99
Figure 41. Spring beans schematic for StaffManagementServiceImpl.....	100
Figure 42. Methods defined by the StaffManagementServiceImpl.....	100
Figure 43. Sequence diagram for the createStaffMember.....	101
Figure 44. Sequence diagram for the authenticate.....	102
Figure 45. Staff classes and dependencies (Generic repository classes omitted for clarity).....	104
Figure 46. Spring beans schematic for TaskManagementServiceImpl.....	106
Figure 47. Methods defined by the TaskManagementServiceImpl.....	106
Figure 48. Task classes and dependencies.....	107
Figure 49. Struts 2 request flow [44].....	108
Figure 50. Package structure for presentation layer.....	109
Figure 51. Office Management Application desktop.....	113
Figure 52. OfficeMA menus and toolbar.....	113
Figure 53. An animated image to indicate the application is loading data.....	114
Figure 54. Message dialogues in the OfficeMA.....	114
Figure 55. Add staff window.....	115
Figure 56. Find staff window.....	116
Figure 57. Class diagram for the client side JavaScript controller.....	117
Figure 58. Boundary class diagram for staff management VSOs.....	118
Figure 59. Edit staff details sequence diagram.....	119
Figure 60. View staff details sequence diagram.....	119
Figure 61. Add staff details sequence diagram.....	120
Figure 62. Find staff sequence diagram.....	120
Figure 63. Boundary class diagram for expenses management VSOs.....	122
Figure 64. Add new expenses item sequence diagram.....	123
Figure 65. Boundary class diagram for task management VSOs.....	124
Figure 66. View Tasks sequence diagram.....	125
Figure 67. Delete/Update Task sequence diagram.....	125
Figure 68. The Office Management Application preliminary E-R diagram.....	129
Figure 69. EHCACHE configurations.....	137
Figure 70. DBCP connection pooling configurations.....	138
Figure 71. Securing OfficeMA application with Apache Web server.....	140
Figure 72. Deployment diagram for OfficeMA.....	143
Figure 73. Add staff use case diagram.....	164
Figure 74. Find staff use case diagram.....	169
Figure 75. View/Edit personal details use case diagram.....	172
Figure 76. View brief/complete staff details use case diagram.....	175
Figure 77. Edit staff use case diagram.....	177
Figure 78. Find expense use case diagram.....	180
Figure 79. View, approve, reject and pay expenses use Case diagram.....	185
Figure 80. Add new expenses use case diagram.....	191
Figure 81. Add staff communication diagram.....	217
Figure 82. Edit Staff communication diagram.....	218
Figure 83. View Staff Details.....	218

Figure 84. Staff management analysis class diagram.....	219
Figure 85. Login communication diagram.....	219
Figure 86. Authentication and Authorisation sequence diagram.....	220
Figure 87. Authentication and Authorisation analysis class diagram.....	221
Figure 88. Add/Edit expenses communication diagram.....	221
Figure 89. Find Expenses communication diagram.....	222
Figure 90. View Expenses communication diagram.....	223
Figure 91. Expenses management analysis class diagrams.....	224
Figure 92. Expenses management analysis class diagrams.....	225
Figure 93. Approve/Cancel holiday communication diagram.....	226
Figure 94. Request/Cancel holiday communication diagram.....	226
Figure 95. View holiday calendar communication diagram.....	227
Figure 96. View Holiday details communication diagram.....	227
Figure 97. Holiday management analysis class diagram.....	228

Abbreviations

ACID	Atomicity, Consistency, Isolation, Durability
BCNF	Boyce-Codd Normal Form
CIO	Chief Information Officer
CRC	Collaboration – Responsibility Card
CRUD	Creating, Retrieving, Updating and Deleting
CSS	Cascaded Style Sheets
DAO	Data Access Object
DBMS	Database Management System
DDL	Data Definition Language
DHTML	Dynamic HTML
DI	Dependency Injection
DML	Data Manipulation Language
DOM	Document Object Model
DTI	Department of Trade & Industry
EJB	Enterprise Java Beans
ER	Entity-Relation
GUI	Graphical User Interface
HR	Human Resources
HTML	Hypertext Markup Language
HTTP	Hyper Text Transfer Protocol
IDE	Integrated Development Environment
IIS	Internet Information Services Server from Microsoft
IT	Information Technology
J2EE	Java 2 Enterprise Edition
JDBC	Java Database Connectivity
JPA	Java Persistence API
JSON	JavaScript Object Notation
JSP	JavaServer Pages from Sun Microsystems
JVM	Java Virtual Machine
MDI	Multiple Document Interface
MVC	Model-View-Controller
OfficeMA	Office Management Application
OO	Object Oriented
ORM	Object Relational Mapping
OS	Operating System
OWASP	Open Web Application Security Project
POJO	Plain Old Java Object
RIA	Rich Internet Applications
TLC	Traditional Waterfall Lifecycle
UI	User Interface
UML	Unified Modeling Language
USDP	Unified Software Development Process
VSO	View Support Object
W3C	World Wide Web Consortium
WUI	Web User Interface
XML	Extensible Markup Language

1 Introduction

1.1 Motivations behind the project

Statistics for 2006 published by the DTI (Department of Trade & Industry) [36] shows that out of 4.5 million businesses in the UK, 99.3% were small firms with fewer than 50 employees. To support the running of their day to day activities and staff management most of these businesses rely on manual processes such as paper forms or spreadsheets. This is due to the fact that these businesses usually do not have the budget to hire IT consultants to develop bespoke applications or the budget to buy out-of-the box software applications that bears a high price tag on licensing.

Web applications can provide a solution and automate these day to day activities, and combined with Web 2.0 features these Web applications can be easy to use, flexible, interactive and cost effective. Such applications can enhance the productivity and enable those businesses to have better control over their operation, keep their mobile workers and customers closer.

The need for the work on this project has risen from the requirements of a small business to be able to automate and manage day to day activities such as expenses, holidays, time booking and electronically store employee data rather than relying on paper and spreadsheets to manage these activities. Based on the large number of small business today the project tried to design and implement a Web application that at least satisfies a generic set of requirements of managing the staff in a small office and making this application configurable where possible.

Most of the small businesses surveyed complained from the fact that these manual processes are time consuming and welcomed the idea of a cost effective Web application which is secure and has access roles so that the administrators are in control and the normal staff roles are restricted.

1.2 Project Aim

The aim of this project is to design and implement a cost effective and configurable Office Management Application as a Rich Internet Application that incorporates at least the staff and expenses human resources areas with the ability to extend the application in the future to cover the other areas such as holidays and time booking.

1.3 Project Objectives

The objectives of this project are summarised as follows:

- Investigate the possibility of applying the USDP boundary classes methodology used to design traditional desktop application in designing and modelling Rich Internet Application user interface. And Devise a methodology that can be used to design similar applications
- Gather and analyse the business requirements for the Office Management Application using the USDP and provide the analysis and detailed design UML models.
- Identify the data requirements for the application and use the relation database design theory to provide the Entity-Relations, relational and physical database models for the application.
- Implement using Java, test and evaluate the application using Open Source technologies and provide the source code, the binaries and the user documentations for the application.

1.4 Organisation of this Dissertation

Section (2) of this report provides a technology background on the methodologies adopted for this project with the main focus on Web applications and the concepts of Web 2.0.

Section (3) on the other hand summarises the project requirements in term of business needs and motivations. This section also briefly evaluates the alternative applications.

Section (4) summarises the design methodology used which is based on the Unified Software Development Process, the user interface design principles and the relation database theory. This section also outlines the deliverables expected from each design step.

Sections (5, 6 and 7) outline the preliminary system design, the implementation strategy and the detailed design respectively. UML diagrams are produced as part of sections 5 and 7, whilst section 6 concentrates on devising the implementation strategy and putting together a new methodology that can be used to implement rich HTML user interfaces.

Sections (7 and 8) provide an evaluation of the application developed and conclusion of the achievement advantages and disadvantages of the new methodology.

2 Technology Background

The work on this project relied upon a number of unconnected concepts in software design and advances in the Web technology. The project tried to link and tie these together and fills the gaps by introducing new concepts and terminology in order to devise a methodology that can be used to effectively design and implement Web 2.0 applications. This methodology is based these principles [24]:

- Applying the concepts of Web 2.0
- Treating the Web as an object oriented user interface, rather than a hypertext medium.
- The use of lightweight dynamic user interfaces that are rich, interactive and responsive, rather than using static Web pages.
- The use of lightweight frameworks and established modelling techniques.

This section provides an overview of the background material available in regards to the four principles mentioned above and the conclusions made. However, before considering these four principles the main drive behind adopting Web applications as a software interface is considered below.

2.1 *Why Web applications?*

A strong argument was presented in most of the referenced literature for using Web applications, for example Rajagopalan et al. [28] has summarised the advantages of Web application as follows:

- Web as an effective medium for information delivery
- The ability to remote access the application
- Platform independent user agent
- Reduced development efforts and cost
- Centralised maintenance efforts and instant upgrade. Also interface changes are centralised

Added to the advantages above is the fact that Web applications have been in existence for quite some time now, which has lead to the development of many

frameworks and design architectures that can be reused when moving on to Web 2.0 technologies [24]. Having said this, traditional Web applications do have some drawbacks such as performance for example, but this project tries to overcome these classical problems by trying out the new Web 2.0 technologies and has come to the conclusion that the advantages of using Web applications outweigh the disadvantages.

The strength of Web application emerged from the ability to integrate a range of backend systems including databases. Elmasri and Navathe [13] has provided a model for providing access to databases on the World Wide Web, although it is outdated in terms of technology, it still represents the same architecture currently used by Web application. Databases on the Internet are now a very popular option for all Web applications. Elmasri and Navathe [13] and Connolly and Begg [8] also explain the relational database design and implementation in great detail.

2.2 The Concepts of Web 2.0

The Web started as a hypertext medium for publishing documents and traditional Web design was all about information architecture and navigational design. As Web technologies evolved, the Web became more interactive in the fact that users could submit and query information dynamically rather than viewing static pages. With the ability to process information, display a rich user interface and interact with the server without the need to submit a whole page through the use of AJAX, the next generation of the Web has emerged; referred to as Web 2.0.

Web 2.0 is being referred to as a platform for deploying software applications [9, 10] and this platform can be used to deploy rich and usable applications on the Web. This approach for developing and deploying software applications has many advantages such as cost reduction, flexibility and usability. However, the problem is that there is much hype and jargon surrounding Web 2.0 technologies with no clear methodology that can be followed to design and implement Web 2.0 applications. This hype is documented by Gartner Research in their Hype Cycle diagram for 2006 [27] (Figure 1) that Web 2.0 technologies which were at the “Peak of Inflated Expectations” should be in mainstream use in less than two years. Two years on, it is now apparent that Web applications are moving towards Web 2.0 technologies as demonstrated by many Web applications today, such as Google Apps.

It is also evident that it is not just big software organisations that want to move towards Web 2.0, but also other large organisations as well. A survey by Forrester Research revealed that 70% of Fortune 2000 Chief Information Officers (CIOs) want to standardize on deploying applications to a Web browser [66]. However, of those surveyed, more than half stated that the limits of HTML prevented them from reaching this objective. But, now with the emergence of

new Web technologies such as AJAX [18] the road is paved for organisations to deploy rich and usable applications on the Web. Hence the Leading Edge Forum [12] has recommended that businesses should adopt Web 2.0 technologies and “move towards using Web browsers and rich internet applications as the standard interface to most applications”.

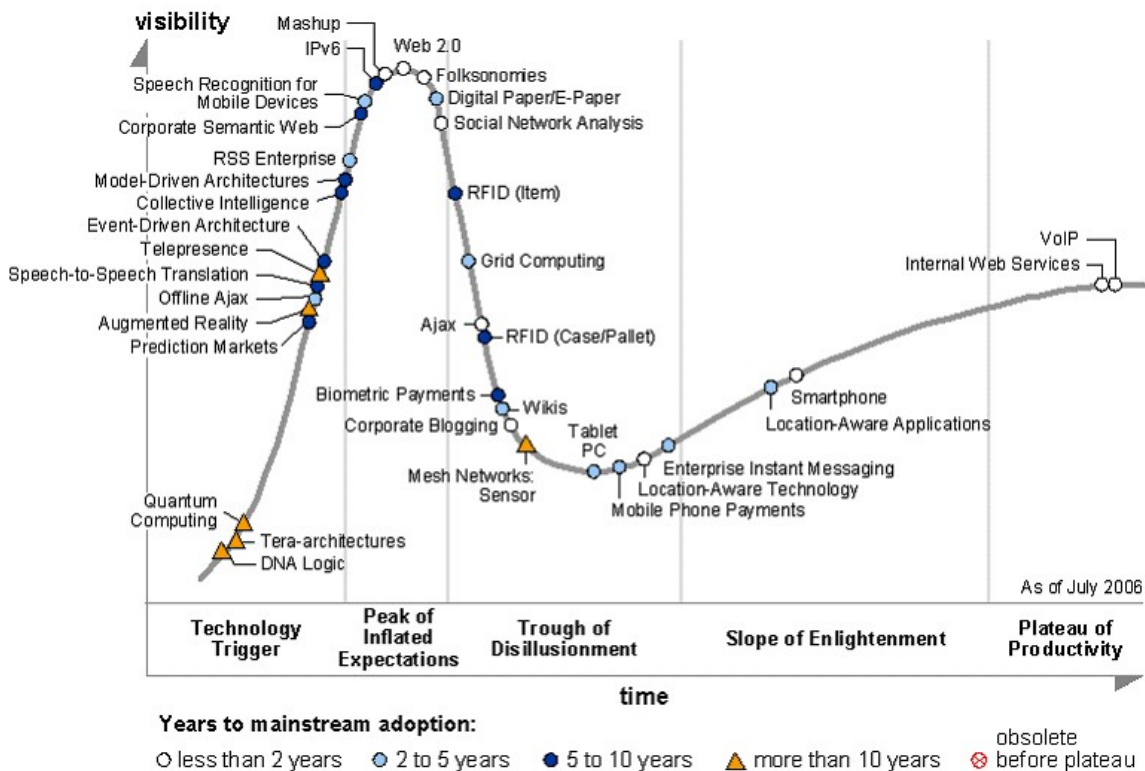


Figure 1. Gartner Hype Cycle for 2006 technologies [27].

From the argument presented above, it's clear that the Web as a platform is now ready for providing a front-end interface for most of the software applications. Nowadays Web 2.0 technologies are regularly talked about and some also believe it holds the answer to HTML limitations. However, a great deal of such hype is over inflated expectations. In this project the author believes that Web 2.0 technologies can deliver usable and effective solutions nevertheless will need careful design and implementation to leverage the best features of these emerging technologies.

2.2.1 Rich Internet Applications

Rich Internet Applications (RIA) on the other hand are Web 2.0 applications that enhance the user interface by incorporating some or all the features and functionality of traditional desktop applications. The similarities between RIAs and desktop applications are demonstrated by Figure 2 below. These similarities are based on how the user interacts with the two user interfaces and how they communicate with the application layer. Both applications still need to use a database server; the desktop client is configured in client-server architecture and accesses the database directly on the other hand the Web application uses the application server for business logic and as a proxy to access the database server. In both cases the user Interface logic and manipulations is performed on the client machine.

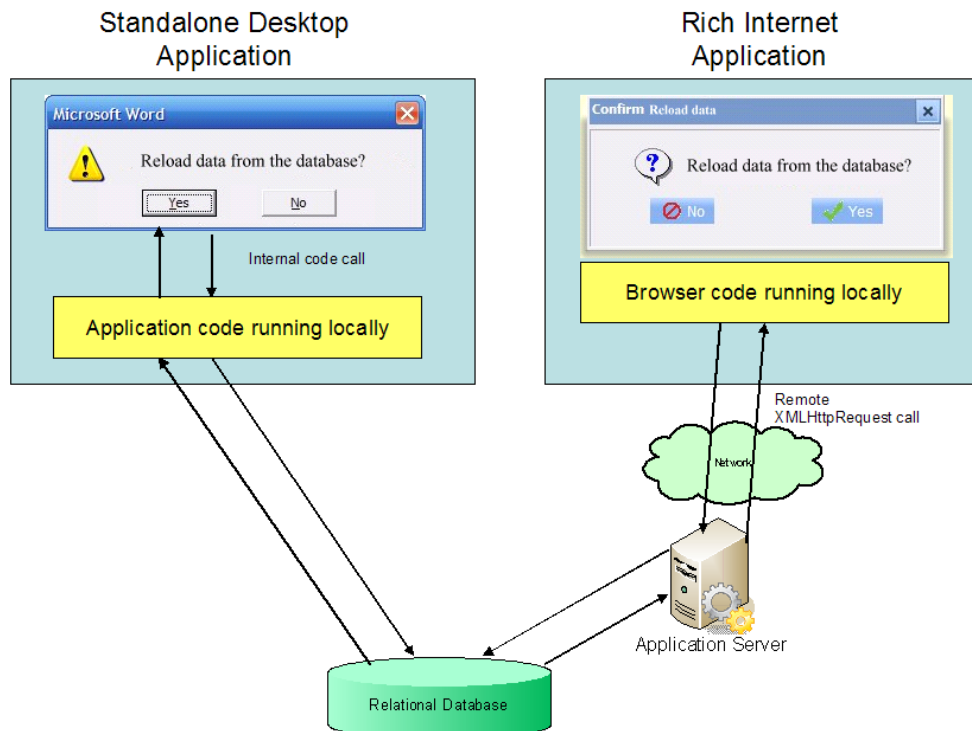


Figure 2. Comparison between Standalone and Rich Internet Applications

2.3 The Web as an OO user interface

Many user interface design textbooks, such as: Stone. et al [34] and Shneiderman [32] conceived the Web as a hyper-textual information space, but with emergence of Web 2.0 technologies and the ability of the Web to be used as a remote software interface the Web, it became clear that the Web now has a dual nature [19]. This dual nature has caused confusion for practitioners as Garrett [19] states:

“.. user experience practitioners have attempted to adapt their terminology to cases beyond the scope of its original application”

Garrett [19] has tried to introduce new terminology to bridge the gap between the two natures of the Web, and provides a good starting point for user interface design of Web software applications. The author has drawn a clear distinction between the use of the Web as a hypertext medium and using the Web as a user interface.

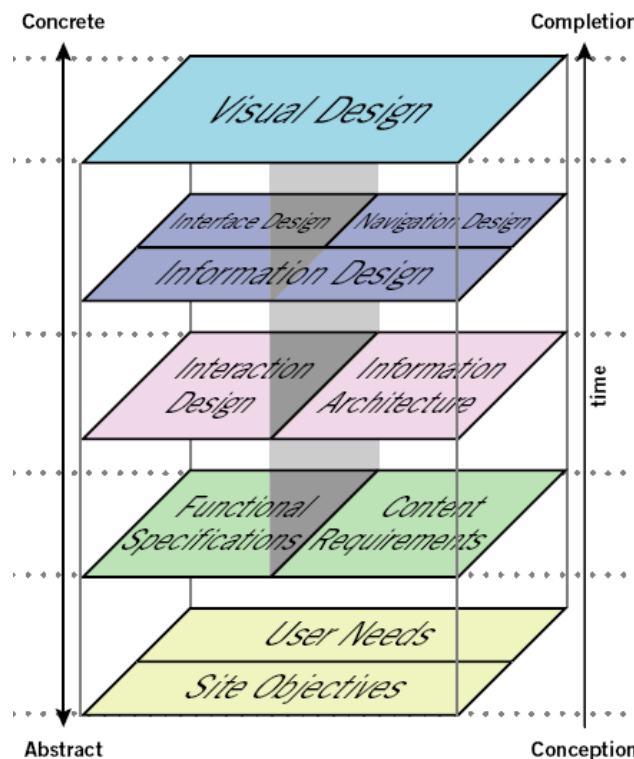


Figure 3.. The duality of the Web [19].

In his model of the Element of User Experience (Figure 3), the author clearly identifies the activities that need to be performed at each plane for the two natures of the Web. This project has treated the Web as a software interface and made the link between the steps highlighted by Garrett [19] and the design methodologies that are used to develop the user interface for traditional desktop applications. These methodologies are not widely used for Web applications, although the process used is deeply rooted in the Unified Software Development Process (USDP) and was used for a long time to model user interfaces for desktop applications developed in languages such as Java and Visual Basic.

Having said that, the fact that the majority of available frameworks for RIA development are Object Oriented such as JavaScript, VBScript, etc..., this enables the Web user interface to be treated as an Object Oriented user interface. At the minimum this can be achieved by the using JavaScript in conjunction with the Document Object Model (DOM), which is the approach that this project has followed. It was obvious that if the Web user interface is treated as OO user interface the same methodology used to design the user interfaces for desktop applications could be used to design the user interface for RIA.

The project has successfully utilised and used these design methodologies that were largely based on the principles outlined by Bennett et al. [4] in regards to designing user interfaces for traditional desktop applications. These can be summarised as follow:

- Prototyping the user interface.
- Designing the boundary classes.
- Modelling the interaction involved in the interface using interaction or communication diagrams.
- Modelling the control of the interface using state machines for complex UI components.

2.4 The use of a dynamic rich user interface

The user interface used for the Office Management Application (OfficeMA) is based on Dynamic HTML for contents, Cascaded Style Sheets for presentation and JavaScript to tie both together. The interaction with the server is achieved through the XMLHttpRequest rather than using the traditional way of clicking on links or submitting a whole page as a HTML Form. To achieve features similar to desktop applications (thick clients) the user interface has adopted the following principles:

- The use of rich visual widgets
- Breaking the page model using AJAX

2.4.1 The use of rich visual widgets:

Visual widgets are components that make up the User Interface and are used to trap user's actions and fire events based on those actions. Visual widgets fulfil a number of interaction styles, which are summarised by Shneiderman [33] as follows:

- Direct Manipulation and Virtual Environments
- Menu Selection, Form Filling, and Dialog Boxes
- Command and Natural Language

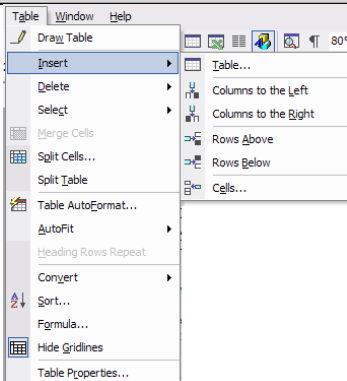
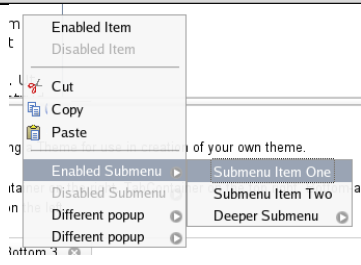
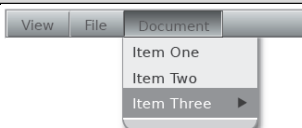
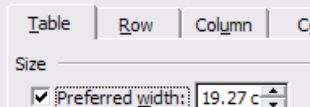
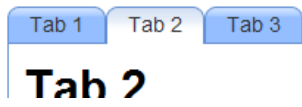
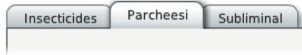
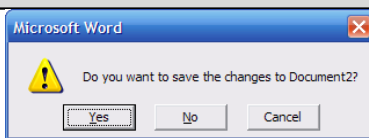
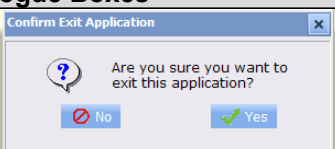
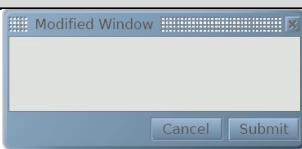
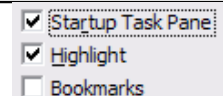
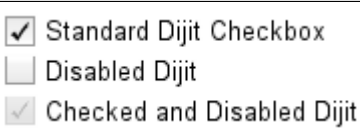
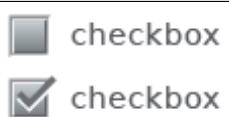
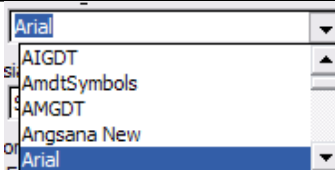

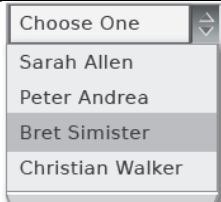
The OfficeMA adopts Menu Selection, Form filling and dialog boxes as the way users will interact with the Application. The advantage of using these interaction styles is the fact these styles come as a natural choice for computer users using and familiar with graphical operating systems such as Windows, this ensures that users can find the application easy to use and can accustom themselves fairly quickly with the controls.

Widgets available to Web applications have evolved considerably and acquired features and properties that outweigh their desktop counterpart. The working of these Web widgets is similar to the one followed by desktop widgets. The user's actions are trapped and used to fire event, which in the case of desktop applications will be based on the operating system events API. On the other hand Web application widgets use the DOM event model to fire events based on the user's actions.

Table 1, below demonstrates the fact that with advances in the Web technologies such as DHTML and Flash plug-in, Web widgets visually resemble the widgets available for traditional desktop applications. The two toolkits considered below

are the Dojo toolkit [42], which is based on DHTML and the OpenLaszlo toolkit which offers Flash as the first option and DHTML as second option. The desktop applications widgets are generally made available by the Operating System (OS) visual library API such as Windows API or through the Java Virtual Machine (JVM) in the case of Java Swing applications.

Table 1 – A comparison of sample widgets used in desktop and Rich Internet applications

Interaction Style	Desktop Client – Based on Windows API	Rich Internet Application	
		Dojo Toolkit	Open Laszlo Toolkit
Menu Selection			
Pull Down Menus			
Tabbed Menus			
Dialogue Boxes			
Dialogue boxes			
Form Elements			
Checkboxes			
Dropdown lists			

2.4.2 Breaking the page model using AJAX

To be able to create a web user interface that is rich and dynamic, the traditional page model will need to be broken and the user will need to be able to submit data to the server without submitting a whole page. The ability to do this will require the use of some logic and control on the client side. This approach can better be explained in terms of the Model-View-Controller (MVC) architectural pattern largely used in Web applications. Figure 4, below shows a traditional MVC Web application with the three components of the architecture based on the server side.

The methodology used in this project is to transfer the view component to the browser on the client machine (Figure 5). This component in itself then follows an MVC pattern on the client side [21]. The author refers to such applications as a third generation Web applications as explained below:

- First generation Web applications transferred HTML mark-up between the client and Web server.
- Second generation Web applications transferred HTML mark-up, but also made use of AJAX technologies to transfer data only.
- Third generation Web applications transferred the HTML mark-up to the client once. The HTML source is built on the client side using JavaScript then only AJAX is used to transfer data thereafter, completely breaking the page model.

The third generation Web applications described above refer to RIA and more specifically to AJAX applications. AJAX applications are a type of RIA that uses XMLHttpRequest browser object as their mechanism of calling the server and breaking the traditional Web pages model. Crane et al. [10] have discussed AJAX applications in detail and provided four principles that can be used to design such applications as summarised below:

1. The browser hosts an application, not content
2. The server delivers data, not content
3. User interaction with the application can be fluid and continuous
4. This is real coding and requires discipline

Breaking the page model enables the use of a rich, interactive and dynamic user interface that only relies on the server for data rather than contents. This provides the ability to use a Multiple Document Interface (MDI) where the user can open and tile multiple windows to aide multitasking and the ability of have different views at one time. This is a feature that is not available in traditional HTML page,

where the user is tied to the page and needs to navigate away in order to view information on different pages.

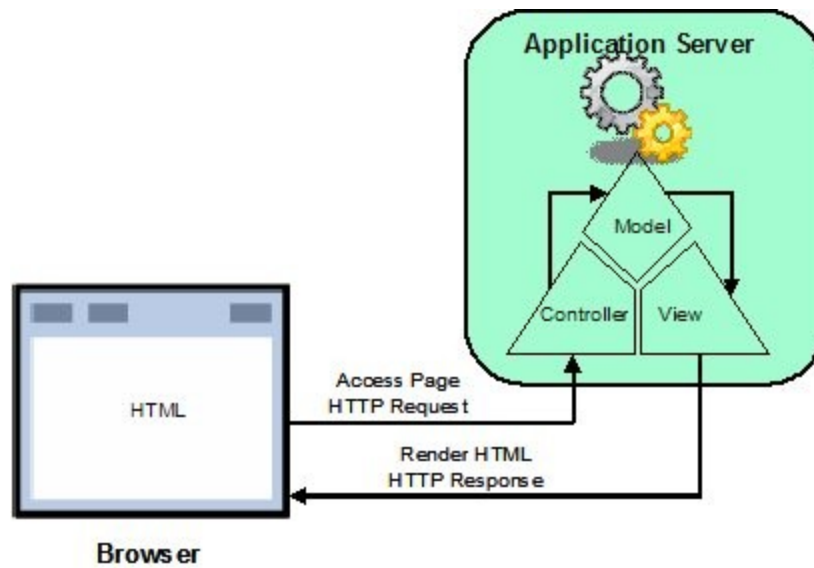


Figure 4. Traditional server based MVC Web applications [21].

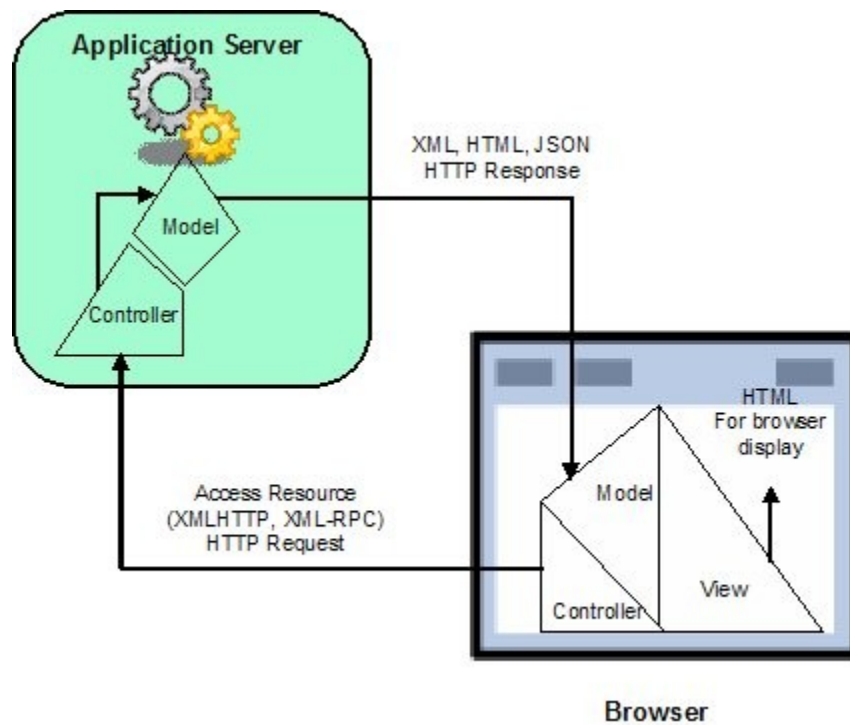


Figure 5. RIA MVC Web applications [21].

2.5 The use of lightweight frameworks and established modelling techniques

The heart of the OfficeMA is the domain model which is the realisation of the use cases developed during the requirement analysis. The project followed the USDP to design an object oriented domain model using UML. This model is use-case-driven as it started off by realising the use cases; this makes it possible to trace the requirements through to their final implementation and hence the ability to test the final application using the use cases. Designing a domain model simplifies the development process and makes it easy to use some of the available Open Source lightweight frameworks to implement the application such as Hibernate and Spring frameworks.

A properly designed domain model that encapsulates all the business logic makes it possible for the user interface to directly represent and manipulate the domain objects. As Pawson [25] has outlined in his Naked Objects design pattern, that for a domain model designed to OO principles the user interface can be automatically generated. This reinforces the fact that domain model is the heart of object oriented software applications and the time well spent during the analysis and design stages saves a great deal of time during implementation.

One of the key areas when designing object oriented applications is the ability to persist the data from objects to database tables. Baur and King [2] present a great deal of Object to Relational Mapping (ORM) concepts. Although the main subject of the book is the popular Hibernate ORM framework, many of the concepts and principles discussed can be applied to address common issues with the conversion from objects to relational database tables.

The ability to effectively map Object into relational database tables also relies on the fact that those Object are implemented as JavaBeans. JavaBeans are simple Java Objects that encapsulate a number of instance variables and provide getters and setters to update their values. These instance variables are usually what is mapped to be persisted in the database. An architectural design pattern that is widely used in the Java world is the POJOs (Plain Old Java Objects) [30], which is an extension to the JavaBeans concept as it adds the concept of separation of concerns in the fact that the POJO is only concerned about its own business logic and does not know about how the data is persisted. Database persistence in the POJOs pattern is done using the Data Access Object (DAO) design pattern [30].

The object oriented analysis and design principles used in this project are largely based on the concepts outlined by Bennett et al. [4]. The authors present a great deal of methodology in the object oriented design using UML, which is largely based on the USDP. The authors have discussed methods such as use case realisation is discussed in detail. On the other hand the popular Domain Analysis

method also termed the Collaboration – Responsibility Card (CRC) is only briefly mentioned. The domain analysis methodology is considered in more detail by Arrington and Rayhan [1]. Although there are differences between the design principles used by Bennett et al. [4] and Arrington and Rayhan [1], they complement each other in terms of producing an analysis and detailed class diagrams. Arrington and Rayhan [1] also consider the implementation using the Java technology which presented some foundations for the methodologies used in this project.

2.6 Usability Requirements

In addition to the OfficeMA functional and technical requirements there were also some usability requirements which directly related to the client requirements that the application should be easy to use, responsive, interactive and user friendly. These requirements are generic and not specific which makes it hard for the designer to be able to translate and apply them to the final product, for this reason a number of established user interface design principles were followed. Some of the usability requirements such as performance and data refresh automatically emerges from the fact that the application will be designed as a RIA.

2.6.1 User Interface (UI) Design Principles

The OfficeMA user interface was designed by using and adapting the user interface design principles outlined by Shneiderman and Plaisant [33] and Stone et al. [34]. The user interface design field is a large area and considering that the OfficeMA is a hybrid of web and desktop applications it was crucial to devise a number of UI design principles and rules that can be used to design the user interface.

Shneiderman and Plaisant [33] outlined the activities required to design and implement an interactive user interface as follow:

- Determine users' skill levels
- Identify the tasks
- Choose an interaction style
- Use of the eight golden rules of interface design.

The authors refer to these eight golden rules as follows:

1. Strive for consistency
2. Cater for universal usability
3. Offer informative feedback
4. Design dialogs to yield closure
5. Prevent errors
6. Permit easy reversal of actions
7. Support internal locus of control
8. Reduce short-term memory load.

Stone et al. [34] on the other hand specify a list of design principles and highlight the fact that these principles are usually abstract and difficult to apply due to their generality. To help make these principles easier to apply the authors indicate that these principles can be translated into design rules, which are low level and highly specific instructions to the designer. The authors then list a number of design principles as follows:

- Visibility - It should be obvious what a control is used for
- Affordance – It should be obvious how a control is used
- Feedback – It should be obvious when a control has been used.
- Simplicity – Keeping the UI as simple as possible
- Structure – Organising the UI in a meaningful and useful way.
- Consistency – Uniformity in appearance, placement and behaviour with the UI
- Tolerance – Preventing the users from making errors

2.6.2 Web pages design principles

Design principles for Web sites are based around the mnemonic HOME-RUN [34] which stands for **H**igh quality contents, **O**ften updated, **M**inimal download time, **E**ase of use, **R**elevant to user's needs, **U**nique to the online medium and **N**et-centric corporate culture. These principles are more applicable to Web pages than functionality-oriented Web applications that are usually restricted to a limited group of people who use it on regular bases. Having said this, the fact that Web applications utilise the Web as their platform some of the above design principles will also need to be incorporated in Web application and in particular RIA.

Even though RIA are designed to behave like desktop application there is a fundamental difference as shown in Figure 2, RIA rely on the server processing for manipulating the data before sending it to the client side for display. This adds an extra layer of processing and potentially a delay and data inconsistency between the client and the server. Hence two of the design principles mentioned above for Web sites will also need to be applied to Web applications in general such as Often updated and Minimal download time principles. These address the performance and data refresh issues in Web applications. Other principles such as High quality contents, Unique to online medium and Net-centric corporate culture are not applicable to the application designed in this project.

2.6.3 Design rules for the OfficeMA

For the purpose of the OfficeMA the following list of design rules have been short listed. This list accommodates principles from both Shneiderman and Plaisant [33] and Stone et al. [34]:

- **Visibility and Affordance** – The user interface will make it clear what controls are used for and how they are used to adopting and using standard control that most graphical operating system users are familiar with.
- **Feedback** – The user interface will provide feedback to user's actions in the form of messages or animated image when data is being retrieved from the server
- **Simplicity and Structure** – The user interface will be design to be simple and to follow an MDI user interface with menu, tool and status bars. This should be a familiar structure to normal desktop users. Complex and long forms will be organised into tabs or trees using the tab or the tree widget. The application will also avoid completely the use of horizontal scrolling by offering the ability to maximize the windows on which the information is displayed. Vertical scrolling in only used where necessary.
- **Consistency** – The user interface will be made consistent and the same approach followed throughout the application.
- **Tolerance** – The user interface will be designed to prevent users from making errors by performing interactive validation to user's input which indicates to the user immediately if they provided an invalid value for an input. The user interface will degrade gracefully by providing error messages and enabling the user to retry the action. The user interface will enable the user to cancel any action they have started before completing it.
- **Closure** – Dialogue boxes and windows will be designed in a way it is clear to the user which dialogue is used for viewing or updating

information. It is also made clear when input is required by the user and when the action has been completed successfully. Feedback should be given to the user after the user submits their updates and current data in the application should be refreshed accordingly.

- **Performance and Data Refresh** – Since the data is not managed locally and application relies on the server to retrieve this data, performance is added as one of the usability rules for the user interface. UI widgets should be rendered in acceptable time frames. The same applies to data retrieval from the server for this reason the server should use caching and other mechanisms to ensure timely responses. The system will also provide the ability for the user to refresh the data where possible or provide automated data refresh updates whenever a window is opened.

2.7 Accessibility

Accessibility on the other hand is to ensure that the system developed is accessible to people with disabilities. Some authors go a step further and define accessibility as making the use of the system easier for all users. The W3C provides a number of Web content accessibility guidelines and defines these under the term universal access, which it describes as follows [34]:

“To make the Web accessible to all by promoting technologies that take into account the vast differences in culture, language, education, ability, material resources, access devices, and physical limitations of users on all content”

It is clear from the description above that the emphasis is to provide accessibility for all rather than just for people with disabilities. For an application such the OfficeMA the scope of the accessibility is even smaller than what is described by the W3C above. This is mainly due to the fact that the application is limited in its use to a number of staff who are computer literate and are constraint in their resources to the ones identified by their employer such as which browser or operating system they use. The need to consider some of the Web accessibility rise from fact that the application is Web based and platform independent so that the users can try it in any operating system with a browser, an issue that does not affect desktop applications because these usually only run in specified OS.

Due to the time and budget constraints of this project accessibility was considered in terms of the ability of the application to run in a specified environment with minimum specifications. Employers can then ensure that these perquisites are met before installing and using the application. If the OfficeMA goes into mainstream use, then future work will be needed to consider the accessibility to people with disabilities.

2.8 Summary

It is clear from the background material presented above that now there is more demand on the Web to host many types of applications, ranging from the basic Web application to the more sophisticated Rich Internet Applications. Basic Web applications attempt to utilise the Web as an application medium by using Web pages and trying to apply some of the Web design principles to these applications. The problem is that these design principles are mainly for information oriented Web sites where navigation and contents is at the heart of the Web design principles. Consequently trying to push functionality-oriented applications to fit this paradigm may raise many issues in some cases as some application might not fit into the Web design paradigm.

The root cause of the above complications is due to the hypertext nature of the Web which is oriented toward contents and information rather than functionality. However, although the Web is intrinsically a hypertext medium, the foundations are there for an Object-Oriented and functional oriented medium similar to the one used for desktop applications. This section has demonstrated these similarities as summarised in the Table 2 below.

Applications such as the OfficeMA designed as part of this project, are more functionality oriented than information oriented. OfficeMA was designed to be used by a group of people possibly on a daily bases to achieve the same tasks, rather than to browser for contents or new information. This type of applications is more suited to be a desktop application than a Web application. However, it is desired to have the best of both worlds, the usability of desktop applications combined with the many features of Web applications such the ability to run on heterogeneous systems, centralised maintenance, remote access, etc... to mention only a few.

Table 2 – Similarities between desktop applications and Web capabilities

Application Type	Object Oriented	Event-Model	Widgets
Native desktop applications	Yes	Based on the OS API	Based on OS GUI widgets
Web applications based on HTML	Yes, based on JavaScript	Based on the DOM event model	Based on HTML widget and customized widgets
Web applications based on Plug-ins such as Adobe AIR and Microsoft Silverlight	Yes, based on Vendor's implementation, but most use JavaScript, C#, Ruby, etc...	Based on Vendor's implementation	Based on Vendor's implementation.

When designing and implementing an application such as the OfficeMA the project used the traditional user interface design principles used for traditional desktop application. By not doing so and trying to follow Web design principles would mean throwing away many years of Graphical User Interface design principles as stated by Nielsen [22]. The project has clearly demonstrated that the application of these traditional design principle to the user interface and treating it as a Graphical User Interface rather than a Web User Interface has resulted in smooth, cost effective and usable design that enabled the implementation of complex functionality.

3 Project Requirements

3.1 *HR Requirements of Small Businesses*

The majority of the requirements for this project have been identified for a small business to manage their human resources efficiently and in a cost effective way. A number of human resources areas have been identified in consultation with the client as candidates for automation using the OfficeMA. These areas can be summarised as follows:

- **Staff** – A module that can be used to create, update and view staff details
- **Expenses** – A module that can be used to create, submit, approve and pay expenses.
- **Tasks** – A module that can be used to indicate to staff members that they have an action to perform in the application.
- **Holidays** – A module that can be used to request and approve holidays
- **Timesheet** – A module that can be used to record the staff timesheet
- **Settings** – A module that can be used to change the system settings so that the application is configurable and can fit the need of a number of small businesses
- **Pay** – A module that can be used to calculate the staff pay and produce payslips

The author of this dissertation also believed that based on the number of Small businesses in the UK, such an application can satisfy the requirements and benefit the majority of other small businesses. Subsequently the author has conducted a survey as part of this project (Appendix A) to identify the methods currently used by small businesses to manage the areas recognized above. The survey also aimed at determining if these small businesses have made use of any existing software, and if they have not, to identify the reasons why existing software applications or the development of customised applications have not been pursued.

The businesses surveyed that have 15 or less staff relied on paper or spreadsheet based process for the areas identified above in except to the Pay area which was either outsourced or done in house using Sage products [59]. Sage has established a strong market hold in the accounting software applications arena and provides products that satisfy the need of most businesses at a reasonable price. Based on this, it was decided to drop the Pay

module from the OfficeMA and to concentrate on filling the gaps by providing a system to cover the other areas that are currently done manually.

3.1.1 Obstacles facing small businesses

Even though all the small businesses surveyed have computer networks in their offices, none of them has utilised it to enable the automation of the above mentioned Human Resources (HR) areas. The main obstacles that faced these businesses in obtaining licensed software or developing their own bespoke applications have been identified as follows:

- The huge licensing fees associated with established software products and the infrastructure that is usually required to run them.
- The complications of large software products that usually contains much functionality than what is required.
- The lack of internal IT skills or budget required to develop in-house bespoke applications.
- The lack of budget needed to outsource the development of a bespoke HR application to an external specialized company.
- Most of the budget applications on the Internet are either standalone to be used by a single person or are sold in hosted mode where the data is held outside the small business office.

3.1.2 What is needed and why?

The small businesses surveyed have expressed a strong need for a cost effective office management application for the following reasons:

- Reducing the time used to manually process and administer staff, expenses, holidays and timesheets.
- Enabling better control and archiving of electronic forms and records.
- The ability to process, complete and submit requests and forms remotely without the need to physically be in the office.

These businesses have also identified the following as must have features and characteristics of such an application:

- Cheap to buy, install and maintain.
- Simple to use and manage without the need for extra skills.

- Can be installed in-house using existing network and computer specification and capabilities
- Can be run on the Intranet and possibly over the Internet.
- An easy to use application that is user friendly, intuitive, interactive and responsive.
- A secure application that can be accessed over the Intranet or the Internet in a secure way. Sensitive data should also be held securely such bank account details and users' passwords.
- Provide roles for application users as follows:
 1. **Administrator** – A staff member who have unlimited access to the application and can create and modify staff details.
 2. **Accountant** – A staff member who is primarily deals with staff pay, taxes and the paying of expenses. Majority of the times in small businesses this role is outsourced to an external accountant who needs to be able to log remotely and have a view of the staff expenses, timesheets and details. The access rules for this role should be controlled by the administrator.
 3. **Regular Staff** – Any one else in the small office who use the system to view and update some of their details and submit expenses, holidays and timesheets. The access rules for this role should be controlled by the administrator.
- Provide authentication, authorisation and role based restrictions to enable administrators, accountants and staff members to login securely and access data that is allowed for their roles.

3.1.3 Alternative applications

Based on the small businesses requirements identified above the project researched a number of alternative software applications to the OfficeMA developed by this project and assessed them in terms of these requirements. Table 3 below summarises these applications and some of the issues that make small businesses reluctant to adopt them:

Table 3 – Alternative software applications and their issues

Software Application	Vendor	Advantages	Small Businesses Issues
SharePoint [53]	Microsoft	<ul style="list-style-type: none">• The ability to customize the forms and templates and add more functionality by adding plug-ins• Suitable for large organisations	<ul style="list-style-type: none">• Big licence cost• Requires Microsoft server operating system and Microsoft SQL server at extra cost• Expertise required to maintain and manage adds extra cost
Oracle HR [55]	Oracle	<ul style="list-style-type: none">• A wide range of functionality.• Suitable for large organisations	<ul style="list-style-type: none">• Big licence cost• Requires other software components at extra cost• Expertise required to maintain and manage adds extra cost
Tommie UK [68]	TOMMIE Systems Ltd	<ul style="list-style-type: none">• Reasonable price per user per day• A wide range of functionality• No required resources as it is a Web based system	<ul style="list-style-type: none">• Contains a great deal of functionality that can be distracting and hinders quick learning• Only offered in hosted mode. This is a problem for small businesses wanting to keep the system in-house and may be on the Intranet only.

3.2 Software used for the Project

The main drive behind this project is the development of a cost effective office management application by trying to use effective design principles to reduce design cost and a number of free and Open Source technologies and tools to reduce the implementation and deployment cost. These free technologies and tools are listed in (Appendix I) and range from the programming languages used to implement the application to the component and frameworks used during the implementation. It is also clear from the requirements that such an application should be cost effective to install and maintain such as the ability to be deployed on Window or Linux based systems.

3.2.1 Java Web Components

For the reasons mentioned above the project has decided to use the Java programming language to implement the OfficeMA. The Java programming language has come along way and has matured and established itself as an effective and cheap Object Oriented programming language. Java is also well suited to multi-tier enterprise applications developed and can benefit from the availability of a wide range of Open Source frameworks and products. Java is also platform independent and Java applications can be run on a variety of operating systems. With the advances in the JVM and the hardware components such as memory and processor speed nowadays, Java now performs well and is highly flexible.

In comparison to other technologies such as Microsoft .NET, there is a requirement and dependency on other proprietary software such as Microsoft Internet Information Services (IIS) Server and the Microsoft Windows operating system. Other Open Source technologies such as PHP are geared towards rapid scripting development for Web sites or simple Web applications. However, the maintainability of the code becomes unmanageable once the application requires a large number of software components.

This project used the Java programming language to implement the business logic and the JavaServer Pages (JSP) to implement the Web pages. Other technologies such as HTML, CSS and JavaScript were also used on the Web pages development. This approach aides the separation of concerns in such a way that the business logic is kept separate from the display and Web pages. This enables the future expansion of the application to use a different display technology if required. Other free Java based frameworks and products were also used to build the OfficeMA such as Hibernate [48], Struts [63], Spring [60] and Tomcat application server [67].

4 Design Methodology

This section outlines the software design methodology used for this project. The methodology is largely based on the USDP detailed by Bennett et al [4]. The main focus is on Model - Driven architecture to enable the business-level functionality to be modelled by standards such as UML and Entity-Relations Model. This approach reinforces the focus on business first then technology, enabling the business model to exist independent of any platform for technology. One advantage of this approach is the ability to freely choose the target platform and technology or switch from one to technology to another without having to modify the business model [15].

4.1 Project Lifecycle

The software development activities of the project followed the Traditional Waterfall Lifecycle (TLC) model to design and implement the OfficeMA (Figure 6). One of the drawbacks of TLC is the unresponsiveness to change in client requirements during the project. Another approach to use to overcome the change in client requirements is the Waterfall lifecycle with feedback loops, yet again it is not without drawbacks and the resulting iterations can prove very costly [4]. For the purpose of this project with fairly clear requirements from the start of the project, the TLC despite its drawbacks offered a very structured approach to systems development.

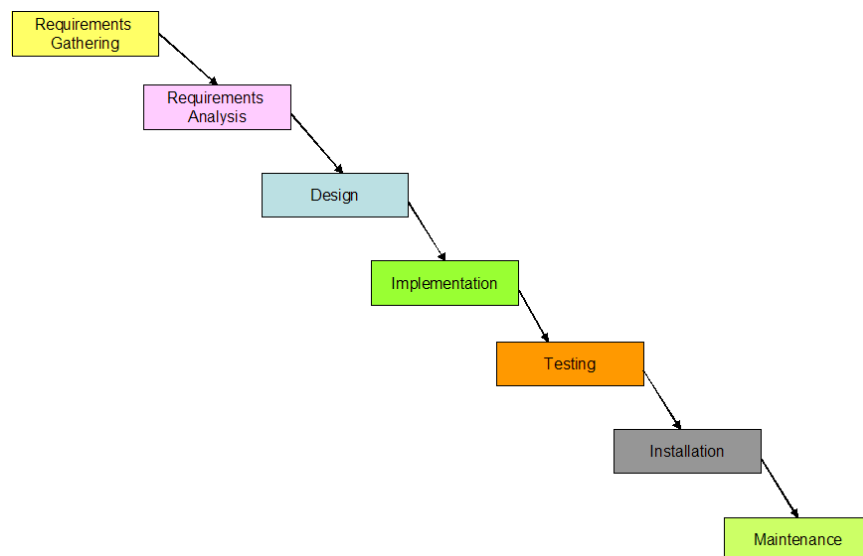


Figure 6. Traditional waterfall lifecycle model.

4.2 The Software Development Process

The project followed the development process outlined by Bennett, et al [4, pp119]. This process is consistent with the USDP and incorporates techniques from other sources such as Arrington and Rayhan [1]. The USDP was favoured over other software development methodologies such as Agile development [65] due to the clear requirements which were not likely to change over the lifecycle of the project. The main activities that were followed are summarised as follows:

- Requirements capture and modelling
- Requirement analysis
- System design
- Class design
- User interface design
- Data management design
- Construction
- Testing
- Implementation

These activities are shown in Figure 7 below. Some of the activities depend on others, but some such as the class design, user interface design and logical database design can be done in parallel.

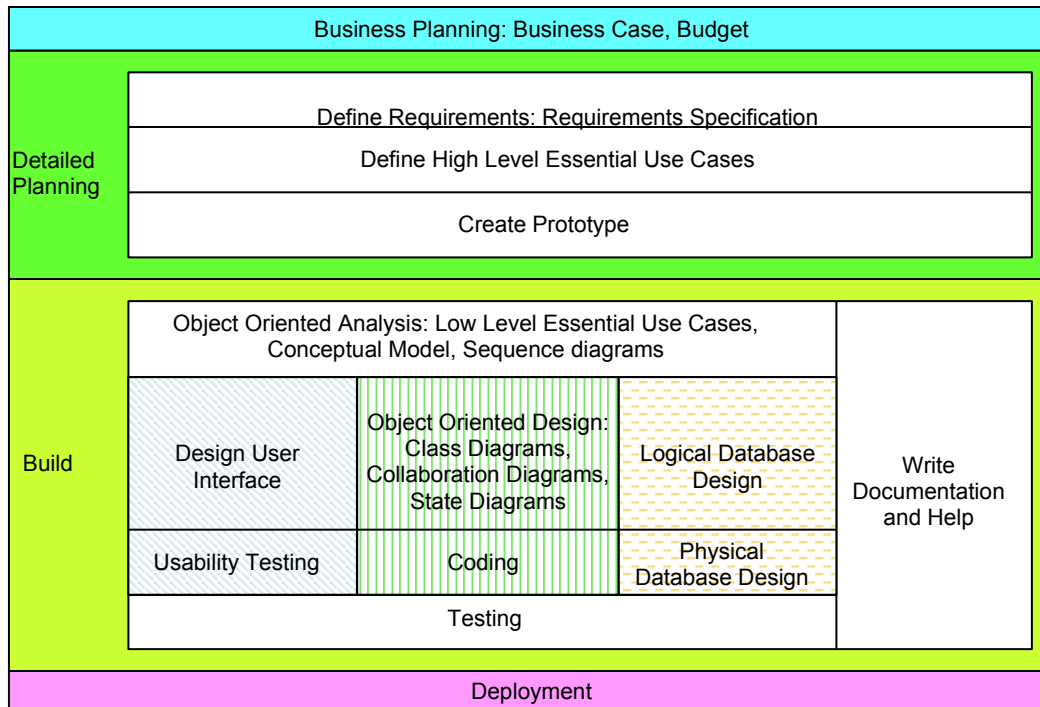


Figure 7. Activities that lead to software deployment, adapted from Grand [21].

Below is a summary of the main activities motioned above alongside their techniques and key deliverables.

4.2.1 Requirement Capture and Modelling

Techniques

1. Requirement elicitation using:

- Background reading
- Fact finding interviews with the client
- Observation
- Document sampling (Appendix B)
- Questionnaires.

In this project two of the above requirement elicitation methods were used, these were: fact finding interviews with the client and document sampling of paper or spreadsheet based forms. These two were sufficient to capture the requirement for the application.

2. **Use case modelling** to be carried out and documented as follows:

- Use case diagram.
- Use case description
- One or more flow of events (i.e. Normal or baseline flow, alternative flows and exception flows)
- Activity diagram. The use of activity diagrams to aid the understanding of the use case diagrams is outlined by Arrington and Rayhan [1]. This is because activity diagrams are less technical than the sequence diagrams and their use at such an early stage can aid the understanding of the business stakeholders.

3. **Initial system architecture** can be developed to help guide subsequent steps during the development process and can be refined and adjusted as the development process progresses. The initial system architecture is usually the package structure of the system.

4. **Prototypes** of some key user interfaces are to be produced to aid the requirement understanding and gathering.

Key Deliverables

- Use case model
- Requirements list
- Initial architecture
- Prototypes

4.2.2 Requirement Analysis

Each of the use cases produced during the requirement gathering and modelling stage are separately analysed to identify the classes that support it.

Techniques

1. **User case realisation** is used to derive communication diagrams to model the object interaction. The models for each use case are then integrated to produce an analysis class diagram.
2. **Domain analysis** can also be used to derive an analysis class diagram. Textual analysis and Class Responsibility Collaboration (CRC) Cards are two of the techniques widely used at this stage.

Use case realisation was favoured over domain analysis and used in this project for two reasons, the first reason is the fact that it is part of the USDP and domain analysis is not, and the second reason is that the extra modelling involved with the use case realisation method help in capturing all the classes that satisfy the use case through the use of collaboration diagrams. The extra modelling involved also helped in understanding the steps required to satisfy the use case.

Analysis class diagrams can be classified into three stereotypes: boundary, control and entity. Control classes represent control, coordination and sequencing. The USDP recommends the use of at least a control class for each use case [4]. On the other hand boundary classes model the interaction between the system and its actors. There are two types of boundary [1]:

- **User Interfaces** – Allow the system to interact with humans. This will form the starting point for the user interface design.
- **System Interfaces** – Allow the system to interact with other systems.

Entity classes are used to model some of the real-life object or real-life events. Instance of entity classes will often require persistent storage of information about the things they represent. This will also be the starting point for data persistence requirements and database design.

Key Deliverables

- Analysis models such as analysis class diagrams and communication diagrams

4.2.3 Class design (Detailed design)

The aim of this stage is to elaborate each use case to include design decisions and enhance the analysis class diagram to produce a detailed design class diagram.

Techniques

1. **Interaction diagrams** are used to show detailed object communications.
2. **State diagrams** are used for objects with complex state behaviour if any.
3. **Detailed design** class diagram is produced by the integration of the separate models additional classes are added to cater for interaction with the user interface and database.
4. **Design patterns** as explained by Gamma et al. [17] are applied to the class diagram to address common problems in the domain model.

Key Deliverables

- Design models such as detailed class diagrams, interaction diagrams and state diagrams.

4.2.4 User Interface Design

Interface design is very dependant on the class design. The user interface boundary classes identified earlier are enhanced by adding more details to model the user's interaction with the system. The user interface followed the design rules outlined in section (2).

Techniques

1. Prototyping the user interface by following these activities.
2. Designing the boundary classes.
3. Modelling the interaction involved in the interface with sequence diagrams.
4. Modelling the control of the interface using state machines.

Key Deliverables

- Design models with interface specification.

4.2.5 Database Design

The database development starts from the persistence requirement for the application and relates to the presence of entity classes in the analysis class diagrams. These entity classes usually require the persistence of some or all of their details. The relational database design principles are followed here (Figure 8) as outlined by Elmasri and Navathe [13] and Connolly and Begg [8].

Besides designing the database layer the project also considered the Object to Relational Mapping (ORM) layer (Figure 17), which was used to decouple classes from the mechanism by which instances are stored in the database [2]. As stated by Bennett et al [4] one favoured approach is the use of a persistence framework, the main feature of which is the use of database brokers or database mappers. These mediate between the business classes and the database and are also responsible for **C**reating, **R**etrieving, **U**psdating and **D**eleting (CRUD) objects.

Techniques

Elmasri and Navathe [13] has summarised the phases of database development as follows:

1. Requirements collection and analysis.
2. Conceptual database design.
3. Choice of DBMS.
4. Logical database design, which involve the mapping the relations from the conceptual model into the target DBMS and View design
5. Physical database design.
6. Database system implementation and tuning.

Connolly and Begg [8] provide a more concise design methodology that was followed on this project, it can be summarised as follows:

- Conceptual database design, such as ER modelling.
- Logical database design, such as relational modelling.

- Physical database design, such as DDL, DML, physical storage, indexes and security measures.

The main reason for choosing these concise steps, is the fact that the focus on this project is on the domain model (classes) developed using the USDP rather than starting from data requirements. As mentioned above the requirements for the database model come from the persistent entities in the analysis classes, which have the required attributes and their types already defined. This also means that a great deal of the design is already done at the analysis stage.

Key Deliverables

- Conceptual data models.
- DDL and DML SQL

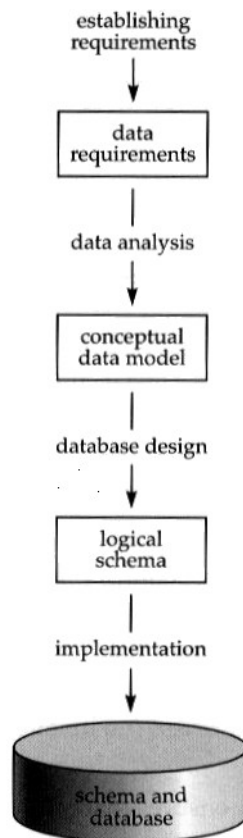


Figure 8. Sample Model of Database Development [23].

4.2.6 Construction, testing and implementation

Techniques

1. Implementing the business logic.
2. Implementing the user interface.
3. Implementing the database.
4. System documentations and help files are written
5. Completed system is deployed and tested. Found bugs are fixed in accordance to severity and priority, some are even left in the final product if these were accepted by the client and the bugs are of low priority. The final tested product is released for deployment

Key Deliverables

- Constructed system documentation.
- Source code for the developed application
- Installed system.

5 Preliminary System Design

The preliminary system design aims at providing an analysis model, which contains a user interface prototype and a number of UML models such as use case diagrams, activity diagrams, communication diagram, sequence diagrams and finally an analysis class diagram.

5.1 Requirements Gathering

The requirements for the OfficeMA were captured through a number of fact finding interviews with the client and the sampling of various documentations (Appendix B). The results of the fact finding interviews have been incorporated into the use case descriptions. These interviews were also conducted during the analysis stage to clarify outstanding issues. The Jude UML CASE tool (Appendix K) was used to construct and produce the various UML models used during the requirements gathering, requirements analysis and design phases of this project.

5.2 Initial System Architecture

The initial package structure of the system is shown in Figure 9 below. This is based on the logical packaging of the high level use cases of the application, and can be summarised as follows:

- Staff management
- Expenses management
- Holidays management
- Task management
- Time booking management
- System settings management

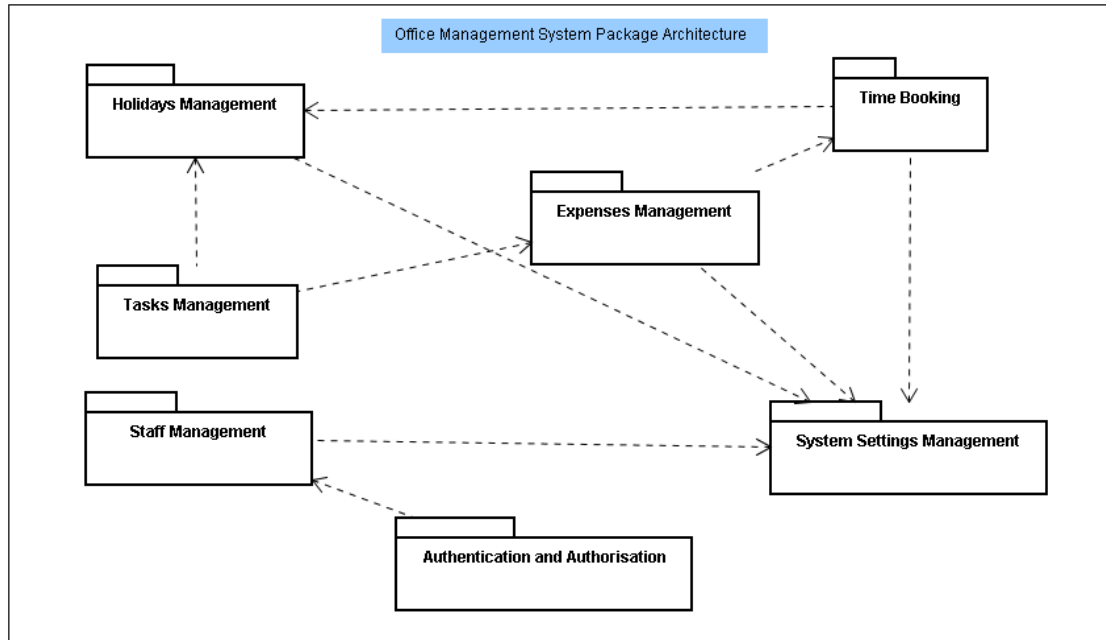


Figure 9. Initial package architecture for OfficeMA

5.3 Requirement Capture and Modelling

5.3.1 Prototyping the User Interface

As part of the requirements capture process, a number of prototypes have been developed to aide the use cases and the clarification of the requirements with the client. Methods used to create prototypes for traditional desktop applications were used in this project. The author has outlined some techniques that can be used to create prototypes for Rich Internet Applications [11]. These techniques can be summarised as follows:

- Get a catalogue of currently used visual widgets such as controls, commands, pointers and windows widgets. An example is the Visual Index on the Microsoft Window Vista User Experience guide [71].
- Short-list the widgets that can be included in the prototype and the completed user interface later on. These are widgets that will be available in the toolkit used for the user interface implementation or can easily be developed.
- Construct the user interface prototype using these widgets.

5.3.2 Staff Management Requirements

Requirements Summary

The application should be able to hold employee details and support user roles, so that different information can be updated by different roles.

- Enable admin users to configure access rights of various users of the system depending on their role
- The ability to update/add employee details such as name, contact numbers, email address, date joined, address, national insurance number, bank details, grade, salary, holiday entitlement, tax code etc...

Requirements List

R1 The system should enable the administration of staff details using the. Such as adding new staff, changing and viewing staff details in which case the staff member will receive a task to review the changes. Administrators should be able to add/view/edit all staff details. Normal staff should be able to edit some of their details in which case the administrator should get a task to verify these updates. Normal staff should also be able to view a brief summary of other staff details.

Table 4 – Requirements summary for staff management

No.	Requirement	Use Case(s)
1.1	Staff to view their personal details	1.1
1.2	Staff to edit some of their personal details, such as contact numbers and address	1.5
1.3	Staff to search for other staff members by name, id or current project, work-stream, employee type, etc...	1.2
1.4	Staff to view a brief summary of other staffs' details, such as email address, name, photo, phone numbers and grade	1.3
1.5	Administrator and Accountant to view complete staff details	1.4
1.6	Administrator to edit staff details, such as change of grade, salary, online access	1.6
1.7	Administrator to add new members of staff such as personal, address, employment and online access details	1.7

Use Cases

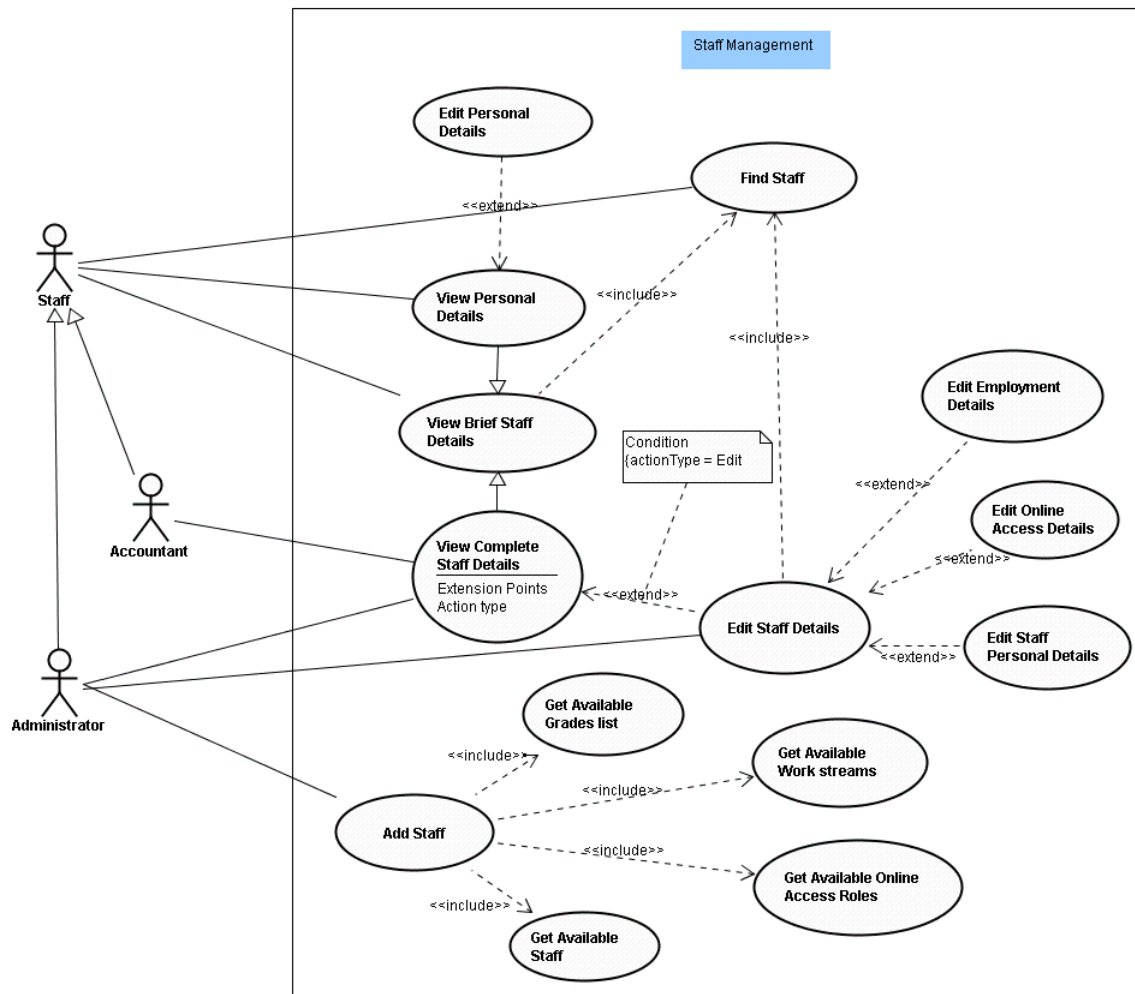


Figure 10. Use cases for staff management.

Use Cases summary

The detailed use case descriptions are provided in Appendix C.

Table 5 – Use cases summary for staff management

No.	Use Case	Description
1.1	View personal details	The current member of staff views their personal details
1.2	Find staff	Search for a member of staff or browse all staff details
1.3	View brief staff details	Regular staff members can view a brief summary of other staff details
1.4	View complete staff details	Admin, accountant users can view all the details for any member of staff

1.5	Edit personal details	Members of staff can edit their personal details. Regular staff can edit only a subset of their details. Admin users can edit all their personal details
1.6	Edit staff details	Admin users can edit other staff details.
1.7	Add Staff	Add a new member of staff. First all the Available grades, work-streams, online access roles and staff names are retrieved. The admin user completes the personal, bank, address, employment and online details.

5.3.3 Expenses Management Requirements

Requirements List

R2 The system should allow members of staff to enter/save/submit their expenses. The system should also notify approvers to approve/pay submitted expenses and allow approvers to either approve or reject an expense

Table 6 – Requirements summary for expenses management

No.	Requirement	Use Case(s)
2.1	Staff to view their saved/submitted expenses	2.1, 2.2
2.2	Staff to be able to edit their saved expenses and save/submit them	2.4
2.2	Staff to be able to add new expenses, the system should detect if saved expenses already exist for chosen period.	2.4, 2.3
2.2	Staff can only edit saved or rejected expenses	
2.3	Expenses period should be configurable, but defaults to a months	
2.4	Staff can only edit/create expenses for periods that does not already have submitted expenses	
2.5	Expenses approver should be notified through tasks/email when an expense is submitted	
2.6	Expenses approver should be able to approve/reject submitted expenses	2.6
2.7	Accountant should be notified when an expense is approved and should be able to pay it	2.7
2.8	Staff should be notified through tasks/email when their submitted expenses status change	
2.9	Staff should be able to search their expenses and categorise them by year/status	2.1, 2.2
2.10	Approvers should be able to search submitted expenses	2.5

	for which they are responsible for approving and categorise them by year/status	
2.11	Staff can either fill in a mileage or an amount. Current mileage is added to previously entered mileage and has different rates of payments.	2.4
2.12	The system should allow the staff to enter a mnemonic when entering expenses, an entered mnemonic updates type, description and amt/miles, any changes to these fields will update the saved mnemonic.	2.4
2.13	Staff should be able to export location entries from timesheets using the location mnemonic	2.4
2.14	A saved or submitted expense should contain at least one expense item.	

Use Cases

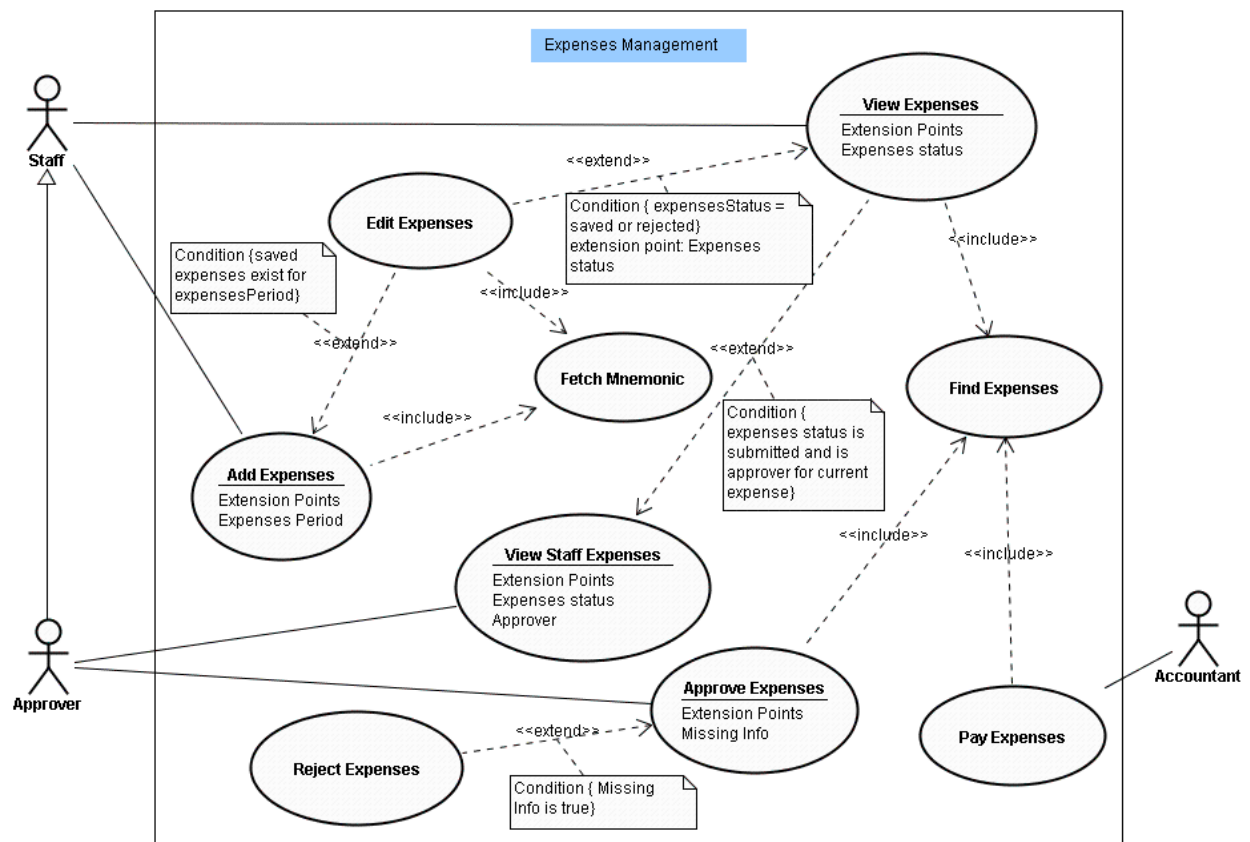


Figure 11. Expenses management use cases

Use Cases Summary

The detailed use case descriptions are provided in Appendix C.

Table 7 – Use cases summary for expenses management

No.	Use Case	Description
2.1	Find Expenses	Staff members can find their expenses using an expenses browser, where they can search by year or expenses status. Approvers can also search expenses they are approving by staff name, year or expenses status
2.2	View Expenses	After finding an expense the staff member can view its details.
2.3	Add Expenses	A staff member can add new expenses after choosing the period of the expense if a saved expense exist for the chosen period then the staff edits the expense using the edit expenses use case
2.4	Edit Expenses	After viewing an editable expense or trying to add a new expense for a period where a saved expense already exist the staff member is allowed to edit the expense and cancel, save or submit their changes
2.5	View Staff Expenses	The expenses approver should be able to view the staff expenses in pending/approved or paid states
2.6	Approve/Reject Expenses	The expenses approver should be able approve or reject the staff expenses in pending state
2.7	Pay Expenses	The expenses approver or the accountant should be able to pay the expenses in approved state

5.3.4 Authentication and Authorisation Requirements

Requirements Summary

R3 The system should authenticate users before allowing them to use it. Each user should have a user role as well to determine if they are authorised to perform a specific action

Requirement List

Table 8 – Requirements summary for authentication and authorisation

No.	Requirement	Use Case(s)
3.1	The system should run on a secure web server using HTTPS	
3.2	The user should provide a username and password to login to the system	3.1
3.3	The system should authenticate the user against stored user details and allow access if details match	3.1
3.4	The system should lock the user account after three unsuccessful login attempts and create a task for the administrator	3.1
3.5	After a user logs in successfully the system should assign the role for the user in order to determine if the user is allowed to perform a specific action or not	3.1

Use Cases

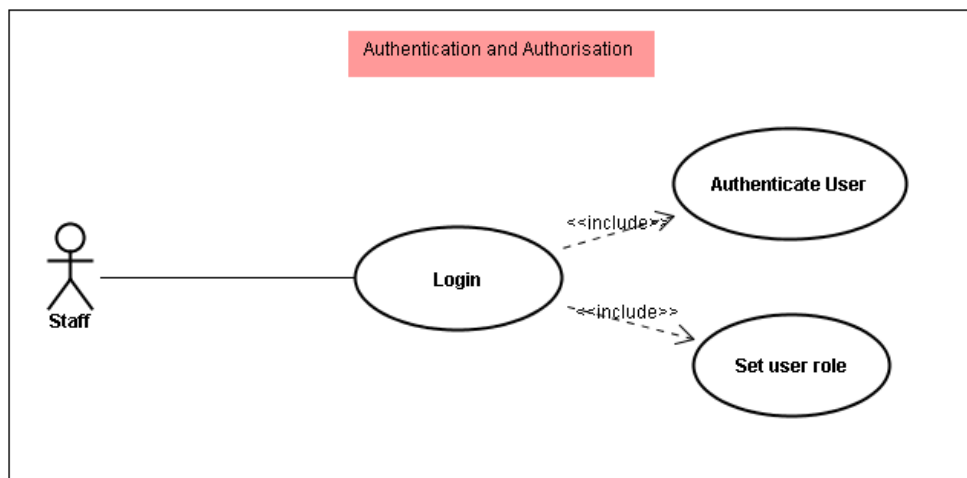


Figure 12. Login use case diagram

Use Cases Summary

The detailed use case descriptions are provided in Appendix C.

Table 9 – Use cases summary for authentication and authorisation

No.	Use Case	Description
3.1	Login	The member of staff provides their username and password to login and the system performs authentication and authorisation on the details provided.

5.3.5 System Settings Management

Requirements Summary

R4 The user should be able to change some of the settings in the system, such as holiday roles, projects, work streams, etc...

Requirement List

Table 10 – Requirements summary for system settings management

No.	Requirement	Use Case(s)
4.1	Administrators should be able to configure time booking management settings: <ul style="list-style-type: none">• Add/Edit a work stream• Add/Edit projects to a work stream• Configure the working days and beginning of the week• Configure timesheet entry unit (e.g. hour/minute)	4.1, 4.2
4.2	Administrators should be able to configure expenses settings: <ul style="list-style-type: none">• Add/Edit expenses categories• Add/Edit list of expenses approvers• Add/Edit expenses period (e.g. 1 month, 1 week)	4.1, 4.2
4.3	Administrators should be able to configure staff management settings: <ul style="list-style-type: none">• Add/Edit list of available grades (consultant, senior consultant, principal consultant, director)	4.1, 4.2

	<ul style="list-style-type: none"> Add/Edit list of online roles (staff, administrator, accountant) Add/Edit list of available personal managers 	
4.4	Administrators should be able to configure holiday management settings: <ul style="list-style-type: none"> Add/Edit list of holiday approvers Add/Edit holiday rules, entitlement, holiday year start/end, carryover limit, carryover cut-off. 	4.1, 4.2
4.5	Administrators should be able to configure task management settings. <ul style="list-style-type: none"> Send emails when creating tasks. 	4.1, 4.2

Use Cases

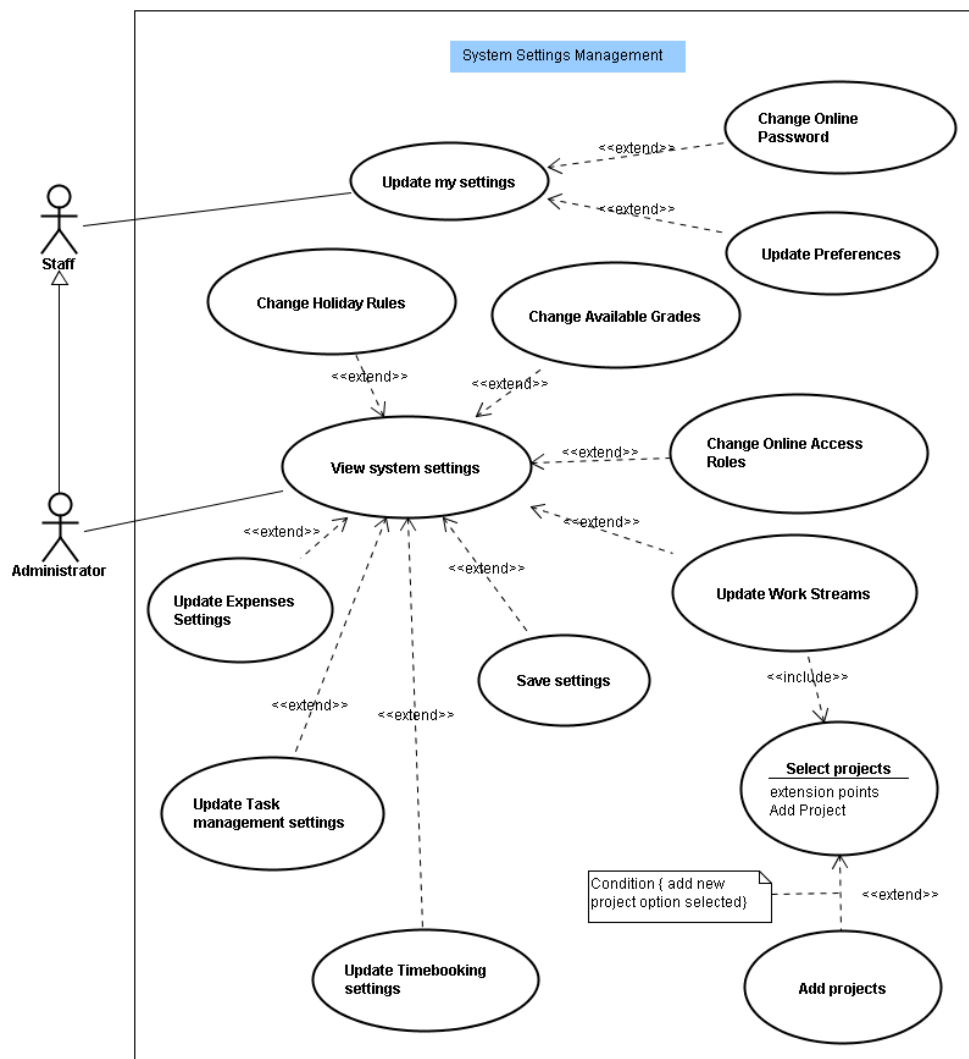


Figure 13. System settings management use cases diagram

Use Cases Summary

The detailed use case descriptions are provided in Appendix C.

Table 11 – Use cases summary for system settings

No.	Use Case	Description
4.1	View system Settings	An administrator can view the system wide settings and update/save them
4.2	Update system setting	After viewing the system settings the administrator updates the setting
4.3	Update my settings	A staff member can change their online password or update their online preferences such as desktop background colour, etc...

5.3.6 Time Booking Requirements

Requirements Summary

R5 The staff members should be able to book their time to project in their work stream. The staff members should also be able to enter any other type of absence through the time booking system, such as unpaid leave, sick leave and holiday. The system should cross check any holiday entered in the timesheet against the holidays request/taken. The system should also be able to auto populate the timesheet with holiday entries when these exist in the holiday subsystem

Requirement List

Table 12 – Requirements summary for time booking

No.	Requirement	Use Case(s)
5.1	Staff member should be able to view their time sheet and if they choose they can add, update or edit entries and then save their timesheet	5.1
5.2	Staff members should be able to view a summary of their timesheet entries for a given period and filter by project	5.2
5.3	Administrators should be able to view other staff timesheets	5.1
5.4	Administrators should be able to view a summary of other staff timesheets and filter by project	5.2

5.5	The system should be able to auto populate holiday from the holiday subsystem when the staff member is completing the relevant period	
-----	---	--

Use Cases

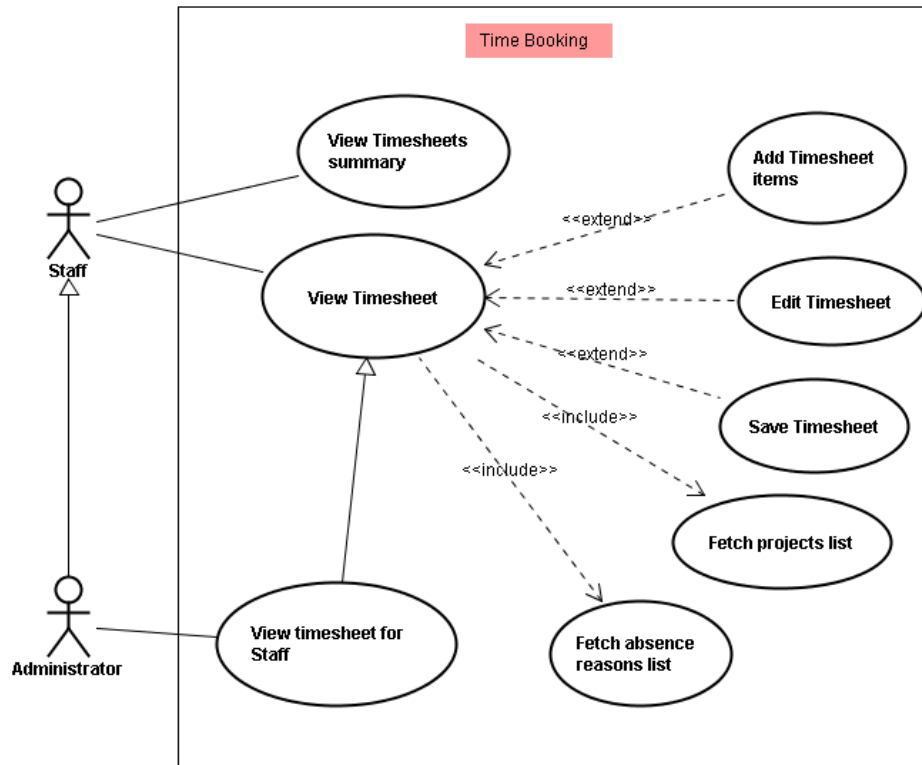


Figure 14. Time booking use cases diagram

Use Cases Summary

The detailed use case descriptions are provided in Appendix C.

Table 13 – Use cases summary for time booking

No.	Use Case	Description
5.1	View/Add Timesheet	Staff members can view their timesheet for a specific period and also update/save their timesheet or add a new entry. Administrator can view or edit timesheet entries for all staff members
5.2	View timesheet summary	Staff members views a summary of their timesheet entries

5.3.7 Holiday Management Requirements

Requirements Summary

R6 Staff members should be able to request holidays through the system. The system should be able to display the holidays entitlement, holidays available, request and taken for previous, current and next year. The system should also enforce the holiday settings such as leave carryover, etc.... The holiday approver should be able to approve or reject requested holiday. The staff member should also be able to cancel their requested holidays if not approved, if approved then only the approver can cancel the request. Approvers should be able to view holiday details for their staff members. Staff member should be able to see a holiday calendar showing the taken and approved and requested holidays for other staff members.

Requirement List

Table 14 – Requirements summary for holiday management

No.	Requirement	Use Case(s)
6.1	Staff to view their requested/approved holidays	6.1
6.2	Staff to be able to cancel their requested if not approved	6.1
6.3	Staff to be able to request new holiday	6.1
6.4	The system should enforce the holiday settings entered as part of the system settings management use cases	
6.5	Holiday approver should be notified through tasks/email when a new holiday request is submitted	
6.6	Holiday approver should be able to approve/reject submitted holidays	6.1
6.7	Staff should be notified through tasks/email when their submitted holiday status change	
6.8	Approvers should be able to view holiday details for staff members they are approving for and change their holiday status	6.1
6.9	Staff should be able to import holiday entries from the timesheet when these don't exist in the holiday subsystem	
6.10	A holiday request should be for at least half a day and weekends are automatically ignored.	
6.11	Staff member should be able to see a holiday calendar showing the taken and approved and requested holidays for other staff members.	6.2

Use Cases

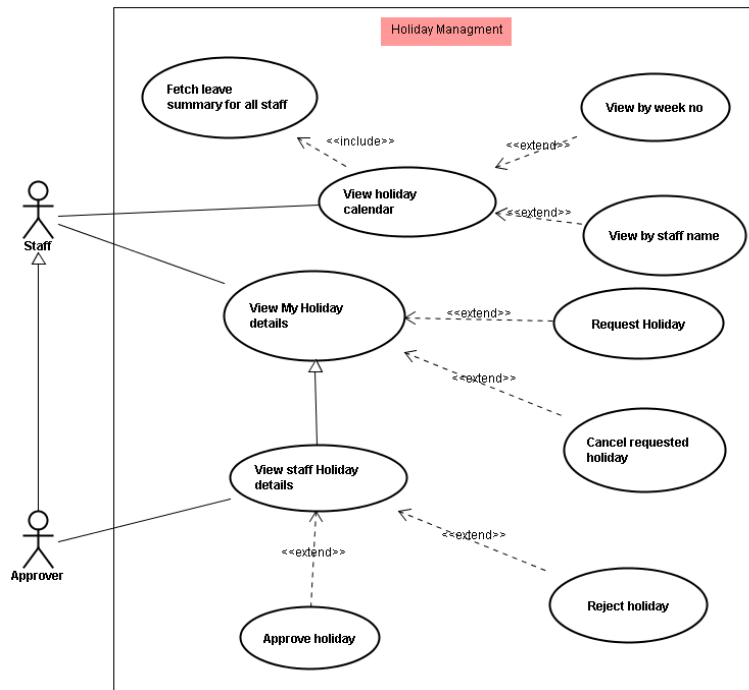


Figure 15. Holiday management use cases

Use Cases Summary

The detailed use case descriptions are provided in Appendix C.

Table 15 – Use cases summary for holiday management

No.	Use Case	Description
6.1	View Holiday details	Staff member can view their holiday summary and request new holiday or cancel a requested holiday. The system should also enforce the holiday roles and import holidays entered in the time booking system. Holiday approvers should be able to view the holiday for their staff and approve or reject their holidays.
6.2	View Holiday Calendar	Staff member can view a calendar with the holiday of all any of the staff members. The staff member can also configure the number of weeks to view

5.3.8 Task Management Requirements

Requirements Summary

R7 Staff members should be able to the tasks assigned to them by the system.
The staff members should be to delete the task or set the task as complete.
The system need not validate that a staff member has actually completed their task

Requirement List

Table 16 – Requirements summary for task management

No.	Requirement	Use Case(s)
7.1	Staff members should be able to view their assigned tasks	7.1
7.2	Staff members should be able to set their tasks to completed	7.1
7.3	Staff members should be able to delete their assigned tasks	7.2

Use Cases

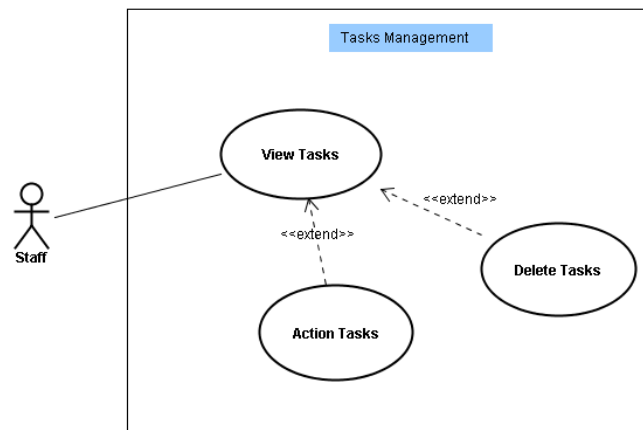


Figure 16. Task management use cases

Use Cases Summary

The detailed use case descriptions are provided in Appendix C.

Table 17 – Use cases summary for task management

No.	Use Case	Description
7.1	View Tasks	Staff members should be able to view their tasks and set them as completed if desired
7.2	Delete tasks	After viewing the tasks the staff members should be to delete these tasks

5.4 Requirement Analysis

Following the use case modelling, the analysis class diagrams were identified using the use case realisation process. Bennett et al [4] defines the use case realisation process as:

“..., use case realisation involves the identification of a possible set of classes, together with an understanding of how those classes might interact to deliver the functionality of the use case”

The collaboration between classes is usually modelled using UML 2.0 Communication diagrams, which are constructed starting from the actor by using a boundary class, then a control class to coordinate the logic in the use case and finally entity classes and the links between them. The author of this dissertation has drawn the conclusion that using use case realisation involves steps similar to the CRC method in trying to identify candidate objects, messages and links. In addition to this, use case realisation offers the ability to easily construct a class diagram from the communication diagram and offers better visibility than CRC. The following steps were used to construct the analysis class diagrams:

- For each use case construct a communication diagram.
- Use the objects in the communication diagram to construct the analysis classes and their stereotypes.
- Use the link between objects to identify association between classes and the multiplicity on each side.
- Use the messages to identify operations for the analysis classes and directions for associations.
- After identifying the operation for the analysis class diagrams, the attributes are identified from the requirements list and the communication diagrams method signature.

Appendix D contains the communication diagrams and the derived analysis class diagrams for each of the use cases developed above.

6 Implementation Strategy

6.1 System Architecture

The project design followed a layered architecture (Figure 17) to ensure encapsulation, maintainability, reuse and separation of concerns. The design covered the user interface layer, business logic layer and the database layer, with special emphasis on the user interface design. For such an application with database requirements to persist the data, the minimum that can be used is two tiers architecture; however this means that the business logic and the presentation logic will be included in one layer. In the case of the OfficeMA the two tiers makes code maintainability and future expansion difficult due to the scope and complexity of the application.

For this reason the three tier architecture was used to aides the separation of concerns in such a way that the business logic is kept separate from the display and Web pages. This enables future expansion of the application to use a different display technology if required.

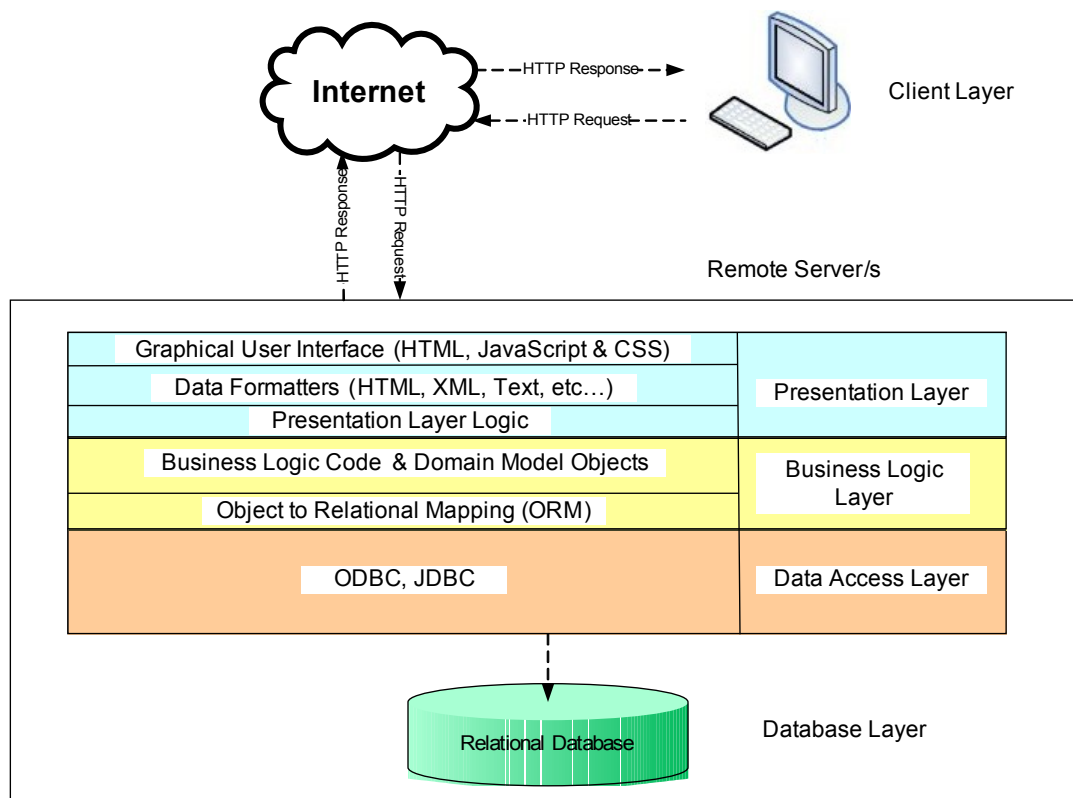


Figure 17. Layered architecture for a typical Web application.

The various implementation strategies and methodologies for the application layers have been identified below following a bottom up approach, starting from the database layer and moving up to the user interface layer. The aim is to develop a robust application that can be ported to different technologies without considerable changes to the business layer.

6.2 Database layer

6.2.1 Choosing a DMBS

The project used a relational database for its persistence requirements. As the project is driven towards cost reduction the first choice was to use an Open source DMBS. The two options that was available for the project was either using MySQL or PostgreSQL (Appendix I). It was decided based on the comparison shown in Table 18 to use the PostgreSQL database server as it strongly conforms to the ANSI - SQL 92/99 standards. This will support the transactional nature of the OfficeMA data requirements and ensures data integrity through the ability to use Check constraints, domains and Triggers. MySQL on the other hand is less compliant with ANSI standards for example Check constraints are not yet supported as indicated in Table 18 below. MySQL is more suited as a backend for Web sites where fact access and data reads are required.

Table 18 – Some of the features of PostgreSQL vs. MySQL [20]

Feature	PostgreSQL 8.0	MySQL 5.0
Performance	Slower	Faster
Sub-selects	Yes	Yes
Triggers	Yes	Yes
Full Joins	Yes	No
Constraints	Yes	No
Cursors	Yes	Partial

6.2.2 Database Implementation

After choosing the target DBMS the logical database design resulting from the detailed class diagram was revised and converted into a physical model to suit the PostgreSQL database. This activity included writing SQL for Data Definition Language (DDL), Data Manipulation Language (DML) and creating database indexes. The implementation strategy is outlined below:

- The candidate entities for the relational model were identified from the entity classes in the detailed class diagram.
- An Entity-Relation model was developed for the above entities.
- A relational model was build using these candidate entities.
- A physical model was build using SQL.
- The database tables were created and populated with sample data using SQL.
- Database testing was done using SQL queries directly on the database layer.

The crow's feet notation [23] was used for the ER modelling to represent the cordiality of the relation and the participation conditions as explained in Table 19 below.

Table 19 – Crow's feet notation used in the ER- modelling

Symbol	Meaning
Open blob - ○	Zero or one participation
Closed blob - ●	Exactly one participation
Open blob and crow's foot - ○	Zero or more participation
Closed blob and crow's foot - ●	One or more participation

6.3 Business Logic Layer

6.3.1 POJO Architectural Pattern

The heart of the OfficeMA is the business logic layer and consists of the domain model classes developed from the detailed class diagram. The detailed class diagram was developed as a direct result of use case modelling, which means that the domain model classes directly represent the business objects and requirements. The implementation of the business logic layer focused on transforming the detailed classes into runtime classes using the Plain Old Java Objects (POJOs) design pattern [30]. POJOs are simply JavaBean objects that enclose their attributes and operations. In JavaBeans, attributes are declared as private and setter / getter methods are defined to access these attributes.

In the developed domain model, entity classes alone were not enough to satisfy the business logic. Evans [15] has identified as part of the POJOs pattern a number of roles in the domain model by which classes can be categorised. A

class's role imply certain type of responsibilities and relationship with other classes in the domain model, these can be summarised as follows [30]:

- **Entities** - Objects with a distinct identity.
- **Value Objects** - Objects with no distinct identity.
- **Factories** - Define methods for creating entities.
- **Repositories** - Manage collections of entities and encapsulate the persistence framework.
- **Services** - Implement responsibilities that can not be assigned to a single class and encapsulates the domain model.

Figure 18 below shows part of the detailed class model for the application. The diagram includes a service class which is invoked from the presentation layer. The service also includes references to the various repository classes that are used to persist or retrieve the entity objects from the database. As seen from the diagram, Interfaces are used so that other classes can have reference to the interface without having to worry about the implementation. The implementation can be changed at any time and as long as the method signatures remain the same, there will be no impact on the other classes as a result of changing the service or repository implementation.

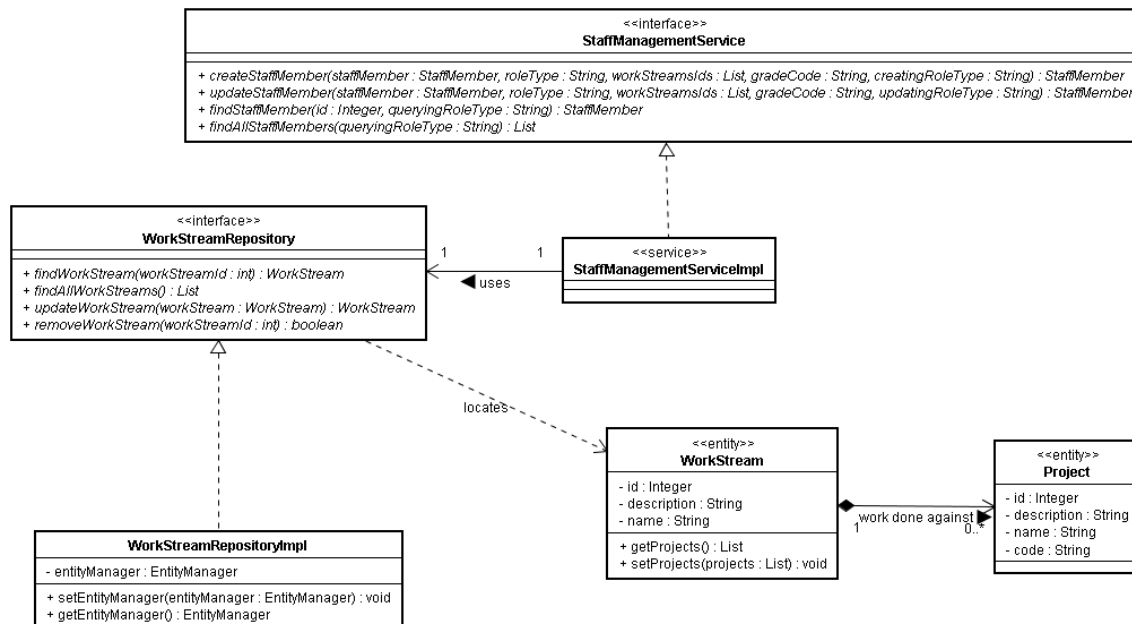


Figure 18. Part of detailed class model showing entities, a repository and a service

6.3.2 Spring Framework and Dependency Injection

There is another unanswered questioning regards to the design shown above in Figure 18. If the service classes reference the repository classes by interface how would they be able to instantiate these interfaces without knowing about the implementation concrete classes. Instantiating these repository classes using the concrete implementation classes tightly couples the code together and makes it hard to modify a single class as the change would likely to ripple through the other code and results in multiple class changes.

There is also another issue, the fact that POJOs by themselves are insufficient to run the application as Richardson [30] has stated:

“In an enterprise application you need services such as transaction management, security and persistence...”

To address the above issues the Spring framework [60] was used. Spring is a lightweight dependency injection, aspect-oriented container and framework. The term dependency injection is very important in regards to the reference by interface issue identified above. Walls [35] describes the benefits of dependency injection as follows:

“The key benefit of DI is loose coupling. If an object only knows about its dependencies by their interface (not their implementation or how they were instantiated) then the dependency can be swapped out with a different implementation without the depending object knowing the difference”

Spring framework can be configured to automatically instantiate and inject the dependency of each object in the domain model, this reduces coupling and encourages programming using interfaces. For example in Figure 18 the repository instances are create by Spring and then injected into the service at runtime. Spring is favoured over Enterprise Java Beans (EJB) as it is lightweight and can also run in a lightweight container such as Tomcat.

The alternatives to Spring such as EJB 2.1 is been largely criticized by the Java community for its shortcoming and declarative programming model. EJB 3.0 on the other hand is considered a step on the right side, but the technology is still new and the project has decided to use aspects of the its standard such as Java Persistence API [50] which when coined with the new features in Java 5 offers an effective way of persisting Java applications. EJB technology requires an EJB compliant container which requires more processing power and mostly geared towards multi-tier enterprise applications development rather than lightweight Web applications.

6.3.3 Domain model classes

The implementation strategy for domain model classes can be summarised as follows:

- The detailed class diagram was developed by applying the domain model roles discussed above.
- The detailed classes' skeletons were then exported to Java code using Jude UML CASE tool (Appendix K).
- Using the Eclipse IDE (Appendix I) the empty skeleton methods were populated with business logic derived from the sequence diagrams. In the simple case of entity classes these methods were getters and setters methods.
- For entity classes UML associations were converted into object references where appropriate.
- Entity classes were annotated with Java Persistence API annotations [50] according to the relational model developed for the database layer. This tells the JPA implementation how to map and persist these classes into database tables.
- For persistent entities a repository interface and implementation was coded. Repository classes were also annotated with the Spring `@Transactional` interface, this ensures that all the CRUD method in the repository are transactional and follow the ACID concept.
- Finally the class of the service responsible for executing the logic in the current package is coded; this service only have a reference to the interfaces of the repository it needs to use to persist the domain model entities. The reference to the objects to be injected was added to the Spring configuration file.
- The above steps are repeated for each of the packages in the detailed class diagram.

6.3.4 Object to relational mapping framework

The repository interfaces defined above need to be implemented somehow to persist the application objects into the database; however, before persisting these objects some mapping needs to be done to convert these objects into database entities. One approach of doing this is to manually map each object and write SQL to persist them into the database directly through the Java Database Connectivity (JDBC) driver. But this approach has some disadvantages as summarised below [2]:

- The mapping from objects to database entities is a time consuming process.
- The implementation is database specific and will need major changes to the repository code, mapping code and SQL if a different database implementation to be used.

A widely used solution to the problem above is the use of Object Relational Mapping (ORM) framework (Figure 19). Baur and King [2] describe the ORM as follows:

“In a nutshell, object/relational mapping is the automated (and transparent) persistence of objects in a Java application to the tables in a relational database, using metadata that describes the mapping between the objects and the database.”

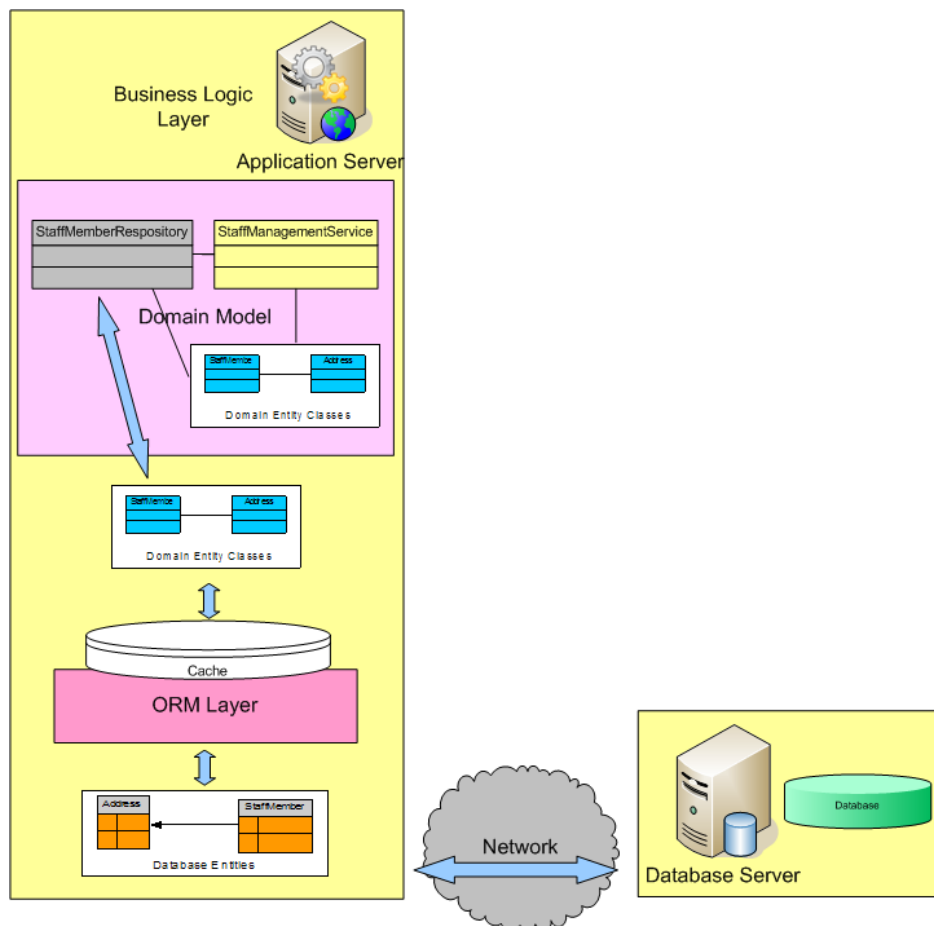


Figure 19. Business logic layer showing domain model and ORM layer

Using an ORM framework has many advantages, some of which are [2]:

- Increased productivity compared to manually mapping objects to relational tables.
- Easy maintainability as the code in the application is purely focused on business logic and the mapping is described using metadata.
- ORM has better performance as it can support caching, lazy loading and many other features.
- The developer can include native SQL statements if it was proven hard to implement the required mapping/functionality through the ORM layer.
- Using an ORM framework such as Hibernate is recommended by the Open Web Application Security Project (OWASP) [57] to avoid common Web application attacks such as Remote SQL Injection, as it provide proper filtering before querying the database.
- Finally, ORMs are vendor independent and can work with a large number of databases, which increases the portability of the application.

This project has used the Java Persistence API standard from Sun's EJB3 specification [50], the JPA is considered by many as a big step in the right direction compared with EJB2 entity beans. The standard uses metadata to describe the object to relational mappings; this metadata can either be in XML configuration files or embedded in the code using the new Java 5 annotation feature. Hibernate [47], a very popular and widely used ORM framework provide an implementation for the JPA and adds many more features making it the best choice for the ORM layer.

When mapping objects into relational entities the author needed to consider the concept of referencing in the object oriented world. For example in the case of One to Many E-R mapping it might be looked as Many to One from object oriented side. This arises from the fact that when loading entities from the database and mapping these into objects a whole object hierarchy will need to be loaded and constructed. So it is crucial to decide on choosing the main object which will be loaded and if the entire objects it is referencing (entire object graph) will also be constructed.

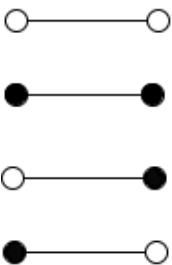
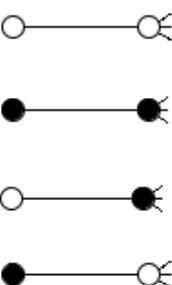
Considering for example the StaffMember object (Figure 45), this object has One to One, One to Many, Many to Many and most importantly Many to One reference with other objects. The Many to One mapping is as a result of looking at the relation from the StaffMember point of view, but in database terms Many to One is simply a One to Many relation as there is no concept of Navigability in database term. Having said this, the relational model can be checked against user transactions using the transaction pathways technique outlined by Connolly and Begg [8]. However, in the application's case this is not required as the

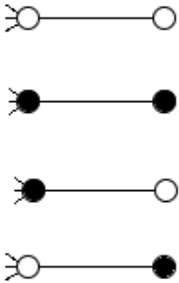
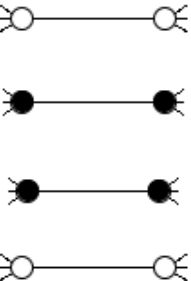
database model was designed from the entities in the class diagram, which already supports the user transactions.

The author has devised an approach that can be used to map relations from the E-R model into JPA annotations which is summarised in Table 20 below. This approach was then followed to map the entity classes into the developed data model using JPA annotations. All the Many to Many relationships were resolved during the logical database design stage by introducing a third dependent relation, and hence there was no need to use @ManyToMany JPA mapping.

It was also revealed that cascade annotation for foreign key updates on the parent table should not be included in the child entity as this will result in the deletion of the parent row whenever a child row is deleted. It was clear that this constraint should be implemented in the database schema and not included in the JPA annotation as it does not have the desired effect.

Table 20 – ER-Model and Relational Mode to JPA and Hibernate mappings

ER-Model	Relational Mode	JPA Annotation
Entity	Relation	@Entity
Identifier	Primary key	@Id
	Alternate Key	@UniqueConstraint(columnNames = {"staff_id", "holiday_year"})
Relationships:		
One to One 	Primary key + Foreign key mechanism – plus declaring the Foreign key as alternate key. <u>Mandatory participation condition:</u> Left side – can be achieved using a constraint. Right side – not allowing null for Foreign key.	@OneToOne <u>Declaring alternate key:</u> @UniqueConstraint(columnNames = {"staff_id", "holiday_year"}) <u>Declaring a check constraint:</u> @org.hibernate.annotations.Check(constraints = "(mileage is not null) or (amount is not null)") <u>Declaring not null:</u> @Column(nullable = false)
One to Many 	Primary key + Foreign key mechanism. <u>Mandatory Participation condition:</u> Left side – can be achieved using a constraint. Right side – not allowing null for Foreign key.	@OneToMany annotation created in the owning class <u>Declaring a check constraint:</u> @org.hibernate.annotations.Check(constraints = "(mileage is not null) or (amount is not null)") <u>Declaring not null:</u> @Column(nullable = false) <u>Cascading:</u>

		<p>Cascading works well when annotating the owning relationship</p> <pre>@OneToMany(cascade = {CascadeType.ALL})</pre> <p>@ManyToOne annotation created in the owned class</p> <p><u>Declaring a check constraint:</u> <pre>@org.hibernate.annotations.Check(constraints = "(mileage is not null) or (amount is not null)")</pre></p> <p><u>Declaring not null:</u> <pre>@Column(nullable = false)</pre></p> <p><u>Cascading:</u> Cascading annotation should not be used in the child class, as will result in the primary key table being updated. Should be included in the database schema</p>
<p>Many to One</p> 	<p>Primary key + Foreign key mechanism.</p> <p><u>Mandatory Participation condition:</u> Left side – not allowing null for Foreign key.</p> <p>Right side – can be achieved using a constraint.</p>	
<p>Many to Many</p> 	<p>Should be resolved at the Logical database design, by resolving the M-N relationships into 3 relations</p> <p><u>Mandatory Participation condition:</u> Implemented as above for One to Many relationships on the One side of the two owning relations</p>	<p>@ManyToMany</p> <p>The expected name for the intermediate table is table1_table2 unless specified otherwise in the annotation.</p>

6.3.5 Coding practices

Code Repository

As the project code was relatively large, with many classes, packages and HTML files, it was decided to use a code repository to store the code. This was important to avoid accidental loss to the code resulting from hardware or software failure, and although only one developer was working on the project, having the code in a central repository made it easy to work on different computers.

Google code was used to host the project development (Appendix L), as it offers a project home page with downloads section, a Wiki, issues area for raising bugs and source area for checking in and out the code. Google code uses a widely known version control system called the Subversion repository. Using Google code has many advantages some of which can be summarised as follows:

- Central repository for the project code with revision history.
- Issues section where bugs can be logged and tracked.
- Web access makes it easy for other people to view or use the code.
- A Wiki page where project related help and documentations can be published.

6.3.6 Unit testing the domain model classes

After developing the Service classes for each of the packages in the OfficeMA, unit tests were written for each of the services. These unit tests focused on testing the Service, Repository and Entity classes in the domain model and ensuring that the correct logic was performed as outlined in the use cases. JUnit, a widely used Java unit testing framework was used to carryout these tests (Appendix I). JUnit framework is also integrated with the Eclipse IDE and can be run from within the Integrated Development Environment (IDE) (Figure 20).

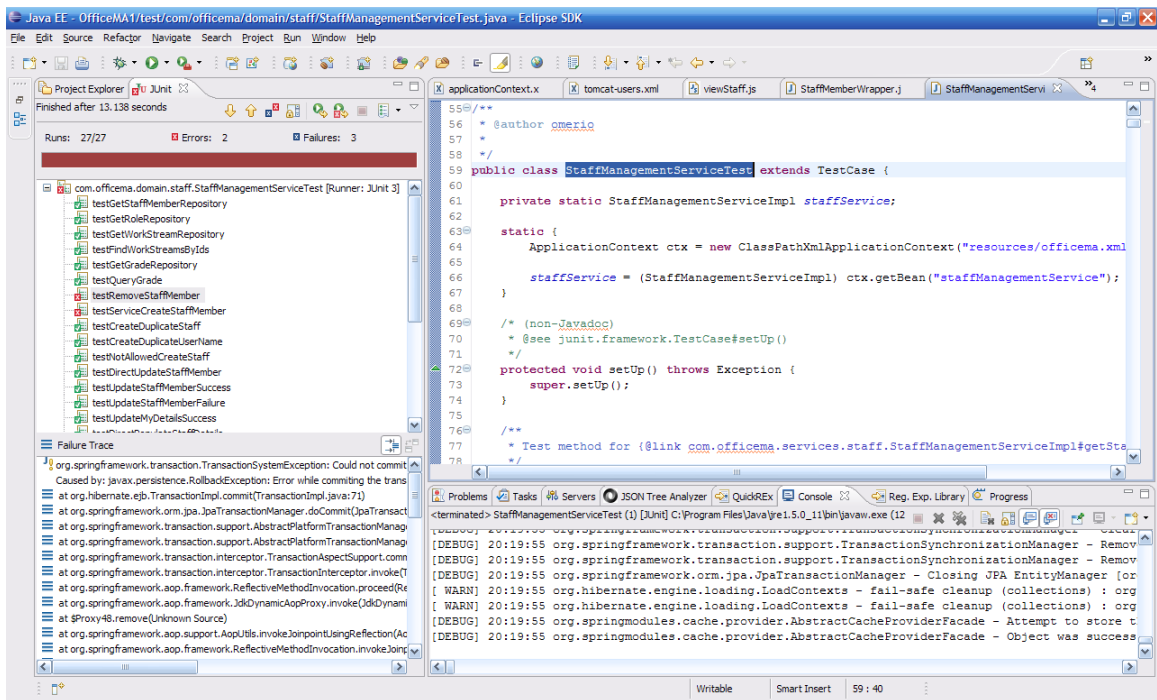


Figure 20. JUnit tests executed from within the Eclipse IDE.

6.4 *Presentation Layer*

As explained in section (2), the user interface used for OfficeMA is a Rich Internet Application that runs on the client side and communicates with the server using AJAX. This approach to designing web applications requires proper design and implementation compared to designing a standard HTML user interface. A development methodology was developed for the user interface in this project based on Crane's four defining principles of AJAX [10] as summarised below:

1. **The browser hosts an application, not content.** Hence the user interface is developed to run entirely on the client's browser and communications with the server need to be done using AJAX. As no HTML is sent to the browser after the main application is loaded the user interface has to be self sufficient and able to deal with all scenarios on the client side and only contact the server for data. For this to be achieved the client application followed the Model-Viewer-Controller design pattern.
2. **The server delivers data, not content.** This means that the server only sends the contents to the application once and any transfer thereafter will be of pure data. That server should be able to convey data and status messages to the client, and the transfer of data should be done in a lightweight protocol that can easily be constructed as objects on the client side.
3. **User interaction with the application can be fluid and continuous.** The user interface should be rich and use visual widgets that enhance the user productivity and experience. This can be achieved by using widgets such as buttons, menus, tabs and dialogues where appropriate (Table 1)
4. **This is real coding and requires discipline.** This is a crucial point for an application that will be running entirely from the browser on the client side and hence the project has seriously considered many issues such as security, error handling, session management and other functionality.

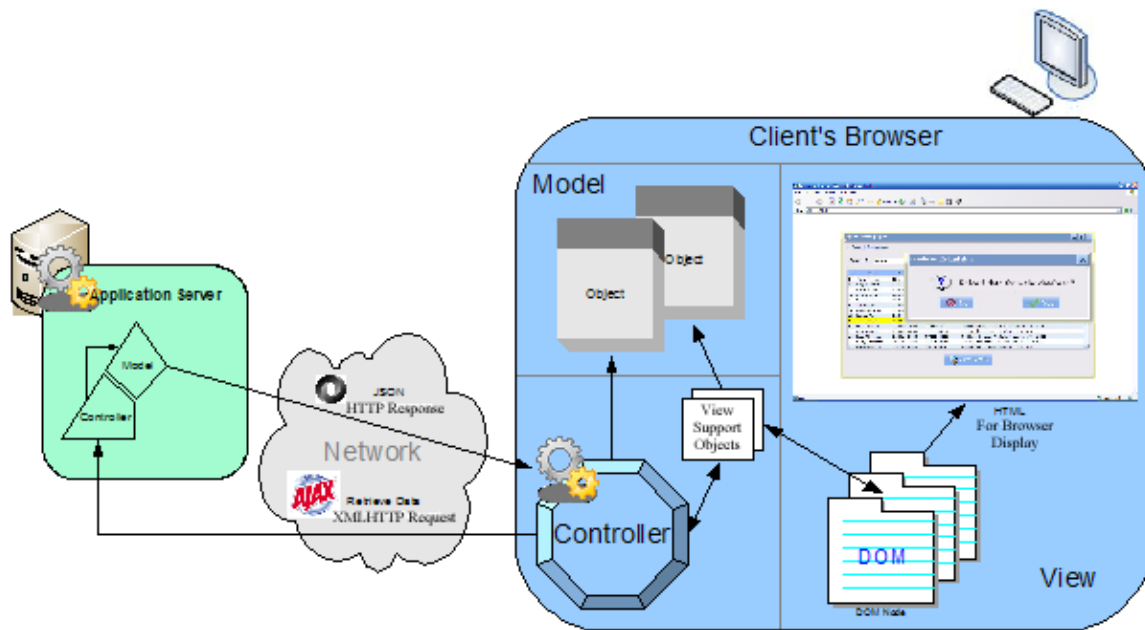


Figure 21. Office Management Application Presentation layer

6.4.1 Presentation Layer logic and data formatters

The presentation layer logic shown as a controller and a model in the Figure 21 above was required to mediate between the business layer and the user interface. This layer also carries out data formatting to suit the presentation technology being used. For this project Struts 2 [63], a widely used and popular web Model-View-Controller framework was used. Struts is build on the top of the Java Servlet and JavaServer Pages technology, it offers many features and enhancement to mainstream web development.

Struts 2 framework

Struts 2 was used for the following reasons:

- Struts 2 supports validation of user input, this validation can be client or server side and is recommended by the Open Web Application Security Project (OWASP) [57] to avoid common Web application attack to do with input validation and type conversion.
- Struts 2 automatically type converts and populate all the string values of the request parameters into the instance variables of the Java class handling the HTTP request.

- Struts 2 enables the developer to use a single class to act as a controller and handle various types of web requests by using a method for each business operation.
- Struts 2 offers the ability to work straight from the business model without the need to create intermediary objects.
- Struts 2 integrates with the Spring framework well.
- Struts 2 defines a concept of interceptors which is similar to the J2EE Filters. Interceptors offer the chance to create custom data formatter to format the data returned on the HTTP response into XML, JSON, etc...

The implementation strategy for this presentation logic layer was to write a class, termed an Action class to handle a number of operations. Validation information for the input for this class was also defined in XML configuration files. The output result of the Action class execution was formatted into the JavaScript Object Notation (JSON) [52] format using the Struts 2 JSON plug-in [62].

JavaScript Object Notation (JSON)

JSON is a string representation of JavaScript objects; it's lightweight and shorter than XML or HTML. JSON strings can be converted into JavaScript object using the **eval** JavaScript function as shown in Figure 22 below.

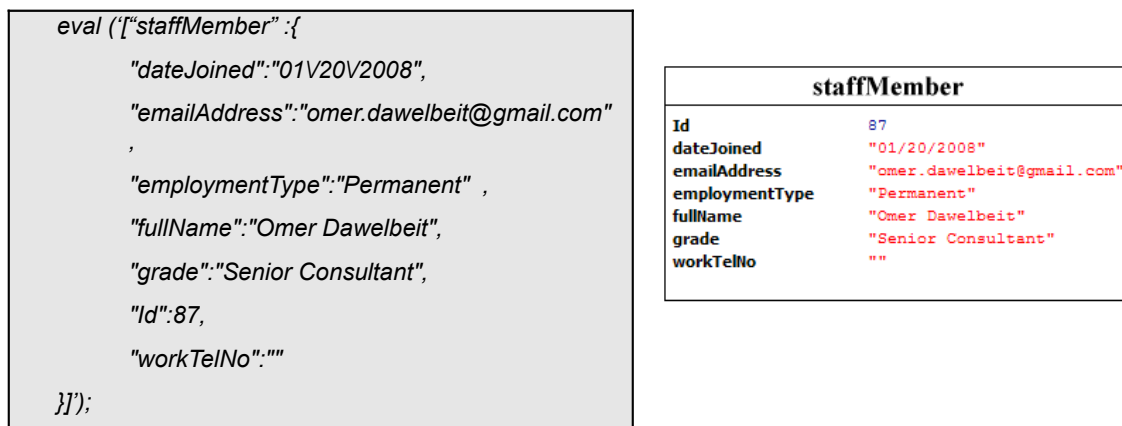


Figure 22. Using JavaScript eval function on a JSON string to create a JavaScript Object

This mechanism was used to transport data from the presentation logic layer to the controller on client side user interface (Figure 21). The following methodology was devised and use when developing the presentation logic layer components to retrieve or process data for the user interface:

- For the functionality required by the user interface, for example query staff details, define an Action class with a method name “query”
- Write validation criteria for the Action class input parameter from the user interface.
- Check the domain model entity can be used directly or a Wrapper object is required, for example to restrict or change the type of some of the fields
- Define the Action class results as JSON type so that Struts 2 can automatically serialize the Java objects into JSON string.
- Return a result status object to indicate the status of execution of the Action class. As shown below the result status object contains a message and a status, the status is either S for success or E for error. This is used by the client side user interface to determine the appropriate action.

```
["resultStatus":{"message":"Staff members details queried successfully","status":"S"}]
```

6.4.2 Graphical user interface

The user interface for the application is transferred to the client only once when the user authenticates successfully to the application. The UI then resides on the client's browser and handles itself according to the actions of the user as shown in Figure 21. This means that the user interface will need to be properly tested on the list of supported browsers and any issues resolved before hand.

The user interface follows the Model-View-Controller pattern and was implemented using Dojo, a rich DHTML, AJAX enabled toolkit [42]. The Dojo toolkit streamlines JavaScript development by providing custom rich widgets (Table 1), many utility functions and an AJAX library. Since JavaScript can be used as an object oriented language based on prototyping, the Dojo toolkit provide a Java like style to defining objects and inheritance in JavaScript. This approach was used for all the JavaScript objects that comprises the user interface.

Widget objects defined in Dojo can also have a HTML template and linked to Document Object Model (DOM) nodes. The widget object then manipulates the

HTML nodes to display information or responds to user's actions such as button clicks, etc....

A new approach to web user interface development

The approach followed to develop the rich HTML user interface is not widely used for Web applications although the process used is deeply rooted in the USDP and was used for a long time to model user interface for desktop applications developed in languages such as Java and Visual Basic. Having said this, the fact that JavaScript supports object orientation, it is theoretically possible to use the same methodology outlined by Bennett et al [4] to design the user interface for Rich Internet Applications.

Bennett et al [4] summarised the steps used in designing the boundary classes as follow:

- Prototyping the user interface.
- Designing the classes.
- Modelling the interaction involved in the interface using interaction or communication diagrams.
- Modelling the control of the interface using state machines.

The client side View

The view component on the client side user interface (Figure 21) consists of DOM nodes and JavaScript Widgets. The HTML in the view is automatically constructed in the browser as DOM nodes. The JavaScript widgets use JavaScript to control a number of DOM nodes that make up the widget. JavaScript widgets were used for complex user interface components such as widgets that control other widgets to perform a specific operation.

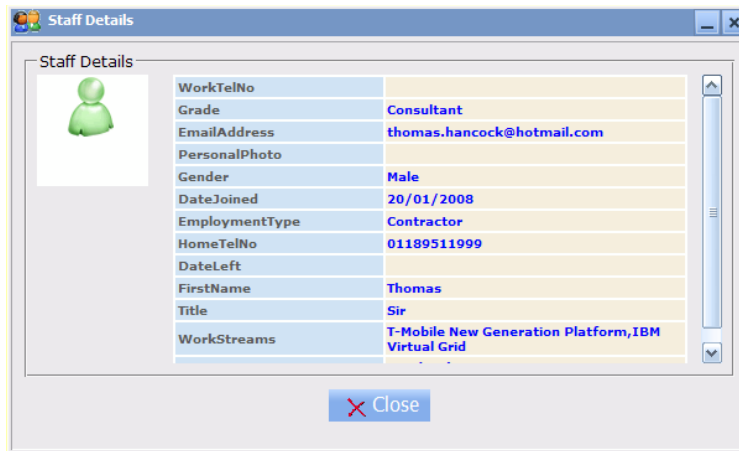


Figure 23. View Staff Details dialogue widget.

For example dialogue boxes widgets contain and control a number of other widgets as shown below for the “Staff Details” widget (Figure 23). Each JavaScript widget consists of the following:

1. JavaScript object, the author used the term View Support Object (VSO) to describe these JavaScript Objects as they control the DOM nodes attached to them (Figure 21).
2. DOM nodes to represent the HTML template for the widget.
3. Other child widgets as widgets can be nested.

The high level container widgets in the user interface such as floating windows and dialogue boxes were implemented using the following approach:

- Define the widget template which consists of HTML, CSS and other widgets, this is defined in a HTML file e.g. **viewStaff.html**. This will later on be constructed as DOM nodes in the browser.
- Define the View Support Object (VSO) for the widget in a JavaScript file, e.g. **viewStaff.js** using a JavaScript class definition from the Dojo toolkit [43] as shown in Figure 24 below.

To define a widget class (supports Mixins, multiple inheritance):

```
dojo.widget.defineWidget("ClassName", [SuperClass1, SuperClass2, ...], {
    property1: "",
    property2: "",
    method1: function() {
        // method code here
    }
})
```

To define a non-widget class (supports Mixins, multiple inheritance):

```

dojo.declare("ClassName",[SuperClass1, SuperClass2, ...], {
    property1: "",
    property2: "",
    // acts as a Java constructor
    initializer : function(urls) {
        this.urls = urls;
        this.user = "";
        this.ajaxTimeOut = 600; // 600s, 10mins to wait for ajax
                                // calls
    },
    method1: function() {
        // method code here
    }
}

```

Figure 24. JavaScript classes declarations in Dojo

- Implement the methods required for the VSO to control the DOM nodes. These methods follow from the user interface models such as boundary classes and interaction diagrams as shown in Figure 25 and 26 below. For the **StaffView** widget a boundary class diagram and an interaction diagram were developed to model the functionality of the **ViewStaff** JavaScript widget.

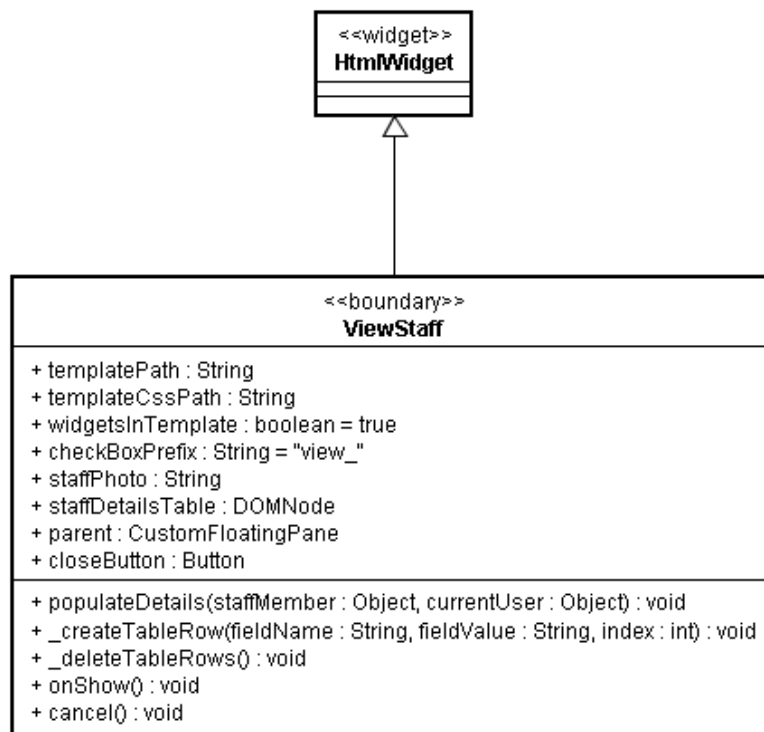


Figure 25. Boundary class diagram for ViewStaff widget

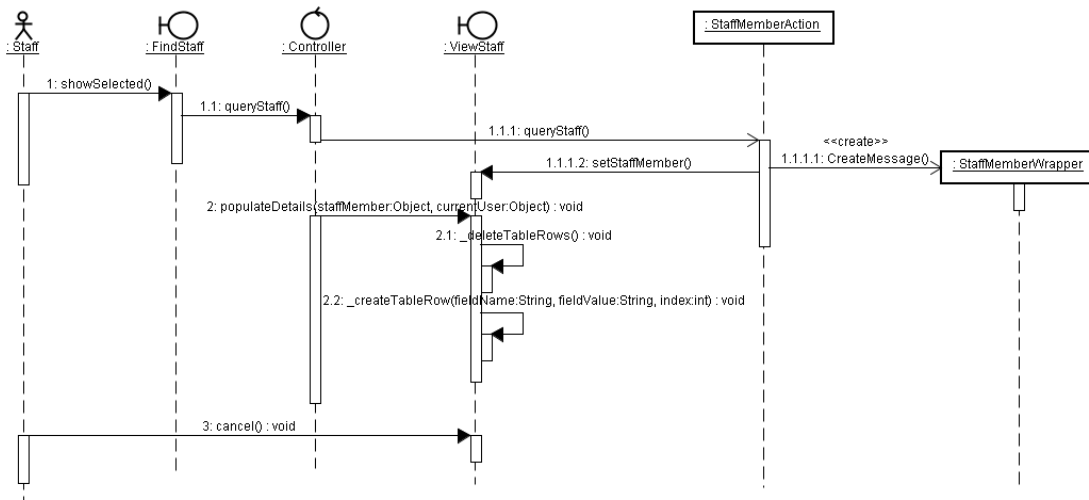


Figure 26. Sequence diagram for the ViewStaff widget

The client side Model (Data)

On the server side the domain model contains the business objects these are then transferred as data to the client side in JSON format. These objects will then be built into client side JavaScript objects as shown above using the 'eval' function. This way the model on the client side will reflect the objects on the server side and there is no need to create class definition for the model on the client side as this is done dynamically during the running of the application.

An example to demonstrate this is the sequence diagram shown Figure 24 above, the server side object StaffMemberWrapper is serialized into JSON format and delivered to the client side; consequently the same object graph will also exist on the client side. The main advantage of this approach is the fact that there is no need to maintain two set of class definitions for StattMember on the client and server side, as the server side definition can be used in both layers.

The client side Controller

The controller (Figure 21) orchestrates the view, the model and facilitates communications with the server to retrieve and save data using AJAX. The controller does the following:

- Holds reference to and controls all the JavaScript widgets.
- Queries data from the server using AJAX and reconstructs the response into JavaScript model objects.
- Saves data to the server from the updated model.

- Enforces security and authorisation on the client side. The same enforcements are done on the server side to ensure no JavaScript tampering was done.

6.5 Overall System

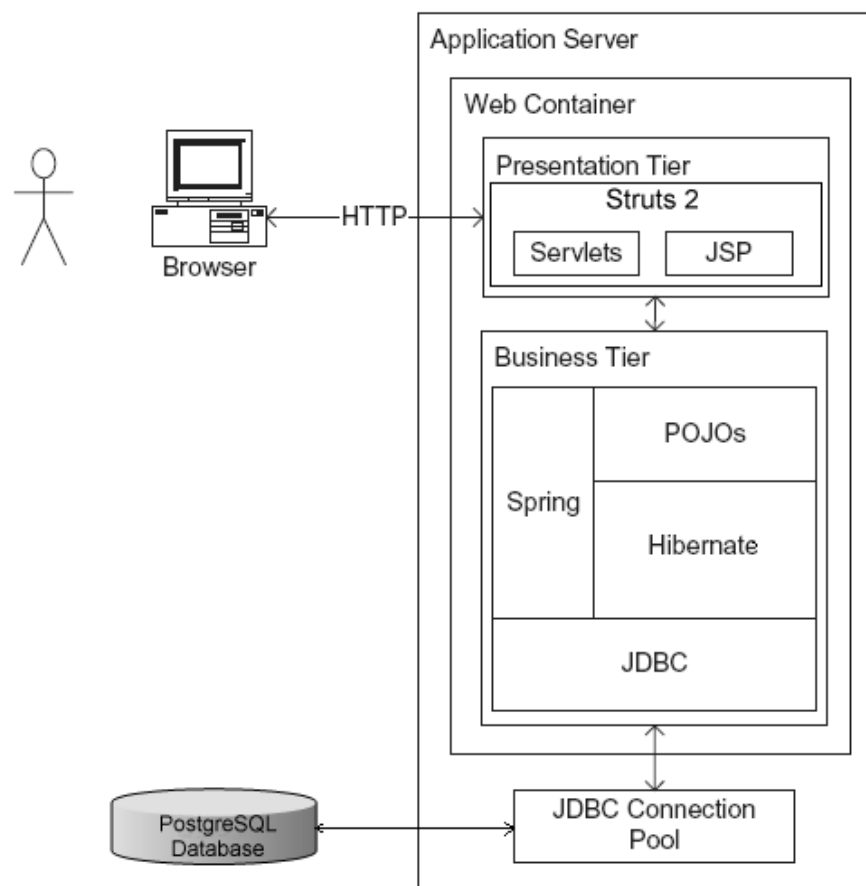


Figure 27. OfficeMA candidate technologies. Adapted from Richardson [30].

Figure 27 above provides an overview of the initial candidate technologies for the Office Management Application and the tier on which they are used. The specific versions used in for the project are summarised in Appendix I. As explained before one of the objectives of the project is to use lightweight Open Source frameworks to add transaction, security and persistence to the application domain model developed using the USDP.

7 Detailed Software Design

The detailed software design followed from the analysis classes by elaborating on the class associations and their multiplicity. Detailed design was also concerned with the specification of the attribute types, how operations function and how objects interact with each other. Other aspects such as object visibility, the use of Interfaces and the use of Design Patterns were also applied. Other areas that were considered during the detailed design are:

- User interface
- Data management

The detailed design was carried out taking into consideration the following candidate technologies chosen for the implementation:

- Java 5 is used to implement the application logic. Other widely used Java frameworks were used such as Hibernate, Struts and Spring. Unit testing was implemented using the JUnit framework.
- DHTML, JavaScript, CSS and the Dojo toolkit is used to implement the user interface.
- PostgreSQL is used to implement the database layer.

7.1 Detailed Class Design and Implementation

The source code developed as part of the detailed design and implementation is included in the CD-ROM attached with this dissertation as summarised in Appendix H. The Java packages structure for the source code is shown in Figure 28 below.

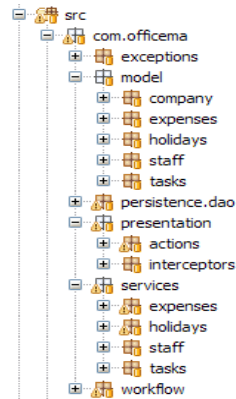


Figure 28. Package structure for OfficeMA detailed classes

7.1.1 Design and architectural patterns

Detailed class design was based on the POJO architectural pattern as explained in subsection (6.3) and the use of the Data Access Object (DAO) design pattern [30]. The advantage of using the DAO pattern was to isolate the business logic from the details of the persistence layer implementation.

The class diagram below shows the DAO pattern and how the various classes interact to provide the necessary persistence isolation. In regards to this project the following terms were used to refer to the classes in the DAO pattern:

- **Service** – Represent the business object that carries out some business logic. The Service also represents a Control class in the USDP, so for each package a Service class was used to support the use cases in that package. (5.3).
- **Repository** – Represent the `DataAccessObject` used to isolate the Service from the implementation details of the persistence layer.
- **EntityManager** – is the persistence manager for the Java Persistence API specification (`javax.persistence.EntityManager`) [51]. This is declared and annotated using the `@PersistenceContext` (`javax.persistence.PersistenceContext`), but not instantiated in the DAO as it is injected by the Spring framework as a result of the declared annotation. The `EntityManager` class provide methods to perform CRUD operations on entities and can also be used to execute SQL queries, which can be in native SQL or JPA query language [50].

- **Model** – Represent the TransferObject or the entity being retrieved or persisted. As discussed in subsection (6.3.4) the entity classes are annotated with JPA mappings to map them to the physical database schema.

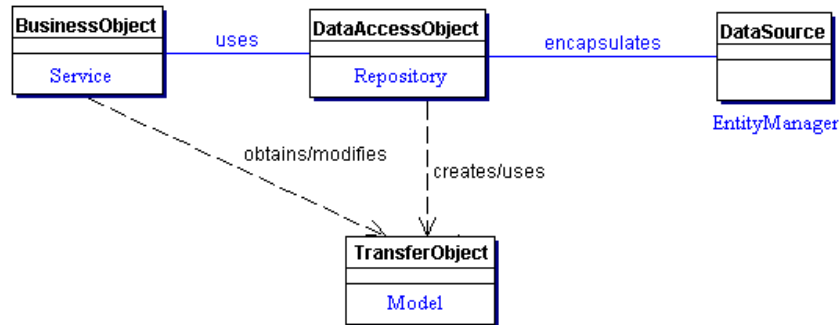


Figure 29. Data Access Object Class Diagram [30].

Below the detailed design for each of the packages in the Office Management Application is considered. A service class was used in each package to support the use case. Repositories were implemented for each class that required persistence and is the main focus of the use case logic. Classes that are persisted as part of these main classes will not have their own repositories as they automatically persisted as part of the object graph for main entities.

7.1.2 Enumerated types

One of the new features in Java 5 the enumerated type [35] was used in the OfficeMA implementation. Most of the classes that are of type Enum ended with the word Type, such as `ExpensesStatusType`, `HolidayStatusTypes`, etc.... Enumerated types are shown in the detailed class diagram as having “enum” stereotype. Enumerations in Java have many advantages such as having a name and an ordinal similar to the C++ counterpart, however the Java enumeration are far powerful and can have static methods and can also be used in switch statements as shown for Enum `OperationType` implemented as part of the staff management module:

```

public boolean isOperationPermitted(OperationType operation) {
    boolean permitted = false;
    switch (operation) {
        case VIEW_PARTIAL_STAFF_DETAILS:
        case UPDATE_OWN_DETAILS:
    }
}

```

```

        case UPDATE_PARTIAL_STAFF_DETAILS:
        case VIEW_OWN_DETAILS:
        case VIEW_OTHERS_DETAILS:
            permitted = true;
            break;
    }
    return permitted;
}

```

7.1.3 Dependency Injection using Spring

As discussed in subsection (6.3.2) the Spring framework was used as a container for the OfficeMA classes, one of the features discussed was the DI used to provide objects with their dependencies at runtime. This feature is in fact just a drop in the ocean compared to the overall features and capabilities offered by the Spring framework. Without using DI, a great deal of boilerplate code would be needed to instantiate objects at runtime. The code would have also been tightly coupled because classes need a concrete class to instantiate which renders the use of interfaces pointless in this case.

The Spring configurations, termed bean configurations are declared in the file “/OfficeMA/WebContent/WEB-INF/applicationContext.xml”. This file includes the declaration of all the beans (classes) that Spring is going to create at runtime and also their wiring configurations. Most importantly the data source configurations such as the database server type, hostname, username, password and connection pooling are all configured in this file. Figures 30 below shows the beans configurations for OfficeMA classes.

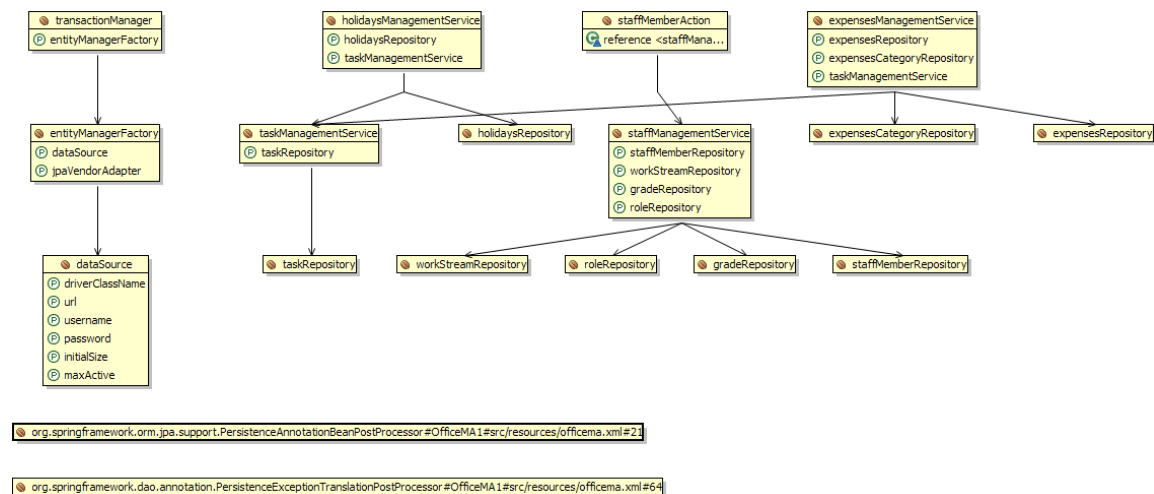


Figure 30. Spring beans schematic for OfficeMA classes

7.1.4 The use of Exceptions

Exceptional conditions in the OfficeMA code were handled using Java Exceptions. Exceptions were thrown in the code to disrupt the normal program flow and to indicate to the caller that an error or some unexpected condition or state has occurred. The business logic layer was programmed so that calling method incorrectly will result in an unchecked runtime exception to be thrown such `java.lang.IllegalArgumentException`, this happens when the developer of the calling code invoke the business layer methods with invalid parameters. Other validation exceptions such as exceptional condition that might happen at runtime as a result of invalid data or state are thrown as checked exceptions so that the caller can handle these and take the appropriate action accordingly.

Implementation details

The application Exception classes were implemented using the following classes including fully qualified package names:

- `com.officema.exceptions.AccountLockedException.java`
- `com.officema.exceptions.DetailsRetrieveUpdateException.java`
- `com.officema.exceptions.InvalidPasswordException.java`
- `com.officema.exceptions.InvalidUsernameException.java`
- `com.officema.exceptions.PermissionException.java`
- `com.officema.exceptions.ValidationException.java`

7.1.5 Generics and Parameterized Classes

A generic repository (DAO) super class and interface were used to provide generic functionality for CRUD (Create, Read, Update and Delete) operations for the domain model entities (Figure 31). The super class uses the new Java 5 Generic feature [45], which is similar to the C++ templates, and offers better readability and compile-time type checking for Collection classes.

For example in the case below Generics made it possible to create a super class and an interface that can be used to retrieve an unknown entity type. This was not possible to achieve without Generics as the user has to either type cast entities or get all entities to implement the same Interface or super class or write multiple super classes, one for each repository. *“Generics are implemented by type erasure: generic type information is present only at compile time, after which it is erased by the compiler”* [6].

As seen below the Generic Java concept is modelled as class parameters in UML. The OMG UML specification [69] states:

"A template is a parameterized element that can be used to generate other model elements using TemplateBinding relationships. The template parameters for the template signature specify the formal parameters that will be substituted by actual parameters (or the default) in a binding. "

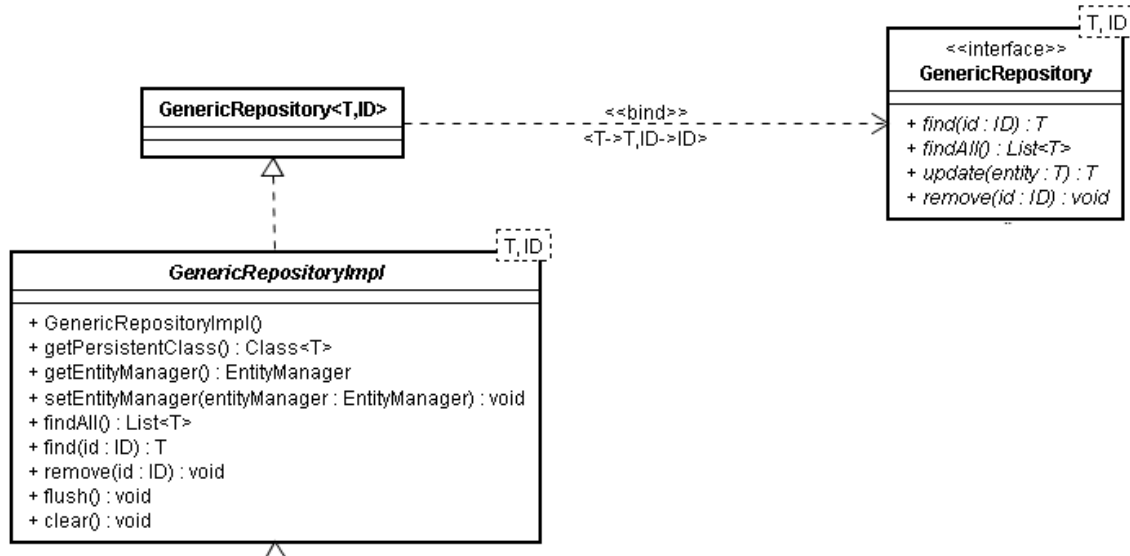


Figure 31. Generic repository super class and interface for CRUD operation

Implementation details

The generic repository classes were implemented using the following classes including fully qualified package names:

- com.officema.persistence.dao.GenericRepository.java
- com.officema.persistence.dao.GenericRepositoryImpl.java

7.2 Application Packages

The detailed class design and the implementation for the following application packages are outlines below. These packages include:

- Company package
- Expenses package
- Holidays package
- Staff package
- System Settings package
- Task package

7.2.1 Company package:

The Company package contains classes such as WorkStream, Project and Grade. These classes were put on a separate package as they belong to the whole company and might be used in other packages. The class diagram below (Figures 32, 33) outlines the detailed classes for the Company package and contains two repositories as follows:

- WorkStreamRepositoryImpl – responsible for saving and retrieving WorkStream and Project from the database.
- GradeRepositoryImpl – responsible for saving and retrieving Grade from the database

Project class does not have its own repository as it is retrieved from the database as part of the WorkStream. This is shown in the class diagram using the composition association to indicate that Projects are part of WorkStreams and can not exist in isolation.

Implementation Details

The Company package classes were implemented using the following classes including fully qualified package names:

- com.officema.model.company.Project.java
- com.officema.model.company.WorkStream.java
- com.officema.persistence.dao.jpa.GradeRepositoryImpl.java
- com.officema.persistence.dao.jpa.WorkStreamRepositoryImpl.java
- com.officema.persistence.dao.GradeRepository.java
- com.officema.persistence.dao.WorkStreamRepository.java

7.2.2 Expenses package:

The Expenses package contains classes such as Expenses, ExpensesCategory, ExpensesItem, ExpensesMnemonic, ExpensesStatus, ExpensesStatusType and MileageCost. As the expenses transition between a number of states, this transition was modelled using a state diagram which is included in Appendix D and was explained in details in the user guide for the application included in Appendix H. The class diagram below outline the detailed classes for the Expenses package and contains two repositories and a service as follows:

- ExpensesCategoryRepositoryImpl – responsible for saving and retrieving Expenses, ExpensesItem and ExpensesStatus from database.
- ExpensesCategoryRepositoryImpl – responsible for saving and retrieving ExpensesCategory from the database

Although the Expenses class depends completely on the StaffMember class as indicated by the composition association in the class diagram below, it is retrieved separately using the ExpensesRepository. This is achieved by including the reference to the StaffMember in the Expenses object instead of referencing a collection of Expenses from the StaffMember object.

The advantage of this strategy is that when loading the StaffMember object in staff management functions the expenses for the staff member are not relevant and are not loaded from the database to increase performance. Only when expenses are loaded then the staff member instance for these expenses will be loaded as well. To achieve this a @ManyToOne JPA annotation was used in the Expenses class to reference the StaffMember instance.

Expenses management service

The ExpensesManagementServiceImpl class provides an implementation of all the business logic that is required to satisfy the various expenses management use cases through the methods shown in Figure 35. The service also provides the interface ExpensesManagementService for clients use to hide the implementation. The service class makes use of other repositories and services such as the ExpensesRepository, ExpensesCategoryRepository and the TaskManagementService to retrieve and update expenses related entities. These private instance variables are not instantiated by the service instead these are injected using the Spring framework at runtime using Dependency Injection (Figure 33).

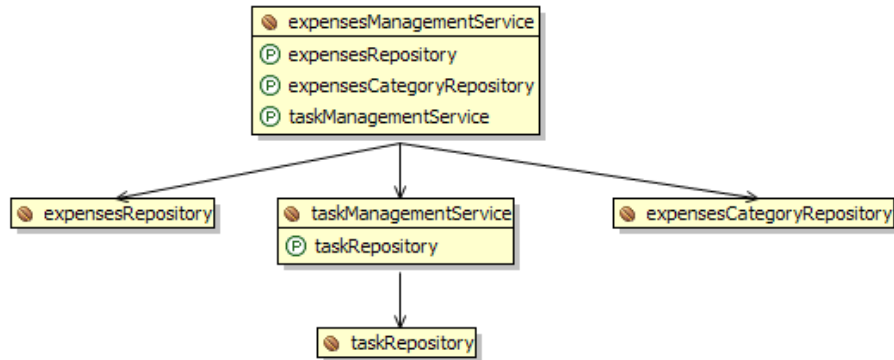


Figure 34. Spring beans schematic for ExpensesManagementServiceImpl

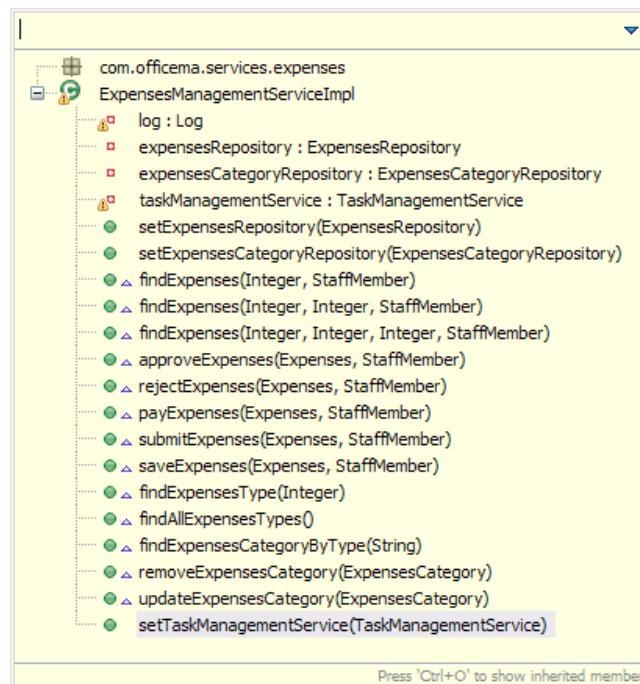


Figure 35. Methods defined by the ExpensesManagementServiceImpl

Implementation Details

The Expenses package classes were implemented using the following classes including fully qualified package names:

- com.officema.model.expenses.types.ExpensesStatusType.java
- com.officema.model.expenses.Expenses.java

- `com.officema.model.expenses.ExpensesCategory.java`
- `com.officema.model.expenses.ExpensesItem.java`
- `com.officema.model.expenses.ExpensesMnemonic.java`
- `com.officema.model.expenses.ExpensesStatus.java`
- `com.officema.model.expenses.MileageCost.java`
- `com.officema.persistence.dao.jpa.ExpensesCategoryRepositoryImpl.java`
- `com.officema.persistence.dao.jpa.ExpensesRepositoryImpl.java`
- `com.officema.persistence.dao.ExpensesCategoryRepository.java`
- `com.officema.persistence.dao.ExpensesRepository.java`
- `com.officema.services.expenses.ExpensesManagementService.java`
- `com.officema.services.expenses.ExpensesManagementServiceImpl.java`

7.2.3 Holiday package:

The Holiday package contains classes such as Holiday, HolidayYear and HolidayStatusType. The class diagram below outline the detailed classes for the Holidays package and contains a repositories and a service as follows:

- HolidaysRepositoryImpl – responsible for saving and retrieving Holiday, HolidayYear and HolidayStatusType from database.

As mentioned above for Expenses class the same applies to the HolidayYear class in the fact that it depends on the StaffMember class as indicated by the composition association in the class diagram below, but it is retrieved separately using the HolidaysRepository. This was achieved by including the reference to the StaffMember in the HolidaysYear object instead of referencing a collection of HolidaysYear from the StaffMember object.

Holiday management service

The HolidaysManagementServiceImpl class provides an implementation of all the business logic that was required to satisfy the various holidays management use cases through the methods shown below in Figure 38. The service also provides the interface HolidaysManagementService for clients use to hide the implementation. The service class makes use of other repositories and services such as the HolidaysRepository and the TaskManagementService to retrieve and update holidays related entities. These private instance variables are not instantiated by the service instead these are injected using the Spring framework at runtime using Dependency Injection (Figure 37).

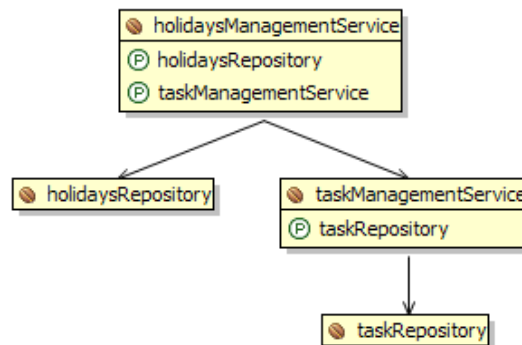


Figure 37. Spring beans schematic for HolidaysManagementServiceImpl

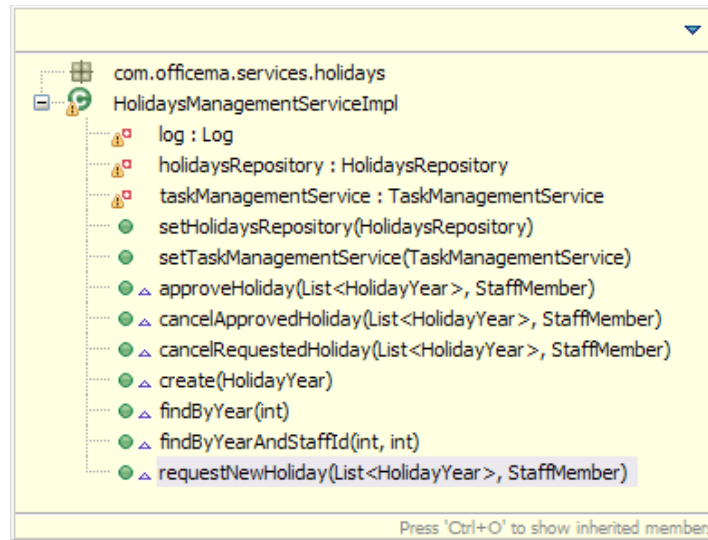


Figure 38. Methods defined by the HolidaysManagementServiceImpl

Implementation Details

The Holidays package classes were implemented using the following classes including fully qualified package names:

- com.officema.model.holidays.types.HolidayStatusType.java
- com.officema.model.holidays.Holiday.java
- com.officema.model.holidays.HolidayYear.java
- com.officema.persistence.dao.jpa.HolidaysRepositoryImpl.java
- com.officema.persistence.dao.HolidaysRepository.java
- com.officema.services.holidays.HolidaysManagementService.java
- com.officema.services.holidays.HolidaysManagementServiceImpl.java

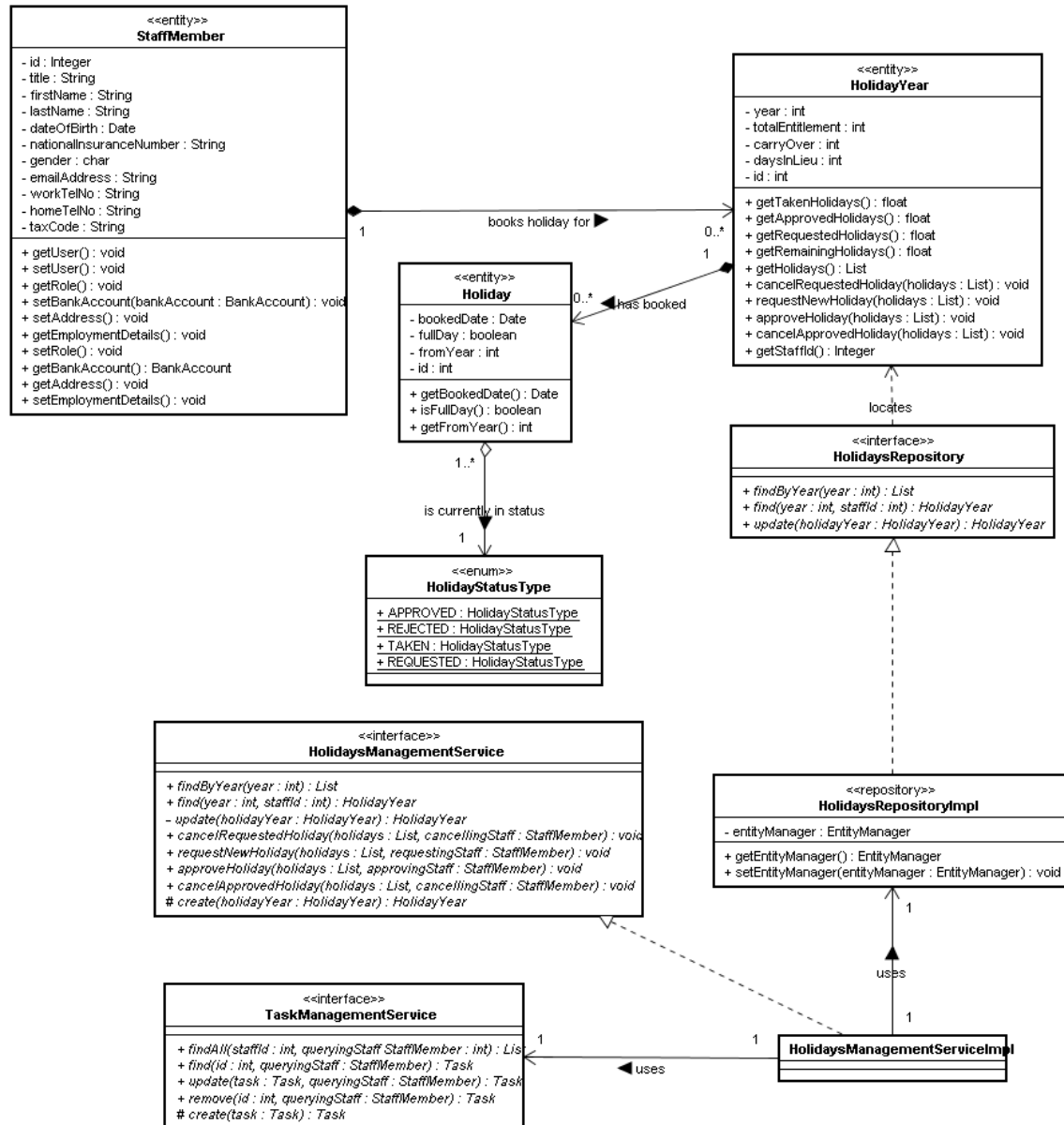


Figure 39. Holidays classes and dependencies (Generic repository classes omitted for clarity)

7.2.4 Staff package:

The Staff package is the largest and the most important package in the application. The package is centric around the StaffMember class, but also include other entity classes such as Grade, Accountant, Administrator, GenericRole, RegularStaff, Role, EmploymentType, Gender, OperationType, Responsibility, ResponsibilityType, RoleType, Title, Address, BankAccount, EmploymentDetails, StaffMember, User. The class diagram in Figure 38 outlines the detailed classes for the Staff package and contains a number of repositories and a service as follows:

- StaffMemberRepositoryImpl – responsible for saving and retrieving Address, BankAccount, EmploymentDetails, StaffMember from database.
- RoleRepositoryImpl – responsible for saving and retrieving GenericRole subclasses Accountant, Administrator, RegularStaff.

Staff members' roles

Roles the staff package is modelled using a number of objects as shown in the diagram below. The staff roles are defined using an interface (Role) that clients can reference, then a generic abstract super class is defined (GenericRole), this class defines all the common functionality required by the various roles. Three subclasses are defined to inherit from the GenericRole super class, these are: Administrator, Accountant and RegularStaff. The role class define the same methods signature, but each class implements these methods differently depending on what each role can do.

This role hierarchy was modelled in the conceptual database design using the enhance ER entity subtypes feature [8]. When mapping these to the database a single table for the GenericRole class hierarchy was used. As Baur and King [2] have outlined that this mapping strategy is the winner in terms of simplicity and performance, and the fact that all the classes contain the same attribute there will not be a problem with some columns holding null for the attributes of a specific role class. The concrete class represented by a particular row is identified by a discriminator column [2] using the JPA annotation in the code snippet below:

```
@Entity
@Table(name = "Roles")
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(
    name="role_type",
    discriminatorType=DiscriminatorType.STRING
)
public abstract class GenericRole implements Role, Serializable {
```

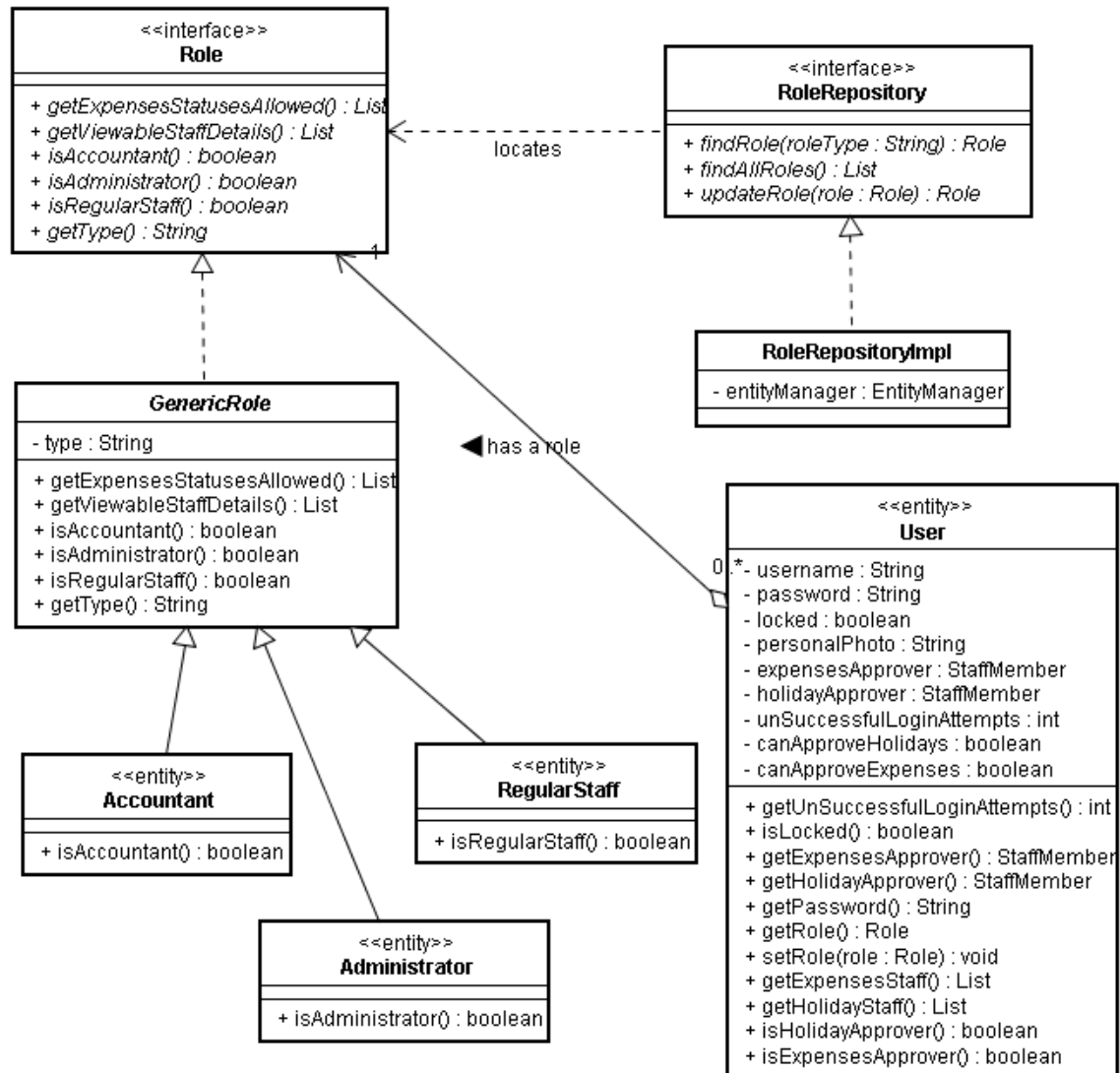


Figure 40. Role classes and dependencies (Generic repository classes omitted for clarity)

Staff management service

The `StaffManagementServiceImpl` class provides an implementation of all the business logic that is required to satisfy the various staff management use cases through the methods shown below in Figure 42. The service also provides the interface `StaffManagementService` for clients use to hide the implementation. The service class makes use of other repositories such as the `WorkStreamRepository`, `RoleRepository`, `GradeRepository` and `StaffMemberRepository` to retrieve and update staff related entities. These private instance variables are not instantiated by the service instead these are

injected using the Spring framework at runtime using Dependency Injection (Figure 41).

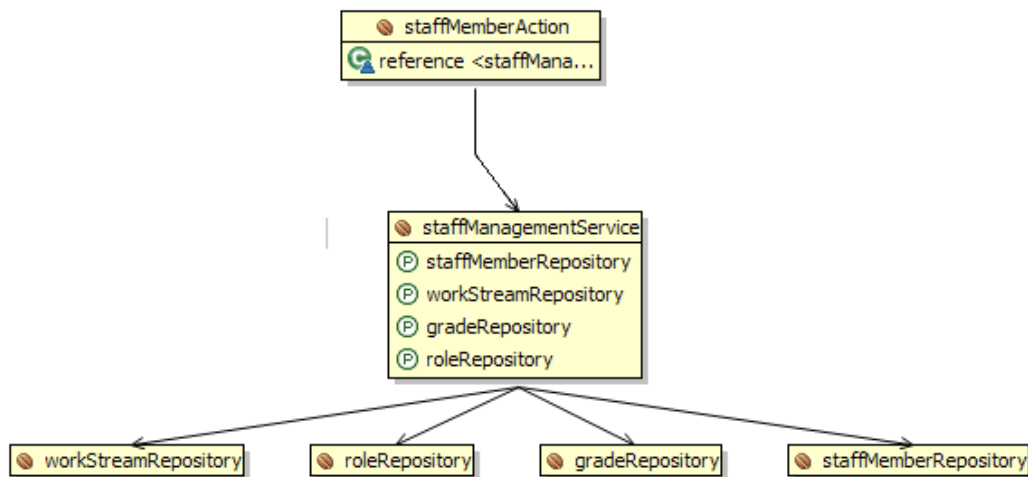


Figure 41. Spring beans schematic for StaffManagementServiceImpl

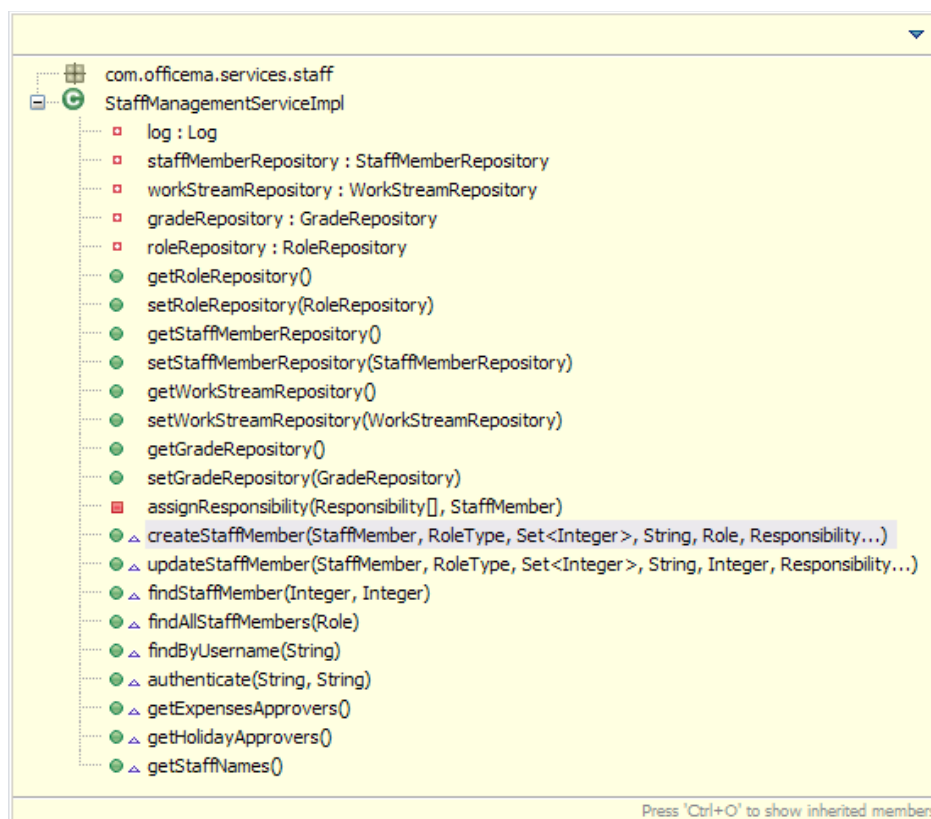


Figure 42. Methods defined by the StaffManagementServiceImpl

Create staff sequence diagram

Figure 43 below show the sequence diagram for the createStaffMember method

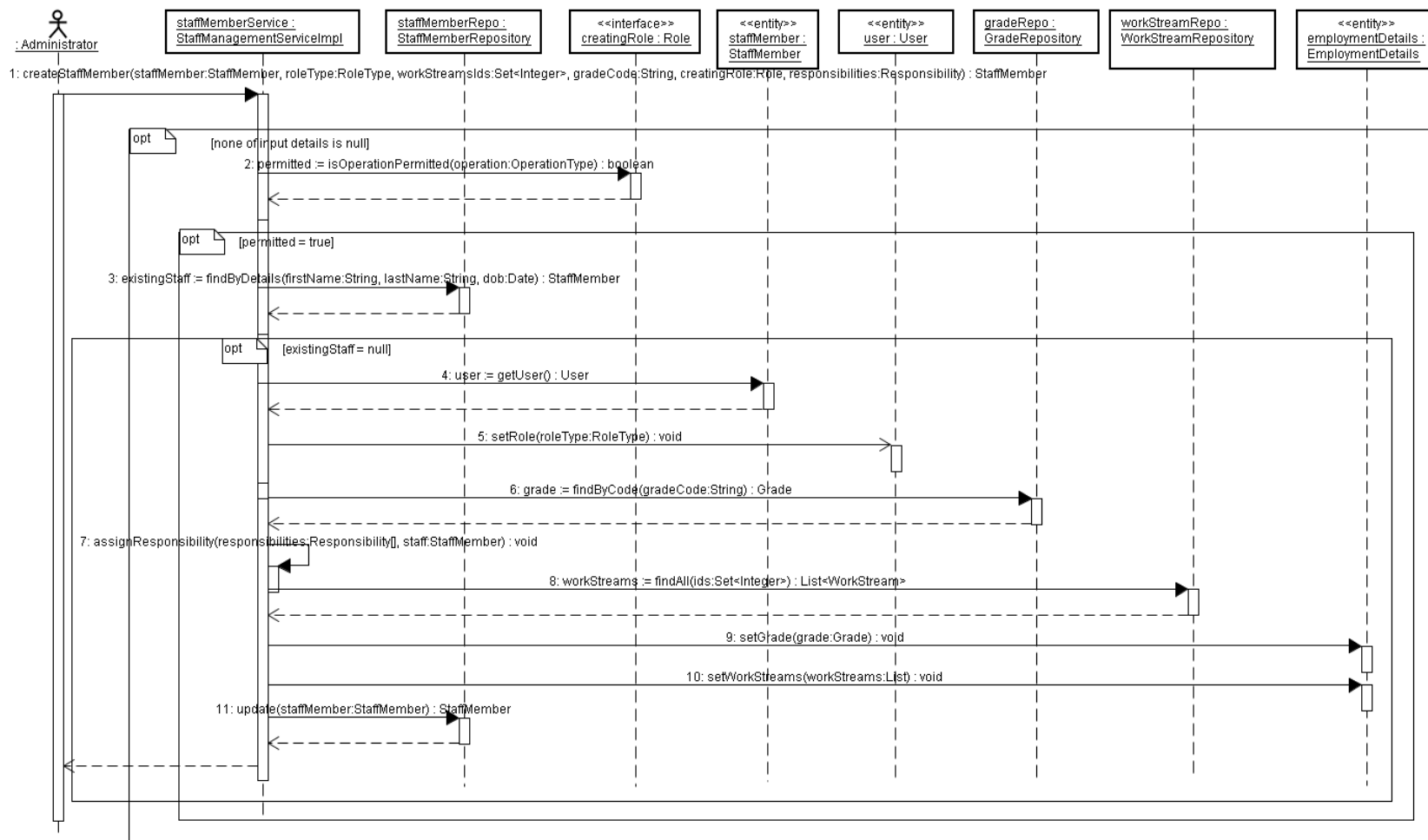


Figure 43. Sequence diagram for the createStaffMember

Authenticate staff sequence diagram

One of the methods implemented by the `StaffManagementServiceImpl` is the `authenticate` method used to authenticate and retrieve the staff member details when they login to the application. The method takes a username and password and tries to retrieve a staff member details from the database that match the supplied details and various exceptions are thrown if a staff member matching the details is not found or if the staff member's account is locked. The method is also responsible for calling the `User.authenticate` method which will check the input password against the user's stored password and lock the account if the maximum invalid login attempts are exceeded.

In the context of a web application, the `authenticate` method alone is not enough because the web application need to remember that the user has authenticate for as long as they are using the application. This is achieved through the use of sessions provided by the Java `javax.servlet.http.HttpSession`, which can be used to store any objects. In this case when the user is authenticated then their `StaffMember` object will be stored in the session and used as an indication that this user has already logged into the application (as will be explained in the presentation layer section below), and these details stays in the session until the user logout or their session expires.

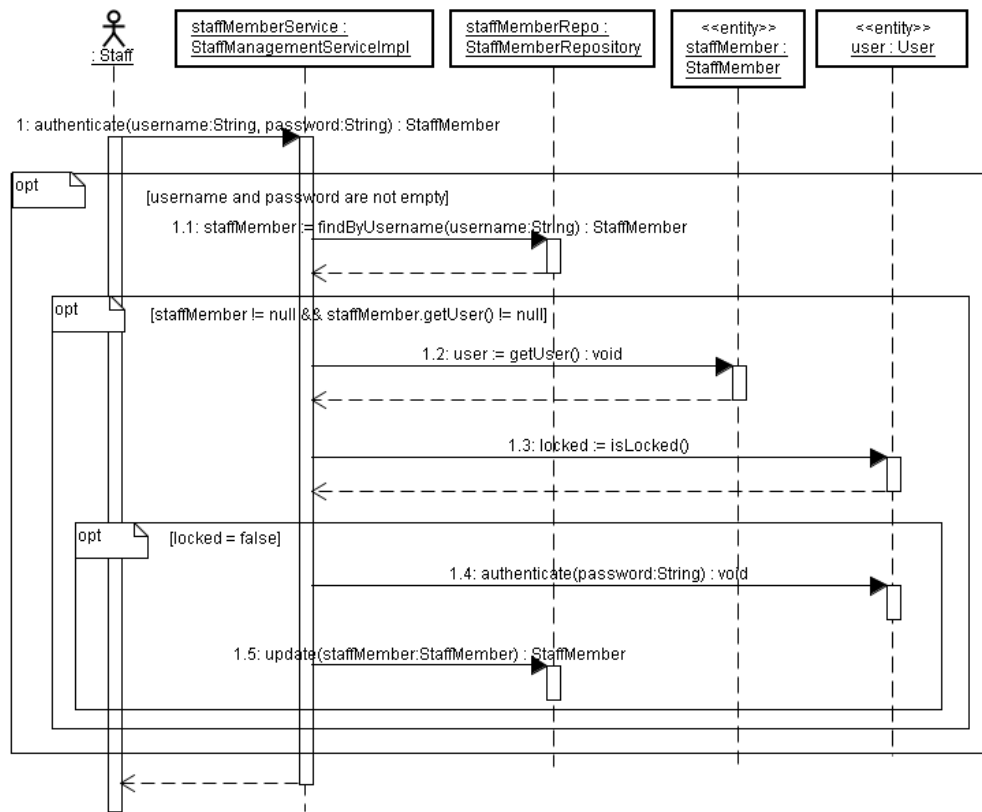


Figure 44. Sequence diagram for the authenticate

Implementation Details

The Staff package classes were implemented using the following classes including fully qualified package names:

- com.officema.model.staff.grades.Grade.java
- com.officema.model.staff.roles.Accountant.java
- com.officema.model.staff.roles.Administrator.java
- com.officema.model.staff.roles.GenericRole.java
- com.officema.model.staff.roles.RegularStaff.java
- com.officema.model.staff.roles.Role.java
- com.officema.model.staff.types.EmploymentType.java
- com.officema.model.staff.types.Gender.java
- com.officema.model.staff.types.OperationType.java
- com.officema.model.staff.types.Responsibility.java
- com.officema.model.staff.types.ResponsibilityType.java
- com.officema.model.staff.types.RoleType.java
- com.officema.model.staff.types.Title.java
- com.officema.model.staff.Address.java
- com.officema.model.staff.BankAccount.java
- com.officema.model.staff.EmploymentDetails.java
- com.officema.model.staff.StaffMember.java
- com.officema.model.staff.User.java

- com.officema.persistence.dao.jpa.GradeRepositoryImpl.java
- com.officema.persistence.dao.jpa.RoleRepositoryImpl.java
- com.officema.persistence.dao.jpa.StaffMemberRepositoryImpl.java
- com.officema.persistence.dao.GradeRepository.java
- com.officema.persistence.dao.RoleRepository.java
- com.officema.persistence.dao.StaffMemberRepository.java

- com.officema.services.staff.StaffManagementService.java
- com.officema.services.staff.StaffManagementServiceImpl.java

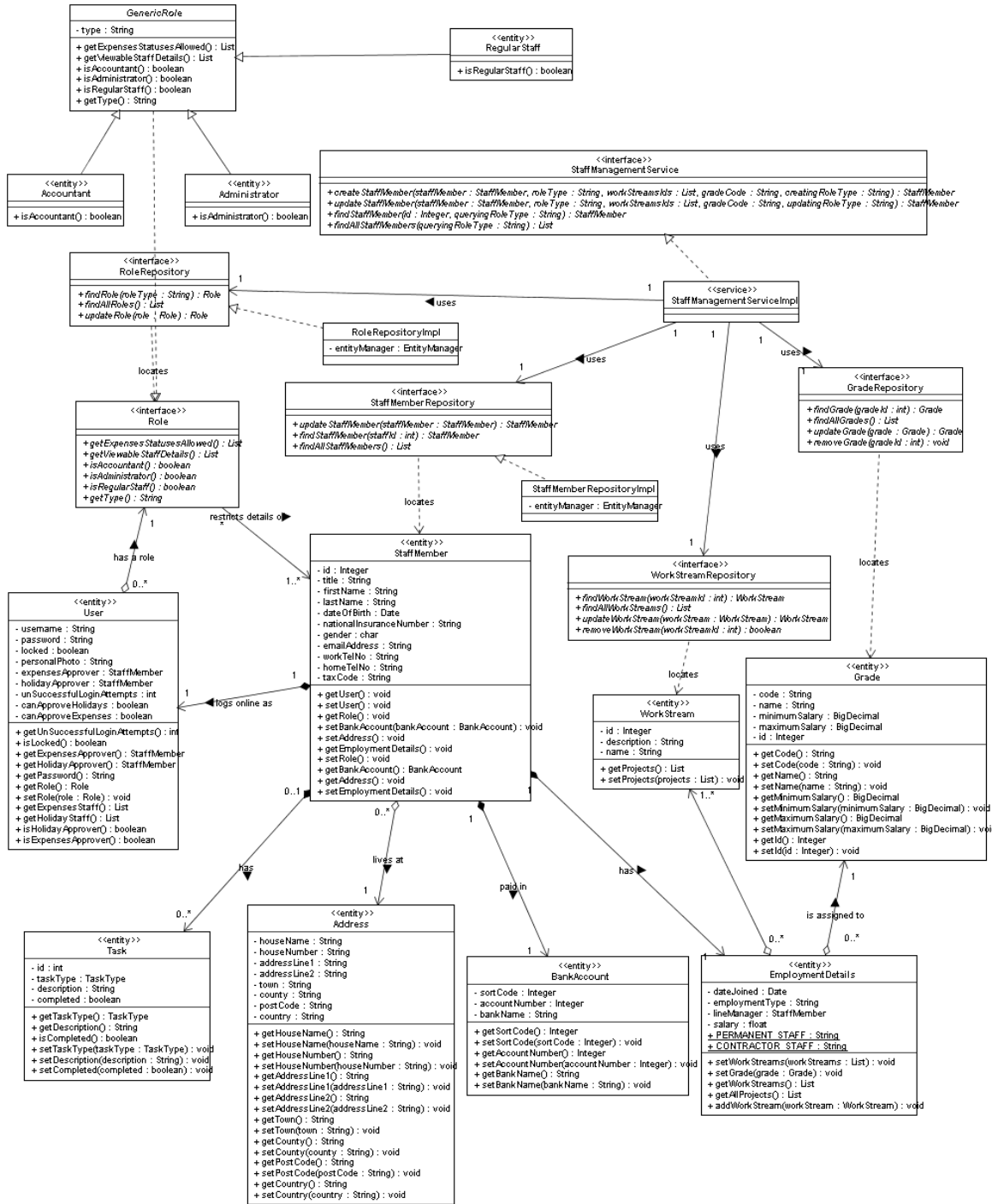


Figure 45. Staff classes and dependencies (Generic repository classes omitted for clarity)

7.2.5 Task package:

The Task package contains classes such as Task and TaskType. The class diagram below outline the detailed classes for the Task package and contains a repositories and a service as follows:

- TaskRepositoryImpl – responsible for saving and retrieving the Task entity from database.

The Task class references an instance of StaffMember the same approach that was used above for Expenses and HolidayYear classes. The class also contains a ManyToOne annotation on the StaffMember instance variable. The Task class is dependent on the StaffMember class as depicted by the class diagram below through the composition association. The types of tasks and their messages were implemented using a Java Enum in the TaskType class. The class demonstrates the strengths of Java Enum by providing a constructor method that was used to read a custom message for each TaskType from a file “MessageResource.properties” so that changes to message can be done without changing the code itself as shown in the code snippet below:

```
private TaskType(String messageKey, String taskName) {
    Properties defaultProps = new Properties();
    try {
        defaultProps.load(TaskType.class.
            getResourceAsStream("MessageResource.properties"));
    } catch (IOException ioe) {
        log.error("Failed to load properties in" +
            " TaskType constructor", ioe);
    }
    this.taskMessage = defaultProps.getProperty(messageKey);
    this.taskName = taskName;
}
```

Task management service

The TaskManagementServiceImpl class provide an implementation of all the business logic that was required to satisfy the various task management use cases through the methods shown below in Figure 47. The service also provides the interface TaskManagementService for clients use to hide the implementation. The service class makes use of TaskRepository to retrieve and update tasks related entities. This private instance variable is injected using the Spring framework at runtime using Dependency Injection (Figure 46).

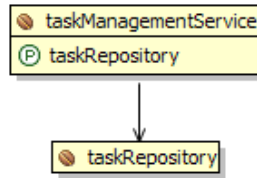


Figure 46. Spring beans schematic for TaskManagementServiceImpl

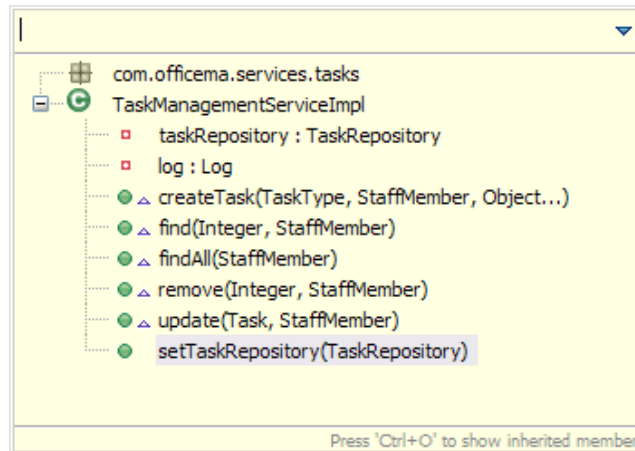


Figure 47. Methods defined by the TaskManagementServiceImpl

Implementation Details

The Task package classes were implemented using the following classes including fully qualified package names:

- com.officema.model.tasks.types.TaskType.java
- com.officema.persistence.dao.jpa.TaskRepositoryImpl.java
- com.officema.persistence.dao.TaskRepository.java
- com.officema.services.tasks.TaskManagementService.java
- com.officema.services.tasks.TaskManagementServiceImpl.java

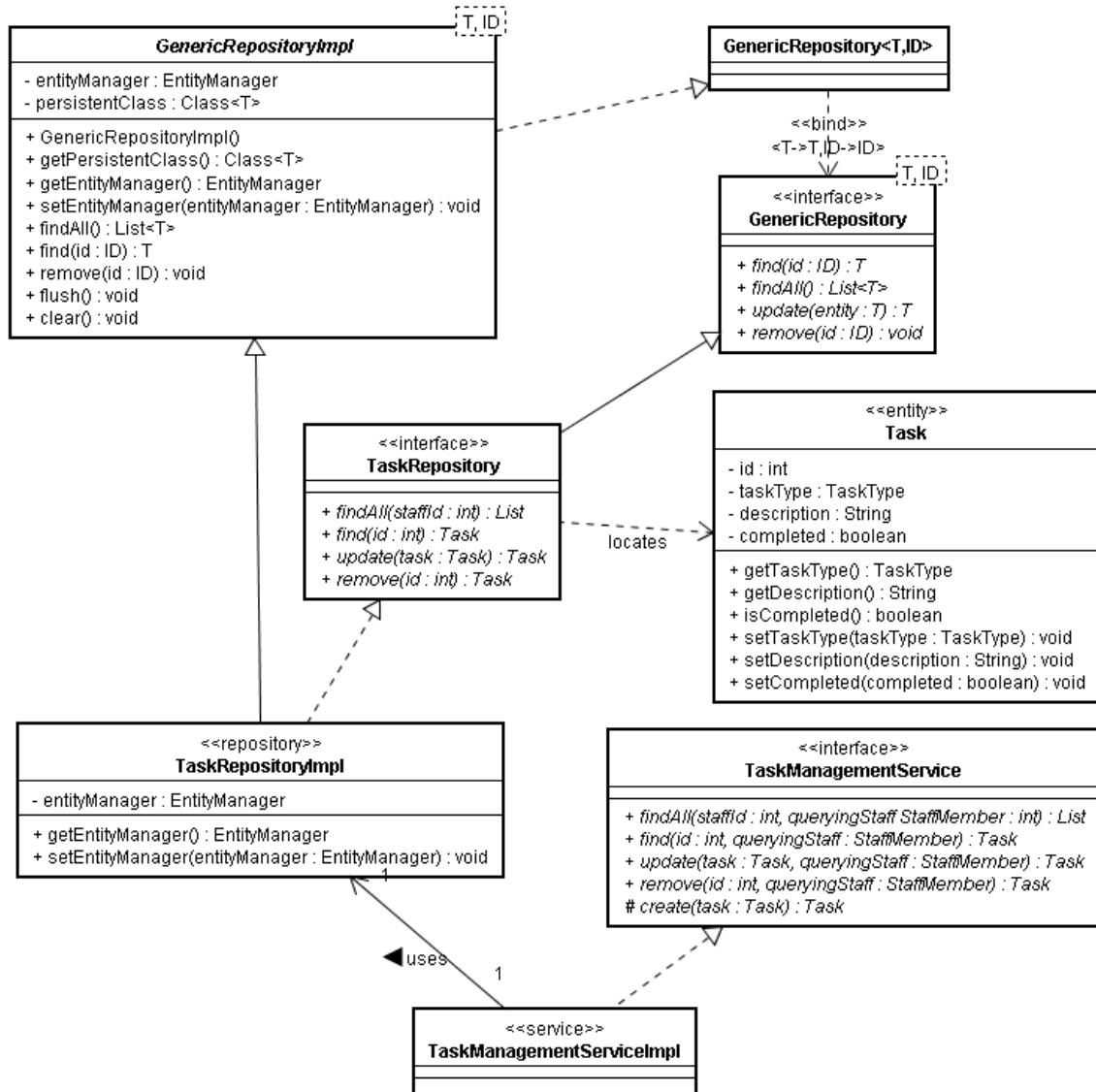


Figure 48. Task classes and dependencies

7.2.6 Testing the domain model

As outlined in subsection (6.3.6) unit testing was done using the JUnit framework to test the business logic and the domain model. The approach used was to test a service class as soon as it was completed in an attempt to phase out any issues or bugs; this ensured that the unit being tested worked successfully as a unit and as part of the whole model. The test classes were implemented in the “/OfficeMA/test” folder.

7.3 User Interface Design and Implementation

The user interface was implemented in two parts, the presentation logic layer which consists of a number of action classes and the user interface which comprises JavaScript, cascaded style sheets, HTML and images.

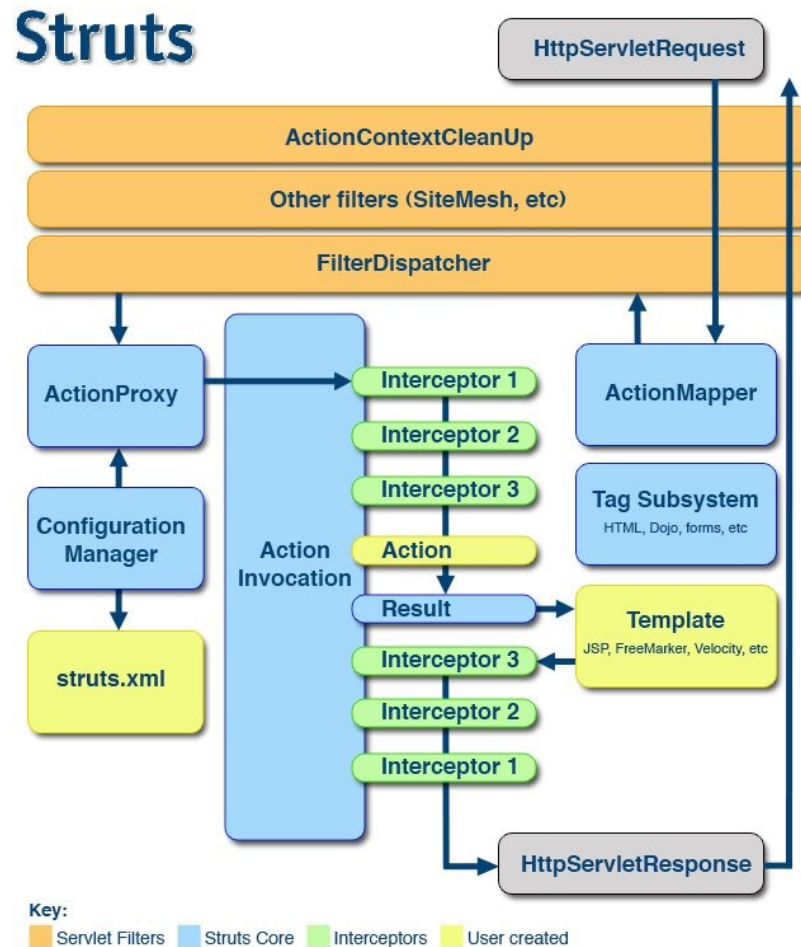


Figure 49. Struts 2 request flow [44].

As explained in subsection (6.4.1) Struts 2 framework was used to implement the presentation logic layer. The presentation classes were implemented in the **com.officema.presentation** package. For the purpose of this application a number of Action classes were created as highlighted in yellow in Figure 47 above. The file **struts.xml** holds the entire configuration for the Action classes and the type of their results (highlighted in blue). An Interceptor class (**com.officema.presentation.interceptors.AuthenticationInterceptor.java**) was also created to validate the user's session upon each HTTP request to check that the user has authenticated, if not then it redirects the user to the login page.

7.3.1 Application action classes

An action class is simply a Java class that has a method named `execute`, which Struts 2 invokes automatically at runtime. The user can also declare any other method and tell Struts to call it by declaring this in the `struts.xml` files as shown in the snippet below:

```
<action name="queryAllStaff" method="queryAll"
        class="queryStaffAction">
    <result type="json">
        <param name="excludeProperties">staffMember,
            staffed
        </param>
    </result>
</action>
```

The snippet shows the mapping for one of the action classes used to query all the staff details as the name imply, also the method name that will be invoked by Struts is declared as “`queryAll`”. The results for the action class on the other hand are declared to be of type “`json`”, this will automatically serialize the Java objects on the HTTP response scope (called the value stack) into JSON strings using the JSON plug-in [62]. The user can also exclude a number of objects in the value stack from being serialized using the “`excludeProperties`”. The Struts result could also be a Java Server Page (JSP), a static HTML page or another action class to chain actions together.

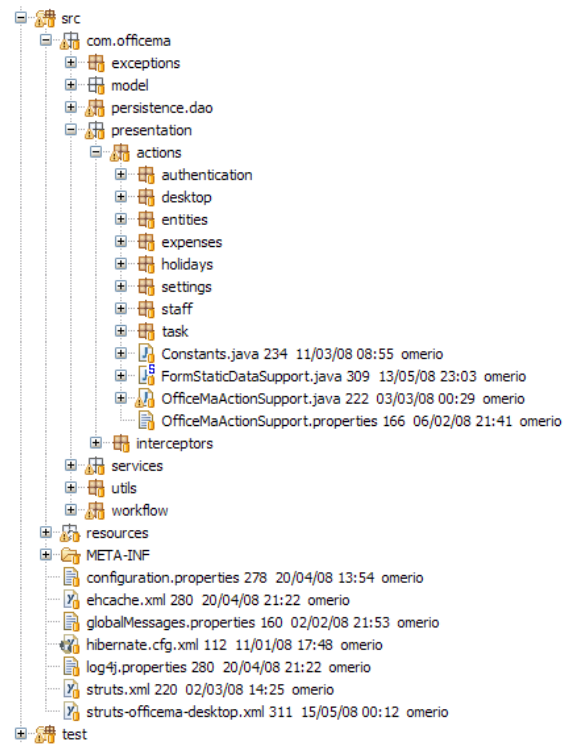


Figure 50. Package structure for presentation layer

Figure 50 above shows the various source files for the presentation logic layer, below is an explanation of the various packages and their significance. Each package also contains the relevant validation definition XML files:

- **com.officema.presentation package** – this package contains the OfficeMaActionSupport super class that all the action classes inherit. This class defines the common functionality required by the application action classes such as the ResultStatus class required by the user interface.
- **Authentication package** – contains the action classes concerned with logging in and out of the application. The classes in this package make use of the StaffManagementService class to authenticate the users to the application.
- **Desktop package** – contains the action classes that are concerned with displaying the application desktop such as the com.officema.presentation.actions.desktop.DesktopLauncherAction.java class. This class declares its result in struts.xml as the main JSP for the application that contains the entire HTML required to build the user interface on the client side.
- **Entities package** – contains user interface model and wrapper classes that wrap the domain model object to either restrict or mutate their properties.
- **Expenses package** – contains the action classes that deal with the expenses functionality and the various requests from the user interface. All the results of the action classes of this package are declared as “json” type. The classes in this package make use of the ExpensesManagementService class to retrieve and update expenses.
- **Holiday package** – contains the action classes that deal with the holiday functionality and the various requests from the user interface. All the results of the action classes of this package are declared as “json” type. The classes in this package make use of the HolidaysManagementService class to retrieve and update holidays.
- **Staff package** – contains the action classes that deal with the staff functionality and the various requests from the user interface. All the results of the action classes of this package are declared as “json” type. The classes in this package make use of the StaffManagementService class to retrieve and update staff details.
- **Settings package** – contains the action classes that deal with updating instances such WorkStream, Project, Grade, ExpensesCategory and Roles. All the results of the action classes of this package are declared as “json” type. The classes in this package make use of the SystemSettingsService class to retrieve and update these details.

- **Task package** – contains the action classes that deal with the task functionality and the various requests from the user interface. All the results of the action classes of this package are declared as “json” type. The classes in this package make use of the TaskManagementService class to retrieve and update tasks.

Implementation Details

Java classes:

- com.officema.presentation.actions.authentication.LoginAction.java
- com.officema.presentation.actions.authentication.LogoutAction.java
- com.officema.presentation.actions.desktop.DesktopLauncherAction.java
- com.officema.presentation.actions.entities.wrappers.EmploymentDetailsWrapper.java
- com.officema.presentation.actions.entities.wrappers.StaffMemberWrapper.java
- com.officema.presentation.actions.entities.wrappers.UserWrapper.java
- com.officema.presentation.actions.entities.CurrentUser.java
- com.officema.presentation.actions.entities.ResultStatus.java
- com.officema.presentation.actions.entities.StaffDetails.java
- com.officema.presentation.actions.entities.StatusType.java
- com.officema.presentation.actions.expenses.ExpensesQueryAction.java
- com.officema.presentation.actions.expenses.ExpensesUpdateAction.java
- com.officema.presentation.actions.holidays.HolidaysQueryAction.java
- com.officema.presentation.actions.holidays.HolidaysUpdateAction.java
- com.officema.presentation.actions.settings.SystemSettingsQueryAction.java
- com.officema.presentation.actions.settings.SystemSettingsUpdateAction.java
- com.officema.presentation.actions.staff.FormStaticDataSupport.java
- com.officema.presentation.actions.staff.QueryStaffAction.java
- com.officema.presentation.actions.staff.StaffMemberAction.java
- com.officema.presentation.actions.staff.UpdateStaffAction.java
- com.officema.presentation.actions.task.TaskManagementAction.java
- com.officema.presentation.actions.Constants.java
- com.officema.presentation.actions.OfficeMaActionSupport.java
- com.officema.presentation.interceptors.AuthenticationInterceptor.java

Other configuration files:

- /OfficeMA/src/struts.xml
- /OfficeMA/src/struts-officema-desktop.xml

- /OfficeMA/src/com/officema/presentation/actions/OfficeMaActionSupport.properties
- /OfficeMA/src/com/officema/presentation/actions/staff/StaffMemberAction-save-validation.xml
- /OfficeMA/src/com/officema/presentation/actions/staff/QueryStaffAction-query-validation.xml
- /OfficeMA/src/com/officema/presentation/actions/staff/package.properties
- /OfficeMA/src/com/officema/presentation/actions/authentication/LoginAction-validation.xml
- /OfficeMA/src/com/officema/presentation/actions/settings/package.properties

7.3.2 User Interface Design

The user interface for the application was implemented by following the design rules identified in subsection (2.6.3). The user interface has adhered to these rules as summarised below. The description of the various windows in the application and their functionality and purpose is described in more details in the user guides developed as part of this project (Appendix H).

Simplicity and Structure

As mentioned before in subsection (2.4.2) the user interface followed an MDI approach (Figure 51) similar to the one used in developing traditional desktop applications. For this approach to work the browser's conventional functions are hidden and the application is loaded on a full browser window to maximize the space used for the application. The browser right click menu was also replaced by a custom menu for the application as the browser functionality such as refresh or back is not relevant in this case. The application desktop source files are listed below, these use the JSP technology:

- /OfficeMA/WebContent/WEB-INF/jsp/bodyContents.jsp
- /OfficeMA/WebContent/WEB-INF/bodyContents.jsp
- /OfficeMA/WebContent/WEB-INF/loader.jsp
- /OfficeMA/WebContent/WEB-INF/menuAndToolbar.jsp
- /OfficeMA/WebContent/WEB-INF/menusDefinitions.jsp
- /OfficeMA/WebContent/WEB-INF/officema.jsp
- /OfficeMA/WebContent/WEB-INF/postLogin.jsp

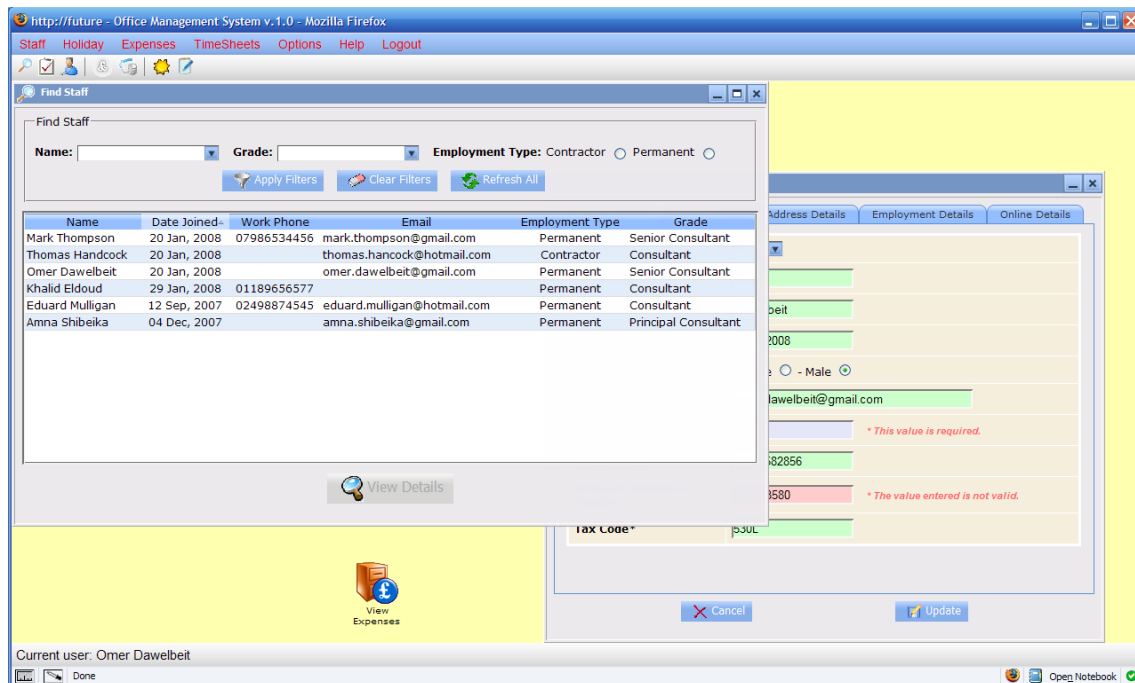


Figure 51. Office Management Application desktop

Visibility, Affordance and Consistency

The interaction styles used for the user interface were summarised in subsection (2.4.1), out of these styles the menu selection has been identified as the main form of interaction in the application. This is achieved by choosing options from the top menu bar. A toolbar is also provided for frequently used options, and as the user hovers on top of any of the toolbar icons a tool-tip is displayed summarising the purpose of the icon. Using this approach which is consistent with Windows applications enables the user to easily figure out how to use the controls to access the various features in the application.

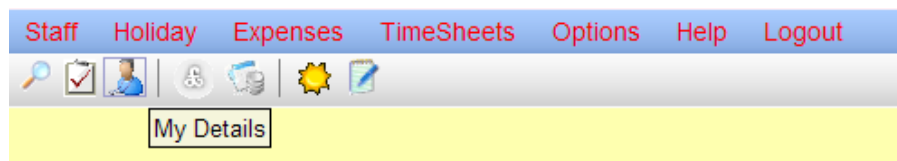


Figure 52. OfficeMA menus and toolbar

The visual design for the project tried to follow some of the good user interface practices outlined in the Window Vista user experience guide from Microsoft [71] such as:

- The use of common controls and common dialogs
- The use of standard window frame.
- Using icons and graphics consistently
- Using task dialogue for various messages and apply a suitable icon to indicate the current situation (Figure 54)

Feedback

Other features include the use of a busy logo whilst using AJAX to retrieve data from the server (Figure 53). As the AJAX call is Asynchronous and the response is returned by the Dojo toolkit using a call-back function; depending on the delay it takes for the server to response the application shows a modal icon which indicate to the user that the application is currently loading some data. This is important from a usability point of view so that the user is aware of the current state of the application. A timeout is also set to check if the server does not response within a configurable timeframe to display an error message to the user.

The application also makes use of information, warning and error dialogues boxes (Figure 54) to convey the status of the application to the user.



Figure 53. An animated image to indicate the application is loading data



Figure 54. Message dialogues in the OfficeMA

Add New Employee

Personal Details | Bank Details | Address Details | Employment Details | Online Details

Title* * This value is required.

Firstname* * This value is required.

Lastname* * This value is required.

Date of Birth* * This value is required.

Gender* Female ☐ - Male ☐

Email Address

Work Phone* * This value is required.

Home Phone

National Insurance Number* * This value is required.

Tax Code* * This value is required.

Figure 55. Add staff window.

Tolerance

The user interface was designed to minimize users from making errors by performing interactive validation to user's input which indicates to the user immediately if they provided an invalid value for an input. An example is the shown in Figure 55 in the Add staff window where mandatory fields are marked on the window with an asterisk and a message indicating the value is required or invalid.

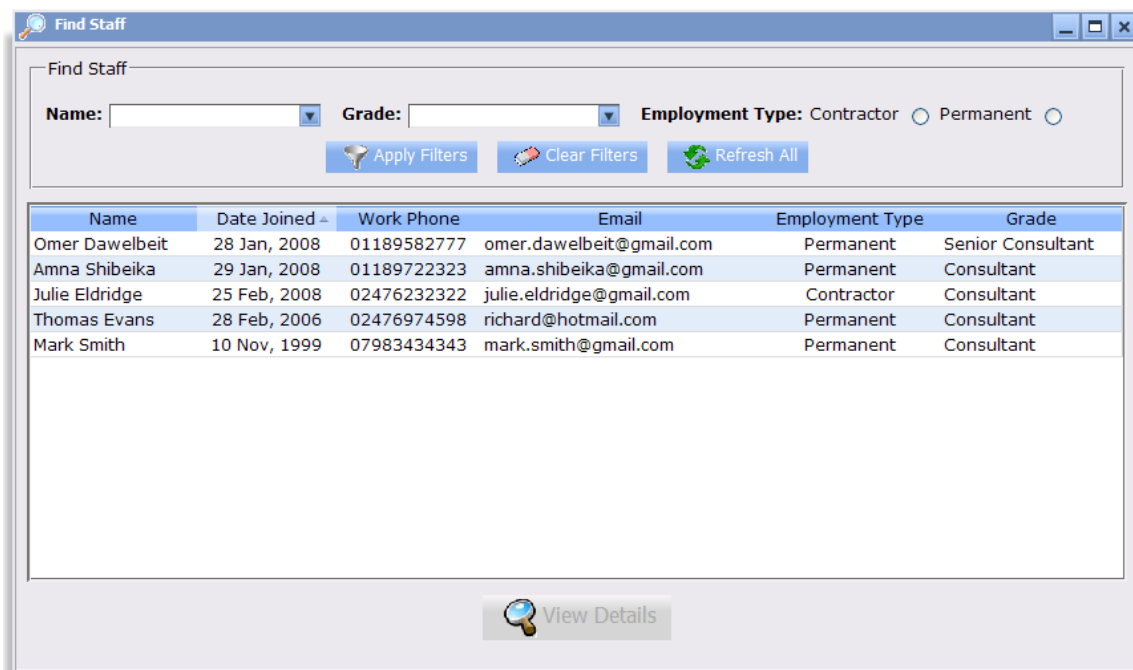
Closure

Dialogue boxes and windows such as the Add staff window shown in Figure 55 are designed in a way it is clear to the user which dialogue is used for viewing or updating information. In this case the window has two buttons one to save and another to cancel. It is also made clear when input is required by the user and when the action has been completed successfully. Feedback is given to the user after the user submits their updates and the current data in the application is refreshed accordingly.

Performance and Data Refresh

Performance in the application user interface is very much dependent on the browser used, some browsers such as Firefox offer high performance compared to others such as Internet Explorer. However, the user interface implemented followed some of the best practices for efficient HTML manipulation such as the use of JavaScript innerHTML function to append HTML elements to a DOM node instead of using the createElement function which is slower. The user interface performance is also dependent on the server's performance when retrieving and saving data, for this purpose many techniques were applied in the server to enhance the performance of the application as discussed in subsection (7.5).

In regards to data refresh the application provides the ability for the user to refresh the data where possible for example by providing a refresh button (Figure 56 below). The application also provides automated data refresh whenever a window is opened or shown. This ensures that the data in the user interface is not stale and is kept in synchronisation with the server data.



The screenshot shows a web application window titled "Find Staff". It contains a search form with the following fields and controls:

- Name:** A text input field.
- Grade:** A dropdown menu.
- Employment Type:** Radio buttons for "Contractor" and "Permanent".
- Buttons:** "Apply Filters" (with a funnel icon), "Clear Filters" (with an eraser icon), and "Refresh All" (with a circular arrow icon).

Below the form is a table with the following data:

Name	Date Joined	Work Phone	Email	Employment Type	Grade
Omer Dawelbeit	28 Jan, 2008	01189582777	omer.dawelbeit@gmail.com	Permanent	Senior Consultant
Amna Shibeika	29 Jan, 2008	01189722323	amna.shibeika@gmail.com	Permanent	Consultant
Julie Eldridge	25 Feb, 2008	02476232322	julie.eldridge@gmail.com	Contractor	Consultant
Thomas Evans	28 Feb, 2006	02476974598	richard@hotmail.com	Permanent	Consultant
Mark Smith	10 Nov, 1999	07983434343	mark.smith@gmail.com	Permanent	Consultant

At the bottom of the window is a "View Details" button with a magnifying glass icon.

Figure 56. Find staff window.

7.3.3 User interface controller

The controller for the user interface was developed using the approach developed in subsection (6.4.2). The controller contains the code required to invoke AJAX call and their call-back function, this approach provide a central

location to parse the server's response, check for errors and notify the user and UI widgets that an error has occurred. Figure 57 below shows the class diagram for the client side controller implemented in JavaScript and used the dojo.Declare notation. The controller was implemented in the “/OfficeMA/WebContent/scripts/officema_controller.js” JavaScript file.

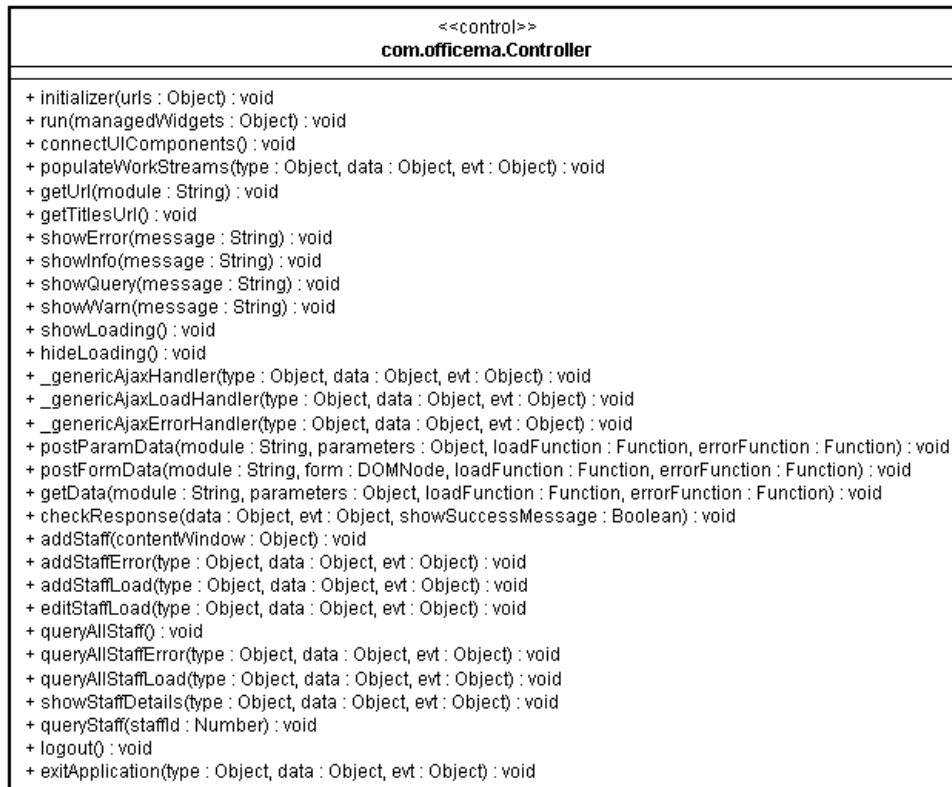


Figure 57. Class diagram for the client side JavaScript controller

7.3.4 User interface modelling

For each of the prototypes developed during the analysis class diagrams (Appendix D), the boundary classes and interaction diagrams were developed to show the overall interaction between the user and the application. This approach of modelling the client side classes alongside the server side classes is only made possible by the fact that the user interface used for the application is object oriented. Such great modelling flexibility would not be possible when modelling conventional Web applications that use normal HTML pages for display.

Staff management boundary classes

Boundary class diagrams for the staff management VSO were developed as shown in Figure 58 and implemented using the source files below. The VSO source comprises a JavaScript file, a HTML file for the template, a CSS file and any image files if any:

- /OfficeMA/WebContent/officemaWidgets/widget/templates/addStaff.css
- /OfficeMA/WebContent/officemaWidgets/widget/templates/addStaff.html
- /OfficeMA/WebContent/officemaWidgets/widget/templates/findStaff.css
- /OfficeMA/WebContent/officemaWidgets/widget/templates/findStaff.html
- /OfficeMA/WebContent/officemaWidgets/widget/templates/ModalAlert.css
- /OfficeMA/WebContent/officemaWidgets/widget/templates/ModalAlert.html
- /OfficeMA/WebContent/officemaWidgets/widget/templates/viewStaff.css
- /OfficeMA/WebContent/officemaWidgets/widget/templates/viewStaff.html
- /OfficeMA/WebContent/officemaWidgets/widget/addStaff.js
- /OfficeMA/WebContent/officemaWidgets/widget/CustomFloatingPane.js
- /OfficeMA/WebContent/officemaWidgets/widget/editStaff.js
- /OfficeMA/WebContent/officemaWidgets/widget/findStaff.js
- /OfficeMA/WebContent/officemaWidgets/widget/ModalAlert.js
- /OfficeMA/WebContent/officemaWidgets/widget/viewStaff.js

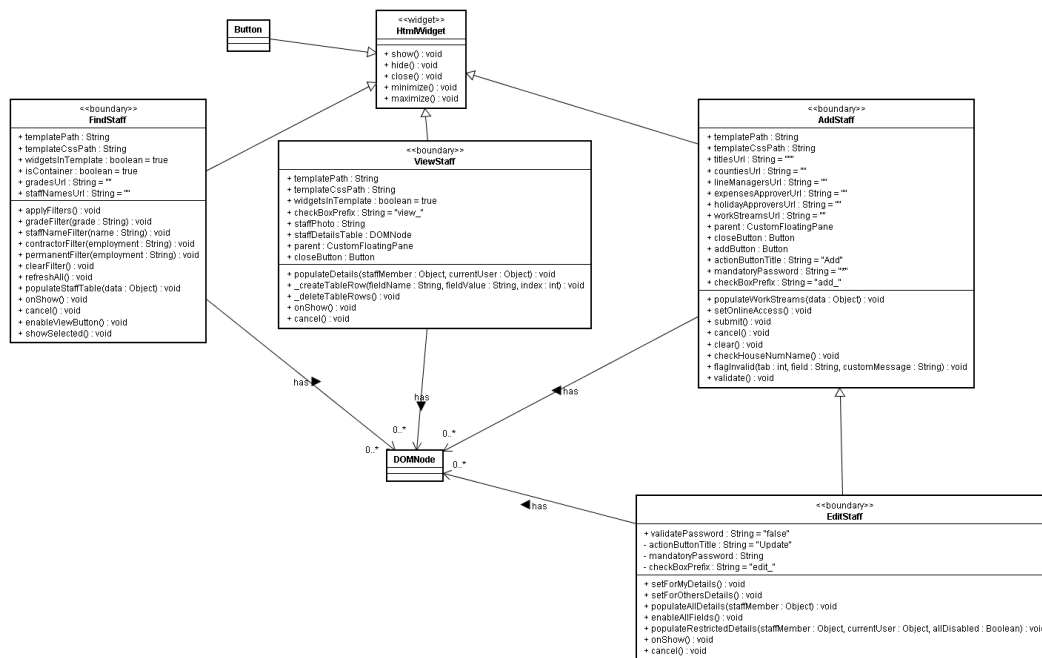


Figure 58. Boundary class diagram for staff management VSOs

Staff management UI interaction diagrams

The sequence diagrams below show the interaction between the user, the View Support Objects, the controller and the server. The messages between the client and the server are shown in the sequence diagram by using Asynchronous AJAX messages.

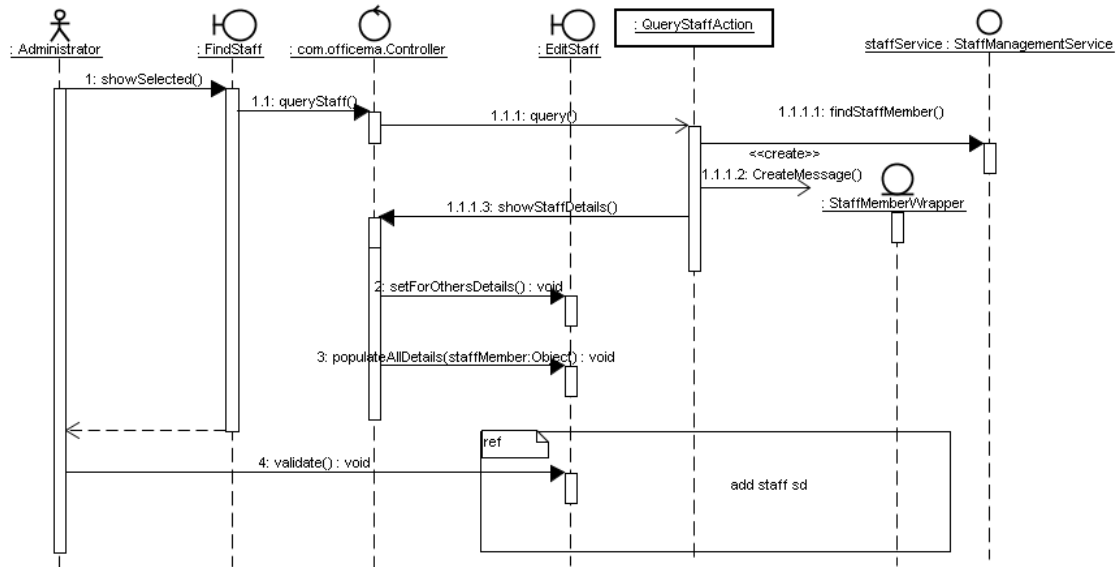


Figure 59. Edit staff details sequence diagram

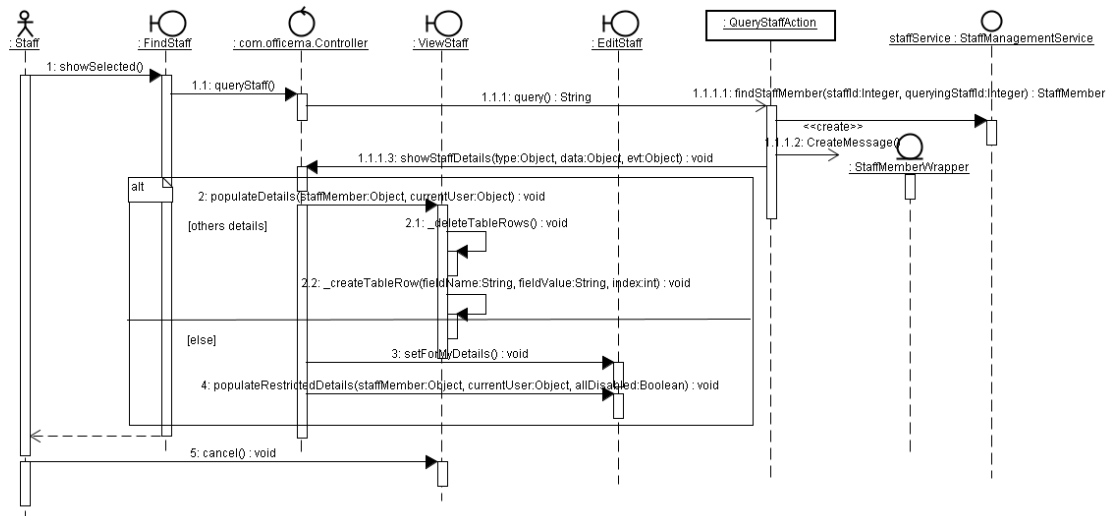


Figure 60. View staff details sequence diagram

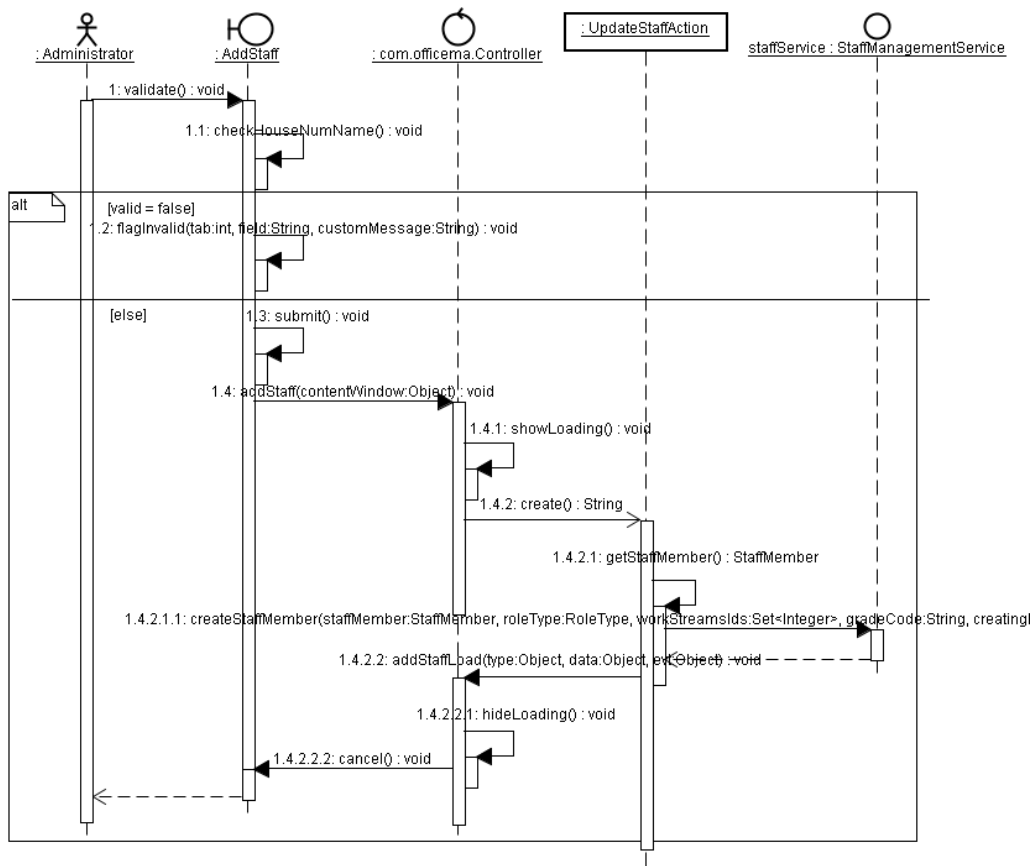


Figure 61. Add staff details sequence diagram

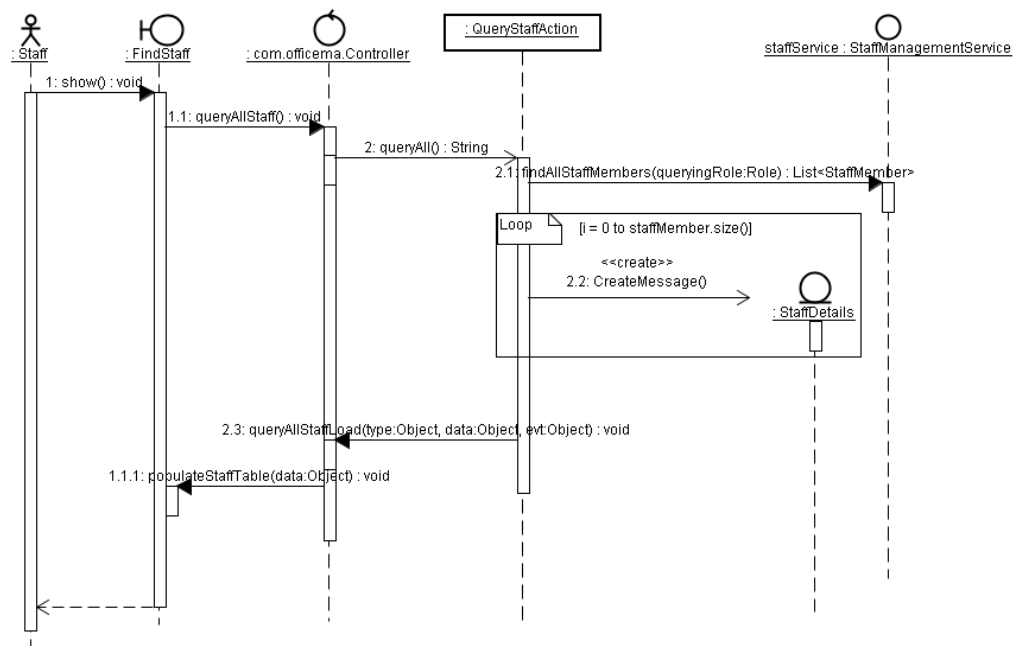


Figure 62. Find staff sequence diagram

Expenses management boundary classes

Boundary class diagrams for the expenses management VSO were developed as shown in Figure 63 and implemented using the source files below. The VSO source comprises a JavaScript file, a HTML file for the template, a CSS file and any image files if any:

- /OfficeMA/WebContent/officemaWidgets/widget/templates/EditExpenses.css
- /OfficeMA/WebContent/officemaWidgets/widget/templates/EditExpenses.html
- /OfficeMA/WebContent/officemaWidgets/widget/templates/ExpensesDetails.css
- /OfficeMA/WebContent/officemaWidgets/widget/templates/ExpensesDetails.html
- /OfficeMA/WebContent/officemaWidgets/widget/templates/ViewExpenses.css
- /OfficeMA/WebContent/officemaWidgets/widget/templates/ViewExpenses.html
- /OfficeMA/WebContent/officemaWidgets/widget/EditExpenses.js
- /OfficeMA/WebContent/officemaWidgets/widget/ExpensesDetails.js
- /OfficeMA/WebContent/officemaWidgets/widget/ExpensesItemRow.js
- /OfficeMA/WebContent/officemaWidgets/widget/ViewExpenses.js
- /OfficeMA/WebContent/officemaWidgets/widget/Map.js

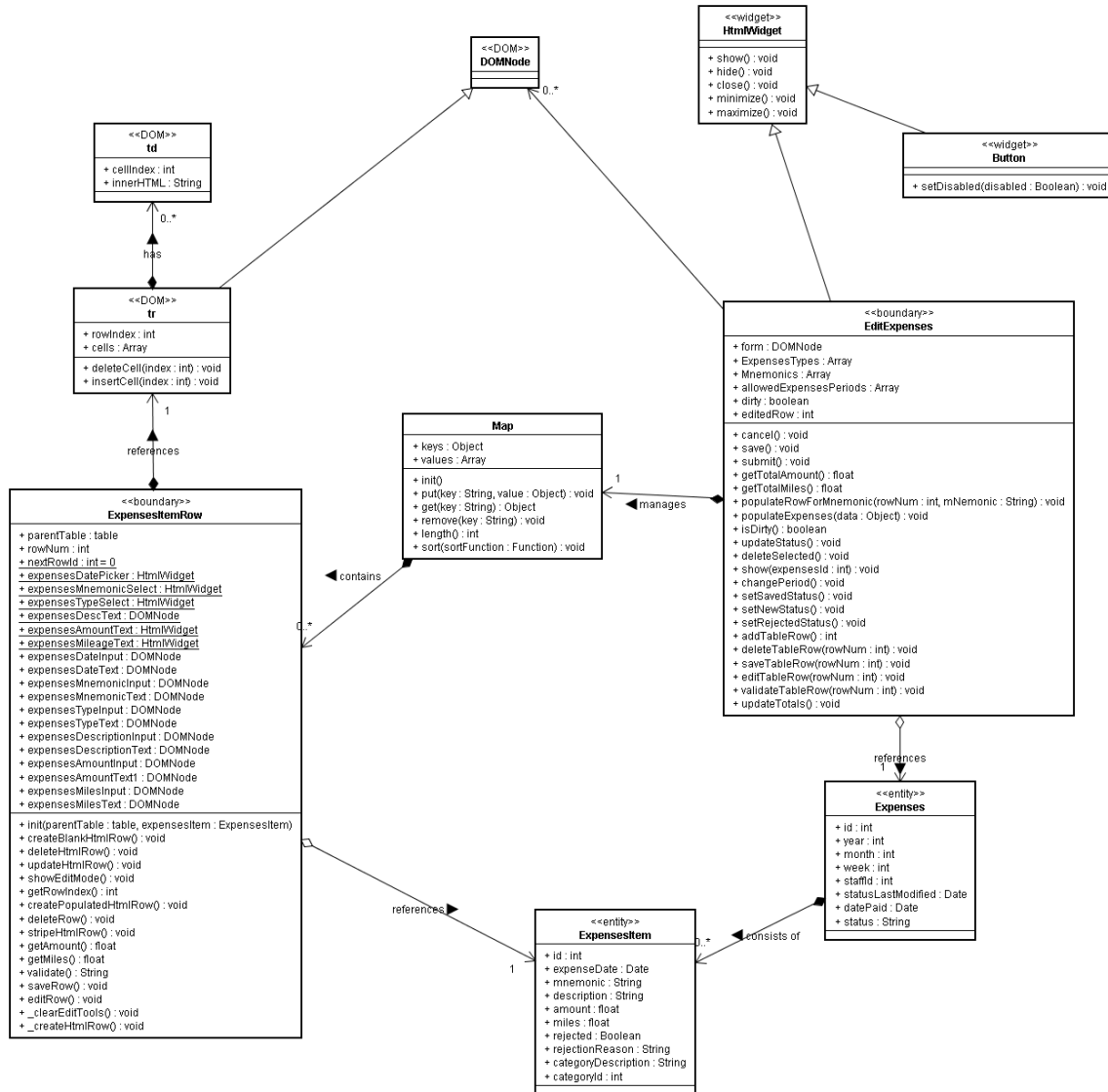


Figure 63. Boundary class diagram for expenses management VSOs

Expenses management UI interaction diagram

The sequence diagrams below show the interaction between the user, the user interface and the View Support Objects to create a new expenses item.

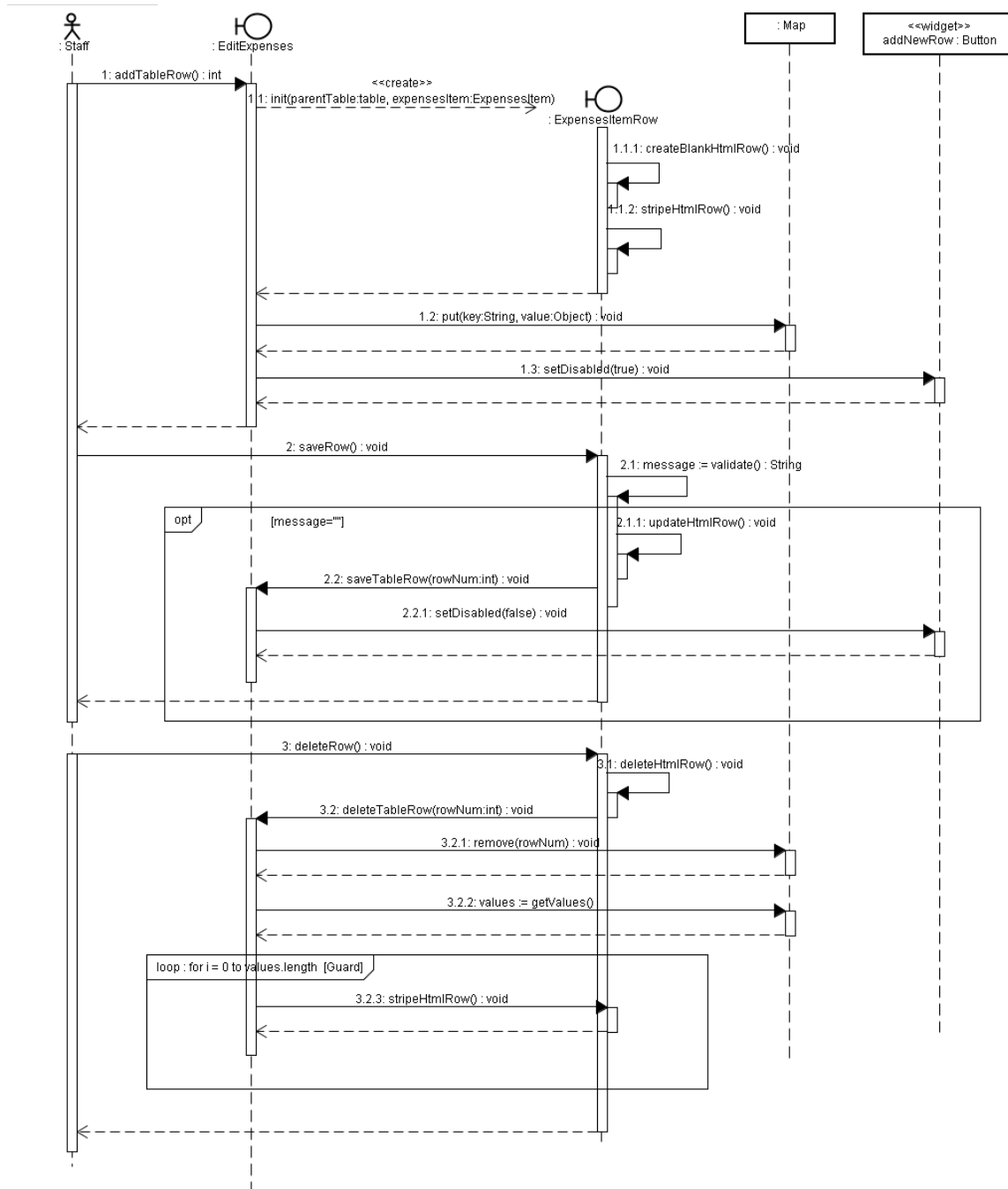


Figure 64. Add new expenses item sequence diagram

Task management boundary classes

Boundary class diagrams for the task management VSO were developed as shown in Figure 65 and implemented using the source files below. The VSO source comprises a JavaScript file, a HTML file for the template, a CSS file and any image files if any:

- /OfficeMA/WebContent/officemaWidgets/widget/templates/manageTasks.css
- /OfficeMA/WebContent/officemaWidgets/widget/templates/manageTasks.html
- /OfficeMA/WebContent/officemaWidgets/widget/ manageTasks.js

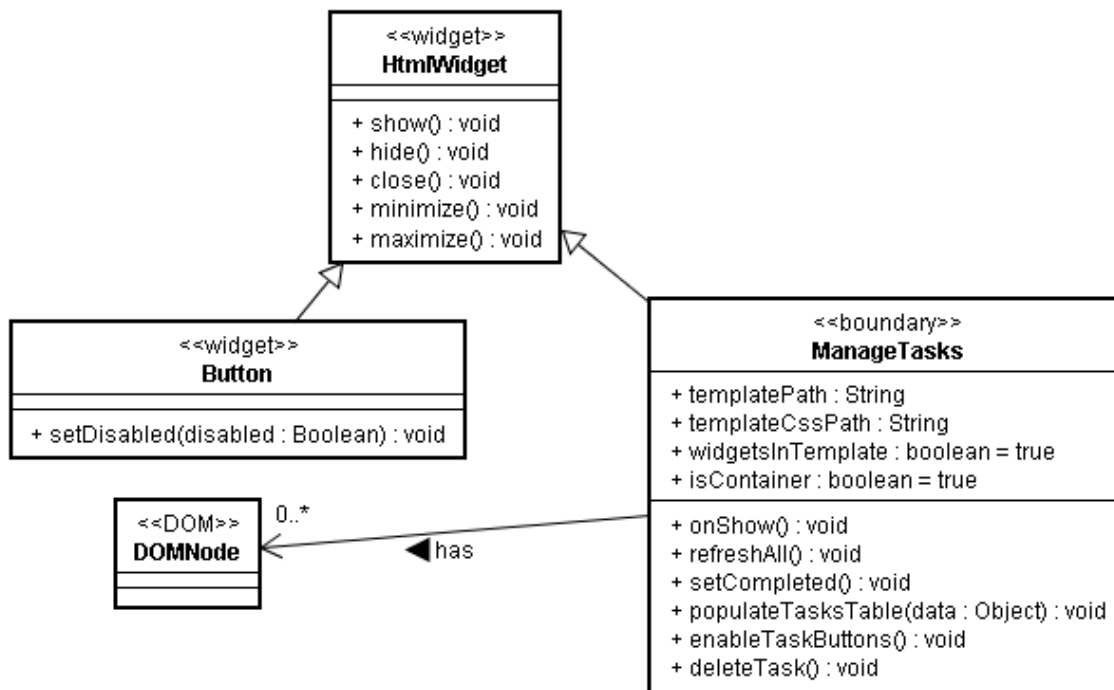


Figure 65. Boundary class diagram for task management VSOs

Task management UI interaction diagrams

The sequence diagrams below show the interaction between the user, the View Support Objects, the controller and the server. The messages between the client and the server are shown in the sequence diagram by using Asynchronous AJAX messages.

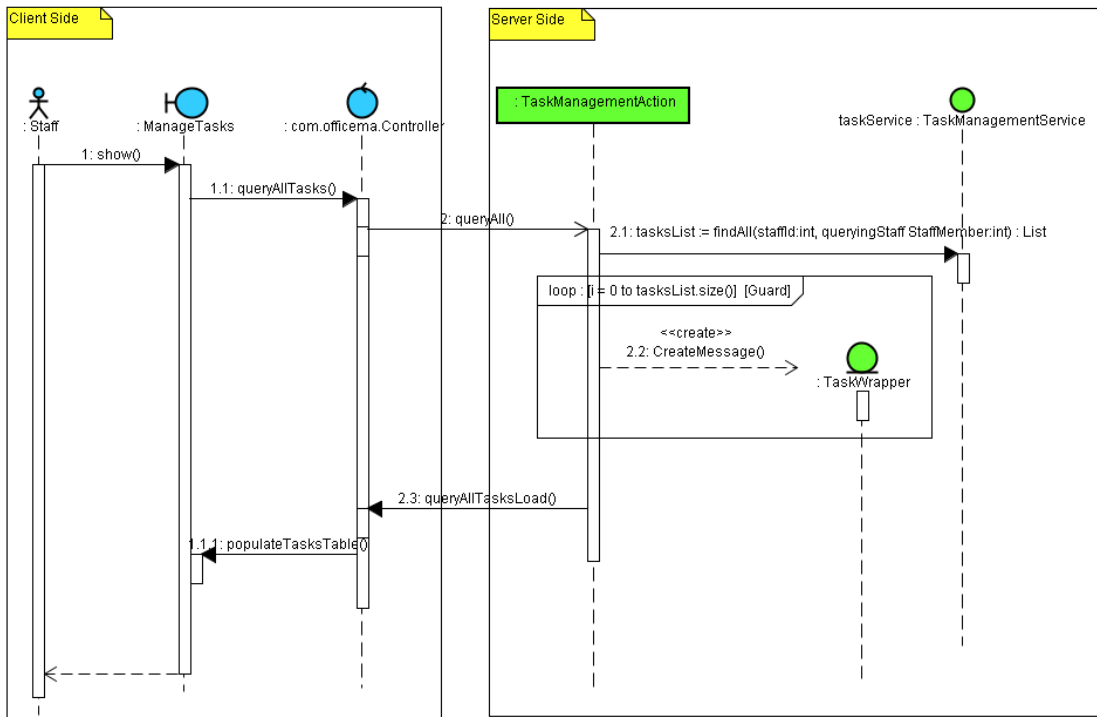


Figure 66. View Tasks sequence diagram

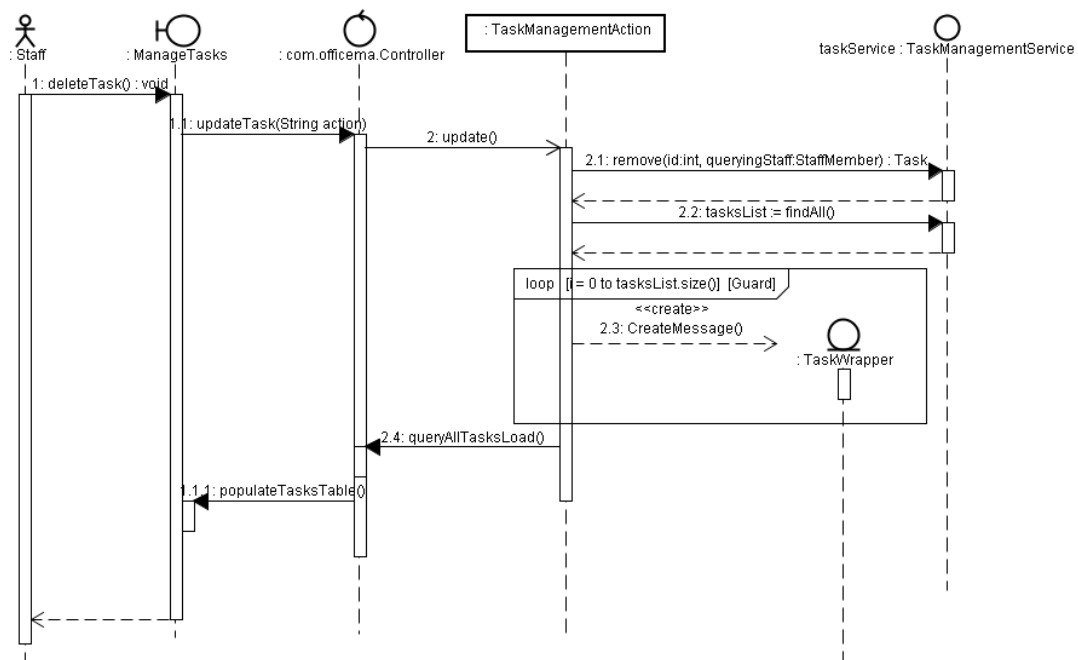


Figure 67. Delete/Update Task sequence diagram

7.4 Database Design and Implementation

7.4.1 Establishing requirements

The initial requirements for the database logical model were derived from the various class models, initially the analysis classes and then the detailed classes. Entities were derived from the classes with stereotype “entity” in the class diagram. Association between classes are also translated into relationship between entities.

7.4.2 Data Analysis

Entity types and relationships

By analysing the analysis class diagram the classes below were identified as candidate entities for the initial Entity-Relations model:

StaffMember, BankAccount, EmploymentDetails, User, Role, Address, Task, Grade, WorkStream, Project, Expenses, ExpensesItem, ExpensesCategory, MileageCost, ExpensesMnemonic, HolidayYear, Holiday

By further analysing the candidate entities above, it is clear that they can be classified as strong or weak entity types. Strong entities are not existence-dependent on some other entity, whilst weak entities are existence-dependent on some other entity [8].

Strong Entities (parent)

- StaffMember
- Grade
- WorkStream
- ExpensesCategory
- User
- EmploymentDetails
- Role

Weak Entities (child)

- BankAccount – dependent on StaffMember
- Address – dependent on StaffMember

- Task – dependent on StaffMember
- Project – dependent on WorkStream
- Expenses – dependent on StaffMember
- ExpensesItem – dependent on Expenses
- MileageCost – dependent on ExpensesCategory
- ExpensesMnemonic – dependent on User
- HolidayYear – dependent on StaffMember
- Holiday – dependent on HolidayYear
- ViewableStaffDetail – dependent on Role
- UpdateableStaffDetails – dependant on Role
- ViewableExpensesStatuses – dependant on Role

It was decided to model some of the candidate entities above as entity types for many reasons, one of which is the multi-valued attributes such as the MileageCost. This is done to fulfil the business requirement that mileage should be paid depending on how many miles the employee has claimed. For example 40pence per mile for less than 10,000 miles and 25pence there after, hence MileageCost was modelled as a separate entity. Other entities include Address and BankAccount which belong to StaffMember, but have been modelled as separate entities to avoid a large StaffMember entity with many attributes. EmploymentDetails was modelled as a strong entity as it represents the employment details of the staff member, such as salary, holiday entitlement, etc...

After completing the data analysis and composing an initial list of entity types and relationships, the identifiers for these entities were decided. Some of the entities have natural identifiers that are used by the business, such as employee id, username, and grade codes. However, for most of the entity types surrogate keys were used as primary keys and natural identifiers as candidate keys. The rationale behind this is to avoid using multi-attribute primary keys or natural keys that might change in the future. Bauer and King [2] have recommended the use of surrogate to ensure that the primary key is unique, constant, always required, and never null or unknown, which is sometimes hard to achieve using natural keys.

Entity subtypes

Inheritance from the UML classes was modelled using entity subtypes (generalization) in the E-R model. An example is the subclasses of the Role class

implemented in the details class model such as Accountant, RegularStaff and Administrator, these are specialized roles that inherit from Role (Figure 68).

One to one and one to many relationships

One to one relationships were modelled by first deciding on which side will have an attribute that will be declared as a foreign key, the same attribute was then declared as an alternate key in the other relation to ensure a one multiplicity instead of many. One to many follows the same approach, but the foreign key is not declared as unique to allow for multiplicity.

Many to many Relationships

Many to many relationship in the E-R model, also called intersection relations can not be represented using the primary key / foreign key mechanism so these were resolved using a dependant entity between the two entities participating in the m:n relationship, hence resulting in three entities participating in two 1:n relationships.

Constraints

Constraints are represented in the E-R models in two ways:

- as a property of a modelling construct
- as a description in the Constraints part of a model

The constraints that were expressed in the conceptual data model are

- Each identifier has a unique value such as staff id
- An entity type participates in a relationship such that an occurrence of the entity type only participates once (at most) for example one to one relationship with optional participation.
- An entity type may be shown to participate in a relationship such that each occurrence of the entity type must participate at least once for example one to one or one to many mandatory participation.

7.4.3 Entity Relationship Model

Below is the Entity Relations model (Figure 68) developed using the convention summarised in Table 4. The model comprises an ER diagram, entity types and constraints.

Entity Types:

StaffMember (StaffId, DateOfBirth, EmailAddress, FirstName, LastName, Gender, HomeTelNo, WorkTelNo, NINumber, TaxCode, Title,)

User (UserName, CanApproveExpenses, CanApproveHolidays, Locked, Password, PersonalPhoto, UnSuccessfulLoginAttempts)

Role (RoleId, RoleType)

ViewableStaffDetails (RoleId, FieldName)

UpdateableStaffDetails (RoleId, FieldName)

ViewableExpesesStatuses (RoleId, StatusName)

ExpensesMnemonics (Name, Amount, Mileage)

BankAccount (StaffId, AccountNumber, sortCode, BankName)

HomeAddress (StaffId, AddressLine1, AddressLine2, Country, County, HouseNumber, HouseName, Locality, PostCode, Town)

Task (Id, Completed, DateCreated, Description, TaskType, Title)

EmploymentDetails (Id, DateJoined, DateLeft, EmploymentType, HolidayEntitlement, Salary)

Grade (Id, Code, MaximumSalary, MinimumSalary, Name)

EmploymentWorkStream (EmploymentDetailsId, WorkStreamId)

WorkStream (Id, Description, Name)

Project (Id, Code, Description, Name)

HolidayYear (Id, CarryOver, DaysInLieu, Entitlement, HolidayYear)

Holiday (Id, AfterNoon, BookedDate, FromYear, FullDay, Status)

Expenses (Id, DatePaid, StatusLastModified, Status, ExpensesMonth, ExpensesWeek, ExpensesYear)

ExpensesItem (Id, Amount, Description, ExpenseDate, Miles, Mnemonic, Rejected, RejectionReason)

ExpensesCategory (Id, Category, HasMileage)

MileageCost (Id, Cost, LowerLimit, UpperLimit)

Constraints

- Each StaffMember participate with the HolidayYear only once for each value HolidayYear attribute
- Each StaffMember participate with the Expenses entity only once for each value of ExpensesMonth, ExpensesWeek and ExpensesYear.

- Only one Holiday should be booked by any employee for the same day
- Either house name or house number or both should be supplied.
- ExpensesMnemonic must have an amount or mileage or both

7.4.4 Normalisation

As the database design for this project followed a top-down approach through ER modelling, normalisation was used as a validation technique to check the structure of relations and ensure that each of these relations were well designed and meet the data requirements as outlined by Connolly and Begg [8]. All the relations above were checked and found to be in the Boyce-Codd Normal Form (BCNF) as explained below:

- **1NF** – all the relations above are in this form because all the non-primary key attributes are functionality dependent on the primary key.
- **2NF** – any relation with a single attribute primary key must be in at least 2NF, which applied to all the relations above.
- **3NF** – there are no transitive dependencies and hence all relations above are in 3NF.
- **BCNF** – all the relations above are in this form as it's safe to assume that relations in 3NF are also in BCNF if these relations have [23]:
 1. only one candidate key (no alternate keys);
or, if there is more than one candidate key, then
 2. the candidate keys are not combinations of attributes;
or, if the candidate keys are combinations of attributes, then
 3. the candidate keys do no overlap.

Below are the functional dependencies for each of the relations above. Relations created to resolve m:n relationships and multi-value attributes have been omitted:

StaffMemberPrimary key FDs

StaffId → DateOfBirth, EmailAddress, FirstName, LastName, Gender,
HomeTelNo, WorkTelNo, NINumber, TaxCode, Title

Candidate key FDs

NINumber → DateOfBirth, EmailAddress, FirstName, LastName, Gender,
HomeTelNo, WorkTelNo, StaffId, TaxCode, Title

User

UserName → CanApproveExpenses, CanApproveHolidays, Locked, Password,
PersonalPhoto, UnSuccessfulLoginAttempts

RolePrimary key FDs

RoleId → RoleType

Candidate key FDs

RoleType → RoleId

ExpensesMnemonics

Name → Amount, Mileage

BankAccount

StaffId → AccountNumber, sortCode, BankName

HomeAddress

StaffId → AddressLine1, AddressLine2, Country, County, HouseNumber,
HouseName, Locality, PostCode, Town

(Although real addresses are dependent on the postcode as well, we are not modelling addresses in this project as there is no facility to validate addresses, and they simply considered an attribute of the staff members, for example if two members of staff live at the same address the address will be stored in the database twice, one for each staff member. To model the address as a foreign key in the staff member relation requires the use of the full postal code database which is not feasible for this project)

Task

Id → Completed, DateCreated, Description, TaskType, Title

EmploymentDetails

Id → DateJoined, DateLeft, EmploymentType, HolidayEntitlement, Salary

GradePrimary key FDs

Id → Code, MaximumSalary, MinimumSalary, Name

Candidate key FDs

Code → Id, MaximumSalary, MinimumSalary, Name
Name → Id, MaximumSalary, MinimumSalary, Code

WorkStream

Primary key FDs

Id → Description, Name

Candidate key FDs

Name → Description, Name, Id

Project

Primary key FDs

Id → Code, Description, Name

Candidate key FDs

Code → Id, Description, Name

Name → Code, Description, Id

HolidayYear

Id → CarryOver, DaysInLieu, Entitlement, HolidayYear

Holiday

Id → AfterNoon, BookedDate, FromYear, FullDay, Status

Expenses

Id → DatePaid, StatusLastModified, Status, ExpensesMonth, ExpensesWeek,
ExpensesYear

ExpensesItem

Id → Amount, Description, ExpenseDate, Miles, Mnemonic, Rejected,
RejectionReason

ExpensesCategory

Primary key FDs

Id → Category, HasMileage

Candidate key FDs

Category → Id, HasMileage

MileageCost

Id → Cost, LowerLimit, UpperLimit

7.4.5 Relational Database Model

The logical database design was carried out using relational modelling. This is a representation of all the relations and constraint independent of any physical implementation. A number of domain definitions were used in the relational model mainly to define custom data for enumerations such as genders, titles, and the various status types for holidays and expenses. This is similar to the use of Java Enum for the classes representing these database entities.

Entity subtypes were mapped using a single table per class hierarchy. For example for the Role entity and its subtypes RegularStaff, Accountant and Administrator, a Role relation was used with a discriminator column "RoleType". RoleType itself was declared as an alternate key to enforce the fact that only one row should exist for each subtype. The relational database model for application is included in Appendix E.

7.4.6 Physical Database Model

The physical database (Appendix F) model was implemented using PostgreSQL relational DBMS for its support to a large part of the SQL standards such as integrity constraints and domain definitions. This model was designed by the translation of the logical data model to suit the PostgreSQL DBMS. The steps followed during the physical database design can be summarised as follows [8]:

- Design base relations
- Design general constraints
- Analyse transactions
- Choose indexes.

Design base relations

For each of the relations in the logical model a table was implemented using the SQL CREATE TABLE statements. Some of the table were named differently as the relation name corresponded to a reserved SQL keyword such as relation Role, for which the tables was named Roles. Other relations such as ViewableStaffDetails, UpdateableStaffDetails and ViewableStaffDetail were created as role_view_staff_details, role_update_staff_details and role_allowed_expenses_statuses respectively to indicate that these are child tables of the roles table. Mandatory table columns were declared as NOT NULL. The data types used are described in Table 21 below alongside their Java counter arts.

Table 21 – Data types for the various models

Java type	Relational model type	Physical model type
String	string	VARCHAR(255)
Decimal	decimal	NUMERIC(19, 2)
Integer	integer	INTEGER
boolean	boolean	BOOL
Date (date only without time)	date	DATE
Date (with time)	timestamp	TIMESTAMP
Enum	Domain definition	Domain definition with check constraint

A database sequence named the `hibernate_sequence` was created to support the automatic generation of surrogate primary keys. The Hibernate ORM uses this sequence to increment the primary key columns annotated in the Java code as using the `@Id @GeneratedValue` annotation.

Designing general constraints

A number of constraints were implemented such as primary keys, foreign key, unique and check constraints. All the constraints from the logical model were translated into SQL statements except the constraint that indicate the mandatory participation on both sides of the relation. These constraints were shown in ER and logical model to indicate that all the records in both tables should participate with each other. The problem with this type of constraints arises when inserting new rows in both tables, the constraint will be violated each time a new row is inserted. This is because a row entry can not be inserted simultaneously in the two tables. It has to be inserted in one table then the other table which will violate the constraint in this case.

These mandatory participation constraints were included in the logical model to indicate that this is how the data should be stored. For example there must not be a record in `StaffMember` that does not have a `HomeAddress` or `BankAccount`. But, it is not possible to enforce this using a Check constraint. This can be achieved using a stored procedure to insert records on a number of tables at once. This stored procedure can then be exposed to clients to use to insert data into the database. In our case the ORM layer manages this by inserting data into a number of tables as one transaction as explained below.

Analyse transactions

To analyse the database transactions we consider the repository classes discussed in subsection (7.1.1). The following tables have a repository defined in the Java code and hence are loaded and updated as parent tables:

- Staff_Member
- Roles
- WorkStream
- Grade
- Expenses
- Task
- Holiday_Year
- Expenses_Category.

The ORM layer will load these tables and their child tables using join statements, and understanding these transactions helps in implementing indexes to improve performance. Appendix G shows sample queries used by the ORM layer to load an instance of StaffMember, these join transactions for StaffMember class are summarised below in the format “Main table → Joined tables” (Joined tables will also join with their child tables):

Staff_Member → Home_Adress, Bank_Account

Employment_Details → Grade, Employment_WorkStream, WorkStream, Project, Staff_Member (for line manager object)

Users → Role, Expenses_Mnemonics, Staff_Member (for expenses approver), Staff_Member (for holiday approver)

Choosing indexes

The PostgreSQL DBMS supports a number of index types, one of which is the B-tree index created by default for primary key and unique constraints (alternate keys). The B-tree index is sufficient for the performance required by the application as most queries are simple queries selecting from tables by primary or alternate keys.

7.5 Caching, Pooling and Transactions Support

7.5.1 Caching

To improve the performance of the web application data caching was used. The data caching was implemented using EHCache [44], a general purpose caching framework. The cache used for the application is a simple in memory cache that only expires when an object is updated. The cache was defined on the methods in the repository classes that retrieve the objects. Whenever an object is requested it is checked in the cache first using its object id and if it is not found it will then be loaded from the database and cached. For this reason all the entity classes in the application implement the Serializable Java interface to indicate that they can be serialized to and from the cache. The cache is flushed whenever an instance is deleted or updated. Caching is configured in the following two files:

- /OfficeMA/WebContent/WEB-INF/ehcache.xml
- /OfficeMA/WebContent/WEB-INF/applicationContext.xml

```
<cache name="officeMACache"
      maxElementsInMemory="500"
      eternal="true"
      overflowToDisk="false"
      memoryStoreEvictionPolicy="LFU" />
```

Figure 69. EHCache configurations

7.5.2 Connection Pooling

Connection pooling was also used to improve performance and cache a number of database connections initially set to 5. The connection pooling saves the time required by the application to create, establish, and then close down a connection each time a request is made to access the database. The DBCP connection pooling component from the Jakarta project [49] was used. The connection pooling was managed by the Spring framework, however this can also be implemented to be managed by the Tomcat container. Pooling is configured in the following file:

- /OfficeMA/WebContent/WEB-INF/applicationContext.xml

```

<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource" destroy-method="close">
  <property name="driverClassName" value="org.postgresql.Driver" />
  <property name="url" value="jdbc:postgresql://localhost/officema" />
  <property name="username" value="officema" />
  <property name="password" value="sayian" />
  <property name="initialSize" value="5"/>
  <property name="maxActive" value="10"/>
</bean>

```

Figure 70. DBCP connection pooling configurations

7.5.3 Transactions

Database transactional support is by all means the most important factor in ensuring the data integrity. The PostgreSQL database is fully ACID compliant, however the application needs to manage the application level transactions and decide when to commit or rollback a transaction. The OfficeMA transactional requirement is to be able to save or persist an object graph into the database, for example a StaffMember instance with associated objects. This operation will update a number of tables and needs to be treated as a single transaction, and in case any update fails the whole operation will need to be rolled back.

The Spring container was used as a transactional framework by declaring all the repository classes as transactional using the Spring `@Transactional` annotation [61]. This ensures that all the methods in the repository class are executed as transactions that will be committed only if no exceptions such as `RuntimeException` occurred.

7.6 Security

Security is one of the big factors affecting the decision to implement a Rich Internet Applications. RIA have many advantages in regards to usability, flexibility, better user interaction and experience, but this comes at a price. By developing a RIA and following an MVC pattern on the client side a great deal of application logic is exposed on the client side in the form of JavaScript code. Some of the risks facing such applications and a way to mitigate this risk are discussed below. Having said this, each RIA should be assessed in terms of requirements and if the advantages of using a RIA outweigh the disadvantages and security risks.

Traditional web application security methods can be used to secure RIAs, but these are not enough and RIA developers will need to take extra care when developing such applications as Edwards [14] has summarised that:

“Although AJAX does not actively make security worse in web application, its approach to software design can encourage mistakes. Developers need to pay more attention ...”

Traditional Web Applications Scanners (WAS) that are not AJAX or JavaScript ready, fail to traverse and detect vulnerabilities in RIA. A recent scan that was carried out on the OfficeMA using the Acunetix [37] WAS has only managed to scan the login page, but failed to scan the rest of the application as the URLs are loaded using the JavaScript function `window.setTimeout()`, so a new generation of WAS that can understand JavaScript and AJAX is needed, in the meantime the developers have to follow good programming practices to ensure these types applications are secure.

Below is a summary of the main security issues that face the RIA, some of which are outlined by the OWASP as being in the top 10 vulnerabilities for 2007 [57].

- Insecure Communications
- Session Hijacking
- JavaScript hijacking
- JavaScript tampering
- SQL Injection, Remote file inclusion and Cross-site scripting

For each of these security risks a solution to eliminate or minimize the risk is discussed below.

7.6.1 Insecure Communications

A common solution for this type of problem is the use of encryption in the form of SSL over HTTP or HTTPS normally used to secure traditional web applications. This ensures that the communication channel between the client and the server is secure. HTTPS can be implemented in Tomcat application server or by using the Apache Web server [38] as a front-end for Tomcat (Figure 71).

This approach has many advantages such as the added advantage of using Mod-Security [54] on Apache. Mod-Security is a well know robust and effective web application firewall that can be used to secure the application. Another added advantage is the ability to provide load balancing between a number of Tomcat servers behind the Apache server hence scaling up and adding resilience to the whole solution. Load balancing is achieved by using the new mod_proxy, and mod_proxy_balancer [39] Apache modules which support the Tomcat AJP protocol.

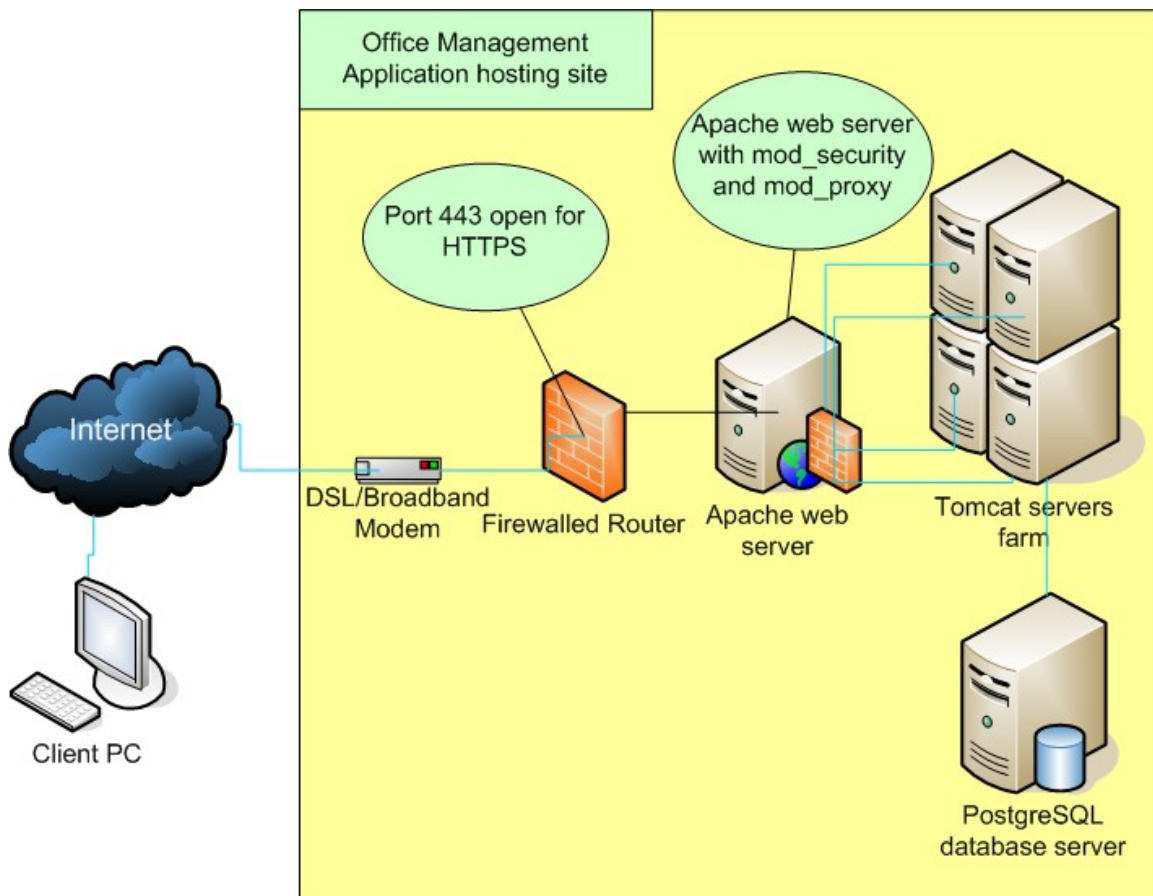


Figure 71. Securing OfficeMA application with Apache Web server

7.6.2 Session Hijacking

This vulnerability occurs when an attacker steals the session information for a logged in user. The session information is usually stored in a browser Cookie. In the case of RIA this risk is amplified because the user interface runs entirely on the client side and the only way for the server to determine if the client is legitimate or not is the session.

An approach that can be used to make RIAs more secure is the use of a unique token per user which will be generated when the user login and sent to the client side JavaScript to store, the client then sends this to the server on each Ajax transmission and the server will validate this token against the one stored in the session to validate the legitimacy of the client [7]. An even more secure approach is for the server to generate a token on each client's request and sends it back to the client. The client then sends this token back to the server upon subsequent request. So stealing the session information alone will not be enough to retrieve data from the server

7.6.3 JavaScript Hijacking

JavaScript Hijacking [7] is an issue that was identified with the use of the JSON notation for transporting JavaScript in Mozilla based browsers. In this vulnerability the hacker can trick the current user into visiting a malicious website where the hacker overrides the constructor for the super class of all JavaScript objects and hence be able to steal sensitive data in JSON format and sends it to the attacker. The solution for this is use some text that will prevent the server response from being constructed in JavaScript objects using the eval function such as using JavaScript comments `/* */`. The application on the client side will need to remove these comments first before executing the eval function on the response.

7.6.4 JavaScript Tampering

By most this is the biggest security issue when using a RIA due to the fact that JavaScript files are transferred to the client side and can be viewed using some browser utilities (Appendix N). The hacker can investigate the JavaScript and tries to understand the inner workings of the server or the URLs that are invoked by the client, although the hacker has to be logged in to be able to invoke any operations on the server. The hacker can then try to perform some action based on this knowledge. The solution for this problem can be summarised as follows:

- The use of JavaScript obfuscation, this will make the JavaScript very hard to read. A gained advantage of this approach is improved loading time for the JavaScript classes as obfuscation also compresses the files. The

developers can work on a fully commented JavaScript code, but this code is then obfuscated before being build into the deploy file.

- Implementing all the business logic, authorisation, authentication and access control on the server side. Relying solely on the client side to validate the user input or to check access roles for the user in RIA is suicide as the hacker can tamper with the JavaScript code and tries to carryout tasks not allowed for role currently used.
- For the hacker to be able to tamper with the JavaScript and invoke this on the server the user need to be logged in, so in this case the server can take an approach where none of the JavaScript code is transferred to the client unless the user is logged in.

7.6.5 SQL Injection, Remote file inclusion and Cross-site scripting

These vulnerabilities are widely known in the traditional web applications and are manifested in the form of injecting some parameters in the request to the server. To encounter such threats a Web Application Firewall (WAF) such as using Apache with ModSecurity as a front-end for Tomcat reduces such a risk by applying negative filtering to parameters supplied by the user. For J2EE applications an application level filter such as Stinger [56] can also be used for application level security. Input validation on the server side is very important in encountering these threats this validation should include type and range checking and all the special characters should be encoded.

7.7 Deployment

The deployment model for the application is shown in Figure 72 below. The application uses three nodes, the PostgreSQL database server, the Tomcat application server and the client's computer and contains two artefacts the **officema.sql** file which contains the data definition language used to create the database for the application and the **OfficeMA.war** Web Archive file that contains the Java classes and required libraries for the application to function. As shown in the diagram below the OfficeMA.war artefact manifests [5] a number of components that runs the application. The database and the application deployment instructions are provided in the installation guide (Appendix H).

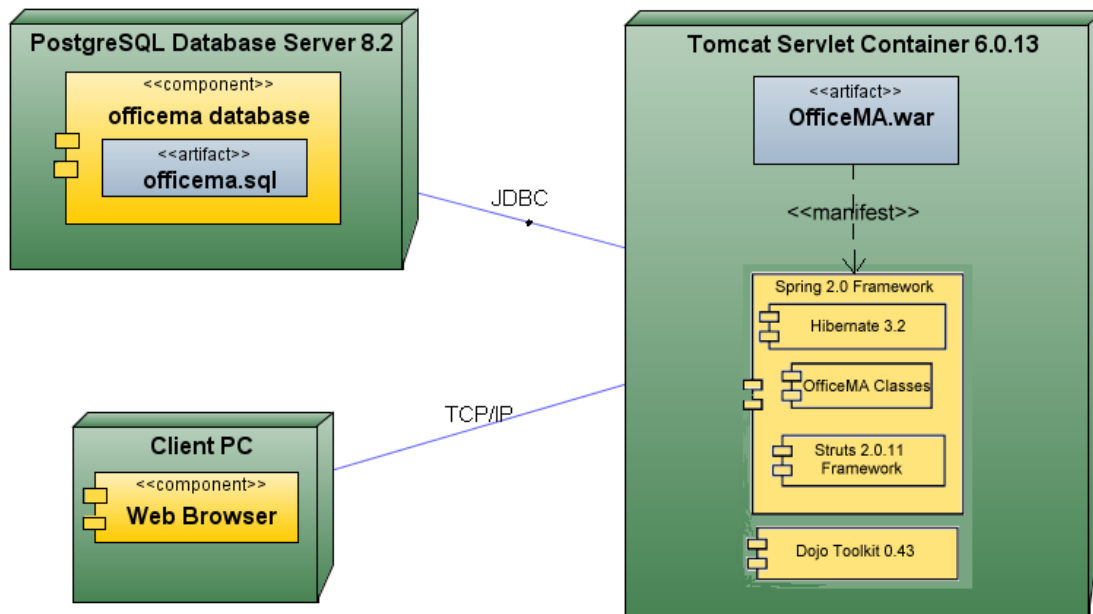


Figure 72. Deployment diagram for OfficeMA

7.8 System Testing

Testing is an important phase in the software development lifecycle. The unit testing carried out for the project code and outlined in subsections (6.3.6 and 7.2.7) has followed a white-box testing [70] approach that concentrated on testing a single unit or block of the application code. This type of testing is sufficient to ensure that a block of code behaves as it is expected, however once all these blocks are put together to form the overall system an end to end system testing is needed. This system testing on the other hand follows a black-box approach and concentrates on the behaviour and the functional requirements of the system. In this project the functional requirements are described using the high level use cases.

In the case of Rich Internet Applications, extra testing is required to validate the user interface and to ensure cross-browser support [29]. Testing the applications started from the business requirements captured in the use cases. The testing steps followed the use case to check that the application satisfied the business requirements. The test strategy followed can be summarised in the steps below:

1. For each use case tests were carried out to validate that the application satisfied the business requirement summarised by the use case.
2. For each carried test the results were checked to ensure they were correct. This was done by checking the results on the user interface, the application log files and finally the database using the tool summarised in Appendix J.
3. For each of the bugs found an issue was raised the Google Code project page for the application (Appendix L). The priority of the bug was set accordingly to its severity.
4. After the test cycle was completed, the raised issues were investigated using Java debugging and the browser tools summarised in Appendix N. High priority bugs were fixed first. Lower priority bugs or nice to have features were fixed if time allowed.
5. The fixed bugs were then tested individually to ensure they were fixed.
6. A new release was then build and deployed into the test environment
7. Step 1 – 6 were repeated again until there was no bugs left or the priority of the remaining bugs was low and they were accepted by the business. These steps were also repeated on all the browsers supported by the application. In the case of this project testing was done on three browsers, Firefox, Internet Explorer and Safari.

The remaining bugs in the application delivered as part of the project are summarised in Appendix H. These bugs were not considered critical by the client and do not hindering the functioning of the application; hence they will be fixed in a subsequent release.

8 Evaluation

The software application developed as part project was evaluated in terms of the following:

- Satisfaction of business requirements by comparing the prototypes with the final developed screens
- Accessibility in terms of browsers support and screen resolution.
- Usability, by taking users' feedback on the usability rules identified in subsection (2.6.3).

8.1 Satisfaction of business requirements

In the case of the application developed as part of this project it is clear that the business requirements for the Staff, Tasks, Expenses and System Settings modules implemented have satisfied the business requirements. This is obvious from the screenshot provided in the user guide in Appendix I and the prototypes provided in Appendix C and as a direct result of using the USDP for the design of the application. One of the advantages of using the USDP and use case driven modelling is the fact that the developed application can be traced back to the use case and the business requirements.

8.2 Accessibility

When the development on the application started it was intended to support all major browsers, however due to time constraints this was not practical. Another accessibility issue was the screen resolution used the various staff. The project has found that any screen resolution below 1024 X 768 will not give enough screen space to implement the various functionality specially with the 'Add Expenses' window which requires a minimum width of 1000 pixels. For this reason it was decided that the application will only support a minimum resolution of 1024 X 768. Given the nature of this project and the fact that the employer can dictate which browser and screen resolution the staff should use to run the application, it was decided to only support a minimum screen resolution of 1024 X 768, and to only support the following browsers:

- Firefox 2.0
- Internet Explorer 6 and 7
- Safari 3.0.

After performing a number of tests using the above browser it was discovered that Firefox has fully supported the application and has shown high performance compared to Internet Explorer 6. Testing on Safari also indicated that the support for the application is quite well apart from one issue which is summarised in the list of outstanding issues in Appendix I. Testing carried on Internet Explorer on the other hand has revealed that this browser was the one with the least performance when running the application compared to the other two browsers.

Many issues have been encountered when running the application using Internet Explorer and these are summarised in Appendix I. If these issues are considered to be serious and time consuming to fix, the support for Internet Explorer can be dropped so that future development of the application can only focus on Firefox and Safari browsers. The project has decided to drop support for the Opera browser due to the time constraints and the numerous issues that were encountered when trying to run the application.

8.3 Usability

The author has interviewed a number of staff after they have used the system for the first time. Users' feedback was then categorised in terms of the usability rules identified in subsection (2.6.3).

8.3.1 Visibility, Affordance and Consistency

Most of the users interviewed have highlighted the fact that the look and feel of the application has automatically conveyed to them the way it should be used. The menu and tool bars were an obvious way of invoking the various functionality of the application. Most users also found the use of the tool-tip on the menu bar very helpful in identifying the purpose of the tool buttons. Users also found that the ability to view a number of windows at the same time very helpful in regards to productivity and multitasking and many were pleased that they do not have to navigate away from one window in order to view another.

Almost all users interviewed did not believe that the application was a Web application, and most of them thought of it as being a desktop application similar to Word and Excel. A few users have suggested the ability to configure the shortcut button on the toolbar so that these can be configure per user rather than be a fixed list.

8.3.2 Closure, Tolerance and Feedback

All the users interviewed have indicated their satisfaction with the way the window and the dialogue boxes function in the application. Users have indicated that the instant feedback on invalid data and expected format when filling forms have enabled them to effectively use the application without unnecessary frustration. The users also indicated that the feedback provided by the application is the form of a loading image or a dialogue box was clear and successfully conveyed to them the current status of the system, which made the users feel in control.

8.3.3 Performance and Data Refresh

Users interviewed were impressed and pleased with the performance of the application. This has enabled the users to efficiently use the application to complete their tasks, submit their expenses or update their details. Many users found that the ability to be able to check their details, others details and manage their expenses efficiently over the Internet very helpful especially in an office with a number of staff who work remotely. However, some users have indicated that as they open a large number of applications on their computer the performance of their browser became sluggish when running the application.

8.4 *Evaluation Summary*

It clear from the application evaluation performed that the application has successfully satisfied its requirements and objectives. The user interface developed has enhanced the users' productivity and experience compared to a static Web application using conventional Web pages for display. Having said this, the application was developed within a constraint environment such as limited browsers support and minimum resolution, the performance of the application is also subject to the performance of the browser and the computer running the user interface. These constraints are the price of utilising some of the desktop clients' features in the Web application as now it will also be constraint with some of the constraints that affects traditional desktop applications. However, in a controlled environment such as workplace the minimum application requirements can easily be met.

9 Conclusions

The proposal of this project has started from the requirements to develop a Web application that can be used to track and manage the expenses for the staff in a small office. However, the author decided to design and implement a full Office Management Application that can be used by small businesses to manage their staff. As part of this dissertation the author has conducted a survey of small businesses to determine the current processes in place used to manage their staff, expenses, holidays and timesheets. The survey has found out that most of these businesses rely on manual processes based on spreadsheet or paper forms. The main reason behind using such methods was the cost involved in trying to purchase and install one of the established software products (Table 3). The project has successfully satisfied the objectives set as follows:

- The project has successfully investigated the possibility of applying the USDP boundary classes methodology used to design traditional desktop application in designing and modelling Rich Internet Application user interface. And has Devised a methodology and applied it to the design and implementation of the OfficeMA user interface
- The project has successfully gathered and analysed the business requirements for the Office Management Application using the USDP and provided the analysis and detailed design UML models in subsections 5.3 and 5.4 and Appendices B and D.
- The project has successfully identified the data requirements for the application and used the relational database design theory to provide the Entity-Relations, relational and physical database models for the application as outlined in subsection 7.4 and Appendices E and F.
- The project has successfully implemented, tested and evaluated the application using Open Source technologies outlined and Appendix I, and provides the source code, the binaries and the user documentations for the application in Appendix H.

9.1 Project Achievements

The project has successfully devised and followed a new design methodology to design and implement the Office Management Application which satisfied the original client's requirements and utilised a number of Open Source technologies to be cost effective. The project has carried out the design and implementation of the business logic the user interface and the database for the application and in doing so explored and utilised a wide range of software design methodologies

and Open Source frameworks. The project has also laid the foundation for the continuing development of the Office Management Application to become a mainstream application for small businesses.

The project has also assessed a number of technologies such as application persistence requirements using a persistence framework such as Java Persistence API. Based on this new Java standard the project has provided a mechanism (summarised in Table 20) by which the Entity-Relationship model can be mapped to and translated into Java Annotations so that the Java objects can easily be serialised to the database tables. Also a transactional framework was needed to coordinate access to the domain objects and manage the security and persistence; a framework such as Spring was used and evaluated against Enterprise Java Beans.

9.2 *Project Issues*

The project was hoping to implement an application that incorporates all the modules identified by the author, however due to time constraints only the Staff, Expenses, Tasks and System Settings modules were implemented. Nevertheless, the project has gathered the requirements for and designed the other modules and future work will be done to implement these remaining modules in a subsequent version of the application. The time constraints on the project have risen from the fact that the project has favoured exploring new avenues for Web development and establishing some methodologies rather than using the conventional Web development methodologies. In doing so the project has faced many issues in regards to utilising and using a wide range of technologies, in particular the limitations of HTML and the incompatibility between the various browsers. The fact that what works in one browser seems to either not work or work differently in other browsers, was one of the major issues faced during this project.

9.3 *Contributions of this Dissertation*

9.3.1 *Problems with adapting functional-oriented UI as content-oriented Web UI*

Besides satisfying the requirements of the Office Management Application the author wanted to investigate the possibility of using Web 2.0 concepts in particular Rich Internet Applications to design and implement the application. Rich Internet Applications are Web based applications that possess many features that are similar to traditional desktop applications, hence utilizing the

features of both worlds. One of the problems that have been identified by this project is that lack of a clear design methodology to design and implement such types of applications in particular the design of the user interface. Using Web pages design principles in an attempt to design functionality-oriented rather than content-oriented user interface adds many complications to the design process, which can be summarised as follows:

- Complex Web pages that relies heavily on the server, which results in poor user experience, extra load and complex logic on the server.
- The in-ability to model together the interactions between the user, the user interface and the application logic in the server, which usually results in designers adopting none standard methods in attempt to bridge the gap between functionality and contents.
- Trying to fit the functional-oriented nature of the user interface into Web pages results in a great deal of desired interactive functionality to be dropped due to the limitations of what Web pages can do.

9.3.2 Utilising the Web as a functional user interface

The root cause of the above complications is due to the hypertext nature of the Web which is oriented toward contents and information rather than functionality. However, although the Web is intrinsically a hypertext medium, the foundations are there for a functional oriented medium similar to the one used for desktop applications as demonstrated in the Table 2.

To address the limitations the project has come up with the idea of adopting an Object-Oriented Web user interface so that the traditional established methodologies can be applied. An Object-Oriented user interface is very close to traditional desktop applications that are developed using Object-Oriented languages such as Visual Basic or Visual C. Methodologies such the ones outlined by Bennett et al. [4] can easily be applied to design and implement such a user interface. The project has successfully devised a methodology to design and implement an Object-Oriented rich user interface in HTML, CSS and JavaScript which can be summarised below (subsection 6.4.2).

The design methodology which is largely based on the concepts outlined by Bennett et al. [4]:

- Prototyping the user interface using the RIA techniques outlined by Dawelbeit [11].
- Designing and elaborating the boundary classes that represent the various widgets on the user interface. Some of these classes will be developed and some already exists as DOM objects.

- Modelling the interaction involved in the interface using interaction or communication diagrams. The interactions modelled should also include the objects on the server that will be handling the messages.
- Modelling the control of the interface using state machines for complex UI components.

The implementation methodology which was developed as part of this project:

- Choose the Web development toolkit also called AJAX toolkit to be used, for example Dojo toolkit as used in this project. Most AJAX toolkits available are Object-Oriented and based on JavaScript.
- For the DOM objects in the boundary classes diagram define the widget template which consists of HTML, CSS and other widgets. This will later on be constructed as DOM nodes in the browser.
- For the other objects in the boundary classes diagram define the View Support Object (VSO) for the widget in using Object-Oriented JavaScript.
- Implement the methods required for the VSO to control the DOM nodes. These methods follow from the user interface models such as boundary classes and interaction diagrams. Some of these methods should use the DOM event model to trap the user's actions.

Advantages of this approach

Below is a summary of the advantages in following the approach summarised above to design and implement functionality-oriented Web user interfaces:

- As the display is not generated by the server upon each request the client can be designed to work offline and have the ability to connect to the server when needed.
- The ability to offer greater usability and interaction to the end user as the user interface is rich and self sufficient.
- Using the server to only transfer data, increases the bandwidth available to serve more client hence enabling scalability and high server performance.
- The capability to model the user interface along side the business logic enables better understanding of the application logic and makes future changes to the applications a lot easier. This results in time savings in developing the final product.

Disadvantages

- With a great deal of logic written in JavaScript and transferred to the client side there is a risk of concealed vulnerabilities and possible security issues such as JavaScript Hijacking and Tempering.
- The approach used in this project relies on the browser capabilities which are limited. To utilise the operating system features such 3D hardware acceleration and local storage a browser plug-in will need to be used such as Java Applets and Flash.
- Trying to adopt this approach requires a high level of experience and proficiency in Web development to overcome the HTML limitations.
- Cross-browser compatibility adds extra cost during implementation and testing for projects adopting this approach.

Summary

It is clear from the advantages and disadvantages presented above that this approach to Web development is only suitable and applicable for functional-oriented Web applications that are used by a limited number of people on a regular basis to achieve some specific tasks. These applications are also likely to be restricted to a secure environment and offer limited browsers support. Browser support and compatibility in itself is one of the big issues that might hinder the adoption of these methodologies for mainstream use, however the use of browser plug-ins such as Microsoft Silverlight or Adobe AIR [11] rather than directly using the browser capabilities, might well be the answer to this issue and the way forward.

9.4 *Suggestions for Future Work*

The project has designed and implemented a HTML based Rich Internet Application and has explored a wide range of principles and methodologies in the software design arena. As this type of applications is relatively new and have not got established principles and methodologies, a great deal of work during the project went towards defining a methodology that can be followed. Although the project has tried to touch all the issues concerned with Rich Internet Applications, it has barely scratched the surface in many areas. Below are suggestions for future work in the areas touched on during this project

9.4.1 Usability and Accessibility of RIA

Usability and accessibility is an area where work has been done for interactive user interfaces and Web sites. Rich Internet Applications are a hybrid of both desktop and Web applications and need a customized set of usability design principles and rules that can be followed. This project has attempted to drive a set of rules by combining the Web design principles with the traditional user interface design principles. The project has also managed to carryout some evaluation on the usability of the application after the initial use by the users; however further future evaluations will be needed as the users become proficient in using the application.

9.4.2 Performance of RIA

Another area that will need investigation and quantative measures is the performance of Rich Internet Applications such as the Office Management Application. It was observed as part of this project that transferring the HTML to the browser only once and transferring only data thereafter using JSON has improved the performance of the application considerably, however future work will be needed to provide actual statistics of the performance of Rich Internet Applications compared to traditional Web applications and desktop clients. This measurement could possibly be performed on three types of application using the same server to ensure the performance being measured only relates to the user interface and its functioning.

9.4.3 Enhancements to the Office Management Application

Future work can also be carried out to implement the remaining modules of the Office Management Application such as the Holidays and Timesheet modules and provides the ability to generate reports. The project has gathered the requirements and carried out the design for these remaining modules. Future work can be done to implement these modules using the same approach followed in this project.

10 References

10.1 Books and Articles

- [1] Arrington, C.T. and Rayhan, S.H. (2003). *Enterprise Java with UML*. 2nd ed. Wiley
- [2] Bauer, C. and King, G. (2007). *Java Persistence with Hibernate*. Greenwich: Manning Publications.
- [3] Bell, D. (15 Jun 2003) *UML basics: An introduction to the Unified Modelling Language*. URL: <http://www-128.ibm.com/developerworks/rational/library/769.html>
[30 September 2007]
- [4] Bennett, S. McRobb, S. and Farmer, R. (2006). *Object-oriented systems analysis and design using UML*. 3rd ed. London: McGraw-Hill Companies.
- [5] Bennett, S. Skelton, J. Lunn, K. (2004). *Schaum's Outline UML*. 2nd ed. McGraw-Hill.
- [6] Bracha, G. (2004). Generics in the Java Programming Language. URL: <http://java.sun.com/j2se/1.5/pdf/generics-tutorial.pdf>
[30 September 2007]
- [7] Chess, B., O'Neil, Y. and West, J. (2007). *JavaScript Hijacking*. URL: http://www.fortifysoftware.com/servlet/downloads/public/JavaScript_Hijacking.pdf
[10 February 2008]
- [8] Connolly, T. Begg, C. (2005). *Database Systems: A practical approach to Design, Implementation, and Management*. 4th ed. Addison Wesley.
- [9] Couch, J. and Steinberg, H. (2002). *Java 2 Enterprise Edition Bible*. Wiley.
- [10] Crane, D. Pascarello, E. James, D. (2006). *Ajax In Action*. Greenwich: Manning Publications.
- [11] Dawelbeit, O. (2008). *Web User Interface from Prototyping to Implementation*. URL: <http://change-vision.blogspot.com/>
[10 April 2008]

- [12] Douglas, N. (2007). Free end-users to cash in on Web 2.0. *Computer Weekly*. **22**, 16-16.
- [13] Elmasri, R. Navathe, S. (1999). *Fundamentals of database systems*. 3rd ed. Addison-Wesley.
- [14] Edwards, C. (2007). *Bandwagon: Has Ajax Over Exposed Itself*. IET Information Professional. **June/July 2007**, 10-11.
- [15] Evans, E. (2003). *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Boston: Addison-Wesley Professional.
- [16] Gadge, V. *Technology options for Rich Internet Applications*. URL: <http://www-128.ibm.com/developerworks/library/wa-richiapp/> [16 September 2007]
- [17] Gamma, E., Helm, R., Johnson, R. and Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley.
- [18] Garrett, J. J. *Ajax: A New Approach to Web Applications*. 18th Feb 2005. URL: <http://www.adaptivepath.com/ideas/essays/archives/000385.php> [20 September 2007]
- [19] Garrett, J. J. (2002). *The Elements of User Experience: User-Centered Design for the Web*. New Riders Press.
- [20] Glowiak, M. MySQL vs. PostgreSQL. URL: http://monstera.man.poznan.pl/wiki/index.php/Mysql_vs_postgres [11 February 2008]
- [21] Grand, M. (1998). *Patterns in Java, Volume 1, A Catalog of Reusable Design Patterns Illustrated with UML*. John Wiley & Sons, 2 vols.
- [22] Nielsen, J. (2002). Usability of *Ephemeral Web-Based Applications*. URL: <http://www.useit.com/alertbox/20021125.html> [12 February 2008]
- [23] Open University (1999). *M358 Block4 : Development of Database Systems*. The Open University, Milton Keynes.

- [24] O'Reilly, T. *What Is Web 2.0, Design Patterns and Business Models for the Next Generation of Software*. URL: <http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html?page=1>
[16 September 2007]
- [25] O'Reilly, T. *Web 2.0: Compact Definition?*. URL: http://radar.oreilly.com/archives/2005/10/web_20_compact_definition.html
[16 September 2007]
- [26] Pawson, R. (2004). *Naked Objects*. PhD thesis. Trinity College, Dublin.
- [27] Pettey, C. Goasduff, L. Gartner's 2006 Emerging Technologies Hype Cycle Highlights Key Technology Themes. URL: <http://www.gartner.com/it/page.jsp?id=495475>
[10 February 2008]
- [28] Rajagopalan, S. Rajamani, R. Krishnaswamy, R. and Vijendran, S. (2002). *Java Servlet Programming Bible*. Wiley.
- [29] Rymer, J. and Stone, J. (2007). Rich Internet Apps Move Beyond the Browser. Forrester's Reports. URL: <http://www.forrester.com/Research/Document/Excerpt/0,7211,42708,00.html>
[10 February 2008]
- [30] Richardson, C. (2006). *POJOs in Action, Developing Enterprise Applications with Lightweight Frameworks*. Greenwich: Manning Publications.
- [31] Shin, S. *Web Application Security Threats and Counter Measures*. URL: <http://www.javapassion.com/>
[12 February 2008]
- [32] Shneiderman, B. (1998). *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. 3rd ed. Addison Wesley.
- [33] Shneiderman, B. and Plaisant, C. (2004). *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. 4th ed. Addison Wesley.
- [34] Stone, D. Jarrett, C. Woodroffe, M. and Minocha, S. (2005). *User*

interface design and evaluation. Amsterdam ; London : Elsevier : Morgan Kaufmann.

- [35] Walls, C. Breidenbach, R. (2006). *Spring in Action*. 2nd ed. Greenwich: Manning Publications.

10.2 Web references

- [36] A government action plan for small business. Department of Trade and Industry. URL: <http://www.berr.gov.uk/files/file39768.pdf>
[12 February 2008]
- [37] Acunetix Web Vulnerability Scanner v4 (Consultant Edition). URL: <http://www.acunetix.com/vulnerability-scanner/>
[12 February 2008]
- [38] Apache Web Server and Modules. URL: <http://httpd.apache.org/>
[12 February 2008]
- [39] Core J2EE Patterns – Data Access Object. URL: <http://java.sun.com/blueprints/corej2eepatterns/Patterns/DataAccessObject.html>
[12 February 2008]
- [40] Crow's Foot notation. URL: http://en.wikipedia.org/wiki/Entity-relationship_diagram#Crow.27s_Feet
[12 February 2008]
- [41] Database design: Choosing a primary key. URL: <http://immike.net/blog/2007/08/14/database-design-choosing-a-primary-key/>
[12 February 2008]
- [42] Dojo Ajax Toolkit. URL: <http://dojotoolkit.org/>
[12 February 2008]
- [43] Dojo Toolkit Object oriented concepts and inheritance. URL:

- <http://dojotoolkit.org/book/dojo-book-0-4/part-3-dojo-programming-model/object-oriented-concepts-and-inheritance>
[12 February 2008]
- [44] EHCACHE general purpose caching framework. URL:
<http://ehcache.sourceforge.net/>
[01 October 2007]
- [45] Enum Java 5 feature. URL:
<http://java.sun.com/j2se/1.5.0/docs/guide/language/enums.html>
[01 October 2007]
- [46] Generics Java 5 feature. URL:
<http://java.sun.com/j2se/1.5.0/docs/guide/language/generics.html>
[01 October 2007]
- [47] Hibernate, Relational Persistence for Java and .NET. URL:
<http://www.hibernate.org/>
[12 February 2008]
- [48] Hibernate, Relational Persistence for Java and .NET. URL:
<http://www.hibernate.org/>
[12 February 2008]
- [49] Jakarta DBCP connection pooling component. URL:
<http://commons.apache.org/dbcp/>
[01 October 2007]
- [50] Java Persistence API blueprints. URL:
<https://blueprints.dev.java.net/bpcatalog/ee5/persistence/index.html>
[12 February 2008]
- [51] Java Persistence API Javadoc. URL:
<http://java.sun.com/javaee/5/docs/api/javax/persistence/package-summary.html>
[01 October 2007]
- [52] JavaScript Object Notation. URL: <http://www.json.org/>
[12 February 2008]
- [53] Microsoft Office SharePoint Server. URL:
<http://www.microsoft.com/sharepoint/default.mspx>
[12 February 2008]
- [54] Mod-Security Web Application Firewall. URL:

- <http://www.modsecurity.org/>
[12 February 2008]
- [55] Oracle Human Resources Management System. URL:
http://www.oracle.com/applications/human_resources/intro.html
[12 February 2008]
- [56] OWASP Stinger Filter for J2EE applications. URL:
http://www.owasp.org/index.php/OWASP_Stinger_Manual
[12 February 2008]
- [57] OWASP Top 10. The ten most critical web application security vulnerabilities, 2007 update. URL:
http://www.owasp.org/images/e/e8/OWASP_Top_10_2007.pdf
[12 February 2008]
- [58] PostgreSQL 8.2 documentation manual online. URL:
<http://www.postgresql.org/docs/8.2/static/index.html>
[01 October 2007]
- [59] Sage Business Support Services. URL:
<http://www.sage.co.uk/home.aspx>
[12 February 2008]
- [60] Spring Application Framework. URL: <http://www.springframework.org/>
[12 February 2008]
- [61] Spring Framework Transaction Management. Spring guide online, chapter 9. URL:
<http://static.springframework.org/spring/docs/2.0.x/reference/transaction.html>
[01 October 2007]
- [62] Struts 2 JSON plug-in. URL: <http://code.google.com/p/jsonplugin/>
[12 February 2008]
- [63] Struts 2 MVC Framework. URL: <http://struts.apache.org/2.x/>
[12 February 2008]
- [64] Struts 2 Request Flow. URL: <http://struts.apache.org/2.0.11/docs/the-struts-2-request-flow.html>
[12 February 2008]

- [65] The New Methodology. URL:
<http://martinfowler.com/articles/newMethodology.html>
[12 February 2008]
- [66] Tibco General Interface, Getting started guide. URL:
www.tibco.com/devnet/resources/gi/3_3/tib_gi_pe_getting_started.pdf,
pp14.
[01 October 2007]
- [67] Tomcat Java Application Server. URL: <http://tomcat.apache.org/>
[12 February 2008]
- [68] Tommie Web Office and Online Diary System. URL:
<http://www.tommie.co.uk/>
[12 February 2008]
- [69] UML Superstructure Specification, v2.1.1, p. 620. URL:
<http://www.omg.org/technology/documents/formal/uml.htm>
[01 October 2007]
- [70] What is black box/white box testing? URL:
<http://www.faqs.org/faqs/software-eng/testing-faq/section-13.html>
[12 February 2008]
- [71] Windows Vista User Experience Guidelines. URL:
<http://download.microsoft.com/download/e/1/9/e191fd8c-bce8-4dba-a9d5-2d4e3f3ec1d3/ux%20guide.pdf>
pp 2-10
[01 October 2007]

11 Appendices

11.1 Appendix A – Office Management Application's Modules Survey

Business	Industry	Size	Dedicated HR Staff	Staff Management	Holiday Management	Expenses Management	Timesheet Management	Pay Management	Computer Network
1	Construction	7	No	Paper based record keeping	Spreadsheet	Spreadsheet	Spreadsheet	Outsourced, done using SAGE	Yes
2	Construction	7	No	Paper based record keeping	Spreadsheet	Spreadsheet	Spreadsheet	In-house, done using SAGE	Yes
3	IT Consultancy	13	No	Spreadsheet	Spreadsheet	Spreadsheet	Web Application	Outsourced	Yes
4	Automotives	16	Yes	Spreadsheet	Spreadsheet	Spreadsheet	No timesheet	Outsourced	Yes
5	Automotives	50	No	Microsoft Access forms	Paper based	Paper based	Spreadsheet	In-house	Yes
6	Health Care	130	Yes	Held and updated by HR using Microsoft SharePoint	Done using SharePoing forms	Done using SharePoing forms	Done using SharePoing forms	Outsourced	Yes
7	Manufacturing	140	Yes	System managed by HR department	Spreadsheet submitted to HR	Spreadsheet submitted to HR	Automated clocking-in system	In-house accountants	Yes
8	Utilities Provider	230	Yes	Updated by HR using Oracle HR	Paper based form submitted to HR. Done using Oracle HR	Paper based form submitted to HR. Done using Oracle HR	No timesheet for office staff, field staff completes paper form.	Outsourced	Yes

11.2 Appendix B – Documents Sampling

11.2.1 Sample holiday control spreadsheet

Staff	Holiday	01-Jan	08-Jan	15-Jan
John Smith	12	H H H H H		
William Hans	5			
Julie Eldrige	3		H H	
Omer Dawelbeit	1			
Thomas Julian	5			H

11.3 Appendix C – Use Case Models

11.3.1 Add Staff Use Case

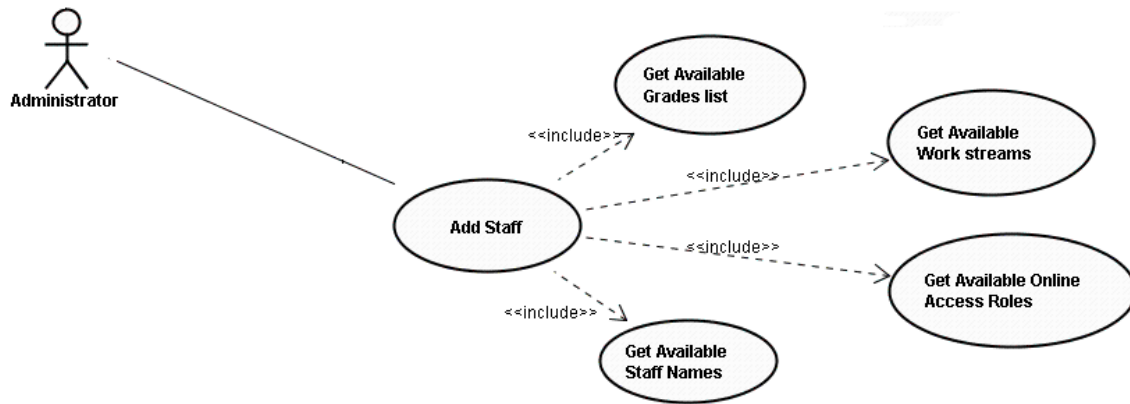


Figure 73. Add staff use case diagram

Name:

Add Staff

Description:

The administrator adds a new member of staff. First all the Available grades, work-streams, online access roles and staff names are retrieved. The admin user completes the personal, bank, address, employment and online details.

Assumptions

- Grades have been added to the system
- Online access roles have been added to the system.
- A number of work-streams have been added to the system.
- The required details for the new member of staff are available

Preconditions

- User is logged in to the system

Post-conditions

- The new member of staff is added successfully to the system and their details can be viewed/edited

Normal flow of events – The administrator successful adds a member of staff

- The administrator selects the add new staff option
- The administrator fills the personal, bank, address, employment and online details for the new member of staff

- The administrator clicks the add button
- A message is displayed indicating that the employee details are successfully added

Alternative flow of events – The administrator populate some invalid details

- The administrator selects the add new staff option
- The administrator fills the personal, bank, address, employment and online details for the new member of staff, but forgets some of the mandatory details.
- The administrator clicks the add button
- The system creates a task for the new staff member to change their password.
- A message is displayed indicating that some of the details provided are invalid; no change is made to existing data.

Alternative flow of events – Employee already exists

- The administrator selects the add new staff option
- The administrator fills the personal, bank, address, employment and online details for the new member of staff, but forgets some of the mandatory details.
- The administrator clicks the add button
- The administrator is notified of the conflict. No change is made to the existing data.

Alternative flow of events – Administrator cancels the process

- The administrator selects the add new staff option
- The administrator fills the personal, bank, address, employment and online details for the new member of staff, but forgets some of the mandatory details.
- The administrator clicks the Cancel button
- No change is made to the existing data.

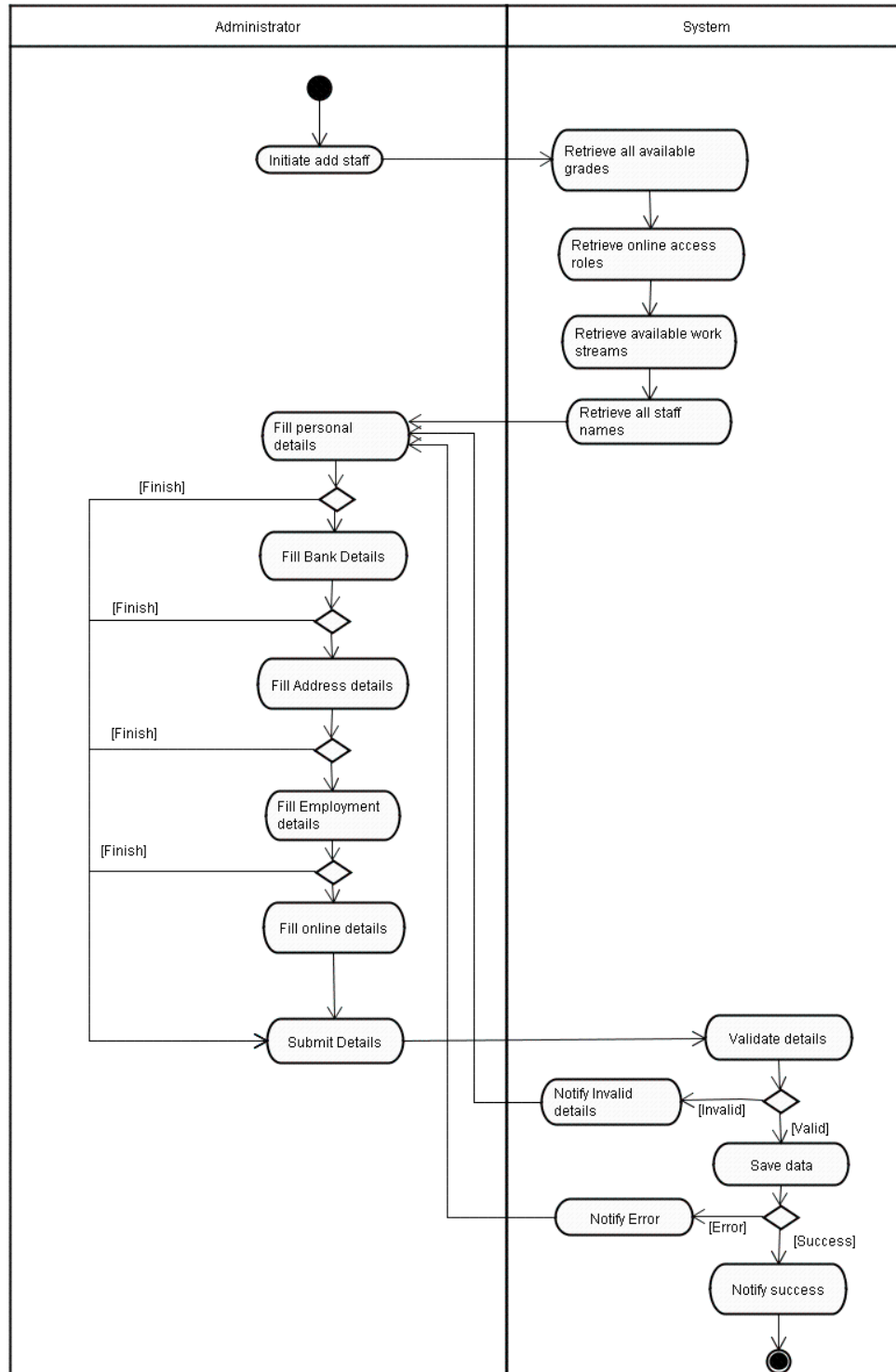
Exception flow of events – The system is unable to add the new staff details due to an error

- The administrator selects the add new staff option
- The administrator fills the personal, bank, address, employment and online details for the new member of staff, but forgets some of the mandatory details.
- The administrator clicks the add button
- The administrator is notified that the details can't be added to the system due to a system error.

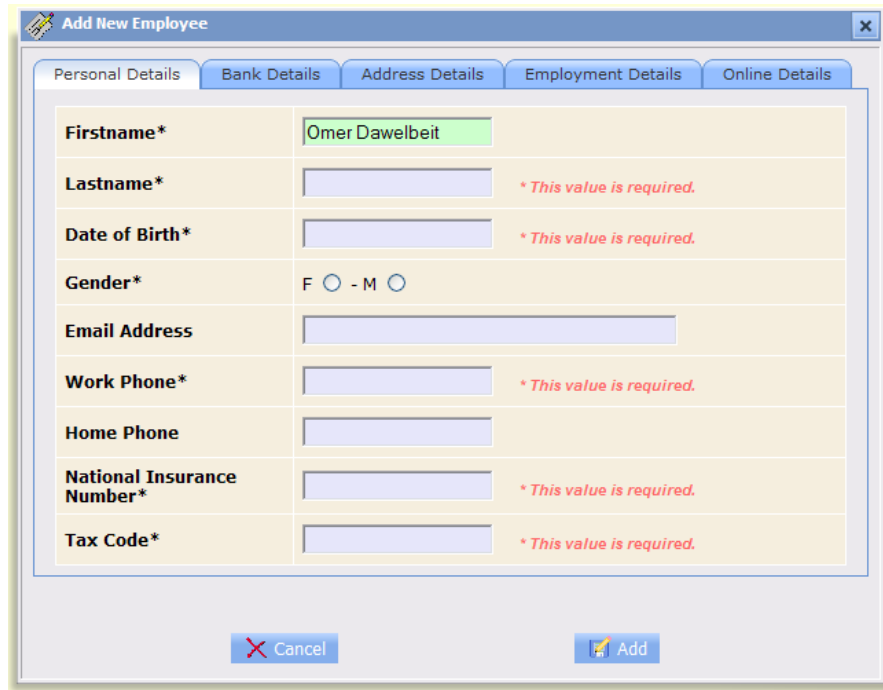
Notes:

Staffs sensitive details such bank account, etc... are stored in encrypted format

Activity Diagram



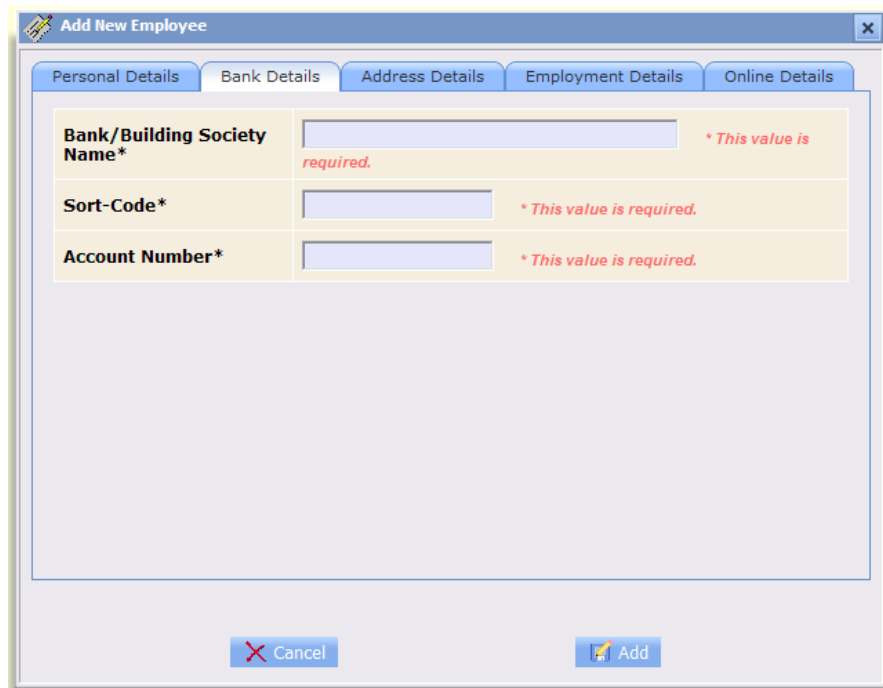
Prototype



Add New Employee

Personal Details | Bank Details | Address Details | Employment Details | Online Details

Firstname*	<input type="text" value="Omer Dawelbeit"/>	
LastName*	<input type="text"/>	* This value is required.
Date of Birth*	<input type="text"/>	* This value is required.
Gender*	F <input type="radio"/> - M <input type="radio"/>	
Email Address	<input type="text"/>	
Work Phone*	<input type="text"/>	* This value is required.
Home Phone	<input type="text"/>	
National Insurance Number*	<input type="text"/>	* This value is required.
Tax Code*	<input type="text"/>	* This value is required.



Add New Employee

Personal Details | Bank Details | Address Details | Employment Details | Online Details

Bank/Building Society Name*	<input type="text"/>	* This value is required.
Sort-Code*	<input type="text"/>	* This value is required.
Account Number*	<input type="text"/>	* This value is required.

Add New Employee

Personal Details
 Bank Details
 Address Details
 Employment Details
 Online Details

House Number	<input type="text"/>	
House Name	<input type="text"/>	
Street Line 1*	<input type="text"/>	* This value is required.
Street Line 2	<input type="text"/>	
Locality	<input type="text"/>	
Town*	<input type="text"/>	* This value is required.
County	<input type="text"/>	
Postcode*	<input type="text"/>	* This value is required.

Cancel
 Add

Add New Employee

Personal Details
 Bank Details
 Address Details
 Employment Details
 Online Details

Date Joined*	<input type="text"/>	
Employment Type*	Contractor <input type="radio"/> - Permanent <input type="radio"/>	
Line Manager*	<input type="text" value="Please Choose..."/>	
Grade*	<input type="text" value="Please Choose..."/>	
Salary*	<input type="text"/>	* This value is required.
Holiday Entitlement*	<input type="text"/>	* This value is required.
Work Stream*	<div> NovaTech New Portal IBM Bluesky O2 Content Management </div>	

Cancel
 Add

Add New Employee

Personal Details Bank Details Address Details Employment Details **Online Details**

Enable Online Access* ☐ Yes

Online Password

Confirm Online Password

Personal Photo **Browse...**

Online Role* Please Choose... ▼

Cancel **Add**

11.3.2 Find Staff Use Case



Figure 74. Find staff use case diagram

Name:

Find Staff

Description:

Search for a member of staff or browse all staff details. The user can search by name, id, employee type, work stream or project

Assumptions

- Staff details are added to the system.
- Only current staff details can be viewed by regular staff. The details for staff who left can only be viewed by administrators

Preconditions

User is logged in to the system.

Post-conditions

- The staff member finds the details of the staff member they are trying to find. Or none if the details searched for are not found

Normal flow of events – The staff member successfully find the details they are searching for, one result found

- The staff member selects the find employee option
- The staff member selects the search type.
- The staff member enters the search criteria details and click the search button
- The details for staff searched for are displayed

Alternative flow of events – The staff member successfully find the details they are searching for, more than one result found

- The staff member selects the find employee option
- The staff member selects the search type.
- The staff member enters the search criteria details and click the search button
- The records matching the search criteria are displayed.

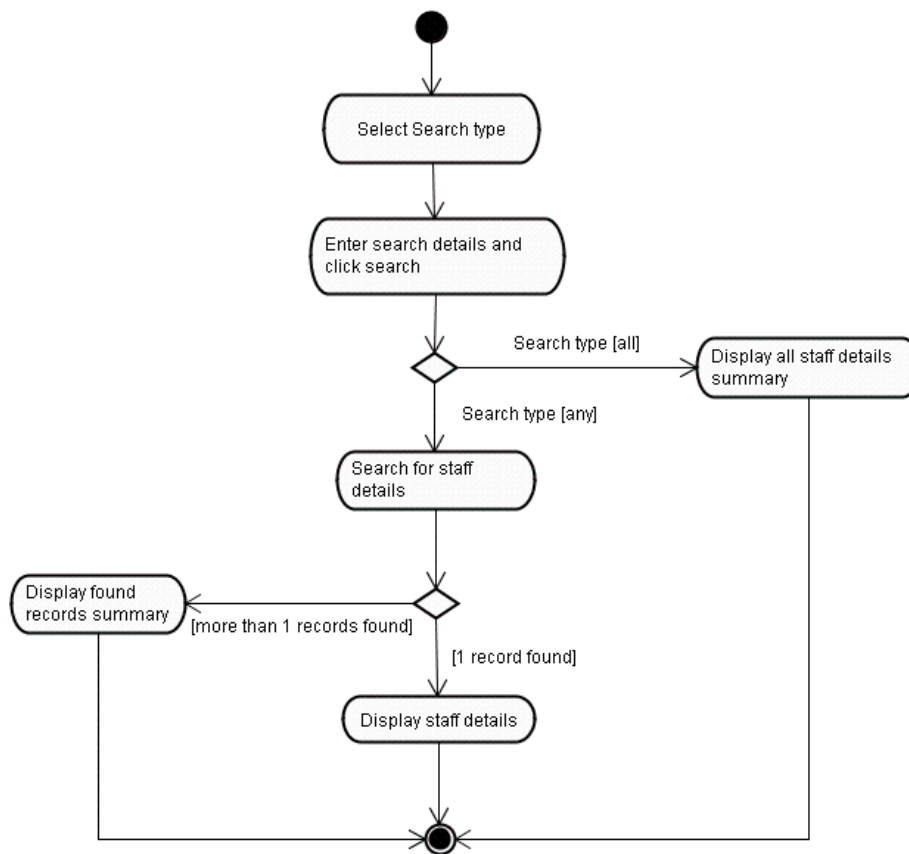
Alternative flow of events – No details found for search

- The staff member selects the find employee option
- The staff member selects the search type.
- The staff member enters the search criteria details and click the search button
- A message is displayed indicating that no details were found.

Exception flow of events – The system is unable to perform the staff details search due to an error

- The staff member selects the find employee option
- The staff member selects the search type.
- The staff member enters the search criteria details and click the search button
- A message is displayed indicating that an error has occurred.

Activity Diagram



Prototype

The screenshot shows a web application titled 'Search Employee'. It features a search bar with a dropdown menu set to 'Name' and a 'Search' button. Below the search bar is a table displaying a list of employees. The table has columns for Id, Name, Date Joined, Phone Number, and Address. The row for David Adams (Id 4) is highlighted in yellow. At the bottom of the application, there is a 'View Details' button.

Id	Name	Date Joined	Phone Number	Address
3	Carla Marcus	23 Apr, 2002	7745453454	70 High Groove, Whitehall, London, W12 3ED
8	Helga Becks	23 Apr, 2002	7745453454	10 Alexander House, Farm Lane, Mortimer, RG10 4RT
13	Carla Marcus	23 Apr, 2002	7745453454	70 High Groove, Whitehall, London, W12 3ED
18	Helga Becks	23 Apr, 2002	7745453454	10 Alexander House, Farm Lane, Mortimer, RG10 4RT
4	David Adams	01 Nov, 2003	2089454545	Alexander House, Farm Lane, Mortimer, RG10 4RT
9	Ianna Tim	01 Nov, 2003	7745453454	7 High Groove, Whitehall, London, W12 3ED
14	David Adams	01 Nov, 2003	2089454545	Alexander House, Farm Lane, Mortimer, RG10 4RT
19	Ianna Tim	01 Nov, 2003	7745453454	7 High Groove, Whitehall, London, W12 3ED
1	Adam Smith	01 Mar, 2004	7786368052	74 Westlands Ave, Reading, RG2 8EW
6	Fred Thomas	01 Mar, 2004	7786368052	70 High Groove, Whitehall, London, W12 3ED
11	Adam Smith	01 Mar, 2004	7786368052	74 Westlands Ave, Reading, RG2 8EW
16	Fred Thomas	01 Mar, 2004	7786368052	70 High Groove, Whitehall, London, W12 3ED
2	Betty Hamilton	15 Jun, 2005	1189582845	7 Horspath Road, Cowley, Oxford, OX4 2QL
7	Greg Gregory	15 Jun, 2005	2089454545	80 Westlands Ave, Reading, RG2 8EW

11.3.3 View Personal Details / Edit Personal Details

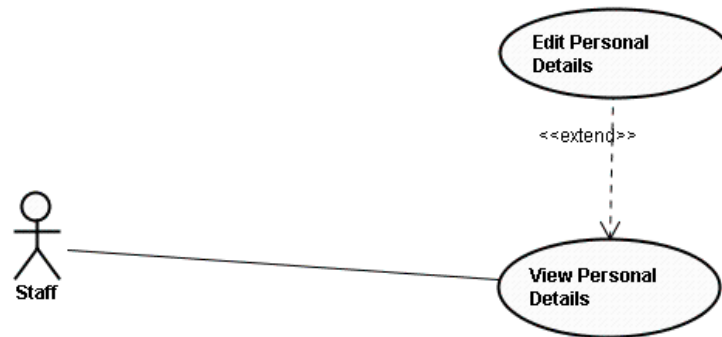


Figure 75. View/Edit personal details use case diagram

Name:

View personal details

Description:

The current member of staff views their personal details. After view their details members of staff can also edit these details. Regular staff can edit only a subset of their details. On the other hand Admin users can edit all their personal details.

Assumptions

- None

Preconditions

User is logged in to the system.

Post-conditions

- If the staff member has edited their personal details, then the new changes are persisted and can be viewed using view my details use case.

Normal flow of events – The staff member successfully views their details

- The staff member selects the my details option
- The details for the staff member are displayed.

Alternative flow of events – The staff member successfully edit their details

- The staff member selects the my details option
- The details for the staff member are displayed.
- The staff member selects to edit his/her personal details.
- The staff member edit their details, not all details are editable for regular staff members

- The staff member save the changes they made to their personal details
- The system generates a task for the Administrator to review the changes

Alternative flow of events – An Administrator successfully edit their details

- The administrator selects the my details option
- The details for the administrator are displayed.
- The administrator selects to edit his/her personal details.
- The administrator edit their details, all details are editable.
- The administrator save the changes they made to their personal details

Alternative flow of events – The staff member edits their details, but cancels the operation

- The staff member selects the my details option
- The details for the staff member are displayed.
- The staff member selects to edit his/her personal details.
- The staff member edit their details
- The staff member cancels their changes. No changes are made to underlying data

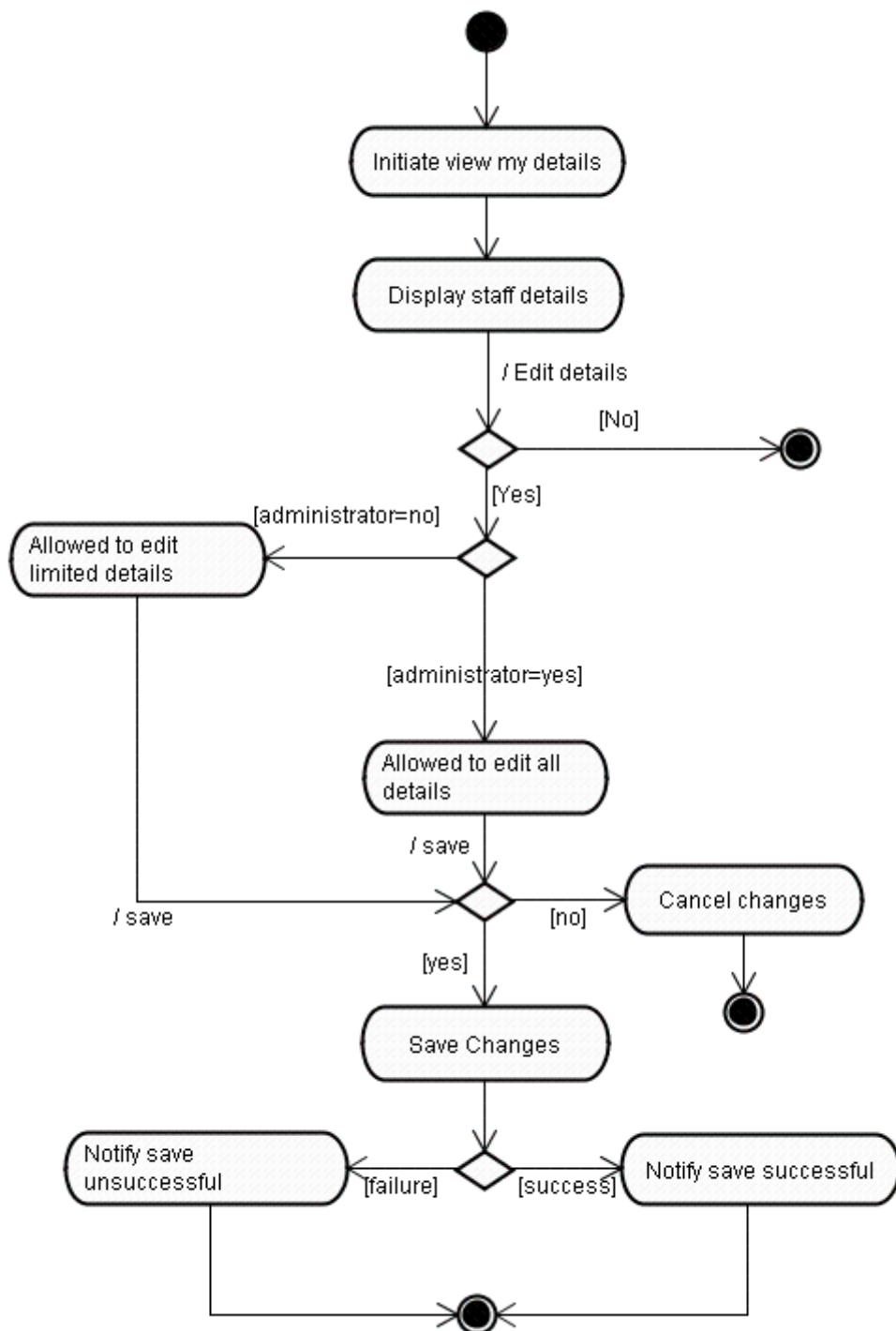
Exception flow of events – The system is unable to display the details of the current staff member due to an error

- The staff member selects the my details option
- A message is displayed indicating that an error has occurred.

Outstanding issues

Clarify which fields will be editable for staff members, some candidates are email, contact numbers and address.

Activity Diagram



11.3.4 View brief / complete staff details

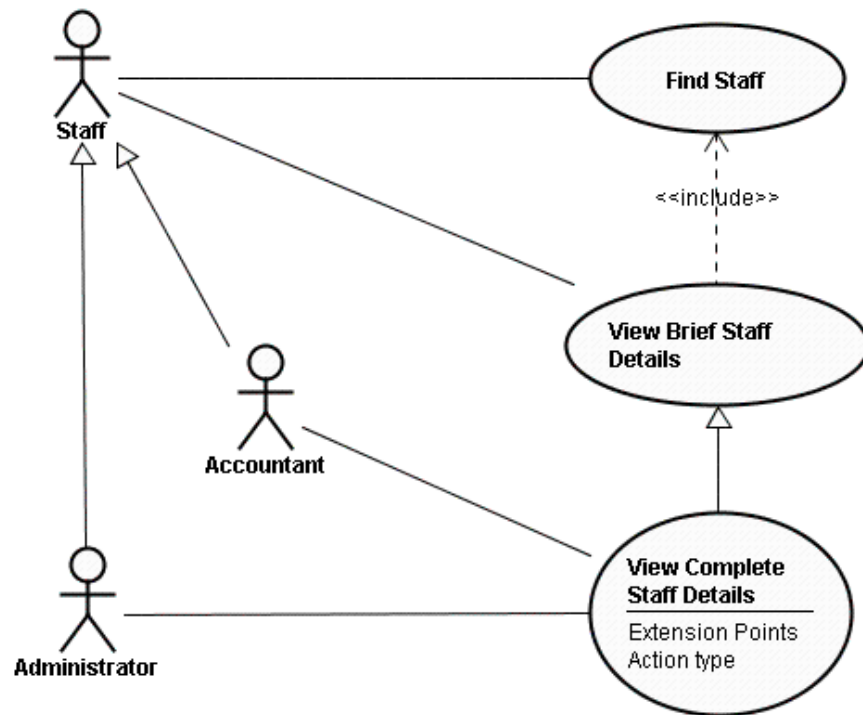


Figure 76. View brief/complete staff details use case diagram

Name:

View staff details

Description:

The current member of staff views the details of another staff member. Regular staff member can view a brief summary of other staff details. Admin and accountant users can view all the details for any member of staff.

Assumptions

- None

Preconditions

- User is logged in to the system.
- The current user has located the details for another staff member using the find staff use case

Post-conditions

- None

Normal flow of events – Regular staff member views the details of another staff member

- The staff member selects to view the staff details of another staff

- The brief details for the other staff member are displayed.

Alternative flow of events – Administrator or accountant views the details of another staff member

- The administrator or accountant selects to view the staff details of another staff
- The full details for the other staff member are displayed

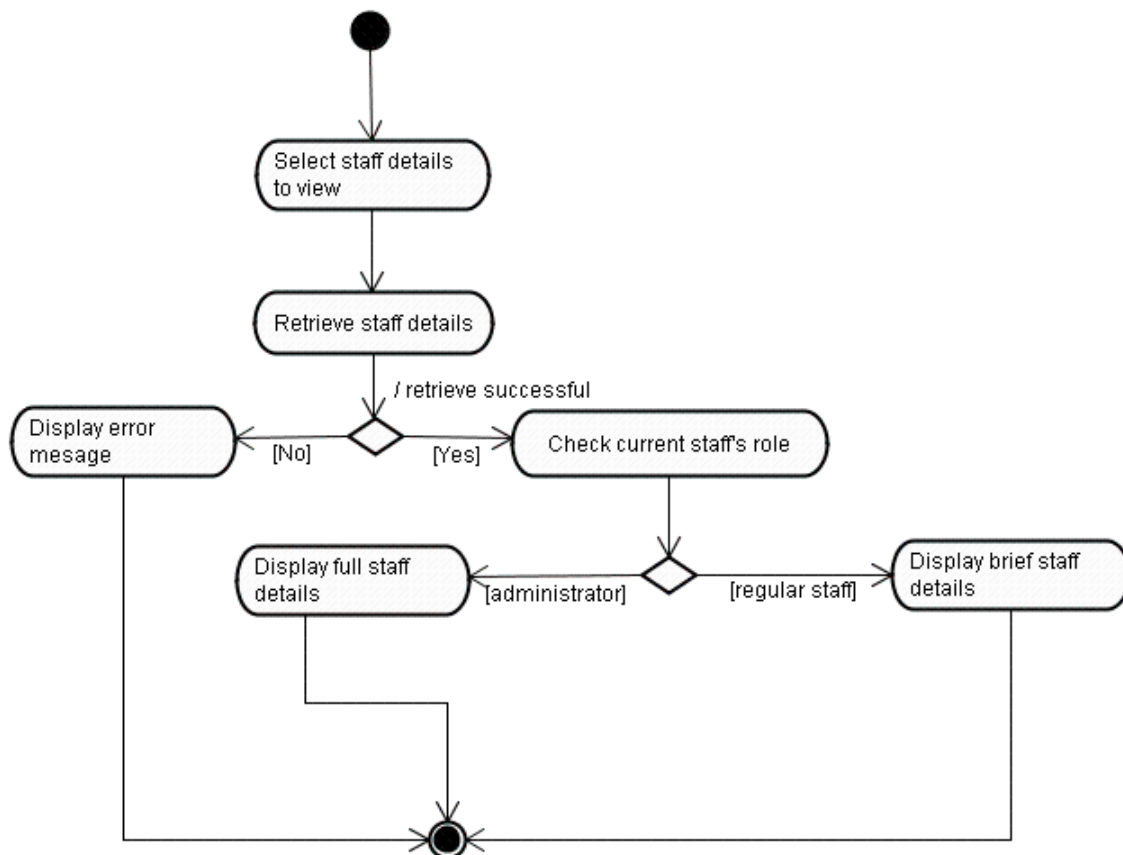
Exception flow of events – The system is unable to display the details of the staff member due to an error

- The staff member selects to view the staff details of another staff
- A message is displayed indicating that an error has occurred.

Outstanding issues

Clarify which staff member fields will be viewable by other regular staff members.

Activity Diagram



11.3.5 Edit staff details

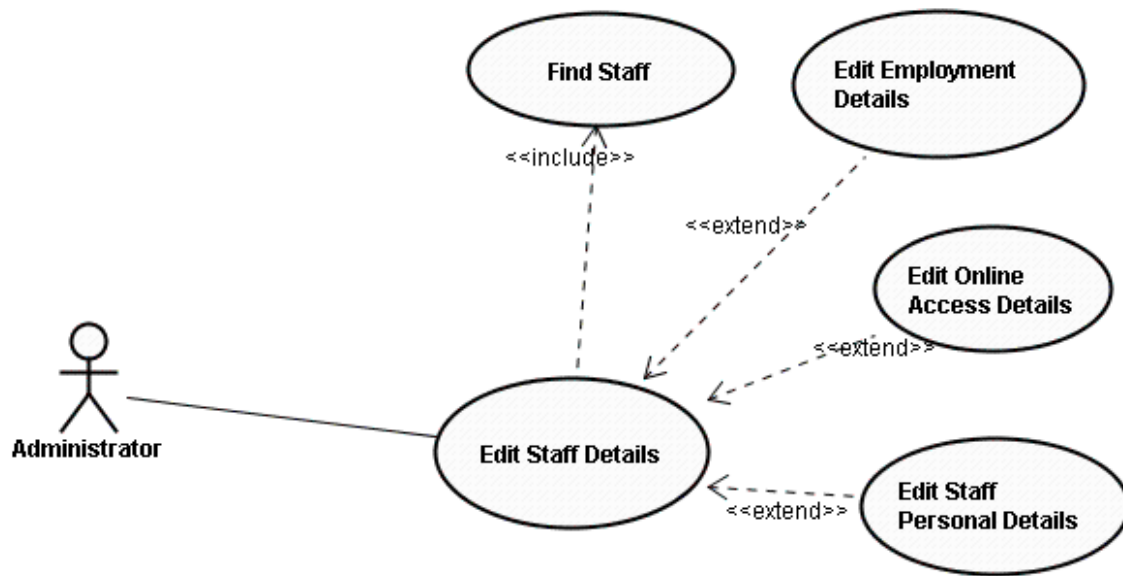


Figure 77. Edit staff use case diagram

Name:

Edit staff details

Description:

Admin users can edit other staff details.

Assumptions

- The member of staff to be edited is already registered

Preconditions

- User is logged in to the system.
- The current user has located the details for another staff member using the find staff use case
- The current user view the staff member details using the view details use case

Post-conditions

- If the edited staff details are saved then these details are persisted and can be viewed using the find staff use case

Normal flow of events – Administrator successfully edits the details of another staff member

- The administrator selects to edit the details of another staff member
- The full details of the other staff member are displayed.

- The administrator edits the personal, bank, address, employment and online details for the staff member.
- The administrator set the leaving date if applicable and the staff member has left.
- The administrator saves the details
- A message is displayed indicating the save was successful.
- The system creates a task to inform the staff member of the changes

Alternative flow of events – The administrator populate some invalid details

- The administrator selects to edit the details of another staff member
- The full details of the other staff member are displayed.
- The administrator edits the personal, bank, address, employment and online details for the staff member.
- The administrator set the leaving date if applicable and the staff member has left.
- The administrator tries to saves the details
- A message is displayed indicating that some of the details provided are invalid; no change is made to existing data.

Alternative flow of events – Administrator edits the details of another staff member, but cancels his/her action

- The administrator selects to edit the details of another staff member
- The full details of the other staff member are displayed.
- The administrator edits the personal, bank, address, employment and online details for the staff member.
- The administrator set the leaving date if applicable and the staff member has left.
- The administrator cancels the changes. No change is made to underlying data.

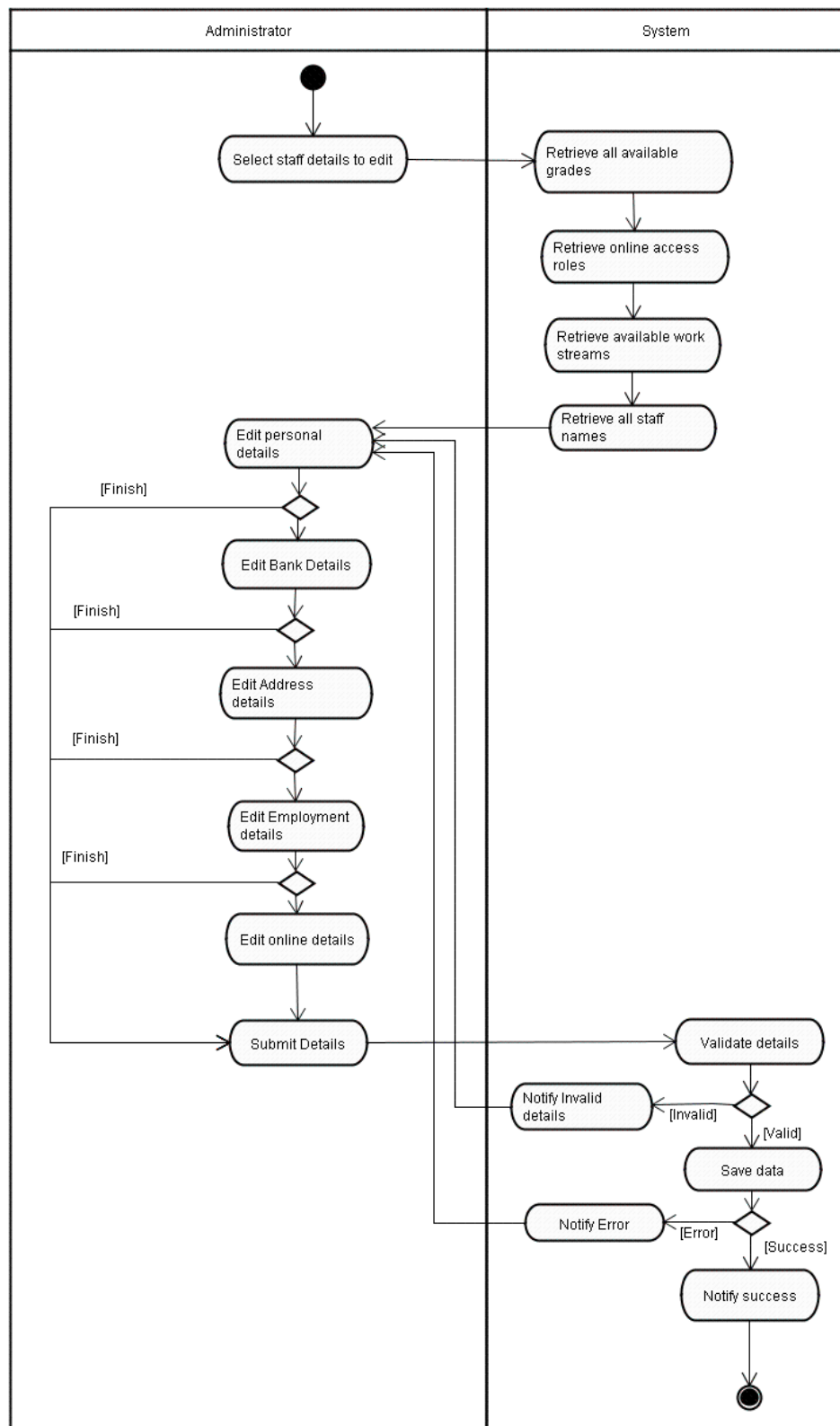
Exception flow of events – The system is unable to save the details of the staff member due to an error

- The administrator selects to edit the details of another staff member
- The full details of the other staff member are displayed.
- The administrator edits the personal, bank, address, employment and online details for the staff member.
- The administrator set the leaving date if applicable and the staff member has left.
- The administrator saves the details
- A message is displayed indicating there was an error saving the details.

Outstanding issues

Are administrators assigned a specific set of staff they can manage, by a super user or they can manage any staff details.

Activity Diagram



11.3.6 Find Expenses

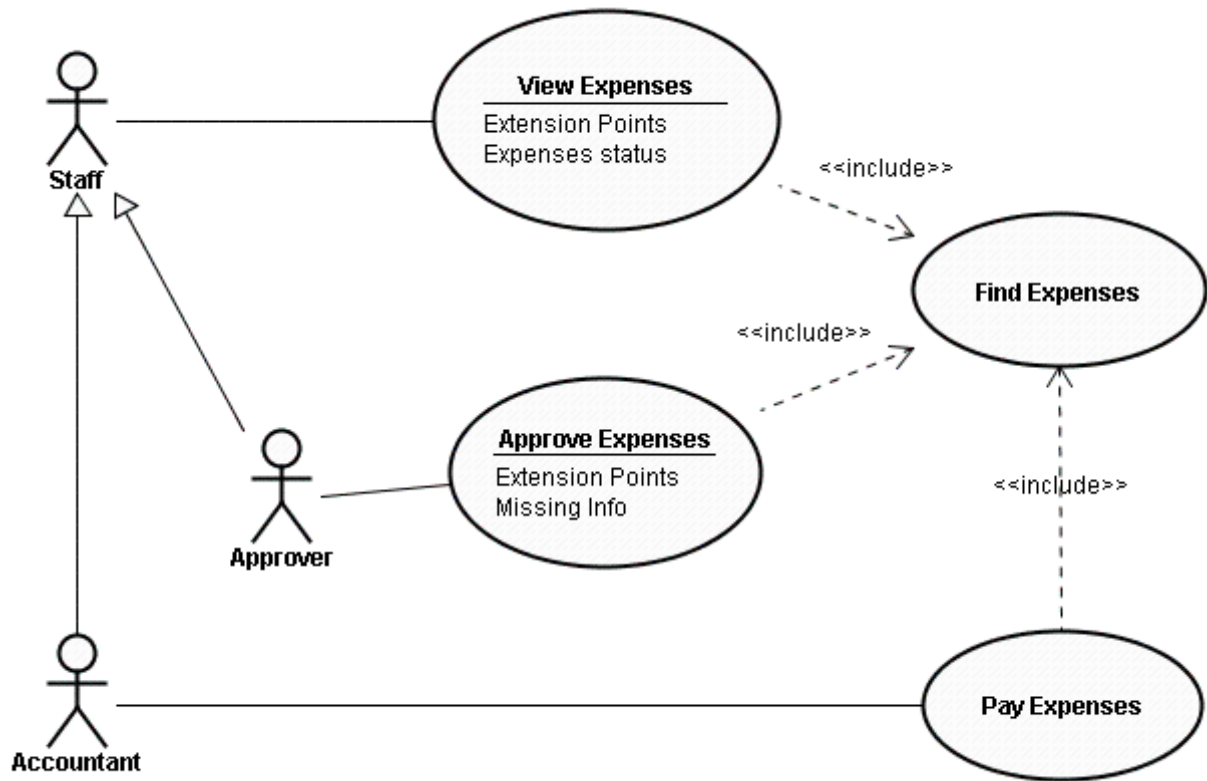


Figure 78. Find expense use case diagram

Name:

Find Expenses

Description:

Staff members can find their expenses using an expenses browser, where they can search by year or expenses status. Approvers can also search expenses they are approving by staff name, year or expenses status. This use case is part of the View Expenses, Approve Expenses and Pay Expenses use cases

Assumptions

- Expenses have the following statuses: New, Saved, Pending, Rejected, Approved and Paid

Preconditions

- User is logged in to the system.

Post-conditions

- None

Normal flow of events – Regular staff member searches for an expense

- The staff member selects the find expense option
- The staff member sets the view criteria, either view all expenses, view by year or view by status.
- The member of staff check the displayed list and select the expense they are looking for

Alternative flow of events – Approver or accountant searches for expenses of another staff member

- The Approver or accountant selects the find expense option
- The Approver selects the staff member name from a list of staff they are allowed to approve expenses for
- The Accountant selects the staff member name from all staff list.
- The staff member sets the view criteria, either view all expenses, view by year or view by status. Available statuses are: Pending, Approved and Paid.
- The Approver or accountant check the displayed list and select the expense they are looking for

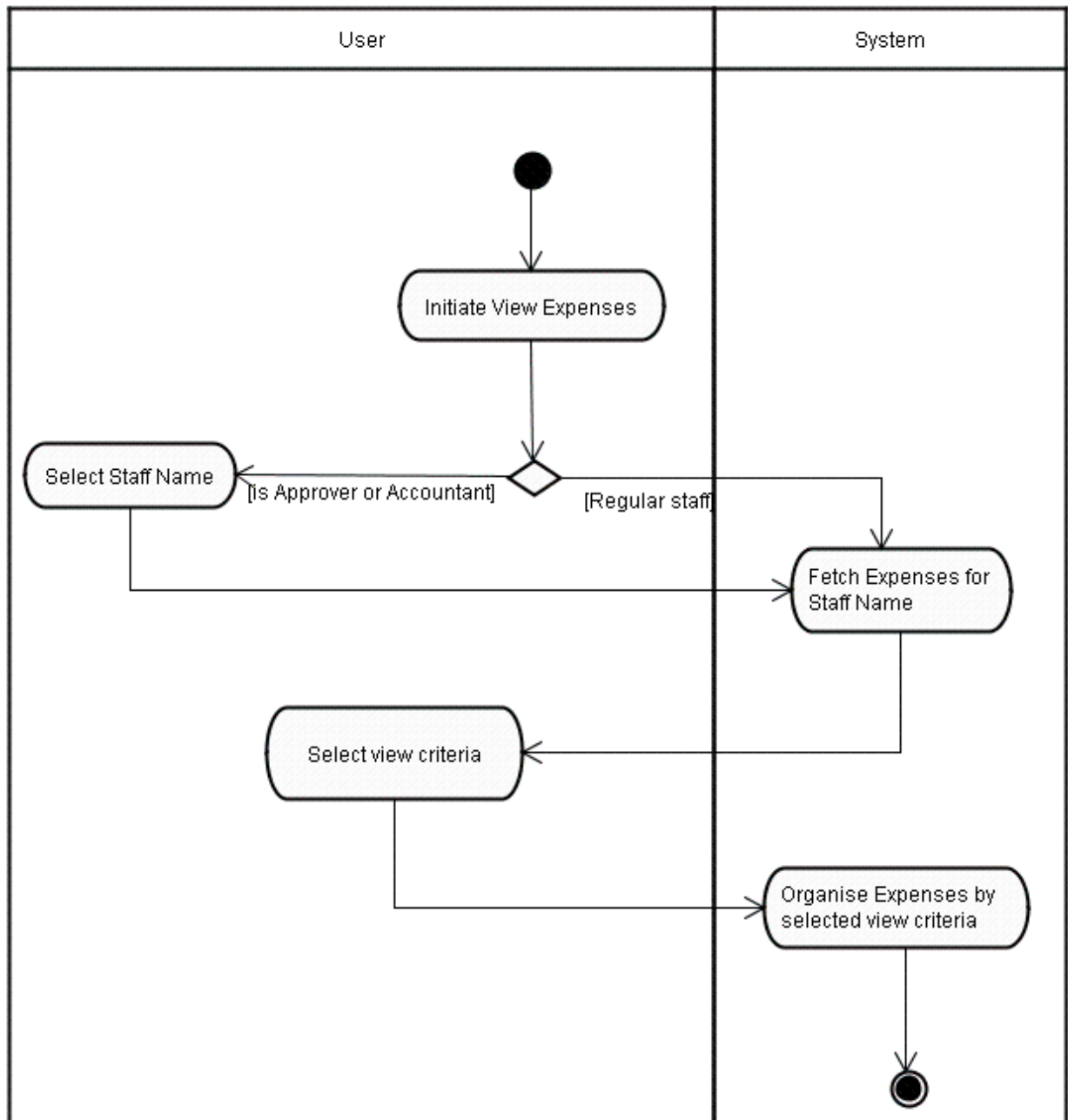
Exception flow of events – The system is unable to find staff expense due to an error

- The staff member selects to view expenses
- A message is displayed indicating that an error has occurred.

Outstanding issues

None

Activity Diagram



Prototype

View Expenses

Expenses Summary

View: Expenses List
Total Approved Expenses: **£1976.34**
Total Approved Miles: **16,456**

Expenses

All

2007 - Oct

2007 - Sep

2007 - Aug

2006 - Dec

2006 - Jul

2006 - Feb

2005 - Dec

Expenses Summary for Period

Status:

Approved

Total mileage for period:

150 miles

Total mileage to date:

12,150 miles

Total claimed exc. VAT:

£1550.35

Total claimed inc. VAT:

£1821.66

Expenses Items Summary

Date Entered	Description	Mileage	Receipt	Amount
10 Oct, 2007	Monthly Broadband	0	N	£25
11 Oct, 2007	Mileage from Home to Newbury	30	N	£8
12 Oct, 2007	Office Equipment	0	Y	£130
16 Oct, 2007	Mileage from Home to Newbury	30	N	£8
17 Oct, 2007	Mileage from Home to Newbury	30	N	£8

View Expenses

Expenses Summary

View: Expenses by Status
Total Approved Expenses: **£1976.34**
Total Approved Miles: **16,456**

Expenses

Status

Approved

2006 - Jul

2006 - Feb

2005 - Dec

Pending

2007 - Aug

2006 - Dec

Rejected

2007 - Oct

Saved

2007 - Sep

Expenses Summary for Period

Status:

Approved

Total mileage for period:

150 miles

Total mileage to date:

12,150 miles

Total claimed exc. VAT:

£1550.35

Total claimed inc. VAT:

£1821.66

Expenses Items Summary

Date Entered	Description	Mileage	Receipt	Amount
10 Oct, 2007	Monthly Broadband	0	N	£25
11 Oct, 2007	Mileage from Home to Newbury	30	N	£8
12 Oct, 2007	Office Equipment	0	Y	£130
16 Oct, 2007	Mileage from Home to Newbury	30	N	£8
17 Oct, 2007	Mileage from Home to Newbury	30	N	£8

View Expenses

Expenses Summary

View: Expenses by Year | Total Approved Expenses: **£1976.34** | Total Approved Miles: **16,456**

Expenses

Year

2007

Oct

Sep

Aug

2006

Dec

Jul

Feb

2005

Dec

Expenses Summary for Period

Status:

Approved

Total mileage for period:

150 miles

Total mileage to date:

12,150 miles

Total claimed exc. VAT:

£1550.35

Total claimed inc. VAT:

£1821.66

Expenses Items Summary

Date Entered	Description	Mileage	Receipt	Amount
10 Oct, 2007	Monthly Broadband	0	N	£25
11 Oct, 2007	Mileage from Home to Newbury	30	N	£8
12 Oct, 2007	Office Equipment	0	Y	£130
16 Oct, 2007	Mileage from Home to Newbury	30	N	£8
17 Oct, 2007	Mileage from Home to Newbury	30	N	£8

11.3.7 View, approve, reject and pay Expenses

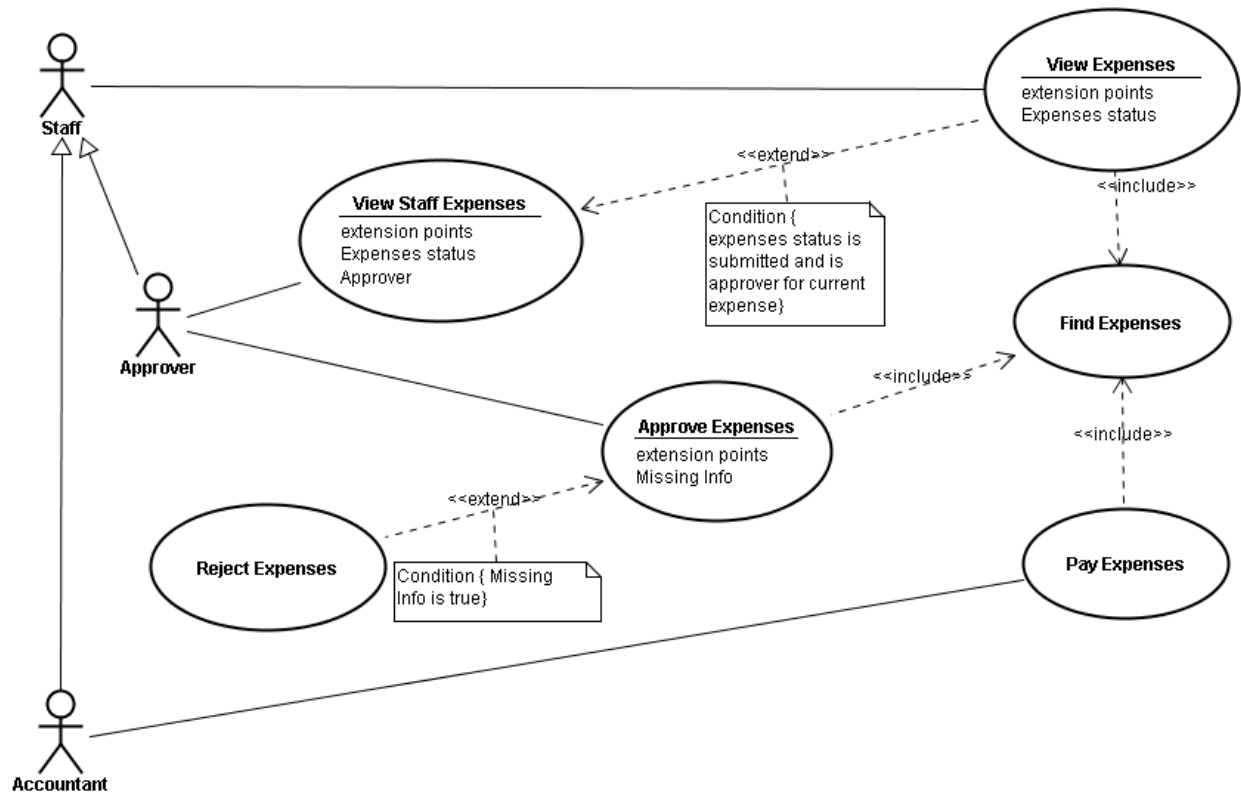


Figure 79. View, approve, reject and pay expenses use Case diagram

Name:

View Expenses

Description:

After finding an expense the staff member can view its details.

Assumptions

- Expenses have the following statuses: New, Saved, Pending, Rejected, Approved and Paid
- Approvers can see Pending, Approved and Paid expenses
- Accountant can see Approved and Paid expenses.

Preconditions

- User is logged in to the system.
- User navigate to the target expense using the find expenses use case

Post-conditions

- None

Normal flow of events – Staff member view an expense

- The staff member click on an expense
- The system displays the expense details
- The staff member views the displayed expense details.

Alternative flow of events – Staff member edits a saved or rejected expense

- The staff member click on an expense
- The system displays the expense details
- The staff member views the displayed expense details.
- The staff member click on the edit button
- The staff member is forwarded to the edit expenses use case

Alternative flow of events – Approver approves or rejects expense

- The approver click on an expense
- The system displays the expense details
- The approver views the displayed expense details.
- The approver click on the approve button or select some expense items enter some text and reject the expenses
- The expense is saved in either approved or rejected status
- If rejected the system generate a task for staff member

Alternative flow of events – accountant pays an expense

- The accountant click on an expense
- The system displays the expense details
- The accountant views the displayed expense details.
- The accountant enters the paid date and click the pay button
- The expense is saved in paid status

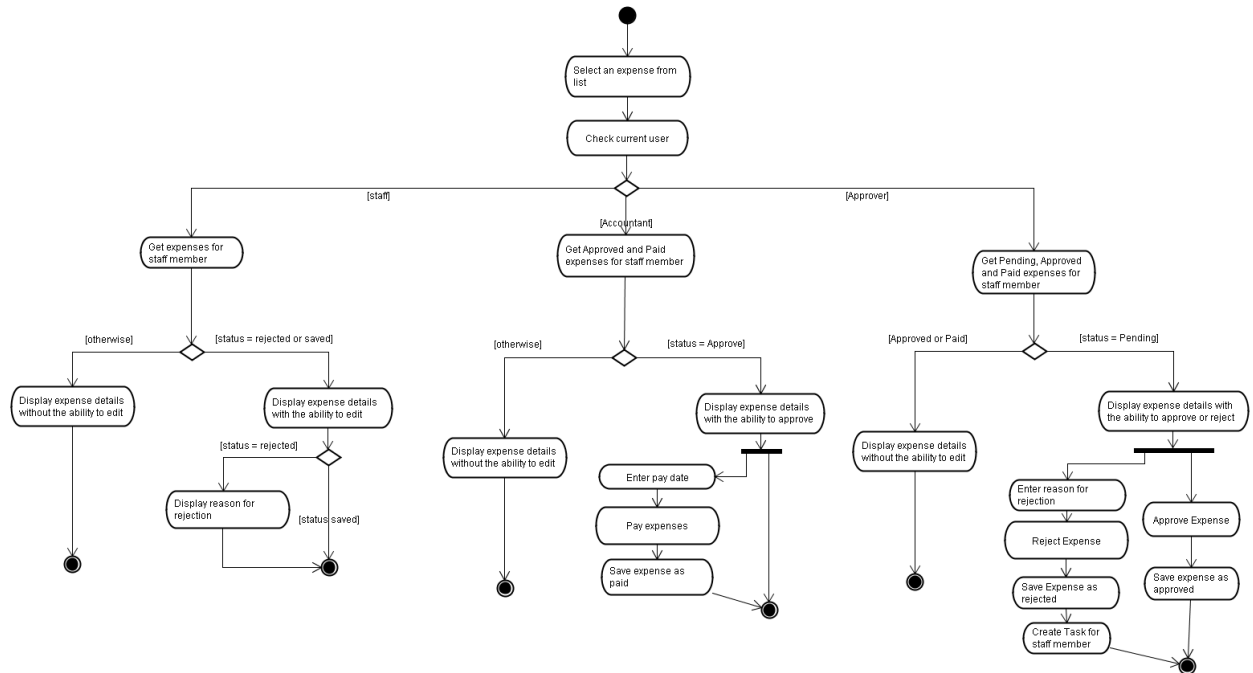
Exception flow of events – The system is unable to find expense due to an error

- The staff member click on an expense
- A message is displayed indicating that an error has occurred.

Outstanding issues

Decide on which expense details to display

Activity Diagram



Prototype

View Expenses

Expenses Summary

View: Expenses List
Total Approved Expenses: **£1976.34**
Total Approved Miles: **16,456**

Expenses

All

2007 - Oct
2007 - Sep
2007 - Aug
2006 - Dec
2006 - Jul
2006 - Feb
2005 - Dec

Expenses Summary for Period

Status: Saved

Edit

Total mileage for period: 150 miles
Total mileage to date: 12,150 miles
Total claimed exc. VAT: £1550.35
Total claimed inc. VAT: £1821.66

Expenses Items Summary

Date Entered	Description	Mileage	Receipt	Amount
10 Oct, 2007	Monthly Broadband	0	N	£25
11 Oct, 2007	Mileage from Home to Newbury	30	N	£8
12 Oct, 2007	Office Equipment	0	Y	£130
16 Oct, 2007	Mileage from Home to Newbury	30	N	£8
17 Oct, 2007	Mileage from Home to Newbury	30	N	£8

Expenses Summary

View: Expenses List | Total Approved Expenses: **£1976.34** | Total Approved Miles: **16,456**

Expenses Summary for Period

Status: **Rejected** [Edit](#)

Total mileage for period: **150 miles**

Total mileage to date: **12,150 miles**

Total claimed exc. VAT: **£1550.35**

Total claimed inc. VAT: **£1821.66**

Expenses Items Summary

Date Entered	Description	Mileage	Receipt	Amount
10 Oct, 2007	Monthly Broadband	0	N	£25
11 Oct, 2007	Mileage from Home to Newbury	30	N	£8
12 Oct, 2007	Office Equipment	0	Y	£130
16 Oct, 2007	Mileage from Home to Newbury	30	N	£8
17 Oct, 2007	Mileage from Home to Newbury	30	N	£8

Reason for rejection: **Please specify which office equipment was purchased**

11.3.8 Edit Expenses Use Case

Name:

Edit Expenses

Description:

After viewing an editable expense or trying to add a new expense for a period where a saved expense already exist the staff member is allowed to edit the expense and cancel, save or submit their changes

Assumptions

- Expenses have the following statuses: New, Saved, Pending, Rejected, Approved and Paid
- When adding expenses for a period and saved expenses exist for the same period then the system displays the saved expenses instead

Preconditions

- User is logged in to the system.
- User views the expense using the view expenses use case
- User selects a period with a saved expense using the add new expenses use case

Post-conditions

- The edited expense details are persisted if the user has submitted them, otherwise no change is made to underlying data

Normal flow of events – Staff member edits an expense and submits it.

- The staff member reviews and/or modifies the current entries.
- The staff member adds new entries.
- The staff member deletes an entry if need be.
- The staff member submit their expenses
- Expense details is saved and status changes to Pending
- A task is generated for the approver
- A confirmation is displayed confirming the submit
- The edit window is closed.

Alternative flow of events – Staff member edits an expense and saves it.

- The staff member reviews and/or modifies the current entries.
- The staff member adds new entries.
- The staff member deletes an entry if need be.
- The staff member save their expenses
- Expense details is saved and status stays as saved the last saved date is updated
- A confirmation is displayed confirming the successful save

Alternative flow of events – Staff member edits an expense, but cancels their changes.

- The staff member reviews and/or modifies the current entries.
- The staff member adds new entries.
- The staff member deletes an entry if need be.
- The staff member cancel their changes
- A confirmation is displayed asking the staff member if they want to abandon their changes

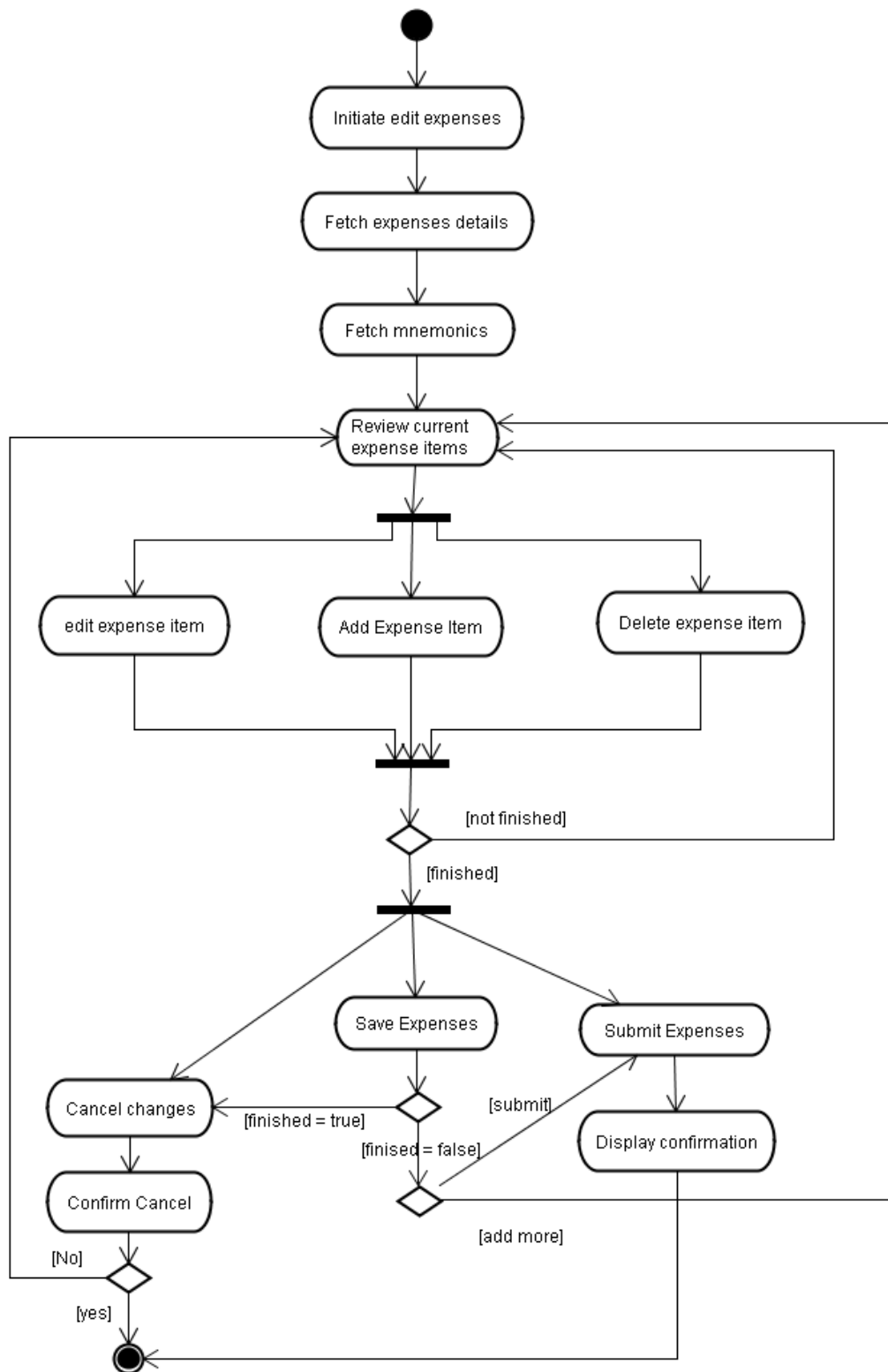
Exception flow of events – The system is unable to save or submit expense due to an error

- The staff member reviews and/or modifies the current entries.
- The staff member adds new entries.
- The staff member deletes an entry if need be.
- The staff member save or submit their expenses
- An error message is displayed indicating an error

Outstanding issues

Decide on which expense details to display

Activity Diagram



Prototype

Add/Edit Expenses

Expenses Period: Status: **Saved** Last Saved: **Mon, 21/10/2007 13:50**

Actions	Date	Mnemonic	Type	Description	Amount	Miles
<input type="checkbox"/>	Mon, 10/09/1978	broadband	Phone & Internet	Monthly broadband charges	£25	
<input type="checkbox"/>	Tue, 10/09/1999	home	Mileage	Mileage from home to Newbury	£15	50
<input type="checkbox"/>	Wed, 10/09/1978	broadband	Phone & Internet	Monthly broadband charges	£25	
<input type="checkbox"/>	Thurs, 10/09/1999	home	Mileage	Mileage from home to Newbury	£15	50
<input type="checkbox"/>	<input type="text"/>	<input type="text"/>	<input type="text" value="Please Choose..."/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
Total					£80	100

With selected:

11.3.9 Add New Expenses Use Case

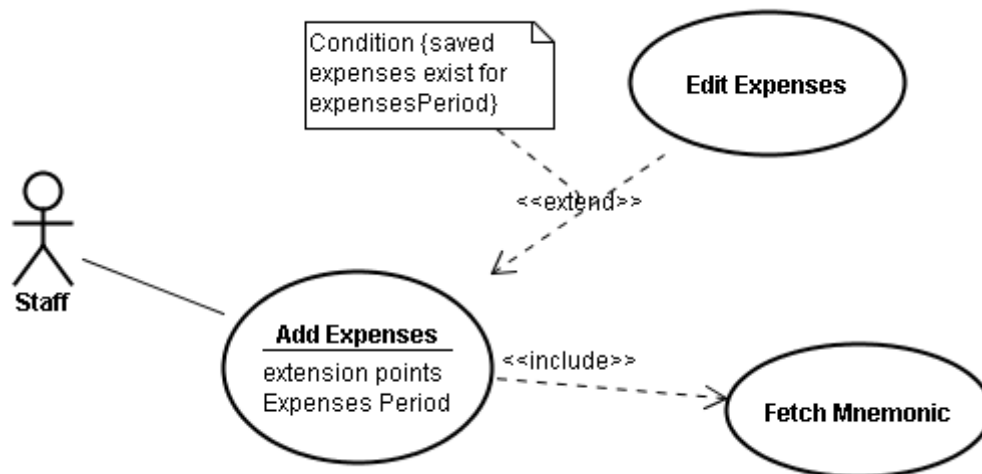


Figure 80. Add new expenses use case diagram

Name:
Add Expenses

Description:

A staff member can add new expenses after choosing the period of the expense if a saved expense exist for the chosen period then the staff edits the expense using the edit expenses use case.

Assumptions

- Expenses have the following statuses: New, Saved, Pending, Rejected, Approved and Paid
- When adding expenses for a period and saved expenses exist for the same period then the system displays the saved expenses instead

Preconditions

- User is logged in to the system.

Post-conditions

- The added expense details are persisted if the user has submitted them, otherwise no change is made to underlying data

Normal flow of events – Staff member adds an expense and submits it.

- The staff member adds new entries.
- The staff member deletes an entry if need be.
- The staff member edits a newly added entry
- The staff member submit their expenses
- Expense details is saved and status changes to Pending
- The system generates a task for the approver
- A confirmation is displayed confirming the submit
- The edit window is closed.

Alternative flow of events – Staff member add an expense for a period for which a saved one already exists.

- The staff member edits the saved expense as outline in the edit expense use case.

Alternative flow of events – Staff member add an expense and saves it.

- The staff member adds new entries.
- The staff member deletes an entry if need be.
- The staff member edits a newly added entry
- The staff member save their expenses
- Expense details is saved and status stays as saved the last saved date is updated
- A confirmation is displayed confirming the successful save

Alternative flow of events – Staff member add an expense, but cancels their changes.

- The staff member adds new entries.

- The staff member edits a newly added entry
- The staff member deletes an entry if need be.
- The staff member cancel their changes
- A confirmation is displayed asking the staff member if they want to abandon their changes

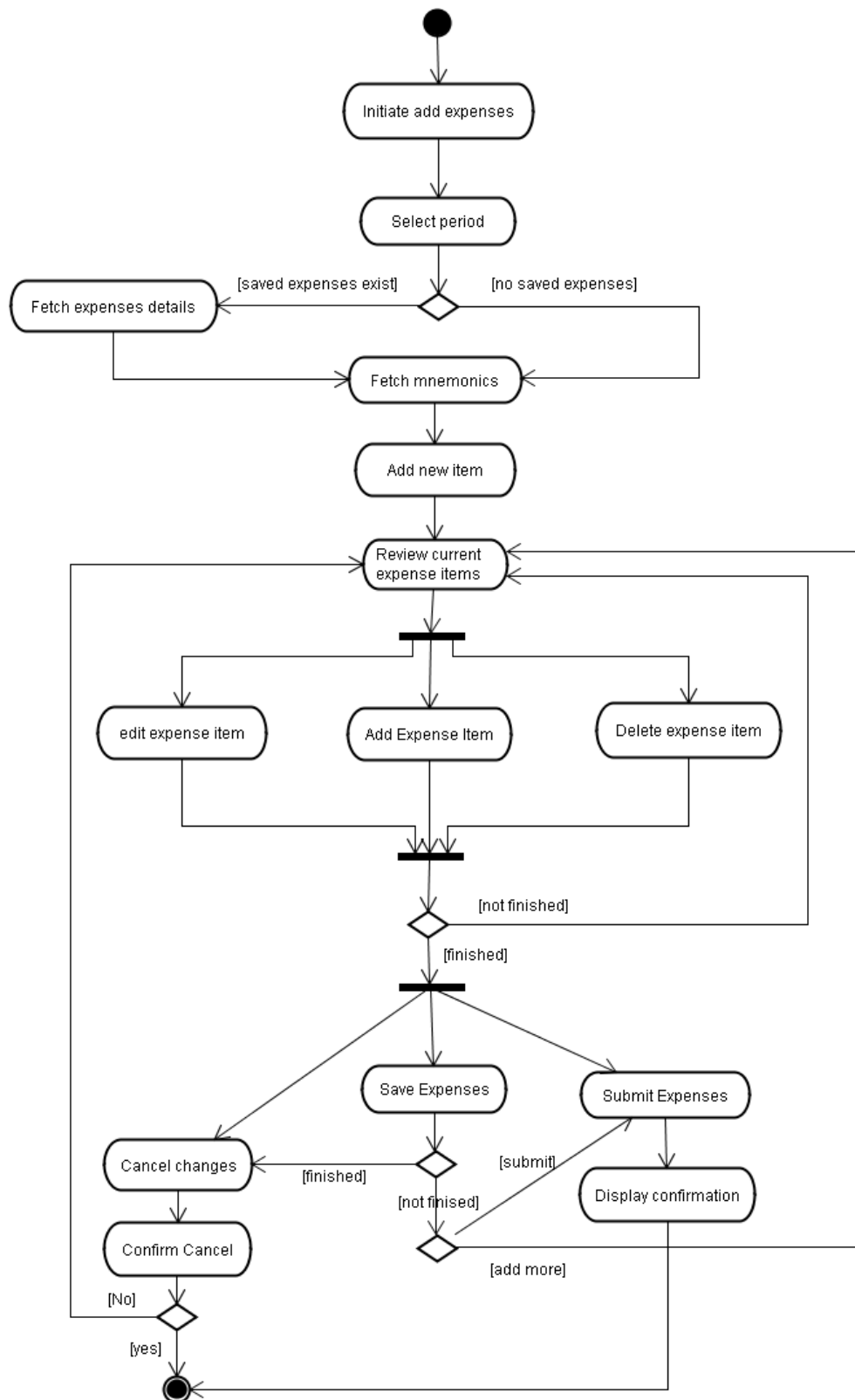
Exception flow of events – The system is unable to save or submit expense due to an error

- The staff member adds new entries.
- The staff member deletes an entry if need be.
- The staff member save or submit their expenses
- An error message is displayed indicating an error

Outstanding issues

Decide on which expense details to display

Activity Diagram



Prototype

Add/Edit Expenses

Expenses Period: Status: **New**

Actions	Date	Mnemonic	Type	Description	Amount	Miles
<input type="checkbox"/>	<input type="text"/>	<input type="text"/>	<input type="text" value="Please Choose..."/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
Total					£0	0

[Check All](#) / [Uncheck All](#) With selected:

11.3.10 Login use case

Name:

Login

Description:

The member of staff provides their username and password to login and the system performs authentication and authorisation on the details provided..

Assumptions

- None

Preconditions

- None.

Post-conditions

- The user is logged in to the system and has the correct role applied

Normal flow of events – Staff member successfully login to the system.

- The staff member enters their username and password.
- The staff member clicks the login button.
- The system authenticates the staff member against stored details.
- The system resets the invalid login details for the staff member
- The system displays the application desktop.

Alternative flow of events – Staff member username is not found or the account is locked.

- The staff member enters their username and password.
- The staff member clicks the login button.
- The system authenticates the staff member against stored details
- The system displays a message that the username and password provided are invalid or that the account is locked and the user should contact an administrator.
- The system creates a task to inform the administrator of the incident.

Alternative flow of events – Staff member password is not correct.

- The staff member enters their username and password.
- The staff member clicks the login button.
- The system authenticates the staff member against stored details
- The system increments the invalid login count
- The system displays a message that the username and password provided are invalid.
- If this the last attempt before the account is locked. The system locks the account

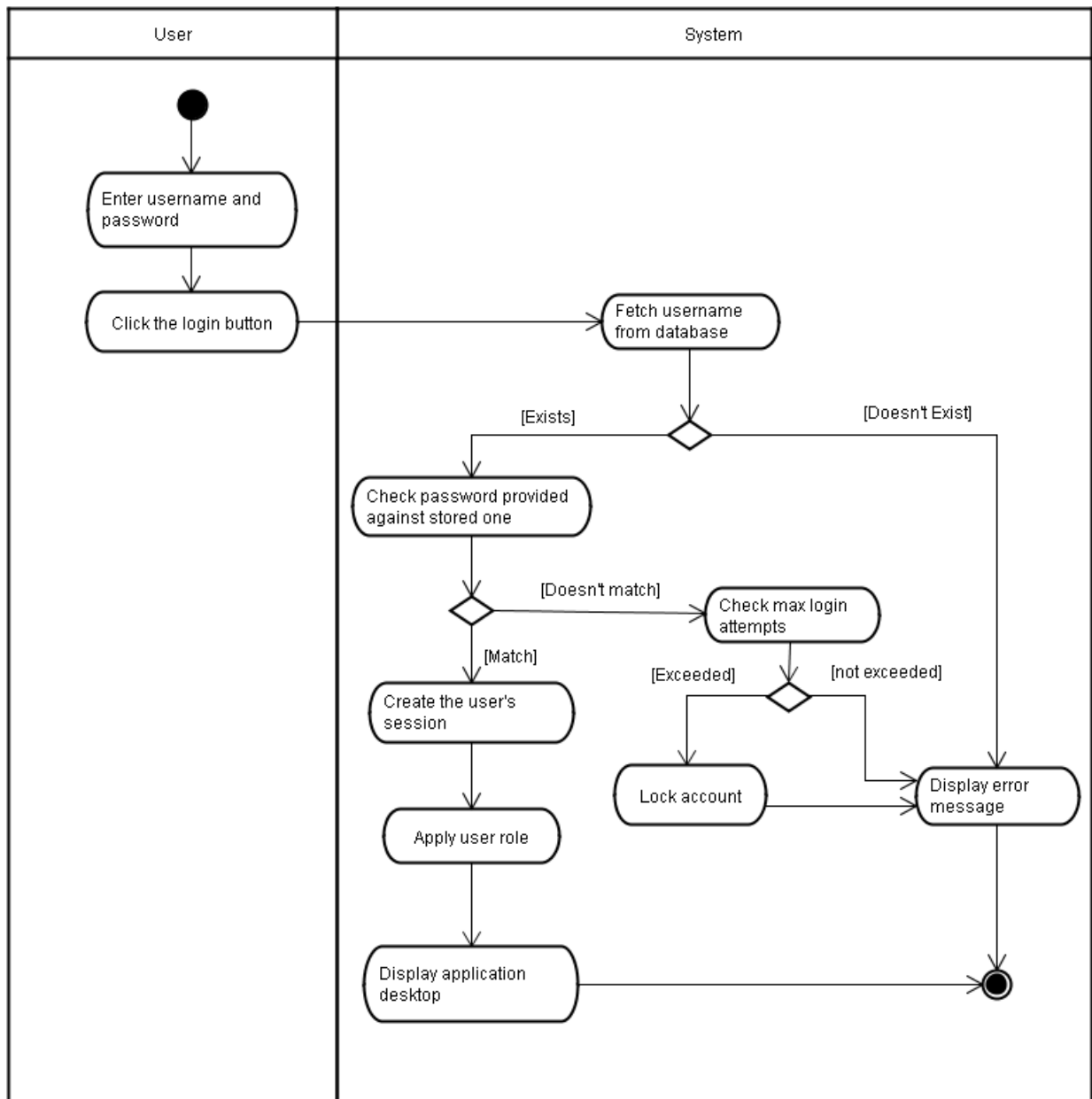
Exception flow of events – The system is authenticate the staff member due to an error

- The staff member enters their username and password.
- The staff member clicks the login button.
- The system authenticates the staff member against stored details
- The system displays a message indicating an error has occurred.

Outstanding issues

None

Activity Diagram



11.3.11 View/Update system settings

Name:

View/Update system settings

Description:

An administrator can view the system wide settings and update/save them.
After viewing the system settings the administrator updates the setting

Assumptions

- None

Preconditions

- None.

Post-conditions

- The updated system settings are successfully saved

Normal flow of events – Administrator successfully views system settings.

- The Administrator clicks on the system setting options.
- The application retrieves the settings currently stored.
- The application displays the system settings.
- The Administrator views the various settings
- The administrator closes the system settings window.

Alternative flow of events – Administrator successfully updates the system settings.

- The Administrator clicks on the system setting options.
- The application retrieves the settings currently stored.
- The application displays the system settings.
- The administrator modifies the system settings such as holiday rules, creates a new workstream, project, user role, etc... and click the save button
- The application displays a message that the settings were saved successfully

Alternative flow of events – Administrator changes the system settings but cancel the changes.

- The Administrator clicks on the system setting options.
- The application retrieves the settings currently stored.
- The application displays the system settings.
- The administrator modifies the system settings and click the cancel button
- The system settings window is closed.

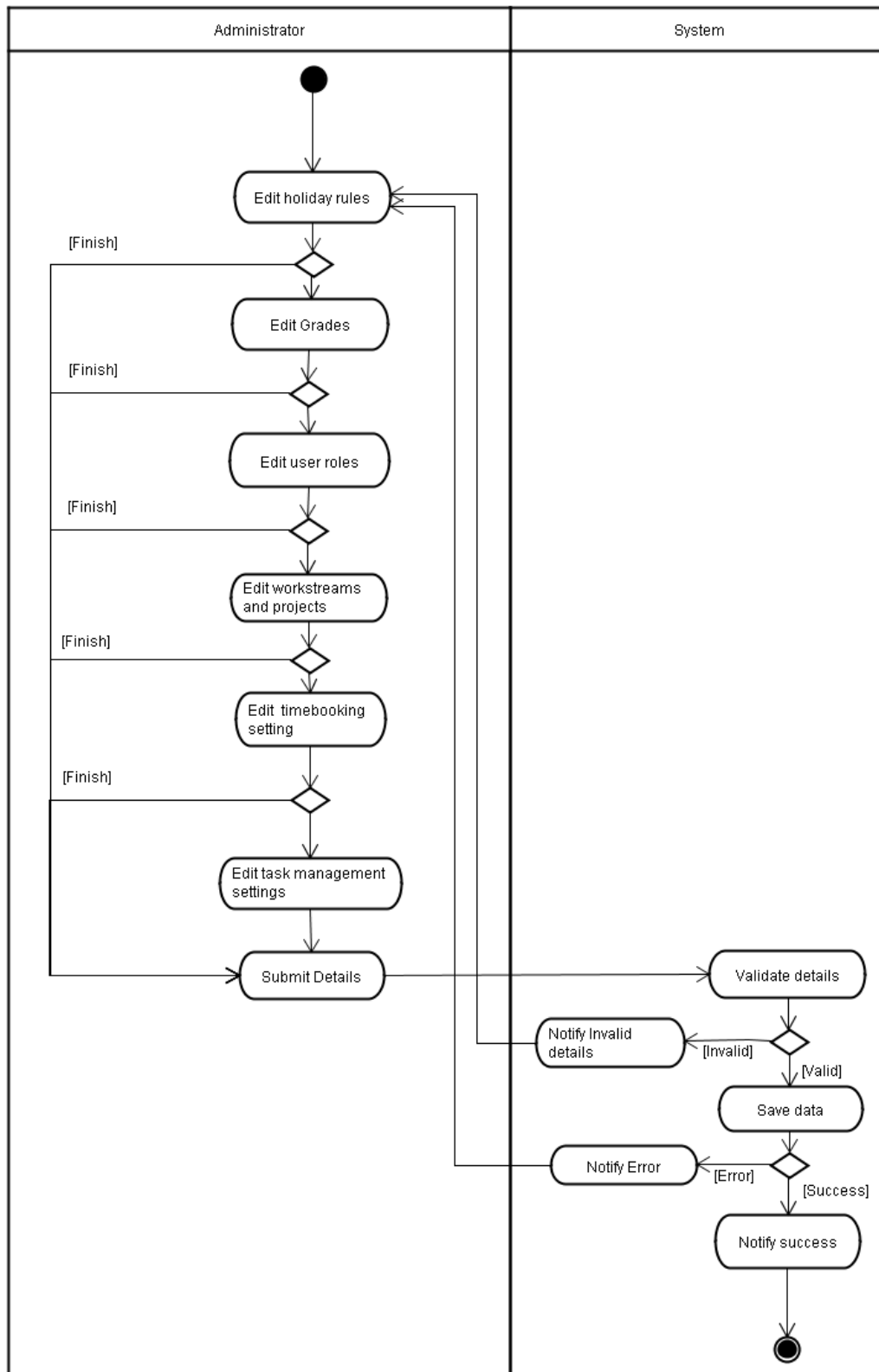
Exception flow of events – The system fails to update the system settings due to an error

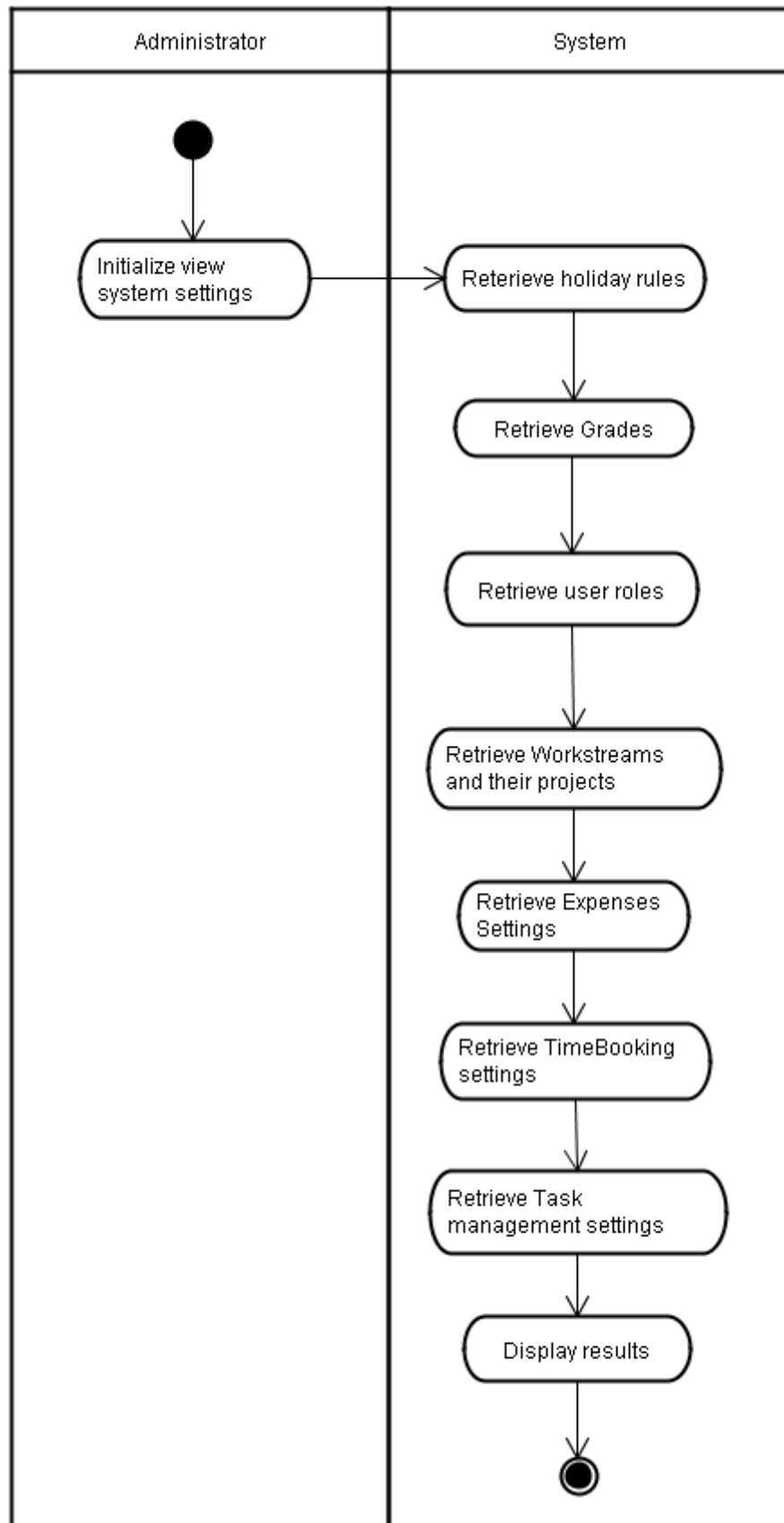
- The Administrator clicks on the system setting options.
- The application retrieves the settings currently stored.
- The application displays the system settings.
- The administrator modifies the system settings and click the save button
- The system displays a message indicating an error has occurred.

Outstanding issues

None

Activity Diagrams





11.3.12 Update my settings

Name:

Update my settings

Description:

A staff member can change their online password or update their online preferences such as desktop background colour, etc...

Assumptions

- None

Preconditions

- None.

Post-conditions

- The updated staff member settings are successfully saved

Normal flow of events – Staff member successfully update their settings.

- The staff member clicks on my setting options.
- The application retrieves the settings currently stored for the staff member.
- The application displays the staff member settings.
- The staff member changes their password by entering their current password and new password
- The staff member changes their preferences
- The staff member then saves the current changes.
- The system displays a success message

Alternative flow of events – Staff member views their settings.

- The staff member clicks on my setting options.
- The application retrieves the settings currently stored for the staff member.
- The application displays the staff member settings.
- The staff member view their settings
- The staff member clicks the cancel button

Alternative flow of events – Staff member changes the system settings but cancel the changes.

- The staff member clicks on my setting options.
- The application retrieves the settings currently stored for the staff member.
- The application displays the staff member settings.
- The staff member changes their password by entering their current password and new password
- The staff member changes their preferences
- The staff member modifies their settings and click the cancel button

- The system settings window is closed, and underlying data is not changed.

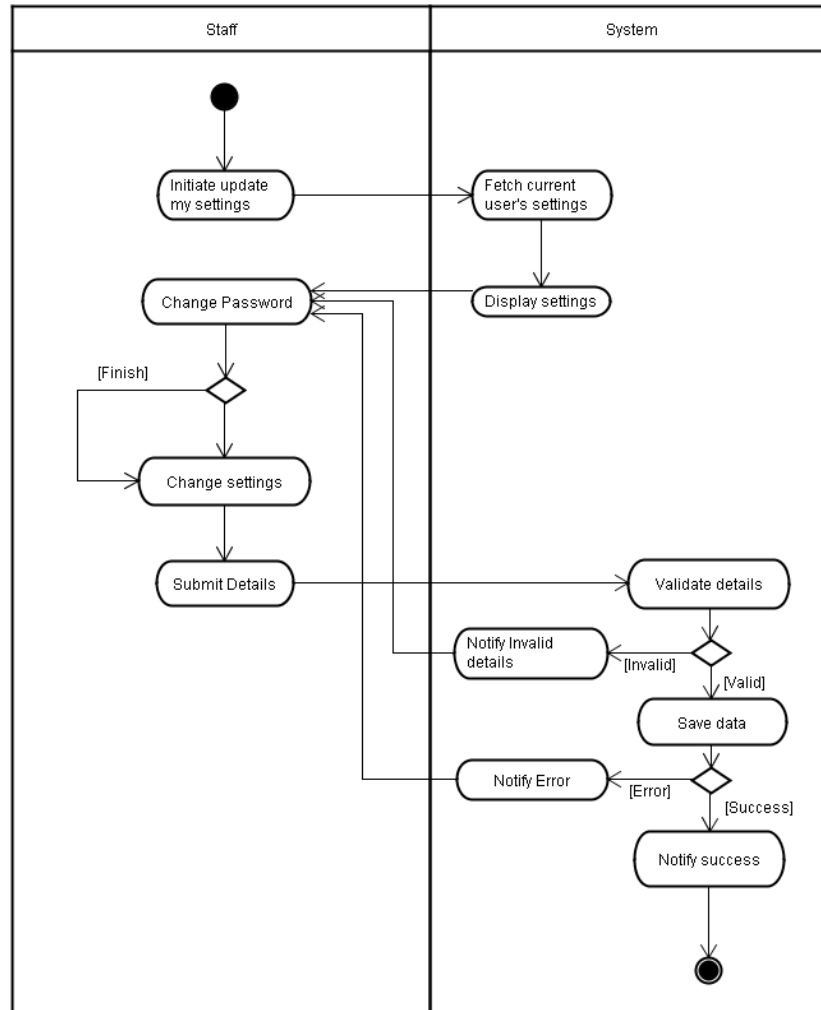
Exception flow of events – The system fails to update the staff member settings due to an error

- The staff member clicks on my setting options.
- The application retrieves the settings currently stored for the staff member.
- The application displays the staff member settings.
- The staff member changes their password by entering their current password and new password
- The staff member changes their preferences
- The staff member then click the save button.
- The system displays a message indicating an error has occurred.

Outstanding issues

None

Activity Diagrams



11.3.13View/Add Timesheet

Name:

View or Add Timesheet

Description:

Staff members can view their timesheet for a specific period and also update/save their timesheet or add a new entry. Administrator can view or edit timesheet entries for all staff members

Assumptions

- Administrators can view and edit timesheet entries for all staff

Preconditions

- None.

Post-conditions

- The updated staff member timesheet are successfully saved

Normal flow of events – Staff member successfully view their timesheet.

- The staff member clicks on view timesheet.
- The application retrieves the timesheet entries for the currently selected period.
- The application pre-populates any holidays requested in the holiday booking system for the current period
- The application displays the staff member timesheet entries.
- The staff member clicks the cancel button

Alternative flow of events – Staff member successfully update their timesheet.

- The staff member clicks on view timesheet.
- The application retrieves the timesheet entries for the currently selected period.
- The application displays the staff member timesheet entries
- The staff member updates their timesheet entries
- The staff member then saves the current changes.
- The system displays a success message

Alternative flow of events – Staff member successfully update their timesheet but cancel the changes.

- The staff member clicks on view timesheet.
- The application retrieves the timesheet entries for the currently selected period.
- The application displays the staff member timesheet entries
- The staff member updates their timesheet entries and click the cancel button
- The system settings window is closed, and underlying data is not changed.

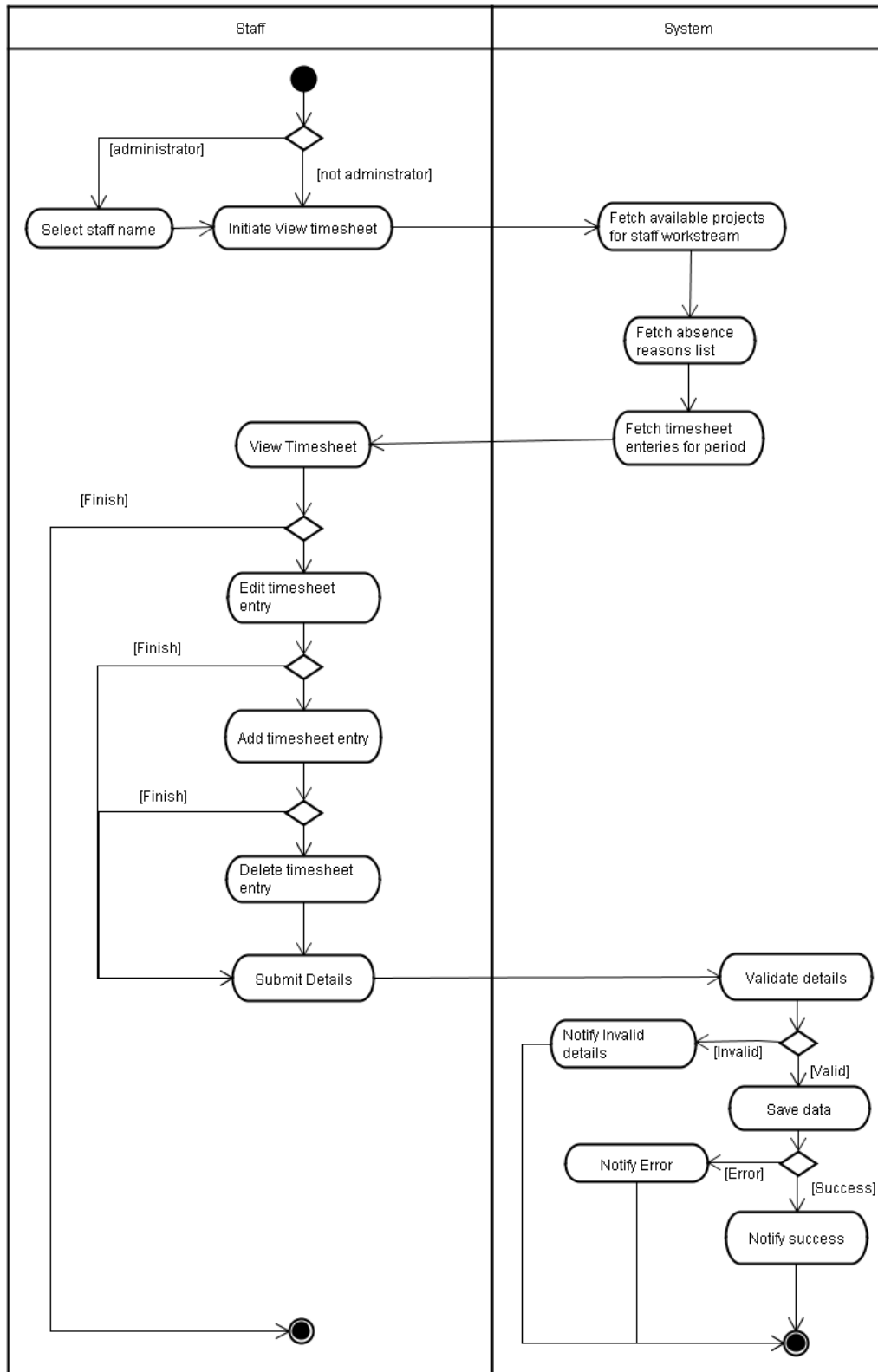
Exception flow of events – The system fails to update the staff member settings due to an error

- The staff member clicks on view timesheet.
- The application retrieves the timesheet entries for the currently selected period.
- The application displays the staff member timesheet entries
- The staff member updates their timesheet entries
- The staff member then saves the current changes.
- The system displays a message indicating an error has occurred.

Outstanding issues

None

Activity Diagrams



11.3.14View timesheet summary

Name:

View timesheet summary

Description:

Staff members views a summary of their timesheet entries

Assumptions

- Administrators can view the timesheet entries summary for all staff

Preconditions

- None.

Post-conditions

- None.

Normal flow of events – Staff member successfully view their timesheet.

- The staff member clicks on view timesheet summary.
- The staff member selects the period to view
- The application retrieves the timesheet entries summary for the currently selected period.
- The application displays the staff member timesheet entries summary.
- The staff member view the summary then clicks the close button

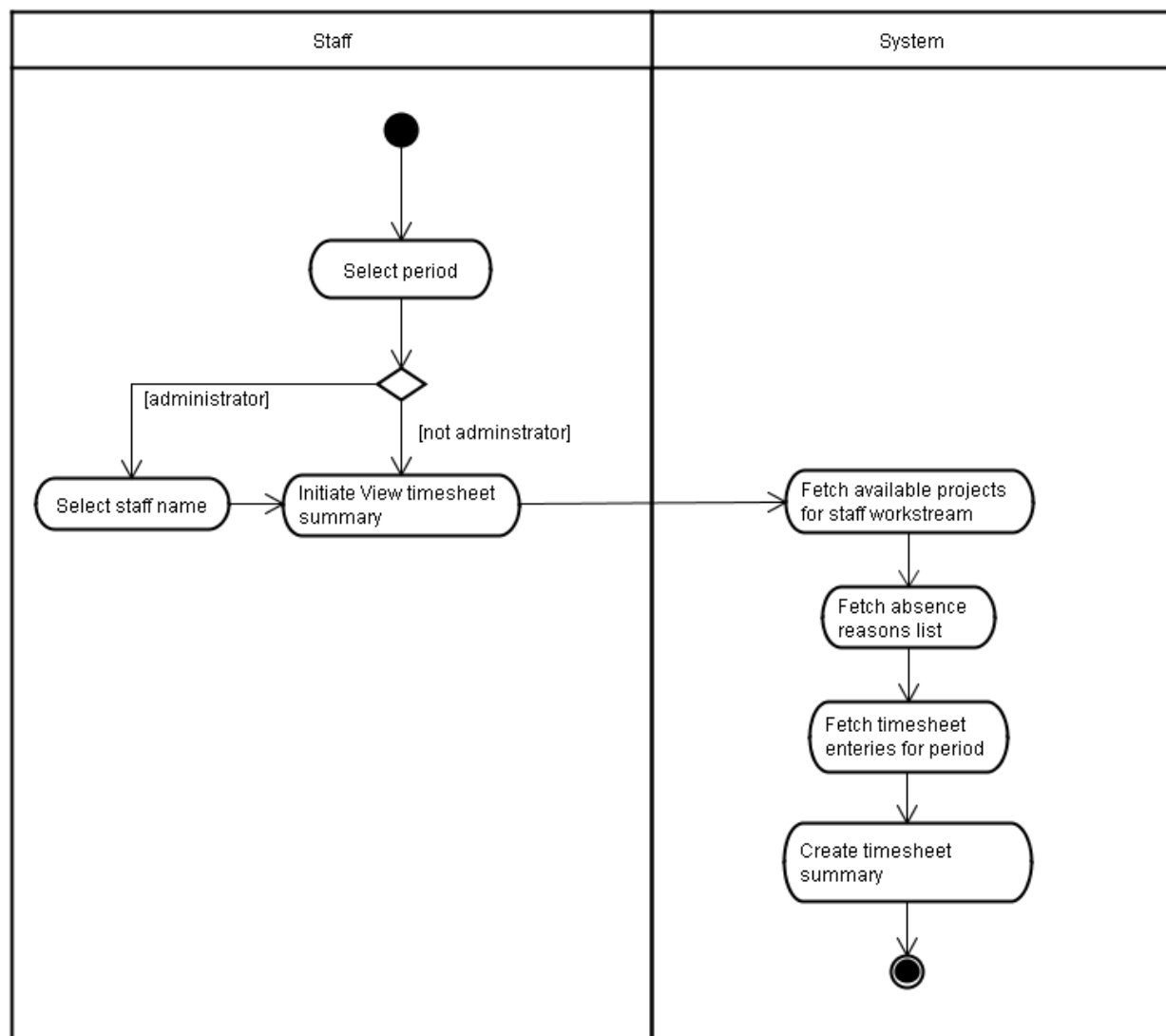
Exception flow of events – The system fails to retrieve the summary for the staff member due to an error

- The staff member clicks on view timesheet summary.
- The staff member selects the period to view
- The system displays a message indicating an error has occurred.

Outstanding issues

None

Activity Diagrams



11.3.15View Holiday Details use case

Name:

View Holiday details

Description:

Staff member can view their holiday summary and request new holiday or cancel a requested holiday. The system should also enforce the holiday roles and import holidays entered in the timebooking system. Holiday approvers should be able to view the holiday for their staff and approve or reject their holidays.

Assumptions

- Holiday approvers can update, approve or reject holiday for their staff.
- Holidays have statuses, which can be requested, approved and taken. The holiday becomes taken when entered in the timesheet.
- Each employee have holiday entitlement for each holiday year

Preconditions

- None.

Post-conditions

- None

Normal flow of events – Staff member successfully view their holiday summary.

- The staff member clicks on view holiday.
- The system retrieves the holiday for the staff member.
- The system retrieves the holidays booked in the time-booking system.
- The system displays the holiday summary for the staff member.
- The system displays approve, reject and cancel buttons for approvers.
- The staff member views the details then clicks the close button

Alternative flow of events – Staff member successfully request new holiday.

- The staff member clicks on view holiday.
- The system retrieves the holiday for the staff member.
- The system retrieves the holidays booked in the time-booking system.
- The system displays the holiday summary for the staff member.
- The staff member then request a new holiday by selecting the start date and finish date, then submits their request.
- The system saves the holiday request and creates a task for the holiday approver.
- The system displays a success message

Alternative flow of events – Staff member successfully cancels a requested holiday.

- The staff member clicks on view holiday.
- The system retrieves the holiday for the staff member.
- The system retrieves the holidays booked in the time-booking system.
- The system displays the holiday summary for the staff member.
- The staff member selects a requested holiday and clicks the cancel button.
- The system deletes the holiday request.
- The system displays a success message

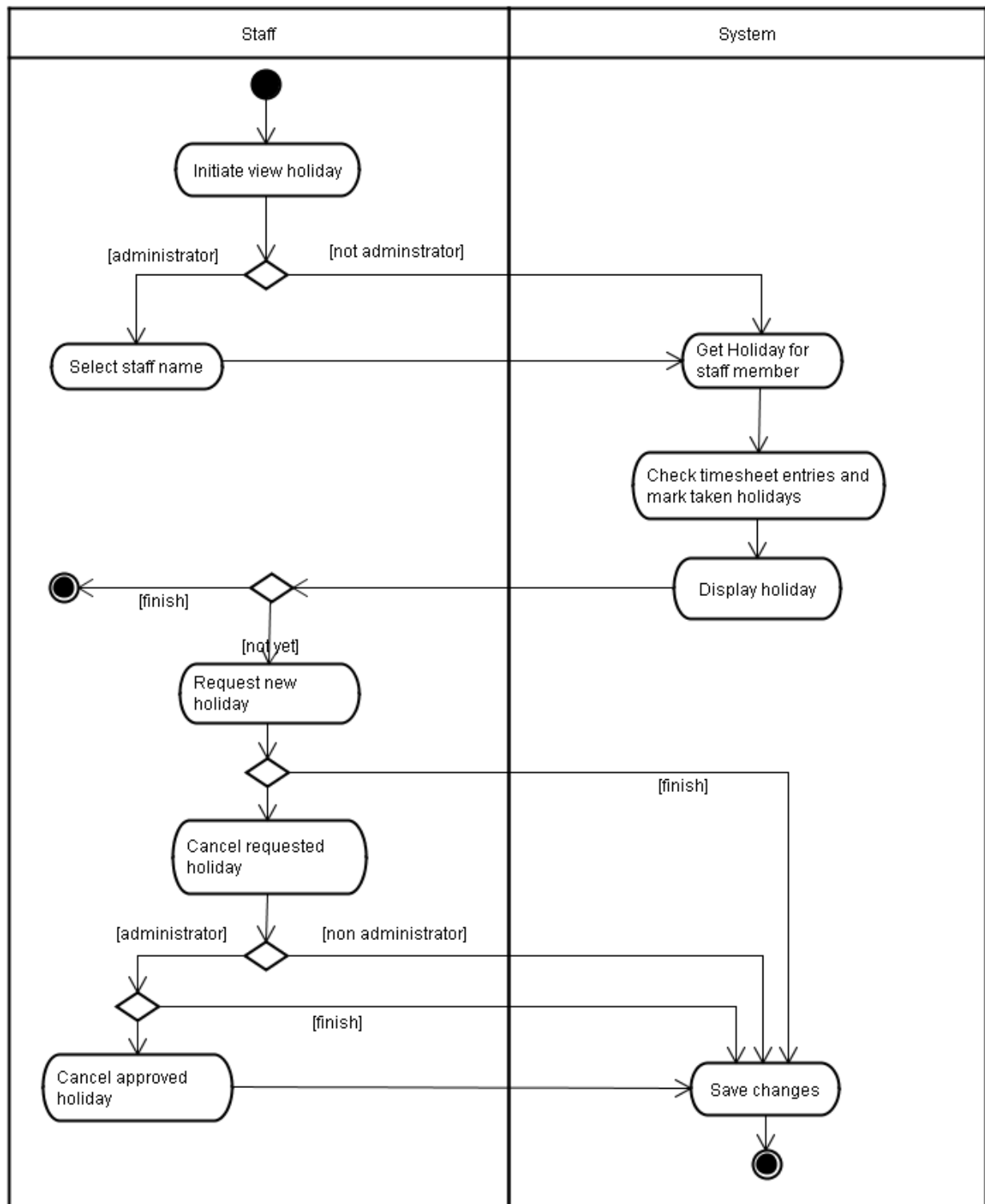
Exception flow of events – The system fails to update the staff holidays due to an error

- The staff member clicks on view holiday.
- The system retrieves the holiday for the staff member.
- The system retrieves the holidays booked in the time-booking system.
- The system displays the holiday summary for the staff member.
- The staff member then request a new holiday by selecting the start date and finish date, then submits their request.
- The system displays a message indicating an error has occurred.

Outstanding issues

None

Activity Diagrams



11.3.16View Holiday Calendar use case

Name:

View Holiday details

Description:

Staff member can view a calendar with the holiday of all any of the staff members. The staff member can also configure the number of weeks to view.

Assumptions

- None

Preconditions

- None.

Post-conditions

- None

Normal flow of events – Staff member successfully view the holiday calendar for all staff.

- The staff member clicks on view holiday calendar.
- The system retrieves the holidays requested, approved and taken for all staff members.
- The staff member views the details then click the close button.

Alternative flow of events – Staff member successfully view the holiday calendar for another staff member.

- The staff member clicks on view holiday calendar.
- The staff member selects the name of another staff member then submits
- The system retrieves the holidays requested, approved and taken for all staff members.
- The staff member views the details then click the close button

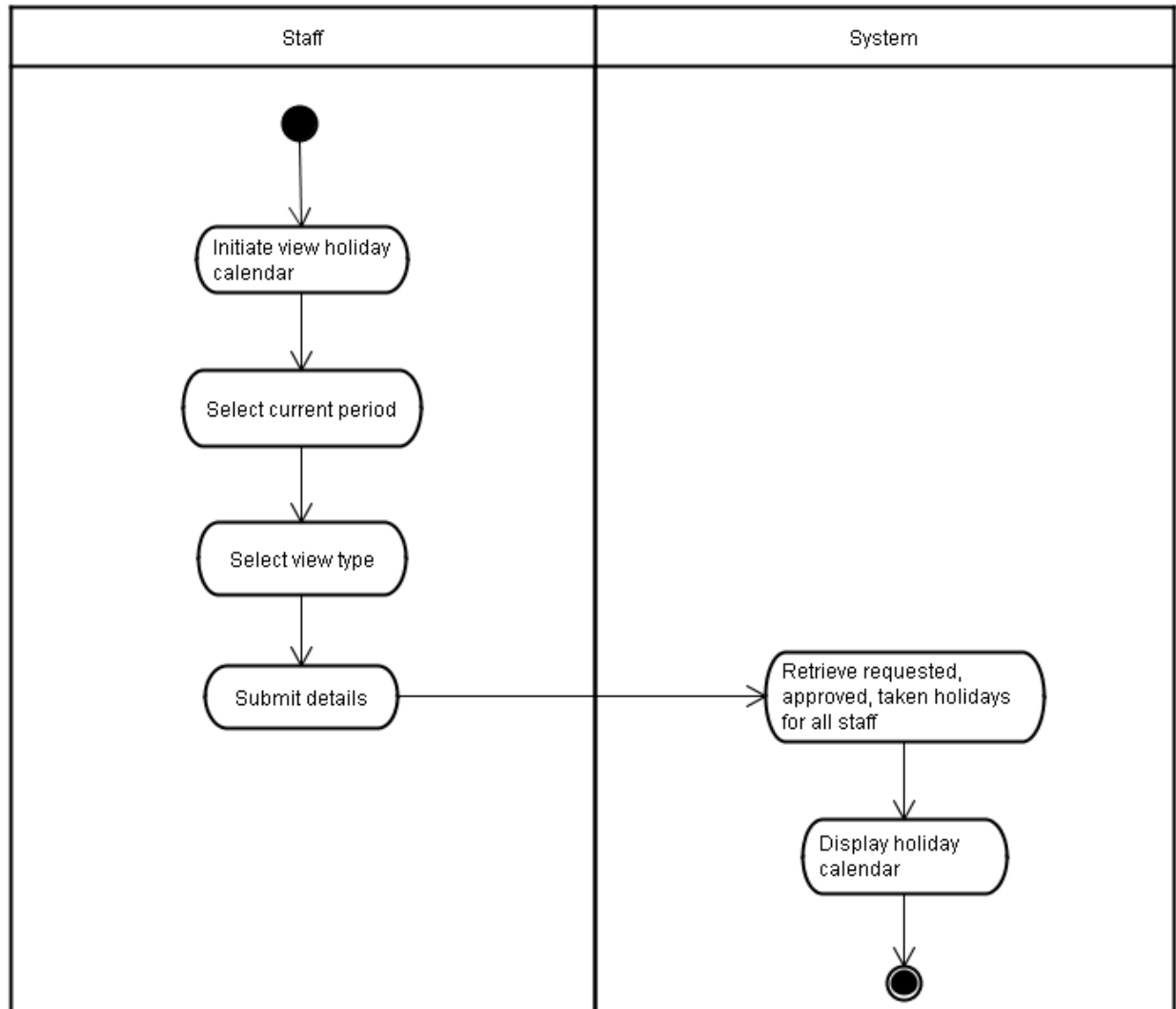
Exception flow of events – The system fails retrieve the staff holidays due to an error

- The staff member clicks on view holiday calendar.
- The staff member selects the name of another staff member then submits
- The system displays a message indicating an error has occurred.

Outstanding issues

None

Activity Diagrams



11.3.17View / Update Tasks use case

Name:

View / Update Tasks

Description:

Staff member can view the tasks assigned to them by the system and can update the task as set it as completed or delete it.

Assumptions

- None

Preconditions

- None.

Post-conditions

- After a staff member deletes a task, the task is removed from the system

Normal flow of events – Staff member successfully view their tasks.

- The staff member clicks on view tasks.
- The system retrieves all the tasks for the staff member.
- The staff member views the tasks then click the close button.

Alternative flow of events – Staff member successfully updates or delete their task.

- The staff member clicks on view tasks.
- The system retrieves all the tasks for the staff member.
- The staff member selects one of the tasks and click 'Set Complete'
- The system updates the task details and shows the task as completed.
- The staff member selects one of the tasks and click 'Delete'
- The system deletes the task details and updates the list.
- The staff member then click the close button

Exception flow of events – The system fails retrieve the staff member tasks due to an error

- The staff member clicks on view tasks.
- The system displays a message indicating an error has occurred.

Outstanding issues

None

Prototype

My Tasks

Please action your task items below

Date	Title	Description	Completed

Refresh All

Delete

Set Completed

11.4 Appendix D – Requirement Analysis Models

11.4.1 Staff management communication diagrams

The communication diagrams for the Add staff, Edit staff and View staff use cases is shown below.

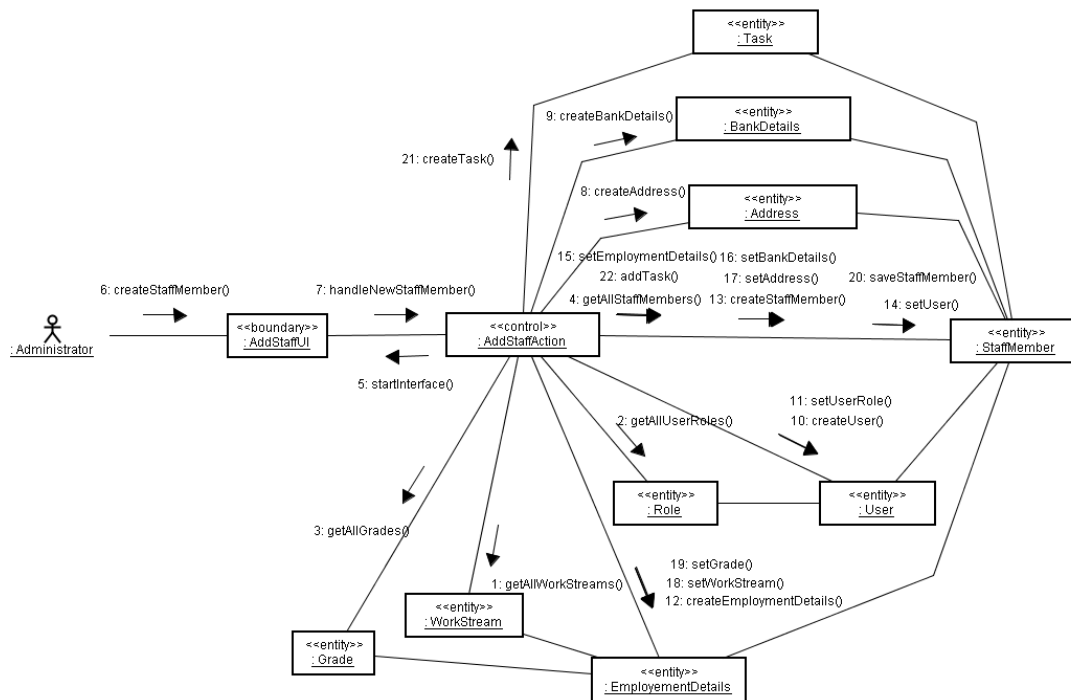


Figure 81. Add staff communication diagram

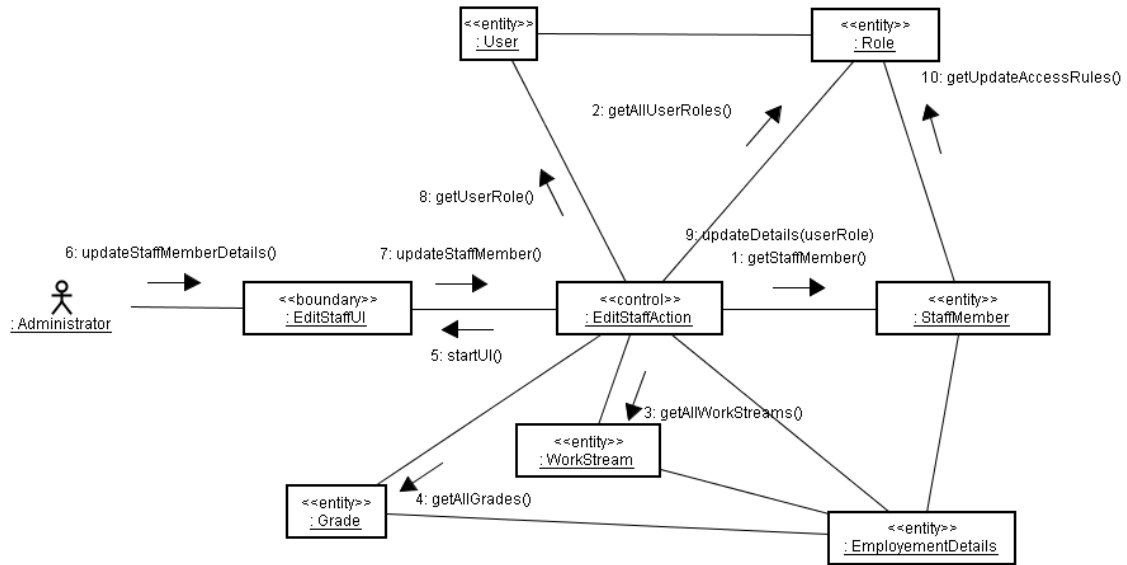


Figure 82. Edit Staff communication diagram

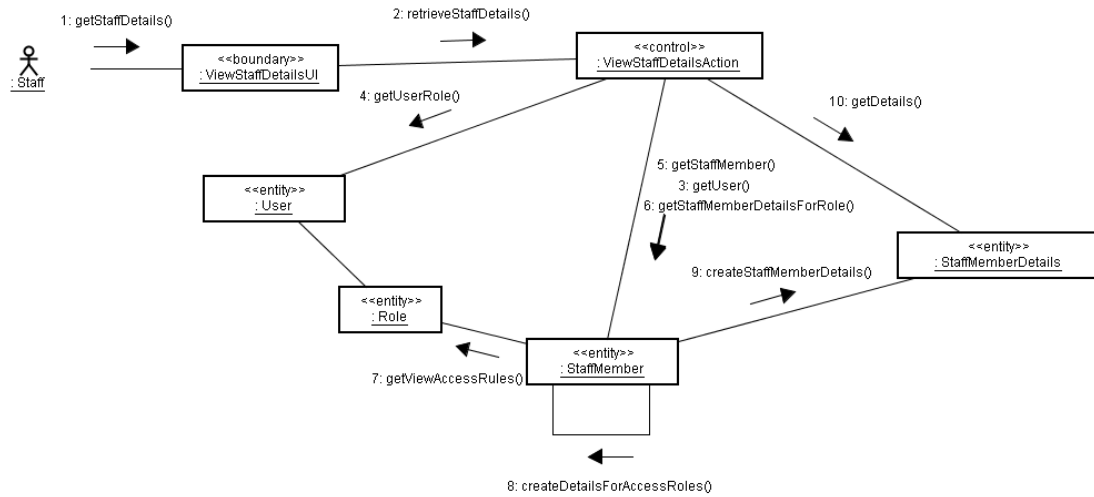


Figure 83. View Staff Details

11.4.4 Authentication and Authorisation sequence diagram

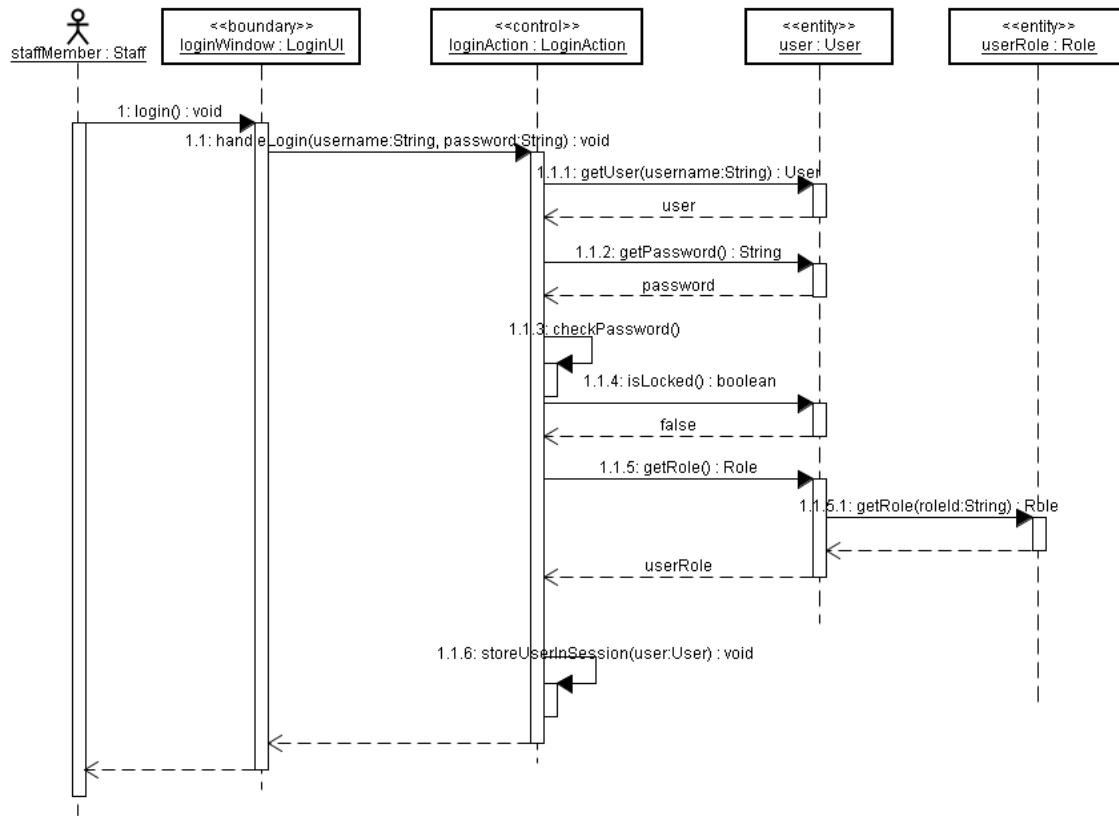


Figure 86. Authentication and Authorisation sequence diagram

11.4.5 Authentication and Authorisation analysis class diagram

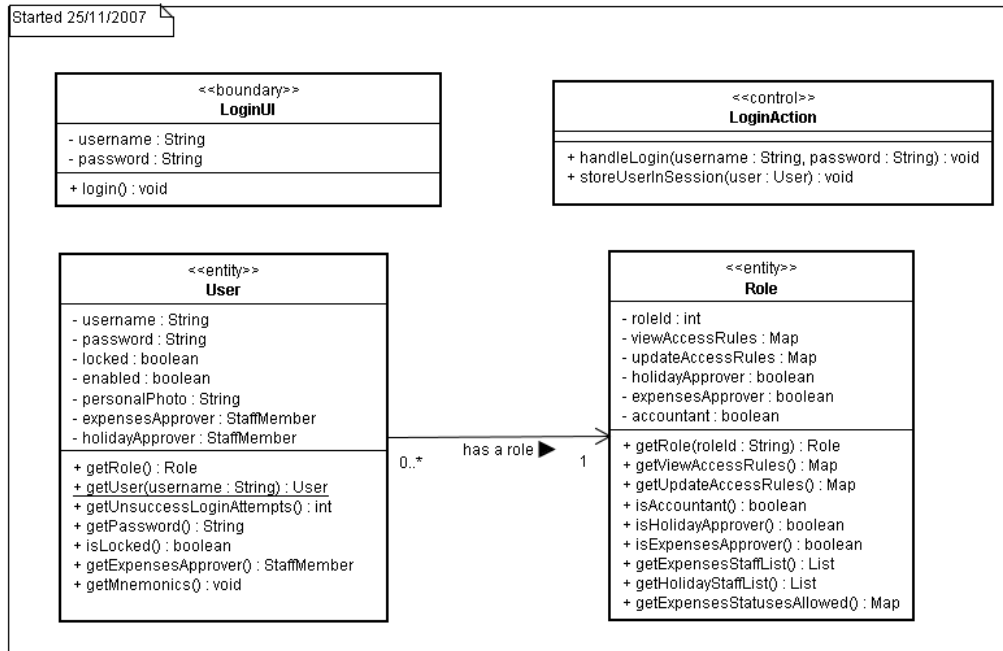


Figure 87. Authentication and Authorisation analysis class diagram

11.4.6 Expenses management communication diagrams

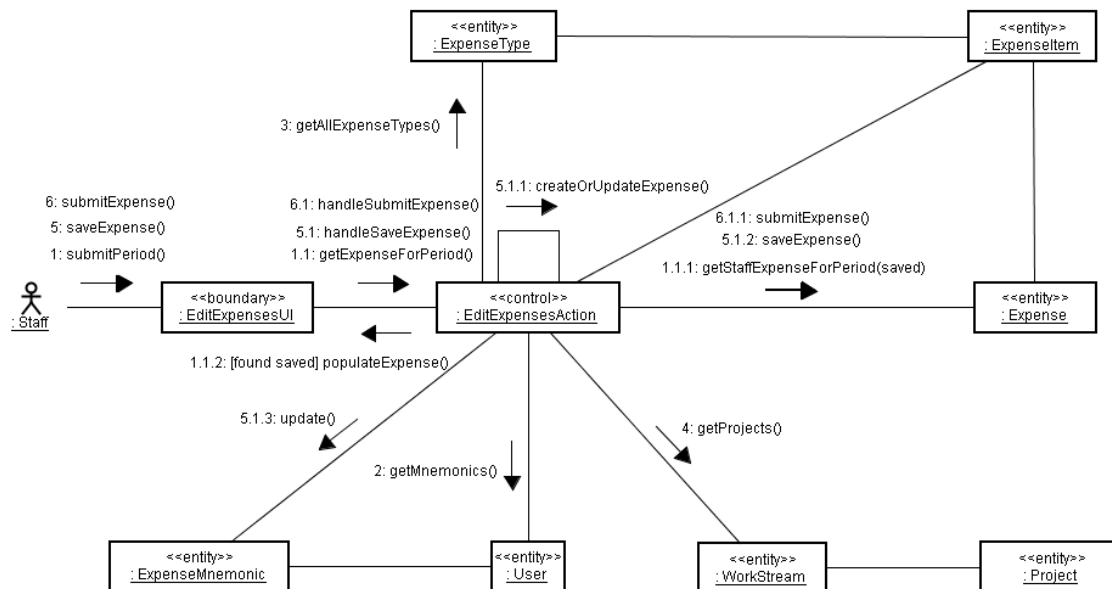


Figure 88. Add/Edit expenses communication diagram

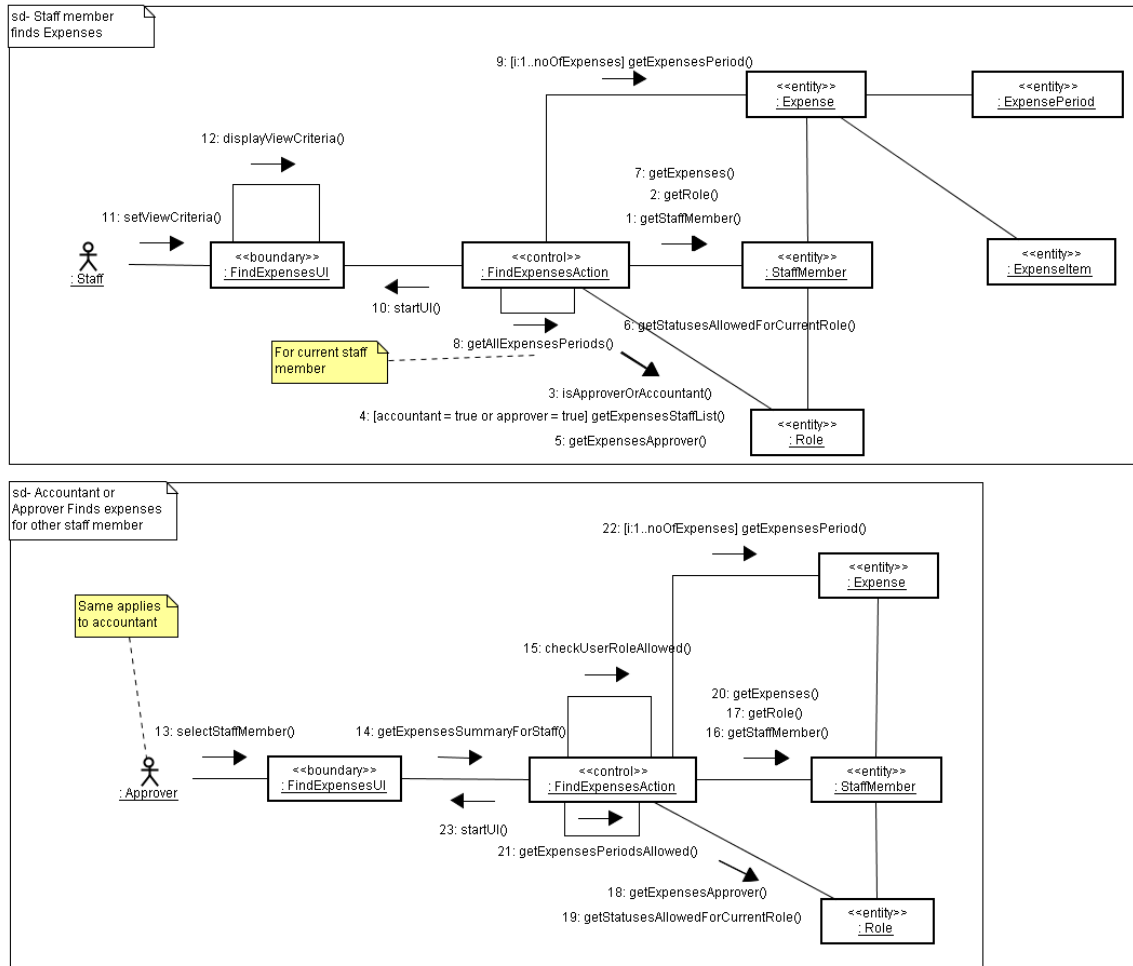


Figure 89. Find Expenses communication diagram

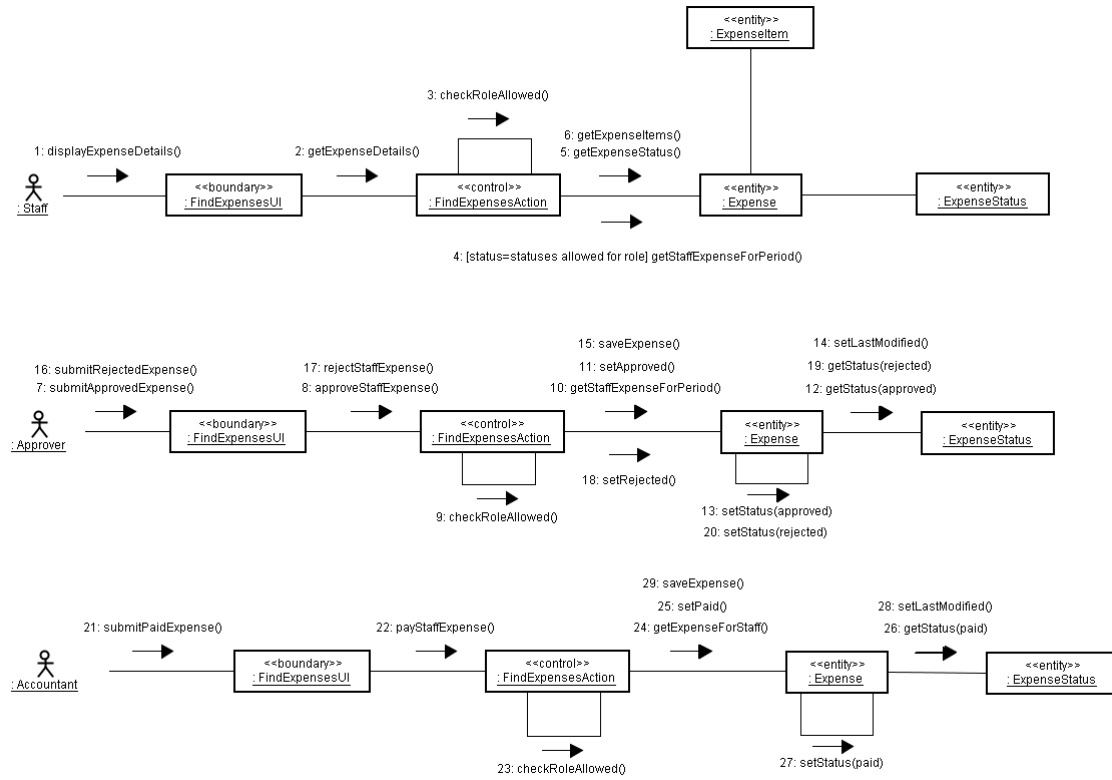
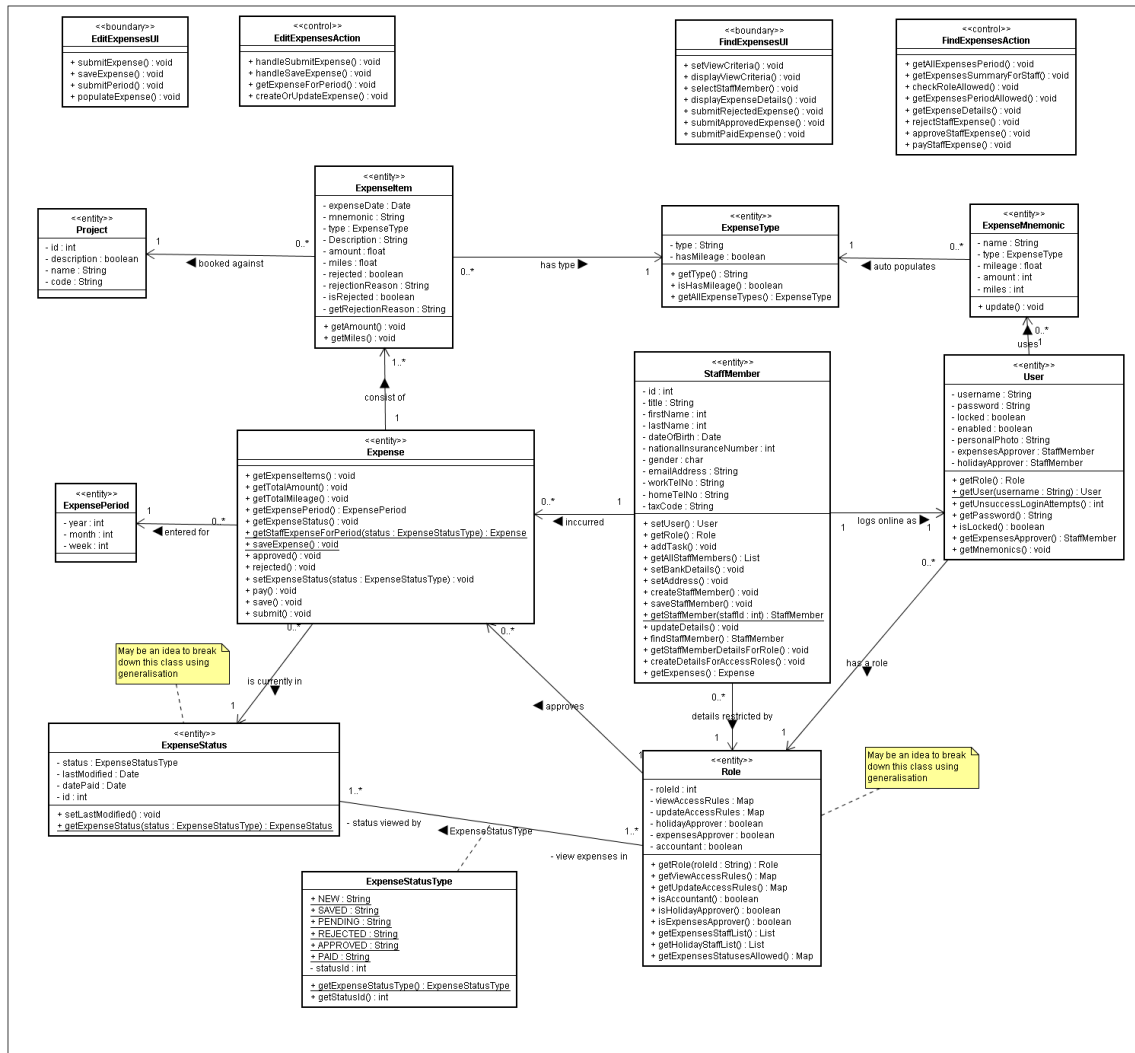


Figure 90. View Expenses communication diagram

11.4.7 Expenses management analysis class diagram



11.4.8 Expenses state diagram

As expenses are transitioned from one state to another a state diagram was constructed to model these transitions as shown below:

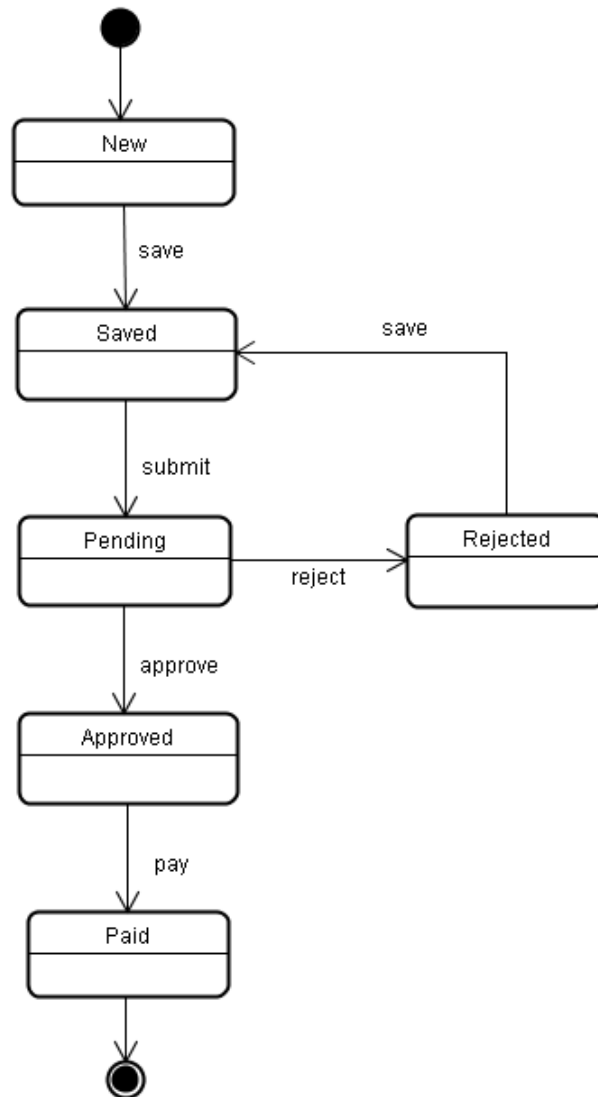


Figure 92. Expenses management analysis class diagrams

11.4.9 Holiday management communication diagrams

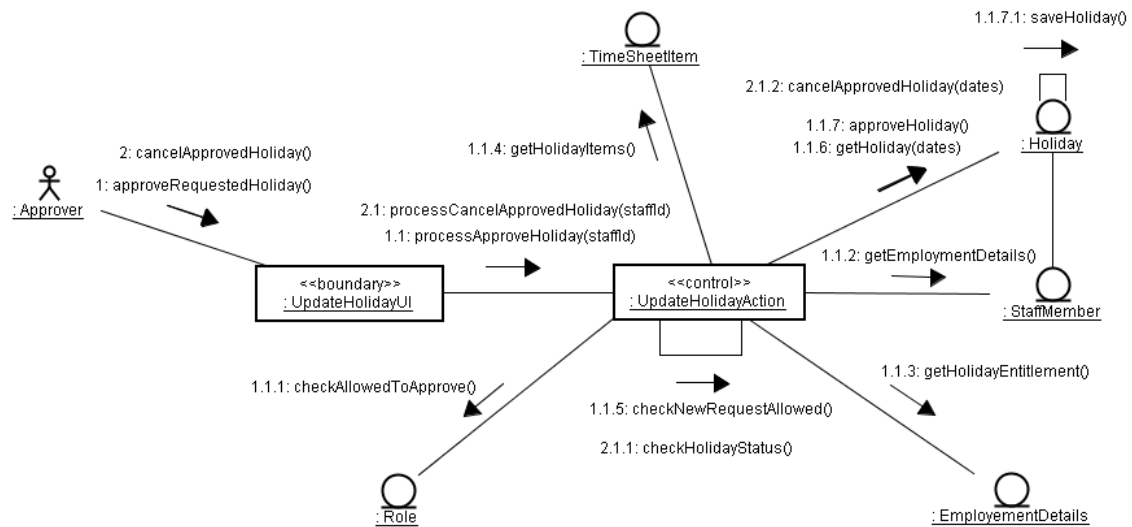


Figure 93. Approve/Cancel holiday communication diagram

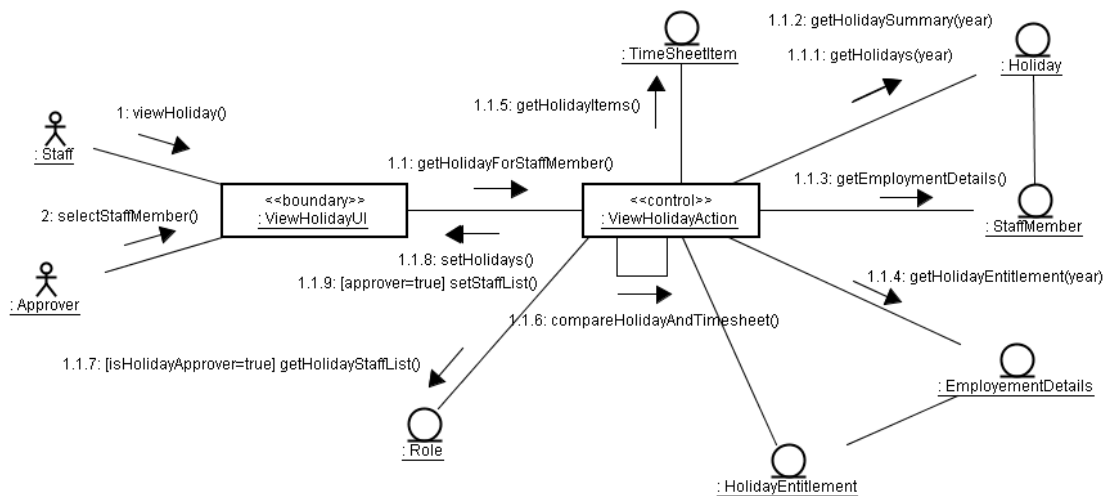


Figure 94. Request/Cancel holiday communication diagram

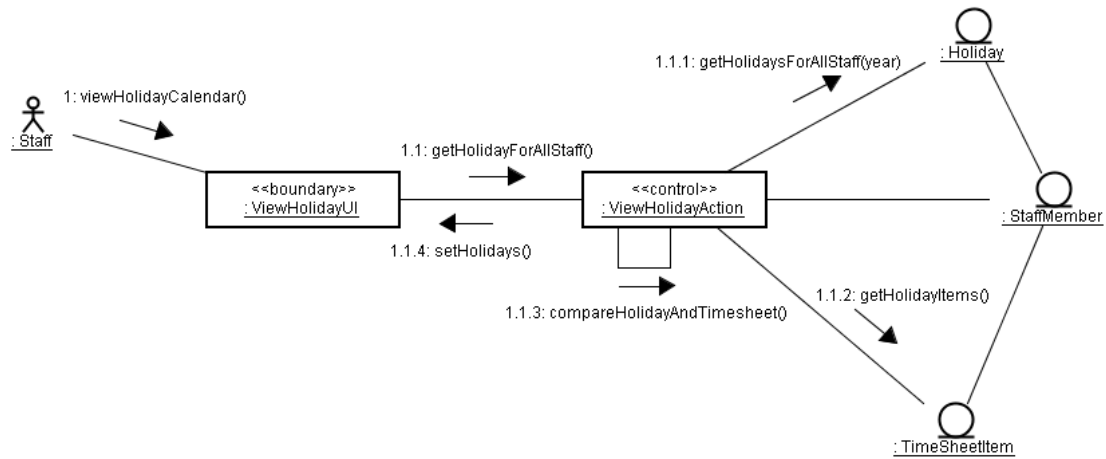


Figure 95. View holiday calendar communication diagram

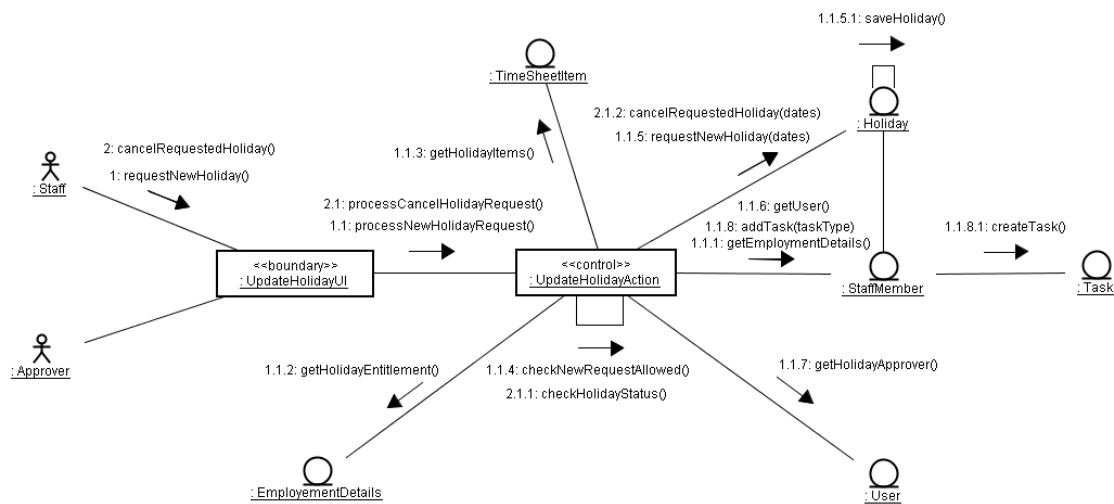


Figure 96. View Holiday details communication diagram

11.4.10 Holiday management analysis class diagram

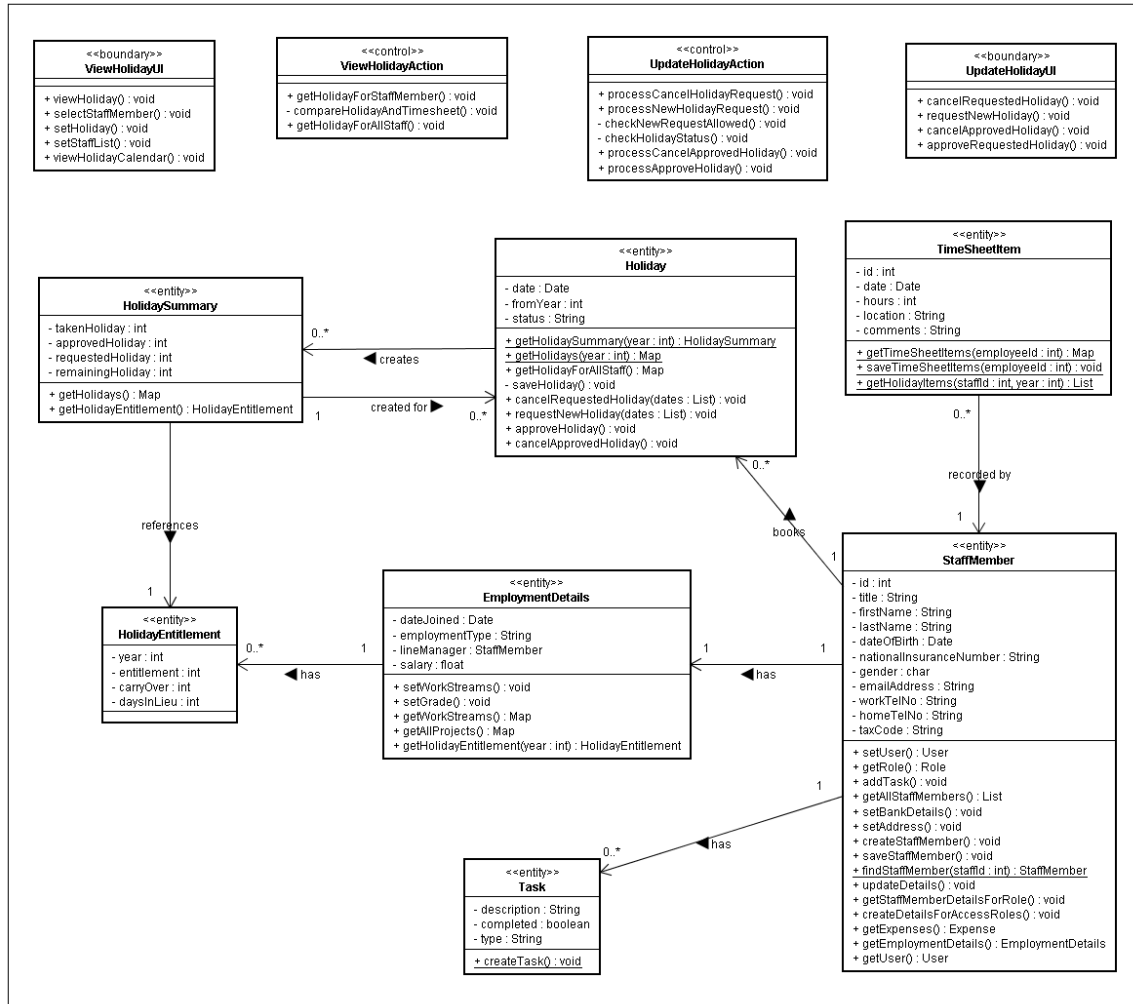


Figure 97. Holiday management analysis class diagram

11.5 Appendix E – Relational Database Model

Notations:

Bold Text	Relational algebra elements
{ }	Comments

model OfficeMA

domains

TitleTypes = (Mr, Mrs, Sir, Miss) **not allowed null**

EmploymentTypes = (Contractor, Permanent) **not allowed null**

ExpensesStatusTypes = (Saved, Pending, Approved, Rejected, Paid) **not allowed null**

HolidayStatusTypes = (Requested, Approved, Taken) **not allowed null**

GenderTypes = (Male, Female) **not allowed null**

RequiredString = string **not allowed null**

RequiredInteger = integer **not allowed null**

relation StaffMember

StaffId: integer

Dob: date **not allowed null**

EmailAddress: string

FirstName: RequiredString

LastName: RequiredString

Gender: GenderTypes

HomeTelNo: string

NiNumber: RequiredString

TaxCode: RequiredString

Title: TitleTypes

WorkTelNo: RequiredString

EmploymentDetailsId: string

Username: string

primary key StaffId

alternate key EmploymentId **not allowed null**

alternate key Username

alternate key NiNumber **not allowed null**

{logs online as}

foreign key Username **references** User

{has}

foreign key EmploymentDetailsId **references** EmploymentDetails (Id) **not allowed null**

{represent mandatory participation with respect to banks with}

constraint (project StaffMember **over** StaffId) **difference (project** BankAccount **over** StaffId) **is empty**

{represent mandatory participation with respect to lives at}

constraint (project StaffMember over StaffId) difference (project HomeAddress over StaffId) is empty

relation BankAccount

AccountNumber: RequiredString

BankName: RequiredString

SortCode: RequiredString

StaffId: integer

primary key StaffId

{banks with}

foreign key StaffId **references** StaffMember **on delete cascade**

relation HomeAddress

AddressLine1: RequiredString

AddressLine2: string

Country: RequiredString

County: string

HouseName: string

HouseNumber: string

Locality: string

PostCode: RequiredString

Town: RequiredString

StaffId: integer

primary key StaffId

{lives at}

foreign key StaffId **references** StaffMember **on delete cascade**

{ Either house name or house number or both should be supplied }

constraint ((HouseName is not null) or (HouseNumber is not null))

relation Task

Id: integer

Completed: boolean **not allowed null**

DateCreated: timestamp **not allowed null**

Description: string

TaskType: string

Title: string

StaffId: integer

primary key StaffId

{has}

foreign key StaffId **references** StaffMember **not allowed null on delete cascade**

relation User

Username: string

CanApproveExpenses: boolean **not allowed null**

CanApproveHolidays: boolean **not allowed null**

Locked: boolean **not allowed null**
 Password: RequiredString
 PersonalPhoto: string
 UnsuccessfulLoginAttempts: integer **not allowed null**
 RoleId: integer
 ExpensesApproverStaffId: integer **default 0**
 HolidayApproverStaffId: integer **default 0**
primary key Username
 {approves expenses for}
foreign key ExpensesApproverStaffId **references** StaffMember **not allowed null on delete set default**
 {approves holidays for}
foreign key HolidayApproverStaffId **references** StaffMember (StaffId) **not allowed null on delete set default**
 {has role}
foreign key RoleId **references** Role **not allowed null**
 {reflect mandatory participation with respect to logs online as}
constraint (project Users over Username) difference (project StaffMember over Username) is empty

relation EmploymentDetails

Id: integer
 DateJoined: date **not allowed null**
 DateLeft: date
 EmploymentType: EmploymentTypes
 HolidayEntitlement: integer
 Salary: decimal
 LineManagerStaffId: integer **default 0**
 GradeId: RequiredInteger
primary key Id
 {is assigned}
foreign key GradeId **references** Grade **not allowed null**
 {has line manager}
foreign key LineManagerStaffId **references** StaffMember (StaffId) **not allowed null on delete set default**
 {reflect mandatory participation with respect to has}
constraint (project EmploymentDetails over Id) difference (project StaffMember over EmploymentDetailsId) is empty

relation Grade

Id: Integer
 Code: RequiredString
 MaximumSalary: decimal
 MinimumSalary: decimal
 Name: string
primary key Id

alternate key Code **not allowed null**
alternate key Name **not allowed null**

relation EmploymentDetailsWorkStream

EmploymentDetailsId: integer

WorkStreamId: integer

{being part of the primary key reflects the mandatory participation condition with regards to perform work and works for}

primary key (EmploymentDetailsId, WorkStreamId)

{perform work}

foreign key EmploymentDetailsId **references** EmploymentDetails (Id) **on**

delete cascade

{works for}

foreign key WorkStreamId **references** WorkStreams(Id) **on delete cascade**

relation Project

Id: integer

Code: string

Description: string

Name: string

WorkStreamId: integer

primary key id

alternate key Code **not allowed null**

alternate key Name **not allowed null**

{contains}

foreign key WorkStreamId **references** WorkStream (id) **not allowed null on delete cascade**

relation WorkStream

Id: integer

Description: string

Name: string

primary key Id

alternate key Name **not allowed null**

relation Role

RoleType: string

RoleId: integer

primary key RoleId

alternate key RoleType **not allowed null**

relation ViewableExpensesStatuses

RoleId: integer

StatusName: string

primary key (RoleId, StatusName)

{view expenses in}
foreign key RoleId **references** Role **on delete cascade**

relation ViewableStaffDetails

RoleId: integer
FieldName: string
primary key (RoleId, FieldName)
{view staff details in}
foreign key RoleId **references** Role **on delete cascade**

relation UpdateableStaffDetails

RoleId: integer
FieldName: string
primary key (RoleId, FieldName)
{update staff details in}
foreign key RoleId **references** Role **on delete cascade**

relation HolidayYear

Id: integer
CarryOver: integer
DaysInLieu: integer
Entitlement: RequiredInteger
HolidayYear: RequiredInteger
StaffId: RequiredInteger
primary key Id
{ Each StaffMember participate with the HolidayYear only once for each
value HolidayYear attribute }
alternate key (StaffId, HolidayYear)
{takes}
foreign key StaffId **references** StaffMember **on delete cascade**

relation Holiday

Id: integer
AfterNoon: boolean **not allowed null**
BookedDate: date **not allowed null**
FromYear: RequiredInteger
FullDay: boolean **not allowed null**
Status: HolidayStatusTypes
HolidayYearId: RequiredInteger
primary key Id
{Only one Holiday should be booked by any employee for the same day}
alternate key (BookedDate, HolidayYearId)
{has}
foreign key HolidayYearId **references** HolidayYear (Id) **on delete cascade**

relation Expenses

Id: integer
 DatePaid: date
 StatusLastModified: timestamp **not allowed null**
 Status: ExpensesStatusTypes
 ExpensesMonth: RequiredInteger
 ExpensesWeek: RequiredInteger
 ExpensesYear: RequiredInteger
 StaffId: RequiredInteger
primary key Id
 {Each StaffMember participate with the Expenses entity only once for each value of ExpensesMonth, ExpensesWeek and ExpensesYear}
alternate key (StaffId, ExpensesYear, ExpensesMonth, ExpensesWeek)
 {books}
foreign key StaffId **references** StaffMember **on delete cascade**

relation ExpensesItem

Id: integer
 Amount: decimal **not allowed null**
 Description: string
 ExpensesDate: date **not allowed null**
 Miles: decimal
 Mnemonic: string
 Rejected: boolean **not allowed null**
 RejectionReason: string
 ExpensesCategoryId: RequiredInteger
 ExpensesId: RequiredInteger
 primary key Id
 {booked against}
foreign key ExpensesCategoryId **references** ExpensesCategory
 {consist of}
foreign key ExpensesId **reference** Expenses **on delete cascade**

relation ExpensesCategory

Id: integer
 Category: RequiredString
 HasMileage: boolean not allowed null
primary key Id
alternate key Category

relation ExpensesMnemonic

Name: RequiredString
 Amount: decimal
 Mileage: decimal
 ExpensesCategoryId: RequiredInteger
 Username: RequiredString
primary key Name

{stored for}
foreign key ExpensesCategoryId **references** ExpensesCategory(Id)
{entered}
foreign key Username **references** User **on delete cascade**
{ExpensesMnemonic must have an amount or mileage or both}
constraint ((Amount **is not null**) **or** (Mileage **is not null**))

relation MileageCost

Id: integer
Cost: decimal **not allowed null**
LowerLimit: decimal **not allowed null**
UpperLimit: decimal **not allowed null**
ExpensesCategoryId: RequiredInteger
primary key Id
{has}
foreign key ExpensesCategoryId **references** ExpensesCategory(Id) **on delete cascade**

11.6 Appendix F – Physical database schema

```
-----
--      Office Managment Application PostgreSQL schema definitions  --
--      Author: Omer Dawelbeit
--
-----

-- Drop all the Foreign keys
ALTER TABLE staff_member DROP CONSTRAINT fk_staff_member_users;
ALTER TABLE staff_member DROP CONSTRAINT
fk_staff_member_employment_details;
ALTER TABLE bank_account DROP CONSTRAINT fk_bank_account_staff_member;
ALTER TABLE home_address DROP CONSTRAINT fk_home_address_staff_member;
ALTER TABLE task DROP CONSTRAINT fk_task_staff_member;
ALTER TABLE users DROP CONSTRAINT fk_users_roles;
ALTER TABLE users DROP CONSTRAINT fk_users_staff_member_1;
ALTER TABLE users DROP CONSTRAINT fk_users_staff_member_2;
ALTER TABLE employment_details DROP CONSTRAINT
fk_employment_details_staff_member;
ALTER TABLE employment_details DROP CONSTRAINT
fk_employment_details_grade;
ALTER TABLE employment_details_workstream DROP CONSTRAINT
fk_employment_details_workstream_workstream;
ALTER TABLE employment_details_workstream DROP CONSTRAINT
fk_employment_details_workstream_employment_details;
ALTER TABLE project DROP CONSTRAINT fk_project_workstream;
ALTER TABLE role_update_staff_details DROP CONSTRAINT
fk_update_staff_details_role;
ALTER TABLE role_view_staff_details DROP CONSTRAINT
fk_view_staff_details_role;
ALTER TABLE role_allowed_expenses_statuses DROP CONSTRAINT
fk_expenses_statuses_role;
ALTER TABLE holiday_year DROP CONSTRAINT fk_holiday_year_staff_member;
ALTER TABLE holiday DROP CONSTRAINT fk_holiday_holiday_year;
ALTER TABLE expenses DROP CONSTRAINT fk_expenses_staff_member;
ALTER TABLE expenses_item DROP CONSTRAINT fk_expenses_item_expenses;
ALTER TABLE expenses_item DROP CONSTRAINT
fk_expenses_item_expenses_category;
ALTER TABLE expenses_item DROP CONSTRAINT fk_expenses_item_project;
ALTER TABLE expenses_mnemonic DROP CONSTRAINT
fk_expenses_mnemonic_expenses_category;
ALTER TABLE expenses_mnemonic DROP CONSTRAINT
fk_expenses_mnemonic_users;
ALTER TABLE mileage_cost DROP CONSTRAINT
fk_mileage_cost_expenses_category;

-- Drop all the primary keys
ALTER TABLE mileage_cost DROP CONSTRAINT mileage_cost_pkey;
ALTER TABLE expenses DROP CONSTRAINT expenses_pkey;
ALTER TABLE staff_member DROP CONSTRAINT staff_member_pkey;
ALTER TABLE employment_details_workstream DROP CONSTRAINT
employment_details_workstream_pkey;
ALTER TABLE users DROP CONSTRAINT users_pkey;
ALTER TABLE bank_account DROP CONSTRAINT bank_account_pkey;
ALTER TABLE expenses_mnemonic DROP CONSTRAINT expenses_mnemonic_pkey;
```

```

ALTER TABLE workstream DROP CONSTRAINT workstream_pkey;
ALTER TABLE roles DROP CONSTRAINT roles_pkey;
ALTER TABLE expenses_item DROP CONSTRAINT expenses_item_pkey;
ALTER TABLE project DROP CONSTRAINT project_pkey;
ALTER TABLE grade DROP CONSTRAINT grade_pkey;
ALTER TABLE expenses_category DROP CONSTRAINT expenses_category_pkey;
ALTER TABLE task DROP CONSTRAINT task_pkey;
ALTER TABLE home_address DROP CONSTRAINT home_address_pkey;
ALTER TABLE employment_details DROP CONSTRAINT employment_details_pkey;
ALTER TABLE holiday_year DROP CONSTRAINT holiday_year_pkey;
ALTER TABLE holiday DROP CONSTRAINT holiday_pkey;
ALTER TABLE role_view_staff_details DROP CONSTRAINT
role_view_staff_details_pkey ;
ALTER TABLE role_update_staff_details DROP CONSTRAINT
role_update_staff_details_pkey;
ALTER TABLE role_allowed_expenses_statuses DROP CONSTRAINT
role_allowed_expenses_statuses_pkey;

-- Drop all the alternate keys
ALTER TABLE staff_member DROP CONSTRAINT
staff_member_employmentdetails_id_key;
ALTER TABLE staff_member DROP CONSTRAINT staff_member_username_key;
ALTER TABLE staff_member DROP CONSTRAINT staff_member_ni_number_key;
ALTER TABLE grade DROP CONSTRAINT grade_code_key;
ALTER TABLE grade DROP CONSTRAINT grade_name_key;
ALTER TABLE project DROP CONSTRAINT project_code_key;
ALTER TABLE project DROP CONSTRAINT project_name_key;
ALTER TABLE workstream DROP CONSTRAINT workstream_name_key;
ALTER TABLE roles DROP CONSTRAINT roles_role_type_key;
ALTER TABLE holiday_year DROP CONSTRAINT holiday_year_staff_id_key;
ALTER TABLE holiday DROP CONSTRAINT holiday_holiday_year_key;
ALTER TABLE expenses DROP CONSTRAINT expenses_staff_id_key;
ALTER TABLE expenses_category DROP CONSTRAINT
expenses_category_category_key;

-- Drop the check constraints
ALTER TABLE home_address DROP CONSTRAINT home_address_check;
ALTER TABLE expenses_mnemonic DROP CONSTRAINT expenses_mnemonic_check;
ALTER TABLE expenses_item DROP CONSTRAINT expenses_item_check;

-- Drop the triggers and functions
DROP TRIGGER mileage_cost_check ON mileage_cost;
DROP FUNCTION mileage_cost_check();

DROP TRIGGER users_check ON staff_member;
DROP FUNCTION users_check();

-- Drop all the tables
DROP TABLE roles;
DROP TABLE grade;
DROP TABLE expenses_mnemonic;
DROP TABLE holiday_year;
DROP TABLE project;
DROP TABLE expenses_item;
DROP TABLE role_allowed_expenses_statuses;
DROP TABLE users;
DROP TABLE expenses_category;

```

```

DROP TABLE role_update_staff_details;
DROP TABLE mileage_cost;
DROP TABLE task;
DROP TABLE holiday;
DROP TABLE employment_details;
DROP TABLE workstream;
DROP TABLE home_address;
DROP TABLE expenses;
DROP TABLE employment_details_workstream;
DROP TABLE role_view_staff_details;
DROP TABLE staff_member;
DROP TABLE bank_account;

-- Drop the sequences
DROP SEQUENCE hibernate_sequence;

-- Drop the domains
DROP DOMAIN titletypes;
DROP DOMAIN employmenttypes;
DROP DOMAIN expensesstatustypes;
DROP DOMAIN holidaystatustypes;
DROP DOMAIN gendertypes;

-- Schema domain definitions
CREATE DOMAIN titletypes
  AS VARCHAR(10) NOT NULL
  CHECK (VALUE IN ('Mr', 'Mrs', 'Sir', 'Miss'));

CREATE DOMAIN employmenttypes
  AS VARCHAR(10) NOT NULL
  CHECK (VALUE IN ('Contractor', 'Permanent'));

CREATE DOMAIN expensesstatustypes
  AS VARCHAR(10) NOT NULL
  CHECK (VALUE IN ('Saved', 'Pending', 'Approved', 'Rejected', 'Paid'));

CREATE DOMAIN holidaystatustypes
  AS VARCHAR(10) NOT NULL
  CHECK (VALUE IN ('Requested', 'Approved', 'Taken'));

CREATE DOMAIN gendertypes
  AS VARCHAR(10) NOT NULL
  CHECK (VALUE IN ('Male', 'Female'));

-- Hibernate sequence used to auto-generate surrogate primary keys
CREATE SEQUENCE hibernate_sequence
  INCREMENT 1
  MINVALUE 10
  MAXVALUE 9223372036854775807
  START 71
  CACHE 1;

-- Table definition statements
CREATE TABLE staff_member (
  staff_id INTEGER NOT NULL,
  dob DATE NOT NULL,
  emailaddress VARCHAR(255),

```

```

        first_name VARCHAR(255) NOT NULL,
        gender gendertypes,
        hometelno VARCHAR(255),
        last_name VARCHAR(255) NOT NULL,
        ni_number VARCHAR(255) NOT NULL,
        tax_code VARCHAR(255) NOT NULL,
        title titletypes,
        worktelno VARCHAR(255) NOT NULL,
        username VARCHAR(255),
        employmentdetails_id INTEGER NOT NULL
    );

CREATE TABLE bank_account (
    account_number VARCHAR(255) NOT NULL,
    bank_name VARCHAR(255) NOT NULL,
    sort_code VARCHAR(255) NOT NULL,
    staff_id INTEGER NOT NULL
);

CREATE TABLE home_address (
    address_line1 VARCHAR(255) NOT NULL,
    address_line2 VARCHAR(255),
    country VARCHAR(8) NOT NULL,
    county VARCHAR(255),
    house_name VARCHAR(255),
    house_number VARCHAR(255),
    locality VARCHAR(255),
    post_code VARCHAR(255) NOT NULL,
    town VARCHAR(255) NOT NULL,
    staff_id INTEGER NOT NULL
);

CREATE TABLE task (
    id INTEGER NOT NULL,
    completed BOOL NOT NULL,
    datecreated TIMESTAMP NOT NULL,
    description VARCHAR(255),
    tasktype VARCHAR(255),
    title VARCHAR(255),
    staff_id INTEGER NOT NULL
);

CREATE TABLE users (
    username VARCHAR(255) NOT NULL,
    canapproveexpenses BOOL NOT NULL,
    canapproveholidays BOOL NOT NULL,
    locked BOOL NOT NULL,
    password VARCHAR(255) NOT NULL,
    personalphoto VARCHAR(255),
    unsuccessfulloginattempts INTEGER NOT NULL,
    role_id INTEGER NOT NULL,
    expensesapprover_staff_id INTEGER NOT NULL DEFAULT 1,
    holidayapprover_staff_id INTEGER NOT NULL DEFAULT 1
);

CREATE TABLE employment_details (
    id INTEGER NOT NULL,

```

```

        datejoined DATE NOT NULL,
        dateleft DATE,
        employmenttype employmenttypes,
        holidayentitlement INTEGER,
        salary NUMERIC(19 , 2),
        grade_id INTEGER NOT NULL,
        linemanager_staff_id INTEGER NOT NULL DEFAULT 1
    );

CREATE TABLE grade (
    id INTEGER NOT NULL,
    code VARCHAR(255) NOT NULL,
    maximumsalary NUMERIC(19 , 2),
    minimumsalary NUMERIC(19 , 2),
    name VARCHAR(255) NOT NULL
);

CREATE TABLE employment_details_workstream (
    employment_details_id INTEGER NOT NULL,
    workstreams_id INTEGER NOT NULL
);

CREATE TABLE project (
    id INTEGER NOT NULL,
    code VARCHAR(255) NOT NULL,
    description VARCHAR(255),
    name VARCHAR(255) NOT NULL,
    workstream_id INTEGER NOT NULL
);

CREATE TABLE workstream (
    id INTEGER NOT NULL,
    description VARCHAR(255),
    name VARCHAR(255) NOT NULL
);

CREATE TABLE roles (
    role_type VARCHAR(31) NOT NULL,
    role_id INTEGER NOT NULL
);

CREATE TABLE role_view_staff_details (
    role_id INTEGER NOT NULL,
    field_name VARCHAR(255)
);

CREATE TABLE role_update_staff_details (
    role_id INTEGER NOT NULL,
    field_name VARCHAR(255)
);

CREATE TABLE role_allowed_expenses_statuses (
    role_id INTEGER NOT NULL,
    status_name VARCHAR(255)
);

CREATE TABLE holiday_year (

```



```

        id INTEGER NOT NULL,
        carry_over INTEGER,
        days_in_lieu INTEGER,
        entitlement INTEGER NOT NULL,
        holiday_year INTEGER NOT NULL,
        staff_id INTEGER NOT NULL
    );

CREATE TABLE holiday (
    id INTEGER NOT NULL,
    after_noon BOOL NOT NULL,
    booked_date DATE NOT NULL,
    from_year INTEGER NOT NULL,
    full_day BOOL NOT NULL,
    status holidaystatustypes,
    holiday_year_id INTEGER NOT NULL
);

CREATE TABLE expenses (
    id INTEGER NOT NULL,
    date_paid DATE,
    status_last_modified TIMESTAMP NOT NULL,
    status expensesstatustypes,
    expenses_month INTEGER NOT NULL,
    expenses_week INTEGER NOT NULL,
    expenses_year INTEGER NOT NULL,
    staff_id INTEGER NOT NULL
);

CREATE TABLE expenses_item (
    id INTEGER NOT NULL,
    amount NUMERIC(19 , 2),
    description VARCHAR(255),
    expense_date DATE NOT NULL,
    miles NUMERIC(19 , 2),
    mnemonic VARCHAR(255),
    rejected BOOL NOT NULL,
    rejection_reason VARCHAR(255),
    project_id INTEGER,
    expenses_category_id INTEGER NOT NULL,
    expenses_id INTEGER NOT NULL
);

CREATE TABLE expenses_category (
    id INTEGER NOT NULL,
    expenses_type VARCHAR(255) NOT NULL,
    hasmileage BOOL NOT NULL
);

CREATE TABLE expenses_mnemonic (
    name VARCHAR(255) NOT NULL,
    amount NUMERIC(19 , 2),
    mileage NUMERIC(19 , 2),
    expenses_category_id INTEGER NOT NULL,
    username VARCHAR(255) NOT NULL
);

```

```

CREATE TABLE mileage_cost (
    id INTEGER NOT NULL,
    cost NUMERIC(19 , 2) NOT NULL,
    lower_limit NUMERIC(19 , 2) NOT NULL,
    upper_limit NUMERIC(19 , 2) NOT NULL,
    expenses_category_id INTEGER
);

-- primary key constraints
ALTER TABLE mileage_cost ADD CONSTRAINT mileage_cost_pkey PRIMARY KEY
(id);
ALTER TABLE expenses ADD CONSTRAINT expenses_pkey PRIMARY KEY (id);
ALTER TABLE staff_member ADD CONSTRAINT staff_member_pkey PRIMARY KEY
(staff_id);
ALTER TABLE employment_details_workstream ADD CONSTRAINT
employment_details_workstream_pkey PRIMARY KEY (employment_details_id,
workstreams_id);
ALTER TABLE users ADD CONSTRAINT users_pkey PRIMARY KEY (username);
ALTER TABLE bank_account ADD CONSTRAINT bank_account_pkey PRIMARY KEY
(staff_id);
ALTER TABLE expenses_mnemonic ADD CONSTRAINT expenses_mnemonic_pkey
PRIMARY KEY (name);
ALTER TABLE workstream ADD CONSTRAINT workstream_pkey PRIMARY KEY (id);
ALTER TABLE roles ADD CONSTRAINT roles_pkey PRIMARY KEY (role_id);
ALTER TABLE expenses_item ADD CONSTRAINT expenses_item_pkey PRIMARY KEY
(id);
ALTER TABLE project ADD CONSTRAINT project_pkey PRIMARY KEY (id);
ALTER TABLE grade ADD CONSTRAINT grade_pkey PRIMARY KEY (id);
ALTER TABLE expenses_category ADD CONSTRAINT expenses_category_pkey
PRIMARY KEY (id);
ALTER TABLE task ADD CONSTRAINT task_pkey PRIMARY KEY (id);
ALTER TABLE home_address ADD CONSTRAINT home_address_pkey PRIMARY KEY
(staff_id);
ALTER TABLE employment_details ADD CONSTRAINT employment_details_pkey
PRIMARY KEY (id);
ALTER TABLE holiday_year ADD CONSTRAINT holiday_year_pkey PRIMARY KEY
(id);
ALTER TABLE holiday ADD CONSTRAINT holiday_pkey PRIMARY KEY (id);
ALTER TABLE role_view_staff_details ADD CONSTRAINT
role_view_staff_details_pkey PRIMARY KEY (role_id, field_name);
ALTER TABLE role_update_staff_details ADD CONSTRAINT
role_update_staff_details_pkey PRIMARY KEY (role_id, field_name);
ALTER TABLE role_allowed_expenses_statuses ADD CONSTRAINT
role_allowed_expenses_statuses_pkey PRIMARY KEY (role_id, status_name);

-- Alternate key constraints
ALTER TABLE staff_member ADD CONSTRAINT
staff_member_employmentdetails_id_key UNIQUE (employmentdetails_id);
ALTER TABLE staff_member ADD CONSTRAINT staff_member_username_key UNIQUE
(username);
ALTER TABLE staff_member ADD CONSTRAINT staff_member_ni_number_key
UNIQUE (ni_number);
ALTER TABLE grade ADD CONSTRAINT grade_code_key UNIQUE (code);
ALTER TABLE grade ADD CONSTRAINT grade_name_key UNIQUE (name);
ALTER TABLE project ADD CONSTRAINT project_code_key UNIQUE (code);
ALTER TABLE project ADD CONSTRAINT project_name_key UNIQUE (name);
ALTER TABLE workstream ADD CONSTRAINT workstream name key UNIQUE (name);

```

```

ALTER TABLE roles ADD CONSTRAINT roles_role_type_key UNIQUE (role_type);
ALTER TABLE holiday_year ADD CONSTRAINT holiday_year_staff_id_key UNIQUE
(staff_id, holiday_year);
ALTER TABLE holiday ADD CONSTRAINT holiday_holiday_year_key UNIQUE
(holiday_year_id, booked_date);
ALTER TABLE expenses ADD CONSTRAINT expenses_staff_id_key UNIQUE
(staff_id, expenses_year, expenses_month, expenses_week);
ALTER TABLE expenses_category ADD CONSTRAINT
expenses_category_category_key UNIQUE (expenses_type);

-- Create default data before creating referential constraints
-- Default Grades
INSERT INTO grade (id, code, maximumsalary, minimumsalary, name) VALUES
(1, 'SC', 52000.00, 42000.00, 'Senior Consultant');
INSERT INTO grade (id, code, maximumsalary, minimumsalary, name) VALUES
(2, 'C', 41999.00, 32000.00, 'Consultant');
INSERT INTO grade (id, code, maximumsalary, minimumsalary, name) VALUES
(3, 'PC', 62000.00, 52000.00, 'Principal Consultant');

-- Default Roles
INSERT INTO roles (role_type, role_id) VALUES ('Accountant', 1);
INSERT INTO roles (role_type, role_id) VALUES ('Administrator', 2);
INSERT INTO roles (role_type, role_id) VALUES ('RegularStaff', 3);

-- Default Administrator (admin/officema)
INSERT INTO staff_member (staff_id, dob, emailaddress, first_name,
gender, hometelno, last_name, ni_number, tax_code, title,
worktelno, username, employmentdetails_id)
VALUES (1, '2008-01-20', NULL, 'Admin', 'Male', NULL, 'Admin',
'XXXXXXX', 'XXX', 'Mr', '0000000000', 'admin', 1);
INSERT INTO employment_details (id, datejoined, dateleft,
employmenttype, salary, linemanager_staff_id, grade_id,
holidayentitlement)
VALUES (1, '2008-01-20', NULL, 'Permanent', 0.00, 1, 1, 0);
INSERT INTO users (username, canapproveexpenses, canapproveholidays,
locked, "password", personalphoto, unsuccessfulloginattempts,
holidayapprover_staff_id, expensesapprover_staff_id, role_id)
VALUES ('admin', true, true, false, 'sxd7SMla0HyxGct2bvF3jw==',
NULL, 0, 1, 1, 2);
INSERT INTO bank_account (account_number, bank_name, sort_code,
staff_id) VALUES ('AC5Ivq3PGmMcx0M20FCpFA==',
'zN3paiWMI9tZwC9sMEwpwg==', 'yewjnIn286k=', 1);
INSERT INTO home_address (address_line1, address_line2, country, county,
house_name, house_number, locality, post_code, town, staff_id) VALUES
('Some Road', NULL, 'UK', NULL, NULL, '1', NULL, 'AA1 1AA', 'Town', 1);

-- Foreign key constraints
ALTER TABLE staff_member ADD CONSTRAINT fk_staff_member_users FOREIGN
KEY (username) REFERENCES users (username);
ALTER TABLE staff_member ADD CONSTRAINT
fk_staff_member_employment_details FOREIGN KEY (employmentdetails_id)
REFERENCES employment_details (id);
ALTER TABLE bank_account ADD CONSTRAINT fk_bank_account_staff_member
FOREIGN KEY (staff_id) REFERENCES staff_member (staff_id) ON DELETE
CASCADE;
ALTER TABLE home_address ADD CONSTRAINT fk_home_address_staff_member
FOREIGN KEY (staff id) REFERENCES staff member (staff id) ON DELETE

```

```

CASCADE;
ALTER TABLE task ADD CONSTRAINT fk_task_staff_member FOREIGN KEY
(staff_id) REFERENCES staff_member (staff_id) ON DELETE CASCADE;
ALTER TABLE users ADD CONSTRAINT fk_users_roles FOREIGN KEY (role_id)
REFERENCES roles (role_id);
ALTER TABLE users ADD CONSTRAINT fk_users_staff_member_1 FOREIGN KEY
(holidayapprover_staff_id) REFERENCES staff_member (staff_id) ON DELETE
SET DEFAULT;
ALTER TABLE users ADD CONSTRAINT fk_users_staff_member_2 FOREIGN KEY
(expensesapprover_staff_id) REFERENCES staff_member (staff_id) ON DELETE
SET DEFAULT;
ALTER TABLE employment_details ADD CONSTRAINT
fk_employment_details_staff_member FOREIGN KEY (linemanager_staff_id)
REFERENCES staff_member (staff_id) ON DELETE SET DEFAULT;
ALTER TABLE employment_details ADD CONSTRAINT
fk_employment_details_grade FOREIGN KEY (grade_id) REFERENCES grade
(id);
ALTER TABLE employment_details_workstream ADD CONSTRAINT
fk_employment_details_workstream_workstream FOREIGN KEY (workstreams_id)
REFERENCES workstream (id) ON DELETE CASCADE;
ALTER TABLE employment_details_workstream ADD CONSTRAINT
fk_employment_details_workstream_employment_details FOREIGN KEY
(employment_details_id) REFERENCES employment_details (id) ON DELETE
CASCADE;
ALTER TABLE project ADD CONSTRAINT fk_project_workstream FOREIGN KEY
(workstream_id) REFERENCES workstream (id) ON DELETE CASCADE;
ALTER TABLE role_update_staff_details ADD CONSTRAINT
fk_update_staff_details_role FOREIGN KEY (role_id) REFERENCES roles
(role_id) ON DELETE CASCADE;
ALTER TABLE role_view_staff_details ADD CONSTRAINT
fk_view_staff_details_role FOREIGN KEY (role_id) REFERENCES roles
(role_id) ON DELETE CASCADE;
ALTER TABLE role_allowed_expenses_statuses ADD CONSTRAINT
fk_expenses_statuses_role FOREIGN KEY (role_id) REFERENCES roles
(role_id) ON DELETE CASCADE;
ALTER TABLE holiday_year ADD CONSTRAINT fk_holiday_year_staff_member
FOREIGN KEY (staff_id) REFERENCES staff_member (staff_id) ON DELETE
CASCADE;
ALTER TABLE holiday ADD CONSTRAINT fk_holiday_holiday_year FOREIGN KEY
(holiday_year_id) REFERENCES holiday_year (id) ON DELETE CASCADE;
ALTER TABLE expenses ADD CONSTRAINT fk_expenses_staff_member FOREIGN KEY
(staff_id) REFERENCES staff_member (staff_id) ON DELETE CASCADE;
ALTER TABLE expenses_item ADD CONSTRAINT fk_expenses_item_expenses
FOREIGN KEY (expenses_id) REFERENCES expenses (id) ON DELETE CASCADE;
ALTER TABLE expenses_item ADD CONSTRAINT
fk_expenses_item_expenses_category FOREIGN KEY (expenses_category_id)
REFERENCES expenses_category (id);
ALTER TABLE expenses_item ADD CONSTRAINT fk_expenses_item_project
FOREIGN KEY (project_id) REFERENCES project (id);
ALTER TABLE expenses_mnemonic ADD CONSTRAINT
fk_expenses_mnemonic_expenses_category FOREIGN KEY
(expenses_category_id) REFERENCES expenses_category (id);
ALTER TABLE expenses_mnemonic ADD CONSTRAINT fk_expenses_mnemonic_users
FOREIGN KEY (username) REFERENCES users (username) ON DELETE CASCADE;
ALTER TABLE mileage_cost ADD CONSTRAINT
fk_mileage_cost_expenses_category FOREIGN KEY (expenses_category_id)
REFERENCES expenses_category (id) ON DELETE CASCADE;

```

```

-- Check constraints definitions
ALTER TABLE home_address ADD CONSTRAINT home_address_check CHECK
(house_name IS NOT NULL OR house_number IS NOT NULL);
ALTER TABLE expenses_mnemonic ADD CONSTRAINT expenses_mnemonic_check
CHECK (mileage IS NOT NULL OR amount IS NOT NULL);
ALTER TABLE expenses_item ADD CONSTRAINT expenses_item_check CHECK
(miles IS NOT NULL OR amount IS NOT NULL);

-- Triggers and functions definitions
CREATE FUNCTION mileage_cost_check() RETURNS trigger AS
$mileage_cost_check$
BEGIN
    IF (TG_OP = 'UPDATE') THEN
        IF EXISTS ( SELECT * FROM mileage_cost AS MC
                     WHERE (MC.lower_limit <= NEW.lower_limit) AND
                           (MC.upper_limit >= NEW.lower_limit) AND
                           (MC.id != NEW.id) AND
                           (MC.expenses_category_id =
NEW.expenses_category_id)
                     )
            THEN
                RAISE EXCEPTION '% lower_limit already exists',
NEW.lower_limit;
            END IF;

        IF EXISTS ( SELECT * FROM mileage_cost AS MC
                     WHERE (MC.lower_limit <= NEW.upper_limit) AND
                           (MC.upper_limit >= NEW.upper_limit) AND
                           (MC.id != NEW.id) AND
                           (MC.expenses_category_id =
NEW.expenses_category_id)
                     )
            THEN
                RAISE EXCEPTION '% upper_limit already exists',
NEW.upper_limit;
            END IF;
        ELSIF (TG_OP = 'INSERT') THEN
            IF EXISTS ( SELECT * FROM mileage_cost AS MC
                         WHERE (MC.lower_limit <= NEW.lower_limit) AND
                               (MC.upper_limit >= NEW.lower_limit) AND
                               (MC.expenses_category_id =
NEW.expenses_category_id)
                         )
                THEN
                    RAISE EXCEPTION '% lower_limit already exists',
NEW.lower_limit;
                END IF;

            IF EXISTS ( SELECT * FROM mileage_cost AS MC
                         WHERE (MC.lower_limit <= NEW.upper_limit) AND
                               (MC.upper_limit >= NEW.upper_limit) AND
                               (MC.expenses_category_id =
NEW.expenses_category_id)
                         )
                THEN
                    RAISE EXCEPTION '% upper limit already exists',
NEW.upper_limit;
                END IF;
        END IF;
    END IF;
END;

```

```

NEW.upper_limit;
    END IF;
    END IF;
    RETURN new;
END;

$mileage_cost_check$ LANGUAGE plpgsql;

CREATE FUNCTION users_check() RETURNS trigger AS $users_check$
BEGIN
    IF (TG_OP = 'UPDATE') THEN
        IF ((NEW.username IS NULL) AND EXISTS (SELECT * FROM users WHERE
(users.username = OLD.username)))
            THEN
                DELETE FROM users WHERE (users.username = OLD.username);
            END IF;
        END IF;
    RETURN new;
END;

$users_check$ LANGUAGE plpgsql;

CREATE TRIGGER mileage_cost_check
    BEFORE INSERT OR UPDATE ON mileage_cost
    FOR EACH ROW
    EXECUTE PROCEDURE mileage_cost_check();

CREATE TRIGGER users_check
    AFTER UPDATE ON staff_member
    FOR EACH ROW
    EXECUTE PROCEDURE users_check();

```

11.7 Appendix G – Sample ORM SQL queries

The queries below are used to load the object graph of a StaffMember instance

```
select staffmembe0_.staff_id as staff1_12_, staffmembe0_.dob as dob12_,
staffmembe0_.emailAddress as emailAdd3_12_,
staffmembe0_.employmentDetails_id as employm13_12_,
staffmembe0_.first_name as first4_12_, staffmembe0_.gender as gender12_,
staffmembe0_.homeTelNo as homeTelNo12_, staffmembe0_.last_name as
last7_12_, staffmembe0_.ni_number as ni8_12_, staffmembe0_.tax_code as
tax9_12_, staffmembe0_.title as title12_, staffmembe0_.username as
username12_, staffmembe0_.workTelNo as workTelNo12_,
staffmembe0_1_.account_number as account1_14_, staffmembe0_1_.bank_name
as bank2_14_, staffmembe0_1_.sort_code as sort3_14_,
staffmembe0_2_.address_line1 as address1_13_,
staffmembe0_2_.address_line2 as address2_13_, staffmembe0_2_.country as
country13_, staffmembe0_2_.county as county13_,
staffmembe0_2_.house_name as house5_13_, staffmembe0_2_.house_number as
house6_13_, staffmembe0_2_.locality as locality13_,
staffmembe0_2_.post_code as post8_13_, staffmembe0_2_.town as town13_
from staff_member staffmembe0_
left outer join bank_account staffmembe0_1_ on
    staffmembe0_.staff_id=staffmembe0_1_.staff_id
left outer join home_address staffmembe0_2_ on
    staffmembe0_.staff_id=staffmembe0_2_.staff_id
where staffmembe0_.username=?

select employment0_.id as id9_12_, employment0_.dateJoined as
dateJoined9_12_, employment0_.dateLeft as dateLeft9_12_,
employment0_.employmentType as employme4_9_12_, employment0_.grade_id as
grade7_9_12_, employment0_.holidayEntitlement as holidayE5_9_12_,
employment0_.lineManager_staff_id as lineMana8_9_12_,
employment0_.salary as salary9_12_, grade1_.id as id10_0_, grade1_.code
as code10_0_, grade1_.maximumSalary as maximumS3_10_0_,
grade1_.minimumSalary as minimumS4_10_0_, grade1_.name as name10_0_,
staffmembe2_.staff_id as staff1_12_1_, staffmembe2_.dob as dob12_1_,
staffmembe2_.emailAddress as emailAdd3_12_1_,
staffmembe2_.employmentDetails_id as employm13_12_1_,
staffmembe2_.first_name as first4_12_1_, staffmembe2_.gender as
gender12_1_, staffmembe2_.homeTelNo as homeTelNo12_1_,
staffmembe2_.last_name as last7_12_1_, staffmembe2_.ni_number as
ni8_12_1_, staffmembe2_.tax_code as tax9_12_1_, staffmembe2_.title as
title12_1_, staffmembe2_.username as username12_1_,
staffmembe2_.workTelNo as workTelNo12_1_, staffmembe2_1_.account_number
as account1_14_1_, staffmembe2_1_.bank_name as bank2_14_1_,
staffmembe2_1_.sort_code as sort3_14_1_, staffmembe2_2_.address_line1 as
address1_13_1_, staffmembe2_2_.address_line2 as address2_13_1_,
staffmembe2_2_.country as country13_1_, staffmembe2_2_.county as
county13_1_, staffmembe2_2_.house_name as house5_13_1_,
staffmembe2_2_.house_number as house6_13_1_, staffmembe2_2_.locality as
locality13_1_, staffmembe2_2_.post_code as post8_13_1_,
staffmembe2_2_.town as town13_1_, employment3_.id as id9_2_,
employment3_.dateJoined as dateJoined9_2_, employment3_.dateLeft as
dateLeft9_2_, employment3_.employmentType as employme4_9_2_,
employment3_.grade_id as grade7_9_2_, employment3_.holidayEntitlement as
```

```

holidayE5_9_2_, employment3_.lineManager_staff_id as lineMana8_9_2_,
employment3_.salary as salary9_2_, workstream4_.employment_details_id as
employment1_14_, workstream5_.id as workStre2_14_, workstream5_.id as
id1_3_, workstream5_.description as descript2_1_3_, workstream5_.name as
name1_3_, projects6_.workstream_id as workstream5_15_, projects6_.id as
id15_, projects6_.id as id0_4_, projects6_.code as code0_4_,
projects6_.description as descript3_0_4_, projects6_.name as name0_4_,
user7_.username as username15_5_, user7_.canApproveExpenses as
canAppro2_15_5_, user7_.canApproveHolidays as canAppro3_15_5_,
user7_.expensesApprover_staff_id as expense10_15_5_,
user7_.holidayApprover_staff_id as holidayA8_15_5_, user7_.locked as
locked15_5_, user7_.password as password15_5_, user7_.personalPhoto as
personal6_15_5_, user7_.role_id as role9_15_5_,
user7_.unSuccessfulLoginAttempts as unSucces7_15_5_,
staffmembe8_.staff_id as staff1_12_6_, staffmembe8_.dob as dob12_6_,
staffmembe8_.emailAddress as emailAdd3_12_6_,
staffmembe8_.employmentDetails_id as employm13_12_6_,
staffmembe8_.first_name as first4_12_6_, staffmembe8_.gender as
gender12_6_, staffmembe8_.homeTelNo as homeTelNo12_6_,
staffmembe8_.last_name as last7_12_6_, staffmembe8_.ni_number as
ni8_12_6_, staffmembe8_.tax_code as tax9_12_6_, staffmembe8_.title as
title12_6_, staffmembe8_.username as username12_6_,
staffmembe8_.workTelNo as workTelNo12_6_, staffmembe8_1_.account_number
as account1_14_6_, staffmembe8_1_.bank_name as bank2_14_6_,
staffmembe8_1_.sort_code as sort3_14_6_, staffmembe8_2_.address_line1 as
address1_13_6_, staffmembe8_2_.address_line2 as address2_13_6_,
staffmembe8_2_.country as country13_6_, staffmembe8_2_.county as
county13_6_, staffmembe8_2_.house_name as house5_13_6_,
staffmembe8_2_.house_number as house6_13_6_, staffmembe8_2_.locality as
locality13_6_, staffmembe8_2_.post_code as post8_13_6_,
staffmembe8_2_.town as town13_6_, expensesmn9_.username as username16_,
expensesmn9_.name as name16_, expensesmn9_.name as name5_7_,
expensesmn9_.amount as amount5_7_, expensesmn9_.expeneses_category_id as
expeneses4_5_7_, expensesmn9_.mileage as mileage5_7_, expensesca10_.id
as id3_8_, expensesca10_.category as category3_8_,
expensesca10_.hasMileage as hasMileage3_8_,
mileagecos11_.expenses_category_id as expenses5_17_, mileagecos11_.id as
id17_, mileagecos11_.id as id6_9_, mileagecos11_.cost as cost6_9_,
mileagecos11_.lower_limit as lower3_6_9_, mileagecos11_.upper_limit as
upper4_6_9_, staffmembel2_.staff_id as staff1_12_10_, staffmembel2_.dob
as dob12_10_, staffmembel2_.emailAddress as emailAdd3_12_10_,
staffmembel2_.employmentDetails_id as employm13_12_10_,
staffmembel2_.first_name as first4_12_10_, staffmembel2_.gender as
gender12_10_, staffmembel2_.homeTelNo as homeTelNo12_10_,
staffmembel2_.last_name as last7_12_10_, staffmembel2_.ni_number as
ni8_12_10_, staffmembel2_.tax_code as tax9_12_10_, staffmembel2_.title
as title12_10_, staffmembel2_.username as username12_10_,
staffmembel2_.workTelNo as workTelNo12_10_,
staffmembel2_1_.account_number as account1_14_10_,
staffmembel2_1_.bank_name as bank2_14_10_, staffmembel2_1_.sort_code as
sort3_14_10_, staffmembel2_2_.address_line1 as address1_13_10_,
staffmembel2_2_.address_line2 as address2_13_10_,
staffmembel2_2_.country as country13_10_, staffmembel2_2_.county as
county13_10_, staffmembel2_2_.house_name as house5_13_10_,
staffmembel2_2_.house_number as house6_13_10_, staffmembel2_2_.locality
as locality13_10_, staffmembel2_2_.post_code as post8_13_10_,
staffmembel2_2_.town as town13_10_, genericroll3_.role_id as

```



```

role2_11_11_, genericrol13_.role_type as role1_11_11_,
expensesst14_.role_id as role1_18_, expensesst14_.status_name as
status2_18_, updateable15_.role_id as role1_19_,
updateable15_.field_name as field2_19_, viewablest16_.role_id as
role1_20_, viewablest16_.field_name as field2_20_
    from employment_details employment0_
    inner join Grade grade1_ on employment0_.grade_id=grade1_.id
    left outer join staff_member staffmembe2_ on
employment0_.lineManager_staff_id=staffmembe2_.staff_id
    left outer join bank_account staffmembe2_1_ on
staffmembe2_.staff_id=staffmembe2_1_.staff_id
    left outer join home_address staffmembe2_2_ on
staffmembe2_.staff_id=staffmembe2_2_.staff_id
    left outer join employment_details employment3_ on
staffmembe2_.employmentDetails_id=employment3_.id
    left outer join employment_details_WorkStream workstream4_ on
employment3_.id=workstream4_.employment_details_id
    left outer join WorkStream workstream5_ on
workstream4_.workStreams_id=workstream5_.id
    left outer join Project projects6_ on
workstream5_.id=projects6_.workstream_id
    left outer join users user7_ on
staffmembe2_.username=user7_.username
    left outer join staff_member staffmembe8_ on
user7_.expensesApprover_staff_id=staffmembe8_.staff_id
    left outer join bank_account staffmembe8_1_ on
staffmembe8_.staff_id=staffmembe8_1_.staff_id
    left outer join home_address staffmembe8_2_ on
staffmembe8_.staff_id=staffmembe8_2_.staff_id
    left outer join expenses_mnemonic expensesmn9_ on
user7_.username=expensesmn9_.username
    left outer join expenses_category expensesca10_ on
expensesmn9_.expenses_category_id=expensesca10_.id
    left outer join mileage_cost mileagecos11_ on
expensesca10_.id=mileagecos11_.expenses_category_id
    left outer join staff_member staffmembel2_ on
user7_.holidayApprover_staff_id=staffmembel2_.staff_id
    left outer join bank_account staffmembel2_1_ on
staffmembel2_.staff_id=staffmembel2_1_.staff_id
    left outer join home_address staffmembel2_2_ on
staffmembel2_.staff_id=staffmembel2_2_.staff_id
    left outer join Roles genericrol13_ on
user7_.role_id=genericrol13_.role_id
    left outer join Role_Allowed_expenses_statuses expensesst14_ on
genericrol13_.role_id=expensesst14_.role_id
    left outer join Role_Update_Staff_Details updateable15_ on
genericrol13_.role_id=updateable15_.role_id
    left outer join Role_View_Staff_Details viewablest16_ on
genericrol13_.role_id=viewablest16_.role_id
    where employment0_.id=?

select user0_.username as username15_12_, user0_.canApproveExpenses as
canAppro2_15_12_, user0_.canApproveHolidays as canAppro3_15_12_,
user0_.expensesApprover_staff_id as expense10_15_12_,
user0_.holidayApprover_staff_id as holidayA8_15_12_, user0_.locked as
locked15_12_, user0_.password as password15_12_, user0_.personalPhoto as
personal6_15_12_ , user0_.role_id as role9_15_12_ ,

```

```

user0_.unSuccessfulLoginAttempts as unSucces7_15_12_,
staffmembre1_.staff_id as staff1_12_0_, staffmembre1_.dob as dob12_0_,
staffmembre1_.emailAddress as emailAdd3_12_0_,
staffmembre1_.employmentDetails_id as employm13_12_0_,
staffmembre1_.first_name as first4_12_0_, staffmembre1_.gender as
gender12_0_, staffmembre1_.homeTelNo as homeTelNo12_0_,
staffmembre1_.last_name as last7_12_0_, staffmembre1_.ni_number as
ni8_12_0_, staffmembre1_.tax_code as tax9_12_0_, staffmembre1_.title as
title12_0_, staffmembre1_.username as username12_0_,
staffmembre1_.workTelNo as workTelNo12_0_, staffmembre1_.account_number
as account1_14_0_, staffmembre1_.bank_name as bank2_14_0_,
staffmembre1_.sort_code as sort3_14_0_, staffmembre1_.address_line1 as
address1_13_0_, staffmembre1_.address_line2 as address2_13_0_,
staffmembre1_.country as country13_0_, staffmembre1_.county as
county13_0_, staffmembre1_.house_name as house5_13_0_,
staffmembre1_.house_number as house6_13_0_, staffmembre1_.locality as
locality13_0_, staffmembre1_.post_code as post8_13_0_,
staffmembre1_.town as town13_0_, employment2_.id as id9_1_,
employment2_.dateJoined as dateJoined9_1_, employment2_.dateLeft as
dateLeft9_1_, employment2_.employmentType as employe4_9_1_,
employment2_.grade_id as grade7_9_1_, employment2_.holidayEntitlement as
holidayE5_9_1_, employment2_.lineManager_staff_id as lineMana8_9_1_,
employment2_.salary as salary9_1_, grade3_.id as id10_2_, grade3_.code
as code10_2_, grade3_.maximumSalary as maximumS3_10_2_,
grade3_.minimumSalary as minimumS4_10_2_, grade3_.name as name10_2_,
staffmembre4_.staff_id as staff1_12_3_, staffmembre4_.dob as dob12_3_,
staffmembre4_.emailAddress as emailAdd3_12_3_,
staffmembre4_.employmentDetails_id as employm13_12_3_,
staffmembre4_.first_name as first4_12_3_, staffmembre4_.gender as
gender12_3_, staffmembre4_.homeTelNo as homeTelNo12_3_,
staffmembre4_.last_name as last7_12_3_, staffmembre4_.ni_number as
ni8_12_3_, staffmembre4_.tax_code as tax9_12_3_, staffmembre4_.title as
title12_3_, staffmembre4_.username as username12_3_,
staffmembre4_.workTelNo as workTelNo12_3_, staffmembre4_.account_number
as account1_14_3_, staffmembre4_.bank_name as bank2_14_3_,
staffmembre4_.sort_code as sort3_14_3_, staffmembre4_.address_line1 as
address1_13_3_, staffmembre4_.address_line2 as address2_13_3_,
staffmembre4_.country as country13_3_, staffmembre4_.county as
county13_3_, staffmembre4_.house_name as house5_13_3_,
staffmembre4_.house_number as house6_13_3_, staffmembre4_.locality as
locality13_3_, staffmembre4_.post_code as post8_13_3_,
staffmembre4_.town as town13_3_, user5_.username as username15_4_,
user5_.canApproveExpenses as canAppro2_15_4_, user5_.canApproveHolidays
as canAppro3_15_4_, user5_.expensesApprover_staff_id as expense10_15_4_,
user5_.holidayApprover_staff_id as holidayA8_15_4_, user5_.locked as
locked15_4_, user5_.password as password15_4_, user5_.personalPhoto as
personal6_15_4_, user5_.role_id as role9_15_4_,
user5_.unSuccessfulLoginAttempts as unSucces7_15_4_,
expensesmn6_.username as username14_, expensesmn6_.name as name14_,
expensesmn6_.name as name5_5_, expensesmn6_.amount as amount5_5_,
expensesmn6_.expenses_category_id as expenses4_5_5_,
expensesmn6_.mileage as mileage5_5_, expensesca7_.id as id3_6_,
expensesca7_.category as category3_6_, expensesca7_.hasMileage as
hasMileage3_6_, mileagecos8_.expenses_category_id as expenses5_15_,
mileagecos8_.id as id15_, mileagecos8_.id as id6_7_, mileagecos8_.cost
as cost6_7_, mileagecos8_.lower_limit as lower3_6_7_,
mileagecos8_.upper limit as upper4_6_7_, staffmembre9_.staff_id as

```

```

staff1_12_8_, staffmembe9_.dob as dob12_8_, staffmembe9_.emailAddress as
emailAdd3_12_8_, staffmembe9_.employmentDetails_id as employm13_12_8_,
staffmembe9_.first_name as first4_12_8_, staffmembe9_.gender as
gender12_8_, staffmembe9_.homeTelNo as homeTelNo12_8_,
staffmembe9_.last_name as last7_12_8_, staffmembe9_.ni_number as
ni8_12_8_, staffmembe9_.tax_code as tax9_12_8_, staffmembe9_.title as
title12_8_, staffmembe9_.username as username12_8_,
staffmembe9_.workTelNo as workTelNo12_8_, staffmembe9_1_.account_number
as account1_14_8_, staffmembe9_1_.bank_name as bank2_14_8_,
staffmembe9_1_.sort_code as sort3_14_8_, staffmembe9_2_.address_line1 as
address1_13_8_, staffmembe9_2_.address_line2 as address2_13_8_,
staffmembe9_2_.country as country13_8_, staffmembe9_2_.county as
county13_8_, staffmembe9_2_.house_name as house5_13_8_,
staffmembe9_2_.house_number as house6_13_8_, staffmembe9_2_.locality as
locality13_8_, staffmembe9_2_.post_code as post8_13_8_,
staffmembe9_2_.town as town13_8_, genericroll10_.role_id as role2_11_9_,
genericroll10_.role_type as role1_11_9_, expensesst11_.role_id as
role1_16_, expensesst11_.status_name as status2_16_,
updateable12_.role_id as role1_17_, updateable12_.field_name as
field2_17_, viewablest13_.role_id as role1_18_, viewablest13_.field_name
as field2_18_, workstream14_.employment_details_id as employment1_19_,
workstream15_.id as workStre2_19_, workstream15_.id as id1_10_,
workstream15_.description as descript2_1_10_, workstream15_.name as
name1_10_, projects16_.workstream_id as workstream5_20_, projects16_.id
as id20_, projects16_.id as id0_11_, projects16_.code as code0_11_,
projects16_.description as descript3_0_11_, projects16_.name as
name0_11_
    from users user0_
    left outer join staff_member staffmembel_ on
user0_.expensesApprover_staff_id=staffmembel_.staff_id
    left outer join bank_account staffmembel_1_ on
staffmembel_.staff_id=staffmembel_1_.staff_id
    left outer join home_address staffmembel_2_ on
staffmembel_.staff_id=staffmembel_2_.staff_id
    left outer join employment_details employment2_ on
staffmembel_.employmentDetails_id=employment2_.id
    left outer join Grade grade3_ on employment2_.grade_id=grade3_.id
    left outer join staff_member staffmembe4_ on
employment2_.lineManager_staff_id=staffmembe4_.staff_id
    left outer join bank_account staffmembe4_1_ on
staffmembe4_.staff_id=staffmembe4_1_.staff_id
    left outer join home_address staffmembe4_2_ on
staffmembe4_.staff_id=staffmembe4_2_.staff_id
    left outer join users user5_ on
staffmembe4_.username=user5_.username
    left outer join expenses_mnemonic expensesmn6_ on
user5_.username=expensesmn6_.username
    left outer join expenses_category expensesca7_ on
expensesmn6_.expeneses_category_id=expensesca7_.id
    left outer join mileage_cost mileagecos8_ on
expensesca7_.id=mileagecos8_.expenses_category_id
    left outer join staff_member staffmembe9_ on
user5_.holidayApprover_staff_id=staffmembe9_.staff_id
    left outer join bank_account staffmembe9_1_ on
staffmembe9_.staff_id=staffmembe9_1_.staff_id
    left outer join home_address staffmembe9_2_ on
staffmembe9_.staff_id=staffmembe9_2_.staff_id

```

```

    left outer join Roles genericroll10_ on
user5_.role_id=genericroll10_.role_id
    left outer join Role_Allowed_expenses_statuses expensesst11_ on
genericroll10_.role_id=expensesst11_.role_id
    left outer join Role_Update_Staff_Details updateable12_ on
genericroll10_.role_id=updateable12_.role_id
    left outer join Role_View_Staff_Details viewablest13_ on
genericroll10_.role_id=viewablest13_.role_id
    left outer join employment_details_WorkStream workstream14_ on
employment2_.id=workstream14_.employment_details_id
    left outer join WorkStream workstream15_ on
workstream14_.workStreams_id=workstream15_.id
    left outer join Project projects16_ on
workstream15_.id=projects16_.workstream_id
    where user0_.username=?

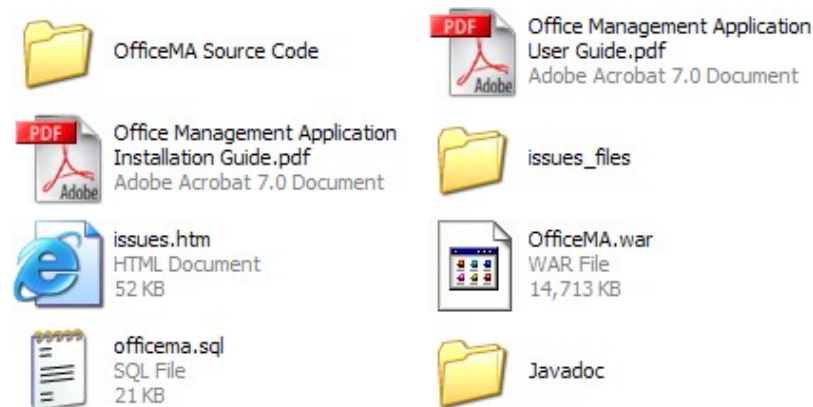
select expensesmn0_.username as username2_, expensesmn0_.name as name2_,
expensesmn0_.name as name5_1_, expensesmn0_.amount as amount5_1_,
expensesmn0_.expeneses_category_id as expeneses4_5_1_,
expensesmn0_.mileage as mileage5_1_, expensesca1_.id as id3_0_,
expensesca1_.category as category3_0_, expensesca1_.hasMileage as
hasMileage3_0_
    from expenses_mnemonic expensesmn0_
    inner join expenses_category expensesca1_ on
expensesmn0_.expeneses_category_id=expensesca1_.id
    where expensesmn0_.username=?

select workstream0_.employment_details_id as employment1_1_,
workstream0_.workStreams_id as workStre2_1_, workstream1_.id as id1_0_,
workstream1_.description as descript2_1_0_, workstream1_.name as
name1_0_
    from employment_details_WorkStream workstream0_
    left outer join WorkStream workstream1_ on
workstream0_.workStreams_id=workstream1_.id
    where workstream0_.employment_details_id=?

```

11.8 Appendix H – Software CD-ROM Contents

The deliverables of this project are included in the CD-ROM attached to this dissertation due to the large size of the source code and supporting documentations such installation and user guides and the Javadoc for the Java source code. Below is a listing of what is included in the CD:



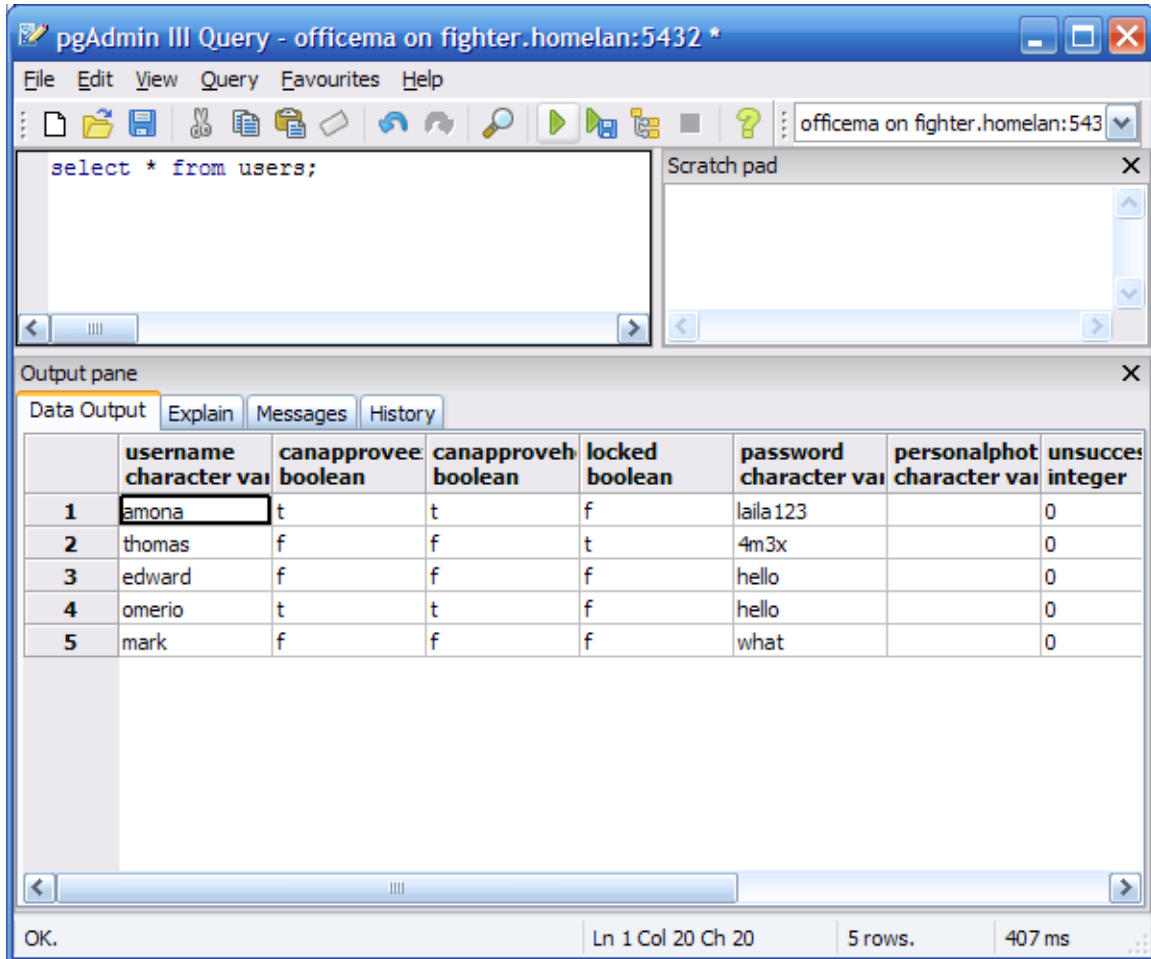
- **OfficeMA Source Code** – Contains the source code for the OfficeMA developed by this project
- **Office Management Application User Guide.pdf** – The user guide for the application.
- **Office Management Application Installation Guide.pdf** – The installation guide for the application.
- **issues.html** – Contains a list of issues currently open in this version of the application.
- **officema.sql** – The PostgreSQL database SQL scripts required to create the OfficeMA database and the default administrator user.
- **OfficeMA.war** – Is the Java Web Archive for the OfficeMA Web application. This will need to be deployed to a Web container such as Tomcat
- **Javadoc** – The Javadoc documentations for the Java source code developed for the OfficeMA.

11.9 Appendix I - Software used for the project

- Eclipse IDE version 3.3 (<http://www.eclipse.org>) – An open source enhanced integrated development environment with Java support.
- Jude Community version 5.0.2 (<https://jude.change-vision.com/>) – Free community UML Case tool.
- PostgreSQL 8.2. DBMS server (<http://www.postgresql.org>) – An open source RDBMS that fully ACID compliant and includes most of SQL92 and SQL99 data types
- Apache Tomcat 6.0.13 (<http://tomcat.apache.org/>) – An open source Servlet and JSP container.
- Dojo Toolkit 0.43 (<http://dojotoolkit.org/>) – Dojo is an Open Source DHTML toolkit written in JavaScript.
- OpenLaszlo (<http://www.openlaszlo.org/>) – OpenLaszlo is an open source platform for creating zero-install web applications with the user interface capabilities of desktop client software.
- Hibernate ORM framework 3.2 (<http://www.hibernate.org/>) – Hibernate is a popular open source ORM framework. Version 3.2 implements the Java Persistence API (JPA)
- Spring 2.0 framework (<http://www.springframework.org/>) – Spring is an open source framework and container that can be used in a domain model to add transactional and security support. Can also supply object dependencies at runtime using dependency injection.
- Struts 2.0.11 framework (<http://struts.apache.org/2.x/index.html>)
- JUnit (<http://www.junit.org/>)

11.10 Appendix J – PostgreSQL database utilities

The PostgreSQL distribution used in this project was version 8.2, included with this download is the pgAdmin III tool shown below. The tool can be used to execute SQL statements and scripts against the database.



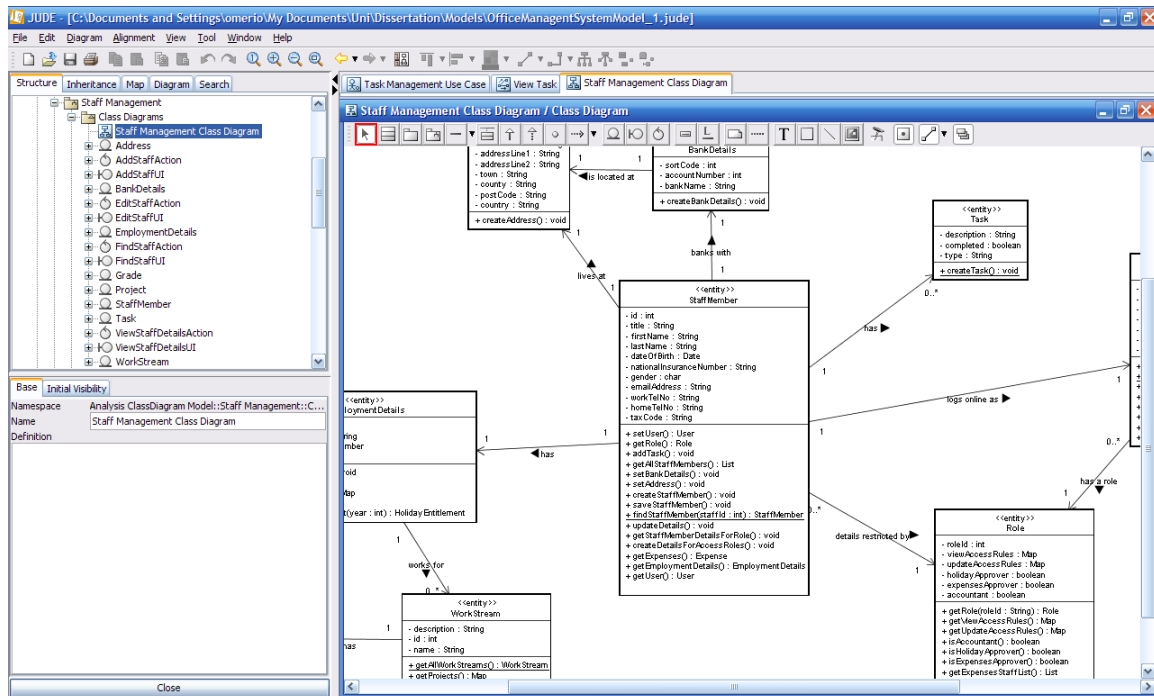
The screenshot shows the pgAdmin III Query tool window. The title bar reads "pgAdmin III Query - officema on fighter.homelan:5432 *". The menu bar includes File, Edit, View, Query, Favourites, and Help. The toolbar contains various icons for file operations, query execution, and help. The query editor contains the SQL statement: `select * from users;`. The output pane shows the results of the query in a table format. The table has 8 columns: username, canapprove, canapproveh, locked, password, personalphot, and unsucces. The data is as follows:

	username character vai	canapprove boolean	canapproveh boolean	locked boolean	password character vai	personalphot character vai	unsucces integer
1	amona	t	t	f	laila123		0
2	thomas	f	f	t	4m3x		0
3	edward	f	f	f	hello		0
4	omerio	t	t	f	hello		0
5	mark	f	f	f	what		0

The status bar at the bottom shows "OK.", "Ln 1 Col 20 Ch 20", "5 rows.", and "407 ms".

11.11 Appendix K – Jude UML CASE tool

Jude Community is a free community tool available from (<https://jude.change-vision.com/>). The version used in the project was version 5.02 which supports UML 1.2. All of the UML models in this project were developed using Jude. The tool also supports exporting UML class diagrams to Java classes skeleton and importing Java code into class diagrams.



11.12 Appendix L – Google code project

officema - Google Code - Microsoft Internet Explorer

Address: <http://code.google.com/p/officema/>

omer.dawelbeit@gmail.com | [Settings](#) | [What's new?](#) | [Help](#) | [My Account](#) | [Sign out](#)

Google Code **officema**
Office Management Application - A development methodology for Rich Internet Applications

[Project Home](#) | [Downloads](#) | [Wiki](#) | [Issues](#) | [Source](#) | [Administer](#)

Office Management Application (OfficeMA) is a Rich Internet Application (RIA) that harnesses the power of Web 2.0 to deliver flexibility, interactivity and enhanced user interface. The work on this project has started as part of my MSc Dissertation with Reading University.

The application is built using Java and the following components: Dojo Toolkit, Hibernate 3.2, Struts 2, Spring 2.0, JSON tags, PostgreSQL database and Tomcat 6 application server.

License: [Apache License 2.0](#)

Labels: [Ajax](#), [Java](#), [Model](#), [Web2](#), [Hibernate](#), [Spring](#), [Struts2](#), [JSON](#), [JavaScript](#), [Dojo](#), [Domain](#), [JPA](#)

Project owners:
[omer.dawelbeit](#)

officema - Google Code - Microsoft Internet Explorer

Address: <http://code.google.com/p/officema/source/browse>

omer.dawelbeit@gmail.com | [Settings](#) | [What's new?](#) | [Help](#) | [My Account](#) | [Sign out](#)

Google Code **officema**
A development methodology for Rich Internet Applications

[Project Home](#) | [Downloads](#) | [Wiki](#) | [Issues](#) | [Source](#) | [Administer](#)

[Checkout](#) | [Browse](#) | [Changes](#) | [Search Trunk](#)

Source Path: [svn/](#) [r14](#) [r15](#)

Filename	Size	Rev	Date	Author
Address.java	3.8 KB	r15	Feb 09 (2 days ago)	omer.dawelbeit
BankAccount.java	2.5 KB	r15	Feb 09 (2 days ago)	omer.dawelbeit
EmploymentDetails.java	5.9 KB	r15	Feb 09 (2 days ago)	omer.dawelbeit
StaffMember.java	17.6 KB	r15	Feb 09 (2 days ago)	omer.dawelbeit
User.java	7.4 KB	r15	Feb 09 (2 days ago)	omer.dawelbeit

This project is currently using 56.1 KB (0.1%) of its 100 MB repository quota.

Bugs raised in the project issues page

The screenshot shows a Mozilla Firefox browser window displaying the Google Code project page for 'officema'. The address bar shows the URL 'http://code.google.com/p/officema/issues/list'. The page header includes the Google Code logo and the project name 'officema' with the description 'Office Management Application - A development methodology for Rich Internet Applications'. The user 'omer.dawelbeit@gmail.com' is logged in, and there are links for 'Settings', 'What's new?', 'Help', 'My Account', and 'Sign out'. The page has tabs for 'Project Home', 'Downloads', 'Wiki', 'Issues', 'Source', and 'Administer'. The 'Issues' tab is active, showing a list of 4 issues. The issues are all 'Defect' type, 'Accepted' status, and 'Medium' priority. The summary of the issues is as follows:

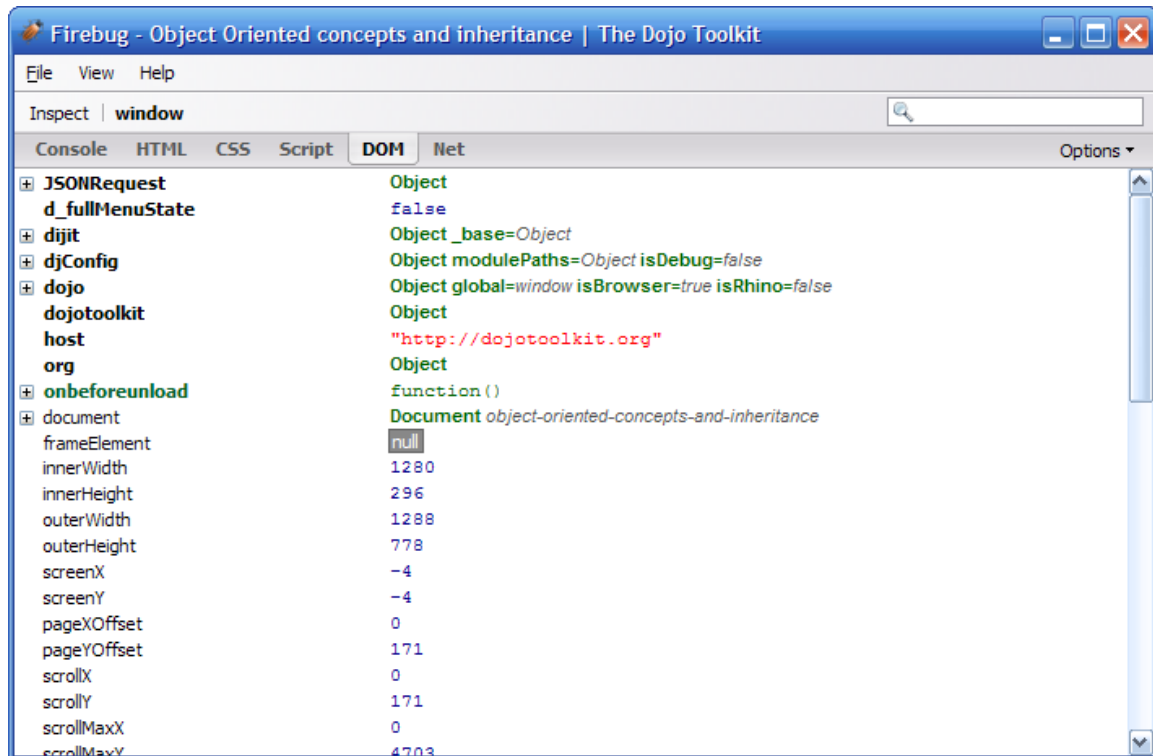
ID	Type	Status	Priority	Milestone	Owner	Summary + Labels
1	Defect	Accepted	Medium	---	omer.dawelbeit	Changing the password for locked account by Admin doesn't unlock the account
2	Defect	Accepted	Medium	---	omer.dawelbeit	DBCP is causing java.sql.SQLException when the connection is already been reset
3	Defect	Accepted	Medium	---	omer.dawelbeit	Accountant sees update button and the address page is editable
4	Defect	Accepted	Medium	---	omer.dawelbeit	Clicking the My Details task bar button results in an error dialogue

At the bottom of the page, there is a copyright notice: '©2008 Google - Code Home - Terms of Service - Privacy Policy - Site Directory'. The browser status bar at the bottom shows 'Done' and 'Open Notebook'.

11.13 Appendix N – Debugging JavaScript and Browser tools

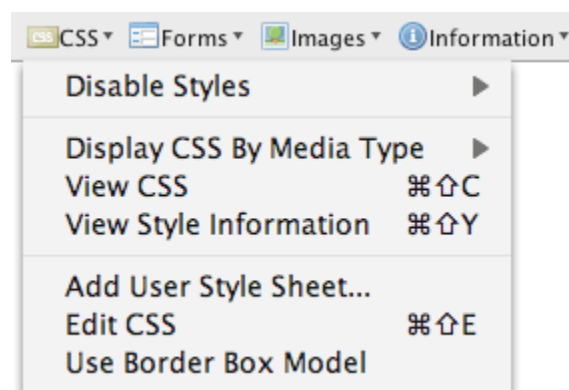
11.13.1 Firefox Firebug

Firebug is a plugin for the Firefox browser and it offers a great number of features such as DOM inspection, performance monitoring, JavaScript debugging, etc... to mention only a few.



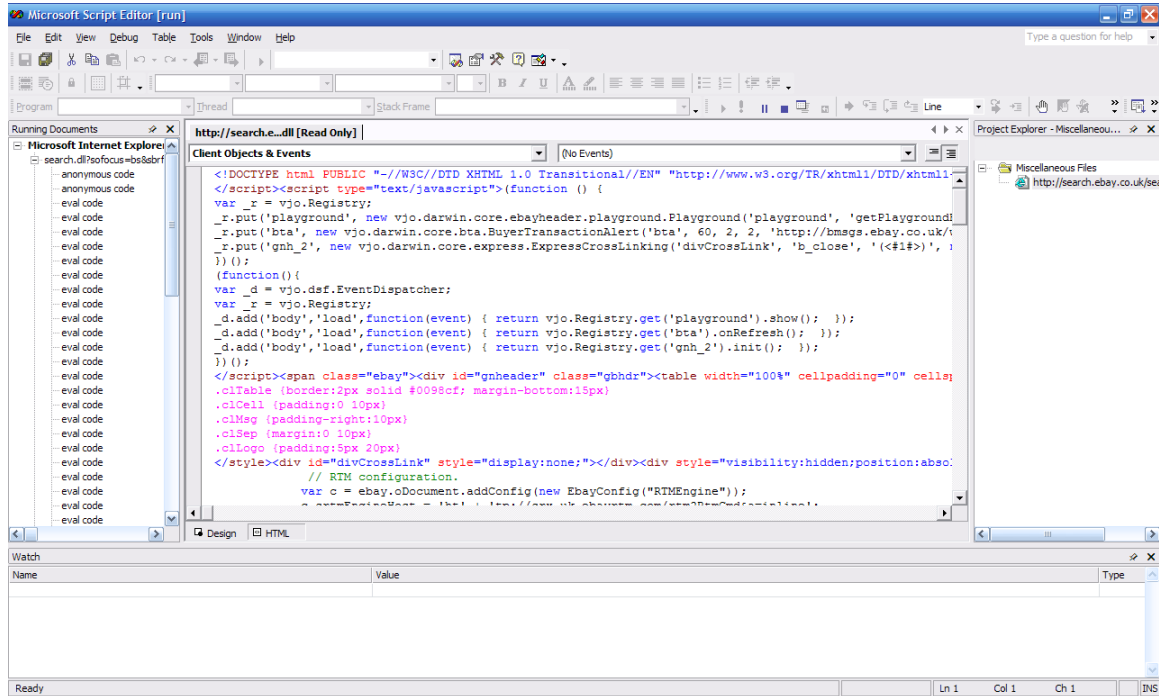
11.13.2 Firefox Web Developer Toolbar

Offers many essential features for web developers
(<http://chrispederick.com/work/web-developer/>)

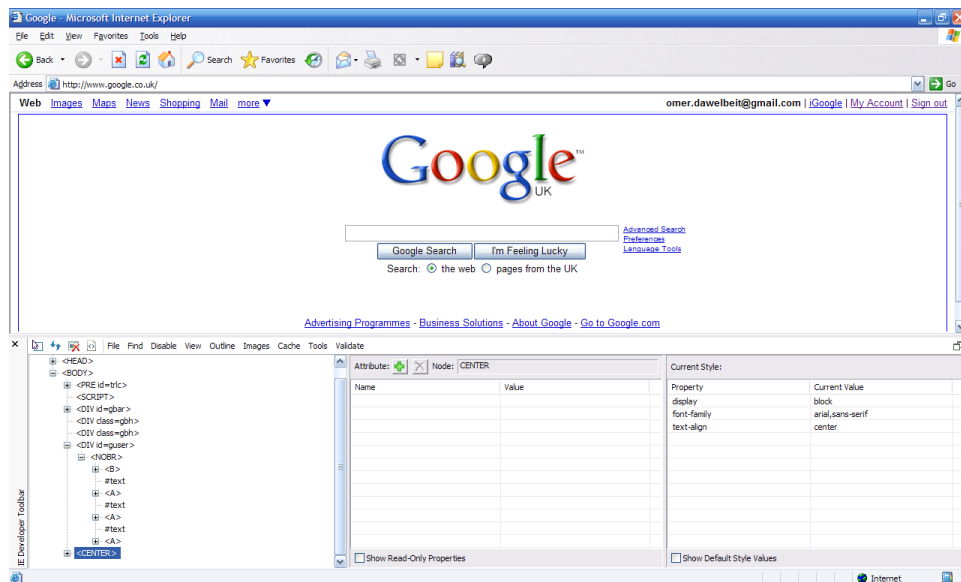


11.13.3 Microsoft Script Debugger for IE

For debugging in Internet Explorer, Microsoft script debugger can be used. Although it does not offer as many features as Firebug, it can be used to find JavaScript bugs.



11.13.4 Microsoft IE Developer Toolbar



11.14 Appendix O - Project Schedule

Table 22 – Project schedule and deadlines

ID	Name	Duration	Start	Finish
1	Requirement Gathering		01/10/2007	19/10/2007
2	Preliminary Report		01/10/2007	01/10/2007
3	Requirement Elicitation		01/10/2007	03/10/2007
4	Use Case Modelling		04/10/2007	17/11/2007
5	Prototype		11/10/2007	01/11/2007
6	Initial system architecture		19/10/2007	19/10/2007
7	Requirement document + prototype		17/11/2007	17/11/2007
8	Requirement Analysis		17/11/2007	07/12/2007
9	Use case realisation		17/11/2007	07/12/2007
10	Domain Analysis		17/11/2007	07/12/2007
11	Analysis class diagrams and communication diagrams		10/12/2007	10/12/2007
12	System Design		29/10/2007	02/11/2007
13	Deployment modelling		29/10/2007	02/11/2007
14	Component modelling		29/10/2007	02/11/2007
15	Architectural modelling		29/10/2007	02/11/2007
16	Overview design and implementation architecture		02/11/2007	02/11/2007
17	Detailed Design		10/12/2007	17/12/2007
18	Detailed class diagram		10/12/2007	17/12/2007
19	State and Interaction diagrams		10/12/2007	17/12/2007
20	Design models		17/12/2007	17/12/2007
21	User Interface design		17/12/2007	24/12/2007
22	User interface modelling and design		17/12/2007	17/12/2007
23	Design models with interface specification		17/12/2007	17/12/2007
24	Database Design		17/12/2007	24/12/2007
25	Data requirements			
26	Conceptual data model			
27	Logical schema			
28	Conceptual data models and SQL		17/12/2007	24/12/2007
29	Interim Report		03/12/2007	03/12/2007
30	Construction, testing and implementation		24/12/2007	21/03/2008
31	Write Java code			
32	Implement the user interface			
33	Database implementation			
34	System documentation			
35	Application source code		21/01/2008	21/03/2008

11.14.1 Project Gantt Chart

