

# Design Document

## WebAgenda

**Project Team:** Daniel Kettle, Daniel Wehr, Mark Hazlett, Noorin Hasan

**Client:** Deerfoot Inn and Casino, Robin Featherstone

**Instructor:** Randy Kaltenbach

February 3, 2010

# Contents

<b>I Preface</b>	<b>12</b>
<b>1 Document Purpose</b>	<b>13</b>
1.1 Introduction . . . . .	13
<b>2 Document Standards</b>	<b>15</b>
2.1 Document Standards List . . . . .	15
<b>II User Requirements</b>	<b>17</b>
<b>3 Business Overviews and Objectives</b>	<b>18</b>
3.1 Statement of the Problem . . . . .	19
3.2 System Environment . . . . .	20
3.3 Current System . . . . .	20
<b>4 System Requirements</b>	<b>21</b>
4.1 Functional Requirements . . . . .	21
4.2 Non-Functional Requirements . . . . .	26
4.3 System Interface Requirements . . . . .	29
4.4 Maintainability and Administration Requirements . . . . .	31
4.5 Usability Requirements . . . . .	32

<b>III Requirements Models</b>	<b>33</b>
<b>5 Use Case Diagram</b>	<b>34</b>
<b>6 Extended Use Case Diagrams</b>	<b>36</b>
6.1 Log-In System . . . . .	36
6.2 Access Schedule . . . . .	38
6.3 Review Employee Request . . . . .	39
6.4 Add Supervisor . . . . .	42
6.5 Booking Emergency Days Off . . . . .	44
6.6 Update User Availability . . . . .	46
6.7 Manager Viewing All Schedules . . . . .	49
6.8 Book Days Off . . . . .	52
6.9 Request Shift Change . . . . .	55
6.10 Distribute Reports . . . . .	60
6.11 Generate Reports . . . . .	62
6.12 Maintain Workgroups . . . . .	64
6.13 Send Announcement . . . . .	67
6.14 View Workgroup Schedule . . . . .	69
6.15 Maintain Locations . . . . .	71
6.16 Maintain Positions . . . . .	74
6.17 Maintain Skills . . . . .	77
<b>7 Problem Domain Class Diagram</b>	<b>80</b>
7.1 Problem Domain . . . . .	80
<b>8 State Machine Diagrams</b>	<b>82</b>

8.1 State Machines . . . . .	82
<b>9 Activity Diagrams</b>	<b>92</b>
9.1 Activity Diagrams . . . . .	93
<b>IV System Design</b>	<b>97</b>
<b>10 Layered Architecture</b>	<b>98</b>
10.1 Package Diagram . . . . .	98
<b>11 Persistence Model</b>	<b>100</b>
11.1 Textual Description Document . . . . .	100
11.2 Entity Relationship Diagrams . . . . .	102
<b>12 Expected Persistent Class Sizes</b>	<b>105</b>
<b>13 Class Diagram</b>	<b>113</b>
13.1 Interface, Persistence, Business and Problem Domain . . . . .	113
<b>14 Interaction Sequence Diagrams</b>	<b>117</b>
14.1 New Schedule Sequence Diagram . . . . .	118
14.2 Request Shift Change Sequence Diagram . . . . .	119
14.3 Viewing Schedule Sequence Diagram . . . . .	120
14.4 Book Days Off Sequence Diagram . . . . .	121
14.5 Administrative Schedule Viewing Sequence Diagram . . . . .	122
14.6 Request Shift Change Sequence Diagram . . . . .	123
14.7 Update Availability Sequence Diagram . . . . .	124
14.8 Access Schedule Sequence Diagram . . . . .	125

14.9 Add Supervisor Sequence Diagram . . . . .	126
14.10 Create Employee Type Sequence Diagram . . . . .	127
14.11 Create New Employee Sequence Diagram . . . . .	128
14.12 Distribute Report Schedule Sequence Diagram . . . . .	129
14.13 Generate Report Schedule Sequence Diagram . . . . .	130
14.14 System Log In Sequence Diagram . . . . .	131
14.15 Maintain Workgroup Sequence Diagram . . . . .	132
14.16 Maintain Position Sequence Diagram . . . . .	133
14.17 Maintain Skill Sequence Diagram . . . . .	134
14.18 Review Request Sequence Diagram . . . . .	135
14.19 Send Event Sequence Diagram . . . . .	136
14.20 Book Days Off Sequence Diagram . . . . .	137
14.21 Manager View Schedules Sequence Diagram . . . . .	138
14.22 Request Shift Change Sequence Diagram . . . . .	139
14.23 Update Availability Sequence Diagram . . . . .	140
14.24 Send Emergency Notification Sequence Diagram . . . . .	141
14.25 View Workgroup Schedule Sequence Diagram . . . . .	142
<b>15 Data Dictionary</b> . . . . .	<b>143</b>
15.1 Interface . . . . .	144
15.2 Persistence . . . . .	156
15.3 Application Domain . . . . .	159
15.4 Business Domain . . . . .	164

<b>V Project Management</b>	<b>176</b>
<b>16 Schedule</b>	<b>177</b>
16.1 Gantt Chart . . . . .	177
16.2 2009-10-08 Requirements Documentation . . . . .	179
16.3 2009-11-05 Requirements Analysis . . . . .	180
16.4 2009-12-17 Design Documentation . . . . .	180
16.5 2009-01-30 Alpha Release . . . . .	181
16.6 2009-02-28 Beta Release . . . . .	181
16.7 2009-04-15 Final Release . . . . .	181
<b>17 Team Configuration</b>	<b>182</b>
17.1 Members And Roles . . . . .	182
17.2 Reporting Relationships and Procedures . . . . .	183
17.3 Contact Information . . . . .	184
<b>18 Project Standards And Procedures</b>	<b>185</b>
18.1 Hardware Architecture . . . . .	187
18.1.1 Hardware Platform for Production System . . . . .	187
18.2 Hardware Platform for Development System . . . . .	188
18.3 Software Platform . . . . .	189
18.3.1 Server Software . . . . .	189
18.3.2 Client Software . . . . .	189
18.4 Interaction Model . . . . .	191
18.4.1 Style . . . . .	191
18.4.2 General Style Guidelines . . . . .	191

18.4.3 System Feedback Style . . . . .	192
18.4.4 Standards . . . . .	193
18.5 Desired User Support . . . . .	194
18.6 Screen Descriptions . . . . .	195
18.6.1 Widgets . . . . .	195
18.6.2 Screens . . . . .	196
18.7 Manual Procedures . . . . .	198
18.7.1 Security . . . . .	198
18.7.2 Operation . . . . .	198
18.7.3 Backup and Restore . . . . .	198
18.7.4 Data Archival . . . . .	199
18.8 Developer Contact . . . . .	199
<b>VI Appendix</b>	<b>200</b>
<b>A Interview Questions</b>	<b>201</b>
<b>B API Documentation</b>	<b>204</b>
B.1 API Document Template . . . . .	205
B.2 API Web Template . . . . .	206
<b>C Prototypes - Second Edition</b>	<b>207</b>
C.1 Adding a User Screen . . . . .	208
C.2 Dashboard View Screen . . . . .	209
C.3 Login Screen . . . . .	210
C.4 Internal Mail Screen . . . . .	211

C.5 Update User Screen . . . . .	212
C.6 Redirect Login Screen . . . . .	213
C.7 Redirect Login Error . . . . .	213
<b>D Prototypes - First Edition</b>	<b>214</b>
D.1 Creating a Schedule . . . . .	214
D.2 Creating a Shift for an Employee . . . . .	216
D.3 Maintaining an Employee . . . . .	218
D.4 Creating a New Employee . . . . .	220
D.5 Taking Emergency Leave . . . . .	222
D.6 Viewing the Schedule . . . . .	223
<b>E Permission Reference Documentation</b>	<b>224</b>
E.1 Permissions Overview . . . . .	224
E.2 Permission Set . . . . .	227
<b>F Glossary</b>	<b>231</b>
<b>G Index</b>	<b>237</b>

# List of Figures

<b>5.1 WebAgenda Use Case Diagram</b>	A graphical overview of the interactions between users and the system. . . . .	35
<b>7.1 Class Diagram</b>	The classes that are utilized by the system with their respective attributes . . . . .	81
<b>8.1 Create New Schedule State Diagram</b>	A representation of different states the system goes through in creating a new schedule . .	83
<b>8.2 Request Shift Change State Diagram</b>	A representation of different states the system goes through in allowing a user to request a shift change . . . . .	88
<b>8.3 Request Time Off State Diagram</b>	A representation of different states the system goes through when an employee requests time off . . . . .	90
<b>9.1 Add New Employee Activity Diagram</b>	The actions required in order to add a new employee to the system . . . . .	93
<b>9.2 Generate Report Activity Diagram</b>	The actions required to generate detailed and relevant reports . . . . .	94
<b>9.3 Request Shift Change Activity Diagram</b>	This diagram shows the status of a shift change request, from being created by an employee, reviewed by a supervisor, and being implemented in future schedules or removed from the system. . . . .	95
<b>9.4 Shift Exchange Activity Diagram</b>	This diagram shows the status of a shift being exchanged or given to another employee. .	96

<b>10.1 Package Diagram</b>	This diagram shows all the packages and objects included in the system . . . . .	99
<b>11.1 Conceptual ERD Diagram</b>	. . . . .	103
<b>11.2 Implementation ERD Diagram</b>	. . . . .	104
<b>13.1 Interface Class Diagram</b>	This diagram shows the objects that make up the interface for WebAgenda and methods associated with them. . . . .	114
<b>13.2 Persistence Class Diagram</b>	This diagram shows the objects that make up the persistence layer, the classes and objects that utilize backend or data objects . . . . .	115
<b>13.3 Application Class Diagram</b>	This diagram shows the objects that make up the application-side real world objects. . . . .	115
<b>13.4 Business Class Diagram</b>	Similar to the problem domain, this diagram displays the business classes. Business classes contain data that reflect real world objects that are used and modified in a business. . . . .	116
<b>14.1 New Schedule Creation Sequence Diagram</b>	A representation of the proper sequence in creating a new schedule . . . . .	118
<b>14.2 Request Shift Change Sequence Diagram</b>	A representation of the proper sequence in requesting a shift change . . . . .	119
<b>14.3 Viewing Schedule Sequence Diagram</b>	A representation of the proper sequence for viewing a schedule . . . . .	120
<b>14.4 Book Days Off Sequence Diagram</b>	A representation of the proper sequence for booking days off . . . . .	121
<b>14.5 Administrative Schedule Viewing Sequence Diagram</b>	A representation of the proper sequence for a manager or authoritative employee to view multiple schedules . . . . .	122
<b>14.6 Request Shift Change Sequence Diagram</b>	A representation of the proper sequence for changing a shift with another employee . . . . .	123
<b>14.7 Update Availability Sequence Diagram</b>	A representation of the proper sequence for updating an Employee's availability . . . . .	124

<b>14.8 Access Schedule Sequence Diagram</b>	A representation of the proper sequence for accessing an Employee's Schedule . . . . .	125
<b>14.9 Add Supervisor Sequence Diagram</b>	A representation of the proper sequence for creating an employee with supervisor permissions	126
<b>14.10 Create Employee Type Sequence Diagram</b>	A representation of the proper sequence for creating an employee to work a specific job . .	127
<b>14.11 Create New Employee Sequence Diagram</b>	A representation of the proper sequence for creating a new employee . . . . .	128
<b>14.12 Distribute Report Sequence Diagram</b>	A representation of the proper sequence for distributing reports that are created . . . . .	129
<b>14.13 Generate Report Schedule Sequence Diagram</b>	A representation of the proper sequence for generating reports . . . . .	130
<b>14.14 System Log In Sequence Diagram</b>	A representation of the proper sequence for logging into the system . . . . .	131
<b>14.15 Maintain Workgroup Sequence Diagram</b>	A representation of the proper sequence for editing workgroups to the desired situation .	132
<b>14.16 Maintain Employee Sequence Diagram</b>	A representation of the proper sequence for editing an employee's position to the desired situation . . . . .	133
<b>14.17 Maintain Employee Sequence Diagram</b>	A representation of the proper sequence for editing an employee's skills regarding the desired situation . . . . .	134
<b>14.18 Review Request Sequence Diagram</b>	A representation of the proper sequence for reviewing an employee request by an authoritative employee . . . . .	135
<b>14.19 Send Event Sequence Diagram</b>	A representation of the proper sequence for posting an event that certain groups of employees will see . . . . .	136
<b>14.20 Book Days Off Sequence Diagram</b>	A representation of the proper sequence for an employee to reduce working days in their schedule . . . . .	137
<b>14.21 Manager View Schedules Sequence Diagram</b>	A representation of the proper sequence for a Manager to view multiple schedules at one time . . . . .	138

**14.22Request Shift Change Sequence Diagram**

A representation of the proper sequence for requesting another user to take or exchange shifts with . . . . . 139

**14.23Update Availability Sequence Diagram**

A representation of the proper sequence for changing an employee's available work times . 140

**14.24Send Emergency Notification Sequence Diagram**

A representation of the proper sequence for handling emergencies that prevent employee's presence at work . . . . . 141

**14.25View Workgroup Schedule Sequence Diagram**

A representation of the proper sequence for viewing schedules of employees working for one job type . . . . . 142

# **Part I**

## **Preface**

# Chapter 1

## Document Purpose

### 1.1 Introduction

This Requirements Documentation is a product of the need to produce a **System** or Entity that helps a business or company perform a job or series of tasks. The document itself contains information that relates to the objectives of this system, the problem that this system is designed to solve, and much of the finer details as well as environmental factors that are taken into consideration. This document will attempt to describe in full detail, the who, what, where, when and why of the System. Here is a brief description of the **Project** this document refers to:

**WHO** : We are targeting Businesses and Organizations that do not have functional, sophisticated and easily maintainable scheduling systems, but are looking to implement one without the frustrations of overly complex licensing, cost, and vendor lock-in. In this Project, the **Deerfoot Inn and Casino** are being directly involved to develop a scheduling system free of such inconveniences. This is partially achieved by licensing our product under the **General Public Licence (GPL)** and creating a product accessible by anyone with a **Web Browser**. The license agreement that the System is distributed with will clearly state any deviation from GPL, if applicable, upon installation. Source code will also be shipped with the product so any programmer or business can use and modify the program according to their needs. Code will be clearly documented for readability and to aid in its development. As our client, the Deerfoot Inn and Casino is going to be represented by their VP of Operations, Robin Featherstone. The Deerfoot Inn and Casino is going to be the main **testing** and **deployment** focus for the system and will be used to fulfill all their scheduling needs in most if not all their **departments**.

**WHAT** : This document describes the Scheduling System "**WebAgenda**" , designed to monitor the schedules of small-to-medium businesses and provide a cheap, easy-to-install-and-use system for scheduling employees. We will be designing it to easily and effectively manage approximately 500 employees for one company. This system does not monitor any other WebAgenda or scheduling systems that may exist on another network, or even on another **server** in the same business or company. Functionality, however, can be built in as users of this software are free to modify it so. This independence may be seen as a security benefit.

**WHERE** : As specified above, small-to-medium business are the target and so this system is meant to reside on those businesses' servers. In the event that a business does not have a server, any computer that runs a server and is connected to the Internet should theoretically be able to run this software. Any computer with a web browser installed should be able to use the system. In fact, most of the standard browsers will be involved in the testing of the final product that this document describes.

**WHEN** : This project should be completed in less than six months. Documentation will be released over that period of time along with revisions of this design document. Once complete, the final product will be supported as long as there are people interested or paid to support it. **Source code** will be freely available if a fix is needed, and as the project developers we may decide to support this product, depending on how this is implemented it may be for a fee.

**WHY** : The world is always in need of powerful, usable, and free software (both as libre and as gratis), especially in the business and IT sector. Many current scheduling solutions have a high cost or limited functionality unless until you purchase the full version. Most, if not all currently marketed scheduling solutions are proprietary and require monthly fees. If you want added functionality you must purchase additional software with possible monthly fees in order to get the job done. With WebAgenda, additional functionality can be built in-house, contracted out, or even mutually shared with other companies; there are no limitations and you always get the full version, for free.

It is not uncommon for businesses to use Excel spreadsheets, paper, or other resourceful but inefficient means to complete scheduling tasks. Security is always an issue. Mass distribution of schedules that are **encrypted** and on a need-to-know basis to hundreds of employees are not always feasable for the employer to put into practice, especially when solutions would cost time and money for little benefit when a simpler centralized system does the job. This project is a college requirement that is designed to have the biggest benefit-to-cost ratio for the widest distribution of potential users and partners.

This project aims to fulfill an often-overlooked aspect of business that time is wasted on needlessly without compromising productivity gains from the old system.

# Chapter 2

## Document Standards

**Document Formatting** All documents included in the project that are physically produced will be created using the following standards below. Web-based documentation does not follow all standards, as many do not apply to an electronic medium.

This document adheres to the following standards:

### 2.1 Document Standards List

- Single Sided
- Bound
- Header: [Chapter Title] - [Sub-Chapter Title] w/ a maximum of two sub-chapters per page, using the latter sub-chapter title if this event occurs. Example: Chapter 2 - Section 2.0 Document Standards. Once the header is located in an Appendix, it will only consist of the word “Appendix” and the letter that represents that specific appendix.  
Font should be a shade of grey.
- Footer: Page number will be:
  1. at the bottom center for single-sided pages or
  2. on the outside (bottom left for left-side page, bottom right for right-side page) if document is double-sided
- Images, Pictures, Charts must have figure text below the image, centered beneath it. If the figure is less than 40% of the page width, it can have text wrapped around the figure. The figure should always be on the left-hand side of the text if text wrapping exists. If figure is greater than 40%, it should be on its own lines with no text wrapping it.

- All paragraphs will start with a tab of 1 cm in length. The document is generated by **Latex** so that there is consistency in these measurements.
- Parts, Chapters and Sections are labelled in the Table of Contents due to the importance of their content. Smaller pieces of the document such as subsections and paragraphs, are not.
- Any bolded words in the document will have a definition in the glossary except for the beginning of paragraphs, numbers in lists, and titles. For glossary definitions, the bolding only applies to the first instance of the word(s) in the document although multiple instances of a word can still be found in the index.

# **Part II**

# **User Requirements**

# **Chapter 3**

## **Business Overviews and Objectives**

The Deerfoot Inn and Casino is a Canadian-owned casino, hotel and conference center located in Calgary, Alberta. The Deerfoot Inn and Casino is Calgary's premier hotel and convention center in the southeast quadrant. They provide a large number of entertainment, conference, and meeting areas to accommodate any group and size. With numerous events and conferences every month, the number of clients that the Deerfoot Inn and Casino need to accommodate is immense. They employ a lot of staff to deal with the large workload, but as the workload changes, so does the number of scheduled staff required. The business objective of the Deerfoot Inn and Casino is to become one of the top casinos in Alberta and to increase profits year after year. They achieve this by sponsoring events, advertising and other marketing means to increase sales, customers, and word-of-mouth.

## 3.1 Statement of the Problem

The Deerfoot Inn and Casino currently schedules their employees by using spreadsheets and paper postings. Creating based on a paper-based system has led to organizational issues and misunderstandings with those who do the scheduling. As the schedules are paper copies posted in common areas, they can get misplaced, damaged (ripped), or removed. In order to change a schedule in circulation, there would have to be a photocopy or an update to the original schedule which could then be printed and distributed. All old copies would have to be removed and there would be a risk of employees misinterpreting shifts, employees not showing up, and discontent. When employees do not work consistently for a period of time (2 weeks straight, for example), checking a schedule that can only be read at the workplace can frustrate staff into making assumptions when they work. If a shift change is completed over a paper based scheduling system, the supervisor is told of the exchange and a note may be written down on the schedule. A note can be modified or forged by an employee, or the specifics of a shift negotiated after an agreement is made. Misunderstandings could lead to supervisor and management frustration, employees showing up for incorrect times or shifts, and having employees that have not worked being paid for another's time.

Communicating schedules between departments becomes a difficult task when using a paper-based system. If an employee works in multiple departments then having proper communication channels is a necessity and requires extra work for both supervisors. To schedule shifts for employees that work in multiple departments, one supervisor may have to physically ask another for the current schedule of an employee or by ask via email which would take longer. Every change to the multi-department employee's shifts would have to be communicated to all corresponding supervisors. The frustration could result in affected employees getting less hours, miscommunications for shift details, working two shifts simultaneously, or high employee turnover rate in a worst-case scenario.

One of the problems that arise in the paper-based system that our client has addressed as a priority, is that supervisors do not know when and whether employees have seen the schedule. Even if a schedule has been posted where employees have had ample time to view it, they cannot be sure; likewise, employees may not always be in a position to check it until the timeframe becomes short and urgent. With an electronic scheduling system, supervisors can send out priority notifications that pop up everytime a user logs in, or while they are using the system. Likewise, employees can check their schedules from virtually anywhere around the globe. One of the features being promoted to the client, is the ability to view when an employee has last logged into the system. This enables supervisors to view trends and follow up on employees that are not checking their schedules for reasons unknown. This function will reduce the number of problem areas for a company.

## 3.2 System Environment

Web Agenda will be hosted on the Deerfoot Inn and Casino's web servers. The scheduling system will run alongside their website and function independant of other systems residing on their server. It will run on a sub-domain of their current site, allowing the scheduler to be accessed from anywhere at any time provided the user has proper authority to use the service. The system will be able to be accessed through their internal computers as well as any external computer or mobile device with internet access. The system does not deal with monitoring equipment, daily employee clock-in and clock-out times, or employee payroll information. This is mostly handled by their current system Aloha that handles employee payroll information and shift monitoring. The human resources department relies on the current hourly data from Aloha to cover employee wages.

## 3.3 Current System

The current scheduling system is seen as workable, but lacking. The supervisors and administrators have different methods of scheduling staff under their authority. The schedules are mainly done in a spreadsheet **format** with templates printed and posted on a wall in employee common areas at the hotel or casino. Employees then write down their availability on the posted sheets for that time period. Supervisors and administration use spreadsheets to create official schedules which are then posted back to those areas. However, none of the schedules are identical and therefore become extremely difficult to track employee data. Also, only a limited number of schedules are printed for any given shift period and if they are lost or destroyed for some reason, the supervisor has to go and re-print the schedule to post. For shift change requests, the employees have to be physically present to complete the transaction. In addition to the validity of the changeover, having both parties present negates the risk of losing the shift change request form or not having it completed properly. There is also the issue of forgeries and miscommunications that the physical presence addresses.

# Chapter 4

## System Requirements

### 4.1 Functional Requirements

Users can export their schedule to a relevant viewer. This allows users to view schedules using different applications assuming the relevant viewer had already been installed.

**Document Viewers** Icons will represent the exporting options. Each icon will hyperlink to display the schedule in the relevant application. The icons displayed will include viewers for:

1. MS Word
2. MS Excel and generic spreadsheets
3. MS Outlook
4. Adobe PDF
5. iCal
6. iPhone calendar
7. Android calendar
8. Blackberry calendar

We will be designing an **Information System** to track and plan schedules for staff. The tracking module will include a way for employees to remotely access their schedules, request schedule changes and book days off.

As with any business, the system will require the ability to save and retrieve old schedules as well as templates and temporarily saved (partially completed) schedules. They will be stored in a database in this case (as opposed to an excel spreadsheet) which should provide more reliable speed and stability. It's requirements include having a very very low (non-existent) corruption rate of data. Data validation shall be in place to ensure that the inputs are proper and abide by any constraint it is given. Reliable **backups** are a must and should be completed easily by any staff member who is authorized to do so.

Old schedules should be stored, but not editable, in the database. They should not be saved based on dates because dates on the computer can be changed or modified. While old schedules can be viewed based on their date, they are not defined by it. New schedules that are based on a future date (as opposed to being currently in use) should be editable by appropriate permission users. Only users who are given permissions to edit schedules should be able to edit them. If intervention is required for a current schedule, staff in higher positions of power can be given permissions to make those changes with notifications being sent out to staff that are affected by the change.

Staff with default permissions can only view current schedules and request changes. Higher privileged users can edit the current schedule, view older schedules, or make requests of higher privileged users. All staff will be able to contact one another through built-in messaging functionality.

The structure of permissions is level-based, with each level having permissions that are equal or greater to the one below. There can be multiple variations of a lower permission level, which can have different levels based off those. Each level has a ‘root’ template: a set of permissions that are the lowest common denominator for that level. A variation of a root template has a suffix that shows that it is a variation.

Example: Permission Level 0 is the root and the lowest permission level. Permission Levels 0a, 0b, and 0c have slightly varying and higher permissions which apply to different types of employees. Level 0b can be higher than root level 1, but permission level 1b would have permissions higher than 0b.

It is possible to elevate permissions from one level to another, provided a higher authority approves of it. Even then, the heightened permissions are temporary.

Common permissions we expect to implement are the following: (see next page)

- Can Read Current Schedule
- Can Read Previous Schedules
- Can Read Future Schedules (Even though they may be blank)
- Can Edit Active Schedule (make changes in real-time)
- Can Write Next Schedule
- Can Write Future Schedules (This would essentially take employees out of the employee-labour-pool that others could use. Example: Admin wants one employee to serve a specific task that is not a regular task, therefore omitting the employee from a supervisor to schedule him or her to a regular task)
- Can View available resources (Employees/Time slots not filled/Report features)
- Can Search Previous, Current, or Future Schedules based on criteria specified by the user.
- Can change permissions of users [every level of tree is a potential possibility]
- Can Read Logs that are generated (What type of logs)
- Can create reports [depending on target: previous schedules, all prev schedules, future schedules, or current. Can **export** to other formats.]
- Can export schedule to format x (current, possibly future depending on restriction)
- Can request days off (to what length of time in the future, how many per time slice)
- Can request vacation (unlike days off, full-time employees may have extra scheduling benefits)
- Can request emergency notifications to supervisors / admin: if in accident, family member dies, act of God occurs and cannot come to work. Able to utilize ‘not in advance’, but immediately. Should create a notification of some sort to proper admin’s that monitor emp.) This only applies to online requests, disabling this does not mean they cannot phone in.
- Can view Inactive Employees (only levels lower than you, not similar authority employees unless granted from higher levels)
- Can work shifts on short notice (# hours/days from shift start)
- Can exchange shifts with other employees working similar jobs
- Able to send notifications to [type of employee]

Permissions list is not comprehensive and subject to change.

Employees should be able to set their availability through the system so that the schedules can be auto-generated in addition to ease of use. Schedules can only be changed so often without admin consent. Auto-scheduling must be able to handle hundred upon hundreds of employees and staff. Should include **RSS** functionality, e-mail notifications, have a notifier when logging in to view schedule, a notification rss feed, schedule to device exporting such as palm and blackberry, and schedule to calendar programs such as mozilla suite, **Mac OS X iCal** and **Microsoft Outlook**.

The system is going to be able to be easily installed on any **Sub Domain**. This meaning that any employer or business that would like to use the system has the ability to easily install it, as long as they have a website capable of running the system. The application will be able to handle any management structure that could come up. As a user of the system with the highest permissions, such as a company owner, they will have the ability to create as many different permission levels as they deem fit. This allows any company, no matter how large or small to reap the benefits of web agenda.

Schedule-by-request feature, where employees can request work on a schedule that is optional (provided by higher-ups in the company, for non-essential work opportunities) The WebAgenda program will be licensed under the General Public Licence 2 so that the company is free to utilize the software to their purposes. Documentation as well as source code will be available to one and all who download/request it.

## 4.2 Non-Functional Requirements

The scheduling system does not monitor employee attendance or payroll functionality, although a stat holiday pay estimate feature will be implemented at the behest of the Deerfoot Inn and Casino Manager. WebAgenda is purely a scheduling, report generating, reference, and distribution system until functionality is extended at a later date. Scheduling is done by department or by a specific unit of the company.

The Deerfoot Inn and Casino needs to provide schedules for a minimum of 500 employees. In a worst-case scenario, the system must be able to provide 500 constant connections reliably. At full capacity, the system should be able to respond to a user within 2000 milliseconds of a request to view a schedule. For more intensive tasks such as searching for an employee or group of employees in addition to additional filtering, anywhere from 3000 to 5000 milliseconds maximum is reasonable under full load. While there may be potential for a company to include more than 500 employees, the system is designed for optimal performance at 500 or less. WebAgenda does not guarantee accuracy above that limit. Multiple WebAgenda scheduling systems can be run in parallel to manage certain departments independently if one scheduler becomes difficult to manage and the hardware is available. System responsiveness will not be guaranteed if more than one system is running at a time. However, it is assumed the hardware will be powerful enough to reach those statistics in such a situation. If multiple scheduling systems are running, staff in one department will only be able to log into the system that holds their department; WebAgenda currently can not communicate with other WebAgenda systems unless data is explicitly exported and imported from database to database. In large corporations, having one scheduling system per department could be effective, provided higher management has access to information they need and maintenance support is available; either in the form of official, paid support or staff / IT department.

It is up to the supervisors, departments, and administrators to ensure that the schedule is indeed accurate and to their liking. They will have the ability to change things as they desire; a generated schedule will not automatically be uploaded and viewable to all. Rather, it will be saved until it is activated. An active schedule can be seen by anyone with schedule read permissions, and will be seen as official when the schedule's date coincides with the current date. The scheduling system will also feature a reporting function, which can list employee and schedule data in different ways. This will also be used to test the system for integrity and stability, although it may be restricted to administrators and maintenance staff. The system must have zero tolerance for data loss during operation. This does not account for human error.

Most, if not all, of the actions performed on the scheduling system are governed by permissions. An employee will have enough permissions to retrieve their schedule, request modifications, book days off within a certain time period from the current date, and change their availability. Depending on the type of employee, permissions may be restricted or granted. A contractor will not be able to book days off or change availability; They can be given an account to remain in contact with the required administration through the messaging system. Supervisors are basically employees with higher permissions including those that allow them to modify the schedule and to some degree, authorize it. **Department** leaders, Managers, and administration will have even higher permissions and ultimately be able to approve or modify a supervisor's decisions in order to suit their needs. Permissions will reflect duty, not overall authority. For maintenance and support of the system, a technician can have access to the system internals while an administrator will have access to creating employees of any type: a permission set that no other employee should have.

We will not be selling the software. The system also does not monitor anything outside of employees and date-related events; equipment, rooms, and other commonly monitored items are not within the scope of this software. Additional functionality can be built on this software for no cost provided it is licensed similarly. Employee shifts that are scheduled will be measured in time, currently to the nearest quarter of the hour. Setting up the server and performing administrative or server-related tasks is in the scope of this project, but exists in another document. As the system is running on a server, uptime is very important and a backup server may be of interest to the client. The general consensus for desired server uptime is 99.9%, which is approximately 364/365 days in the year that the system needs to be accessible. Backing up should not interrupt the function of the server, although restoring a previously backed up database may cause some time for the system to be inaccessible or partially functional.

## 4.3 System Interface Requirements

The employees at the Deerfoot Inn and Casino will have varying experience levels using computers. As such, the system should be built with a descriptive and intuitive **interface**. The system will be running as webpage for the front-end, allowing any computer with a web browser and **javascript** to interface with it. The employee's interface should have quick access to their schedule and job-related information with minimal effort. Supervisors and employees with higher permissions should have quick access to functions specific to their job, such as administration. The interface reads and writes information after validating it to the system backend. The backend consists of a database filled with employee and schedule information which can be interpreted by the interface when it is requested by the user. With the addition of a plug-in system, any user of the system can take relevant data from the backend and export it to an array of different formats.

Web Agenda is a full web based system and therefore in order to get communications out to the employees using the system, it needs to be able to interface with other applications. The following applications need to be able to be communicated with to ensure full functionality and ease of use of the system.

**Microsoft Outlook:** This mail application for Microsoft Windows operating system is the main mail client that the Deerfoot Inn and Casino uses for inter-departmental communication. Our system will need to be able to export schedules to a format that Microsoft Outlook will be able to import into its calendar function.

**Mac OS X iCal:** This application is an optional application that the application will need to interface with, and may be introduced at a later date as a plug-in as opposed to being added as a full feature upon release. The application needs to be able to export calendars and contact information to be added to Mac OS X iCal. Since Mac OS X iCal is not used directly by the casino but may be used by employees using the system this will still need to be put into the system at some point.

**GNU/Linux KCal:** This application is an interface for KOrganizer, one of the more common calendar applications that WebAgenda will need to communicate with. Kcal is included in one of the two major Linux desktop environments and although it's not used directly by the Deerfoot Inn and Casino, it could still be used by the end user of the system. Netbooks and specialized phones may have this software installed.

**Mobile Web:** The application will need to be able to work well with mobile devices. That being said the best way to have full functionality across multiple mobile platforms is to have a separate web interface to be able to work well with all mobile web enabled devices. Some business tasks are going to be handled by other applications at the Deerfoot Inn and Casino. Web Agenda will not need to interface with these applications as they handle aspects of the program that are outside of its scope.

**Aloha:** Aloha is an employee management system that handles tracking when employees clock in and clock out for a shift. This doesn't handle scheduling of the employees at all. We will not need to interface with Aloha as that would be outside the scope of the project.

Our system is designed to export to calendar formats, spreadsheets, text, and potentially any format the user wants via a plug-in. This generally applies to schedules, although employees with proper permissions can create and output reports. The system will also have to be usable on mobile devices such as phones, personal data assistants, and small laptops. The interface works closely with a permission set that each employee is given. It governs what tasks an employee is able to perform and will display the interface accordingly. The actual WebAgenda software is distributed with one employee active by default and having all permissions required to allow other users to utilize the system.

The hardware required to use the system are a pointing device, usually a mouse or touch screen depending on what computer is used to access it. A wireless or wired connection to a network that can contact the WebAgenda server is essential. Although not a necessity, having a keyboard or method of character input into the system will be needed to access some of the functionality such as sending messages, editing employee information, and search input. The possibility of using a computer or device without this hardware is almost non-existent as of this document's production.

## 4.4 Maintainability and Administration Requirements

WebAgenda will be hosted on web servers that are already in service and maintained by the Deerfoot Inn and Casino. IT Staff are already in place for maintaining the hardware for these servers so hardware maintenance will be outside the scope of this project.

All employee and scheduling information will be stored in a database. The WebAgenda system will provide the means for automatic backups of all data, though selective and manual backup functionality may be introduced at a later date.

A robust plug-in system will be included so that additional functionality can easily be added to the system, such as inclusion of data from Aloha or the ability to send notifications through **SMS**. Additional development time will be required to create plug-ins, though the plug-in system should significantly reduce the amount of time that it will take for these features to be created and added compared to more conventional methods.

Staff may require additional training to review and understand any error messages produced by the system, which can then be used to troubleshoot and fix errors. A knowledgebase will be provided that contains an in-depth explanation of all components of WebAgenda and its error messages with appropriate solutions.

At least one manager must be in the system with full privileges at all times to act as an administrator over other managers and supervisors. Without this, certain employees could not have their permissions upgraded; it would not be possible to add new supervisors or managers.

The previous scheduling system must be maintained for at least several months, and will be used in tandem with the WebAgenda system after it is first brought online. This will allow for a more comfortable transition between the two systems for all employees, and allow them to fall back on the old system in the event that there are any issues with WebAgenda during this transition. An emergency plan, in the unlikely event that WebAgenda should ever fail, will be implemented during the transition.

## 4.5 Usability Requirements

The system needs to be able to be used by a variety of clientelle. Users of the system will vary in experience so the system needs to take this into account. At one end of the spectrum, the system needs to be customizable and intuitive to allow for maximum usability. On the other end, the system needs to have a lot of functionality and features that advanced users of the system can access upon request without being buried under too many layers. With the amount of functionality that the system will bring, it's important that the system is usable by every user to do their jobs. This is a crucial aspect of the system and the interface design of the system that the project team will need to keep in mind at every level of the development process.

# **Part III**

## **Requirements Models**

# Chapter 5

## Use Case Diagram

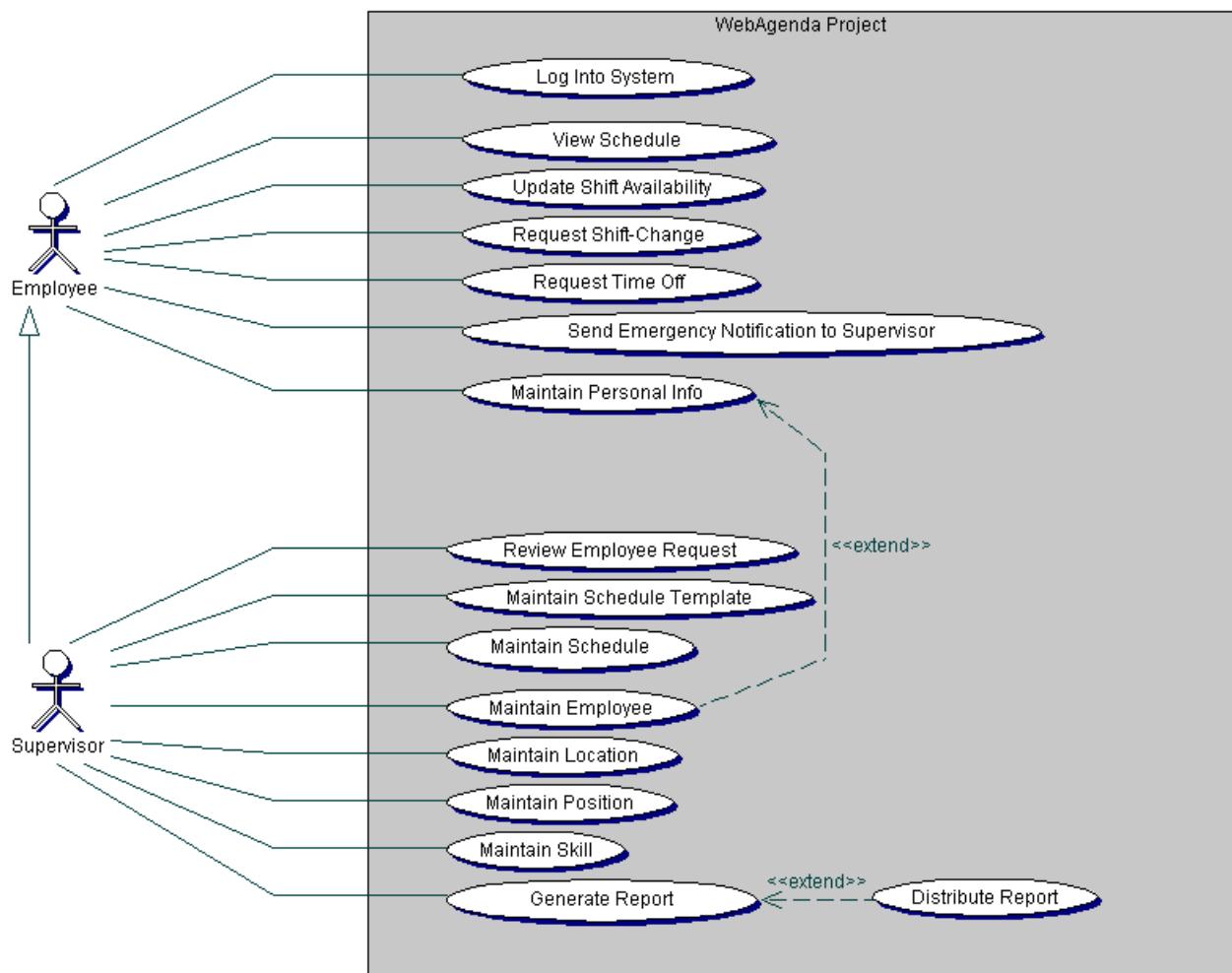


Figure 5.1: **WebAgenda Use Case Diagram**

A graphical overview of the interactions between users and the system.

# Chapter 6

## Extended Use Case Diagrams

### 6.1 Log-In System

**Description:**

The login use case goes through how a typical user would log into a system. If the user does not enter the correct information they are given an error and they can then re-enter their credentials.

**Pre Condition:**

User must access the online system and have the login page open.

**Post Condition:**

Dashboard is displayed to a newly logged in user

**Actor:**

Employee of a company using WebAgenda

#### Normal Flow

Actor	System
1. User enters a string into the username field of the login screen.	
2. User enters a password into the password field of the login screen.	
3. User presses the “Log In” button to log into the system.	

Actor	System
	<b>4.</b> System takes the value that was entered from the username field and the password field when the log in button was pressed, queries the database and compares the values.
	<b>5.</b> When the values match the values that were queried from the database, proceed to allow access and display the dashboard screen. (A 5.1)
	<b>6.</b> Presents the dashboard screen and populates it with user-specific data.

### Alternate Path 5.1

User enters data into the fields that do not match the database records.

Actor	System
	<b>5.1.1</b> System queries the data entered by the user and they do not match the data found in the database.
	<b>5.1.2</b> System displays error message above the username field advising the user that the username/password combination was invalid.
<b>5.1.3</b> User enters data in username field again	
<b>5.1.4</b> User enters data in the password field again	
	<b>5.1.5</b> Return to Normal Flow Step 5.

## 6.2 Access Schedule

**Description:**

This use case will be used for the actors accessing their schedule through the schedule screen.

**Pre Condition:**

A user needs to successfully be logged into the system and the dashboard screen being displayed.

**Post Condition:**

A schedule widget is displayed to the user with user specific information

**Actor:**

Employee of a company using WebAgenda

### Normal Flow

Actor	System
1. User clicks on the Users tab	
2. User clicks on Current Schedule link <i>Note to Mark: changed from schedule button</i>	
	3. System accesses schedule data based on what user is logged in and accessing the schedule screen. (A 3.1)
	4. System displays the schedule screen.
	5. System retrieves the schedule widget and displays it on the schedule screen with information specific to the user that is currently logged in.

### Alternate Path 3.1

User is not added to any current schedule

Actor	System
	<b>3.1.1</b> System does not pull any schedule data for the current user meaning they are not currently added to any schedule.
	<b>3.1.2</b> System displays a message to the user saying they are not currently scheduled in the system.

## 6.3 Review Employee Request

### Description:

This use case is used for the process of a supervisor reviewing an employee request and either accepting it, deleting it, or updating it.

### Pre Condition:

The user must be successfully logged into the system and on the dashboard screen

### Post Condition:

A schedule widget is displayed to the user with user specific information

### Actor:

An Employee logged into the system with a permission level above other employees in the same job category

### Normal Flow

Actor	System
1. Clicks on the Employee Requests Widget.	2. System displays the new Employee Requests screen and loads a new copy of the Employee Requests widget.
4. Loop: For Each Request - User chooses to approve the request (A 4.1, A 4.2)	3. System loads all the of the requests for that supervisor that are in the database.
	5. System accepts the change and modifies data in the system based on what type of request was approved. (Example: shift change modifies the shift, etc.)
	6. Notification is sent using the notification in the API to however many users are involved in the request.
8. End Loop	7. Removes the request that was approved

### **Alternate Path 4.1**

User chooses to modify the request

Actor	System
<b>4.1.1</b> User clicks the Modify button on the Employee Request widget	
	<b>4.1.2</b> System displays text boxes in place of all fields to allow the supervisor to modify values.
<b>4.1.3</b> User Clicks the Submit button to submit the new values (A 4.1.3.1)	
	<b>4.1.4</b> System will accept the new values and change the values in the database according to the new values entered. (E1)
	<b>4.1.5</b> Return to Normal Flow Step 6

### **Alternate Path 4.1.3**

User Clicks the Cancel button to cancel any values entered

Actor	System
<b>4.1.3.1</b> Actor Clicks the Cancel button	
	<b>4.1.3.2</b> System reverts back to the original request that was sent by the employee
	<b>4.1.3.3</b> Return to Normal Flow Step 4

### **Alternate Path 4.2**

User Chooses to delete the request

Actor	System
<b>4.2.1</b> Actor Presses the Delete button	
	<b>4.2.2</b> System erases the request from the screen, however it will still be stored as a disabled user request in the database to be accessed if need be.

**Error Flow 1**

User Enters invalid data after being request is modified

Actor	System
<b>1.1</b> Actor Enters invalid data for the Modify Employee Request.	
	<b>1.2</b> System queries database to check data.
	<b>1.3</b> System displays error message pertaining to what fields were entered incorrectly.
<b>1.4</b> User Clicks OK button and enters correct data for missing fields.	

## 6.4 Add Supervisor

### Description:

This use case is used when promoting an employee to a supervisor position, accepting it, deleting it, or updating it.

### Pre Condition:

A manager must be logged into the system and on the maintain employees screen

### Post-Condition:

Success message telling the logged in user that the employees permissions were successfully changed

### Actor:

User with Permission Level higher than an employee considered to have supervisor authority

### Normal Flow

Actor	System
1. Actor presses the promote button	2. System produces a message that asks the user if they are sure they want to promote the user to supervisor.
3. Actor Clicks the Confirm button (A 3.1)	4. System raises the permission level of the user that is promoted by 1. (A 4.1) 5. System displays a confirmation box displaying the employee's current permission level and an overview of their now current permissions.

### Alternate Path 3.1

Actor Clicks the Cancel button

Actor	System
3.1.1 Actor Clicks the Cancel Button	3.1.2 Return to Normal Flow Step 1.

**Alternate Path 4.1**  
Employee does not have a workgroup

Actor	System
	<b>4.1.1</b> Actor is Redirected to the Maintain Employee screen (See Maintain Employee extended use case)

## 6.5 Booking Emergency Days Off

### Description:

This use case describes how a user would go about using the WebAgenda web interface to send an emergency notification for not coming into work. This case assumes that an emergency has happened and the user is not using the system for another reason.

### Pre Condition:

The user must be successfully logged into the system, presented the dashboard screen and the permission for sending emergency notifications is enabled

### Post Condition:

A schedule widget is displayed to the user with user specific information

### Actor:

An employee for the company that can book off days (which may exclude contracting positions)

### Normal Flow

Actor	System
1. Click the “Users” tab on the dashboard	
2. Click the “Days Off” item on the sidebar	
3. Click on the “Emergency” sub-item on the sidebar	
4. Click the “Next Shift” radio button option to affect the employee’s next working shift. If the shift has already started, that shift will be the one that is booked off. (A 4.1, A 4.2)	
	5. Present user with an overview of the date(s) booked off and a textbox for them to fill in an optional reason for their absence.
6. User clicks the “Confirm” button.	
	7. Send a notification to the user’s supervisor, including the reason for the absence if user entered any text into the textbox.

**Alternate Path 4.1**

Book off an extended period of time

Actor	System
<b>4.1.1.</b> Click on the extended period radio button	
<b>4.1.2.</b> User types a number that represents a period that they will be unscheduleable for work. Maximum will be determined by the system permissions set. (E 1)	
<b>4.1.3.</b> Clicks Confirm button	
	<b>4.1.4.</b> System changes notification message to supervisor including the number of days the employee expects to be away.

**Alternate Path 4.2**

Book off an unknown period of time

Actor	System
<b>4.2.1.</b> Click on the unknown period radio button	
<b>4.2.2.</b> Clicks Confirm button	
	<b>4.1.3.</b> System changes notification message to supervisor requesting that they contact the employee to understand the situation as the employee may be gone for an unspecified length of time.

**Error Path 4.1.2.**

A non-number is entered into the extended period of days to book off field

Actor	System
	<b>4.1.2.1</b> Characters that are not numbers will just not appear in the field.

## 6.6 Update User Availability

**Description:**

An employee that wants to change the days they work has the option to configure a template schedule populated with shifts they can work, called an availability. As long as the template schedule meets minimum shift and hour requirements, within a period of time, they will be able to work that new schedule at their job. Note that the shifts placed in the template are not guaranteed, but schedule creators will have the option of using those employees at those times.

**Pre Condition:**

The user must be successfully logged into the system, presented the dashboard screen and the permission for changing availability is enabled

**Post Condition:**

Supervisor does not reject the new availability user creates, duration for availability to take effect passes and new schedule after that date will contain the new availability. Old availability will be archived, current employee availability will be overwritten by new one.

**Actor:**

An employee for the company that is in a position to fill out their availability (contractors may not be able to)

**Normal Flow**

Actor	System
1. User clicks on the “Users” tab on the dashboard.	
2. User clicks on the Availability link in the sidebar.	
	3. A blank week-long schedule template is produced and presented to the user. A Finish or Complete button will be shown but disabled until a time that the schedule meets requirements.
	4. Weekly Schedule option is selected by default (A 4.1)

Actor	System
<b>LOOP</b>	
5. User clicks on one of the 7 columns for each day of the week. Columns are divided up into rows that represent time. Clicking on a row will create a shift object starting at the time representation of the row.	
6. [Optional] User can expand or shrink the shift anywhere in the day by grabbing the start or end row of the shift object and dragging it to another row. User can move an entire shift without resizing it by clicking the middle and dragging it, even between different days.	
7. User creates shifts until the minimum number of shifts (as defined in the system permission set) is met and/or until the required number of hours is met.	
<b>END LOOP</b>	
8. User clicks on “Complete” button. (A8.1)	
	9. Availability is saved alongside their current availability, which can be blank if they are new.
10. [Optional] User clicks on “Activate Availability” button to request that the current user’s availability be changed to the saved one.	
	11. Supervisor of the employee is notified that a new availability was filled out.

**Alternate Path 4.1**  
Monthly Schedule is selected

Actor	System
<b>LOOP</b>	
<b>4.1.1.</b> Continue Normal Flow steps 5 through 7. Once step 7 is finished, user clicks on a button with an arrow icon at the top right of the schedule, another blank template will appear for the user to fill out.	
	<b>4.1.2.</b> Once all the shifts that can be added for the schedule to be valid are added (minimum for all 4 weeks in a month), Activate Schedule button becomes enabled.
	<b>4.1.3.</b> Continue from Normal Flow, Step 8

**Alternate Path 8.1**  
User clicks on Clear button

Actor	System
<b>LOOP</b>	
	<b>8.1.1.</b> System removes all shifts from any applicable weeks that have been edited.
	<b>8.1.2.</b> Presents user with notifications saying that the available template has been cleared.
	<b>8.1.3.</b> Continue from Normal Flow, Step 3.

## 6.7 Manager Viewing All Schedules

**Description:**

The managers may want a snapshot of all the schedules that are being created / produced within a certain month, as well as quickly accessible links to any period of schedules from the current time forward. A manager being anyone with a specific level of permissions.

**Pre Condition:**

The user must be successfully logged into the system, presented the dashboard screen and is at a permission level where User can read current and future schedules and has employees that can create schedules under their level. User must have a permission level greater than 0 at a minimum.

**Actor:**

An employee for the company that may need confirmation, access to, or a general overview of the business' scheduling specifics

### Normal Flow

Actor	System
1. User clicks on the Administration tab.	
2. Click the View Schedules button	
	3. System checks the User's permission level and fetches all of the employees with a lower level than User.
	4. A clickable miniature representation image of the schedule is shown on the page in a grid for each employee found in Step 3 for the current date in a widget.
5. User views the schedule representations they want to examine. (A 5.1, A 5.2)	

Actor	System
	<p><b>6.</b> If the User wants to view a schedule in greater detail, they will click on the representation schedule to display it in the widget in full detail, as the creator would view it.</p>
	<p><b>7.</b> The other schedule representations are moved to a short horizontal widget below the detailed view of the currently viewed schedule. They appear as boxes that have tooltips when the mouse is hovered over them or they are given focus by usually pressing the Tab key.</p>
	<p><b>8.</b> [Optional] Clicking on a schedule representation in the smaller widget below the schedule displaying widget will swap the currently viewed schedule with the clicked-on schedule, so the selected schedule will be viewed in full detail.</p>

#### **Alternate Path 5.1**

Different Date is selected to view

Actor	System
<b>5.1.1.</b> User clicks the Next arrow located above and to the right of the main schedule display widget to view the next month of schedules.	
	<b>5.1.2.</b> New schedule representation images are displayed and the old ones are removed from the widget.

**Alternate Path 5.2**

Different Date is selected to view

Actor	System
<b>5.2.1.</b> User clicks the Month Drop Down Menu and selects a month.	
<b>5.2.2.</b> User types in a year to display. (E1, E2)	
	<b>5.2.3.</b> New schedule representation images are displayed and the old ones are removed from the widget.

**Error Path 1**

Number is not entered into the year text field

Actor	System
	<b>1.1</b> Any non-number detected input will be ignored.

**Error Path 2**

No schedules are found for the year entered

Actor	System
	<b>2.1</b> No schedule representation images are displayed. Text is displayed stating that no schedules exist for that year while displaying the current year.

## 6.8 Book Days Off

**Description:**

How an employee should go about booking days off in advance. Unlike emergency requests which is to accomidate rare and difficult circumstances, this requires a supervisor's permission level or greater to be granted or denied. There are a pre-defined number of days, weeks, or time periods that can be booked off.

**Pre Condition:**

The user must be successfully logged into the system, presented the dashboard screen and have permissions to book days off. The user must also have a supervisor for this to have any effect. Schedules cannot be accessed previous to the current date plus the buffer period for requesting days off as specified by the system permission set.

**Post Condition:**

Schedules are modified, supervisor is notified.

**Actor:**

Employee who logs into the system can have any permission set as long as they have a supervisor to approve of the request.

**Normal Flow**

Actor	System
1. Click the "Users" tab on the dashboard. 2. Click the "Days Off" item on the sidebar.	
	3. Displays the number of days off that the user can take (There is a limit imposed every year-month- or week) and a schedule starting at the next available day to book off (as specified by system, defaulted to 2 week wait if not set).

Actor	System
4. Selects a period of time by clicking on a day object in the schedule for the number of times the day off counter is above 0. Clicking on a day that is already selected will deselect it and add back to the day off counter. No partial days can be booked off. If the user wants to work a partial day, they will have to ask a supervisor to edit their current schedule to accompany the request.	
6. User clicks the Confirm button. (A 6.1)	5. For every booked off day, a day off counter is removed ( -1 ) (A 5.1)
	7. Is shown details of selection (in a list) of days booked off and asked to confirm again. (A 6.1)
	8. System notifies supervisor of the book off requests (A 8.1) and supervisor accepts the request.
	9. Schedule is modified.

**Alternate Path 5.1**  
No counters, no more days off to book

Actor	System
5.1.1. Clicks on a day off when counter is at 0	5.1.2. Nothing happens.

**Alternate Path 6.1**  
Clear off booked days

Actor	System
6.1.1. Clicks Clear button	6.1.2. Any days off are deselected and counters restored for every day.

**Alternate Path 8.1**  
Supervisor rejects the request

Actor	System
	<b>8.1.1.</b> Counters are returned to the user, schedule is not modified.
	<b>8.1.2.</b> Notification is sent to user with an optional explanation as to why their request was refused.

## 6.9 Request Shift Change

**Description:**

An employee wants to get rid of a shift, or trade a shift because the shift is at an inconvenient time. Instead of booking the day off, leaving the business short-staffed, or if employee is unable to book it off as the due date is past, they can ask their fellow employees to take the shift instead. A supervisor has permissions to revert the changes if they do not want the changes to take place. The supervisors supervisor has the ability to overrule that decision, to keep a reign on lower authority supervisors in case decisions are found to be biased.

**Pre Condition:**

The user must be successfully logged into the system, presented the dashboard screen and have permissions to request the shift be taken. Only an employee with the same permission set or higher can accept a request.

**Post Condition:**

Schedule is updated with the involved employees according to the shift(s) being modified.

**Actor:**

Employee who logs into the system has a job they are working. Higher permission set employees may have trouble finding replacements as generally there are less high-position workers in a company than lower-position workers.

**Normal Flow**

Actor	System
1. User clicks on the Users tab	
2. User clicks on the Request Shift Change link in the sidebar.	
3. User selects one shift from their schedule (shown in a widget) that they want to trade.	
4. User clicks Confirm. (A 4.1)	
	5. The request for the shift change is placed in the Shift Exchange Pool, a list of all the shifts that employees want to get rid of.
	6. System waits for another employee to accept the proposed shift. (A 6.1)
	7. Shift is accepted from another employee, notifications are sent to Users supervisor (A 7.1)
	8. Schedules of both employees are changed. (A 8.1)

### **Alternate Path 4.1**

User requested specific employee to change shifts with.

<b>Actor</b>	<b>System</b>
	<b>4.1.1.</b> List is populated of people that have the same job type that User works.
<b>4.1.2.</b> User clicks on an employee name (which is a link) .	
	<b>4.1.3.</b> Displays information on the request and asks for confirmation. (A 4.1.3.1., A 4.1.4.1.)
<b>4.1.4</b> Clicks Confirm to send request.	
	<b>4.1.5.</b> Looks up employee and sends request to them.
	<b>4.1.6.</b> Notification is sent to supervisor with option to revert the request.
	<b>4.1.7.</b> Resume at Normal Flow Step 8.

### **Alternate Path 4.1.3.1.**

Cancel pending shift request to employee

<b>Actor</b>	<b>System</b>
<b>4.1.3.1.1.</b> Clicks on “Users” tab.	
<b>4.1.3.1.2.</b> Clicks on Request Shift Change link in sidebar.	
	<b>4.1.3.1.3.</b> Pending Request in progress will be shown in sidebar below Request Shift Change link.
<b>4.1.3.1.4.</b> Click on Pending Request link.	
	<b>4.1.3.1.5.</b> Displays information on the pending request and a button to drop request.
<b>4.1.3.1.6.</b> Clicks “Drop Request“ button.	
	<b>4.1.3.1.7.</b> Notification is sent to the employee mentioning that a shift request was cancelled by User.

**Alternate Path 4.1.4.1.**

Employee can work as requested by user

Actor	System
	<b>4.1.4.1.1.</b> Requested employee is notified that a shift change request is pending.
<b>4.1.4.1.2.</b> Employee denies request (A 4.1.4.1.2.1.).	
	<b>4.1.4.1.3.</b> Notification is sent back to user explaining request was denied. (A 4.1.4.1.3.1.).
	<b>4.1.4.1.4.</b> System removes Pending Request.

**Alternate Path 4.1.4.1.2.1.**

Employee accepts request that is sent by User

Actor	System
	<b>4.1.4.1.2.1.1.</b> Employee accepts request to take Users shift.
	<b>4.1.4.1.2.1.2.</b> Notification of acceptance is sent to user.
	<b>4.1.4.1.2.1.3.</b> Schedule of both users are modified accordingly.

**Alternate Path 4.1.4.1.3.1.**

Employee silently ignores the request

Actor	System
	<b>4.1.4.1.3.1.1.</b> Employee requests to have request ignored.
	<b>4.1.4.1.3.1.2.</b> Shift is seen as pending until time for shift approaches. Unless User cancels the request and gets someone else to work it, they will be expected to work that shift.

**Alternate Path 6.1.**

No user accepts the shift in time (either request pool or specific request)

Actor	System
	<b>6.1.1.</b> Notification is sent to User saying that the shift could not be traded.
	<b>6.1.2.</b> Remove the pending request.

**Alternate Path 7.1.**

Employee and User have different supervisors

Actor	System
	<b>7.1.1.</b> Notifications are sent to both supervisors. (A 8.1.)

**Alternate Path 8.1.**

Supervisor reverts decision to have shift change

Actor	System
	<b>8.1.1.</b> Supervisor is notified of the exchange and clicks on the Reverse Decision button.
	<b>8.1.2.</b> Employees that had their schedules modified have changes reverted.
	<b>8.1.3.</b> Notifications are sent to both employees informing them that the schedules are reversed.
	<b>8.1.4.</b> Supervisor has information displayed that encourages them to contact the employees to explain why decision was reversed.
	<b>8.1.5.</b> Notification is sent to supervisor notifying them that a decision was overruled. (A 8.1.5.1.)

**Alternate Path 8.1.4.1.**

Supervisor's Supervisor reverts the previous decision

Actor	System
	<b>8.1.4.1.1.</b> Involved employees are sent a notification that mentions their shifts have been changed yet again.
	<b>8.1.4.1.2.</b> Schedules are changed.
	<b>8.1.4.1.3.</b> Notification is sent back to supervisor explaining that their decision was overturned.

## 6.10 Distribute Reports

### Description:

Once a report has been generated, it can be sent to others as an electronic copy through email, or it can be printed and distributed by paper.

### Pre Condition:

A report has been generated and is displayed on the screen.

### Post Condition:

A report has been printed or emailed, and the user has returned to the report view.

### Actor:

An Employee who has Supervisor permissions.

### Normal Flow

Actor	System
1. User clicks distribute report button.	2. System displays distribution option list.
3. User clicks the print report link. (A 3.1)	4. Generate print-friendly version of the report. 5. Display report on screen. 6. Initiate web browser's print action.
7. Prints report (Browser functions outside project scope)	
8. User clicks return to report view link.	9. Reload report view.

**Alternate Path 3.1**  
E-Mail report link selected

Actor	System
	<b>3.1.1.</b> Display email entry field.
<b>3.1.2.</b> User enters e-mail address.	<b>3.1.3.</b> System verifies that email is in proper format. (E 1)
	<b>3.1.4.</b> Enable Send Emails button.
<b>3.1.5.</b> Press send Emails button.	<b>3.1.6.</b> Send report to listed emails.
	<b>3.1.7.</b> Display emails sent successfully.
<b>3.1.8.</b> Clicks OK Button.	<b>3.1.9.</b> Use case resumes at step 9.

**Error Path 1.1**  
Entered email(s) detected as invalid

Actor	System
	<b>1.1.</b> Highlight invalid emails
	<b>1.2.</b> Request that user corrects or removes invalid emails.
<b>1.3.</b> Correct email addresses.	<b>1.4.</b> Use Case resumes at Step 3.1.3.

## 6.11 Generate Reports

### Description:

Supervisors can generate reports give information on schedules, employees and their usage of the system.

Schedule Reports list information on previously implemented schedules, or future schedules that have yet to take effect.

Employee Resource Reports list information such as employees that are free during a given time, or future timeslots that have yet to be filled.

Employee Usage Reports will allow supervisors to see statistics on how employees are using the system. This will include information such as when they last viewed their schedule, which will allow supervisors to see who might not yet know about changes that have been made.

### Pre Condition:

The supervisor has successfully logged in; Dashboard is displayed.

### Post Condition:

A specific report is displayed, and can be printed or emailed if desired.

### Actor:

An Employee who has Supervisor permissions.

### Normal Flow

Actor	System
1. User clicks button to enter report view.	2. Display front page of report view, listing report types.
3. User selects Schedule Report, enters date range for report. (A 3.1, A 3.2)	4. Collect schedule data.
	5. Generate schedule data.
	6. Display report.

**Alternate Path 3.1**

Employee Resource Report selected

Actor	System
	<b>3.1.1.</b> Collect employee and employee's shift data.
	<b>3.1.2.</b> Generate resource report.
	<b>3.1.3.</b> Use case continues at Step 6.

**Alternate Path 3.2**

E-Mail Employee Usage Report selected

Actor	System
	<b>3.2.1.</b> Collect employee and employee's schedule data.
	<b>3.2.2.</b> Generate usage report.
	<b>3.2.3.</b> Use case continues at Step 6.

## 6.12 Maintain Workgroups

**Description:**

Managers are able to create, edit and delete employee workgroups. When a new workgroup is created, the manager may choose to assign employees to the workgroup, or leave it empty for employees to be added at a later date.

**Pre Condition:**

The manager has successfully logged in; Dashboard is displayed.

**Post Condition:**

The manager has finished creating/editing/deleting a workgroup, and has returned to the available workgroups list.

**Actor:**

An Employee who has Manager permissions.

**Normal Flow**

Actor	System
1. User clicks the Administration tab.	
2. User clicks maintain workgroups link in sidebar.	3. Load maintain workgroups widget. 4. Populate available workgroups list.
5. Presses create new workgroup button. (A 4.1, A 4.2)	
	6. Creates new workgroup. 7. Populate available employees list. 8. Display edit workgroup view.
9. Enters new workgroup name.	
10. Selects employees to add to workgroup.	
11. Clicks add employees to workgroup button.	
	12. Show dialog requesting confirmation and displaying employee info.
13. Clicks accept changes button (A 13.1.)	
	14. Save workgroup.
	15. Return to available workgroups list.

**Alternate Path 4.1**  
Manager edits workgroup

Actor	System
4.1.1. Selects workgroup from available workgroup list.	
4.1.2. Clicks edit workgroup button.	
	4.1.3. Load selected workgroup.
	4.1.4. Populate available employees list.
4.1.5. Selects employees to add to workgroup. (A 4.1.4.1.)	
4.1.6. Clicks add employees to workgroup button.	
	4.1.7. Use case resumes at Step 11.

**Alternate Path 4.1.4.1.**  
Manager edits workgroup

Actor	System
4.1.4.1.1. Clicks remove employees button.	
	4.1.4.1.2. Use case resumes from Step 11.

**Alternate Path 4.2.**  
Manager deletes workgroup

Actor	System
4.2.1. Selects workgroup from available workgroup list.	
4.2.2. Clicks delete workgroup button.	
	4.2.3. Use case resumes from Step 11.

**Alternate Path 12.1.**

Manager clicks cancel changes button.

Actor	System
	<b>12.1.1.</b> Undo changes, deselect employees.
	<b>12.1.2.</b> Use case resumes from Step 9.

## 6.13 Send Announcement

**Description:**

Custom announcements can be created that will be sent to specific employees, employee types, or all employees working under the supervisor.

**Pre Condition:**

The supervisor has successfully logged in; Dashboard is displayed.

**Post Condition:**

The supervisor has finished sending an announcement, and is able to send another or return to the dashboard.

**Actor:**

An Employee who has Supervisor permissions.

### Normal Flow

Actor	System
1. Clicks send announcement button.	2. Load announcement widget.
	3. Display announcement type selection view.
4. Presses workgroup announcement button. (A 4.1)	5. Populate and display list of workgroups for supervisor.
6. Selects target workgroup from list.	
7. Enters announcement message.	
8. Presses send button.	9. Send announcement to all employees within selected workgroup.
	10. Display messages sent alert window.
11. Clicks ok button on alert window.	12. Return to announcement type selection view.

**Alternate Path 4.1.**  
Presses employee type announcement button

Actor	System
	<b>4.1.1.</b> Populate and display list of employee types for supervisor.
<b>4.1.2.</b> Selects target employee type.	
<b>4.1.3.</b> Enters announcement message.	
<b>4.1.4.</b> Presses send button.	
	<b>4.1.5.</b> Send announcement message to all employees within selected employee type.
	<b>4.1.6.</b> Use case resumes from Step 9.

## 6.14 View Workgroup Schedule

**Description:**

After entering the schedule view, supervisors can select to view a schedule for a specific workgroup of employees. Weekly schedules are displayed by default, which can be switched to a daily schedule if desired.

**Pre Condition:**

The supervisor has successfully logged in; Dashboard is displayed.

**Post Condition:**

The schedule for a specific workgroup is displayed.

**Actor:**

An Employee who has Supervisor permissions.

**Normal Flow**

Actor	System
1. Clicks schedules button to enter schedule view. (A 1.1)	
	2. Get schedule data on super workgroup (all employees) for supervisor.
	3. Populate workgroup dropdown list. (E 1)
	4. Display weekly schedule view of super workgroup for current date, listing workgroups in dropdown box.
5. Selects workgroup from dropdown box.	
	6. Reload schedule view, showing employees only from selected workgroup.
	7. Get schedule data for employees in the selected workgroup for the given time period.
	8. Display workgroup schedule.

**Alternate Path 1.1.**

Daily schedule selected

Actor	System
	1.1.1. Get daily schedule data for employees in selected workgroup only.
	1.1.2. Use case resumes from Step 1.

**Error Path 1.1.**

No additional workgroups exist

Actor	System
	<b>1.1.1.</b> Disable workgroup dropdown list.
	<b>1.1.2.</b> Use case resumes at Step 4.

## 6.15 Maintain Locations

**Description:**

Supervisors are able to create, edit and delete locations. Once created, employees can choose or be assigned a preferred location, used for automated scheduling.

**Pre Condition:**

The manager has successfully logged in; Dashboard is displayed.

**Post Condition:**

The manager has finished creating/editing/deleting a location, and has returned to the dashboard.

**Actor:**

An Employee who has Supervisor permissions.

**Normal Flow**

Actor	System
1. Clicks maintain locations button.	2. Load maintain locations widget.
	3. Display location list view, populating location list.
4. Clicks add location button. (A4.1)	5. Display empty edit location view.
6. Enters location name and description.	
7. Clicks save button.	8. Confirm location does not already exist. (E1)
	9. Save location in database
	10. Display location saved alert.
11. Clicks OK button on alert	
	12. Display location list view, populating location list.
13. Clicks return to dashboard link.	

**Alternate Path 4.1.**  
Selects existing location

Actor	System
<b>4.1.1.</b> Clicks edit location button. (4.1.1.1)	
	<b>4.1.2.</b> Display edit location view.
	<b>4.1.3.</b> Populate fields with data from selected location.
<b>4.1.4.</b> Changes location name or description.	
<b>4.1.5.</b> Clicks save button.	
	<b>4.1.6.</b> Use case resumes from step 9.

**Alternate Path 4.1.1.1.**

No additional workgroups exist

Actor	System
	<b>4.1.1.1.1.</b> Display confirm delete dialog.
<b>4.1.1.1.2.</b> Clicks ok button on dialog.	<b>4.1.1.1.3.</b> Remove location from database.
	<b>4.1.1.1.4.</b> Display location removed alert.
	<b>4.1.1.1.5.</b> Use case resumes from step 11.

**Error Path 1**

Location already exists

Actor	System
	<b>1.1</b> Display location already exists alert.
<b>1.2.</b> Clicks ok button on dialog.	<b>1.3.</b> Use case resumes from step 6.

## 6.16 Maintain Positions

**Description:**

Supervisors are able to create, edit and delete positions. Once created, employees can choose or be assigned a preferred position, used for automated scheduling.

**Pre Condition:**

The manager has successfully logged in; Dashboard is displayed.

**Post Condition:**

The manager has finished creating/editing/deleting a position, and has returned to the dashboard.

**Actor:**

An Employee who has Supervisor permissions.

**Normal Flow**

Actor	System
1. Clicks maintain positions button.	2. Load maintain positions widget.
	3. Display position list view, populating position list.
4. Clicks add position button. (A4.1)	5. Display empty edit position view. Populate skill list.
6. Enters position name and description. Chooses required skills.	
7. Clicks save button.	8. Confirm position does not already exist. (E1)
	9. Save position in database
	10. Display position saved alert.
11. Clicks OK button on alert	
	12. Display position list view, populating position list.
13. Clicks return to dashboard link.	

**Alternate Path 4.1.**  
Selects existing position

Actor	System
<b>4.1.1.</b> Clicks edit position button. (4.1.1.1)	
	<b>4.1.2.</b> Display edit position view. <b>4.1.3.</b> Populate fields with data from selected position. Populate Skill List
<b>4.1.4.</b> Changes position name, description or required skills.	
<b>4.1.5.</b> Clicks save button.	
	<b>4.1.6.</b> Use case resumes from step 9.

**Alternate Path 4.1.1.1.**  
Clicks delete position button.

Actor	System
	<b>4.1.1.1.1.</b> Display confirm delete dialog.
<b>4.1.1.1.2.</b> Clicks ok button on dialog.	<b>4.1.1.1.3.</b> Remove position from database.
	<b>4.1.1.1.4.</b> Display position removed alert.
	<b>4.1.1.1.5.</b> Use case resumes from step 11.

**Error Path 1**  
Position already exists

Actor	System
	<b>1.1</b> Display position already exists alert.
<b>1.2.</b> Clicks ok button on dialog.	<b>1.3.</b> Use case resumes from step 6.

## 6.17 Maintain Skills

**Description:**

Supervisors are able to create, edit and delete skills. Once created, skills may be assigned to employees and positions. Skills will be used in the automatic generation of schedules, limiting what positions an employee can be assigned to, and for keeping track of extra skills an employee has outside of their position.

**Pre Condition:**

The manager has successfully logged in; Dashboard is displayed.

**Post Condition:**

The manager has finished creating/editing/deleting a skill, and has returned to the dashboard.

**Actor:**

An Employee who has Supervisor permissions.

**Normal Flow**

Actor	System
1. Clicks maintain skills button.	2. Load maintain skills widget.
	3. Display skill list view, populating skill list.
4. Clicks add skill button. (A4.1)	5. Display empty edit skill view. Populate skill list.
6. Enters skill name and description. Chooses required skills.	
7. Clicks save button.	8. Confirm skill does not already exist. (E1)
	9. Save skill in database
	10. Display position saved alert.
11. Clicks OK button on alert	
	12. Display skill list view, populating skill list.
13. Clicks return to dashboard link.	

**Alternate Path 4.1.**

Selects existing skill

Actor	System
<b>4.1.1.</b> Clicks edit skill button. (4.1.1.1)	
	<b>4.1.2.</b> Display edit skill view.
	<b>4.1.3.</b> Populate fields with data from selected skill.
<b>4.1.4.</b> Changes skill name, description.	
<b>4.1.5.</b> Clicks save button.	
	<b>4.1.6.</b> Use case resumes from step 9.

**Alternate Path 4.1.1.1.**

Clicks delete skill button.

Actor	System
	<b>4.1.1.1.1.</b> Display confirm delete dialog.
<b>4.1.1.1.2.</b> Clicks ok button on dialog.	<b>4.1.1.1.3.</b> Remove skill from database.
	<b>4.1.1.1.4.</b> Display skill removed alert.
	<b>4.1.1.1.5.</b> Use case resumes from step 11.

**Error Path 1**

Skill already exists

Actor	System
	<b>1.1</b> Display skill already exists alert.
<b>1.2.</b> Clicks ok button on dialog.	<b>1.3.</b> Use case resumes from step 6.

# **Chapter 7**

## **Problem Domain Class Diagram**

### **7.1 Problem Domain**

### Class Diagrams



#### Problem Domain

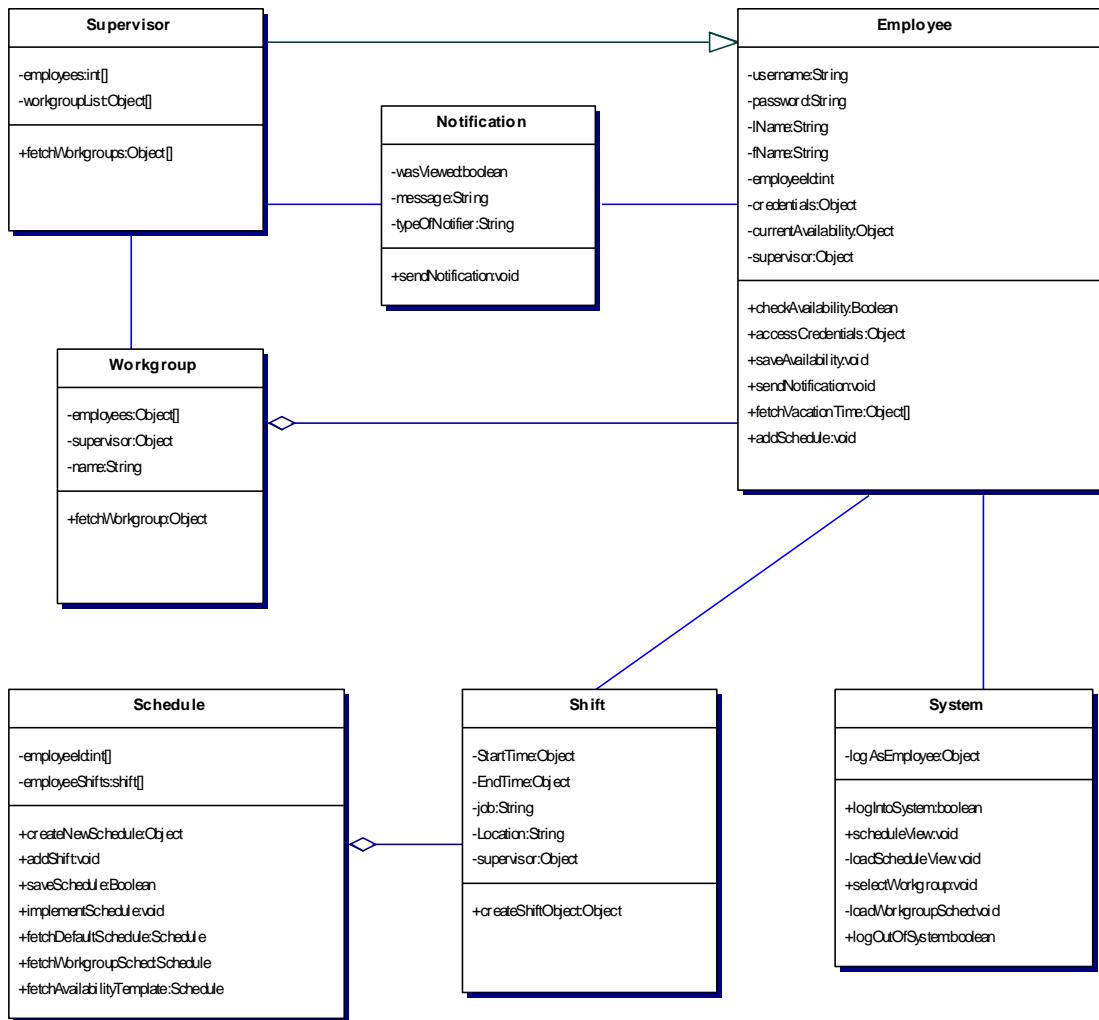


Figure 7.1: **Class Diagram** The classes that are utilized by the system with their respective attributes

# **Chapter 8**

## **State Machine Diagrams**

### **8.1 State Machines**

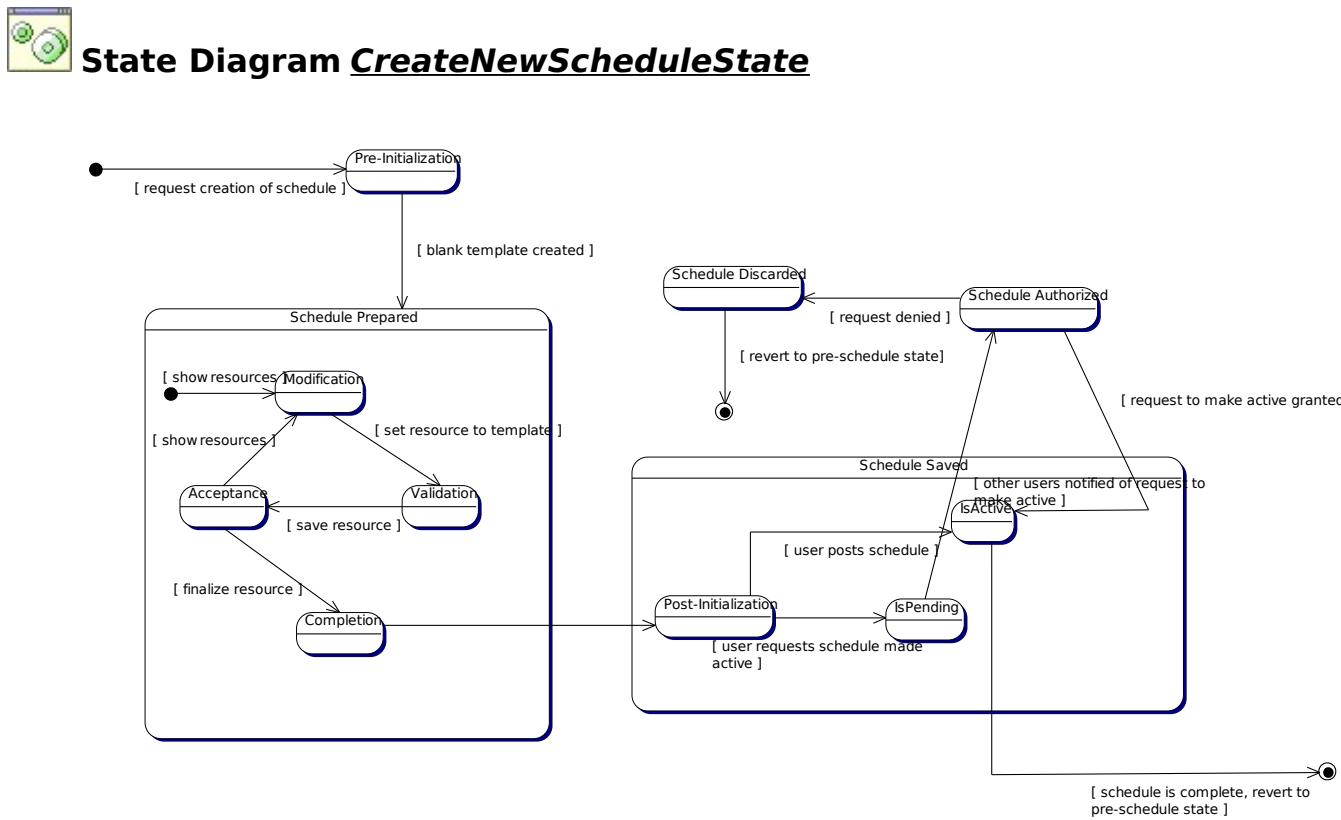


Figure 8.1: **Create New Schedule State Diagram**

A representation of different states the system goes through in creating a new schedule

### **Diagram Contents Summary**

- State Pre-Initialization**
- State Schedule Authorized**
- State Schedule Discarded**
- State Schedule Prepared**
- State Acceptance**
- StartState ActionsEnabled**
- State Completion**
- State Modification**
- State Validation**
- State Schedule Saved**
- State IsActive**
- State IsPending**
- State Post-Initialization**
  - EndState scheduleAvailable**
  - EndState scheduleRefuse**
- StartState StartScheduleCreation**

### **State Detail**

#### **State Pre-Initialization**

This state is when user permissions are checked to ensure they have the right to create a schedule, as well as checking for temporary data that might exist from previous schedule creations that have not been saved. The user is assumed to have schedule creation privileges.

##### **Transition links**

to **State Schedule Prepared**

**Event:**

[ blank template created ]

#### **State Schedule Authorized**

A pending schedule is given approval from users not in the scope of the creator and is made active.

##### **Transition links**

to **State IsActive**

**Event:**

[ request to make active granted ]

to **State Schedule Discarded**

**Event:**

[ request denied ]

#### **State Schedule Discarded**

The saved object of the schedule may have been deleted

##### **Transition links**

to **EndState scheduleRefuse**

**Event:**

[ revert to pre-schedule state]

 **State Schedule Prepared**

The schedule has been initialized so that the user starts off with a blank schedule. If temporary values are found, they can be restored or deleted; that is out of scope for this diagram.

**Substates**

 **State Acceptance**

When a resource is accepted as it is placed in a valid position, it is written to a temporary value in memory which is routinely saved to the backend.

**Transition links**

to [\*\*State Modification\*\*](#)

**Event:**

[ show resources ]

to [\*\*State Completion\*\*](#)

**Event:**

[ finalize resource ]

 **StartState ActionsEnabled**

At this start point, the user is allowed to perform some actions on the blank schedule.

**Transition links**

to [\*\*State Modification\*\*](#)

**Event:**

[ show resources ]

 **State Completion**

The user requests the schedule as complete. The temporary object is saved to the backend, temporary value is deleted, and user begins the implementation process if they have permissions to.

**Transition links**

to [\*\*State Post-Initialization\*\*](#)

 **State Modification**

A resource that is available to place on the schedule can be added, edited, or removed.

**Transition links**

to [\*\*State Validation\*\*](#)

**Event:**

[ set resource to template ]

 **State Validation**

Has a resource been applied when it is not available or sufficient for a task? Any modification that goes against the set of rules that apply to a schedule are undone and not saved.

#### ***Transition links***

to **State Acceptance**

**Event:**

[ save resource ]

#### **State Schedule Saved**

The template has been saved officially so that it is recognized as a schedule that can have actions applied to it (as opposed to modifications and view requests).

#### **Substates**

#### **State IsActive**

Schedule has been made active, so it can be seen by everyone who requests viewing a schedule.

#### ***Transition links***

to **EndState scheduleAvailable**

**Event:**

[ schedule is complete, revert to pre-schedule state ]

#### **State IsPending**

If user permissions require authorization to activate a schedule template, it will be placed in a pending state and users with higher permissions will be notified. Only users that are directly above the user will be automatically notified, although notifications to other higher authorities can be requested, but outside the scope of this diagram.

#### ***Transition links***

to **State Schedule Authorized**

**Event:**

[ other users notified of request to make active ]

#### **State Post-Initialization**

Checks are done for permissions: user may not be able to make the schedule 'active' or may require other users to accept the changes.

#### ***Transition links***

to **State IsPending**

**Event:**

[ user requests schedule made active ]

to **State IsActive**

**Event:**

[ user posts schedule ]

#### **EndState scheduleAvailable**

**backgroundColor:**

0,0,0

#### **EndState scheduleRefuse**

The schedule was refused by higher authority to be made active. User who created the schedule will be notified of its denial

status. A denied schedule may also be deleted by those denying it.

**backgroundColor:**

0,0,0

● **StartState *StartScheduleCreation***

This is the point where the system will start preparing for new schedule creation.

**backgroundColor:**

0,0,0

***Transition links***

to **State Pre-Initialization**

**Event:**

[ request creation of schedule ]



### Request Shift-Change

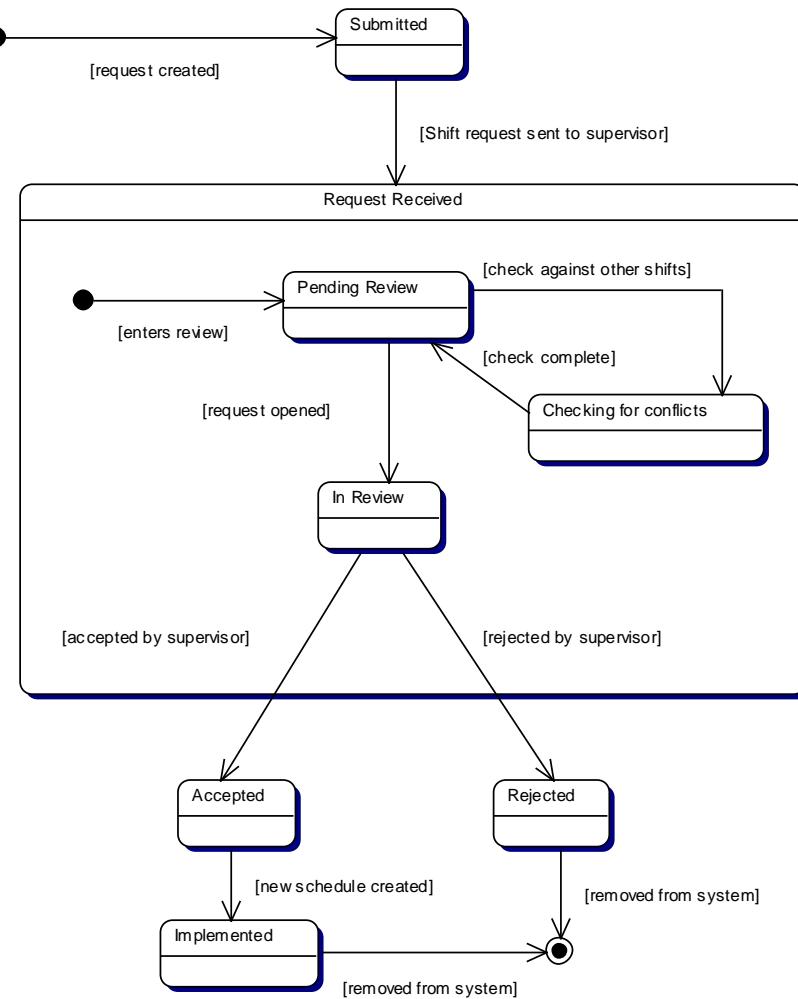


Figure 8.2: **Request Shift Change State Diagram**

A representation of different states the system goes through in allowing a user to request a shfit change

### **State Detail**

#### **State Accepted**

The shift change has been accepted by the supervisor. Notifications will be sent informing the employee of the decision, and the new shift for this employee will await inclusion in a future schedule. The request will persist in the system until it has been included in a schedule.

#### **State Implemented**

The shift change is now included within a future schedule for the employee. Additional notifications will be sent to the employee to remind them that their work hours will be changing at this time.

#### **State Rejected**

The shift change has been rejected by the supervisor. Notifications will be sent informing the employee of the decision, and the shift change will be removed from the system.

#### **State Request Received**

The request has been received by the supervisor's account, and is in the review process.

### **Substates**

#### **State Checking for conflicts**

The shift request is in the process of being checked for any conflicts with any current shifts from other employees. Conflicts include conditions such as the shift being requested not having room for an additional employee.

#### **State In Review**

The shift change is in the process of being reviewed by the supervisor, including the review of any conflicts that the system has detected with the new shift times the employee has requested. It is awaiting a decision on whether or not the shift change will be accepted or rejected.

#### **State Pending Review**

A pending shift change is one that has been sent to the supervisor including any notifications that there is a new employee request for them to view. The supervisor has yet to view the request at this time.

#### **State Submitted**

The shift change request has been submitted by the employee and has been created in the system.

 Request Time Off

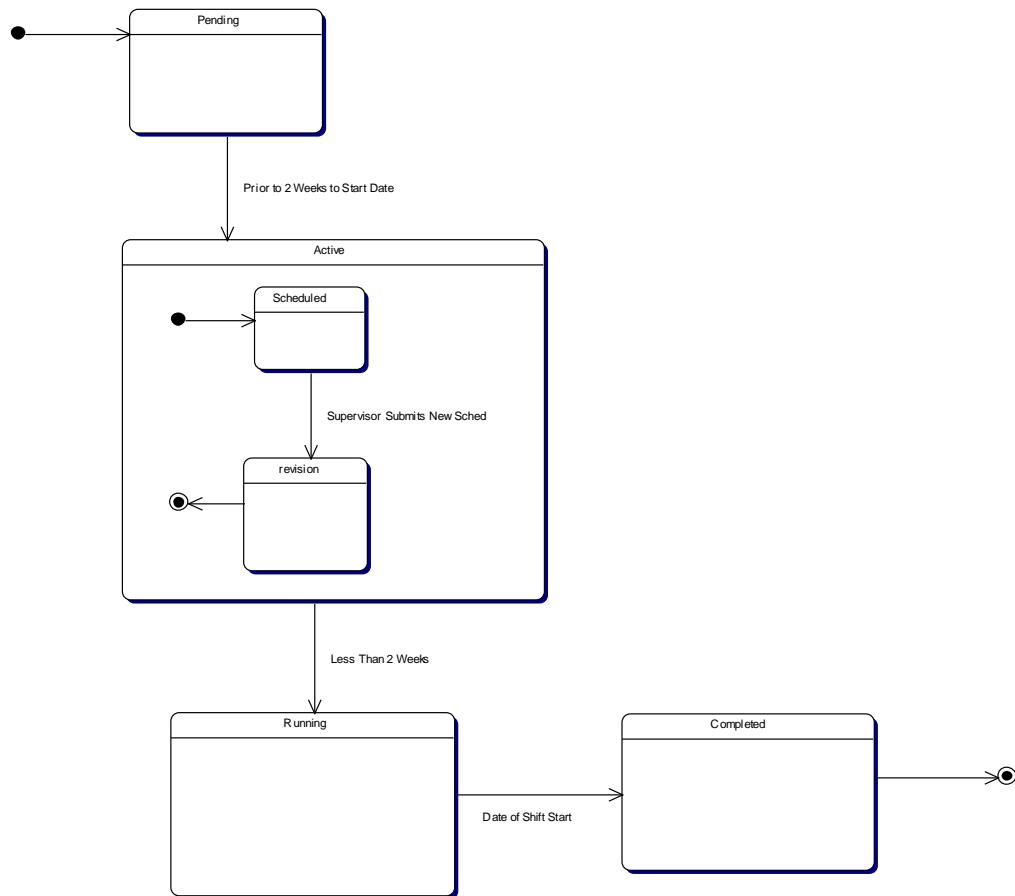


Figure 8.3: **Request Time Off State Diagram**

A representation of different states the system goes through when an employee requests time off

### **State Detail**

#### **State Active**

This state can be achieved by a supervisor taking the shift change request out of the pending state.

#### **Substates**

#### **State revision**

This state can be accessed if a supervisor is viewing a schedule and decides that it should be changed. The state can then proceed to the end of a revision stage by the supervisor sending the modified schedule back.

#### **State Scheduled**

This is an internal state inside of active. This state can be achieved if the schedule is currently being viewed by a supervisor after an employee has submitted the shift change request.

#### **State Completed**

The state of the shift change request goes to completed after the date of the shift change request has passed.

#### **State Pending**

This state is when the shift change request is sent to the supervisor and it is pending approval. This can no longer be changed once the 2 weeks to start date has occurred.

#### **State Running**

A shift change request can be put into a running state after all the changes have been made and a supervisor accepts the shift change request.

# Chapter 9

## Activity Diagrams

## 9.1 Activity Diagrams

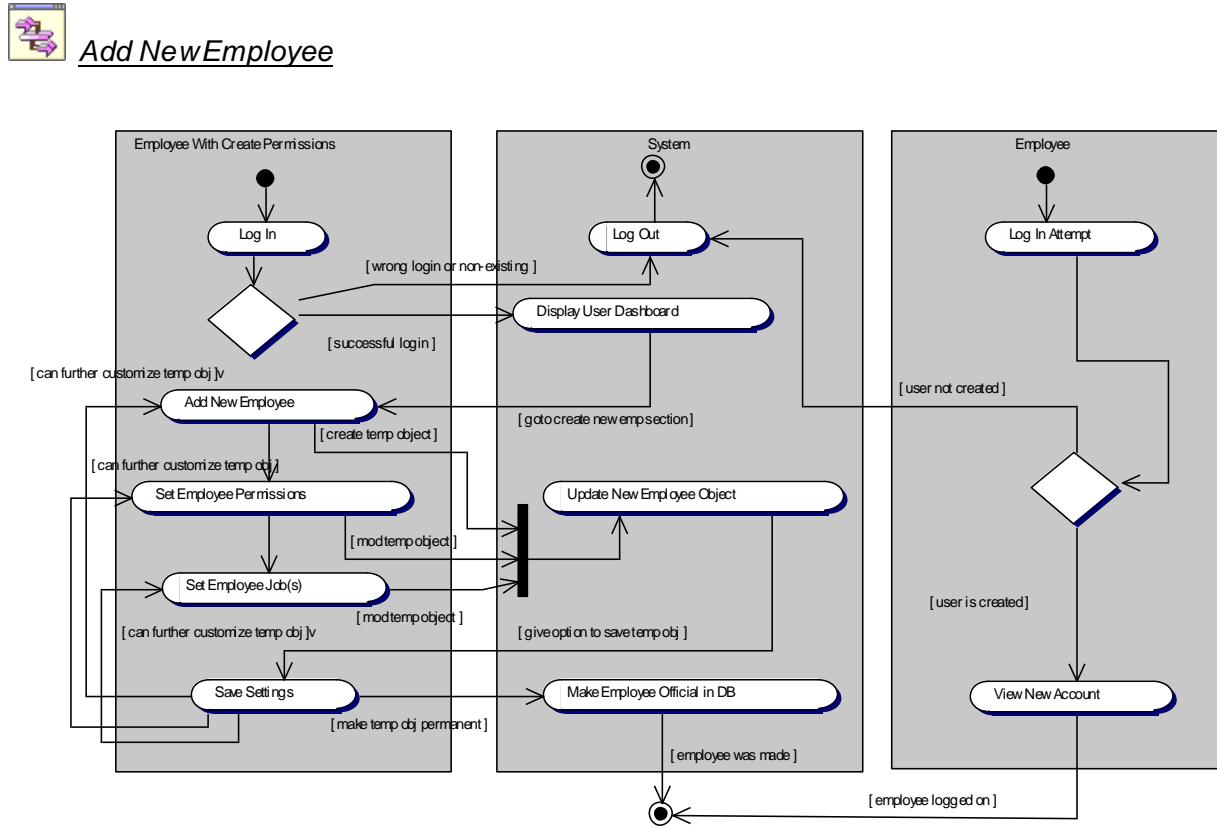
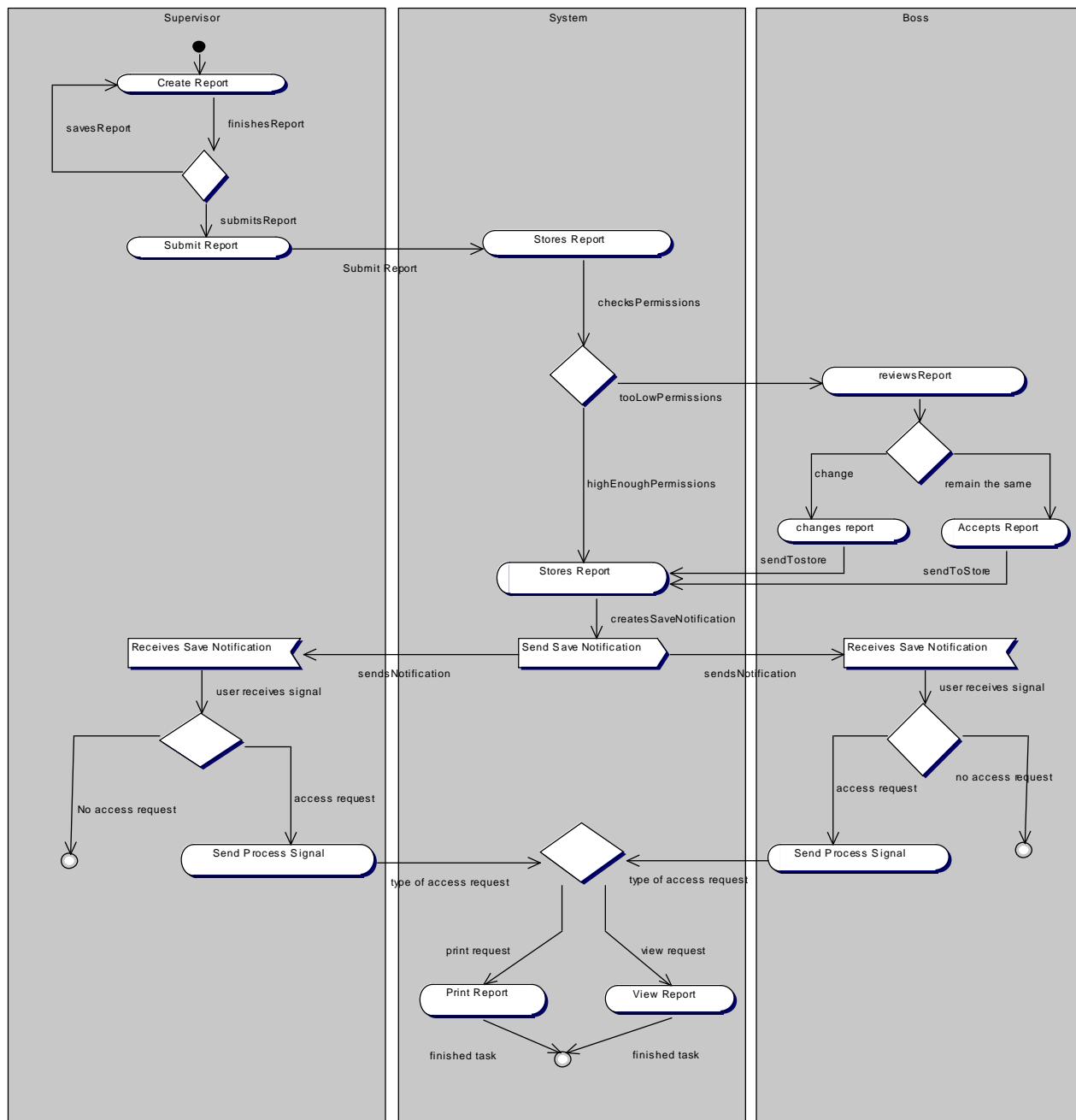


Figure 9.1: **Add New Employee Activity Diagram**

The actions required in order to add a new employee to the system



## Generate Report



**Figure 9.2: Generate Report Activity Diagram**  
The actions required to generate detailed and relevant reports

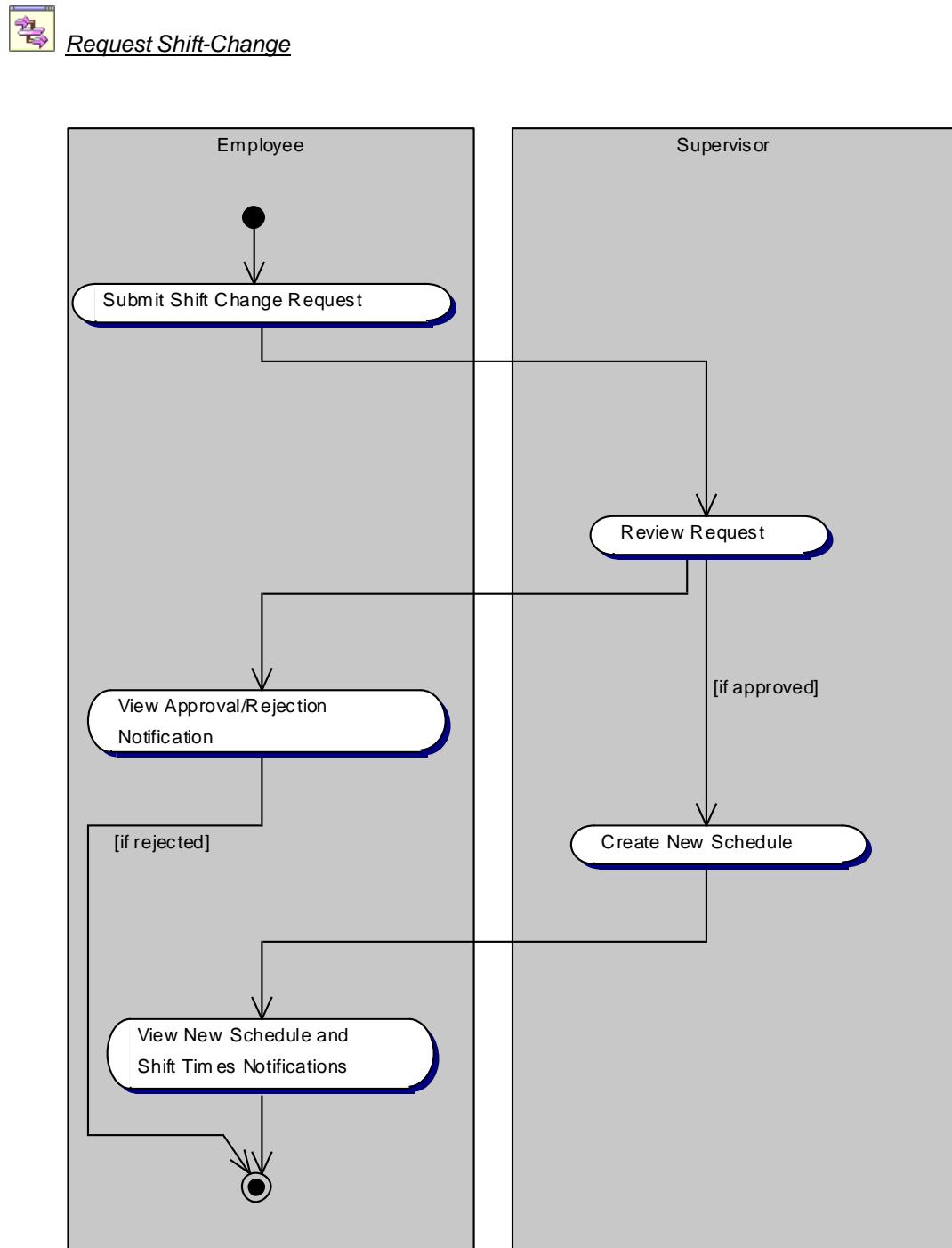


Figure 9.3: **Request Shift Change Activity Diagram**

This diagram shows the status of a shift change request, from being created by an employee, reviewed by a supervisor, and being implemented in future schedules or removed from the system.

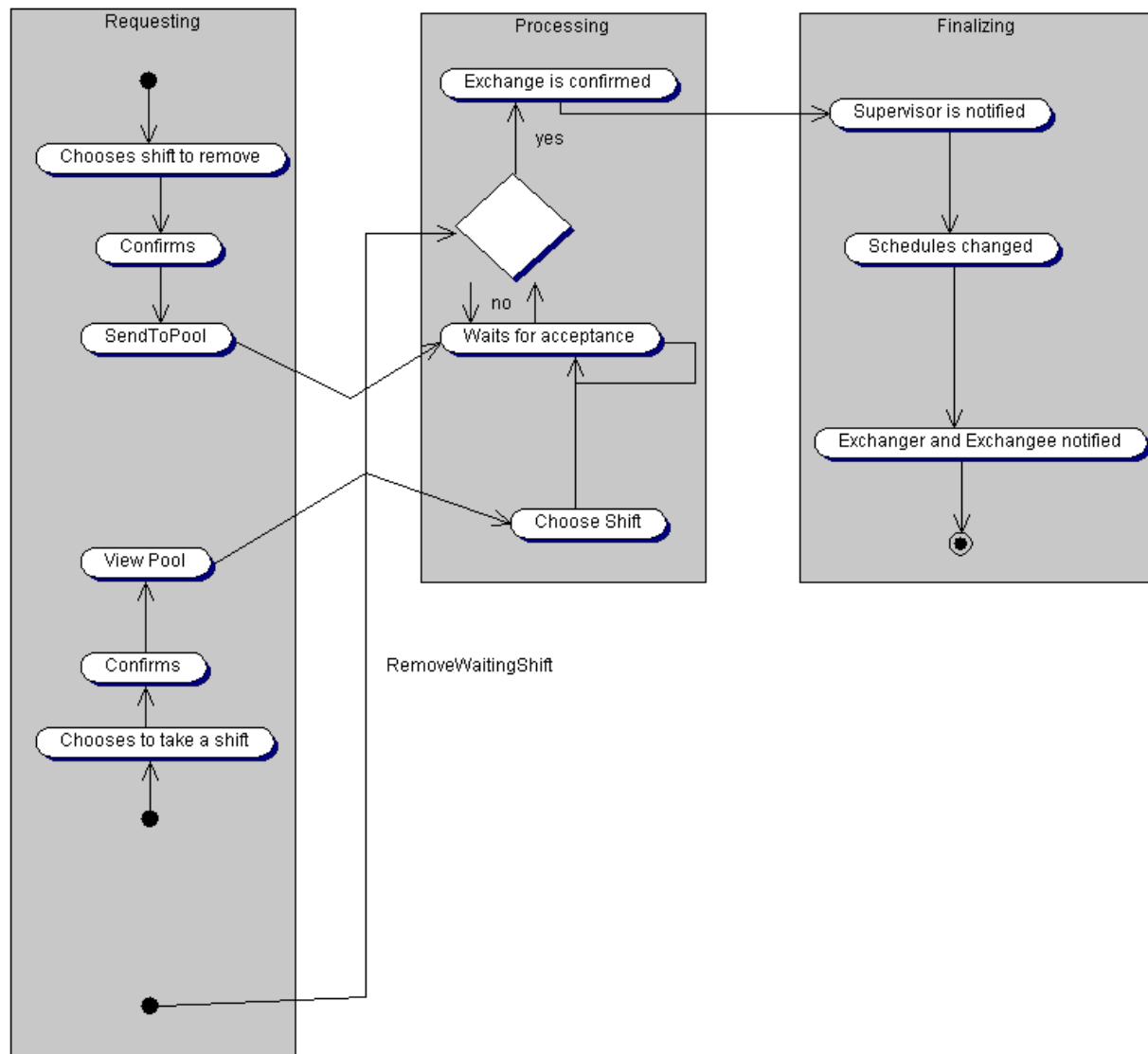


Figure 9.4: Shift Exchange Activity Diagram

This diagram shows the status of a shift being exchanged or given to another employee.

# Part IV

## System Design

# Chapter 10

## Layered Architecture

### 10.1 Package Diagram

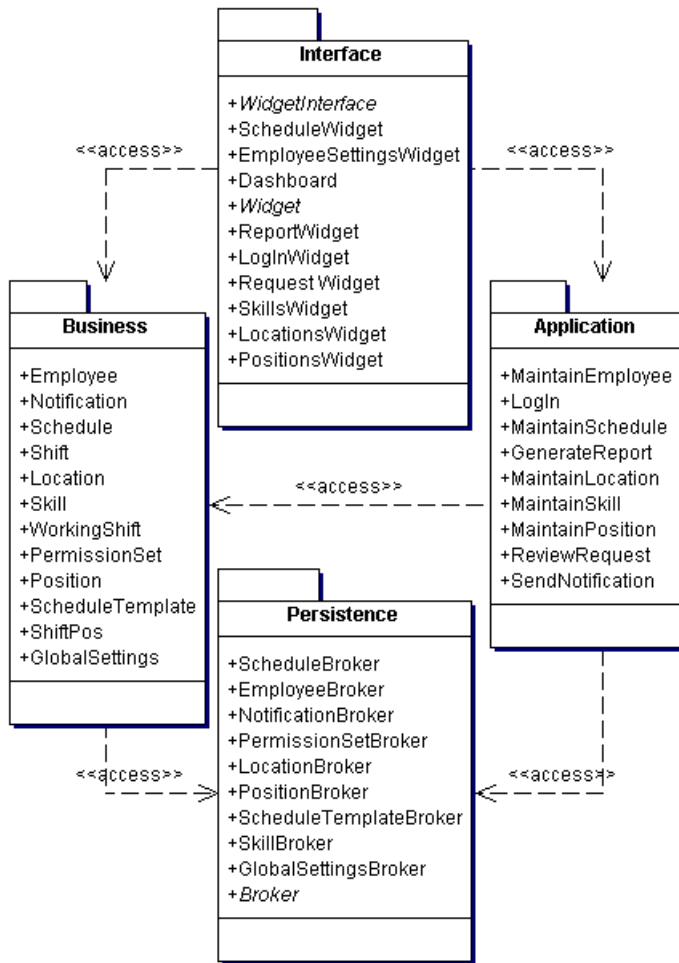


Figure 10.1: **Package Diagram**

This diagram shows all the packages and objects included in the system

# Chapter 11

## Persistence Model

### 11.1 Textual Description Document

#### Control Classes

Our system will have control classes and brokers that will be in charge of directly interacting with the database. The control pattern will implement the application logic layer of our system. Any insertion, deletion or updating of data occurs with the assistance of these classes. The control classes are also used to perform any interaction with the interface and problem domain layers. Basically they are the link between the physical data in the database and what the user sees in the end. The broker classes are in place to put forth an application programming interface to allow developers to get data from the database without directly interacting with the data. This is in place to ensure that the data integrity of the database stays consistent throughout system use. Ensuring data integrity is a key aspect of a system with a four-tier architecture. Allowing the developers to access direct data and information from the database without physically letting them grab the data is the key point of having the control classes.

The interface layer is implemented in the interface classes which will be used to communicate with the users in an easy to use fashion. The interface classes will then interpret the user interaction and send that parsed information to the broker classes to get the information that the user requested. From there the interface class will parse the information received from the broker classes back into a format that the user can understand and interact with.

The business layer consists of data that represents real life objects that interact with each other and persistence layer classes.

The persistence classes will be used as direct data from the database. Mostly consists of objects that are persisted into the database as actual data. The persistence classes will be hidden from the developers due to the broker classes. So in fact, the developers will not be able to directly interact with these classes without first going through the broker classes. This again, is to ensure data integrity in the persistence layer and at the physical data layer.

The application layer contains classes that control the flow of how the interface and business objects interact. While the Broker manages back-end processes, application manages front-end processes including validation.

## 11.2 Entity Relationship Diagrams

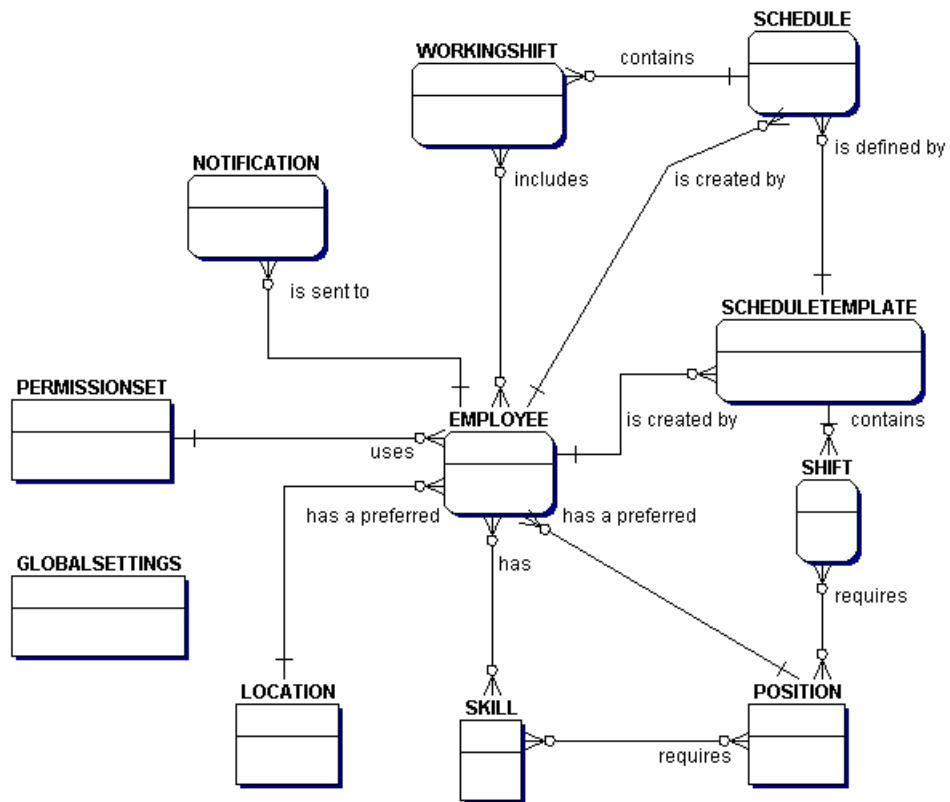


Figure 11.1: Conceptual ERD Diagram

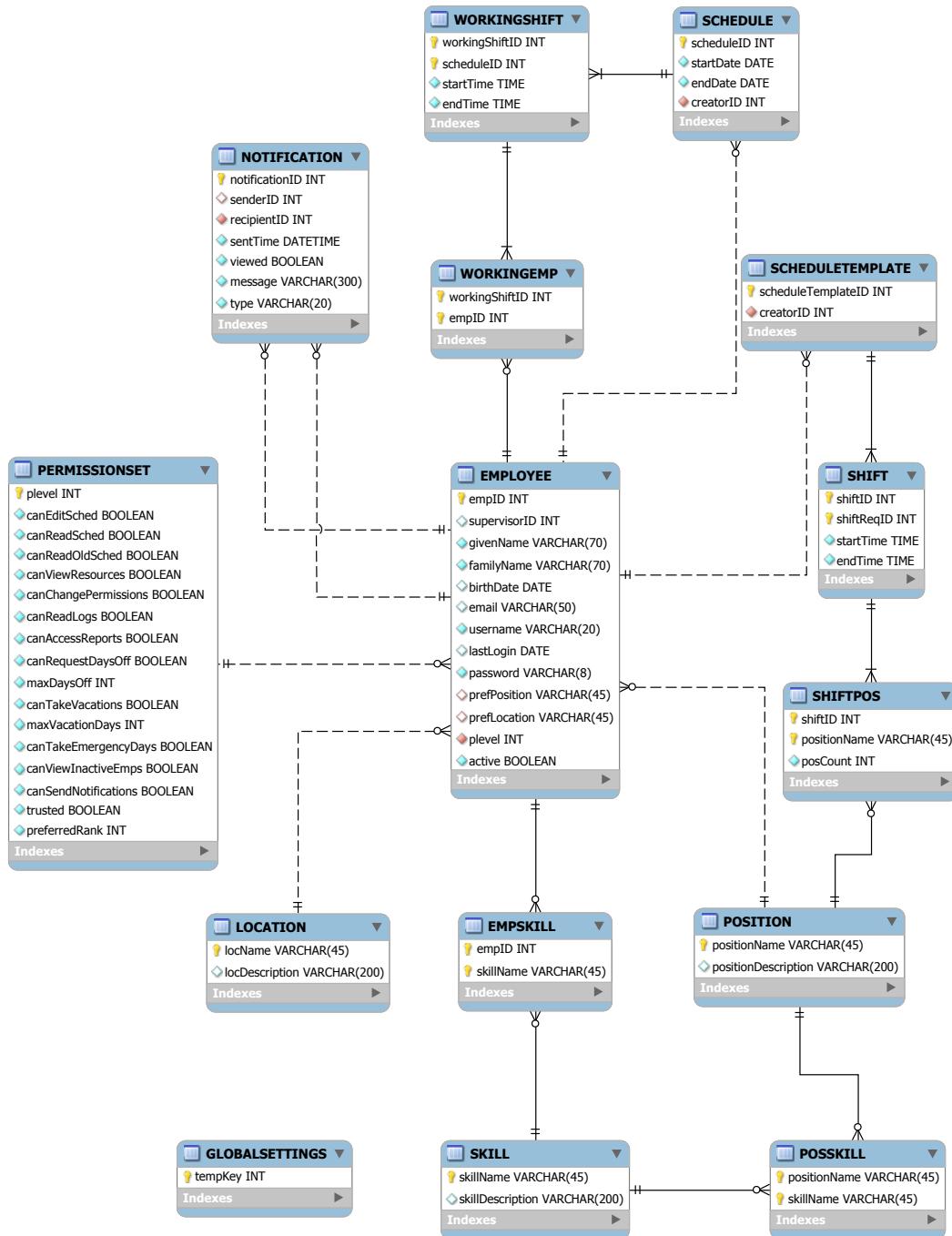


Figure 11.2: Implementation ERD Diagram

# **Chapter 12**

## **Expected Persistent Class Sizes**

### 11.3 Expected Persistent Class Sizes

Table	Attributes	Attribute type	Data size(Bytes)	Total
Employee	*EmpID <sup>i</sup>	INT	4	341
	supervisorID	INT	4	
	givenName	VARCHAR(70)	72	
	familyName	VARCHAR(70)	72	
	birthDate	DATE	3	
	email	VARCHAR(50)	52	
	username	VARCHAR(20)	22	
	password	VARCHAR(8)	10	
	prefPosition	VARCHAR(45)	47	
	prefLocation	VARCHAR(45)	47	
	plevel	INT	4	
	active	BOOLEAN	1	
	lastLogin	DATE	3	

Table	Attributes	Attribute type	Data size(Bytes)	Total
Schedule	*schedID	INT	4	14
	startDate	DATE	3	
	endDate	DATE	3	
	creatorID	INT	4	

Table	Attributes	Attribute type	Data size(Bytes)	Total
PermissionSet	*plevel	INT	4	29
	canReadSched	BOOLEAN	1	
	canReadOldSched	BOOLEAN	1	
	canEditSched	BOOLEAN	1	
	canViewResources	BOOLEAN	1	
	canChangePermissions	BOOLEAN	1	
	canReadLogs	BOOLEAN	1	
	canAccessReports	BOOLEAN	1	
	canRequestDayesOff	BOOLEAN	1	
	maxDaysOff	INT	4	
	canTakeVacations	BOOLEAN	1	
	maxVacationDays	INT	4	
	canTakeEmergencyDays	BOOLEAN	1	
	canViewInactiveEmps	BOOLEAN	1	
	canSendNotifications	BOOLEAN	1	

Chapter 12 - Section:

---

	Trusted	BOOLEAN	1	
	preferredRank	INT	4	

Table	Attributes	Attribute type	Data size(Bytes)	Total
Notification	*notificationID	INT	4	230
	senderID	INT	4	
	recipientID	INT	4	
	viewed	BOOLEAN	1	
	message	VARCHAR(300)	202	
	type	VARCHAR(20)	12	
	sentTime	DATETIME	3	

Table	Attributes	Attribute type	Data size(Bytes)	Total
WorkingShift	*workShiftID	INT	4	14
	scheduleID	INT	4	
	startTime	TIME	3	
	endTime	TIME	3	

Table	Attributes	Attribute type	Data size(Bytes)	Total
Skill	*skillName	VARCHAR(45)	47	249
	skillDescreption	VARCHAR(200)	202	

Table	Attributes	Attribute type	Data size(Bytes)	Total
Position	*positionName	VARCHAR(45)	47	249
	positionDescreption	VARCHAR(200)	202	

Table	Attributes	Attribute type	Data size(Bytes)	Total
EmpSkill	*empID	INT	4	51
	skillName	VARCHAR(45)	47	

Table	Attributes	Attribute type	Data size(Bytes)	Total
PosSkill	*positionName	VARCHAR(45)	47	94
	*skillName	VARCHAR(45)	47	

Table	Attributes	Attribute type	Data size(Bytes)	Total
Location	*locName	VARCHAR(45)	47	249
	locDescreption	VARCHAR(200)	202	

Table	Attributes	Attribute type	Data size(Bytes)	Total
Shift	*shiftID	INT	4	14
	shiftReqID	INT	4	
	startTime	TIME	3	
	endTime	TIME	3	

Table	Attributes	Attribute type	Data size(Bytes)	Total
ShiftPos	*shiftID	INT	4	55
	* positionName	VARCHAR(45)	47	
	posCount	INT	4	

Table	Attributes	Attribute type	Data size(Bytes)	Total
ScheduleTemplate	*scheduleTemplateID	INT	4	8
	creatorID	INT	4	

Table	Attributes	Attribute type	Data size(Bytes)	Total
WorkingEmp	*WorkingShiftID	INT	4	8
	*empID	INT	4	

Table	Attributes	Attribute type	Data size(Bytes)	Total
GlobalSettings	*tempKey	INT	4	4

## Expected Data Size

Expected max number of object is derived from Deerfoot Inn and Casino captured requirements:

### *EXPECTED MAX NUMBER OF STORED OBJECTS*

- **Employee**- Expected to hold 500 active employees. But since disabled employees are not deleted, our system is expected to store disabled employees who have been replaced. On average the total is 800 active and disabled employees. According to the requirements analysis employees are expected to increase on an average of 50 annually.

- **Schedule**- Expected to hold 34 schedules per a week. Since there are 500 employees with 15 in a single work group. Each workgroup has a schedule. In a year the number is expected to grow to 1768 schedules.
- **PermissionSet**- Expected to hold a maximum of 20 different permission sets.
- **Notification**- Expected to hold around 50 notifications per active employee at any given time. Therefore, the total at any given time would be 25, 000 notifications. Notifications are to be deleted every 3 months. Therefore, by the end of a 3 month period there would be a maximum of
- **WorkingShift**- Expected to hold 15 per week. This would grow to 780 by the end of the year.
- **Skill**- Expected to hold 1000 skills for all the positions
- **Position**- Expected to hold 200 positions.
- **EmpSkill**- Expected to hold a maximum of 50 skills per employee. By the end of the year that would grow to 25000.
- **PosSkill**- Expected to hold 3000 since positions will not require more than 15 positions each.
- **Location**- A maximum of 50 locations. This includes 9 restaurants, 2 hotels, casino floor, customer service ...etc
- **Shift** – An employee is expected to cover 12 shifts in 6 days. The number is expected to grow to 624 by the end of the year.
- **ShiftPos**- Expected to hold a maximum of 3120 by the end of the year.
- **ScheduleTemplate**- Expected to hold 52
- **WorkingEmp**- For 6 days, it is expected to hold 3060. This is derived from the multiplication of days by the max number of schedules per week. In a year that would grow to 159 120.
- **GlobalSettings**- With the given information it is not known yet. This value is to be calculated in the next phase.

*EXPECTED DATA SIZE OF STORED OBJECTS*

The following equation had been used to calculate the size:

Expected data size (data size for each persistent class \* expected max number of stored objects of each class)

Each object will approximately have the following data sizes in bytes:

- **Employee**- 170 500
- **Schedule**- 24752
- **PermissionSet**- 580
- **Notification**- 2250000
- **Skill**- 249 000
- **Position**- 49800
- **EmpSkill**- 1275000
- **PosSkill**- 282000
- **Location**- 12450
- **Shift**- 8736
- **ShiftPos**- 171600
- **ScheduleTemplate**- 416
- **WorkingShift**- 10920
- **WorkingEmp**- 1272960
- **GlobalSettings**- With the given information it is not known yet. This value is to be calculated in the next phase.

**Attributes with an asterisk next to them represent primary keys.**

# **Chapter 13**

## **Class Diagram**

### **13.1 Interface, Persistence, Business and Problem Domain**

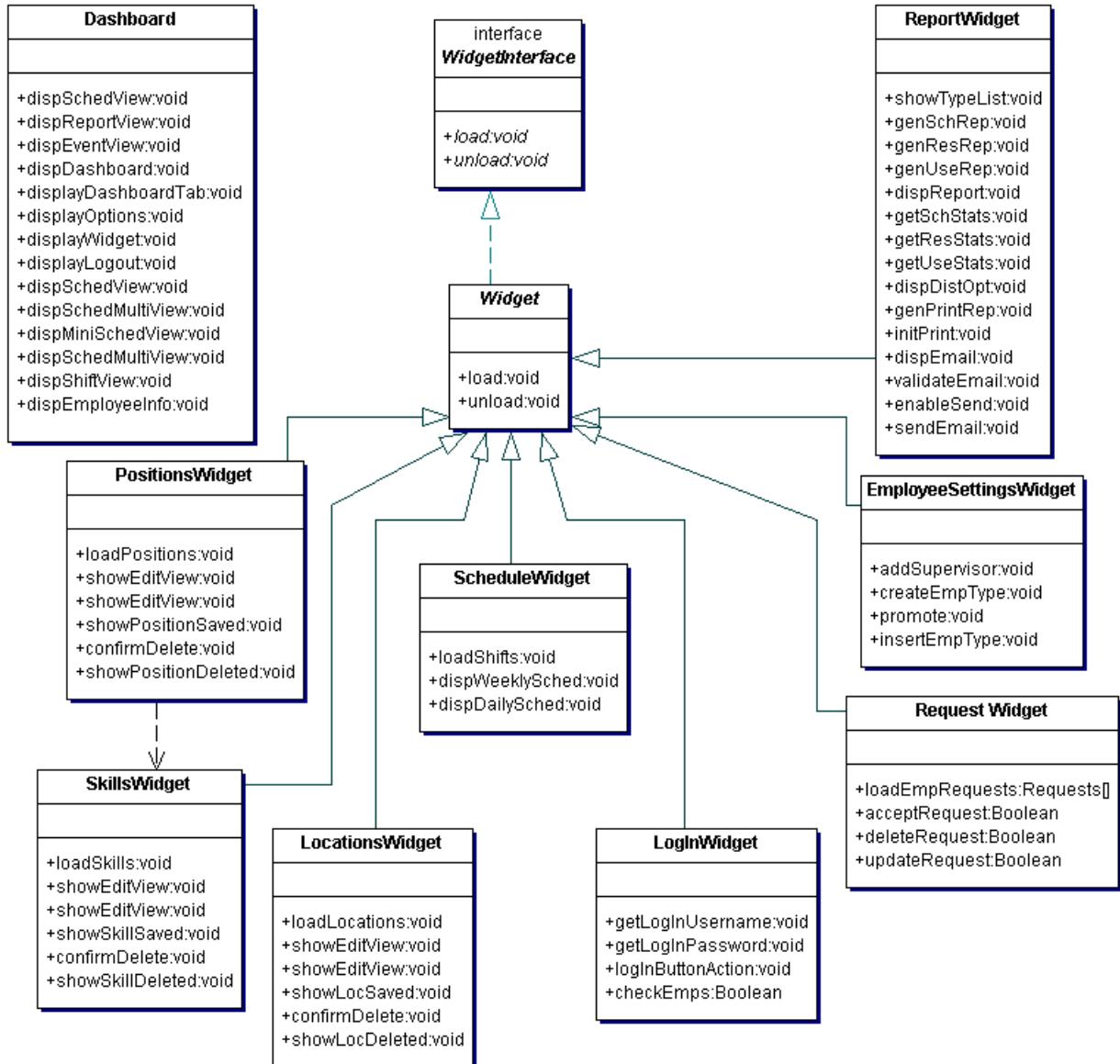


Figure 13.1: Interface Class Diagram

This diagram shows the objects that make up the interface for WebAgenda and methods associated with them.

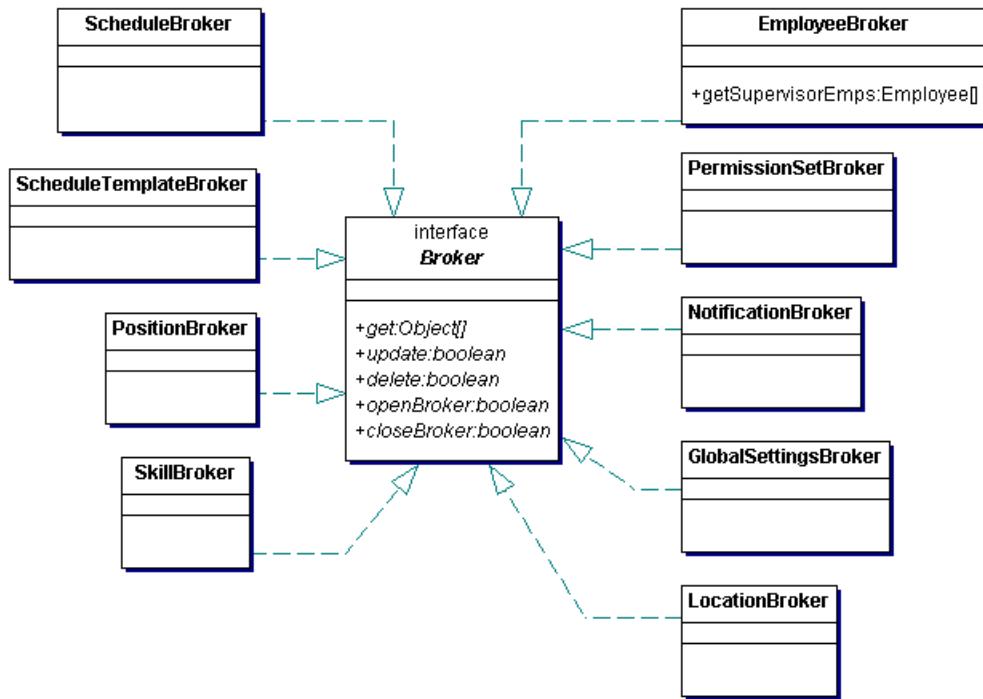


Figure 13.2: Persistence Class Diagram

This diagram shows the objects that make up the persistence layer, the classes and objects that utilize backend or data objects

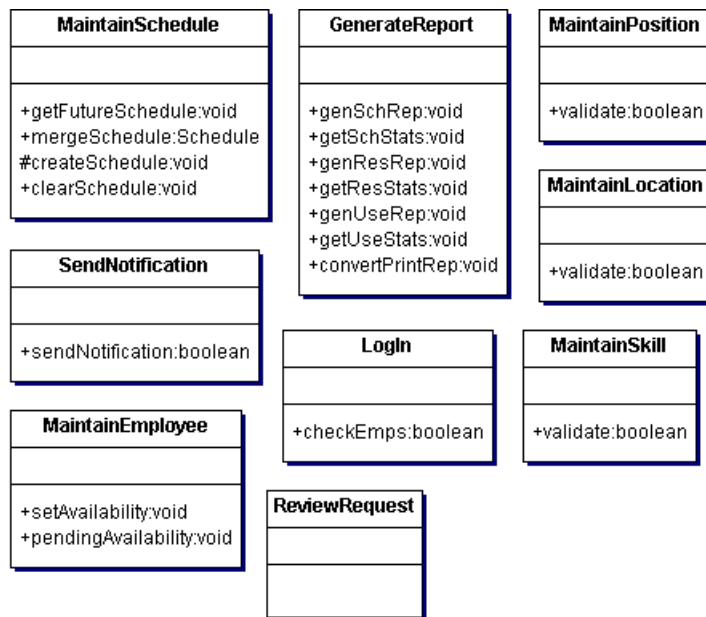


Figure 13.3: Application Class Diagram

This diagram shows the objects that make up the application-side real world objects.

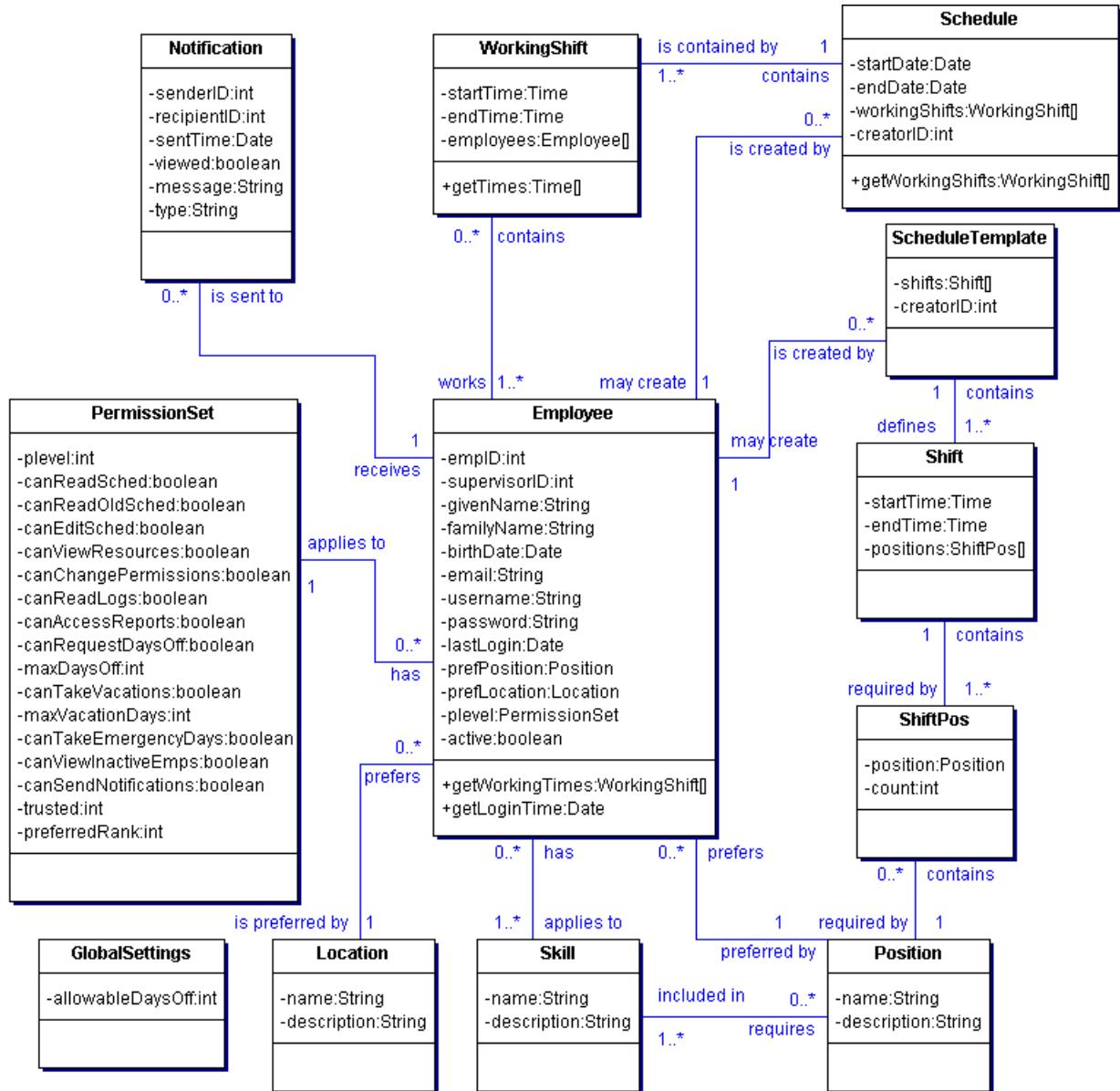


Figure 13.4: Business Class Diagram

Similar to the problem domain, this diagram displays the business classes. Business classes contain data that reflect real world objects that are used and modified in a business.

# Chapter 14

## Interaction Sequence Diagrams

## 14.1 New Schedule Sequence Diagram

### Interaction Diagrams



#### Create New Schedule

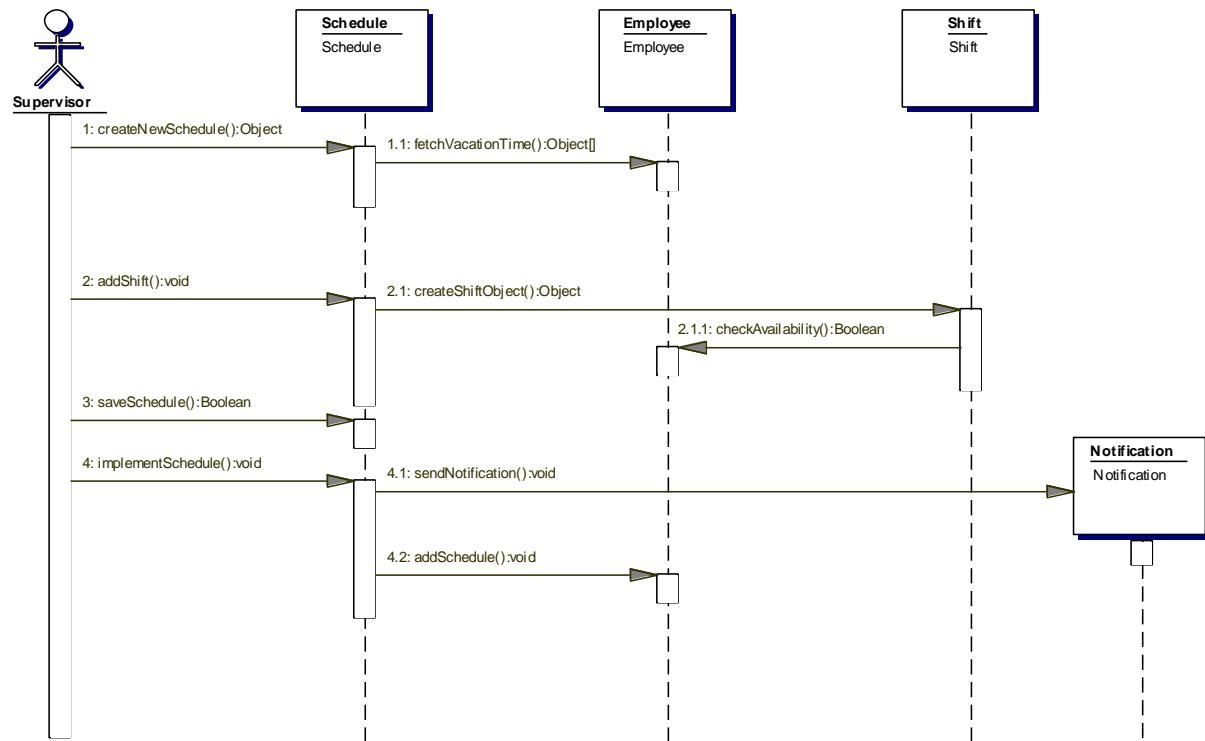


Figure 14.1: New Schedule Creation Sequence Diagram

A representation of the proper sequence in creating a new schedule

## 14.2 Request Shift Change Sequence Diagram



Request Shift-Change

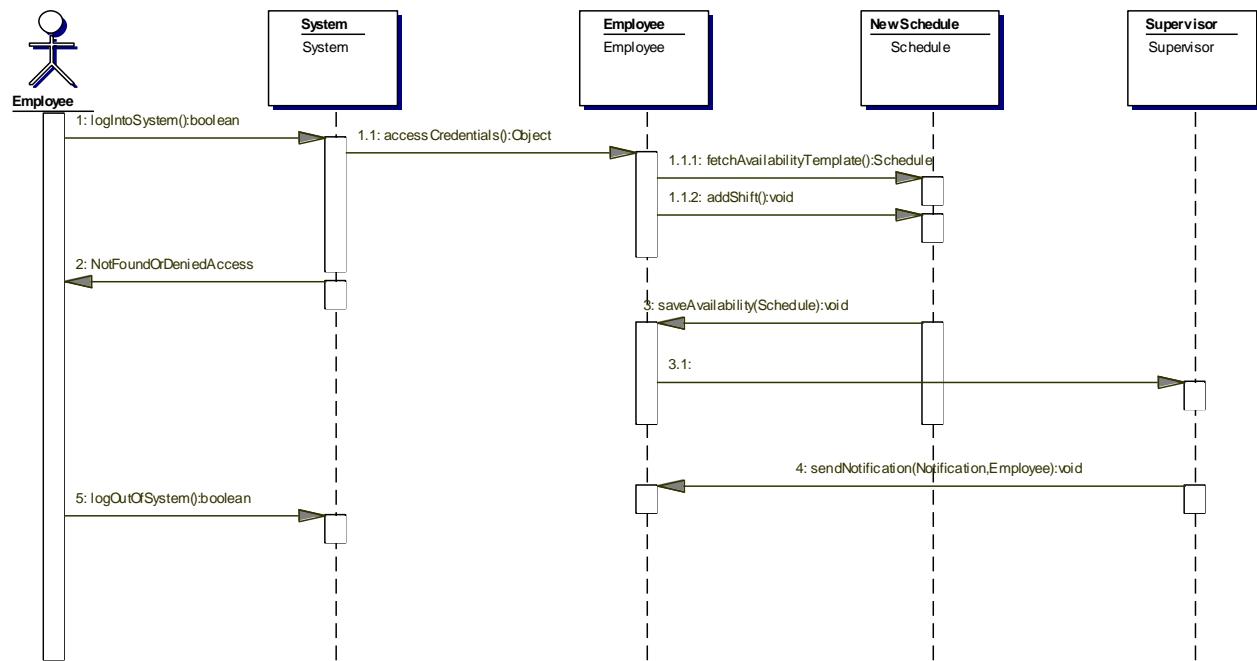


Figure 14.2: Request Shift Change Sequence Diagram

A representation of the proper sequence in requesting a shift change

## 14.3 Viewing Schedule Sequence Diagram



### View Workgroup Schedule

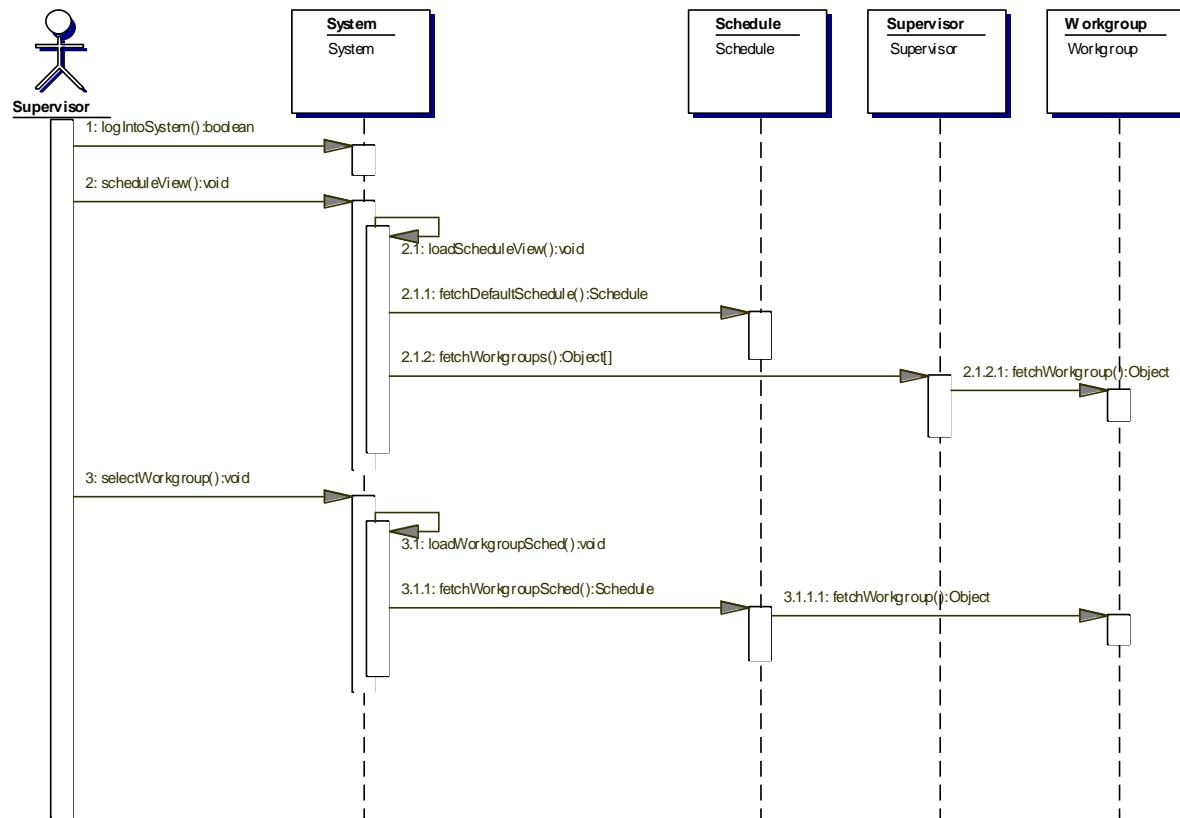
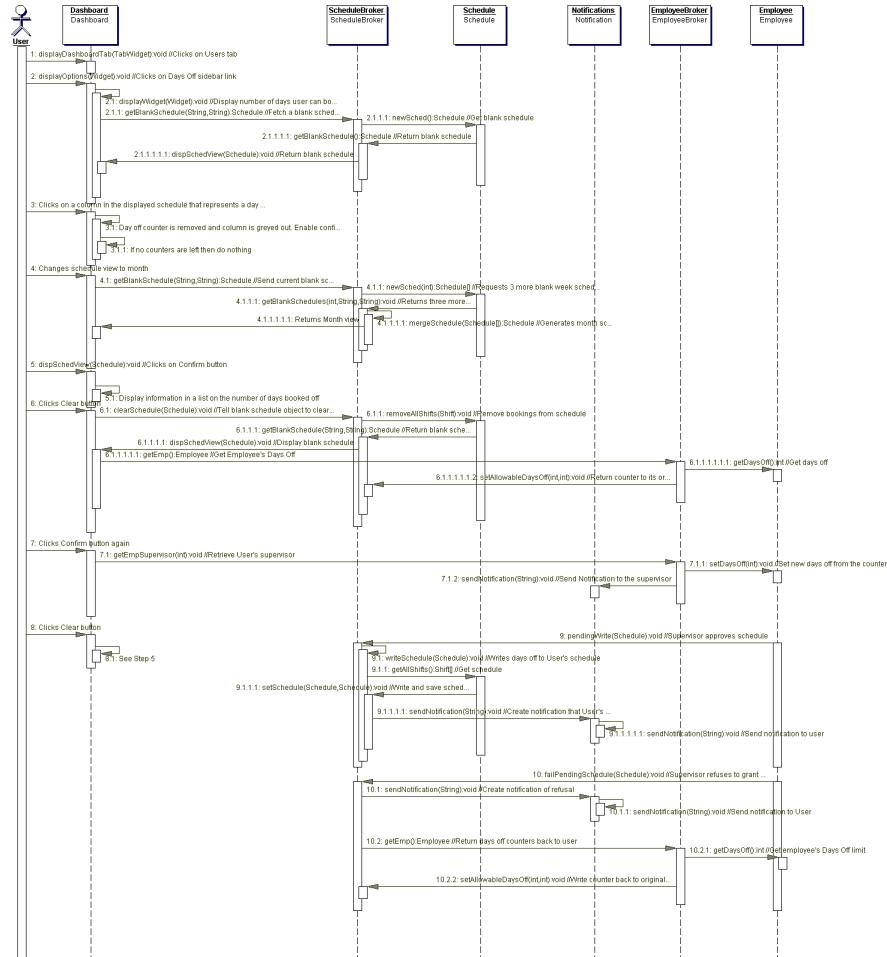


Figure 14.3: Viewing Schedule Sequence Diagram

A representation of the proper sequence for viewing a schedule

## 14.4 Book Days Off Sequence Diagram



**Figure 14.4: Book Days Off Sequence Diagram**  
A representation of the proper sequence for booking days off

## 14.5 Administrative Schedule Viewing Sequence Diagram

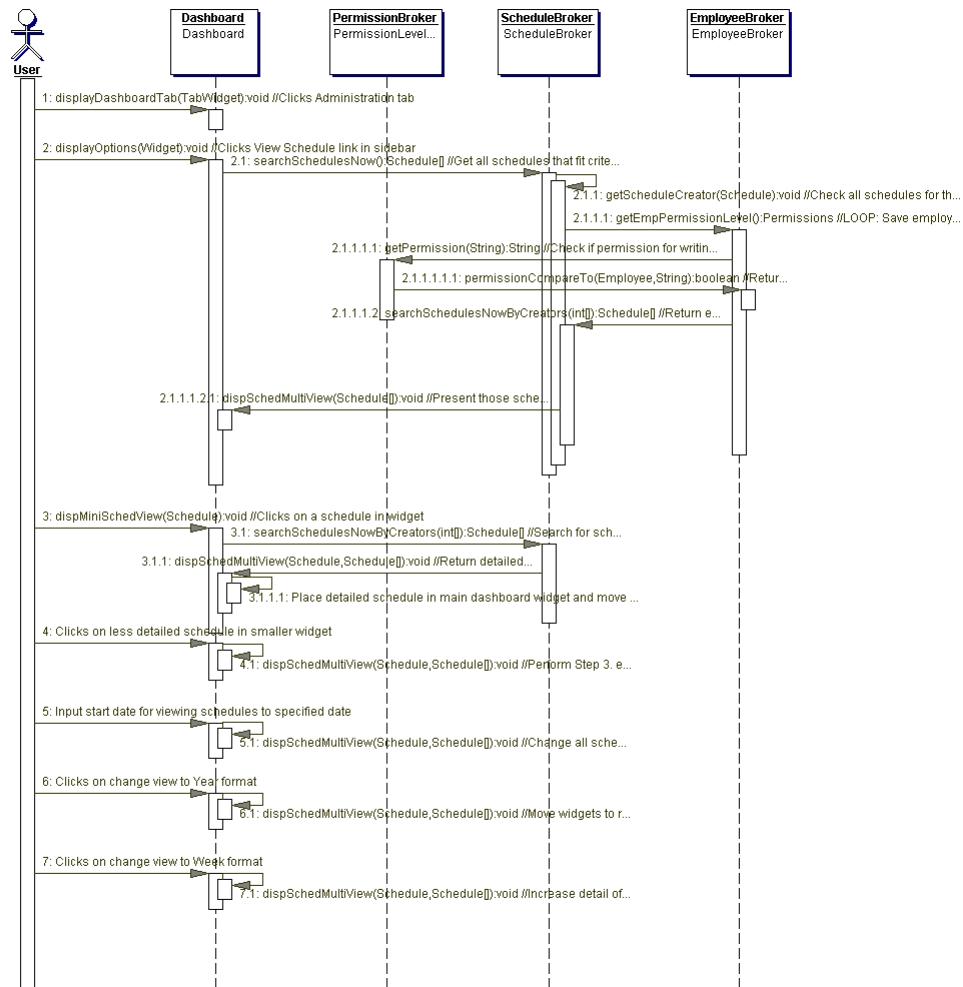


Figure 14.5: Administrative Schedule Viewing Sequence Diagram

A representation of the proper sequence for a manager or authoritative employee to view multiple schedules

## 14.6 Request Shift Change Sequence Diagram

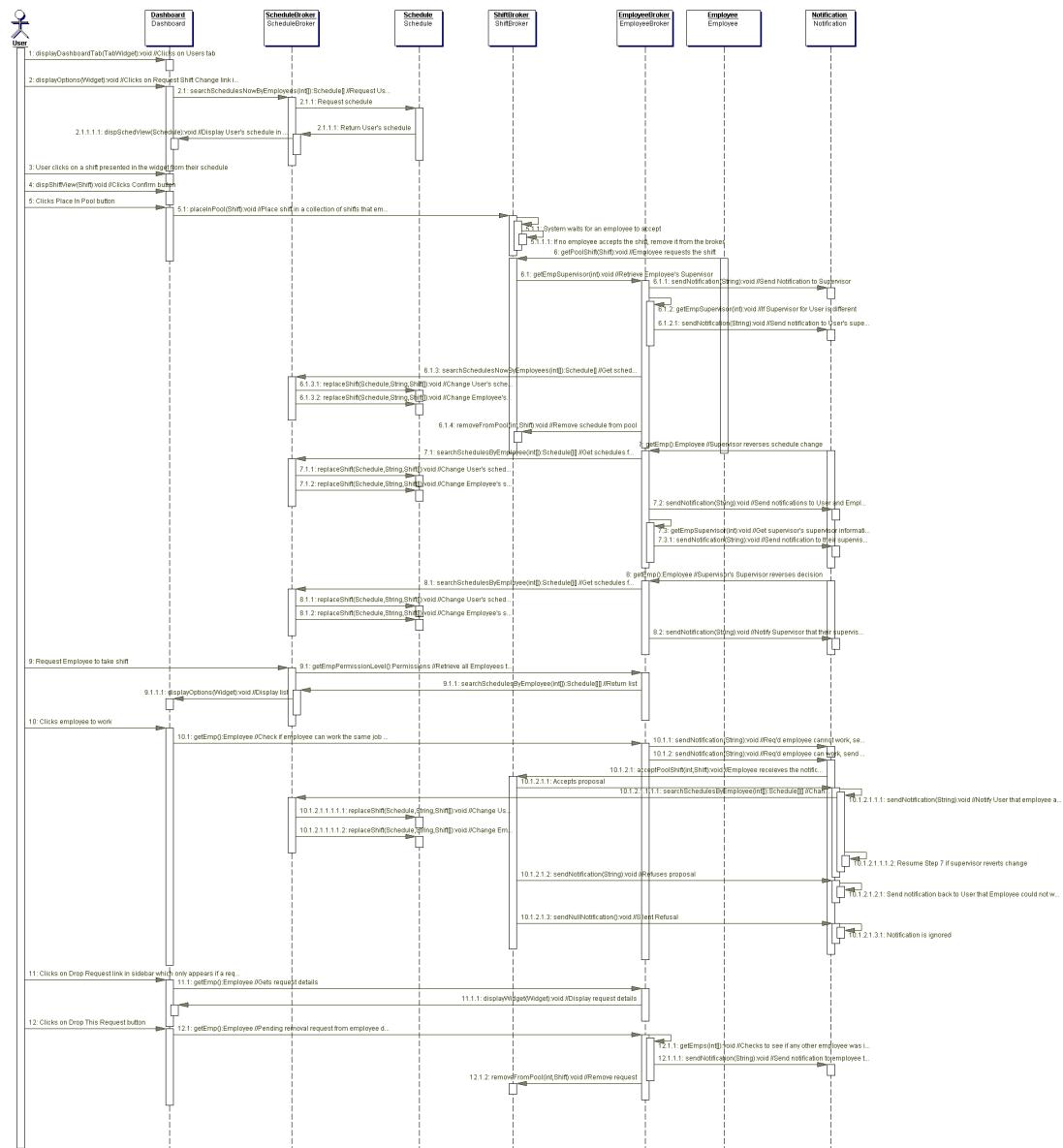
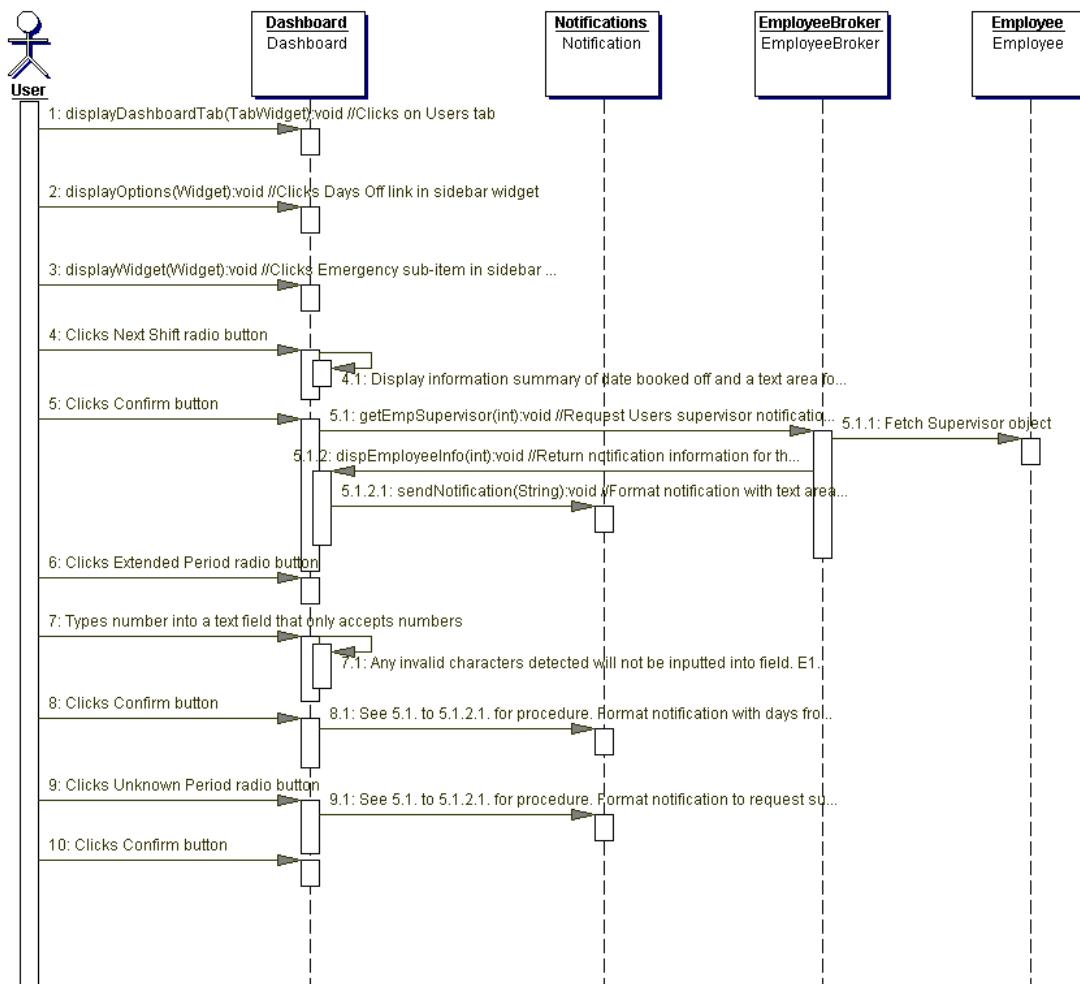


Figure 14.6: Request Shift Change Sequence Diagram

A representation of the proper sequence for changing a shift with another employee

## 14.7 Update Availability Sequence Diagram



**Figure 14.7: Update Availability Sequence Diagram**

A representation of the proper sequence for updating an Employee's availability

## 14.8 Access Schedule Sequence Diagram

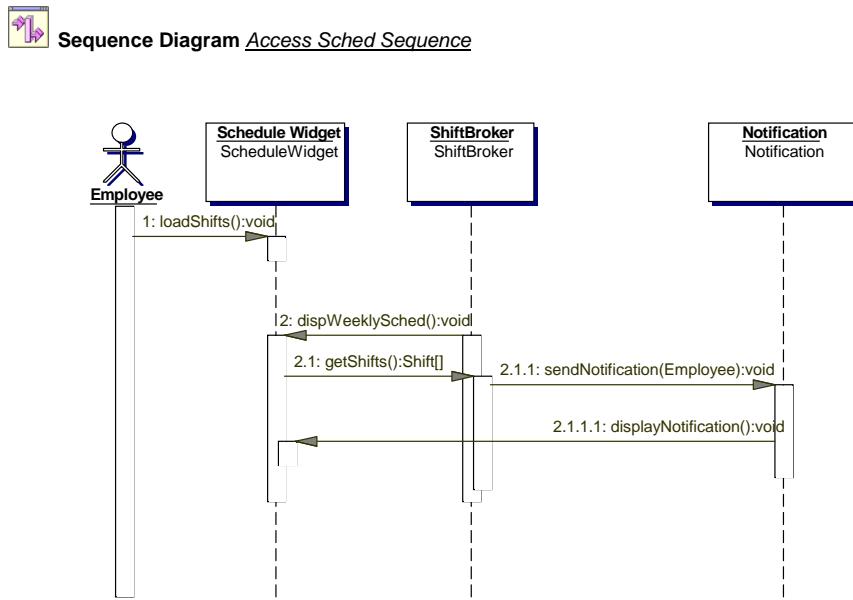


Figure 14.8: **Access Schedule Sequence Diagram**

A representation of the proper sequence for accessing an Employee's Schedule

## 14.9 Add Supervisor Sequence Diagram

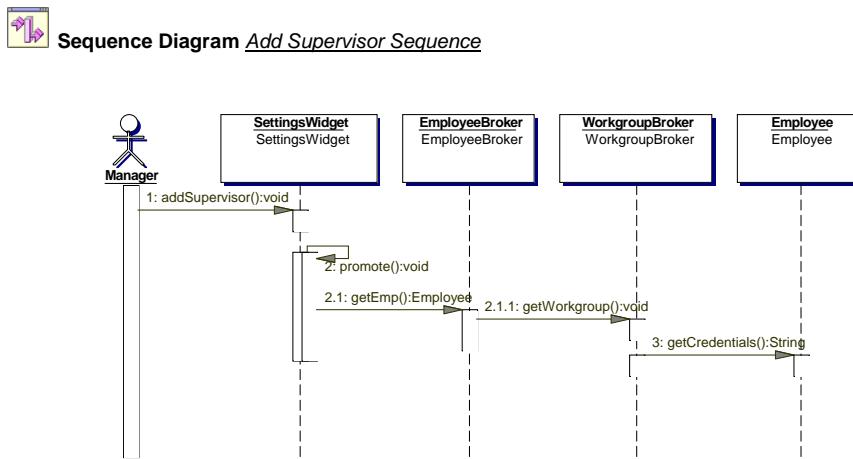


Figure 14.9: **Add Supervisor Sequence Diagram**

A representation of the proper sequence for creating an employee with supervisor permissions

## 14.10 Create Employee Type Sequence Diagram



Sequence Diagram Create Emp Type Sequence

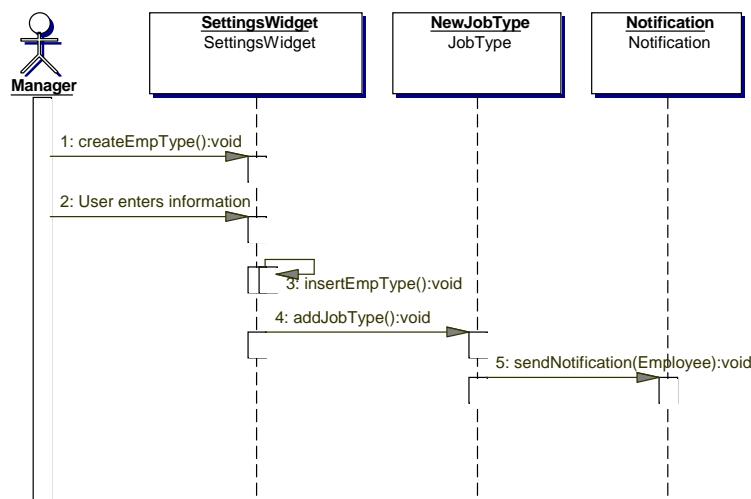


Figure 14.10: **Create Employee Type Sequence Diagram**

A representation of the proper sequence for creating an employee to work a specific job

## 14.11 Create New Employee Sequence Diagram

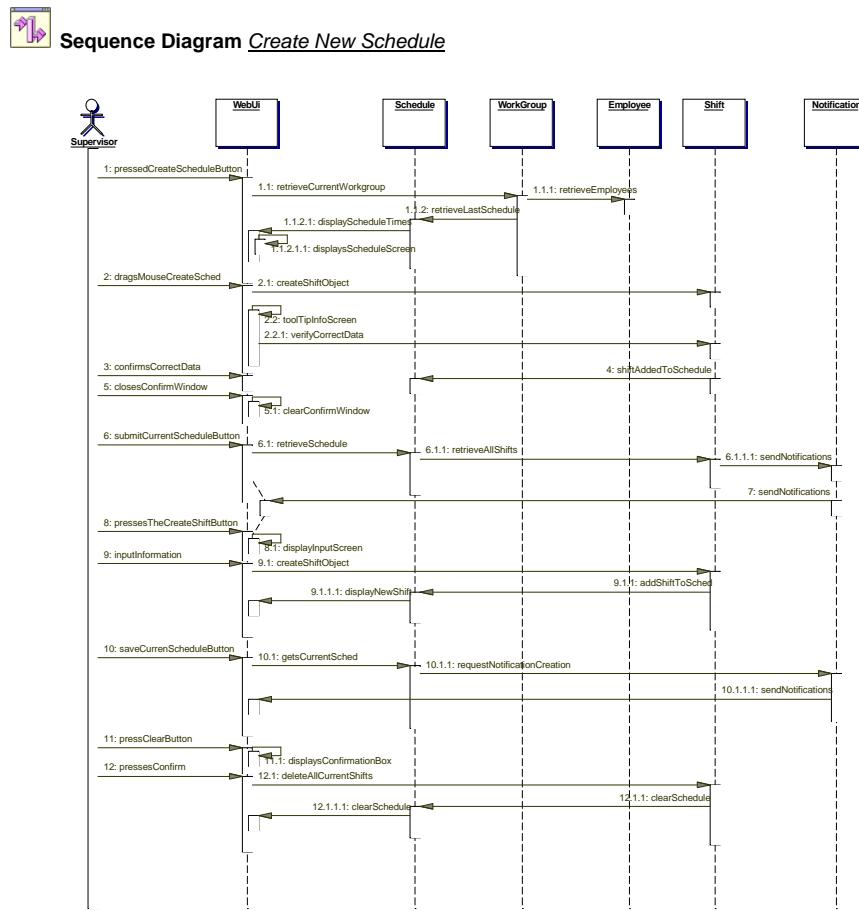


Figure 14.11: **Create New Employee Sequence Diagram**  
A representation of the proper sequence for creating a new employee

## 14.12 Distribute Report Schedule Sequence Diagram

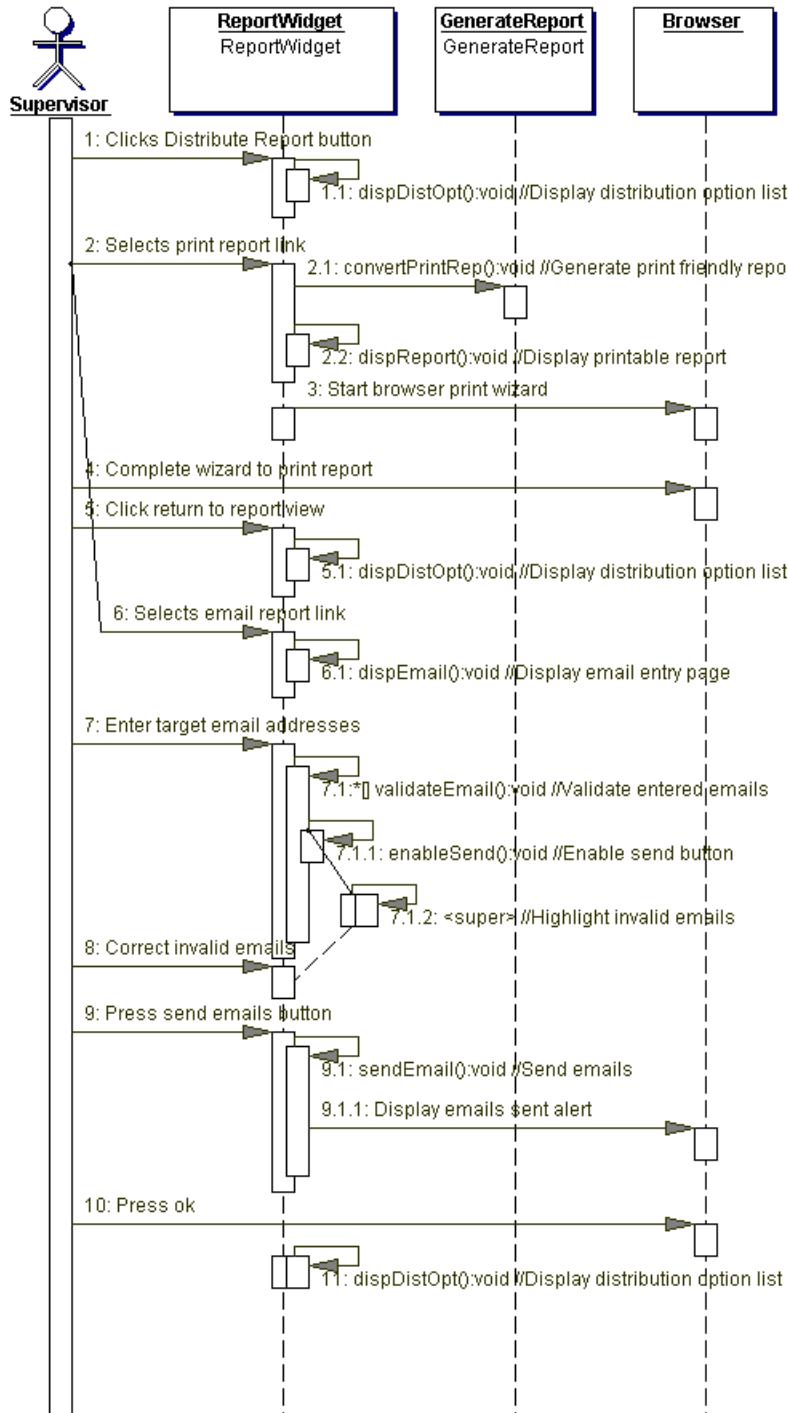


Figure 14.12: **Distribute Report Sequence Diagram**

A representation of the proper sequence for distributing reports that are created

## 14.13 Generate Report Schedule Sequence Diagram

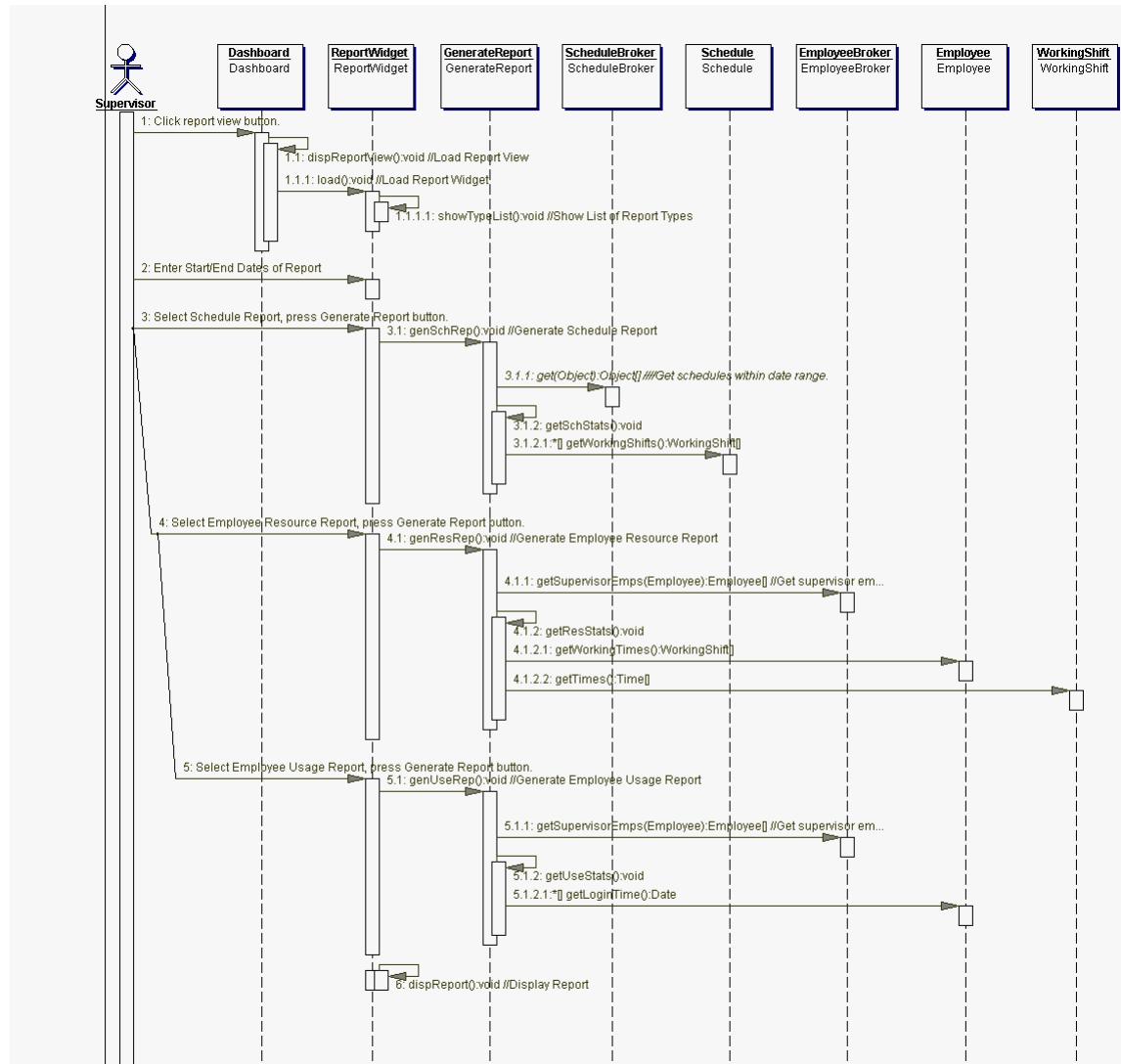


Figure 14.13: **Generate Report Schedule Sequence Diagram**  
A representation of the proper sequence for generating reports

## 14.14 System Log In Sequence Diagram

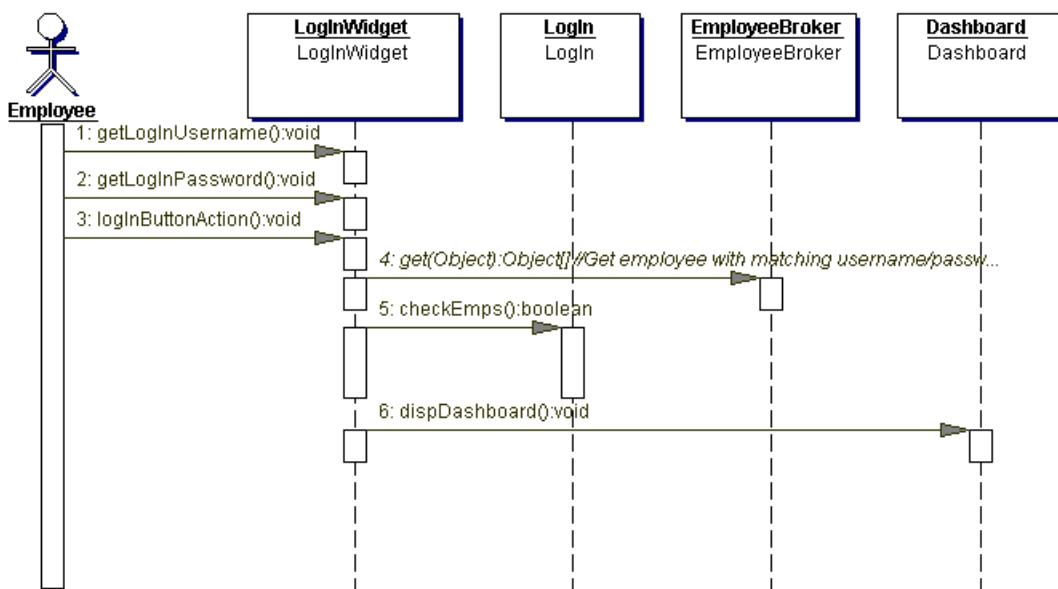


Figure 14.14: System Log In Sequence Diagram

A representation of the proper sequence for logging into the system

## 14.15 Maintain Workgroup Sequence Diagram

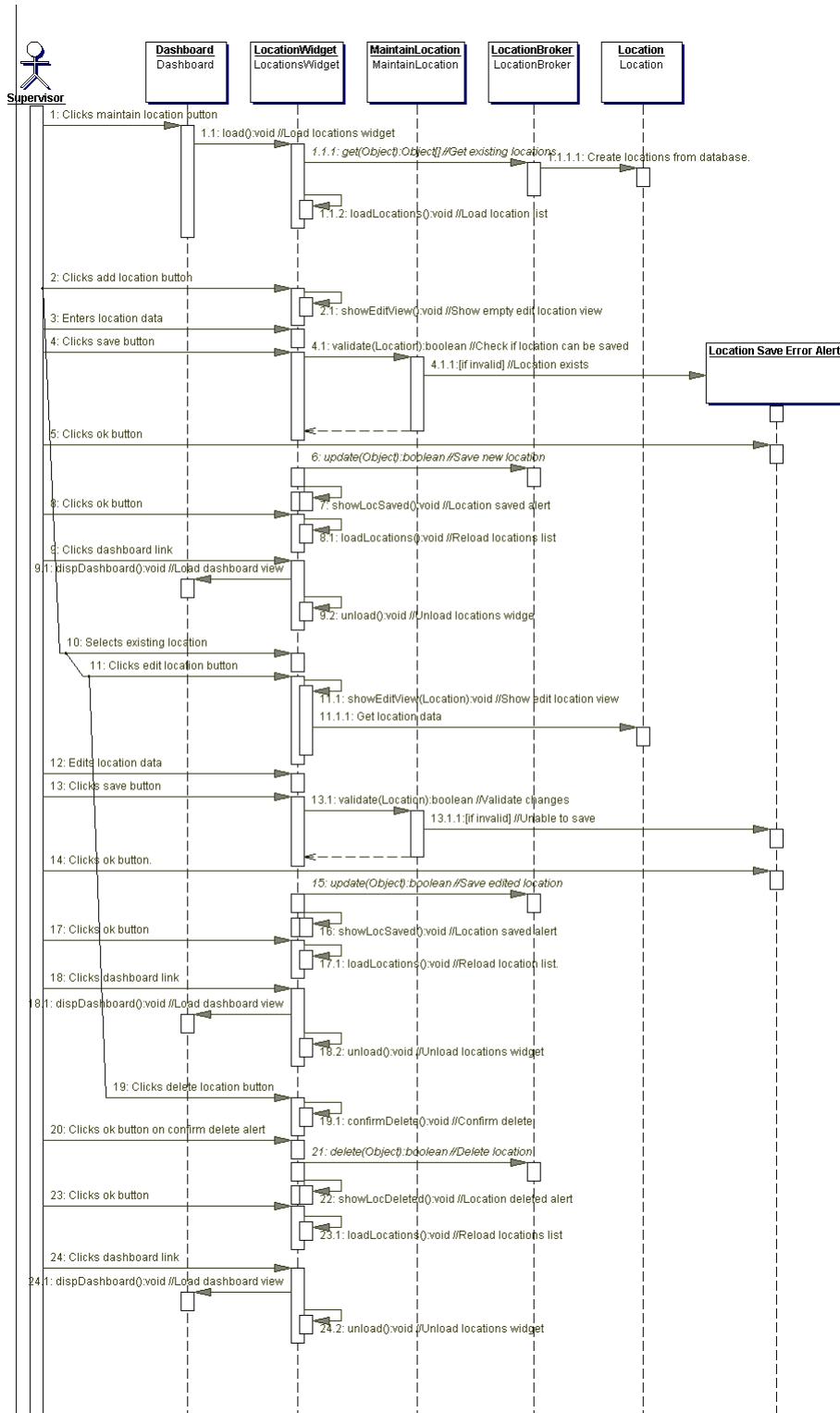


Figure 14.15: Maintain Workgroup Sequence Diagram

A representation of the proper sequence for editing workgroups to the desired situation

## 14.16 Maintain Position Sequence Diagram

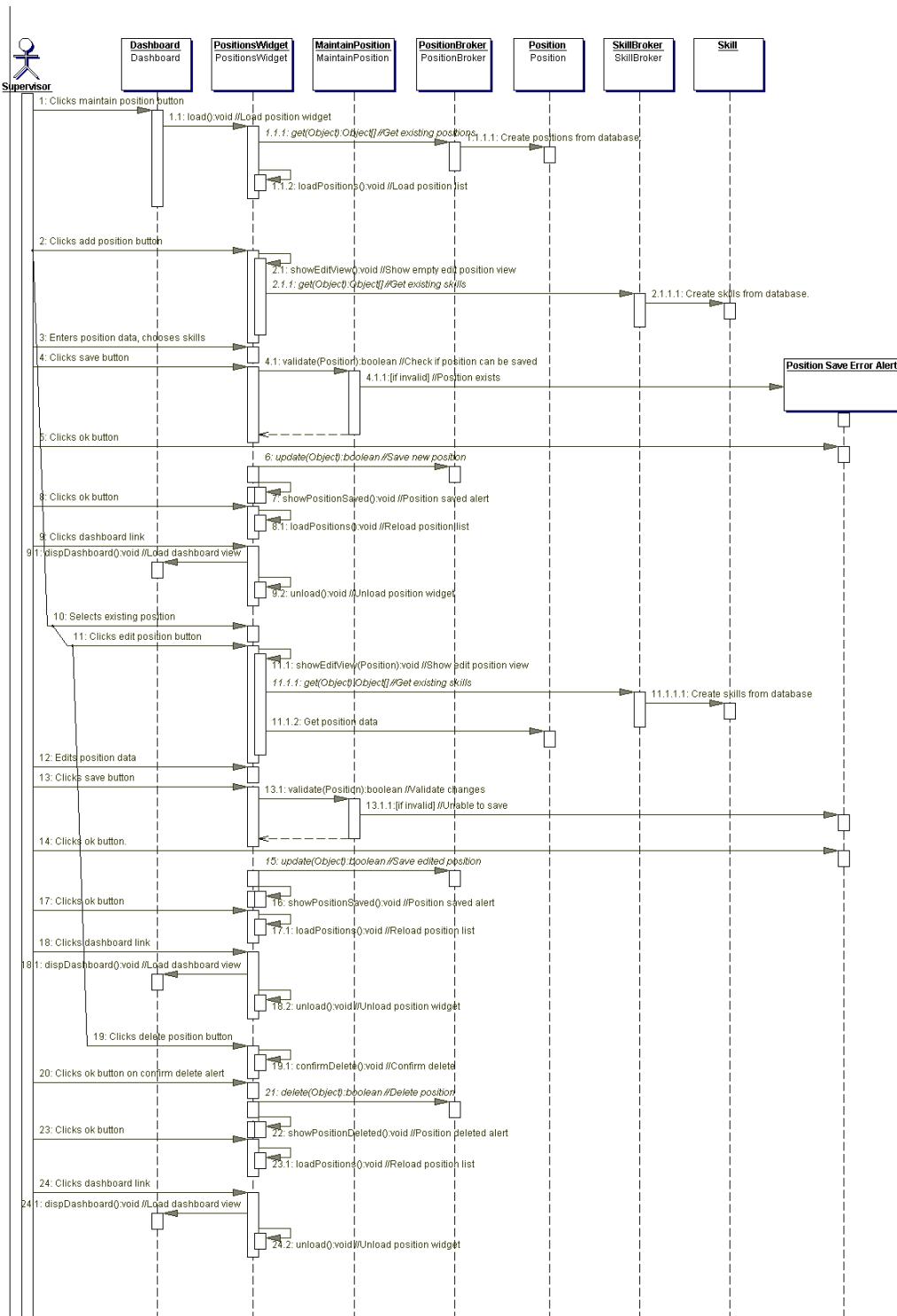


Figure 14.16: Maintain Employee Sequence Diagram

A representation of the proper sequence for editing an employee's position to the desired situation

## 14.17 Maintain Skill Sequence Diagram

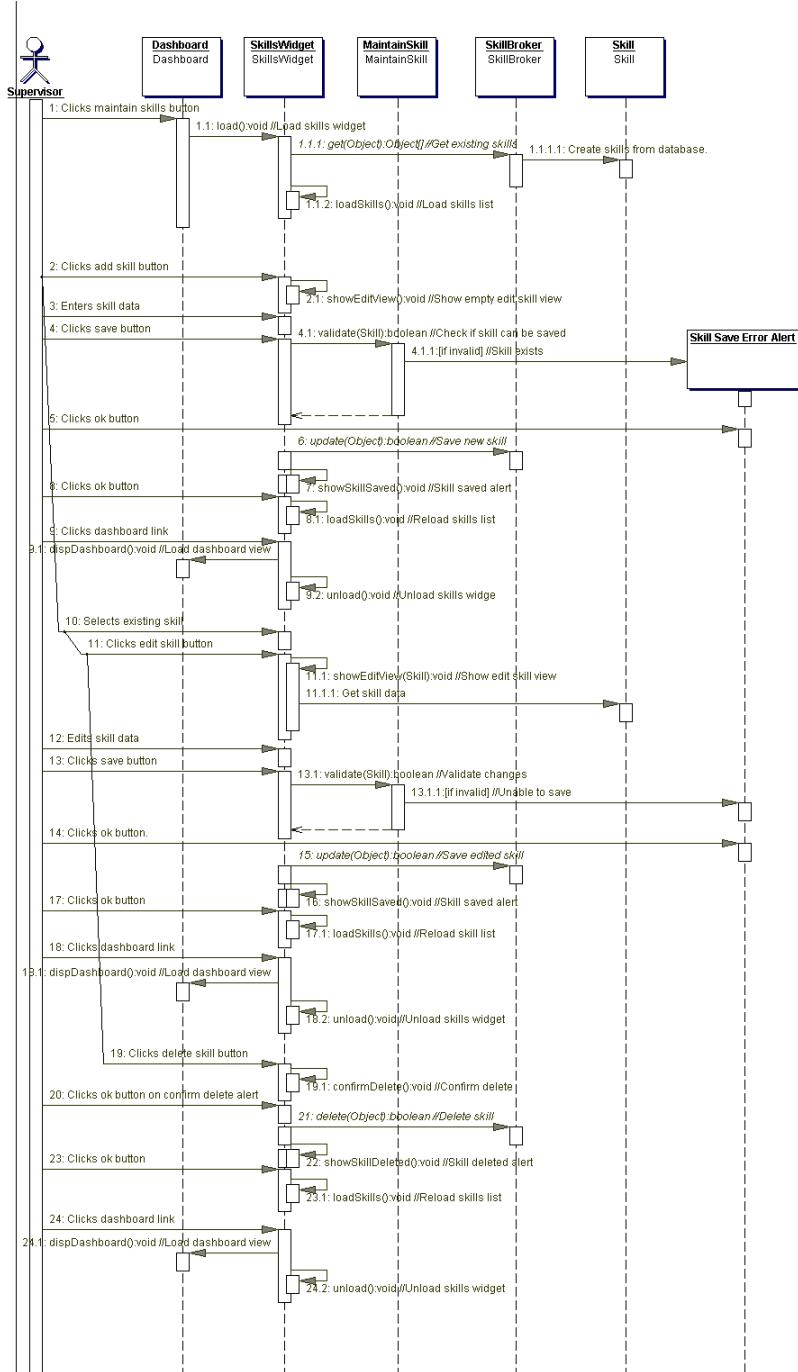


Figure 14.17: Maintain Employee Sequence Diagram

A representation of the proper sequence for editing an employee's skills regarding the desired situation

## 14.18 Review Request Sequence Diagram

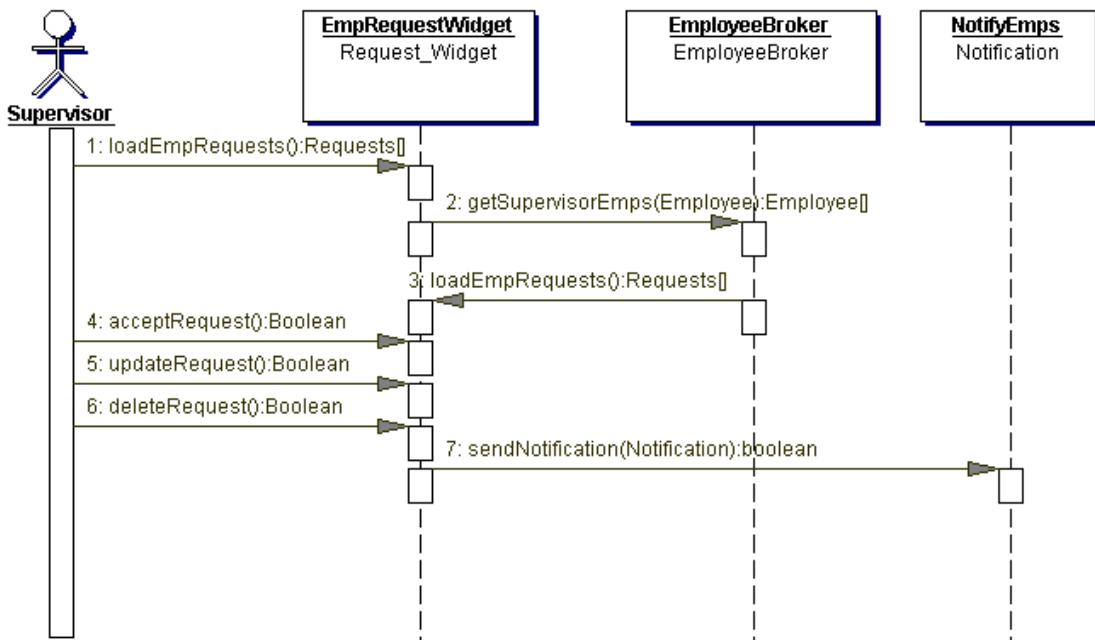


Figure 14.18: **Review Request Sequence Diagram**

A representation of the proper sequence for reviewing an employee request by an authoritative employee

## 14.19 Send Event Sequence Diagram

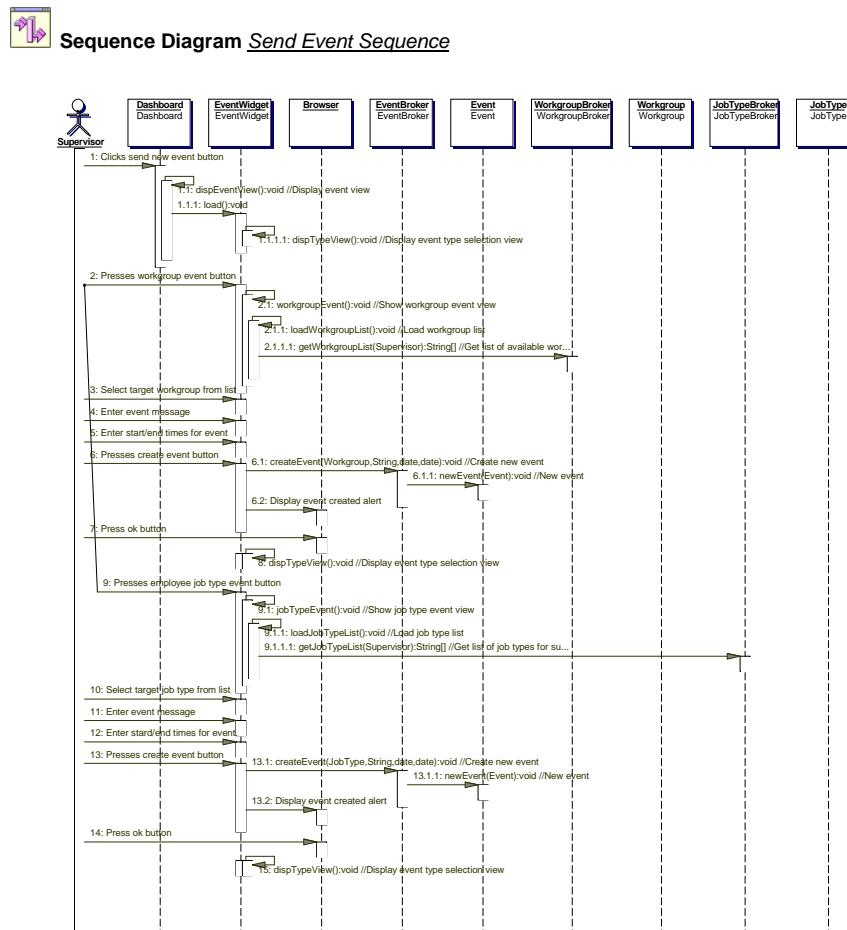


Figure 14.19: **Send Event Sequence Diagram**

A representation of the proper sequence for posting an event that certain groups of employees will see

## 14.20 Book Days Off Sequence Diagram

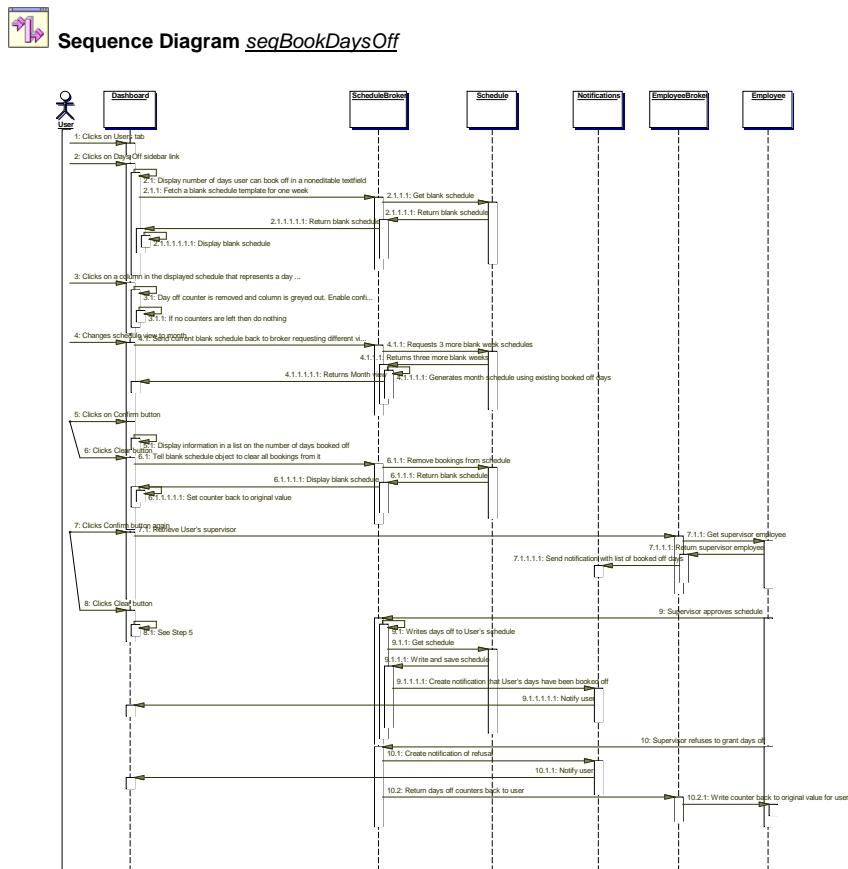


Figure 14.20: Book Days Off Sequence Diagram

A representation of the proper sequence for an employee to reduce working days in their schedule

## 14.21 Manager View Schedules Sequence Diagram

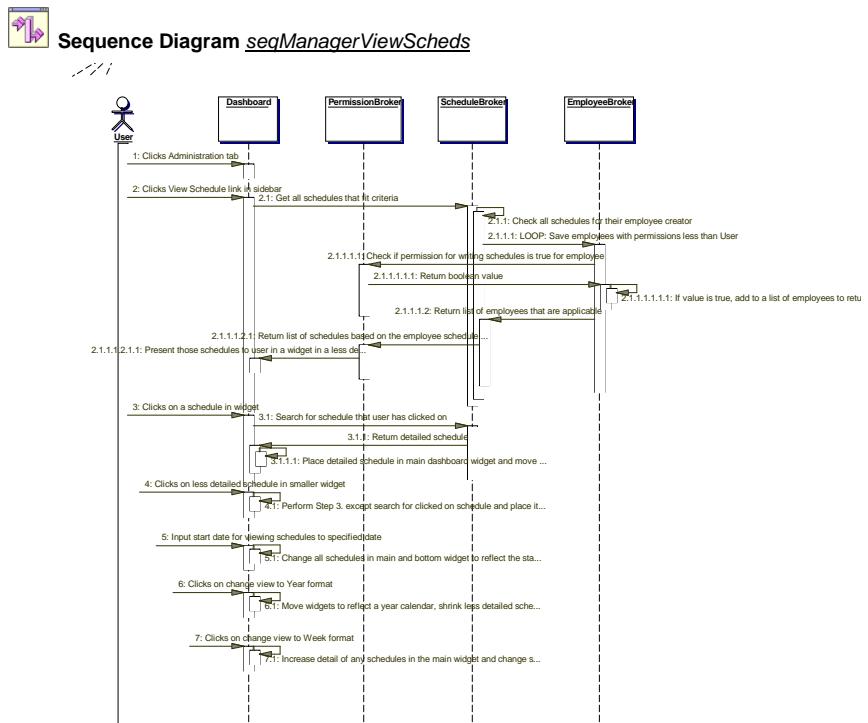


Figure 14.21: Manager View Schedules Sequence Diagram

A representation of the proper sequence for a Manager to view multiple schedules at one time

## 14.22 Request Shift Change Sequence Diagram

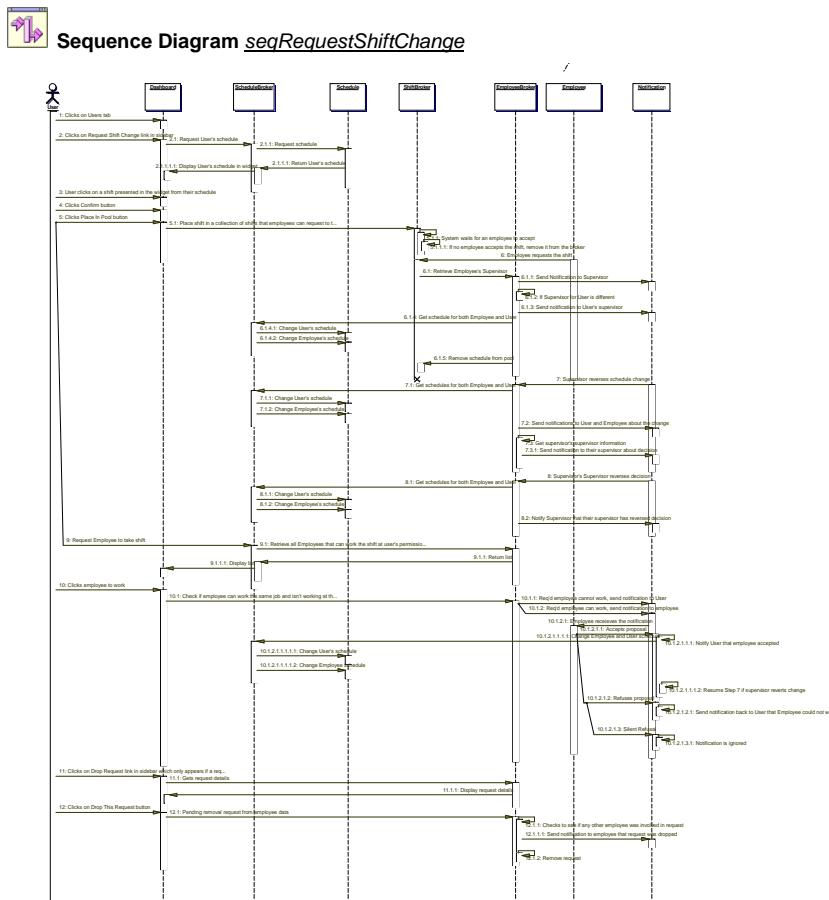


Figure 14.22: Request Shift Change Sequence Diagram

A representation of the proper sequence for requesting another user to take or exchange shifts with

## 14.23 Update Availability Sequence Diagram

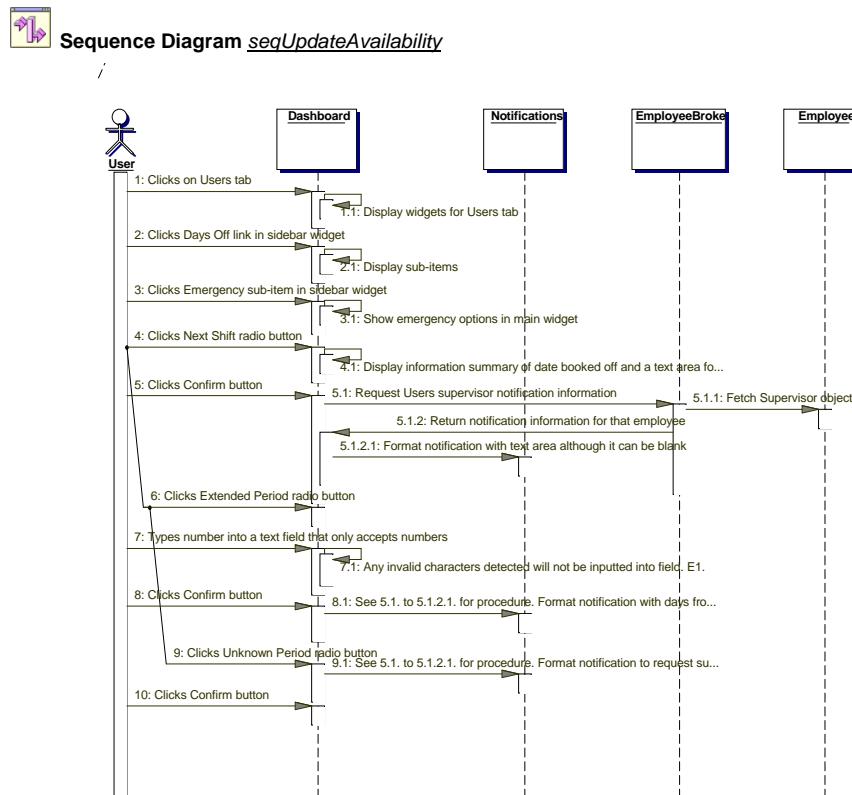


Figure 14.23: **Update Availability Sequence Diagram**

A representation of the proper sequence for changing an employee's available work times

## 14.24 Send Emergency Notification Sequence Diagram

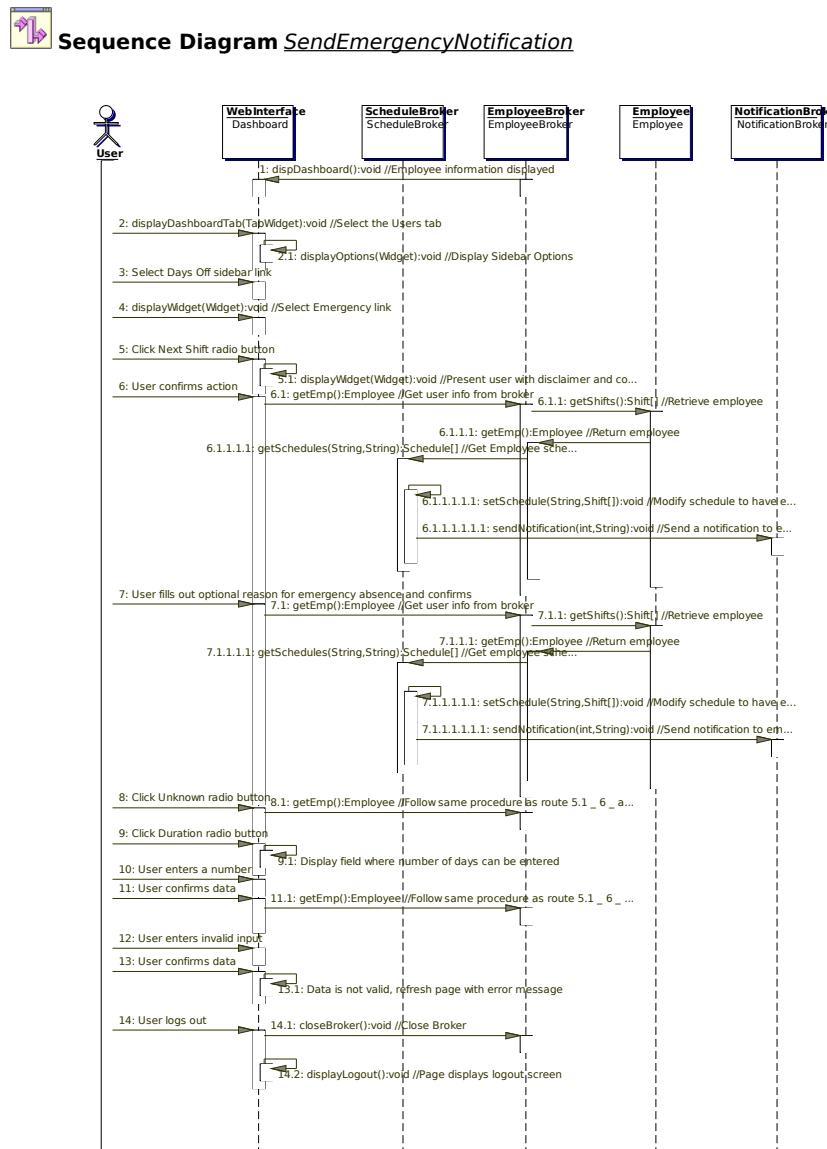


Figure 14.24: Send Emergency Notification Sequence Diagram

A representation of the proper sequence for handling emergencies that prevent employee's presence at work

## 14.25 View Workgroup Schedule Sequence Diagram

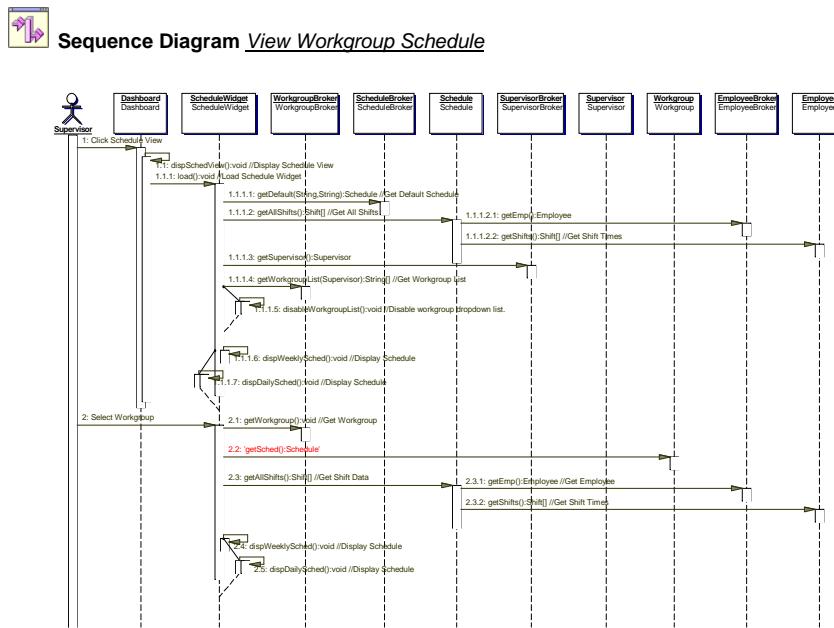


Figure 14.25: View Workgroup Schedule Sequence Diagram

A representation of the proper sequence for viewing schedules of employees working for one job type

# Chapter 15

## Data Dictionary

## 15.1 Interface

### **Class Detail**

#### **Class Dashboard**

package: [Interface](#)

```
public class Dashboard
```

The main interface object of the system that provides access to all other pages and widgets, such as schedule or report widgets. Available sections will be determined by the permissions of the employee that is currently logged in.

#### **Method Summary**

public <a href="#">void</a>	<a href="#">dispDashboard()</a>	Displays the dashboard itself.
public <a href="#">void</a>	<a href="#">dispEmployeeInfo(int p0)</a>	
public <a href="#">void</a>	<a href="#">dispEventView()</a>	Displays the report view, loading and displaying the Event Widget that allows new events to be created and sent to workgroups and job types.
public <a href="#">void</a>	<a href="#">displayDashboardTab(TabWidget p0)</a>	Displays a Tab widget, a larger widget that contains a main widget area that does not interfere with other widget's main areas
public <a href="#">void</a>	<a href="#">displayLogout()</a>	Logs user out along with displaying the page they are directed to after logging out from their dashboard.
public <a href="#">void</a>	<a href="#">displayOptions(Widget p0)</a>	Displays relevant options in the sidebar widget, related tasks based on the main widget.
public <a href="#">void</a>	<a href="#">displayWidget(Widget p0)</a>	Displays a widget set by the method in the main widget area
public <a href="#">void</a>	<a href="#">dispMiniSchedView(Schedule p0)</a>	Displays a schedule in a view that based on colours in a week/month/year form so that an understanding of shift intensity and employee density can be understood.
public <a href="#">void</a>	<a href="#">dispReportView()</a>	Displays the report view, loading and displaying the Report Widget that provides all report functionality.
public <a href="#">void</a>	<a href="#">dispSchedMultiView(Schedule p0, Schedule[] p1)</a>	Displays the standalone schedule parameter in a detailed form in a larger widget with the rest seen as smaller simple view widgets.
public <a href="#">void</a>	<a href="#">dispSchedMultiView(Schedule[] p0)</a>	Displays an array of schedules in a multi-view.
public <a href="#">void</a>	<a href="#">dispSchedView()</a>	Displays the report view, loading and displaying the expanded version of the Schedule Widget that provides all functionality for managing schedules.
public <a href="#">void</a>	<a href="#">dispSchedView(Schedule p0)</a>	Displays a specific schedule in the main widget
public <a href="#">void</a>	<a href="#">dispShiftView(Shift p0)</a>	Displays a detailed view of a shift's properties

#### **Method Detail**

##### **dispDashboard**

```
public void dispDashboard()
```

Displays the dashboard itself.

---

## **dispEmployeeInfo**

Chapter 15 [Section 15.1](#) [Interface](#)

---

### **dispEventView**

```
public void dispEventView()
```

Displays the report view, loading and displaying the Event Widget that allows new events to be created and sent to workgroups and job types.

---

### **displayDashboardTab**

```
public void displayDashboardTab(TabWidget p0)
```

Displays a Tab widget, a larger widget that contains a main widget area that does not interfere with other widget's main areas

---

### **displayLogout**

```
public void displayLogout()
```

Logs user out along with displaying the page they are directed to after logging out from their dashboard.

---

### **displayOptions**

```
public void displayOptions(Widget p0)
```

Displays relevant options in the sidebar widget, related tasks based on the main widget.

---

### **displayWidget**

```
public void displayWidget(Widget p0)
```

Displays a widget set by the method in the main widget area

---

### **dispMiniSchedView**

```
public void dispMiniSchedView(Schedule p0)
```

Displays a schedule in a view that based on colours in a week/month/year form so that an understanding of shift intensity and employee density can be understood.

---

### **dispReportView**

```
public void dispReportView()
```

Displays the report view, loading and displaying the Report Widget that provides all report functionality.

---

### **dispSchedMultiView**

```
public void dispSchedMultiView(Schedule p0, Schedule[] p1)
```

Displays the standalone schedule parameter in a detailed form in a larger widget with the rest seen as smaller simple view widgets.

---

### **dispSchedMultiView**

```
public void dispSchedMultiView(Schedule[] p0)
```

Displays an array of schedules in a multi-view. All schedules are in a mini-view, a small widget that displays a simple overview of what the schedule contains.

---

### **dispSchedView**

```
public void dispSchedView()
```

Displays the report view, loading and displaying the expanded version of the Schedule Widget that provides all functionality for managing schedules.

## Chapter 15 - Section: 15.1. Interface

---

### **dispSchedView**

```
public void dispSchedView(Schedule p0)
```

Displays a specific schedule in the main widget

### **dispShiftView**

```
public void dispShiftView(Shift p0)
```

Displays a detailed view of a shift's properties

## **Class EmployeeSettingsWidget**

package: [Interface](#)

```
Interface.Widget
|
+--Interface.EmployeeSettingsWidget
```

```
public class EmployeeSettingsWidget
```

**Extends:**

[Interface.Widget](#)

Provides the user interface that will be used to change employee specific permission settings.

### **Method Summary**

public void	<a href="#">addSupervisor()</a>	Adds an employee as a supervisor within the system, and sets their permissions accordingly.
public void	<a href="#">createEmpType()</a>	Creates a new employee type, which can then be applied to any employees within the system.
public void	<a href="#">insertEmpType()</a>	Inserts a new employee type, based off of data from outside the system.
public void	<a href="#">promote()</a>	Increases an employees permission level.

### **Method Detail**

#### **addSupervisor**

```
public void addSupervisor()
```

Adds an employee as a supervisor within the system, and sets their permissions accordingly.

#### **createEmpType**

```
public void createEmpType()
```

Creates a new employee type, which can then be applied to any employees within the system.

#### **insertEmpType**

```
public void insertEmpType()
```

Inserts a new employee type, based off of data from outside the system.

#### **promote**

```
public void promote()
```

Increases an employees permission level.

## Class LocationsWidget Interface

package: [Interface](#)

```
Interface.Widget
|
+--Interface.LocationsWidget
```

public class LocationsWidget

Extends:

[Interface.Widget](#)

Provides functionality for adding, editing and deleting locations.

### Method Summary

public void	<a href="#">confirmDelete()</a>	Prompts the user to confirm that they wish to delete the location.
public void	<a href="#">loadLocations()</a>	Populates the existing locations list with positions from the database.
public void	<a href="#">showEditView()</a>	Shows the edit location view with all fields blank for the user to create a new location.
public void	<a href="#">showEditView(Location p0)</a>	Shows the edit location view with all fields filled with data from the chosen location.
public void	<a href="#">showLocDeleted()</a>	Display an alert, showing the user that the chosen location was deleted.
public void	<a href="#">showLocSaved()</a>	Display an alert, showing the user that the locations was saved or created.

### Method Detail

#### **confirmDelete**

public void confirmDelete()

Prompts the user to confirm that they wish to delete the location.

#### **loadLocations**

public void loadLocations()

Populates the existing locations list with positions from the database.

#### **showEditView**

public void showEditView(Location p0)

Shows the edit location view with all fields filled with data from the chosen location.

#### **showEditView**

public void showEditView()

Shows the edit location view with all fields blank for the user to create a new location.

#### **showLocDeleted**

public void showLocDeleted()

Display an alert, showing the user that the chosen location was deleted.

## **showLocSaved**

public `void` showLocSaved()  
 Chapter 15 - Section: 15.1. Interface  
 Display an alert, showing the user that the locations was saved or created.

---

### **Class LogInWidget**

package: [Interface](#)

```
Interface.Widget
|
+--Interface.LogInWidget
```

public class LogInWidget

**Extends:**

[Interface.Widget](#)

Provides the user interface that will verify employee credentials and provides access to the system.

#### **Method Summary**

public <code>Boolean</code>	<a href="#">checkEmps()</a>
	Verifies the given username and password by checking it against employees that currently exist within the system.
public <code>void</code>	<a href="#">getLogInPassword()</a>
	Retrieves the password that was entered by the user.
public <code>void</code>	<a href="#">getLogInUsername()</a>
	Retrieves the username that was entered by the user.
public <code>void</code>	<a href="#">logInButtonAction()</a>
	Activates when the log in button is clicked by the user, initiating validation procedures.

#### **Method Detail**

##### **checkEmps**

public `Boolean` checkEmps()

Verifies the given username and password by checking it against employees that currently exist within the system.

##### **getLogInPassword**

public `void` getLogInPassword()

Retrieves the password that was entered by the user.

##### **getLogInUsername**

public `void` getLogInUsername()

Retrieves the username that was entered by the user.

##### **logInButtonAction**

public `void` logInButtonAction()

Activates when the log in button is clicked by the user, initiating validation procedures.

### **Class PositionsWidget**

package: [Interface](#)

```
Interface.Widget
|
+-Interface.PositionsWidget
```

## Chapter 15 - Section: 15.1. Interface

```
public class PositionsWidget
```

### Extends:

[Interface.Widget](#)

Provides functionality for adding, editing and deleting positions.

### Method Summary

public void	<a href="#">confirmDelete()</a>	Prompts the user to confirm that they wish to delete the position.
public void	<a href="#">loadPositions()</a>	Populates the existing positions list with positions from the database.
public void	<a href="#">showEditView()</a>	Shows the edit position view with all fields blank for the user to create a new position.
public void	<a href="#">showEditView(Position p0)</a>	Shows the edit position view with all fields filled with data from the chosen position.
public void	<a href="#">showPositionDeleted()</a>	Display an alert, showing the user that the chosen position was deleted.
public void	<a href="#">showPositionSaved()</a>	Display an alert, showing the user that the position was saved or created.

### Method Detail

#### **confirmDelete**

```
public void confirmDelete()
```

Prompts the user to confirm that they wish to delete the position.

#### **loadPositions**

```
public void loadPositions()
```

Populates the existing positions list with positions from the database.

#### **showEditView**

```
public void showEditView(Position p0)
```

Shows the edit position view with all fields filled with data from the chosen position.

#### **showEditView**

```
public void showEditView()
```

Shows the edit position view with all fields blank for the user to create a new position.

#### **showPositionDeleted**

```
public void showPositionDeleted()
```

Display an alert, showing the user that the chosen position was deleted.

#### **showPositionSaved**

```
public void showPositionSaved()
```

Display an alert, showing the user that the position was saved or created.

## Class ReportWidget

Chapter 15 - Section: 15.1. Interface

```
Interface.Widget
|
+-Interface.ReportWidget
```

```
public class ReportWidget
```

**Extends:**

[Interface.Widget](#)

Provides the user interface for distributing and managing events that exist within the system.

### Method Summary

public void	<a href="#">dispDistOpt()</a>	Display options for distributing a report after it has been generated.
public void	<a href="#">dispEmail()</a>	Displays the email entry page where the user may enter which emails the report should be sent to.
public void	<a href="#">dispReport()</a>	Displays a report after it has been generated.
public void	<a href="#">enableSend()</a>	Enables the send button so that reports can be sent to the entered email address.
public void	<a href="#">genPrintRep()</a>	Generates the print-friendly version of a currently displayed report.
public void	<a href="#">genResRep()</a>	Generates a resource report.
public void	<a href="#">genSchRep()</a>	Generates a schedule report.
public void	<a href="#">genUseRep()</a>	Generates an employee usage report.
public void	<a href="#">getResStats()</a>	Collects resource statistics as part of generating a resource report.
public void	<a href="#">getSchStats()</a>	Collects schedule statistics as part of generating a schedule report.
public void	<a href="#">getUseStats()</a>	Collects usage statistics as part of generating an employee usage report.
public void	<a href="#">initPrint()</a>	Initializes the print wizard of the browser that is currently being used to report.
public void	<a href="#">sendEmail()</a>	Sends the report to the target emails.
public void	<a href="#">showTypeList()</a>	Shows a list allowing the user to choose what type of report they wish to generate.
public void	<a href="#">validateEmail()</a>	Validates entered emails ensuring they are properly formatted before trying to send emails to them.

### Method Detail

#### dispDistOpt

```
public void dispDistOpt()
```

Display options for distributing a report after it has been generated.

## **dispEmail**

`public void dispEmail()`  
Chapter 15 - Section: 15.1. Interface  
Displays the email entry page where the user may enter which emails the report should be sent to.

---

## **dispReport**

`public void dispReport()`  
Displays a report after it has been generated.

---

## **enableSend**

`public void enableSend()`  
Enables the send button so that reports can be sent to the entered email address. This will only be possible after the entered emails have been validated

---

## **genPrintRep**

`public void genPrintRep()`  
Generates the print-friendly version of a currently displayed report.

---

## **genResRep**

`public void genResRep()`  
Generates a resource report.

---

## **genSchRep**

`public void genSchRep()`  
Generates a schedule report.

---

## **genUseRep**

`public void genUseRep()`  
Generates an employee usage report.

---

## **getResStats**

`public void getResStats()`  
Collects resource statistics as part of generating a resource report.

---

## **getSchStats**

`public void getSchStats()`  
Collects schedule statistics as part of generating a schedule report.

---

## **getUseStats**

`public void getUseStats()`  
Collects usage statistics as part of generating an employee usage report.

---

## **initPrint**

`public void initPrint()`  
Initializes the print wizard of the browser that is currently being used to report.

---

## **sendEmail**

public `void` sendEmail()  
 Chapter 15 - Section: 15.1. Interface  
 Sends the report to the target emails.

---

## **showTypeList**

public `void` showTypeList()

Shows a list allowing the user to choose what type of report they wish to generate.

---

## **validateEmail**

public `void` validateEmail()

Validates entered emails ensuring they are properly formated before trying to send emails to them.

---

## **Class Request\_Widget**

package: [Interface](#)

```
Interface.Widget
  |
  +-Request_Widget
```

public class Request\_Widget

**Extends:**

[Interface.Widget](#)

Provides the user interface that employees will use to send requests to their supervisor, and will also be used by supervisors to manage any requests they have received.

### **Method Summary**

public <code>Boolean</code>	<a href="#">acceptRequest()</a>	Accepts a selected request, notifying the employee of the decision.
public <code>Boolean</code>	<a href="#">deleteRequest()</a>	Deletes a selected request.
public <code>Requests[]</code>	<a href="#">loadEmpRequests()</a>	Gets the list of all current requests that have been sent by employees.
public <code>Boolean</code>	<a href="#">updateRequest()</a>	Updates a request, allowing notifications to be sent to the requestee that

### **Method Detail**

#### **acceptRequest**

public `Boolean` acceptRequest()

Accepts a selected request, notifying the employee of the decision.

---

#### **deleteRequest**

public `Boolean` deleteRequest()

Deletes a selected request. This is equivalent to rejecting a request, and the employee that sent it will be notified accordingly.

---

#### **loadEmpRequests**

public `Requests[]` loadEmpRequests()

Gets the list of all current requests that have been sent by employees.

---

## **updateRequest**

public Boolean updateRequest()

Chapter 15 - Section: 15.1. Interface

Updates a request, allowing notifications to be sent to the requestee that

## **Class ScheduleWidget**

package: [Interface](#)

```
Interface.Widget
  |
  +-Interface.ScheduleWidget
```

public class ScheduleWidget

**Extends:**

[Interface.Widget](#)

Provides the user interface for creating and managing schedules that exist within the system.

### **Method Summary**

public void	<a href="#">dispDailySched()</a>	Displays the schedule in a daily format.
public void	<a href="#">dispWeeklySched()</a>	Displays the schedule in a weekly format.
public void	<a href="#">loadShifts()</a>	Loads all shifts that are contained within a given schedule.

### **Method Detail**

#### **dispDailySched**

public void dispDailySched()

Displays the schedule in a daily format.

#### **dispWeeklySched**

public void dispWeeklySched()

Displays the schedule in a weekly format.

#### **loadShifts**

public void loadShifts()

Loads all shifts that are contained within a given schedule.

## **Class SkillsWidget**

package: [Interface](#)

```
Interface.Widget
  |
  +-Interface.SkillsWidget
```

public class SkillsWidget

**Extends:**

[Interface.Widget](#)

Provides functionality for adding, editing and deleting skills.

<b>Method Summary</b>	
Chapter 15 - Section: 15.1.	<p><b>public void</b> <code>confirmDelete()</code> Prompts the user to confirm that they wish to delete the skill.</p>
	<p><b>public void</b> <code>loadSkills()</code> Populates the existing skills list with skills from the database.</p>
	<p><b>public void</b> <code>showEditView()</code> Shows the edit skill view with all fields blank for the user to create a new skill.</p>
	<p><b>public void</b> <code>showEditView(Skill p0)</code> Shows the edit skill view with all fields filled with data from the chosen skill.</p>
	<p><b>public void</b> <code>showSkillDeleted()</code> Display an alert, showing the user that the chosen skill was deleted.</p>
	<p><b>public void</b> <code>showSkillSaved()</code> Display an alert, showing the user that the skill was saved or created.</p>

## **Method Detail**

### **confirmDelete**

```
public void confirmDelete()
Prompts the user to confirm that they wish to delete the skill.
```

### **loadSkills**

```
public void loadSkills()
Populates the existing skills list with skills from the database.
```

### **showEditView**

```
public void showEditView(Skill p0)
Shows the edit skill view with all fields filled with data from the chosen skill.
```

### **showEditView**

```
public void showEditView()
Shows the edit skill view with all fields blank for the user to create a new skill.
```

### **showSkillDeleted**

```
public void showSkillDeleted()
Display an alert, showing the user that the chosen skill was deleted.
```

### **showSkillSaved**

```
public void showSkillSaved()
Display an alert, showing the user that the skill was saved or created.
```

## **Class Widget**

package: [Interface](#)

```
public abstract class Widget
```

**Implements:**

[Interface.WidgetInterface](#)

Default widget that can not itself be loaded, but must be extended by all other widgets so that they all contain the same base functionality.

<b>Method Summary</b>	
Chapter 15 - Section: 15.1.	<p>public <code>void</code> <code>load()</code> Creates a new widget instance and allows it to be referenced and displayed.</p>
	<p>public <code>void</code> <code>unload()</code> Removes a widget from the system, releasing all resources it is using and saving any related data.</p>

## Method Detail

### load

```
public void load()
```

Creates a new widget instance and allows it to be referenced and displayed.

### unload

```
public void unload()
```

Removes a widget from the system, releasing all resources it is using and saving any related data.

## Interface Detail

### Interface WidgetInterface

package: [Interface](#)

#### All Known Implementing Classes:

[Widget](#)

```
public interface WidgetInterface
```

Interface object that determines what methods all widgets must have in the system.

<b>Method Summary</b>	
	<p>public <code>void</code> <code>load()</code> Creates a new widget instance and allows it to be referenced and displayed.</p>
	<p>public <code>void</code> <code>unload()</code> Removes a widget from the system, releasing all resources it is using and saving any related data.</p>

## Method Detail

### load

```
public void load()
```

Creates a new widget instance and allows it to be referenced and displayed.

### unload

```
public void unload()
```

Removes a widget from the system, releasing all resources it is using and saving any related data.

## 15.2 Persistence

### Class Detail

#### Class EmployeeBroker

package: [Persistence](#)

public class EmployeeBroker

Implements:

[Persistence.Broker](#)

Provides functionality for retrieving, creating, updating and deleting Employee data from the database.

#### Method Summary

public Employee[]	getSupervisorEmps(Employee parameter0)
-------------------	--

#### Method Detail

##### getSupervisorEmps

public Employee[] getSupervisorEmps(Employee parameter0)

#### Class GlobalSettingsBroker

package: [Persistence](#)

public class GlobalSettingsBroker

Implements:

[Persistence.Broker](#)

Provides functionality for retrieving, creating, updating and deleting GlobalSettings data from the database.

#### Class LocationBroker

package: [Persistence](#)

public class LocationBroker

Implements:

[Persistence.Broker](#)

Provides functionality for retrieving, creating, updating and deleting Location data from the database.

#### Class NotificationBroker

package: [Persistence](#)

public class NotificationBroker

Implements:

[Persistence.Broker](#)

Provides functionality for retrieving, creating, updating and deleting Notification data from the database.

#### Class PermissionSetBroker

package: [Persistence](#)

public class PermissionSetBroker

**Implements:**[Persistence.Broker](#)

## Chapter 15 - Section 15.2 - Persistence

Provides functionality for retrieving, creating, updating and deleting PermissionSet data from the database.

### Class PositionBroker

package: [Persistence](#)

```
public class PositionBroker
```

**Implements:**[Persistence.Broker](#)

Provides functionality for retrieving, creating, updating and deleting Position data from the database.

### Class ScheduleBroker

package: [Persistence](#)

```
public class ScheduleBroker
```

**Implements:**[Persistence.Broker](#)

Provides functionality for retrieving, creating, updating and deleting Schedule data from the database.

### Class ScheduleTemplateBroker

package: [Persistence](#)

```
public class ScheduleTemplateBroker
```

**Implements:**[Persistence.Broker](#)

Provides functionality for retrieving, creating, updating and deleting ScheduleTemplate data from the database.

### Class SkillBroker

package: [Persistence](#)

```
public class SkillBroker
```

**Implements:**[Persistence.Broker](#)

Provides functionality for retrieving, creating, updating and deleting Position data from the database.

## Interface Detail

### Interface Broker

package: [Persistence](#)

**All Known Implementing Classes:**

[EmployeeBroker](#), [GlobalSettingsBroker](#), [LocationBroker](#), [NotificationBroker](#), [PermissionSetBroker](#), [PositionBroker](#), [ScheduleBroker](#), [ScheduleTemplateBroker](#), [SkillBroker](#)

```
public interface Broker
```

Broker interface, determining global functions that all broker classes should contain. Provides functionality for retrieving, creating, updating and deleting data from the database.

<b>Method Summary</b>	
Chapter 15 - Section: 15.2.	<b>closeBroker()</b> Closes the broker and it's accompanying connection to the database, ensuring all pending transactions are completed first.
	<b>delete(Object parameter0)</b> Deletes the record for the given object from the database.
	<b>get(Object parameter0)</b> Retrieves the type of object from the database.
	<b>openBroker()</b> Opens the broker and prepares it's connection to the database.
	<b>update(Object parameter0)</b> Applies any changes in the object to the database, or saves it as a new record if it does not yet exist.

## **Method Detail**

### **closeBroker**

```
public boolean closeBroker()
```

Closes the broker and it's accompanying connection to the database, ensuring all pending transactions are completed first.

### **delete**

```
public boolean delete(Object parameter0)
```

Deletes the record for the given object from the database.

### **get**

```
public Object get(Object parameter0)
```

Retrieves the type of object from the database. The search criteria is given as a new object of that type, with all attributes empty except for those that will be used to match results. This allows for single objects to be retrieved, or any combination of search terms to be used through the same command.

### **openBroker**

```
public boolean openBroker()
```

Opens the broker and prepares it's connection to the database. Once complete, the broker can be used to interact with the database.

### **update**

```
public boolean update(Object parameter0)
```

Applies any changes in the object to the database, or saves it as a new record if it does not yet exist.

## 15.3 Application Domain

### Package *Application*

#### **Class Detail**

##### **Class *GenerateReport***

package: [Application](#)

```
public class GenerateReport
```

Provides functionality for all actions used by the Generate Report and Distribute Report use cases.

##### **Method Summary**

public <a href="#">void</a>	<a href="#"><u>convertPrintRep()</u></a>	Converts a generated report into a print-friendly format.
public <a href="#">void</a>	<a href="#"><u>genResRep()</u></a>	Generates a new employee resource report.
public <a href="#">void</a>	<a href="#"><u>genSchRep()</u></a>	Generates a new schedule report.
public <a href="#">void</a>	<a href="#"><u>genUseRep()</u></a>	Generates a employee system usage report.
public <a href="#">void</a>	<a href="#"><u>getResStats()</u></a>	Calculates employee resource statistics, used when generating an employee resource report.
public <a href="#">void</a>	<a href="#"><u>getSchStats()</u></a>	Calculates scheduling statistics, used when generating a schedule report.
public <a href="#">void</a>	<a href="#"><u>getUseStats()</u></a>	Generates employee usage statistics, used when generating an employee usage report.

##### **Method Detail**

###### **convertPrintRep**

```
public void convertPrintRep()
```

Converts a generated report into a print-friendly format.

---

###### **genResRep**

```
public void genResRep()
```

Generates a new employee resource report.

---

###### **genSchRep**

```
public void genSchRep()
```

Generates a new schedule report.

---

###### **genUseRep**

```
public void genUseRep()
```

Generates a employee system usage report.

### **getResStats**

```
public void getResStats()  
    Calculates employee resource statistics, used when generating an employee resource report.
```

### **getSchStats**

```
public void getSchStats()  
    Calculates scheduling statistics, used when generating a schedule report.
```

### **getUseStats**

```
public void getUseStats()  
    Generates employee usage statistics, used when generating an employee usage report.
```

## **Class Login**

package: **Application**

```
public class LogIn
```

Provides functionality for all actions used by the LogIn use case.

### **Method Summary**

public boolean	<b>checkEmps()</b> Verifies the employee credentials, comparing the entered username and password against what was in the database.
----------------	--

## **Method Detail**

### **checkEmps**

```
public boolean checkEmps()  
    Verifies the employee credentials, comparing the entered username and password against what was in the database.
```

## **Class MaintainEmployee**

package: **Application**

```
public class MaintainEmployee
```

Provides functionality for all actions used by the Maintain Employee use case.

### **Method Summary**

public void	<b>pendingAvailability(Schedule p0)</b> Send a schedule into pending state for a change in availability.
public void	<b>setAvailability(Schedule p0)</b> Sets an employee's availability with a new value.

## **Method Detail**

### **pendingAvailability**

```
public void pendingAvailability(Schedule p0)  
    Send a schedule into pending state for a change in availability.
```

**setAvailability**

```
public void setAvailability(Schedule p0)
```

Sets an employee's availability with a new value.

**Class MaintainLocation**

package: [Application](#)

```
public class MaintainLocation
```

Provides functionality for all actions used by the Maintain Location use case.

**Method Summary**

public boolean	<a href="#">validate(Location p0)</a>
----------------	---------------------------------------

Confirms that the location has no errors that would prevent it from being written to the database.

**Method Detail****validate**

```
public boolean validate(Location p0)
```

Confirms that the location has no errors that would prevent it from being written to the database. Will also check if the location already exists in the database.

**Class MaintainPosition**

package: [Application](#)

```
public class MaintainPosition
```

Provides functionality for all actions used by the Maintain Position use case.

**Method Summary**

public boolean	<a href="#">validate(Position p0)</a>
----------------	---------------------------------------

Confirms that the position has no errors that would prevent it from being written to the database.

**Method Detail****validate**

```
public boolean validate(Position p0)
```

Confirms that the position has no errors that would prevent it from being written to the database. Will also check if the position already exists in the database.

**Class MaintainSchedule**

package: [Application](#)

```
public class MaintainSchedule
```

Provides functionality for all actions used by the Maintain Schedule use case.

**Method Summary**

public void	<a href="#">clearSchedule(Schedule p0)</a>
-------------	--

Clears a schedule by removing all shifts.

protected void	<a href="#">createSchedule(String p0, Shift[] p1)</a>
----------------	---

Sets the schedule for a certain day (the string param is a date) with the shift[] 's specified.

**Method Summary**

<p>public <code>void</code> <a href="#"><code>getFutureSchedule(int p0)</code></a></p>	Returns a schedule that is saved but not active.
<p>public <code>Schedule</code> <a href="#"><code>mergeSchedule(Schedule p0)</code></a></p>	Takes an array of schedules and adds the shifts of all into one, which is returned.

**Method Detail****clearSchedule**

```
public void clearSchedule(Schedule p0)
```

Clears a schedule by removing all shifts.

**createSchedule**

```
protected void createSchedule(String p0, Shift[] p1)
```

Sets the schedule for a certain day (the string param is a date) with the shift[] 's specified.

**getFutureSchedule**

```
public void getFutureSchedule(int p0)
```

Returns a schedule that is saved but not active. Returns null if no future schedule exists.

**mergeSchedule**

```
public Schedule mergeSchedule(Schedule p0)
```

Takes an array of schedules and adds the shifts of all into one, which is returned.

**Class MaintainSkill**

package: [Application](#)

```
public class MaintainSkill
```

Provides functionality for all actions used by the Maintain Skill use case.

**Method Summary**

<p>public <code>boolean</code> <a href="#"><code>validate(Skill p0)</code></a></p>	Confirms that the skill has no errors that would prevent it from being written to the database.
--	---

**Method Detail****validate**

```
public boolean validate(Skill p0)
```

Confirms that the skill has no errors that would prevent it from being written to the database. Will also check if the skill already exists in the database.

**Class ReviewRequest**

package: [Application](#)

```
public class ReviewRequest
```

Provides functionality for all actions used by the Review Employee Requests use case.

### **Class *SendNotification***

package: [Application](#)

public class SendNotification

Provides functionality for all actions used by the Send Notification use case.

#### **Method Summary**

public boolean	<a href="#"><u>sendNotification(Notification parameter0)</u></a>
	Prepares a new notification object to be sent to an employee.

#### **Method Detail**

##### **sendNotification**

public boolean sendNotification([Notification](#) parameter0)

Prepares a new notification object to be sent to an employee.

## 15.4 Business Domain

### Package *Business*

#### **Class Detail**

##### **Class Employee**

package: **Business**

```
public class Employee
```

Contains information on an employee within the system including username and passwords to log in, and basic employee information such as first and last name.

#### **Field Summary**

private boolean	<u>active</u> The active state of the employee.
private Date	<u>birthDate</u> The birthday of the employee.
private String	<u>email</u> The email at which the employee can be contacted.
private int	<u>empID</u> The ID number of the employee within the system.
private String	<u>familyName</u> The family name of the employee.
private String	<u>givenName</u> The given name of the employee.
private Date	<u>lastLogin</u> Holds the last day and time that the employee logged in.
private Location	<u>lnkLocation</u>
private Notification	<u>lnkNotification</u> Employees will receive and view notifications that are sent automatically, or manually sent by their supervisor.
private PermissionSet	<u>lnkPermissionLevel</u> All employees have an assigned permission set that determines employee-specific permissions within the system regarding what they can and can't access, as well as numerical values relating to time off.
private Position	<u>lnkPosition</u>
private Skill	<u>lnkSkill</u>
private String	<u>password</u> The password that the employee uses to log into the system.
private PermissionSet	<u>pLevel</u> The current permission level of the employee, tied to a permission set.
private Location	<u>prefLocation</u> The preferred location within the business that the employee will be working in.
private Position	<u>prefPosition</u> The preferred position that the employee will be working in.
private int	<u>supervisorID</u> The supervisor of the employee.
private String	<u>username</u> The username that the employee uses to log into the system.

<b>Method Summary</b>	
public <a href="#">Date</a>	<a href="#">getLoginTime()</a> Retrieves the last time that the employee logged into the system.
public <a href="#">WorkingShift</a>	<a href="#">getWorkingTimes()</a> Retrieves the list of active working shifts that the employee is currently scheduled for.

### **Field Detail**

#### **active**

private [boolean](#) active

The active state of the employee. When active, an employee can be included in shifts and schedules.

An inactive employee is equivalent to one that has been deleted, and is kept in the system so they may still be included in reports and auditing procedures.

#### **birthDate**

private [Date](#) birthDate

The birthday of the employee. Not required by default.

#### **email**

private [String](#) email

The email at which the employee can be contacted.

#### **empID**

private [int](#) empID

The ID number of the employee within the system.

#### **familyName**

private [String](#) familyName

The family name of the employee.

#### **givenName**

private [String](#) givenName

The given name of the employee.

#### **lastLogin**

private [Date](#) lastLogin

Holds the last day and time that the employee logged in.

#### **InkLocation**

private [Location](#) InkLocation

#### **InkNotification**

private [Notification](#) InkNotification

Employees will receive and view notifications that are sent automatically, or manually sent by their supervisor.

#### **InkPermissionLevel**

private [PermissionSet](#) InkPermissionLevel

All employees have an assigned permission set that determines employee-specific permissions within the system regarding what they can and can't access, as well as numerical values relating to time off.

---

**InkPosition**

private [Position](#) InkPosition

---

**InkSkill**

private [Skill](#) InkSkill

---

**password**

private [String](#) password

The password that the employee uses to log into the system.

---

**plevel**

private [PermissionSet](#) plevel

The current permission level of the employee, tied to a permission set.

---

**prefLocation**

private [Location](#) prefLocation

The preferred location within the business that the employee will be working in.

---

**prefPosition**

private [Position](#) prefPosition

The preferred position that the employee will be working in.

---

**supervisorID**

private [int](#) supervisorID

The supervisor of the employee.

---

**username**

private [String](#) username

The username that the employee uses to log into the system.

---

**Method Detail****getLoginTime**

public [Date](#) getLoginTime()

Retrieves the last time that the employee logged into the system.

---

**getWorkingTimes**

public [WorkingShift](#) getWorkingTimes()

Retrieves the list of active working shifts that the employee is currently scheduled for.

---

**Class GlobalSettings**

package: [Business](#)

public class GlobalSettings

Used to keep track of global attributes that apply system-wide to every employee or schedule. There is only ever one Global Settings object stored in the database.

**Field Summary**private int **allowableDaysOff**

The maximum number of days off that any employee can take.

**Field Detail****allowableDaysOff**

private int allowableDaysOff

The maximum number of days off that any employee can take.

**Class Location**package: **Business**

public class Location

Locations are used to keep track of specific real-world places. An employee may have a preferred location to work in, used by the schedule and report generators.

**Field Summary**private String **description**

The description of the location, if desired.

private String **name**

The name of the location, uniquely identifying it apart from other locations.

**Field Detail****description**

private String description

The description of the location, if desired.

**name**

private String name

The name of the location, uniquely identifying it apart from other locations.

**Class Notification**package: **Business**

public class Notification

A notification is a type of message object that is sent to employees. These can be sent out automatically to inform an employee of the status of requests they have made, and can also be sent manually if a supervisor wishes to send a message to his/her employees.

**Field Summary**private String **message**

Contains the message text of the notification.

private int **recipientID**

The ID of the employee that the notification is being sent to.

private int **senderID**

The ID of the employee that sent the notification, or 0 if sent by the system.

private Date **sentTime**

The time and day that the notification was sent.

**Field Summary**

private <code>String</code>	<b>type</b> Contains the type of notification that this object represents, such as an accept/reject notification for an employee request, or a general notification sent by the supervisor.
private <code>boolean</code>	<b>viewed</b> Shows whether or not the intended recipient of this notification has opened and viewed it.

**Field Detail****message**private `String` message

Contains the message text of the notification.

**recipientID**private `int` recipientID

The ID of the employee that the notification is being sent to.

**senderID**private `int` senderID

The ID of the employee that sent the notification, or 0 if sent by the system.

**sentTime**private `Date` sentTime

The time and day that the notification was sent.

**type**private `String` type

Contains the type of notification that this object represents, such as an accept/reject notification for an employee request, or a general notification sent by the supervisor.

**viewed**private `boolean` viewed

Shows whether or not the intended recipient of this notification has opened and viewed it.

**Class PermissionSet**package: **Business**

public class PermissionSet

All employees have a set of permission settings that determine what levels of access they have within the system.

**Field Summary**

private <code>boolean</code>	<b>canAccessReports</b> Sets whether or not the employee can access the reporting system, allowing them to both generate and distribute reports.
private <code>boolean</code>	<b>canChangePermissions</b> Sets whether or not the employee is able to change the permission settings of other employees that are below their permission level.
private <code>boolean</code>	<b>canEditSched</b> Sets whether or not the employee is able to edit schedules, either by creating new ones or by editing future schedules that have not yet come into effect.

<b>Field Summary</b>	
private boolean	<u>canReadLogs</u> Sets whether or not the employee is able to view system logs.
private boolean	<u>canReadOldSched</u> Employee is able to access archived schedules that have ended before the current date.
private boolean	<u>canReadSched</u> Employee can access times on a schedule under their name and lower levels for the current date or period.
private boolean	<u>canRequestDaysOff</u> Sets whether or not the employee is able to request days off.
private boolean	<u>canSendNotifications</u> Sets whether or not the employee can send notifications which are messages internal to the system that can be sent to specific employees, workgroups, or job types.
private boolean	<u>canTakeEmergencyDays</u> Sets whether or not the employee is able to take emergency days off.
private boolean	<u>canTakeVacations</u> Sets whether or not the employee is able to take vacation days.
private boolean	<u>canViewInactiveEmps</u> Sets whether or not the employee can view details on inactive employees who are not currently working, such as ones that have left the company.
private boolean	<u>canViewResources</u> Sets whether or not the employee is able to view employee resources, such as employees that are available, booked off, or are already working at a given time.
private int	<u>maxDaysOff</u> Sets the maximum number of days off that an employee can request for a given schedule period.
private int	<u>maxVacationDays</u> Determines the maximum number of vacation days that an employee can take off over a schedule period, assuming they have the required permission for vacation days.
private int	<u>bLevel</u> Each collection of unique permission settings are tied to a specific permission level number.
private int	<u>preferredRank</u> Starting with 0 as the highest priority employee and then incrementing, this differentiates seniority within the same level.
private int	<u>trusted</u> Similar to a permission promotion (which cannot be given by self), trusted is a permission that allows any employee to perform actions that affect the next highest level without requiring authorization from a superior.

### Field Detail

#### canAccessReports

private boolean canAccessReports

Sets whether or not the employee can access the reporting system, allowing them to both generate and distribute reports.

#### canChangePermissions

private boolean canChangePermissions

Sets whether or not the employee is able to change the permission settings of other employees that are below their permission level.

#### canEditSched

private boolean canEditSched

Sets whether or not the employee is able to edit schedules, either by creating new ones or by editing future schedules that have not yet come into effect.

**canReadLogs**

private boolean canReadLogs

Sets whether or not the employee is able to view system logs.

---

**canReadOldSched**

private boolean canReadOldSched

Employee is able to access archived schedules that have ended before the current date.

---

**canReadSched**

private boolean canReadSched

Employee can access times on a schedule under their name and lower levels for the current date or period.

---

**canRequestDaysOff**

private boolean canRequestDaysOff

Sets whether or not the employee is able to request days off.

---

**canSendNotifications**

private boolean canSendNotifications

Sets whether or not the employee can send notifications which are messages internal to the system that can be sent to specific employees, workgroups, or job types.

---

**canTakeEmergencyDays**

private boolean canTakeEmergencyDays

Sets whether or not the employee is able to take emergency days off. This will be on by default, but can be turned off if desired.

---

**canTakeVacations**

private boolean canTakeVacations

Sets whether or not the employee is able to take vacation days.

---

**canViewInactiveEmps**

private boolean canViewInactiveEmps

Sets whether or not the employee can view details on inactive employees who are not currently working, such as ones that have left the company. This is primarily used for reports and auditing.

---

**canViewResources**

private boolean canViewResources

Sets whether or not the employee is able to view employee resources, such as employees that are available, booked off, or are already working at a given time.

---

**maxDaysOff**

private int maxDaysOff

Sets the maximum number of days off that an employee can request for a given schedule period. This value will only be accessed if the employee already has permission to request days off.

---

**maxVacationDays**

private int maxVacationDays

Determines the maximum number of vacation days that an employee can take off over a schedule period, assuming they have the required permission for vacation days.

---

**plevel**

```
private int plevel
```

Each collection of unique permission settings are tied to a specific permission level number. Employee permissions are determined by setting them to a permission level, rather than having to set all permission settings individually for each employee.

**preferredRank**

```
private int preferredRank
```

Starting with 0 as the highest priority employee and then incrementing, this differentiates seniority within the same plevel.

**trusted**

```
private int trusted
```

Similar to a permission promotion (which canot be given by self), trusted is a permission that allows any employee to perform actions that affect the next highest level without requiring authorization from a superior. However, a notification will be sent notifying the supervisor (n/a if highest plevel) explaining actions. This only applies to permissions that are currently enabled. Ex: If an employee cannot book days off, a trusted employee canot book days off either.

**Class Position**

package: **Business**

```
public class Position
```

The primary object used to match an employee with a shift. Positions will in most cases be equivalent to a specific job title in the business.

**Field Summary**

private String	<b>description</b>
The description of the position, if desired.	
private Skill	<b>lnkSkill</b>
private String	<b>name</b>
The name of the position, uniquely identifying it apart from other positions.	

**Field Detail****description**

```
private String description
```

The description of the position, if desired.

**lnkSkill**

```
private Skill lnkSkill
```

**name**

```
private String name
```

The name of the position, uniquely identifying it apart from other positions.

**Class Schedule**

package: **Business**

```
public class Schedule
```

A group of working shifts that apply within a given range of dates.

### Field Summary

private int	<code>creatorID</code>	The ID of the employee that created the schedule.
private Date	<code>endDate</code>	The end date for when this schedule will no longer be in effect for all applicable employees.
private Employee	<code>lnkEmployee</code>	
private Date	<code>startDate</code>	The start date for when this schedule will be in effect for all applicable employees.
private WorkingShift	<code>workingShifts</code>	The collection of working shifts that will be active between the dates set by the schedule.

### Method Summary

public WorkingShift	<code>getWorkingShifts()</code>	Retrieves an array of all working shifts held by the schedule.
---------------------	---------------------------------	--

### Field Detail

#### creatorID

private int creatorID

The ID of the employee that created the schedule.

#### endDate

private Date endDate

The end date for when this schedule will no longer be in effect for all applicable employees.

#### lnkEmployee

private Employee lnkEmployee

#### startDate

private Date startDate

The start date for when this schedule will be in effect for all applicable employees.

#### workingShifts

private WorkingShift workingShifts

The collection of working shifts that will be active between the dates set by the schedule.

### Method Detail

#### getWorkingShifts

public WorkingShift getWorkingShifts()

Retrieves an array of all working shifts held by the schedule.

### Class ScheduleTemplate

package: **Business**

public class ScheduleTemplate

A grouping of shifts that serve as a template for new schedules, determining what positions and how many are required for given times during the week. This is used as the basis for auto-generating a new schedule.

### Field Summary

private int	<a href="#">creatorID</a>	The employee who created the Schedule Template.
private <a href="#">Employee</a>	<a href="#">lnkEmployee</a>	
private <a href="#">Shift</a>	<a href="#">lnkShift</a>	
private <a href="#">Shift</a>	<a href="#">shifts</a>	The collection of shifts that belong to the Schedule Template.

### Field Detail

#### creatorID

```
private int creatorID
```

The employee who created the Schedule Template.

#### lnkEmployee

```
private Employee lnkEmployee
```

#### lnkShift

```
private Shift lnkShift
```

#### shifts

```
private Shift shifts
```

The collection of shifts that belong to the Schedule Template.

### Class Shift

package: [Business](#)

```
public class Shift
```

Contains a list of what positions must be filled by employees within a given range of time.

### Field Summary

private <a href="#">Time</a>	<a href="#">endTime</a>	The time at which work ends for the employees assigned to the shift.
private <a href="#">ShiftPos</a>	<a href="#">lnkShiftPos</a>	
private <a href="#">ShiftPos</a>	<a href="#">positions</a>	The positions that are required for the shift.
private <a href="#">Time</a>	<a href="#">startTime</a>	The time at which work begins for the employees assigned to the shift.

### Field Detail

#### endTime

```
private Time endTime
```

The time at which work ends for the employees assigned to the shift.

#### lnkShiftPos

```
private ShiftPos lnkShiftPos
```

#### positions

```
private ShiftPos positions
```

## Chapter 15 - Section: 15.4. Business Domain

The positions that are required for the shift.

### **startTime**

private [Time](#) startTime

The time at which work begins for the employees assigned to the shift.

### **Class ShiftPos**

package: [Business](#)

public class ShiftPos

Used by Shifts to record what positions they required, as well as a count of how many of each unique position they require.

#### **Field Summary**

private int	<a href="#">count</a>
	The number of people that must fill this position for the containing shift.
private <a href="#">Position</a>	<a href="#">lnkPosition</a>
private <a href="#">Position</a>	<a href="#">position</a>
	The position that is required.

#### **Field Detail**

##### **count**

private int count

The number of people that must fill this position for the containing shift.

##### **lnkPosition**

private [Position](#) lnkPosition

##### **position**

private [Position](#) position

The position that is required.

### **Class Skill**

package: [Business](#)

public class Skill

Skills are assigned to both Employees and Positions. Together, they keep track of what skills an employee must have to work a certain position, and are checked against an employee's skill list when an attempt is made to assign a preferred position to them.

#### **Field Summary**

private String	<a href="#">description</a>
	The description of the skill, if desired.
private String	<a href="#">name</a>
	The name of the skill, uniquely identifying it apart from other skills.

#### **Field Detail**

##### **description**

private String description

The description of the skill, if desired.

**name**

```
private String name
```

The name of the skill, uniquely identifying it apart from other skills.

**Class WorkingShift**

package: **Business**

```
public class WorkingShift
```

Working shifts determine which employees are working at what times for a specific schedule.

**Field Summary**

private Employee	<b>employees</b>	The employees that will be working during the times set by the working shift.
private Time	<b>endTime</b>	The end time of this working shift.
private Employee	<b>lnkEmployee</b>	
private Schedule	<b>Inkschedule</b>	
private Time	<b>startTime</b>	The starting time of this working shift.

**Method Summary**

public Time[]	<b>getTimes()</b>	Returns an array containing the start and end time of a working shift.
---------------	-------------------	--

**Field Detail****employees**

```
private Employee employees
```

The employees that will be working during the times set by the working shift.

**endTime**

```
private Time endTime
```

The end time of this working shift.

**InkEmployee**

```
private Employee InkEmployee
```

**InkSchedule**

```
private Schedule InkSchedule
```

**startTime**

```
private Time startTime
```

The starting time of this working shift.

**Method Detail****getTimes**

```
public Time[] getTimes()
```

Returns an array containing the start and end time of a working shift.

# Part V

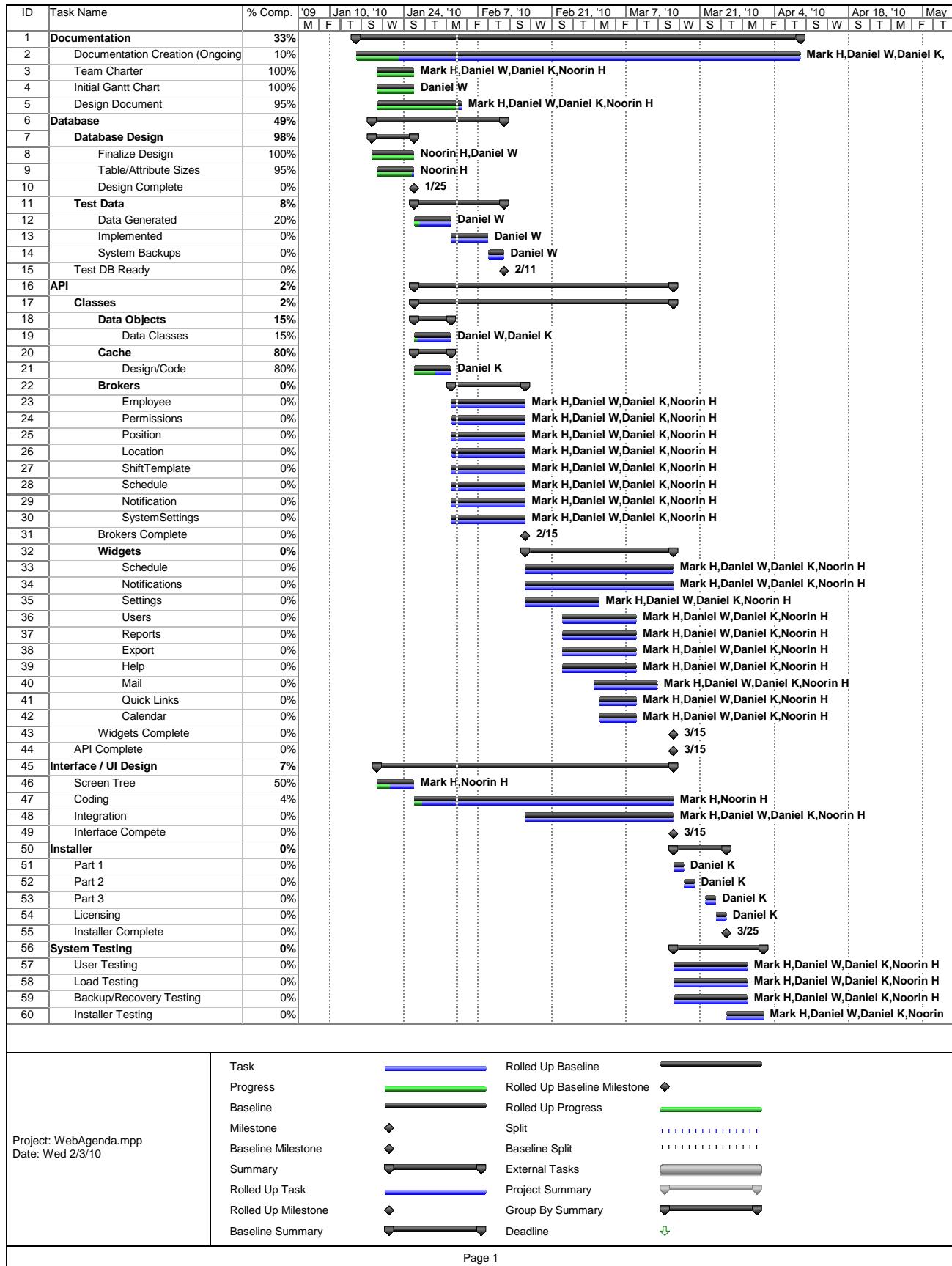
# Project Management

# **Chapter 16**

## **Schedule**

### **16.1 Gantt Chart**

## Chapter 16 - Section: 16.1. Gantt Chart



## 16.2 2009 -10-08 Requirements Documentation

The requirements documentation is a compilation of details on the WebAgenda project. This document is used to outline the specific user requirements of the project, the **system requirements** of the project, the **system interface requirements** of the project, and the project management details of the project.

Inside the user requirements portion of the document, the business overview and objectives need to be defined. This includes the nature of the clients business, the mission of the business and all other relative information in regards to the clients business model. A project overview should also be included in the documentation. It should include things like a statement of the current problem or issue the client has with their current system. The **project scope** should also be included with the project overview. This should include things like what the system needs to be able to do and include a summary of the major functional requirements of the system. The system environment will also be included in the project overview. The scope of the project details what the system will be able to do and what it will not be able to do. In other words, the scope just outlines the project boundaries. The system environment is also included in the project overview and details where the project will be deployed upon **beta release** and **final releases**. The project overview will include a description of the current system and what the type of functionality it has in addition to its downfalls and the desired improvements.

The next section of the requirements document is the project system requirements. The first point of the system requirements is the fact-finding methodology. This will include a description of the approach project team members took to find facts regarding the system. Any questions that were asked in the interview process will also be included in the fact-finding methodology section of the system requirements. A use case diagram that was derived from the fact-finding process should also be included in the system requirements section. A description of the use case elements should also be included in this section, things like actor definition, and roles should be defined in this section. A non-functional requirement should also be derived here from the use case and the fact-finding section of the system requirements. A system interface requirements section should also be included and derived from the use case and fact-finding sections. **Maintainability** and Administration requirements should also be included in this section. This means things like maintenance and backup schedules, and all administration aspects that the system needs to include to have the full functionality that the current system has. Usability requirements should also be included in this section. The usability requirements includes things like how advanced the users are at using software/computers in general. This should also include how many people will be using the software and how familiar they are with software/computers in general.

## **16.3 2009-11-05 Requirements Analysis**

The requirements analysis is incorporated into the project planning stages of the project by identifying how close the requirements are to the actual project features. Once the project team meets with the client and determines exactly what the client wants, the updated features list then is located in the requirements analysis. Changes to the original requirements documentation will go into the requirements Analysis report. The client should receive both a copy of the requirements documentation and the requirements analysis to ensure that it is up to date and accurate.

## **16.4 2009-12-17 Design Documentation**

The design documentation is the final piece of documentation in the planning stages of the project. It is an all inclusive summary of all the clients technical and informational requirements for the project. This includes budget, time constraints, and any final technical specifications for the project. The design documentation should be used by the developers as a reference when building the project. If any questions arise as to features of the project the developers should consult the design documentation for conclusion. It is the most up to date compilation from the client and the developers about the project.

Included in the design document is design related functionality. How the system needs to be implemented needs to be considered at this stage. Information regarding the human computer interface needs to be seriously considered here. All interface related documentation needs to be reflected in both the sequence diagrams and the problem domain class diagram.

Hardware architecture needs to be described in this document. Minimum hardware specifications for properly running the software need to be outlined in this document. Software architecture also needs to be implemented. Minimum software requirements need to be outlined in this document as well. Any software dependencies need to be outlined and documented in this document so the user can see what software they need in order to run the system.

## 16.5 2009-01-30 Alpha Release

The pre-beta release, known as an **alpha release**, is a basic functional release of the software. The software may have next to no features of the full release and the ones that do exist are partially tested and potentially have **bugs**. The alpha release is used to start working out bugs and get any input from the client as to features that need to be modified or changed to meet the clients needs. The interface may need some changes at this stage to incorporate other features.

## 16.6 2009-02-28 Beta Release

The beta release will have full functionality of the final software, with the exception of plugins. However, there will probably be bugs in the program. This release of the software is meant to work out any bugs and ensure that the software is acceptably tested. The software at this stage will also get stress tested, meaning that testers and possibly the client need to use the program and purposely try to break it. This way we can find bugs that may not have surfaced at the alpha stage and fix them before the final release. This stage should include any minor adjustments that the client requests.

## 16.7 2009-04-15 Final Release

The final release will be a full featured, and fully tested release. The software will have all the features that the client requested as well as all the other features that the system implements within the scope of the project. Having tested the software at both the alpha release and the beta release, most of the major bugs should be worked out by the final release period. Some minor bugs may still be in the system but they should not influence productivity or stability at all. The software needs to be in a safe usable state. All **encryption** and **SSL** features need to be implemented so that the users can fully use the software without worrying about any security flaws. The software should also be deployment ready, so that it can be installed and go live on the users system.

# Chapter 17

## Team Configuration

The following section provides detail on the team members involved in the project, as well as their current roles. Contact information is included for team members as well as other key stakeholders.

### 17.1 Members And Roles

The following people are the members of the project team. Current role assignments are included and described.

Mark Hazlett

- Team Leader Ensures all team members understand their roles and what work is required from them each week.
- Meeting Organizer Schedules future meetings and ensures all team members are aware and able to attend.

Daniel Kettle

- Standards Checker Ensures all documentation meets the guidelines listed in the Documentation Standards section of the Preface.
- Archivist Handles storage of documentation and other project related files, ensuring they are freely accessible to the team.

Daniel Wehr

- Scribe Distributes audio recordings of all meetings to group members on the same day they were recorded. Meeting minutes will be typed up and submitted to the team for review before they are finalized and included in the project documentation.
- Quality Assurance Ensures non-documentation project files such as source code meet the standards of the team and have been reviewed for efficiency.

## 17.2 Reporting Relationships and Procedures

All group members will be responsible for reporting the status of the project to the client. The reporting position will be filled by individual members, and be rotated through on a regular basis as per agreement with the client. Current reporting periods are once a month, and since they will be in person the reports can be printed and sent electronically. Reports will be written, and will be sent to the client by email or fax. The client will be given time to review the report, after which they will be contacted for their input on the progress that has been made to ensure the project is following their requirements. All reports will be included in the project documentation for later reference.

## 17.3 Contact Information

### **Mark Hazlett**

Phone: (403) 978-8485  
SAIT Email: [Mark.Hazlett@edu.sait.ca](mailto:Mark.Hazlett@edu.sait.ca)  
Personal Email: [markhazlett9@gmail.com](mailto:markhazlett9@gmail.com)

### **Daniel Kettle**

Phone: (403) 390-2478  
SAIT Email: [Daniel.Kettle@edu.sait.ca](mailto:Daniel.Kettle@edu.sait.ca)  
Personal Email: [initial.dann@gmail.com](mailto:initial.dann@gmail.com)

### **Daniel Wehr**

Phone: (403) 389-3155  
SAIT Email: [Daniel.Wehr@edu.sait.ca](mailto:Daniel.Wehr@edu.sait.ca)  
Personal Email: [danwehr@ftml.net](mailto:danwehr@ftml.net)

### **Noorin Hasan**

Phone: (403) 401-6466  
SAIT Email: [Noorin.Hasan@edu.sait.ca](mailto:Noorin.Hasan@edu.sait.ca)  
Personal Email: [noorin.mohammed@hotmail.com](mailto:noorin.mohammed@hotmail.com)

### **Pauline Turnbull**, SAIT Instructor

Phone: (403) 284-8353  
Email: [Pauline.Turnbull@sait.ca](mailto:Pauline.Turnbull@sait.ca)  
Work Address: Room N409C, SAIT Campus  
1301 16th Ave NW  
Calgary, Alberta T2M 0L4

### **Robin Featherstone**, VP of Operations, Deerfoot Inn & Casino

Phone: (403) 284-8353

# Chapter 18

## Project Standards And Procedures

**LaTeX2E**, also known as LaTeX, is a document preparation system for high-quality typesetting. It has been used in various academic institutions and produces standardized and consistant pages based on tags specified in the LaTeX files. It is not a word processor; appearance is generated from the source file the user creates. Final documents will be generated as a pdf file. LaTeX is also free and available for Windows, Mac, and Linux-Unix systems. The current version is 2E; it will be used for generating all digital-based final documentation. Since WebAgenda is licensed under the GPL 2, source code for the documentation in LaTeX will be available. LaTeX will be used for developing the documentation required by the system in forms of both manuals (instructions, support, and overviews) as well as documents we will upload on the server, which should include troubleshooting, tutorials and Frequently Asked Questions for a variety of basic tasks, downloadable copies of the perviously mentioned document manuals, and the scheduling system documents (WebAgenda's plugin and compatibility documents).

**Date Formats** Dates, when inputted or displayed to the user, should be in a Mon-DD-YYYY format. Regardless of whether the code beneath the user translates this to another format, it is irrelevant. (This is subject to change, but allows for a common understanding of dates to this point.)

**Interface:** This layer of the architecture will handle all user related events, interface requirements, displaying functional data, etc. User events are handled through the use of text boxes, buttons, links, radio buttons and other event handlers to accept user input into the system. The user interface handles everything that the user sees and interacts with. The user interface provides a method to send items to the problem domain so that they can be persisted and manipulated. Error messages are also displayed using this layer.

**Problem Domain:** This layer handles all the **API** calls for the system. The API calls allow the interface to grab specific pieces of data to be displayed by the interface. Through the use of API calls, the interface will be able to grab data from the persistence layer and display it using the **interface layer**. (Refer to Appendix B for full API documentation).

**API:** The application programming interface is the main method to access the persistence layer. The API is a set of pre-defined methods that have specific functions to the program. Only developers can add API methods as to keep the integrity of the system.

**Persistence:** The persistence layer is the layer that handles all database access and storage. The database access, inserting, deleting, copying, etc will all be handled by this layer. This layer will be the only layer of the architecture that will handle directly touching and accessing the database.

**Application Logic:** This layer of architecture will handle the manipulation of entities in the system. It will consist of all the controller classes. It will handle the maintain classes as well as any interaction between the presentation and domain layers.

## 18.1 Hardware Architecture

### 18.1.1 Hardware Platform for Production System

The hardware to run the system will be different than that needed to utilize the system. For the server side of the system that will actually be hosting the application and displaying it, the hardware qualifications are much larger. It will require a web server to host the system along with appropriate storage space for a database. The database can be located directly on the server if needed. In this case, where the database is located on the same hard drive running the web server, then the hardware architecture diagram will change to include only a central unit for both the server and the database. In order for the system to be capable of running, the server must meet the following general requirements:

- Capable of running all software on the server
- Capable of running an estimated minimum of 500 simultaneous **HTTP** and **HTTPS** connections
- Capable of staying online for extended periods of time without shutting down (99.9% uptime is the industry standard)

The more hardware that is installed on the server (memory, hard drive, processors), the faster the server should respond. The server's performance should be scalable as the use of the system increases. As per the "Document Purpose" chapter of this document, our scope aims to satisfy approximately 500 employees. The system may suffer no ill performance if this estimate is exceeded and due to possible additional programs that run on a server, exact hardware requirements will vary. As such, there are no specific specifications for running the system as it depends on many external factors aside from the size and labour force of the company or business.

If the database is not installed alongside the webserver, it should be installed on another server. Doing so will require that server to be connected through a network so that the web server can access the database. If the database is located nexted to the web server, then the server needs to have enough storage space and processing power to hold the number of employees in the system and run any web services required. When the number of employees increase, the server will require extra disk space to accomidate them. As usage goes up, the server will access the database more frequently. If the database cannot handle the demand or runs out of space, then the server will need to be reconfigured or extended. The **MySQL** database is licensed under the same license as our product, the GNU Public License, and is efficient and lightweight. It should be more than capable of satisfying the needs of a 500 employee business. If another database is preferred or required, moving data from a MySQL database is a straightforward process.

## 18.2 Hardware Platform for Development System

The hardware platform requirements for the development system are significantly less than those needed for the production system. However, if the developers are testing with large amounts of data and users then the development system requirements would increase proportionally. The development system could be as little as a generic laptop running an HTTP server to a rack mount server with an exponential increase of processors and memory to run the system. The test data that the developers decide to use will help determine the minimum specifications needed to run a development version of the system. The database needed for the development system should reflect the database used in the development server. Depending on the data used in testing, accurate estimates of disk usage and certain performance tweaking can then be used to enhance the production server.

## 18.3 Software Platform

### 18.3.1 Server Software

- **J2EE** 1.5+ of the **Java EE** libraries
- **JSP** and **Servlet** technologies
- **Apache** 2.0+
- MySql 5.0+
- Storage Space dependent on usage
- Bandwidth determined by usage

### 18.3.2 Client Software

#### Browsers

- Safari 3.0+ (Windows or Mac), Safari Mobile
- Chrome 1.0+ (Windows, Mac or Linux)
- IE 7.0+ (Windows)
- Firefox 2.0+
- Opera 8+, Opera Mini
- Konqueror 3.5+
- Javascript Enabled Browser

#### Operating Systems

1. Windows XP+
2. Mac OS X 10.4.8+
3. Linux 2.6.x -based
4. RIM (Blackberry OS)
5. Unix & BSD
6. iPhone OS

The minimum system requirements are a set of guidelines will allow the system to function as smoothly as possible. If the minimum software requirements are not followed, then the system cannot be guaranteed to work properly, although it may still work. The system will be optimized to function on the above specifications to ensure that there are no environment variables that could cause some issues with the system. Again, the system requirements are not completely required, however to get a smoothly running system the minimum specifications are encouraged.

## 18.4 Interaction Model

### 18.4.1 Style

The system will have the same look and feel throughout the entire system. Each screen will be using the same style standards for fonts, text boxes, and general human to system interaction. Using CSS for our style standards, we can define user modifiable values and standard values to keep a consistent look and feel throughout the entire main system as well as third party plug-ins for the system.

### 18.4.2 General Style Guidelines

#### Dialog Boxes

The dialog boxes that appear in the system will all be clearly outlined using a darkening technique for the rest of the system around the dialog box. This allows the user a clear understanding of what window they need to work with at what time.

#### Text Fields

Text fields will be highlighted when the user selects them so that the user can easily understand what field they need to enter data into. If the user enters incorrect data or an error occurs, then a clearly defined error message will occur.

#### Error Messages

If an error message is needed to be displayed, the priority of the error message is taken into account when determining how the error will be displayed. If the error message is a system error and is not recoverable, then the same principles will apply to displaying the error message as a dialog box. If the error is minor, for example if the user enters a piece of data incorrectly, then an error message will be displayed to alert the user what was entered incorrectly.

#### Widgets

A widget will have some areas that are available for user customization; however the overall look and feel will be unavailable to change. This is to ensure that all widgets, both internal and external, have the same look and feel throughout the system.

#### Page

A page is a generic site layout without any widgets installed. A page will have a default header and footer that are customizable by the user. The header will have a logo and a company name that will be able to be customizable by the user. A page is generally just a blank template with the basic areas such as a footer, header, and content area.

### 18.4.3 System Feedback Style

#### Dialog Boxes

Dialog boxes are going to be used for system feedback when the system encounters and irreversible error. The dialog boxes will be used for this type of error because they draw the most attention from the user to address the issue at hand. Included in the dialog box will not only be a description of the error that occurred but also help articles and links to help resources to allow the user to try and fix the problem him/herself.

#### Error Messages

Error messages, like dialog boxes will be used if a minor error occurs to provide feedback to the user. The error messages will be positioned over the area that the error occurred to provide optimal feedback to the user. The error message will display information about what the error is and how to fix the error. The error message will also provide a link to help articles and areas of the system that can help the user solve the issue at hand.

Any piece of information that needs to be immediately communicated to the user with the up most importance will need to be communicated using a dialog box. Otherwise, the error message method should be sufficient for most types of errors.

## 18.4.4 Standards

### Font

As a general rule of thumb, the Helvetica font will be used throughout the system. Helvetica is a professional font that is standard across all browsers and operating systems.

### Colours

Web 2.0 colors are all valid colors to be using the system. As a general theory, the colors used by the system or third party plug-ins need to be compatible with all large browsers and operating systems.

### Widget Outline

The outline of the widgets, or the placeholders, need to be consistent with the standards set during the initial release. However because the users will be able to modify the look and feel of the widgets, the widgets do not have a specific look and feel that they must adhere to.

### Page

A page will determine the look and feel of the widgets and everything that is displayed on the page. The CSS to change the look and feel of the internal widgets will all be handled by how the page looks. If you want to change the look and feel of the widgets, then that must be changed on the page.

**NOTE** The following information is from Desired User Support. Unsure if this should be in its own user-related section or under a ‘project’ heading. Dan suggests that we have a basic help section for new users, and another for developer-related issues.

## 18.5 Desired User Support

Throughout the process of using the system, Users may require help or assistance with certain aspects of the program. The development team would like to have as many help options available to the users as possible to increase ease of use of the system. Available to all users using the system will be the in system help, the wiki, and developer contact.

### System Help

Throughout the system there will be a help menu option to allow users at any time during the system to access the in system help resources. The in system help resources is an internal help resource that comes complete with recent issues, popular issues, frequently asked questions, plug-in, and help articles to help out with aspects of using the system.

### Recent Issues List

The recent issues list will include bugs and problems that users have recently submitted to the system. Users will be able to look through the recent issues list to view if the issue that they are having and see if that is the same issue they are having. If the help article listed is what the users are looking for they can click on the article and it will pull up the article.

### Popular Issues List

The popular issues list will pull a list of the most popular issues, or the issues that are currently getting the most attention. These issues are listed on the main help screen because they are the issues that the users are most likely to come across. The users can then click on any of the articles to view the article on how to solve their issue or perform the action that they are looking to accomplish.

### Frequently Asked Questions

A frequently asked questions section will be available in the help section to allow users to view answers to commonly asked questions about the system. These may change over time with updates and dependent on features.

### Plug-in Specific Help

Plug-in developers have a section in the help section for plug-in specific help articles. Developers can include how to articles specific to their plug-ins here. They can also include frequent issues, and other issues that users of the system may take advantage of.

### Wiki

The wiki is a resource that includes all help articles written by the Web Agenda development team. How to articles, recent issues, HCI guidelines, and all other relevant information can be found on the wiki. Currently the wiki is located at <http://webagenda.googlecode.com/wiki/> but could change at a later point. The wiki will also include developer resources such as design info, API information and plug-in development information.

## 18.6 Screen Descriptions

### 18.6.1 Widgets

**Widgets** - Mini programs that can be run on a webpage. Each widget has different functionality that uses the system's API to get and receive information from the system. Widgets will be openly available to any developer wanting to develop a widget to increase functionality of the system.

**Schedule Summary Widget** - The schedule summary widget is a basic view widget that will allow users to view the current schedule for the date and time by default. If the user that is logged in is a supervisor that has created schedules, this widget will show the user all the schedules for the current date and time for the users inside his/her workgroup.

**Admin Widget** - The admin widget is used to provide an easy access to other screens in the system. By default, this widget will be available only on the main screen but can be embedded on any screen at the users request. The widget provides a number of customizable links that allow the users to navigate through the system to the different pages. The users can choose from a pre-defined group of links to add to the admin widget.

**Sidebar Widget** - The sidebar widget is an embeddable widget that will contain a number of customizable widgets that the user can add to the sidebar. The sidebar is by default visible on every page, however if more screen real-estate is needed then the sidebar can be hidden or even disabled by the user in the user preferences.

**Events Widget** - The events widget is used to send notifications to employees on just about anything that is going on. The widget displays event notifications in real-time to the users. If users do not want real-time updates they have the ability to turn them off at any time.

**Scheduling Widget** - The scheduling widget handles one of two tasks depending on who is logged into the system. If the user logged in has adequate permissions to create a schedule then the user has access to the ability to create schedules using the schedule widget. If the user that is logged in does not have adequate permissions to create schedules then they will be able to view the scheduling widget as a read-only widget that will allow them to check their schedules on a daily, weekly, or monthly basis.

**Browse Users Widget** - The browse users widget is used to browse the users of the system. Employees can only see information that is specified by administrative users. Users will have the ability to filter information based on items such as workgroup, job title, etc to find the information they need. The main function for the browse users widget is to allow employees to retrieve contact and shift information from other employees.

**Search Users Widget** - The search users widget is used to search for a specific employee or shift based on parameters set by the user. Again, information that can be retrieved is determined by the administrative users of the system.

**Settings Widget** - The settings widget is used to display settings information for the users profile and settings for the system. Based on the users permission level, certain settings may be available to change and certain settings may not be. The administrators of the system will be able to change what information if available to be changed at which permission level.

**User Admin Widget** - The user admin widget is used to add, remove and modify users if the user logged in has adequate permissions to do so. The widget will contain user information and allow administrators to add users to the system, update users, and delete or disable users from the system.

**Internal Mail Widget** - The mail widget is the main method of communication for internal communications within the system. If at any point the users would like to send/receive email to any other user in the system, they can do so by using the built in mail widget. It will have the ability to send, receive and update messages from users in the system.

## 18.6.2 Screens

**Dashboard** - The dashboard screen is the main screen after a user successfully logs into the system. The dashboard is the window that displays a summary of information for the user as well as other links to navigate to different areas of the system. Tabs along the top of the dashboard will allow the user to navigate to the main subsection of the site. The main section of the screen will include different widgets to increase functionality of the system. The Dashboard by default will contain multiple widgets: a schedule summary widget, an admin widget, and a sidebar widget which will contain any number of widgets.

**Schedule** - The Schedule section of the system is where the main scheduling widget is located. Here, supervisors will have the ability to create schedules, modify schedules, delete schedules, and perform any other action when it comes to dealing with the scheduling aspect of the system. If a user logs into the system to check his/her schedule then they can view schedules by day, week and month. This screen can contain up to 2 widgets: the schedule widget as well as the sidebar widget which I enabled by default.

**Users** - The users screen is a section of the program that will allow users to browse, search, and communication of the users of the system. The users screen contains a browse users widget, a search users widget and a communication widget that is embedded within the other widgets on the users screen.

**User Administration** - The user admin screen contains a user admin widget that allows users with adequate permissions to add users, delete users, and modify users in the system. The user admin screen is a sub screen of the users screen and is only available if the user logged in has adequate permissions to add, modify and delete users.

**Settings** - The settings screen contains a settings widget that allows the user to modify settings in their profile. The sidebar widget on the settings screen is disabled by default to allow for maximum screen real estate being dedicated to the settings widget. Any changes made to the users settings will be stored for the next time that they login to the system.

**Mail** - The mail screen is the main communication method between users of the system. Although the system has the ability to send emails to external emails, private internal emails are also available to be sent and received using the internal mail widget of the system. The sidebar widget on the mail screen is by default enabled for this screen.

## 18.7 Manual Procedures

### 18.7.1 Security

The system stores confidential and sensitive data about employees, administrators and the business. Therefore, below are the safety procedures that will be implemented to prevent Web application attacks. This will ensure that none of the data is threatened at any point.

- All employees require a password to login into the system. They are assigned usernames and generated passwords at first. Once the administrator creates an account, an email is sent to the employee with the generated password. In case the employees email was missing, the person creating the account will be notified with the generated password. Users are then prompted to change their passwords as soon as they login. Passwords must be 6 -8 alphanumeric characters and are going to be encrypted in the database.
- Application auto-logout if the system was left idle for 15 minutes. Reliable mechanisms will logout the users by popping up Time Out or Session Expired messages. This will prevent loss of data in case of a power outage, and will prevent unauthorized individuals from accessing the system through an idle computer.
- Setting specific permission sets for different workgroups. This will allow the administrator to configure the credentials for all employees and workgroups. Thus allowing flexibility as it provides the option of granting temporary credentials for particular employees at given times.

### 18.7.2 Operation

The system mainly operates over an online application that allows Direct Manipulation through multiple interface screens. Those screens are to be accessed using different devices like mobiles and pcs taking in consideration the systems requirements. The entire system is based on widgets and allowing the flexibility of adding plug-ins for future scalability.

### 18.7.3 Backup and Restore

Application data (from the database) must be backed up on regular basis. Our system will provide automatic ongoing data backup for time consuming input procedures i.e. schedules. Due to the systems nature, providing ongoing data backup for all procedures will cost a lot more than the cost of an error for example, recreating a single account. Instead full and incremental backups are to be scheduled automatically by the systems administrator. Users with the right permission set can perform manual full data backup. They can create restore points using the system's wizard.

The wizard will walk them through the steps required to back up all their data. All backed up files could then be removed for off site storage using external storage devices. This will protect users against hardware failures or any other unforeseen problems. Ultimately users could simply refer to a restore point from an existing backup file. They are then walked through in a wizard to complete the procedure.

#### **18.7.4 Data Archival**

Most of the old data is to be archived annually. This will include schedules, shifts, and employment records. Other records like Notifications and e-mail messages are not to be archived due to their irrelevance to the system and thus they are to be deleted every 3 months. Similarly to backup and restore, an administrator or an employee with the appropriate set of permissions must manually schedule to archive the data using the systems wizard.

### **18.8 Developer Contact**

This feature of the help section can only be accessed by higher authorities and employees with adequate permissions by default, but can be changed to accommodate other employees if desired. This feature can be used to contact the developers via bug tracking, issue tracking and feature requests.

# **Part VI**

# **Appendix**

# Appendix A

## Interview Questions

### 1. What are your current scheduling procedures?

We currently run a paper based scheduling system. We have some checkout of employee data tied into blackberrys, but that is mostly on the supervisors end. The amount of time that a schedule is generated differs per group. Some groups may generate schedules 2 weeks in advance and some may be a month. Our current scheduling procedures, although are done on paper, differ from department to department.

### 2. What are some of the down sides to you current system?

Since the system is completely based on paper, it can be time consuming for supervisors or department heads to make up schedules. It can be especially time consuming for certain departments that have employees working in multiple departments. This results in the supervisors and department heads to wait until the other department has scheduled their hours and schedule around it. Also, a paper based system takes a long time to complete, and because there is no specific template for each department, the supervisors are in charge of creating one.

### 3. What are some of the benefits to your current system?(What do you want to keep)?

Once supervisors and department heads have completed the template for creating schedules, it isn't very difficult to create schedules, just time consuming. I would like the system to still be easy to use.

### 4. What would you like the system to accomplish?

The system needs to be able to take time off the current scheduling system. It needs to be more efficient and easier to use. The system needs to be able to communicate between departments in the event that an employee works in multiple departments.

**5. Are there any specific features that you would like the system to have?**

Nothing specific at my request. But I havent done actual scheduling in a while. The better person to talk to about this would be Michael the chef. He schedules the largest group of employees covering all the restaurants and bars.

**6. What are some of the requirements that you would like the system to have? Ex. Communicate with specific applications, run on certain machines?**

I would like it very much if the system could do a budget summary of pay periods for the maximum number of hours that a group of employees worked. Also, I would like it if it had the ability to produce reports on whatever data that the supervisor would like in a variety of formats.

**7. What does your current schedule look like?**

We currently use a paper based system at the casino. I can have my assistant provide you with a copy of the current housekeeping schedule as an example.

**8. What programs do you require compatibility with? Ex. Outlook, excel**

I cant think of any programs off the top of my head that we would require compatibility with. Again, Michael the chef would be able to answer this question a little better.

**9. In addition, what would you like to see compatibility with (non-essential but nice)?**

I would like to see a portion of the system to be available on blackberry and potentially the iphone as well. So that employees had the ability to access their schedules on some type of mobile device. This would increase the usefulness and ease of access of the system.

**10. What does security look like currently on your scheduling system? Are there anything you want for security? Ex. Encryption, secure connection, use of VPNs**

Currently, as our system is completely paper based at the moment, we dont have much in the form of security. Obviously since we are dealing with personal employee information, there has to be some type of security implemented so that the data cannot be accessed from outside the system.

**11. When using the new system, who should have access to personal information (employees, supervisors, higher) and to what extent?**

Obviously employees shouldnt be able to change certain things. The supervisors however maybe should have the ability to change personal information for the employees. Things like passwords and stuff should be ok for employees to change but birth date should not.

**12. How are vacations scheduled (Christmas day, new years) and are there requirements when working a holiday? How about forcing employees to work days based on some other requirements?**

Again, I havent dealt with employee scheduling in a while. This would be a better question for Michael the chef or Dorine from payroll. Dorine would be better to talk to about the vacations.

**13. Do hours change during holidays based on the regular schedule?**

Dorine from payroll would be able to tell you about the specifics on holidays, or vacation information for employees.

**14. What is the management structure at the Deerfoot?**

Each department has their own supervisors/department heads. They are the ones that schedule the employees directly under them for their department. Also, higher executives should be able to pull certain information from the system such as budget information from different departments.

**15. How far in advance are new employee schedules created?**

This defers from department to department. It can range anywhere from 2 weeks to one month. This would have to be able to be set by the supervisors in order to accommodate all the different departments.

## **Appendix B**

### **API Documentation**

## B.1 API Document Template

In addition to the following template for API method calls, the external API document will have sections on our licence agreement, API usage or "Legend" for the document, Widget usage, and the collection of methods that can be used. (This is only a prototype visual representation and is subject to change)

### Example

Description

Usage

This is the colour and font of code

Parameters

Examples

Notes

Pre-Conditions

Post-Conditions

Errors

This is the colour of an error block

Warnings

This is the colour of warnings to the user

Date [In Bottom Right]

[Date]

## B.2 API Web Template

```
<h2>Title Of API Method</h2>
<h3>Description</h3>
<p>The desiption of the API method goes here</p>

<h3>Usage</h3>
<p>Usage Description if required</p>
<code>Code Goes Here for an example</code>

<h3>Parameters</h3>
<p>Parameter Descriptions if required</p>
<ul>
    <li>Parameter 1 - Parameter description</li>
    <li>Parameter 2 - Parameter description</li>
    <li>Parameter 3 - Parameter description</li>
</ul>

<h3>Examples</h3>
<p>Example description if required</p>
<code>Code Goes Here for an example on how to use it in a real life
scenario</code>

<h3>Notes</h3>
<p>Notes go here</p>

<h3>Pre-Condition</h3>
<p>Pre conditions go here</p>

<h3>Post-Conditions</h3>
<p>Post conditions go here</p>

<h3>Errors</h3>
<p>Errors go here</p>

<h3>Related Methods</h3>
<p><a href="#">Related Method 1 Link</a></p>
```

# **Appendix C**

## **Prototypes - Second Edition**

The following prototypes have been build upon the first edition prototype and include a more recent snapshot of how the final product will look and act like.

## C.1 Adding a User Screen

### Appendix C

Welcome: Admin | Settings | Logout

### Deerfoot Inn and Casino

Add | Update | Delete

User Icon  Upload

Instructions:  
Please fill in all the fields available. If the data is not available at this time then a default value will be given to any fields left blank.

**Personal:**

First Name:   
Address:   
City:  Calgary  
Home Phone:   
Postal Code:

Country:  Canada  
Bus Phone:   
Personal Email:   
Work Email:

**Work:**

Employee ID:   
Workgroup:   
Supervisor:   
Job Type:   
 Save  Cancel

**Mail(1)**

Inbox(1)  
Compose  
Drafts  
Notifications

**Help**

Search Help  
Contact Admin  
Online Help

## C.2 Dashboard View Screen

Welcome Admin | Settings | Logout

Deerfoot Inn and Casino

Event Notification - Joe Blow has requested a shift change [view it now...](#) [Close](#)

**Dashboard**

Statistics

**Schedule**

View Schedule

Request Change

Availability

**Users**

Add

Delete

Update

**Mail(1)**

Inbox(1)

Compose

Drafts

Notifications

**Help**

Search Help

Contact Admin

Online Help

**Quick Links**

[Add User](#) [Delete User](#) [Schedule](#) [Workgroup](#) [Permissions](#) [Some Link](#) [Settings](#)

**Schedule**

8:00 8:30 9:00 10:00 10:30 11:00 11:30 12:00 12:30 1:00 1:30 2:00 2:30 5:00 4:00 5:30 6:00 6:30 7:00

Person	8:00	8:30	9:00	10:00	10:30	11:00	11:30	12:00	12:30	1:00	1:30	2:00	2:30	5:00	4:00	5:30	6:00	6:30	7:00
Person 1																			
Person 2																			
Person 3																			
Person 4																			
Person 5																			
Person 6																			
Person 7																			
Person 8																			

**Notifications**

Joe Blow has requested a shift change [View](#) [Delete](#)

Your Supervisor has sent your team a notification [View](#) [Delete](#)

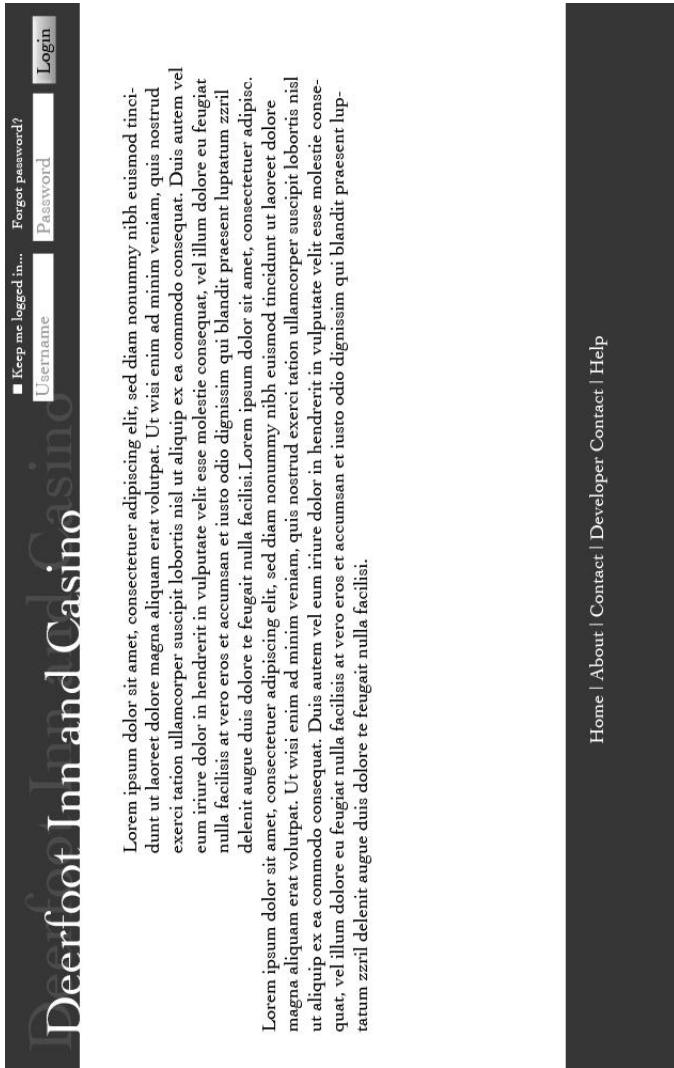
**Mail - Inbox**

Options

From	Subject	Date
Joe Blow	Can you switch with...	Jan 10
Sam B	Congratulations!	Jan 9
John Doe	Tomorrow	Jan 7
Martin T	Shift Change?	Jan 5
System	Happy New Year!	Jan 1
System	Merry Christmas!	Dec 25
System	Welcome!	Dec 4

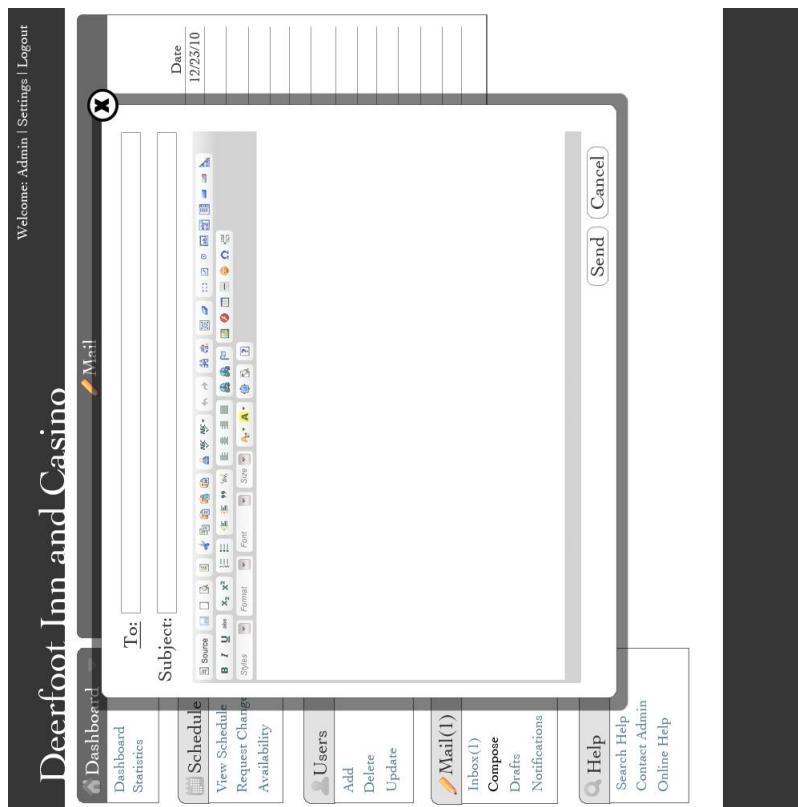
### C.3 Login Screen

Appendix C



## C.4 Internal Mail Screen

### Appendix C



## C.5 Update User Screen

### Appendix C

Welcome: Admin | Settings | Logout

### Deerfoot Inn and Casino

Updating Joseph Smith's Information

User Icon

**Personal:**

First Name: <input type="text" value="Joseph"/>	Last Name: <input type="text" value="Smith"/>
Address: <input type="text" value="122 Marda Loop"/>	Postal Code: <input type="text" value="T4W 5H4"/>
City: <input type="text" value="Calgary"/>	Country: <input type="text" value="Canada"/>
Home Phone: <input type="text" value="(403)555-5555"/>	Bus Phone: <input type="text" value="(403)555-5555"/>
Personal Email: <input type="text" value="joe.smith@gmail.com"/>	Work Email: <input type="text" value="joe.smith@deerfootinn.co"/>

**Work:**

Employee ID: <input type="text" value="450607"/>	Supervisor: <input type="text" value="John Williams"/>
Workgroup: <input type="text" value="Restaurant Staff"/>	Job Type: <input type="text" value="Bartender"/>

**Mail(1)**

Inbox(1)
Compose
Drafts
Notifications

**Help**

[Search Help](#) [Contact Admin](#) [Online Help](#)

## C.6 Redirect Login Screen

The screenshot shows the Deerfoot Inn and Casino login interface. At the top, the logo "Deerfoot Inn and Casino" is visible. Below it, the title "Web Agenda Login" is centered. There are two input fields: "Username:" and "Password:", both currently empty. To the right of the password field is a "Remember me?" checkbox and a "Login" button. Below the input fields is a link "Forgot your Password?". At the bottom of the form, there is a horizontal line.

## Appendix C

## C.7 Redirect Login Error

The screenshot shows the Deerfoot Inn and Casino login interface. At the top, the logo "Deerfoot Inn and Casino" is visible. Below it, the title "Web Agenda Login" is centered. There are two input fields: "Username:" and "Password:", both currently empty. To the right of the password field is a "Remember me?" checkbox and a "Login" button. Below the input fields is a link "Forgot your Password?". At the bottom of the form, there is a horizontal line. A red error message box is displayed, containing the text "There was an error in your username and password combination, please try again."

The screenshot shows the Deerfoot Inn and Casino login interface. At the top, the logo "Deerfoot Inn and Casino" is visible. Below it, the title "Web Agenda Login" is centered. There are two input fields: "Username:" and "Password:", both currently empty. To the right of the password field is a "Remember me?" checkbox and a "Login" button. Below the input fields is a link "Forgot your Password?". At the bottom of the form, there is a horizontal line. A red error message box is displayed, containing the text "There was an error in your username and password combination, please try again."

The screenshot shows the Deerfoot Inn and Casino login interface. At the top, the logo "Deerfoot Inn and Casino" is visible. Below it, the title "Web Agenda Login" is centered. There are two input fields: "Username:" and "Password:", both currently empty. To the right of the password field is a "Remember me?" checkbox and a "Login" button. Below the input fields is a link "Forgot your Password?". At the bottom of the form, there is a horizontal line. A red error message box is displayed, containing the text "There was an error in your username and password combination, please try again."

# **Appendix D**

## **Prototypes - First Edition**

The following prototypes are the first conceptual drawings of the system and there is no effort put towards consistency at this point.

### **D.1 Creating a Schedule**

## Appendix D

The image displays two screenshots of a web-based scheduling application for the Deerfoot Inn And Casino.

**Screenshot 1 (Top):** This screenshot shows a modal dialog titled "Schedule" over a main dashboard. The modal has tabs for "Jan 6th, 2009" and "Jan 7th, 2009". The "Jan 7th, 2009" tab is active, showing a timeline from 8:00am to 11:30pm for 12 employees. A tooltip for "Employee 1" indicates a shift from 8:30am to 12:00pm. Buttons for "Save", "Clear", and "Submit" are at the bottom. The main dashboard includes tabs for Dashboard, Users, Inbox(1), Settings, and Schedule. A top navigation bar offers links for Welcome, Username, Log Out, settings, and Log Out.

**Screenshot 2 (Bottom):** This screenshot shows a similar "Schedule" modal for "Jan 7th, 2009". It features a "Create Shift" button highlighted in red. The timeline and employee shifts are identical to the first screenshot. Buttons for "Save", "Clear", and "Submit" are at the bottom. The main dashboard tabs and top navigation bar are also present.

## D.2 Creating a Shift for an Employee

The screenshot shows the software interface for 'Deerfoot Inn And Casino'. The top navigation bar includes links for 'Welcome', 'username | settings | Log Out'. Below the navigation is a header with the logo and the text 'Deerfoot Inn And Casino'. The main menu has tabs for 'Dashboard', 'Users' (which is selected), 'Inbox(1)', and 'Settings'. A sub-menu for 'Schedule' is open, showing a grid of employee names (Employee 1 to Employee 12) against dates (Jan 6th, 2009, Jan 7th, 2009, Jan 8th, 09). The grid displays various shift durations for each employee. In the bottom right corner of the schedule grid, there are 'Save', 'Clear', and 'Submit' buttons. A large red arrow points from the 'Create Shift' button in the bottom left of the schedule grid towards the 'Edit' button in the top right corner of the same grid.

Jan 6th, 2009      Jan 7th, 2009      Jan 8th, 09

Employee	8:00am	8:30am	9:00am	9:30am	10:00am	10:30am	11:00am	11:30am	12:00pm	12:30am
Employee 1										
Employee 2										
Employee 3										
Employee 4										
Employee 5										
Employee 6										
Employee 7										
Employee 8										
Employee 9										
Employee 10										
Employee 11										
Employee 12										

**Create Shift**

**Save** **Clear** **Submit**

Jan 6th, 2009      Jan 7th, 2009      Jan 8th, 09

Employee	8:00am	8:30am	9:00am	9:30am	10:00am	10:30am	11:00am	11:30am	12:00pm	12:30am
Employee 1										
Employee 2										
Employee 3										
Employee 4										
Employee 5										
Employee 6										
Employee 7										
Employee 8										
Employee 9										
Employee 10										
Employee 11										
Employee 12										

**Create Shift**

**Save** **Clear** **Submit**

Home Administration Settings Contact  
(c) Web Agenda 2009

## Appendix D

The screenshot shows the Deerfoot Inn And Casino software interface. At the top, there is a navigation bar with links for Dashboard, Users, Inbox(1), Settings, and a sign-in/out section. The main area is titled "Schedule" and displays a timeline from Jan 6th, 2009, to Jan 8th, 2009. The timeline shows various employees assigned to shifts at different times. A modal window titled "Create a Shift" is open, prompting the user to enter details: Employee (Employee 6 is selected), Start Time (11:30am), End Time (12:00pm), Position, and Location. The "Save" button is highlighted in red, while the "Cancel" button is greyed out.

## D.3 Maintaining an Employee

The screenshot shows a two-step process for maintaining an employee record:

**Step 1: Employee Lookup**

The user is on the "Employee Lookup" page. The URL is [http://www.agenda2009.com/DeerfootInnAndCasino/EmployeeLookUp.aspx](#). The page title is "Deerfoot Inn And Casino". The navigation bar includes links for Dashboard, Users, Inbox(1), Settings, and a menu with Welcome, username, settings, and Log Out.

The main content area displays a list of employees:

Employee Result
Employee Result 1
Employee Result 2
Employee Result 3
Employee Result 4

Below the list are several navigation links: Manage Schedules, Current Schedule, Employee Lookup, Notification Panel, Request Privileges, View Hierarchy, and User Status.

**Step 2: Employee Maintenance**

The user has selected the first employee from the list, which has been highlighted with a blue border. The URL is now [http://www.agenda2009.com/DeerfootInnAndCasino/EmployeeMaintenance.aspx?EmployeeID=1](#).

The page title is "Deerfoot Inn And Casino". The navigation bar includes links for Home, Administration, Settings, and Contact.

The main content area shows a form for maintaining the employee record:

Employee ID:

Buttons: SUBMIT and CANCEL

Navigation links at the bottom include: Manage Schedules, Current Schedule, Employee Lookup, Notification Panel, Request Privileges, View Hierarchy, and User Status.

## Appendix D

The screenshot displays a web-based application interface for 'Deerfoot Inn And Casino'.

**Header:** Welcome, username | Settings | Log Out

**Main Navigation:** Dashboard | Users | Inbox(1) | Settings

**Login Form:**

Username	Password: .....*
User ID	
First Name	
Last Name	

**Dashboard Menu:**

- Manage Schedules
- Current Schedule
- Employee Lockup
- Notification Panel
- Request Privileges
- View Hierarchy
- User Status

**Permissions Configuration:**

Job Rellevance

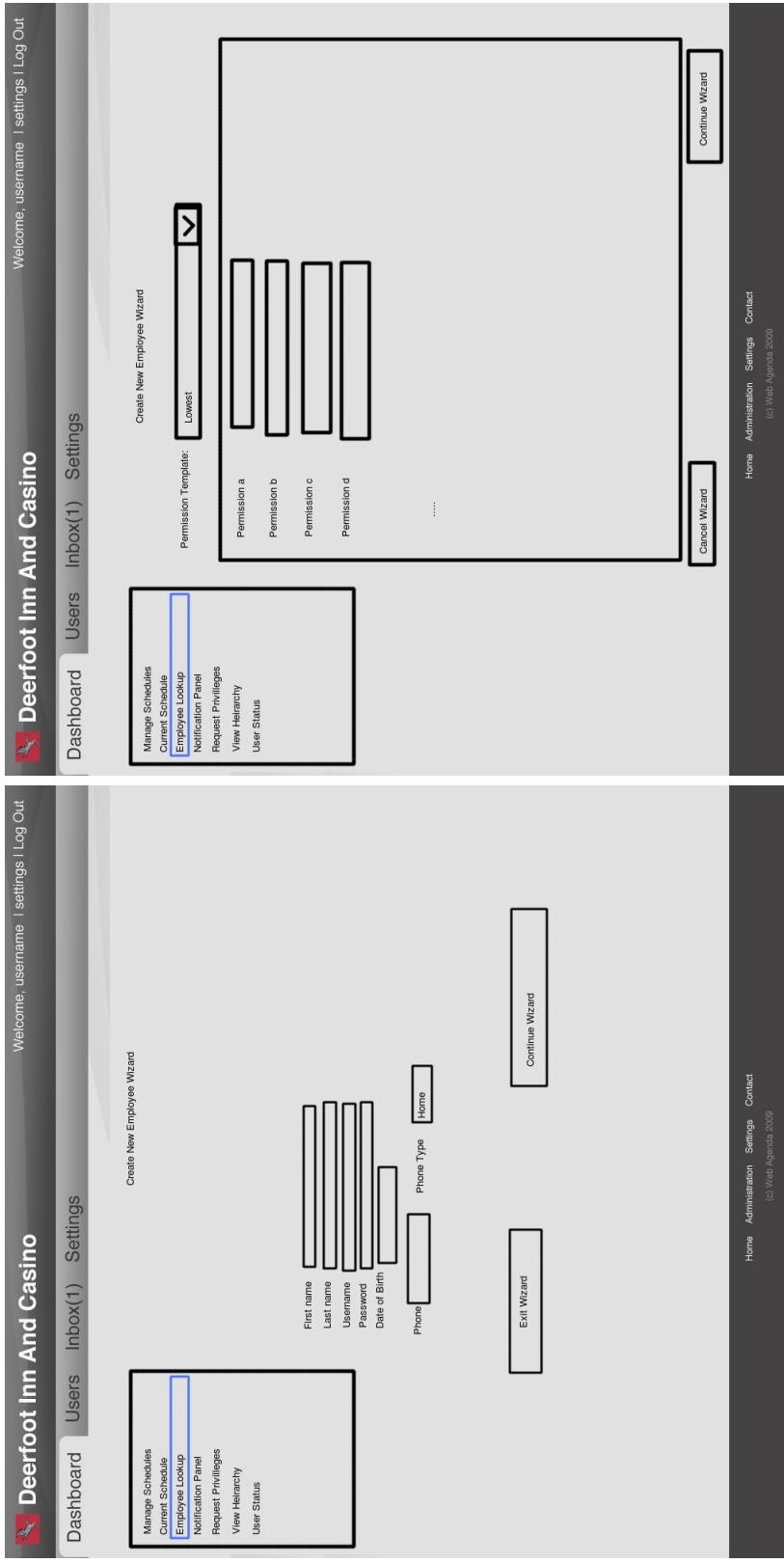
Positions Known:	Perm. Example 1: [Color Gradient]
Position	Perm. Example 2: [Color Gradient]
Position Permissions	Position Supervisor [Color Gradient]

Employee Information

Note x:

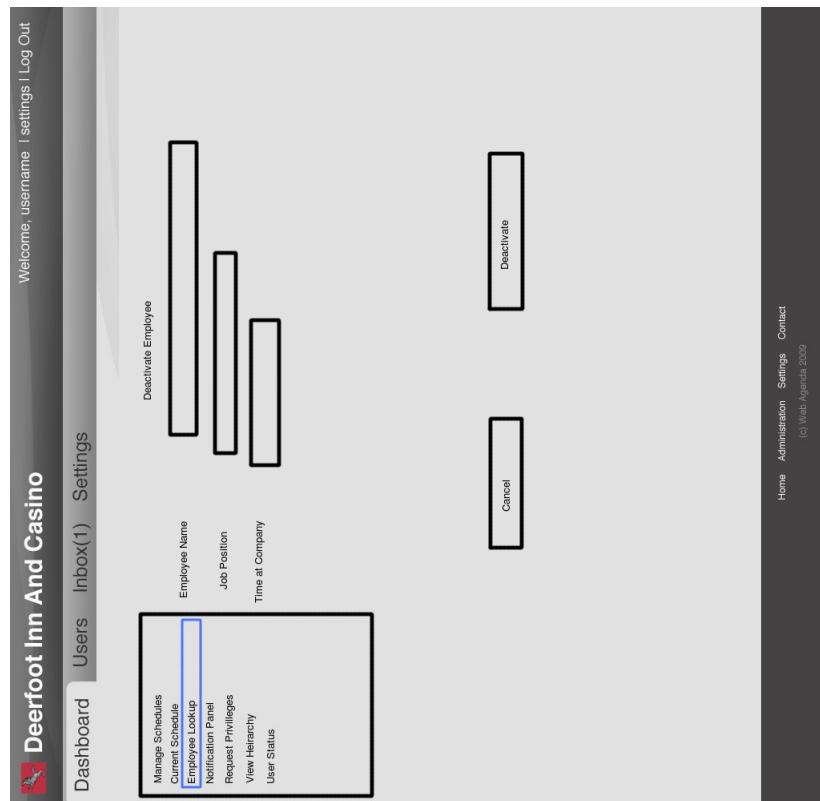
**Footer:** Home | Administration | Settings | Contact  
(C) Web Agenda! 2009

## D.4 Creating a New Employee



## Appendix D

## Appendix D



## D.5 Taking Emergency Leave

## Appendix D

Welcome, username | settings | Log Out

# Deerfoot Inn And Casino

Dashboard Users Inbox(1) Settings

Schedules Days Off Emergency Availability Form Supervisor Promotions

Your supervisor will be notified of your absence.  
Confirm your emergency absence

Type: Date: xxx

Give a reason (Optional)

Confirm

Home Administration Settings Contact

(c) Web Agenda 2009

## D.6 Viewing the Schedule

### Appendix D

The screenshot shows a web-based scheduling application for the Deerfoot Inn And Casino. The top navigation bar includes links for Dashboard, Users, Inbox(1), Settings, Welcome, username, settings, and Log Out. The main content area is titled "Schedule - Housekeeping". A header at the top of the grid indicates the period from "16th - 21st" on the left to "29th - 5th" on the right. The grid itself is titled "November 22nd - 28th". It contains 8 rows, each representing an employee (Employee 1 through Employee 8) and their daily work shifts. The columns represent the days of the week from Monday to Saturday. The shift times for most employees are 5pm - 9pm, except for Employee 1 who works 8am - 2pm. The "Off" status is indicated by a red background color.

Employee	Sun. 22	Mon. 23	Tue. 24	Wed. 25	Thu. 26	Fri. 27	Sat. 28
Employee 1	Off	5pm - 9pm	5pm - 9pm	Off	8am - 2pm	8am - 2pm	Off
Employee 2	Off	8am - 2pm	8am - 2pm	Off	5pm - 9pm	5pm - 9pm	Off
Employee 3	Off	5pm - 9pm	5pm - 9pm	Off	8am - 2pm	8am - 2pm	Off
Employee 4	Off	8am - 2pm	8am - 2pm	Off	5pm - 9pm	5pm - 9pm	Off
Employee 5	Off	5pm - 9pm	5pm - 9pm	Off	8am - 2pm	8am - 2pm	Off
Employee 6	Off	8am - 2pm	8am - 2pm	Off	5pm - 9pm	5pm - 9pm	Off
Employee 7	Off	5pm - 9pm	5pm - 9pm	Off	8am - 2pm	8am - 2pm	Off
Employee 8	Off	8am - 2pm	8am - 2pm	Off	5pm - 9pm	5pm - 9pm	Off

Workgroup: Housekeeping ▾

Print...

Home Administration Settings Contact  
(c) Web Agenda 2009

# Appendix E

## Permission Reference Documentation

### E.1 Permissions Overview

#### What are permissions?

Permissions are variables that determine what is accessible by a user based on their governing permission level.

#### What is a permission level?

A level of permissions are a general pre-defined list of permissions, usually based around the job and functionality that a user needs from the system. The lowest permission level is 0, and each level above that must contain permissions that are equal or higher in value. There are possible multiple variations of one permission level. The level defines authority, not the job's requirements.

## How do these permission levels work?

By default, a user is assigned a permission level of 0, the lowest possible level, when created. This should allow for general viewing of the current schedule, sending messages to other employees via the internal messaging system, being able to request booking days off, and being able to export their schedule into another format. Every consecutive permission level must have equal, or greater permission values set.

Every level that is represented by a number is a 'template' permission level, meaning that a user can create alternate permission levels that are recognized as a certain level but have varying degrees of permissions. These alternate permission levels are seen as the template number followed by a letter.

Example: 0 is a template for the minimum employee, 0a and 0b are alternate level 0 versions.

The reason for having template and alternate levels is so that employees can be grouped by their permission levels while retaining any varying permissions that they may require for their job AND so that supervisors can track and understand employees' needs better. If a new employee is hired from another division of the company as a supervisor, they might be given a new user account with permission level 0, but also have the permissions required to create a schedule (but not activate it, of course. They would have to notify a superior to have it put in an active state) which could then be called permission level 0x. As well, if an employee is found to be spamming the internal messaging system, they can have that privilege revoked under level 0z, where they are denied access to any component of the messaging system.

Levels are also useful for managers that need to temporary grant access to other employees. For instance, a manager is going on vacation. They can pass on their permission levels to a supervisor that they trust for a period of time. In this case, permission level 4 is granted to employee with permission level 3b for a period of 2 weeks. (Due to the nature of permissions, no user can elevate their own permission levels. Only one account has full permissions for a level, and that level is known as the Highest permission level +1.)

## How is this a secure method of permission setting?

Good question. With levels, no one user can go above what they are given. No user can create an employee above their own level. When WebAgenda is first put into action, only one account exists: the administrator, or root account. It contains what is known as Highest permission level +1, where the permission level will always be the highest one. The actual level number is dynamic. If a company has 5 permission levels created (0-4), then the root account will be 5. If the company drops two levels, then root account will be 3, and so on and so forth. The root account is the only account that can create or delete permission levels. However, there is one exception if all goes according to plan. The root account can enable a maintenance account, one that is disabled by default. Its purpose is for full access to the entire system. **Database** backend, all employee information, all permission levels are ignored for this user. It should only be enabled when there is actual maintenance to be done, then disabled after. It requires a long password that is changed with each login to the system under it.

One more note. With permission level templates, anyone who understands the permissions for that level can make an educated guess on how everyone on that level functions; all 0 level employees are basic bread-and-butter workers for the company. All level 1's would be supervisors or senior employees. In any case, once the permission levels are set up, it makes monitoring employees much easier for the scheduling portion of a business. Once plug-ins get created, more and more functionality can be expected and perhaps WebAgenda will evolve into something more than a scheduling system.

## Any additional information?

Permission level can have an alias that helps describe it; for instance, level 0b can be known as Supervisor's assistant, which has the ability to edit schedules that the supervisor works on. Level 3 can be known as Manager, even though it's a template.

A permission level can have a duration; this usually refers to an employee's working period a date that is used to determine when an employee's contract or work stops. For most employees, this will not have a value (no known date for termination) while for contract workers, this might end on dd/mm/yyyy. Having a permission level end on a date for a contract worker means the supervisor doesn't have to go in and disable them, it will be automatic. That also means that once an employee is fired, their account is deactivated, but until then won't be.

## E.2 Permission Set

### Employee Permission Set Revision 0.1 (First edition)

A series of permissions that affect employee actions on the system

Permission Name	Values	Description
CanReadActiveSched	Boolean	Employee can access times on a schedule under their name and lower plevels for the current date or period
CanReadPreviousSched	Boolean	Employee has access to times in the archive of schedules under their name and lower plevels
CanReadFutureSched	Boolean	Employee has access to times in schedules ahead of the current date under their name and lower plevels. Most likely they will be blank.
CanEditActiveSched	Boolean	! Employee can make changes to their time in the active schedule under their name and lower plevels
CanEditFutureSched	Boolean	! Employee can add or remove lower plevel employees and self to a schedule ahead of the active schedule.
CanViewResources	Boolean	! Resource view enabled: when viewing schedule, show employees <b>that are available</b> , are not available, have time <b>booked off</b> for that view, <b>booked off for emergency</b> , and are <b>already working</b> that view.
CanSearchResources	Boolean	! Employee can look up employees below their plevel and view information (provided permissions allow it)
CanChangePermissions	Boolean	! Although it only applies to lower plevel permissions, and cannot be raised past the default plevel set for Employee, this permission allows changing of other employee's permissions.
CanReadLogs	Boolean	!! This permission is held for the highest permission level and the SysMaintain account; system logfiles generated by the server can be read. Permission can be assigned to another plevel by highest plevel.
CanCreateReports	Boolean	! Can choose from a selection of reports to generate. When a report is generated, it must be sent to an employee.
CanReadReports	Boolean	If a report is sent to employee, it won't be discarded and logged if they have this permission.
CanExportReports	Boolean	! Administrators may want information to be manipulated by trusted employees. This option may provide export methods when viewing report.
CanExportSchedule	Boolean	Why you wouldn't want this is beyond me. But there might be use for it.
CanRequestDaysOff	Boolean	Disable this for abuse or contractors
MaxDaysOff	Number	How many days off an employee can take during a

## Appendix E

---

		schedule period. Decimal values should be allowed, or base this off a month/year value.
CanTakeVacations	Boolean	Days off generally refers to days employee is available that are turned to not available. Vacation is a period of time and generally reserved for full-time employees.
MaxVacationDays	Number	Number of total days in a time period that employee can take.
CanTakeEmergencyDays Off	Boolean	If this is abused, it can be disabled
CanViewInactiveEmployees	Boolean	! View information on inactive employees
CanSendNotifications	Boolean	! Unlike messages, internal e-mails, this actually displays a notification on an employee's dashboard.
Trusted	Boolean	Similar to a permission promotion (which cannot be given by self), trusted is a permission that allows any employee to perform actions that affect the next highest level without requiring authorization from a superior. However, a notification will still be sent notifying the supervisor (n/a if highest plevel) explaining actions. This only applies to permissions that are currently enabled. Ex: If an employee cannot book days off, a trusted employee cannot book days off either.
PreferredRank	Number	Starting with 0 as the highest priority employee and then incrementing, this differentiates seniority within the same plevel. For instance, if there are 3 supervisors (emps with same plevel) in the same workgroup, only one can override the decisions of the other two. While the Trusted permission trumps any rank, if more than one employee is trusted or none are, preferred rank is used to set decisions. If two supervisors are promoted at the same time and no rank 0 supervisor is found, promoter will be prompted to set the preferred supervisor. If only one supervisor is set in a workgroup, rank is automatically 0. If more are added, system will assign ranks by the specified System permission rule. If two employees with the same preferred rank exist, the System's SeniorityRule permission takes precedence.

Descriptions prefixed with a '!' mean that permissions are not recommended for permission levels assigned to anyone lower than a supervisor or manager position. The business should determine who retains such control permissions.

'plevels' is the short form for permission levels, a set of the above permissions that define an employee's authority in the system.

**Schedule Permission Set**

A series of permissions that affect system interaction with schedules. The employee's permission level that created the schedule becomes the schedule's permission level.

Permission Name	Values	Description
LowestReadPermission	Number	The lowest permission level that can read the schedule when it's active Default = 0. Future schedules are determined by employee permission levels and permissions.
DefaultTimeRef	Char	When referring to time values, when 'Y' use a per year reference, 'M' use per month, 'W' use week, 'D' use day. Can be changed, can be converted to and from this value. Exists for ease of use. Defaults to Week.
Active	Date	The date that schedule is active from.
ScheduleDuration	Number	The number of days that a schedule can stay active for. DefaultTimeRef will be converted into days if a different value is given. (Ex: 2W = 14, month would be determined by # of days in it)
EmployeeCreator	Number	Employee ID that created schedule, value will automatically be assigned when created.

If no schedules are active, no one is expected to come into work. A warning system should be implemented so that if there is a period that no schedules exist, affected employees that can create schedules should be warned.

Anyone with higher permission levels than the EmployeeCreator value can view and modify the schedule regardless, provided they have CanEdit[x]Sched permission. (Where x = Future or Active)

**System Permission Set**

<b>Permission Name</b>	<b>Value</b>	<b>Description</b>
DenyShiftRequest	Date	The time duration before a shift starts that any requests for that shift will be cancelled and user is notified that no takers for shift were found.
MinShiftDuration	String (hh:mm)	How long (in hours/minutes) that a shift will last. Shifts may actually be smaller, but payment for a shift will last for that duration. (Ex: work for an hour, but minimum shift is 3 so 3hours of work is paid out. Value=03:00)
MinAvailabilityShifts	Number	The number of minimum shifts that must be applied for an availability to be proper, as per week. Can be 0.
AvailabilityChangePause	Date	How long user must wait before changing availability again (ie. Cannot change every day, must wait n amount of weeks)
SeniorityRule	Number	0 – Prompt for preferred rank (display in a list from highest to lowest) 1 – Default: Oldest worker is given authority 2 – Greatest availability (works most hours) 3 – Most permissions enabled (if there is a tie, will set to 0) 4 – First come, first serve (FIFO, aka queue) 5 – Alphabetical 6 – Reverse Alphabetical
MaxEmergencyDuration	Number	Number of consecutive days can be taken off. Default is 2 weeks. If more time is required, employee must speak to administration or an authoritative figure in the company.

# Appendix F

## Glossary

### A

**Actor** Something or someone that interacts with the system, yet exist outside of it. Though actors are often people, they may also be hardware, or even other systems.

**Administration Requirements** Minimum expectations set by those in power that aid in management.

**Aloha** The Aloha Corporation provides payroll and payroll related services to small and midsize businesses. Aloha offers direct deposit, workers' compensation and other convenient, time-saving services.

**Alpha Release** A basic release of the software for testing purposes.

**Apache** An open source HTTP (and HTTPS) server for \*nix, Windows NT, and other computing platforms.

**API** The Application Programming Interface is a set of methods for interacting with data of all types in the database. This item is mostly used for getting data from the database and storing data in the database.

**Application Layer** All classes specifically related to application specific processes.

### B

**Backup** A copy made of existing information and data that is stored in a separate location. Should a failure result in the loss of the original source, the information and data can be restored from the backup to reduce the impact of the failure, and the time it takes to recover from it.

**Beta Release** A feature rich release of the software. This release may still have errors that need to be fixed.

**Bug** An error or unexpected / undesired behaviour in a program. Bugs can have large, small, or negligible impacts.

**Business Layer** All classes that are specifically related to business processes.

## D

**Database** A Method of storing data on a computer/server that allows access of that data in an extremely efficient manner.

**Department** A specific unit or section of a company that performs or monitors a specific company-related task, such as food-service, cleaning, IT, etc.

**Deployment** The installation of a program or application to be used by the end user.

**Deerfoot Inn and Casino** The business client for which WebAgenda is currently being based off of.

**Document Viewers** An application that is installed on a client computer that displays a file containing relevant data.

## E

**Encrypted** Data that has been modified and cannot be used until decrypted, usually by specifying a password.

**Encryption** A series of modifications to the contents of data so that it cannot be read without undoing the modifications first.

**Export** The ability of a system to send information out to be used by other applications

## F

**Format** A common language that multiple applications can understand.

**Functional Requirements** Features and components of the system that must be provided if the system is to fully meet its intended usage.

**Fact-Finding Methodology** A detailed account of the methods used to gain information about the needs of the client business, the work environment the system will be deployed in and, if applicable, any current systems that will be replaced by the new one.

**Final Release** A full featured release with little or no errors ready to be installed.

## G

**GPL (General Public License)** A widely used free software license, originally written by Richard Stallman that provides free access to software published under its terms. Users are allowed to copy, modify, and redistribute GPL software as long as its conditions are met.

**GNU/Linux KCal** A calendar plugin used by KOrganizer, part of the personal information management suite found in the K Desktop Environment, one of the two most popular GNU/Linux system desktop managers.

## J

**J2EE** Also known as Java 2 Platform Enterprise Edition, Java EE is an enterprise mainframe programming language derived from java to provide simple application development for thin clients, or low-powered computers.

**Java** An object-oriented programming language written for cross-compatibility and server-side speed

**Java EE** The server side libraries from the Java language.

**JSP** Java Server Page is a server-side technology that integrate html code from webpages into the Java and Java EE programming languages.

## H

**HTTP (Hyper Text Transfer Protocol)** Used by web servers to transfer web pages and web applications so that users can use them.

**HTTPS (Hyper Text Transfer Protocol)** Secure is a protocol used to securely transfer data such as passwords to the server.

**Javascript** A simple, cross-platform, World-Wide Web scripting language created by Netscape that is used on browsers and servers to provide additional functionality.

## I

**Interface** The presentation of communications between application and user.

**Interface Layer** All classes related to the display of content and data.

## L

**LaTeX** A programming language and word processor that is used for typesetting technical data into professional documents.

## M

**Mac OS X iCal** Default calendar client included in Macintosh OS X installations and on products owned by Apple, such as the iPhone.

**Maintainability** The degree to which a product can be readily supported (upgraded, secured) over a period of time; usually the predicted lifecycle of the product.

**Microsoft Outlook** Default e-mail client included in Microsoft Windows installations that includes a calendar. Web versions of outlook exist with calendar functionality as well.

**Mobile Web** Refers to web browsing on a mobile device, where the screen size is much smaller than conventional computers that are used. Formats that a Mobile Web device can view must be clear, simple, reasonably small and concise. Can be in the form of a webpage, text file, or image.

**MySQL** The world's most popular open source database which can be distributed for no charge and is licensed under the GPL.

## N

**Non-Functional Requirements** Requirements placed on a system that focus on the quality of the actions it provides and any constraints they must function within.

## P

**Persistence Layer** All classes responsible for persisting data to a database.

**Project** A temporary process, that provides a product to a client.

**Project Scope** Defines the limits of what a project will and will not cover. Elements determined to be outside of the project scope will not be addressed by the project team, though they may need to be covered by an outside party if the elements are desired.

## R

**RSS** (Really simple Syndication) Family of web feed formats to publish frequent updates.

## S

**SSL** (Secure Sockets Layer) Provides a method to send data across a network securely.

**Server** A very powerful computer that hosts a variety of services to clients.

**Servlet** A server-side java program and / or applet

**SMS** Short Message Service, a cellphone-based messaging system that can send 160 character messages delivered by the digital cellular system.

**Source Code** Groupings of text that a computer can understand and create a product as result.

**Sub Domain** A domain that resides in a larger domain that is seen on a network. Is generally used to separate tasks that perform a function for clarity and purpose.

**System** A program or application used to help the user to accomplish a task.

**System Interface** How the end user interacts with the program or application to accomplish a task.

**System Interface Requirements** How easy the system is to use based on the end user.

**System Requirements** The minimum requirements that a program or application needs to run.

## T

**Testing** A process through which aspects of the system are examined to ensure they work as expected, and meet all business requirements. This process may be repeated many times, as resolving some issues may reveal others that were previously unknown.

**TSL** (Transport Security Layer) A secure communications protocol that allows data to be sent over a computer network (such as the internet) while preventing outside parties from viewing or tampering with it.

## U

**Use Case** A description of a system's behavior from the user's point of view, detailing the events that must occur for a user to accomplish a specific task.

## W

**WebAgenda** Name of the Scheduling System developed by Daniel Kettle, Daniel Wehr, and Mark Hazlett as a useful and free web-based alternative to the proprietary commercial systems with emphasis on 24/7 access to schedules.

# **Appendix G**

## **Index**

# Index

- Activity Diagrams, 92
- Actor, 39
- actor, 36–79, 179
- Administration Requirements, 179
- Aloha, 20, 30, 31
- Alpha Release, 181
- Apache, 189
- API, 39, 186, 194, 195, 204–206
- application, 115
- Backup, 22, 28, 31, 179
- Beta Release, 179, 181
- budget, 202
- bug, 181, 194, 199
- Business Overviews and Objectives, 18
- Contact Information, 184
- current scheduling procedures, 201
- Current System, 20
- current system, 201
- Data Dictionary, 143
- Database, 22, 28, 29, 31, 37, 39–41, 100, 101, 186–188, 226
- Date Formats, 185
- Deerfoot Inn and Casino, 1, 13, 18–20, 26, 27, 29–31
- Department, 13, 19, 20, 26, 27, 201, 203
- Deployment, 13, 181
- Design Documentation, 180
- Document Formatting, 15
- encrypted, 14
- encryption, 181, 202
- export, 24, 29, 30, 225
- Extended Use Case Diagrams, 36
- Fact-Finding Methodology, 179
- figure, 15
- Final Release, 181
- Footer, 15
- format, 20, 24, 29
- Functional Requirements, 21, 179
- General Public Licence, 13, 25, 185, 187
- GNU/Linux KCal, 29
- Header, 15
- holidays, 203
- HTTP, 187, 188, 194
- HTTPS, 187
- human resources, 20
- Information Systems, 22
- interface, 29, 30, 32, 44, 100, 180, 181, 185, 186
- interface layer, 186
- Internet, 14
- J2EE, 189
- Java EE, 189
- Javascript, 189
- javascript, 29
- JSP, 189
- LaTeX, 185
- Latex, 16
- LaTeX2E, 185
- licensing, 13
- Mac OS X iCal, 25, 29
- Maintainability, 179
- Maintainability and Administration Requirements, 31
- management structure, 203
- Members And Roles, 182
- Microsoft Outlook, 29, 202

## Appendix G

---

Mobile Web, 30  
MySQL, 187, 189  
Non-Functional Requirements, 26  
    passwords, 202  
    Permissions, 25  
    Preface, 13  
    Project, 13  
    project, 14, 15, 28, 30–32, 60, 179–183  
    Project Management, 177  
    project scope, 179  
    Project Standards and Procedures, 185  
    Reporting Relationships and Procedures, 183  
    requirements, 202  
    Requirements Analysis, 180  
    Requirements Documentation, 13, 179  
    Requirements Models, 34  
    RSS, 25  
    Schedule, 83, 118, 177  
    schedules, 19  
    scheduling procedures, 201  
    security, 14, 202  
    server, 14, 20, 28, 30, 185, 187–189  
    Servlet, 189  
    Shift Change, 88  
    SMS, 31  
    source code, 13, 14, 25, 183, 185  
    SSL, 181  
    State Machine Diagrams, 82  
    Statement of the Problem, 19  
    Sub Domain, 25  
    system, 13, 14, 19, 20, 22, 25–32, 35–79, 81, 83,  
        88, 90, 93, 95, 100, 179–181, 185–188,  
        190–197, 201–203, 214, 224–226  
    System Environment, 20  
    system interface, 179  
    System Interface Requirements, 29, 179  
    System Requirements, 21, 179, 188, 190  
    Team Configuration, 182  
    testing, 13, 14, 188, 231  
    Usability Requirements, 32  
    Use Case, 35, 36, 38, 39, 42–44, 61, 63, 65, 66,  
        68–70, 179  
    Use Case Diagram, 34  
    User Requirements, 18  
    vacations, 202  
    WebAgenda, 14, 25–27, 29–31, 35, 36, 38, 44,  
        93, 114, 179, 185, 194, 226  
    WebBrowser, 13