CEN4010 Section U01

Golden Pages

Team 4

Adam Merille

Shadeh Ferris-Francis

Jorge Travieso

Andy Martinez

Sergio Saucedo

Michael Machin

November 11, 2014

Professor Tariq King

## Executive Summary

This System Design Document is a description of the architectural and design patterns being used in Golden Pages, as well as the explanation of all the subsystems involved in the creation of Golden Pages. The document starts by introducing the purpose of the system, the functional and non-functional requirements, the design methodology, definitions, acronyms, abbreviations, and an overview of the document. This is followed by a discussion of the software architecture that is being proposed. This is done by first showing an overview of each subsystem, then a detailed description of the major subsystems and how each use cases are associated with the subsystems, then followed by a map of the subsystems to the hardware and software. Additionally, this document identifies the data that needs to be stored and any security issues related to it. Then, a description of how Golden Pages deals with security and privacy is given.

Then, the document provides a detailed discussion of the design by first giving an overview by means of a minimal class diagram, then, the sequence diagrams showing the interaction of the objects, and finally, a detailed class design, including the purpose of each class. The final portion of this document contains a glossary of the terms used, and various appendices pertaining to use case diagrams, use cases with nonfunctional requirements, detailed class diagrams, class interfaces for the implemented subsystems, and meeting diaries.

Table of Contents Page

1. Introduction…………………………………………………….……. 4
   1. Purpose of System…………………………………..….……. 4
   2. Functional and Non-functional Requirements………….…… 4
   3. Design Methodology………………………………………… 8
   4. Definitions, Acronyms, and Abbreviations……….…….….... 8
   5. Overview of Document….…………………………….…….. 9
2. Proposed Software Architecture..……………………………….…...10
   1. Overview……………………………………………………..10
   2. Subsystem Decomposition………………………………….. 11
   3. Hardware and Software Mapping…………………………... 12
   4. Persistent Data Management………………………………...13
   5. Security/Privacy……………………………………………..14
3. Detailed Design………………………………………..………….... 15
   1. Overview…………………………………………………….15
   2. Object Interaction………………………………………..…. 16
   3. Detailed Class Design……………………………………..... 20
4. Glossary……………………………………….……………………..25
5. Appendix…..………………………………………………………... 26
   1. Appendix A - Use Case Diagram ……………………........... 26
   2. Appendix B - Uses Cases ……………………………………26
   3. Appendix C - Detailed Class Diagrams ……………………...45
   4. Appendix D - Class Interfaces ……………………………….47
   5. Appendix E - Diary of Meetings and Tasks………………….54

1. Introduction

In this chapter, we are presenting overall design methodology along with the purpose and overview of our system, The Golden Pages. After the system purpose is summarized, a brief description of the functional and nonfunctional requirement of the use cases is first introduced. In addition, we are including the definitions of some of the most frequently used acronyms, terms, and abbreviations.

* 1. Purpose of system

The purpose of The Golden Pages is to allow Businesses to create pages wherein their potential customers can view, follow, and set up appointments. Our goal is to facilitate effortless communication between businesses and consumers, where appointments can be made and confirmed, and reviews can be written and viewed. The central feature of this system is to automate searching for and creating appointments.

This system is meant to remove some of the confusing and time consuming nature of making appointments by presenting to the customer a full calendar view of available times and dates of businesses. The appointments the customer requests will then be automatically stored for the business to keep track of.

* 1. Functional and Non-functional Requirements.
* Business Account Creation (See use case GP01 in Appendix B)
  + Functionality:
    - New users should be able to create a business account by inserting the required information into the system.
  + Constraints:

1. Usability:
   1. No training time as it follows web standard account creation
   2. On average the user should take 5 minutes to complete
2. Reliability:

Mean time to Failure - 2% failures for every 5,000 attempts

1. Performance:
   1. Request should be sent, created, and saved within 5 seconds.
   2. System should be able to handle 50 requests in 1 minute.
2. Supportability:

The request should be correctly handled by IE, Mozilla, Chrome, Safari and their mobile equivalents.

* Consumer Account Creation (See use case GP02 in Appendix B)
  + Functionality:
    - New users should be able to create a consumer account by inserting the required information into the system.
  + Constraints:

1. Usability:
   1. No training time as it follows web standard account creation.
   2. On average the user should take 5 minutes to complete.
2. Reliability:

Mean time to Failure - 2% failures for every 5000 attempts.

1. Performance:
   1. Request should be sent, created, and saved within 5 seconds.
   2. System should be able to handle 50 requests in 1 minute.
2. Supportability:

The request should be correctly handled by IE, Mozilla, Chrome, Safari and their mobile equivalents.

* Login for Account (See use case GP03 in Appendix B)
  + Functionality:
    - Users should be able to log into their accounts, which allows them to access and edit their profiles.
  + Constraints:

1. Usability
   1. No training time needed, follows web standard login
   2. User should be able to login in under a minute
2. Reliability

Mean time to Failure - 5% every 10,000 attempts\

1. Performance

The system should be able to sustain multiple users logging in at the same time.

1. Supportability

The request should be correctly handled by IE, Mozilla, Chrome, Safari, and their mobile equivalents.

* Password Recovery (See use case GP04 in Appendix B)
  + Functionality:
    - Current system users should be able to retrieve their password by using the “Forgot Password” functionality in the login page. A valid email or username is necessary for the recovery.
  + Constraints:

1. Usability:
   1. No previous Training Time
   2. Help facility should provide 1 help frame on the topic
   3. On average the user should take 5 mins to complete the send request form.
2. Reliability:
   1. Mean time to Failure – 5% failures for every twenty four hours of operation is acceptable.
   2. Availability – Down time for Login Back-up 30 minutes in a 24 hour period.
3. Performance:
   1. Request should be sent and saved within 5 secs.
   2. System should be able to handle 100 request in 1 minute
4. Supportability:

The request should be correctly handled by Internet Explorer, Firefox, Chrome, Safari and their mobile equivalents.

* (Security 1) Storage Manipulation Prevention (See use case GP11 in Appendix B)
  + Functionality:
    - The system shall be able to filter out user inputs that are possible SQL injections or Cross-site scripting (XSS).
  + Constraints:

1. Usability:

System should perform automatically.

1. Reliability:

Mean time to Failure - 2% for every 1,000 attacks.

1. Performance:

System should check without user realizing, should finish < 1 sec.

1. Supportability:

The request should be correctly handled by IE, Mozilla, Chrome, Safari and their mobile equivalents.

* (Security 2) Multiple Accounts for One Person/Business Creation (See use case GP12 in Appendix B)
  + Functionality:
    - The system shall check and prevent that the any given user cannot create/have more than one account under the same email or username.
  + Constraints:

1. Usability:

System should perform this security measure automatically.

1. Reliability:

Mean time to Failure – 10% failures for every 1,000 attempts

1. Performance:
   1. System should check for duplicated data within 3 seconds.
   2. System should be able to identify foul account creation and record the numbers of tries tried.
2. Supportability:

The request should be correctly handled by IE, Mozilla, Chrome, Safari and their mobile equivalents.

* Consumer Account Editing (See use case GP07 in Appendix B)
  + Functionality:
    - The system shall allow the users to modify their account information, such as password, email, name or description.
  + Constraints:

1. Usability:
   1. No training time as it follows web standard account editing
   2. On average should take less than 5 minutes to complete dependant on amount of information changed.
2. Reliability

5% failures for a 24 hour run time is acceptable

1. Performance
   1. Request for edit should be sent, updated, and saved within 5 seconds.
   2. System should be able to handle 50 requests per minute
2. Supportability

The request should be correctly handled by Internet Explorer, Firefox, Chrome, Safari and their mobile equivalents.

* Creating Appointment (See use case GP09 in Appendix B)
  + Functionality:
    - The system shall allow the users to add new appointment entries to the calendar.
  + Constraints:

1. Usability:
   1. On average the user should take less than 5 minutes to complete.
2. Reliability:
   1. Mean time to Failure – 1% failures for every 5,000 attempts
3. Performance:
   1. Appointment creation should take less than 4 seconds.
   2. System should be able to handle 100 requests in 1 minute.
4. Supportability:

The request should be correctly handled by IE, Mozilla, Chrome, Safari and their mobile equivalents.

* 1. Design Methodology

We are using the unified software development process (USDP) to work through the entire project. This iterative and incremental approach allows us to revisit various facets of the project as we move forward.

The Software Requirements Document has allowed our team to analyze the critical areas of the system in order to subdivide them into various subsystems and classes. The models we have used are based on the Unified Modeling Language and they include:

* Class diagrams (minimal and detailed)
* Sequence diagrams
* Use case diagrams
* Deployment diagrams
  1. Definitions, Acronyms, and Abbreviations

**Functional requirement**: defines a function of a [system](http://en.wikipedia.org/wiki/System) and its components

**Nonfunctional requirement**: is a [requirement](http://en.wikipedia.org/wiki/Requirement) that specifies criteria that can be used to judge the operation of a system, rather than specific behaviors

**System**: the software that is being developed; ours is called Golden Pages

**Subsystem**: an independent working part of the system

**Unauthorized**: to not be allowed to access certain content

**Fields**: in a webpage or other document, a place where text is input

M**ethodology**: in a software development context, it is a framework used to structure, plan, and control the process of developing software

**USDP**: Unified Software Development Process – an iterative, incremental, risk-focused, and use-case driven software development framework which can be refined to the needs of individual software development teams

**UML**: Unified Modeling Language – a set of rules used to model a system in a textual or graphical way to remove the ambiguities of natural language

**OCL**: Object Constraint Language – a set of rules used to describe the invariants, pre-, and postconditions of a system

**Invariant**: in the context of object oriented programming, it is a detail which is *always* true for instances of a given class

**Precondition**: in the context of software development, something which must be true before a particular method is called

**Postcondition**: in the context of software development, something which must be true after a particular method has returned

* 1. Overview of document

This design document is composed of 5 sections. The first section introduces the overall document flow. The second section focuses on of the software architecture that the Golden Pages web application will be utilizing. Section 2 is further broken into the decomposition of the subsystems, the use cases associated with those subsystems, the hardware and software being used, the persistent data, and security concerns.

Section 3 is composed of the system design. The first subsection gives a big picture view of the classes that are going to be implemented based on the subsystems. The second subsection is a series of sequence diagrams describing how the system flows over time for a given use case. The final subsection dives into the purpose of each class and uses the object constraint language to exemplify how the methods will be functioning.

Section 4 is the glossary of terms pertinent to the project and section 5 consists of the appendices. The appendices cover the use case diagram, the detailed class diagrams, the class interfaces, and the diary of meetings.

1. Proposed Software Architecture

In this chapter, we are presenting the overall system model and architecture for the Golden Pages. Our system utilizes a MVC architectural design. An overview of this design will be described in section 2.1, with the succeeding sections detailing our system itself.

* 1. Overview

The Golden Pages system will consist of 7 major subsystems, as shown in the diagram below. The MVC architectural pattern will be used to reduce coupling and dependencies between the system layers, and to have multiple views from the same model, as native mobile interfaces could be developed in the future. As a secondary pattern, the Pipes and Filters architectural style will used for the authorization and authentication of requests to the system.

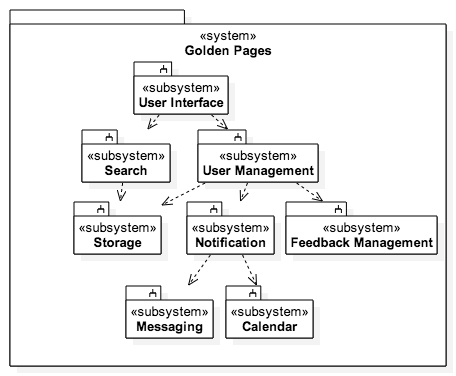


Figure 2.1

1. **User Interface**: space where the user interact with the system.
2. **User Management**: manages the user accounts creation, deletion, modification and the relations between users.
3. **Search**: finds and retrieves entities in the system.
4. **Notification**: retrieves notification alerts from messaging and calendar events.
5. **Feedback Management**: manages the user business reviews and recommendations.
6. **Messaging**: sends and retrieves messages between internal users.
7. **Calendar**: manages calendar entries for creation, deletion and modification.
8. **Storage**: manages the access to the database
   1. Subsystem Decomposition
9. **The User Interface (UI) Subsystem** is responsible for all interaction between the users and the system. The UI can be decomposed into group of dynamically loaded HTML pages, which are rendered in standard web browsers, e.g, Google Chrome, Mozilla Firefox. The entry, modification, and deletion of data will be handled directly through the UI.
10. **The User Management Subsystem** is responsible for the creation, modification, and deletion of the user accounts. In addition, this subsystem will handle the user authorization and authentication inquires, login requests, and will be able to identify the users account type. The subsystem will handle basic security issues such as checking duplicates accounts and the users input for code injections. The main controller can be found within this subsystem.

Use Cases

* Business Account Creation (#1)
* Consumer Account Creation (#2)
* Login for Account (#3)
* Password Recovery (#4)
* (Security 1) Storage Manipulation Prevention (#11)
* (Security 2) Multiple Accounts for One Person/Business Creation (#12)
* Consumer Account Editing (#7)

1. **The Search Subsystem** enables the user to search for public entities in the system using natural language or from advanced search interfaces. The search tool will implement a page ranking algorithm.
2. **The Notification Subsystem** is responsible for the formulation of alerts to the user from events that occur in the calendar and messaging subsystems. In addition, another task of the notification subsystem is to classify alerts with unique priority levels. Depending on the priority level, the notification of events may be transmitted to user through direct messages to the user or with alerts/changes in the UI.

1. **The Feedback Management Subsystem** will be responsible for managing recommendations and user reviews to any business account, as well as maintaining a rating score. It will enable the users to recommend the Golden Pages to potential users, as well . In addition, it will handle the consumers reviews by classifying them as *helpful* and *less helpful.* It will check for non-valid or duplicate reviews to a business account that might affect their rating.
2. **The Messaging Subsystem** is responsible for the processing, transport, and delivery of digital messages through the systems entities. In addition, it is responsible of maintaining an unique conversation between sender and receiver. The subsystem should maintain a queue of incoming and outgoing messages as they are processed. New events in the messaging subsystems will be pushed to the notification for classification.
3. **The Calendar Subsystem** is responsible for the creation, modification and deletion of calendar entries within the system. The subsystem will enable the business accounts to add invitations to the consumers as well as the consumers to request appointments to a business. Moreover, the subsystem will necessary maintain the calendar entries collision-free for the consumer account, i.e. no events may occur at the same time.

Uses Cases:

* Creating Appointment (#9)

1. **The Storage Subsystem** is responsible for all insertion, reading, updating and deletion of all data within the model, i.e. the database and the classes . No data will be accessed in any way except through and by the storage subsystem.
   1. Hardware and Software Mapping

The diagram below shows the deployment diagram for the Golden Pages System. The Golden Pages System is made of the following components:

* Business Computer
  + Browser
* Consumer Computer
  + Browser
* Web Host
  + Web Server – Apache
    - Messaging Subsystem
    - Calendar Subsystem
    - Search Subsystem
    - User Management Subsystem
    - Feedback Management Subsystem
* Unix Host
  + SQL Database

The Business’s and Consumer’s computers communicate with the Web Host via HTTP protocol using the Browser in their respective computers. The Web Host runs a Tomcat Application Server. All subsystems, the Messaging, Calendar, Search, User Management, and Feedback Management Subsystems, are located in the web server. The Web Host communicates with the Unix Host, which contains an SQL Database, using the JDBC API.

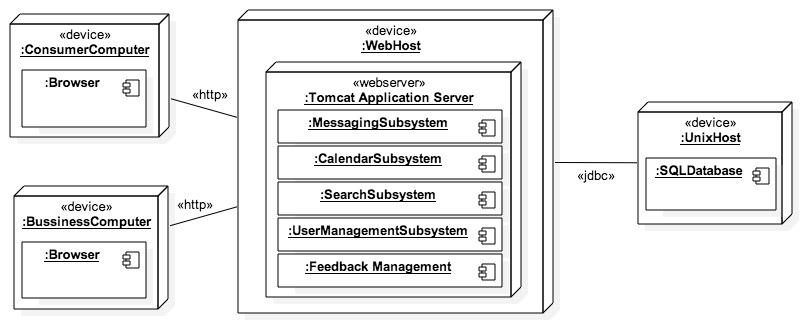


Figure 2.2

* 1. Persistent Data Management

Persistent data is stored data that needs to live or be accessible through multiple instances of the program’s runtime. Persistent data concerning Golden Pages is divided into two types: One of static nature but expandable in size at any time and the other which grows as more customer/business users join and create accounts. The latter consists mainly of user profile information and the former of events and data associated with business/customer accounts independent of their profiles.

* Customer/Business user profile data:

1. Unique username (associated with Full Name, Business Name, E-mail, and Password)
2. Full name (associated with unique username)
3. Business Name (associated with unique username)
4. Passwords (associated with unique username)
5. E-mail addresses (associated with unique username)
6. Miscellaneous user data (e.g. summary description, address, profile photo) (associated with unique username)

* Events and Data associated with accounts:

1. Appointments and events (associated between unique Business and Customer username)
2. Messages (associated between unique usernames)
3. Reviews and feedback (associated with unique Business usernames)
4. Notifications (associated with unique username)

Security is a main concern as personal data from profile accounts are stored in the database. For this reason the system will have different layers of protection. One is the decoupling of the User Interface subsystem from the Storage subsystem preventing users from accessing the database directly. Also, passwords will be encrypted to secure information in case of any database leakage and users input will never be treated as code to be executed by the system.

* 1. Security/Privacy

User authentication processes will be handled by the User Management subsystem as well as manage account creations, modifications and deletions. The subsystem will be the core of basic security measures involving use cases concerning with security and privacy, such as password recovery and checking for duplicate accounts.

Apart from validating login inquires and authorizations, in the use case of password recovery, the User Management subsystem will verify the business/customer account with the username or email provided through the User Interface subsystem, and send a confirmation request to the email associated with the account in the system for the user to change the account password.

In addition, the User Management subsystem will check for duplicate accounts at several times, one being at account creation by identifying the type of account being created and looking for usernames taken and duplicated business names (For business account creation). Furthermore, the subsystem will check periodically for spam customer accounts by analyzing identical data and malicious activities.

The only data to be encrypted is user passwords as the system does not store significant personal information requiring the encryption of large chunks of the database. For firewalls and server security measures, we are relying on the security services provided by the SQL database.

1. Detailed Design

Throughout this document, we have mainly been concerned with the system’s decomposition in terms of subsystems and use cases. However, the subsystems are sets of classes in an object oriented programming sense, and the use cases describe the behavior of those classes. This chapter focuses on the aspect of the design concerned with the classes themselves including design patterns, control flow, and contracts between classes.

* 1. Overview

The decomposition of the subsystems being implemented is represented in the following class diagram:

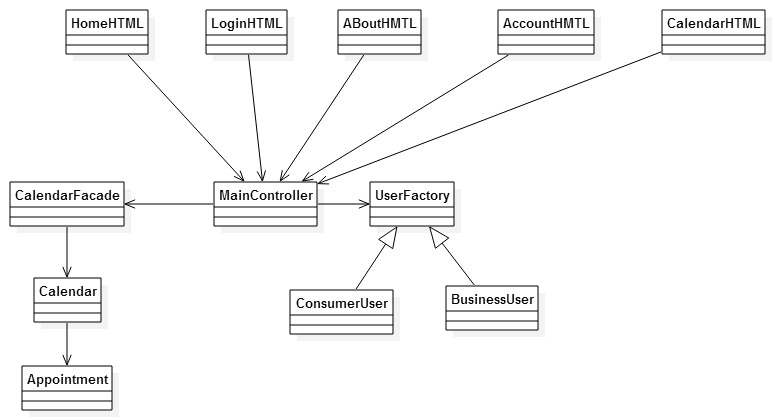


Figure 3.1

* The MainController class oversees the interactions between the user, userinterface, and the surrounding subsystems.
* The CalendarFacade class is the gateway to the Calendar class, this uses the facade design pattern to make sure there is only one entry point into the Calendar and Appointment classes.
* The Calendar class maintains the calendar for a given user as well as the appointment creation and deletion.
* The Appointment class contains information regarding an appointment with the client. The UserFactory class is used in the creation of both the business and consumer account, this class is using the abstract factory design.
* The ConsumerUser class manages a consumer user’s information.
* The BusinessUser class manages a business user’s information.

The ConsumerUser and BusinessUser both implement the command design structure in terms of operations that edit certain features, such as the business name. The HomeHTML, LoginHTML, AboutHTML, AccountHTML, and CalendarHTML represent the User Interface subsystem. That subsystem does not contain any classes, rather it contains pages that are dynamic based on what is passed back fromt the MainController class.

* 1. Object Interaction

The following are a series of sequence diagrams representing the use cases to be implemented. For a more detailed description of the use cases see Appendix B.

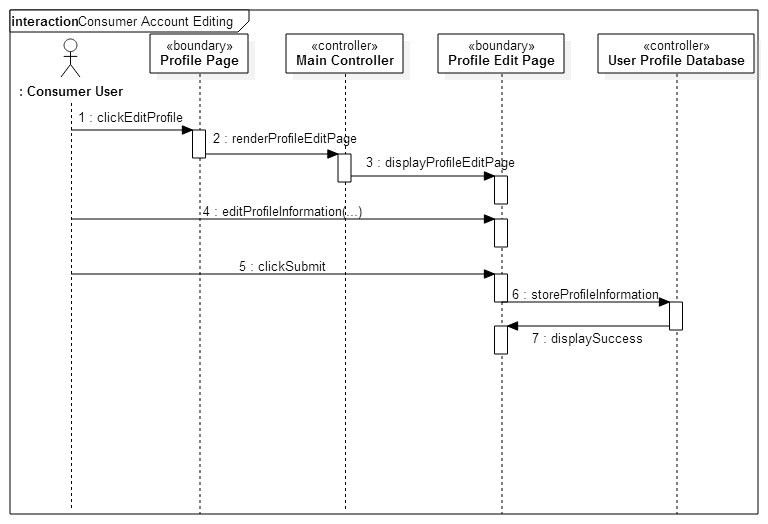


Figure 3.2 Consumer Account Editing

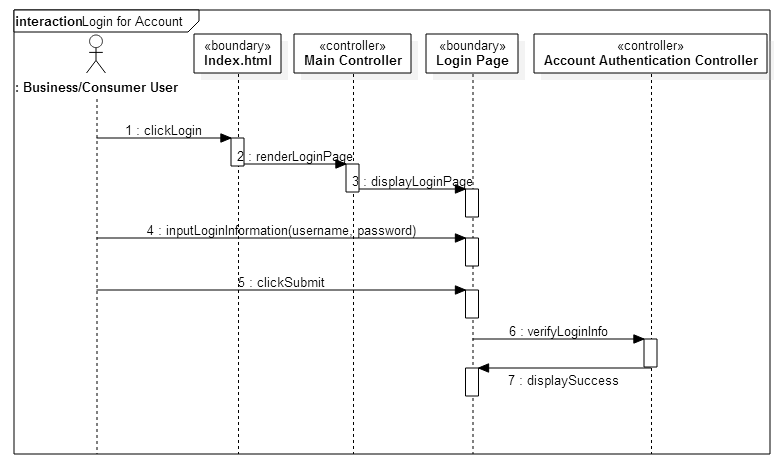
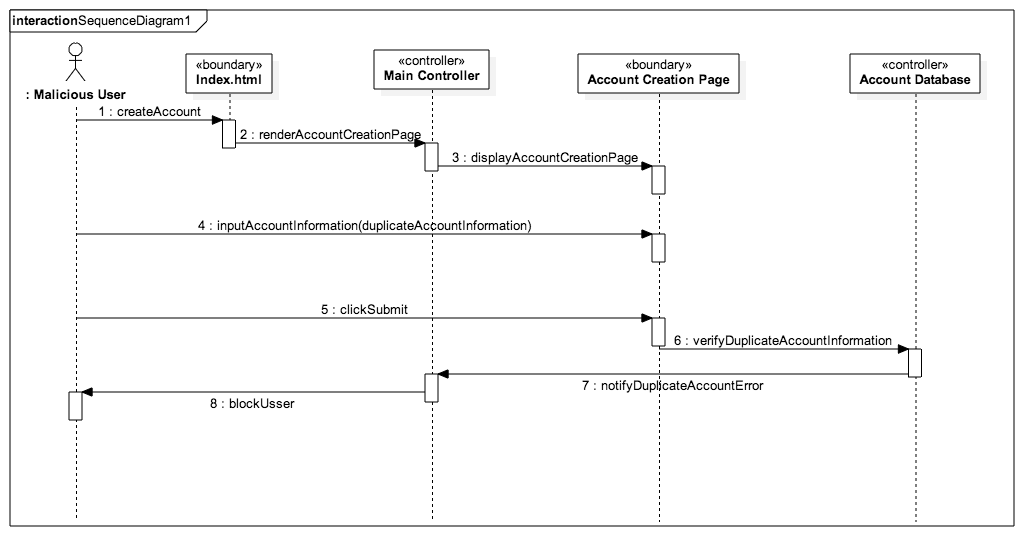
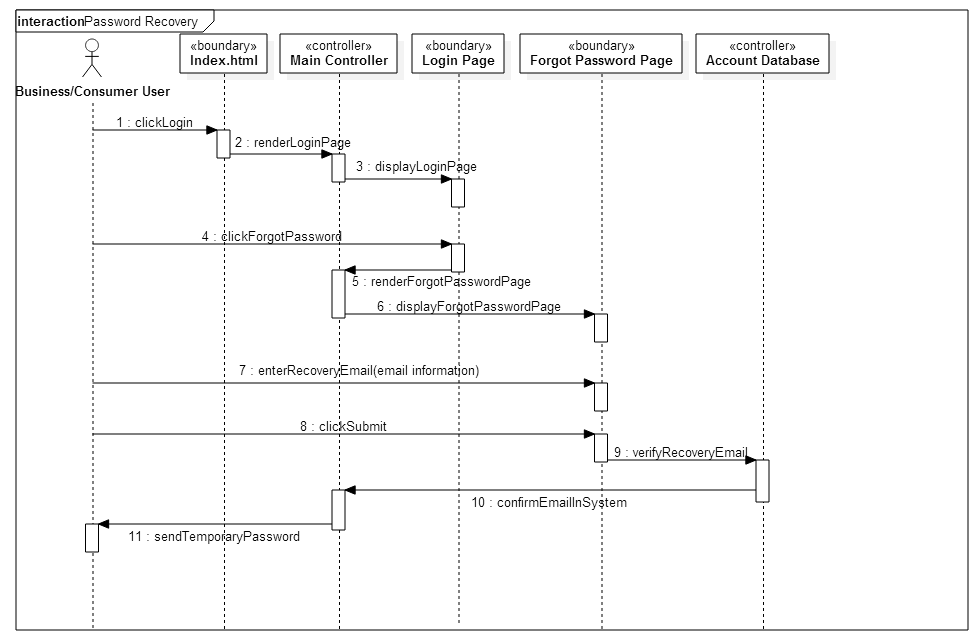


Figure 3.3 Login

Figure 3.4 Block multiple account creation

Figure 3.5 Password recovery

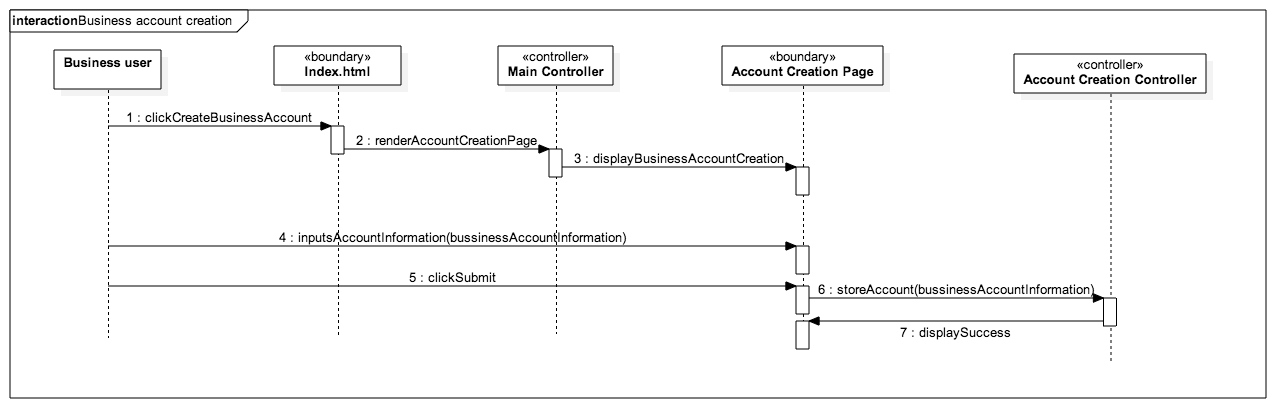


Figure 3.6 Create Business account

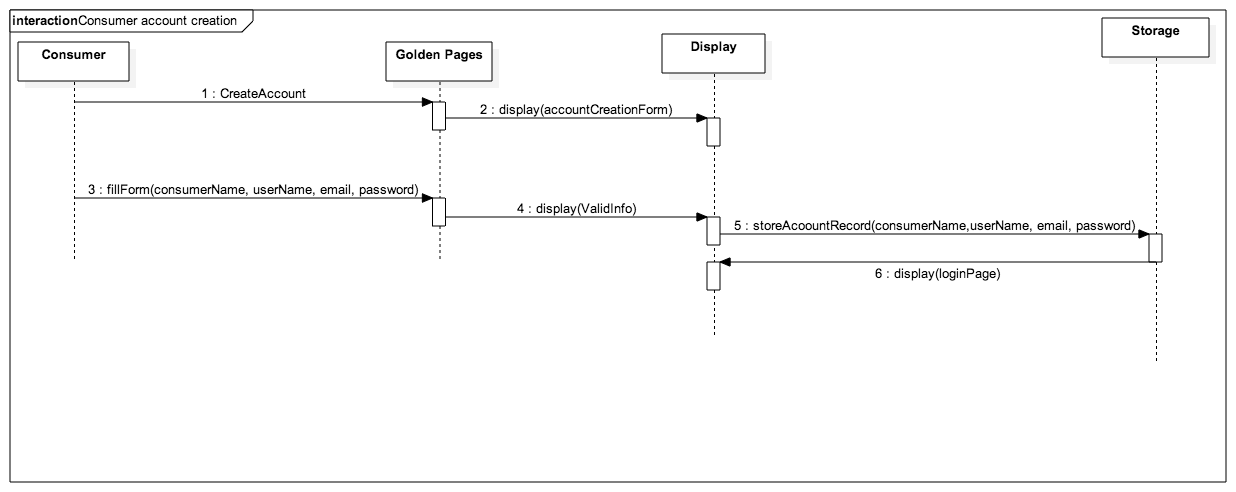
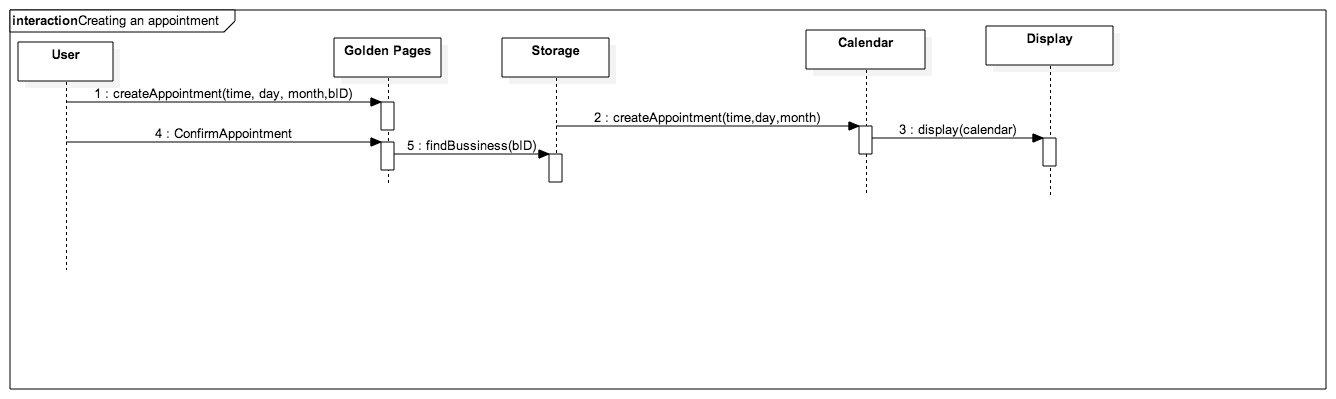
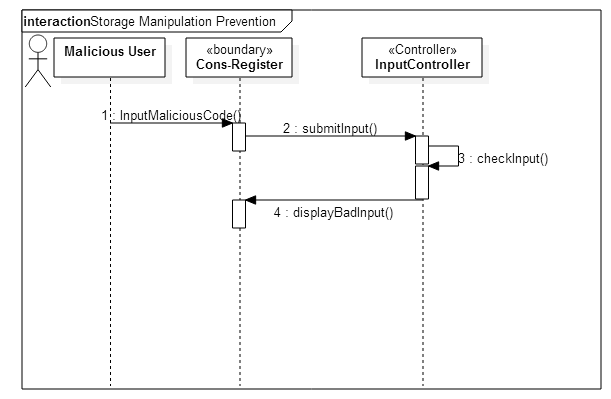


Figure 3.7 Create Consumer account

Figure 3.8 Appointment creation

Figure 3.9 Storage Manipulation prevention

* 1. Detailed Class Design

Each class in each subsystem has a specialized purpose, either to represent a critical component or to abstract the data for security reasons. This section goes into detail about each class and it’s role in the system.

* MainController
  1. Purpose: This class’ primary concern is handling the input coming from the user and passing it on to the appropriate subsystem, and controls which user is currently logged in at any given time. The class handles the logging in and logging out by attaching or removing the user id# to the current session. It also contains facade’s for the user’s system to be able to call upon without accessing the underlying logic.
  2. Attributes:
     + CalendarFacade myCalendarFacade
     + MessageFacade myMsgFacade
     + NotificationFacade myNotificationFacade
     + UserFactory myUserFactory
  3. Methods:
     + boolean doLogin(String username, String password, Session session)
     + void signup(String args[])
     + void logout(Session session)
     + void forgotPassword(String username)
     + boolean changePassword(String args[])
     + boolean changeEmail(String args[])
* BusinessUser
  1. Purpose: This class handles the creation and deletion of business users from the database. It also serves as a representation of the account for other subsystems.
  2. Attributes:
     + String businessName
     + String name
     + int uniqueID
     + String username
     + Calendar myCalendar
     + String email
  3. Methods:
     + String getName()
     + void createUser()
     + void deleteUser()
* ConsumerUser
  1. Purpose: This class handles the creation and deletion of Consumer users from the database. It also serves as a representation of the account for other subsystems.
  2. Attributes:
     + List<Appointments>: myAppts
     + String name
     + int uniqueID
     + String username
     + Calendar myCalendar
     + String email
  3. Methods:
     + String getName()
     + void createUser()
     + void deleteUser()
* CalendarFacade
  1. Purpose: This class’ purpose is to provide the user access to the needed methods of a calendar object but without giving them direct access to the calendar object. It does so by calling the appropriate methods for the calendar class within its own methods.
  2. Attributes: None
  3. Methods:
     + Calendar createNewCalendar()
     + void confirm()
     + void deny()
     + boolean isConfirmed()
* Calendar
  1. Purpose: This class’ purpose is manage the collection of appointments a Business User has. Maintaining the list of confirmed, pending, and free appointments, limited by time and how many concurrent appointments the user specifies as their limit.
  2. Attributes:
     + Map<Map Time> appts
     + int numAppts
     + long apptTimeRange
     + int numOfConcurrentAppts
     + long minApptLength
     + List freeApptSlots
  3. Methods:
     + void addAppt(Appointmet appt)
     + Appointment getApptByDate(Date)
     + void confirmAppt(Appointment)
     + Appointment getApptByTime(Time)
* Appointment()
  1. Purpose: This class’ purpose is to manage the individual appointment a consumer user makes with a business user. It is sent like a request to the business user for them to either confirm or deny.
  2. Attributes:
     + Time timeOfDay
     + String nme
     + String location
     + String dayOfWeek
     + Date dayByDate
  3. Methods:
     + void setTime(Time)
     + Time getTime()
     + Date getDate()
     + String getLocation()
     + void setLocation(String)
     + String getDayofWeek()
     + void setDate(Date)

We are using mainController as our “Main logic class” since it is the one that delegates all duties to the respective classes, as well as serving as the main gateway for all requests. The OCL for mainController is provided as follows.

doLogin():

context mainController::doLogin(String useName, String pWord, Session tSess) pre:

tSess.getAttribute(“user”) == NULL

context mainController::doLogin(String useName, String pWord, Session tSess) inv:

useName != NULL

context mainController::doLogin(String useName, String pWord, Session tSess) inv:

pWord != NULL

context mainController::doLogin(String useName, String pWord, Session tSess) inv:

tSess != NULL

context mainController::doLogin(String useName, String pWord, Session tSess) post:

Session.getAttribute(“user”) != NULL or user\_redirect\_AuthenticationError ==true

logout():

context mainController::logout(Session tSess) pre: tSess.getAttribute(“user”) != NULL

context mainController::logout(Session tSess) inv: tSess != NULL

context mainController::logout(Session tSess) pre: tSess.getAttribute(“user”) == NULL

signUp():

context mainController::signUp(List input, Session tSess) pre: Session.getAttribute(“user”) == NULL

context mainController::signUp(List input, Session tSess) inv: input != NULL

context mainController::signUp(List input, Session tSess) inv: tSess != NULL

context mainController::signUp(List input, Session tSess) inv: input.getSize() >= 6

context mainController::signUp(List input, Session tSess) inv: input.getFirst() == “Business” or input.getFirst() == “Consumer”

context mainController::signUp(List input, Session tSess) post: tSess.getAttribute(“user”) !=NULL or user\_redirect\_Sign\_up\_form\_error == true

forgotPassword():

context mainController::forgotPassword(String username) inv: username != null

context mainController::forgotPassword(String username) post: Reset\_PW\_Sent == true or user\_redirect\_username\_not\_found == true

changePassword():

context changePassword::changePassword(String args[]) pre: Session.getAttribute(“user”) !=NULL

context changePassword::changePassword(String args[]) inv: args[0] == userPass

context changePassword::changePassword(String args[]) inv: args[1] == args[2]

context changePassword::changePassword(String args[]) post: userPass == args[2]

changeEmail():

context changePassword::changePassword(String args[]) pre: Session.getAttribute(“user”) !=NULL

context changePassword::changePassword(String args[]) inv: args[0] == userPass

context changePassword::changePassword(String args[]) inv: args[1] == args[2]

context changePassword::changePassword(String args[]) post: userEmail == args[2]

1. Glossary

Business Account: An account created that represents a business, where a

business user can offer services.

Business User: A user that has and can manage a business account.

Consumer Account: An account created that represents an individual

consumer, that can be used to procure services from a

Business Account.

Consumer User: A user that has and can manage a user account.

Malicious User/System: A user or an automated system, that has the intention or

was designed to subvert the intended functionality of the system.

System Administrator: A privileged user that is able to oversee all accounts and

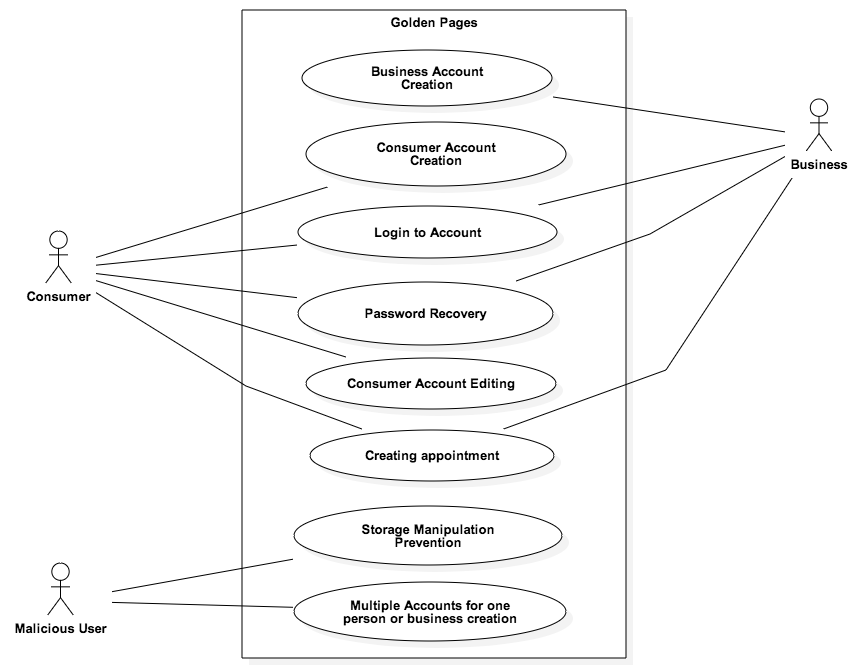
databases.

APPT: Abbreviation for “appointment”

GP: Acronym for “Golden Pages”

UML: Acronym for “Unified Modeling Language”

1. Appendix
   1. Appendix A – Use case diagram.



* 1. Appendix B - Use cases with nonfunctional requirements (from the SRD).

**Use Case ID:** GP01 – Business Account Creation

**Details:**

*Actor:*Business user

*Pre-conditions:*

1. Business user has navigated to main web page.
2. Business user has clicked link to Create Business Account.
3. Business user Creation page loads.

*Description:*

1. Use case begins when Business user clicks on Create Business Account
2. The system shall provide Business user with a template for data entry (See appendix C).
3. The Business user shall enter the following data: **business name, user full name, username, email, password, confirm password**
4. The Business user will then select create to have the system create the account.
5. The system shall notify the user if the information was entered correctly.
6. The system will generate a unique id and store the account record
7. The system will redirect the user to the logged in page

*Post-conditions:*

1. The user is logged in and the message “Account has been created successfully!” is displayed.
2. The system has stored a record of the new account.
3. Account has been created and stored on the system.

**Alternative Courses of Action**

1. In step D.4 the user resets the page or goes to a different page.
2. In step D.5 the system notifies user of any fields that are incorrect

**Exceptions:**

1. The account creation page is not loading/active
2. The system fails to store the record
3. The system does not redirect to the logged in page correctly

**Related Use Cases:**

GP03 User Log In.

**-----------------------------------------------------------------------------------------------------------------**

**Decision Support**

*Frequency:*Once per business account

*Criticality:*High. Allows the business account to be created which then allows t he rest of the system to function properly

*Risk:* Medium. Implementing this use case employs standard web technology

**-----------------------------------------------------------------------------------------------------------------**

**Constraints:**

1. Usability:
   1. No training time as it follows web standard account creation
   2. On average the user should take 5 minutes to complete
2. Reliability:

Mean time to Failure - 2% failures for every 5,000 attempts

1. Performance:
   1. Request should be sent, created, and saved within 5 seconds.
   2. System should be able to handle 50 requests in 1 minute.
2. Supportability:

The request should be correctly handled by IE, Mozilla, Chrome, Safari and their mobile equivalents.

**Modification History:**

*Owner:* Adam Merille

*Initiation date:*9/26/2014

*Date last modified:* 9/26/2014

**Use Case ID:** GP02 – Consumer Account Creation

**Details:**

*Actor:*Consumer user

*Pre-conditions:*

1. Consumer user has navigated to main web page.
2. Consumer user has clicked link to Create Consumer Account.
3. Consumer user Creation page loads.

*Description***:**

1. Use case begins when Consumer user clicks on Create Consumer Account
2. The system shall provide Consumer user with a template for data entry (See appendix C).
3. The Consumer user shall enter the following data: **consumer full name, username, email, password, confirm password**
4. The Consumer user will then select create to have the system create the account.
5. The system shall notify the user if the information was entered correctly.
6. The system will generate a unique id and store the account record
7. The system will redirect the user to the logged in page

*Post-conditions***:**

1. The user is logged in and the message “Account has been created successfully!” is displayed.
2. The system has stored a record of the new account.
3. Account has been created and stored on the system.

**Alternative Courses of Action**

1. In step D.4 the user resets the page or goes to a different page.
2. In step D.5 the system notifies user of any fields that are incorrect

**Exceptions:**

1. The account creation page is not loading/active
2. The system fails to store the record
3. The system does not redirect to the logged in page correctly

**Related Use Cases:**

GP03 User Log In.

**-----------------------------------------------------------------------------------------------------------------**

**Decision Support**

*Frequency:*Once per consumer account

*Criticality:*High. Allows the consumer account to be created which then allows the rest of the system to function properly

*Risk:*Medium. Implementing this use case employs standard web technology

**-----------------------------------------------------------------------------------------------------------------**

**Constraints:**

1. Usability:
   1. No training time as it follows web standard account creation.
   2. On average the user should take 5 minutes to complete.
2. Reliability:

Mean time to Failure - 2% failures for every 5000 attempts.

1. Performance:
   1. Request should be sent, created, and saved within 5 seconds.
   2. System should be able to handle 50 requests in 1 minute.
2. Supportability:

The request should be correctly handled by IE, Mozilla, Chrome, Safari and their mobile equivalents.

**Modification History:**

*Owner:* Adam Merille

*Initiation date:*9/26/2014

*Date last modified:*9/26/2014

**Use Case ID:** GP03 - Login to Account

**Details:**

*Actor:*User (Business User or Consumer User)

*Pre-conditions:*

User was able to create an account.

*Description:*

1. Use case begins when the User clicks the login option on the homepage.
2. The System shall display the login page to the User.
3. The User shall enter their username and password and click enter.
4. The System shall verify if the information of the User corresponds to that of a User in the database of registered accounts.
5. Use case ends when system responds by giving the user access to their account.

*Post-conditions:*

User has access to their account.

**Alternative Courses of Action:**

1. In step D.1 the User can exit from login page and return to homepage by clicking Cancel.
2. In step D.5 the system will alert the User if username and/or password are invalid.

**Exceptions:**

User cannot access the Login page.

**Related Use Cases:**

GP 01 - Business Account Creation

GP 02 - Consumer Account Creation

----------------------------------------------------------------------------------------------------------------------------

**Decision Support**

*Frequency:* The use of this system shall occur each time the user accesses their account and after an hour of inactivity.

*Criticality:*High. Successful functioning of this program will allow the user to access their account and prevent non-owners of the account from accessing the account.

*Risk:*High. Incorrect implementation of the login process could jeopardize an account.

----------------------------------------------------------------------------------------------------------------------------

**Constraints:**

1. Usability
   1. No training time needed, follows web standard login
   2. User should be able to login in under a minute
2. Reliability

Mean time to Failure - 5% every 10,000 attempts\

1. Performance

The system should be able to sustain multiple users logging in at the same time.

1. Supportability

The request should be correctly handled by IE, Mozilla, Chrome, Safari, and their mobile equivalents.

**Modification History:**

**Owner:** Shadeh Ferris-Francis

**Initiation date:** 09/13/2014

**Date Last Modified:** 09/27/2014

**Use Case ID:** GP04 – Password Recovery

**Details:**

*Actor*:Business, Consumer user

*Pre-conditions:*

1. The user has already created an account
2. User navigates to login page

*Description*:

1. Use Case begins when the user clicks on Forgot Password
2. The system provides the user with a template to write require information
3. User enters the following data: email or username
4. System notifies that email or username exists and that email was sent.
5. Use Case ends when the system sends an email to the user with its current password.

*Post-conditions:*

1. Email has been sent successfully.

**Alternative Courses of Action:**

In step D.4 system notifies the user that the username or email entered does not exist.

**Exceptions:**

1. The “forgot password” page fails to load
2. Server fails to send email
3. Email is rejected by the receiver server

**Related Use Cases:**

GP03- Login

**-------------------------------------------------------------------------------------------------------------------**

**Decision Support**

*Frequency*: On average, once every 90 days.

*Criticality:* High. Without it the user will not be able to access its account.

*Risk*: Medium. Implementing this use case employs standard web-based technology.

**---------------------------------------------------------------------------------------------------------------------**

**Constrains:**

1. Usability:
   1. No previous Training Time
   2. Help facility should provide 1 help frame on the topic
   3. On average the user should take 5 mins to complete the send request form.
2. Reliability:
   1. Mean time to Failure – 5% failures for every twenty four hours of operation is acceptable.
   2. Availability – Down time for Login Back-up 30 minutes in a 24 hour period.
3. Performance:
   1. Request should be sent and saved within 5 secs.
   2. System should be able to handle 100 request in 1 minute
4. Supportability:

The request should be correctly handled by Internet Explorer, Firefox, Chrome, Safari and their mobile equivalents.

**Modification History**

*Owner*: Andy D. Martinez

*Initiation date*: 09/26/2014

*Date last modified*: 09/27/2014

**Use Case ID:** GP07 – Consumer Account Editing

**Details:**

Actor: Customer user

Pre-Conditions:

1. Customer has logged into their account successfully.
2. Customer is currently viewing their personal page.

Description:

1. Use case begins when Customer selects edit account from menu.
2. The system shall provide a template for data entry, and the current data for the user will be filled into each respective form
3. The user edits information within the form.
4. The user, when complete, will select confirm.
5. The system will prompt the user to re-enter their password for confirmation.
6. The system will confirm that the password entered is correct.
7. If the password is correct the system will store the new information and overwrite the old.
8. The system will then return the user to their account page displaying the updated information.
9. Use Case ends.

**Alternative Courses of Action**

1. In step D4 the user leaves the page or reloads the page canceling all changes.
2. In step D6 the password is found to be incorrect, the System will then prompt the user again.

**Exceptions.**

1. 3The account editing page is not loading/active.
2. The system fails to update the record.
3. The system does not redirect the user to their account page.

**Related Use Cases:**

GP02 Consumer Account Creation

GP03 User login

----------------------------------------------------------------------------------------------------------------------------

**Decision Support**

*Frequency:* Multiple times per account.

*Criticality*: Medium. Allows consumers to have current contact information for businesses.

*Risk:* Medium. Implementing this use case employs standard web technology

----------------------------------------------------------------------------------------------------------------------------

**Constraints:**

1. Usability:
   1. No training time as it follows web standard account editing
   2. On average should take less than 5 minutes to complete dependant on amount of information changed.
2. Reliability

5% failures for a 24 hour run time is acceptable

1. Performance
   1. Request for edit should be sent, updated, and saved within 5 seconds.
   2. System should be able to handle 50 requests per minute
2. Supportability

The request should be correctly handled by Internet Explorer, Firefox, Chrome, Safari and their mobile equivalents.

**Modification History:**

*Owner:* Michael Machin

*Initiation date:*9/25/2014

*Date last modified:* 9/26/2014

**Use Case ID:** GP09 – Creating Appointment

**Details:**

*Actor:*Business/Consumer user

*Pre-conditions:*

1. Respective user has navigated to main web page.
2. Respective user has created an account.
3. Respective user has logged into account. .
4. Respective user has clicked in Create Appointment feature in its main page.

*Description:*

1. Use case begins when the user clicks on Create Appointment button in its main/profile page.
2. The system analyzes the required information such as **Time**, **Day**, **Month**, and **Year** from the server.
3. The system displays a weekly calendar for the user to choose the day for the appointment.
4. The system opens up a pop-up window where the user shall enter the **Business** and **Time** slot available to create the appointment.
5. The user confirms the creation of the appointment.
6. The system creates an appointment.
7. The system closes the pop-up window and displays the calendar with the appointments created.

*Post-conditions:*

1. The system shall notify the user the appointment was created successfully.
2. The system has associated and stored the appointment to the respective account.
3. The system sends a notification to the business associated with the appointment for confirmation.

**Exceptions:**

1. The system fails to get the required information to display the calendar correctly: Time, Day, Month, and Year.
2. The system fails to create the appointment.
3. The system fails to display the calendar with the appointments after successful creation.

**Related Use Cases:**

GP14 Consumer Selection of Appointment Time

GP15 Business Acceptance of Appointment Time

**-----------------------------------------------------------------------------------------------------------------**

**Decision Support**

*Frequency:*As many times per account.

*Criticality:*High. Main functionality of the system

*Risk:* Medium. Implementing this use case employs standard web technology.

**-----------------------------------------------------------------------------------------------------------------**

**Constraints:**

1. Usability:
   1. On average the user should take less than 5 minutes to complete.
2. Reliability:
   1. Mean time to Failure – 1% failures for every 5,000 attempts
3. Performance:
   1. Appointment creation should take less than 4 seconds.
   2. System should be able to handle 100 requests in 1 minute.
4. Supportability:
   1. The request should be correctly handled by IE, Mozilla, Chrome, Safari and their mobile equivalents.

**Modification History:**

*Owner:* Sergio Saucedo

*Initiation date:*9/28/2014

*Date last modified:* 9/30/2014

**Use Case ID:** GP11 – Storage Manipulation Prevention

**Details:**

*Actor:*Malicious user or system

*Pre-conditions:*

1. User is on page with text input box that is linked to system storage.

*Description***:**

1. Use case begins when user is on page with text input.
2. User inputs malicious code in text box
3. User clicks a button that updates system storage.
4. System checks input for malicious code.
5. System notifies user of improperly formatted input.

*Post-conditions***:**

1. The system storage is unaffected.
2. The system displays “Input format incorrect” to user.

**Alternative Courses of Action**

1. In step D.3 the user navigates to a different page

**Exceptions:**

1. The system fails to check input correctly
2. The system fails to notify user of improper format

**Related Use Cases:**

GP01 Business account creation

GP02 Consumer account creation

GP10 Message creation

**-----------------------------------------------------------------------------------------------------------------**

**Decision Support**

*Frequency:*15 times per hour

*Criticality:*High. Without the system storage could be corrupted and therefore affect the entire system and other users.

*Risk:*Medium. Implementing this use case employs standard web technology

**-----------------------------------------------------------------------------------------------------------------**

**Constraints:**

1. Usability:

System should perform automatically.

1. Reliability:

Mean time to Failure - 2% for every 1,000 attacks.

1. Performance:

System should check without user realizing, should finish < 1 sec.

1. Supportability:

The request should be correctly handled by IE, Mozilla, Chrome, Safari and their mobile equivalents.

**Modification History:**

*Owner:* Adam Merille

*Initiation date:*9/26/2014

*Date last modified:*9/26/2014

**Use Case ID:** GP12 – Multiple Accounts for One Person/Business Creation

**Details:**

*Actor:*Malicious user

*Pre-conditions:*

1. Malicious user has navigated to the main web page.
2. Malicious user has a registered account already.
3. Malicious user has clicked link to Create Business/Consumer Account.

*Description:*

1. Use case begins when malicious user clicks on Create Business/Consumer Account.
2. The system shall provide malicious user with a template for data entry.
3. The malicious user is asked to provide the following data: **business name** (if creating a business account), **user full name**, **username**, **email**, **password**, and **confirm password**.
4. The system will check for duplicated emails associated with other accounts and business names.
5. If the system finds duplicated data with stored accounts, an error will be displayed, stopping any request from creating a new account.
6. If more than 3 faulty tries are done, the system will timed out any future request from the computer by blocking any communication coming from the malicious IP number.

*Post-conditions:*

1. The system will display a timed out error and notify the user to try later again.
2. In the case accounts are made, the system will check periodically for troll accounts and suspend them.
3. If no appeals are made, they will be eventually banned and removed.

**Exceptions:**

1. The system fails to time out after 3 faulty tries.
2. The system fails to check for duplicated data stored in the server.
3. The system fails to identify troll accounts.

**Related Use Cases:**

GP01 Business Account Creation

GP02 Consumer Account Creation

**-----------------------------------------------------------------------------------------------------------------**

**Decision Support**

*Frequency:*More than once per hour.

*Criticality:*High. Provides safety and security for genuine accounts

*Risk:* High. Implementing this use case employs advance security protocols

**-----------------------------------------------------------------------------------------------------------------**

**Constraints:**

1. Usability:

System should perform this security measure automatically.

1. Reliability:

Mean time to Failure – 10% failures for every 1,000 attempts

1. Performance:
   1. System should check for duplicated data within 3 seconds.
   2. System should be able to identify foul account creation and record the numbers of tries tried.
2. Supportability:

The request should be correctly handled by IE, Mozilla, Chrome, Safari and their mobile equivalents.

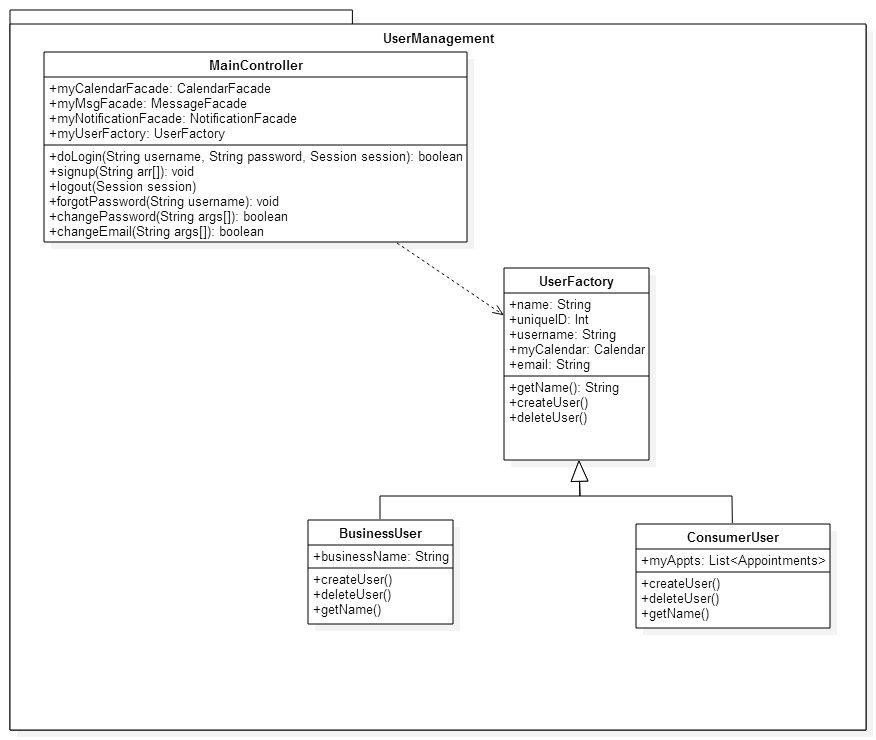
**Modification History:**

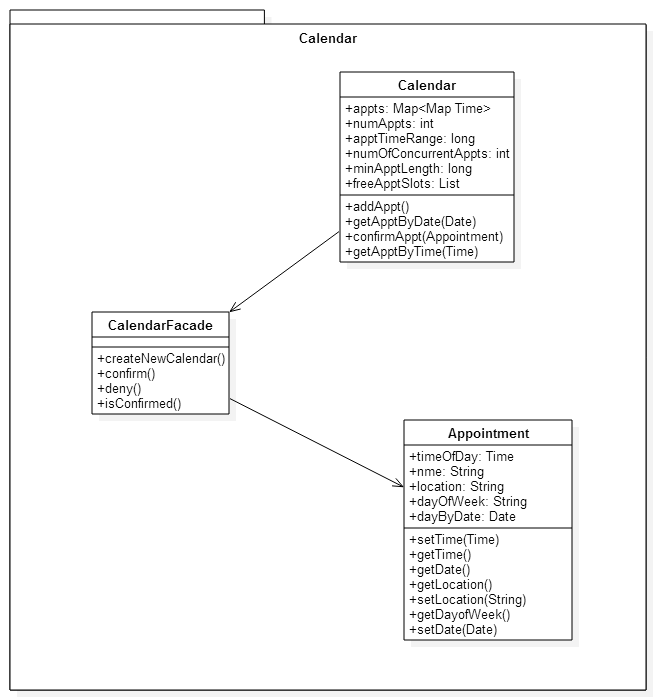
*Owner:* Sergio Saucedo

*Initiation date:*9/28/2014

*Date last modified:* 9/28/2014

* 1. Appendix C – Detailed class diagrams (per package).





5.4 Appendix D – Class Interfaces

/\*\*

\*

\* This class’ primary concern is handling the input coming from the

\* user and passing it on to the appropriate subsystem, and controls which user

\* is currently logged in at any given time. The class handles the logging in

\* and logging out by attaching or removing the user id# to the current session.

\* It also contains facade’s for the user’s system to be able to call upon

\* without accessing the underlying logic.

\*/

public class MainController {

/\*\*

\* These Serve as the Facades to access the hidden logic

\*/

CalendarFacade myCalendarFacade;

MessageFacade myMsgFacade;

NotificationFacade myNotificationFacde;

/\*\*

\* myUserFactory is used to create and store different users

\*/

UserFactory myUserFactory

/\*\*

\* Receives the login name and password, if the the Session does not already

\* have a user logged in it searches the database for the user and matches

\* with the password. If either of these fail the method returns false

\* @param username

\* @param password

\* @param session

\* @return wheather or not the login was succesful

\*/

boolean doLogin(String username, String password, Session session){ /\* ... \*/ }

/\*\*

\* Receives all the information from the sign up page fields, if there is not

\* a user currently logged in and the user name doesn't already exist an

\* account is created and stored in the database and a true is returned else

\* false

\* @param args

\*/

void signup(String args[]){ /\* ... \*/ }

/\*\*

\* Removes the user id from session paramater, logging them out.

\* @param session

\*/

void logout(Session session){ /\* ... \*/ }

/\*\*

\* Locates the account associated with the username and sends an automated

\* email to the stored email address with the new randomly generated

\* password

\* @param username

\*/

void forgotPassword(String username){ /\* ... \*/ }

/\*\*

\* Confirms that the first item in args matches the password of the current

\* user, and confirms that the second two arguments are exactly the same,

\* then updates the user recored with the new password and returns true else

\* false.

\* @param args

\* @return

\*/

boolean changePassword(String args[]){ /\* ... \*/ }

/\*\*

\* Confirms that the first item in args matches the password of the current

\* user, and confirms that the second two arguments are exactly the same,

\* then updates the user recored with the new email and returns true else

\* false.

\* @param args

\* @return

\*/

boolean changeEmail(String args[])

}

/\*\*

\*This class’ purpose is to provide the user access to the needed methods of a

\* calendar object but without giving them direct access to the calendar object.

\* It does so by calling the appropriate methods for the calendar class within

\* its own methods.

\*

\*/

public class CalendarFacade {

/\*\*

\* Creates a new calendar for the business user

\* @return The new calendar

\*/

Calendar CreateNewCalendar(){ /\* ... \*/ }

/\*\*

\*

\* Confirms an appointment on the calendar, and messages the consumer.

\*/

void confirm(){ /\* ... \*/ }

/\*\*

\* Denies an appointment on the calendar and removes it from the list and

\* messages the user.

\*/

void deny(){ /\* ... \*/ }

/\*\*

\* Confirms if an appointment has been confirmed

\* @return True if appointment has been confirmed else false

\*/

boolean isConfirmed(){ /\* ... \*/ }

}

/\*\*

\*This class’ purpose is manage the collection of appointments a Business User

\* has. Maintaining the list of confirmed, pending, and free appointments,

\* limited by time and how many concurrent appointments the user specifies as

\* their limit.

\*/

public class Calendar {

/\*\*

\* appts stores all the pending and confirmed appointments

\* and numAppts keeps a running tally

\*/

Map<Map Time> appts;

int numAppts;

/\*\*

\* numOfConcurrentAppts stores how many appointments can be made for

\* a certain time block, while minApptLength maintains how long to divide

\* each time block into

\*/

int numOfConcurrentAppts;

long minApptLength;

/\*\*

\* freeApptSlots stores all the available appointments that have not been requested

\*/

List freeApptSlots;

/\*\*

\* This method Creates a calendar object with default attributes

\*/

public Calendar() { /\* ... \*/ }

/\*\*

\* This method adds the param appt to the Map appts with the

\* confirmed set to false

\* @param appt

\*/

void addAppt(Appointment appt){ /\* ... \*/ }

/\*\*

\* This method returns all appointments associated with the parameter

\* date from the map appts

\* @param date

\* @return List<Appointment>

\*/

Appointment getApptByDate(Date date){ /\* ... \*/ }

/\*\*

\* This method locates the param appt in the map and sets it to be confirmed

\* @param appt

\*/

void confirmAppt(Appointment appt){ /\* ... \*/ }

/\*\*

\* This method returns all appointments associated with the parameter

\* time from the map appts

\* @param time

\* @return List<Appointment>

\*/

Appointment getApptByTime(Time time){ /\* ... \*/ }

}

/\*\*

\*This class’ purpose is to manage the individual appointment a consumer user

\* makes with a business user. It is sent like a request to the business user

\* for them to either confirm or deny.

\*/

public class Appointment {

/\*\*

\* Stores all pertinent information regarding appointments

\*/

Time timeOfDay;

String nme;

String location;

String dayOfWeek;

Date dayByDate;

/\*\*

\* Constructor using the time, Consumer name, address, and date

\* @param time

\* @param Name

\* @param Location

\* @param date

\*/

public Appointment(Time time, String Name, String address, Date date)

{ /\* ... \*/ }

/\*\*

\* Sets the time of the appointment

\* @param time

\*/

void setTime(Time time){ /\* ... \*/ }

/\*\*

\* Returns the time of the appointment

\* @return time of appointment

\*/

Time getTime(){ /\* ... \*/ }

/\*\*

\* Returns the date of the appointment

\* @return date of appointment

\*/

Date getDate(){ /\* ... \*/ }

/\*\*

\* Returns the address of the appointment

\* @return address

\*/

String getLocation(){ /\* ... \*/ }

/\*\*

\* Changes the address of the appointment

\* @param address of the business

\*/

void setLocation(String address){ /\* ... \*/ }

/\*\*

\* returns the stored day of week

\* [Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday]

\* @return day of week

\*/

String getDayofWeek(){ /\* ... \*/ }

/\*\*

\* Changes the date of the appointment

\* @param date

\*/

void setDate(Date date){ /\* ... \*/ }

}

5.5 Appendix E – Diary of Meetings and Tasks.

**Diary Entry** - October 28, 2014

**Location:** FIU ECS Building

**Start time:** 3:30 pm

**End time:** 4:00 pm

**In attendance:** Jorge, Shadeh, Adam, Michael, Andy, Sergio

**Late:** N/A

**Agenda:** Discuss requirements for Deliverable 2

**Summary of discussion:** Planned a course of action for the weeks up until the due date for Deliverable 2. Voted on which mode of meeting was more convenient: meeting in person, or meeting on Skype.

**Assigned tasks:** Go over class notes for a better understanding of the topics that will be covered in Deliverable 2

**Diary Entry** - November 2, 2014

**Location:** Skype, Google Docs

**Start time:** 10am

**End time:** 10:30am

**In attendance:** Jorge, Shadeh, Andy

**Late:** N/A

**Agenda:** Divide the Tasks for Deliverable 2

**Summary of discussion:** Teammates selected portions of Deliverable 2 to work on

**Assigned tasks:** Begin working on respective parts of Deliverable 2

**Diary Entry** - November 10, 2014

**Location:** FIU ECS Building, Google Docs

**Start time:** 11am

**End time:**  12pm

**In attendance:** Jorge, Shadeh, Adam, Michael, Andy, Sergio

**Late:** N/A

**Agenda:** Refine Diagrams, Finish Deliverable 2

**Summary of discussion:** Teammates gave input on how the subsystems should be laid out and what classes should be present for the software.

**Assigned tasks:** Complete diagrams, code, and finish Deliverable 2

**Diary Entry** - November 11, 2014

**Location:** FIU ECS Building, Google Docs, Skype

**Start time:** 1pm

**End time:** 7pm

**In attendance:** Jorge, Shadeh, Adam, Michael, Andy, Sergio

**Late:** N/A

**Agenda:** Finish Deliverable 2

**Summary of discussion:** Finished Deliverable 2, completed and cleaned up diagrams.

**Assigned tasks:** Finish Deliverable 2