Panther Buddy

Design Document

CEN 5011 Advanced Software Engineering

Prof. Peter Clarke

Team 6:

Abhinav Dutt

Allan Shaji Manamel

Sharat Kedari

Yulong Qiu

Abdur Rahman

Bin Shahid

Yue Wu

James Angelo

November 4, 2015

Abstract

Panther Buddy is a web application through which users interested in Florida International University can get access to relevant information related to majors, housing, etc. On Panther Buddy, different types of information will largely be generated by users. Once a user is registered on Panther Buddy, he or she will be able to post questions, view profile of other users. As a result, other users will be benefited from this sort of interaction.

In the SRD, i.e. first deliverable, we introduce the use case models for major functionalities. Using the 12 most important use cases, the team also created scenarios, object diagrams, sequence diagrams, and activity diagrams. During requirement elicitation and analysis, the team defined the actors, scenarios, and use cases. These artifacts were formalized using the use case, object, class, and sequence diagrams. As the second deliverable, in this design document, we presents the functional and non-functional requirements, design methodology, proposed architecture, and detailed design of PantherBuddy in detail. The primary architecture used is the 3-tier Architecture (A subset of the Multi-Tier architecture) with the goal of achieving flexibility and reusability, with the secondary architecture as Model View Controller (MVC) architecture.

Table of Content

1. Introduction

This chapter covers an overview of the system, including the purpose of the system, different terminologies and technical jargons required to understand the document. It also covers a detailed presentation of the functional and non-functional requirements and design methodology.

1.1. Purpose of System

This project consists of the creation of a web application called Panther Buddy, which will allow any user interested in Florida International University to have access to an online message board in order to post relevant information regarding majors, housing, etc. and also view messages posted by other users of the system. Any user can register on Panther Buddy. Once a user is registered, he or she will have access to posts and questions on the message board. A registered user can also post questions and insights that can benefit other users.

1.2. Requirements

1.2.1 Functional Requirements

- The System shall allow a registered user to login (Appendix B – Use case PB_UC20_LOGIN “User Login”).
- The system shall allow a logged-in registered user to log out (Appendix B – Use case PB_UC21_LOGOUT “User Logout”).
- The system shall provide a way for a first time user, to register (Appendix B – Use Case PB_UC1_USER_REGISTRATION “User Registration”).
- The system shall activate the profile of a newly-registered user (Appendix B – Use Case PB_UC08_ACTIVATE_USER “Activate User”).
- The system shall allow a registered user to post messages (Appendix B – Use Case PB_UC09_POST_MESSAGE “Post Message”).
- The system shall provide a way for a registered user to change his or her password (Appendix B – Use Case PB_UC03_CHANGE_PASSWORD “Change Password”).
- The system shall provide a way for a registered user to recover his or her password (Appendix B – Use Case PB_UC02_RECOVER_PASSWORD “Recover Password”).

- The system shall allow a registered user to delete his or her posted messages (Appendix B – Use case PB_UC14_DELETE_MESSAGE “Delete Message”).
- The system shall be able to prevent XSS attacks (Appendix B – Use Case PB_SC2_XSS_FILTERING “XSS Filtering”).
- The system shall be able to prevent SQL injection-based attacks on login (Appendix B – Use Case PB_SC1_FILTER_INPUT “SQL Injection Prevention on Login”).
- The system shall allow a registered User to view posts (Appendix B – Use Case PB_UC16_VIEW_POST “View Post”).

1.2.2. Non-functional Requirements

- **Performance** – Data will be saved or viewed in 2 secs. See use cases PB_UC01_USER_REGISTRATION, PB_UC20_LOGIN, PB_UC02_RECOVER_PASSWORD, PB_UC11_POST_MESSAGE, PB_UC04_VIEW_OWN_PROFILE, PB_UC08_ACTIVATE_USER, PB_UC21_LOGOUT, PB_UC16_VIEW_POST, PB_UC14_DELETE_MESSAGE (Appendix B)

- **Supportability** – For supportability, any modern browser such as Chrome, Firefox, Safari, and Edge that supports HTML5 should support the application. All use cases in Appendix B.

- **Implementation** – The frontend or presentation tier of the system will be implemented using the Bootstrap framework and JSP. For the business tier of the system, Wildfly 8.2.0 server with Java will be used. For the data tier of the system, MySQL database 5.6, MySQL workbench 6.3, Connector/ODBC 5.3.5, and connector/J 5.1.36 will be used. See all use cases in Appendix B

- **Usability** – No previous training needed for any use case.

- **Reliability** –

    1. One failure in 2 months operation. See use cases PB_UC01_USER_REGISTRATION, PB_UC20_LOGIN, PB_UC02_RECOVER_PASSWORD, PB_UC04_VIEW_OWN_PROFILE, PB_UC21_LOGOUT, PB_UC16_VIEW_POST (Appendix B)
    2. One failure every six months for change password use case. See use case PB_UC03_CHANGE_PASSWORD, (Appendix B)
    3. One failure in 24 hours see use cases PB_UC11_POST_MESSAGE, PB_UC08_ACTIVATE_USER, PB_UC14_DELETE_MESSAGE (Appendix B)

1.3. Design Methodology

In this section, we present and describe the design methodology that we use for our project. We also identify the software process models to design our project. Additionally, we will introduce the types of UML models used to represent our design. For this project, we use the Unified Software Development Process (USDP) model. USDP is a component-based and usecase-driven model that uses the Unified Modeling Language (UML) to represent the different models of our software system. USDP also allows incremental development of the system. That's way, we can iterate from analysis to design and vice versa. In order to present a visual blueprint of the different artifacts of the system, we use the UML 2.0 notation. Use of UML 2.0 allows us to create use case diagrams, class diagrams, sequence diagrams, state machines, deployment diagram and UML profiles.

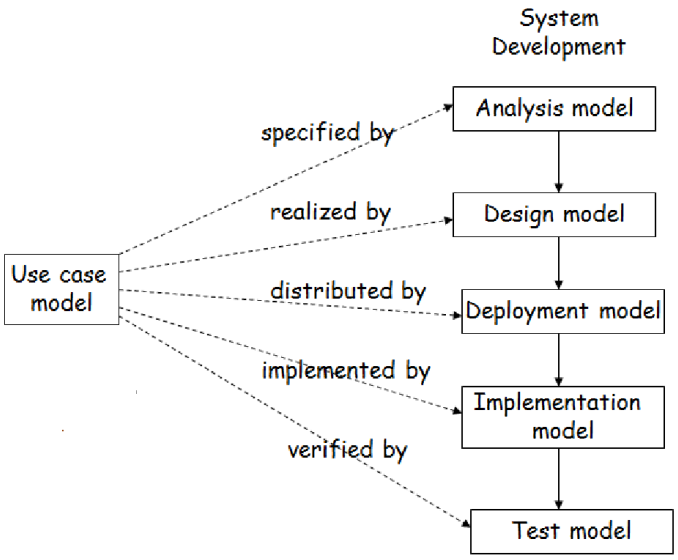


Figure: Shows the USDP process

1.4. Definitions, Acronyms, and Abbreviations

UML - Unified Modeling Language

USDP – Unified Software Development Process Model

ER Diagram – Entity Relationship Diagram

XSS - Cross-site scripting

HTML - Hyper Text Markup Language

MVC – Model View Controller architecture for software systems.

1.5. Overview of Documents

In section 2 of the document, we give an in-depth description of the system in terms of its software architecture, system decomposition, hardware to software mapping, data management, and security management. In Section 3, we describe the class diagrams, state machines, object interactions, and class designs. We present the glossaries and references in section 4 and 5, respectively. Finally, in section 6, we present the appendix.

2. Proposed System Software Architecture

In this chapter, architectural patterns and design patterns are determined. Firstly, we will introduce the system architecture used in our project along with the identification and description of the architecture pattern used. In section 2.1 an overview of the system in terms of its subsystems will be given. In section 2.2 the subsystem decomposition will be given in terms of responsibilities and dependencies. The mapping of the subsystems to hardware described by the software architecture will be presented in subsections 2.3. In section 2.4 persistent data management will be explained. In section 2.5 security management will be specified for the system.

2.1. Overview

In this part, package diagrams including the major and minor subsystem will be given showing, an overview of the system. In the package diagram shown below we can observe the main components of our system architecture implementing the architectural patterns.

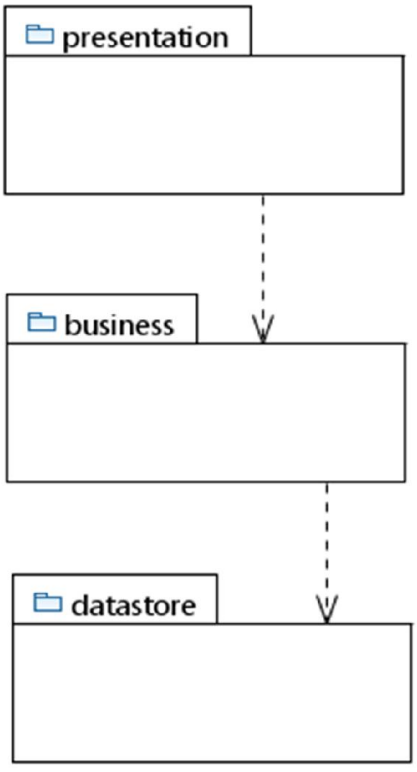


Figure: Represents the High Level 3-Tier Architecture.

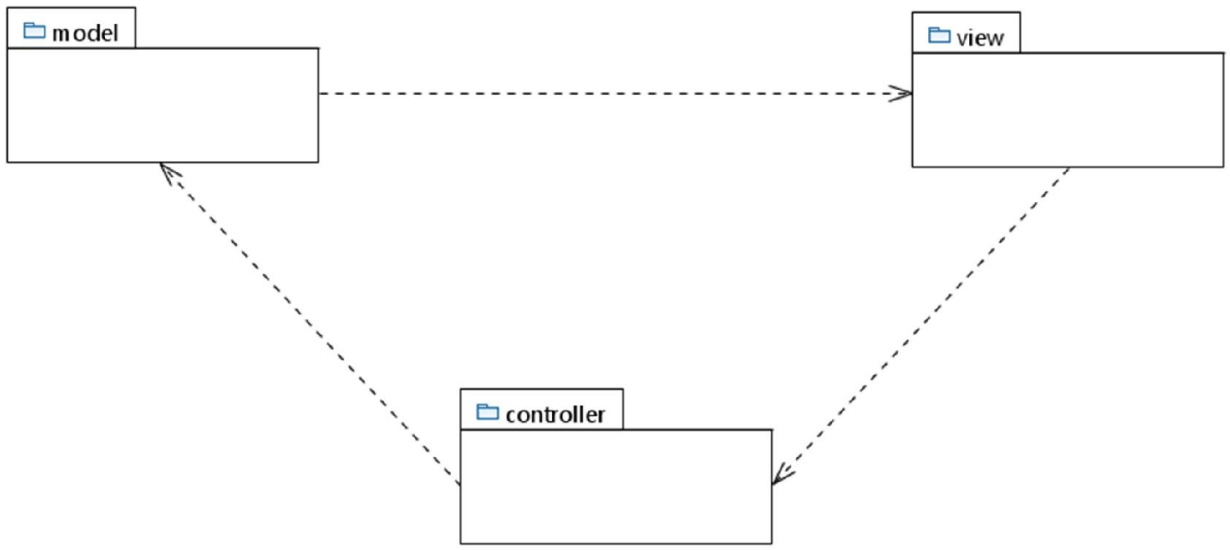


Figure: MVC Architecture

System Architecture:

The architectural patterns that are used for the system design are the three-tier (3-tier) and Model View Controller (MVC) architecture.

The main reason for using three-tier (3-tier) was based on

- **Maintainability and reusability**

Single responsibility principle: GUIs change at a very different rate, and for very different reasons, than business rules. Database schemas change for very different reasons, and at very different rates than business rules. Keeping these concerns (GUI, business rules, and database) separate is good design and makes it easier to apply object oriented concept. Define any logic once within the business layer and that logic can be shared among any number of components in the presentation layer.

The other reasons are:

1. Flexibility

By separating the business logic of an application from its presentation logic, a 3-Tier architecture makes the application much more flexible to changes

2. Maintainability

Changes to the components in one layer should have no effect on any others layers. Also, if different layers require different skills (such as HTML/CSS is the presentation layer, Java in the business layer, SQL in the data access layer), changes can be managed by independent teams with skills in those specific areas.

3. Reusability

Separating the application into multiple layers makes it easier to implement re-usable components. A single component in the business layer, for example, may be accessed by multiple components in the presentation layer, or even by several different presentation layers (such as desktop and the web) at a time.

4. Scalability

A 3-Tier architecture allows distribution of application components across multiple servers thus making the system much more scalable.

5. Reliability

A 3-Tier architecture, if deployed on multiple servers, makes it easier to increase the reliability of a system by implementing multiple levels of redundancy.

6. Security

A physically separate middle-tier application server can increase security because it adds an extra level of indirection between the web server and the database. This means no direct route from the web server to the database server and (e.g.) SQL protocols/ports don't need to be allowed/opened. If your web server gets hacked, your application server is safe. Also, the attack surface on the web server is reduced because there is a reduced amount of code, etc. running on the web server.

7. Availability

If the Application tier server is down and caching is sufficient, the Presentation tier can process Web requests using a cache.

The main reason for using **Model View Controller (MVC)** was based on:

1. Same data is being rendered in different ways on the page
2. Presence of highly reusable visual elements on the screen.
3. Many trivial interactions on the page.

2.2. Subsystem Decomposition

1) **Presentation subsystem:** Occupies the top level and displays information related to services available on a website. This tier communicates with other tiers by sending results to the browser and other tiers in the network. The presentation subsystem is responsible for the implementation and display of the user interface as well as managing user interaction. The **Presentation subsystem**, consists of the browser classes. The use cases associated with this subsystem are- Signup, Login, Logout, post message, view message, edit message, edit message and delete message.
2) **Business subsystem**: Also called the middle tier, logic tier, business logic or logic tier, this tier is pulled from the presentation tier. It controls application functionality by performing detailed processing. In business tier we further implement MVC (Model View Controller architecture).

2.1) **View:** A view is the visual representation of the system, that is pushed to the client in the presentation tier. It can also provide generic mechanisms to inform the controller of client input. A view produces an output presentation to the client taking into account changes in the model. The use cases associated in this subsystem are registration, login, user profile, change password, recover password, post message, edit message and delete message, edit user profile (See Appendix B for details regarding the use cases).

2.2) **Model:** A model advises its related view/views and controllers when there has been a change in its state. This notification permits views to update their presentation, and the controllers to change the accessible arrangement of orders. The model subsystem contains user model and message model which hold data related user account and messages respectively. The use cases associated in this subsystem are registration, login, user profile, change password, recover password, post message, edit message and delete message (See Appendix B for details regarding the use cases).

2.3) **Controller:** A controller can send commands to the model to redesign the model's state. It can also send commands to its related store and change. This subsystem bridges the gap between the application business logic and the boundary. Back end processing and bindings between external applications are provided by this subsystem. The use cases associated in this subsystem are like registration, login, user profile, change password, recover password, post message, edit message, delete message, logout, activate user, SQL injection and XSS (See Appendix B).

3) **Data subsystem:** Houses database servers where information is stored and retrieved. Data in this tier is kept independent of application servers or business logic. The **Database subsystem,** is responsible for the data storage of the application. It stores user's information, messages posted by user. The use cases associated in this subsystem are like registration, login, user profile, change password, recover password, post message, and edit message and delete message (See Appendix B).

2.3. Hardware and Software Mapping

In this section, we show the hardware configuration of the system, which node is responsible for which functionality. How is communication between nodes realized? Which services are realized using existing software components? How are these components encapsulated? Addressing hardware/software mapping issues often leads to the definition of additional subsystems dedicated to moving data from one node to another, dealing with concurrency, and reliability issues.

This part shows the 3-Tier deployment of the system, including the clients, the application server, and the database server.

1) **Clients**: the client indicates all the devices that are accessing the website through web browser like Chrome, IE, etc. It can be any platform like Windows, Linux, OS X, android, IOS. The client connect to the server through the Internet.

2) **Application Server**: the application server indicates the hardware which runs the website application. The OS platform is 'Windows XP' or higher. The website platform is WildFly8. The server application is implemented in Java, so it runs on Java EE JVM. The Application server connects the Database server by local network (100MB/s Ethernet).

3) **Database Server**: the database server indicates the hardware which runs the database instance. The platform is 'Windows XP' or higher, the database platform is MySQL5.

Detail:

1. **For Presentation subsystem (Desktop):**

   1.1.Hardware:

- Professor Intel(R) Core(TM) i3 2.4GHz or higher
- Memory 2GB or higher
- Internal Memory need: None

  1.2.Software:

- Windows XP or higher, Ubuntu 12.04, MAC OS X or main stream system.
- Chrome 48 or higher, IE 10 or higher, Firefox 40 or higher

2. **For MVC:**

   2.1. Hardware:

- Server: WildFly 8.2
- Professor: intel
- Memory: 4 GB RAM
- Hard drive: 1TB SATA

  2.2. Software:

- Windows XP or higher, Mac OS X
- Eclipse (Luna)
- JDK 1.8
- Boostrap
- J2EE 5 or higher

3. **For Database subsystem**

3.1. Hardware:

- Server: MySQL data base server 5.5
- Professor: Intel Quad-Core 1.80 GHz
- Memory: 12GB
- Hard drive: 1TB

3.2. Software:

- MySQL Work bench

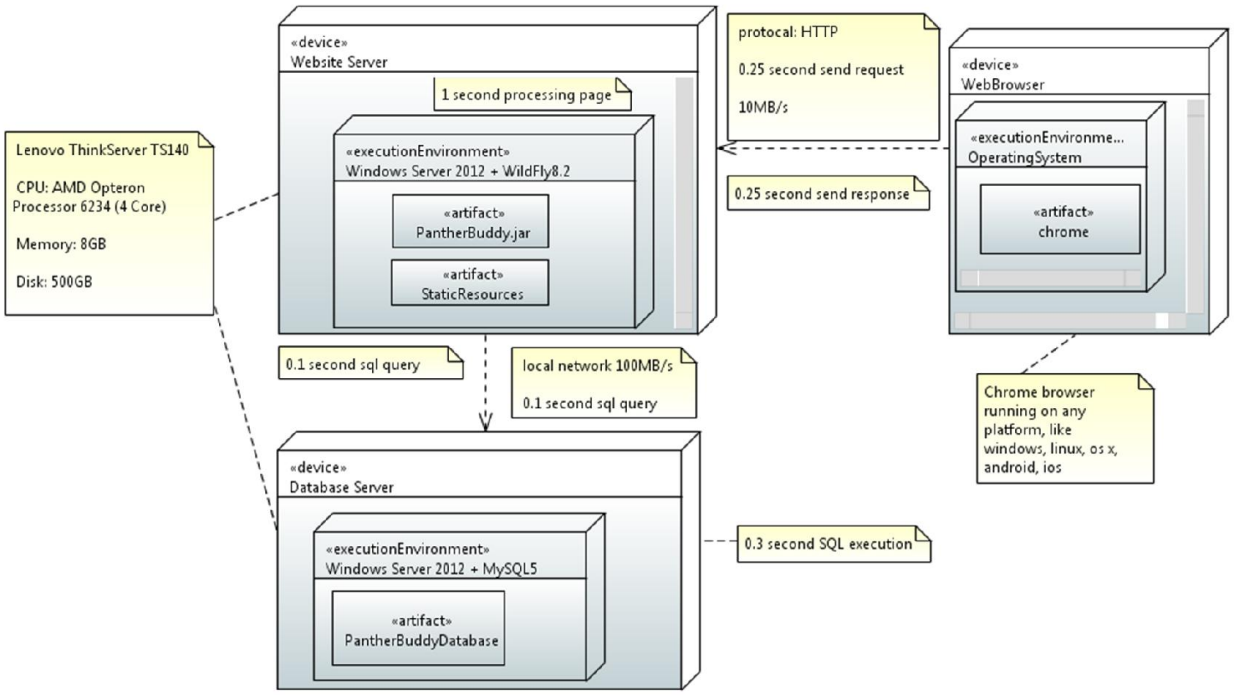


Figure: The figure shows the deployment diagram for the PantherBuddy system

2.4. Persistent Data Management

Persistent data represents a bottleneck in the system on many different fronts: most functionality in the system is concerned with creating or manipulating persistent data. For this reason, access to the data should be fast and reliable**.** In computing, a persistent data structure is a data structure that always preserves the previous version of itself when it is modified. In this section, we describe how we identify data that needs to be stored. This is similar to a data dictionary, i.e. the structure of the data used in the entity objects. We show how we used ER diagrams to identify data that needs to be stored.

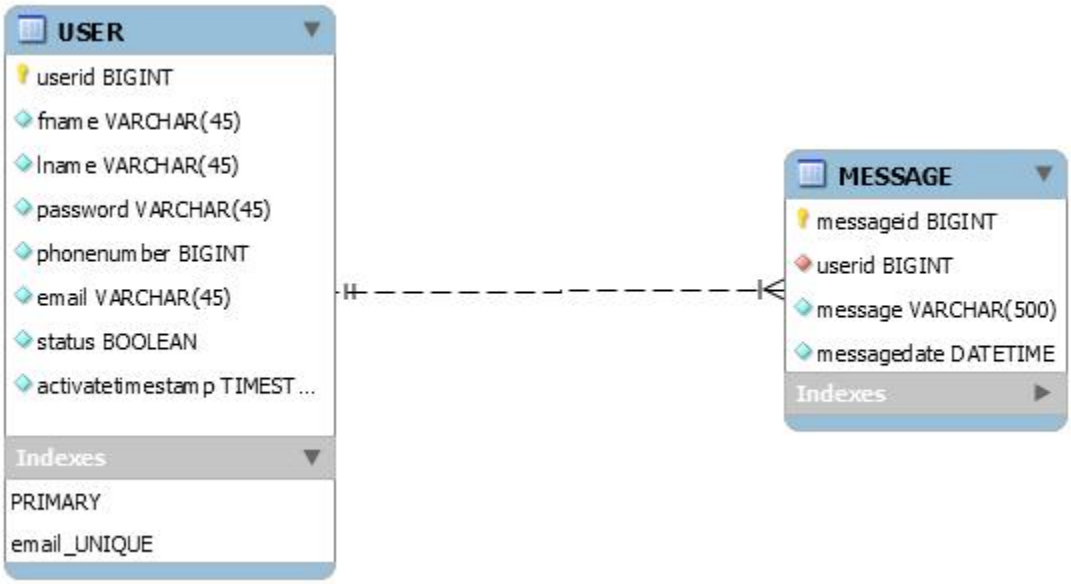


Figure: The ER diagram represents the entities in the database for the Panther buddy system

2.5. Security Management

Security management deals with development, documentation and implementation of policies and protocols. In this section, we describe security protocols that are used in the design.

- **Confidentiality:**

Confidentiality refers to limiting information access and disclosure to authorized users "the right people" and preventing access by or disclosure to unauthorized ones "the wrong people." Authentication methods like user-IDs and passwords, that uniquely identify data systems' users and control access to data systems' resources, underpin the goal of confidentiality.

- **Integrity**

Integrity refers to the trustworthiness of information resources. It includes the concept of "data integrity" namely, that data have not been changed inappropriately, whether by accident or deliberately malign activity. It also includes "origin" or "source integrity" that is, that the data actually came from the person or entity you think it did, rather than an imposter.

- **Availability**

Availability is the property that information is accessible and modifiable in a timely fashion by those authorized to do so.

- **Assurance**

  Assurance refers to how trust is provided and managed in computer systems.

- **Authenticity**

  Authenticity is the ability to determine that statements, policies, and permissions issued by persons or systems are genuine.

- **Anonymity**

  Anonymity is the property that certain records or transactions not to be attributable to any individual.

- **Economy of mechanism**

  Economy of mechanism principle stress simplicity in the design and implementation on security measures.

- **Fail-safe defaults**

  Fail-safe defaults states that the default configuration of a system should have a conservative protection scheme

- **Complete mediation**

  Complete mediation is that every access to resources must be checked for a compliance with a protection scheme.

- **Separation of Privileges**

  Separation of Privileges dictates that multiple conditions should be required to achieve access to restricted resources or have a program perform some action.

- **Least privilege**

  Least privilege principle states that user should operate with the bare minimum privileges necessary to function property.

- **Least common mechanism**

Least common mechanism principle states that allowing resources to be shared by more than should be minimized.

- **Psychological acceptability**

Psychological acceptability principle states that the user interfaces should be well designed and intuitive, and all security related settings should adhere to what an ordinary user might expect.

- **Avoiding SQL injection**

  For avoiding SQL injection, special symbols are not allowed in input text.
- **Avoiding XSS injection**

  For avoiding XSS injection, special symbols are not allowed in input text

3. Detailed Design

In this chapter, we will introduce our object design by presenting a minimal class diagram for the subsystems we will implement. We will identify and describe the design patterns we used to create our class diagram and the justification for using them. **Section 3.1.** Shows the minimal class diagrams of the subsystems that are used. **Section 3.2.** Shows the state machine for the overall system and the main control object in each major subsystem. **Section 3.3.** Shows the sequence diagrams, a refinement of sequence diagrams from the analysis model. **Section 3.4.** Explains the detailed class design, the purpose of each class. In **Section 3.4.1.** we explain the purpose of each class and refer to the appropriate class diagram in Appendix C. In **Section 3.4.2.** we describe OCL for the class invariants, pre and post conditions for the methods in the main control objects of the subsystems.

3.1. Minimal Class Diagram Overview

In this section, we will provide an overview of the minimal class diagram for the subsystems, along with a brief description of each class diagram, and identify the design patterns that are used.

3.1.1. Minimal Class Diagram

The PantherBuddy system is broken into three major subsystems, the presentation, business and Data source. There are different classes implemented in each subsystem. The minimal class diagram of our system is shown below.

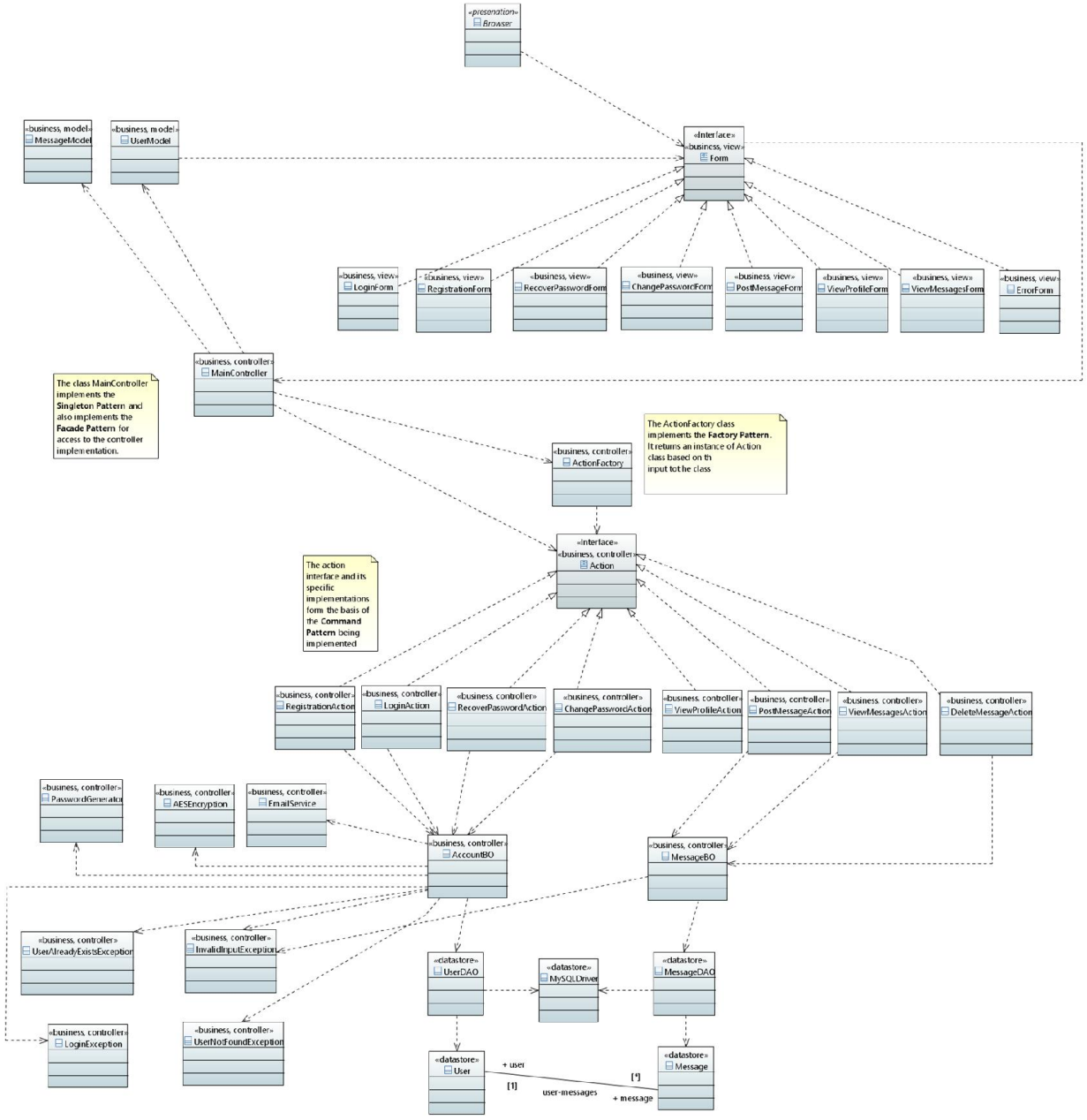


Figure: Minimal Class Diagram

3.1.2. Purpose of classes in brief

The PantherBuddy system is broken into three major subsystems, the presentation, business and Data source. There are different classes implemented in each subsystem. The purpose of different classes, implemented in of our system are described below.

3.1.2.1) Presentation tier

The package contains the implementation of the presentation tier of the 3-tier architecture followed for developing the PantherBuddy system. It occupies the top level and displays information related to services available of a website on the client machine. The brief description of classes are as follows

- **Browser:** The abstract class shows the interactions available for a user on the website being shown on his browser.

3.1.2.2) Business

The package contains the implementation of the business logic tier of the 3-tier architecture. It implements the MVC architecture within it. It contains the following packages

3.1.2.2.1) Model

The package contains the implementation of the Model component of the MVC architecture followed for developing the PantherBuddy system application tier. The classes in this package are:

- **MessageModel:** This is the model class to hold the data related to the messages for views showing or accessing messages
- **UserModel:** This is the model class to hold the data related to the user for views showing or accessing user data.

3.1.2.2.2) View

The package contains the implementation of the View component of the MVC architecture followed for developing the PantherBuddy system application tier. The classes in this package are:

- **ChangePasswordForm:** Represents the view for the change password use case.
- **ErrorForm:** Represents the view for showing the error which has occurred.
- **LoginForm:** Represents the view for the user login use case.
- **PostMessageForm:** Represents the view for the post message use case.

- **RecoverPasswordForm:** Represents the view for the recover password use case.
- **RegistrationForm:** Represents the view for the register user use case.
- **ViewMessagesForm:** Represents the view for the view messages use case.
- **ViewProfileForm:** Represents the view for the view profile use case.

3.1.2.2.3) Controller

The package contains the implementation of the Controller component of the MVC architecture followed for developing the PantherBuddy system application tier. The packages in this package are:

3.1.2.2.3.1) Control

- **MainController:** This is the entry point into the controller of the MVC architecture.

3.1.2.2.3.2) Action

- **ActionFactory:** The class follows the factory pattern and is used to get the specific implementation of the **Action** class.
- **Action:** The interface is implemented by every specific action class.
- **ChangePasswordAction:** Action class for Change Password.
- **DeleteMessageAction:** Action class for Delete Message.
- **LoginAction:** Action class for Login.
- **LogoutAction:** Action class for Logout.
- **PostMessageAction:** Action class for Post Message.
- **RecoverPasswordAction**: Action class for Recover Password.
- **RegistrationAction:** Action class for Registration.
- **ViewMessagesAction:** Represents the view for View Messages.
- **ViewProfileAction:** Represents the view for View Profile.

3.1.2.2.3.3) Account

Contains the business object classes pertaining to user account.

- **AccountBO:** The business object class that implement related to the account user.

3.1.2.2.3.4) Utility

The package contains the utility classes used by the system.

- **AESEncryption:** The class is used to encrypt and decrypt the user password.
- **EmailService:** The class is used to send the user his password.
- **PasswordGenerator:** The utility class is used to generate a random password for an user.

3.1.2.2.3.5) Exceptions

The package contains the business exceptions thrown by the system.

- **InvalidInputException:** Exception class for invalid input.
- **LoginException:** Exception class for login failure.
- **UserAlreadyExistsException:** Exception for user already exists.
- **UserNotFoundException:** Exception thrown when user with input data is not found.

3.1.2.2.3.6) Message

Contains the business object classes pertaining to the messages.

- **MessageBO:** The business object class that implements the logic for all operations related to the messages.

3.1.2.3) Data store

The package contains the implementation of the data tier of the 3-tier architecture followed for developing the PantherBuddy system. It occupies the bottom most level and includes the data persistence mechanisms (database servers, file shares, etc.) and the data access layer that encapsulates the persistence mechanisms and exposes the data. The data access layer provides an interface to the application tier that exposes methods of managing the stored data without exposing or creating dependencies on the data storage mechanisms. Avoiding

dependencies on the storage mechanisms allows for updates or changes without the application tier clients being affected by or even aware of the change. It contains the following packages.

3.1.2.3.1) DAO

The package contains the implementation of data access objects for user account and messages.

- **MessageDAO:** This class represents the data access object pertaining to message entity. It allows CRUD operations on message entity.
- **UserDAO:** This class represents the data access object pertaining to user entity. It allows CRUD operations on user entity.

3.1.2.3.2) Entity

The package contains the entities used by the Panther Buddy system. They are a reflection of the entities in the database.

- **Message:** The class represents the message entity.
- **User:** The class represents the user entity.

3.1.3. Design Patterns

The following are the design patterns that we implemented.

- **Command Pattern**: In object-oriented programming, the command pattern is a behavioral design pattern in which an object is used to represent and encapsulate all the information needed to call a method at a later time. This information includes the method name, the object that owns the method and values for the method parameters. The classes in the 'action' package representing actions implement the command pattern.

- **Facade pattern**: Facade pattern hides the complexities of the system and provides an interface to the client using which the client can access the system. This type of design pattern comes under structural pattern as this pattern adds an interface to the existing system to hide its complexities. This pattern involves a single class which provides simplified methods required by client and delegates calls to methods of existing system classes. The class 'MainController' implements the facade to the controller of the MVC

architecture being used.

- **Factory pattern:** Factory pattern is one of the most used design pattern in Java. This type of design pattern comes under creational pattern as this pattern provides one of the best ways to create an object. In Factory pattern, we create an object without exposing the creation logic to the client and refer to a newly created object using a common interface. The class 'ActionFactory' implements the factory pattern to get the action classes.

- **Singleton pattern**: singleton pattern is one of the simplest design patterns in Java. This type of design pattern comes under creational pattern as this pattern provides one of the best ways to create an object. This pattern involves a single class which is responsible to create an object while making sure that only single object gets created. This class provides a way to access its only object which can be accessed directly without need to instantiate the object of the class. The class 'MainController' implements the singleton pattern to the controller of the MVC architecture being used.

3.2. State machine diagrams

A state diagram is a type of diagram used in computer science and related fields to describe the behavior of systems. State diagrams require that the system described is composed of a finite number of states; sometimes, this is indeed the case, while at other times this is a reasonable abstraction. Many forms of state diagrams exist, which differ slightly and have different semantics. We show two level of state machine diagrams. High level diagrams shows the transitions between the states. In the low level diagram we considered main control object of each subsystem being decomposed and show its state transition.

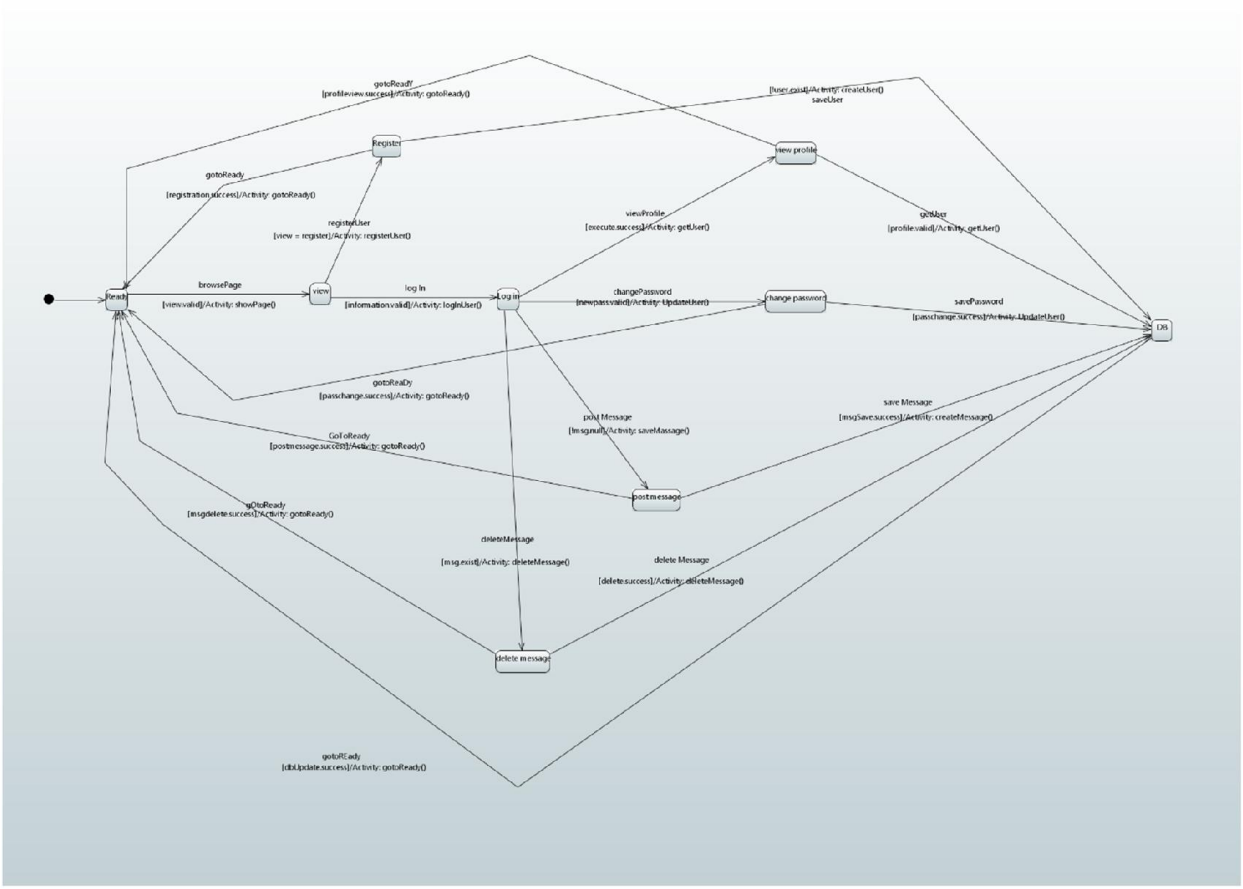


Figure: Shows the overall state machine for the flow in the system.

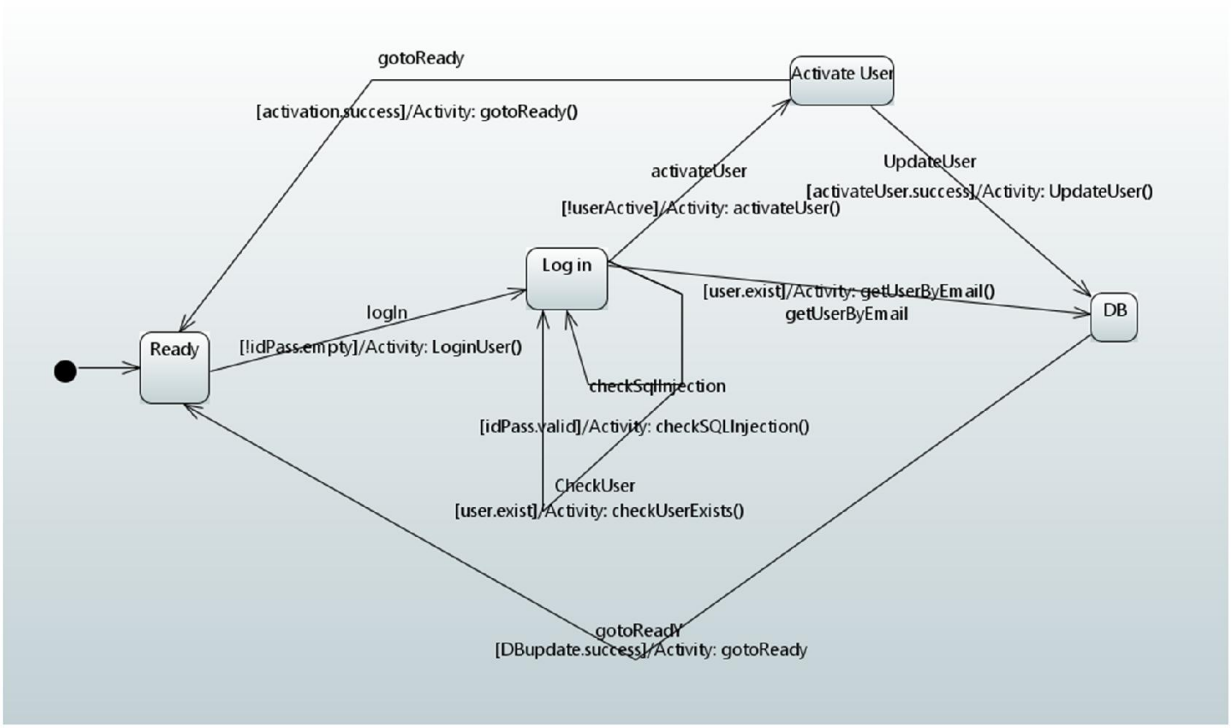


Figure: State machine for Login and Activate user use cases.

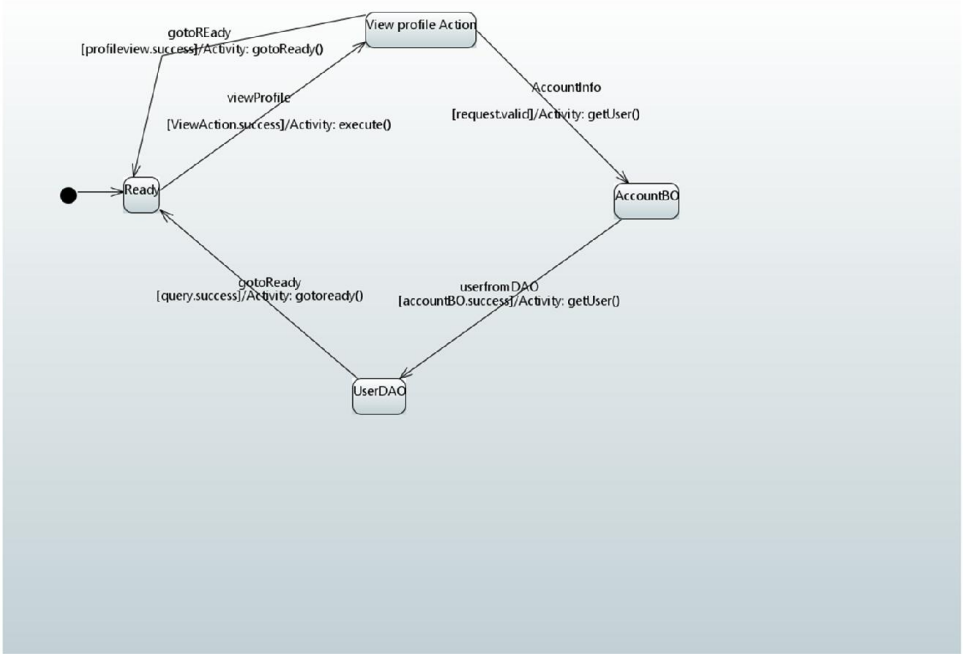


Figure: State machine for View Profile use case.

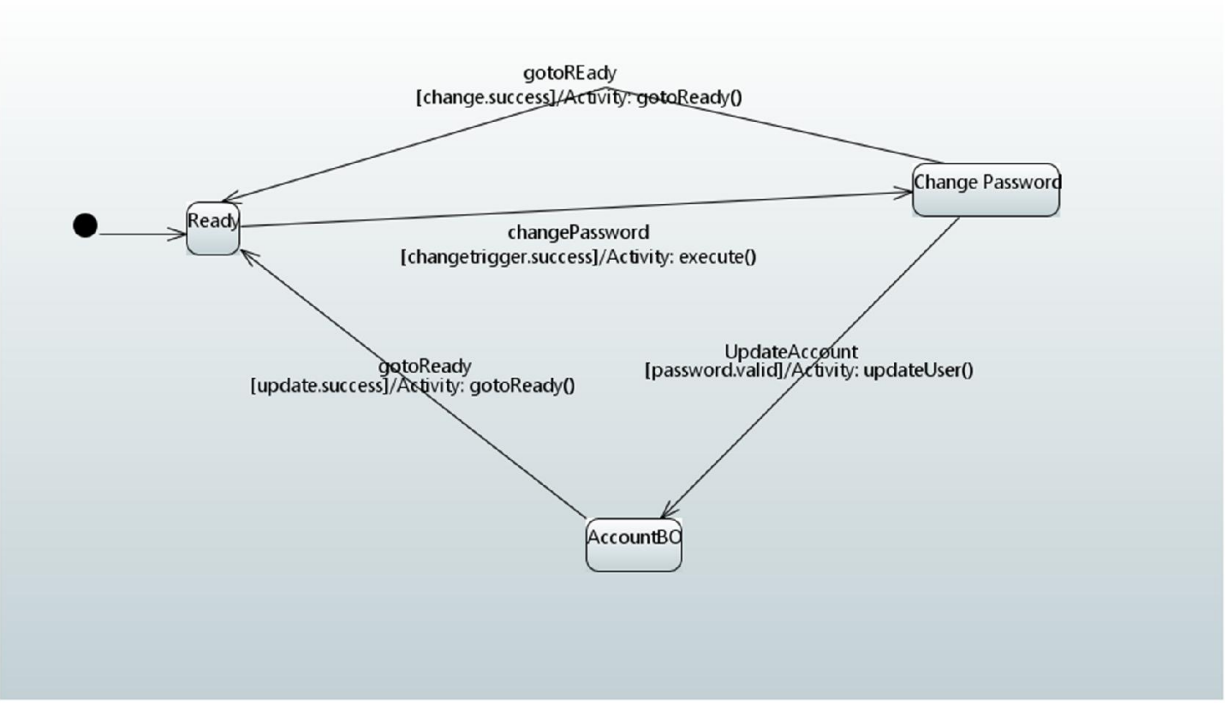


Figure: State machine for Change Password use case

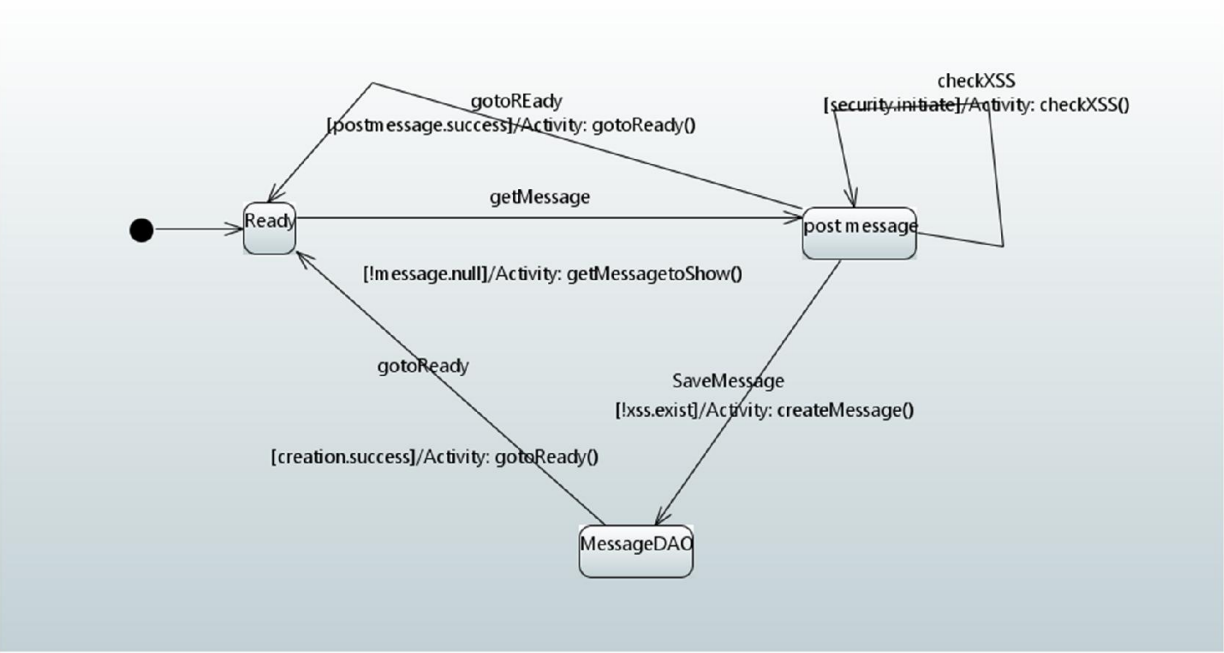


Figure: State machine for Post Message use case

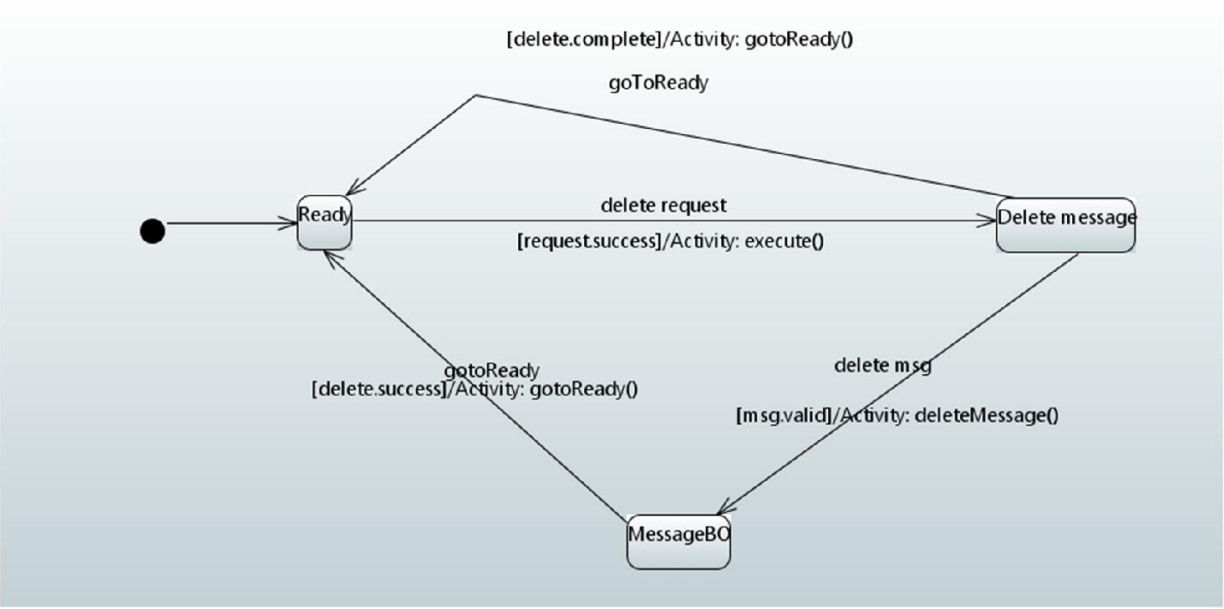


Figure: State Machine for Delete Message use case

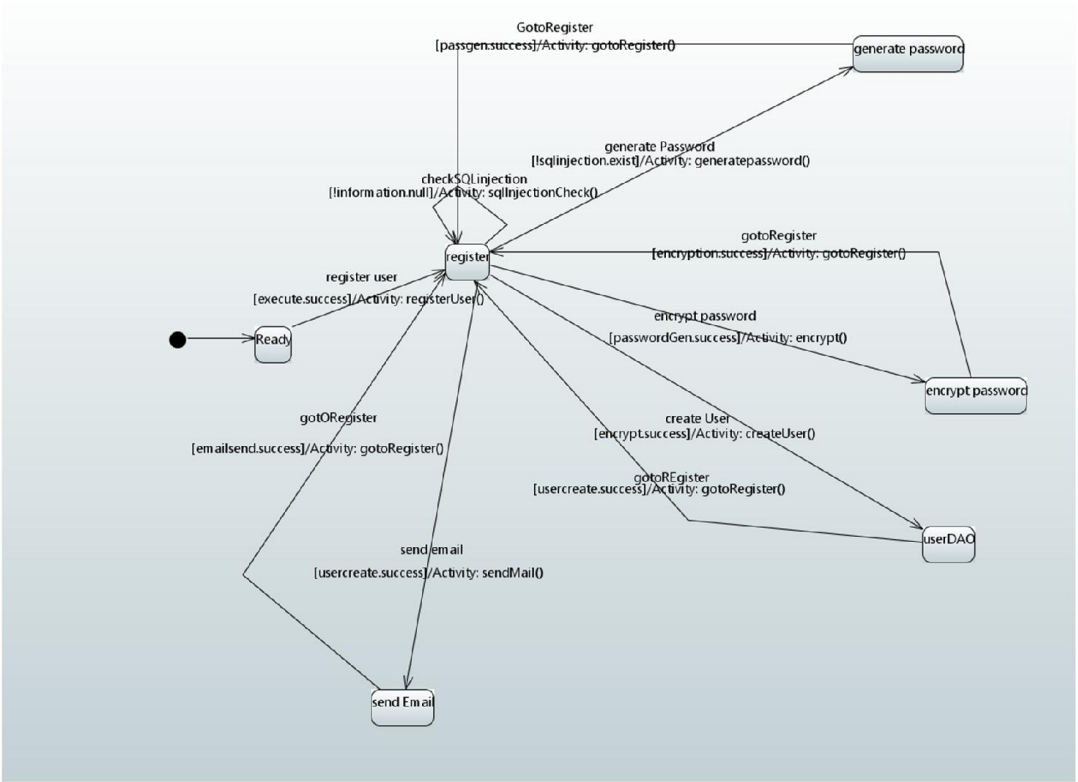


Figure: State Machine for Register User use case

3.3. Sequence diagrams

A **Sequence diagram** is an interaction diagram that shows how processes operate with one another and in what order. The following diagrams shows the how the processes operates in our '**Panther Buddy**' system.

- **Login, SQL injection and Active user**
  The diagram below shows the sequence diagram which covers the use case for login, activate user and SQL injection.

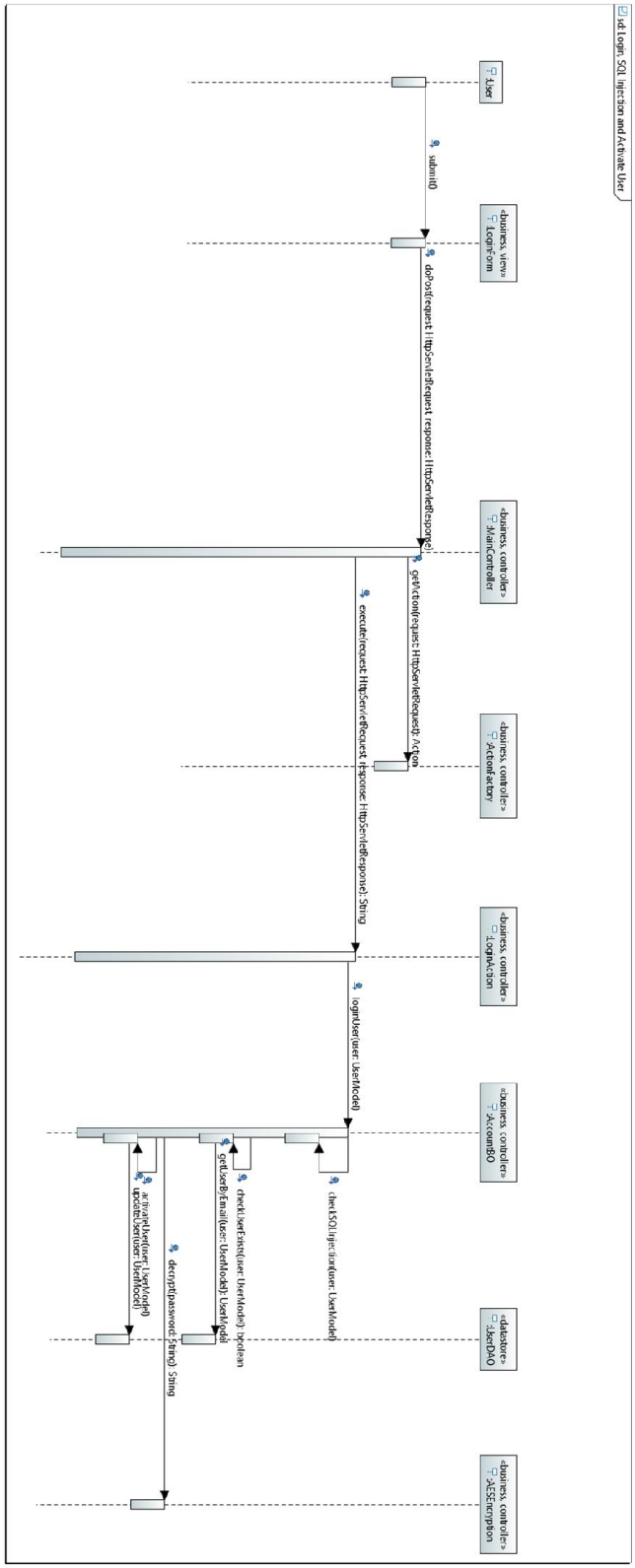


Figure 3.3.1 Login, SQL injection and Active user

- **Logout**

  The diagram below shows the sequence diagram which covers the use case for logout

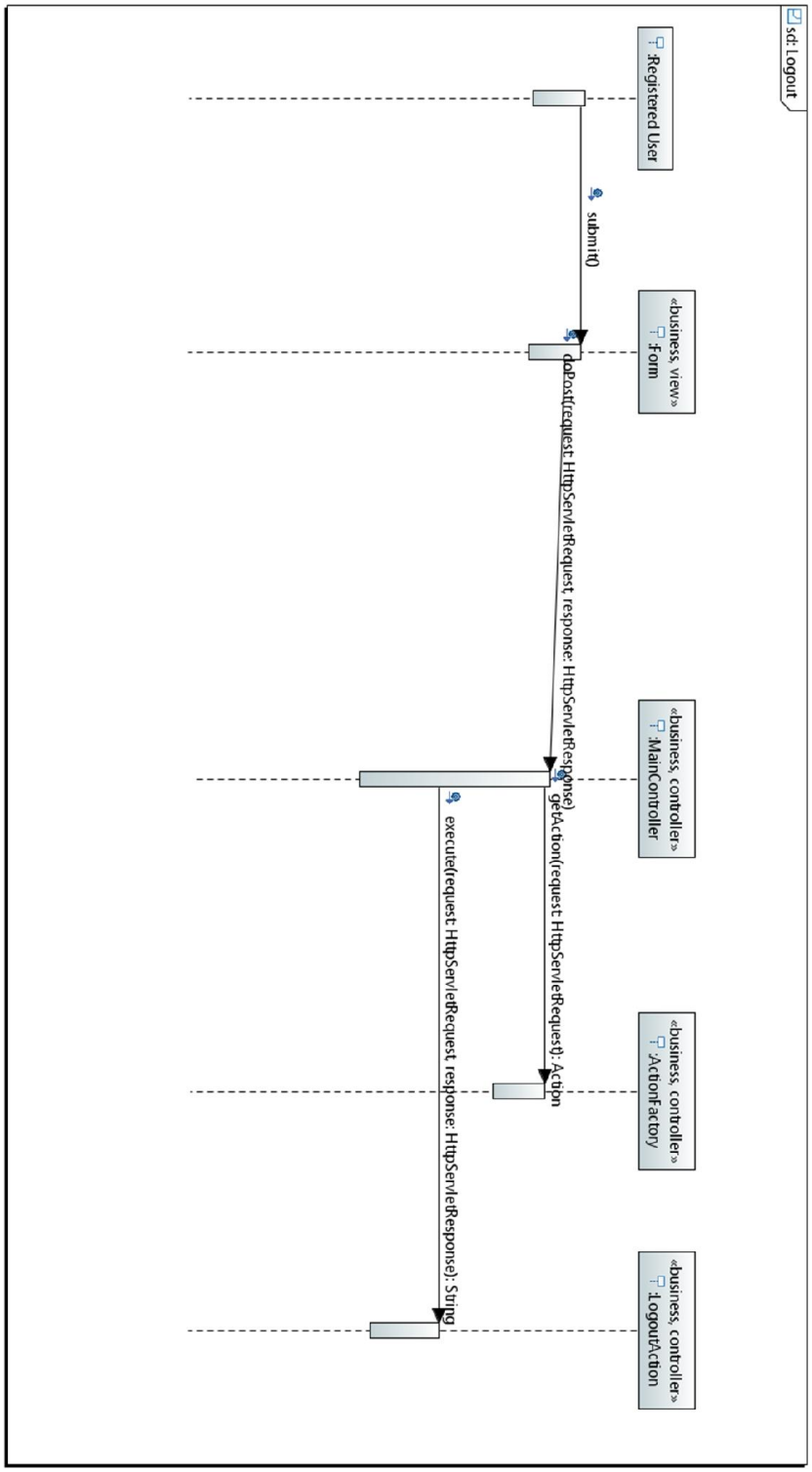


Figure 3.3.2 Logout

- **Registration**

  The diagram below shows the sequence diagram which covers the use case for registration

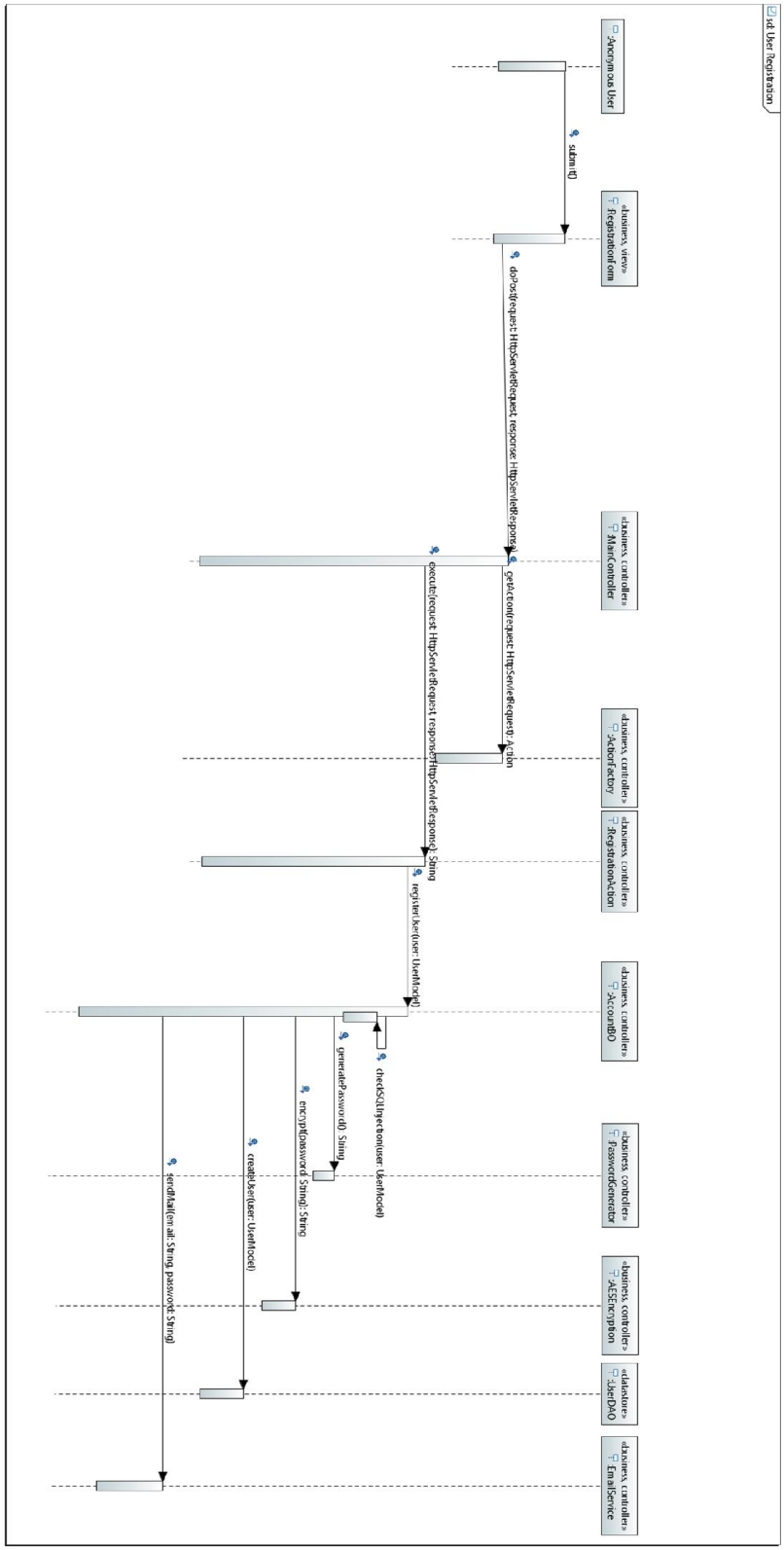


Figure 3.3.3 Registration

- **Change Password**

  The diagram below shows the sequence diagram which covers the use case for change password.

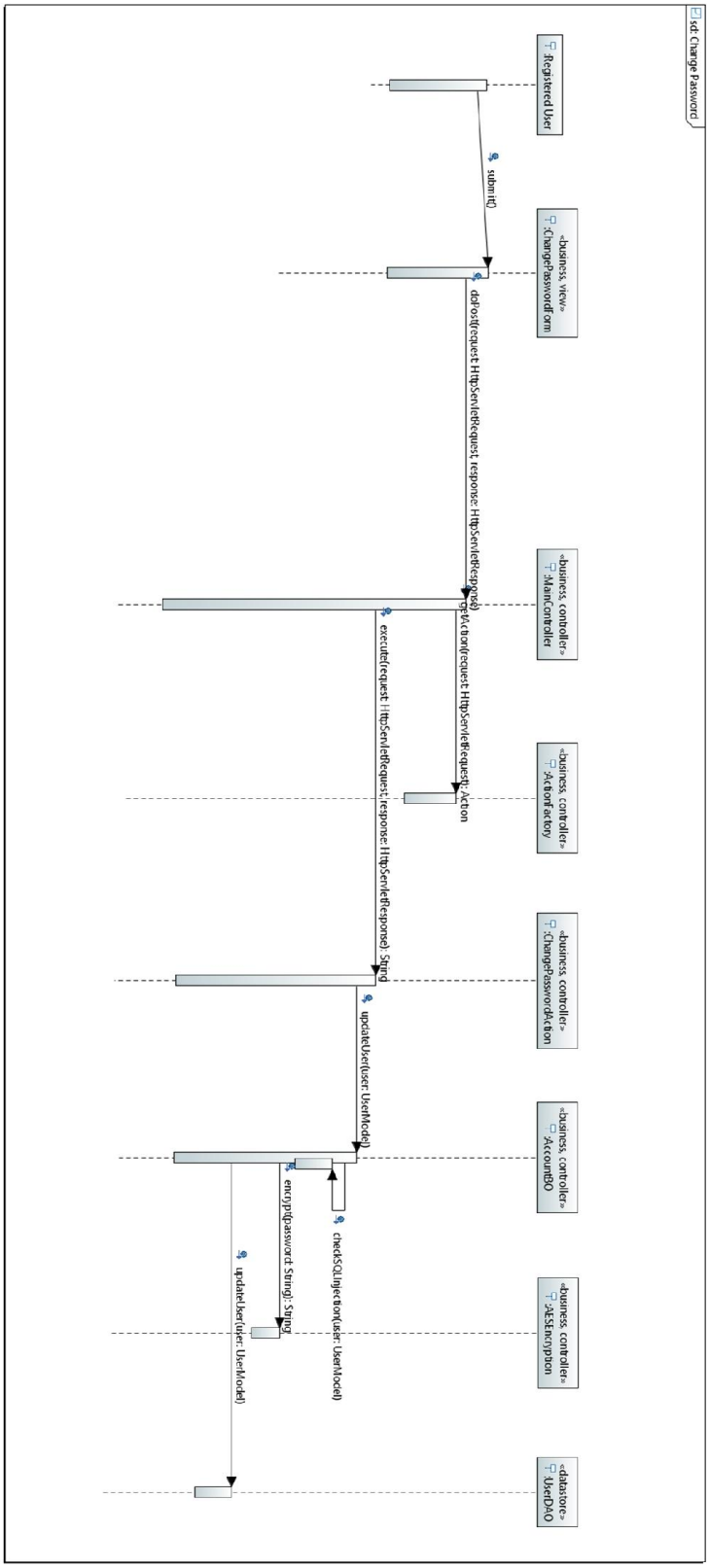


Figure 3.3.4 Change Password

- **View Message and Delete Message**

  The diagram below shows the sequence diagram which covers the use cases for view and delete message.

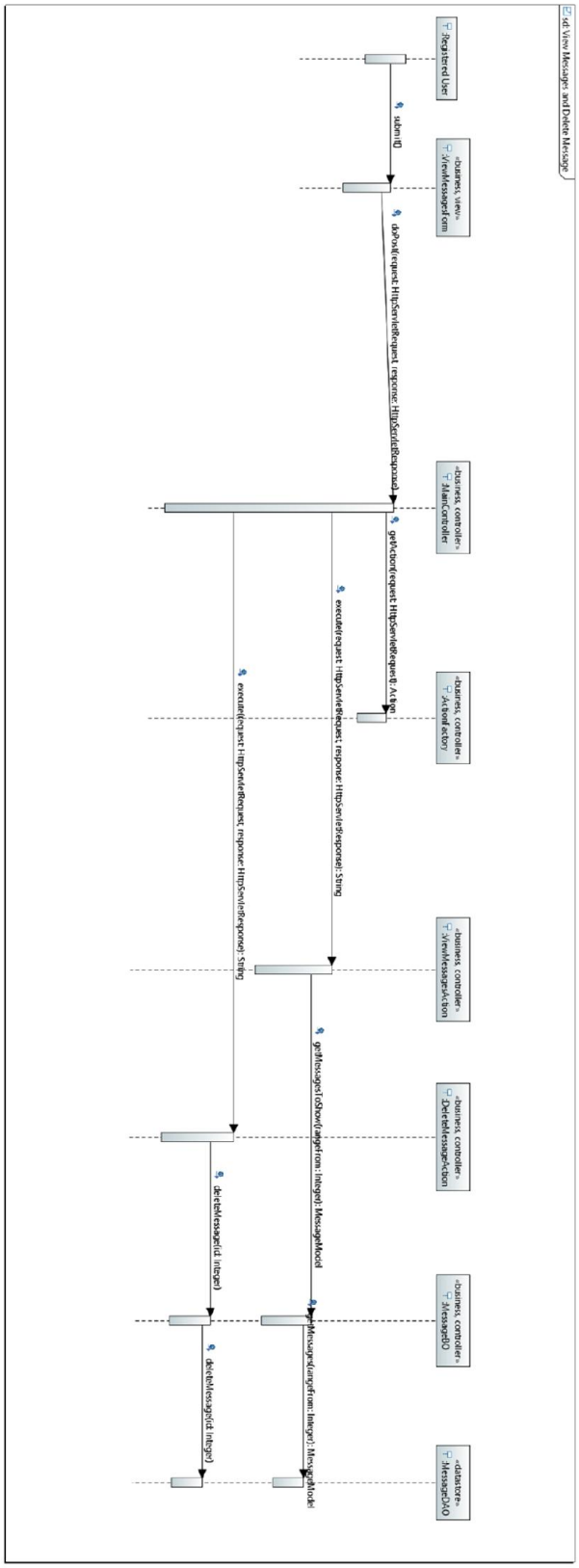


Figure 3.3.5 View Message and Delete Message

- **Post Message and XSS Filtering**

  The diagram below shows the sequence diagram which covers the use cases for post message and XSS filtering.

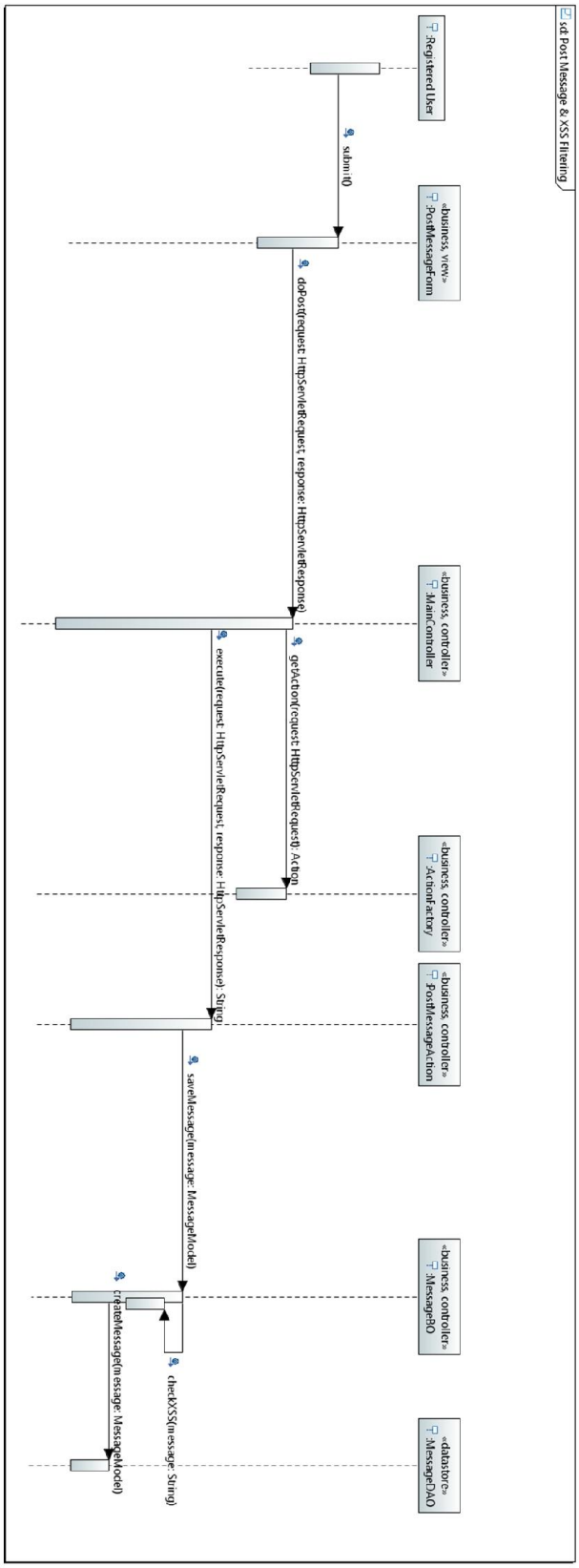


Figure 3.3.6 Post Message and XSS Filtering

- **Recover Password**

  The diagram below shows the sequence diagram which covers the use case for recover password.

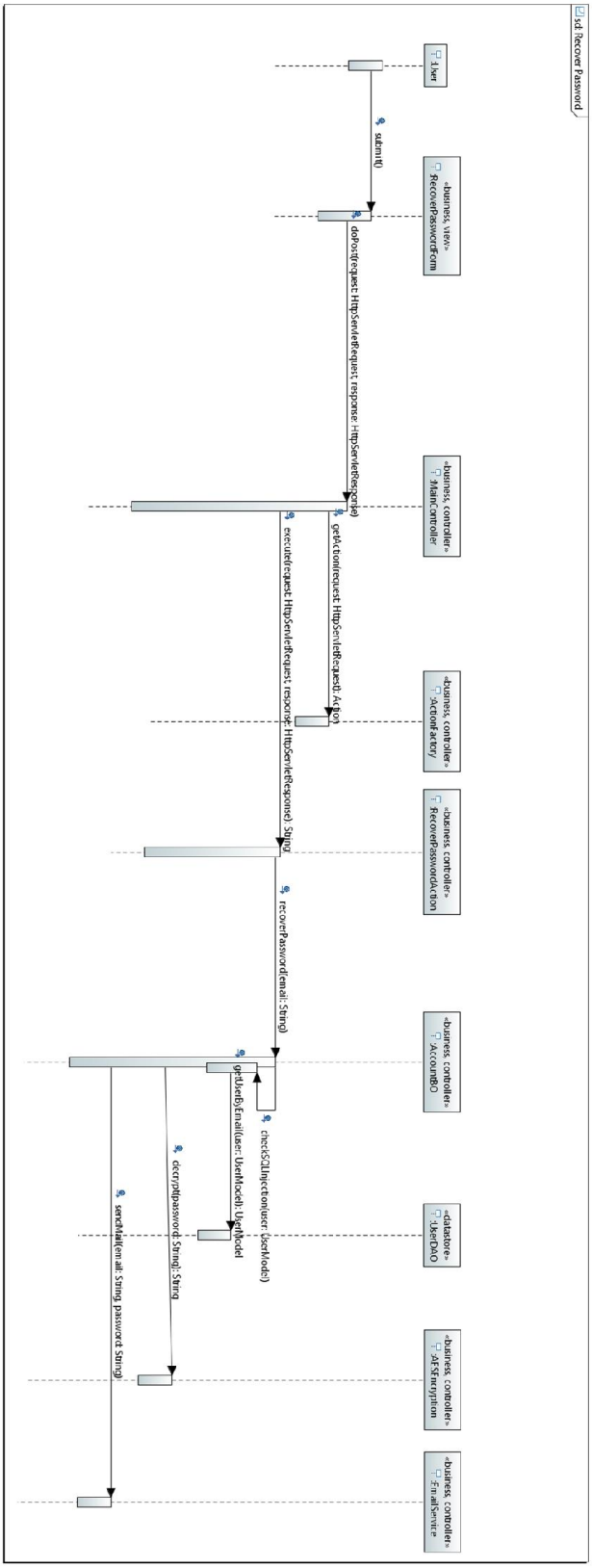


Figure 3.3.7 Recover Password

- **View Profile**

  The diagram below shows the sequence diagram which covers the use case for view profile

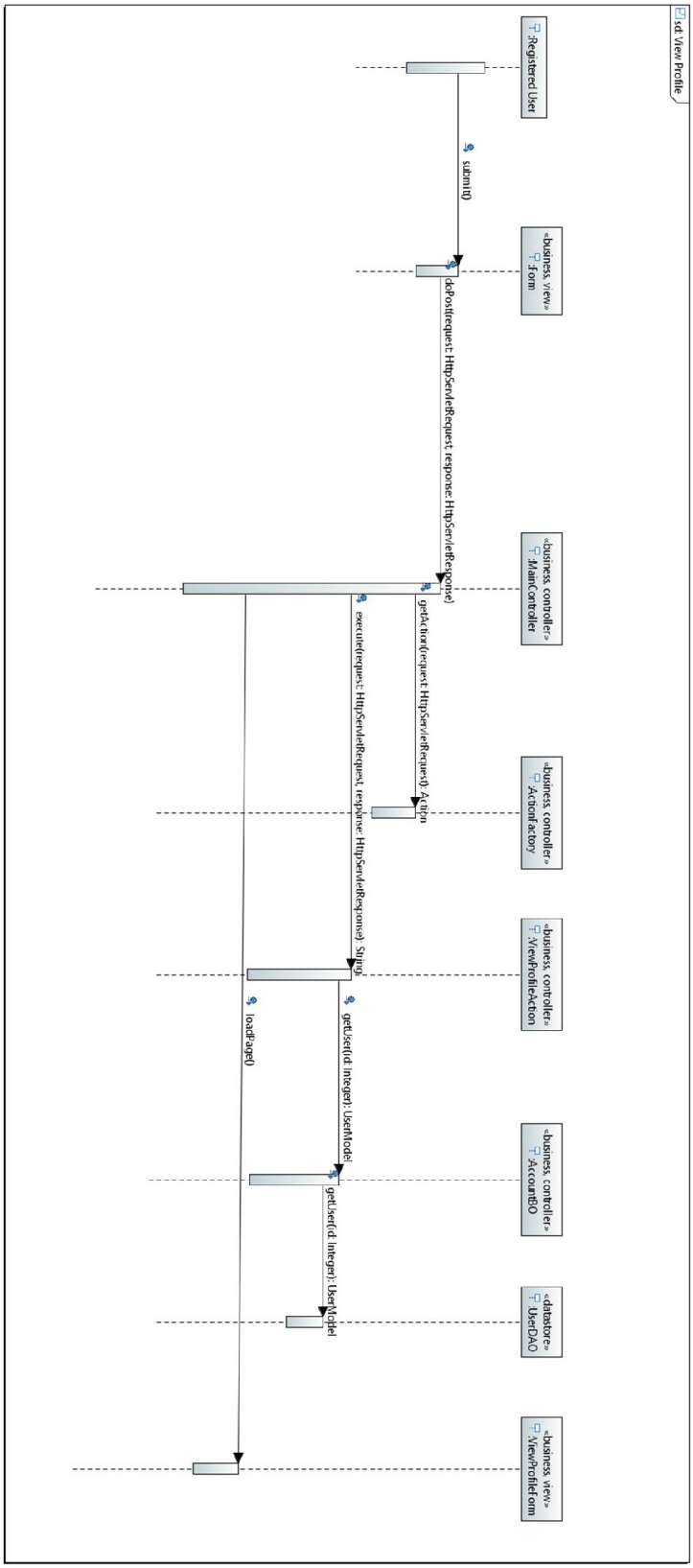


Figure 3.3.8 View Profile

3.4. Detailed Class Design

This section will briefly describe the purpose of each detailed class diagram for the system and a reference to the appropriate class diagram in Appendix C

3.4.1. Purpose of classes

The PantherBuddy system is broken into three major subsystems, the presentation, business and Data source. There are different classes implemented in each subsystem. The purpose of different classes implemented in our system are described below.

1. **Presentation (presentation tier)**

   The package contains the implementation of the presentation tier of the 3-tier architecture followed for developing the PantherBuddy system. It occupies the top level and displays information related to services available of a website on the client machine. The description of classes implemented in this package are as follows

   - **Browser:** This is an abstract class that represents the general interactions available to a user on his client to interact with the view being shown on his browser. It has two methods namely loadPage() and submit(). The method loadPage() is used to load the page and the method submit() is used to submit the data and actions from an user to the controller. (See Appendix C point 1)

2. **Business**

   The package contains the implementation of the Controller component of the MVC architecture followed for developing the PantherBuddy system application tier. (See Appendix C point 3). The brief description of packages and classes in this package are as follows

   2.1. Model

   The package contains the implementation of the Model component of the MVC architecture followed for developing the PantherBuddy system application tier. The view updates itself using this model. (See Appendix C point 4)

   - **MessageModel:** This is the model class to hold the data related to the messages for views

showing or accessing messages.

- **UserModel:** This is the model class to hold the data related to the user for views showing or accessing user data.

2.2. View

The package contains the implementation of the View component of the MVC architecture followed for developing the PantherBuddy system application tier. (See Appendix C point 5)

- **Form:** Is an interface that every view form should implement. It has two methods namely loadPage() and submit(). The method loadPage() is used to load the page and the method submit() is used to submit the data and actions from an user to the controller.
- **ChangePasswordForm:** Represents the view for the change password use case. This class implements the Form interface. The submit method implemented passes the password and new password attributes to the controller to change the users password.
- **ErrorForm:** Represents the view for showing the error which has occurred. This class implements Form interface.
- **LoginForm:** Represents the view for the user login use case. This class implements Form interface. The submit method passes the password, email id attributes to the controller process login for the user.
- **PostMessageForm:** Represents the view for the post message use case. This class implements Form interface. The submit method passes the message attribute to the controller to create and save a new message in the database.
- **RecoverPasswordForm:** Represents the view for the recover password use case. This class implements Form interface. The submit page pass the email id, last name, first name, phone number attributes to the controller to load the messages to view.
- **RegistrationForm:** Represents the view for the register user use case. This class implements Form interface. The submit method passes the email id, last name, first name, phone number attributes to the controller to register an user.
- **ViewMessagesForm:** Represents the view for the view messages use case. This class implements Form interface. The submit method passes the rangeFrom attribute to the controller to load the messages to view.

- **ViewProfileForm:** Represents the view for the view profile use case. This class implements Form interface. The submit method passes the id attribute representing the id of a user in the system to the controller to get the user details for the user.

2.3. Controller

The package contains the implementation of the Controller component of the MVC architecture followed for developing the PantherBuddy system application tier. (See Appendix C point 6). The packages and classes under it are:

2.3.1. Control

This package contains the entry point to the controller. (See Appendix C point 7)

- **MainController:** This is the entry point into the controller of the MVC architecture. It follows singleton pattern. All communications to and from the controller of MVC architecture is handled by this class.

2.3.2. Action

The package contains all the actions that can be executed in the system. (See Appendix C point 8)

- **ActionFactory:** The class follows the factory pattern and is used to get the specific implementation of the Action class.
- **Action:** Action is the interface every class need to implement the action interface. Action interface contains single method called execute. Any action class implementing this interface should put logic for particular action in this method.
- **ChangePasswordAction:** Action class for Change Password. This class implements the Action interface. This class contains a single method execute that encapsulates the logic for the change password action to be performed.
- **DeleteMessageAction:** Action class for Delete Message. This class implements the Action interface. This class contains a single method execute that encapsulates the logic for the delete message action to be performed.
- **LoginAction:** Action class for Login. This class implements the Action interface. This class contains a single method execute that encapsulates the logic for the login action to be performed.

- **LogoutAction:** Action class for Logout. This class implements the Action interface. This class contains a single method execute that encapsulates the logic for the logout action to be performed.
- **PostMessageAction:** Action class for Post Message. This class implements the Action interface. This class contains a single method execute that encapsulates the logic for the post message action to be performed.
- **RecoverPasswordAction**: Action class for Recover Password. This class implements the Action interface. This class contains a single method execute that encapsulates the logic for the recover password action to be performed.
- **RegistrationAction:** Action class for Registration. This class implements the Action interface. This class contains a single method execute that encapsulates the logic for the registration action to be performed.
- **ViewMessagesAction:** Represents the Action class for View Messages. This class implements the Action interface. This class contains a single method execute that encapsulates the logic for the view messages action to be performed.
- **ViewProfileAction:** Represents the Action class for View Profile. This class implements the Action interface. This class contains a single method execute that encapsulates the logic for the view profile action to be performed.

2.3.3. Account

Contains the business object classes pertaining to user account. (See Appendix C point 9)

- **AccountBO:** The business object class that implements methods related to the user account.

2.3.4. Utility

The package contains the utility classes used by the system. (See Appendix C point 11)

- **AESEncryption:** The class is used to encrypt and decrypt the user password. It uses AES 128bit algorithm using a key value.
- **EmailService:** The class is used to send the user his password.
- **PasswordGenerator:** The utility class is used to generate a random password for a user.

2.3.5. Exceptions

The package contains the business exceptions thrown by the system. (See Appendix C point 12)

- **InvalidInputException:** Exception class for invalid input. Thrown when there is an exception due to invalid input.
- **LoginException:** Exception class for login failure. Thrown when invalid login occurs due to password or email or both don't represent a valid entity in the system.
- **UserAlreadyExistsException:** Exception for user already exists. Thrown when user trying to give constraints same as an existing user.
- **UserNotFoundException:** Exception thrown when user with input data is not found. Thrown when there is an invalid user being searched.

2.3.6. Message

Contains the business object classes pertaining to the messages. (See Appendix C point 10)

- **MessageBO:** The business object class that implements the logic for all operations related to the messages.

3. Datastore

The package contains the implementation of the data tier of the 3-tier architecture followed for developing the PantherBuddy system. It occupies the bottom most level and includes the data persistence mechanisms (database servers, file shares, etc.) and the data access layer that encapsulates the persistence mechanisms and exposes the data. The data access layer provides an interface to the application tier that exposes methods of managing the stored data without exposing or creating dependencies on the data storage mechanisms. Avoiding dependencies on the storage mechanisms allows for updates or changes without the application tier clients being affected by or even aware of the change. (See Appendix C point 13). The brief description of packages and classes are as follows

3.1. DAO

The package contains the implementation of data access objects for user account and messages. (See Appendix C point 14)

- **MessageDAO:** This class represents the data access object pertaining to message entity. It allows CRUD operations on message entity.
- **UserDAO:** This class represents the data access object pertaining to user entity. It allows CRUD operations on user entity.

3.2. Entity

The package contains the entities used by the Panther Buddy system, they are a reflection of the entities in the database. (See Appendix C point 15)

- **Message:** The class represents the message entity.
- **User:** The class represents the user entity.

3.4.2. Object Constraint Language (OCL)

In this section, we introduce the OCL constraints for the main controller in the main subsystem.

The Object Constraint Language (OCL) is a declarative language for describing rules that apply to Unified modeling language (UML) models. OCL supplements UML by providing expressions that have neither the ambiguities of natural language nor the inherent difficulty of using complex mathematics. OCL is also a navigation language for graph-based models. OCL is a language that allows constraints to be formally specified on single model elements (e.g., attributes, operations, classes) or groups of model elements (e.g., associations and participating classes).

The main control object in the application tier are:

1. **AccountBO**

   1.1 Method loginUser

   Context: loginUser(user :UserModel)

   Pre-condition:

   self.user.username<>null && self.user.password<>null

   Post-condition:

not execThrown('LoginException')

1.2 Method registerUser

Context: registerUser(user : UserModel)

Pre-condition:

self.user.username<> null && self.user.password<> null && self.user.fname <> null && self.user.lname <> null

Post-condition:

not execThrown('UserAlreadyExistsException')

1.3 Method getUser

Context: getUser(id : Integer)

Pre-condition:

self.id<>null

Post-condition:

return userModel <> null

1.4 Method recoverPassword

Context: recoverPassword (email : String)

Pre-condition:

self.email<>null

Post-condition:

self.user = null implies execThrown('UserNotFoundException')

1.5 Method updateUser

Context: updateuser(user : UserModel)

Pre-condition:

self.user <> null

Post-condition:

not execThrown('IllegalArgumentException')

1.6 Method checkUserExists

Context: checkUserExists(user : UserModel)

Pre-condition:

self.user <> null

Post-condition:

return <> null

1.7 Method checkSQLInjection

Context: checkSQLInjection(user : UserModel)

Pre-condition:

self.user.username<>null && self.user.password<>null

Post-condition:

not execThrown('InvalidInputException')

1.8 Method activateUser

Context: activateUser(user : UserModel)

Pre-condition:

self.user.status<>true

Post-condition:

self.user.status==true

2. **MessageBO**

2.1 Method checkXSS

Context: checkXSS (message:String)

Pre-condition:

self.message<>null

Post-condition:

not execThrown('IllegalArgumentException')

2.2 Method deleteMessage

Context: deleteMessage (id :Integer)

Pre-condition:

self.id<>null

Post-condition:

not execThrown('IllegalArgumentException')

2.3 Method getMessagesToShow

Context: getMessagesToShow (rangeFrom :Integer)

Pre-condition:

self.rangeFrom>=0

Post-condition:

return = MessageModel[1..*]<>null

2.4 Method saveMessage

Context: saveMessage (message :MessageModel)

Pre-condition:

self.message.messageText<>null

Post-condition:

not execThrown('IllegalArgumentException')

The main control objects in the data tier are:

1. **UserDAO**

1.1 Method createUser

Context: createUser (user:UserModel)

Pre-condition:

self.user<>null

Post-condition:

not execThrown('IllegalArgumentException')

1.2 Method getUser

Context: getUser (id:Integer)

Pre-condition:

self.user.id <> null

Post-condition:

return = UserModel <> null

1.3 Method updateUser

Context: getUserByEmail (user:UserModel)

Pre-condition:

self.user<>null

Post-condition:

not execThrown('IllegalArgumentException')

2. MessageDAO

2.1 Method createMessage

Context: createMessage (message :MessageModel)

Pre-condition:

self.message<>null

Post-condition:

not execThrown('IllegalArgumentException')

2.2 Method deleteMessage

Context: deleteMessage (id :Integer)

Pre-condition:

self.id<>null

Post-condition:

not execThrown('IllegalArgumentException')

2.3 Method getMessages

Context: getMessages (rangeFrom :Integer)

Pre-condition:

self.rangeFrom>=0

Post-condition:

return = MessageModel[1..*]<>null

4. Glossary

SRD	Stands for System Reference Document
MVC	Stands for Model View Controller, which is a software architecture pattern
USDP	Stands for Unified Software Development Process Model, which is a software development process
ER- diagram	Stands for entity relationship diagram, showing the entities in the database and the relation between them.
XSS	Short for Cross Site Scripting, which is a form of attack on the website.
HTML	Stands for Hyper Text Markup Language, a web technology used to build websites.
JSP	Stands for Java Server Pages, which is a server side web technology.
CRUD	Stands for Create, Read, Update and Delete. These are basic atomic operations defined for a database.
OCL	The Object Constraint Language (OCL) is a declarative language for describing rules that apply to Unified Modeling Language (UML) models developed at IBM and now part of the UML standard.
AES	The Advanced Encryption Standard (AES), also known as Rijndael[4][5] (its original name), is a specification for the encryption of electronic data established by the U.S. National Institute of Standards and Technology (NIST) in 2001.[6]
J2EE	J2EE is a platform-independent, Java-centric environment from Sun for developing, building and deploying Web-based enterprise applications online. The J2EE platform consists of a set of services, APIs, and protocols that provide the functionality for developing multitiered, Web-based applications.
JDK	The Java Development Kit (JDK) is an implementation of either one of the Java SE, Java EE or Java ME platforms[1] released by Oracle Corporation in the form of a binary product aimed at Java developers on Solaris,Linux, Mac OS X or Windows.
CSS	Cascading Style Sheets (CSS) is a style sheet language used for describing the presentation of a document written in a markup language.
Bootsrap	Bootstrap, a sleek, intuitive, and powerful mobile first front-end framework for faster and easier web development.
MySQL	MySQL is an open-source relational database management system (RDBMS)

5. References

Bryan Basham, K. S. (2004). *Head First Servlets and JSP* . 08: O'Reilly Media.

Foundation, E. (n.d.). *Papyrus*. Retrieved from Eclipse: https://eclipse.org/papyrus/

Point, T. (2015). . Retrieved from Design Patterns: http://www.tutorialspoint.com/design_pattern/command_pattern.htm

Point, T. (2015). *factory pattern*. Retrieved from tutorialsPoint: http://www.tutorialspoint.com/design_pattern/factory_pattern.htm

Points, T. (2015). *strategy pattern*. Retrieved from Tutorial Points: http://www.tutorialspoint.com/design_pattern/strategy_pattern.htm

Scholtz, B. (2010, 08 22). *Using the MVC design pattern with JSP/Servlet*. Retrieved from The BalusC Code: http://balusc.omnifaces.org/2010/08/using-mvc-design-pattern-with.html

Thiemann, D. P. (2009). *Software Engineering.* Retrieved from http://proglang.informatik.uni-freiburg.de/: http://proglang.informatik.uni-freiburg.de/teaching/swt/2009/v11-ocl.en.pdf

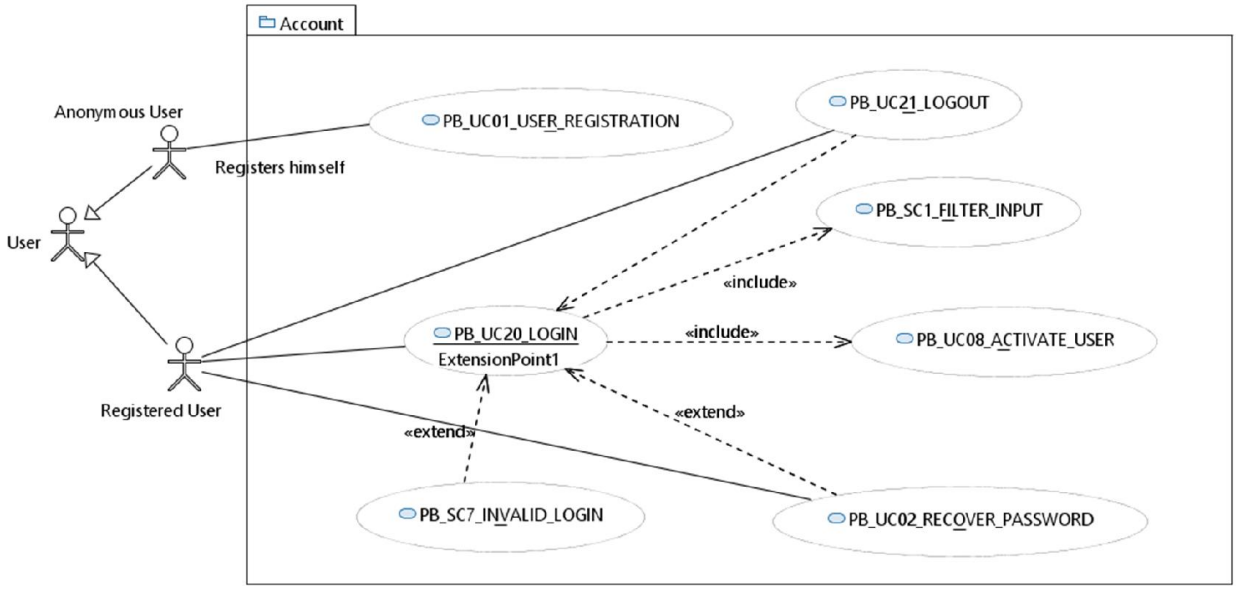
visual-paradigm. (n.d.). *Deployment diagram*. Retrieved from visual-paradigm: http://www.visual-paradigm.com/VPGallery/diagrams/Deployment.html

6. Appendix

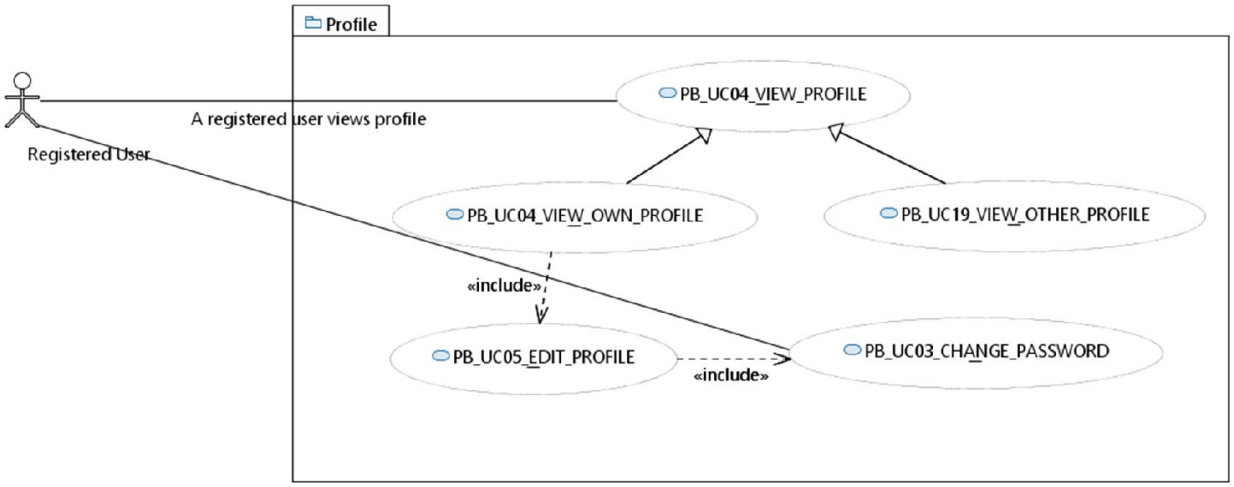
6.1. Appendix A – Use case Diagram

This appendix contains the use case diagrams for the use cases being implemented.

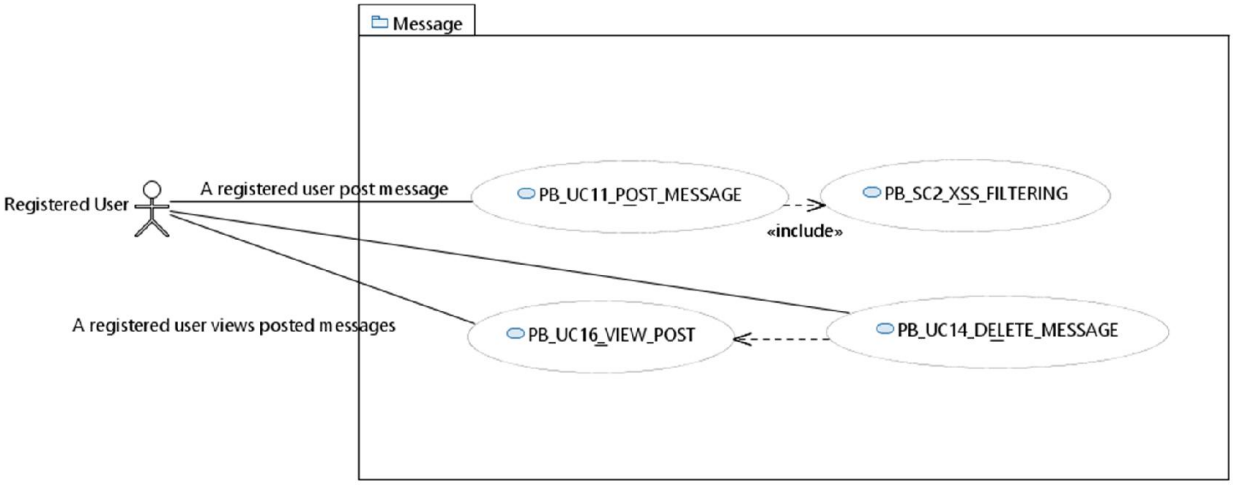
1. Registration, Login, Logout, Recover Password and SQL Injection Security



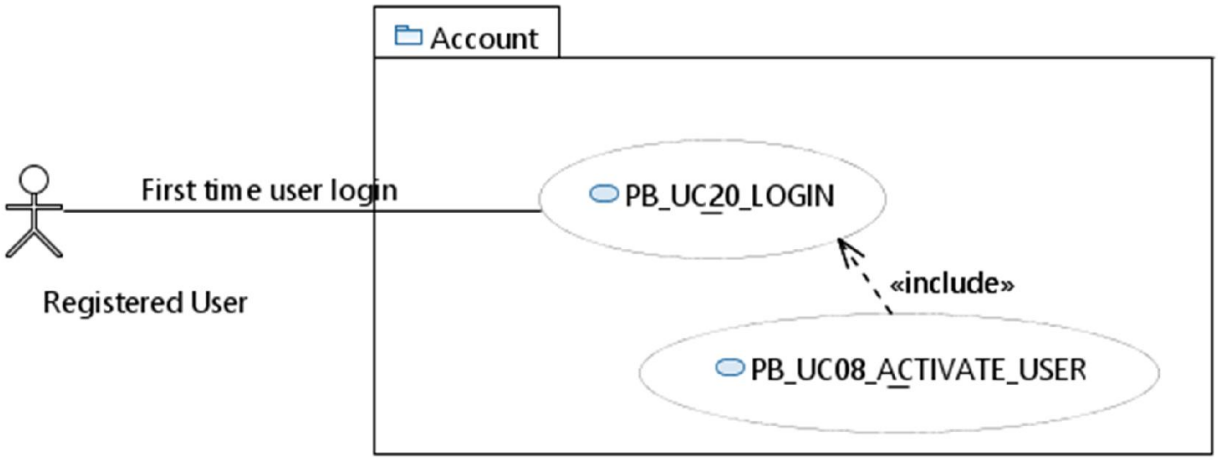
2. View Profile and Change User Password



3. Post, view and delete message and XSS Security



4. Activate User



6.2. Appendix B – Use Cases with Non-functional Requirements

1) USER REGISTRATION

Use case ID: PB_UC01_USER_REGISTRATION

Use case level: System level (End-to-End)

Details:

Actors: A non-registered User.

Pre-conditions: The non-registered User has the registration page for Panther Buddy open in a web browser.

Description: The User registers himself on "Panther Buddy". The user will be asked to enter his or her credentials.

Trigger: The non-registered User clicks on the “Register” button.

The system responds by

1. The user enters his or her first name in the first name box.
2. The user enters his or her last name in the last name box.
3. The user enters his or her email address in the email address box.
4. The user enters his or her phone number.
5. The user clicks the submit button to submit the data.
6. The user’s data is saved in the database along with a generated password. His status is marked as inactive and a default date value (Sun, 2 Dec 292269055 BC 16:47:04 +0000) is put for his activation date.
7. A default password is sent to the user via email.
8. Once the user checks his or her email, he or she is directed to the login page.

Post condition: The user is marked as an inactive registered user in database.

Alternative courses of action:

At any time during step 1, the user can click the cancel button to return to the 'Panther Buddy' login page.

Extensions:

None

Exceptions:

- If, at step 3, the user enters an email address that is already registered in the system. The user is asked to re-enter a new email address or to login instead.

Related use cases:

- PB_UC2_LOGIN, PB_UC3_ACTIVATE_USER, PB_UC4_INVALID_LOGIN,

- PB_UC5_RECOVER_PASSWORD.

Decision support:

Frequency: Moderate - Performed every time a user wants to register on 'Panther Buddy' (30 times per day).

Criticality: Very Critical - If the user wants to use the 'Panther Buddy' site he needs to register. We also need to verify the user's data input by him during registration for completeness and malicious activity.

Risk: High

Constraints:

Usability

- No previous training needed.

Mean time to failure

- One failure for every two months of operation is acceptable.

Performance

- The user data will be saved in 2 seconds.

Supportability

- The software will support all browsers that support HTML 5.

Implementation

- Client requests the backend implementation to be done in Java Server Faces (Wildfly 8.2.0 and the Mojarra 2.2.9 Library). We will use the Bootstrap framework for the frontend. For the data tier, we will use MySQL database 5.6.

Modification History

Owner: Allan Shaji Manamel

Initiation date: 09/07/2015

Date last modified: 09/14/2015

Example Scenario:

A new user wants to register.

Actors: Allan Shaji Manamel

Pre-conditions: Allan Shaji Manamel has the user registration page for panther buddy open in a web browser.

Description: Allan Shaji Manamel need to registers himself on "Panther Buddy". The Allan Shaji Manamel will be asked to enter his details.

Trigger: The non-registered User clicks on the “Register” button.

The system responds by

1. First name: Allan
2. Last name: Manamel
3. Email address: amana010@fiu.edu
4. Phone number: 987 654 2134
5. Allan Shaji Manamel data is saved in the database along with a generated password (D43sx21). His status is marked as inactive and a default date value (Sun, 2 Dec 292269055 BC 16:47:04 +0000) is put for his activation date.
6. Allan Manamel sent his password (D43sx21) by email.
7. The user is then redirected to the login page.

Post-conditions: Allan Shaji Manamel is marked as an inactive registered user in database.

2) LOGIN

Use case ID: PB_UC20_LOGIN

Use case level: System level (End-to-End)

Details:

Actors: Any user.

Pre-conditions: The user has the login page open.

Description: The user logins into the system using his email and password.

Trigger: The user clicks on the login button.

The system responds by

1) User enters his email into the email field
2) The user enters his password in the password field
3) The user clicks the login button to submit his data.

Post condition: The user is redirected to the Panther Buddy home page.

Alternative Courses of Action:

If the user decides not to proceed with login, he or she can just press the cancel button.

Extensions:

PB_SC7_INVALID_LOGIN

Exceptions:

- If the user entered data that do not correspond to any registered user in the system, he or she is asked to re-enter the data or register on Panther Buddy.

Related use cases:

- PB_SC1_FILTER_INPUT

Decision support:

Frequency: Moderate - Performed every time a user logins to the Panther buddy system

Criticality: Very Critical - If the user wants to use the 'Panther Buddy' site he needs to be able to login into the system.

Risk: High

Constraints

Usability:

- No previous training needed.

Mean time to failure:

- One failure for every two months of operation is acceptable.

Performance:

- The user will be redirected to the home page in two seconds

Supportability:

- The software will support all browsers that support HTML 5.

Implementation:

- Client requests the backend implementation to be done in Java and the Eclipse framework. He requests us to use MySQL as database provider and Java Server Pages and Bootstrap as frontend.

Modification history

Owner: Allan Shaji Manamel

Initiation date: 09/07/2015

Date last modified: 09/14/2015

Example scenario:

Allan forgets his password.

Actors: Any user.

Pre-conditions: The user has the login page open.

Description: The user logins into the system using his email and password.

Trigger: The user clicks on the login button.

1) Allan opens the 'Panther Buddy' login page on his laptop browser that supports HTML 5.

2) Allan enters his email amana100@fiu.edu into the email field.
3) Allan enters his password DX345tv in the password field.
4) Allan clicks the login button to submit the data.

Post Conditions: Allan is then able to view his Panther Buddy home page.

3) RECOVER PASSWORD

Use case ID: PB_UC02_RECOVER_PASSWORD

Use case level: System level (End-to-End)

Details:

Actors: Any user.

Pre-conditions: The user has the login page open.

Description: The user tries to recover his password, which is sent to him at his registered email.

Trigger: The user clicks on forget password after entering the wrong password more than three times.

The system responds by

1) User clicks the recover password button on the login page
2) The user is redirected to the recover password page.
3) The user enters his email id used to register in the system.
4) The user clicks the submit button to submit his email id.
5) The user with the email-id is recovered and his password is sent to him by email.

Post condition: The user gets his password on his registered email id.

Alternative courses of action: At any time during step 1 to 4, the user can click the cancel button to return to the 'Panther Buddy' login page.

Extensions:

PB_UC20_LOGIN

Exceptions:

- The user at step 3 enters an email id that is not registered in the system. The user is asked to re-enter a valid email address.

Related use cases:

- None

Decision support:

Frequency: Moderate - Performed every time a use forgets his password. (Used twice a day)

Criticality: Very Critical - If the user wants to use the 'Panther Buddy' site he needs to be able to login into the system, for which he needs his password.

Risk: High

Constraints:

Usability:

- No previous training needed.

Mean time to failure:

- One failure for every two months of operation is acceptable.

Performance:

- The user will be sent a mail in 2 seconds.

Supportability:

- The software will support all browsers that support HTML 5.

Implementation:

- Client requests the backend implementation to be done in Java and the Eclipse framework. He requests us to use MySQL as database provider and Java Server Pages and Bootstrap as frontend.

Modification history

Owner: Allan Shaji Manamel

Initiation date: 09/07/2015

Date last modified: 09/07/2015

Example scenario:

Allan forgets his password.

Actors: Allan.

Pre-conditions: Allan has the login page open.

Description: Allan tries to recover his password, which is sent to him at his registered email.

Trigger: Allan clicks on forget password after entering the wrong password more than three times.

The system responds by

1) Allan opens the 'Panther Buddy' login page on his laptop browser that supports HTML 5.
2) Allan clicks the recover password button.
3) Allan is redirected to the recover password page.
4) Allan enters his email id: amana576@fiu.edu used to register into the system.
5) Allan clicks the submit button to submit the data.
6) Allan's password: DX56ssUT is recovered by the system and sent to his email account.

Post-conditions: A temporary email is then sent to Allan. He will have to change it upon login.

4) CHANGE PASSWORD

Use case ID: PB_UC03_CHANGE_PASSWORD

Use case level: System level (End-to-End)

Details

Actors: Logged in registered user.

Pre-conditions: The user has his own profile page open on 'Panther Buddy'.

Description: The user tries to change his password.

Trigger: The user clicks “change password” on his Panther Buddy profile.

The system responds by

1. User clicks the change password button on his profile page
2. Three fields appear on the page. They are:
   Current Password.

   New Password

   Confirm Password

3. The user enters the data asked
4. User enters his current password in the current password field
5. User enters his new desired password in the new password field.
6. User re-enters his desired new password in the confirm password field.
7. The user clicks the submit button to submit his data.
8. The 3 Fields for password disappear.

Post condition: The user’s password is changed as per his desire.

Alternative courses of action:

At any time during step 1 to 4, the user can click the cancel button to return to the 'Panther Buddy' login page.

Extensions:

None

Exceptions:

- If the user enters a wrong password at step 3, he or she is asked to correct his entered password.
- If the user enters a password that does not fulfil the password criteria for Panther Buddy, he is asked to correct his entered password.
- At step 3, if the user enters a password that does not match his existing password, he or she is asked to correct the entered password.

Related use cases:

- PB_UC02_RECOVER_PASSWORD

Decision support:

Frequency: Moderate - Performed every time a user wants to change his password. (Used twice a day)

Criticality: Moderate

Risk: High

Constraints:

Usability:

- No previous training needed.

Reliability:

- The system shall allow the user to change his password at any time.

Mean time to failure:

- One failure for every six months of operation is acceptable.

Performance:

- The user password will be changed in 2 seconds

Supportability:

- The software will support all browsers that support HTML 5.

Implementation:

- Client requests the backend implementation to be done in Java and the Eclipse framework. He requests us to use MySQL as database provider and Java Server Pages and Bootstrap as frontend.

Modification History

Owner: Allan Shaji Manamel

Initiation date: 09/18/2015

Date last modified: 09/18/2015

Example scenario:

Allan wants to change his password.

Actor: Allan Shaji Manamel

Pre-conditions: Allan Shaji Manamel is at his own profile page open on 'Panther Buddy'.

Description: Allan wants to change his password.

Trigger: Allan clicks “change password” on his Panther Buddy profile.

The system responds by

1. Allan logs in to the 'Panther Buddy' system and opens his profile page on laptop browser that supports HTML 5.
2. Allan clicks the change password button.
3. Fields appear on the page. They are:
   Current Password.

   New Password

   Confirm Password

4. Allan enters his current password = DX56ssUT in current password field.
5. Allan enters his new desired password = FS566ssCV in the new password field.
6. Allan re-enters his new desired password = FS566ssCV in the confirm password field.
7. Allan clicks the submit button to submit his data.
8. Allan's password is changed by the system to his desired password.

Post condition: Allan Shaji Manamel's password is changed as per his desire.

5) POST MESSAGE

Use case ID: PB _UC11_POST_MESSAGE

Use case level: System level (End-to-End)

Details:

Actors: A registered user.

Pre-conditions: The user must be logged-in. The user must fill the information that he wants and post it.

Description: The user will add post by filling the information that he wants and post it or attach file.

Trigger: The non-registered User clicks on the "Register" button.

The system responds by

1. A post window will appear.
2. The user can specify the post URL or browse to the specific file or write in the winnow to post.
3. By clicking "Open" button, the image will be added to the post.

Post condition: The post will be saved in the backend and its name will include the id of the post.

Alternative courses of action: At steps 2 and 3, the user can hit “Cancel” to cancel post.

Extensions:

Drag-and-drop file.

Exceptions:

The post need to be accepted by moderator.

Related Use Cases: PB_UC_16_VIEW_POST

Decision support:

Frequency: Medium - Performed every time a user wants to post message on 'Panther Buddy'.

Criticality: Moderate - to provide better visualization of the post.

Risk: High - attacker may try to post virus.

Constraints:

Usability

- No previous training needed.

Mean time to failure

- One failure for every 24 hours of operation is acceptable.

Performance

- The message will be saved in 2 secs.

Supportability

- The application will be web-based. It will have supported on all sorts of modern devices.

Implementation

- Client requests the backend implementation to be done in Java Server Pages (Wildfly 8.2.0 and the Mojarra 2.2.9 Library). We will use the Bootstrap framework for the frontend. For the data tier, we will use MySQL database 5.6.

Modification History:

Owner: Kedari Sharat

Initiation date: 09/05/2015

Date last modified: 09/06/2015

Example Scenario:

A user wants to post message

Actors: Kedari Sharat

Pre-conditions: Kedari Sharat must be logged-in. Kedari Sharat must fill the information that he wants and post it.

Description: Kedari Sharat will add post by filling the information that he wants and post it or attach file.

Trigger: Kedari Sharat clicks add post button from the menu.

The system responds by

1. A post window will appear.
2. Sharat specify the post URL or browse to the specific file or write in the winnow to post.
3. By clicking “post” button, the image will be added to the post.

Post-conditions: The post will be saved in the backend and its name will include the id of the post.

6) VIEW OWN PROFLE

Use case ID: PB_UC04_VIEW_OWN_PROFILE

Use case level: System level (End-to-End)

Details

Actors: A registered user

Pre-conditions: The user has panther buddy open in a web browser and logged in.

Description: The user views his profile on "Panther Buddy".

Trigger: The user clicks the view user profile button to submit the request.

The system responds by showing the details of the user given below:

1. Name
2. Email
3. Phone
4. Option to change his/her profile data including password.

Post condition: The user can view his/her profile and choose to edit his/her profile or navigate back to home page.

Alternative courses of action:

None

Extensions:

None

Exceptions:

None

Related use cases: PB_UC_20_LOGIN, PB_UC05_EDIT_PROFILE

Decision support:

Frequency: High - Performed every time a user wants to view profile on 'Panther Buddy'

Criticality: High - If the user wants to view profile on the 'Panther Buddy' site, follow users and edit his profile he need to access view profile first.

Risk: Moderate

Constraints:

Usability: No previous training needed.

Mean time to failure: One failure in 2 months of operation.

Performance:

The user data will be showed in 2 second from backend.

Supportability:

The software will support all browsers that support HTML 5.

Implementation:

Client requests the backend implementation to be done in Java and the Eclipse framework. He requests us to use MySQL as database provider and Java Server Pages and Bootstrap as frontend.

Modification history

Owner: Yue Wu

Initiation date: 09/06/2015

Date last modified: 10/18/2015

Example Scenario:

Actors: Allan

Pre-conditions: Allan has panther buddy open in a web browser and logged in.

Description: Allan need view her profile on "Panther Buddy".

Trigger: Allan clicks the view user profile button to submit the request.

The system responds by showing the details of Allan given below:

5. Name : Allan Manamel
6. Email: amana100@fiu.edu
7. Phone: 7888889898
8. Option to change his/her profile data including password.

Post-conditions: Allan can choose edit profile of herself or return to the home page.

7). ACTIVATE USER

Use Case ID: PB_UC_08_ACTIVATE_USER

Use Case Level: System level (End-to-End)

Details:

Actor: User
Pre-conditions: The user should have the login page open in a browser and is logging in for the first time into the system.
Description: The user will hit "Login" button to login on panther buddy system.
Trigger: The user clicks the "Login" button on the main page.

1. The system prompts the user for their Panther account credentials.

2. The user enters their username and password.

3. The user clicks the Login button.

4. The system authenticates the username and password

5. The system activates the user on the system.

Post-conditions:

The user is logged into the system and navigated to the home page.

Alternative Courses of Action:

Extensions: None.

Exceptions: Incorrect login credentials will not allow the user to login and activate himself.

Related Use Cases: PB_UC01_USER_REGISTRATION, PB_UC20_LOGIN.

Decision Support

Frequency: Moderate

Criticality: High.

Risk: High- This is the core feature.

Constraints:

Usability:

No previous training needed

Reliability:

Mean time to failure - 1 failure for every 24 hours of operation is acceptable

Performance:

The user will be navigated to the home page in 2 seconds.

Supportability:

The application will support all major browsers supporting HTML 5.

Implementation

Implementation will be done in JSP, java and Eclipse framework.

Modification History:

Owner: Abhinav Dutt

Initiation date: 09/06/2015

Date last modified: 09/06/2015

Example Scenario:

User Activation

- **Actors:** John
- **Pre-conditions:** John has login page for panther buddy open in a web browser.
- **Trigger:** John clicks the Login button to login to the system.
- **Description:**
  John needs to login himself on "Panther Buddy". John will be asked to enter his login credentials given below:

  User Id: John@fiu.edu

  Password: Manamel0012

  John Clicks the 'Login' button.
- **Post-conditions:** John is marked as an active user in database.

8) LOGOUT

Use case ID: PB_UC21_LOGOUT

Use case level: System level (End-to-End)

Details

Actors: Registered user.

Pre-conditions: The user has logged into the system.

Trigger: The user clicks the logout button

Description:

1. The user clicks the logout button on the webpage.
2. The user session is destroyed
3. The user is redirected to the login page.

Post condition: The user is redirected to the profile page of the user he wants to view details of.

Extensions: None

Exceptions: None

Related use cases: PB_UC20_LOGIN

Decision support:

- **Frequency:** Moderate - Performed every time a user logs out from the Panther buddy system.
- **Criticality:** High
- **Risk:** low

Constraints:

- **Usability:** No previous training needed.
- **Reliability:**

Mean time to failure - 1 failure for every 2 months of operation is acceptable.

- **Performance:**

The user will be redirected to the login page in 2 seconds

- **Supportability:** The software will support all browsers that support HTML 5.
- **Implementation:** Client requests the backend implementation to be done in Java and the Eclipse framework. He requests us to use MySQL as database provider and Java Server Pages and Bootstrap as frontend.

Modification history

Owner: Allan Shaji Manamel

Initiation date: 09/18/2015

Date last modified: 09/18/2015

Example scenario:

Actor: Allan Shaji Manamel

Pre-conditions: Allan Shaji Manamel has logged into the system and is on the home page

Trigger: Allan Shaji Manamel clicks the logout button

Description:

1. Allan Shaji Manamel clicks the logout button on the homepage.
2. Allan Shaji Manamel's session is destroyed
3. Allan Shaji Manamel is redirected to the login page.

Post condition: Allan Shaji Manamel is logged out of the system and redirected to the login page.

9). VIEW POST

Use Case ID: PB_UC16_VIEW_POST

Use Case Level: High-level

Details:

Actor: Registered User

Pre-conditions: The registered user has to login in order to be able to view a post.

Description: The user can view the posts on the home page in chronological order

Trigger: The clicks on the home page button/User redirected after login to home page.

1. The Home page is loaded.
2. The messages in the system are shown in chronological order.

Post-conditions: The user can view the messages in system

Exceptions: None

Related Use Cases: None

Decision Support

Frequency: More than 30 times in an hour

Criticality: Moderate

Risk: Moderate

Constraints:

- **Usability:** The registered user should be able to view any post at any moment
- **Reliability:** Mean time to failure is about 3 failures for every 24 hours of operation
- **Performance:** The post should be displayed in less than two seconds
- **Supportability:** The application will support any text format
- **Implementation:** Bootstrap frontend framework and Java JSP

Modification History:

Owner: James W. Angelo

Initiation date: 09/22/2015

Date last modified: 09/22/2015

Example scenario:

Actor: James
Pre-conditions: James is logged in to the system.
Description: James can view the posts on the home page in chronological order
Trigger: James clicks on the home page button

1. The home page is loaded.
2. The messages in the system are shown in chronological order.

Post-conditions: James can view the messages in system

10). DELETE MESSAGE

Use Case ID: PB_UC14_DELETE_MESSAGE

Use Case Level: System level (end to end)

Details:

- **Actor:** Registered user
- **Pre-conditions:** The user must be logged-in. The user must be owner of that post that he wants to delete

- **Description:** The user will delete the post that he has posted.
- **Trigger:** The user clicks delete post button near the message.
  The system responds by
    1. A window will appear.
    2. The user needed to confirm whether he want to delete the post.
    3. By clicking “ok” button, the post will be deleted.
- **Post-conditions:** The page will be loaded after deleting the post
- **Alternative Courses of Action:**
  At steps 2 the user can hit “Cancel” to cancel deletion.
- **Extensions:** none.
- **Exceptions:** The post need to be owned by user.
- **Related Use Cases:** PB_UC16_VIEW_POST

Decision Support

Frequency: Medium - User

Criticality: Moderate - to provide better visualization of the post.

Risk: Medium.

Constraints:

Usability: No previous training needed.

Reliability

Mean time to failure - 1 failure for every 24 hours of operation is acceptable.

Performance

The message will be deleted in 2 seconds.

Supportability

The application will be web-based and support all major browsers supporting HTML 5.

Implementation

Implementation will be done in HTML 5, java and Eclipse framework.

Modification History:

Owner: Kedari Sharat

Initiation date: 09/05/2015

Date last modified: 09/06/2015

Example scenario:

- **Actor:** Kedari Sharat
- **Pre-conditions:** Kedari Sharat be logged-in. Kedari Sharat must be owner of that post that he wants to delete
- **Description:** Kedari Sharat will delete the post that he owns.
- **Trigger:** Kedari Sharat clicks delete post button.
  The system responds by
  1. A window will appear for confirmation with ‘Ok’ or ‘Cancel’ button.
  2. Sharat needed to confirm wither he want to delete the post.
  3. By clicking “ok” button, the post will be deleted.
- Post-conditions: The page will be loaded after deleting the post

11) XSS FILTERING

Use case ID: PB_SC2_XSS_FILTERING

Use case level: System level

Details

- **Actors:** Registered user.
- **Pre-conditions:** The user has the post message page open.

- **Description:** The user message is filtered to check for XSS attack.
- **Trigger:** The user clicks the 'Post Message' button
    1. The user writes his message in the space provided for writing his message.
    2. The user clicks the post message button to submit the message.
    3. The system checks the message string for malicious code.
    4. System saves the message for moderation by moderator.
- **Post condition:** The users post is saved in the system for moderation by moderator.
- Alternative courses of action:

  At step 1 the user can click the cancel button to stop the process.
- **Extensions:** None
- **Exceptions:**

  If the user at step 1 puts malicious code in his message. The system shows an error for in proper content after step 3.
- **Related use cases:** None

Decision support

- **Frequency:** High - Performed every time a user posts his message. (Used 20 times a day)
- **Criticality:** Very
- **Risk:** High

Constraints

- **Usability:** No previous training needed.
- **Reliability:**

  Mean time to failure - 1 failure for every 2 months of operation is acceptable.
- **Performance:**

  The user will be able to submit his message in 2 sec
- **Supportability:**

  The software will support all browsers that support HTML 5.

- **Implementation:** Client requests the backend implementation to be done in Java and the Eclipse framework. He requests us to use MySQL as database provider and Java Server Pages and Bootstrap as frontend.

Modification history

Owner: Allan Shaji Manamel

Initiation date: 09/18/2015

Date last modified: 09/18/2015

Example scenario:

Actor: Allan Shaji Manamel

Pre-conditions: Allan Shaji Manamel has the post message page open.

Trigger: Allan submits his message.

- Allan logs into the 'Panther Buddy' system and opens the post message page on his laptop browser that supports HTML 5.
- Allan writes his message "Used book - Introduction to Algorithms by Thomas Cormen up for sale at 19$".
- Allan clicks the submit button to submit his message.
- The system checks if the message contains any malicious code or not.
- The system verifies the message to be ok.
- The system saves the message to be moderated by a moderator.

Post condition: Allan Shaji Manamel's post is saved in the system for moderation by moderator.

Misuse case

Use case ID: PB_MC2_XSS_FILTERING

Use case level: System level

Details

- **Actors:** Misuser.
- **Pre-conditions:** The misuser has the post message page open.
- **Description:** The misuser embeds malicious scripts.
- **Trigger:** The misuser clicks the post message button.
    1. The misuser writes his message in the space provided for writing his message.
    2. The misuser clicks the post message button to submit the message.
- **Post condition:** The user's message when rendered on page runs the script to hack into the system to fetch/manipulate unauthorized data from the system.
- **Criticality:** Very
- **Risk:** High

12). SQL INJECTION PREVENTION ON LOGIN

Use case ID: PB_SC1_FILTER_INPUT

Use case level: System level (End-to-End)

Details

- **Actors:** Any user.
- **Pre-conditions:** The user has the login page open.
- **Description:** The user logins into the system using his email and password.
- **Trigger:** The user clicks the 'Login' button
    1. User enters his email into the email field
    2. The user enters his password in the password field
    3. The user clicks the login button to submit his data.
    4. The system checks the email and password for malicious content before querying for the user with the entered login credentials
- **Post condition:** The user is redirected to the Panther Buddy home page.
- **Extensions:** None

- **Exceptions:** None
- **Related use cases:** PB_UC_20_LOGIN, PB_SC7_INVALID_LOGIN

Decision support:

Frequency: Moderate - Performed every time a user logins to the Panther buddy system

Criticality: Very Critical

Risk: High

Constraints:

Usability: No previous training needed.

Reliability:

Mean time to failure - 1 failure for every 2 months of operation is acceptable.

Performance:

The user will be redirected to the home page in 2 seconds

Supportability:

The software will support all browsers that support HTML 5.

Implementation: Client requests the backend implementation to be done in Java and the Eclipse framework. He requests us to use MySQL as database provider and Java Server Pages and Bootstrap as frontend.

Modification history

Owner: Allan Shaji Manamel

Initiation date: 09/18/2015

Date last modified: 09/18/2015

Example scenario:

Actor: Allan Shaji Manamel

Pre-conditions: Allan has the login page open.

Trigger: Allan clicks the ‘Login’ button

1. Allan opens the 'Panther Buddy' login page on his laptop browser that supports HTML 5.

2. Allan enters his email amana100@fiu.edu into the email field.

3. Allan enters his password DX345tv in the password field.

4. Allan clicks the login button to submit the data.

5. The system checks the input email amana100@fiu.edu and password DX345tv for malicious content

Post condition: Allan is redirected to the Panther Buddy home page.

Misuse case

Use case ID: PB_MC1_FILTER_INPUT

Use case level: System level (End-to-End)

Details

Actors: Misuser.

Pre-conditions: The misuser has the login page open.

Description: The misuser gains entry into the system.

Trigger: The misuser clicks the ‘Login’ button.

1. The misuser enters his email into the email field

2. The misuser enters a password: dummy’ OR 1=1 OR ‘’ = ‘

3. The misuser clicks the login button to submit his data.

The system will put the string in the where clause of a SQL statement used to compare existing data for input data like WHERE password = ‘dummy’ OR 1=1 OR ‘’ = ‘’, thereby gaining entry into the system.

Post condition: The user is redirected to the Panther Buddy home page.

Related use cases: PB_UC_20_LOGIN, PB_SC7_INVALID_LOGIN

Criticality: Very Critical

Risk: High

6.3. Appendix C – Detailed class Diagrams

This Appendix contains the detail class diagram.

1. High Level Package Architecture

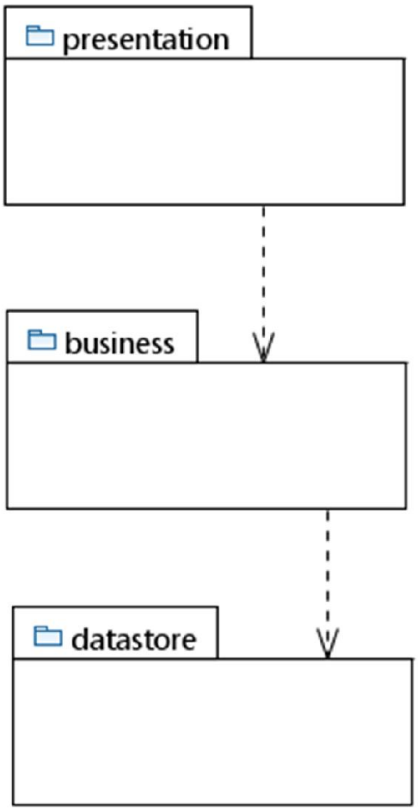


Figure: Shows the high level package architecture. It shows the 3-tier package architecture we use.

2. Presentation Package

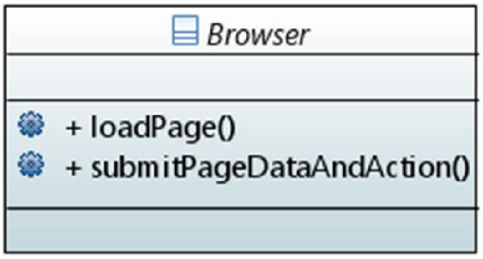


Figure: The abstract class Browser

3. Business Package

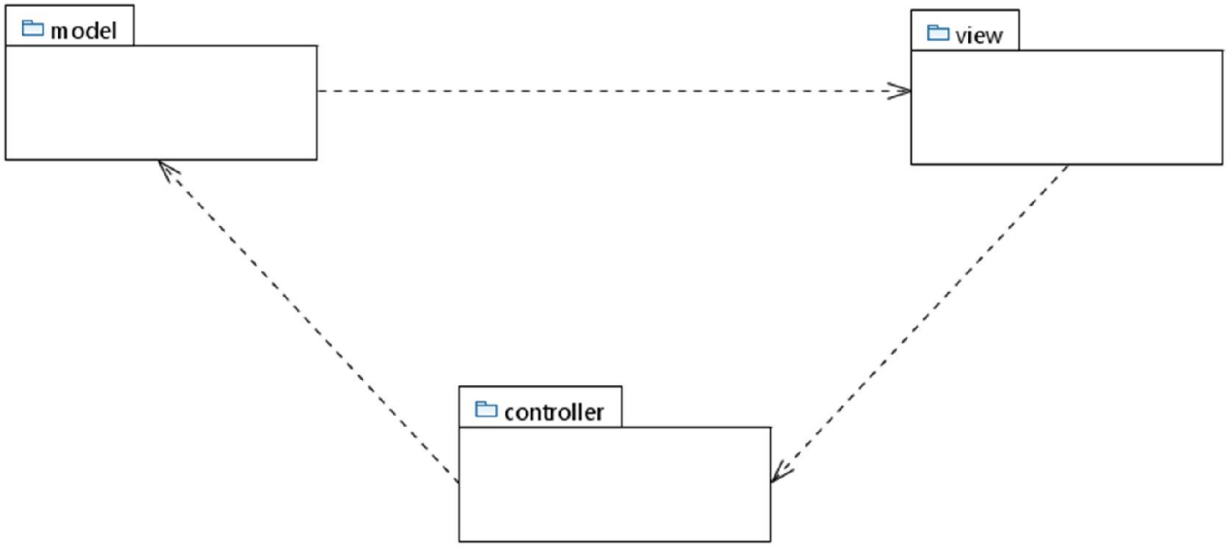


Figure: Shows the package structure present inside the business package.

4. Model Package

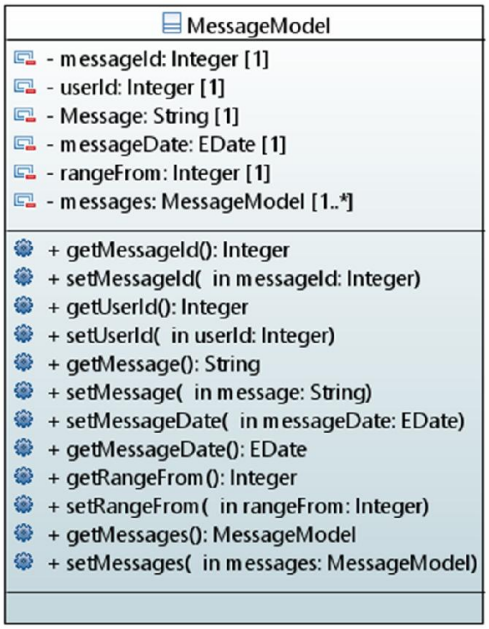


Figure: Shows the model classes of the MVC architecture implemented within the business tier

5. View Package

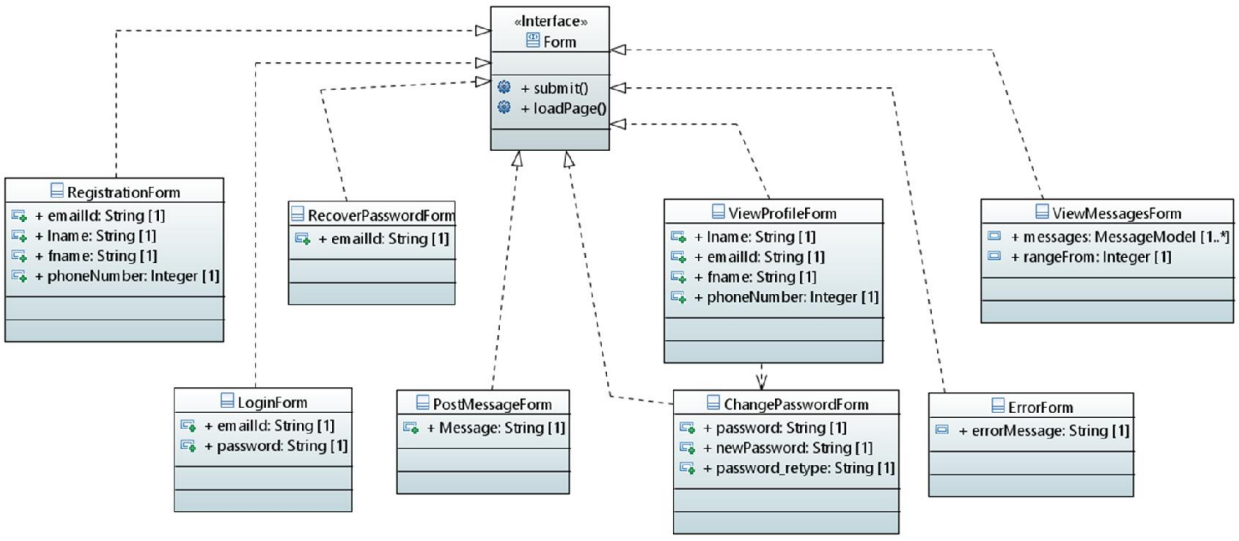


Figure: Shows the classes in the view package.

6. Controller Package

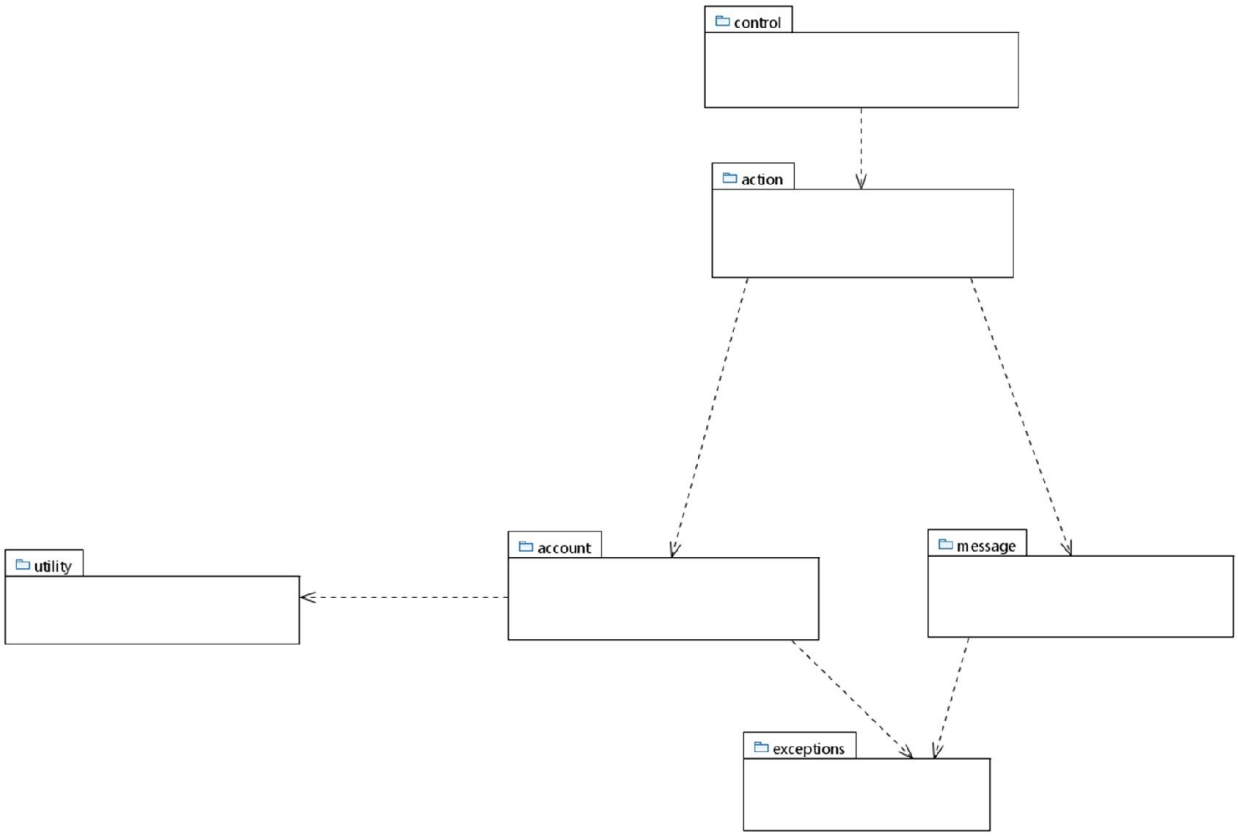


Figure: Shows the packages in the controller package.

7. Control Package

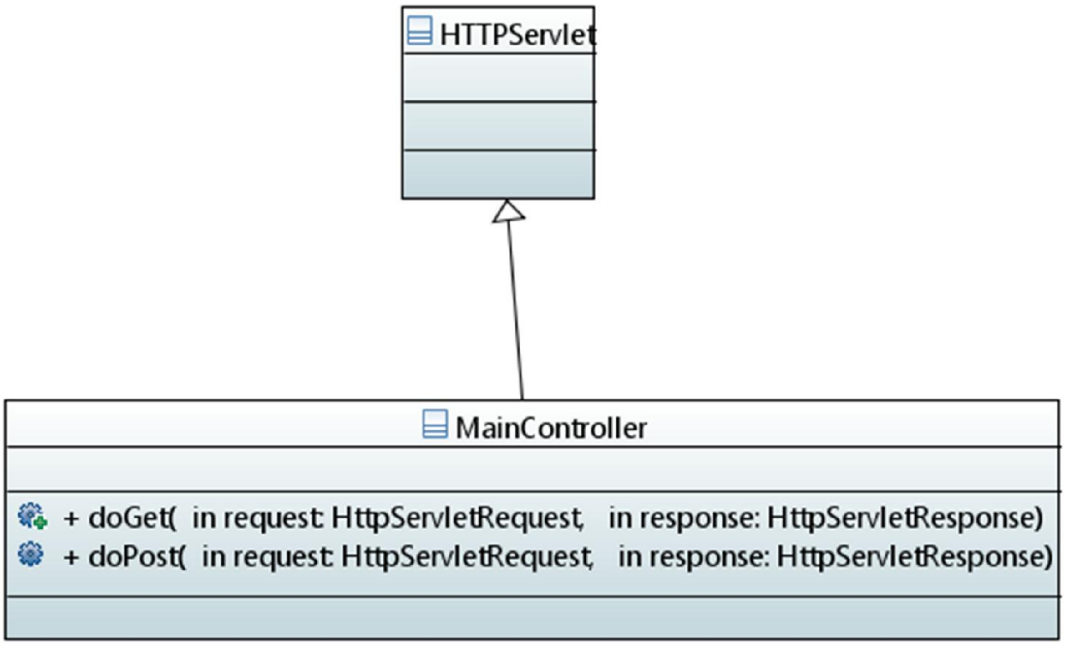


Figure: Shows the classes in the control package

8. Action package

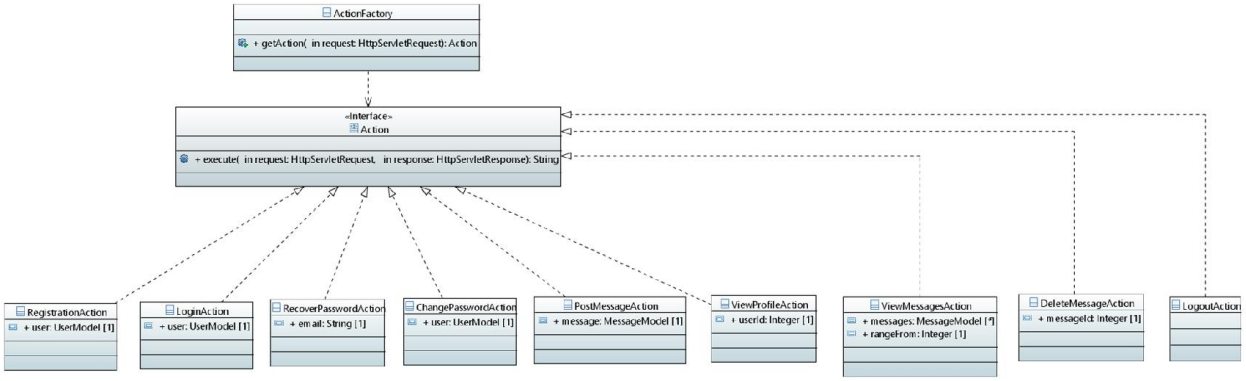


Figure: Shows the classes inside the action package

9. Account package

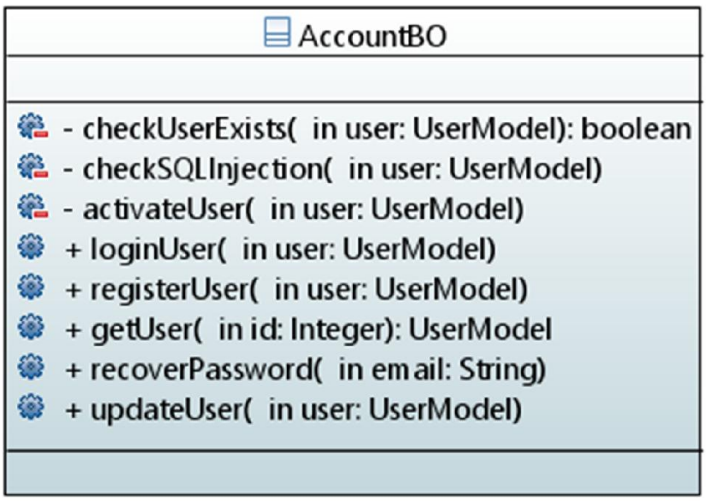


Figure: Shows the AccountBO class in account package.

10. Message package

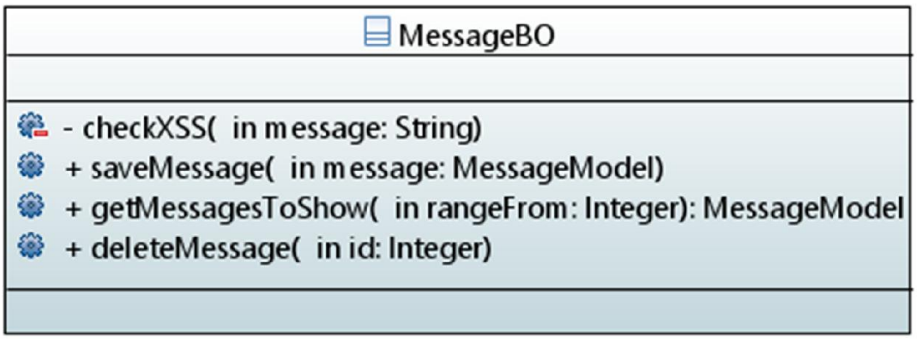


Figure: Shows the MessageBO class in message package

11. Utility package

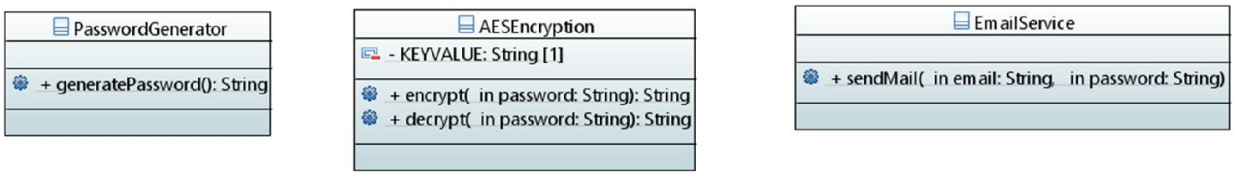


Figure: Shows the utility classes in the utility package

12. Exceptions package

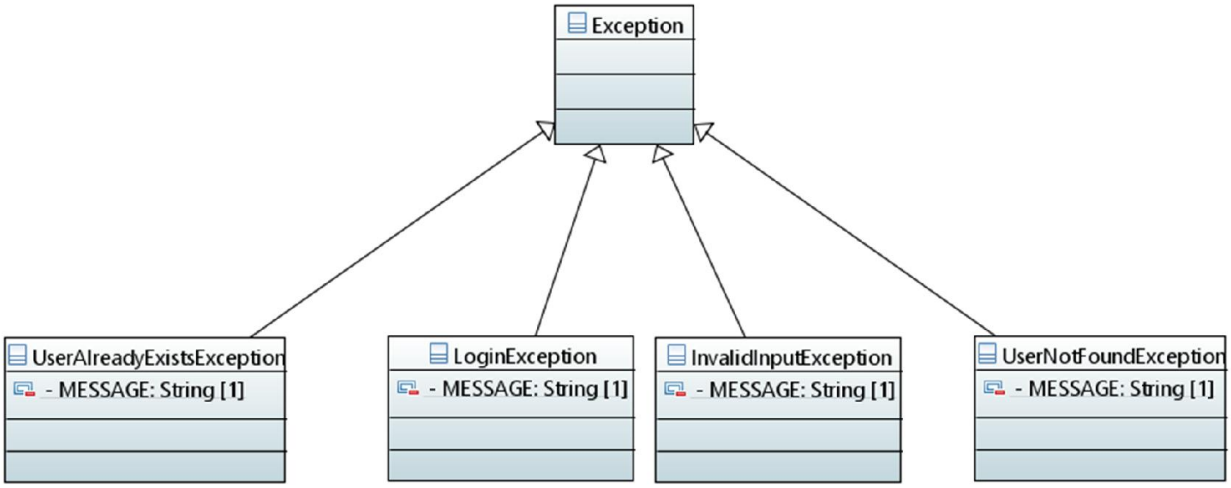


Figure: Shows the business exception classes of the system.

13. Datastore package

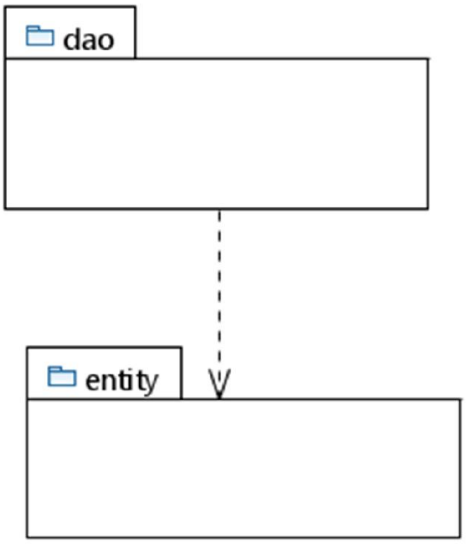


Figure: Shows the package structure of the data tier of the high level 3-tier architecture.

14. DAO package

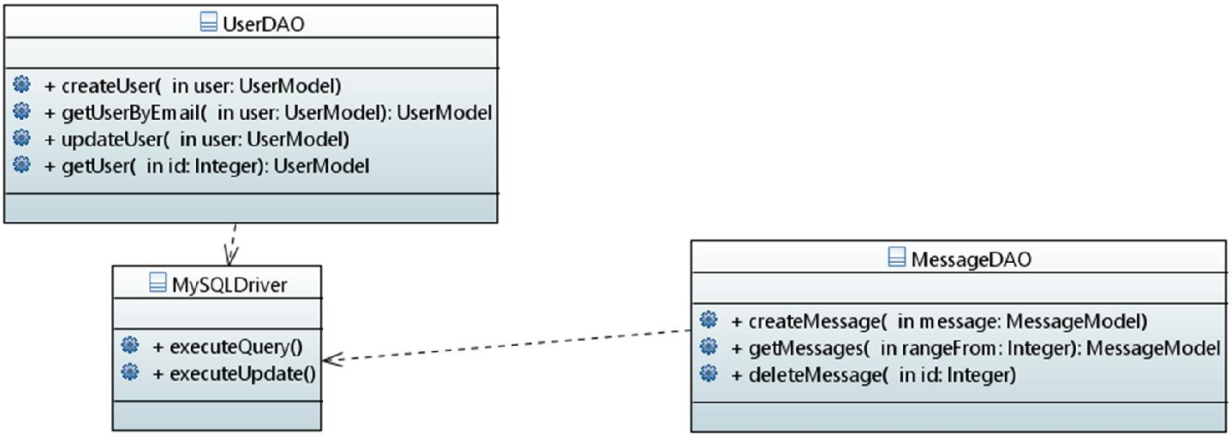


Figure: Shows the classes in the DAO package.

15. Entity package

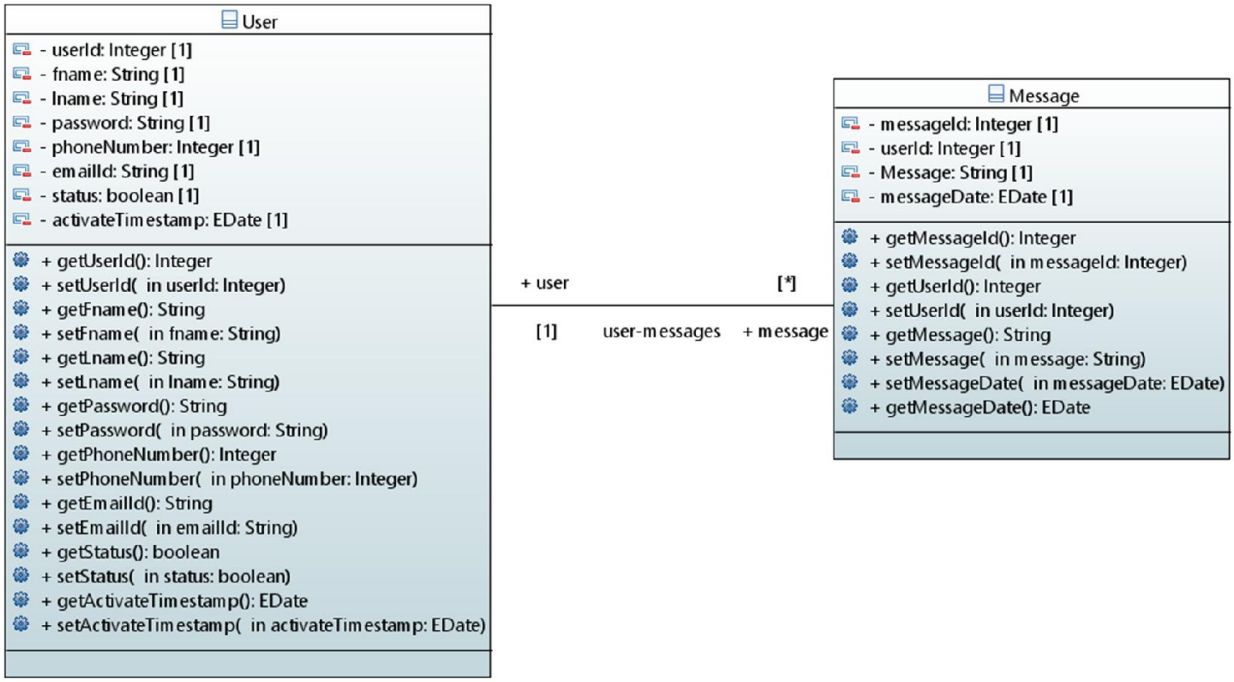


Figure: Shows the entity classes in the entity package.

6.4. Appendix D – Class Interfaces

Note: Attached with Document. Please use index.html in doc folder supplied.

6.5. Appendix E – Diary of Meetings and Tasks

Meeting Dairy 1

Project Name: Panther Buddy

Date: 10/12/15

Time: 8 pm – 9:30 pm

Location: ECS 252

Attendance: Abdur, James, Yue Wu, Allan, Abhinav, Sharat, Yulong Qiu.

Agenda: Discuss the design patterns. Prepare everything such as set up sever and review diagrams. Install sever in eclipse follow instructions.

Summary of discussion: Discussed the role of deliverable 2 and assigned tasks.

Assigned Tasks: Download Wildfly8.2.0 server. Follow instructions at the website Allan sent to everyone. Use mojarra library 2.2.9, and checkout the project from SVN location - trunk/src/pantherbuddy-web

Meeting Dairy 2

Project Name: Panther Buddy

Date: 10/14/15

Time: 8 pm – 9:30 pm

Location: ECS 252

Attendance: Abdur, James, Yue Wu, Allan, Abhinav, Sharat, Yulong Qiu.

Agenda: Discuss the design patterns and architectural patterns. Make the time schedule of the project. Assign task to everyone.

Summary of discussion: Use 3-Tier architecture.

Assigned Tasks: Abdur do minimal class diagram. Abhinav do State machine diagrams, James have sequence diagram, Sharat, and Yulong Qiu take care of the diagram of subsystems. Yue Wu do the UML Profile documentations and power point. Allan reviews every other's work.

Meeting Dairy 3

Project Name: Panther Buddy

Date: 10/19/15

Time: 8 pm – 9:30 pm

Location: ECS 252

Attendance: Abdur, James, Yue Wu, Allan, Abhinav, Sharat, Yulong Qiu.

Agenda: UML Profile, Determine the design patterns and architectural patterns. Report the progress of task assigned to everyone.

Summary of discussion: Use Model View Controller (MVC) and use 3-Tier architecture. We chose these patterns because they represent the way our system operates.

Assigned Tasks: Analysis the requirement and work on the architecture diagram

Meeting Dairy 4

Project Name: Panther Buddy

Date: 10/23/15

Time: 7 pm – 9:30 pm

Location: ECS 252

Attendance: Abdur, James, Yue Wu, Allan, Abhinav, Sharat, Yulong Qiu.

Agenda: Review all the minimal class diagram, the architecture diagram and sequence diagrams.

Summary of discussion: determine and correct all the diagrams.

Assigned Tasks: Preparing all the documents and review it. Abdur do review DD document part 1: Introduction. Abhinav do Part 2: Proposed Software Architecture, James do part 3: Detailed Design, Sharat and Yulong Qiu take care of part 4: Glossary. Yue Wu have part 6: Appendix. Allan do part 5: References and review every other's work.

Meeting Dairy 5

Project Name: Panther Buddy

Date: 10/30/15

Time: 6:00 pm – 9:30 pm

Location: ECS 252

Attendance: Abdur, James, Yue Wu, Allan, Abhinav, Sharat, Yulong Qiu.

Agenda: Review all the diagrams and documents.

Summary of discussion: Preparing for the presentation and PowerPoint.

Assigned Tasks: Everyone prepare for the presentation. Yue Wu and James take care of the PowerPoint of phase 2.

Meeting Dairy 6

Project Name: Panther Buddy

Date: 11/02/15

Time: 8:00 pm – 9:30 pm

Location: ECS 252

Attendance: Abdur, James, Yue Wu, Allan, Abhinav, Sharat, Yulong Qiu.

Agenda: Review all the diagrams and documents.

Summary of discussion: Everyone in the group submitted their assigned task. Tasks had been reviewed by everyone. Identified the mistakes. Discussed mistakes and changes were assigned.

Assigned Tasks: Everyone prepare for the presentation