Team 6: Panther Buddy

Structure of the Final Systems Document

Prof. Peter Clarke

Abhinav Dutt

Allan Shaji Manamel

Sharat Kedari

Yulong Qiu

Abdur Rahman

Bin Shahid

Yue Wu

James Angelo

November 30, 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 Design Documents in deliverable2, we presents the functional and non-functional requirements, we use unified software development model and 3-tier Architecture (A subset of the Multi-Tier architecture) for the project, with the secondary architecture as Model View Controller (MVC) architecture, we include package diagrams, minimal class diagram, state machine and sequence diagram for each use case. We separate the use cases to four scenario: Registration, Login, Logout, Recover Password and SQL Injection Security. View Profile and Change User Password. Post, view and delete message and XSS Security and activate User. As the third deliverable, we already have all the system design, software architecture and detail design. We finish the coding part and do unit test, system tests and subsystem tests. And we do evaluation of tests.

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. Scope of System

When delivered the system will allow any user to get registered with panther buddy application.

The system will allow users to view and post messages in the panther buddy application, users can also delete his/her own messages, users can also view others profile.

The boundary of the system would be all kinds of end users except hackers. We will built security artifacts to prevent hackers, also there is no limitations of using method, and user can use both smartphone, laptop and portable iOS and Android devices to access it.

1.3. Development methodology

In this part, 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 use

case-driven model that uses the Unified Modeling Language (UML) to represent the different models of our software system.

We have 5 design phases, the first one is analysis model, we analysis both the functional and nonfunctional requirement of our system. The second one is design model, we design the software architecture and detail design. The third one is deployment model, use case model distributed by deployment model. And the forth one is implementation model, use case model implemented by this model. The last one is test model, we do unit test, subsystem tests and systems tests. 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.

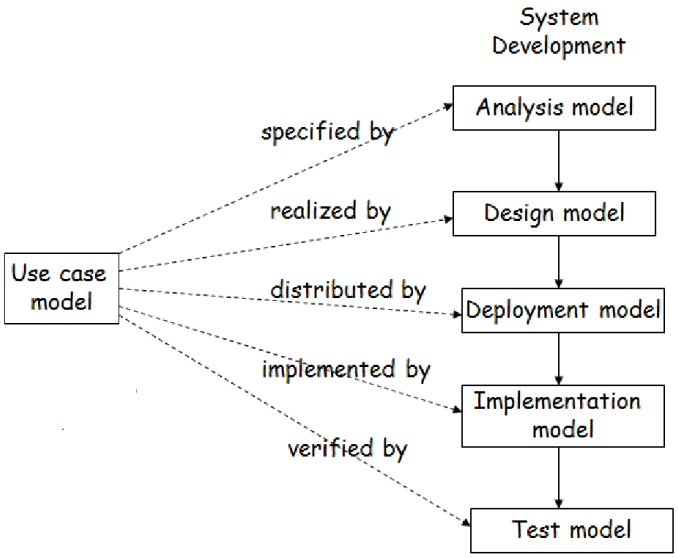


Figure: Shows the USDP process

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

This is the Final Systems document for the deliverable of Advance Software Engineering Panther Buddy System. This document contains seven chapters which are further divided into a number of subchapters. In the chapter1, we do a brief introduction of our system' purpose, scope and design methodology. In the following chapters 2 of the document, we analysis our system's limitations and problems. In chapter 3 we make project plan. And in chapter 4 we analysis both the functional and nonfunctional requirements of system. In chapter 5 we give an in-depth description of the current system in terms of its software architecture, system decomposition, hardware to software mapping, data management, and security management. In chapter 6 we describe the class diagrams, state machines, object interactions, and class designs. We present the test process both unit tests, subsystem tests, system tests and do evaluation of tests. We present the glossaries and references in chapter 8, 9 and 10, respectively. Finally, in section 11, we present the appendix.

2. Current System

There are systems currently that do a similar job, but they are distributed in functionality i.e. they deal only with a particular line of queries for e.g. furniture, books, etc. We wish to collate these different lines of queries in a single place for the convenience of the user.

The Panther Buddy System may also have many limitations.

- Possible Downtime:

There will be maintenance downtime for one hour on every first of the month.

- Technical Issues:

We should be aware of the fact that this technology is always prone to outages and other technical issues. Besides, we will need an Internet connection to be logged onto the system at all times. We will invariably be stuck in case of network and connectivity problems.

- User limitations:

Only in campus user will have interest in using our system. The number of user may not be big enough for us. It's not good for us to gathering funding from investigator.

- Security problem:

Our security use case still need to be improved and we need manual update and maintain every time after facing new security problem.

- System limitation of speed:

Panther Buddy use MySQL database, encrypt and decrypt procedure take time. For example, a user use "dddddd" as his password, the system need to encrypt it as 16 charter into database, and every time he log in, system need to decrypt the 16 charter into the original password "dddddd" from database. Also, XSS filtering and These procedure often decimate the user experience by making it a little bit slow to load when user login our system.

- User waiting time:

If user forget password and request to recovery password. It take time for user to wait for the recovery password email to change the password.

In summary, Although Panther Buddy Systems itself have some limitation and problems, we still can provide user information resources with high performance and usability.

3. Project Plan

This section will briefly describe the project plan for the system and a reference to the appropriate project schedule in Appendix A

3.1 Introduction

The project plan is explained in this part of the document, this includes the roles assigned to every team member, the hardware and software needed to complete the project and the work breakdown that will be presented in a Gantt chart. The tasks presented in this chart may change and new tasks might be added and another might be deleted or modified. The System shall allow a registered user to login (Appendix B – Use case PB_UC20_LOGIN "User Login").

3.2 Project organization – assignment of roles for the entire project.

The organizational structure of the roles involved in this project for each of the three phases of the project is described below:

Team Member	Phase 1	Phase 2	Phase 3
Abhinav Dutt	Team leader	Architect and requirements engineer	Tester
Allan Shaji Manamel	Modeler	Team larder & architect and requirements engineer	Coder and Tester
Yue Wu	Document writer	Technical writer	Team leader
Sharat Kedari	Time keeper& Minute taker	Technical writer & Minute taker& Time keeper	UI designer
Yulong Qiu	Document writer and modeler	Developer	Tester

Abdur Rahman Bin Shahid	Modeler	Developer	Coder and test evaluator
James Angelo	Presentation	Technical writer	Test evaluator

Leader: Oversees all project tasks and ensures all milestones are reached. He/she will be 9 primary facilitator for the group meetings, questions regarding the project, and assigns work to each member of the group. And also work with some scenarios, use cases and diagrams.
Document writer: Will oversee and do the creation of the required documents for the phase of software engineering life cycle.
Modeler: Handles the modelling of the system. Each of the members will do some use cases and scenarios and their related diagrams. Finally, the lead combines all the diagrams in one model and validates the model.
Presentation: The person will do the creation of the presentation for the deliverable.
Time keeper: Is responsible for keeping track of time and notifying the facilitator if a discussion consumes more time than is allocated. A vote might be required to continue discussion or move on to the next point.
Minute keeper: Is responsible for recording the meeting i.e. information for the diary.
Architect: Will be responsible for architecture design and implementation for the software.
Developer: Will help the architect in developing the software.
Document writer: Will oversee and do the creation of the required documents for the phase of software.
Tester: Will test the system for validation and verification.
Test evaluator: Will organize the code coverage for the unit, subsystem and system testing all the test results and comment on how to possibly fix the bugs.

3.3 Cost Estimation

This section shows the cost estimation done for the project. We have used COCOMO 2 Early Design model to estimate the cost of the project.

The function point cost estimation approach is based on the amount of functionality in a software project and a set of individual project factors. Function points are useful estimators since they are based on information that is available early in the project life cycle.

Function points measure a software project by quantifying the information processing functionality associated with major external data or control input, output, or file types. Five user function types should be identified as defined in table below.

- **Project task timelines:**

#	Task name	Duration (In days)	Start	Finish	Predecessor
1	**Panther Buddy System**	**83.04**	**8/25/2015**	**11/15/2015**	
2	**Deliverable 1**	**20**	**8/25/2015**	**9/13/2015**	
3	**System requirements**	**11**	**8/25/2015**	**9/4/2015**	
4	Feature diagram design	1	8/25/2015	8/25/2015	
5	Functional requirements	7	8/26/2015	9/1/2015	4
6	Non functional requirements	8	8/26/2015	9/2/2015	4
7	Use case definitions and diagrams	10	8/26/2015	9/4/2015	4
8	**System analysis**	**9**	**9/5/2015**	**9/10/2015**	
9	Scenario description for use cases	5	9/5/2015	9/9/2015	7
10	Object model	6	9/5/2015	9/10/2015	7
11	Dynamic model	6	9/5/2015	9/10/2015	7
12	**User interface**	**2**	**9/11/2015**	**9/12/2015**	**11**
13	**Presentation**	**1**	**9/13/2015**	**9/13/2015**	**12**
14	**Deliverable 2**	**25**	**9/17/2015**	**10/11/2015**	**2**
15	**Software architecture**	**8**	**9/17/2015**	**9/24/2015**	
16	Package diagram design	8	9/17/2015	9/24/2015	13
17	UML Profile	8	9/17/2015	9/24/2015	13
18	Subsytem decomposition	8	9/17/2015	9/24/2015	13
19	**Object design**	**16**	**9/25/2015**	**10/10/2015**	
20	Class diagram for subsystem	8	9/25/2015	10/2/2015	18
21	Object Interaction	8	9/25/2015	10/2/2015	18
22	Detail class design	8	9/3/2015	10/10/2015	21
23	OCL constraints	8	9/3/2015	10/10/2015	21
24	**Presentation**	**1**	**10/11/2015**	**10/11/2015**	**23**
25	**Deliverable 3**	**28.04**	**10/19/2015**	**11/15/2015**	**14**
26	**Validation**	**5**	**10/19/2015**	**10/23/2015**	
27	Validation of use case model	5	10/19/2015	10/23/2015	24
28	Validation of analysis model	5	10/19/2015	10/23/2015	24
29	Validation of system model	5	10/19/2015	10/23/2015	24
30	Validation of detail design model	5	10/19/2015	10/23/2015	24
31	**Implementation**	**20**	**10/24/2015**	**11/12/2015**	**30**
32	**Final Presentation**	**3**	**11/13/2015**	**11/15/2015**	**31**

- **Effort estimation**

Category	Effort	Comments/Assumptions
Requirements/Design Effort	93.32	Outputs are Technical Design and Information Architecture doc. Information Architecture doc will have guidelines for interface design.
Development Effort	79.99	Source Code
Test Planning	21.33	Output are Test Plan and Test Cases
Testing Effort	26.66	Output is Test Results
Documentation Effort	13.33	Outputs are Deployment and Build Documents
Deployment Effort	2.67	Effort to deploy application
Off-shore Management	26.66	Output is Detailed Project Plans, Risk and Issues tracking, Weekly status reports etc.
On-site Management Effort	2.67	Client Communication. Output is usually any list of issues or improvement opportunities
Total Effort	**266.62**	

- **Costs**

Cost Units

Units	per hour	per days
Architect/Designer	20	160
Developer (Blended rate for Senior and Junior Developer)	15	120
Testing Lead	20	160
Tester	15	120
Technical Writing Cost	15	120
Manager	25	200
On-site Manager	60	480

Total Project Cost

Category	Effort (Days) Estimated	Effort (Days) to Quote	Cost
Requirements/Design Effort	93.3	39.0	$6,240
Development Effort	80.0	19.0	$2,280
Test Planning	21.3	2.0	$320
Testing Effort	26.7	3.0	$360
Documentation Effort	13.3	5.0	$600
Deployment Effort	2.7	1.0	$160
Off-shore Management	26.7	3.0	$600
On-site Management Effort	2.7	1.0	$480
Total	**266.6**	**73.0**	**$11,040**
		Blended Rate	**$18.90**

Total Support Cost

Man Months	1
Man Days	5

Category	Hours per day	Total Hours	Cost
Designers	1	5	$100
Developers	8	40	$600
Testers	1	5	$75
Managers	0.5	2.5	$63
		Total	**$838**

4. Requirements of System

In this chapter we will describe the functional and nonfunctional requirements of the system so we can justify our solution. In addition we will talk about specific functions that the system must perform together with client-defined constraints.

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.

4.1. Functional and Nonfunctional Requirements

In the following subsections we will discuss the overall functionality of the system, what it should do and how it should do.

4.1.1. 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”).
- The system shall allow the user to edit his profile (Appendix B – Use Case PB_UC05_EDIT_PROFILE “edit profile”)
- The system shall allow the admin user to delete his profile (Appendix B – Use Case PB_UC06_DELETE_PROFILE “delete profile”)
- The system shall allow the user to follow posts of a user (Appendix B – Use Case PB_UC07_FOLLOW_USER “follow user”)
- The system shall allow the user to flag a post (Appendix B – Use Case PB_UC09_FLAG_MESSAGE “flag message”)
- The system shall allow the user to view flagged post (Appendix B – Use Case PB_UC10_VIEW_FLAGGED_MESSAGE “view flag message”)
- The system shall allow the user to edit a post (Appendix B – Use Case PB_ UC12_EDIT_MESSAGE “edit message”)
- The system shall allow the user to subscribe to another users posts (Appendix B – PB_UC13_SUBSCRIBE “subscribe”)
- The system shall allow the user to subscribe to another users posts (Appendix B – PB_UC13_SUBSCRIBE “subscribe”)
- The system shall allow the user to search messages based on search criteria (Appendix B – PB_UC15_SEARCH_POST “search post”)
- The system shall allow the user sort messages based on a criteria (Appendix B – PB_UC17_SORT_POST “search post”)

4.1.2. 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

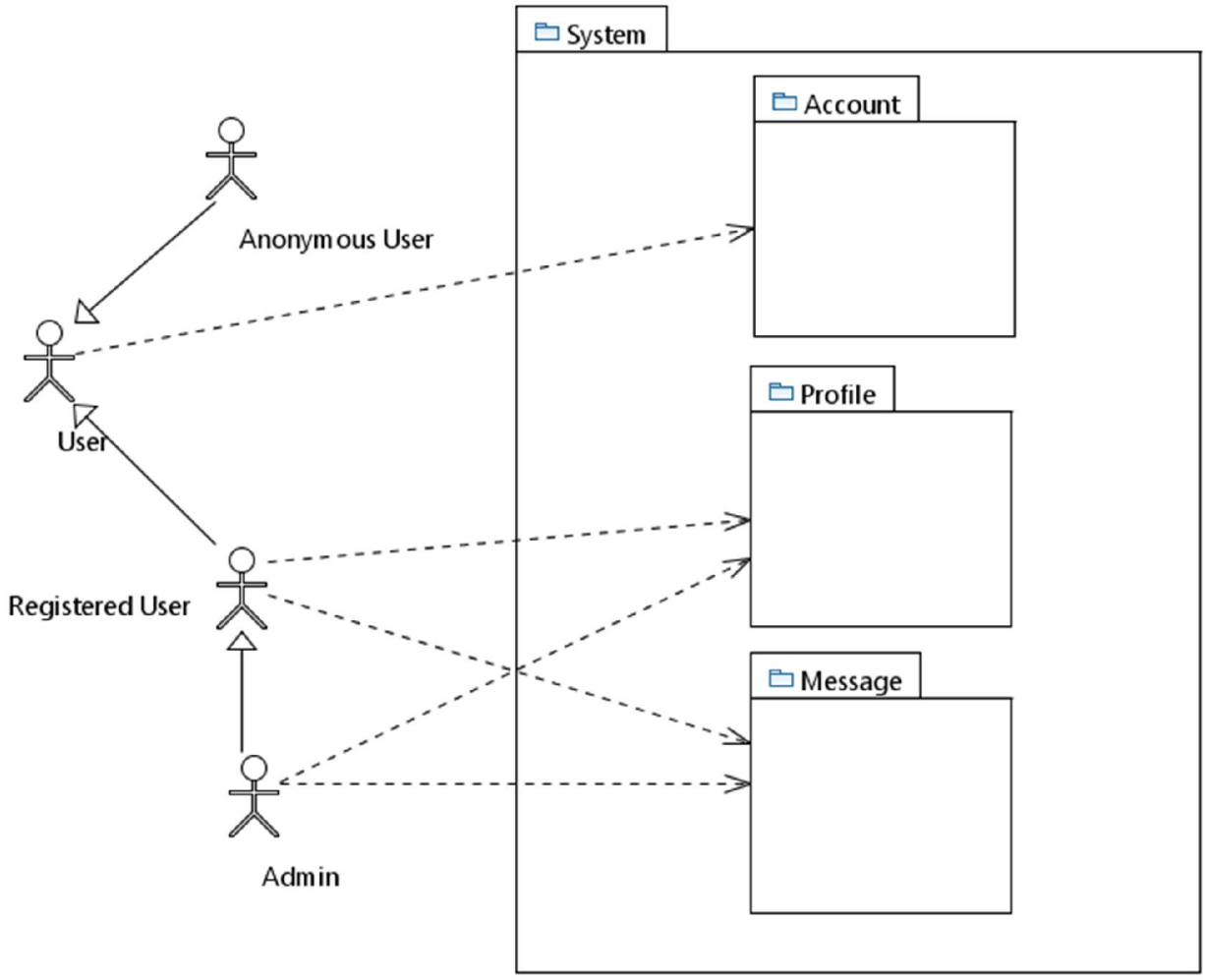
- **Usability** – No previous training needed for any use case.

- **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)

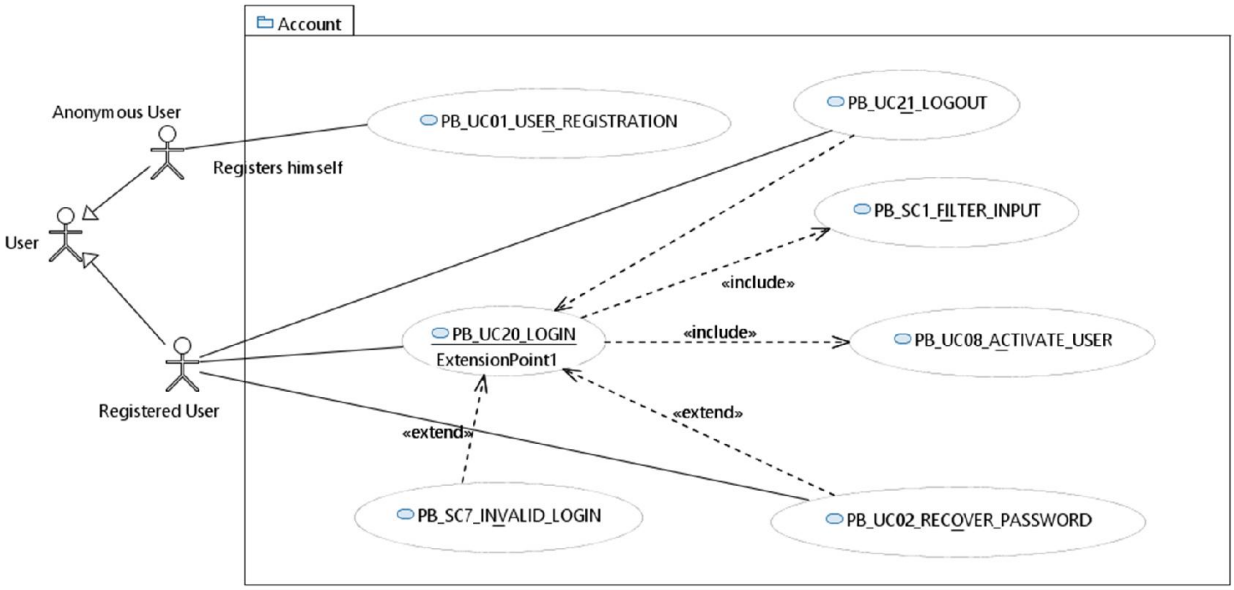
4.2. Use case diagram for all use cases and those implemented.

Below is the high level use case diagram

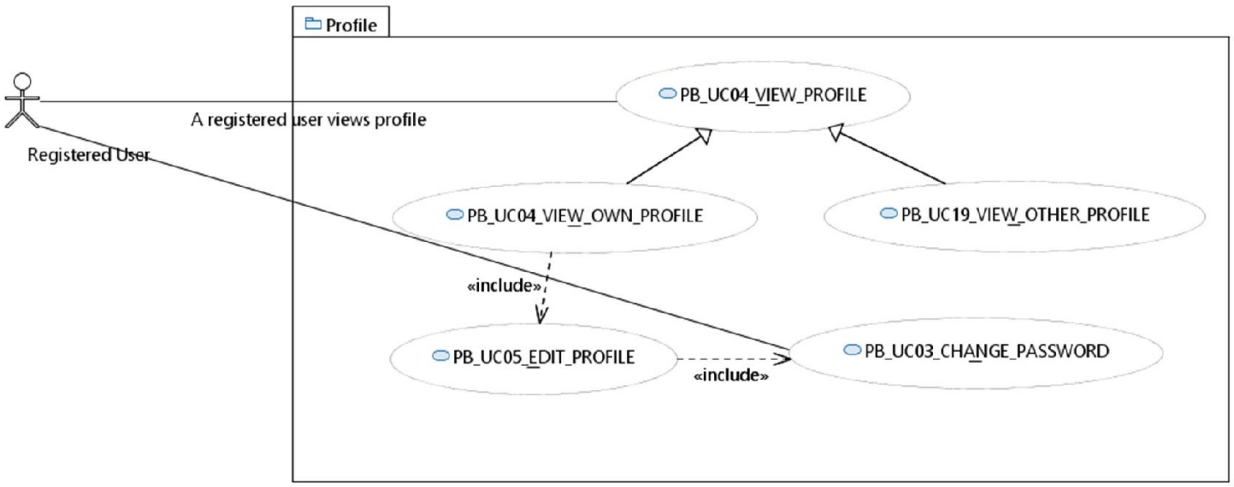


Below are the use case diagrams for the use cases being implemented.

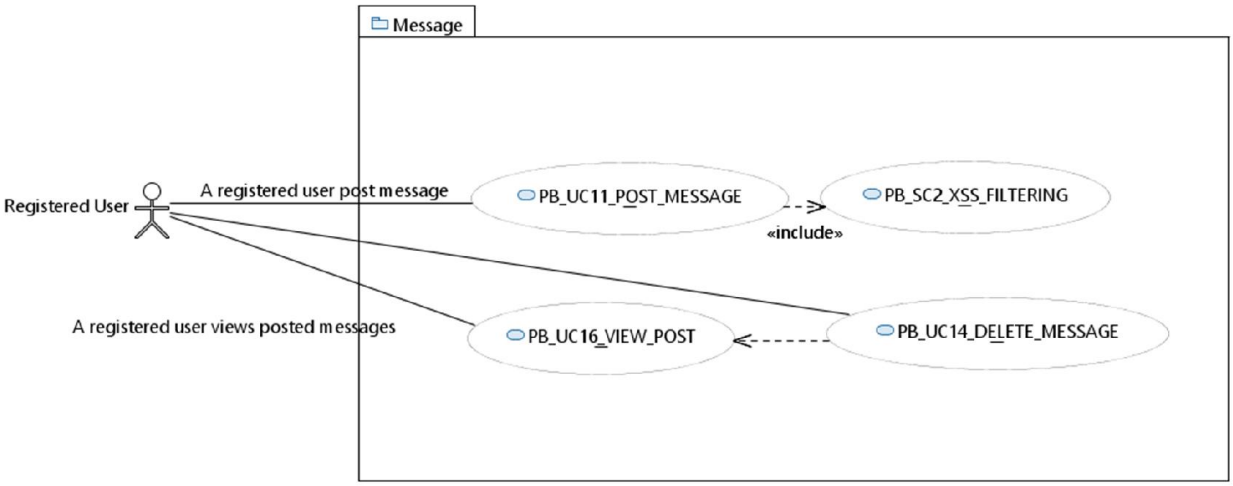
1. Registration, Login, Logout, Recover Password and SQL Injection Security



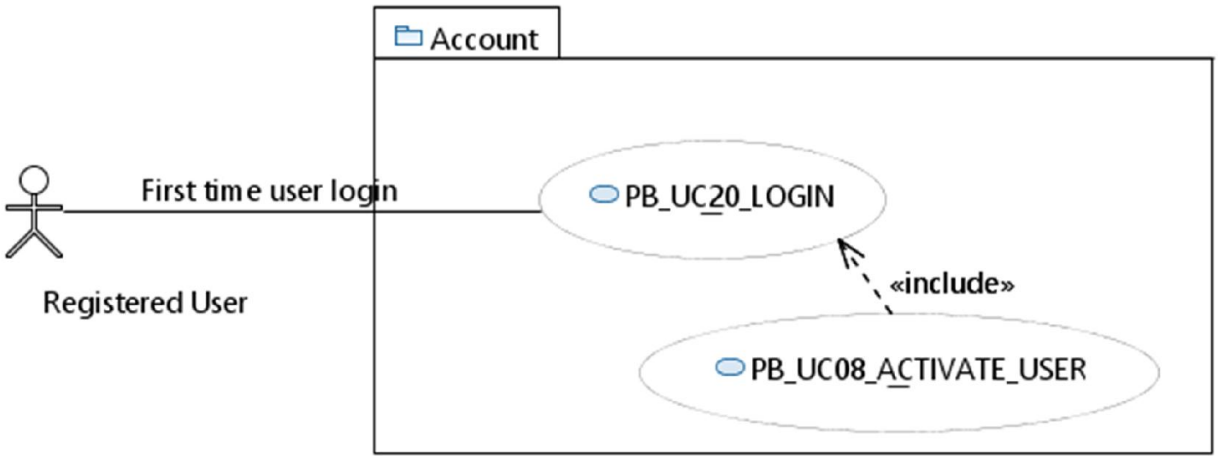
2. View Profile and Change User Password



3. Post, view and delete message and XSS Security



4. Activate User



4.3. Requirements Analysis

In this chapter we present models that show both static and dynamic views of the system. 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.

4.3.1. Description of ten (10) scenarios.

1. USER 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.

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.

Extensions: PB_SC7_INVALID_LOGIN

Exceptions: None

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

Usability: No previous training needed.

Reliability:

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

Performance:

1. 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 PostgreSQL as database provider and Java Server Faces 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 login page open.

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

Post condition: Allan Shaji Manamel is redirected to the home page of Panther Buddy system.

2. 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 and is on the home page.

Description: The user logs out from the system.

1. User clicks the logout button on the screen
2. The system destroys the users session and redirects him to the login page

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

Extensions:

Exceptions:

Related use cases: PB_SC3_SESSION_TIMEOUT

Decision support:

Frequency: High - Performed every time a user logs out from the Panther buddy system

Criticality: low

Risk: low

Usability: No previous training needed.

Reliability:

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

Performance:

1. 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 PostgreSQL as database provider and Java Server Faces and Bootstrap as frontend.

Modification history

Owner: Allan Shaji Manamel

Initiation date: 09/19/2015

Date last modified: 09/19/2015

Example scenario:

Actor: Allan

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

1. Allan clicks on the logout button on the page.
2. Allan's session is destroyed and he is redirected to the Panther Buddy login page

Post condition: Allan Shaji Manamel is redirected to the Panther Buddy login page.

3. USER REGISTRATION

Use case ID: PB_UC1_USER_REGISTRATION

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

Details

Actors: Any anonymous user or an already registered user.

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

Description: The user registers himself on "Panther Buddy".

1. The user will be asked to enter his details.
    - First name: The first name of the user.
    - Last name: The last name or surname of the user
    - Email id: The email id the user wants to register on 'Panther Buddy' with.
    - Phone number: The user primary phone number.
2. The user clicks the submit button to submit the data.

3. The user 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.
4. The user is sent his password by mail.
5. The user is then redirected 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:

Exceptions:

The user at step 1.3 enters an email id that is already used to register by a user in the 'Panther Buddy' system. The user is asked to re-enter a new email id.

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 new 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

Usability: No previous training needed.

Mean time to failure: 1 failure for every 2 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 and the Eclipse framework. He requests us to use PostgreSQL as database provider and Java Server Faces and Bootstrap as frontend.

Modification history:

Owner: Allan Shaji Manamel

Initiation date: 09/07/2015

Date last modified: 09/14/2015

Example Scenario:

User registration

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".

Allan Shaji Manamel will be asked to enter his details.

- First name: Allan
- Last name: Manamel
- Email id: amana010@fiu.edu
- Phone number: 987 654 2134

Trigger: Allan Shaji Manamel clicks the submit button to submit the data.

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.

Allan Manamel sent his password (D43sx21) by email.

The user is then redirected to the login page.

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

4. USER ACTIVATION

Use Case ID: PB_UC08_ACTIVATE_USER

Use Case Level: System Level

Details:

Actor: Registered user

Pre-conditions:

1. The user should be registered on panther buddy

Description: The user will hit "Login" button to activate his/her account on panther buddy after entering his login credentials.

Trigger: The user clicks the "Login" button on the main page after entering his login credentials.

The system responds by:

1. The system authenticates the username and password
2. The system marks the user as activated
3. The user gains access to the systems functionality.

Post-conditions:

1. The user status is changed to active in the Panther Buddy system.

Alternative Courses of Action:

Extensions: None.

Exceptions: Incorrect login credentials will not activate the user.

Related Use Cases: PB_UC_20_LOGIN, PB_UC01_USER_REGISTRATION

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:

User will be activate in 2 seconds.

Supportability

The application will rely on HTML 5.

Implementation

Implementation will be done in HTML 5, 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.

Description: John needs to login himself on "Panther Buddy".

The John will be asked to enter his details.

- User Id: John@fiu.edu
- Password: Manamel0012

Trigger: The user clicks the Login button to Login.

Post-conditions: John's profile is activated and John is directed to home page.

5. POST MESSAGE

Use Case ID: PB _Posting message

Use Case Level: System Level (end to end)

Details:

Actor: Registered user

Pre-conditions: The user must be logged-in and the post message page should be opened in a web browser.

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

1. The user writes his message to be posted in the space provided.
2. The user clicks the add post button to submit his message

Post-conditions: The post will be saved in the system.

Alternative Courses of Action:

At steps 1 the user can hit "Cancel" button to cancel post.

Extensions:

Exceptions: The message will be rejected if it contains illegal characters or keywords or their combination.

Related Use Cases: PB_SC2_XSS_FILTERING

Decision Support

Frequency: Medium - User

Criticality: High

Risk: Medium - attacker may try to attack using XSS

Constraints:

Usability: No training is needed.

Reliability:

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

Performance:

The file or text should be saved along with the post.

Supportability:

The software will support all browsers that support HTML 5.

Implementation:

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

Modification History:

Owner: Abhinav Dutt

Initiation date: 09/06/2015

Date last modified: 09/06/2015

Example Scenario:

Post message

Actors: Kedari Sharat

Pre-conditions: Kedari Sharat must be logged-in and the post message page should be opened in a web browser.

Description:

1. Kedari Sharat writes his message to be posted in the space provided.
2. Kedari Sharat clicks the add post button to submit his message

Post conditions: The post will be saved in the system.

6. 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.

1. User clicks the change password button on his profile page
2. 3 fields appear on the page. They are:
   a) Current Password.
   b) New Password
   c) Confirm Password
3. The user enters the data asked
   3.1 User enters his current password in the current password field
   3.2 User enters his new desired password in the new password field.
   3.3 User re-enters his desired new password in the confirm password field.
4. The user clicks the submit button to submit his data.
5. The 3 Fields for password disappear.

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

Alternative courses of action:

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

Extensions:

Exceptions:

1. The user at step 3.1 enters a wrong password. He is asked to correct his entered password.
2. The user at step 3.2 enters a password that does not fulfil the password criteria for 'Panther Buddy'. He is asked to correct his entered password.
3. The user at step 3.3 enters a password that does not match the password entered in step 3.2. He is asked to correct his entered password.

Related use cases:

Decision support:

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

Criticality: Moderate

Risk: High

Usability: No previous training needed.

Reliability:

Mean time to failure: 1 failure for every 6 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 PostgreSQL as database provider and Java Server Faces 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 is at his own profile page open on 'Panther Buddy'.

Allan wants to change his password.

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:
   a) Current Password.
   b) New Password
   c) 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.

7. PASSWORD RECOVERY

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.

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:

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 id.

Related use cases:

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

Usability: No previous training needed.

Mean time to failure: 1 failure for every 2 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 PostgreSQL as database provider and Java Server Faces 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: The user has the login page open.

Allan forgets his password.

3. Allan opens the 'Panther Buddy' login page on his laptop browser that supports HTML 5.
4. Allan clicks the recover password button.
5. Allan is redirected to the recover password page.
6. Allan enters his email id: amana100@fiu.edu used to register into the system.
7. Allan clicks the submit button to submit the data.
8. Allan's password: DX56ssUT is recovered by the system and sent to his email account.

Post condition: Allan Shaji Manamel gets his password on his registered email id: amana100@fiu.edu.

8. 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 has privileges to do so

. Trigger: The user clicks delete post button near the message.

The system responds by

3. A window will appear.
4. The user needed to confirm wither he want to delete the post .
5. 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 and 3, the user can hit “Cancel” to cancel deletion.

Extensions: none.

Exceptions: The post need to be owned by user.

Related Use Cases: log-in, Post messages.

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 file or text should be saved along with the post.

Supportability

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

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 has privileges to do so.

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

The system responds by:

1. A window will appear.
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

9. 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.

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 any time during step 1 to 4, the user can click the cancel button to return to the home page.

Extensions:

Exceptions:

The user at step 1 puts malicious code in his message. The system shows an error for in proper content at asks user to rectify it.

Related use cases:

Decision support

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

Criticality: Very

Risk: High

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 PostgreSQL as database provider and Java Server Faces 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.

Allan submits his message.

1. Allan logs into the 'Panther Buddy' system and opens the post message page on his laptop browser that supports HTML 5.
2. Allan writes his message "Used book - Introduction to Algorithms by Thomas Cormen up for sale at 19$".
3. Allan clicks the submit button to submit his message.
4. The system checks if the message contains any malicious code or not.
5. The system verifies the message to be ok.
6. 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.

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 unauthorised data from the system.

Criticality: Very

Risk: High

10. 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.

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:

Exceptions:

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

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 PostgreSQL as database provider and Java Server Faces 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: The user has the login page open.

1. Allan forgets his password.
2. Allan opens the 'Panther Buddy' login page on his laptop browser that supports HTML 5.
3. Allan enters his email amana100@fiu.edu into the email field.
4. Allan enters his password DX345tv in the password field.
5. Allan clicks the login button to submit the data.
6. The system checks the input email amana100@fiu.edu and password DX345tv for malicious content

Post condition: The user 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.

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.
4. 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

11. VIEW POST

Use Case ID: PB_UC_16_VIEW_POST

Use Case Level: System leve (end to end)

Details:

Actor: Registered User

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

Description

1. The user clicks the home page button on the screen he is at.
2. User is redirected to the homepage where he can view a list of posts ordered by most recent at top.

Post-conditions: The user can view the already posted messages.

Alternative Courses of Action:

Extensions:

Exceptions: The system might not be able to display a longer version of the message.

Related Use Cases: PB_UC15_SEARCH_MESSAGE

Decision Support

Frequency: More than 30 times in an hour

Criticality: Moderate

Risk: Moderate

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 JSF

Modification History:

Owner: James W. Angelo

Initiation date: 09/22/2015

Date last modified: 09/18/2015

Example scenario:

Actor: James Angelo

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

Description:

1. The user clicks the home page button on the screen he is at.
2. User is redirected to the homepage where he can view a list of posts ordered by most recent at top.

Post-conditions: The user can view the already posted messages.

12. 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.

Description:

1. The user clicks the name of the user he wants the profile of.
2. The user is redirected to the profile page for the user

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

Extensions:

Exceptions:

Related use cases: PB_UC03_CHANGE_PASSWORD

Decision support:

Frequency: Moderate - Performed every time a user logs out from the Panther buddy system

Criticality: low

Risk: low

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 profile 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 PostgreSQL as database provider and Java Server Faces 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

Description:

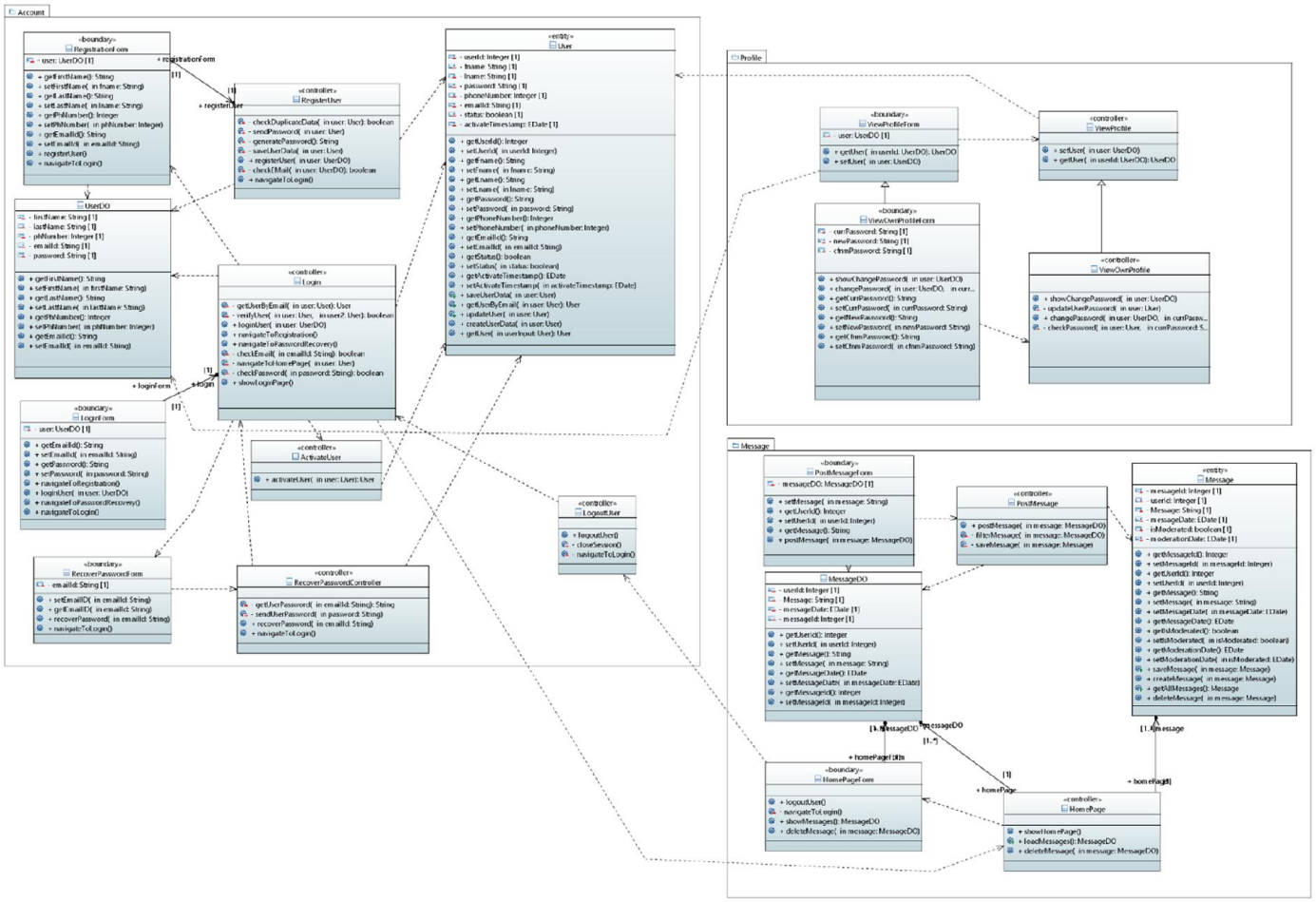
1. Allan clicks on the name of Abhinav, which appears above a post created by Abhinav.
2. Allan is redirected to Abhinav's profile page.

Post condition: Allan Shaji Manamel can view Abhinav's profile.

4.3.2 Static model – object diagrams (one per scenario), class diagram

4.3.2.1 Class Diagram

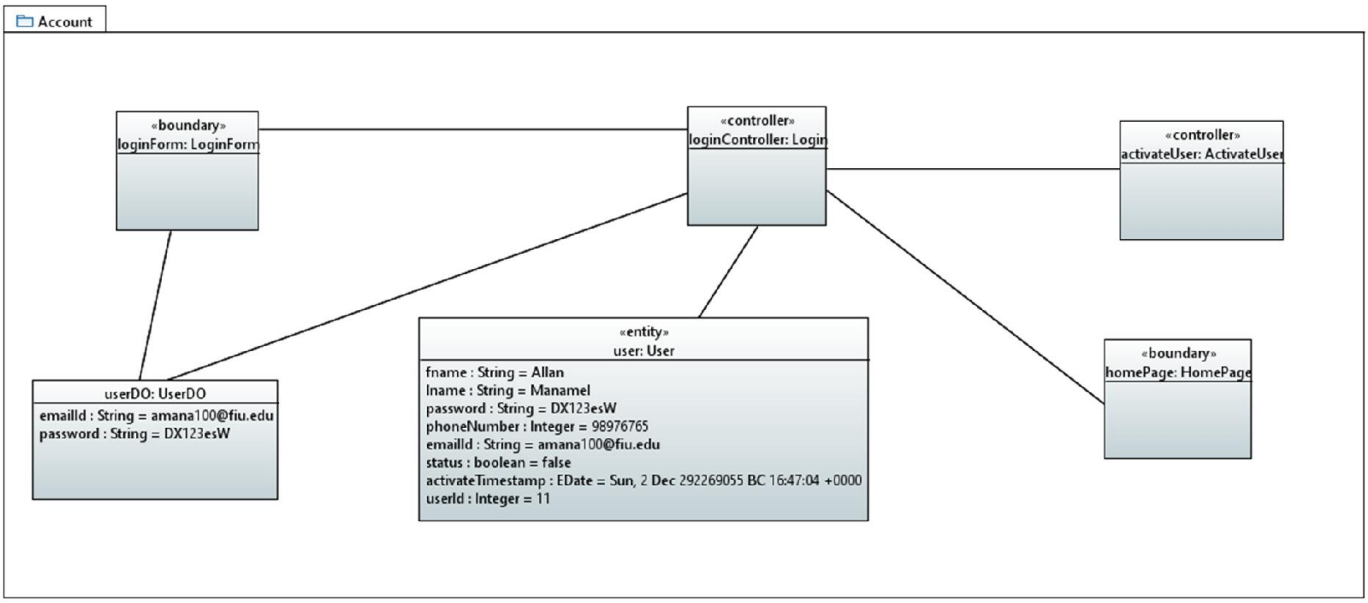
Below is the high level class diagram.



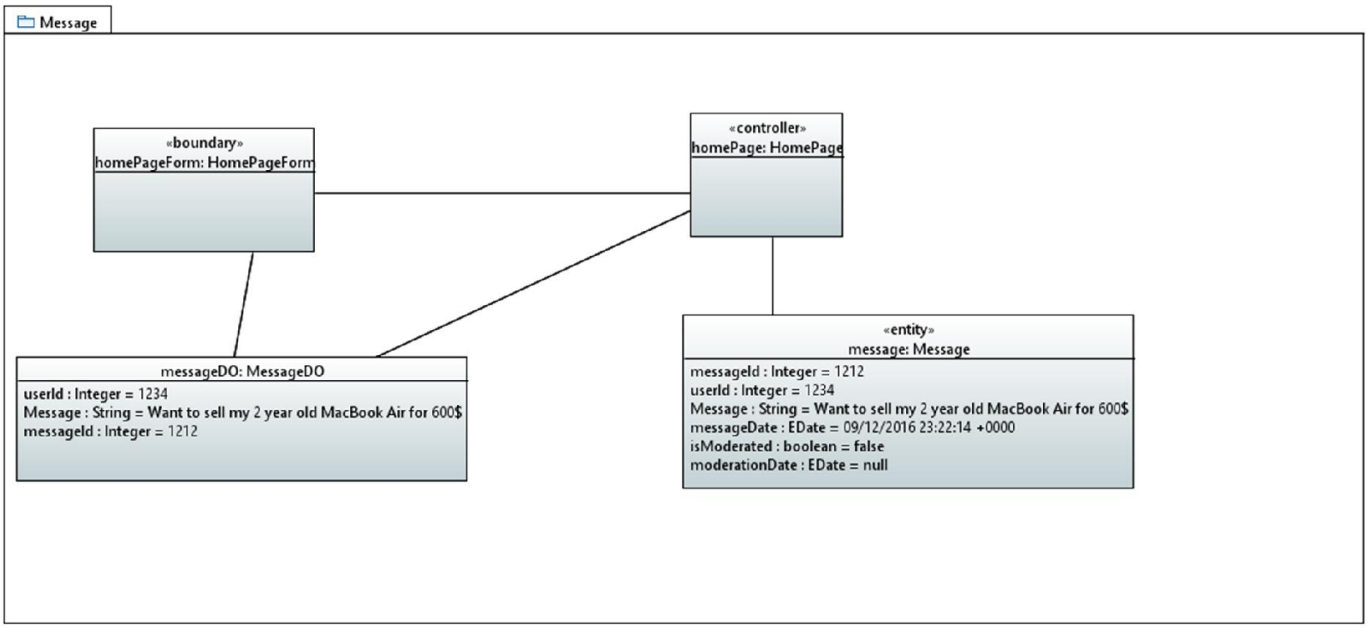
4.3.2.2 Object Diagrams

Below are the object diagrams for the use cases being implemented.

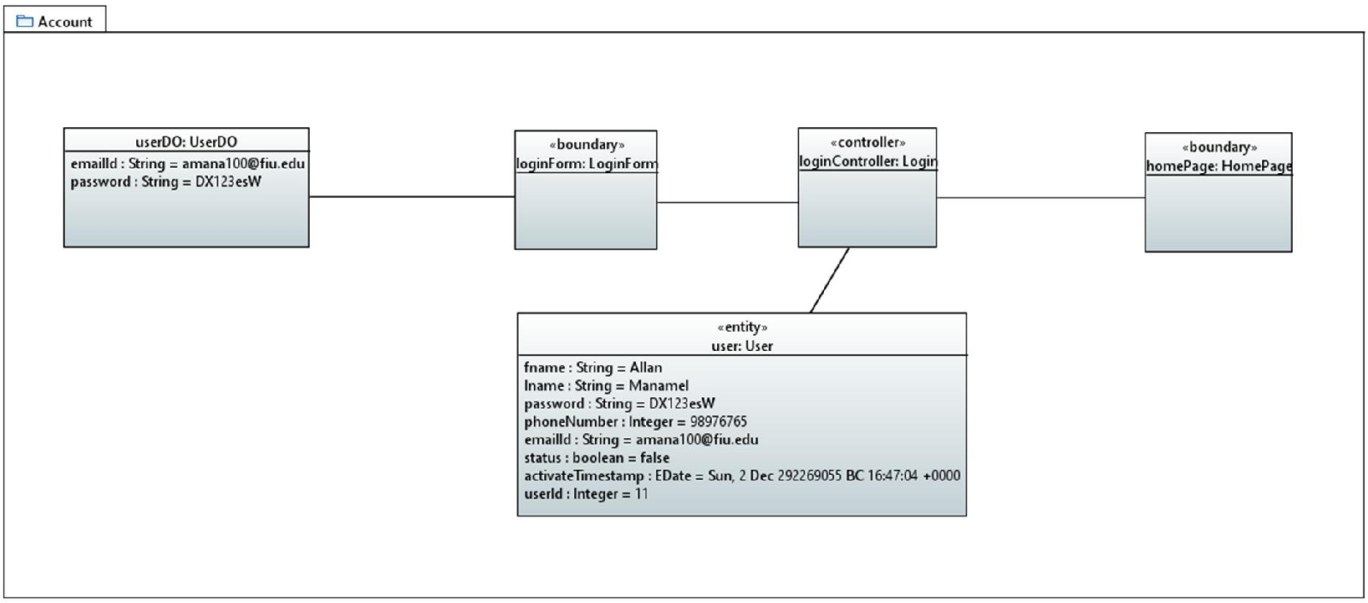
Activate User



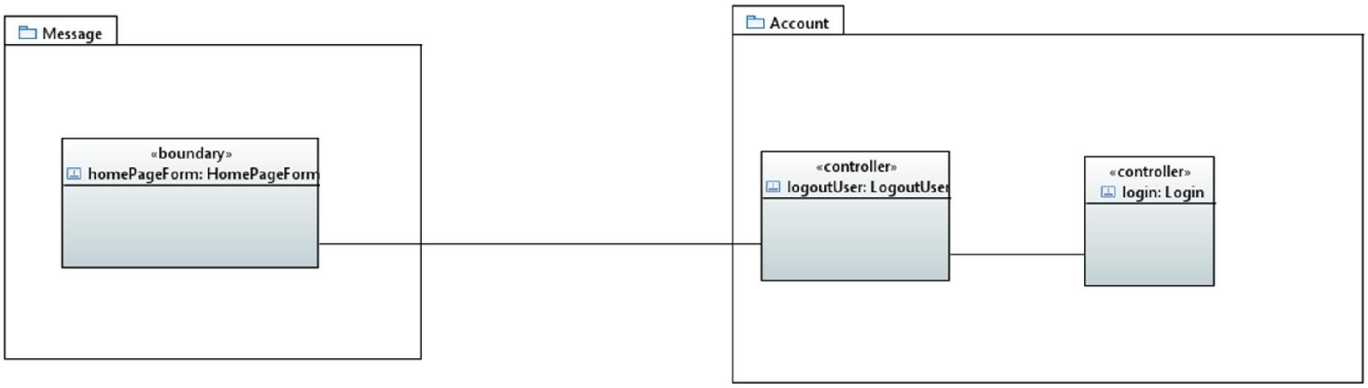
Delete Message



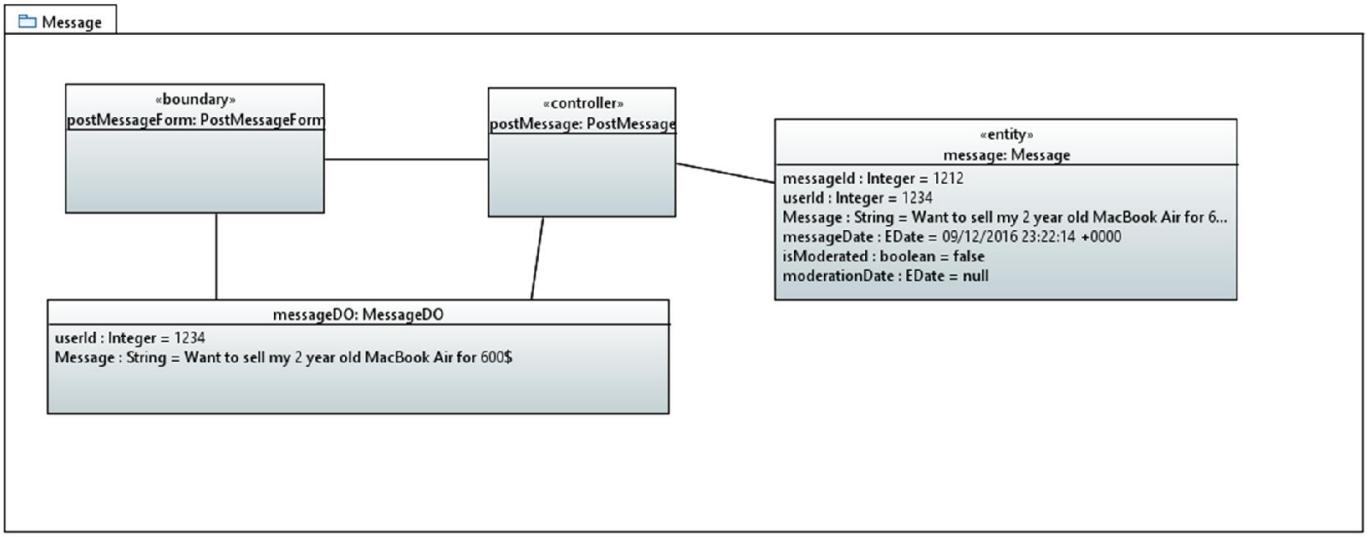
User Login and SQL Injection


Account
userDO: UserDO
emailId : String = amana100@fiu.edu
password : String = DX123esW
«boundary»
loginForm: LoginForm
«controller»
loginController: Login
«boundary»
homePage: HomePage
«entity»
user: User
fname : String = Allan
lname : String = Manamel
password : String = DX123esW
phoneNumber : Integer = 98976765
emailId : String = amana100@fiu.edu
status : boolean = false
activateTimestamp : EDate = Sun, 2 Dec 292269055 BC 16:47:04 +0000
userId : Integer = 11

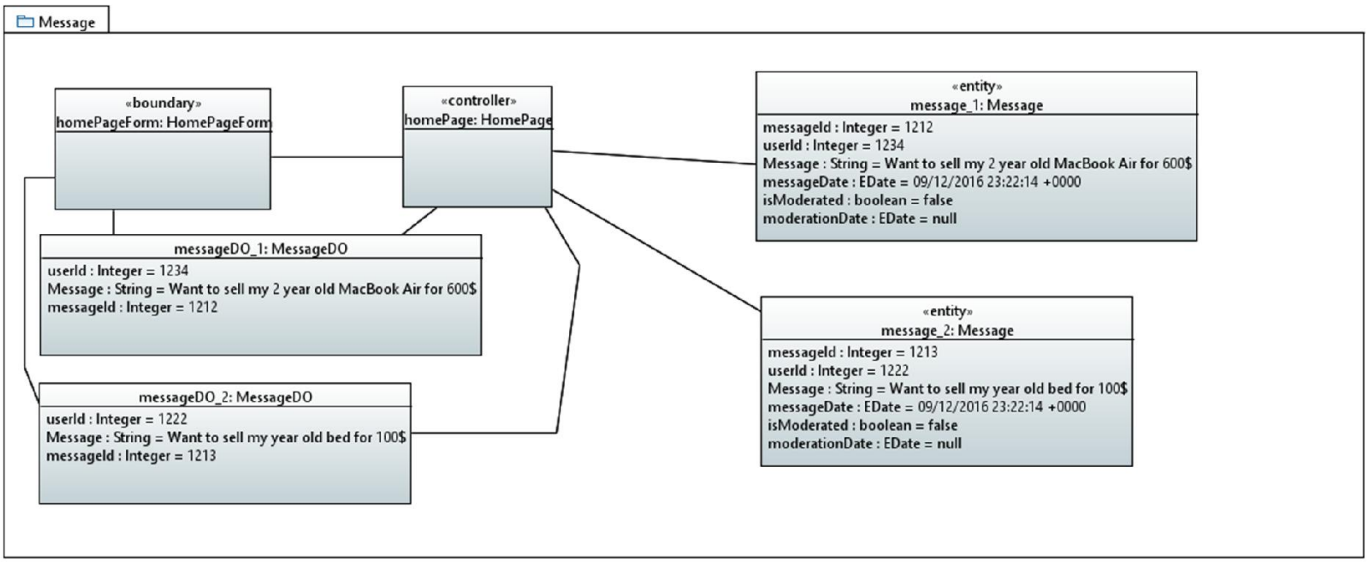
User Logout


Message
«boundary»
homePageForm: HomePageForm
Account
«controller»
logoutUser: LogoutUser
«controller»
login: Login

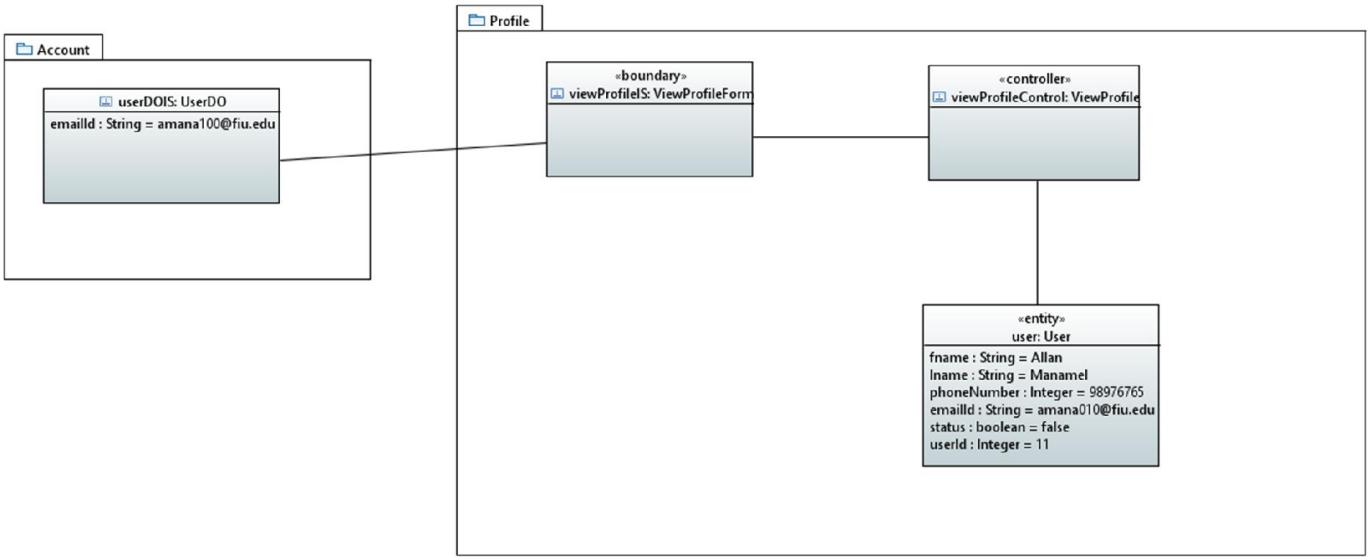
Post Message and XSS Filtering



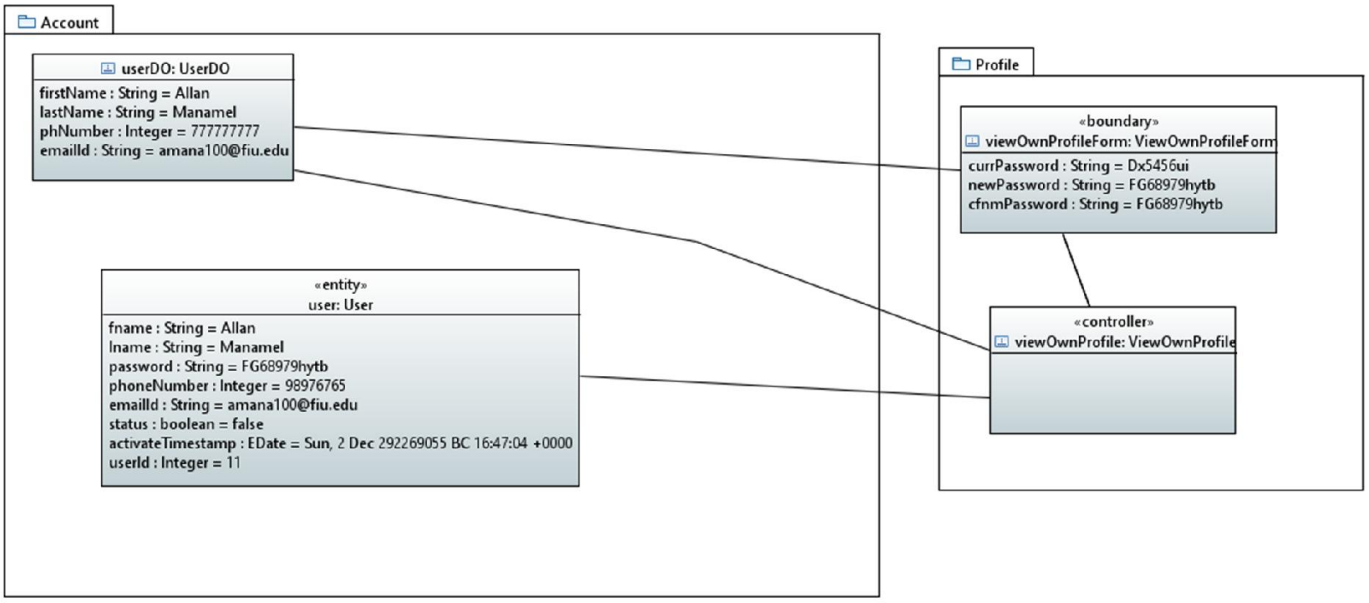
View Post



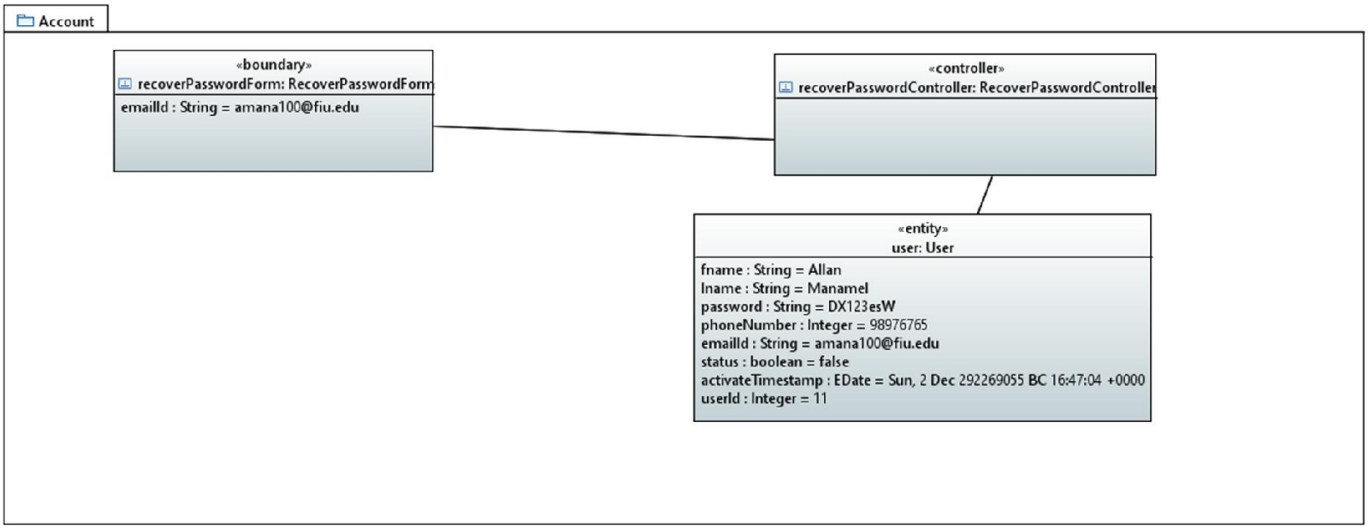
View Profile



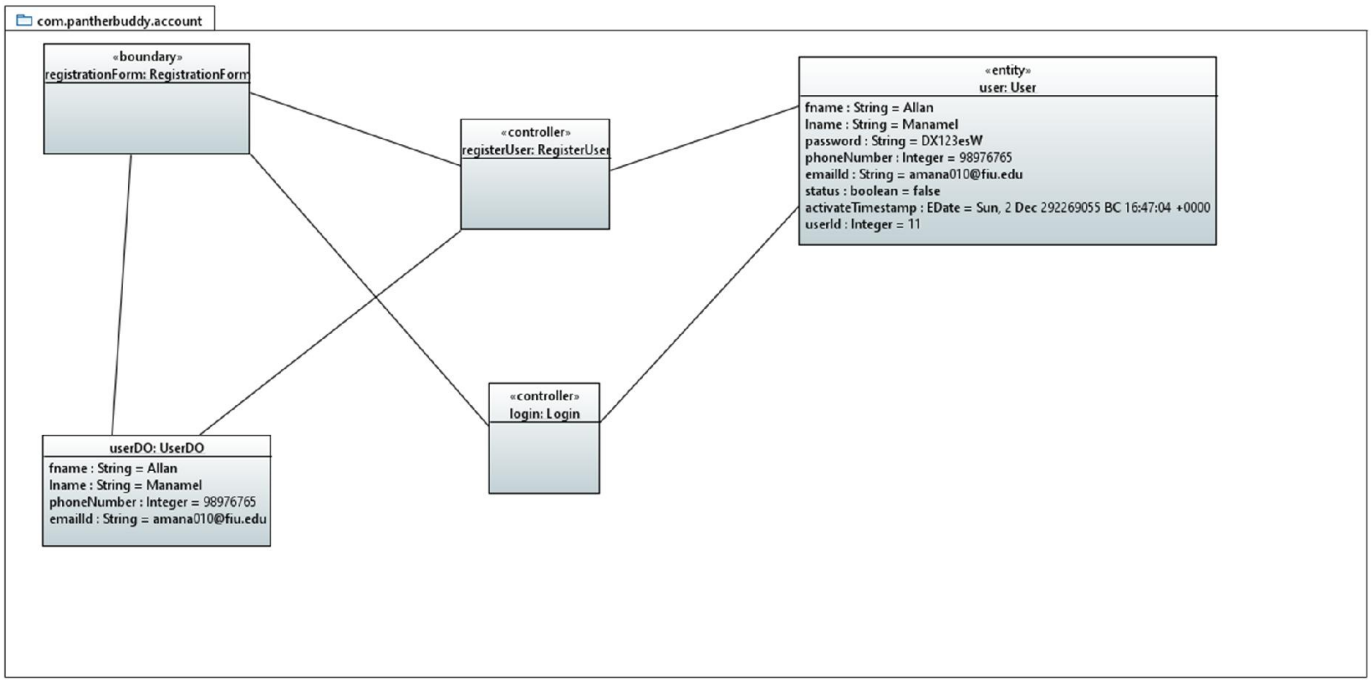
Change Password



Recover Password



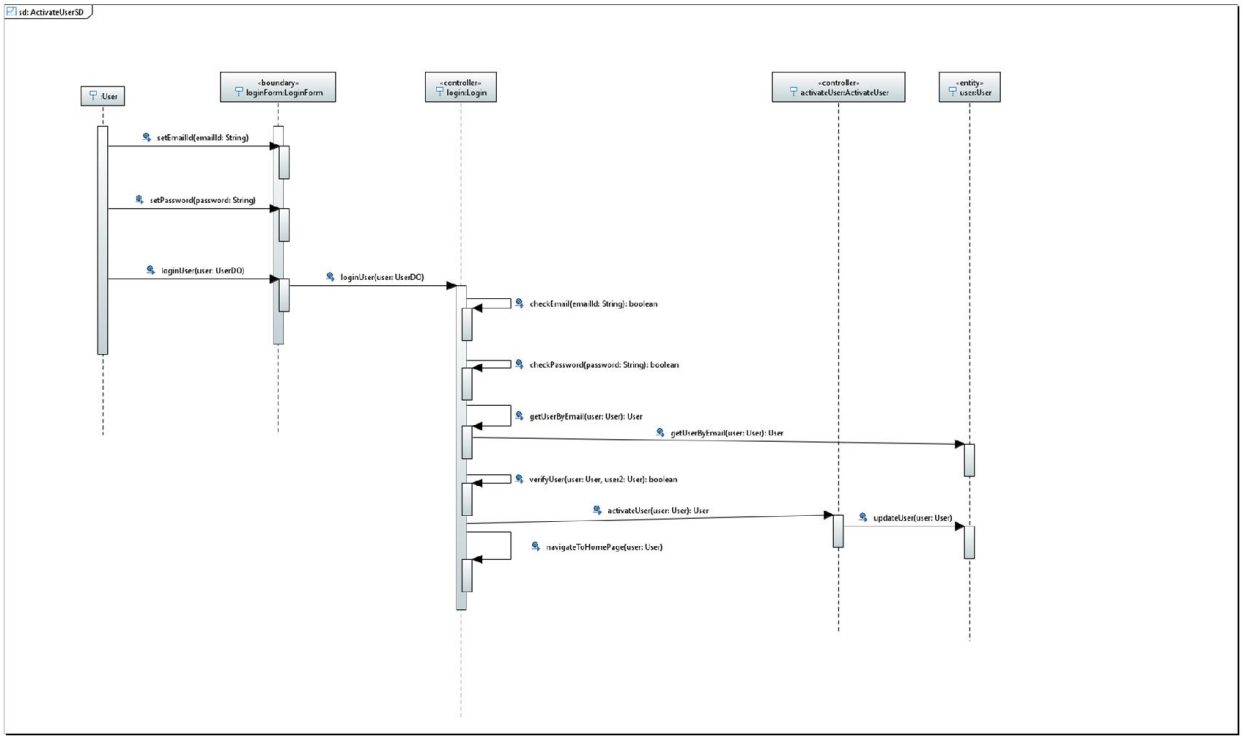
User Registration



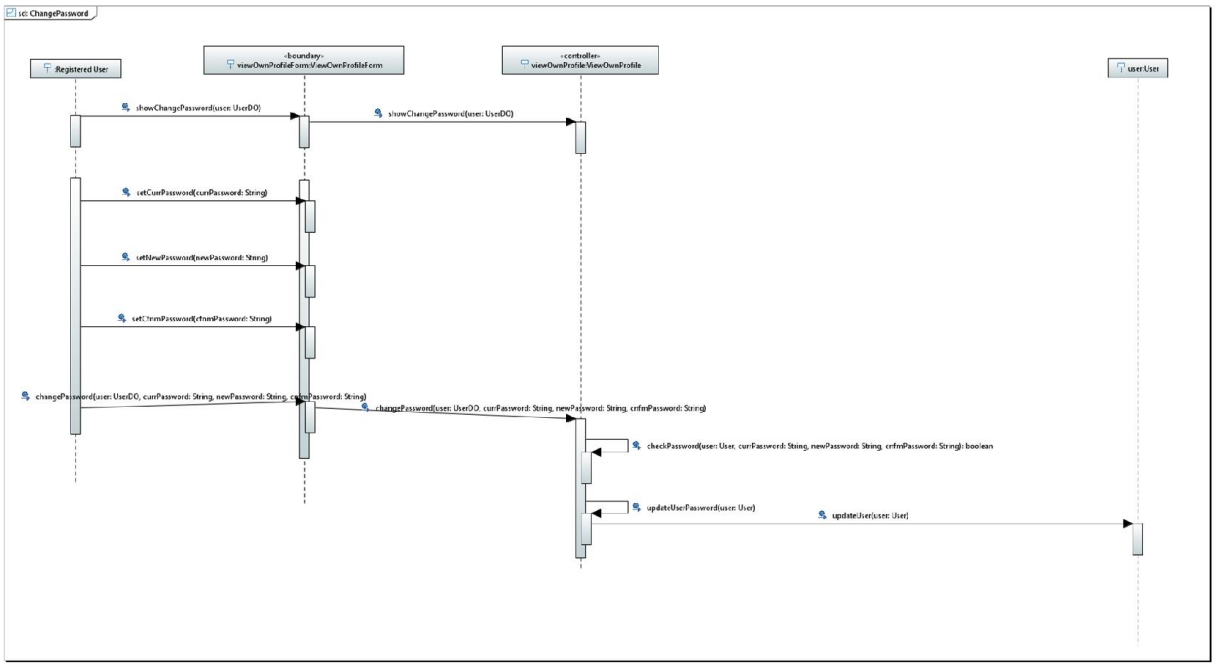
4.3.3 Dynamic model – Sequence diagrams (one per scenario)

Sequence Diagrams.

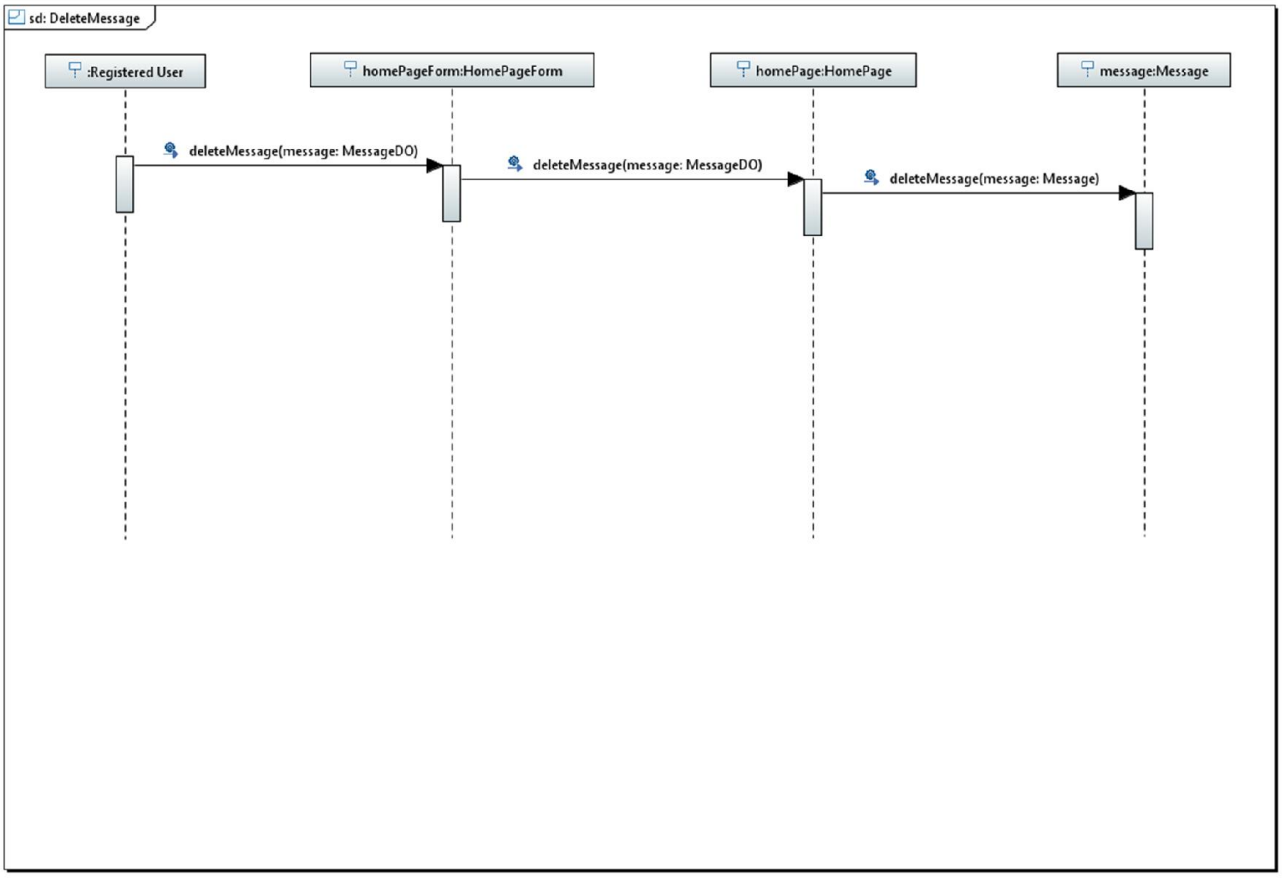
Activate User



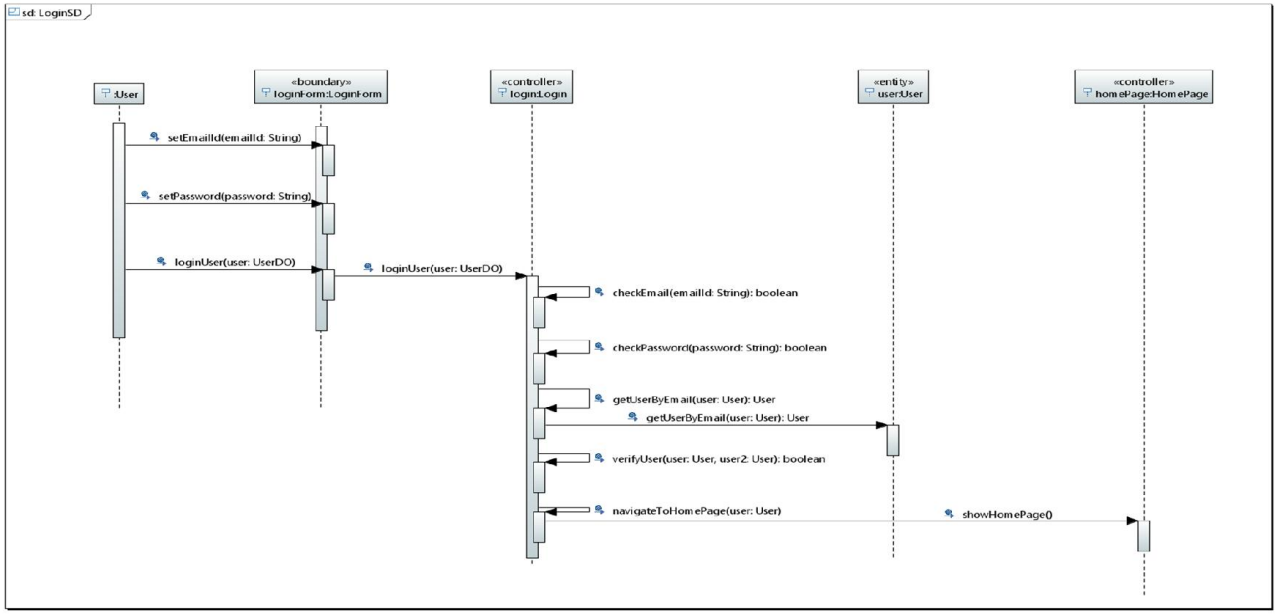
Change Password



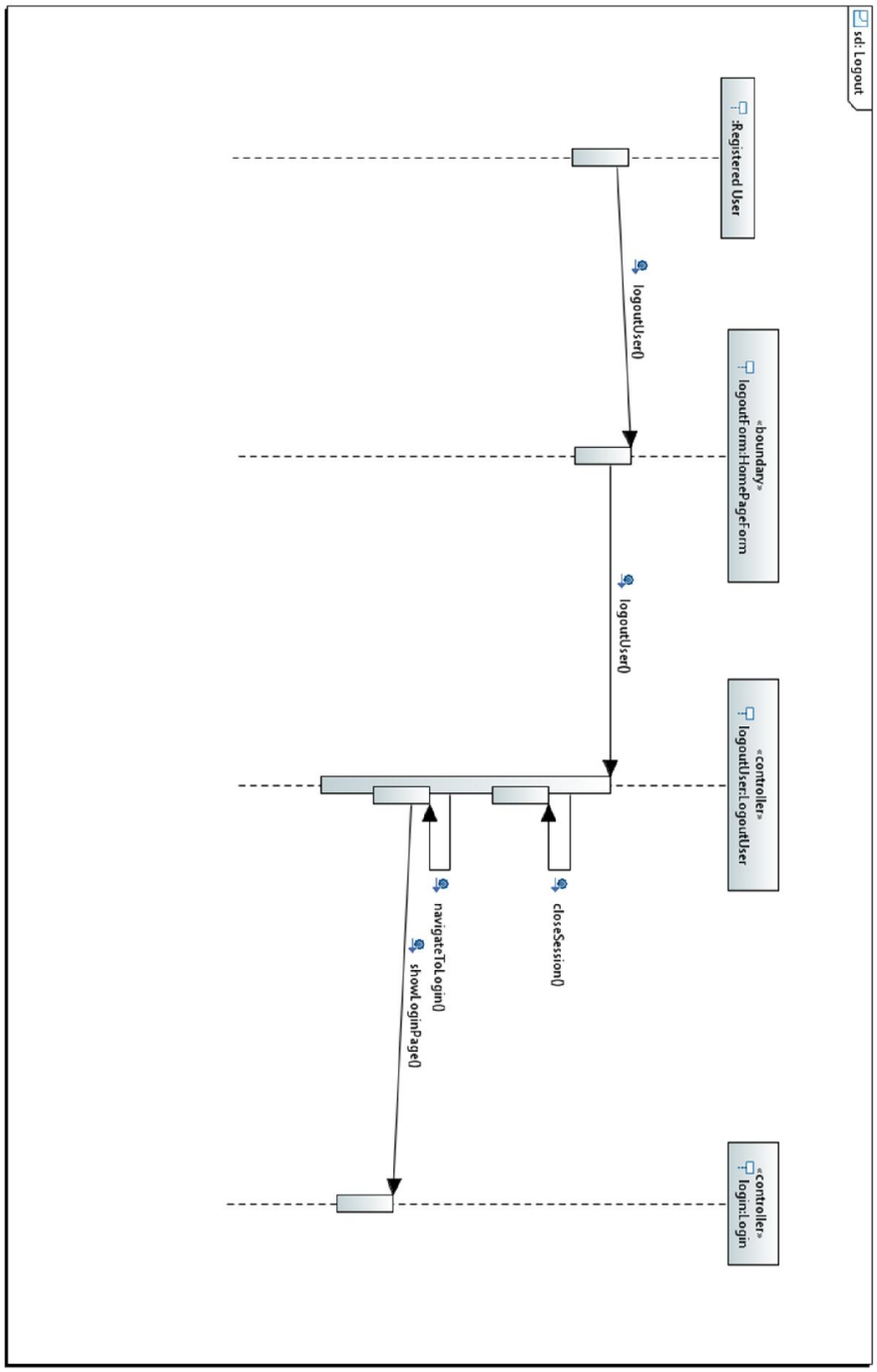
Delete Message



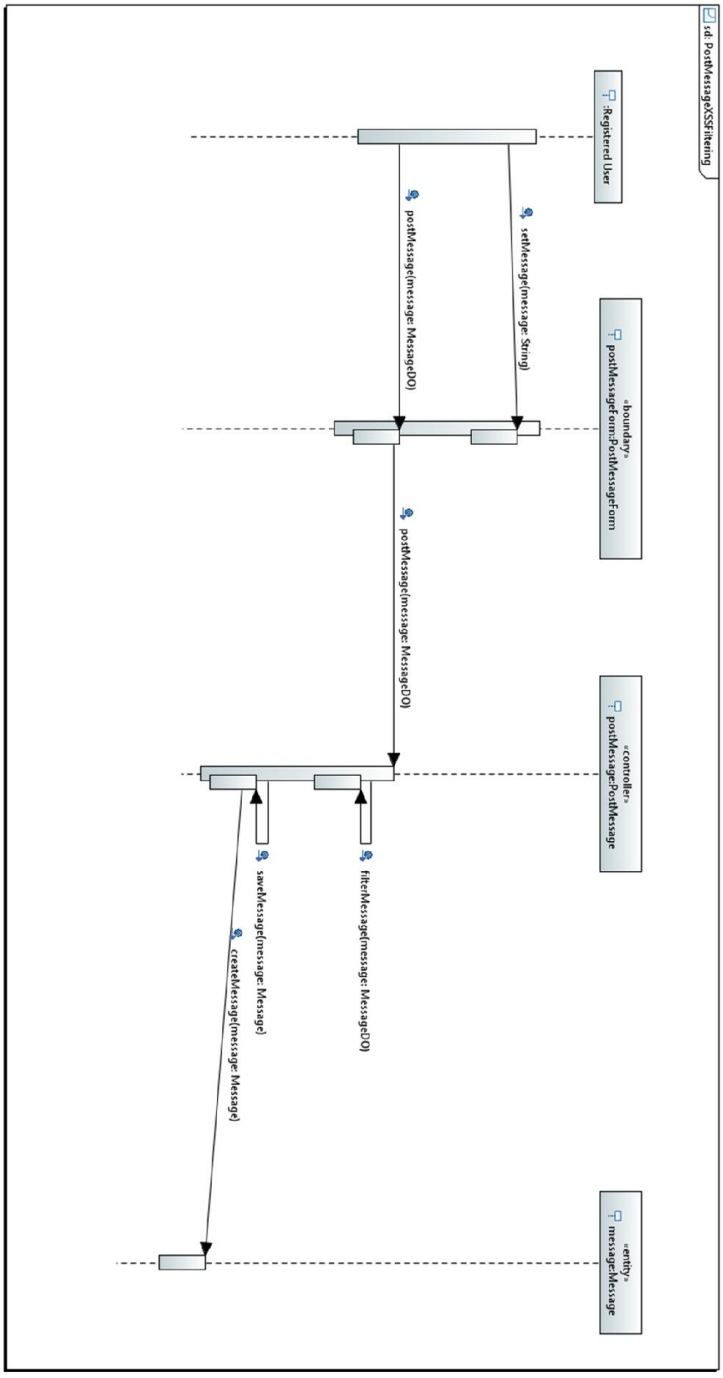
User Login and SQL injection prevention



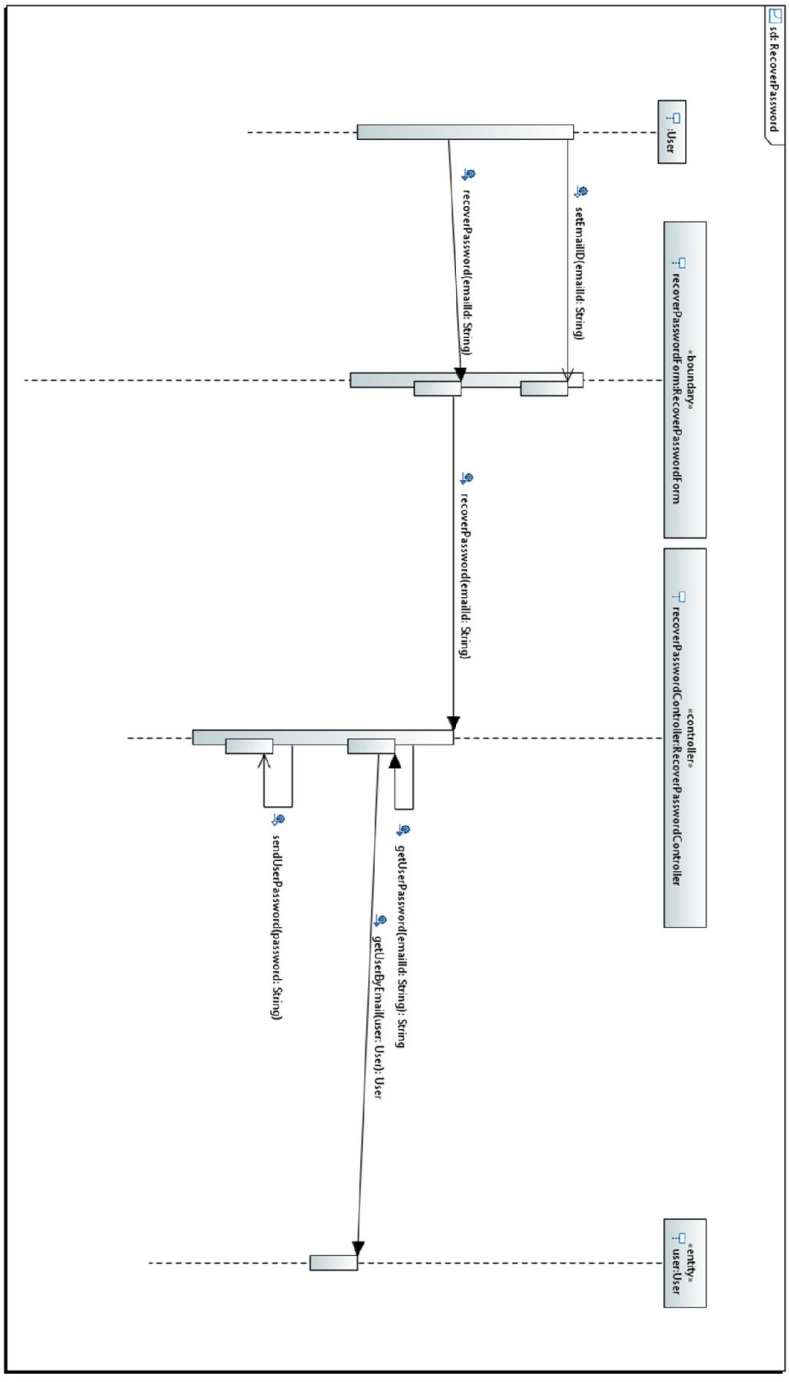
User Logout



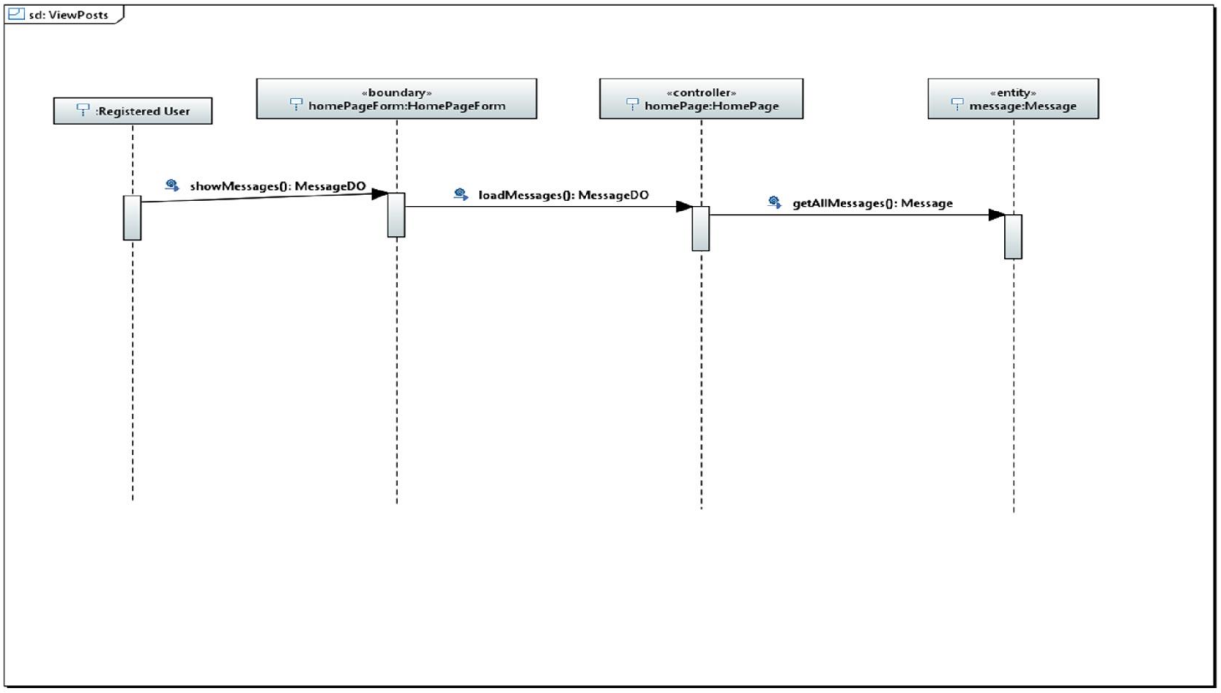
Post Message&XSS Filtering



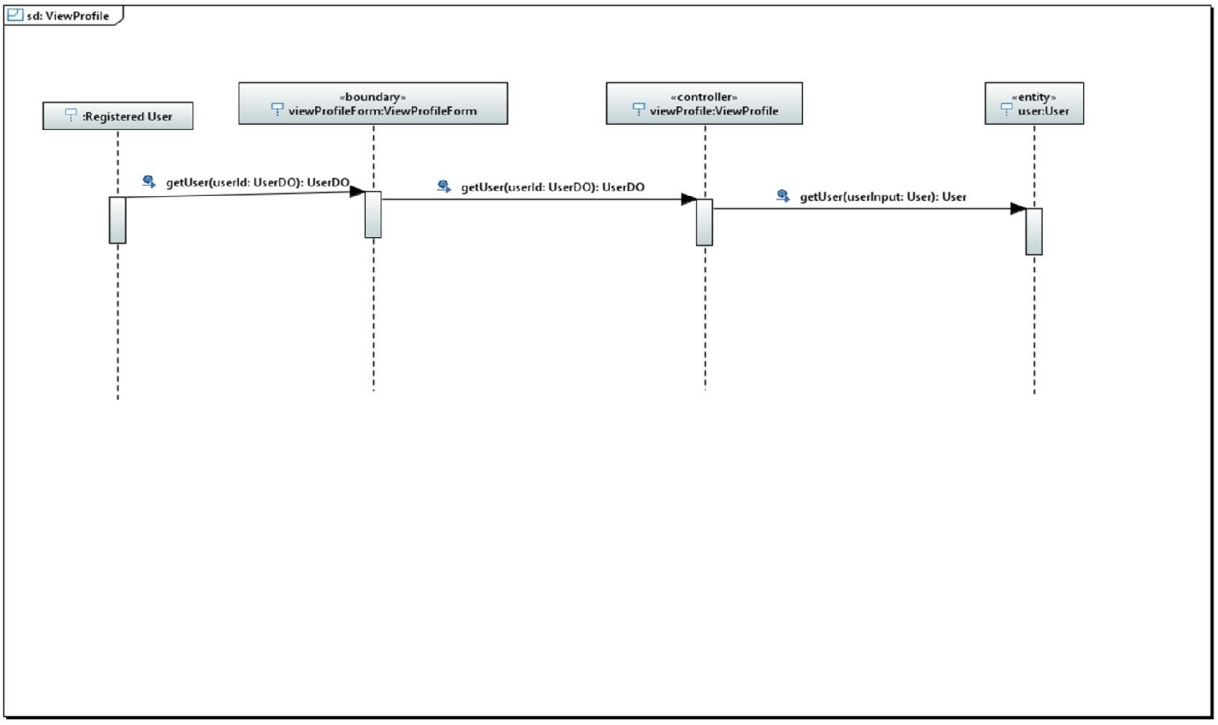
Recover Password


cd: RecoverPassword
: User
recoverPassword(emailId: String)
setEmailID(emailId: String)
«boundary»
recoverPasswordFormRecoverPasswordForm
recoverPassword(emailId: String)
«controller»
recoverPasswordController:RecoverPasswordController
setUserPassword(password: String)
getUserPassword(emailId: String): String
getUserByEmail(user: User): User
«entity»
user:User

View post



View Profile



5. 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.

5.1. 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.

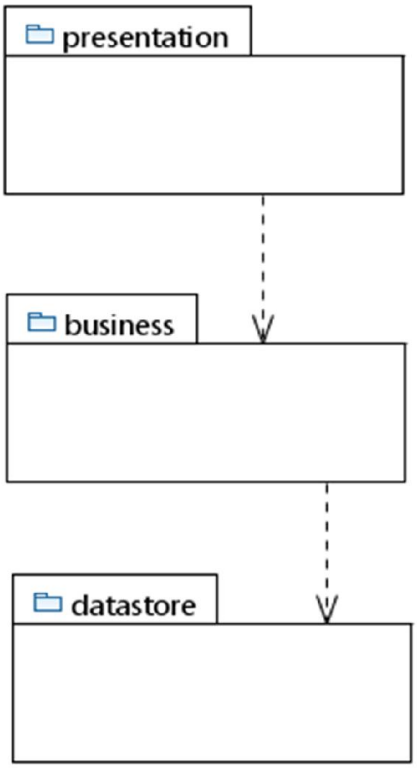


Figure: Represents the High Level 3-Tier Architecture.

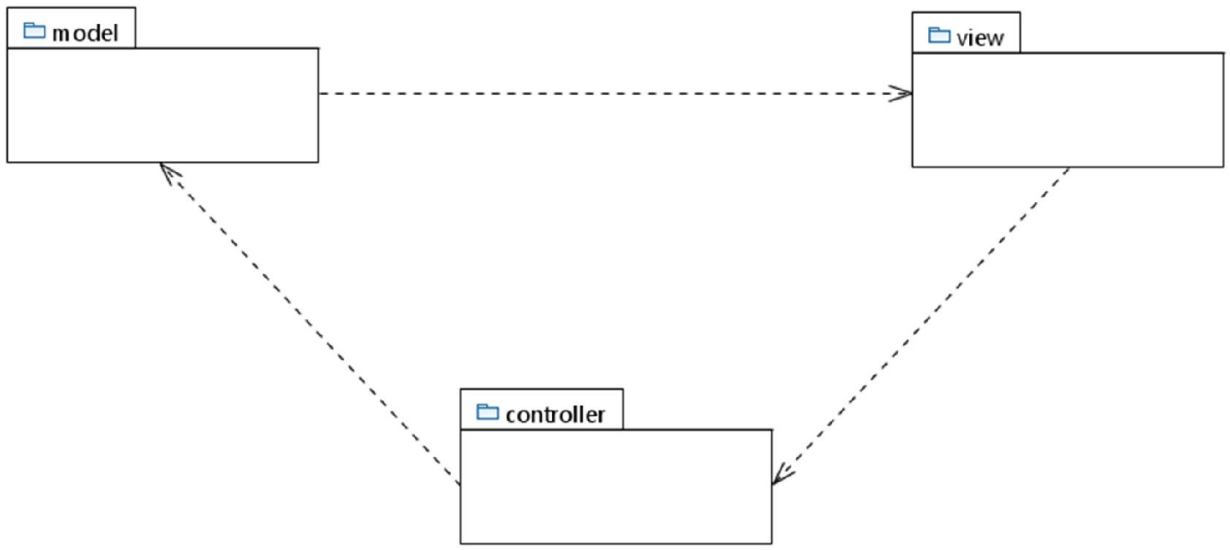


Figure: MVC Architecture

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.

5.2. 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).

5.3. 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

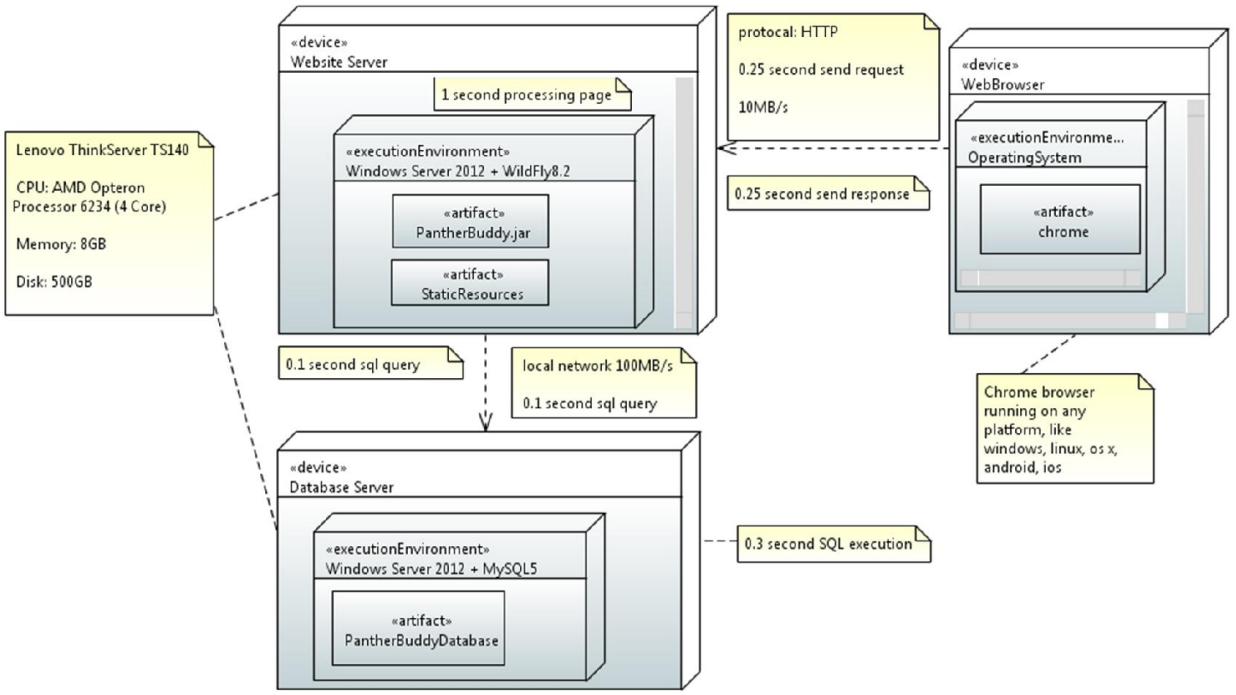


Figure: The figure shows the deployment diagram for the Panther Buddy system

5.4. 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.

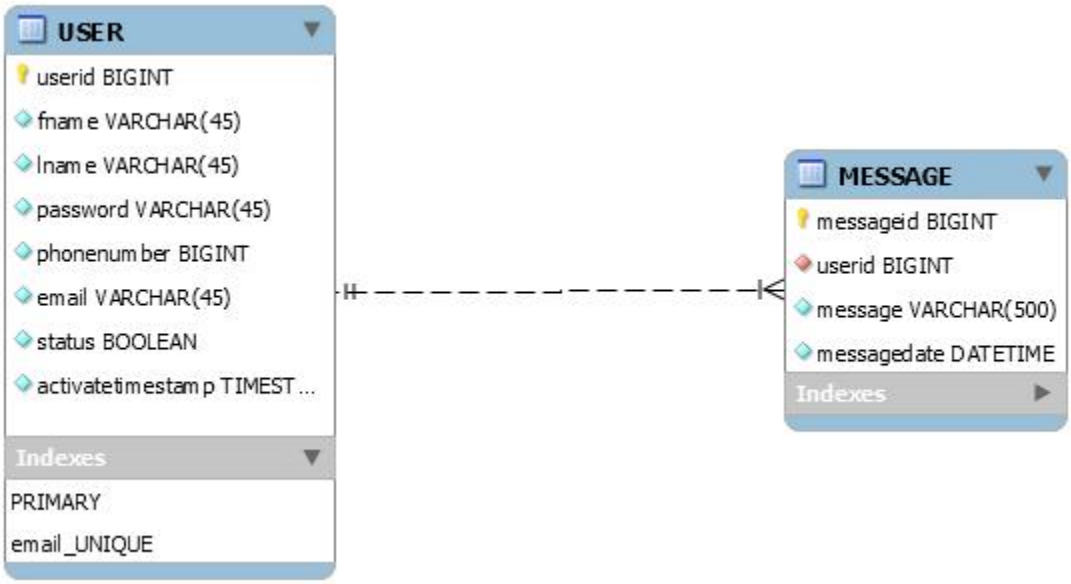


Figure: The ER diagram represents the entities in the database for the Panther buddy system

5.5. 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

6. 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.

6.1 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.

6.1.1 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.

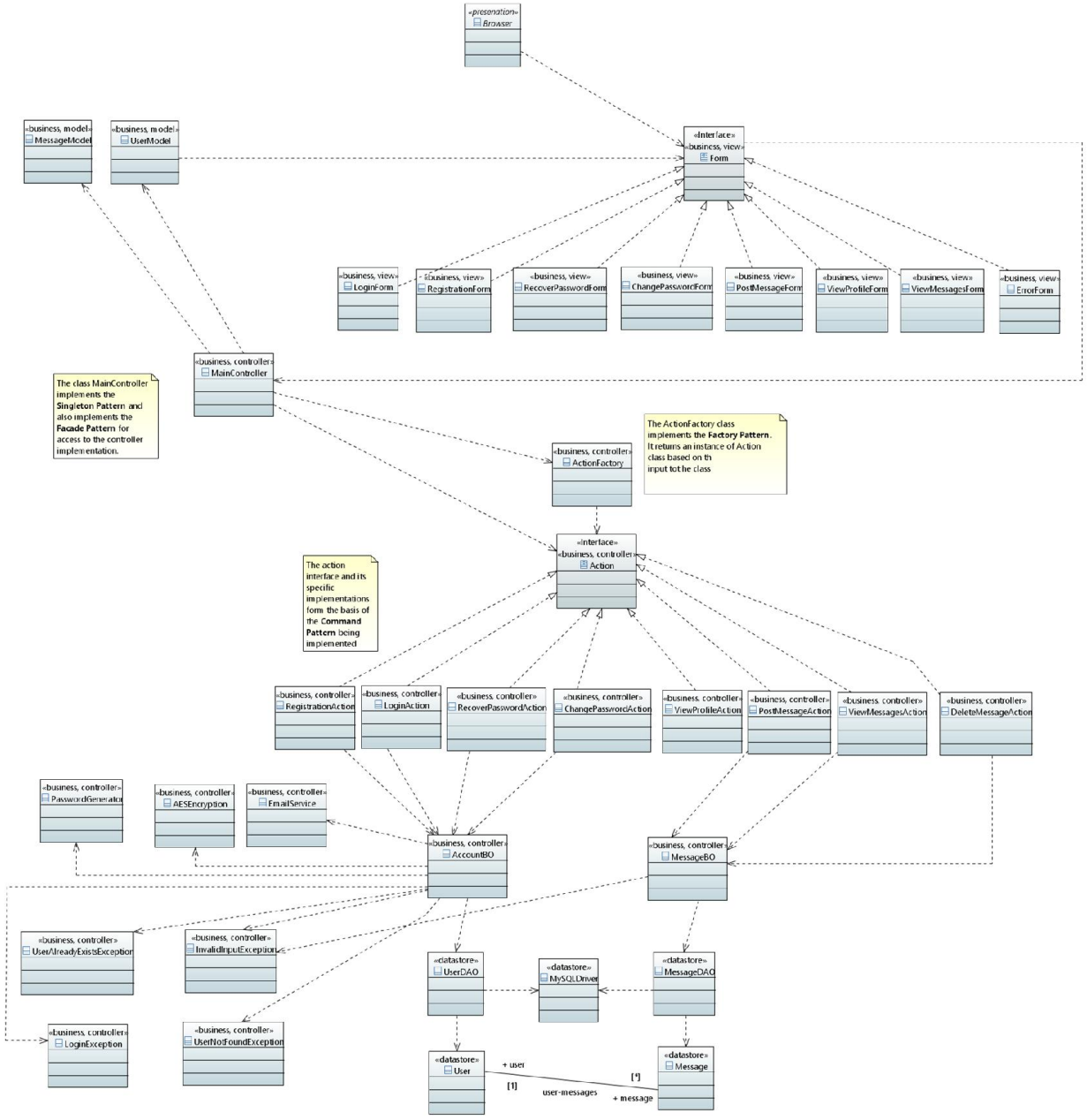


Figure: Minimal Class Diagram

6.1.2 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.

6.1.2.1) 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.

6.1.2.2) 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

6.1.2.2.1) 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.

6.1.2.2.2) 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.

6.1.2.2.3) 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:

6.1.2.2.3.1) Control

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

6.1.2.2.3.2) 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.

6.1.2.2.3.3) Account

Contains the business object classes pertaining to user account.

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

6.1.2.2.3.4) 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.

6.1.2.2.3.5) 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.

6.1.2.2.3.6) 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.

6.1.2.3) 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.

6.1.2.3.1) 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.

6.1.2.3.2) 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.

6.1.3 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.

6.2 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.

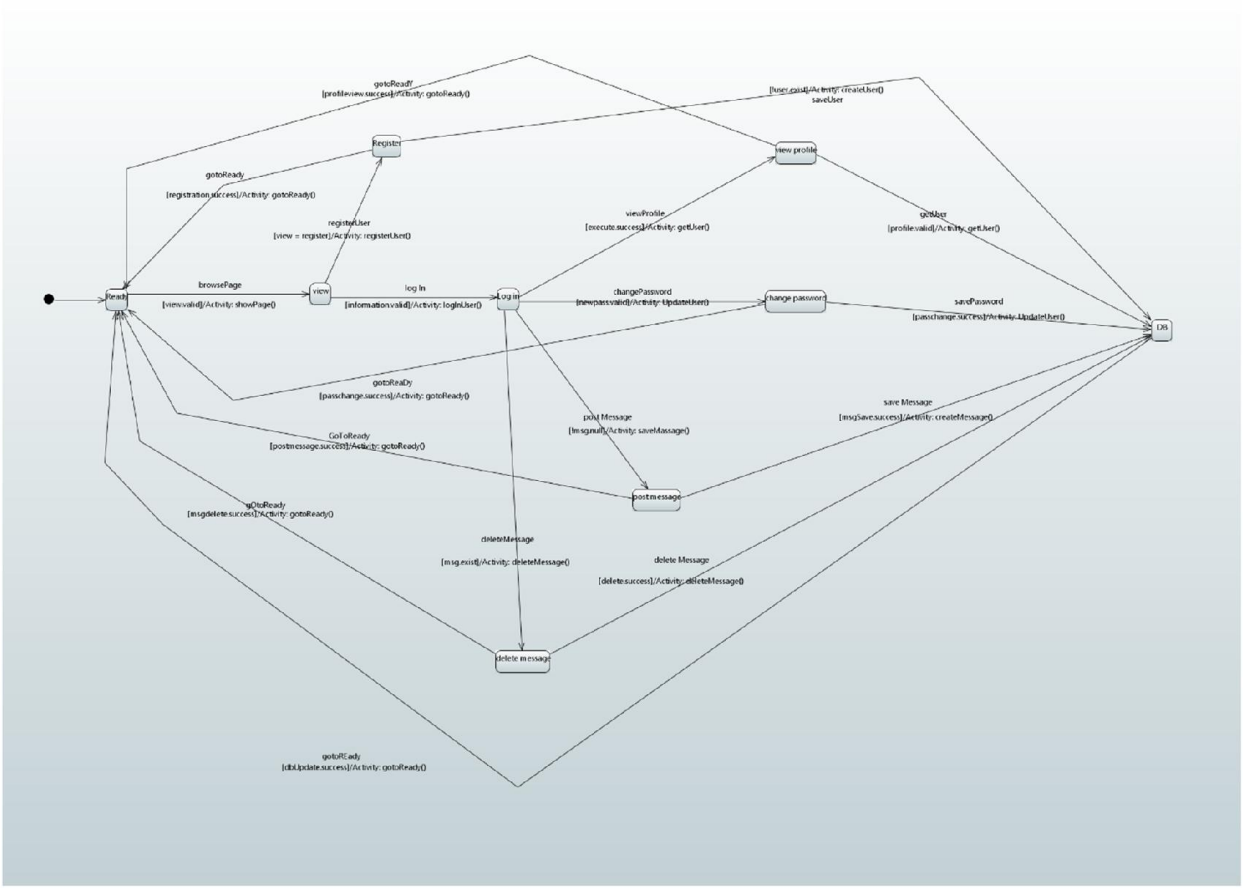


Figure: Shows the overall state machine for the flow in the system.

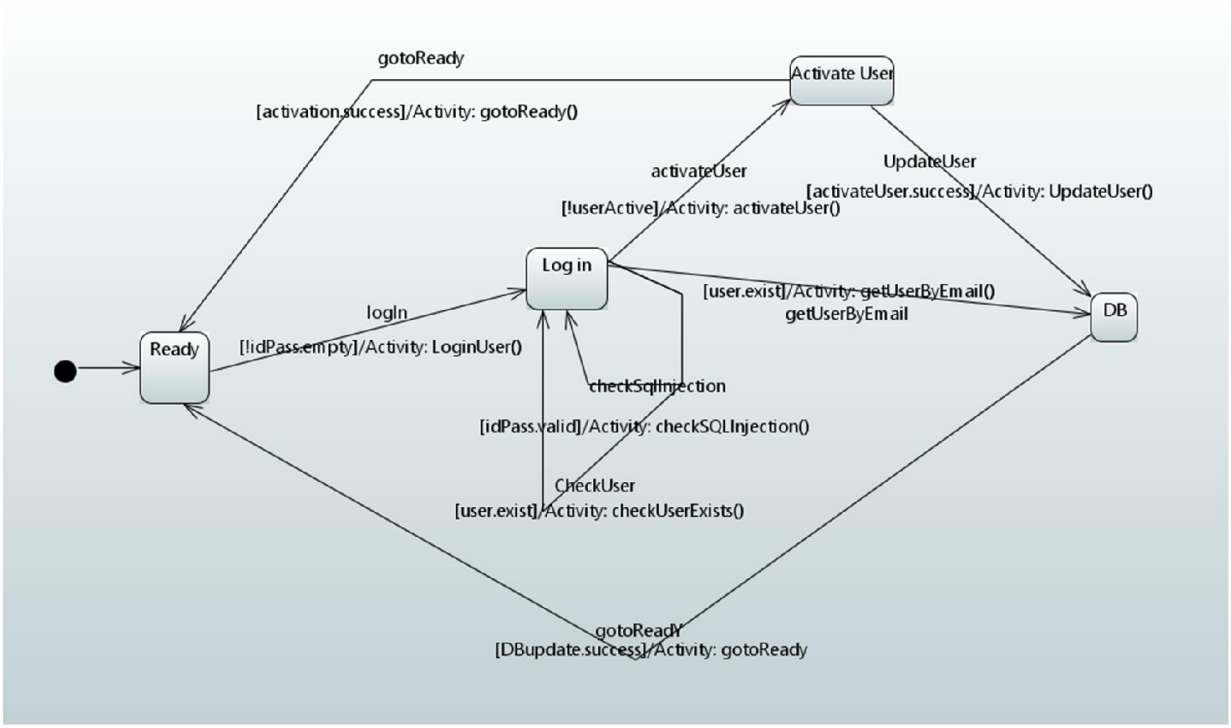


Figure: State machine for Login and Activate user use cases.

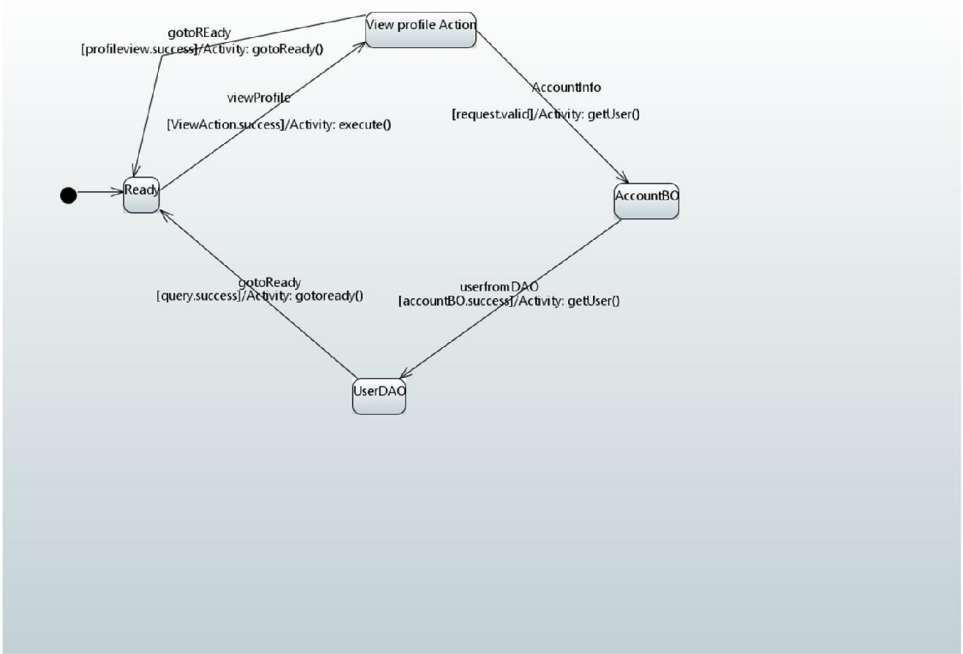


Figure: State machine for View Profile use case.

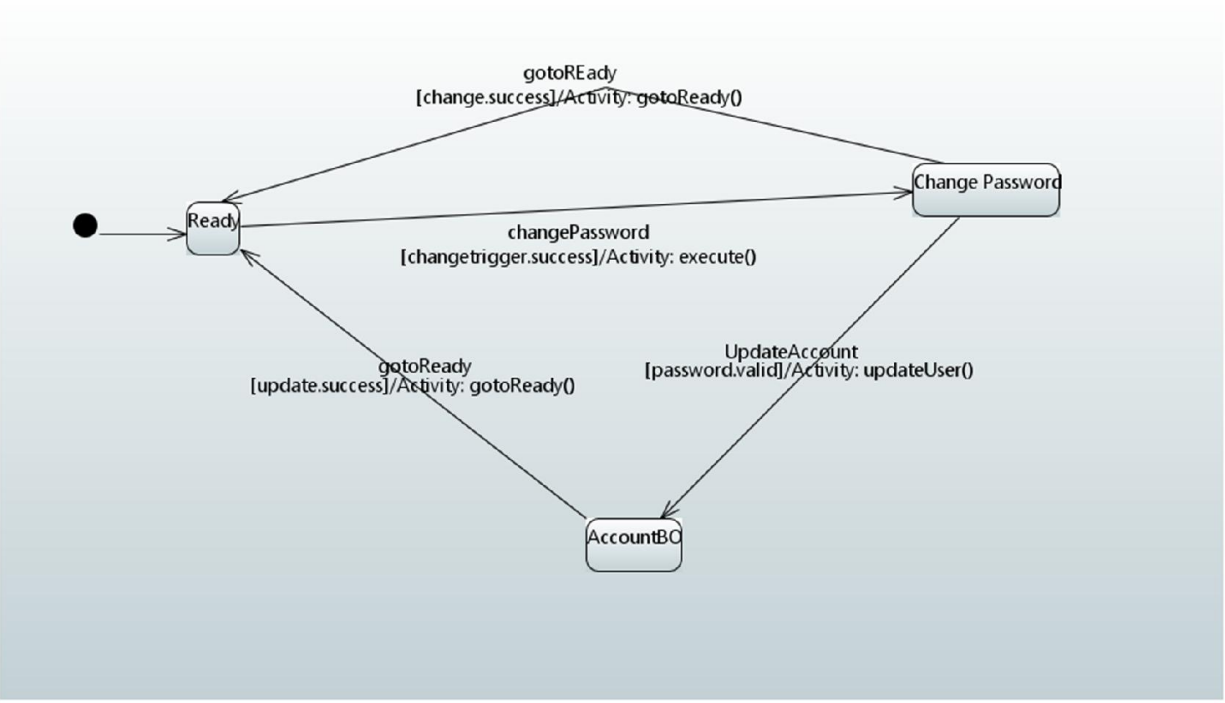


Figure: State machine for Change Password use case

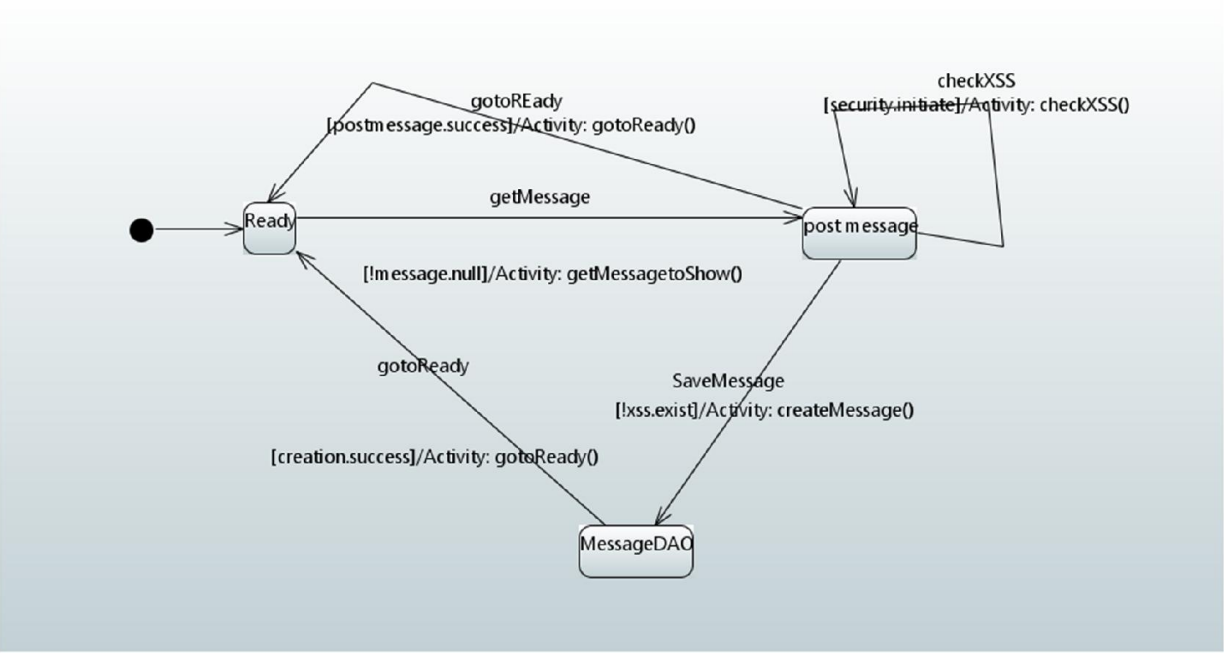


Figure: State machine for Post Message use case

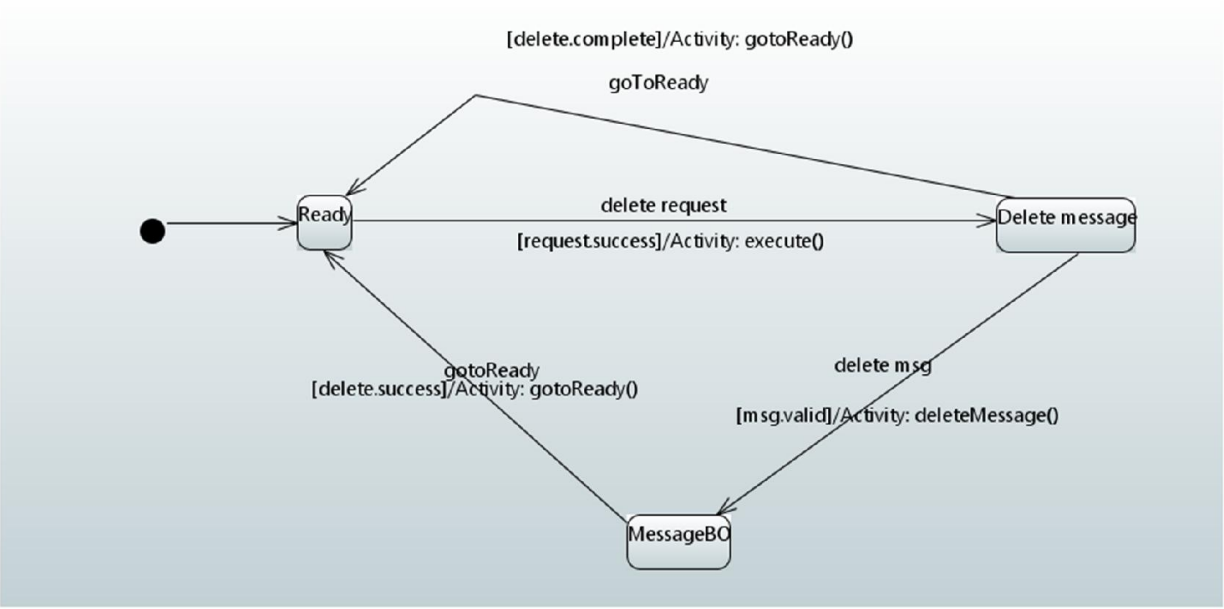


Figure: State Machine for Delete Message use case

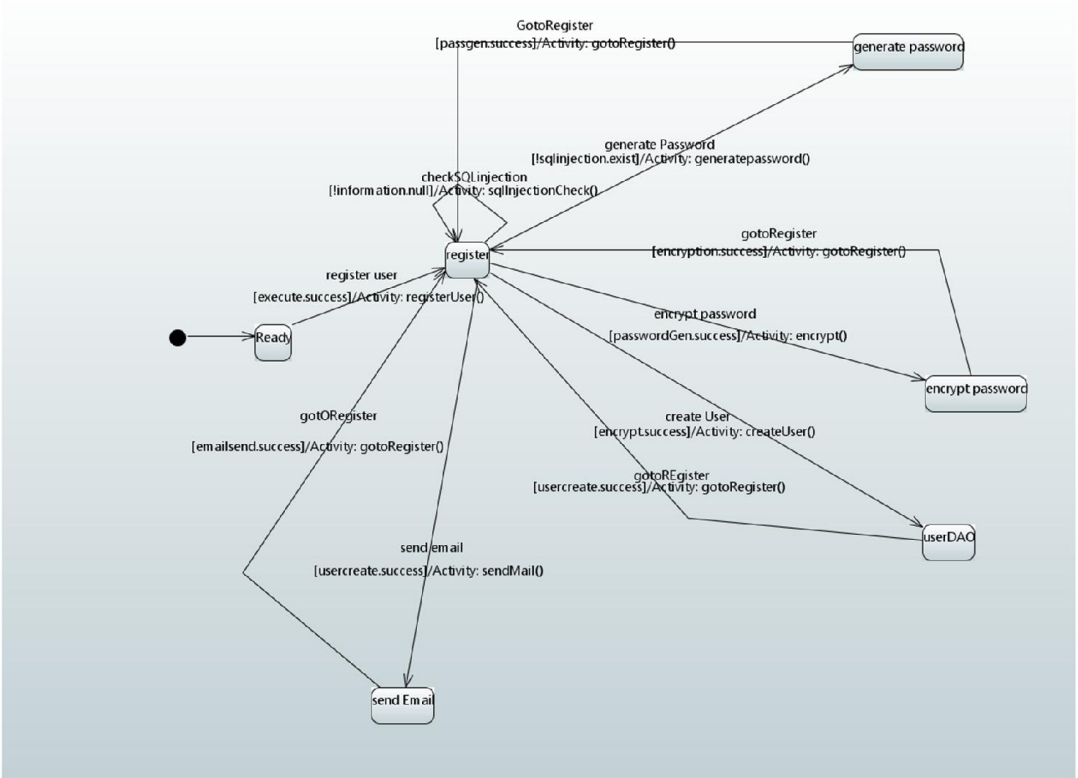


Figure: State Machine for Register User use case

6.3 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.

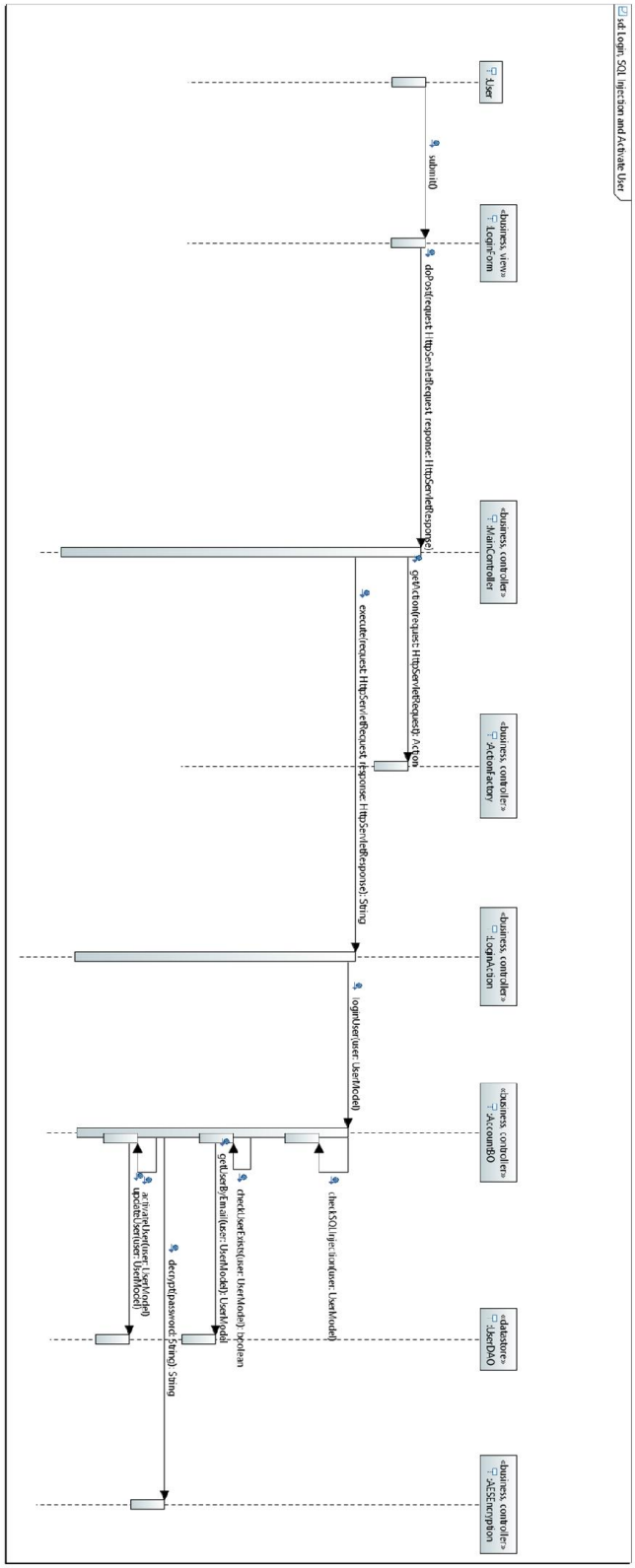


Figure 3.3.1 Login, SQL injection and Active user

- **Logout**

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

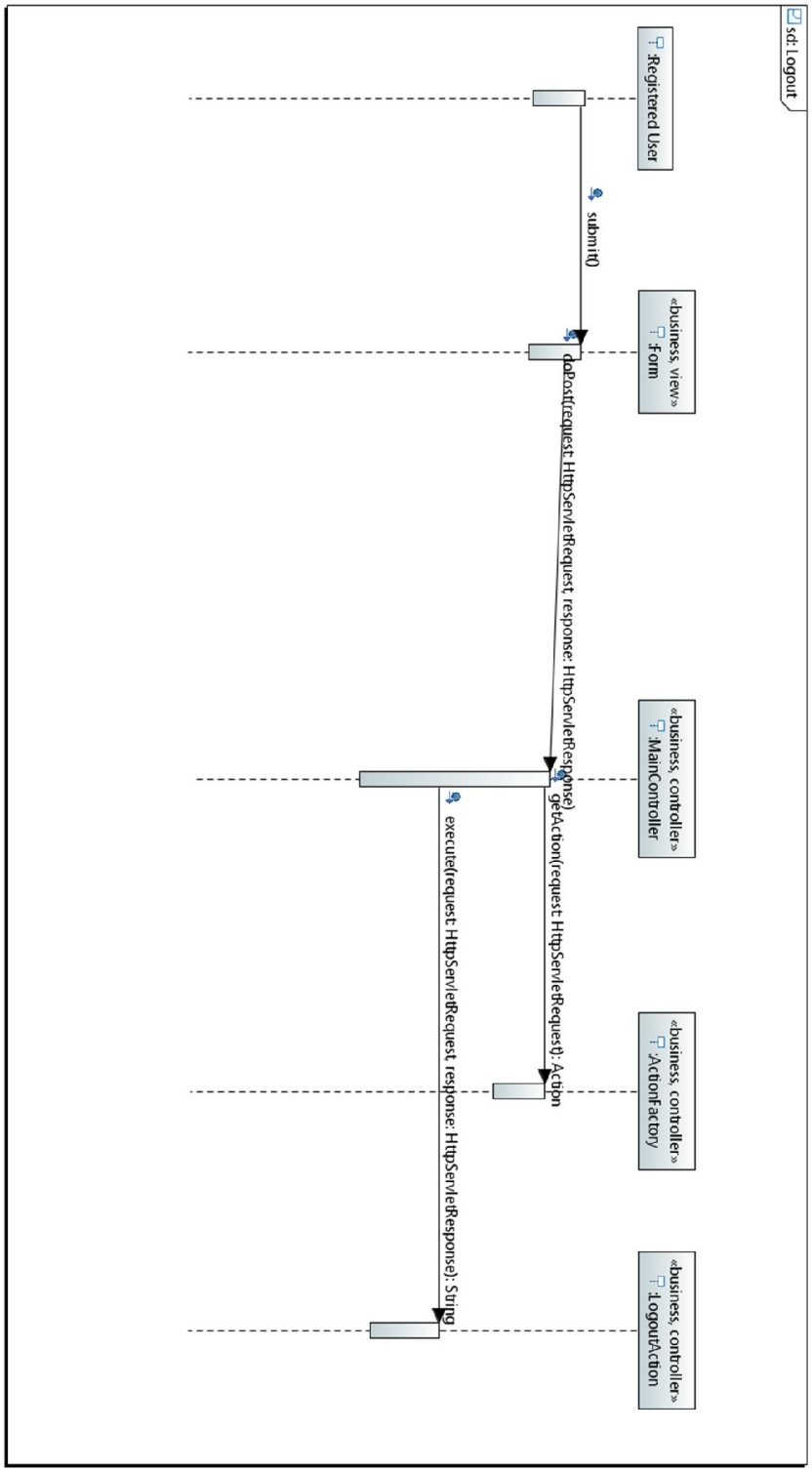


Figure 3.3.2 Logout

- **Registration**

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

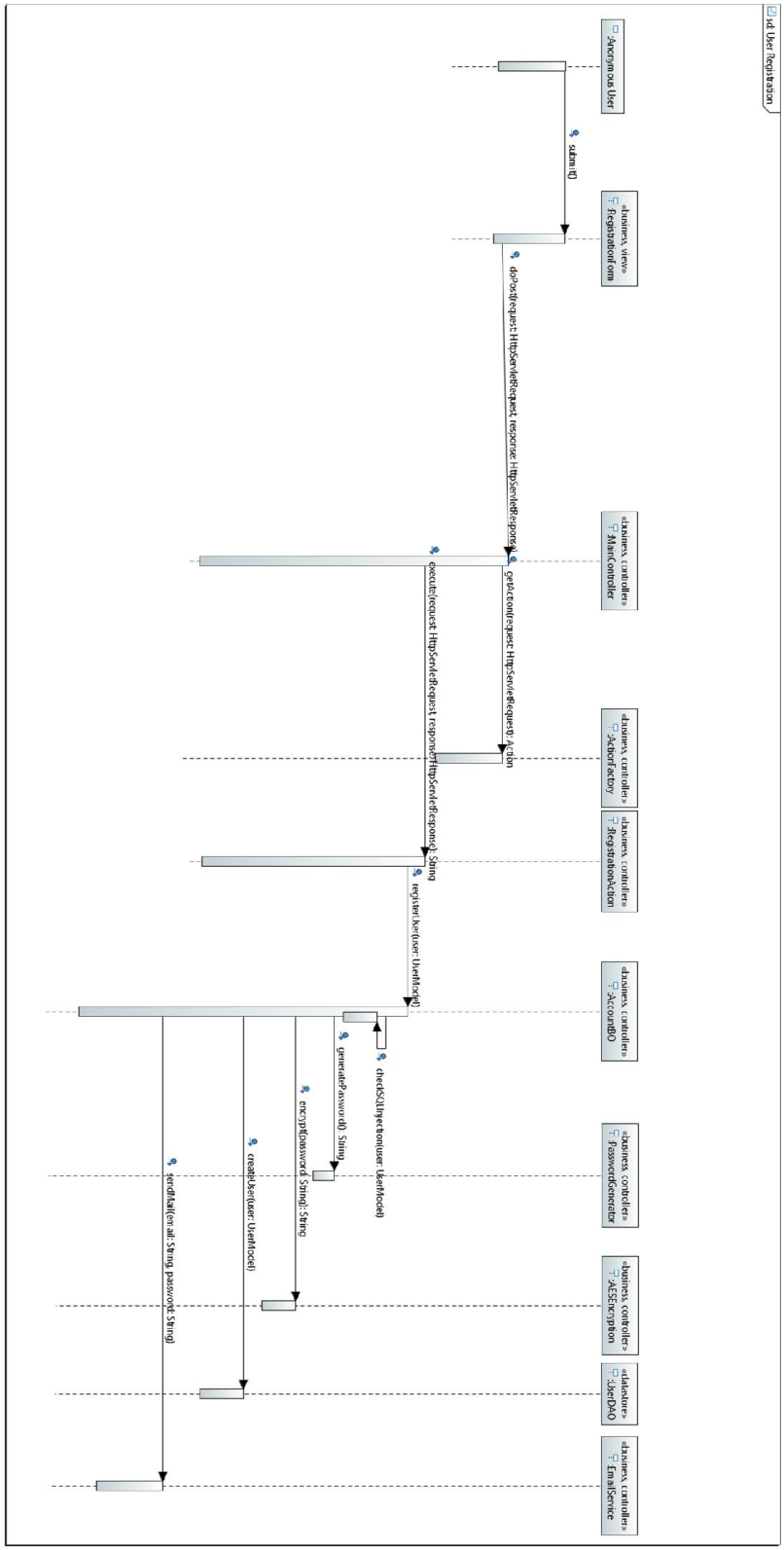


Figure 3.3.3 Registration

- **Change Password**

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

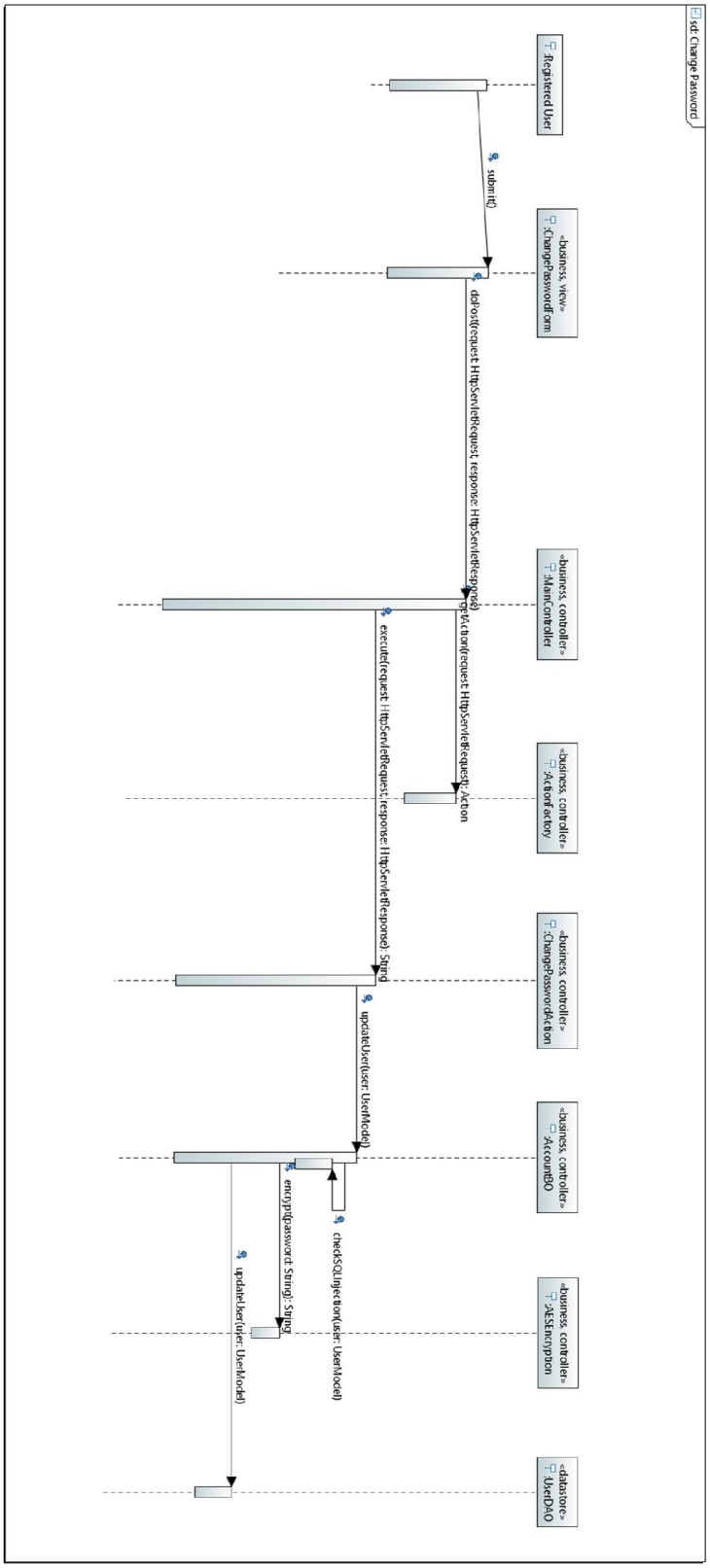


Figure 3.3.4 Change Password

- **View Message and Delete Message**

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

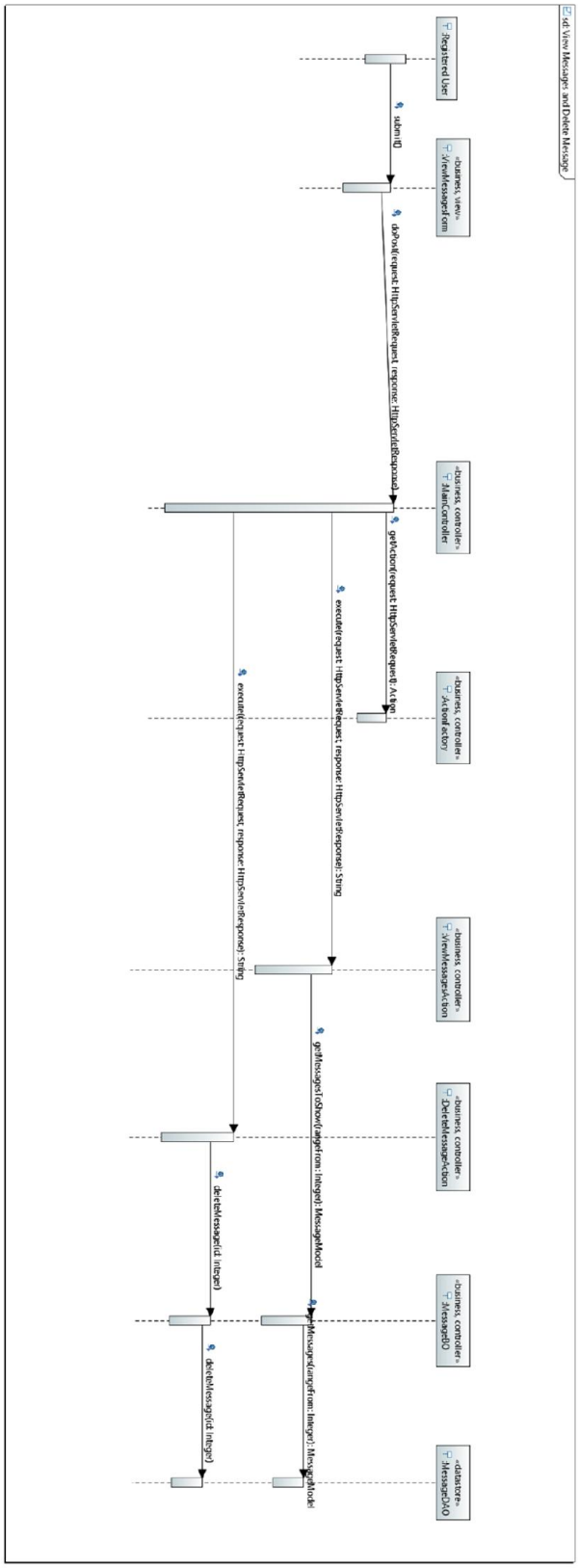


Figure 3.3.5 View Message and Delete Message

- **Post Message and XSS Filtering**

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

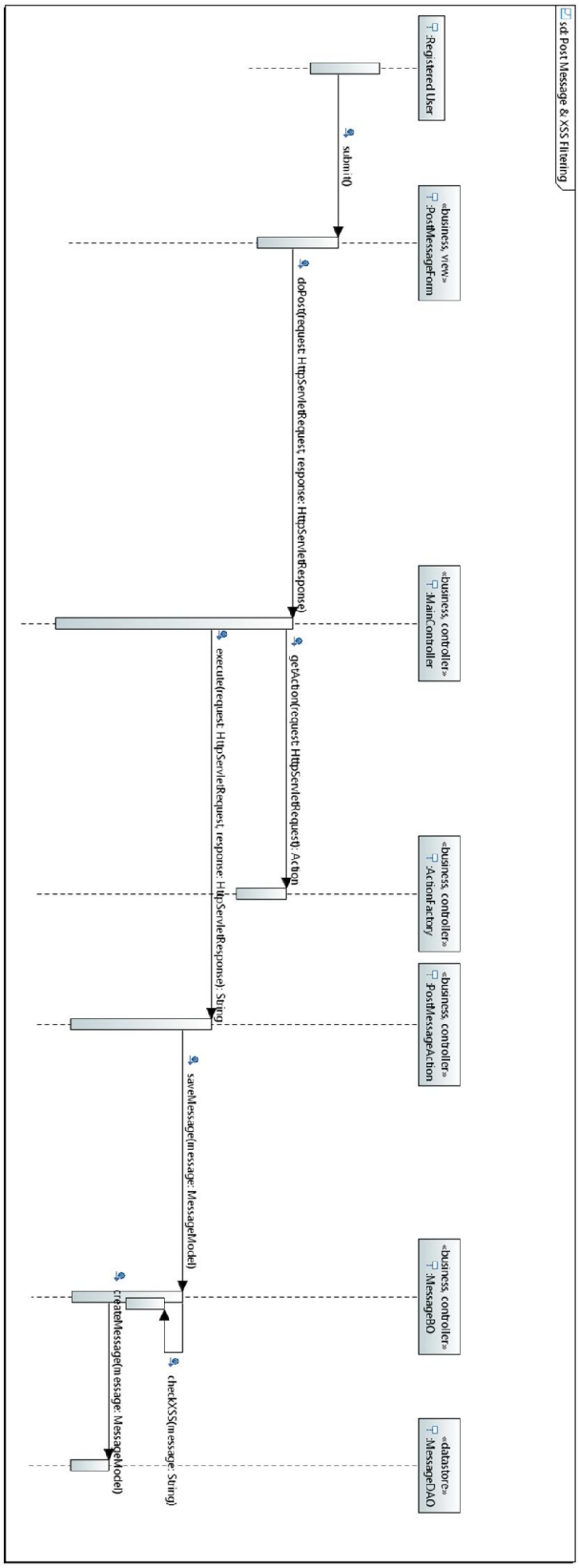


Figure 3.3.6 Post Message and XSS Filtering

- **Recover Password**

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

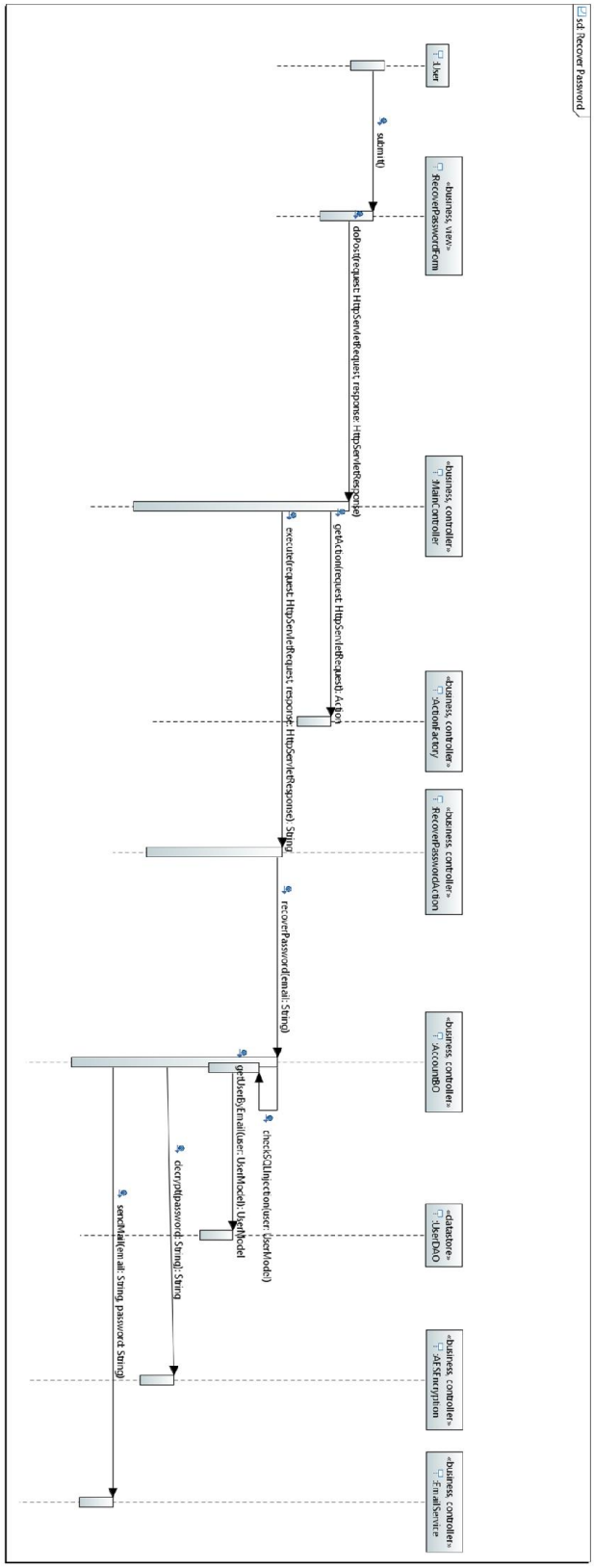


Figure 3.3.7 Recover Password

- **View Profile**

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

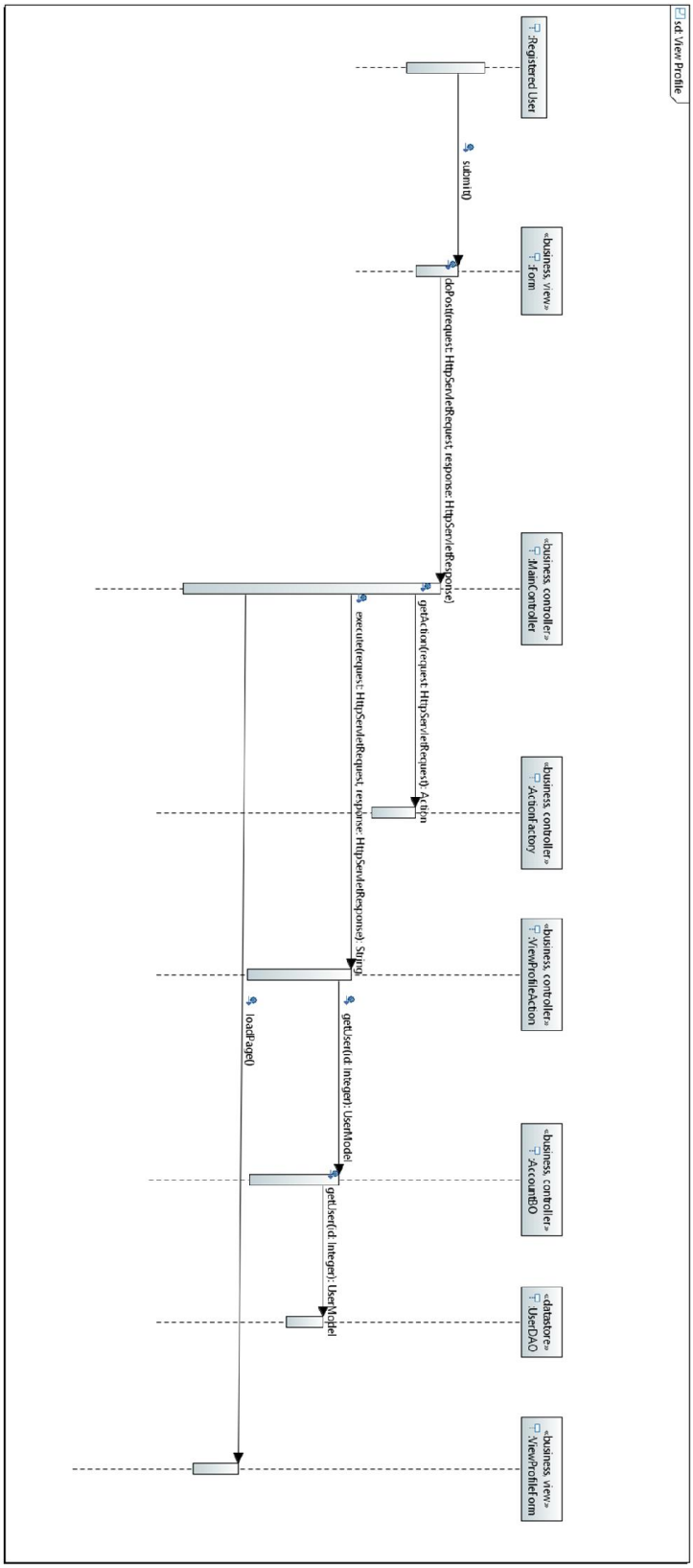


Figure 3.3.8 View Profile

6.4 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

6.4.1. 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.

6.4.2. 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

7. Testing Process

In this chapter, we test our Panther Buddy by using different testing methodologies such as system testing, unit testing, and subsystem testing. Unit testing is the testing of individual components such as functions and procedures. For unit tests, we use Junit in Java. After the unit tests, we perform subsystem testing. Subsystem testing is the testing of collections of modules which have been integrated into subsystems. JUnit is also used for subsystem testing. After the subsystem tests, we perform system testing. System testing is the testing of the subsystems that make up the entire system. For system testing, we use Selenium. Selenium is a tool that automates browsers. The code coverage tool that we use is EcLEmma. EcLEmma is code coverage tool for Eclipse.

7.1. Unit tests

Unit testing is individually tested for proper operation to ensure that each unit functions as designed. We test of individual components such as functions and procedures. For unit tests, we use Junit in Java. JUnit is a unit testing framework for the Java programming language. JUnit has been important in the development of test-driven development, and is one of a unit testing frameworks.

Test Case ID	Test_01_SystemTest_PB_UC01_ USER_REGISTRATION _Sunny1
Purpose	Test if the Registration feature functions properly with user's proper input data.
Test setup/Precondition	1) The registration page must be opened in the browser. 2) The server is up and running.
Input	1) The user enters valid information: a. first name - "abdur" b. last name - "rahman" c. email id: ashah044@fiu.edu d. phone number: 7863972558 2) The user clicks the submit button.
Expected output	1) The user will be sent password to the email. 2) The user will be directed to the login page.

Test Case ID	Test_02_SystemTest_PB_UC01_ USER_REGISTRATION _Sunny2
Purpose	Test if the Registration feature functions properly with user's proper input data.

Test setup/Precondition	1) The registration page must be opened in the browser. 2) The server is up and running.
Input	1) The user enters valid information: a. first name - “sharat” b. last name - “kedari” c. email id: sharathkedari@gmail.com d. phone number: 1234567890 2) The user clicks the submit button.
Expected output	1) The user will be sent password to the email. 2) The user will be directed to the login page.

Test Case ID	Test_03_SystemTest_PB_UC01_ USER_REGISTRATION _Rainy
Purpose	Test if the Registration feature functions properly with user’s already registered email.
Test setup/Precondition	1) The registration page must be opened in the browser. 2) The server is up and running. 3) User with ashah044@fiu.edu email already registered to system.
Input	1) The user enters valid information: a. first name - “abdur” b. last name - “rahman” c. email id: ashah044@fiu.edu d. phone number: 7863972558 2) The user clicks the submit button.
Expected output	1) The user will be shown error message.

Test Case ID	Test_04_SystemTest_PB_UC20_ LOGIN _Sunny1
Purpose	Test if the Login feature functions properly with user’s proper credential.
Test setup/Precondition	1) The Login page must be opened in the browser. 2) The server is up and running. 3) user ashah044@fiu.edu is already registered to the system and activated in the database.
Input	1) The user enters valid information: a. email - “ashah044@fiu.edu”

	b. password - “dddd” 2) The user clicks the login button.
Expected output	1) The user is redirected to the Panther Buddy home page.

Test Case ID	Test_05_SystemTest_PB_UC20_ LOGIN _Sunny2
Purpose	Test if the Login feature functions properly with user’s proper credential.
Test setup/Precondition	1) The Login page must be opened in the browser. 2) The server is up and running. 3) The user with sharathkedari@gmail.com is registered to the system and activated in the database.
Input	1) The user enters valid information: a. email - “sharathkedari@gmail.com” b. password - “dddd” 2) The user clicks the login button.
Expected output	1) The user is redirected to the Panther Buddy home page.

Test Case ID	Test_06_SystemTest_PB_UC20_ LOGIN _Rainy
Purpose	Test if the Login feature functions properly with user’s improper credential.
Test setup/Precondition	1) The Login page must be opened in the browser. 2) The server is up and running.
Input	1) The user enters email: ashah044@fiu.edu 2) wrong password: abcde
Expected output	1) The user is shown message error message and retry.

Test Case ID	Test_07_SystemTest_PB_UC02_RECOVER_PASSWORD _Sunny1

Purpose	Test if the Password recovery feature functions properly with user's proper email.
Test setup/Precondition	1) The server is up and running. 2) The user enters forgot his/her password.
Input	1) The user click recover password button and the recover password page is open in the web browser. 2) The user enters valid registered email: "ashah044@fiu.edu" 3) The user clicks the submit button.
Expected output	1) The user gets his recovered password at his registered email.

Test Case ID	Test_08_SystemTest_PB_UC02_RECOVER_PASSWORD _Sunny2
Purpose	Test if the Password recovery feature functions properly with user's proper email.
Test setup/Precondition	1) The server is up and running. 2) The user forgot his/her password.
Input	1) The user click recover password button and the recover password page is open in the web browser. 2) The user enters valid registered email: "sharathkedari@gmail.com" 3) The user clicks the submit button.
Expected output	1) The user gets her recovered password at her registered email.

Test Case ID	Test_09_SystemTest_PB_UC02_ RECOVER_PASSWORD _Rainy
Purpose	Test if the Password recovery feature functions properly with user's invalid email.
Test setup/Precondition	1) The server is up and running. 2) the email fakeemail@fake.com is not registered to the system.
Input	1) The user click recover password button and the recover password page is open in the web browser. 2) The user enters unregistered email: fakeemail@fake.com. 3) The user clicks the submit button.

Expected output	1) The user is asked to re-enter a valid email address.

Test Case ID	Test_10_SystemTest_PB_UC21_ LOGOUT _Sunny1
Purpose	Test if the Logout feature functions properly.
Test setup/Precondition	1) The server is up and running. 2) The user, abdur rahman is logged in into the pantherBuddy System.
Input	1) The user click logout button.
Expected output	1) System destroys the user session and redirects him to the login page.

Test Case ID	Test_11_SystemTest_PB_UC21_ LOGOUT _Sunny2
Purpose	Test if the Logout feature functions properly.
Test setup/Precondition	1) The server is up and running. 2) The user, sharath kedari is logged in into the pantherBuddy System.
Input	1) The user click logout button.
Expected output	1) System destroys the user session and redirects her to the login page.

Test Case ID	Test_12_SystemTest_PB_UC21_LOGOUT_Rainy
Purpose	Test if the UserActivation feature functions properly.
Test setup/Precondition	3) The server is up and running.

	1) The user, abdur rahman is registered into the pantherBuddy System, but never logged in to the system.
Input	1) The user enters his login credentials. 2) The user will hit “Login” button.
Expected output	1) System marks the user as activated. 2) The user gains access to the systems functionality.

Test Case ID	Test_13_SystemTest_PB_UC08_ACTIVATE_USER _Sunny1
Purpose	Test if the UserActivation feature functions properly.
Test setup/Precondition	1) The server is up and running. 2) The user, sharat kedari is registered into the pantherBuddy System, but never logged in to the system.
Input	1) The user enters his login credentials. 2) The user will hit “Login” button.
Expected output	1) System marks the user as activated. 2) The user gains access to the systems functionality.

Test Case ID	Test_14_SystemTest_PB_UC08_ACTIVATE_USER _Sunny2
Purpose	Test if the UserActivation feature functions properly with incorrect login credential.
Test setup/Precondition	1) The server is up and running. 2) The user, abdur rahman is registered into the pantherBuddy System, but never logged in to the system. (his email: ashah044@fiu.edu and password: dddd)
Input	1) The user enters her login credentials with correct email address but wrong password (abcde). 2) The user will hit “Login” button.
Expected output	1) System will show log in error and will not activate the user.

Test Case ID	Test_15_SystemTest_PB_UC09_ POST_MESSAGE _Sunny1
Purpose	Test if the PostMessage feature functions properly with user's valid input.
Test setup/Precondition	1) The server is up and running. 2) The user, abdur rahman is logged-in into the pantherBuddy System.
Input	1) The user writes the post "I need Introduction to Algorithm book by Cormen". 2) The user hits "post" button
Expected output	1) Message will be posted successfully.

Test Case ID	Test_16_SystemTest_PB_UC09_POST_MESSAGE_Sunny2
Purpose	Test if the PostMessage feature functions properly with user's valid input.
Test setup/Precondition	1) The server is up and running. 2) The user, Amy Lee is logged-in into the pantherBuddy System.
Input	1) The user writes the post "I have Introduction to Algorithm book by Cormen. If any wants to take it, let me know". 2) The user hits "add post" button.
Expected output	1) Message will be posted successfully.

Test Case ID	Test_17_SystemTest_PB_UC09_ POST_MESSAGE _Rainy
Purpose	Test if the PostMessage feature functions properly with user's illegal characters or keywords input.
Test setup/Precondition	1) The server is up and running. 2) The user, abdur rahman is logged-in into the pantherBuddy System.
Input	1) The user enters illegal characters or keywords in his post: <script> trying to hack pantherbuddy </script> 2) The user hits "add post" button.

Expected output	1) System detects the illegal characters or keywords and rejects the post.

Test Case ID	Test_18_SystemTest_PB_UC14_ DELETE_MESSAGE _Sunny1
Purpose	Test if the Delete Message feature functions properly.
Test setup/Precondition	3) The user must be logged in and user must be owner of the post he/she wants to delete. 4) The server is up and running.
Input	4) The user click delete post button near the message.
Expected output	2) The post will be deleted and page will be reloaded after deleting the post.

Test Case ID	Test_19_SystemTest_PB_UC14_ DELETE_MESSAGE _Sunny2
Purpose	Test if the Delete Message feature functions properly.
Test setup/Precondition	1) The user must be logged in and user must be owner of the post he/she wants to delete. 2) The server is up and running.
Input	1) The user click delete post button near the message.
Expected output	1) The post will be deleted and page will be reloaded after deleting the post.

Test Case ID	Test_20_SystemTest_PB_UC03_ CHANGE_PASSWORD _Sunny1
Purpose	Test if the Change Password feature functions properly.

Test setup/Precondition	1) The user (abdur rahman) has his own profile page open on 'Panther Buddy. 2) The server is up and running.
Input	1) User clicks the change password button on his profile page 2) 3 fields appear on the page. They are: Current Password. dddd New Password : ssss Confirm Password: ssss 3) The user clicks the submit button to submit his data
Expected output	1) The User's password is changed.

Test Case ID	Test_21_SystemTest_PB_UC03_ CHANGE_PASSWORD _Sunny1
Purpose	Test if the Change Password feature functions properly.
Test setup/Precondition	1) The user (sharath kedari) has his own profile page open on 'Panther Buddy. 2) The server is up and running.
Input	1) User clicks the change password button on his profile page 2) 3 fields appear on the page. They are: Current Password. dddd New Password ssss Confirm Password ssss 3) The user clicks the submit button to submit his data
Expected output	1) The User's password is changed.

Test Case ID	Test_22_SystemTest_PB_UC03_ CHANGE_PASSWORD _Rainy
Purpose	Test if the Change Password feature functions properly if user enter his/her current password wrong.
Test setup/Precondition	1) The user (abdur rahman) has his own profile page open on 'Panther Buddy.

	2) The server is up and running.
Input	1) User clicks the change password button on his profile page 2) 3 fields appear on the page. They are: Current Password. s New Password: ssss Confirm Password ssss 3) The user clicks the submit button to submit his data
Expected output	1) The User's password is not changed.

Test Case ID	Test_23_SystemTest_PB_UC16_ VIEW_POST _Sunny1
Purpose	Test if the View Post feature functions properly.
Test setup/Precondition	1) The registered user has to login in order to be able to view a post. 2) The server is up and running.
Input	1) The user clicks the home page button on the screen he is at. 2) User is redirected to the homepage where he can view a list of posts ordered by most recent at top.
Expected output	1) The user can view the already posted messages.

Test Case ID	Test_24_SystemTest_PB_UC16_ VIEW_POST _Sunny2
Purpose	Test if the View Post feature functions properly.
Test setup/Precondition	1) The registered user has to login in order to be able to view a post. 2) The server is up and running.

Input	1) The user clicks the home page button on the screen he is at. 2) User is redirected to the homepage where he can view a list of posts ordered by most recent at top.
Expected output	1) The user can view the already posted messages.

Test Case ID	Test_25_SystemTest_PB_UC16_ VIEW_POST _Rainy
Purpose	Test if the View Post feature functions properly ,when user tries to view post when his session is expired.
Test setup/Precondition	1) The registered user has to login in order to be able to view a post. 2) The server is up and running.
Input	1) The user clicks logout button. 2) the user tries to enter “pantherbuddy/viewmessage.jsp” url directly into the system.
Expected output	1) User is redirected to the login.jsp to login to the system

Test Case ID	Test_26_SystemTest_PB_SC02_ XSS_FILTERING _Sunny1
Purpose	Test if the XSS Filtering feature functions properly.
Test setup/Precondition	1) The user has the post message page open. 2) The server is up and running.
Input	1) The user message is filtered to check for XSS attack 2) The user writes his message in the space provided for writing his message. 3) The user clicks the post message button to submit the message. 4) The system checks the message string for malicious code.

	5) System saves the message for moderation by moderator.
Expected output	The user's message is checked for malicious code. Once verified they are saved in the system for moderation by moderator.

Test Case ID	Test_27_SystemTest_PB_SC02_ XSS_FILTERING _Sunny2
Purpose	Test if the XSS Filtering feature functions properly.
Test setup/Precondition	1) The user has the post message page open. 2) The server is up and running.
Input	1) The user message is filtered to check for XSS attack 2) The user writes his message in the space provided for writing his message. 3) The user clicks the post message button to submit the message. 4) The system checks the message string for malicious code. 5) System saves the message for moderation by moderator.
Expected output	The user's message is checked for malicious code. They are saved in the system for moderation by moderator.

Test Case ID	Test_28_SystemTest_PB_SC02_ XSS_FILTERING _Rainy
Purpose	Test if the XSS Filtering feature functions properly, when malicious message is posted
Test setup/Precondition	1) The user has the post message page open. 2) The server is up and running.
Input	1) The user message is filtered to check for XSS attack 2) The user writes his message in the space provided for writing his message.

	3) The user clicks the post message button to submit the message. 4) The system checks the message string for malicious code. 5) System saves the message for moderation by moderator.
Expected output	The user's message is checked for malicious code. The message is not saved as it contains malicious code.

Test Case ID	Test_30_SystemTest_PB_SC01_ FILTER_INPUT _Sunny1
Purpose	Test if the SQL Injection feature functions properly.
Test setup/Precondition	1) The user has login page open 2) The server is up and running.
Input	1) The user logins into the system using his email and password. a. User enters his email into the email field b. The user enters his password in the password field c. The user clicks the login button to submit his data. d. The system checks the email and password for malicious content before querying for the user with the entered login credentials
Expected output	1) The user is redirected to the Panther Buddy home page.

Test Case ID	Test_31_SystemTest_PB_SC01_ FILTER_INPUT _Sunny2
Purpose	Test if the SQL Injection feature functions properly.

Test setup/Precondition	1) The user has login page open 2) The server is up and running.
Input	3) The user logins into the system using his email and password. e. User enters his email into the email field f. The user enters his password in the password field g. The user clicks the login button to submit his data. h. The system checks the email and password for malicious content before querying for the user with the entered login credentials
Expected output	1) The user is redirected to the Panther Buddy home page.

Test Case ID	Test_32_SystemTest_PB_SC01_FILTER_INPUT_Rainy
Purpose	Test if the SQL Injection feature functions properly, when user enters his email/password wrong
Test setup/Precondition	1) The user has login page open 2) The server is up and running.
Input	3) The user logins into the system using his email and password. i. User enters his email into the email field j. The user enters his password in the password field k. The user clicks the login button to submit his data. l. The system checks the email and password for malicious content before querying for the user with the entered login credentials
Expected output	1) The user is not redirected to the Panther Buddy home page.

Test Case ID	Test_32_SystemTest_PB_UC04_ VIEW_OWN_PROFILE _Sunny1
Purpose	Test if the ViewProfile feature functions properly with if user logged in.
Test setup/Precondition	1) The user is logged in to the system. 2) The system is up and running.
Input	1) User clicks the Profile option on the menu to view his profile. button on his profile page
Expected output	1) The profile of the user with change password option will be shown to him.

Test Case ID	Test_33_SystemTest_PB_UC04_ VIEW_OWN_PROFILE _Sunny2
Purpose	Test if the ViewProfile feature functions properly with if user logged in.
Test setup/Precondition	1) The system is up and running. 2) The user is in his home page. 3) The user sees post from another user, 'Amy lee'.
Input	1) The user clicks the name of 'Amy lee'.
Expected output	1) The User sees the profile of Amy lee.

7.2. Subsystem tests

This part includes subsystem verification test. It is performed as a prelude to system testing. It is performed in the operational environment using installed system hardware and software. Subsystem testing is the testing of collections of modules which have been integrated into subsystems. JUnit is also used for subsystem testing. This part is subsystem test for "datastore" subsystem. Datastore subsystem is the third layer of system, which is responsible for

data maintain with database, including two class: UserDAO and MessageDAO. Below is the subsystem test for these two class.

Test Case ID	Test_03_SubsystemTest_UserDAO: testCreateUserPositive
Purpose	Test if the create user functions properly with user's proper input data.
Test setup/Precondition	Database connected. The user is not in database presently.
Input	new user Fname="qiu"; Lname="qiu"; Password="123"; Phonenumber=7869256598L; EmailId="qiu15@gmail.com";
Expected output	User is added into database.

Test Case ID	Test_03_SubsystemTest_ UserDAO : testCreateUserNegative
Purpose	Test if the create user functions properly with user's improper input data.
Test setup/Precondition	Database connected. The user is already in database at present.
Input	new user: Fname="qiu"; Lname="qiu"; Password="123"; Phonenumber=7869256598L;

	EmailId=”qiu15@gmail.com”;
Expected output	Throw exception.

Test Case ID	Test_03_SubsystemTest_ UserDAO: testDeleteUserPositive
Purpose	Test if the delete user functions properly with user’s proper input data.
Test setup/Precondition	Database connected. The user with id 39 is already in database.
Input	Input: user id = 39.
Expected output	delete success.

Test Case ID	Test_03_SubsystemTest_ UserDAO: testDeleteUserNegative
Purpose	Test if the delete user functions properly with user’s improper input data.
Test setup/Precondition	Database connected. The user with id 9999 is not in database.
Input	userid = 9999.
Expected output	throw exception.

Test Case ID	Test_03_SubsystemTest_ UserDAO: testGetUserPositive
Purpose	Test if the get user functions properly with user’s proper input data.
Test setup/Precondition	Database connected.

	The user with id 1 is already in database.
Input	user id = 1.
Expected output	get user success.

Test Case ID	Test_03_SubsystemTest_ UserDAO: testGetUserNegative
Purpose	Test if the get user functions properly with user's improper input data.
Test setup/Precondition	Database connected. The user with id 9999 is not in database.
Input	user id = 9999;
Expected output	get nothing.

Test Case ID	Test_03_SubsystemTest_ UserDAO: testUpdateUserPositive
Purpose	Test if the update user functions properly with user's proper input data.
Test setup/Precondition	Database connected. The user with id 1 is already in database.
Input	change user (id=1) status to 1.
Expected output	user status changed.

Test Case ID	Test_03_SubsystemTest_ UserDAO: testUpdateUserNegative
Purpose	Test if the update user functions properly with user's proper input data.

Test setup/Precondition	Database connected. User with id 9999 is not in database.
Input	change user (id=9999) status to 1.
Expected output	nothing changed.

Test Case ID	Test_03_SubsystemTest_ UserDAO: testGetUserByEmailPositive
Purpose	Test if the get user by email functions properly with user's proper input data.
Test setup/Precondition	Database connected. User with email "qiu@gmail.com" is already in database.
Input	user email = "qiu@gmail.com".
Expected output	get success.

Test Case ID	Test_03_SubsystemTest_ UserDAO: testGetUserByEmailNegative
Purpose	Test if the get user by email functions properly with user's proper input data.
Test setup/Precondition	Database connected. The user with email "xxxxxx @gmail.com" is not in database.
Input	user email = "xxxxxx @gmail.com".
Expected output	get nothing.

Test Case ID	Test_04_SubsystemTest_MessageDAO: testCreateMessagePositive
Purpose	Test if the create message functions properly with user's proper input data.
Test setup/Precondition	Database connected.
Input	user id = 1, message="hello world".
Expected output	create success.

Test Case ID	Test_04_SubsystemTest_ MessageDAO: testCreateMessageNegative
Purpose	Test if the create message functions properly with user's proper input data.
Test setup/Precondition	Database connected. User id 9999 is not in database.
Input	user id = 9999, message="hello world".
Expected output	throw exception

Test Case ID	Test_04_SubsystemTest_ MessageDAO: testDeleteMessagePositive
Purpose	Test if the delete message functions properly with user's proper input data.
Test setup/Precondition	Database connect. Message with id 1 is already in database.
Input	message id = 1;
Expected output	delete success.

Test Case ID	Test_04_SubsystemTest_ MessageDAO: testDeleteMessageNegative
Purpose	Test if the delete message functions properly with user's proper input data.
Test setup/Precondition	Database connected. Message with id 9999 is not in database.
Input	message id = 9999.
Expected output	nothing changed.

Test Case ID	Test_04_SubsystemTest_ MessageDAO: testGetMessagePositve
Purpose	Test if the get message functions properly with user's proper input data.
Test setup/Precondition	Database connected. Message with id 1 is already in database.
Input	message id = 1.
Expected output	get message.

Test Case ID	Test_04_SubsystemTest_ MessageDAO: testGetMessageNegative
Purpose	Test if the get message functions properly with user's proper input data.
Test setup/Precondition	Database connected. Message with id 9999 is not in database.

Input	message id = 9999.
Expected output	get nothing.

7.3. System tests

After the subsystem tests, we perform system testing. System testing is the testing of the subsystems that make up the entire system. For system testing, we use Selenium. Selenium is a tool that automates browsers. The code coverage tool that we use is EcLEmma. EcLEmma is code coverage tool for Eclipse.

Test Case ID	Test_01_SystemTest_PB_UC01_Sunny1
Purpose	Test if the Registration feature functions properly with user's proper input data.
Test setup/Precondition	3) The registration page must be opened in the browser. 4) The server is up and running.
Input	3) The user enters valid information: a. first name - "abdur" b. last name - "rahman" c. email id: ashah044@fiu.edu d. phone number: 7863972558 4) The user clicks the submit button.
Expected output	3) The user will be sent password to the email. 4) The user will be directed to the login page.

Test Case ID	Test_02_SystemTest_PB_UC01_Sunny2
Purpose	Test if the Registration feature functions properly with user's proper input data.
Test setup/Precondition	3) The registration page must be opened in the browser. 4) The server is up and running.
Input	3) The user enters valid information: a. first name - "sharat" b. last name - "kedari" c. email id: sharathkedari@gmail.com

	d. phone number: 1234567890 4) The user clicks the submit button.
Expected output	3) The user will be sent password to the email. 4) The user will be directed to the login page.

Test Case ID	Test_03_SystemTest_PB_UC01_Rainy
Purpose	Test if the Registration feature functions properly with user's already registered email.
Test setup/Precondition	4) The registration page must be opened in the browser. 5) The server is up and running. 6) User with ashah044@fiu.edu email already registered to system.
Input	3) The user enters valid information: a. first name - "abdur" b. last name - "rahman" c. email id: ashah044@fiu.edu d. phone number: 7863972558 4) The user clicks the submit button.
Expected output	2) The user will be shown error message.

Test Case ID	Test_04_SystemTest_PB_UC20_Sunny1
Purpose	Test if the Login feature functions properly with user's proper credential.
Test setup/Precondition	4) The Login page must be opened in the browser. 5) The server is up and running. 6) user ashah044@fiu.edu is already registered to the system and activated in the database.
Input	3) The user enters valid information: a. email - "ashah044@fiu.edu" b. password - "dddd" 4) The user clicks the login button.
Expected output	2) The user is redirected to the Panther Buddy home page.

Test Case ID	Test_05_SystemTest_PB_UC20_Sunny2
Purpose	Test if the Login feature functions properly with user's proper credential.
Test setup/Precondition	4) The Login page must be opened in the browser. 5) The server is up and running. 6) The user with sharathkedari@gmail.com is registered to the system and activated in the database.
Input	3) The user enters valid information: a. email - "sharathkedari@gmail.com" b. password - "dddd" 4) The user clicks the login button.
Expected output	2) The user is redirected to the Panther Buddy home page.

Test Case ID	Test_06_SystemTest_PB_UC20_Rainy
Purpose	Test if the Login feature functions properly with user's improper credential.
Test setup/Precondition	3) The Login page must be opened in the browser. 4) The server is up and running.
Input	3) The user enters email: ashah044@fiu.edu 4) wrong password: abcde
Expected output	2) The user is shown message error message and retry.

Test Case ID	Test_07_SystemTest_PB_UC02_Sunny1
Purpose	Test if the Password recovery feature functions properly with user's proper email.
Test setup/Precondition	5) The server is up and running. 6) The user enters forgot his/her password.
Input	5) The user click recover password button and the recover password page is open in the web browser.

	6) The user enters valid registered email: "ashah044@fiu.edu" 7) The user clicks the submit button.
Expected output	3) The user gets his recovered password at his registered email.

Test Case ID	Test_08_SystemTest_PB_UC02_Sunny2
Purpose	Test if the Password recovery feature functions properly with user's proper email.
Test setup/Precondition	3) The server is up and running. 4) The user forgot his/her password.
Input	4) The user click recover password button and the recover password page is open in the web browser. 5) The user enters valid registered email: "sharathkedari@gmail.com" 6) The user clicks the submit button.
Expected output	2) The user gets her recovered password at her registered email.

Test Case ID	Test_09_SystemTest_PB_UC02_Rainy
Purpose	Test if the Password recovery feature functions properly with user's invalid email.
Test setup/Precondition	3) The server is up and running. 4) the email fakeemail@fake.com is not registered to the system.
Input	4) The user click recover password button and the recover password page is open in the web browser. 5) The user enters unregistered email: fakeemail@fake.com. 6) The user clicks the submit button.
Expected output	2) The user is asked to re-enter a valid email address.

Test Case ID	Test_10_SystemTest_PB_UC21_Sunny1
Purpose	Test if the Logout feature functions properly.
Test setup/Precondition	4) The server is up and running. 5) The user, abdur rahman is logged in into the pantherBuddy System.
Input	2) The user click logout button.
Expected output	2) System destroys the user session and redirects him to the login page.

Test Case ID	Test_11_SystemTest_PB_UC21_Sunny2
Purpose	Test if the Logout feature functions properly.
Test setup/Precondition	3) The server is up and running. 4) The user, sharath kedari is logged in into the pantherBuddy System.
Input	2) The user click logout button.
Expected output	2) System destroys the user session and redirects her to the login page.

Test Case ID	Test_12_SystemTest_PB_UC08_Sunny1
Purpose	Test if the UserActivation feature functions properly.
Test setup/Precondition	6) The server is up and running. 3) The user, abdur rahman is registered into the pantherBuddy System, but never logged in to the system.
Input	3) The user enters his login credentials. 4) The user will hit “Login” button.
Expected output	3) System marks the user as activated. 4) The user gains access to the systems functionality.

Test Case ID	Test_13_SystemTest_PB_UC08_Sunny2
Purpose	Test if the UserActivation feature functions properly.
Test setup/Precondition	2) The server is up and running. 4) The user, sharat kedari is registered into the pantherBuddy System, but never logged in to the system.
Input	3) The user enters his login credentials. 4) The user will hit “Login” button.
Expected output	3) System marks the user as activated. 4) The user gains access to the systems functionality.

Test Case ID	Test_14_SystemTest_PB_UC08_Rainy
Purpose	Test if the UserActivation feature functions properly with incorrect login credential.
Test setup/Precondition	3) The server is up and running. 4) The user, abdur rahman is registered into the pantherBuddy System, but never logged in to the system. (his email: ashah044@fiu.edu and password: dddd)
Input	3) The user enters her login credentials with correct email address but wrong password (abcde). 4) The user will hit “Login” button.
Expected output	2) System will show log in error and will not activate the user.

Test Case ID	Test_15_SystemTest_PB_UC09_Sunny1
Purpose	Test if the PostMessage feature functions properly with user’s valid input.
Test setup/Precondition	3) The server is up and running. 4) The user, abdur rahman is logged-in into the pantherBuddy System.

Input	3) The user writes the post "I need Introduction to Algorithm book by Cormen". 4) The user hits "post" button
Expected output	2) Message will be posted successfully.

Test Case ID	Test_16_SystemTest_PB_UC09_Sunny2
Purpose	Test if the PostMessage feature functions properly with user's valid input.
Test setup/Precondition	3) The server is up and running. 4) The user, Amy Lee is logged-in into the pantherBuddy System.
Input	3) The user writes the post "I have Introduction to Algorithm book by Cormen. If any wants to take it, let me know". 4) The user hits "add post" button.
Expected output	2) Message will be posted successfully.

Test Case ID	Test_17_SystemTest_PB_UC09_Rainy
Purpose	Test if the PostMessage feature functions properly with user's illegal characters or keywords input.
Test setup/Precondition	3) The server is up and running. 4) The user, abdur rahman is logged-in into the pantherBuddy System.
Input	3) The user enters illegal characters or keywords in his post: <script> trying to hack pantherbuddy </script> 4) The user hits "add post" button.
Expected output	2) System detects the illegal characters or keywords and rejects the post.

Test Case ID	Test_18_SystemTest_PB_UC14_Sunny1

Purpose	Test if the Delete Message feature functions properly.
Test setup/Precondition	7) The user must be logged in and user must be owner of the post he/she wants to delete. 8) The server is up and running.
Input	8) The user click delete post button near the message.
Expected output	4) The post will be deleted and page will be reloaded after deleting the post.

Test Case ID	Test_19_SystemTest_PB_UC14_Sunny2
Purpose	Test if the Delete Message feature functions properly.
Test setup/Precondition	3) The user must be logged in and user must be owner of the post he/she wants to delete. 4) The server is up and running.
Input	2) The user click delete post button near the message.
Expected output	2) The post will be deleted and page will be reloaded after deleting the post.

Test Case ID	Test_20_SystemTest_PB_UC03_Sunny1
Purpose	Test if the Change Password feature functions properly.
Test setup/Precondition	3) The user (abdur rahman) has his own profile page open on 'Panther Buddy. 4) The server is up and running.
Input	4) User clicks the change password button on his profile page 5) 3 fields appear on the page. They are: Current Password. dddd

	New Password : ssss Confirm Password: ssss 6) The user clicks the submit button to submit his data
Expected output	2) The User's password is changed.

Test Case ID	Test_21_SystemTest_PB_UC03_Sunny2
Purpose	Test if the Change Password feature functions properly.
Test setup/Precondition	3) The user (sharath kedari) has his own profile page open on 'Panther Buddy. 4) The server is up and running.
Input	4) User clicks the change password button on his profile page 5) 3 fields appear on the page. They are: Current Password. dddd New Password ssss Confirm Password ssss 6) The user clicks the submit button to submit his data
Expected output	2) The User's password is changed.

Test Case ID	Test_22_SystemTest_PB_UC03_Rainy
Purpose	Test if the Change Password feature functions properly if user enter his/her current password wrong.
Test setup/Precondition	3) The user (abdur rahman) has his own profile page open on 'Panther Buddy. 4) The server is up and running.
Input	4) User clicks the change password button on his profile page 5) 3 fields appear on the page. They are: Current Password. s New Password: ssss Confirm Password ssss

	6) The user clicks the submit button to submit his data
Expected output	2) The User's password is not changed.

Test Case ID	Test_23_SystemTest_PB_UC16_Sunny1
Purpose	Test if the View Post feature functions properly.
Test setup/Precondition	3) The registered user has to login in order to be able to view a post. 4) The server is up and running.
Input	3) The user clicks the home page button on the screen he is at. 4) User is redirected to the homepage where he can view a list of posts ordered by most recent at top.
Expected output	2) The user can view the already posted messages.

Test Case ID	Test_24_SystemTest_PB_UC16_Sunny2
Purpose	Test if the View Post feature functions properly.
Test setup/Precondition	3) The registered user has to login in order to be able to view a post. 4) The server is up and running.
Input	3) The user clicks the home page button on the screen he is at. 4) User is redirected to the homepage where he can view a list of posts ordered by most recent at top.
Expected output	2) The user can view the already posted messages.

Test Case ID	Test_25_SystemTest_PB_UC16_Rainy
Purpose	Test if the View Post feature functions properly ,when user tries to view post when his session is expired.
Test setup/Precondition	3) The registered user has to login in order to be able to view a post. 4) The server is up and running.
Input	3) The user clicks logout button. 4) the user tries to enter “pantherbuddy/viewmessage.jsp” url directly into the system.
Expected output	2) User is redirected to the login.jsp to login to the system

Test Case ID	Test_26_SystemTest_PB_SC02_Sunny1
Purpose	Test if the XSS Filtering feature functions properly.
Test setup/Precondition	3) The user has the post message page open. 4) The server is up and running.
Input	6) The user message is filtered to check for XSS attack 7) The user writes his message in the space provided for writing his message. 8) The user clicks the post message button to submit the message. 9) The system checks the message string for malicious code. 10) System saves the message for moderation by moderator.
Expected output	The user’s message is checked for malicious code. Once verified they are saved in the system for moderation by moderator.

Test Case ID	Test_27_SystemTest_PB_SC02_Sunny2
Purpose	Test if the XSS Filtering feature functions properly.
Test setup/Precondition	3) The user has the post message page open. 4) The server is up and running.
Input	6) The user message is filtered to check for XSS attack 7) The user writes his message in the space provided for writing his message. 8) The user clicks the post message button to submit the message. 9) The system checks the message string for malicious code. 10) System saves the message for moderation by moderator.
Expected output	The user's message is checked for malicious code. They are saved in the system for moderation by moderator.

Test Case ID	Test_28_SystemTest_PB_SC02_Rainy
Purpose	Test if the XSS Filtering feature functions properly, when malicious message is posted
Test setup/Precondition	3) The user has the post message page open. 4) The server is up and running.
Input	6) The user message is filtered to check for XSS attack 7) The user writes his message in the space provided for writing his message. 8) The user clicks the post message button to submit the message. 9) The system checks the message string for malicious code. 10) System saves the message for moderation by moderator.
Expected output	The user's message is checked for malicious code. The message is not saved as it contains malicious code.

Test Case ID	Test_29_SystemTest_PB_SC01_Sunny1
Purpose	Test if the SQL Injection feature functions properly.
Test setup/Precondition	3) The user has login page open 4) The server is up and running.
Input	2) The user logins into the system using his email and password. a. User enters his email into the email field b. The user enters his password in the password field c. The user clicks the login button to submit his data. d. The system checks the email and password for malicious content before querying for the user with the entered login credentials
Expected output	2) The user is redirected to the Panther Buddy home page.

Test Case ID	Test_30_SystemTest_PB_SC01_Sunny2
Purpose	Test if the SQL Injection feature functions properly.
Test setup/Precondition	4) The user has login page open 5) The server is up and running.
Input	6) The user logins into the system using his email and password. e. User enters his email into the email field f. The user enters his password in the password field g. The user clicks the login button to submit his data.

	h. The system checks the email and password for malicious content before querying for the user with the entered login credentials
Expected output	2) The user is redirected to the Panther Buddy home page.

Test Case ID	Test_31_SystemTest_PB_SC01_Rainy
Purpose	Test if the SQL Injection feature functions properly, when user enters his email/password wrong
Test setup/Precondition	4) The user has login page open 5) The server is up and running.
Input	6) The user logins into the system using his email and password. i. User enters his email into the email field j. The user enters his password in the password field k. The user clicks the login button to submit his data. l. The system checks the email and password for malicious content before querying for the user with the entered login credentials
Expected output	2) The user is not redirected to the Panther Buddy home page.

Test Case ID	Test_32_SystemTest_PB_UC04_Sunny1
Purpose	Test if the ViewProfile feature functions properly with if user logged in.
Test setup/Precondition	3) The user is logged in to the system. 4) The system is up and running.
Input	2) User clicks the Profile option on the menu to view his profile. button on his profile page

Expected output	2) The profile of the user with change password option will be shown to him.

Test Case ID	Test_33_SystemTest_PB_UC04_Sunny2
Purpose	Test if the ViewProfile feature functions properly with if user logged in.
Test setup/Precondition	4) The system is up and running. 5) The user is in his home page. 6) The user sees post from another user, 'Amy lee'.
Input	2) The user clicks the name of 'Amy lee'.
Expected output	2) The User sees the profile of Amy lee.

7.4. Evaluation of tests

7.4.1. Test result and comment

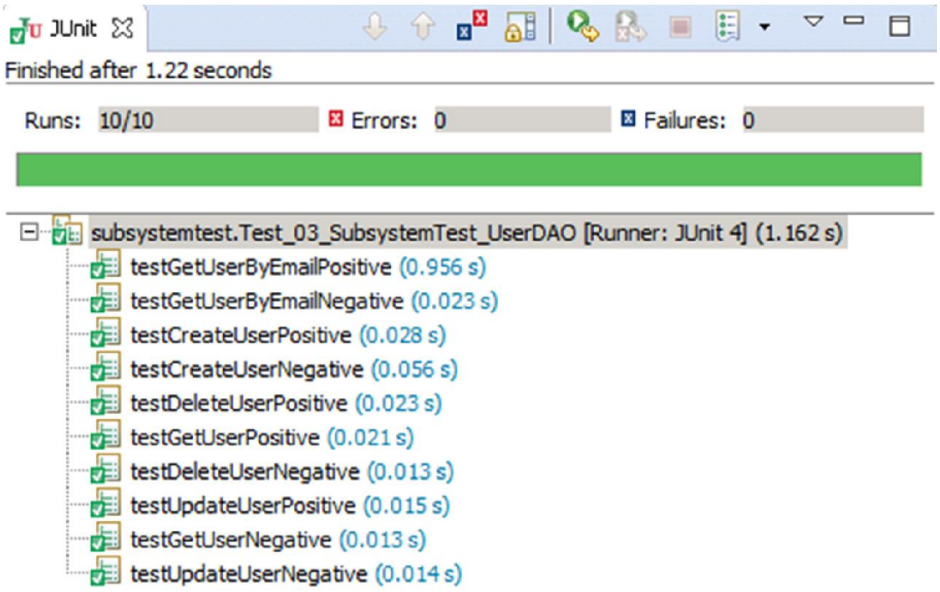
Subsystem Test

TestCaseID	PassFail	Fix
Test_03_SubsystemTest_UserDAO: testCreateUserPositive	Pass	no
Test_03_SubsystemTest_ UserDAO : testCreateUserNegative	Pass	no
Test_03_SubsystemTest_ UserDAO: testDeleteUserPositive	Pass	no
Test_03_SubsystemTest_ UserDAO: testDeleteUserNegative	Pass	no
Test_03_SubsystemTest_ UserDAO: testGetUserPositive	Pass	no
Test_03_SubsystemTest_ UserDAO: testGetUserNegative	Pass	no

Test_03_SubsystemTest_ UserDAO: testUpdateUserPositive	Pass	no
Test_03_SubsystemTest_ UserDAO: testUpdateUserNegative	Pass	no
Test_03_SubsystemTest_ UserDAO: testGetUserByEmailPositive	Pass	no
Test_03_SubsystemTest_ UserDAO: testGetUserByEmailNegative	Pass	no
Test_04_SubsystemTest_MessageDAO: testCreateMessagePositive	Pass	no
Test_04_SubsystemTest_ MessageDAO: testCreateMessageNegative	Pass	no
Test_04_SubsystemTest_ MessageDAO: testDeleteMessagePositive	Pass	no
Test_04_SubsystemTest_ MessageDAO: testDeleteMessageNegative	Pass	no
Test_04_SubsystemTest_ MessageDAO: testGetMessagePositve	Pass	no
Test_04_SubsystemTest_UserDAO: testGetMessageNegative	Pass	no

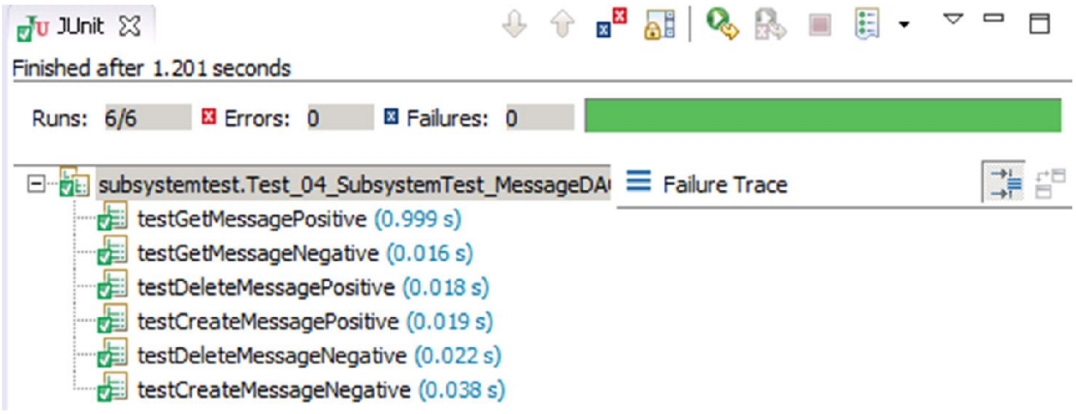
Class: Test_03_SubsystemTest_UserDAO.java

JUNIT test: all passed



Class: Test_04_SubsystemTest_MessageDAO.java

Junit Test: all passed



Unit Test

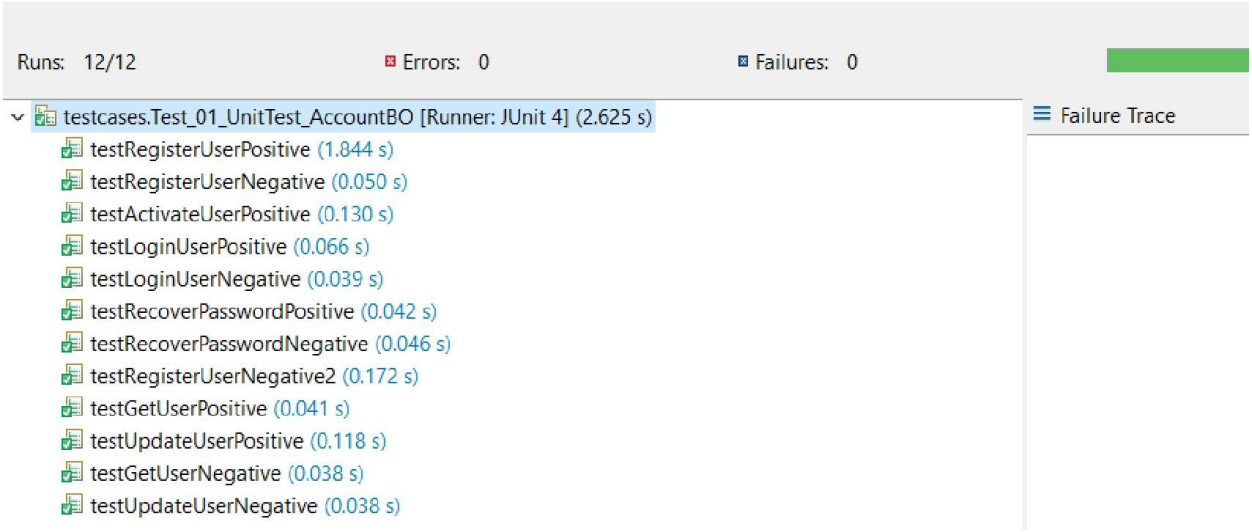
Test Case ID	Pass/Fail	Fix
Test_04_UnitTest_AccountBO: testLoginUserPositive	Pass	NO
Test_04_UnitTest_AccountBO: testLoginUserNegative	Pass	NO

Test_04_UnitTest_AccountBO: testActivateUserPositive	Pass	NO
Test_04_UnitTest_AccountBO: testRegisterUserPositive	Pass	NO
Test_04_UnitTest_AccountBO: testRegisterUserNegative	Pass	NO
Test_04_UnitTest_AccountBO: testGetUserPositive	Pass	NO
Test_04_UnitTest_AccountBO: testGetUserNegative	Pass	NO
Test_04_UnitTest_AccountBO: testRecoverPasswordNegative	Pass	NO
Test_04_UnitTest_AccountBO: testRecoverPasswordPositive	Pass	NO
Test_04_UnitTest_AccountBO: testUpdateUserPositive	Pass	NO
Test_04_UnitTest_AccountBO: testUpdateUserNegative	Pass	NO
Test_02_UnitTest_MessageBO: testSaveMessagePositive	Pass	NO
Test_02_UnitTest_MessageBO: testSaveMessageNegative	Pass	NO
Test_02_UnitTest_MessageBO: testGetMessagesToShowPositive	Pass	NO
Test_02_UnitTest_MessageBO: testGetMessagesToShowNegative	Pass	NO
Test_02_UnitTest_MessageBO: testDeleteMessagePositive	Pass	NO
Test_02_UnitTest_MessageBO: testDeleteMessageNegative	Pass	NO

Test_04_UnitTest_AccountBO: testRegisterUserNegative2	Pass	NO

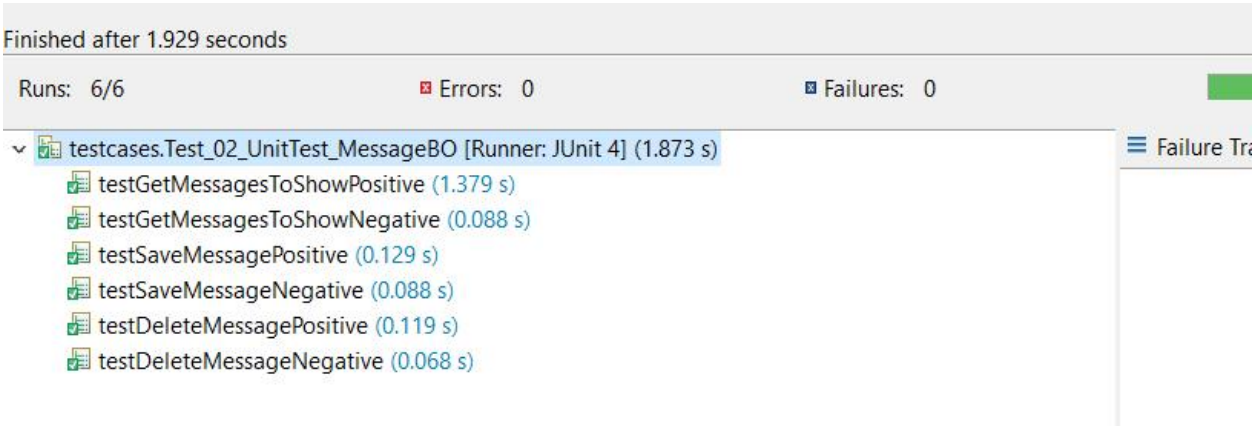
Class: Test_01_UnitTest_AccountBO.java

JUNIT test: all passed



Class: Test_01_UnitTest_AccountBO.java

JUNIT test: all passed

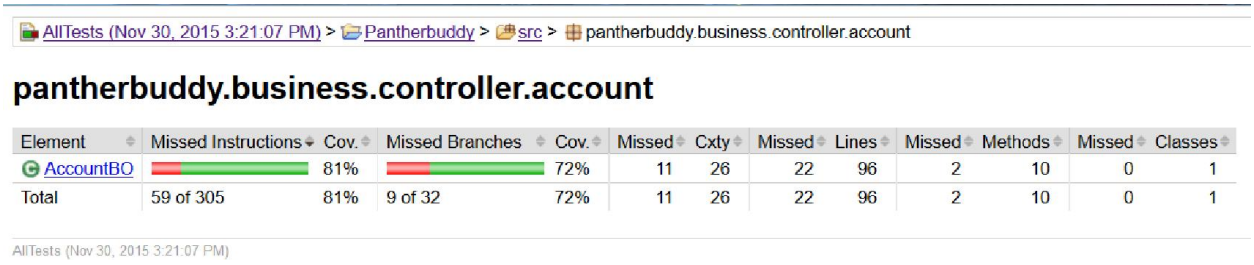


7.4.2. Code coverage and number of bugs

Type of testing	*Number of Test Cases*	*Number of Bugs*	*Code Coverage*

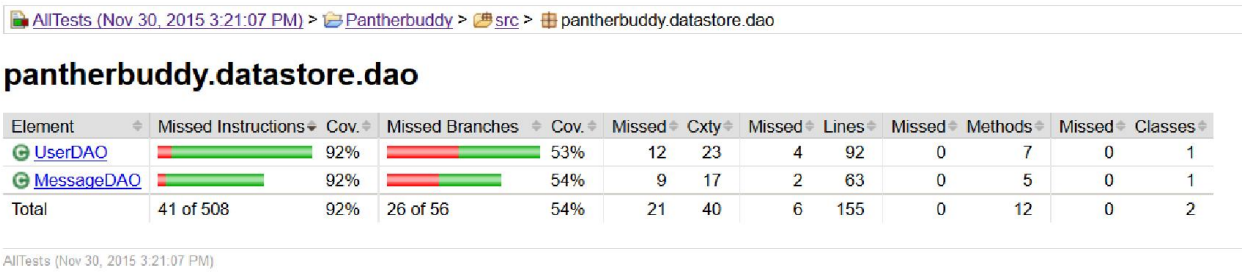
Subsystem test	16	0	92%
System test	33	0	Not applicable
Unit test	18	0	86.6%

Code coverage for unit test on AccountBO controller class.



Code coverage for unit test on MessageBO controller class.

Code coverage for subsystem test on data layer of the main 3-Tier architecture.



8. 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.
Bootstraps	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)

9. Signature page

TEAM NAME: Panther Buddy Project Team

Team members of the Panther Buddy participated in the preparation of this project, understand i contents, approve the plan as team's charter and operating plan, and agree to be held mutually accountable for adherence to the plan. Evidence of agreement is reflected by each team membe signature affixed below.

Yue Wu — *Team Lead Print name* — 11/29/15 — Date

Abhinav Dutt — *Team Member Name* — 11/30/15 — Date

kedari Sharath — *Team Member Name* — 11/30/15 — Date

ABDUR RAHMAN BIN SHAHLO — *Team Member Name* — 11/30/15 — Date

ALLAN SHAJI MANAMEL — *Team Member Name* — 11/30/15 — Date

YULONG QIU — *Team Member Name* — 11/30/15 — Date

James Angelo — *Team Member Name* — 11/30/15 — Date

Project Planning Document Approval History

Approving Party	Version Approved	Signature	Date
Project Manager			

Project Planning Document Review History

Reviewer	Version Reviewed	Signature	Date
Group Member	2.0		11/29/15
Group Member	2.0	Shwetha	11/29/15
Group Member	2.0		11/29/15
Group Member	2.0		11/30/15
Group Member	2.0.		11/30/15
Group Member	2.0		11/30/15
Group Member	2.0.		11/30/15

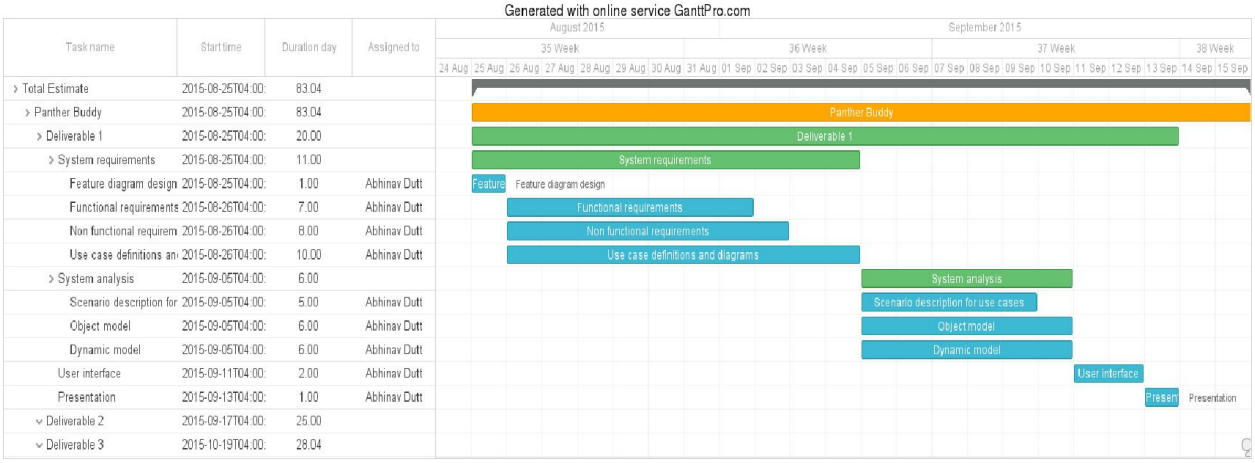
10. References

1. Acamedia.stackexchange. (2015). *Post list*. Retrieved from Stackexchange: http://academia.stackexchange.com/
2. Ambler, S. W. (2009). *UML 2 Class Diagrams.* Weldon .
3. Burnett, M. (2007). *Blocking Brute Force Attacks.* Retrieved from Virginia Tech CS: http://www.cs.virginia.edu/~csadmin/gen_support/brute_force.php
4. C, J. D., & Darwen, H. (1997). *A Guide to the SQL standard : a user's guide to the standard database language SQL* (Vol. 4). USA: Addison Wesley.
5. Craiglist. (n.d.). *Show Contact*. Retrieved from Craiglist: http://www.craigslist.org/about/sites
6. Facebook. (2015). *Report*. Retrieved from Facebook help center: https://www.facebook.com/help/181495968648557
7. FIU. (2015). *student housing*. Retrieved from FIU student affairs: http://housing.fiu.edu/
8. Google . (2014). *Google reCaptcha*. Retrieved from Google reCAPTCHA: https://www.google.com/recaptcha/intro/index.html
9. group, F. s. (2014). *FIU Students Selling Textbooks*. Retrieved from Facebook: https://www.facebook.com/groups/287168468030148/
10. Group, O. M. (2005). *UML Superstructure Specification.*
11. Holus Associates. (2007). *UML Reference Card* (Vol. 2). Retrieved March 12, 2011
12. Microsoft. (2011). *SQL injection.* Retrieved from technet Microsoft: https://technet.microsoft.com/en-us/library/ms161953(v=sql.105).aspx
13. Owasp. (2014, 04 22). *Cross-site Scripting (XSS).* Retrieved from Owasp: https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)
14. Stackexchange. (2015). *Questions (Post)*. Retrieved from Stackexchange: http://codegolf.stackexchange.com/questions/ask
15. Stackexchange. (2015). *Tagging post*. Retrieved from stackexchange: http://codegolf.stackexchange.com/tags
16. Tanji, M. (2013, 11). *CryptoLocker Decryption Engine.* Retrieved from KYRUS: http://www.kyrus-tech.com/2013/11/12/cryptolocker-decryption-engine/

17. Twitter. (n.d.). *Twitter Follow User*. Retrieved from Twitter: https://dev.twitter.com/web/follow-button

11. Appendix

11.1. Appendix A – Project schedule.



11.2. Appendix B – All 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_SC02_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_MC02_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_SC01_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_MC01_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

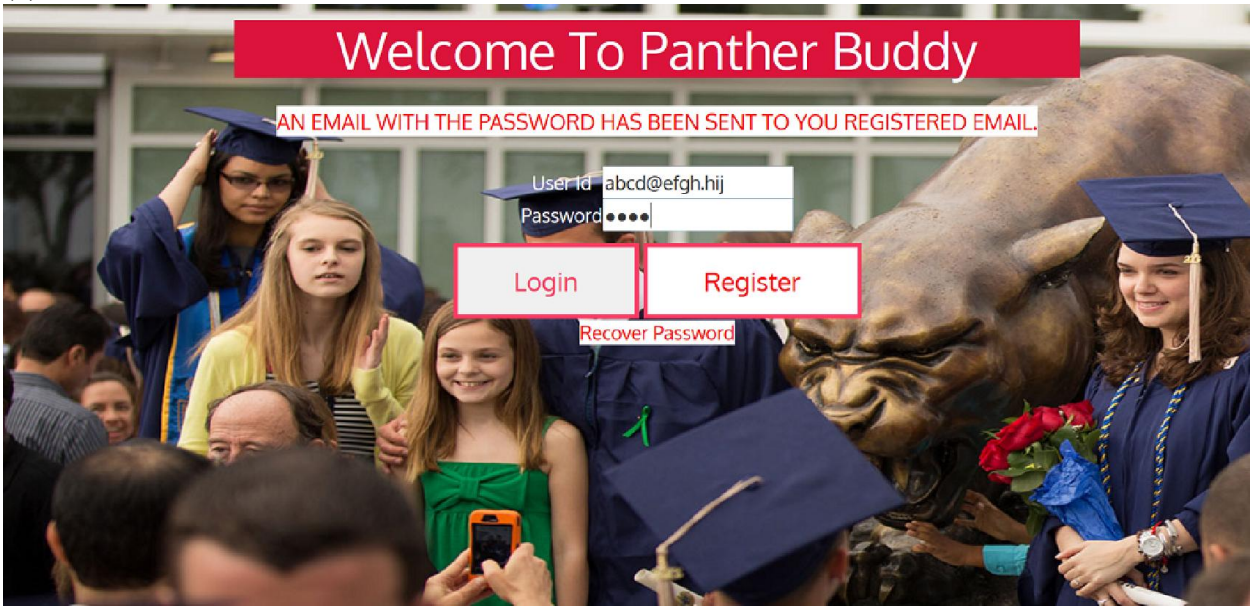
Criticality: Very Critical

Risk: High

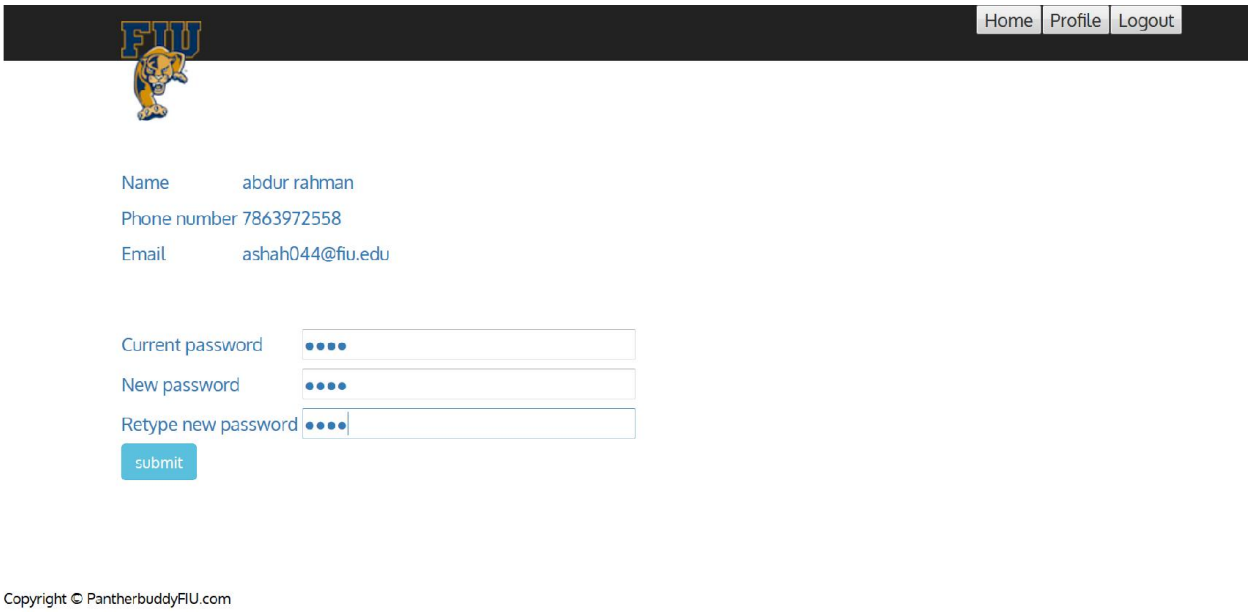
11.3. Appendix C – User Interface designs.

In this section we present the UIs of the use cases that we will implement in this project, including activate user, change password, delete post, login, invalid login, post message, recover password, registration, SQL injection, view own profile, view post and XSSfiltering.

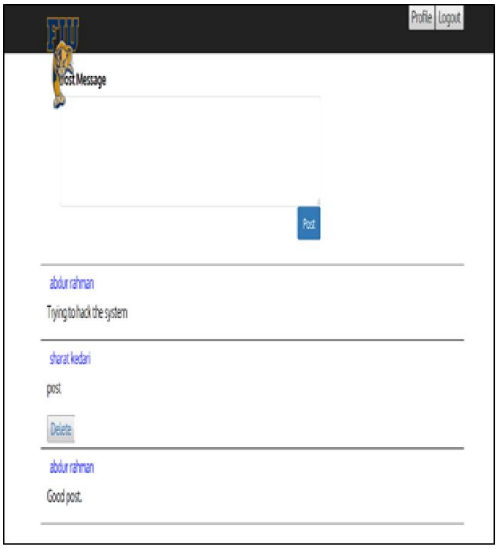
(1) Activate user

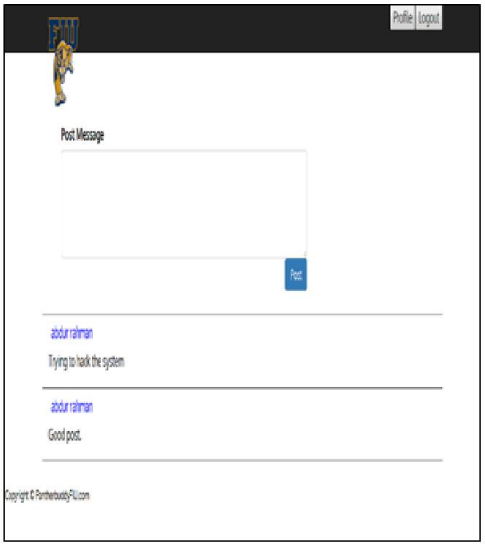


(2) Change password

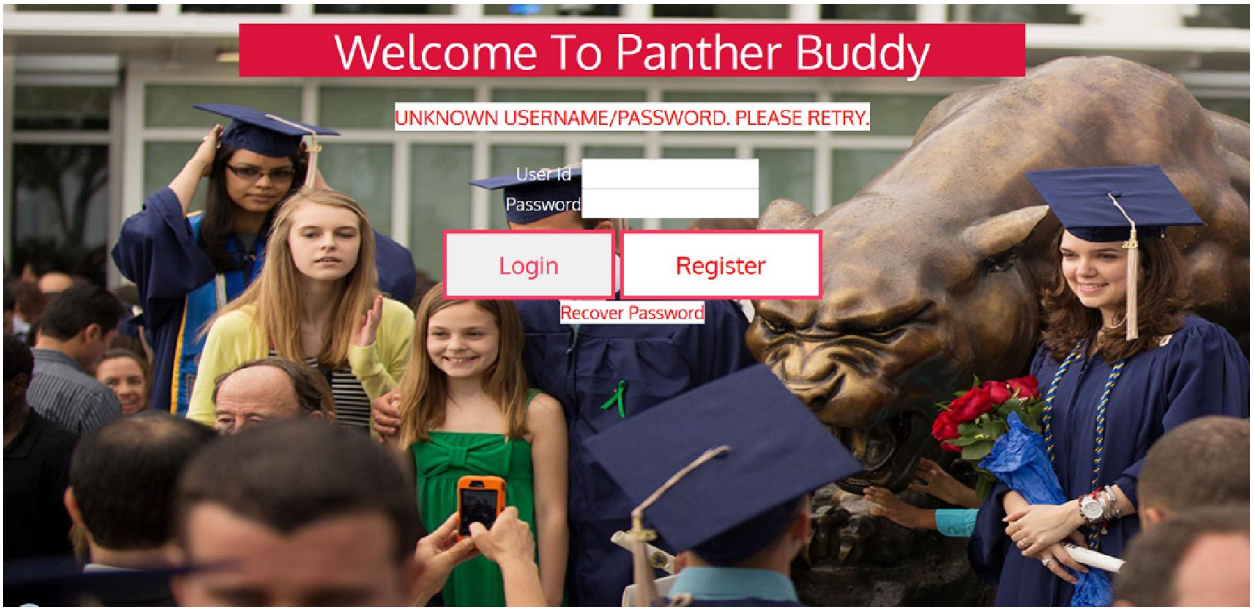


(3) Delete Post

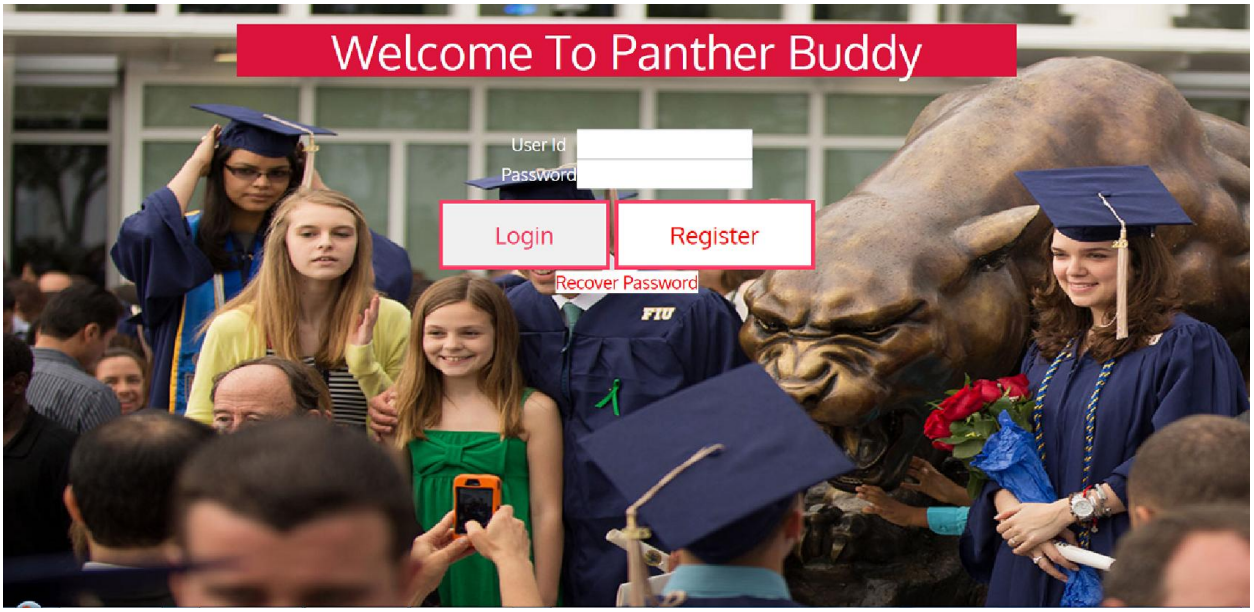




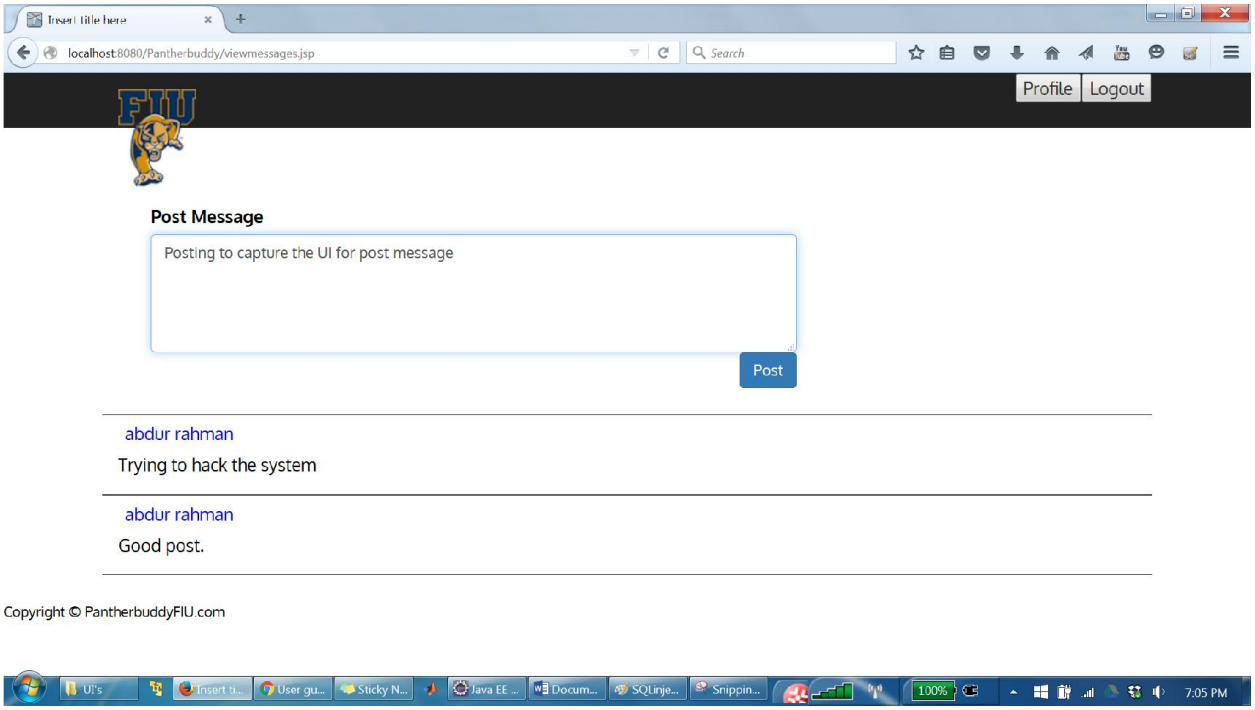
(4)Invalid Login



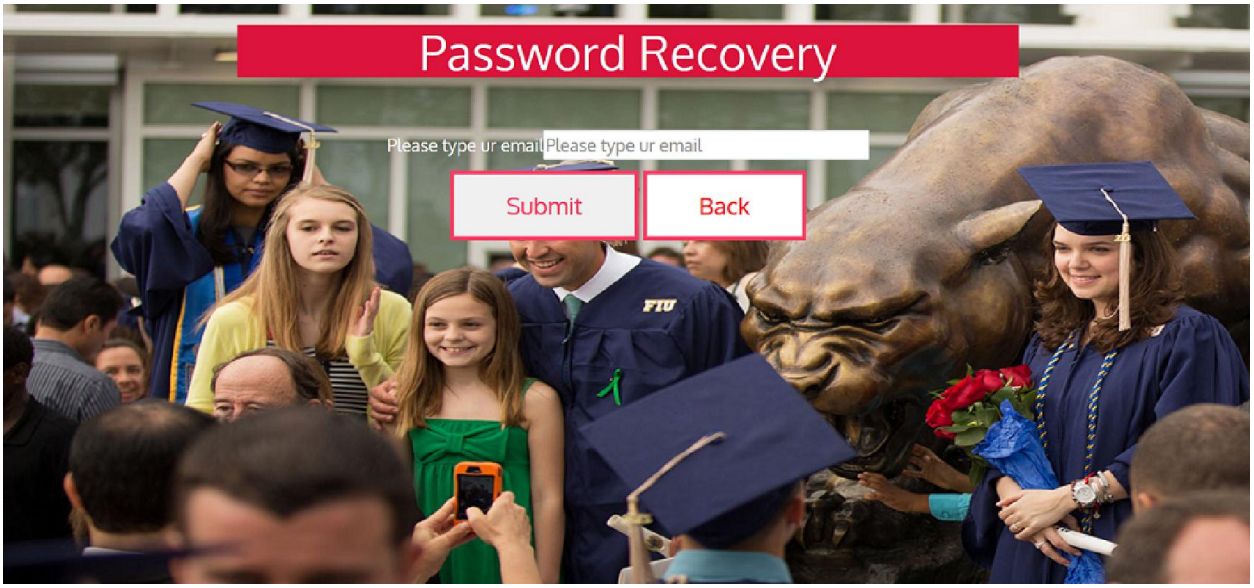
(5)Login



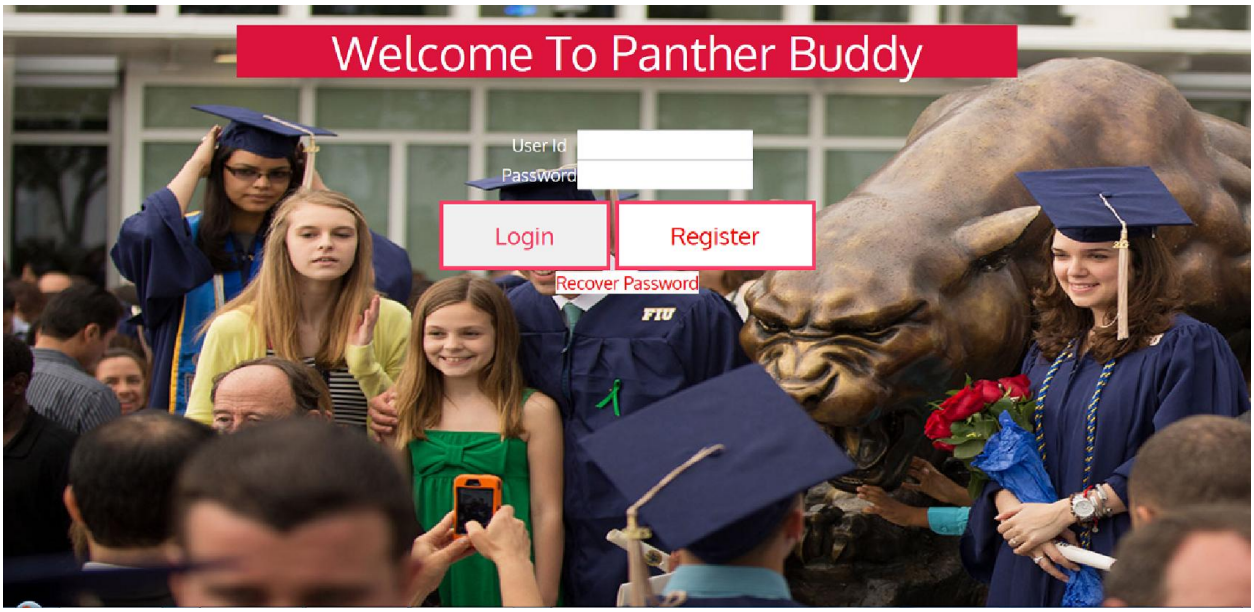
(6) Post Message



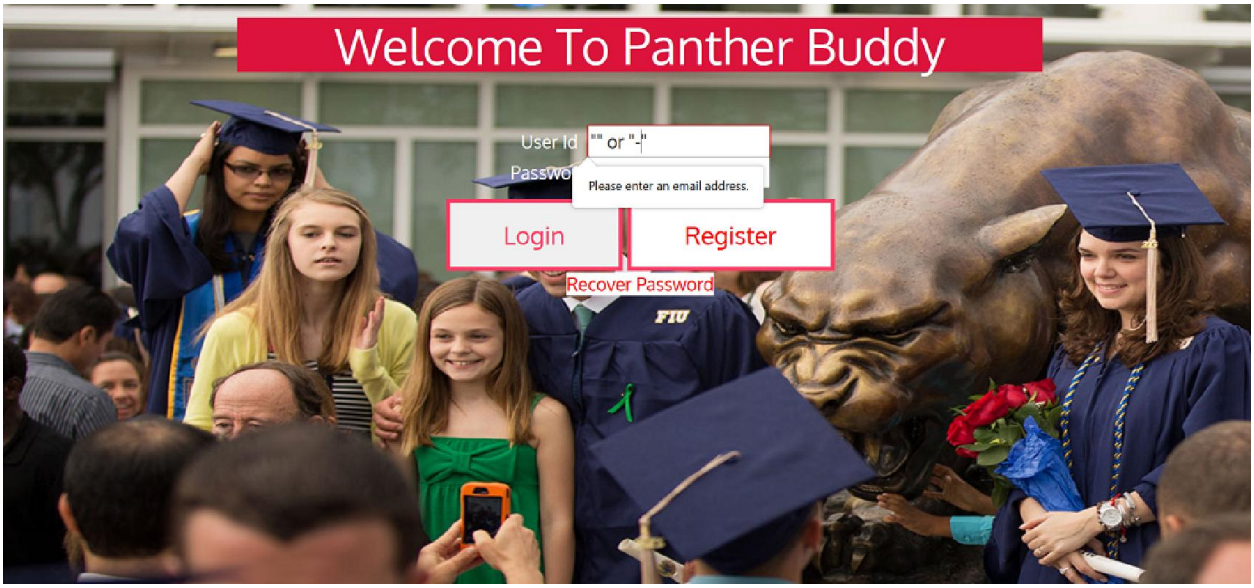
(7) Recover password



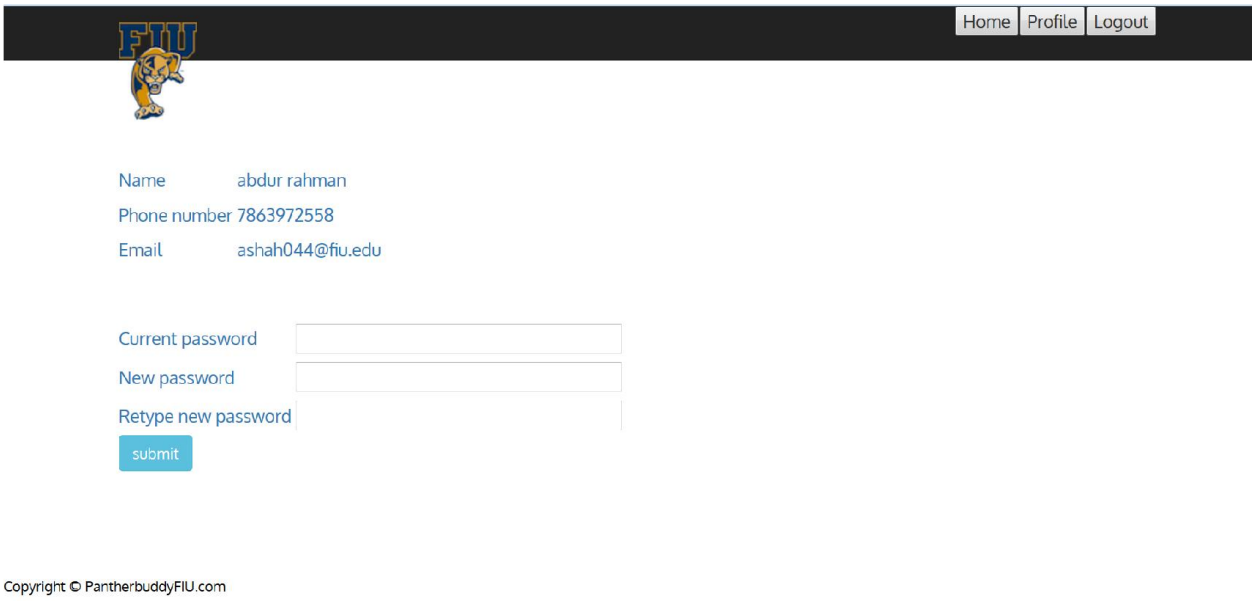
(8) Registration



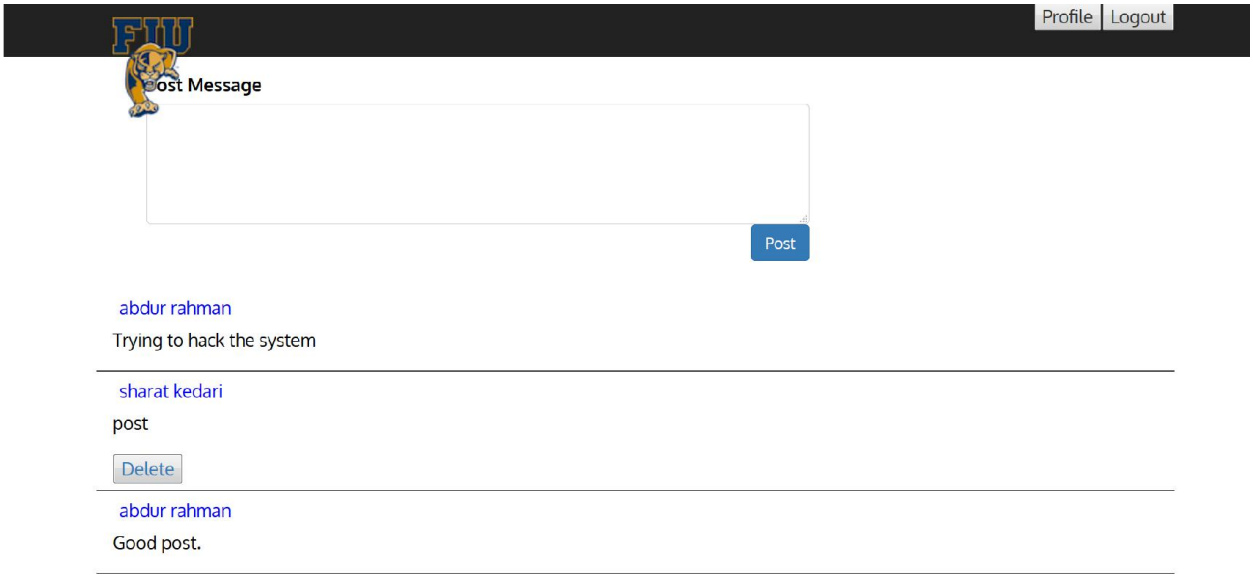
(9) SQL Injection



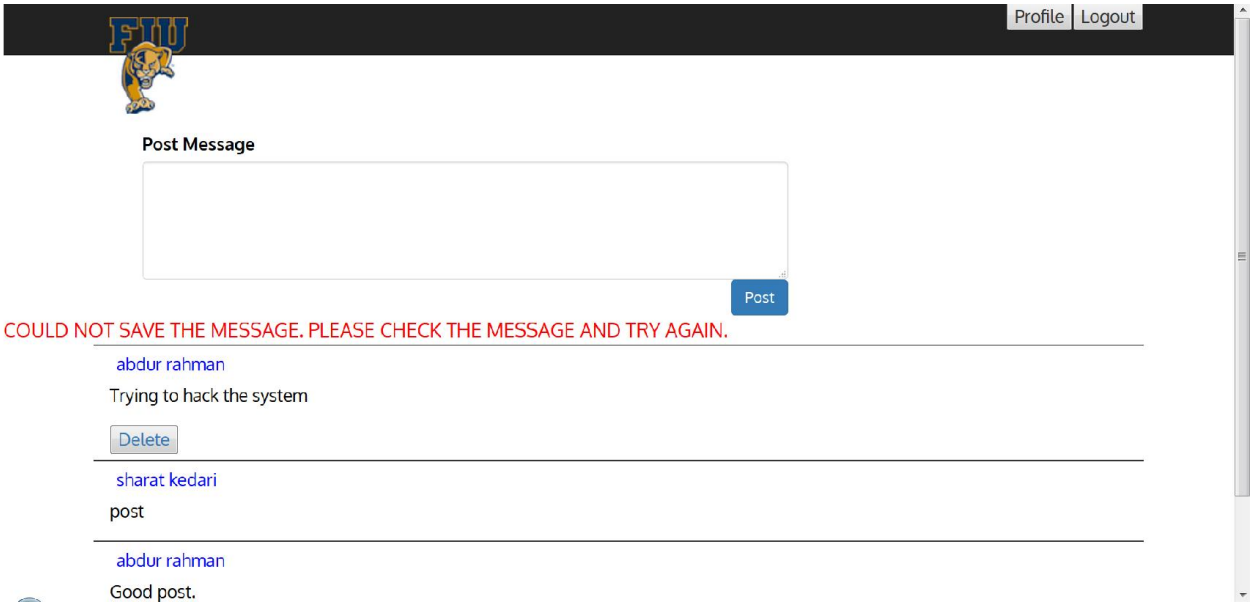
(10) View Own Profile



(11) View Post



(12) XSS Flitering



11.4. Appendix D – Detailed class Diagrams

This Appendix contains the detail class diagram.

1. High Level Package Architecture

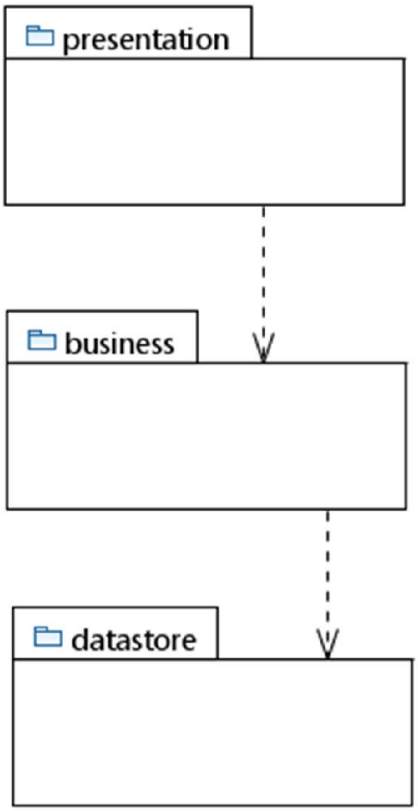


Figure: Shows the high level package architecture. It shows the 3-tier package architecture we use.

2. Presentation Package

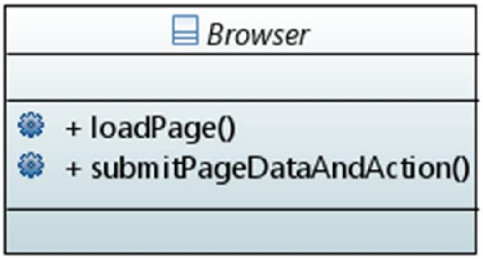


Figure: The abstract class Browser

3. Business Package

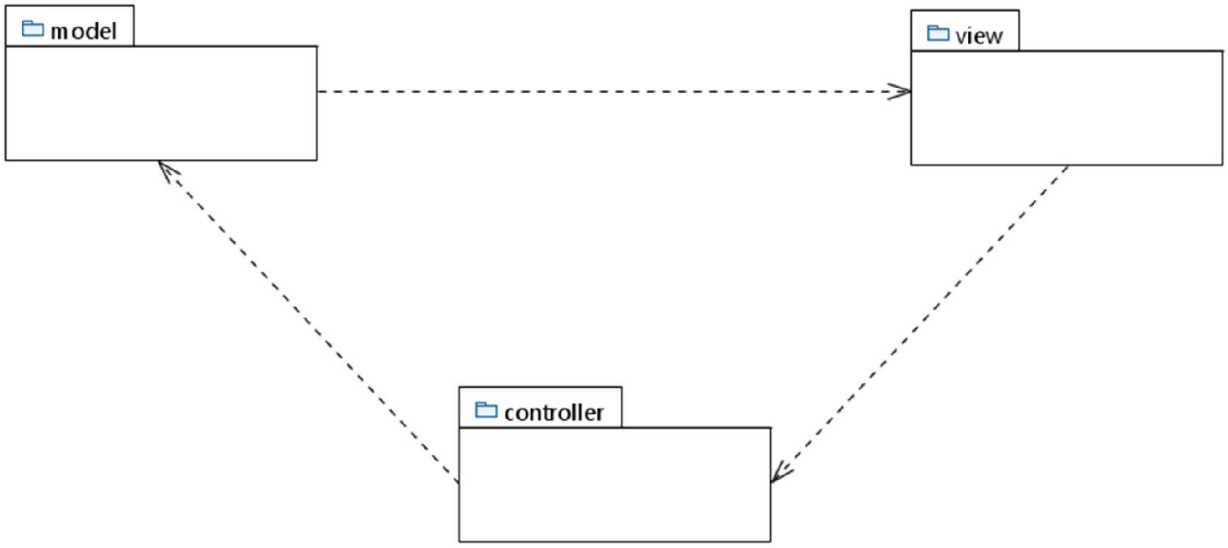


Figure: Shows the package structure present inside the business package.

4. Model Package

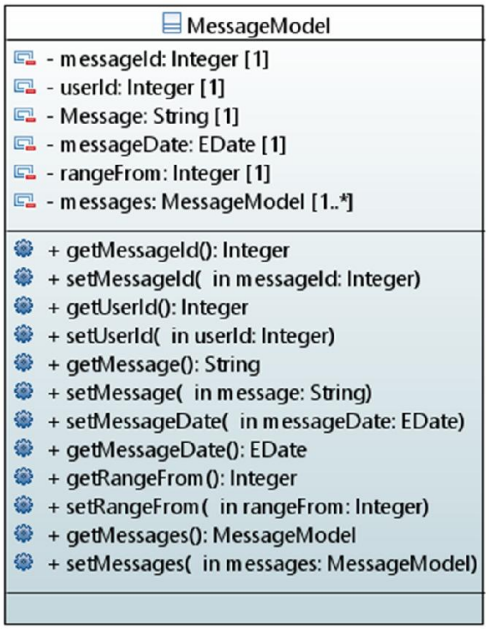


Figure: Shows the model classes of the MVC architecture implemented within the business tier

5. View Package

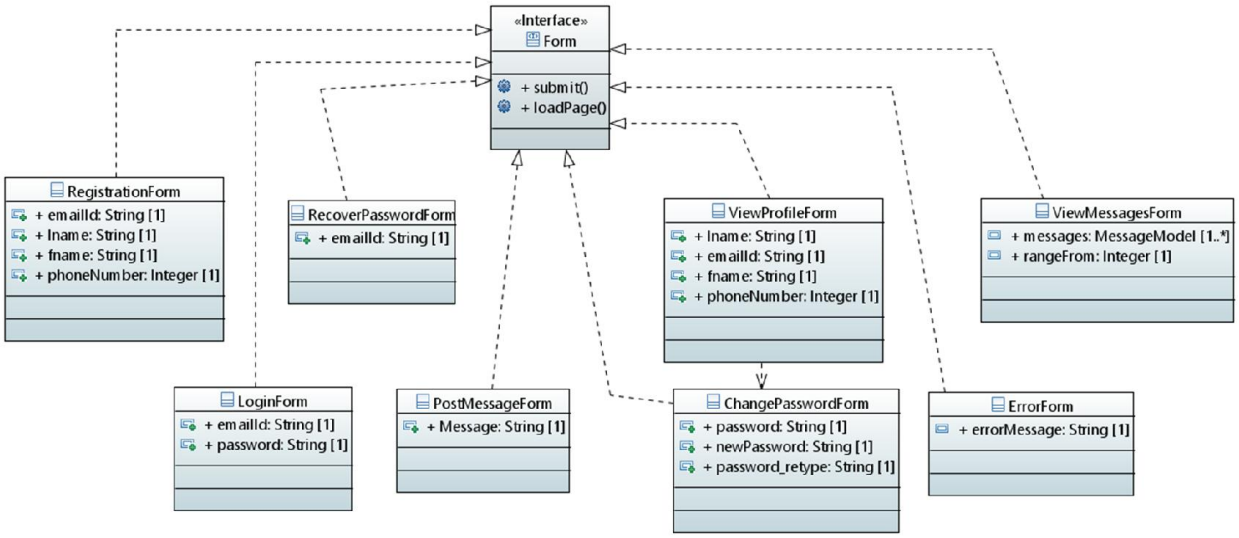


Figure: Shows the classes in the view package.

6. Controller Package

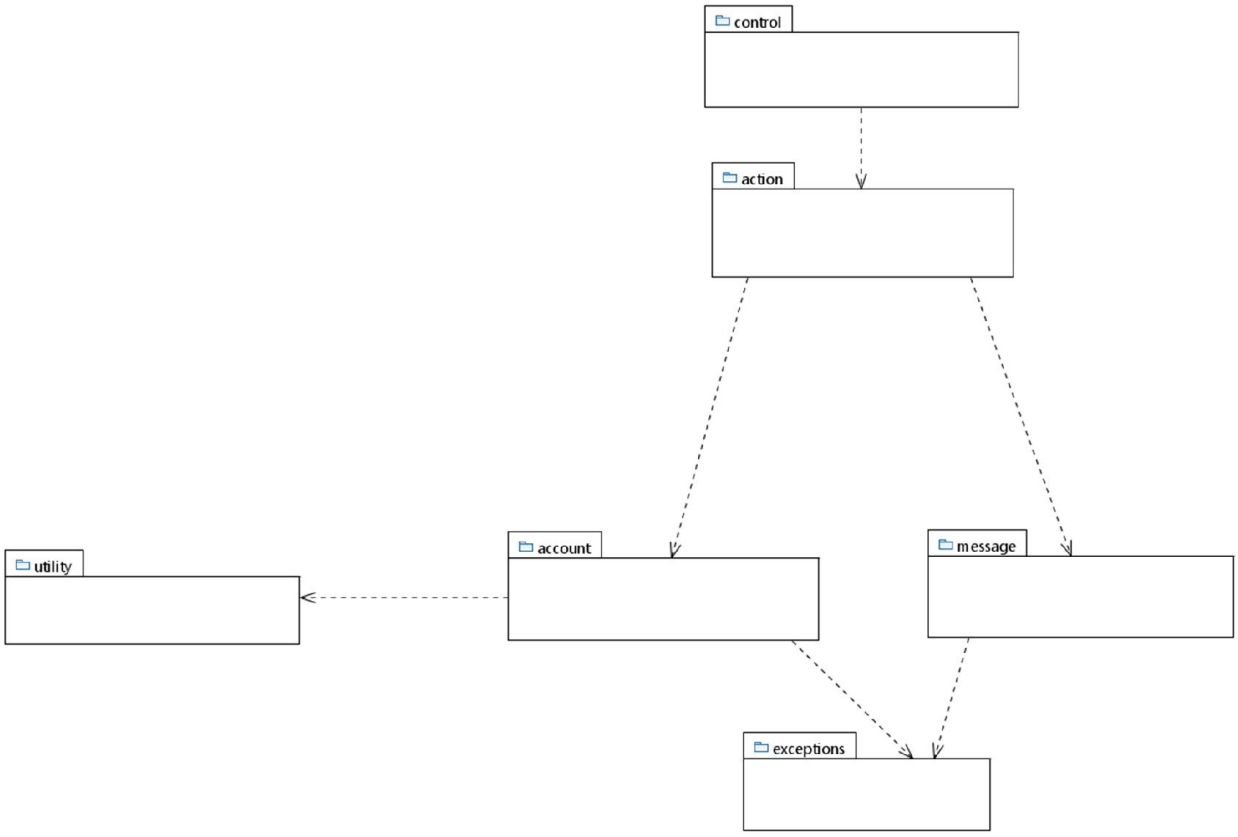


Figure: Shows the packages in the controller package.

7. Control Package

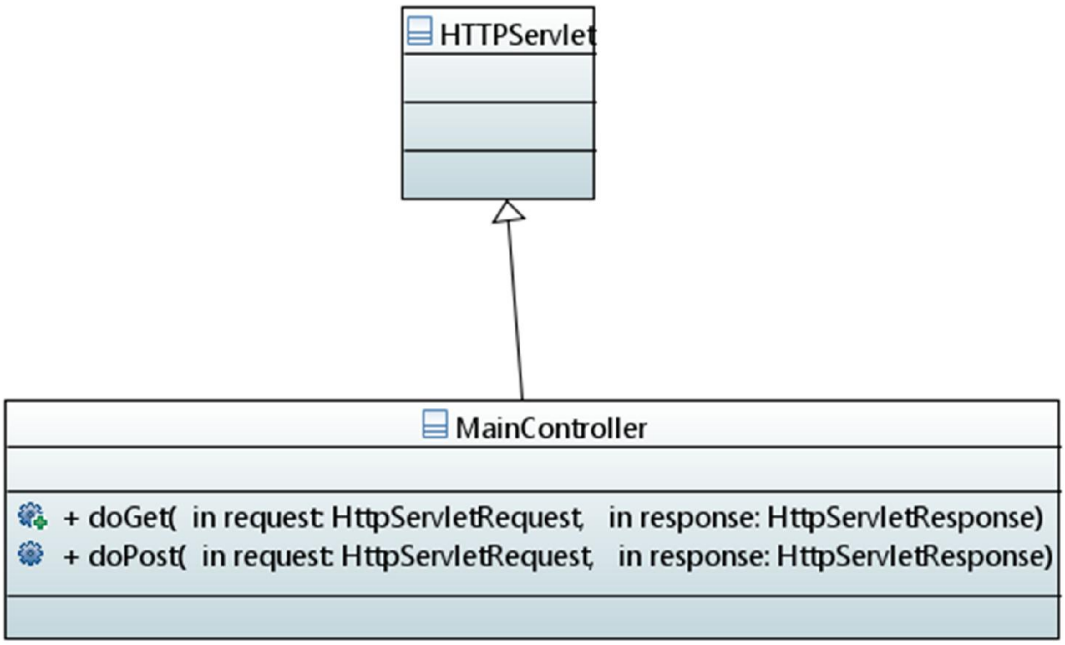


Figure: Shows the classes in the control package

8. Action package

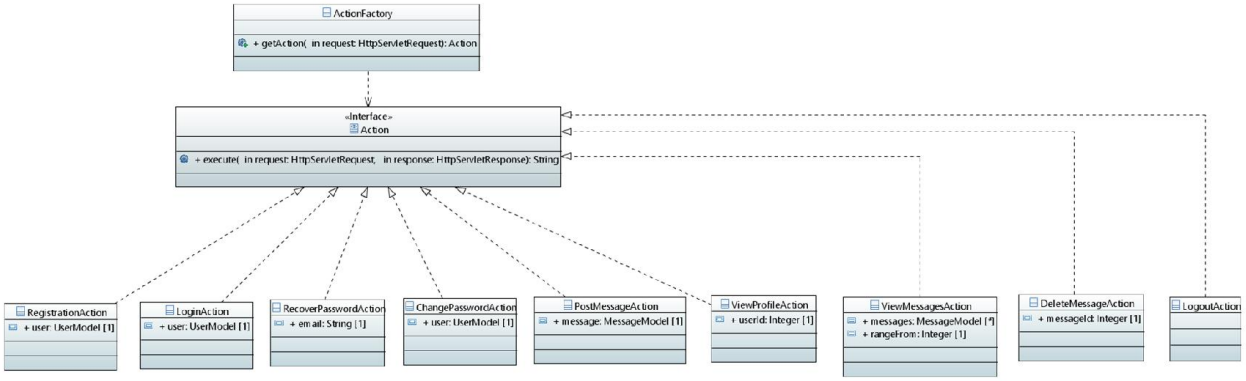


Figure: Shows the classes inside the action package

9. Account package

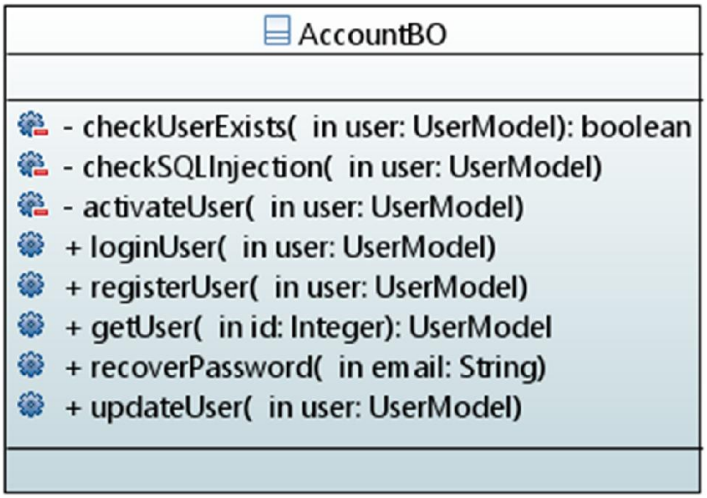


Figure: Shows the AccountBO class in account package.

10. Message package

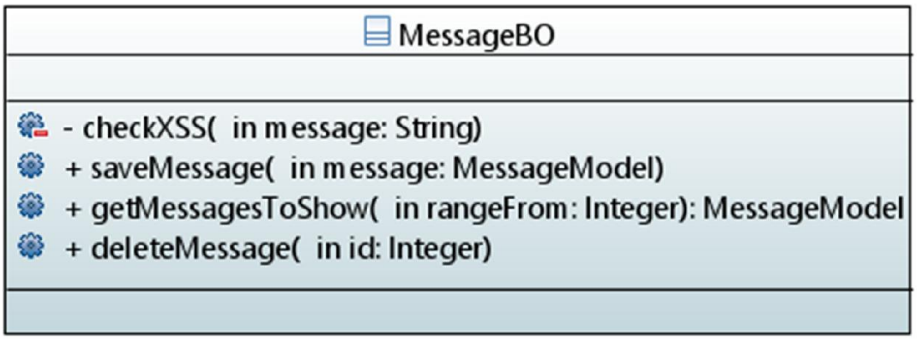


Figure: Shows the MessageBO class in message package

11. Utility package

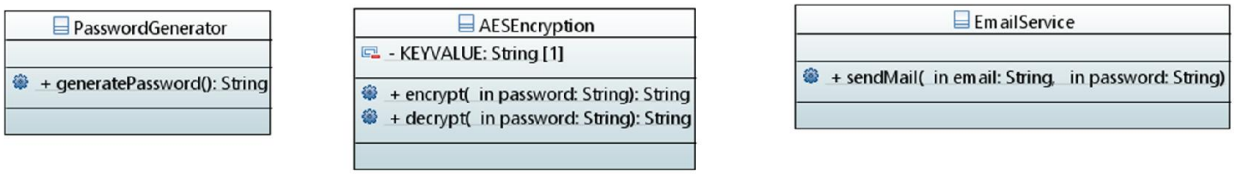


Figure: Shows the utility classes in the utility package

12. Exceptions package

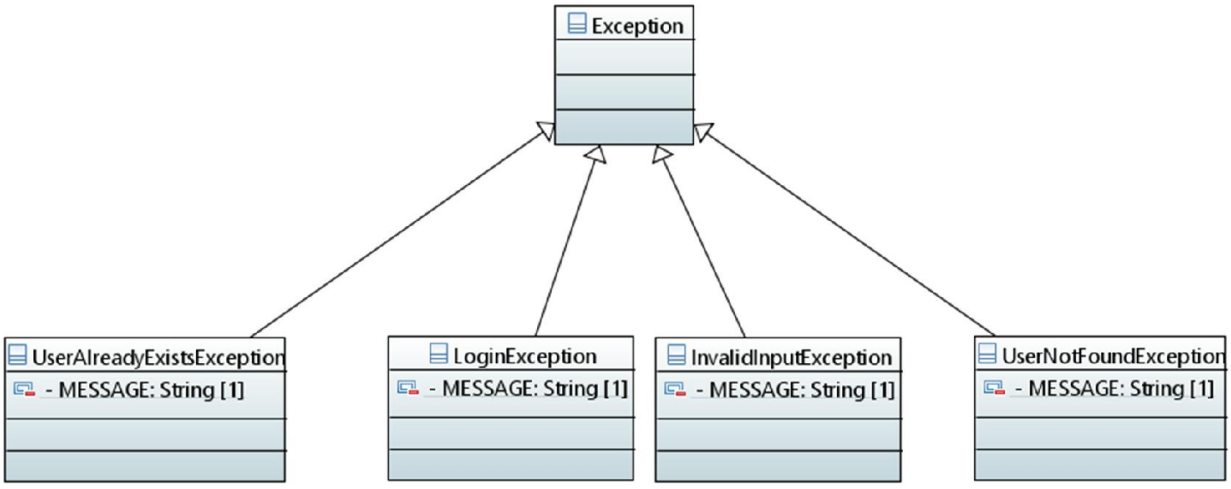


Figure: Shows the business exception classes of the system.

13. Datastore package

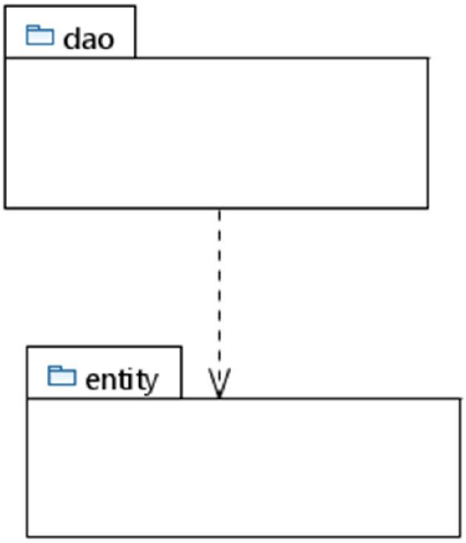


Figure: Shows the package structure of the data tier of the high level 3-tier architecture.

14. DAO package

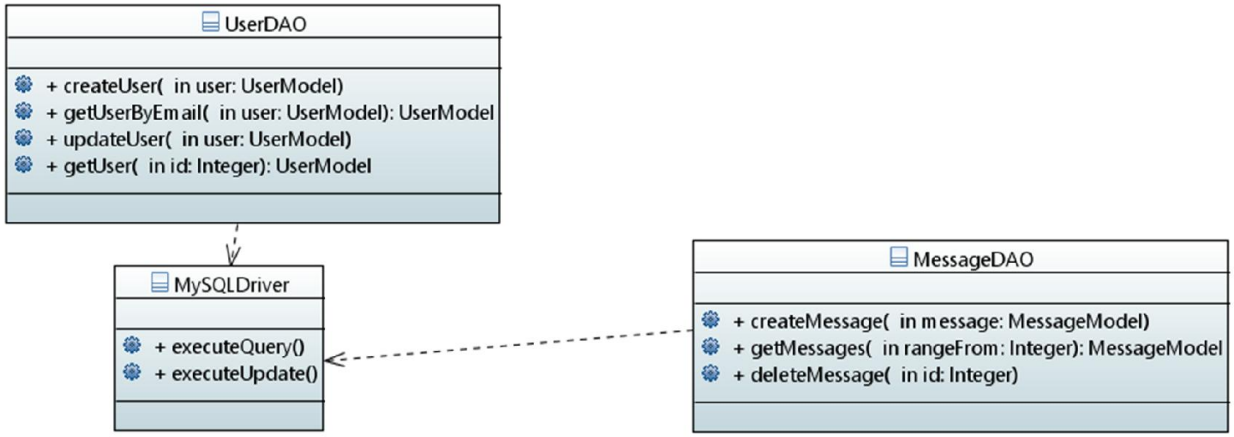


Figure: Shows the classes in the DAO package.

15. Entity package

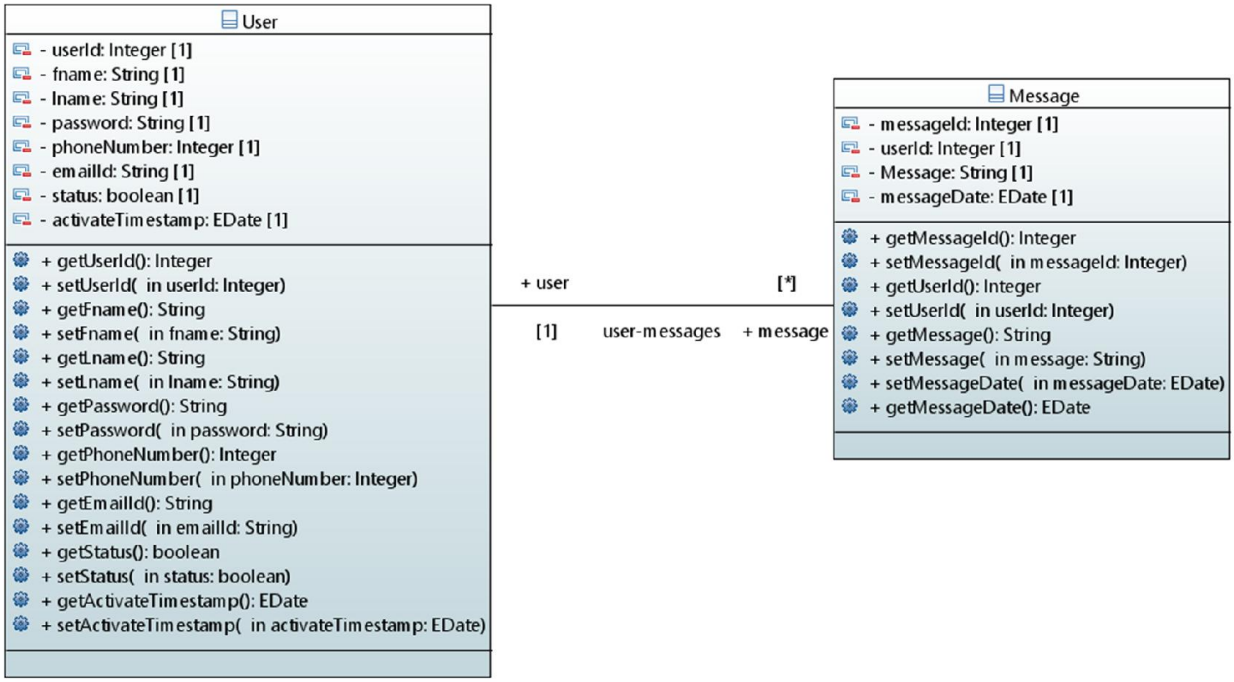


Figure: Shows the entity classes in the entity package.

11.5. Appendix E – Class Interfaces

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

11.6. Appendix F – Documented code for Test Driver.

Below is the code for test driver, for unit test and sub system test.

1. Unit test driver
   1.1. Test_01_UnitTest_AccountBO

```
2. package testcases;
3.
4. import static org.mockito.Matchers.any;
5. import static org.mockito.Mockito.when;
6.
7. import java.sql.Connection;
8. import java.sql.PreparedStatement;
9. import java.sql.SQLException;
10.      import java.util.Date;
11.
12.      import javax.mail.MessagingException;
13.      import javax.naming.NamingException;
14.
15.      import org.junit.After;
16.      import org.junit.Assert;
17.      import org.junit.Test;
18.      import org.mockito.Mockito;
19.
20.      import pantherbuddy.business.controller.account.AccountBO;
21.      import
   pantherbuddy.business.controller.exceptions.InvalidInputException;
22.      import
   pantherbuddy.business.controller.exceptions.LoginException;
23.      import
   pantherbuddy.business.controller.exceptions.UserAlreadyExistsException;
24.      import
   pantherbuddy.business.controller.exceptions.UserNotFoundException;
25.      import pantherbuddy.business.controller.utility.UpdateColumn;
26.      import pantherbuddy.business.model.UserModel;
27.      import pantherbuddy.datastore.dao.UserDAO;
28.
29.      public class Test_01_UnitTest_AccountBO {
30.
31.            @Test
32.            public void testLoginUserPositive() throws
   InstantiationException,
33.            IllegalAccessException, ClassNotFoundException,
   SQLException,
34.            LoginException {
35.                  UserModel model = new UserModel();
36.                  model.setEmailId("xyz@fiu.edu");
37.                  model.setPassword("dddd");
38.
39.                  AccountBO accountBO = Mockito.mock(AccountBO.class);
40.
       when(accountBO.checkUserExists(any())).thenCallRealMethod();
41.
       when(accountBO.loginUser(model)).thenCallRealMethod();
42.
```

```
43.                   Integer id = accountBO.loginUser(model);
44.
45.                   Assert.assertTrue(id == 1);
46.             }
47.
48.             @Test(expected = LoginException.class)
49.             public void testLoginUserNegative() throws
   InstantiationException,
50.             IllegalAccessException, ClassNotFoundException,
   SQLException,
51.             LoginException {
52.                   UserModel model = new UserModel();
53.                   model.setEmailId("xyz@fiu.edu");
54.                   model.setPassword("ddss");
55.
56.                   AccountBO accountBO = Mockito.mock(AccountBO.class);
57.
        when(accountBO.checkUserExists(any())).thenReturn(getUserModel())
   ;
58.
        when(accountBO.loginUser(model)).thenCallRealMethod();
59.                   accountBO.loginUser(model);
60.             }
61.
62.             @Test
63.             public void testActivateUserPositive() throws
   InstantiationException,
64.             IllegalAccessException, ClassNotFoundException,
   SQLException {
65.                   UserModel model = new UserModel();
66.                   model.setEmailId("test@gmail.com");
67.                   model.setPassword("dddd");
68.                   model.setUserId(2);
69.
70.                   AccountBO accountBO = Mockito.mock(AccountBO.class);
71.
        when(accountBO.activateUser(model)).thenCallRealMethod();
72.
73.                   boolean success = accountBO.activateUser(model);
74.
75.                   UserDAO dao = new UserDAO();
76.                   UserModel userModel = dao.getUser(2);
77.
78.                   Assert.assertTrue(success);
79.                   Assert.assertTrue(userModel.isStatus());
80.             }
81.
82.             @Test
83.             public void testRegisterUserPositive() throws
   InstantiationException,
84.             IllegalAccessException, ClassNotFoundException,
   SQLException,
85.             UserAlreadyExistsException, InvalidInputException,
86.             MessagingException, NamingException {
87.                   UserModel model = new UserModel();
88.                   model.setEmailId("xyz1@fiu.edu");
89.                   model.setFname("Abhinav");
```

```
90.                   model.setLname("Dutt");
91.                   model.setPhoneNumber(1234567890L);
92.
93.                   AccountBO accountBO = Mockito.mock(AccountBO.class);
94.
        when(accountBO.checkUserExists(any(UserModel.class))).thenReturn(
   null);
95.
        Mockito.when(accountBO.registerUser(any())).thenCallRealMethod();
96.                   Mockito.doNothing().when(accountBO).sendMail(any(),
   any());
97.
98.                   boolean success = accountBO.registerUser(model);
99.                   Assert.assertTrue(success);
100.            }
101.
102.            @Test(expected = UserAlreadyExistsException.class)
103.            public void testRegisterUserNegative() throws
   InstantiationException,
104.            IllegalAccessException, ClassNotFoundException,
   SQLException,
105.            UserAlreadyExistsException, InvalidInputException,
106.            MessagingException, NamingException {
107.                  UserModel model = new UserModel();
108.                  model.setEmailId("xyz2@fiu.edu");
109.                  model.setFname("Abdur");
110.                  model.setLname("Rehman");
111.                  model.setPhoneNumber(1224567890L);
112.
113.                  AccountBO accountBO = Mockito.mock(AccountBO.class);
114.
        when(accountBO.checkUserExists(any(UserModel.class))).thenReturn(
   model);
115.
        Mockito.when(accountBO.registerUser(any())).thenCallRealMethod();
116.                  Mockito.doNothing().when(accountBO).sendMail(any(),
   any());
117.                  accountBO.registerUser(model);
118.            }
119.
120.            @Test(expected = InvalidInputException.class)
121.            public void testRegisterUserNegative2() throws
   InstantiationException,
122.            IllegalAccessException, ClassNotFoundException,
   SQLException,
123.            UserAlreadyExistsException, InvalidInputException,
124.            MessagingException, NamingException {
125.
126.                  UserModel model = new UserModel();
127.                  model.setEmailId("xyz1@fiu.edu");
128.                  model.setFname("Abhinav");
129.                  model.setLname("Dutt");
130.                  model.setPhoneNumber(1234567890L);
131.
132.                  AccountBO accountBO = Mockito.mock(AccountBO.class);
```

```
133.
        when(accountBO.checkUserExists(any(UserModel.class))).thenReturn(
   null);
134.
        Mockito.when(accountBO.registerUser(any())).thenCallRealMethod();
135.
        Mockito.doThrow(MessagingException.class).when(accountBO).sendMai
   l(any(), any());
136.                    accountBO.registerUser(model);
137.            }
138.
139.            @Test
140.            public void testGetUserPositive() throws
   InvalidInputException {
141.                    AccountBO accountBO = Mockito.mock(AccountBO.class);
142.
        Mockito.when(accountBO.getUser(any())).thenCallRealMethod();
143.                    UserModel model = accountBO.getUser(1);
144.                    Assert.assertTrue(model.getUserId() == 1);
145.            }
146.
147.            @Test(expected = InvalidInputException.class)
148.            public void testGetUserNegative() throws
   InvalidInputException {
149.                    AccountBO accountBO = Mockito.mock(AccountBO.class);
150.
        Mockito.when(accountBO.getUser(any())).thenCallRealMethod();
151.                    accountBO.getUser(3);
152.            }
153.
154.            @Test
155.            public void testRecoverPasswordPositive() throws
   UserNotFoundException,
156.            MessagingException, NamingException {
157.                    String email = "xyz@fiu.edu";
158.
159.                    AccountBO accountBO = Mockito.mock(AccountBO.class);
160.
        Mockito.when(accountBO.recoverPassword(any())).thenCallRealMethod
   ();
161.                    Mockito.doNothing().when(accountBO).sendMail(any(),
   any());
162.                    boolean success = accountBO.recoverPassword(email);
163.                    Assert.assertTrue(success);
164.            }
165.
166.            @Test(expected = UserNotFoundException.class)
167.            public void testRecoverPasswordNegative() throws
   MessagingException,
168.            NamingException, UserNotFoundException {
169.                    String email = "notexist@fiu.edu";
170.
171.                    AccountBO accountBO = Mockito.mock(AccountBO.class);
172.
        Mockito.when(accountBO.recoverPassword(any())).thenCallRealMethod
   ();
```

```
173.                 Mockito.doNothing().when(accountBO).sendMail(any(),
   any());
174.                 boolean success = accountBO.recoverPassword(email);
175.                 Assert.assertTrue(success);
176.           }
177.
178.           @Test
179.           public void testUpdateUserPositive() throws
   InvalidInputException {
180.                 UserModel model = new UserModel();
181.                 model.setUserId(1);
182.                 model.setPassword("dddd");
183.                 model.setNewPassword("ssss");
184.                 model.setPassword_retype("ssss");
185.
186.                 AccountBO accountBO = Mockito.mock(AccountBO.class);
187.
      when(accountBO.updateUser(any())).thenCallRealMethod();
188.                 boolean success = accountBO.updateUser(model);
189.                 Assert.assertTrue(success);
190.           }
191.
192.           @Test(expected = InvalidInputException.class)
193.           public void testUpdateUserNegative() throws
   InvalidInputException {
194.                 UserModel model = new UserModel();
195.                 model.setUserId(1);
196.                 model.setPassword("ddss");
197.                 model.setNewPassword("ssss");
198.                 model.setPassword_retype("ssss");
199.
200.                 AccountBO accountBO = Mockito.mock(AccountBO.class);
201.
      when(accountBO.updateUser(any())).thenCallRealMethod();
202.                 accountBO.updateUser(model);
203.           }
204.
205.           private UserModel getUserModel() {
206.                 UserModel model = new UserModel();
207.                 model.setActivateTimestamp(new Date());
208.                 model.setEmailId("xyz@fiu.edu");
209.                 model.setFname("Allan");
210.                 model.setLname("Shaji");
211.                 model.setPassword("8oLsJNn7V7UMU9+tRhQFGg==");
212.                 model.setPhoneNumber(7894561L);
213.                 model.setStatus(false);
214.                 model.setUserId(1);
215.                 return model;
216.           }
217.
218.           @After
219.           public void tearDown() throws Exception {
220.
221.                 UserDAO dao = new UserDAO();
222.
223.                 UserModel model1 = new UserModel();
224.                 model1.setEmailId("xyz1@fiu.edu");
```

```
225.                 model1 = dao.getUserByEmail(model1);
226.                 if (model1 != null)
227.                       dao.deleteUser(model1.getUserId());
228.
229.                 UserModel model2 = new UserModel();
230.                 model2.setUserId(1);
231.                 model2.setPassword("8oLsJNn7V7UMU9+tRhQFGg==");
232.                 dao.updateUser(model2, UpdateColumn.PASSWORD);
233.
234.                 Connection connection = dao.getConnection();
235.                 PreparedStatement preparedStatement = connection
236.                             .prepareStatement("UPDATE
   pantherbuddy.user SET status = FALSE WHERE userid = 2;");
237.                 preparedStatement.executeUpdate();
238.                 connection.close();
239.           }
240.     }
```

1.2 Test_02_UnitTest_MessageBO

```
2. package testcases;
3.
4. import java.sql.Connection;
5. import java.sql.DriverManager;
6. import java.sql.PreparedStatement;
7. import java.sql.SQLException;
8. import java.util.List;
9.
10.   import org.junit.After;
11.   import org.junit.Assert;
12.   import org.junit.Before;
13.   import org.junit.Test;
14.   import org.mockito.Mockito;
15.
16.   import
   pantherbuddy.business.controller.exceptions.InvalidInputException;
17.   import pantherbuddy.business.controller.message.MessageBO;
18.   import pantherbuddy.business.model.MessageModel;
19.
20.   public class Test_02_UnitTest_MessageBO {
21.
22.         @Before
23.         public void setUp() throws Exception {
24.               Class.forName("com.mysql.jdbc.Driver").newInstance();
25.               Connection conn = DriverManager
26.
      .getConnection("jdbc:mysql://localhost/pantherbuddy?"
27.                                         +
   "user=allanshaji&password=o1pktp");
28.
29.               PreparedStatement preparedStatement = conn
30.                           .prepareStatement("DELETE FROM
   pantherbuddy.message");
31.               int rs = preparedStatement.executeUpdate();
32.
```

```
33.                 MessageModel message = new MessageModel();
34.                 message.setMessage("1 Message from test");
35.                 message.setUserId(1);
36.
37.                 PreparedStatement preparedStatement2 = conn
38.                             .prepareStatement("INSERT INTO
   pantherbuddy.message values (100,?,?,NOW())");
39.                 preparedStatement2.setInt(1, message.getUserId());
40.                 preparedStatement2.setString(2,
   message.getMessage());
41.                 int rs2 = preparedStatement2.executeUpdate();
42.
43.                 conn.close();
44.           }
45.
46.           @After
47.           public void tearDown() throws Exception {
48.           }
49.
50.           @Test
51.           public void testSaveMessagePositive() throws
   InstantiationException,
52.           IllegalAccessException, ClassNotFoundException,
   SQLException,
53.           InvalidInputException {
54.                 MessageModel message = new MessageModel();
55.                 message.setMessage("2 Message from test");
56.                 message.setUserId(1);
57.
58.                 MessageBO bo = Mockito.mock(MessageBO.class);
59.
      Mockito.when(bo.saveMessage(Mockito.any())).thenCallRealMethod();
60.
61.                 boolean success = bo.saveMessage(message);
62.                 Assert.assertTrue(success);
63.           }
64.
65.           @Test(expected = InvalidInputException.class)
66.           public void testSaveMessageNegative() throws
   InstantiationException,
67.           IllegalAccessException, ClassNotFoundException,
   SQLException,
68.           InvalidInputException {
69.                 MessageModel message = new MessageModel();
70.                 message.setMessage("<script>");
71.                 message.setUserId(1);
72.
73.                 MessageBO bo = Mockito.mock(MessageBO.class);
74.
      Mockito.when(bo.saveMessage(Mockito.any())).thenCallRealMethod();
75.
76.                 bo.saveMessage(message);
77.           }
78.
79.           @Test
80.           public void testGetMessagesToShowPositive() {
81.                 MessageBO bo = Mockito.mock(MessageBO.class);
```

```
82.
83.
      Mockito.when(bo.getMessagesToShow(Mockito.any())).thenCallRealMet
   hod();
84.
85.               List<MessageModel> list = bo.getMessagesToShow(0);
86.
87.               Assert.assertTrue(list != null);
88.               boolean found = false;
89.               for(MessageModel model : list){
90.                     if(model.getMessageId() == 100){
91.                           found = true;
92.                     }
93.               }
94.               if(!found){
95.                     Assert.fail();
96.               }
97.         }
98.
99.         @Test
100.        public void testGetMessagesToShowNegative() {
101.              MessageBO bo = Mockito.mock(MessageBO.class);
102.
103.
      Mockito.when(bo.getMessagesToShow(Mockito.any())).thenCallRealMet
   hod();
104.
105.              List<MessageModel> list = bo.getMessagesToShow(20);
106.
107.              Assert.assertTrue(list != null && list.isEmpty());
108.        }
109.
110.        @Test
111.        public void testDeleteMessagePositive() {
112.              MessageBO bo = Mockito.mock(MessageBO.class);
113.
114.
      Mockito.when(bo.deleteMessage(Mockito.any())).thenCallRealMethod(
   );
115.
116.              boolean success = bo.deleteMessage(100);
117.
118.              Assert.assertTrue(success);
119.        }
120.
121.        @Test
122.        public void testDeleteMessageNegative() {
123.              MessageBO bo = Mockito.mock(MessageBO.class);
124.
125.
      Mockito.when(bo.deleteMessage(Mockito.any())).thenCallRealMethod(
   );
126.
127.              boolean success = bo.deleteMessage(200);
128.
129.              Assert.assertTrue(!success);
130.        }
```

```
131.
132.  }
```

2. Subsystem Test Driver

2.1. Test_03_SubsystemTest_UserDAO

```
package subsystemtest;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;

import org.junit.After;
import org.junit.Assert;
import org.junit.Before;
import org.junit.Test;

import pantherbuddy.business.controller.utility.UpdateColumn;
import pantherbuddy.business.model.UserModel;
import pantherbuddy.datastore.dao.UserDAO;

public class Test_03_SubsystemTest_UserDAO {
	private Connection conn;
	private UserDAO userdao;

	@Before
	public void setUp() throws Exception{
		userdao = new UserDAO();
		conn =
DriverManager.getConnection("jdbc:mysql://localhost/pantherbuddy?"
                                    + "user=allanshaji&password=o1pktp");
	}

	@After
	public void tearDown() throws Exception{
		if(!conn.isClosed())
			conn.close();
	}

	@Test
	public void testCreateUserPositive() throws SQLException{
		UserModel newuser = new UserModel();
		newuser.setFname("qiu");
		newuser.setLname("qiu");
		newuser.setPassword("123");
		newuser.setPhoneNumber(7869256598L);
		newuser.setEmailId("qiu15@gmail.com");

		try {
			userdao.createUser(newuser);
		} catch (Exception e) {
			Assert.fail();
```

```
		}
	}

	@Test(expected = Exception.class)
	public void testCreateUserNegative() throws SQLException,
InstantiationException, IllegalAccessException, ClassNotFoundException{
		UserModel newuser = new UserModel();
		newuser.setFname("qiu");
		newuser.setLname("qiu");
		newuser.setPassword("123");
		newuser.setPhoneNumber(7869256598L);
		newuser.setEmailId("qiu@gmail.com");

		userdao.createUser(newuser);
	}

	@Test
	public void testDeleteUserPositive() throws SQLException,
InstantiationException, IllegalAccessException, ClassNotFoundException{
		Integer userid = 39;
		try{
			userdao.deleteUser(userid);
		}catch(Exception e){
			Assert.fail();
		}
	}

	@Test(expected = Exception.class)
	public void testDeleteUserNegative() throws SQLException,
InstantiationException, IllegalAccessException, ClassNotFoundException{
		Integer userid = 9999;
		userdao.deleteUser(userid);
	}

	@Test
	public void testUpdateUserPositive() throws SQLException,
InstantiationException, IllegalAccessException, ClassNotFoundException{
		UserModel newuser = new UserModel();
		Integer uid = 1;
		newuser.setUserId(uid);

		userdao.updateUser(newuser, UpdateColumn.STATUS);

		String sql = "select status from user where userid = "+uid;
		ResultSet rs = conn.createStatement().executeQuery(sql);
		rs.next();
		int newstatus = rs.getInt("status");
		boolean success = (newstatus == 1);
		Assert.assertTrue(success);
	}

	@Test
	public void testUpdateUserNegative() throws SQLException,
InstantiationException, IllegalAccessException, ClassNotFoundException{
		UserModel newuser = new UserModel();
		Integer uid = 1;
		newuser.setUserId(uid);
```

```
            userdao.updateUser(newuser, UpdateColumn.STATUS);

            String sql = "select status from user where userid = "+uid;
            ResultSet rs = conn.createStatement().executeQuery(sql);
            rs.next();
            int newstatus = rs.getInt("status");
            boolean success = (newstatus == 1);
            Assert.assertTrue(success);
      }
      @Test
      public void testGetUserPositive() throws SQLException,
InstantiationException, IllegalAccessException, ClassNotFoundException{
            Integer uid = 1;
            UserModel user = userdao.getUser(uid);
            boolean success = (user != null);
            Assert.assertTrue(success);
      }

      @Test
      public void testGetUserNegative() throws SQLException,
InstantiationException, IllegalAccessException, ClassNotFoundException{
            Integer uid = 9999;
            UserModel user = userdao.getUser(uid);
            boolean success = (user == null);
            Assert.assertTrue(success);
      }
      @Test
      public void testGetUserByEmailPositive() throws SQLException,
InstantiationException, IllegalAccessException, ClassNotFoundException{
            UserModel user = new UserModel();
            user.setEmailId("qiu@gmail.com");
            UserModel result = userdao.getUserByEmail(user);
            boolean success = (result != null);
            Assert.assertTrue(success);
      }

      @Test
      public void testGetUserByEmailNegative() throws SQLException,
InstantiationException, IllegalAccessException, ClassNotFoundException{
            UserModel user = new UserModel();
            user.setEmailId("xxxxxx@gmail.com");
            UserModel result = userdao.getUserByEmail(user);
            boolean success = (result == null);
            Assert.assertTrue(success);
      }
}
```

2.2. Test_04_SubsystemTest_MessageDAO

```
package testcases;

import java.sql.Connection;
import java.sql.DriverManager;
```

```
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.List;

import org.junit.After;
import org.junit.Assert;
import org.junit.Before;
import org.junit.Test;

import pantherbuddy.business.model.MessageModel;
import pantherbuddy.datastore.dao.MessageDAO;

public class Test_04_SubsystemTest_MessageDAO {
	private Connection conn;
	private MessageDAO messagedao;
	@Before
	public void setUp() throws Exception{
		messagedao = new MessageDAO();
		conn =
DriverManager.getConnection("jdbc:mysql://localhost/pantherbuddy?"
				+ "user=allanshaji&password=o1pktp");

		int id = 0;
		PreparedStatement preparedStatement =
conn.prepareStatement("SELECT * FROM pantherbuddy.message WHERE messageid =
109");
		ResultSet rs = preparedStatement.executeQuery();
		if(rs.next()){
			id = rs.getInt("messageid");
		}
		if(id != 109){
			PreparedStatement ps = conn.prepareStatement("INSERT INTO
pantherbuddy.message values (109,1, 'test message',NOW())");
			ps.executeUpdate();
		}
	}

	@After
	public void tearDown() throws Exception{
		conn.close();
	}

	@Test
	public void testCreateMessagePositive() throws SQLException,
InstantiationException, IllegalAccessException, ClassNotFoundException{
		MessageModel message = new MessageModel();
		Integer uid = 1;
		String content = "hello world";
		message.setUserId(uid);
		message.setMessage(content);
		try{
			messagedao.createMessage(message);
		}catch(Exception e){
			Assert.fail();
		}
	}
```

```
	@Test(expected = Exception.class)
	public void testCreateMessageNegative() throws SQLException,
InstantiationException, IllegalAccessException, ClassNotFoundException{
		MessageModel message = new MessageModel();
		Integer uid = 9999;
		String content = "hello world";
		message.setUserId(uid);
		message.setMessage(content);
		messagedao.createMessage(message);
	}
	@Test
	public void testDeleteMessagePositive() throws SQLException{
		Integer mid = 109;
		try{
			messagedao.deleteMessage(mid);
		}catch(Exception e){
			Assert.fail();
		}
	}

	@Test(expected = Exception.class)
	public void testDeleteMessageNegative() throws SQLException,
InstantiationException, IllegalAccessException, ClassNotFoundException{
		Integer mid=1;
		messagedao.deleteMessage(mid);
	}
	@Test
	public void testGetMessagePositive() throws SQLException,
InstantiationException, IllegalAccessException, ClassNotFoundException{
		Integer rf = 1;
		List<MessageModel> mm = messagedao.getMessages(rf);
		boolean success = (mm.size()>0);
		Assert.assertTrue(success);
	}

	@Test
	public void testGetMessageNegative() throws SQLException,
InstantiationException, IllegalAccessException, ClassNotFoundException{
		Integer rf = 9999;
		List<MessageModel> mm = messagedao.getMessages(rf);
		boolean success = (mm.size()==0);
		Assert.assertTrue(success);
	}
}
```

11.7. Appendix G – Diary of Meetings and Tasks

Meeting Dairy

Meeting Dairy 1

Project Name: Panther Buddy

Date: 08/25/2015

Time: 1 hour 30 minutes

Location: Grad Lab ECS 252

Person attended: ALL

Late: none Minutes Taker: Sharat Kedari

Assigned Task: Project Idea

Summary of Discussion: Discussion on what type project we can implement. Every gave their own ideas. And finalized the project that need to be implemented.

Meeting Dairy 2

Project Name: Panther Buddy

Date: 08/26/2015

Time: 1 hour

Location: Grad Lab ECS 252

Person attended: ALL

Late: None

Minutes Taker: Sharat Kedari Assigned

Task: Project Idea generation

Summary of Discussion: Finalizing PantherBuddy project and what are the different functionalities that need to be implemented.

Meeting Dairy 3

Project Name: Panther Buddy

Date: 08/27/2015

Time: 3 hour

Location: Grad Lab ECS 266

Person attended: ALL

Late: none

Minutes Taker: Sharat Kedari

Assigned Task: Functionalities for project. Assigned 4 functionalities each

Summary of Discussion: Designing about what project need to contain 21 functionalities and 7 securities.

Meeting Dairy 4

Project Name: Panther Buddy

Date: 09/01/2015

Time: 3 hour

Location: Grad Lab ECS 266

Person attended: ALL

Late: none

Minutes Taker: Sharat Kedari

Assigned Task: Functionalities for project

Summary of Discussion: Assigned 3 functionalities and 1 security to each team member.

Meeting Dairy 5

Project Name: Panther Buddy

Date: 09/06/2015

Time: 3 hour

Location: Grad Lab ECS 266

Person attended: ALL

Late: none

Minutes Taker: Sharat Kedari

Assigned Task: Functionalities for project

Summary of Discussion: Assigned 3 functionalities and 1 security each.

Meeting Dairy 6

Project Name: Panther Buddy

Date: 09/10/2015

Time: 3 hour

Location: Grad Lab ECS 258

Person attended: ALL

Late: none

Minutes Taker: Sharat Kedari

Assigned Task: UML diagrams for the user registration functionalities

Summary of Discussion: Discussed UML diagrams for the user registration functionalities.

Meeting Dairy 7

Project Name: Panther Buddy

Date: 09/11/2015

Time: 3 hour

Location: Grad Lab ECS 266

Person attended: ALL

Late: none

Minutes Taker: Sharat Kedari

Assigned Task: Functionalities for project

Summary of Discussion: Assigned 3 functionalities and 1 security each. And gave a task to draw other use cases assigned to them.

Meeting Dairy 8

Project Name: Panther Buddy

Date: 09/12/2015

Time: 1 hour 30 minutes

Location: Grad Lab ECS 252

Person attended: ALL

Late: none

Minutes Taker: Sharat Kedari

Assigned Task: Functionalities for project

Summary of Discussion: Decision about what are the different types of functionalities that we are implementing

Meeting Dairy 9

Project Name: Panther Buddy

Date: 09/17/2015

Time: 1 hour 30 minutes

Location: Grad Lab ECS 252

Person attended: ALL

Late: none

Minutes Taker: Sharat Kedari

Assigned Task: Functionalities for project

Summary of Discussion: Designing of the functionalities for project in Papyrus.

Meeting Dairy 10

Project Name: Panther Buddy

Date: 09/21/2015

Time: 1 hour 30 minutes

Location: Grad Lab ECS 252

Person attended: ALL

Late: none

Minutes Taker: Sharat Kedari

Assigned Task: Functionalities for project

Summary of Discussion: Designing Functionalities for project in Papyrus

Meeting Dairy 11

Project Name: Panther Buddy

Date: 09/24/2015

Time: 1 hour 30 minutes

Location: Grad Lab ECS 252

Person attended: ALL Late: none

Minutes Taker: Sharat Kedari

Assigned Task: Functionalities for project

Summary of Discussion: Designing Functionalities for project in Papyrus and discussion about the doubts

Meeting Dairy 12

Project Name: Panther Buddy

Date: 10/01/2015

Time: 2 hour 30 minutes

Location: Grad Lab ECS 252

Person attended: ALL

Late: none

Minutes Taker: Sharat Kedari

Assigned Task: SRD writing

Summary of Discussion: Completion of the diagrams in Papyrus. Distribution of tasks for writing the project

Meeting Dairy 13

Project Name: Panther Buddy

Date: 10/02/2015

Time: 4 hour 30 minutes

Location: Grad Lab ECS 252

Person attended: ALL

Late: none

Minutes Taker: Sharat Kedari

Assigned Task: SRD writing.

Summary of Discussion: Completion of tasks for SRD writing.

Meeting Dairy 14

Project Name: Panther Buddy

Date: 10/03/2015

Time: 4 hour 30 minutes

Location: Grad Lab ECS 252

Person attended: ALL

Late: none

Minutes Taker: Sharat Kedari

Assigned Task: SRD writing.

Summary of Discussion: Completion of tasks for SRD writing.

Meeting Dairy 15

Project Name: Panther Buddy

Date: 10/04/2015

Time: 4 hour 30 minutes

Location: Grad Lab ECS 252

Person attended: ALL

Late: none

Minutes Taker: Sharat Kedari

Assigned Task: SRD writing.

Summary of Discussion: Completion of tasks for SRD writing.

Meeting Dairy 16

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 17

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 18

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 19

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 20

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 21

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

Meeting Dairy 22

Name: Panther Buddy

Date: 11/13/15

Time: 8:00 pm – 9:30 pm

Location: ECS 252

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

Agenda: Discussion about third deliverable

Summary of discussion: Everyone in the group need to code their functionalities and submitted their assigned task.

Assigned Tasks: Need to code their functionalities

Meeting Dairy 23

Name: Panther Buddy

Date: 11/16/15

Time: 8:00 pm – 9:30 pm

Location: ECS 252

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

Agenda: Discussion about third deliverable

Summary of discussion: Everyone in the group need to code their functionalities and submitted their assigned task.

Assigned Tasks: Need to code their functionalities

Meeting Dairy 24

Name: Panther Buddy

Date: 11/23/15

Time: 8:00 pm – 9:30 pm

Location: ECS 252

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

Agenda: Review of the code.

Summary of discussion: Discussion about test cases. Working on testing and code coverage of the system and subsystem.

Assigned Tasks: Working on testing

Meeting Dairy 25

Name: Panther Buddy

Date: 11/25/15

Time: 8:00 pm – 9:30 pm

Location: ECS 252

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

Agenda: Working on testing

Summary of discussion: Discussion about test cases. Working on testing and code coverage of the system and subsystem.

Assigned Tasks: Working on testing

Meeting Dairy 26

Name: Panther Buddy

Date: 11/29/15

Time: 8:00 pm – 9:30 pm

Location: ECS 252

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

Agenda: Preparing all the documents and review it.

Summary of discussion: Working on the document. Modifying the mistakes that were made in previous deliverable. Preparing for the presentation and PowerPoint.

Assigned Tasks: Prepare for the presentation.