



**UTM**  
UNIVERSITI TEKNOLOGI MALAYSIA

SECJ2203: Software Engineering

---

## **System Documentation**

**Project Title: FABU Alumni System**

Version 2.0

10th June 2021

School of Computing, Faculty of Engineering

Prepared by: Seventh

## Revision Page

### a. Overview

The software design in this report is the first version of the alumni management system. This report consists of the introduction, system-specific requirement, system architectural design, detailed description of components, data design, user interface design, requirement matrix, test cases, and test approach analysis.

### b. Target Audience

The target audiences of the alumni management system are the alumni in the Faculty of Built Environment and Surveying(FABU) and the FABU staff in the University of Technology Malaysia (UTM).

### c. Project Team Members

Name	Matric Number	Role	Task	Status
Ng Jing Er	A19EC0115	Team Leader	Communication Interfaces Other Requirements State Diagram Use Case( Mobile - View news,View event,View charity,View report) 4.2.3 Package mobile 5.0 Data Design	Completed
Goh Jo Ey	A19EC0047	Member	User Interface Performance Requirements Use Case(User - Login, Add User, Remove User, Create Account) 3.1 Architecture Style and Rationale 3.2 Component Model 4.1 Complete Package Model 4.2 Detailed Description	Completed
Chiam Wooi Chin	A19EC0034	Member	Hardware Interface Design Constraints Use Case ( Alumni - Update Architecture Licensing, Mobile - View alumni profile,View alumni list, View dashboard)	Completed

			Package user 5.0 Data Design	
Ong Yin Ren	A19EC0204	Member	Software Interfaces Software System Attributes State Diagram Use Case(Alumni - Filter friend, Add friend, Delete friend, Search friend) 4.2.1 Package alumni 6.0 User Interface Design	Completed
Siti Fatimah Az Zahra	A19EC3032	Member	Introduction Domain diagram Use Case(Alumni-View status,Upload Status,Edit Status,Delete Status , User-Update Profile) 4.2.1 Package alumni ERD	Completed

**d. Version Control History**

Version	Primary Author(s)	Description of Version	Date Completed
<b>Version 1.0</b>	<b>Ng Jing Er Goh Jo Ey Ong Yin Ren Chiam Wooi Chin Siti Fatimah Az Zahra</b>	<b>This version includes interface requirements, system features, performance requirements, design constraints, software system attributes and other requirements of the alumni system.</b>	<b>25th May 2021</b>
<b>Version 2.0</b>	<b>Ng Jing Er Goh Jo Ey Ong Yin Ren Chiam Wooi Chin Siti Fatimah Az Zahra</b>	<b>This version includes the system architecture design(architecture style and rationale, component model), detailed description of components(package diagram, detailed description, class diagram and sequence diagram for each</b>	<b>10th May 2021</b>

		<p>package), data design(data description, data dictionary) as well as the user interface design(overview and the screen images)</p>	
--	--	--	--

# Table of Contents

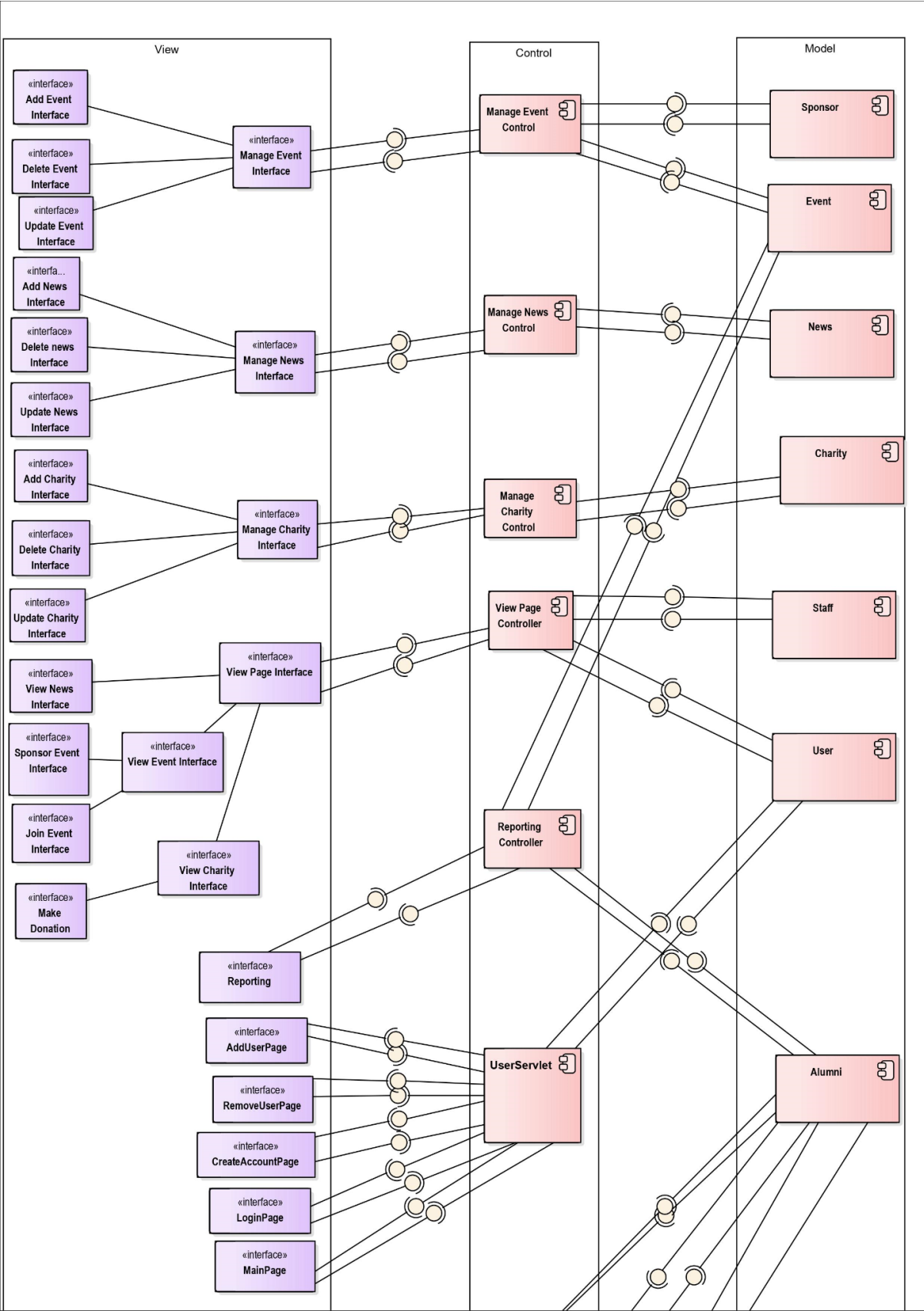
<b>3. System Architectural Design</b>	<b>1</b>
3.1 Architecture Style and Rationale	1
3.2 Component Model	2
<b>4. Detailed Description of Components</b>	<b>4</b>
4.1 Complete Package Diagram	4
4.2 Detailed Description	6
4.2.1 P001: <Alumni> Subsystem	7
4.2.1.1 Class Diagram	7
4.2.1.2 Sequence Diagram	18
<b>4.2.2 P002: User Subsystem</b>	<b>28</b>
4.2.2.1 Class Diagram	29
4.2.2.2 Sequence Diagram	36
4.2.3 P003: Package Mobile	40
4.2.3.1 Class Diagram	41
4.2.3.2 Sequence Diagram	49
<b>5. Data Design</b>	<b>56</b>
5.1 Data Description	56
5.2 Data Dictionary	58
5.2.1 Entity: User	58
5.2.2 Entity: Staff	58
5.2.3 Entity: Alumni	59
5.2.4 Entity: Account	60
5.2.5 Entity: Status	60
5.2.6 Entity: Friend	60
5.2.7 Entity: Architecture License	61
5.2.8 Entity: Mobile	61
5.2.9 Entity: Dashboard	62
5.2.10 Entity: Charity	62
5.2.11 Entity: News	62
5.2.12 Entity: Event	62
5.2.13 Entity: Report	62
<b>6. User Interface Design</b>	<b>63</b>
6.1 Overview of User Interface	63
6.2 Screen Images	63

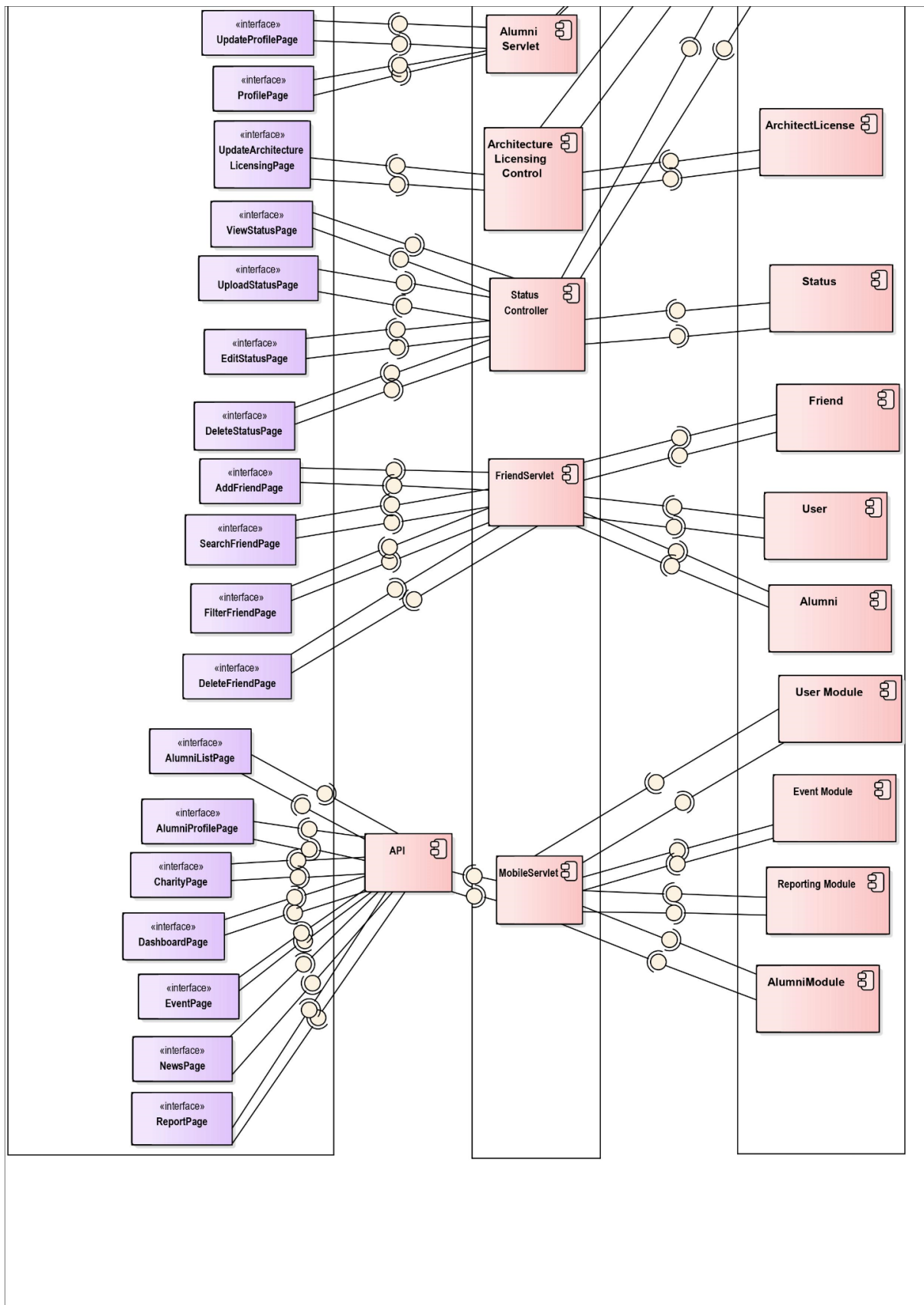
### **3. System Architectural Design**

#### **3.1 Architecture Style and Rationale**

The chosen architecture style for the FABU alumni system is Model View Control(MVC). There are three main components in this architecture which are model component, control component and the view component. The model components handle the system's data and the processes that go along with it, the view components represent the presentation of the model data to the user as well as the control component represents the bridge between the user and the system. MVC architecture style can speed up the development process of the project because the model is well defined and each person can concentrate on its own part. This architecture pattern provides the separation between the presentation and interaction from the system data. It is appropriate for this project since it permits data to change without concern to its representation and vice versa. The modifications of the model do not affect the entire model of the project. It makes updating the system much easier. It also enables the display of the same data in many forms, with changes to one representation mirrored in the others. By applying this architecture style, the implementation of the alumni system in web browsers and also the mobile phone can be more easier.

### 3.2 Component Model



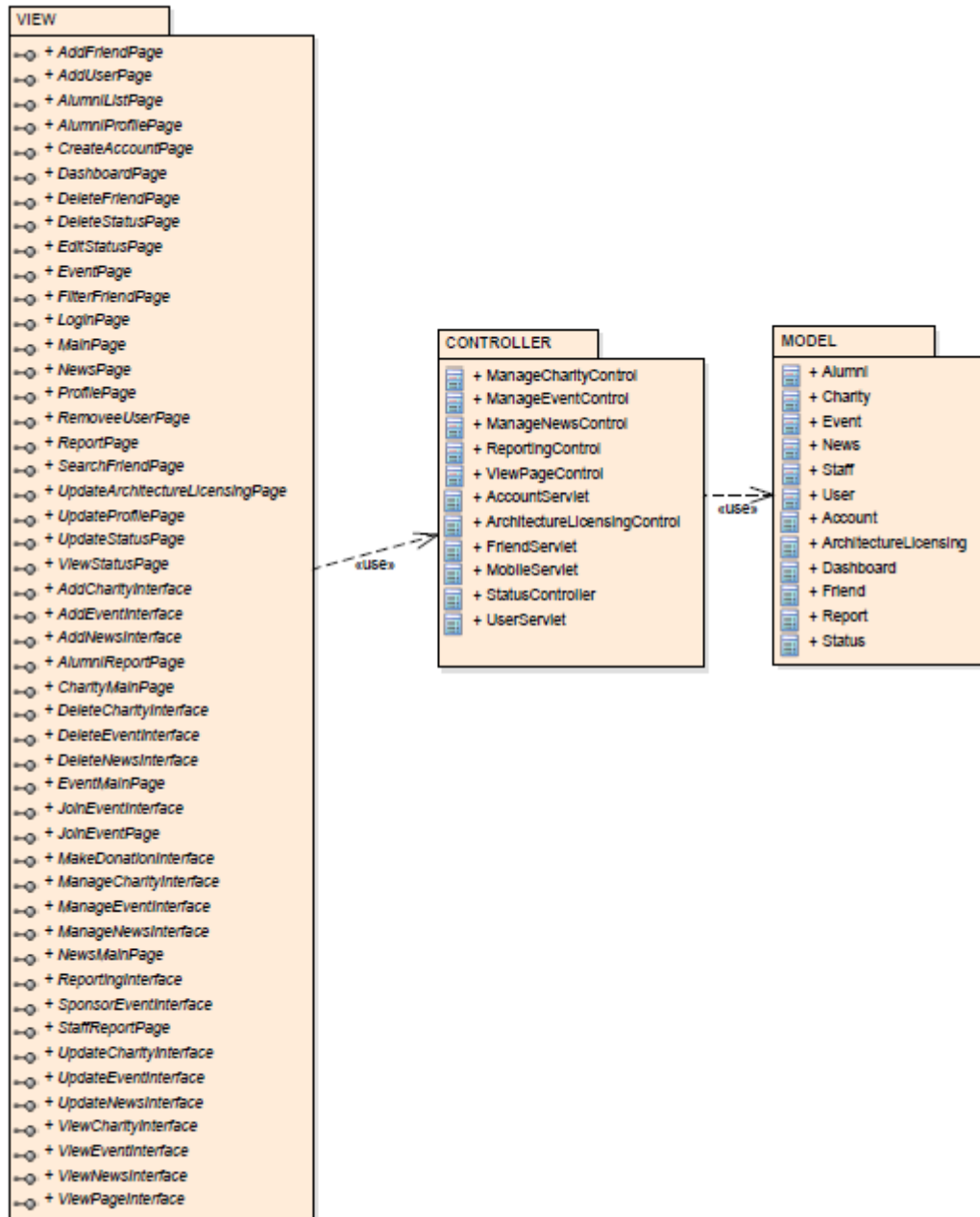


**Figure 3.1: Component Diagram of <FABU ALUMNI SYSTEM>**



## 4. Detailed Description of Components

### 4.1 Complete Package Diagram



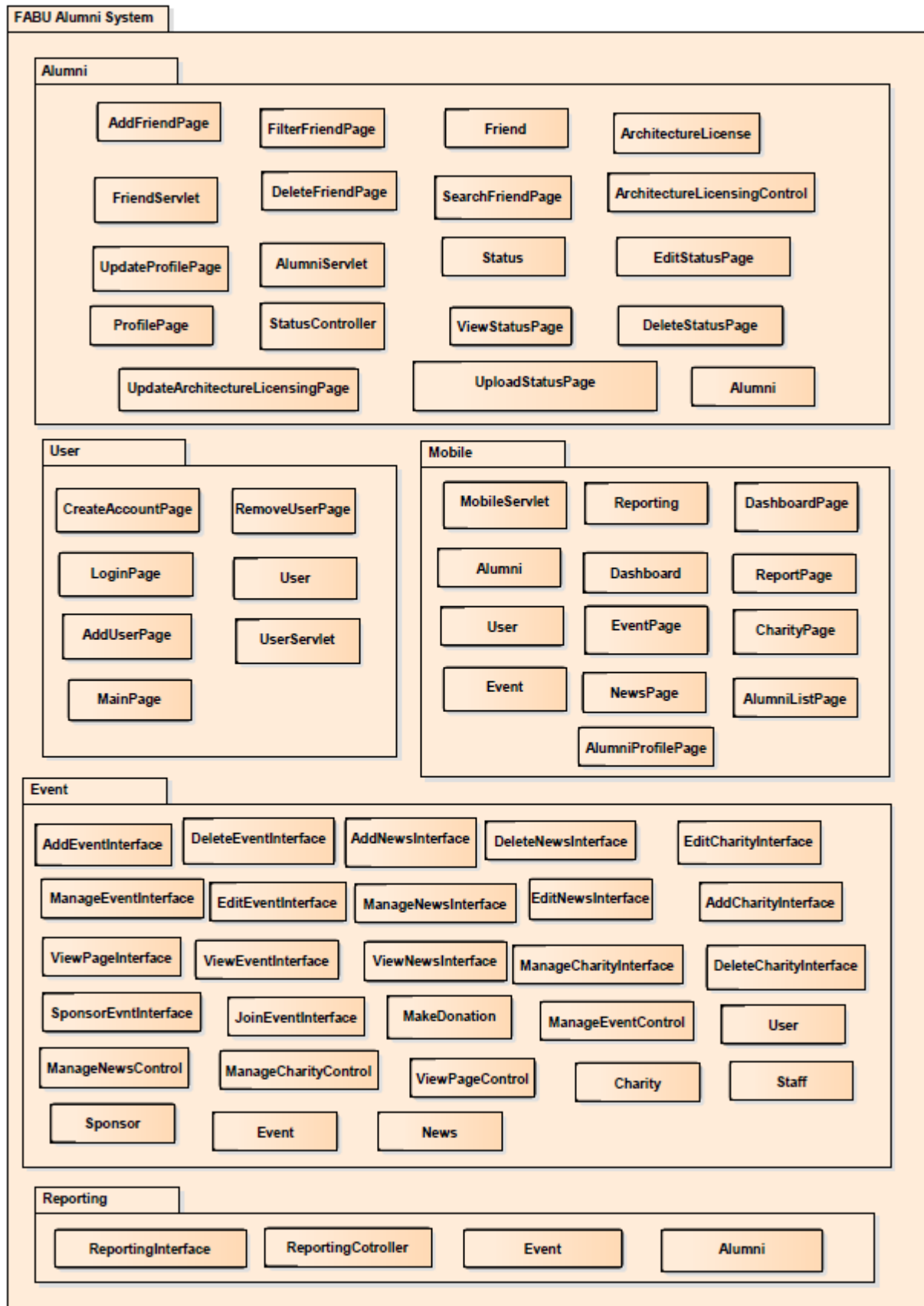


Figure 4.1: Package Diagram for <FABU ALUMNI SYSTEM>

## 4.2 Detailed Description

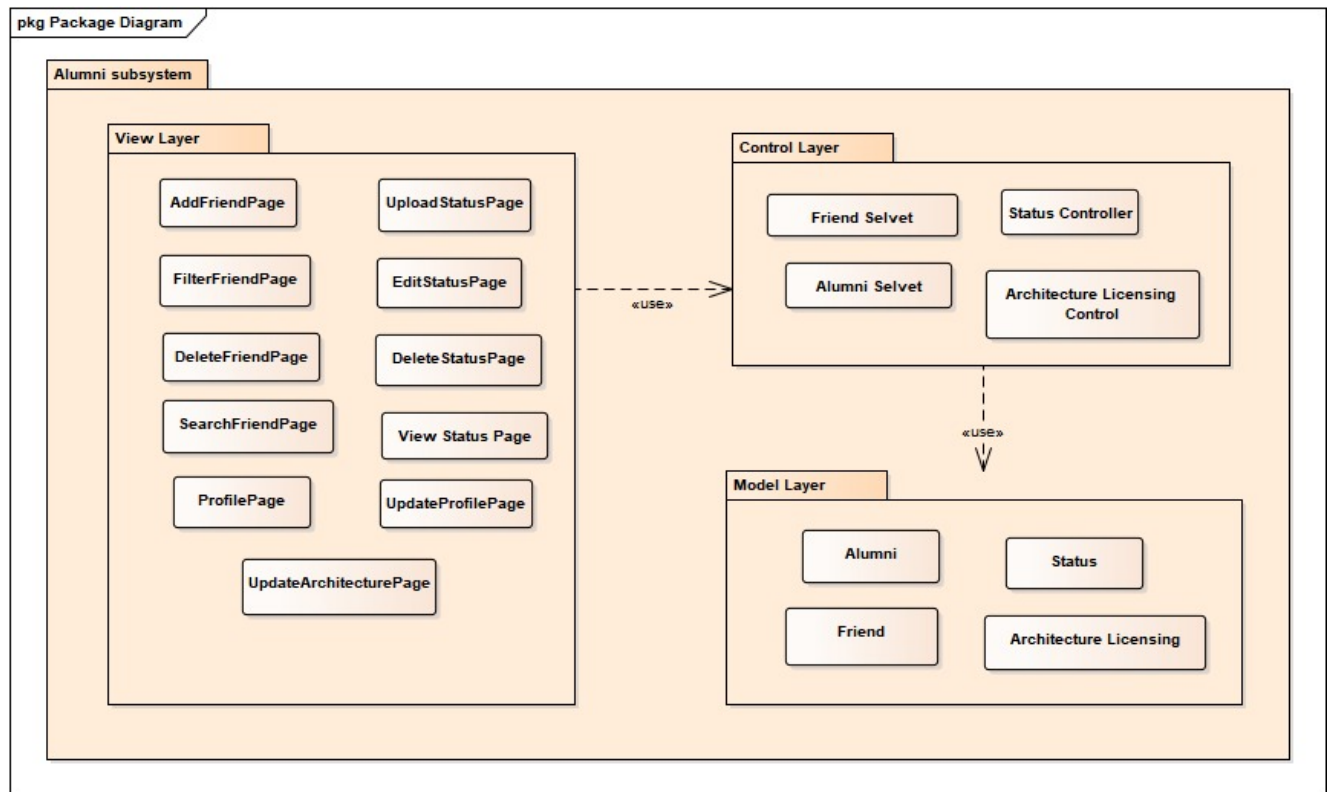
FABU alumni system consists of 4 subsystems: alumni, mobile, user, event and reporting subsystems. Each subsystem is divided into model, view and control.

The user subsystem consists of the user servlet, user model and also the user interfaces which includes the login page, create account page, remove user page, update profile and also add user page. It allows the user to add(UC0001) and remove(UC0004) users to the system; the added user also can create their account(UC0002) in the user subsystem. Once they create their account, they can login(UC0003) into the system by entering their username and password. For every user, they are allowed to update their profile(UC0017) info in this subsystem.

The alumni subsystem consists of the alumni, alumni control, architecture license model, architecture licensing control, friend model, friend control and also the status interface, status control as well as the status model. The alumni can update their architecture licensing(UC0009) in this subsystem. The alumni also have the right to manage their friends by filtering the friends(UC0013), adding a friend(UC0014), deleting a friend(UC0015), and also searching for a friend(UC0016). The alumni also can manage their status in the alumni subsystem with the function of view status(UC0018), upload status(UC0019), edit status(UC0020) and also delete status(UC0021).

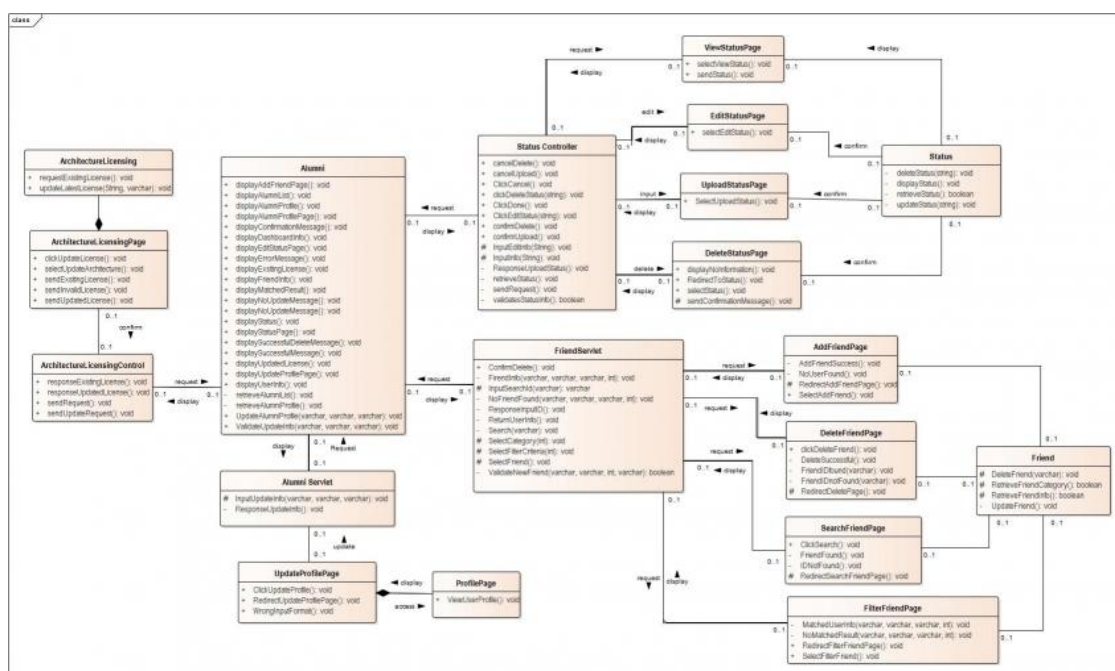
The mobile subsystem consists of the mobile servlet and the mobile interfaces which include event page, reporting page, news page, charity page alumni list, alumni profile and also dashboard. It includes the event module, alumni module, user module, and reporting module as the system data and the operation(model) of the subsystem. The mobile subsystem can only be accessed by the alumni user. The alumni user can view news(UC0005), events(UC0006), charity(UC0007), report(UC0008), alumni list(UC0010), alumni profile(UC0011) and dashboard(UC0012).

#### 4.2.1 P001: <Alumni> Subsystem



**Figure 4.2.1: Package diagram for <Alumni> Subsystem**

#### 4.2.1.1 Class Diagram



**Figure 4.2.1: Class diagram for <Alumni> Subsystem**

<b>Entity Name</b>	Alumni
<b>Method Name</b>	displayAlumniList(), displayAlumniprofile(), displayAlumniProfilePage(), displayConfirmationMessage(), displayDashboardInfo(), displayAddFriend(), displayEditStatusPage(), displayErrorMessage(), displayExistingLicense(), displayFriendInfo(), displayMatchedResult(), displayMoUpdateMessage(), displayStatus(), displayStatusPage(), displaySuccessfulDeleteMessage(), displaySuccessfulmessage(), displayUpdatedLicense(), displayUpdateProfilePage(), displayuserInfo(), retrieveAlumniList(), retrieveAlumniProfile(), updateAlumniProfile(), validateUpdateInfor()
<b>Input</b>	Request from user
<b>Output</b>	Display out based on the request
<b>Algorithm</b>	<ol style="list-style-type: none"> <li>1. Start</li> <li>2. The user request a specific task (e.g update alumni profile)</li> <li>3. System receive and process request</li> <li>4. Display output based on the request</li> <li>5. End</li> </ol>

<b>Entity Name</b>	Alumni Servlet
<b>Method Name</b>	ResponseUpdateInfo()
<b>Input</b>	Update Info
<b>Output</b>	Update info true or false
<b>Algorithm</b>	<ol style="list-style-type: none"> <li>1. Start</li> <li>2. User receive update info</li> <li>3. System validate the input format</li> <li>4. The system return 'True' for correct format and 'False' for wrong format</li> <li>5. End</li> </ol>

<b>Entity Name</b>	UpdateProfilePage
<b>Method Name</b>	ClickUpdateProfile(),RedirectUpdateProfilePage(), WrongInputFormat()
<b>Input</b>	Update Info
<b>Output</b>	Display successful or error message
<b>Algorithm</b>	<ol style="list-style-type: none"> <li>1. Start</li> <li>2. User click on "Update Profile"</li> <li>3. User receive update info</li> <li>4. System will validate input format</li> <li>5. If input format accepted, system will display successful message, else the system will display error message</li> <li>6. End</li> </ol>

<b>Entity Name</b>	ProfilePage
<b>Method Name</b>	ViewUserProfile()
<b>Input</b>	Profile page access request
<b>Output</b>	Display alumni profile page
<b>Algorithm</b>	<ol style="list-style-type: none"> <li>1. Start</li> <li>2. User must have login as an alumni</li> <li>3. User request to view user profile</li> <li>4. System displays user (alumni) profile page</li> <li>5. End</li> </ol>

<b>Entity Name</b>	ArchitectureLicensingControl
<b>Method Name</b>	ResponseExistingLicense(),responseUpdatedLicense(), sendRequest(), sendUpdateRequest()
<b>Input</b>	Request to update architecture license
<b>Output</b>	Update architect license status

<b>Algorithm</b>	<ol style="list-style-type: none"> <li>1. Start</li> <li>2. Receive request to update architecture license from user</li> <li>3. Request existing license from system</li> <li>4. System display existing license</li> <li>5. Receive update request from user</li> <li>6. Send update info to the system</li> <li>7. System returns update status</li> <li>8. Send update status to Update Architecture Licensing Page</li> <li>9. End</li> </ol>
------------------	--

<b>Entity Name</b>	ArchitectureLicensingPage
<b>Method Name</b>	clickUpdateLicense(),selectUpdateArchitecture(), sendExistingLicense(),sendInvalidLicense(), sendUpdateLicense()
<b>Input</b>	Request to update architecture license
<b>Output</b>	Update architect license status
<b>Algorithm</b>	<ol style="list-style-type: none"> <li>1. Start</li> <li>2. Receive request to update architecture license from user</li> <li>3. Request existing license from system</li> <li>4. System display existing license</li> <li>5. Receive update request from user - clickUpdateLicense()</li> <li>6. Send update info to the system</li> <li>7. System returns update status</li> <li>8. Display updated license if update successful or display error message if update fails..</li> <li>9. End</li> </ol>

<b>Entity Name</b>	ArchitectureLicensing
<b>Method Name</b>	request ExistingLicense(), updatelatestLicense()
<b>Input</b>	Request to update architecture license and update info
<b>Output</b>	Update confirmation

<b>Algorithm</b>	<ol style="list-style-type: none"> <li>1. Start</li> <li>2. Receive request to display existing architecture license</li> <li>3. Display existing license</li> <li>4. Receive information to update architecture license</li> <li>5. Validate information format</li> <li>6. Return update status (valid/invalid)</li> <li>7. End</li> </ol>
------------------	--

<b>Entity Name</b>	Status
<b>Method Name</b>	deleteStatus(), displayStatus(), retrieveStatus(), updateStatus()
<b>Input</b>	Request to view status/ New status/ status ID
<b>Output</b>	Status information/ Response message
<b>Algorithm</b>	<ol style="list-style-type: none"> <li>1. Start</li> <li>2. The system receive request to view/ new status</li> <li>3. Retrieve information ALT: The system validates the information</li> <li>4. Display status information or response message</li> </ol>

<b>Entity Name</b>	Status Controller
<b>Method Name</b>	cancelDelete(),cancelUpload(), ClickCancel(), clickDeleteStatus(), clickDone(),clickEditStatus(),confirmDelete(),confirmUpload(), InputEditInfo(),InputInfo(),ResponseUploadStatus(), retrieveStatus(), sendRequest(), validatesStatusInfo()
<b>Input</b>	<ol style="list-style-type: none"> <li>a. Request to view</li> <li>b. Status edit info</li> <li>c. Input status</li> <li>d. delete</li> </ol>
<b>Output</b>	<ol style="list-style-type: none"> <li>a. Send retrieve status to status page</li> <li>b. Display status page</li> <li>c. Display successful or cancelation message</li> </ol>
<b>Algorithm</b>	<ol style="list-style-type: none"> <li>1. Start <ol style="list-style-type: none"> <li>2.1a Receive request to view status</li> </ol> </li> </ol>



	<p>2.2a Retrieve status from system</p> <p>2.3a System returns status</p> <p>2.4a Send retrieved status to status page</p> <p>2.1b Receive status edit info</p> <p>2.2b Confirm and update info in system</p> <p>ALT: 2.2b User cancel the update the system will display status page</p> <p>2.1c Receive status input info</p> <p>2.2c User confirm upload status</p> <p>2.2c System validate the status info</p> <p>2.3c System returns the validation result</p> <p>2.4c If validation is true: system upload the status</p> <p>2.5c System display upload successful message</p> <p>ALT: 2.4c If validation is false: System display cancel message</p> <p>2.1d Receive request to delete status</p> <p>2.2d Display delete confirmation message</p> <p>2.3d If user confirm delete, system deletes the status</p> <p>2.4d System display successful message</p> <p>ALT: 2.3d if user cancel delete, system display status page</p> <p>3. End</p>
--	---

<b>Entity Name</b>	ViewStatusPage
<b>Method Name</b>	selectViewStatus(), sendStatus()
<b>Input</b>	Request to view status
<b>Output</b>	Display status or error message
<b>Algorithm</b>	<p>1. Start</p> <p>2. Receive request to view status</p> <p>3. System retrieve status</p> <p>4. Display status or error message (if status is not available)</p> <p>5. End</p>

<b>Entity Name</b>	EditStatusPage
<b>Method Name</b>	selectEditStatus()
<b>Input</b>	Request to edit status info
<b>Output</b>	Display status page
<b>Algorithm</b>	<ol style="list-style-type: none"> <li>1. Start</li> <li>2. Receive request to display existing architecture license</li> <li>3. Display existing license</li> <li>4. Receive information to update architecture license</li> <li>5. Validate information format</li> <li>6. Return update status (valid/invalid)</li> <li>7. End</li> </ol>

<b>Entity Name</b>	UploadStatusPage
<b>Method Name</b>	selectUploadStatus()
<b>Input</b>	Request to upload status
<b>Output</b>	Display successful response or cancel message
<b>Algorithm</b>	<ol style="list-style-type: none"> <li>1. Start</li> <li>2. Receive request to upload status</li> <li>3. User insert new status</li> <li>4. System displays confirmation request</li> <li>5. User confirm to upload     ALT: 5. User cancel to upload</li> <li>6. Display successful/ cancel response</li> <li>7. End</li> </ol>

<b>Entity Name</b>	DeleteStatusPage
<b>Method Name</b>	displayNoInformation(),RedirectToStatus(),selectStatus, sendConfirmationMessage()
<b>Input</b>	Request to delete status

<b>Output</b>	Display status page and display successful message
<b>Algorithm</b>	<ol style="list-style-type: none"> <li>1. Start</li> <li>2. Receive request to delete selected status</li> <li>3. System display confirmation request</li> <li>4. User confirm deletion <ul style="list-style-type: none"> <li>ALT: 4. User cancel deletion</li> </ul> </li> <li>5. Display successful or cancel message</li> <li>6. End</li> </ol>

<b>Entity Name</b>	Friend
<b>Method Name</b>	DeleteFriend(), RetrieveFriendCategory(), RetrieveFriendInfo(), UpdateFriend()
<b>Input</b>	Friend category/ friend ID/ friend criteria
<b>Output</b>	User or friend information
<b>Igorithm</b>	<ol style="list-style-type: none"> <li>1. Start</li> <li>2. System request to retrieve user info</li> <li>3. System returns user info</li> <li>4. End</li> </ol>

<b>Entity Name</b>	FriendServlet
<b>Method Name</b>	ConfirmDelete(), FriendInfo(), InoutSearchId(), NoFriendFound(), ReponseInputID(), ReturnUserInfo(), Search(), SelectCategory(), SelectFilterCriteria(), SelectFriend(), ValidateNewFriend()
<b>Input</b>	<ol style="list-style-type: none"> <li>a. Request to add friend- input category</li> <li>b. Request to delete friend- input friend ID</li> <li>c. Request to filter friends- input criteria</li> <li>d. Request to search friend- input friend ID</li> </ol>
<b>Output</b>	<ol style="list-style-type: none"> <li>a. Display friend info</li> <li>b. Delete confirmation message and successful or error message</li> <li>c. Display matched result or display error message</li> <li>d. Return search status - Found or ID Not Found</li> </ol>

<b>Algorithm</b>	<p>1. Start</p> <p>2.1a Receive request to add friend</p> <p>2.2a User select a category</p> <p>2.3a System retrieve user info</p> <p>2.4a Display friend info</p> <p>2.1b Receive request to delete friend</p> <p>2.2b Search friend ID</p> <p>2.3b System retrieve friend information</p> <p>2.4b If friend is found, display delete confirmation request</p> <p>ALT: 2.4b If friend is not found, display error message</p> <p>2.5b Redirect to delete page</p> <p>2.5b User confirm deletion</p> <p>2.6b System delete friend from database</p> <p>2.1c Receive request to filter friends</p> <p>2.2c User insert filter criteria</p> <p>2.3c Retrieve friends based on criteria</p> <p>2.4c Display matched result from friends filter</p> <p>ALT 1: 2.4c No friends found, display error message</p> <p>ALT 2: 2.4c No matched friends, display error message</p> <p>2.1d Receive request to search friend</p> <p>2.2d User insert friend ID</p> <p>2.3d Select category of friend</p> <p>2.4d Retrieve friend info</p> <p>2.5d If friend is found, return search status as Found</p> <p>ALT: 2.5d If friend is not found, return search status as ID Not Found</p> <p>3. End</p>
------------------	--

<b>Entity Name</b>	AddFriendPage
<b>Method Name</b>	AddFriendSuccess(), NoUserFound(), RedirectAddFriendPage(), SelectAddFriend()

<b>Input</b>	Request to add friend- category
<b>Output</b>	Display successful or error message
<b>Algorithm</b>	<ol style="list-style-type: none"> <li>1. Start</li> <li>2. Receive request to add friend</li> <li>3. System display user info</li> <li>4. Add friend</li> <li>5. Display add friend successful message</li> <li>6. End</li> </ol>

<b>Entity Name</b>	DeleteFriendPage
<b>Method Name</b>	clickDeleteFriend(), DeleteSuccessful(), FriendIDFound(), FriendIDnotFound(), redirectDeletePage()
<b>Input</b>	Receive request to delete friend- friend ID
<b>Output</b>	Display successful or error message
<b>Algorithm</b>	<ol style="list-style-type: none"> <li>Start</li> <li>2. Receive request to delete friend</li> <li>3. User insert friend ID</li> <li>4. If friend is found, display delete confirmation message</li> <li>5. User confirm deletion</li> <li>6. Display successful message</li> <li>ALT: 4. If friend is not found, display error message</li> <li>5. Redirect to delete page</li> <li>7. End</li> </ol>

<b>Entity Name</b>	FilterFriendPage
<b>Method Name</b>	MatchedUserInfo(), NoMatchedUserInfo(), redirectFilterPage(), SelectFilterFriend()
<b>Input</b>	Request to filter status- criteria
<b>Output</b>	Display matched result or error message

<b>Algorithm</b>	<ol style="list-style-type: none"> <li>1. Start</li> <li>2. Receive request to filter friend</li> <li>3. User insert filter criteria</li> <li>4. System retrieve information</li> <li>5. If matched user found, display matched result <ul style="list-style-type: none"> <li>ALT: If matched user not found, display error message and redirect to filter friend page</li> </ul> </li> <li>6. End</li> </ol>
------------------	---

<b>Entity Name</b>	SearchFriendPage
<b>Method Name</b>	ClickSearch(),FriendFound(),IDNotFound(), RedirectSeachFriendPage()
<b>Input</b>	Request to search friend- friend ID
<b>Output</b>	Display friend information or error message
<b>Algorithm</b>	<ol style="list-style-type: none"> <li>1. Start</li> <li>2. Receive request to search friend</li> <li>3. User insert friend ID</li> <li>4. System retrieve information</li> <li>5. If friend found, display friend information <ul style="list-style-type: none"> <li>ALT: If friend is not found, display error message and redirect to search friend page</li> </ul> </li> <li>6. End</li> </ol>

### 4.2.1.2 Sequence Diagram

a) SD001: Sequence diagram for Update Architecture Licensing

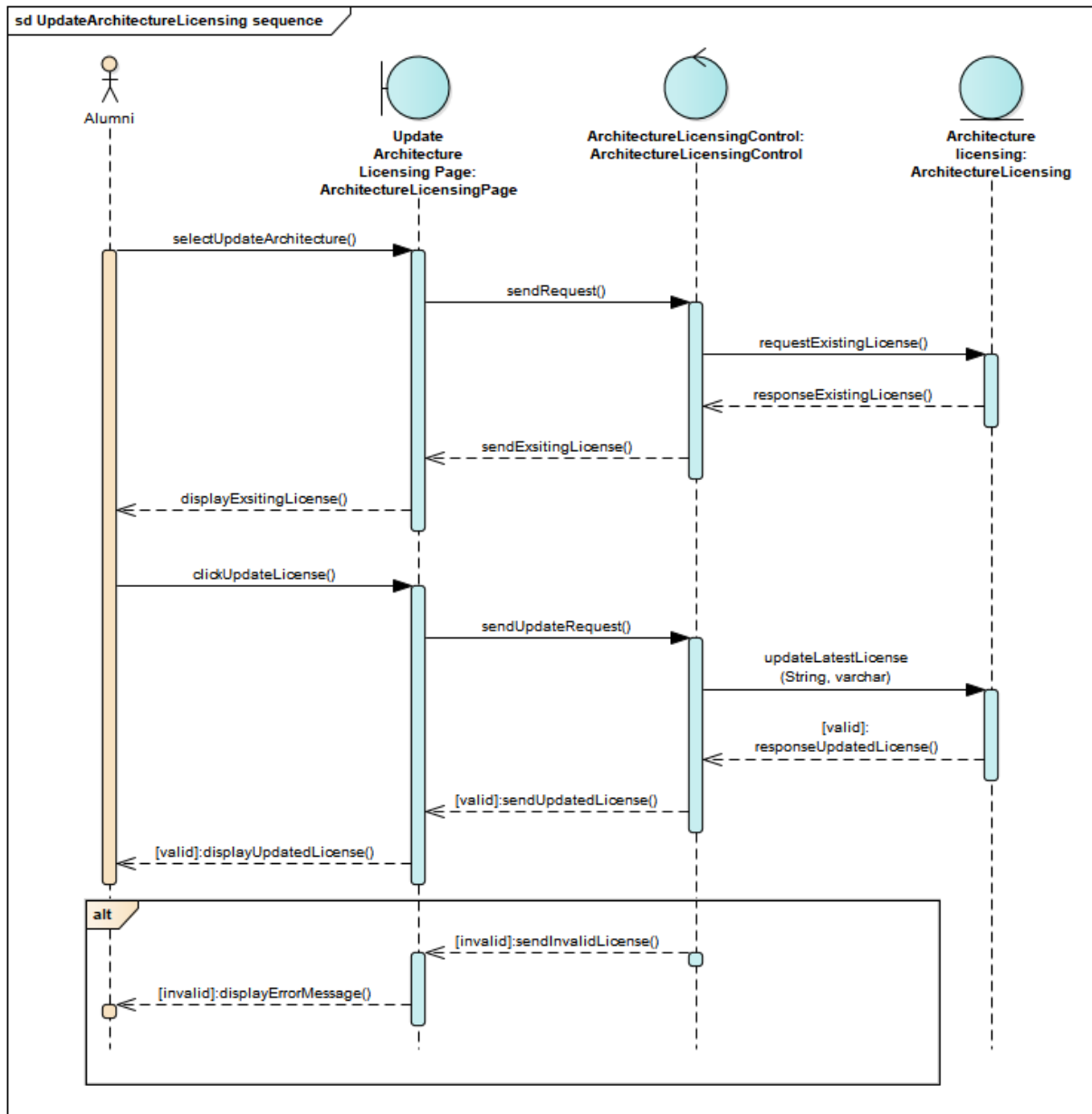


Figure 4.2.1.2.1: Sequence Diagram for <Update Architecture Licensing>

b) SD002: Sequence diagram for Update Profile Page

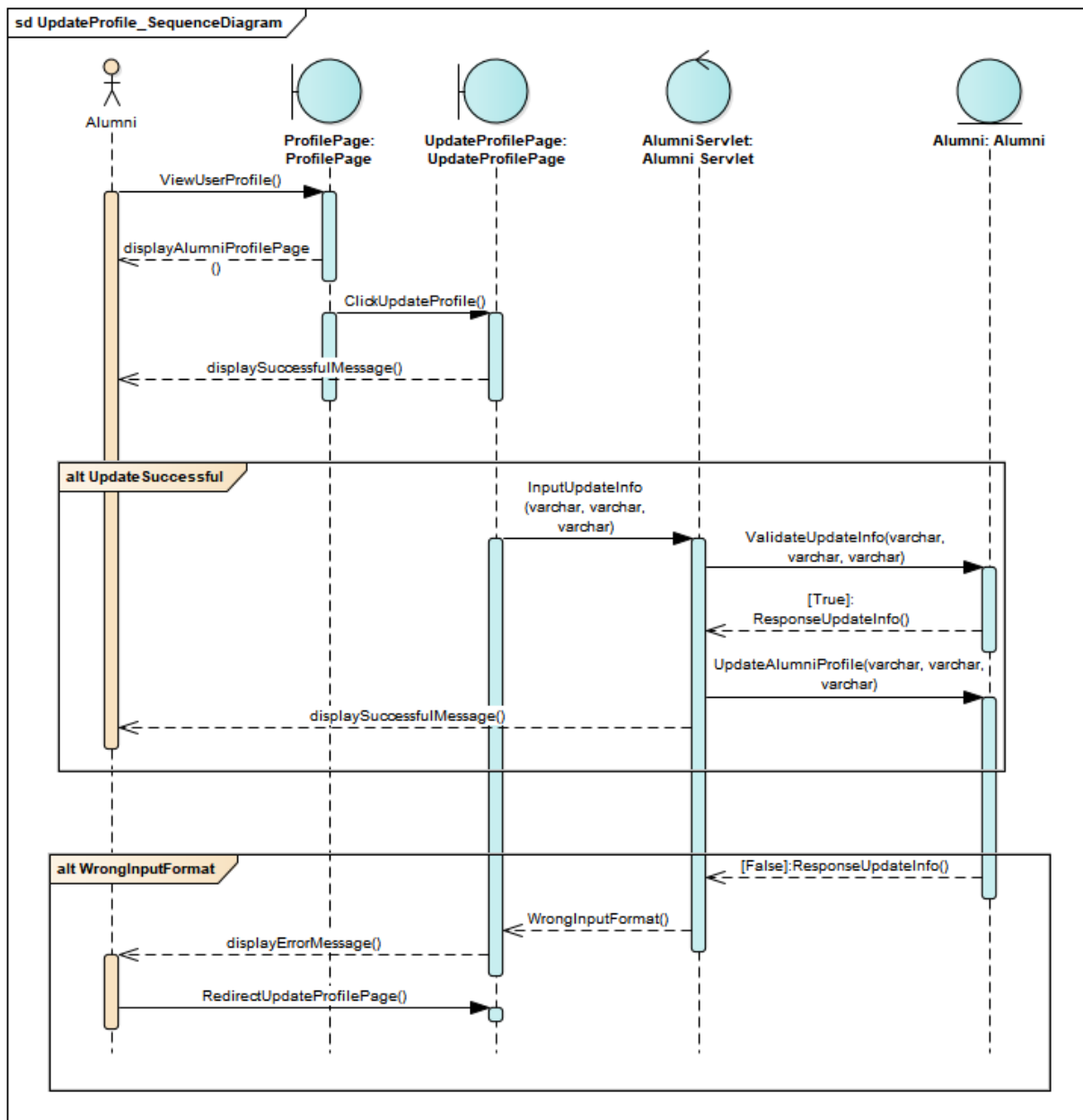


Figure 4.2.1.2.2: Sequence Diagram for <Update Profile Page>



c) SD003: Sequence diagram for View Status

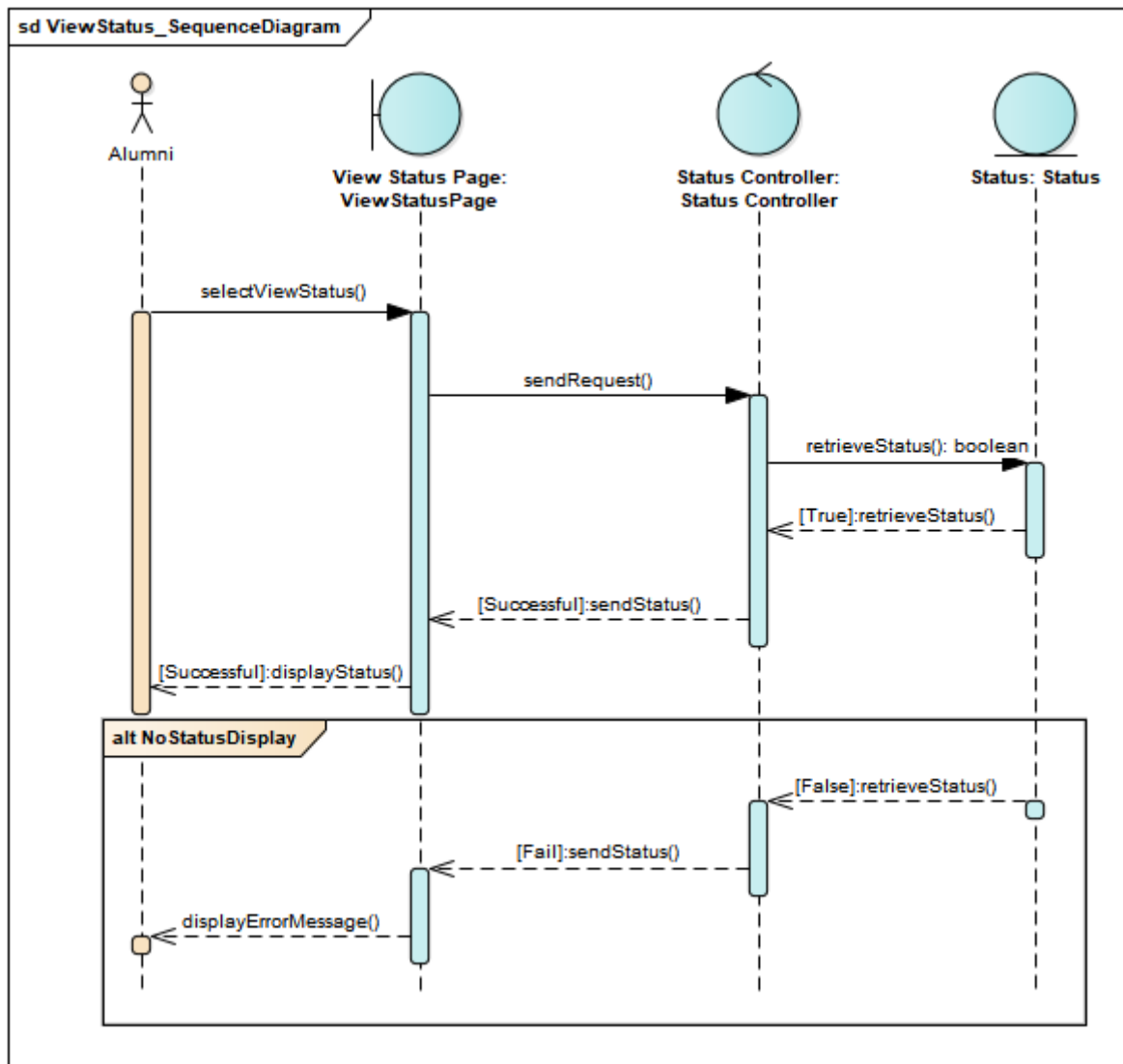


Figure 4.2.1.2.3: Sequence Diagram for <View Status>

d) SD004: Sequence diagram for EditStatus

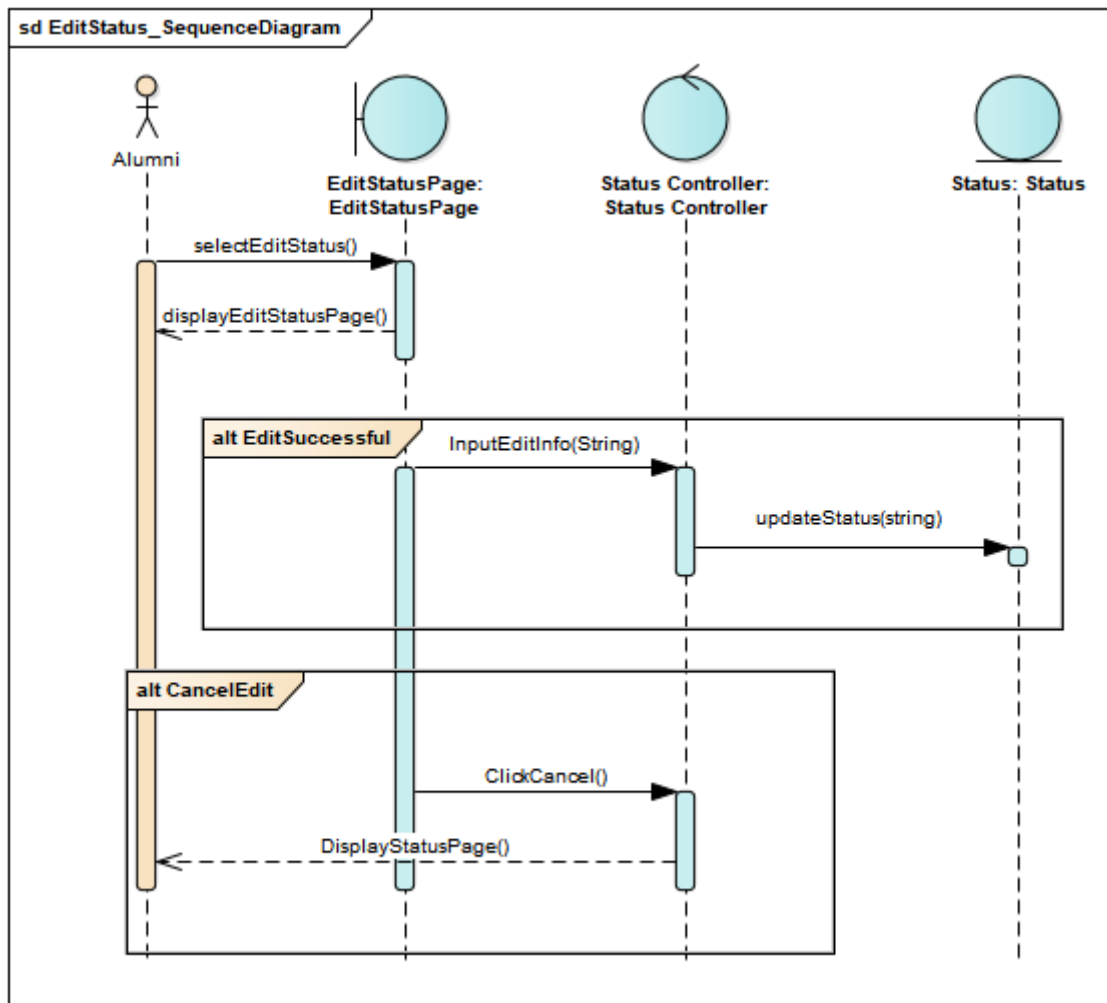


Figure 4.2.1.2.4: Sequence Diagram for <Edit Status>

e) SD005: Sequence diagram for Upload Status

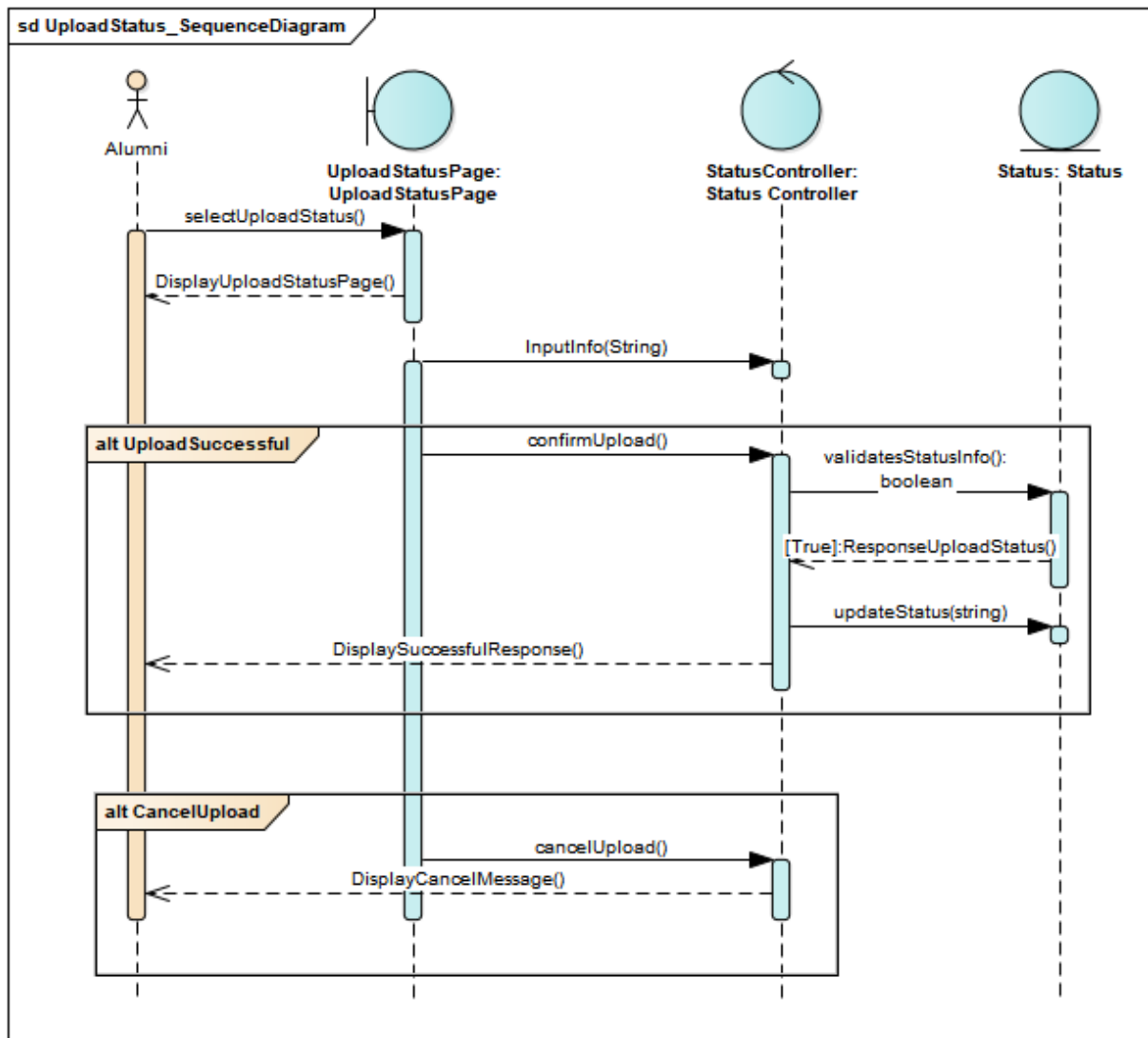


Figure 4.2.1.2.5: Sequence Diagram for <Upload Status>

f) SD006: Sequence diagram for Delete Status

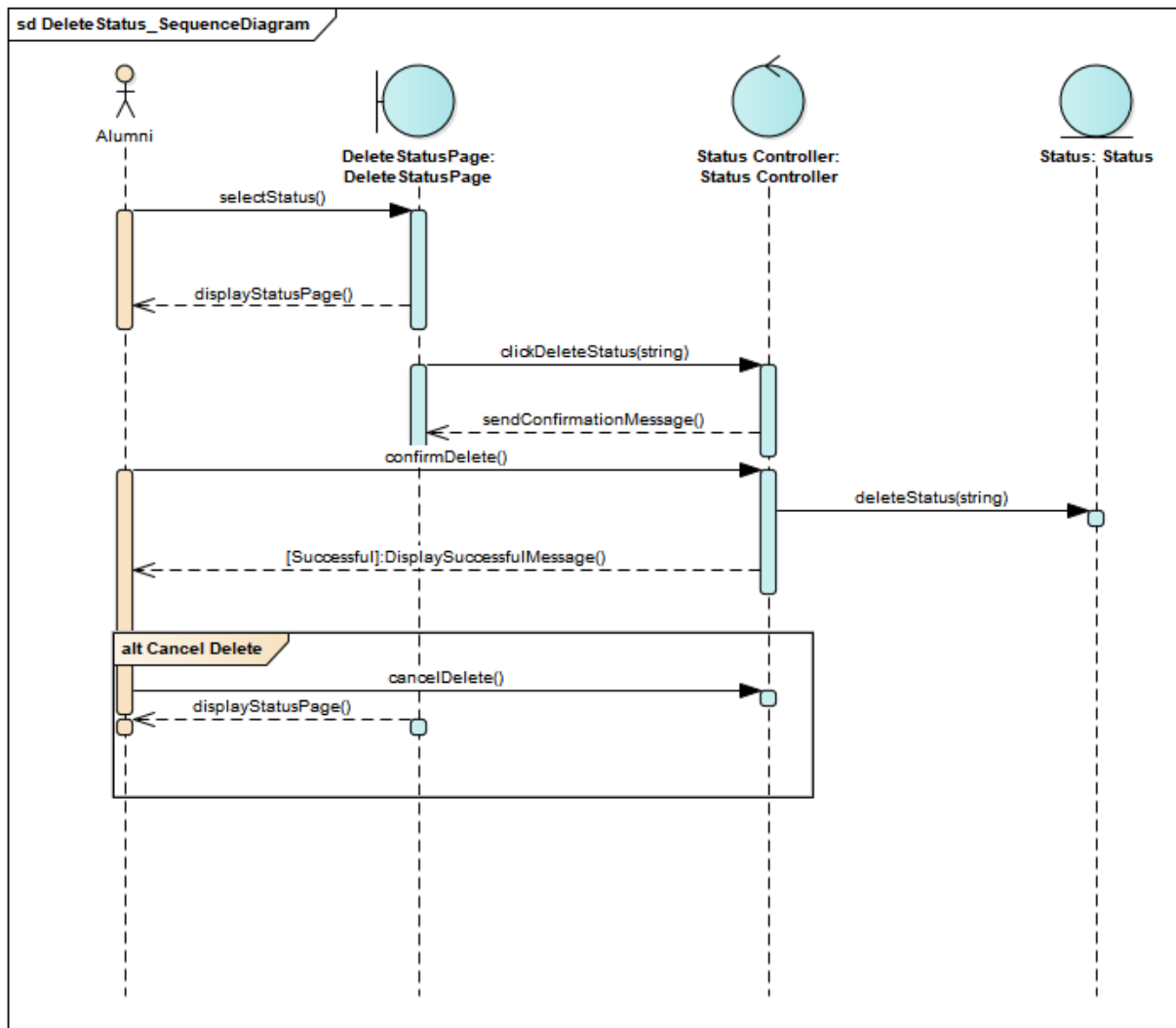


Figure 4.2.1.2.6: Sequence Diagram for <Delete Status>

g) SD007: Sequence diagram for Add Friend

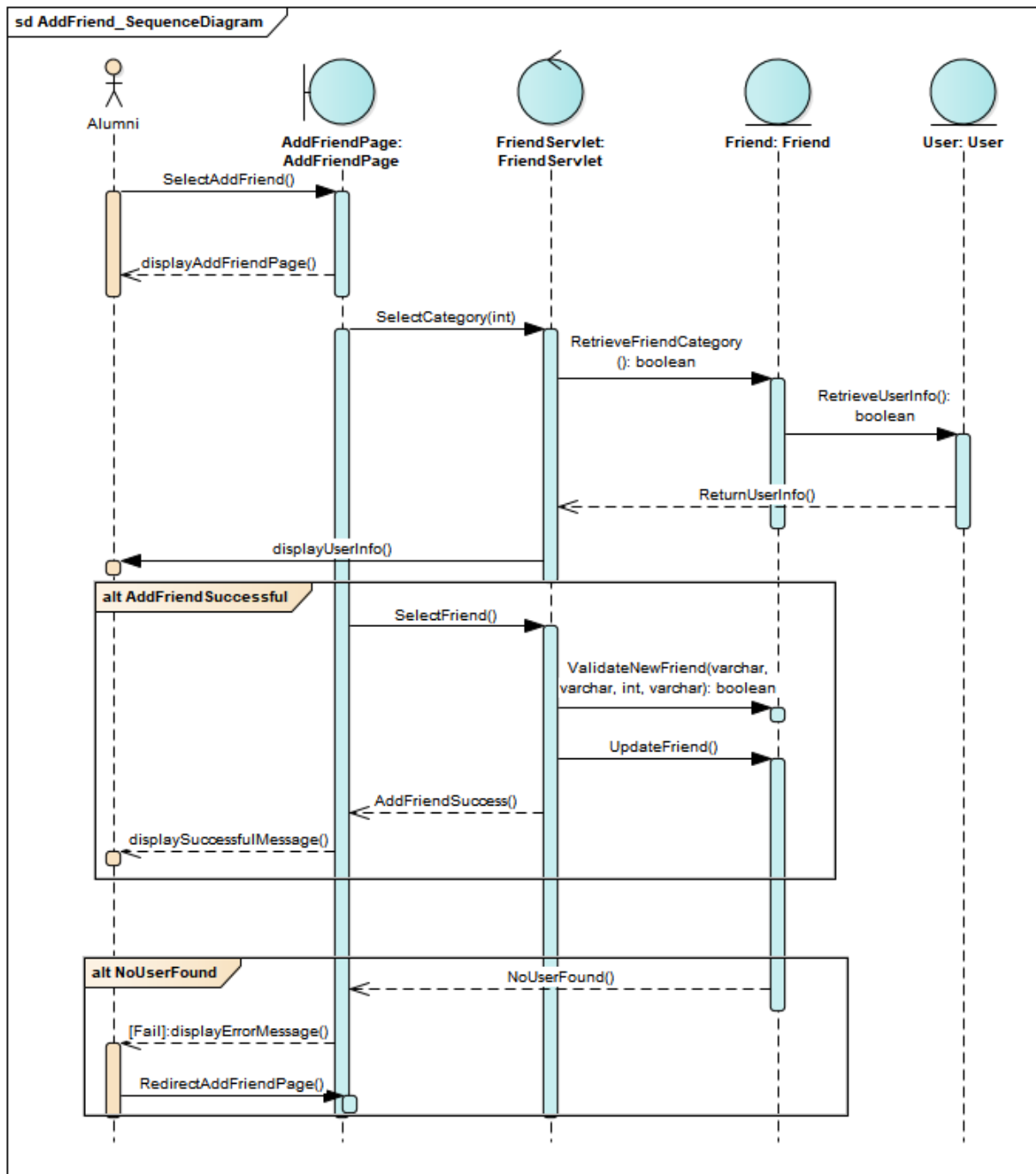


Figure 4.2.1.2.7: Sequence Diagram for <Add Friend>

h) SD008: Sequence diagram for Delete Friend

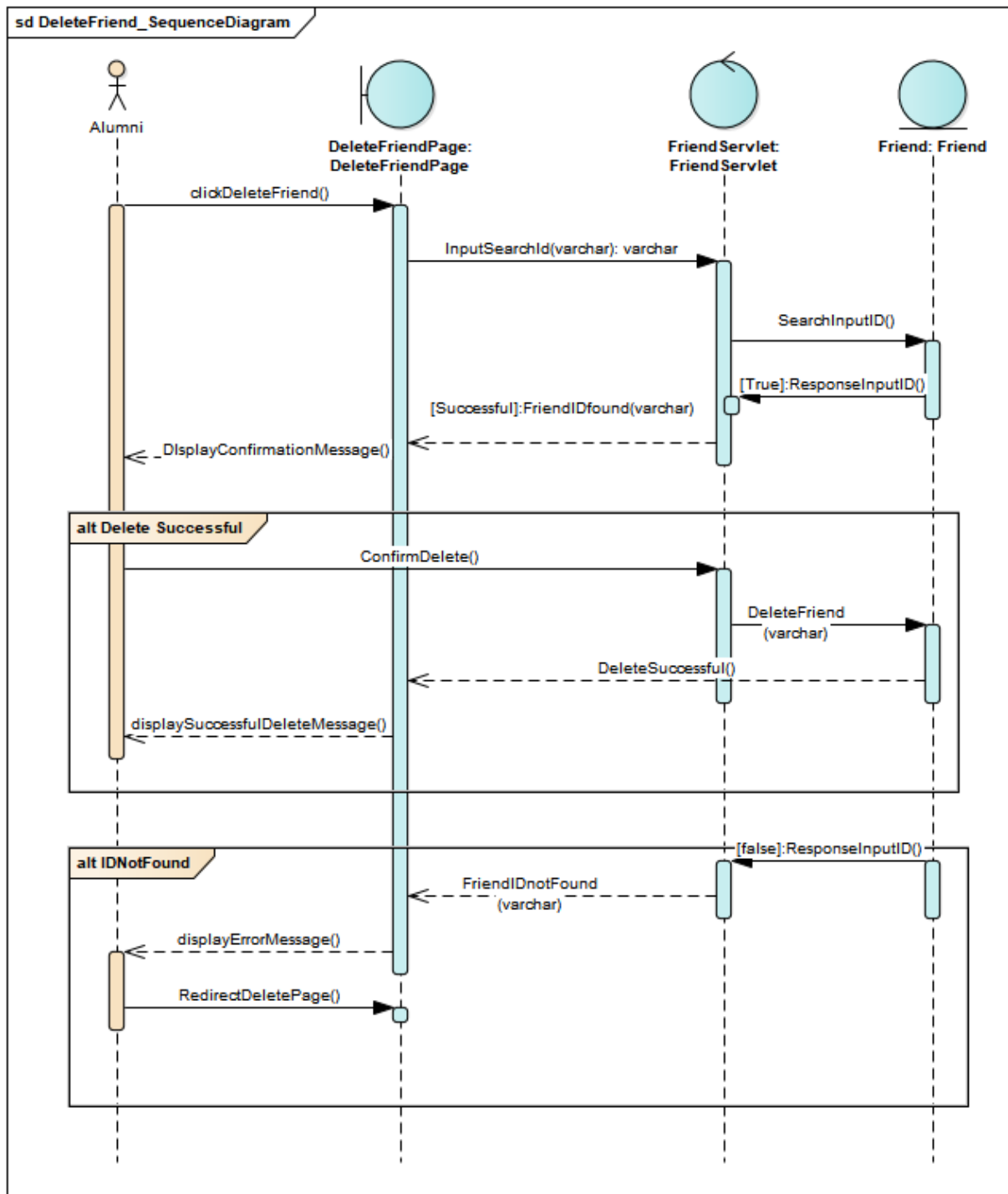


Figure 4.2.1.2.8: Sequence Diagram for <Delete Friend>

i) SD009: Sequence diagram for Search Friend

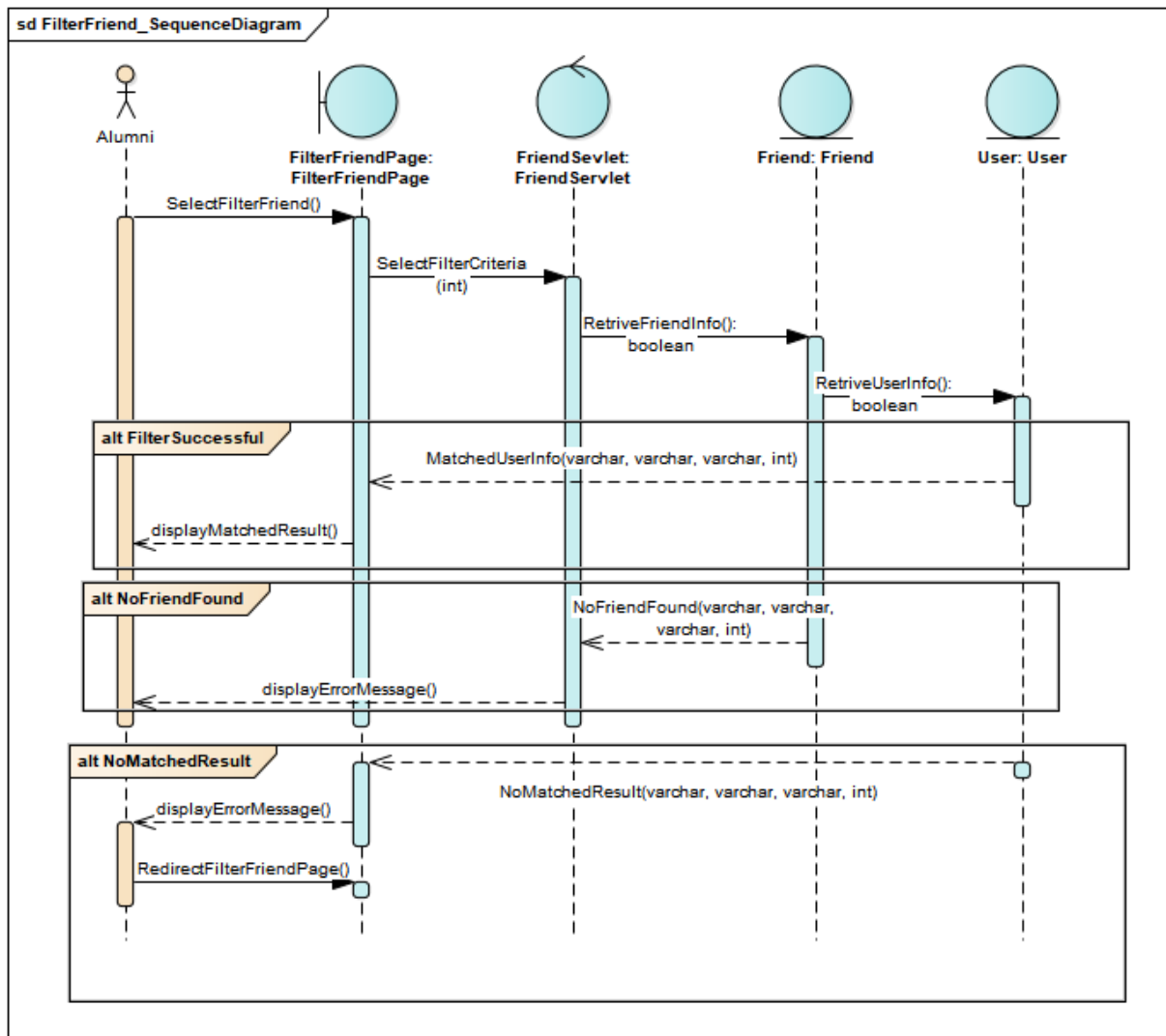


Figure 4.2.1.2.9: Sequence Diagram for <Search Friend>

j) SD010: Sequence diagram for Filter Friend

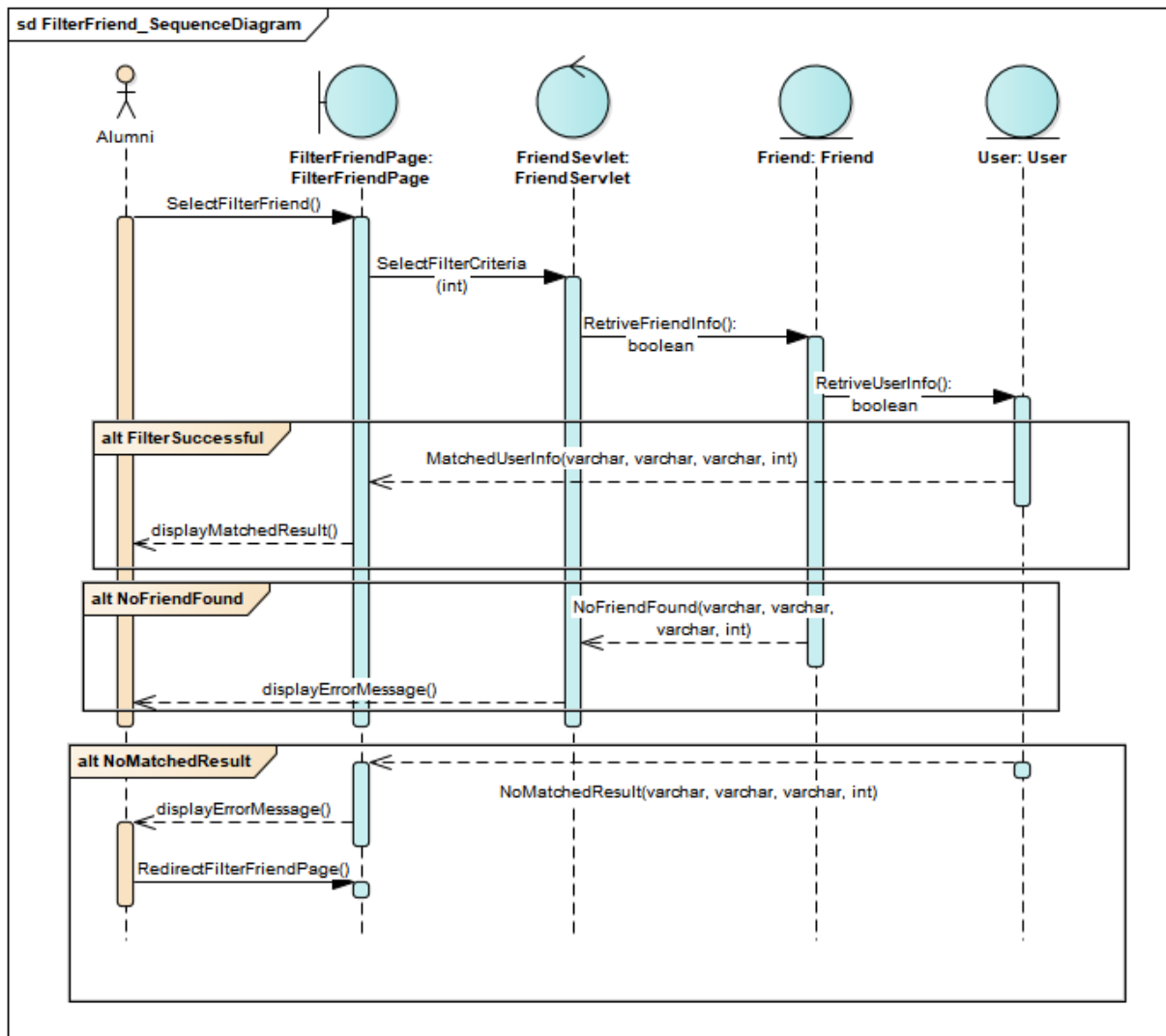
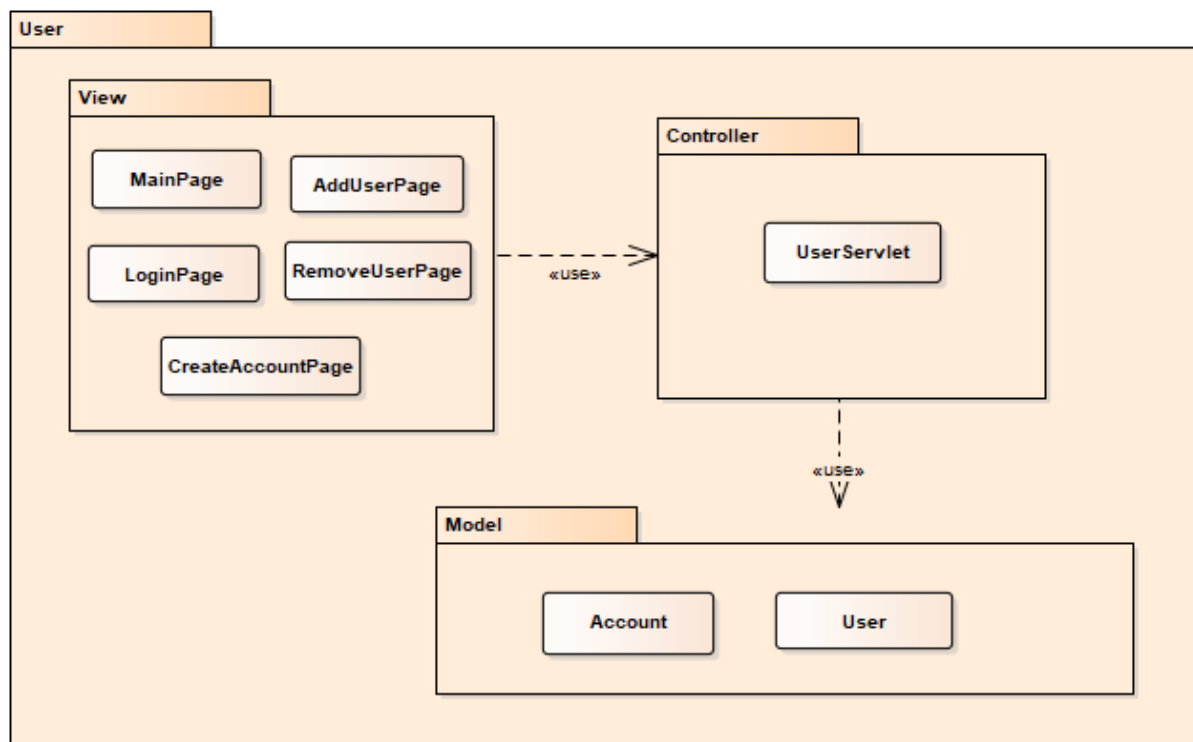


Figure 4.2.1.2.10: Sequence Diagram for <Filter Friend>

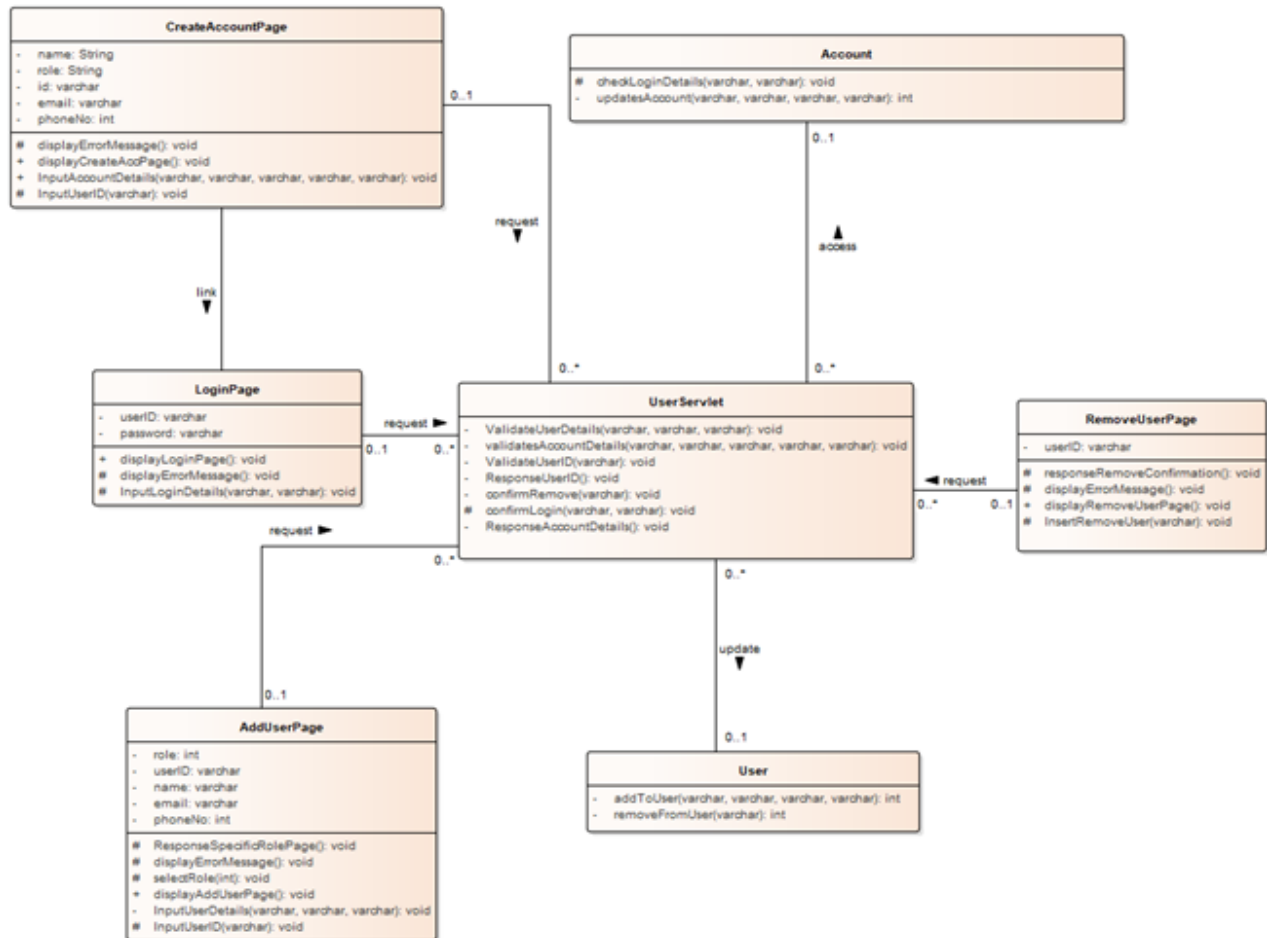


#### 4.2.2 P002: User Subsystem



**Figure 4.2.2 : Package Diagram for User Subsystem**

#### 4.2.2.1 Class Diagram



**Figure 4.2.2.1 : Class Diagram for User Subsystem**

<b>Entity Name</b>	LoginPage
<b>Method Name</b>	InputLoginDetails (userID,password)
<b>Input</b>	userID, password
<b>Output</b>	UserServlet get input and confirm login details
<b>Algorithm</b>	<ol style="list-style-type: none"> <li>1. User input userID and password on LoginPage</li> <li>2. UserServlet gets input and confirms login details.</li> </ol>

<b>Entity Name</b>	CreateAccountPage, AddUserPage
<b>Method Name</b>	InputUserID (userID)
<b>Input</b>	userID
<b>Output</b>	Servlet get input and validate userID
<b>Algorithm</b>	<ol style="list-style-type: none"> <li>1. User input userID.</li> <li>2. Servlet gets input and validates userID.</li> </ol>

<b>Entity Name</b>	CreateAccountPage
<b>Method Name</b>	InputAccountDetails (name, id, role, email, phoneNo)
<b>Input</b>	name, id ,role, email, phoneNo
<b>Output</b>	Servlet get input and validate account details
<b>Algorithm</b>	<ol style="list-style-type: none"> <li>1. User input account details on CreateAccountPage</li> <li>2. Servlet gets input and validates account details.</li> </ol>

<b>Entity Name</b>	AddUserPage
<b>Method Name</b>	SelectRole (role)
<b>Input</b>	role
<b>Output</b>	Request specific role page from Servlet
<b>Algorithm</b>	<ol style="list-style-type: none"> <li>1. User input role on AddUserPage.</li> <li>2. AddUserPage requests Servlet to respond to a specific role page.</li> </ol>

<b>Entity Name</b>	AddUserPage
<b>Method Name</b>	ResponseSpecificRolePage
<b>Input</b>	Request from SelectRole
<b>Output</b>	Response of SpecificRolePage

<b>Algorithm</b>	<ol style="list-style-type: none"> <li>1. UserServlet gets a request from SelectRole.</li> <li>2. UserServlet returns a response to AddUserPage.</li> <li>3. AddUserPage get response SpecificRolePage.</li> </ol>
------------------	--

<b>Entity Name</b>	AddUserPage
<b>Method Name</b>	InputUserDetails (name, id, role, email, phoneNo)
<b>Input</b>	name, id ,role, email, phoneNo
<b>Output</b>	UserServlet get input and validate user details
<b>Algorithm</b>	<ol style="list-style-type: none"> <li>1. User input user details on AddUserPage</li> <li>2. UserServlet gets input and validates user details.</li> </ol>

<b>Entity Name</b>	LoginPage, CreateAccountPage, AddUserPage, RemoveUserPage
<b>Method Name</b>	DisplayErrorMessage
<b>Input</b>	Invalid message from UserServlet
<b>Output</b>	Display error message
<b>Algorithm</b>	<ol style="list-style-type: none"> <li>1. Pages receive invalid message from UserServlet</li> <li>2. Pages displays the error message.</li> </ol>

<b>Entity Name</b>	RemoveUserPage
<b>Method Name</b>	InsertRemoveUser (userID)
<b>Input</b>	userID
<b>Output</b>	UserServlet get input and validate userID
<b>Algorithm</b>	<ol style="list-style-type: none"> <li>1. User input userID.</li> <li>2. UserServlet gets input and validates userID.</li> </ol>

<b>Entity Name</b>	RemoveUserPage
--------------------	----------------

<b>Method Name</b>	responseRemoveConfirmation
<b>Input</b>	HttpServletRequest confirm remove user
<b>Output</b>	RemoveUserPage get response remove confirmation
<b>Algorithm</b>	<ol style="list-style-type: none"> <li>1. HttpServletRequest confirm remove user</li> <li>2. HttpServletRequest send the response remove confirmation</li> </ol>

<b>Entity Name</b>	HttpServletRequest
<b>Method Name</b>	ValidateUserDetails (name,id,role, email, phoneNo)
<b>Input</b>	Request from user details input
<b>Output</b>	User details send to user entity
<b>Algorithm</b>	<ol style="list-style-type: none"> <li>1. HttpServletRequest get user details input request from AddUserPage</li> <li>2. HttpServletRequest validate the user details</li> <li>3. User details are sent to user entity</li> </ol>

<b>Entity Name</b>	HttpServletRequest
<b>Method Name</b>	ValidateUserID (userID)
<b>Input</b>	Request from userID input
<b>Output</b>	Send to user entity for check
<b>Algorithm</b>	<ol style="list-style-type: none"> <li>1. HttpServletRequest get userID input request from CreateAccountPage, AddUserPage and RemoveUserPage</li> <li>2. HttpServletRequest validate the userID</li> <li>3. HttpServletRequest sent to the user entity for checking.</li> </ol>

<b>Entity Name</b>	User
<b>Method Name</b>	ValidateAccountDetails (name,id,role, email, phoneNo)
<b>Input</b>	Request from account details input

<b>Output</b>	Account details sent to account entity
<b>Algorithm</b>	<ol style="list-style-type: none"> <li>1. UserServlet get account details input request from CreateAccountPage</li> <li>2. UserServlet validate the account details</li> <li>3. Account details are sent to account entity</li> </ol>

<b>Entity Name</b>	UserServlet
<b>Method Name</b>	confirmRemove (UserID)
<b>Input</b>	Request from remove user input
<b>Output</b>	Send to user entity for remove
<b>Algorithm</b>	<ol style="list-style-type: none"> <li>1. UserServlet gets request from remove user input of RemoveUserPage</li> <li>2. UserServlet send the input to user entity to remove the user.</li> </ol>

<b>Entity Name</b>	UserServlet
<b>Method Name</b>	confirmLogin (UserID, Password)
<b>Input</b>	Request from login details input
<b>Output</b>	Send to account entity to check login details
<b>Algorithm</b>	<ol style="list-style-type: none"> <li>1. UserServlet gets request from login details input of LoginPage</li> <li>2. UserServlet sends the login details to the account entity for check login details.</li> </ol>

<b>Entity Name</b>	UserServlet
<b>Method Name</b>	ResponseAccountDetails
<b>Input</b>	Checking result from account entity
<b>Output</b>	response of account details

<b>Algorithm</b>	<ol style="list-style-type: none"> <li>1. Account details checked in account entity.</li> <li>2. UserServlet get response of account details from account entity</li> </ol>
------------------	---

<b>Entity Name</b>	UserServlet
<b>Method Name</b>	ResponseUserID
<b>Input</b>	Checking result from user entity
<b>Output</b>	response of userID
<b>Algorithm</b>	<ol style="list-style-type: none"> <li>1. userID checked in user entity.</li> <li>2. UserServlet get response of userID from user entity</li> </ol>

<b>Entity Name</b>	User
<b>Method Name</b>	addToUser (name,id,role, email, phoneNo )
<b>Input</b>	User details from UserServlet
<b>Output</b>	User details add to user entity
<b>Algorithm</b>	<ol style="list-style-type: none"> <li>1. User servlet sent the user details to user entity</li> <li>2. User details are added to user entity</li> </ol>

<b>Entity Name</b>	User
<b>Method Name</b>	removeFromUser (userID )
<b>Input</b>	Remove confirmation from UserServlet
<b>Output</b>	UserID remove from user entity
<b>Algorithm</b>	<ol style="list-style-type: none"> <li>1. UserServlet sent the remove confirmation to the user entity</li> <li>2. UserID removed from user entity.</li> </ol>

<b>Entity Name</b>	Account
--------------------	---------

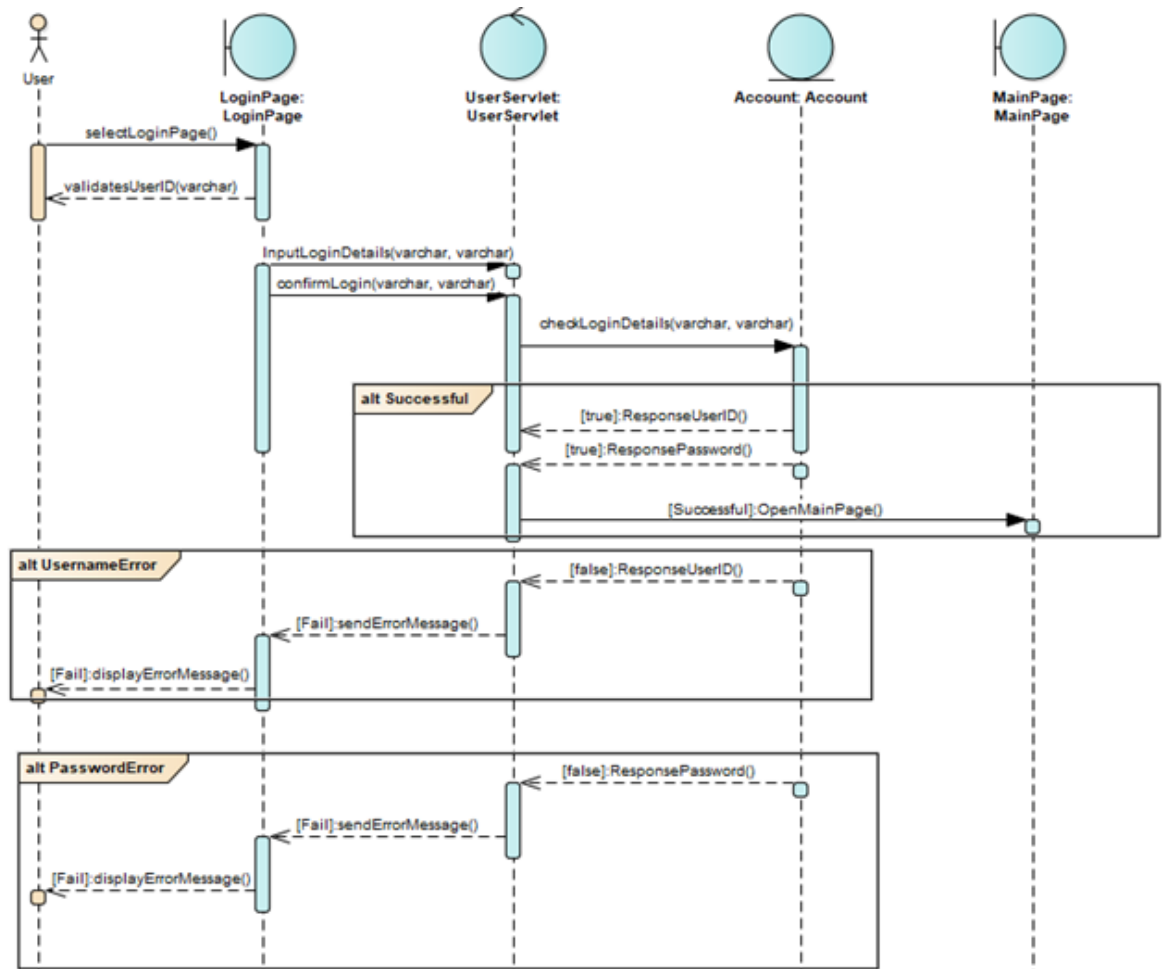
<b>Method Name</b>	checkLoginDetails (userID,password )
<b>Input</b>	Login confirmation from UserServlet
<b>Output</b>	Check login details
<b>Algorithm</b>	<ol style="list-style-type: none"> <li>1. UserServlet sent the login confirmation to the account entity.</li> <li>2. Login details checked in account entity.</li> </ol>

<b>Entity Name</b>	Account
<b>Method Name</b>	updateAccount (name,id,role, email, phoneNo)
<b>Input</b>	Account details from UserServlet
<b>Output</b>	Account details add to account entity
<b>Algorithm</b>	<ol style="list-style-type: none"> <li>1. User servlet sent the account details to account entity</li> <li>2. User details are updated to account entity</li> </ol>



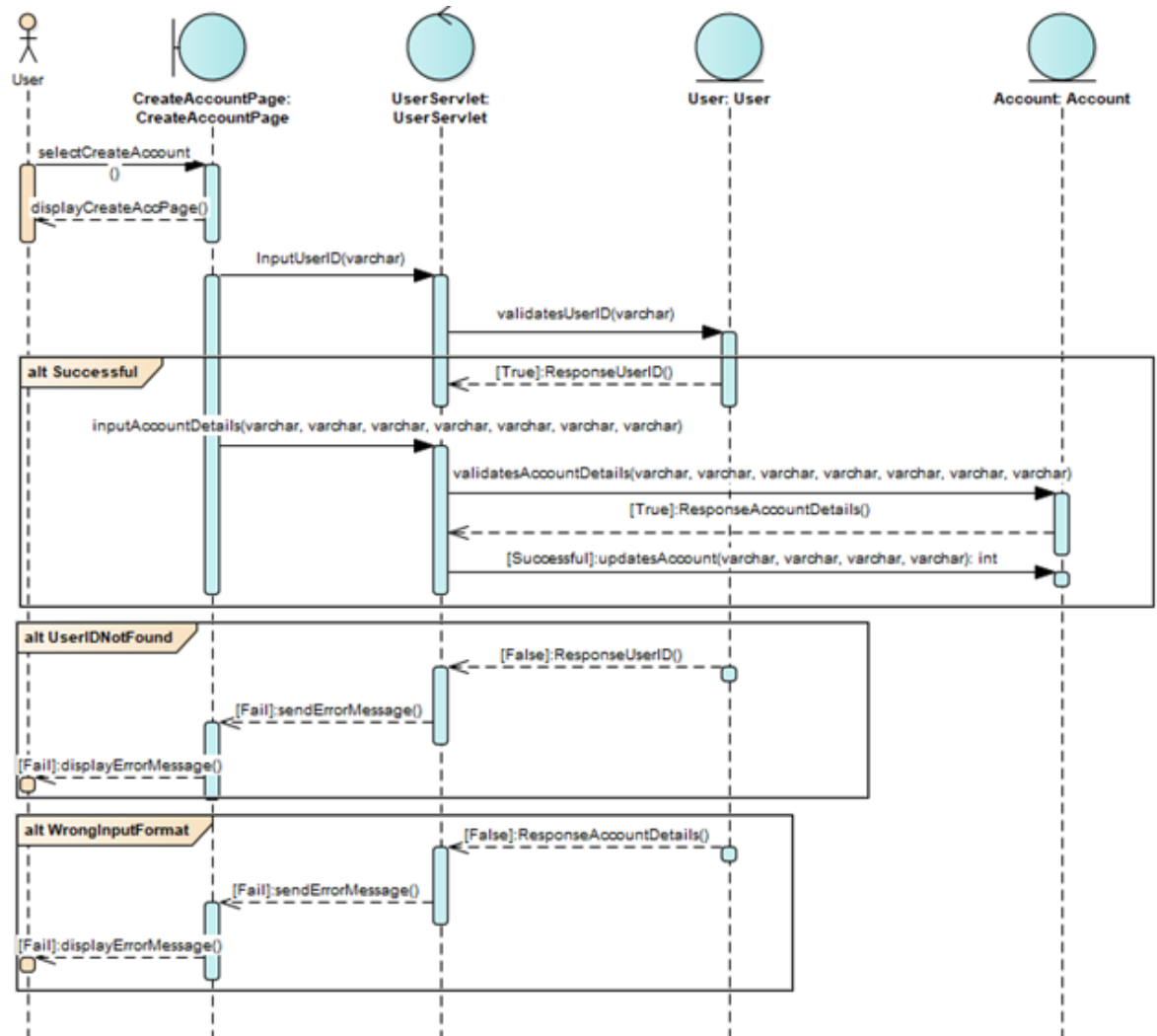
#### 4.2.2.2 Sequence Diagram

##### a) SD001: Sequence Diagram for Login



**Figure 4.2.2.2 : Sequence Diagram for Login**

## b) SD002: Sequence Diagram for Create Account



**Figure 4.2.2.2 : Sequence Diagram for Create Account**

c) SD003: Sequence Diagram for Add User

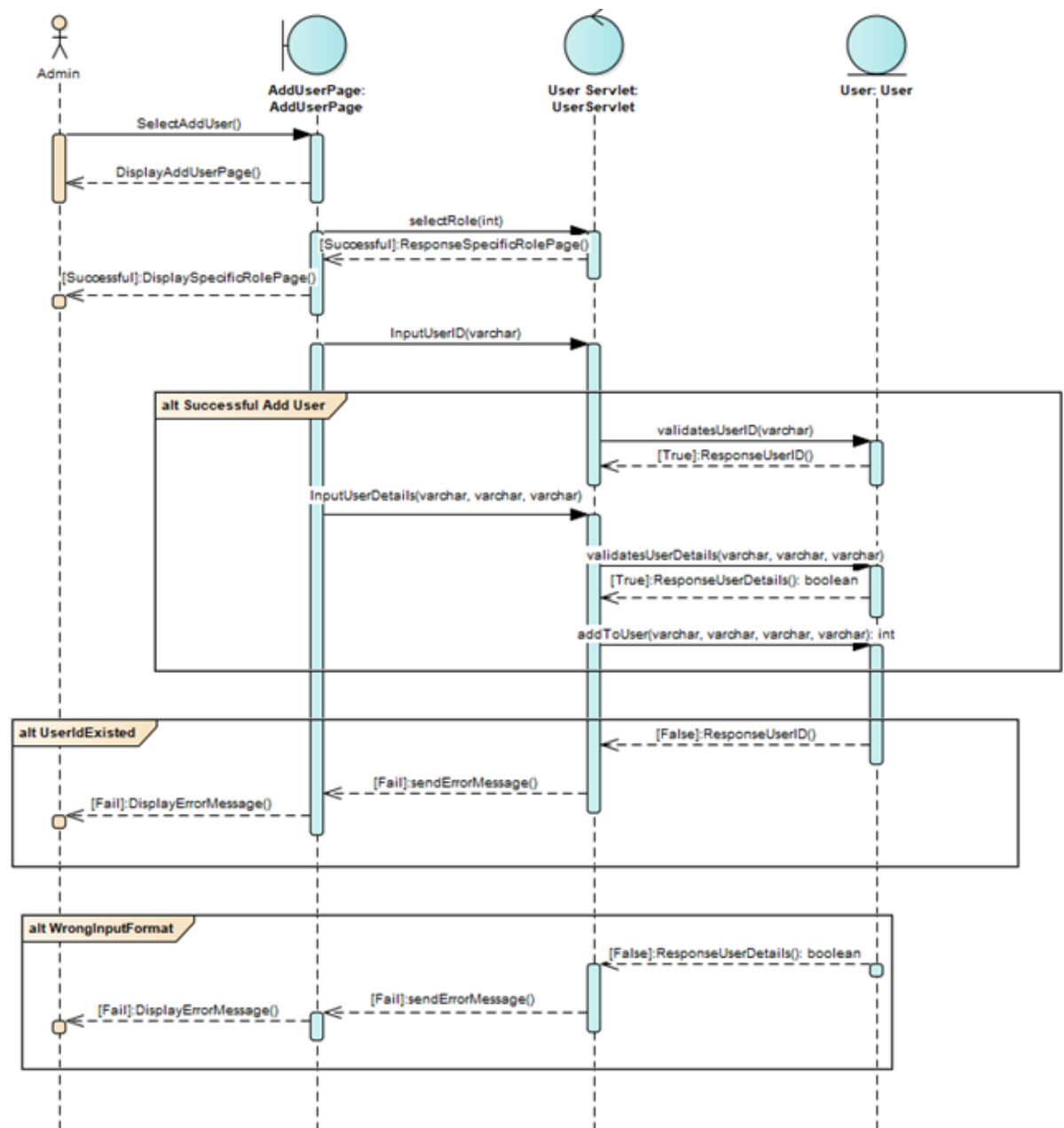


Figure 4.2.2.2 : Sequence Diagram for Add User

d) SD004: Sequence Diagram for Remove User

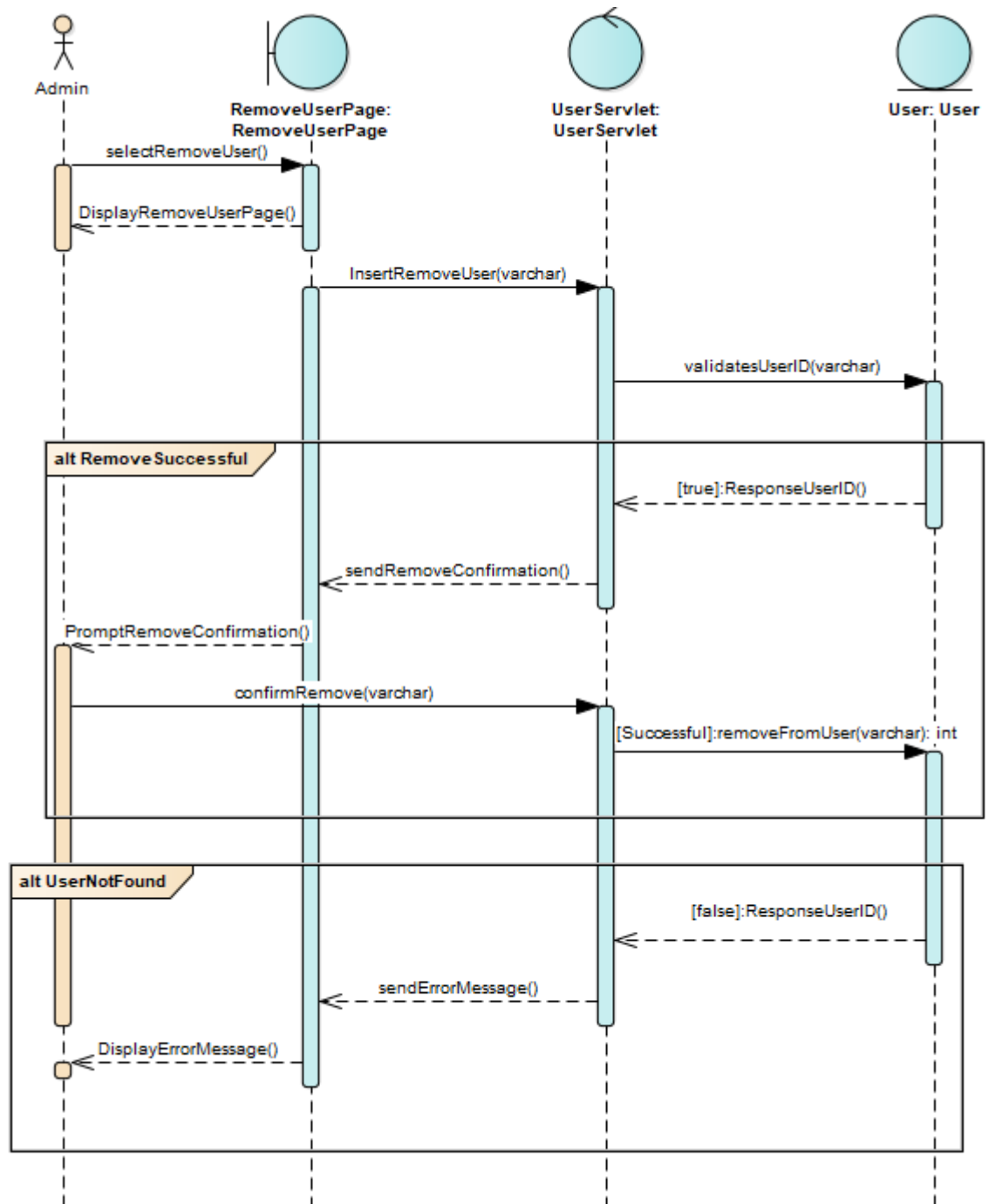


Figure 4.2.2.2 : Sequence Diagram for Remove User

### 4.2.3 P003: Package Mobile

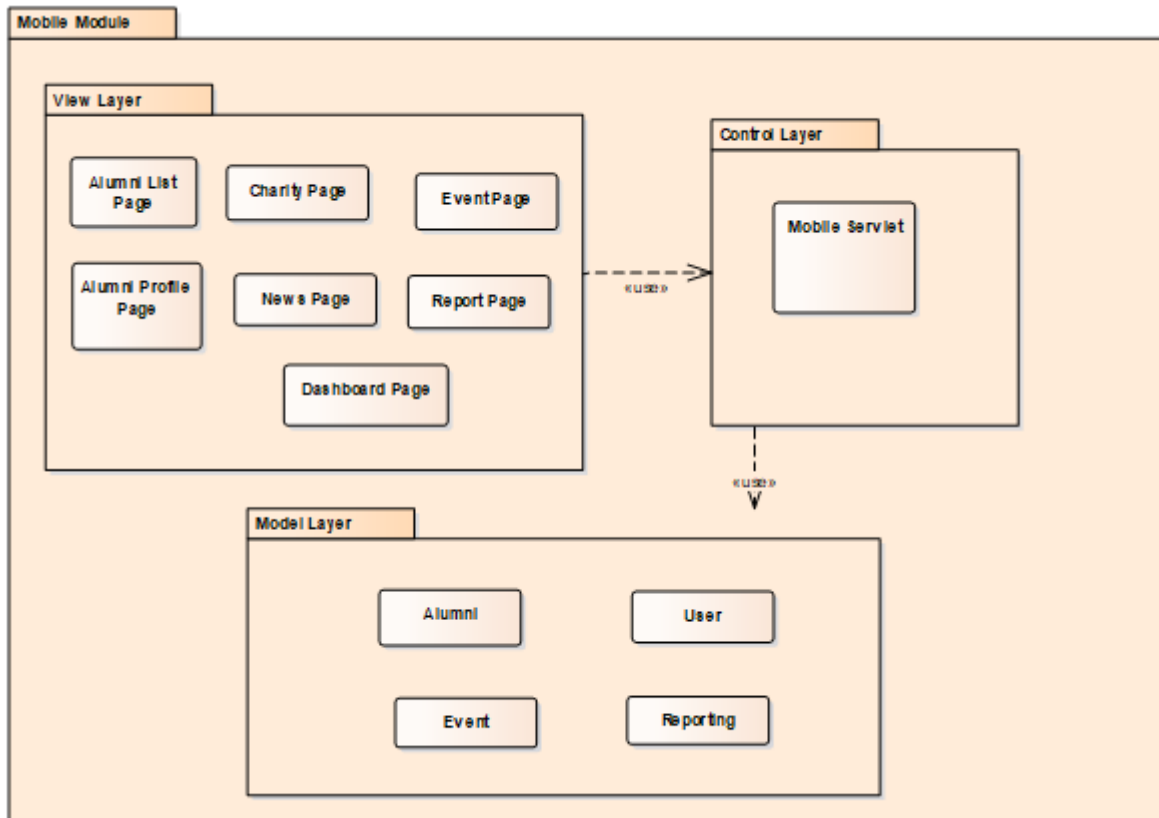


Figure 4.2.3: Package diagram for Mobile Module

#### 4.2.3.1 Class Diagram

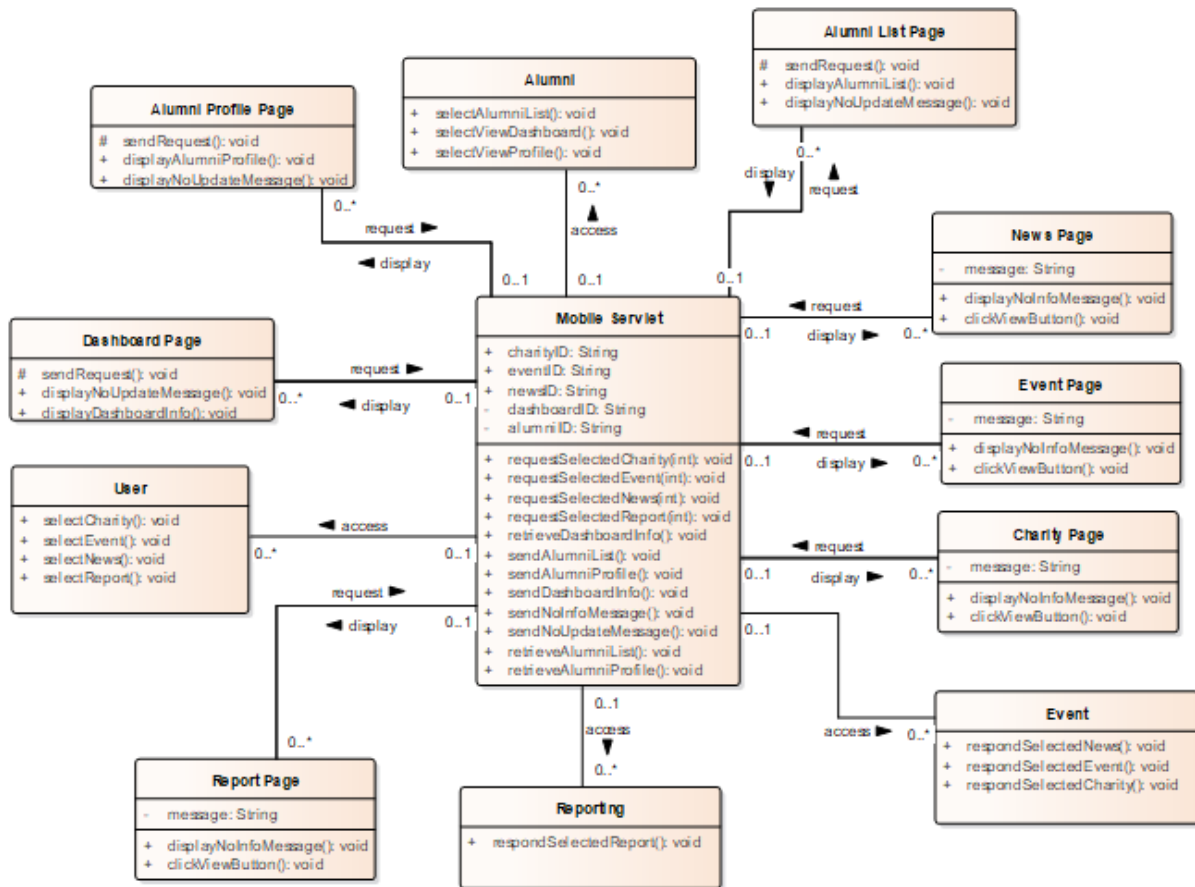


Figure 4.2.3.1: Class diagram for Mobile Package

Entity Name	Alumni List Page, Alumni Profile Page
Method Name	sendRequest()
Input	(Alumni List,/Alumni Profile) Page request
Output	Redirect to page information that the user selected.
Algorithm	<ol style="list-style-type: none"> <li>1. The user selects the option of Alumni List Page.</li> <li>2. The Alumni List Page sends a request to the mobile servlet to display Alumni List page information.</li> </ol>

<b>Entity Name</b>	<b>Mobile Servlet</b>
<b>Method Name</b>	<b>retrieveAlumniProfile()</b>
<b>Input</b>	<b>alumniID</b>
<b>Output</b>	<b>Return the requested alumni profile from the database.</b>
<b>Algorithm</b>	<ol style="list-style-type: none"> <li>1. Mobile Servlet retrieves the alumni profile information from the alumni database based on alumniID.</li> </ol>

<b>Entity Name</b>	<b>Alumni Profile Page</b>
<b>Method Name</b>	<b>displayAlumniProfile()</b>
<b>Input</b>	<b>Requested alumni profile</b>
<b>Output</b>	<b>Display the requested alumni profile.</b>
<b>Algorithm</b>	<ol style="list-style-type: none"> <li>1. The Alumni Profile Page displays information of the requested alumni profile.</li> </ol>

<b>Entity Name</b>	<b>Dashboard Page</b>
<b>Method Name</b>	<b>sendRequest()</b>
<b>Input</b>	<b>Dashboard Page request</b>
<b>Output</b>	<b>Redirect to page information that the user selected.</b>
<b>Algorithm</b>	<ol style="list-style-type: none"> <li>1. The user selects the option of Dashboard Page.</li> <li>2. The Dashboard Page sends a request to the mobile servlet to display Dashboard page information.</li> </ol>

<b>Entity Name</b>	<b>Mobile Servlet</b>
<b>Method Name</b>	<b>retrieveDashboardInfo()</b>
<b>Input</b>	<b>dashboardID</b>
<b>Output</b>	<b>Return the requested dashboard information from the database.</b>
<b>Algorithm</b>	<b>1. Mobile Servlet retrieves the dashboard information from the dashboard database based on dashboardID.</b>

<b>Entity Name</b>	<b>Dashboard Page</b>
<b>Method Name</b>	<b>displayDashboardInfo()</b>
<b>Input</b>	<b>Requested dashboard information</b>
<b>Output</b>	<b>Display the requested dashboard information.</b>
<b>Algorithm</b>	<b>1. The Dashboard Page displays information of the requested dashboard.</b>

<b>Entity Name</b>	<b>Alumni List Page, Alumni Profile Page</b>
<b>Method Name</b>	<b>displayNoUpdateMessage()</b>
<b>Input</b>	<b>Receive no update message from mobile servlet.</b>
<b>Output</b>	<b>Display no update message.</b>
<b>Algorithm</b>	<b>1. The Alumni List/Alumni Profile Page displays no update message to the user.</b>



<b>Entity Name</b>	<b>Mobile Servlet</b>
<b>Method Name</b>	<b>requestSelectedCharity(String )</b>
<b>Input</b>	<b>charityID</b>
<b>Output</b>	<b>Return selected charity page</b>
<b>Algorithm</b>	<ol style="list-style-type: none"> <li>1. The user selects an option from the Charity Page.</li> <li>2. The Charity Page sends a request to the mobile servlet to display selected charity page information based on charityNo.</li> </ol>

<b>Entity Name</b>	<b>Mobile Servlet</b>
<b>Method Name</b>	<b>requestSelectedEvent(String)</b>
<b>Input</b>	<b>eventID</b>
<b>Output</b>	<b>Return selected event page</b>
<b>Algorithm</b>	<ol style="list-style-type: none"> <li>1. The user selects an option from the Event Page.</li> <li>2. The Event Page sends a request to the mobile servlet to display selected event page information based on eventNo.</li> </ol>

<b>Entity Name</b>	<b>Mobile Servlet</b>
<b>Method Name</b>	<b>requestSelectedNews(String)</b>
<b>Input</b>	<b>newsID</b>
<b>Output</b>	<b>Return selected news page.</b>
<b>Algorithm</b>	<ol style="list-style-type: none"> <li>1. The user selects an option from the News Page.</li> <li>2. The News Page sends a request to the mobile servlet to display selected charity page information based on eventNo.</li> </ol>

<b>Entity Name</b>	<b>Mobile Sevlet</b>
<b>Method Name</b>	<b>requestSelectedReport(String)</b>
<b>Input</b>	<b>reportID</b>
<b>Output</b>	<b>Return selected report page.</b>
<b>Algorithm</b>	<ol style="list-style-type: none"> <li>1. The user selects an option from the Report Page.</li> <li>2. The Report Page sends a request to the mobile servlet to display selected report page information based on reportNo.</li> </ol>

<b>Entity Name</b>	<b>Event</b>
<b>Method Name</b>	<b>respondSelectedCharity()</b>
<b>Input</b>	<b>Selected charity information</b>
<b>Output</b>	<b>Return the selected charity information.</b>
<b>Algorithm</b>	<ol style="list-style-type: none"> <li>1. The database checks the existence of selected charity information.</li> <li>2. The database returns the information of the selected charity.</li> </ol>

<b>Entity Name</b>	<b>Event</b>
<b>Method Name</b>	<b>respondSelectedEvent()</b>
<b>Input</b>	<b>Selected event information</b>
<b>Output</b>	<b>Return the selected event information.</b>
<b>Algorithm</b>	<ol style="list-style-type: none"> <li>1. The database checks the existence of selected event information.</li> <li>2. The database returns the information of the selected event.</li> </ol>

<b>Entity Name</b>	<b>Event</b>
<b>Method Name</b>	<b>respondSelectedNews()</b>
<b>Input</b>	<b>Selected news information</b>
<b>Output</b>	<b>Return the selected news information.</b>
<b>Algorithm</b>	<ol style="list-style-type: none"> <li>1. The database checks the existence of selected news information.</li> <li>2. The database returns the information of the selected news.</li> </ol>

<b>Entity Name</b>	<b>Reporting</b>
<b>Method Name</b>	<b>respondSelectedReport()</b>
<b>Input</b>	<b>Selected report information</b>
<b>Output</b>	<b>Return the selected report information.</b>
<b>Algorithm</b>	<ol style="list-style-type: none"> <li>1. The database checks the existence of selected report information.</li> <li>2. The database returns the information of the selected report.</li> </ol>

<b>Entity Name</b>	<b>Mobile Servlet</b>
<b>Method Name</b>	<b>OpenCharityInfoPage()</b>
<b>Input</b>	<b>Selected charity information request</b>
<b>Output</b>	<b>Open and display the selected charity information.</b>
<b>Algorithm</b>	<ol style="list-style-type: none"> <li>1. Get selected charity information from the database.</li> <li>2. Display page of selected charity information.</li> </ol>

<b>Entity Name</b>	<b>Mobile Servlet</b>
<b>Method Name</b>	<b>OpenEventInfoPage()</b>
<b>Input</b>	<b>Selected event information request</b>
<b>Output</b>	<b>Open and display the selected event information.</b>
<b>Algorithm</b>	<ol style="list-style-type: none"> <li>1. Get selected event information from the database.</li> <li>2. Display page of selected event information.</li> </ol>

<b>Entity Name</b>	<b>Mobile Servlet</b>
<b>Method Name</b>	<b>OpenNewsInfoPage()</b>
<b>Input</b>	<b>Selected news information request</b>
<b>Output</b>	<b>Open and display the selected news information.</b>
<b>Algorithm</b>	<ol style="list-style-type: none"> <li>1. Get selected news information from the database.</li> <li>2. Display page of selected news information.</li> </ol>

<b>Entity Name</b>	<b>Mobile Sevlet</b>
<b>Method Name</b>	<b>OpenReportInfoPage()</b>
<b>Input</b>	<b>Selected report information request</b>
<b>Output</b>	<b>Open and display the selected report information.</b>
<b>Algorithm</b>	<ol style="list-style-type: none"> <li>1. Get selected report information from the database.</li> <li>2. Display page of selected report information.</li> </ol>

<b>Entity Name</b>	<b>Charity Page, Event Page, News Page, Report Page</b>
<b>Method Name</b>	<b>displayNoInfoMessage()</b>
<b>Input</b>	<b>Receive no information message from mobile servlet.</b>
<b>Output</b>	<b>Display no information message.</b>
<b>Algorithm</b>	<ol style="list-style-type: none"> <li>1. The page displays no information message to the user.</li> </ol>

### 4.2.3.2 Sequence Diagram

a) SD001: Sequence diagram for View Alumni List

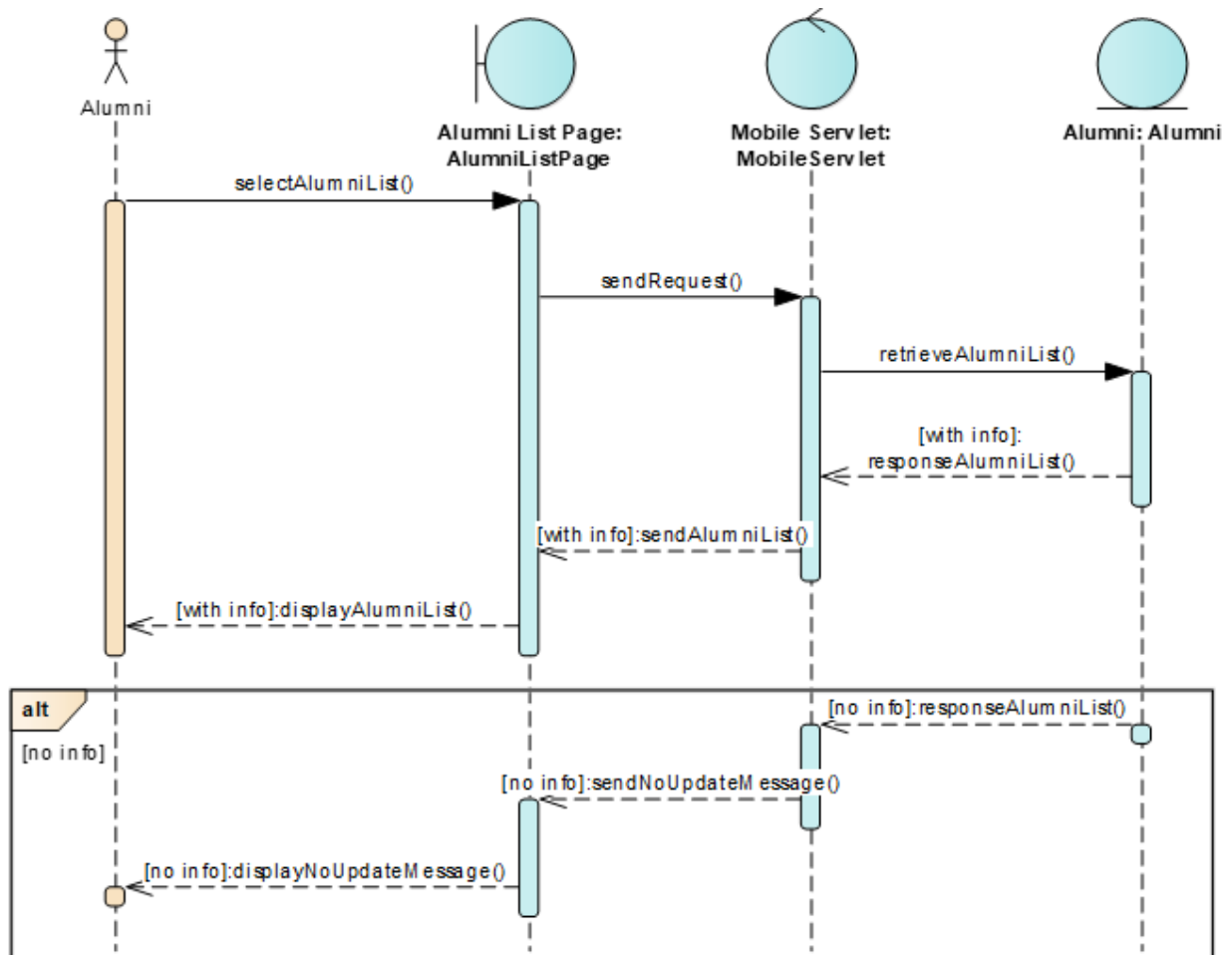


Figure 4.2.3.2.1: Sequence Diagram for View Alumni List

b) SD002: Sequence diagram for View Alumni Profile

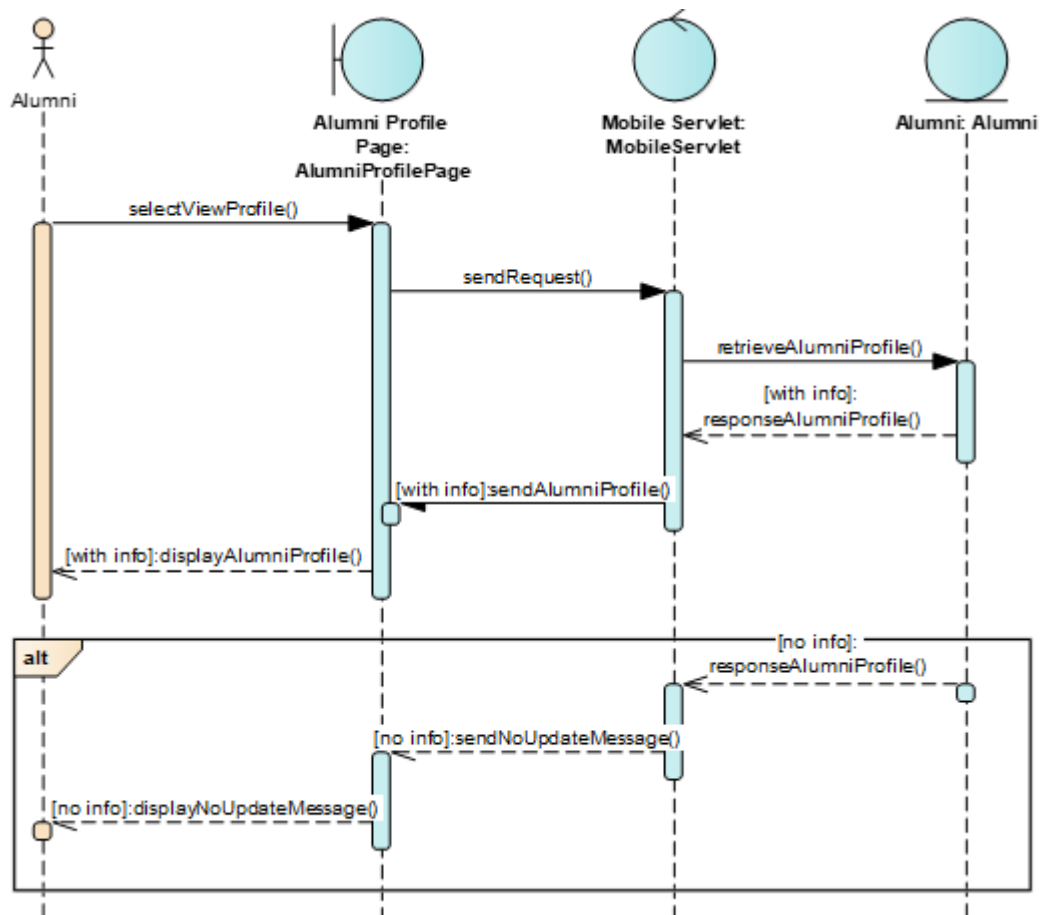


Figure 4.2.3.2.2: Sequence Diagram for View Alumni Profile

c) SD003: Sequence diagram for View Dashboard

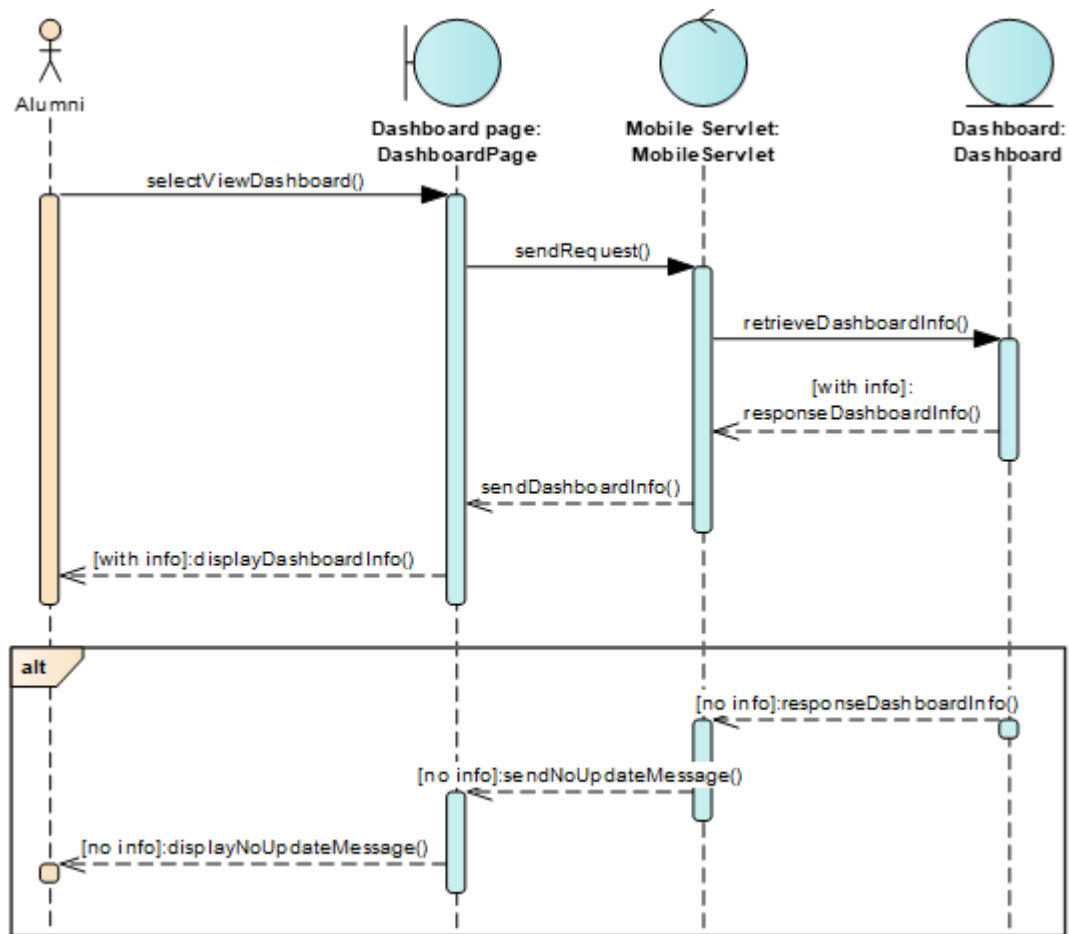


Figure 4.2.3.2.3: Sequence Diagram for View Dashboard



d) SD004: Sequence diagram for View Charity

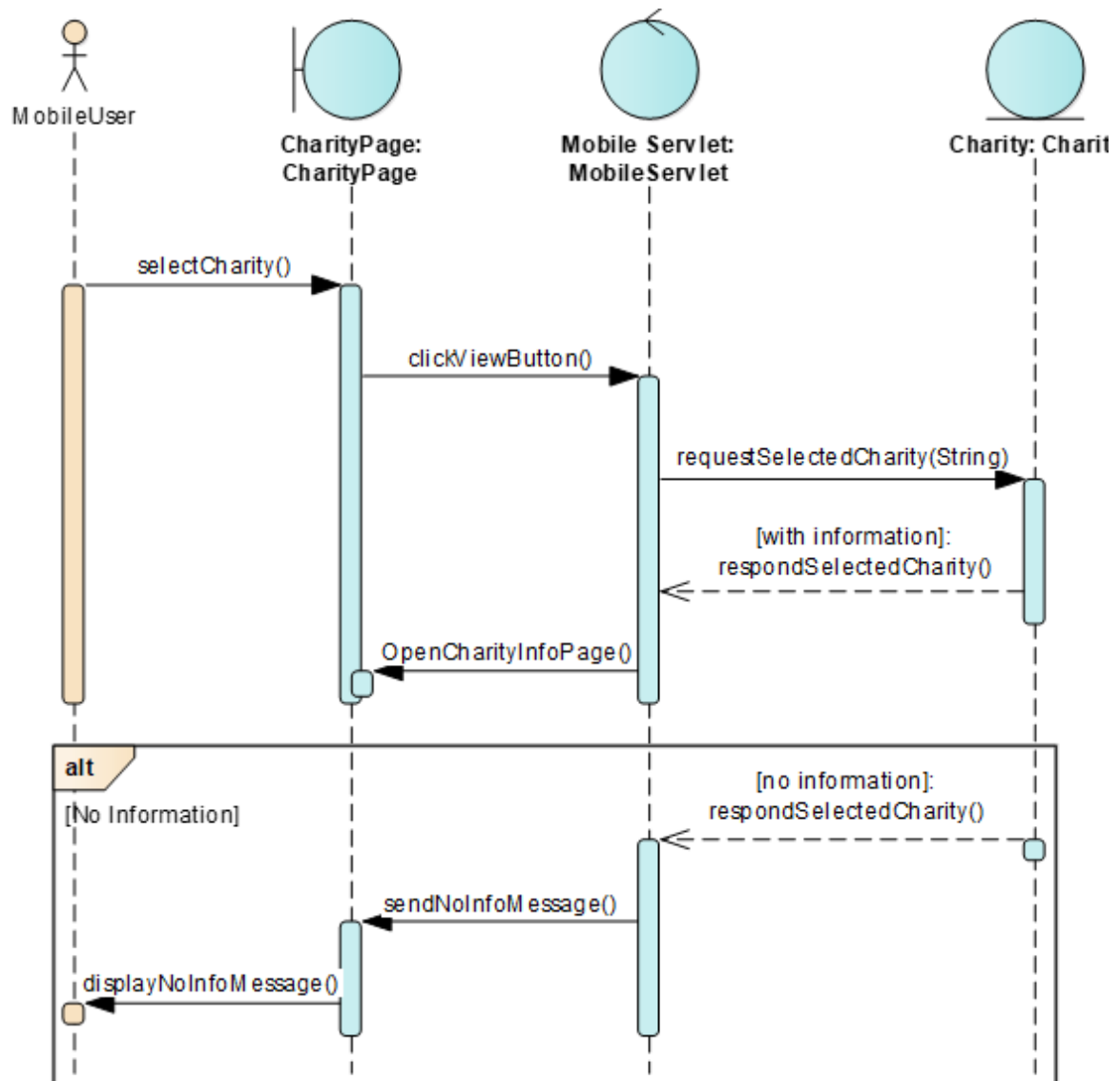


Figure 4.2.3.2.4: Sequence Diagram for View Charity

e) SD005: Sequence diagram for View Event

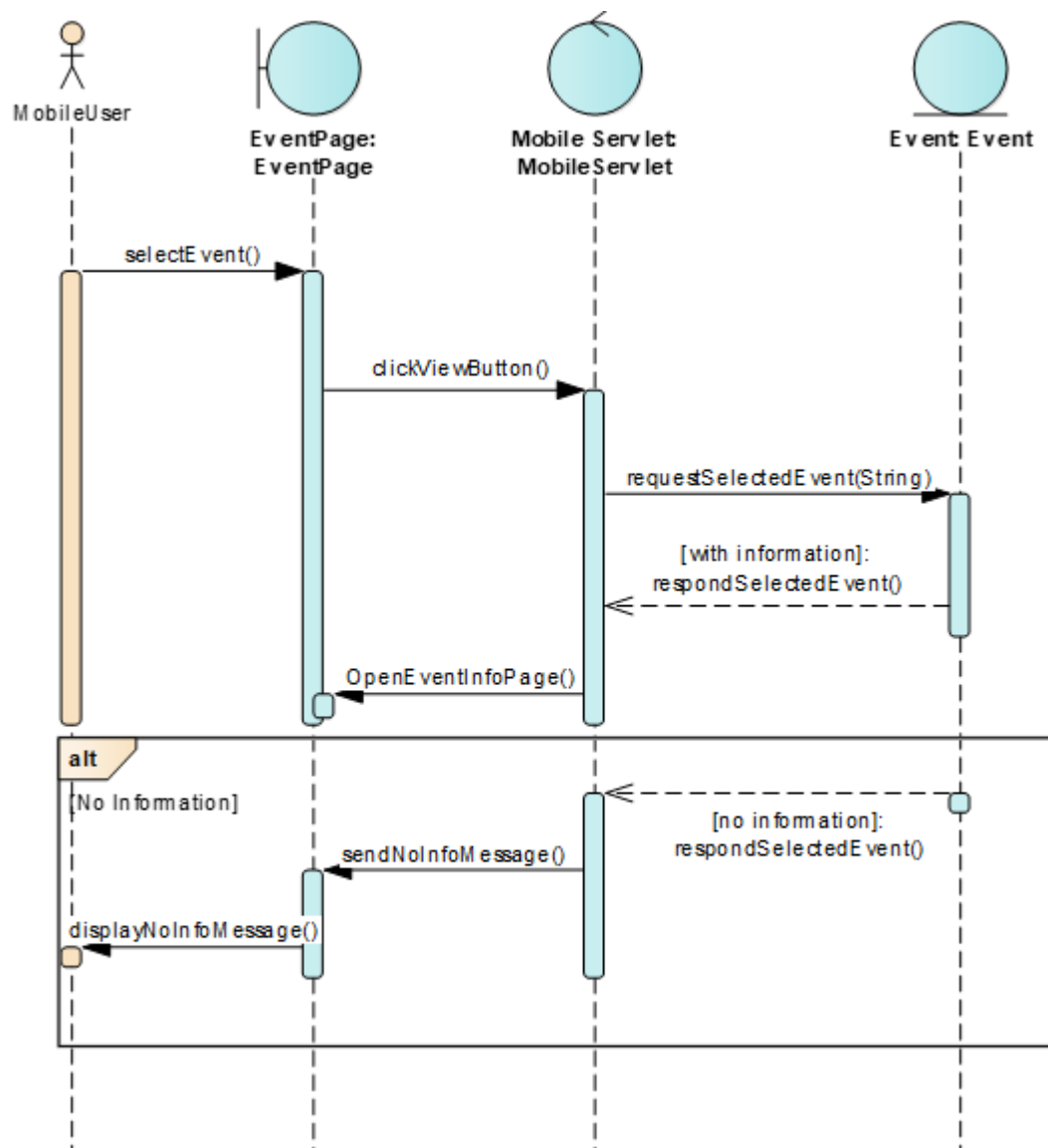


Figure 4.2.3.2.5: Sequence Diagram for View Event

f) SD006: Sequence diagram for View News

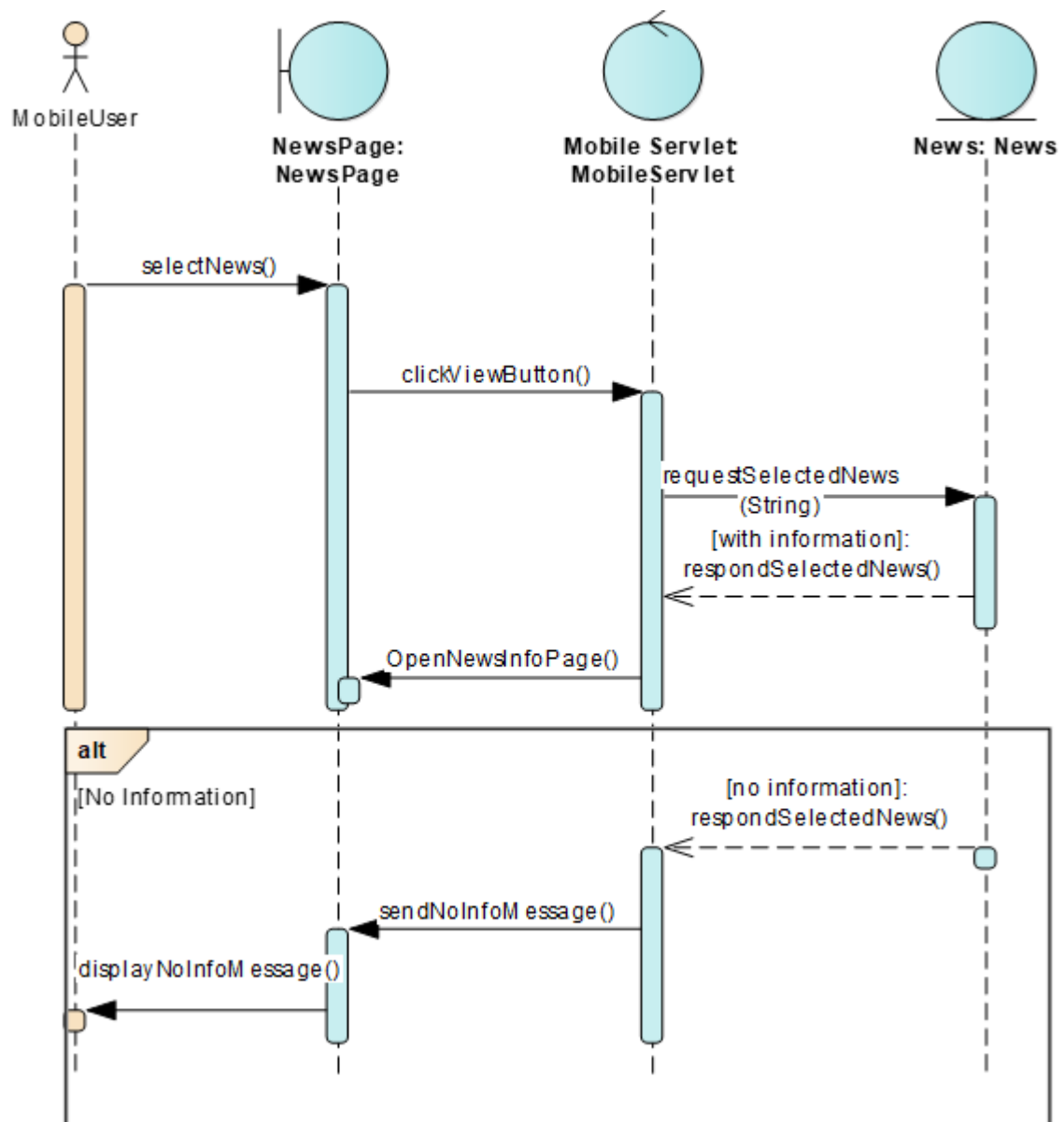


Figure 4.2.3.2.6: Sequence Diagram for View News

g) SD007: Sequence diagram for View Report

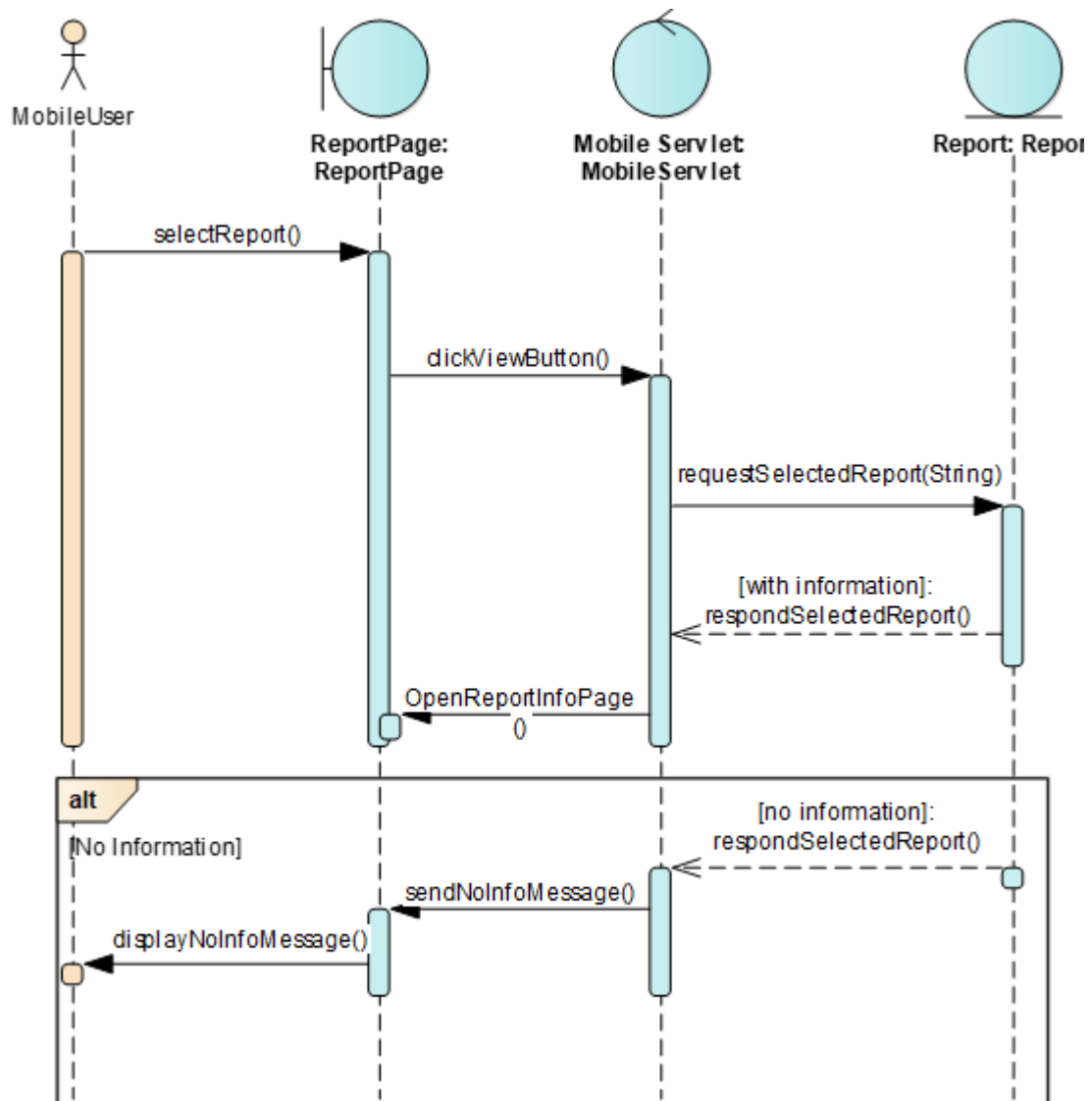
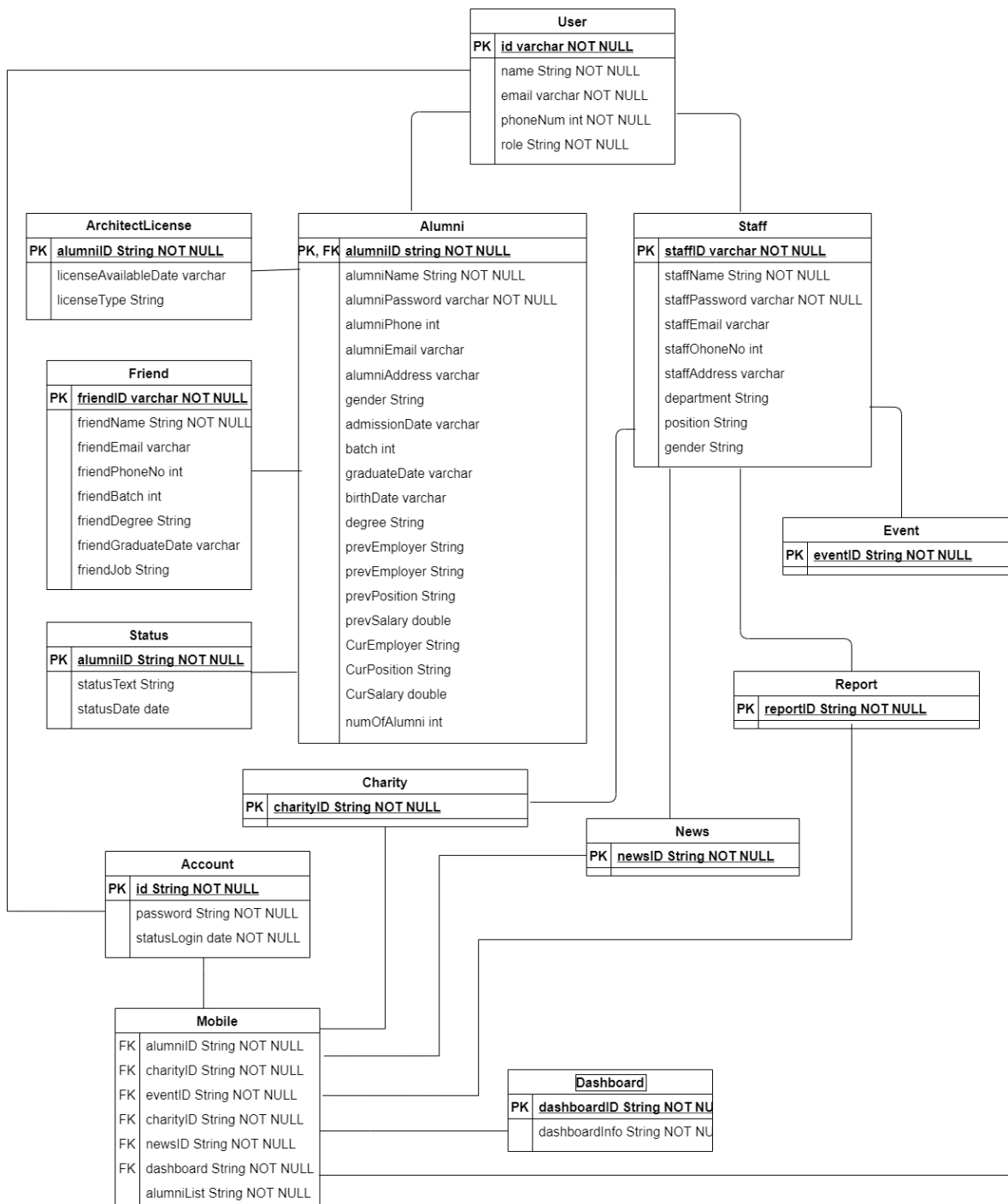


Figure 4.2.3.2.7: Sequence Diagram for View Report

## 5. Data Design



### 5.1 Data Description

The major data or system entities are stored in the system database. The data retrieved and stored into the database through controllers or servlets for example,

user servlet, alumni servlet, architecture license control, status controllers, friend servlet and mobile servlet.

The data is organized into attributes with recognizable names. To enable precise search and retrieval of the data, the file name of the document will reflect the content of the files

Entity Name	Description
User	The entity consists of the data related to the user.
Staff	The entity consists of the data related to the personal details of staff.
Alumni	The entity consists of the data related to the personal details of alumni.
Account	The entity consists of the data related to the account.
Status	The entity consists of the data related to the status.
Friend	The entity consists of the data related to the friend.
Architecture License	The entity consists of the data related to the architecture license.
Mobile	The entity consists of the data related to the alumni, dashboard, charity, news, event, report.
Dashboard	The entity consists of the data related to the dashboard.
Charity	The entity consists of the data related to the charity.
News	The entity consists of the data related to the news.
Event	The entity consists of the data related to the event.
Report	The entity consists of the data related to the report.

## 5.2 Data Dictionary

### 5.2.1 Entity: User

Attribute Name	Type	Description
id	varchar	Unique identifier for the users.
name	varchar	Name of the users.
role	varchar	Role of the users in the system.
email	varchar	Email of the users.
phoneNum	int	Telephone number of the users.

### 5.2.2 Entity: Staff

Attribute Name	Type	Description
staffID	String	Unique identifier for the staff.
staffName	String	Name of the staff.
staffEmail	varchar	Email of the staff.
staffPassword	varchar	Password for login into the system.
staffPhoneNo	int	Telephone number of the staff.
staffAddress	varchar	Home address.
position	String	Current position job.
gender	String	Gender of the staff.
department	String	Working department of the staff.

### 5.2.3 Entity: Alumni

Attribute Name	Type	Description
alumniID	String	Unique identifier for the alumni.
alumniName	String	Name of the alumni.
alumniEmail	varchar	Email of the alumni.
alumniPassword	varchar	Password for login into the system.
alumniPhone	int	Telephone number of the alumni.
alumniAddress	varchar	Home address.
birthDate	varchar	Birthdate of the alumni.
gender	String	Gender of the alumni.
batch	int	Batch year of the alumni
degree	String	Degree name of the alumni
admissionDate	varchar	Date of alumni admission.
graduateDate	varchar	Graduate date of the alumni.
CurEmployer	String	Current employer of the alumni
CurPosition	String	Current position of the alumni
curSalary	double	Current salary of the alumni
prevEmployer	String	Previous employer of the alumni
prevPosition	String	Previous position of the alumni
prevSalary	double	Previous salary of the alumni
employerAddress	varchar	Current employer address.



numOfAlumni	int	The number of alumni.
-------------	-----	-----------------------

#### 5.2.4 Entity: Account

Attribute Name	Type	Description
id	varchar	Unique identifier for the account.
password	varchar	Password for login into the system.
statusLogin	String	The account login status.

#### 5.2.5 Entity: Status

Attribute Name	Type	Description
statusText	string	Status text posted by the alumni.
alumniID	varchar	Unique identifier for the alumni.
statusDate	date	Date of the status posted.

#### 5.2.6 Entity: Friend

Attribute Name	Type	Description
friendID	varchar	Unique identifier for the alumni friend.
friendName	String	Name of the alumni friend.
friendEmail	varchar	Email of the alumni friend.
friendPhoneNo	int	Telephone number of the alumni friend.

friendDegree	String	Batch year of the alumni friend
friendBatch	int	Degree name of the alumni friend
friendGraduateDate	varchar	Graduate date of the alumni friend.
friendJob	String	Job of the alumni friend

### 5.2.7 Entity: Architecture License

Attribute Name	Type	Description
licenseType	String	Type of the architecture license.
licenseAvailableDate	varchar	Available date of the architecture license.

### 5.2.8 Entity: Mobile

Attribute Name	Type	Description
alumniID	String	Unique identifier for the alumni.
dashboardID	String	Unique identifier for the dashboard.
eventID	String	Unique identifier for the event.
charityID	String	Unique identifier for the charity.
newsID	String	Unique identifier for the news.
reportID	String	Unique identifier for the report.

### 5.2.9 Entity: Dashboard

Attribute Name	Type	Description
----------------	------	-------------

dashboardID	String	Unique identifier for the dashboard.
dashboardInfo	String	Dashboard detail information.

#### 5.2.10 Entity: Charity

Attribute Name	Type	Description
charityID	String	Unique identifier for the charity.

#### 5.2.11 Entity: News

Attribute Name	Type	Description
newsID	String	Unique identifier for the event.

#### 5.2.12 Entity: Event

Attribute Name	Type	Description
eventID	String	Unique identifier for the event.

#### 5.2.13 Entity: Report

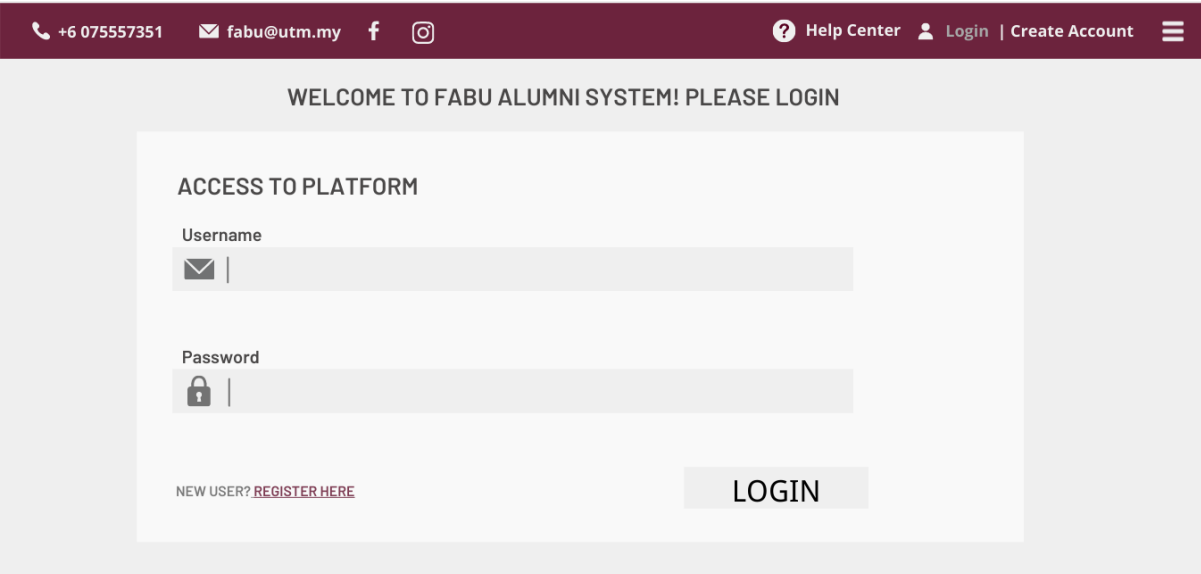
Attribute Name	Type	Description
reportID	String	Unique identifier for the report.

## 6. User Interface Design

### 6.1 Overview of User Interface

There are three user roles in this alumni management system which are alumni, staff and also the admin of the system. All three types of users need to login to the account by using their user ID and password. If they haven't registered an account, they need to register their account and fill up the details in the create account page. Once they created their account, they can log into the system by using the user ID and password registered. The alumni user may access event, news, charity, report, and dashboard data, as well as participate in events, sponsor events, donate to charities, modify and see their user profile, add, search, and remove friends, post status, and update architecture license. While for the staff, they can edit and view their user profile, add, edit and delete event, news, charity and reports

### 6.2 Screen Images



The screenshot displays the login interface of the FABU Alumni System. At the top, a dark red header bar contains contact information (+6 075557351, fabu@utm.my) and social media icons (Facebook, Instagram). It also includes links for 'Help Center', 'Login', and 'Create Account', along with a hamburger menu icon. Below the header, a light gray box contains the text 'WELCOME TO FABU ALUMNI SYSTEM! PLEASE LOGIN'. Inside this box is a white form titled 'ACCESS TO PLATFORM'. The form has two input fields: 'Username' with an envelope icon and 'Password' with a lock icon. Below the password field is a link for 'NEW USER? REGISTER HERE'. A 'LOGIN' button is positioned to the right of the form.

Screen Image 1: Login page of FABU Alumni System

If the users haven't registered an account, User need to create an account

The screenshot shows a web application header with a dark red background. On the left, there is a phone icon, the number '+6 075557351', an email icon, the address 'fabu@utm.my', and social media icons for Facebook and Instagram. On the right, there is a question mark icon, the text 'Help Center', a user icon, the text 'Login | Create Account', and a hamburger menu icon. Below the header, the main content area has a light gray background. At the top of this area, the text 'CREATE YOUR ACCOUNT' is centered. Below this, there is a white rectangular box containing two input fields. The first field is labeled 'Role' and the second is labeled 'ID'. Below these fields is a dark red button with the text 'Next' in white.

+6 075557351 fabu@utm.my f i ? Help Center Login | Create Account

CREATE YOUR ACCOUNT

Role

ID

Next

The screenshot shows the same web application header as the first image. Below the header, the main content area has a light gray background. At the top of this area, the text 'CREATE YOUR ACCOUNT' is centered. Below this, there is a white rectangular box containing six input fields. The labels for these fields are 'Role', 'User Id', 'Email', 'Password', 'Full Name', and 'Phone Number', arranged vertically from top to bottom.

+6 075557351 fabu@utm.my f i ? Help Center Login | Create Account

CREATE YOUR ACCOUNT

Role

User Id

Email

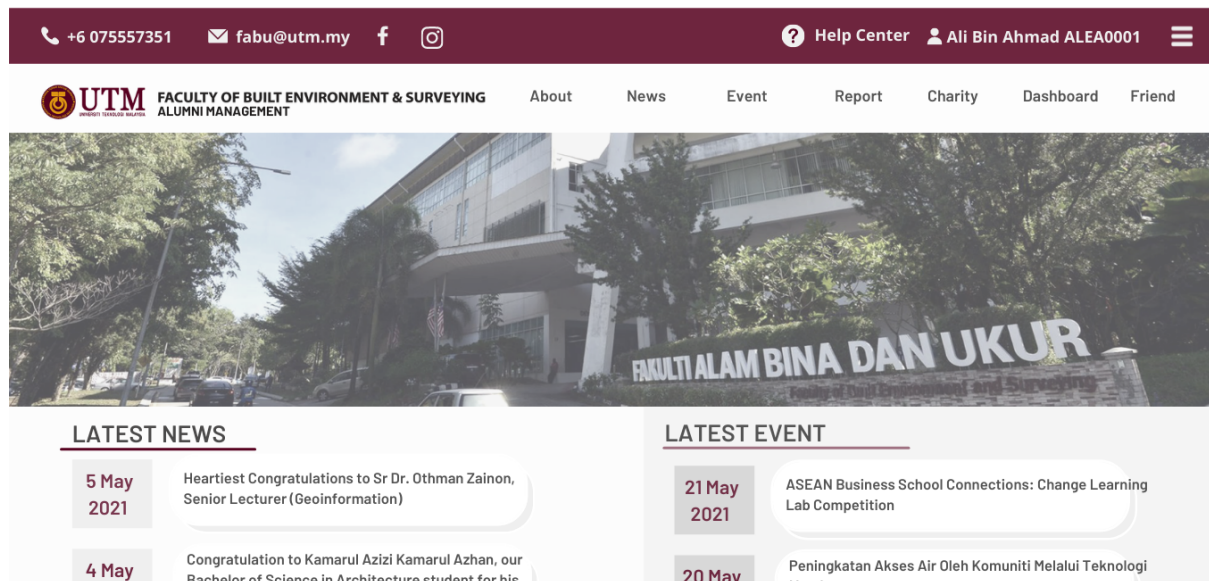
Password

Full Name

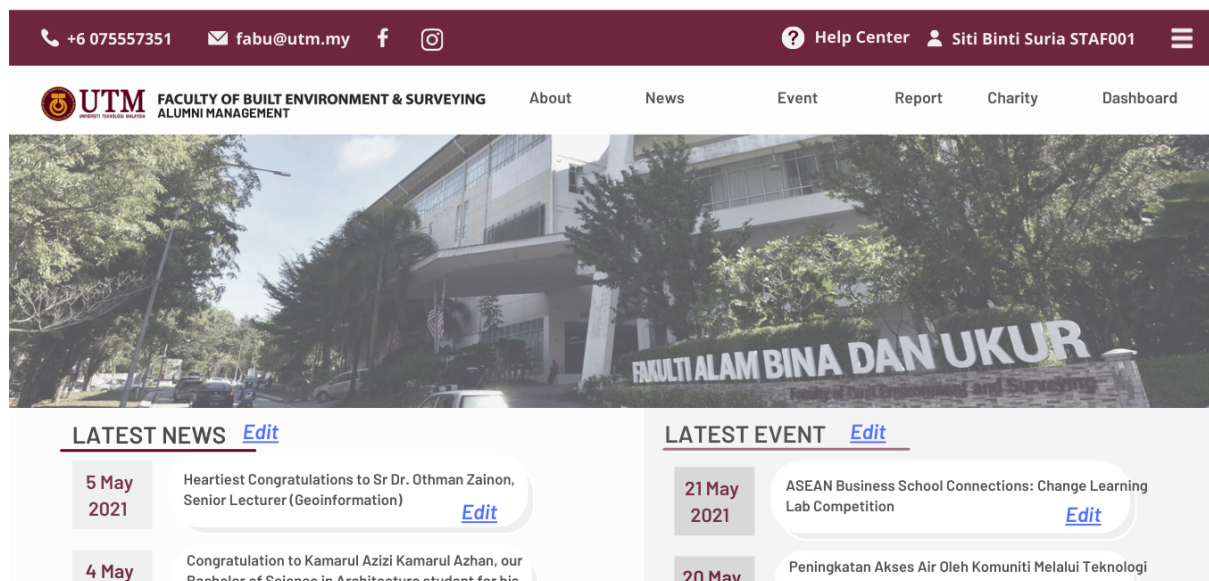
Phone Number

Screen Image 2: Create account page

After the alumni user or staff successfully login into the system, they will go to the main page of the system.



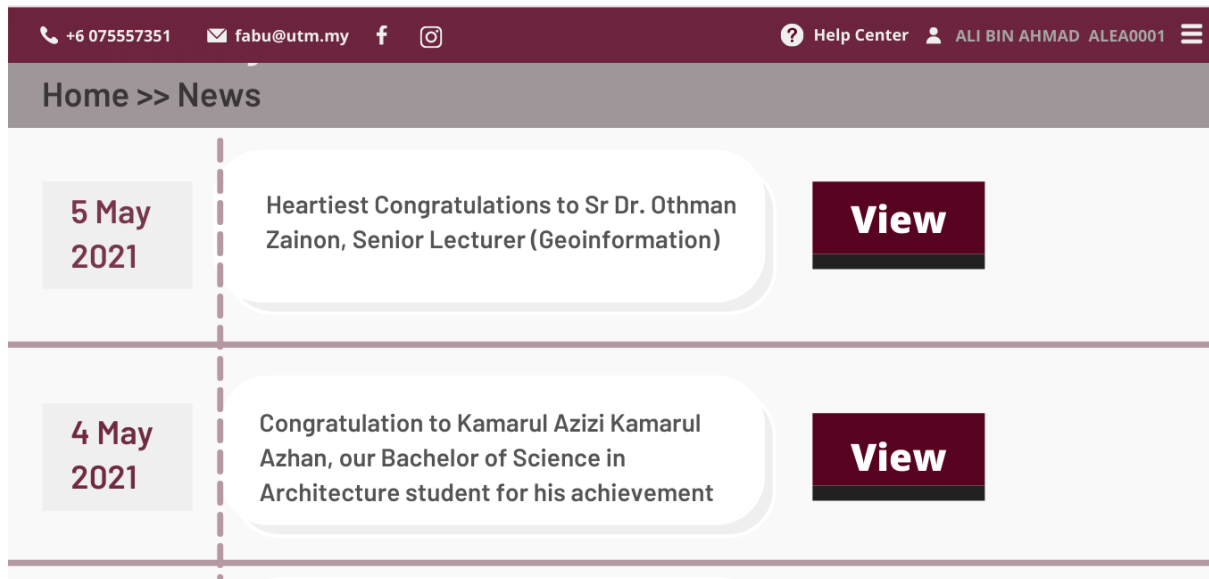
Screen Image 3: The main page of staff after successful login



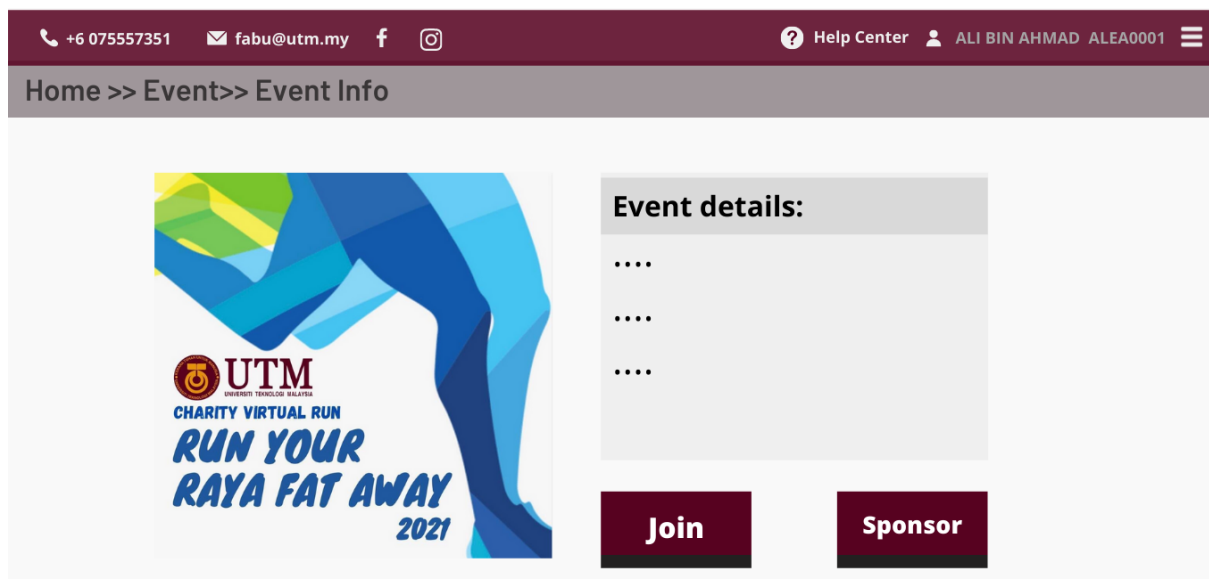
Screen Image 4: The main page of alumni after successful login

After the alumni user or staff successfully login into the alumni system, they can use some features from this system.

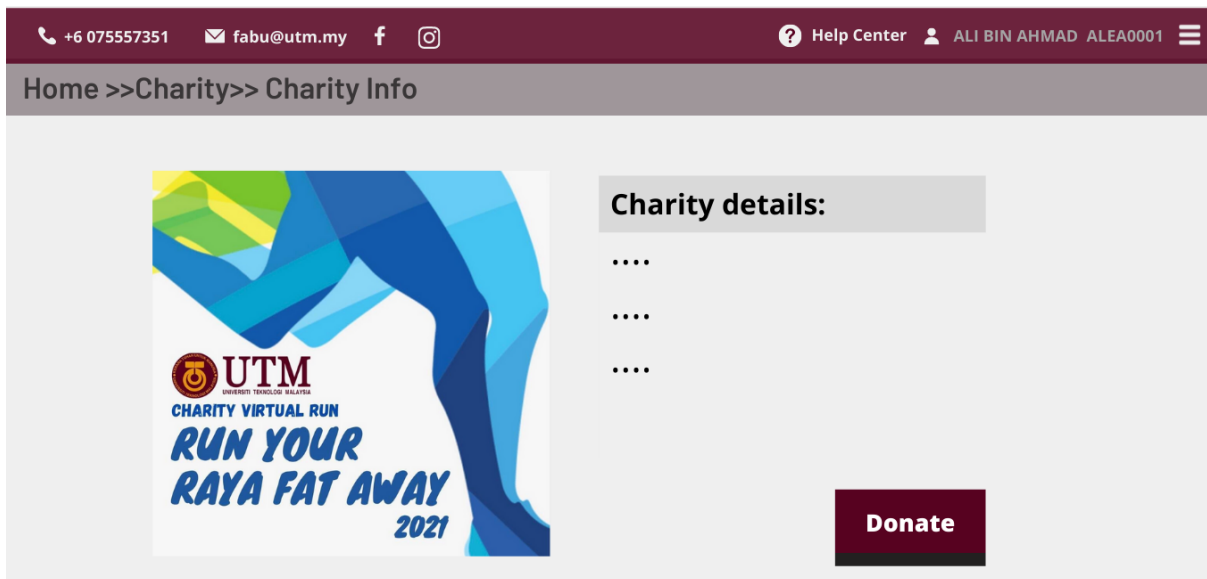
If it is login as an alumni user, they can view the event, news, charity, report and dashboard data by clicking the related options at the main page.



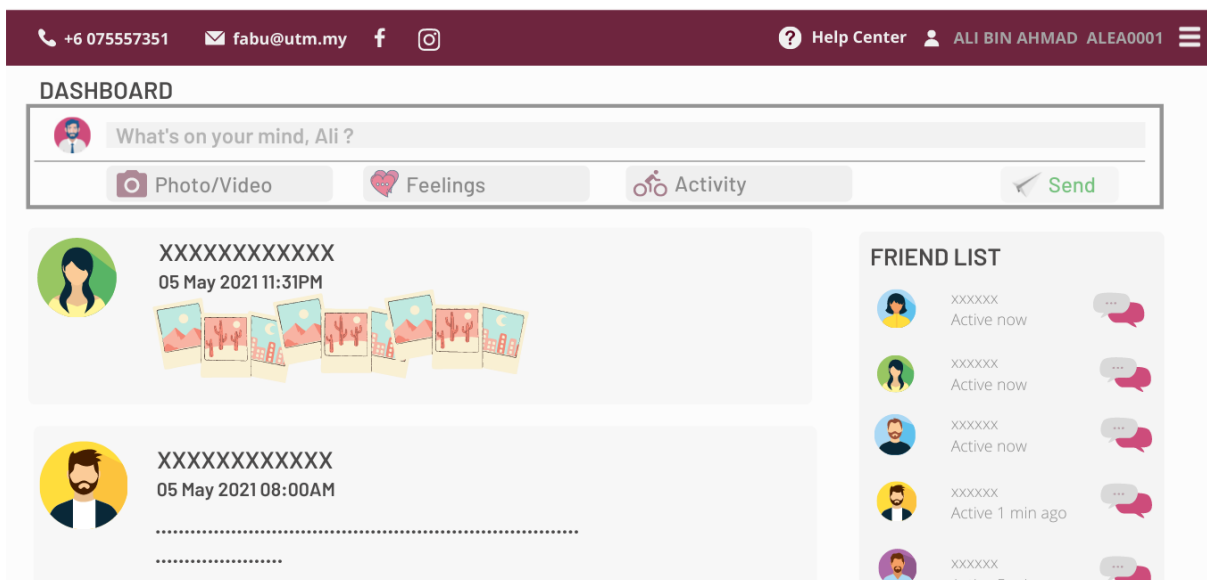
Screen Image 5:: The news page for alumni user to view the news



Screen Image 6: The event page for alumni user to view the event

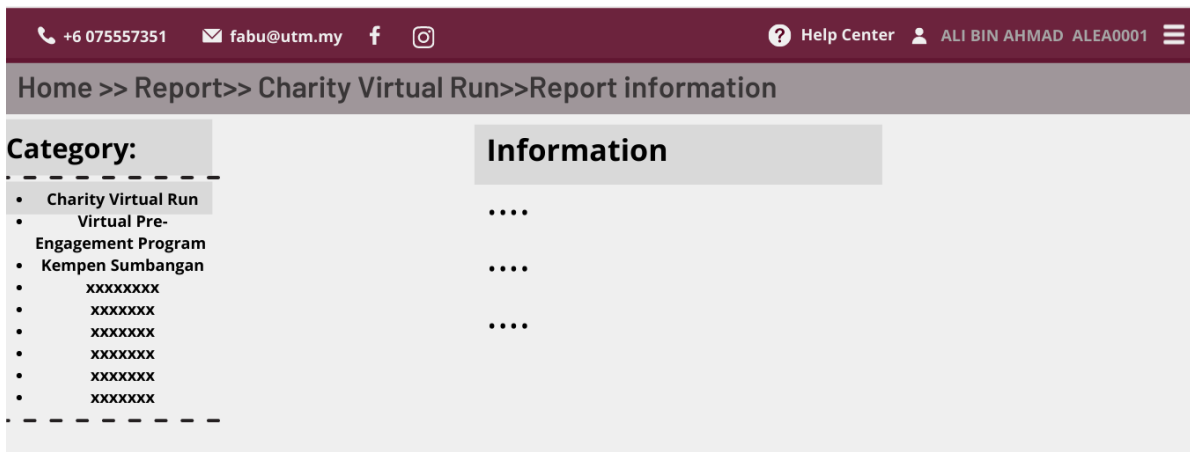


Screen Image 7: The charity page for alumni user to view the charity and make donation if interested



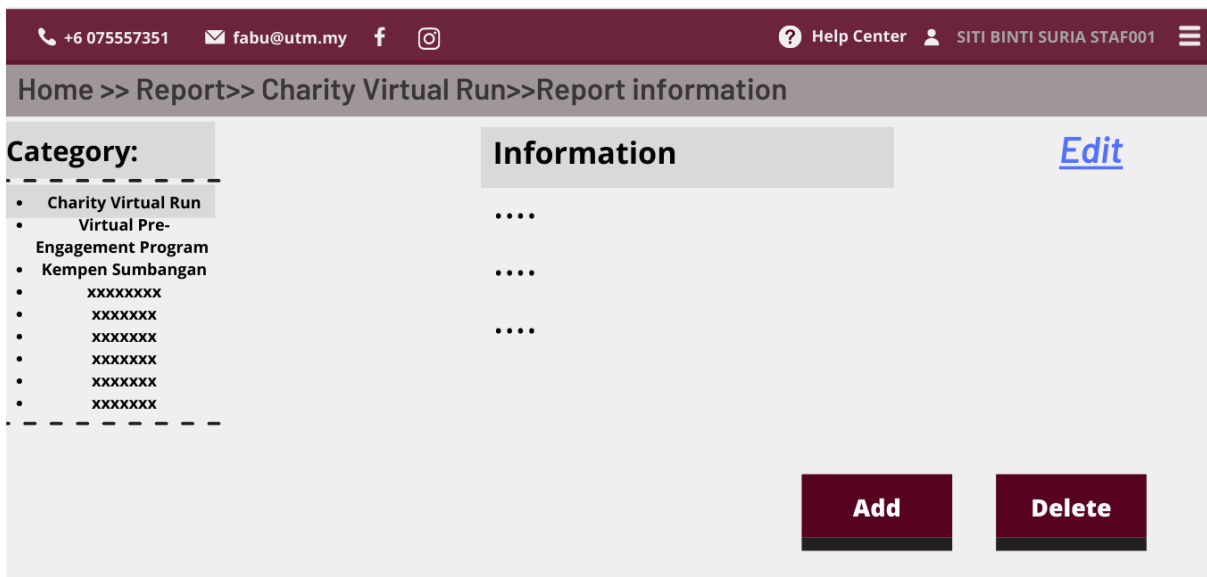
Screen Image 8: The dashboard page for alumni user to view the dashboard data



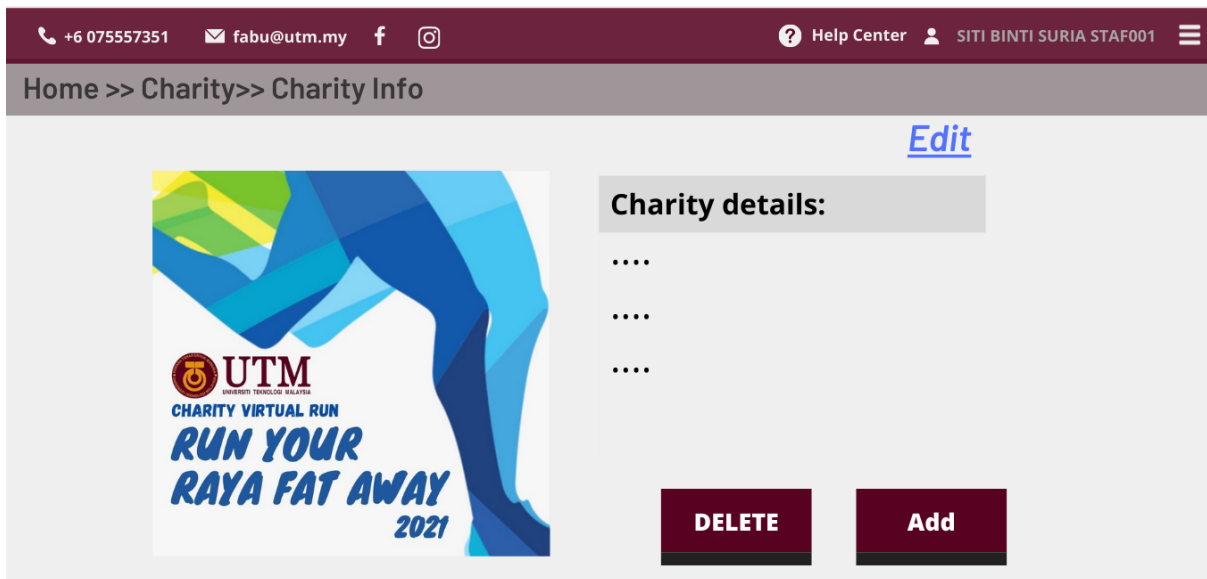


Screen Image 9: The report page for alumni user to view the report

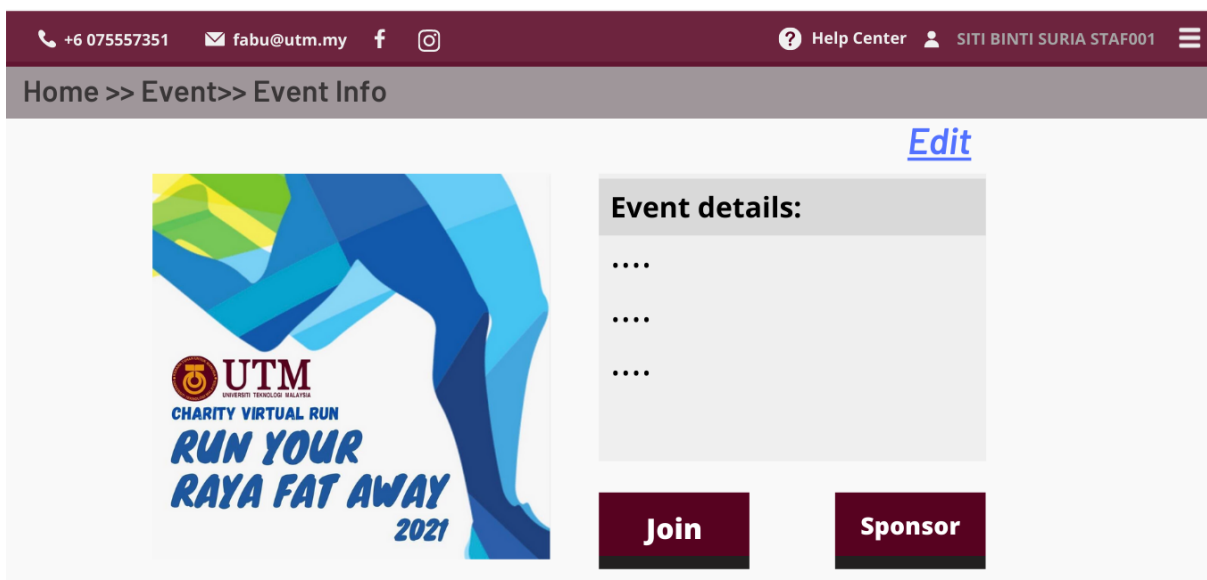
If it is login as a staff, they can add, edit and delete event, news, charity and reports by clicking the related options at the main page.



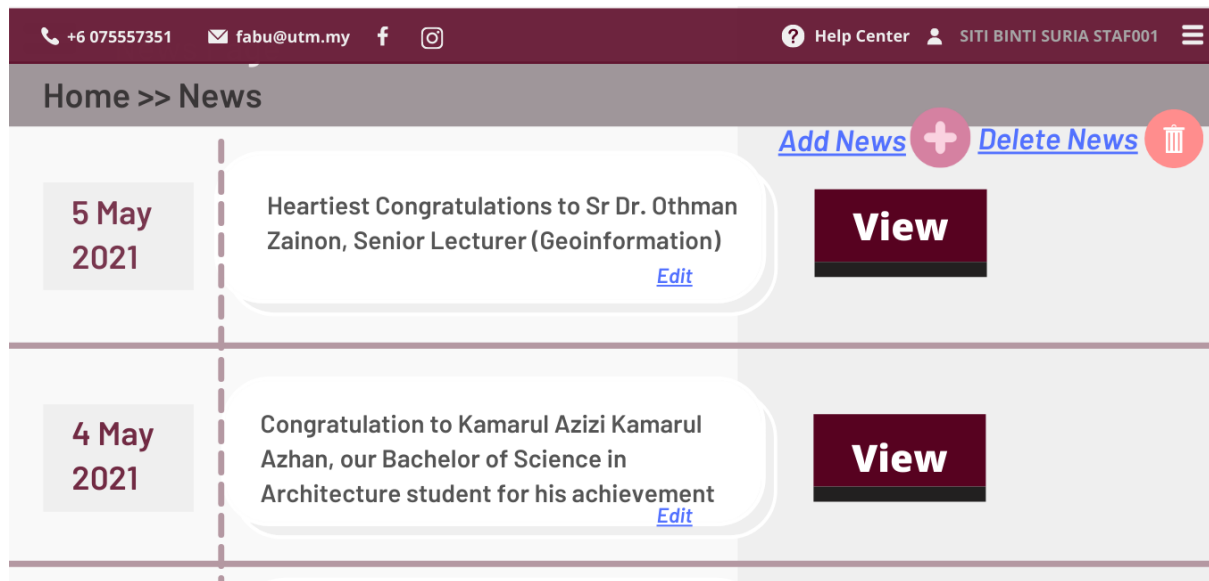
Screen Image 10: The report page for staff to add, edit and delete the report



Screen Image 11: The charity page for staff to add, edit and delete the report

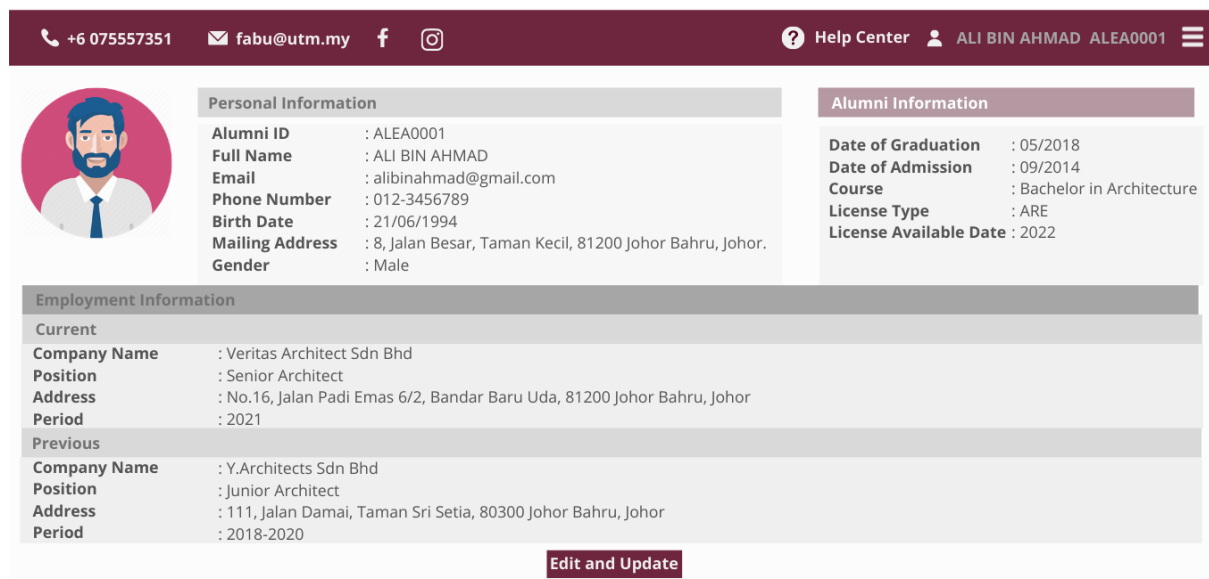


Screen Image 12: The event page for staff to add, edit and delete the event




Screen Image 13: The news page for staff to add, edit and delete the news

Besides, alumni users and staff also can view their user profile by clicking the profile at the top of the page to enter the profile page.



Screen Image 14: The alumni profile page for alumni user to view or edit and update their profile

+6 075557351
fabu@utm.my
f
i
? Help Center
SITI BINTI SURIA STAF001



Personal Information	
Staff ID	: STAF001
Full Name	: Siti Binti Suria
Email	: sitibintisuria@gmail.com
Phone Number	: 012-3456789
Position	: Senior Manager
Mailing Address	: 8, Jalan Besar, Taman Kecil, 81200 Johor Bahru, Johor.
Gender	: Female

Staff Information	
Department	: Academic Office
Faculty	: Faculty of Built Environment and Surveying
University	: Universiti Teknologi Malaysia
Start	: 01/2020
Tasks	: Manage the alumni system

Edit and Update

Screen Image 15: The profile page for staff to view or edit and update their profile

Furthermore, alumni users can update their architecture licensing by clicking Update Architecture Licensing from the main menu. After that, alumni user will get into the Update Architecture Licensing Page.

+6 075557351
fabu@utm.my
f
i
? Help Center
ALI BIN AHMAD ALEA0001

Home >> Update Architecture Licensing

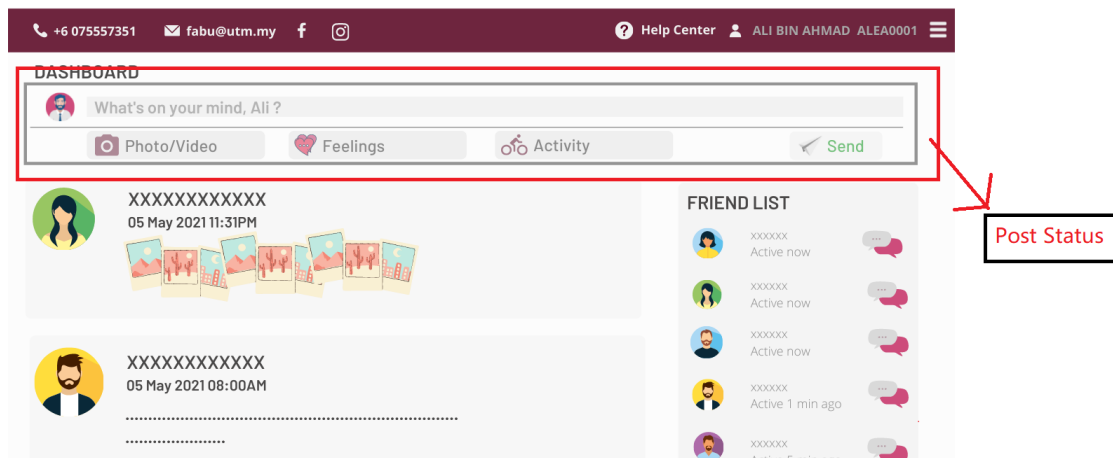
Architecture Licensing

Name:	xxxxxxxx
Type:	xxxxxxxx
Renewed Date:	xx/xx/xxxx
Expired Date:	xx/xx/xxxx

Update

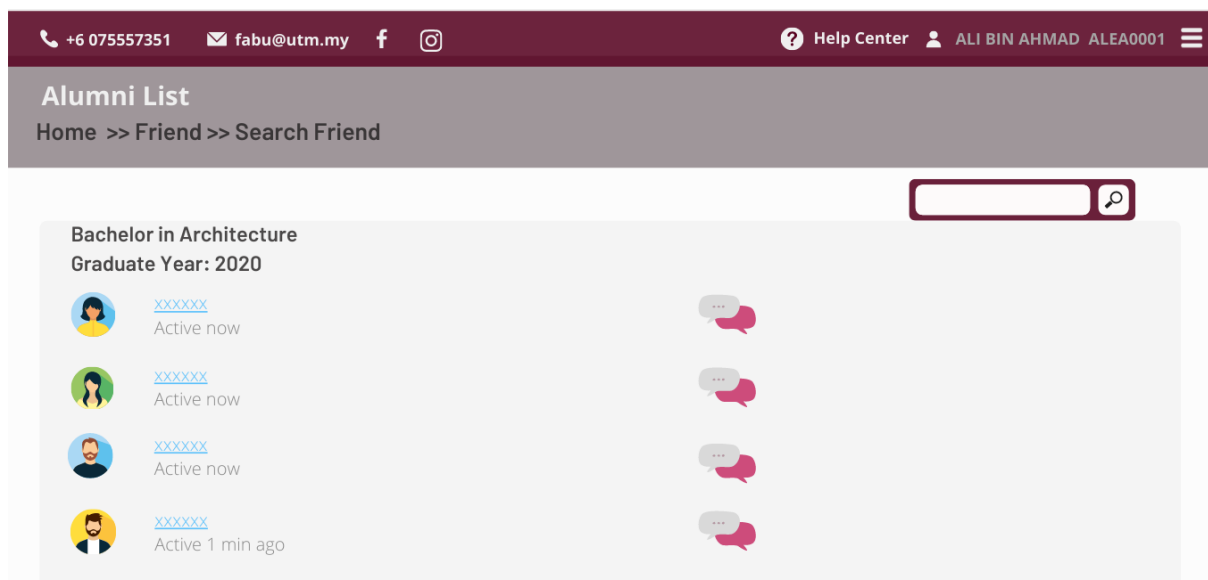
Screen Image 16: The update architecture licensing page for alumni user updated.

Besides, if an alumni user wants to post a status, they can go to the main menu by clicking the dashboard option to enter the dashboard page.



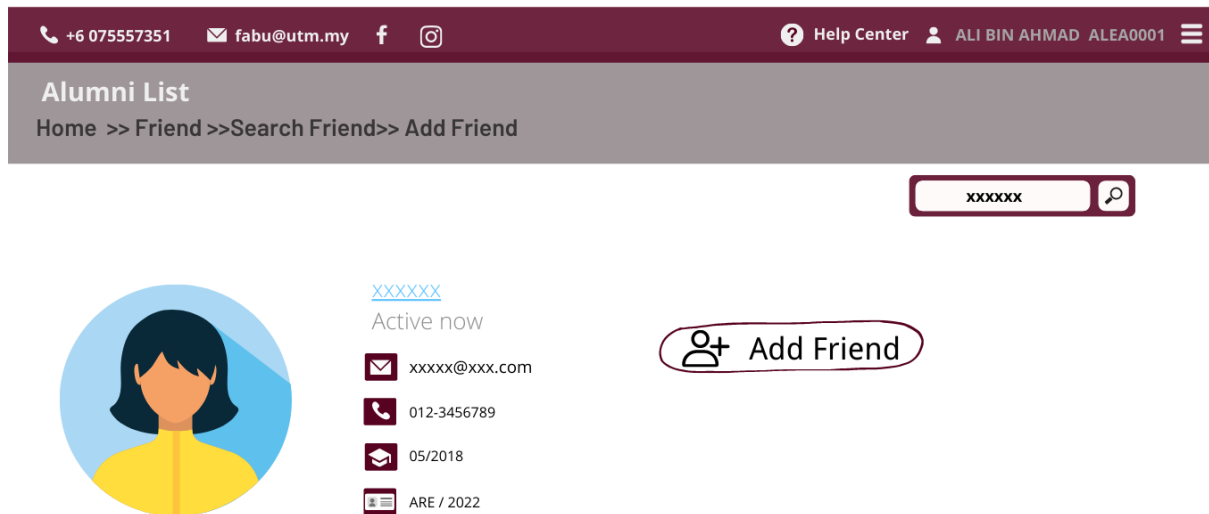
Screen Image 17: The dashboard page for alumni user to post a status

Moreover, alumni user also can search, add, delete friend by clicking the friend options in the main menu to enter the friend page and choose which option interested. If alumni user wants to search friend, they will then enter the search friend page. Alumni user click the search bar to search friend



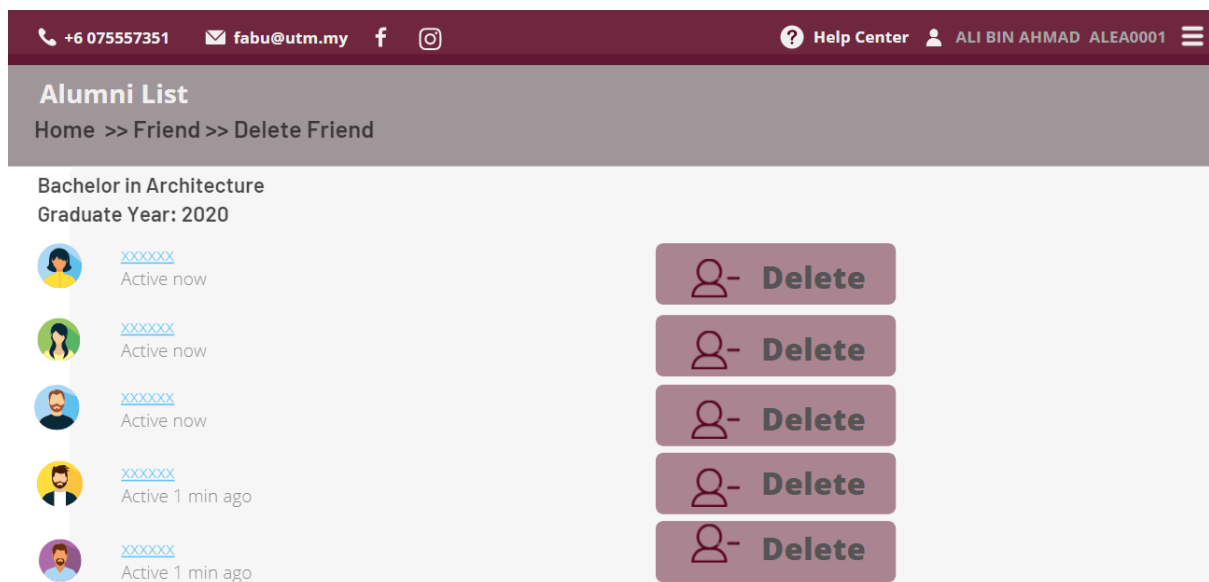
Screen Image 18: The search friend page for alumni user to search friend

However, if an alumni user wants to add a friend, they will then enter the add friend page. Alumni enter the friend they want.



Screen Image 19: The add friend page for alumni user to add friend

However, if an alumni user wants to delete a friend, they will then enter the delete friend page. Alumni can click delete to delete their friend



Screen Image 20: The delete friend page for alumni user to delete friend