SECP3204: Software Engineering WBL

System Design Descriptions (SDD)

UTM Faculty Of Computing Student Engagement System

Version 1.0

18/06/2023

School of Computing, Faculty of Engineering

Prepared by: Systema

Table of Contents

3. System Architectural Design

3.1 Architecture Style and Rationale

Architectural styles play a crucial role in software development as they provide a structured approach to designing and organizing complex systems. By employing the appropriate architectural style, such as the Model-View-Controller (MVC) model, developers can create scalable and maintainable modular software solutions. Although several architectural styles exist, this proposal will focus on the utilization of MVC in the student engagement system. The MVC architecture divides an application into three main components: model, view, and controller. This division is intended to separate concerns and promote code organization, thereby making the system easier to understand, develop, and maintain.

The model component serves as the foundation of the MVC architecture. It encapsulates the application's data and implements the business logic of the student engagement system. The model component is responsible for data access, operations, and verification. It applies business rules and validates user input. Various technologies or frameworks can be employed to realize the model component. For example, a relational database management system (RDBMS) can be used to store and manage student-related data. Object-Relational Mapping (ORM) frameworks can simplify the interaction between the application and the database by providing an abstraction layer.

The view component focuses on the presentation layer of the student engagement system. It is responsible for presenting data to users in a user-friendly manner and handling the user interface modules. To enhance the user experience, the view component should provide an intuitive and interactive interface. Web-based technologies like HTML, CSS and JavaScript can be used to simplify the implementation of the view component.Additionally, front-end frameworks such as Bootstrap and jQuery can facilitate development by offering reusable UI components.

The controller component acts as an intermediary between the model and the view. It receives user input, executes appropriate model actions, and updates the view accordingly. The controller component ensures the separation of the model and view, enabling independent development and testing. Programming languages like JavaScript can be employed to implement the controller component.

The decision to utilize the MVC framework in the student engagement system was driven by several factors. Firstly, the separation of concerns in MVC allows each component to be developed and tested independently, enhancing code maintainability and facilitating future system enhancements or updates. Secondly, MVC encourages code organization and reusability. Adhering to the MVC pattern enables developers to organize their codebase into distinct components, making it easier to navigate and comprehend. This promotes collaboration among team members and reduces development time. Lastly, the MVC architecture aligns with the needs and goals of the student engagement system. The model component handles data access, operations, and verification, ensuring the accuracy and reliability of student-related information. The view component focuses on creating an appealing and user-friendly interface, while the controller component manages user input and updates the system accordingly.

In summary, employing the MVC architecture in the student engagement system offers numerous advantages. The separation of concerns, code organization, and modularity inherent in MVC contribute to the system's scalability, maintainability, and extensibility. By implementing the model, view, and controller components, developers can meet the project's requirements and create a robust and user-friendly student engagement system. The architectural design stage plays a critical role in the software development process, laying the groundwork for a successful implementation. By carefully considering the use of the MVC framework, developers can ensure a well-structured and effective system.

3.2 Component Model

A component model refers to a framework or methodology used in software engineering to design and develop software systems by decomposing them into modular and reusable components. A component is a self-contained unit of software that encapsulates a set of related functionalities or services and can be independently deployed and replaced without affecting the overall system.

The component model defines the rules, guidelines, and standards for constructing and integrating components within a software system. It specifies how components should be designed, implemented, packaged, deployed, and interact with each other and the underlying infrastructure.

The following figures refer to the component diagram of UTM Faculty of Computing Student Engagement System:

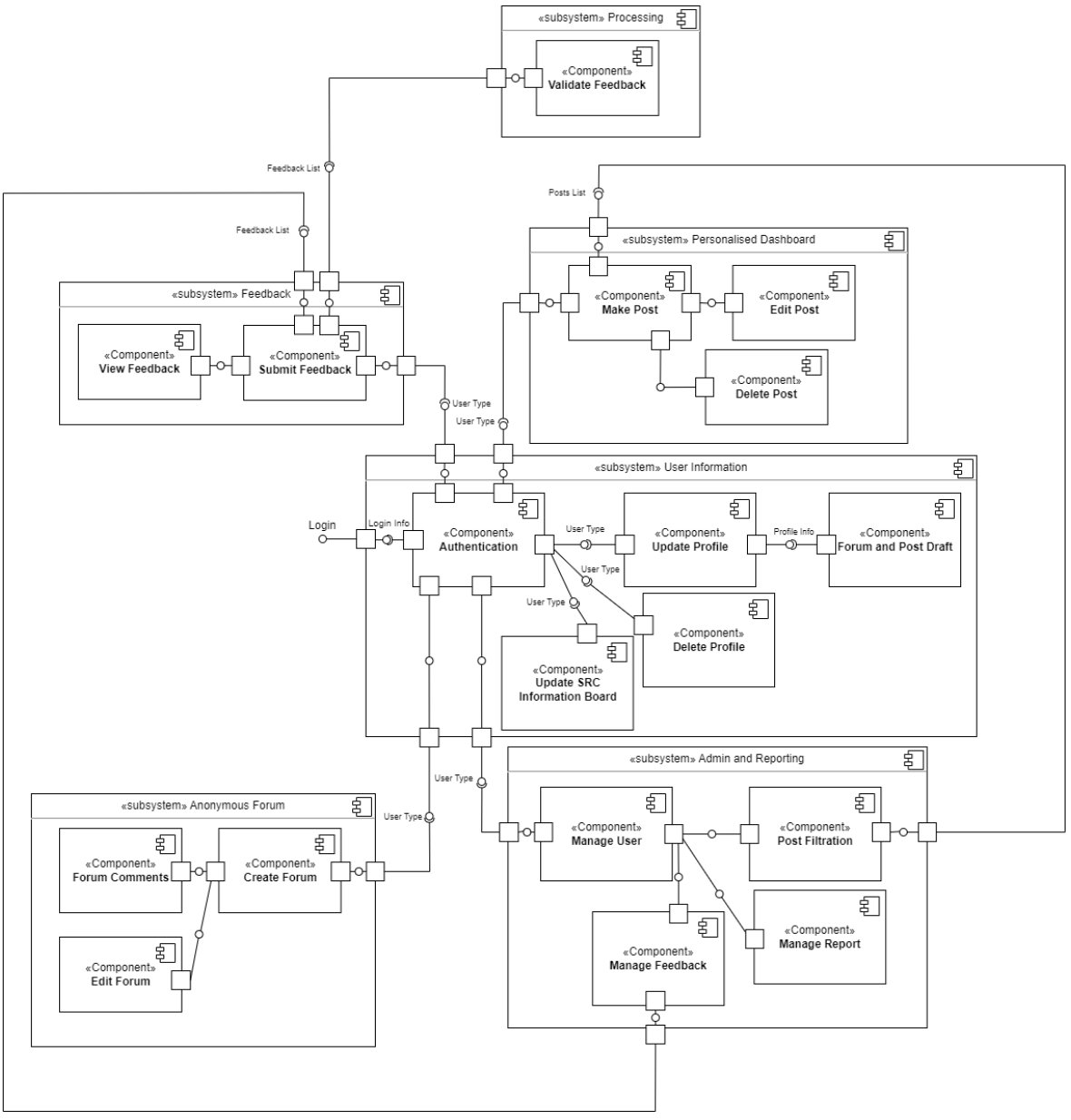


Figure 3.1: Component Diagram of <UTM Faculty of Computing Student Engagement System>

Our component model consists of six subsystems which each contain their respective components to perform different functions. The six subsystems are “User Information”, “Processing”, “Feedback”, “Personalized Dashboard”, “Admin and Reporting” and “Anonymous Forum”.

First and foremost, the “User Information” subsystem. It consists of five components such as “Authentication”, “Update Profile”, “Delete Profile”, “Update SRC Information Board” and “Forum and Post Draft”. Each time users login into our system, they will be led to the “Authentication”. This component enables users to sign up or login their account based on the user type such as admin, member of Student Representative Council(SRC), student or lecturer. During signing up their account, they might need an email address to be used as verification. Besides, if they forgot their password, they are able to reset their password in this component. For the “Update Profile” and “Update SRC Information Board” component, users are able to edit their profile based on their latest information and user type. Only members of SRC are able to update their information board while other user types cannot. On the other hand, if the users wish to delete their profile, they are able to do it based on the “Delete Profile” component.

Apart from that, for the “Processing” subsystem, it consists of a component only which is “Validate Feedback”. The MVC architecture’s controller is in charge of processing feedback data. It accepts feedback information, filters, validates and communicates with the model component to store and retrieve feedback information.

A “Feedback” subsystem consists of three components which are “View Feedback” and “Submit Feedback”. The MVC architecture’s model component will be in charge of analyzing the type of users in order to determine who is able to submit and view feedback. After users submit their feedback, they are able to view it.

In addition, “Make Post”, “Edit Post” and “Delete Post” are part of components of the “Personalized Dashboard” subsystem. Users are able to create posts, edit them and also delete unwanted posts in this subsystem. The controller is in charge of analyzing the type of users in order to perform the functions of this subsystem.

Coming to the “Admin and Reporting” subsystem, it consists of four components such as “Manage Report”, “Manage User”, “Manage Feedback” and “Post Filtration”. Same goes with the two subsystems above, the MVC architecture’s model component in charge of analyzing the type of users to determine who can access each of the components in this subsystem. However, there exists a difference with the two subsystems above which is that in this subsystem, there is one more responsibility of the controller which is it will analyze, categorize and arrange the reports as well as feedback and posts in a clear and easy access view to the users as well as communicates with the model component to retrieve the required data. For “Manage Feedback”, a specific type of user will be responsible to filter the feedback based on their priority level such as light, moderate and heavy while for post filtration, they will filter the post in order to create a user friendly and green environment of the system.

Last but not least, the “Anonymous Forum”. It consists of three components such as “Create Forum”, “Forum Comments” and “Edit Forum”. In this subsystem, users are allowed to create a forum based on what they wanted to share, comments on others’ forum and edit forum posts by themselves. All the actions done in this subsystem are all anonymous so that users may feel free to post and comment about what they want as nobody might know who they are.

4. Detailed Description of Components

4.1 Complete Package Diagram

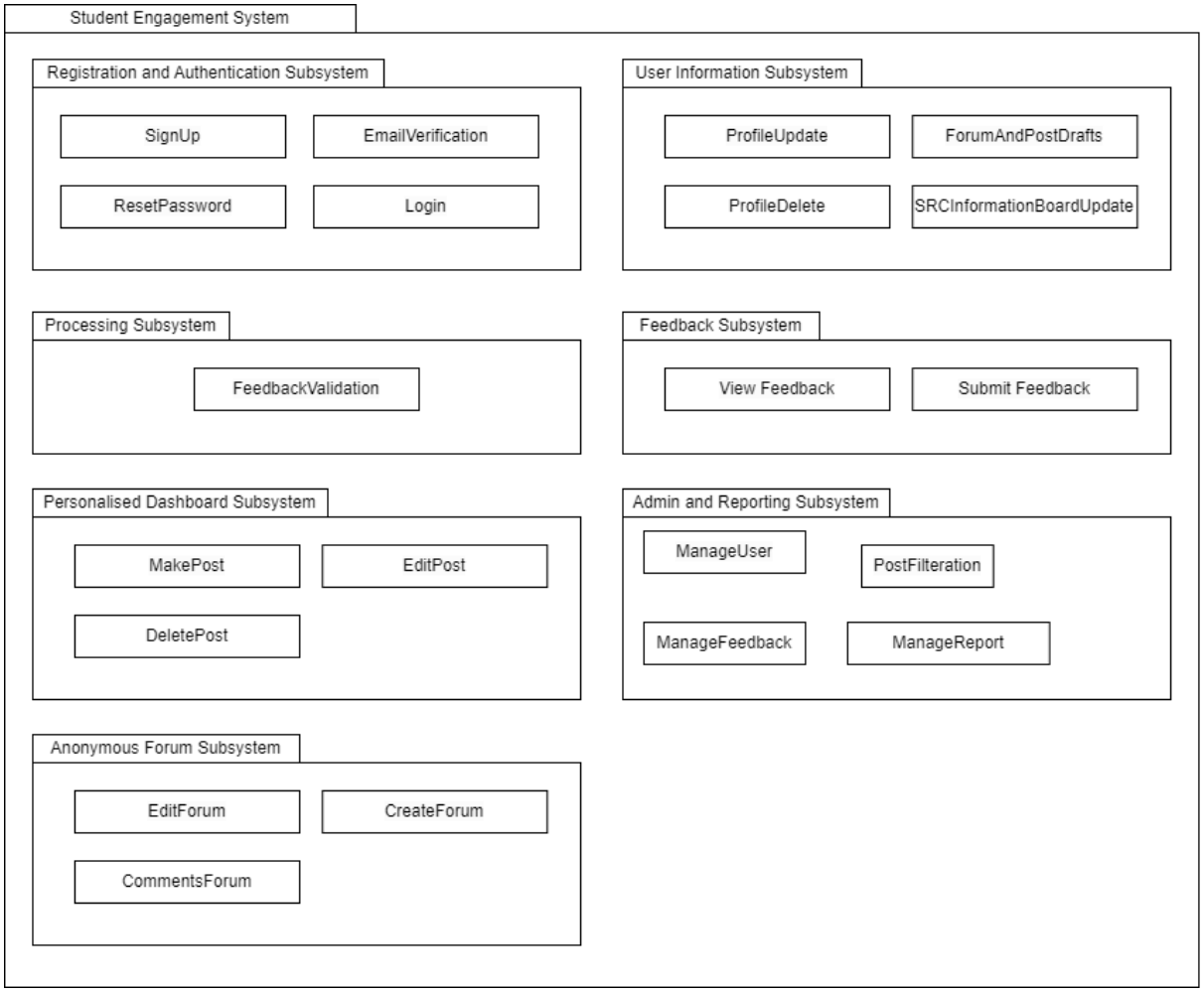


Figure 4.1: Package Diagram for <UTM Faculty of Computing Student Engagement System>

4.2 Detailed Description

4.2.1.1 P001: <Registration and Authentication> Subsystem

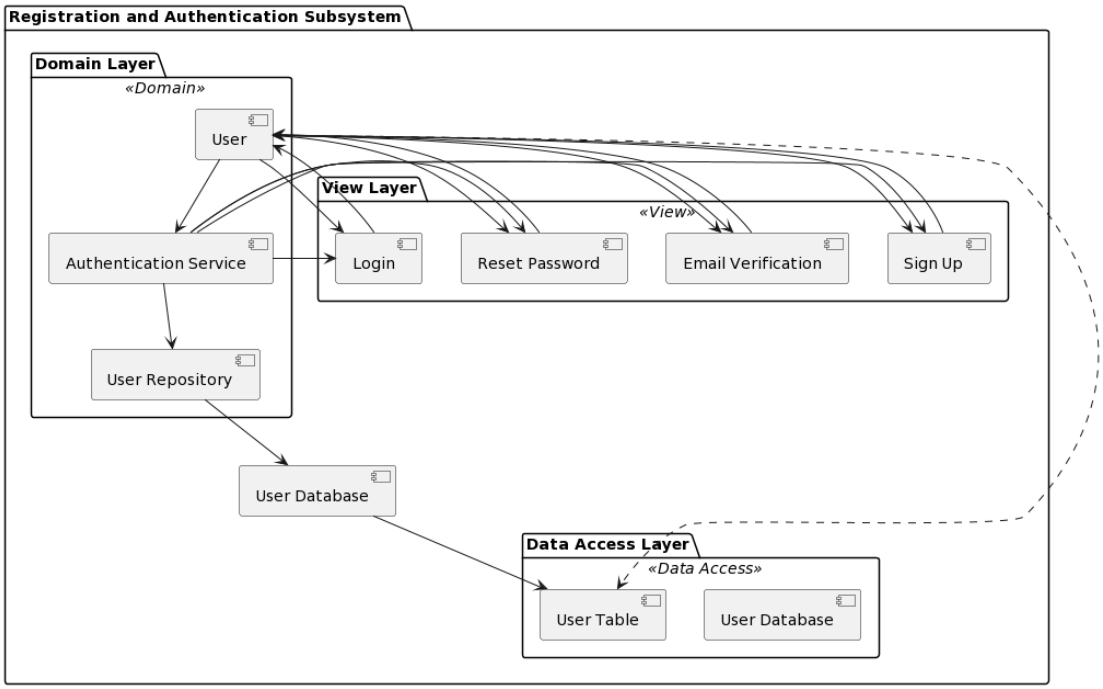


Figure 4.2.1: Package Diagram for <Registration and Authentication> Subsystem

4.2.1.2 P002: <User Information> Subsystem

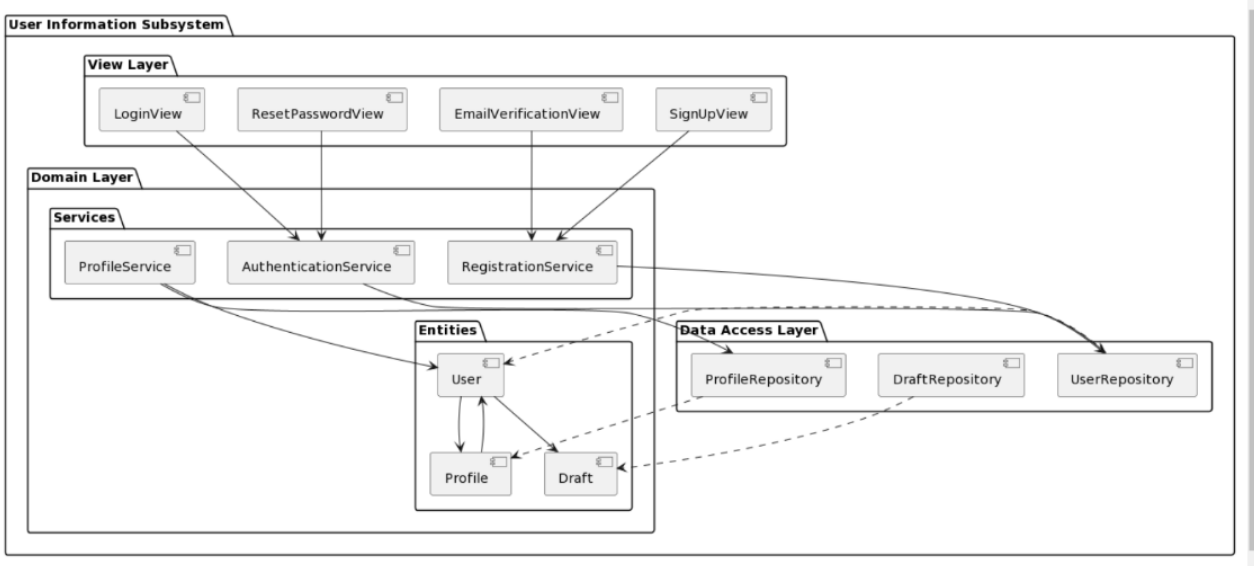


Figure 4.2.2: Package Diagram for <User Information> Subsystem

4.2.1.3 P003: <Processing> Subsystem

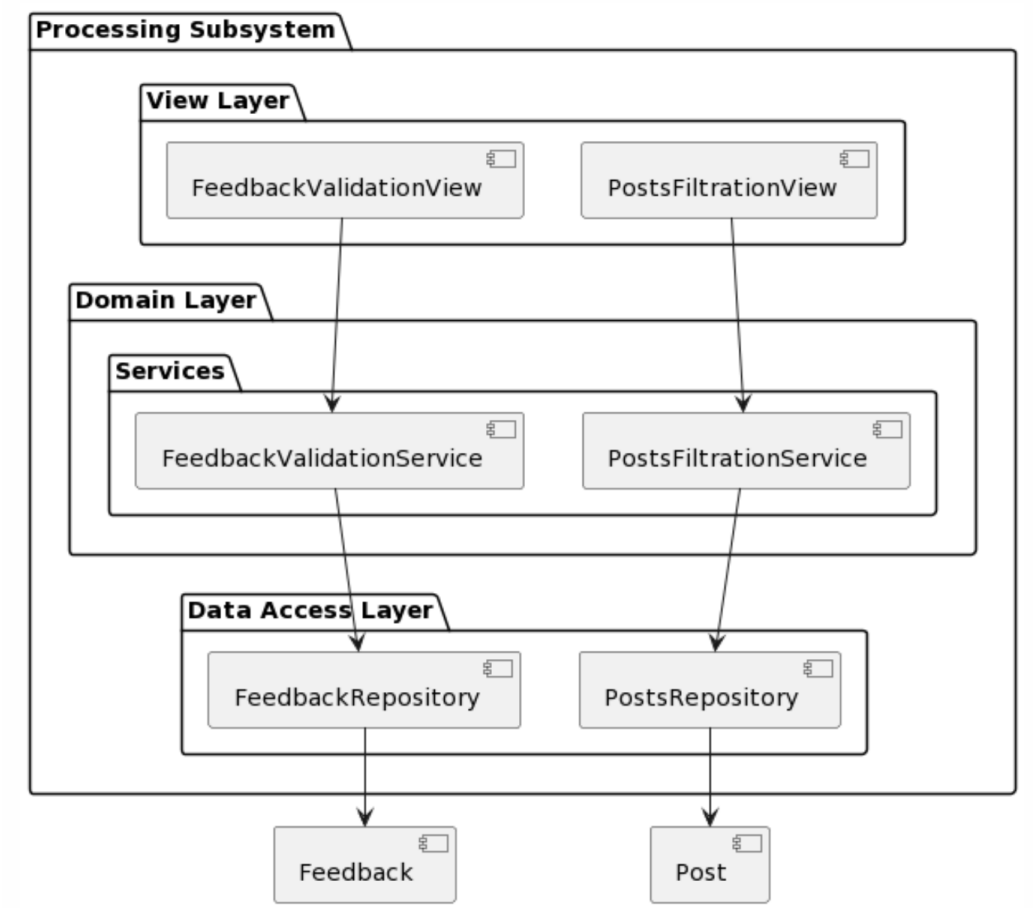


Figure 4.2.3: Package Diagram for <Processing> Subsystem

4.2.1.4 P004: <Admin and Reporting> Subsystem

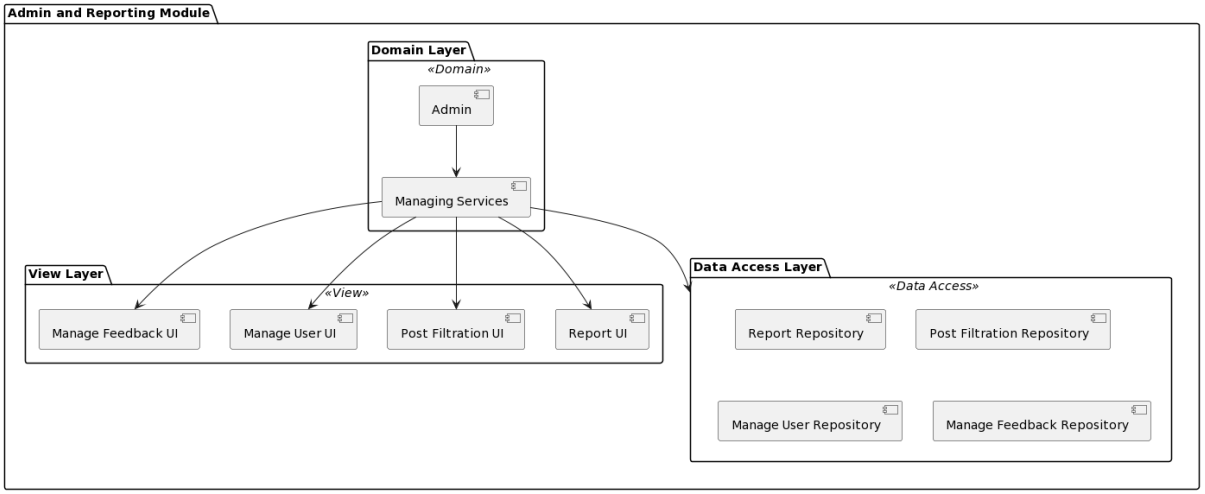


Figure 4.2.4: Package Diagram for <Admin and Reporting> Subsystem

4.2.1.5 P005: <Personalized Dashboard> Subsystem

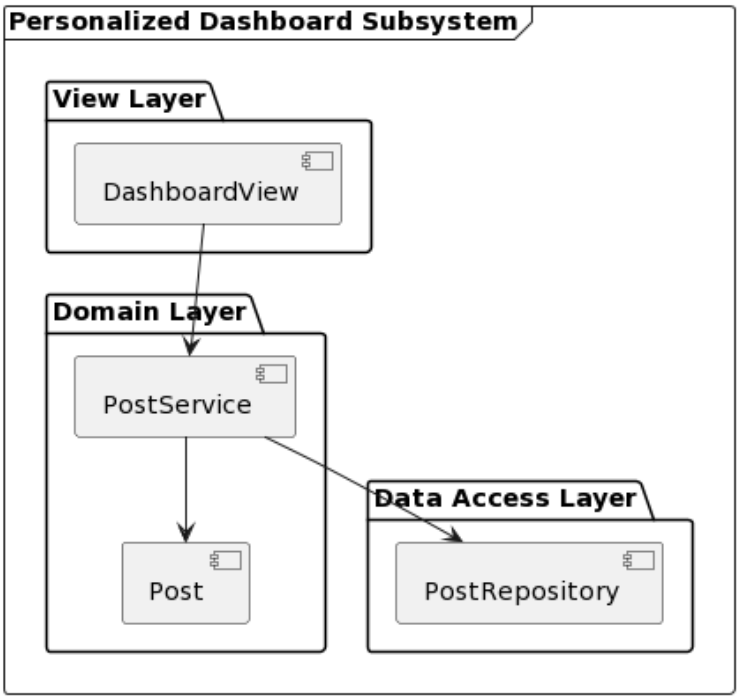


Figure 4.2.5: Package Diagram for <Personalized Dashboard> Subsystem

4.2.1.6 P006: <Feedback> Subsystem

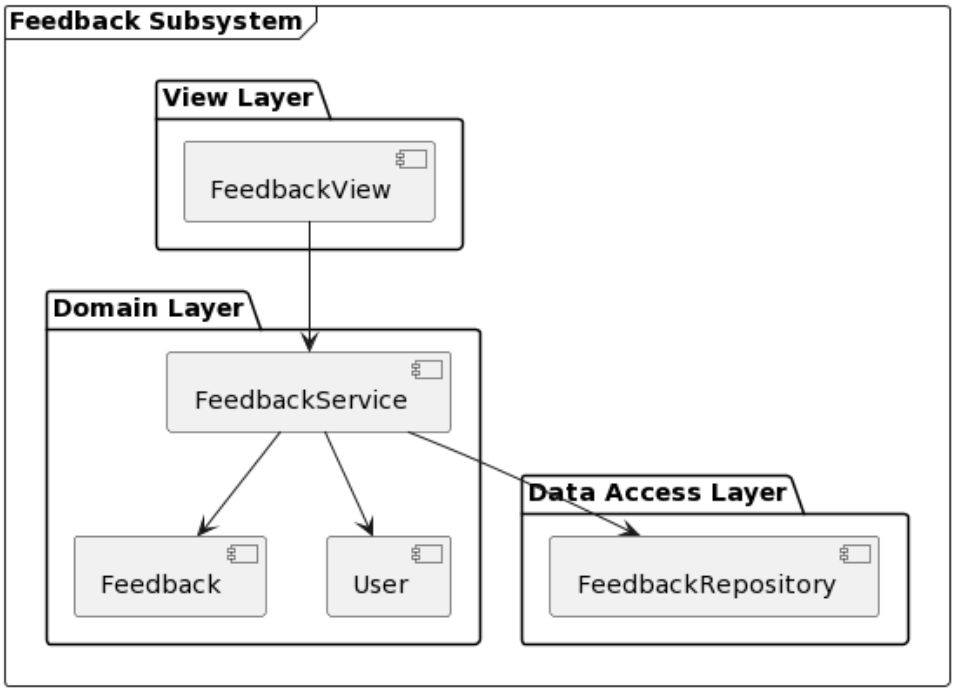


Figure 4.2.6: Package Diagram for <Feedback> Subsystem

4.2.1.7 P007: <Anonymous Forum> Subsystem

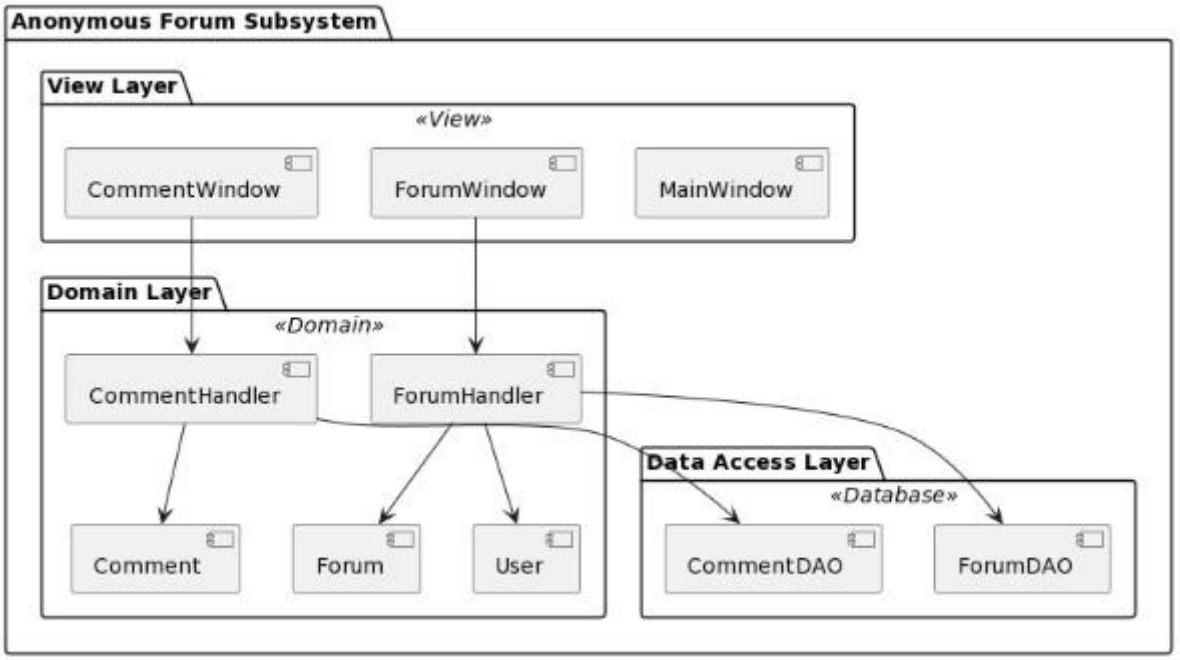
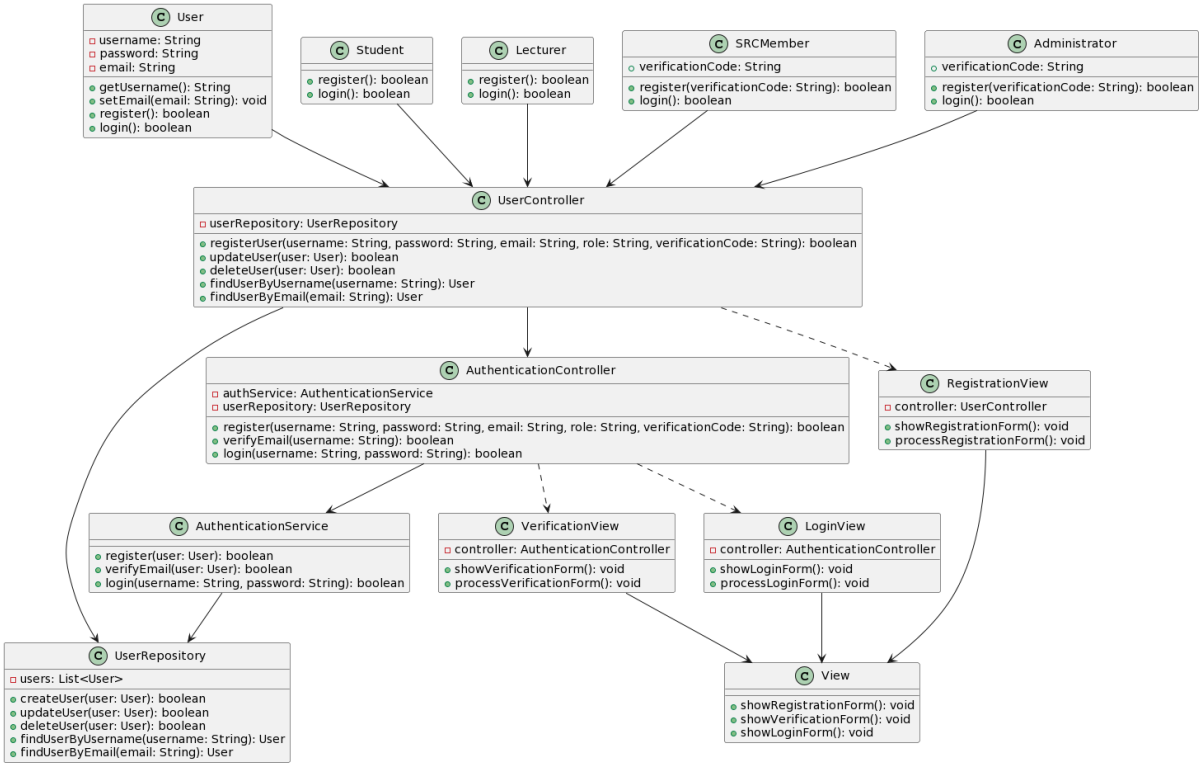


Figure 4.2.7: Package Diagram for <Anonymous Forum> Subsystem

4.2.2 Class Diagram

Figure 4.2.2.1: Class Diagram for <Registration and Authentication> Subsystem



Entity Name	User
Method Name	getUsername
Input	
Output	String (username)
Algorithm	1. Start 2. Return the value of the username attribute 3. End

Entity Name	User
Method Name	setEmail
Input	email
Output	void
Algorithm	1. Start 2. Set the email attribute to the provided value 3. End

Entity Name	User
Method Name	register
Input	
Output	boolean (indicating success or failure)
Algorithm	1. Start 2. Add the user to the UserRepository using the createUser method 3. Return true if successful, false otherwise 4. End

Entity Name	User
Method Name	login
Input	
Output	boolean (indicating success or failure)
Algorithm	1. Start 2. Retrieve the user from the UserRepository based on the provided username 3. If the user is found and the provided password matches the stored password, return true 4. Otherwise, return false 5. End

Entity Name	Student
Method Name	register
Input	
Output	boolean (indicating success or failure)
Algorithm	1. Start 2. Call the register method of the User superclass 3. Return the result of the register method call 4. End

Entity Name	Student
Method Name	login
Input	
Output	boolean (indicating success or failure)
Algorithm	1. Start 2. Call the login method of the User superclass

	3. Return the result of the login method call 4. End

Entity Name	Lecturer
Method Name	register
Input	
Output	boolean (indicating success or failure)
Algorithm	1. Start 2. Call the register method of the User superclass 3. Return the result of the register method call 4. End

Entity Name	Lecturer
Method Name	login
Input	
Output	boolean (indicating success or failure)
Algorithm	1. Start 2. Call the login method of the User superclass 3. Return the result of the login method call 4. End

Entity Name	SRCMember
Method Name	register
Input	verificationCode
Output	boolean (indicating success or failure)
Algorithm	1. Start 2. Add the verificationCode to the SRCMember object 3. Call the register method of the User superclass 4. Return the result of the register method call 5. End

Entity Name	SRCMember
Method Name	login
Input	
Output	boolean (indicating success or failure)

Algorithm	1. Start 2. Call the login method of the User superclass 3. Return the result of the login method call 4. End

Entity Name	Administrator
Method Name	register
Input	verificationCode
Output	boolean (indicating success or failure)
Algorithm	1. Start 2. Add the verificationCode to the Administrator object 3. Call the register method of the User superclass 4. Return the result of the register method call 5. End

Entity Name	Administrator
Method Name	login
Input	
Output	boolean (indicating success or failure)
Algorithm	1. Start 2. Call the login method of the User superclass 3. Return the result of the login method call 4. End

Entity Name	UserRepository
Method Name	createUser
Input	user
Output	boolean (indicating success or failure)
Algorithm	1. Start 2. Check if the user's username or email already exists in the UserRepository 3. If the username or email is not already taken, add the user to the UserRepository 4. Return true if successful, false otherwise 5. End

Entity Name	UserRepository

Method Name	updateUser
Input	user
Output	boolean (indicating success or failure)
Algorithm	1. Start 2. Retrieve the user from the UserRepository based on the provided username 3. If the user is found, update the user's attributes with the provided user object 4. Return true if successful, false otherwise 5. End

Entity Name	UserRepository
Method Name	deleteUser
Input	user
Output	boolean (indicating success or failure)
Algorithm	1. Start 2. Remove the user from the UserRepository based on the provided user object 3. Return true if successful, false otherwise 4. End

Entity Name	UserRepository
Method Name	findUserByUsername
Input	username
Output	User
Algorithm	1. Start 2. Search the UserRepository for a user with the provided username 3. Return the found user or null if not found 4. End

Entity Name	UserRepository
Method Name	findUserByEmail
Input	email
Output	User
Algorithm	1. Start

	2. Search the UserRepository for a user with the provided email 3. Return the found user or null if not found 4. End

Entity Name	UserController
Method Name	registerUser
Input	username, password, email, role, verificationCode
Output	boolean (indicating success or failure)
Algorithm	1. Start 2. Create a new User object with the provided username, password, and email 3. Set the role attribute based on the provided role 4. If the role is SRCMember or Administrator, pass the verificationCode to the register method of the respective class 5. Otherwise, call the register method of the User object 6. Return the result of the register method call 7. End

Entity Name	UserController
Method Name	updateUser
Input	user
Output	boolean (indicating success or failure)
Algorithm	1. Start 2. Call the updateUser method of the UserRepository with the provided user object 3. Return the result of the updateUser method call 4. End

Entity Name	UserController
Method Name	deleteUser
Input	user
Output	boolean (indicating success or failure)
Algorithm	1. Start 2. Call the deleteUser method of the UserRepository with the provided user object 3. Return the result of the deleteUser method call

	4. End

Entity Name	UserController
Method Name	findUserByUsername
Input	username
Output	User
Algorithm	1. Start 2. Call the findUserByUsername method of the UserRepository with the provided username 3. Return the found user or null if not found 4. End

Entity Name	UserController
Method Name	findUserByEmail
Input	email
Output	User
Algorithm	1. Start 2. Call the findUserByEmail method of the UserRepository with the provided email 3. Return the found user or null if not found 4. End

Entity Name	AuthenticationService
Method Name	register
Input	user
Output	boolean (indicating success or failure)
Algorithm	1. Start 2. Call the register method of the UserService with the provided user object 3. Return the result of the register method call 4. End

Entity Name	AuthenticationService
Method Name	verifyEmail
Input	user
Output	boolean (indicating success or failure)

Algorithm	1. Start 2. Call the verifyEmail method of the UserService with the provided user object 3. Return the result of the verifyEmail method call 4. End

Entity Name	AuthenticationService
Method Name	login
Input	username, password
Output	boolean (indicating success or failure)
Algorithm	1. Start 2. Call the login method of the UserService with the provided username and password 3. Return the result of the login method call 4. End

Entity Name	AuthenticationController
Method Name	register
Input	username, password, email, role, verificationCode
Output	boolean (indicating success or failure)
Algorithm	1. Start 2. Call the registerUser method of the UserController with the provided parameters 3. Return the result of the registerUser method call 4. End

Entity Name	AuthenticationController
Method Name	verifyEmail
Input	username
Output	boolean (indicating success or failure)
Algorithm	1. Start 2. Call the verifyEmail method of the AuthenticationController with the provided username 3. Return the result of the verifyEmail method call 4. End

Entity Name	AuthenticationController
Method Name	login
Input	username, password
Output	boolean (indicating success or failure)
Algorithm	1. Start 2. Call the login method of the AuthenticationController with the provided username and password 3. Return the result of the login method call 4. End

Entity Name	RegistrationView
Method Name	showRegistrationForm
Input	
Output	void
Algorithm	1. Start 2. Display the registration form to the user 3. End

Entity Name	RegistrationView
Method Name	processRegistrationForm
Input	
Output	void
Algorithm	1. Start 2. Collect the user's input from the registration form 3. Call the register method of the AuthenticationController with the collected data 4. Display a success message if the registration is successful, or an error message otherwise 5. End

Entity Name	VerificationView
Method Name	showVerificationForm
Input	
Output	void
Algorithm	1. Start

	2. Display the verification form to the user 3. End

Entity Name	VerificationView
Method Name	processVerificationForm
Input	
Output	void
Algorithm	1. Start 2. Collect the user's input from the verification form 3. Call the verifyEmail method of the AuthenticationController with the collected data 4. Display a success message if the verification is successful, or an error message otherwise 5. End

Entity Name	LoginView
Method Name	showLoginForm
Input	
Output	void
Algorithm	1. Start 2. Display the login form to the user 3. End

Entity Name	LoginView
Method Name	processLoginForm
Input	
Output	void
Algorithm	1. Start 2. Collect the user's input from the login form 3. Call the login method of the AuthenticationController with the collected data 4. Display a success message if the login is successful, or an error message otherwise 5. End

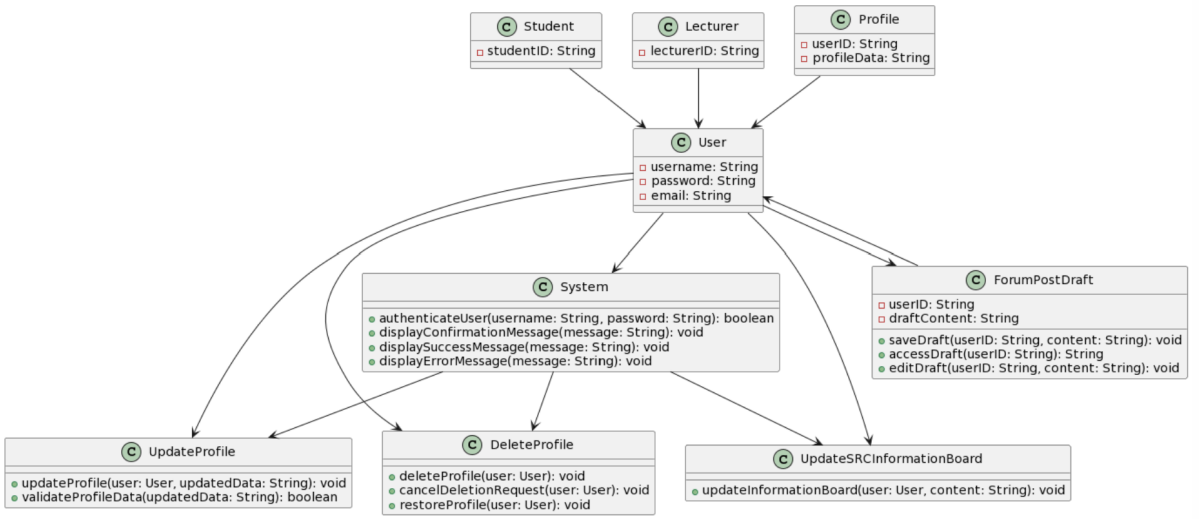
Entity Name	View

Method Name	showRegistrationForm
Input	
Output	void
Algorithm	1. Start 2. Display the registration form to the user 3. End

Entity Name	View
Method Name	showVerificationForm
Input	
Output	void
Algorithm	1. Start 2. Display the verification form to the user 3. End

Entity Name	View
Method Name	showLoginForm
Input	
Output	void
Algorithm	1. Start 2. Display the login form to the user 3. End

Figure 4.2.2.2: Class Diagram for <User Information> Subsystem



Entity Name	User
Method Name	authenticate
Input	username, password
Output	boolean(success or failure)
Algorithm	1. Start 2. Read or input username and password 3. Call the authenticateUser method of System class 4. Return the authentication result 5. End

Entity Name	Delete Profile
Method Name	confirmProfileDeletion
Input	
Output	boolean(confirmation result)
Algorithm	1. Start 2. Display a confirmation message to the user 3. Read user's confirmation input 4. Return the confirmation result as a boolean value 5. End

Entity Name	Delete Profile
Method Name	deleteProfile

Input	userID
Output	boolean(success or failure delete result)
Algorithm	1. Start 2. Read or input userID 3. Remove the user's profile and associated data from the system 4. Return the deletion result 5. End

Entity Name	SaveDraft
Method Name	confirmCancel
Input	
Output	boolean(confirmation result)
Algorithm	1. Start 2. Display a confirmation message to the user 3. Read user's confirmation input 4. Return the confirmation result as a boolean value 5. End

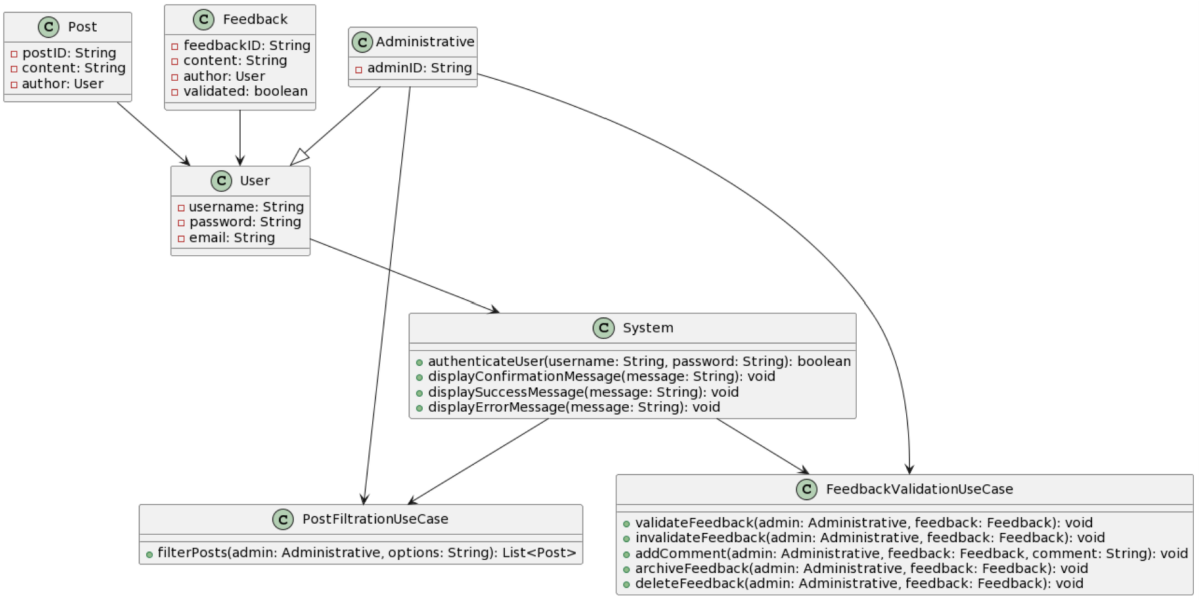
Entity Name	SaveDraft
Method Name	SaveAsDraft
Input	draftContent
Output	boolean(saving result)
Algorithm	1. Start 2. Read or input draftContent 3. Save the draft content to the user's draft box 4. Return the saving result 5. End

Entity Name	UpdateInfoBoard
Method Name	presentContent
Input	
Output	Current content of info board
Algorithm	1. Start 2. Retrieve the current content of the info board 3. Return the current content to the user 4. End

Entity Name	UpdateInfoBoard
Method Name	editContent
Input	newContent
Output	
Algorithm	1. Start 2. Read or input newContent 3. Update the content of the info board with the new content 4. End

Entity Name	UpdateInfoBoard
Method Name	saveChanges
Input	
Output	boolean(saving result)
Algorithm	1. Start 2. Save the updated content of the info board 3. Return the saving result 4. End

Figure 4.2.2.3: Class Diagram for <Processing> Subsystem



Entity Name	AdministrativeController
Method Name	selectPostFiltrationOption
Input	
Output	user ID
Algorithm	1. Start 2. Present the available post filtration options 3. Wait for the user to select an option 4. End

Entity Name	AdministrativeController
Method Name	selectFilteringOptions
Input	
Output	user ID
Algorithm	1. Start 2. Present the available filtering options 3. Wait for the user to select an option 4. End

Entity Name	AdministrativeController
Method Name	displayFilteredPosts
Input	

Output	filteredPosts
Algorithm	1. Start 2. Retrieve the filtered posts 3. Display the filtered posts 4. End

Entity Name	AdministrativeController
Method Name	validateFeedback
Input	FeedbackID, validation
Output	
Algorithm	1. Start 2. Validate the feedback based on the validate 3. End

Entity Name	AdministrativeController
Method Name	reviewFeedback
Input	feedbackID
Output	feedbackDetails
Algorithm	1. Start 2. Retrieve the details of the feedback with the given feedbackID 3. Returns the feedback details 4. End

Entity Name	AdministrativeController
Method Name	addComments
Input	feedbackID, comment
Output	
Algorithm	1. Start 2. Add comment to the feedback with the given feedbackID 3. End

Entity Name	AdministrativeController
Method Name	processValidatedFeedback

Input	feedbackID
Output	
Algorithm	1. Start 2. Process the validated feedback based on the provided feedbackID 3. End

Entity Name	Administrative
Method Name	presentFilteringOptions
Input	
Output	
Algorithm	1. Start 2. Present the available filtering options 3. End

Entity Name	Administrative
Method Name	markPostValidated
Input	postID
Output	
Algorithm	1. Start 2. Mark the post with the given postID as validated 3. End

Entity Name	Administrative
Method Name	markPostInvalidated
Input	postID
Output	
Algorithm	1. Start 2. Mark the post with the given postID as invalidated 3. End

Entity Name	Administrative
Method Name	notifyUser
Input	userID

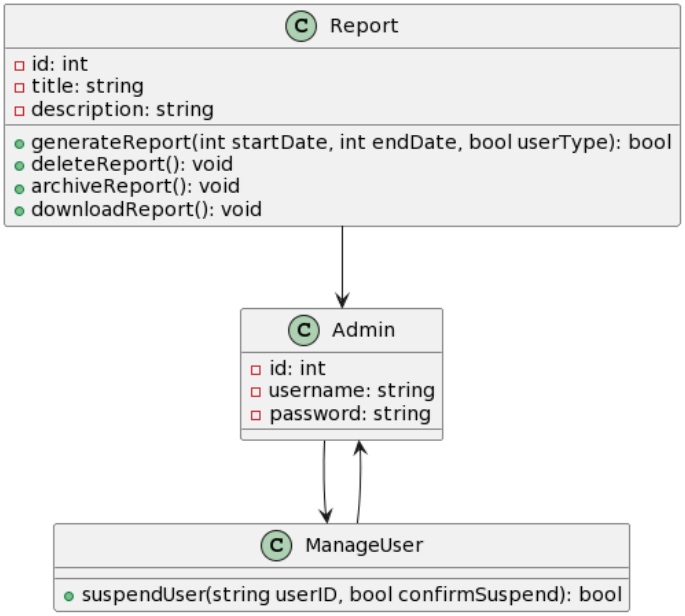
Output	
Algorithm	1. Start 2. Notify the user with the given userID 3. End

Entity Name	Feedback
Method Name	validate
Input	feedbackID
Output	
Algorithm	1. Start 2. Validate the feedback with the given feedbackID 3. End

Entity Name	Feedback
Method Name	addComments
Input	feedbackID, comment
Output	
Algorithm	1. Start 2. Add the comment to the feedback with the given feedbackID 3. End

Entity Name	Post
Method Name	selectedFilters
Input	
Output	
Algorithm	1. Start 2. Apply the selected filters to the post 3. End

Figure 4.2.2.4: Class Diagram for <Admin and Reporting> Subsystem



Entity Name	Report
Method Name	generateReport
Input	startDate, endDate, userType
Output	boolean
Algorithm	1. Start 2. Retrieve the data based on the provided input 3. Generate a list of report 4. Return true if successfully generated report, else false 5. End

Entity Name	Report
Method Name	deleteReport
Input	reportID, delete
Output	boolean
Algorithm	1. Start 2. Retrieve the report based on the provided reportID 3. Delete report 4. Return true if successfully deleted report, else return false 5. End

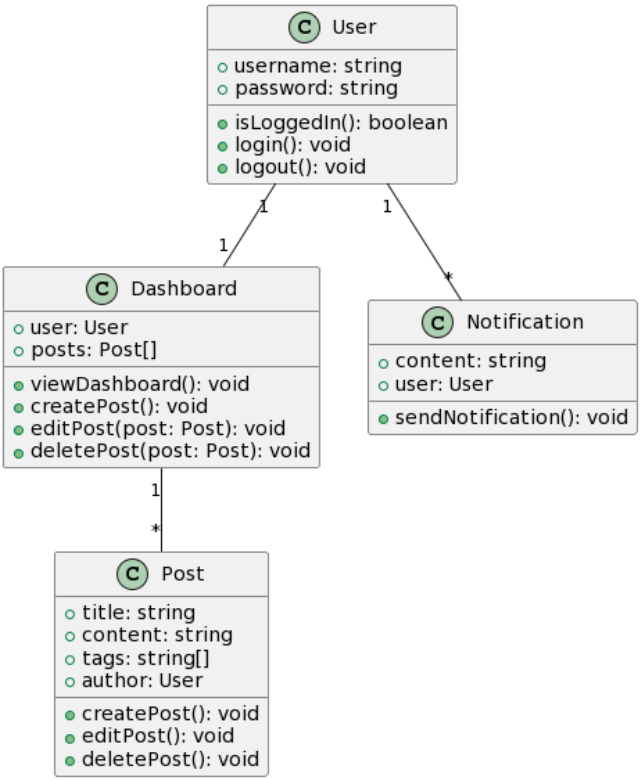
Entity Name	Report
Method Name	archiveReport
Input	reportID, archive

Output	boolean
Algorithm	1. Start 2. Retrieve the report based on the provided reportID 3. Archive report 4. Return true if successfully archived report, else return false 5. End

Entity Name	Report
Method Name	downloadReport
Input	reportID, download
Output	boolean
Algorithm	1. Start 2. Retrieve the report based on the provided reportID 3. Download report 4. Return true if successfully downloaded report, else return false 5. End

Entity Name	ManageUser
Method Name	suspendUser
Input	userID, confirmSuspend
Output	boolean
Algorithm	1. Start 2. Retrieve the user based on the provided userID 3. Return true if confirmSuspend is true, else return false 4. End

Figure 4.2.2.5 Class Diagram for <Personalized Dashboard> Subsystem



Entity Name	User
Method Name	isLoggedIn
Input	-
Output	boolean
Algorithm	1. Start 2. Return true if the user is currently logged in, else false 3. End

Entity Name	User
Method Name	login
Input	-
Output	void
Algorithm	1. Start 2. Validate the user's credentials(username and password) 3. If the credentials are valid, create a session for the user and set the login status to true 4. If the credentials are not valid, display an error message

	5. End

Entity Name	User
Method Name	logout
Input	-
Output	void
Algorithm	1. Start 2. Destroy the user's session and set the login status to false 3. Redirect the user to the login page or a logged-out state 4. End

Entity Name	Post
Method Name	createPost
Input	-
Output	void
Algorithm	1. Start 2. Retrieve the input values for the post(title, content and tags) 3. Create a new instance of the Post class with the input values 4. Set the author of the post to the current user 5. Add the new post to the posts list in the dashboard 6. End

Entity Name	Post
Method Name	editPost
Input	-
Output	void
Algorithm	1. Start 2. Retrieve the input values for editing the post(updated content) 3. Update the content of the selected post with the new input values 4. End

Entity Name	Post
Method Name	deletePost
Input	-
Output	void
Algorithm	1. Start 2. Remove the selected post from the posts list in the dashboard 3. End

Entity Name	Dashboard
Method Name	viewDashboard
Input	-
Output	void
Algorithm	1. Start 2. Fetch the user's posts from the database or storage 3. Display the user's posts in the personalized dashboard 4. End

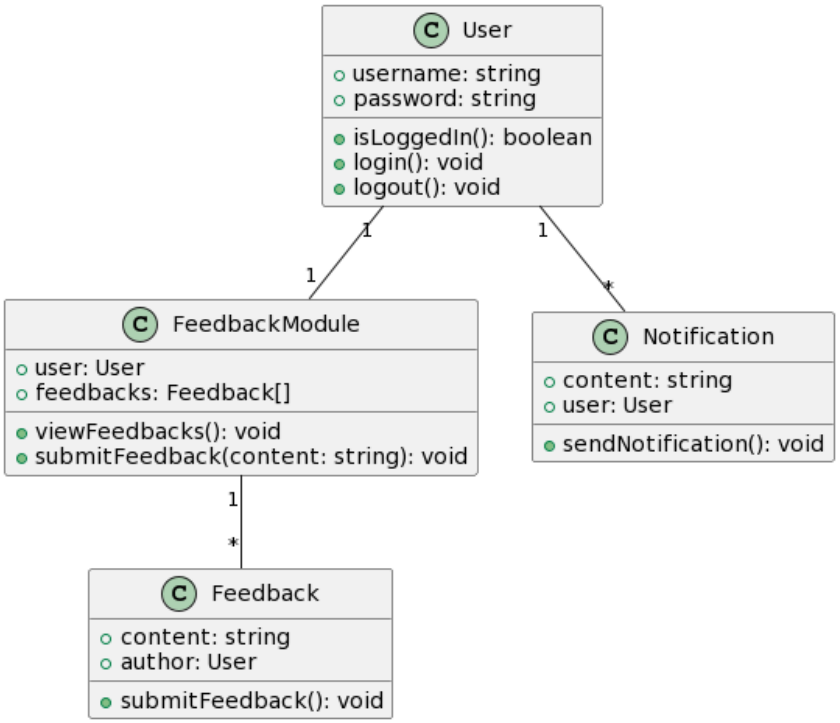
Entity Name	Dashboard
Method Name	createPost
Input	-
Output	void
Algorithm	1. Start 2. Open a form or editor for the user to input the post details 3. Retrieve the input values for the new post 4. Call the createPost() method of the Post class to create a new post 5. End

Entity Name	Dashboard
Method Name	editPost
Input	post: Post
Output	void
Algorithm	1. Start 2. Retrieve the updated content or other input values for editing the post 3. Call the editPost() method of the Post class, passing the post object and the updated input values 4. End

Entity Name	Dashboard
Method Name	deletePost
Input	post: Post
Output	void
Algorithm	1. Start 2. Remove the selected post from the posts list in the dashboard 3. End

Entity Name	Notification
Method Name	sendNotification
Input	-
Output	void
Algorithm	1. Start 2. Compose the notification message or content 3. End

Figure 4.2.2.6: Class Diagram for <Feedback> Subsystem



Entity Name	User
Method Name	isLoggedIn
Input	-
Output	boolean
Algorithm	1. Start 2. Return true of the user's session is active and they are logged in, else false 3. End

Entity Name	User
Method Name	login
Input	-
Output	void
Algorithm	1. Start 2. Validate the user's credentials(username and password) 3. If the credentials are valid, create a session for the user and set the login status to true 4. If the credentials are not valid, display an error message 5. End

Entity Name	User
Method Name	logout
Input	-
Output	void
Algorithm	1. Start 2. Destroy the user's session and set the login status to false 3. Redirect the user to the login page or a logged-out state 4. End

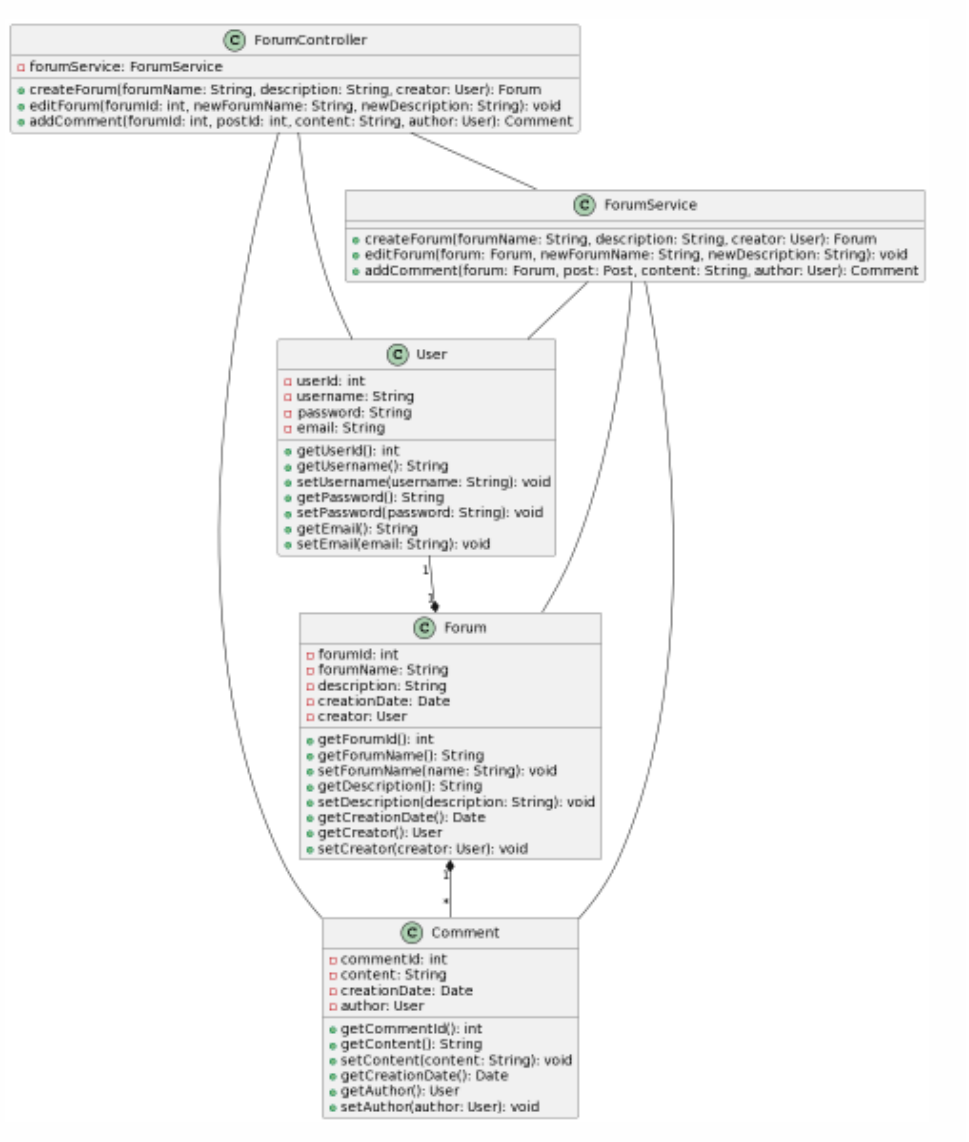
Entity Name	Feedback
Method Name	submitFeedback
Input	-
Output	void
Algorithm	1. Start 2. Retrieve the input content for the feedback 3. Create a new instance of the feedback to the current user 4. Add the new feedback to the feedbacks list in the FeedbackModule 5. End

Entity Name	FeedbackModule
Method Name	viewFeedbacks
Input	-
Output	void
Algorithm	1. Start 2. Fetch the feedbacks from the database or storage 3. Display the feedbacks in the feedback view 4. End

Entity Name	FeedbackModule
Method Name	submitFeedback
Input	content: string
Output	void
Algorithm	1. Start 2. Create a new instance of the Feedback class with the input content 3. Set the author of the feedback to the current user 4. Add the new feedback to the feedbacks list 5. Save the updated feedbacks list to the database or storage 6. End

Entity Name	Notification
Method Name	sendNotification
Input	-
Output	void
Algorithm	1. Start 2. Compose the notification message or content 3. Assign the notification to the user 4. Send the notification to the user 5. End

Figure 4.2.2.7: Class Diagram for <Anonymous Forum> Subsystem



Entity Name	Forum
Method Name	createForum
Input	forumName(string) , description(string) , creator(user)
Output	forum(Forum)
Algorithm	1. Start 2. Create a new instance of Forum 3. Set the forum name description and creator 4. Add the forum to the system 5. Return the created forum. 6. End

Entity Name	Forum
Method Name	editForum
Input	forum(Forum) , newForumName(string) , newDescription(string)
Output	void
Algorithm	1. Start 2. Update the forum's name and descriptions 3. Save the changes to the system 4. End

Entity Name	Forum
Method Name	addComment
Input	forum(Forum) , content(string) , author(user)
Output	comment(comment)
Algorithm	1. Start 2. Create a new instance of comment 3. Set the comment's content, creation date and author 4. Add the comment to the forum and post 5. Return to the created comment 6. End

Entity Name	User
Method Name	getUserId
Input	
Output	userId(int)
Algorithm	1. Start 2. Return the userId of the user 3. End

Entity Name	User
Method Name	getUsername
Input	
Output	username(string)
Algorithm	1. Start 2. Return the username of the user 3. End

Entity Name	User
Method Name	setUserName
Input	username(string)
Output	void
Algorithm	1. Start 2. Set the username of the user 3. End

Entity Name	User
Method Name	getPassword
Input	-
Output	password(string)
Algorithm	1. Start 2. Return the password of the user 3. End

Entity Name	User
Method Name	setPassword
Input	password(string)
Output	void
Algorithm	1. Start 2. Set the password of the user 3. End

Entity Name	User
Method Name	getEmail
Input	-
Output	email(string)
Algorithm	1. Start 2. Return the email of the user 3. End

Entity Name	User
Method Name	setEmail
Input	email(string)
Output	void
Algorithm	1. Start 2. Set the email of the user 3. End

Entity Name	Comment
Method Name	getCommentId
Input	-
Output	commentId
Algorithm	1. Start 2. Return the commentId of the comment 3. End

Entity Name	Comment
Method Name	getContent
Input	-
Output	content(string)
Algorithm	1. Start 2. Return the content of the comment 3. End

Entity Name	Comment
Method Name	setContent
Input	content(string)
Output	void
Algorithm	1. Start 2. Set the content of the comment 3. End

Entity Name	Comment

Method Name	getCreationDate
Input	-
Output	creationDate(Date)
Algorithm	1. Start 2. Return the creation date of the comment 3. End

Entity Name	Comment
Method Name	getAuthor
Input	-
Output	author(user)
Algorithm	1. Start 2. Return the author of the comment 3. End

Entity Name	Comment
Method Name	setAuthor
Input	author(user)
Output	void
Algorithm	1. Start 2. Set the author of the comment 3. End

4.2.3 Sequence Diagram

a) SD001: Sequence diagram for Sign Up

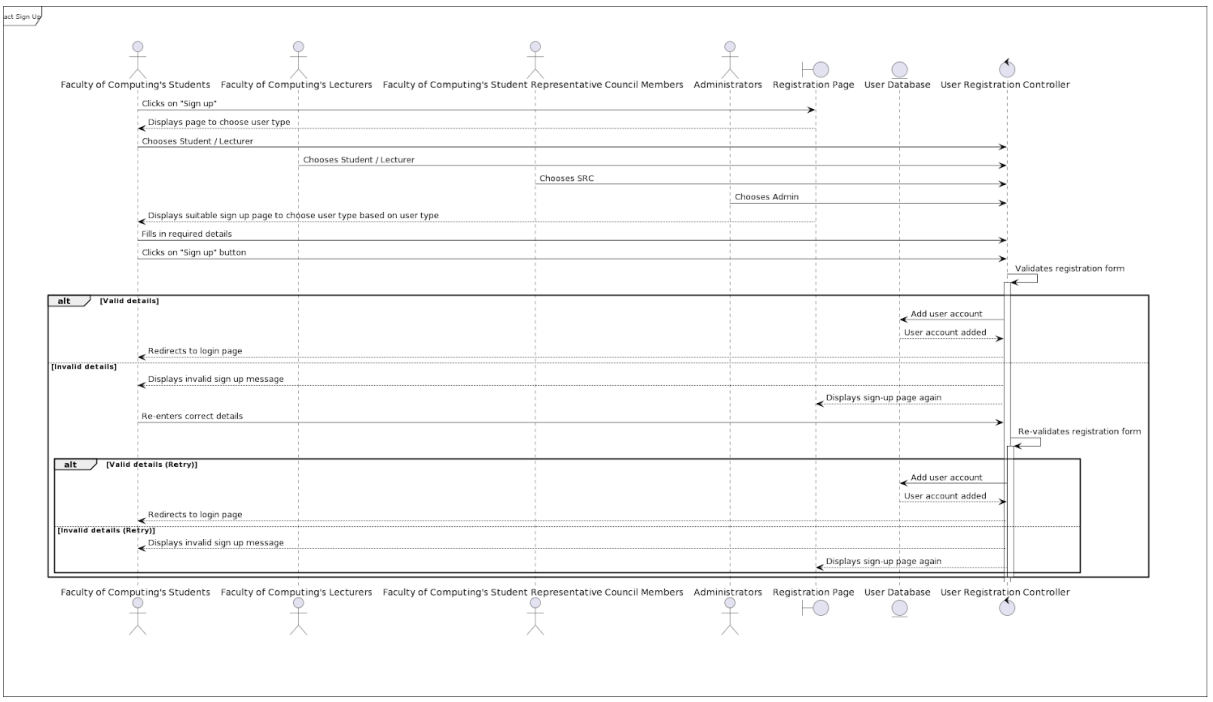


Figure 4.2.3.1: Sequence Diagram for <Sign Up>

b) SD002: Sequence diagram for Email Verification

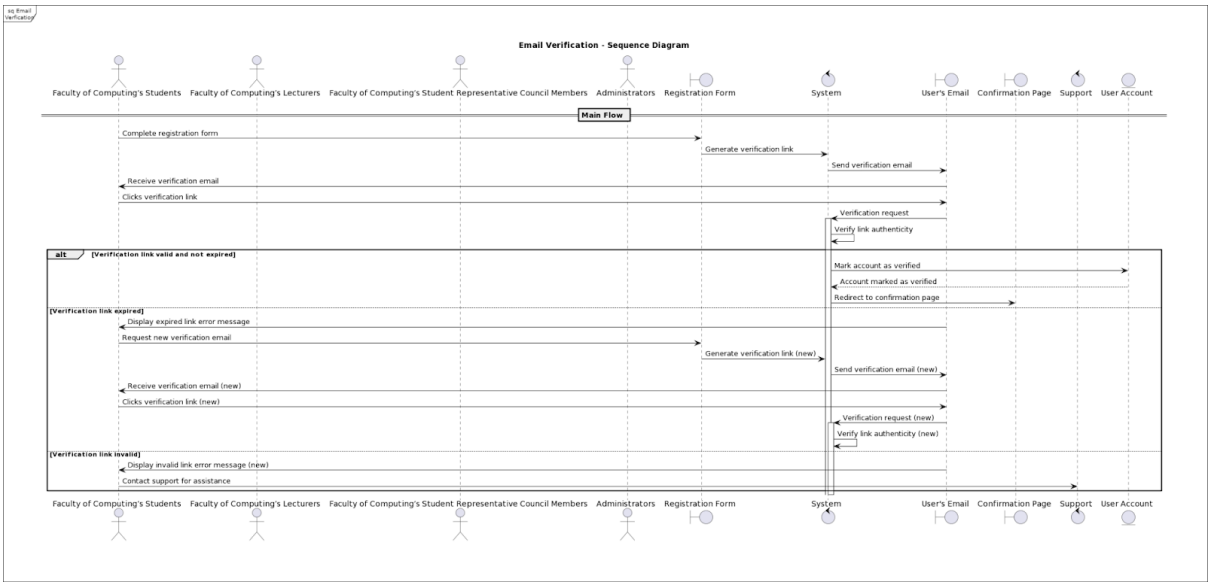


Figure 4.2.3.2: Sequence Diagram for <Email Verification>

c) SD003: Sequence diagram for Reset Password

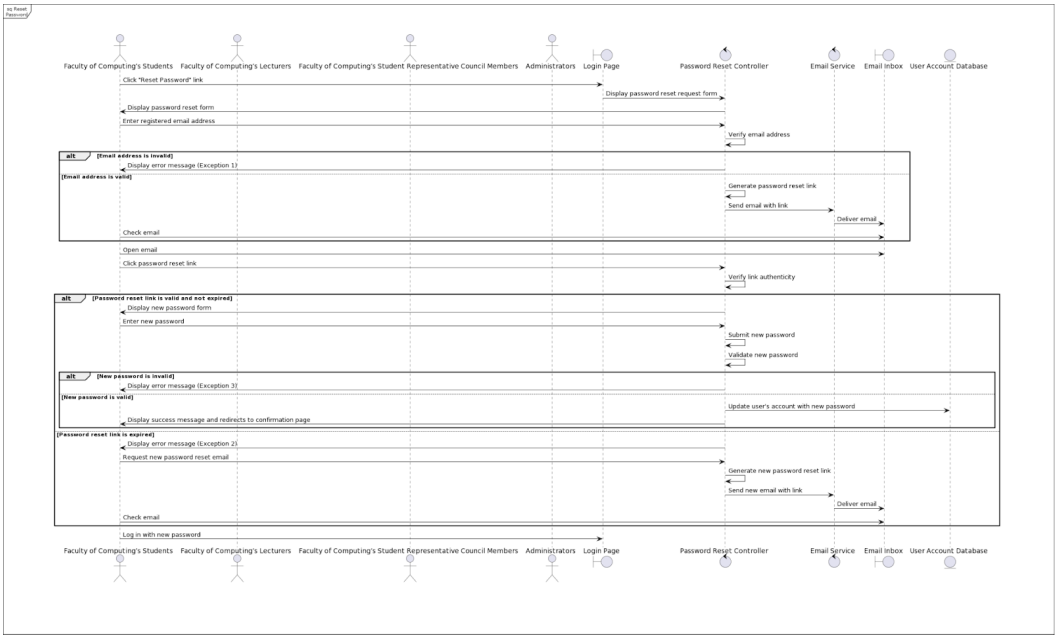


Figure 4.2.3.3: Sequence Diagram for <Reset Password>

d) SD004: Sequence diagram for Login

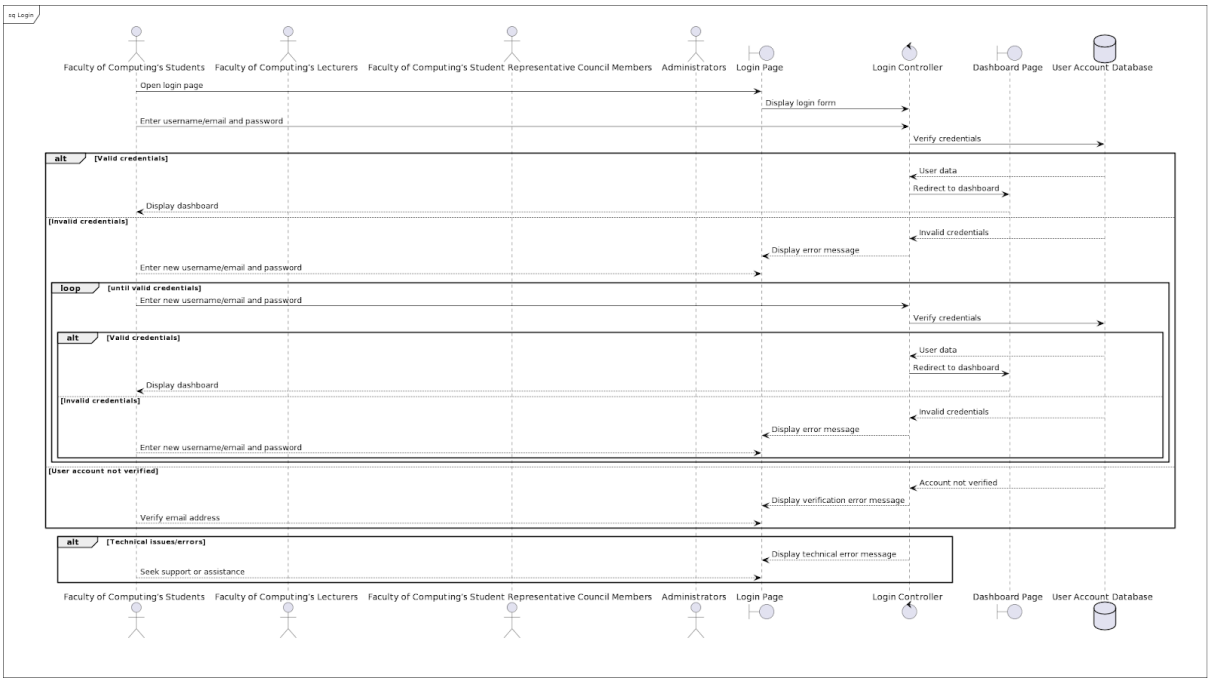


Figure 4.2.3.4: Sequence Diagram for <Login>

e) SD005: Sequence diagram for Update Profile

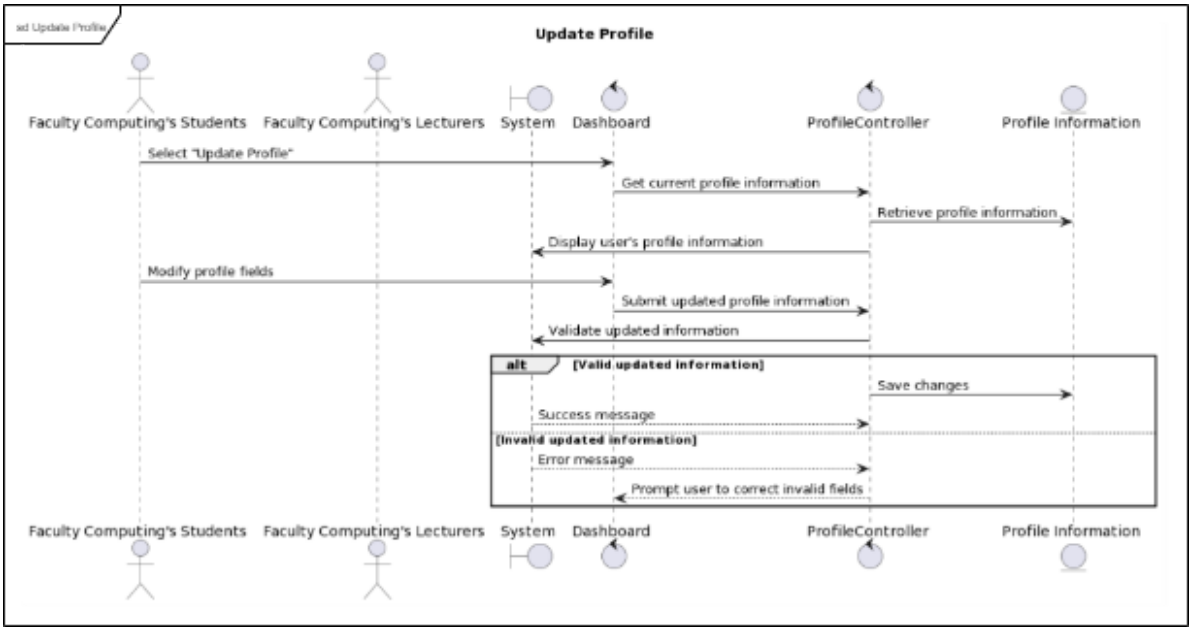


Figure 4.2.3.5: Sequence Diagram for <Update Profile>

f) SD006: Sequence diagram for Delete Profile

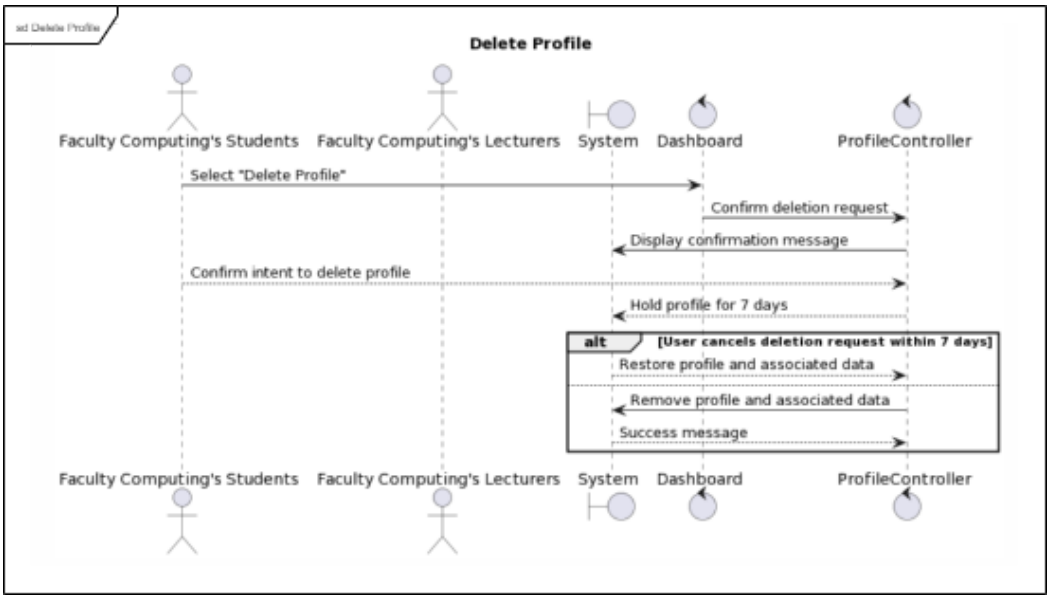


Figure 4.2.3.6: Sequence Diagram for <Delete Profile>

g) SD007: Sequence diagram for Forum and Post Draft

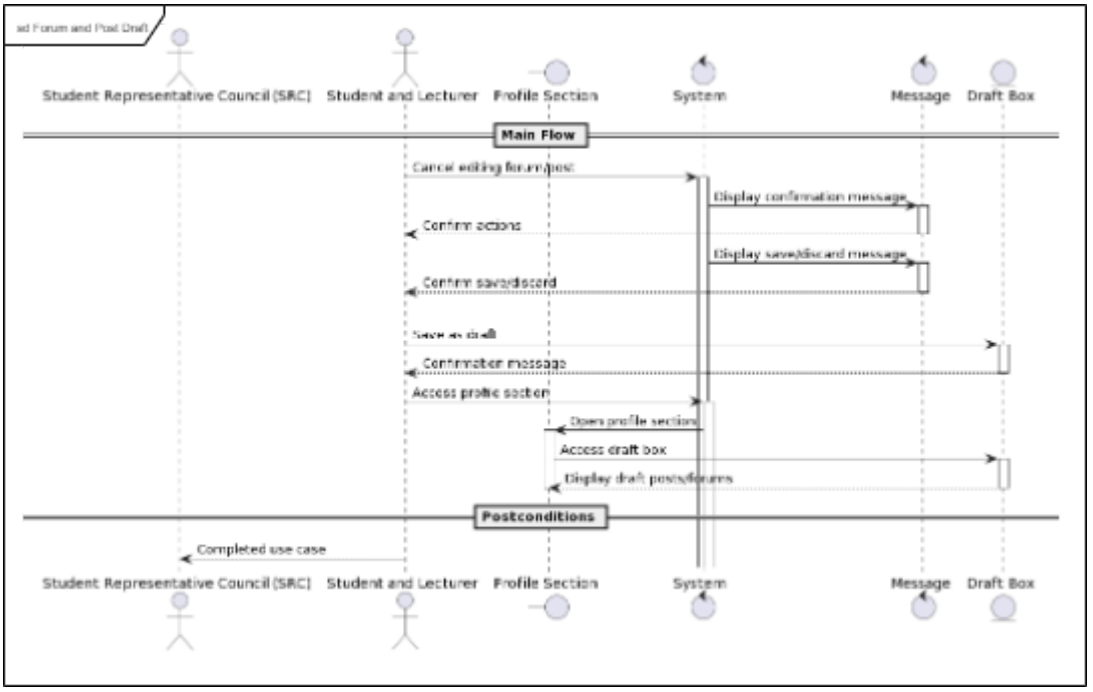


Figure 4.2.3.7: Sequence Diagram for <Forum and Post Draft

h) SD008: Sequence diagram for Update SRC Information Board

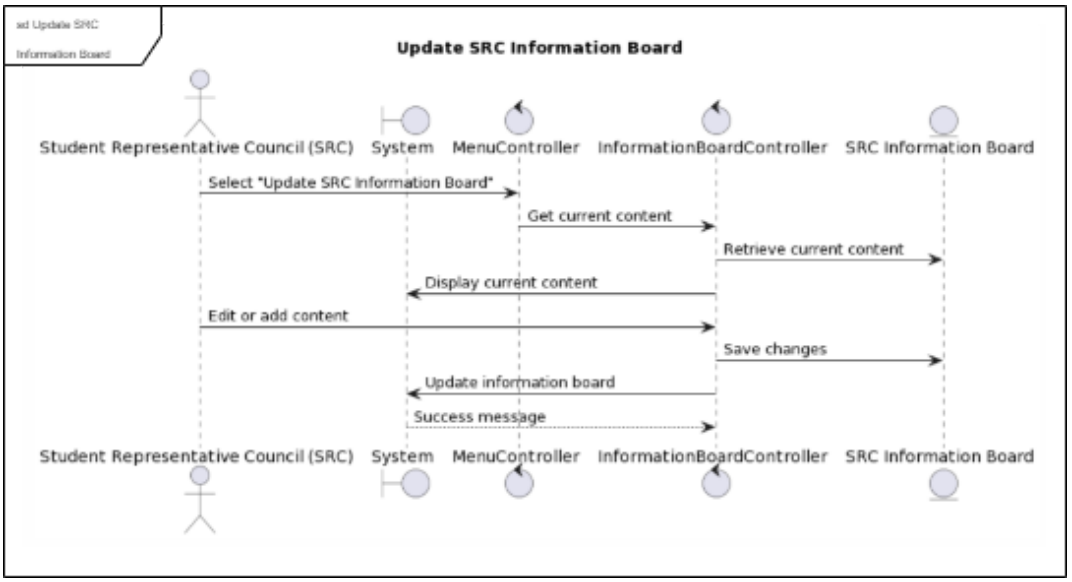


Figure 4.2.3.8: Sequence Diagram for <Update SRC Information Board>

i) SD009: Sequence diagram for Post Filtration

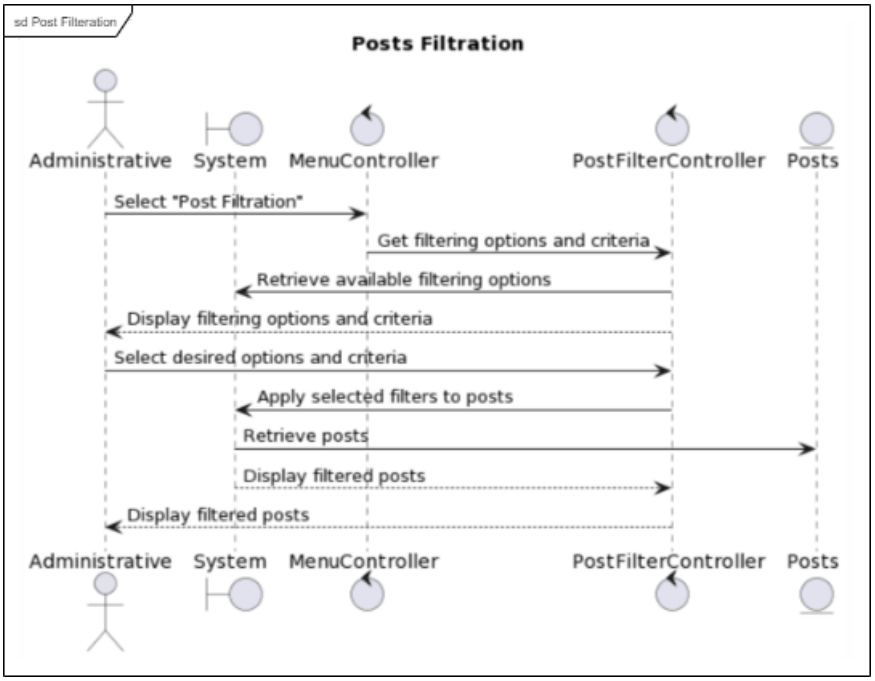


Figure 4.2.3.9: Sequence Diagram for <Post Filtration>

j) SD010: Sequence diagram for Validate Feedback

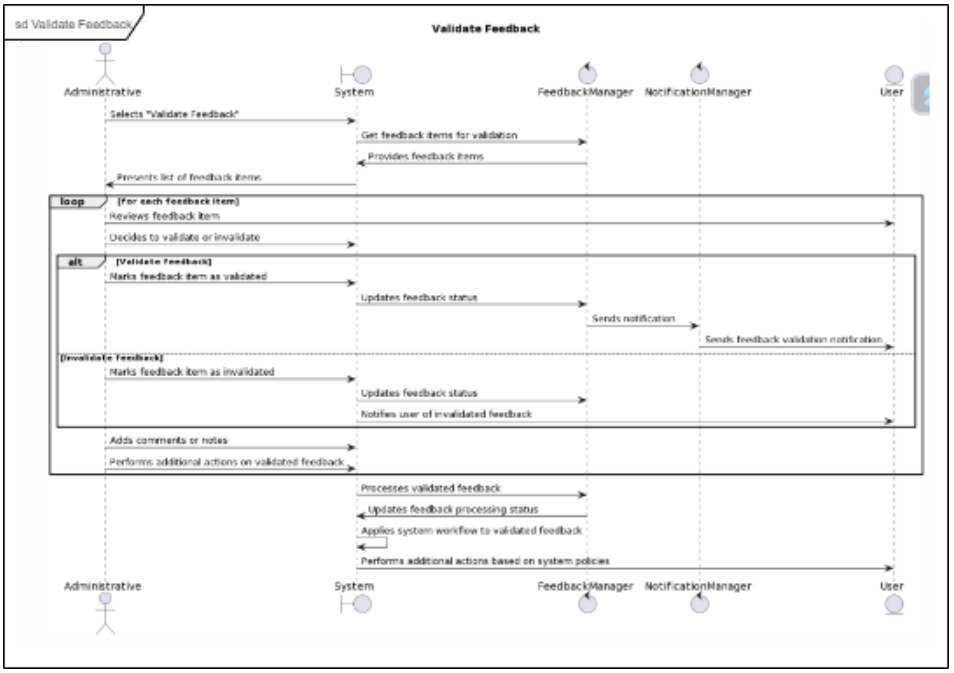


Figure 4.2.3.10: Sequence Diagram for <Validate Feedback>

k) SD011: Sequence diagram for Generate Report

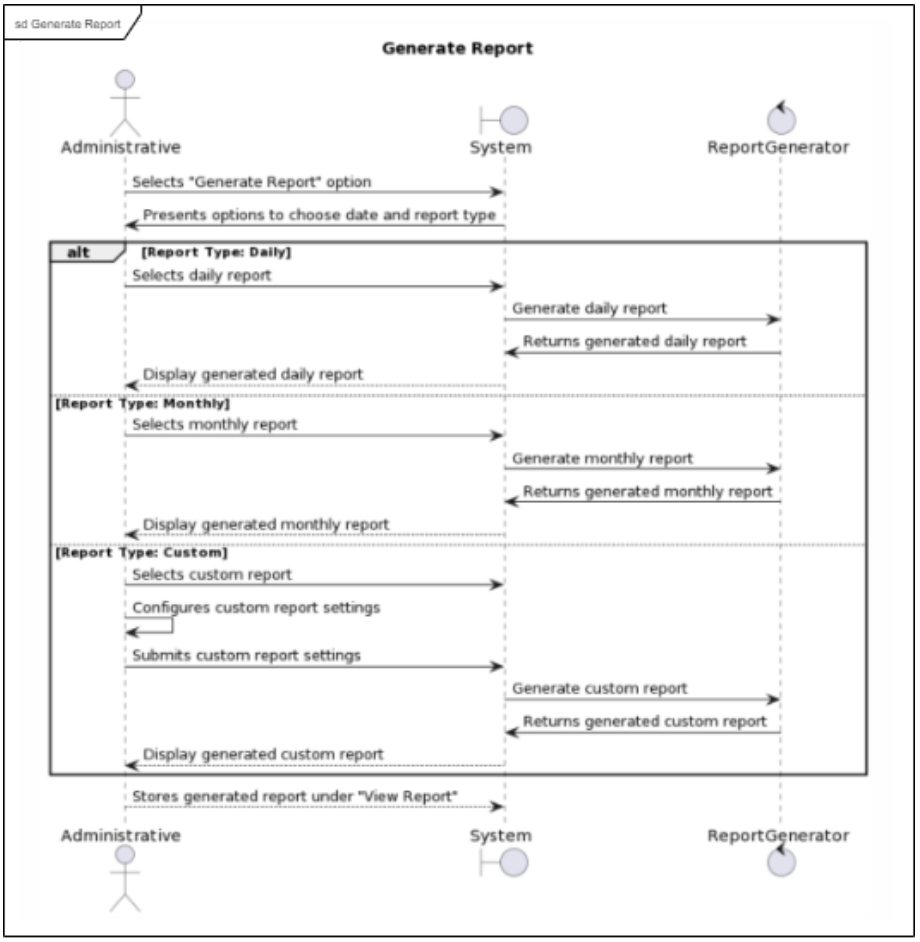


Figure 4.2.3.11: Sequence Diagram for <Generate Report>

l) SD012: Sequence diagram for Download Report

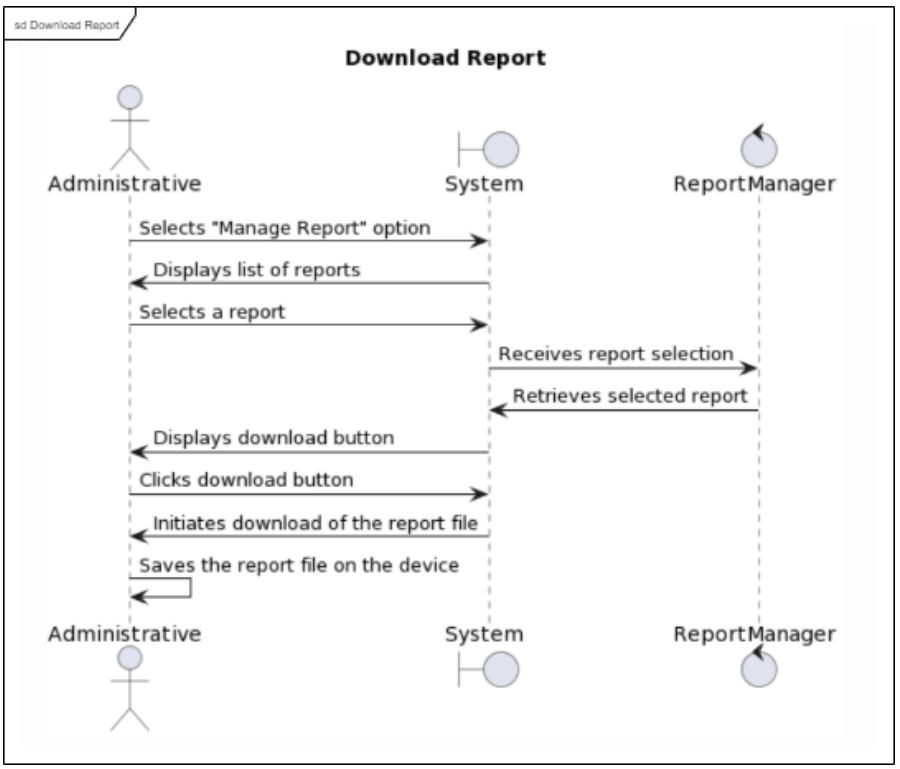


Figure 4.2.3.12: Sequence Diagram for <Download Report>

m) SD013: Sequence diagram for Manage User

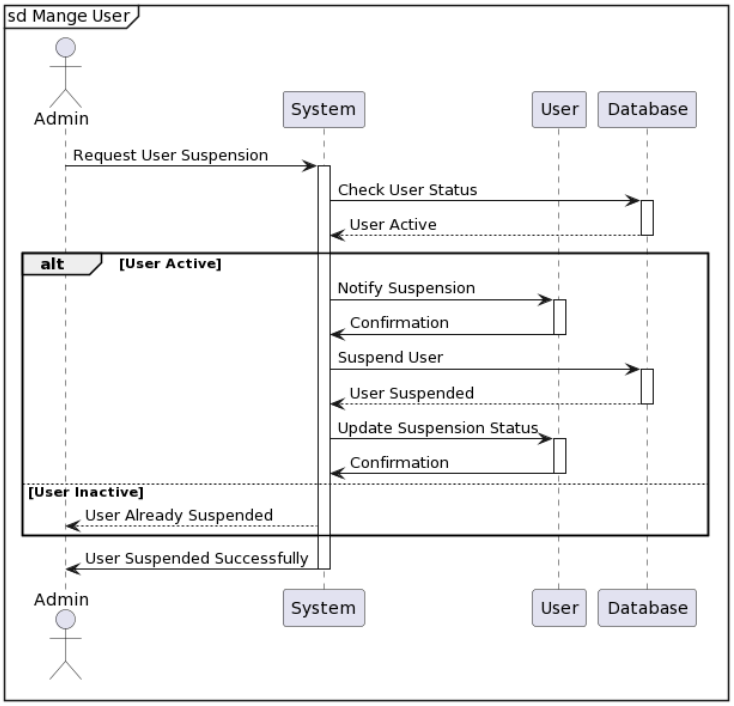


Figure 4.2.3.13: Sequence Diagram for <Manage User>

o)SD014: Sequence diagram for Make Post

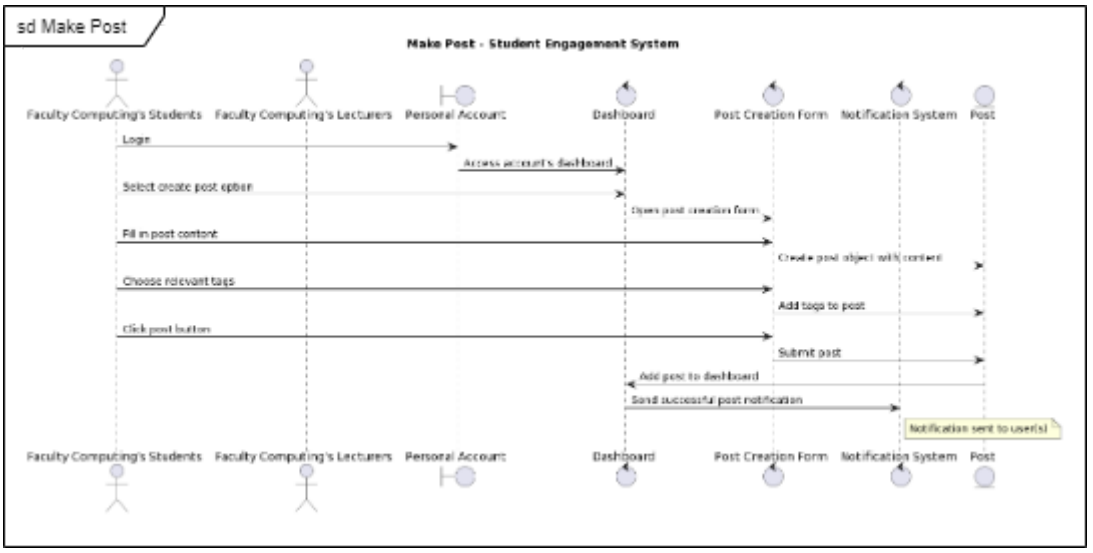


Figure 4.2.3.14: Sequence Diagram for <Make Post>

p)SD015: Sequence diagram for Edit Post

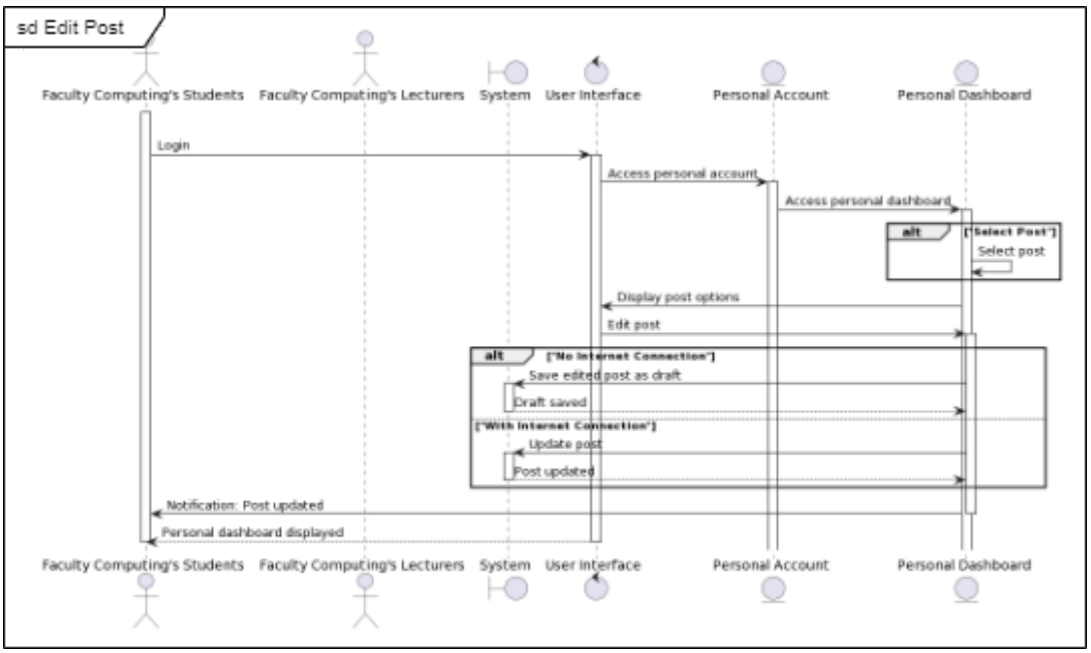


Figure 4.2.3.15: Sequence Diagram for <Edit Post>

q)SD016: Sequence diagram for Delete Post

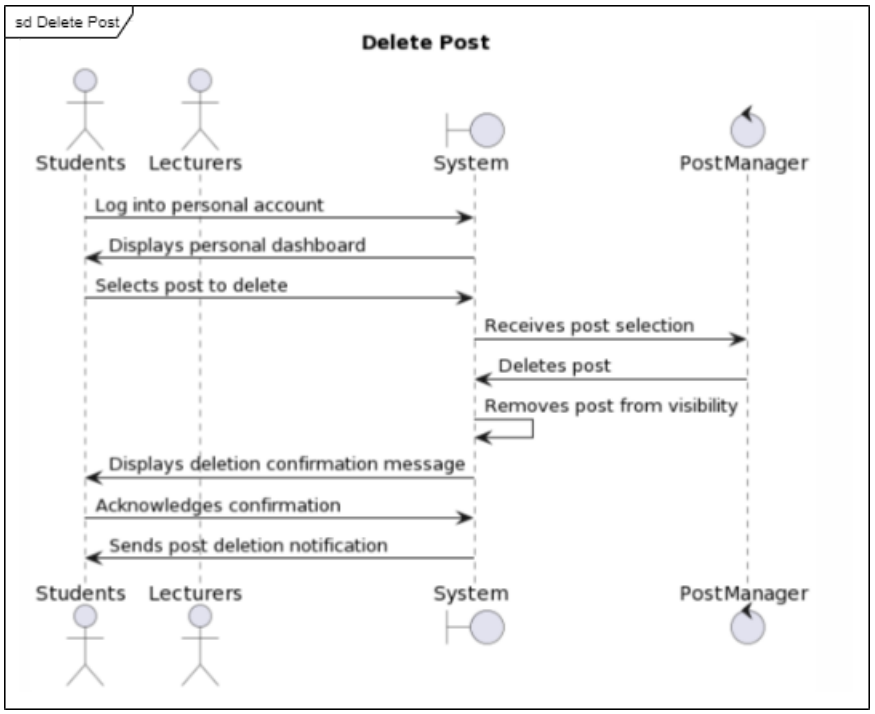


Figure 4.2.3.16: Sequence Diagram for <Delete Post>

r)SD017: Sequence diagram for View Feedback

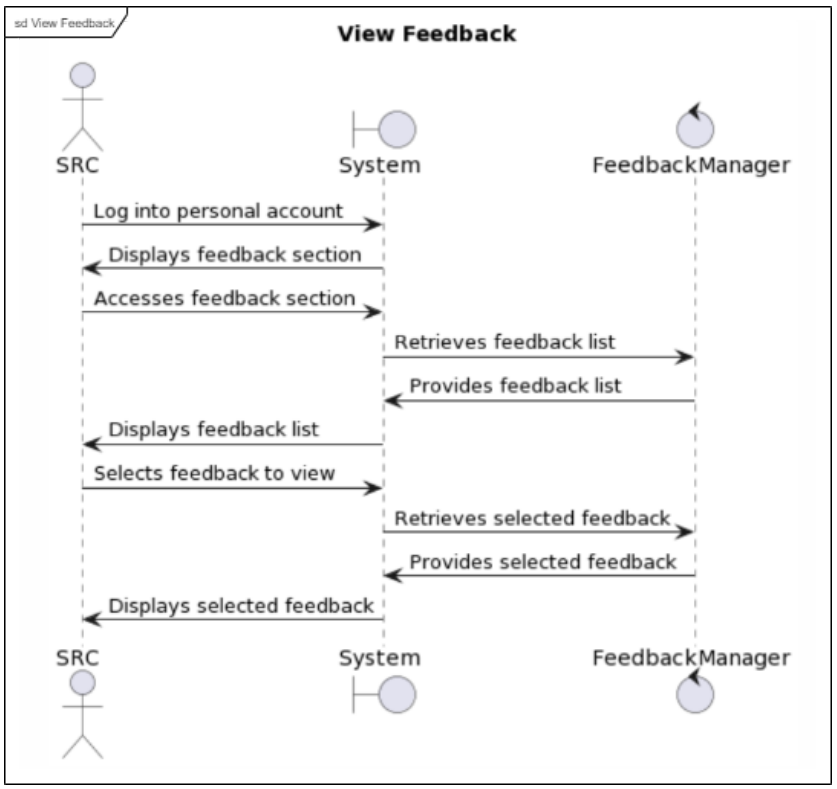


Figure 4.2.3.17: Sequence Diagram for <View Feedback>

s)SD018: Sequence diagram for Submit Feedback

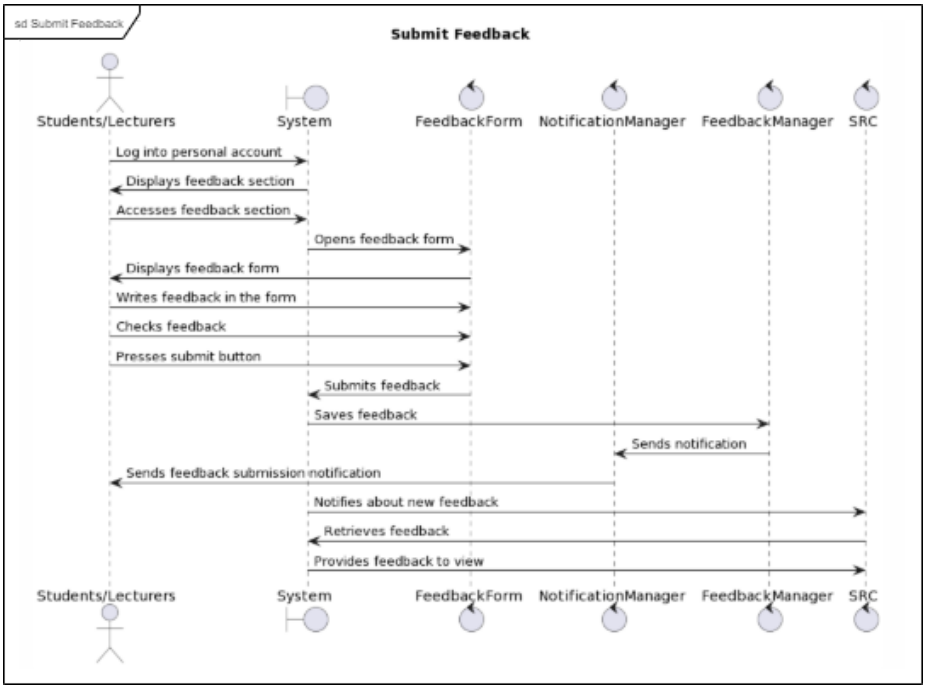


Figure 4.2.3.18: Sequence Diagram for <Submit Feedback>

t) SD019: Sequence diagram for Create Forum

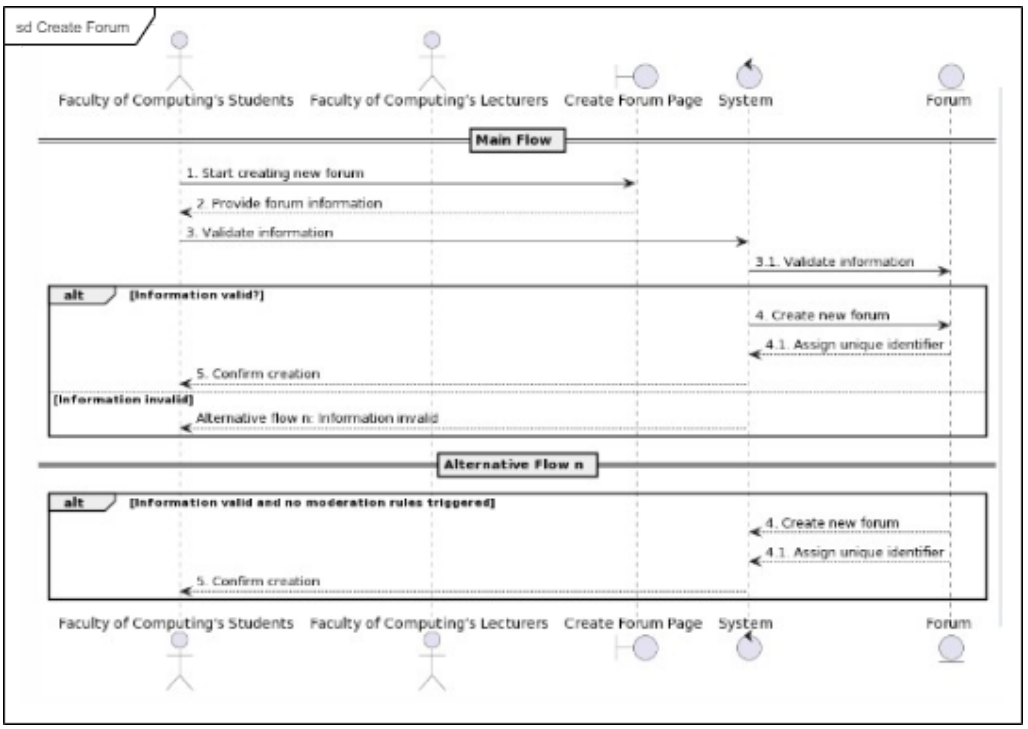


Figure 4.2.3.19: Sequence Diagram for <Create Forum>

u) SD020: Sequence diagram for Forum Comments

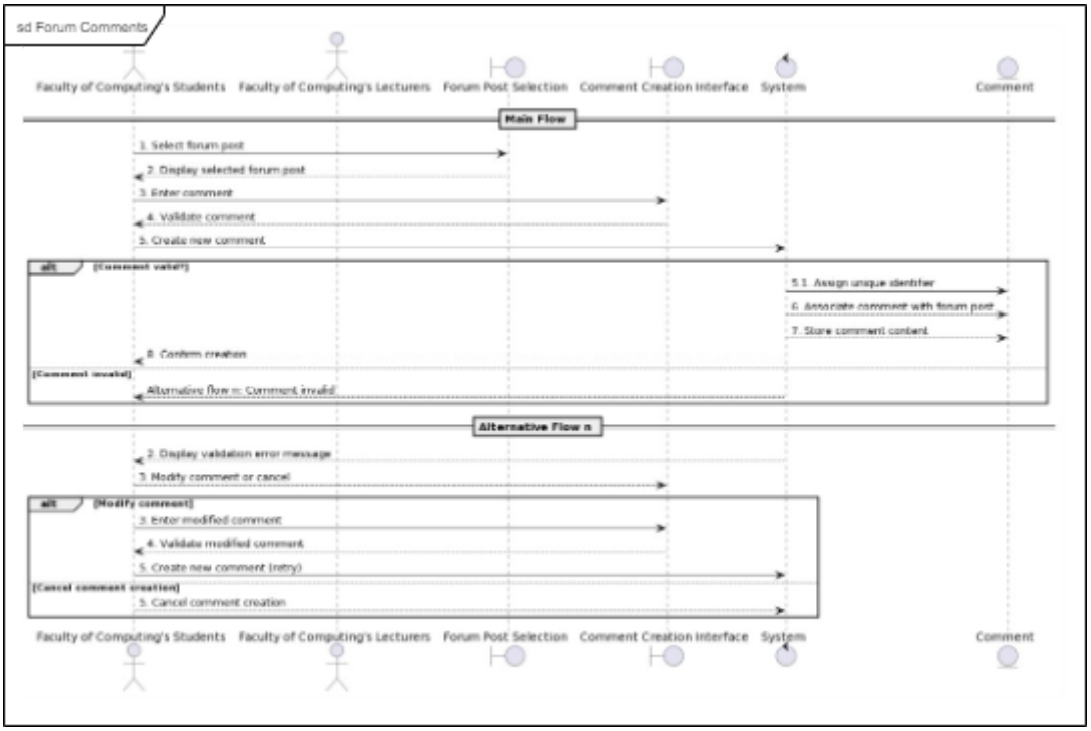


Figure 4.2.3.20: Sequence Diagram for <Forum Comments>

v)SD021: Sequence diagram for Edit Forum

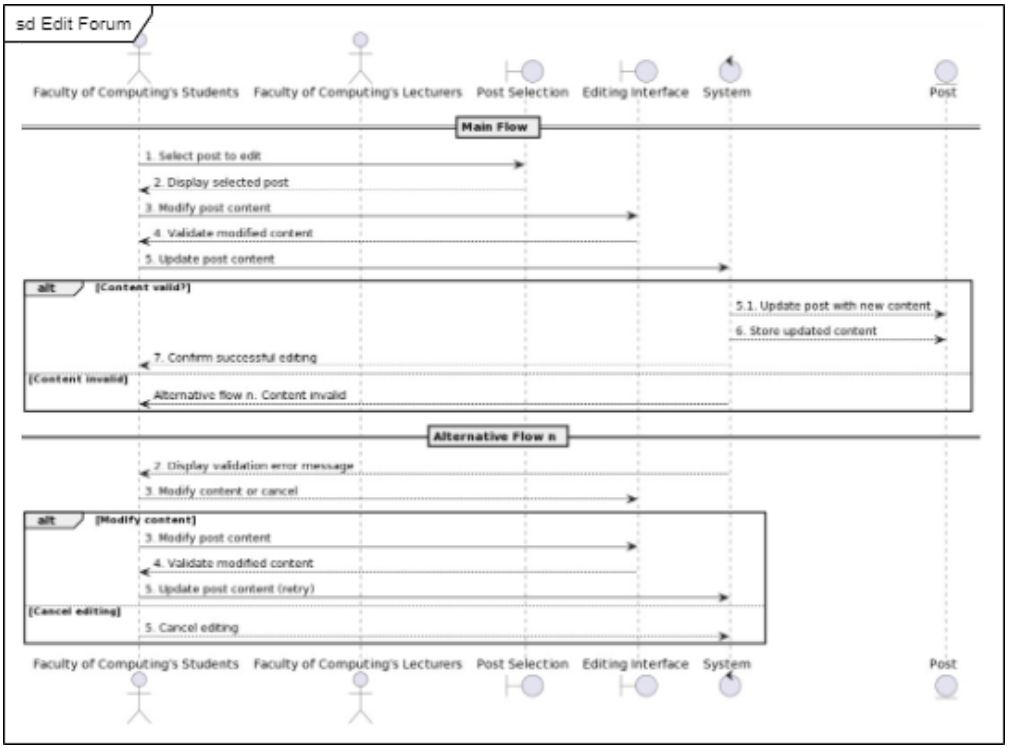


Figure 4.2.3.21: Sequence Diagram for <Edit Forum>

5. Data Design

5.1 Data Description

The major data or systems entities are stored into a relational database named as…, processed and organized into *n* entities as listed in Table 5.1.

Table 5.1: Description of Entities in the Database

No.	Entity Name	Description
1.	User	Represents a user of the system.
2.	Student	Represents a student at the UTM Faculty of Computing.
3.	Lecturer	Represents a lecturer at the UTM Faculty of Computing.
4.	SRC Member	Represents a student representative council member.
5.	Administrator	Represents an administrator of the system.
6.	UserRepository	Manages the storage and retrieval of user data.
7.	UserController	Handles user-related operations and interactions.
8.	AuthenticationService	Handles authentication and authorization processes.
9.	AuthenticationController	Manages the authentication and authorization operations.
10.	RegistrationView	Provides a view for user registration.
11.	VerificationView	Provides a view for user verification.
12.	LoginView	Provides a view for user login.
13.	View	Represents a general view component for the system's user interface.
14.	ForumPostDraft	The ForumPostDraft entity represents a draft of a forum post or discussion created by a user. It is used to store the content that a user has composed but has not yet submitted or published. The ForumPostDraft entity allows users to save their work in progress and come back to it later for further editing or submission. It typically includes attributes such as the draft ID, content, author, and timestamp. The content attribute holds the actual text or body of the draft post, allowing users to write and edit their thoughts, ideas, or questions before finalizing and posting them on the forum. The ForumPostDraft entity

		helps users manage and organize their unfinished forum posts, ensuring that their work is not lost and can be easily retrieved for completion or further refinement.
15.	Administrative	Administrative represents an administrative user or role within a system or organization. Administrators are responsible for managing and maintaining the system, performing administrative tasks, and ensuring the smooth operation of the system.
16.	Feedback	The Feedback entity represents feedback provided by users of a system or application. Feedback allows users to express their opinions, suggestions, or concerns regarding the system's features, usability, or any other aspect. Here contains feedbackID to distinguish each feedback entry, content contains actual feedback message or comments, author refers to the user who submitted the feedback, and validate indicating whether the feedback has been validated or not. It serves as a valuable source of information for ensuring a continuous feedback loop between the system developers and the user community.
17.	Post	The Post entity represents a post or message created by users within a system or application. Posts allow users to share information, ask questions, or engage in discussions. The Post entity includes attributes such as postID, content, author. It enables the creation of a community-driven platform where users can interact, provide feedback, and exchange knowledge or ideas. Posts can be displayed, sorted, and searched based on various criteria, providing users with easy access to relevant discussions and facilitating effective communication within the system.
18.	Report	
19.	ManageUser	
20.	Dashboard	The dashboard entity represents all the posts that users have posted. They can either view, edit their posted post, create post or delete current post.

21.	Notification	In this entity, the system will send notifications to users regarding their status of posted posts and also forums.
22.	FeedbackModule	In this entity, users are able to view the forum that posted by themselves and also others as well as submit written feedback
23.	Forum	A Forum represents an online discussion platform within the Anonymous Forum Module. It provides a space for users, such as Faculty Computing's Students and Lecturers, to create and participate in discussions on various topics. Each forum has a unique name, a description to provide context, and can contain multiple posts and comments. Users can create new forums, edit existing forums, and interact with posts and comments within the forum.
24.	Comment	A Comment is a user-generated response or contribution within a forum's post. It represents a user's opinion, feedback, or additional information related to the discussion. Each comment has a unique identifier, content that includes the text of the comment, a creation date to track when it was posted, and an associated author who wrote the comment. Comments can be added to posts within forums, allowing users to engage in conversations and express their thoughts on specific topics.

5.2 Data Dictionary

5.2.1 Entity: <User>

Attribute Name	Type	Description
username	string	● Represents username of the user
password	string	● Represents password of the user
email	string	● Represents UTM email of the user

5.2.2 Entity: <Student>

Attribute Name	Type	Description
-	-	-

5.2.3 Entity: <Lecturer>

Attribute Name	Type	Description
=	-	-

5.2.4 Entity: <SRC Member>

Attribute Name	Type	Description
-	-	-

5.2.5 Entity: <Administrator>

Attribute Name	Type	Description
-	-	-

5.2.14 Entity: <ForumPostDraft>

Attribute Name	Type	Description
userID	string	● Represents the unique identifier for each user
draftContent	string	● Represents the saved draft content

5.2.15 Entity: <Administrative>

Attribute Name	Type	Description

adminID	string	• Represents the unique identifier for admin

5.2.16 Entity: <Feedback>

Attribute Name	Type	Description
feedbackID	string	• Represents the unique identifier of the feedback • Used for referencing and identifying the feedback
content	string	• Represent the description of the feedback • Stores the feedback content
author	User	• Represent the user who submit the feedback • Stores the username
validate	boolean	• Represent the validation status of the feedback • Indicates whether the feedback has been validated or not

5.2.17 Entity: <Post>

Attribute Name	Type	Description
postID	string	• Represents the unique identifier of the post • Used for referencing and identifying the post
content	string	• Represent the description of the post • Stores the content
author	user	• Represent the user who created the post • Stores the username

5.2.18 Entity: <Report>

Attribute Name	Type	Description
-	-	-

5.2.19 Entity: <Manage User>

Attribute Name	Type	Description
-	-	-

5.2.20 Entity: <Dashboard>

Attribute Name	Type	Description
user	User	● Represent each of the user accounts
posts	Post[]	● Represents the post that had been posted by each user accounts

5.2.21 Entity: <Notification>

Attribute Name	Type	Description
content	string	● Represent the statement of notification that would be send to the users
user	User	● Represent each of the user accounts

5.2.22 Entity: <Feedback Module>

Attribute Name	Type	Description
user	User	● Represent each of the user accounts
feedbacks	Feedback[]	● Represent the posted feedback by each of the user accounts

5.2.23 Entity: <Forum>

Attribute Name	Type	Description
forumId	Integer	● Represents the unique identifier of the forum. ● Used for referencing and identifying the forum.
forumName	String	● Represents the name or title of the forum ● Provides a concise description of the forum's topic
forumDescriptions	String	● Contains a brief description of the forum's topic ● Gives users an overview of the forum's purpose or content
forumCreationDate	Date	● Represents the date when the forum was created ● Helps track the forum's age and chronological order

creater	String	● Represent the user who created the forum（with anonymous name) ● Stores the username

5.2.24 Entity: <Comment>

Attribute Name	Type	Description
commentId	String	● Represents the unique identifier for a comment ● Used to uniquely identify and reference a specific comment
content	String	● Contains the actual text or content of the comment ● Stores the message or information shared by the commenter
forumId	forum	● Reference to the forum entity ● Represent the forum to which the comment is associated
author	String	● Represent the user who authored the comment ● Stores the information or reference of the author of the comment (with anonymous name)

5.2.25 Entity: <Admin>

Attribute Name	Type	Description
id	Integer	● Represents the unique identifier for admin
username	String	● Contains the actual text of username
password	String	● Represent a length of text to access admin account

6. User Interface Design

6.1 Overview of User Interface

The interface includes a range of features that cater to the needs of users in the UTM Faculty of Computing Student Engagement System.

First and foremost, there is a registration and authentication process. Users can sign up for the system using their UTM email address. Users can sign up as students, lecturers, Student Representative Council (SRC) members, or administrators. The user needs to sign up with their username, UTM email address, and password. For SRC and Administrator sign-up, a verification code has to be entered. After the sign-up process, users verify their email through an interface. If the verification link is expired, users can click on the resend verification link to resend the email verification link. If the verification link is invalid, users can click on contact support to request help. In case users forget their passwords, there is password reset functionality available. Once registered, users can login to the system using their unique username and password. There is a drop down in the login screen which allows users to login as student / lecturer, SRC or Admin. The user will go to the homepage of student / lecturer, SRC or Admin, depending on their choice of user type.

The system also provides options for users to manage their user information. Students and lecturers have an interface to update their profiles, allowing them to keep their personal details up-to-date. Additionally, they have the option to delete their profiles if needed. A convenient interface allows students, lecturers, and SRC to save their forum posts or draughts, enabling them to work on their content before posting it. Moreover, the Student Representative Council (SRC) has access to update the information board, providing them with a platform to share important updates with other users. To ensure the quality of content, the system includes processing features. Posts submitted by users undergo content filtration by the admin through a special interface to ensure appropriate and relevant information is shared.

The administration and reporting capabilities of the system offer additional functionality. There is an interface for Administrators to generate reports. Once generated, administrators can view the reports to gain insights into various aspects of the system. They also have the ability to manage reports, organizing and categorizing them as needed. Furthermore, administrators can download reports for further analysis or sharing with other stakeholders.

Students, lecturers, and SRC are provided with a personalized dashboard, offering a user-centric experience. Within their dashboard, users can create posts and engage in discussions. The system allows users to edit their posted content, ensuring they can make necessary changes or updates. If users decide they no longer want to keep a post, they have the option to delete it from their dashboard. They can also see posts from other users in their dashboard. They can then go into their post to comment or like it.

Feedback plays a crucial role in the system, and users have dedicated features for this purpose. There is an interface for students and lecturers to submit their feedback. Students' and lecturers' feedback can be viewed by the SRC, enabling improvements based on the valuable input received. Admins have an interface to manage their submitted feedback, making it easier to track and update their responses over time.

Lastly, the system supports an anonymous forum where students, lecturers, and SRC members can freely express their thoughts and opinions. Students, lecturers, and SRC members have the ability to create forums and initiate discussions. Additionally, all users are allowed to comment on posts within the forum. A flexible feature allows users to edit their forum posts even after they have been published.

For every interface, there is a button to allow the user to logout.

6.2 Screen Images

Registration and Authentication Subsystem

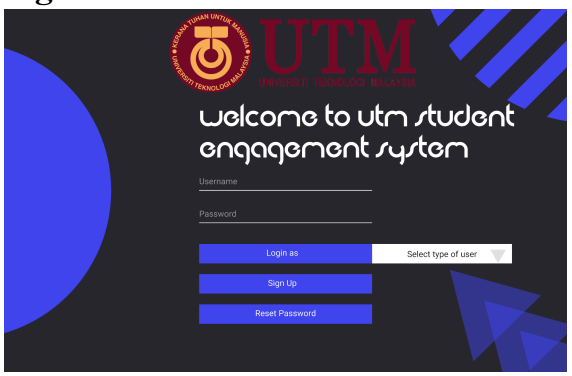


Figure 6.2.1: Interface for <Login Main>

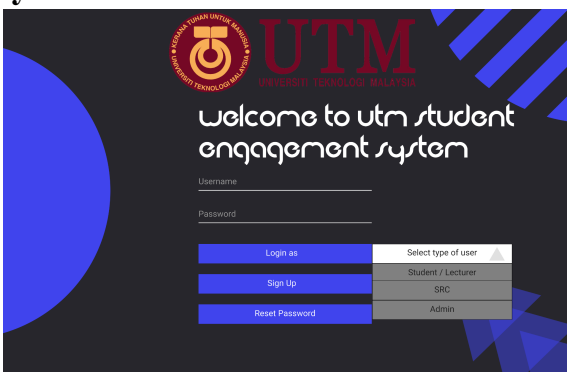


Figure 6.2.2: Interface for <Login Main (Select User Type)>

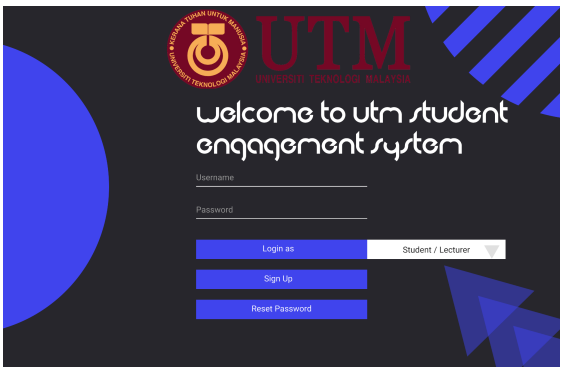


Figure 6.2.3: Interface for <Login Main (Student / Lecturer)>

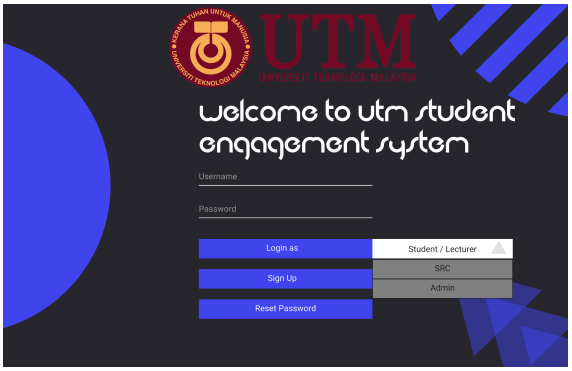


Figure 6.2.4: Interface for <Login Main (Student / Lecturer) - Choose Another User Type>

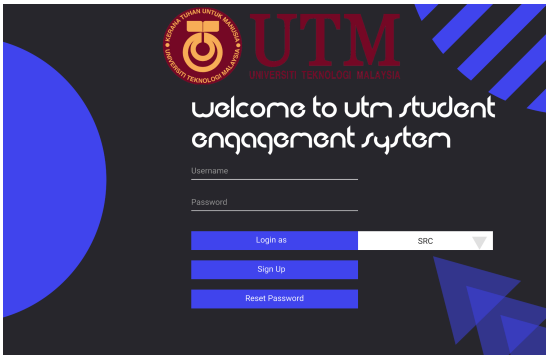


Figure 6.2.5: Interface for <Login Main (SRC)>

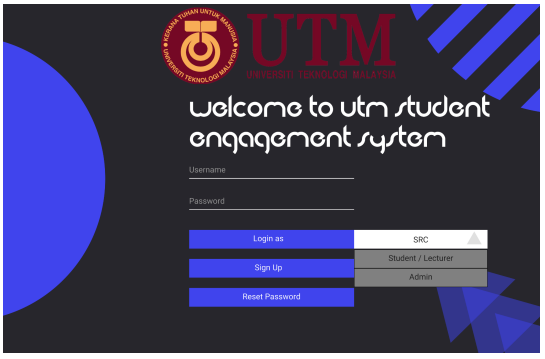


Figure 6.2.6: Interface for <Login Main (SRC) - Choose Another User Type>

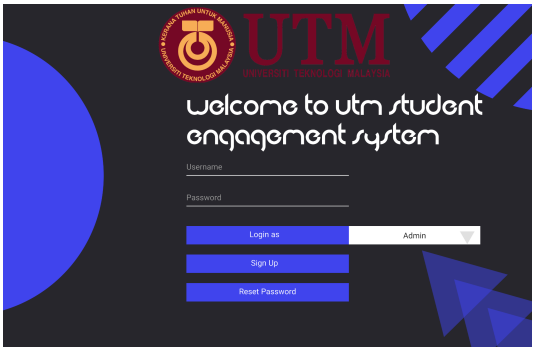


Figure 6.2.7: Interface for <Login Main (Admin)>

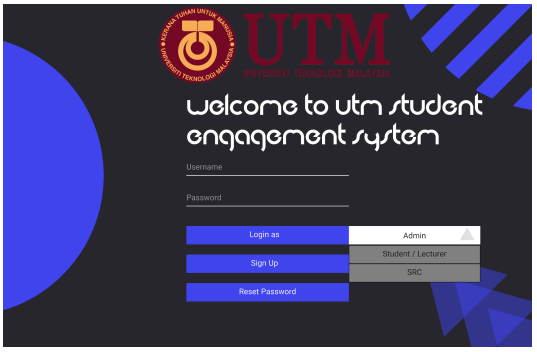


Figure 6.2.8: Interface for <Login Main (Admin) - Choose Another User Type>

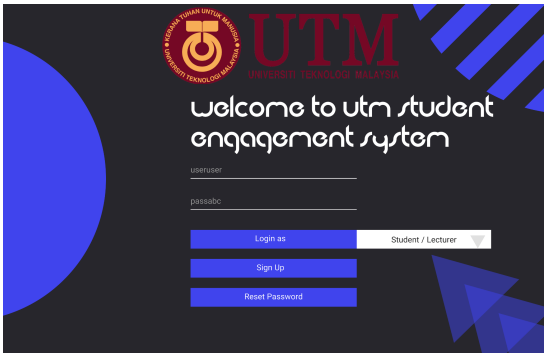


Figure 6.2.9: Interface for <Login With Input (Student / Lecturer)>

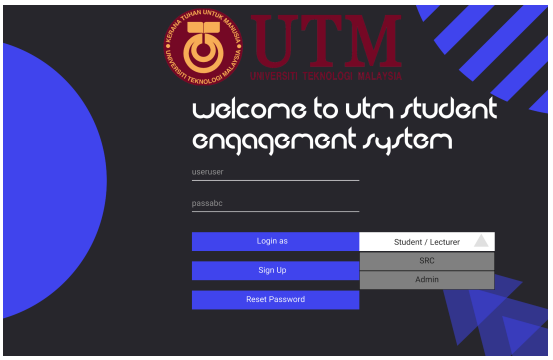


Figure 6.2.10: Interface for <Login With Input (Student / Lecturer) (Change User Type)>

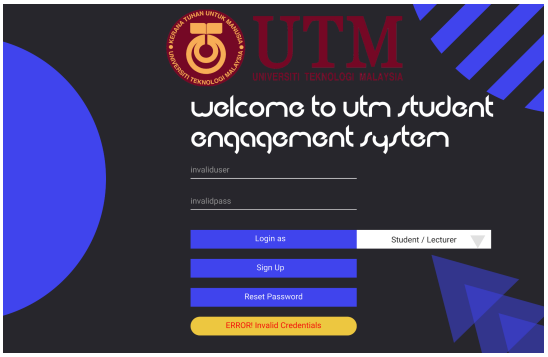


Figure 6.2.11: Interface for <Login (Student / Lecturer) (Invalid Credentials)>

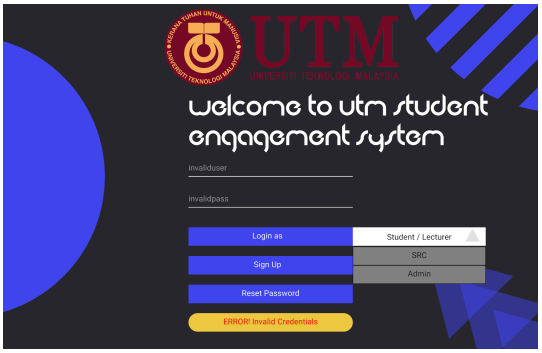


Figure 6.2.12: Interface for <Login (Student / Lecturer) (Invalid Credentials) (Change User Type)>

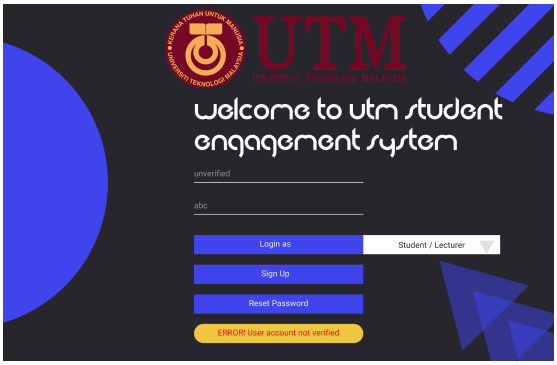


Figure 6.2.13: Interface for <Login (Student / Lecturer) (Unverified User)>

Figure 6.2.14: Interface for <Login (Student / Lecturer) (Unverified User) (Change User Type)>

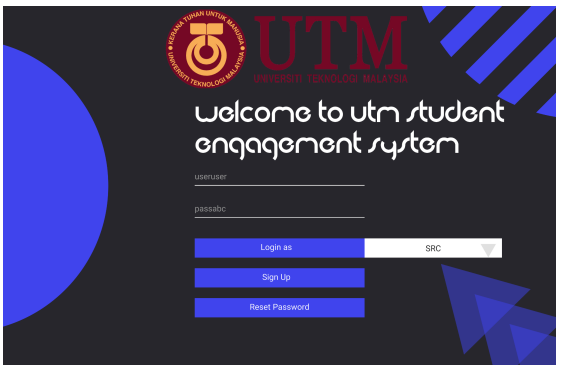


Figure 6.2.15: Interface for <Login With Input (SRC)>

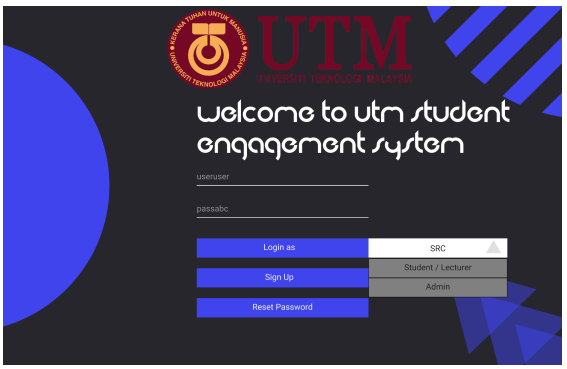


Figure 6.2.16: Interface for <Login With Input (SRC) (Change User Type)>

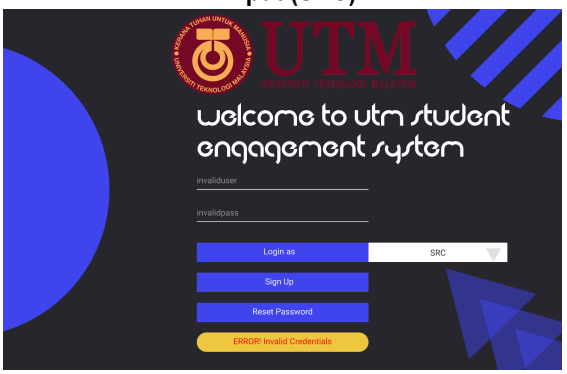


Figure 6.2.17: Interface for <Login (SRC) (Invalid Credentials)>

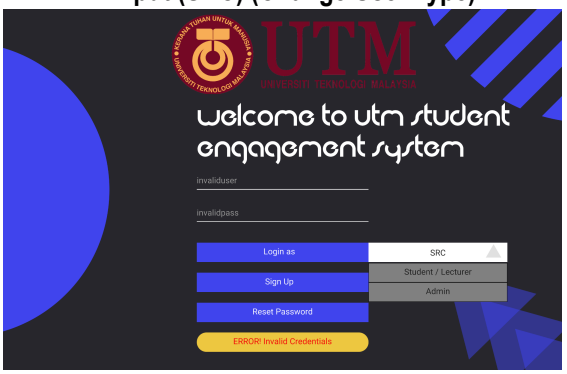


Figure 6.2.18: Interface for <Login (SRC) (Invalid Credentials) (Change User Type>

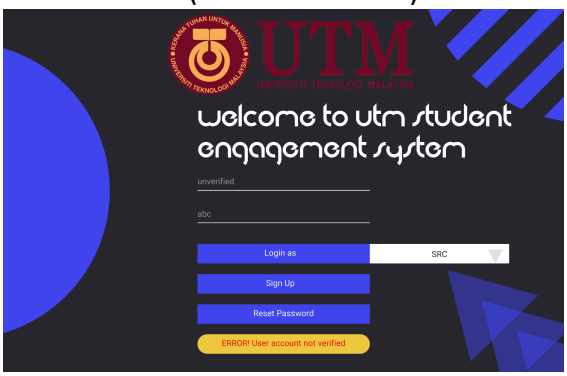


Figure 6.2.19: Interface for <Login (SRC) (Unverified User)>

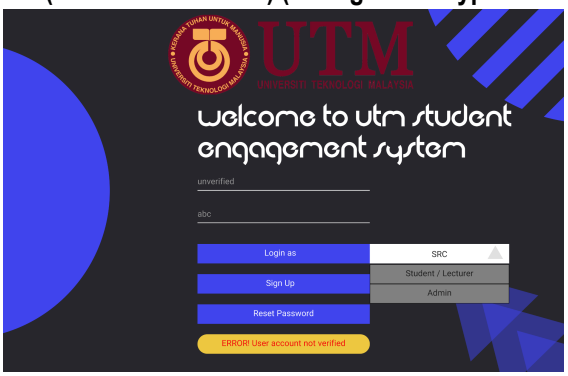


Figure 6.2.20: Interface for <Login (SRC) (Unverified User) (Change User Type)>

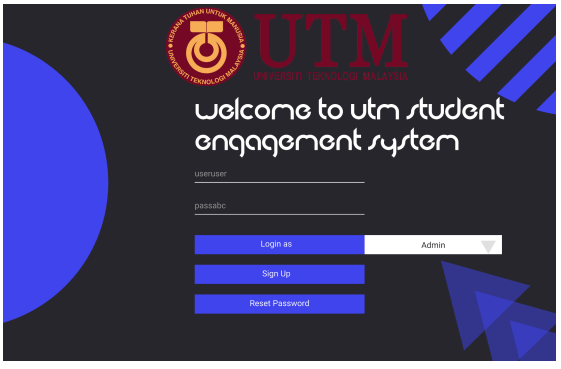


Figure 6.2.21: Interface for <Login With Input (Admin)>

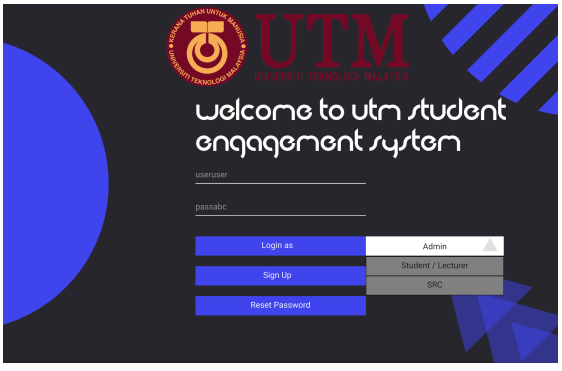


Figure 6.2.22: Interface for <Login With Input (Admin) (Change User Type)>

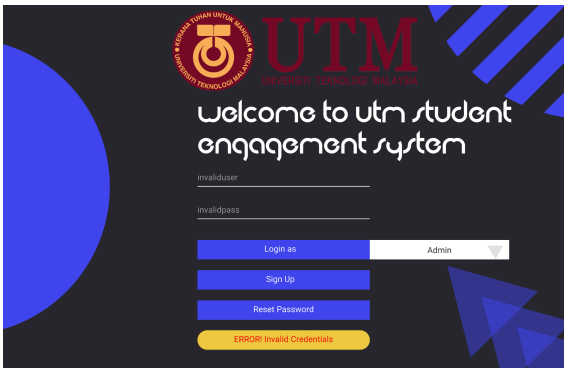


Figure 6.2.23: Interface for <Login (Admin) (Invalid Credentials)>

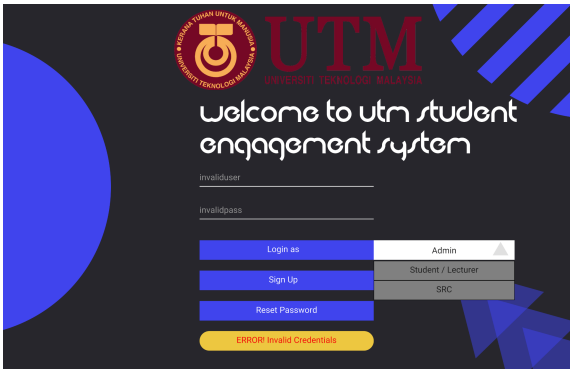


Figure 6.2.24: Interface for <Login (Admin) (Invalid Credentials) (Change User Type)>

Figure 6.2.25: Interface for <Login (Admin) (Unverified User)>

Figure 6.2.26: Interface for <Login (Admin) (Unverified User) (Change User Type)>

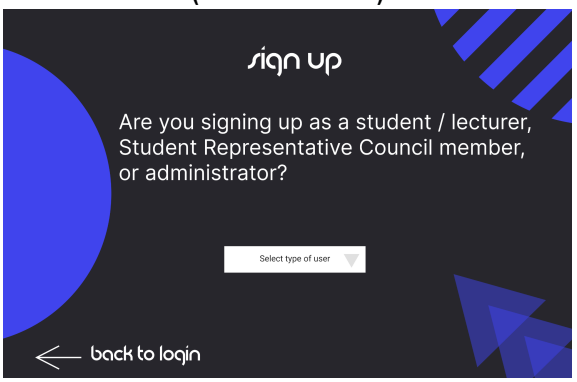


Figure 6.2.27: Interface for <Sign Up (Choose User Type)>

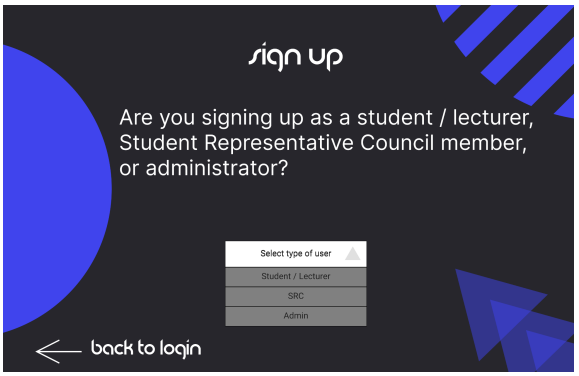


Figure 6.2.28: Interface for <Sign Up (Choose User Type Dropdown)>

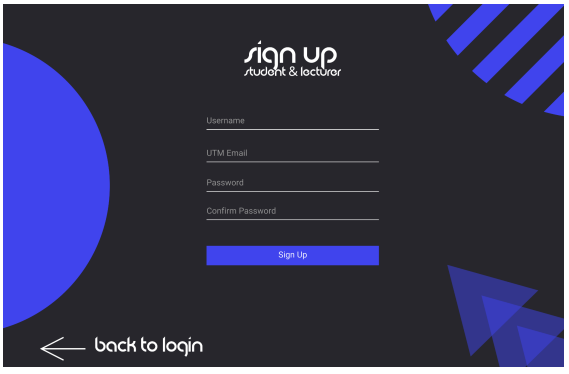


Figure 6.2.29: Interface for <Sign Up -Student & Lecturer>

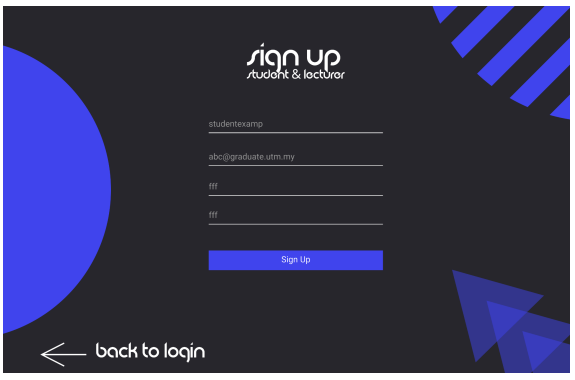


Figure 6.2.30: Interface for <Sign Up -Student & Lecturer Filled>

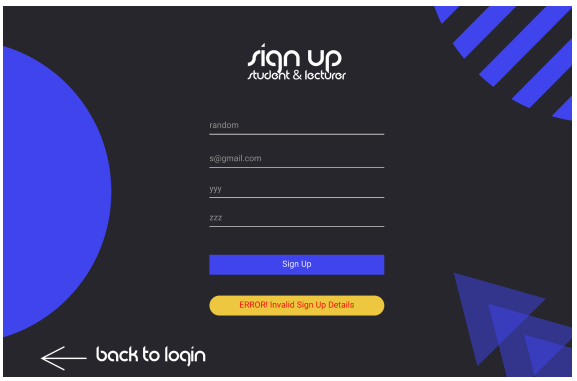


Figure 6.2.31: Interface for <Sign Up -Student & Lecturer (Invalid Sign Up)>

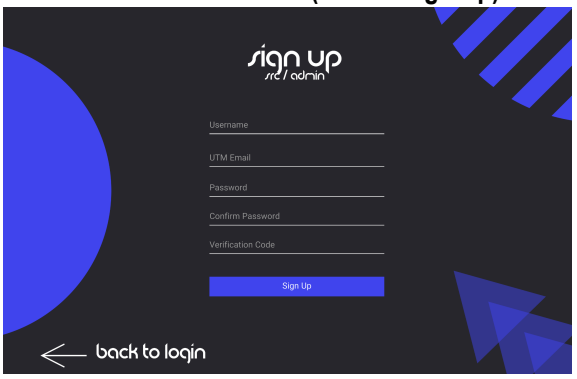


Figure 6.2.32: Interface for <Sign Up -SRC/ Admin>

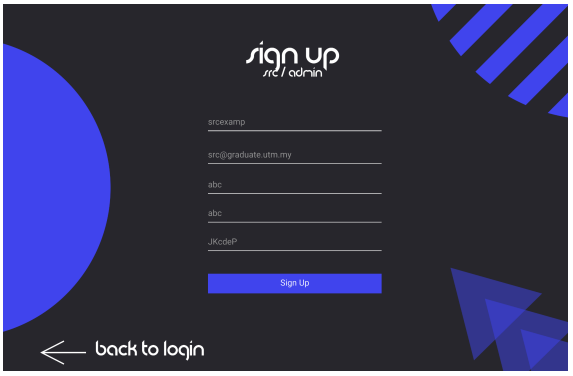


Figure 6.2.33: Interface for <Sign Up -SRC/ Admin Filled>

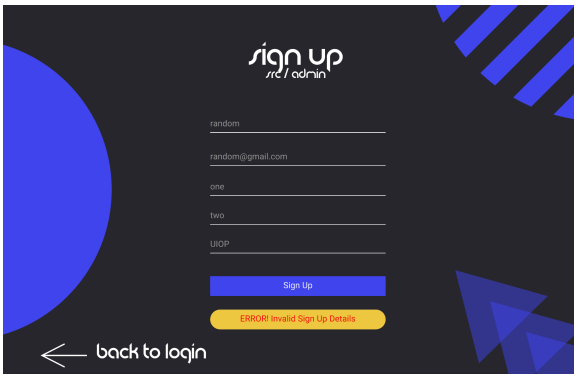


Figure 6.2.34: Interface for <Sign Up -SRC / Admin (Invalid Sign Up)>

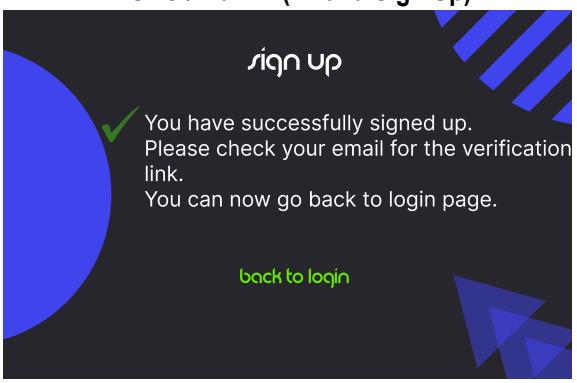


Figure 6.2.35: Interface for <Sign Up (Success)>

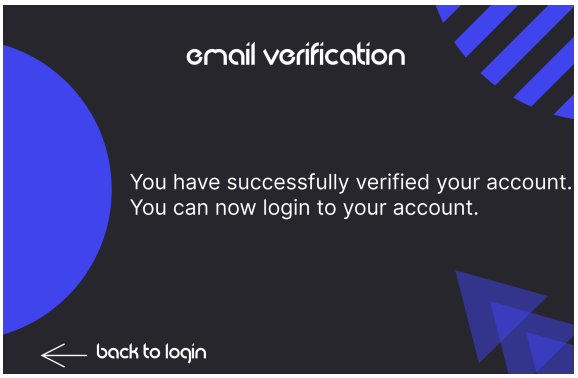


Figure 6.2.36: Interface for <Email Verification (Success)>

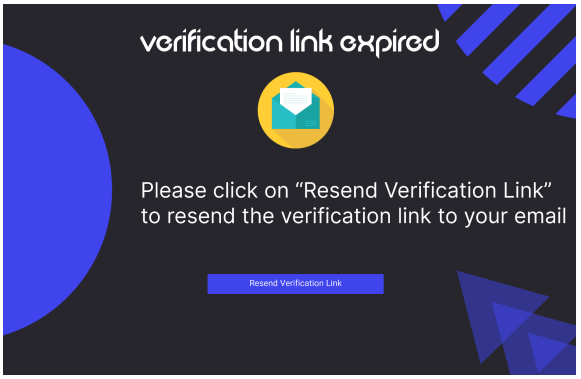


Figure 6.2.37: Interface for <Email Verification (Expired Link)>

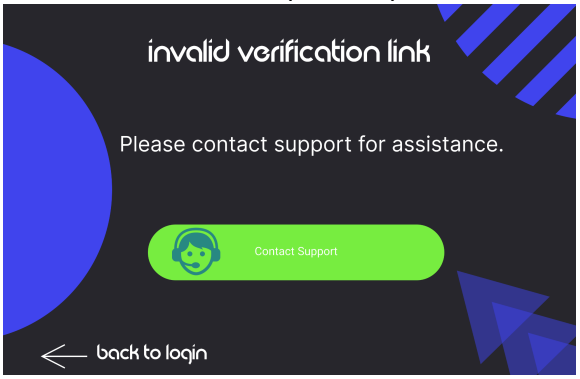


Figure 6.2.38: Interface for <Email Verification (Invalid Link)>

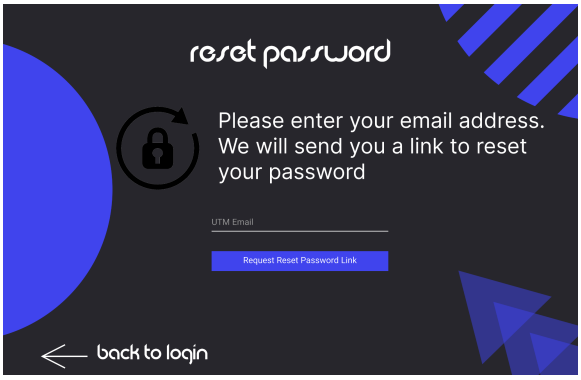


Figure 6.2.39: Interface for <Reset Password (Request Reset Password)>

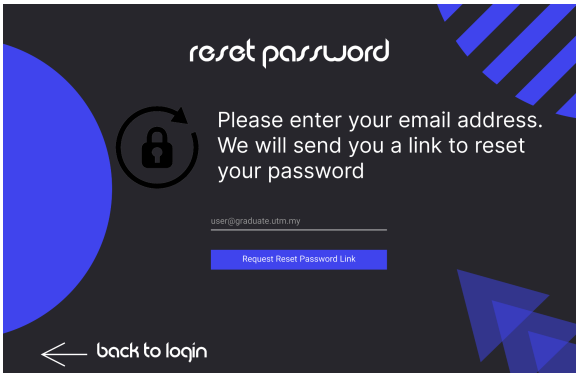


Figure 6.2.40: Interface for <Reset Password (Request Reset Password Filled)>

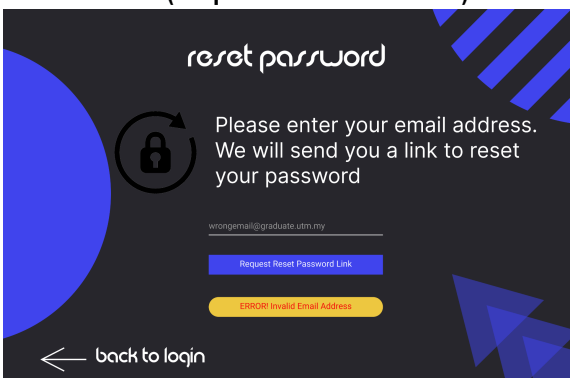


Figure 6.2.41: Interface for <Reset Password (Request Reset Password - Invalid Email)>

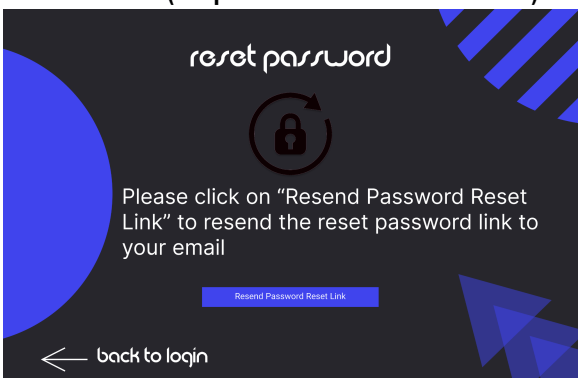


Figure 6.2.42: Interface for <Password Reset Link Expired)>

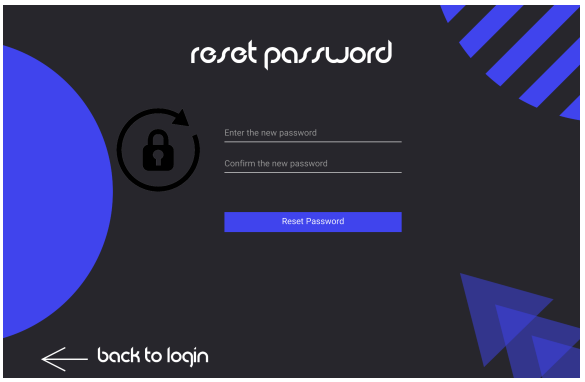


Figure 6.2.43: Interface for <Reset Password (New Password)>

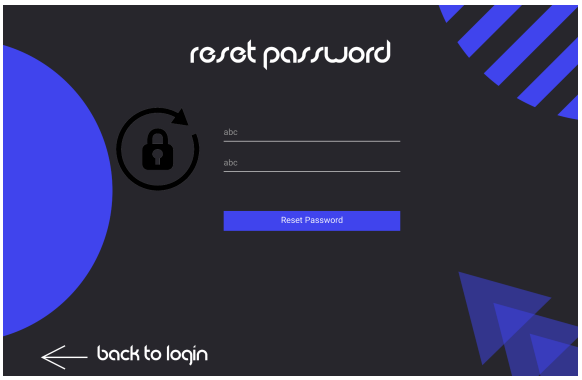


Figure 6.2.44: Interface for <Reset Password (New Password FIlled)>

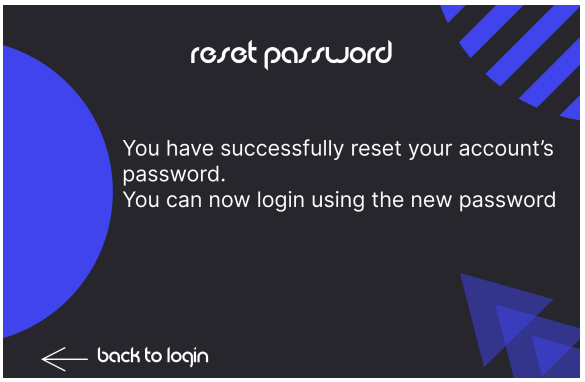


Figure 6.2.45: Interface for <Password Reset Successful>

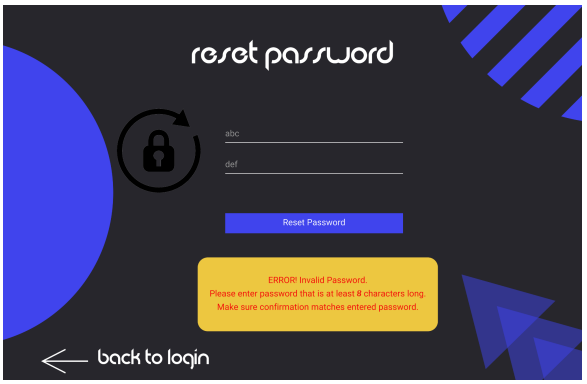


Figure 6.2.46: Interface for <Reset Password (Invalid Password)>

User information Subsystem

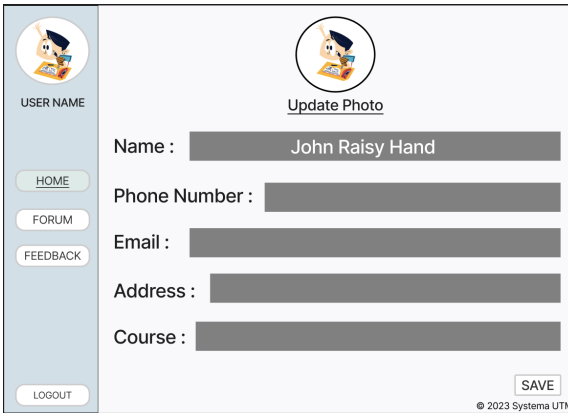


Figure 6.2.47: Interface for <Update User Information(Insert Personal Information)>

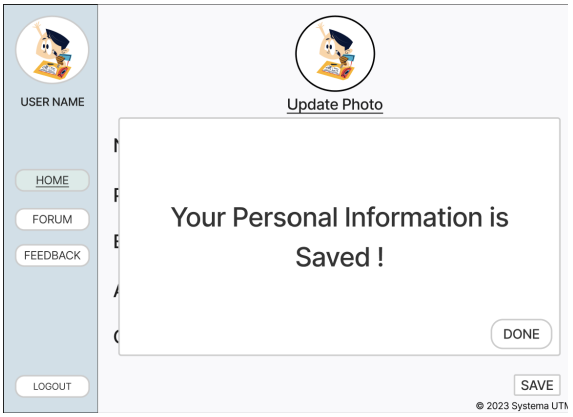


Figure 6.2.48: Interface for <Update User Information(Successfully Updated)>

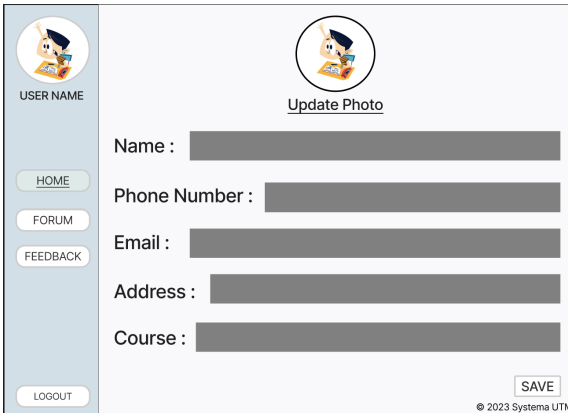


Figure 6.2.49: Interface for <Update User Information (not inserting any informations)>

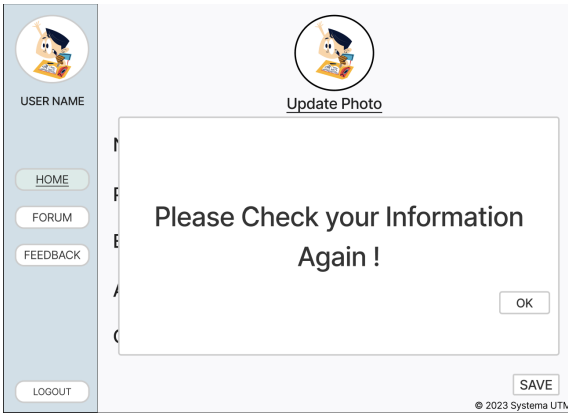


Figure 6.2.50: Interface for <Update Information(information is not complete)>

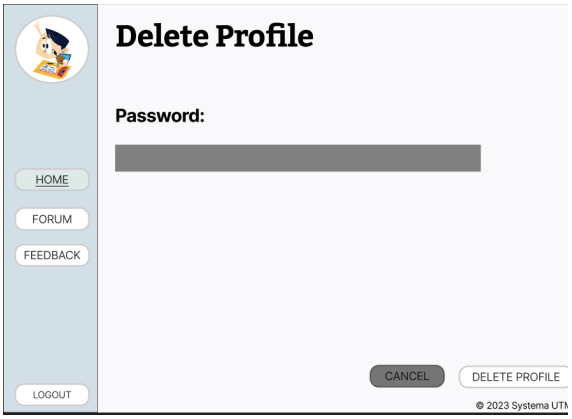


Figure 6.2.51: Interface for <Delete Profile>

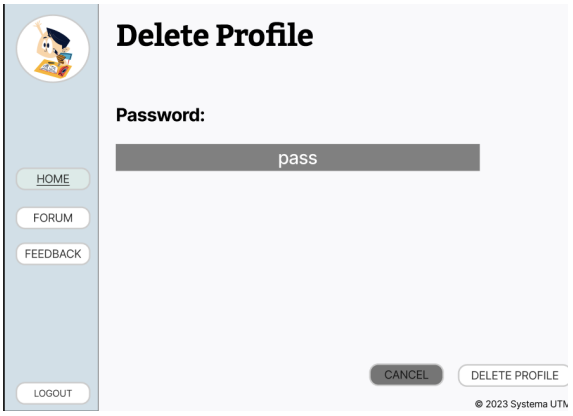


Figure 6.2.52: Interface for <Delete Profile(Enter Password)>

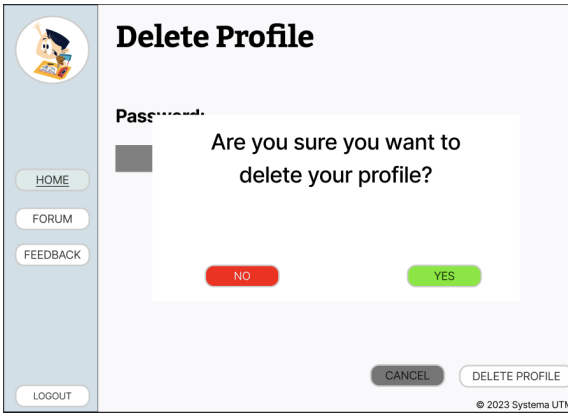


Figure 6.2.53: Interface for <Delete Profile(Confirmation)>

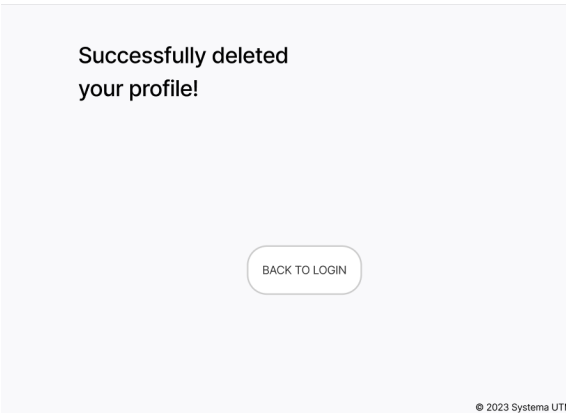


Figure 6.2.54: Interface for <Delete Profile(Profile is Successfully Deleted)>

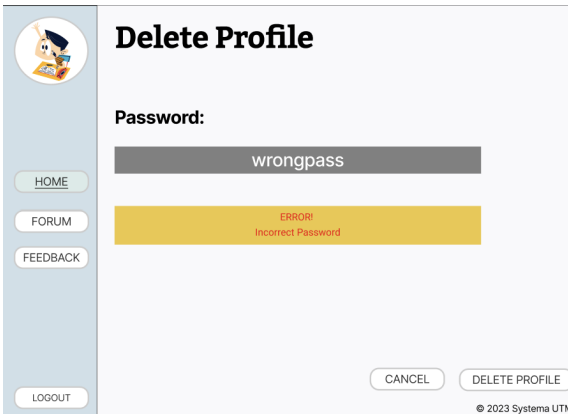


Figure 6.2.55: Interface for <Delete Profile(Entering the Wrong Password)>

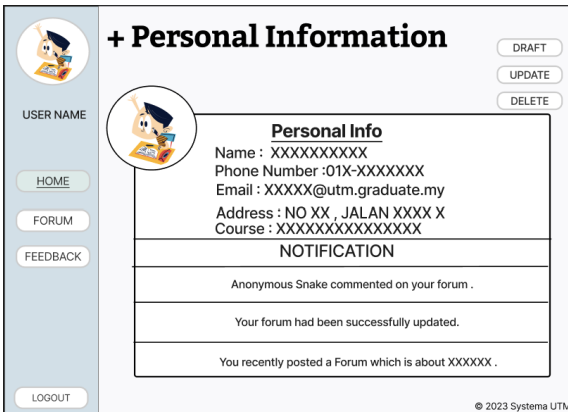


Figure 6.2.56: Interface for <Personal information>

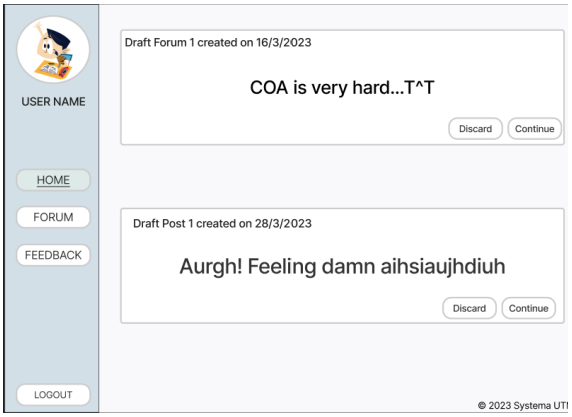


Figure 6.2.57: Interface for <Forum Drafts>

Figure 6.2.58: Interface for <Forum Creation>

Figure 6.2.59: Interface for <Forum Creation(Message of Successfully Created the Forum>

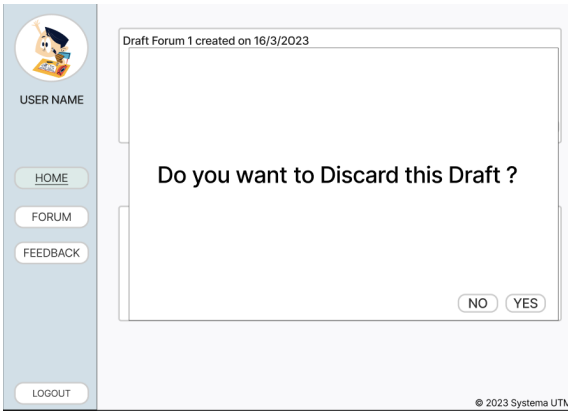


Figure 6.2.60: Interface for <Discard Forum Draft>

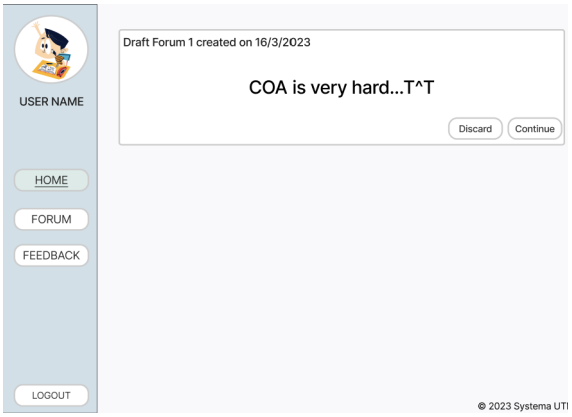


Figure 6.2.61: Interface for <Forum Draft Discarded>

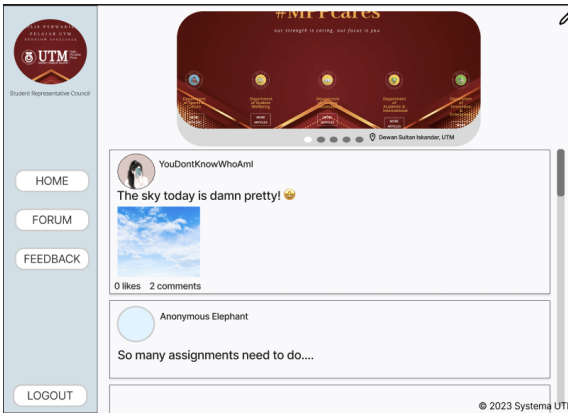


Figure 6.2.62: Interface for <Homepage of SRC>

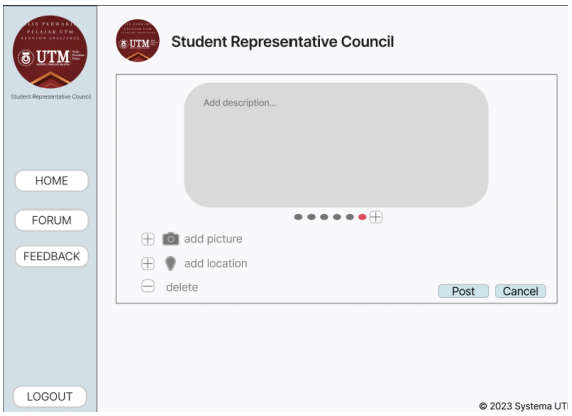


Figure 6.2.63: Interface for <Update SRC Information Board>

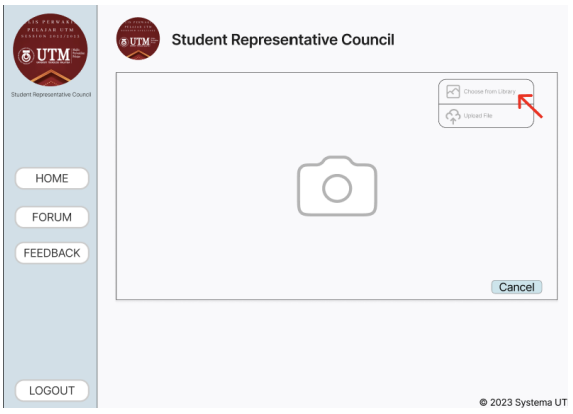


Figure 6.2.64: Interface for <Update SRC Information Board(Choose picture from Computer Photo Library)>

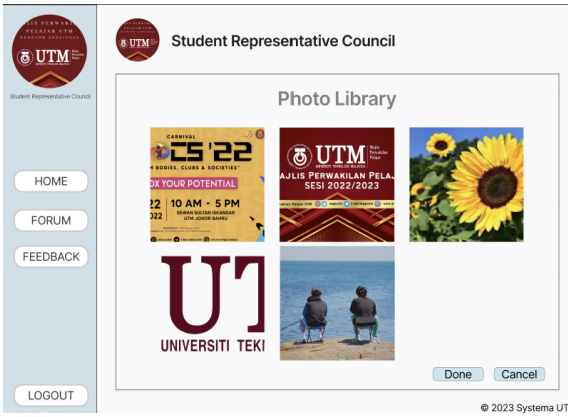


Figure 6.2.65: Interface for <Update SRC Information Board(Computer Photo Library)>

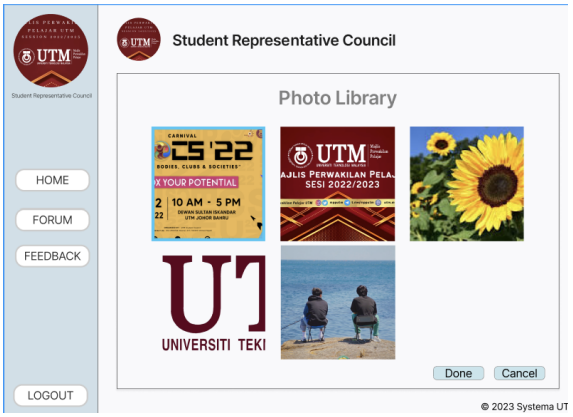


Figure 6.2.66: Interface for <Update SRC Information Board(Choosing Photo)>

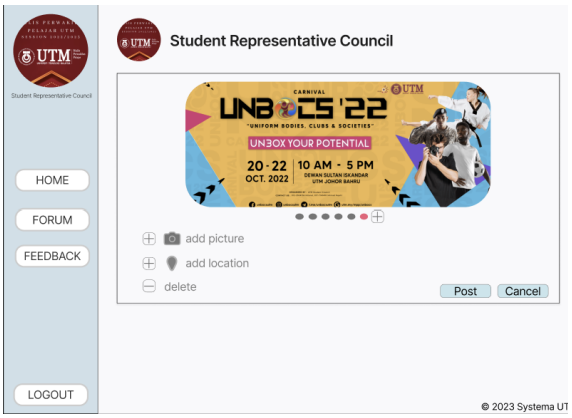


Figure 6.2.67: Interface for <Feedback been successfully submitted>

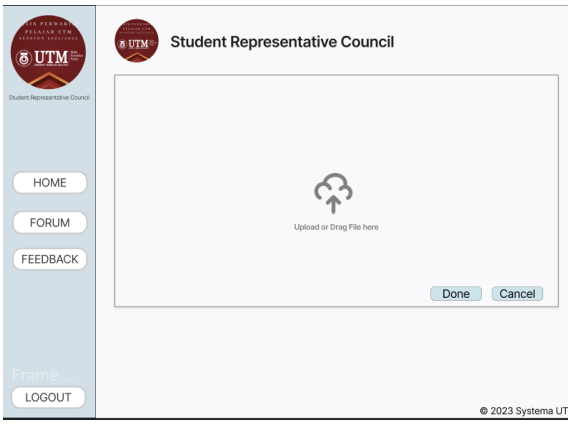


Figure 6.2.68: Interface for <Feedback been successfully submitted>

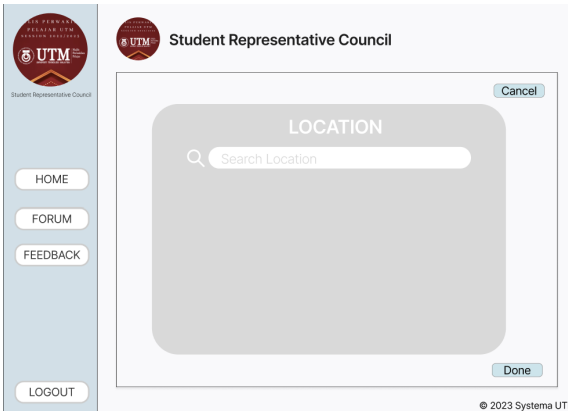


Figure 6.2.69: Interface for <Feedback been successfully submitted>

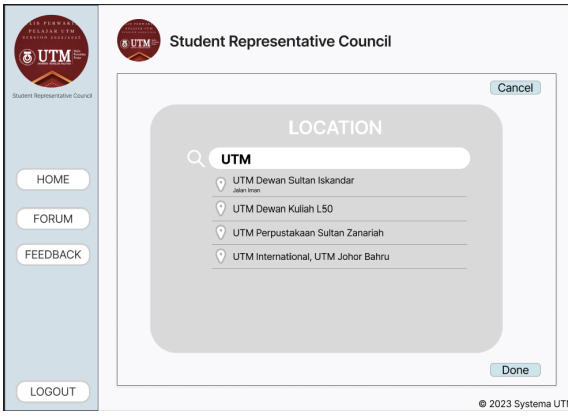


Figure 6.2.70: Interface for <Feedback been successfully submitted>

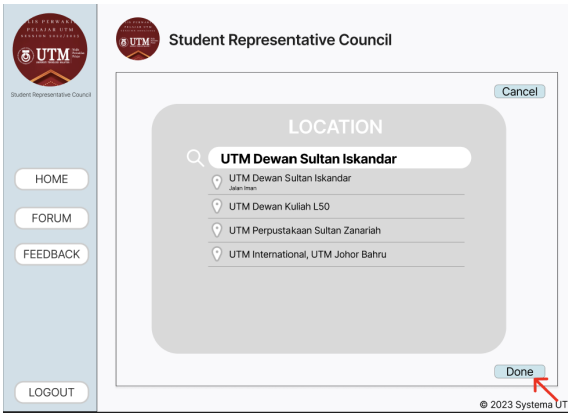


Figure 6.2.71: Interface for <Feedback been successfully submitted>

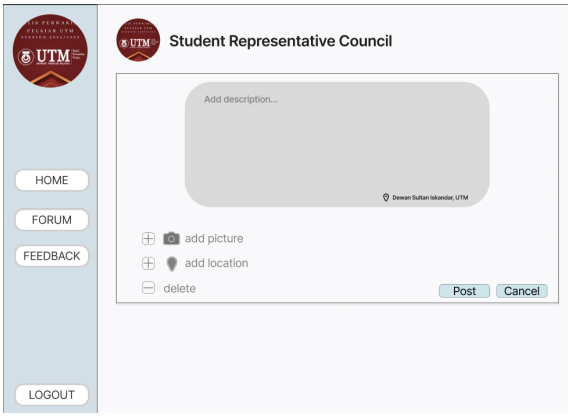


Figure 6.2.72: Interface for <Feedback been successfully submitted>

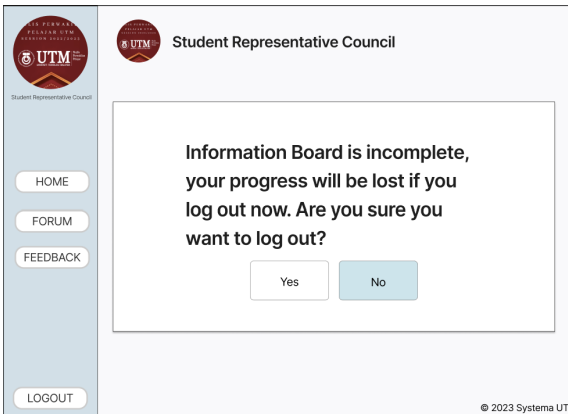


Figure 6.2.73: Interface for <Feedback been successfully submitted>

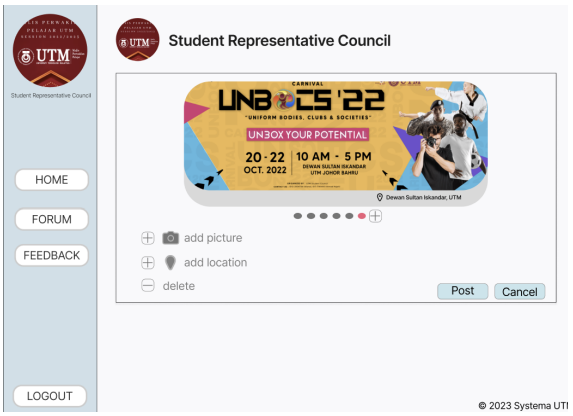


Figure 6.2.74: Interface for <Feedback been successfully submitted>

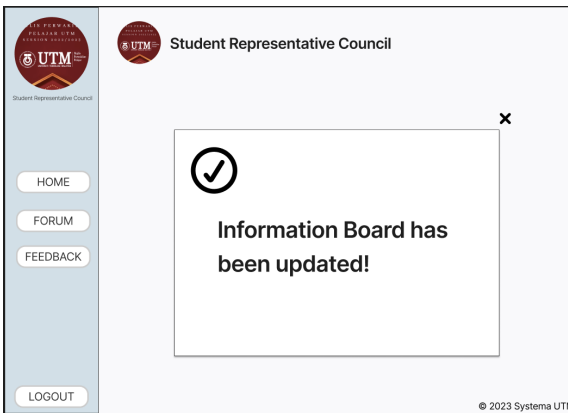


Figure 6.2.75: Interface for <Feedback been successfully submitted>

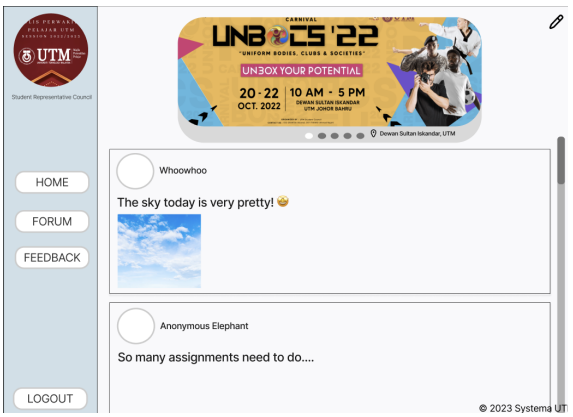


Figure 6.2.76: Interface for <Feedback been successfully submitted>

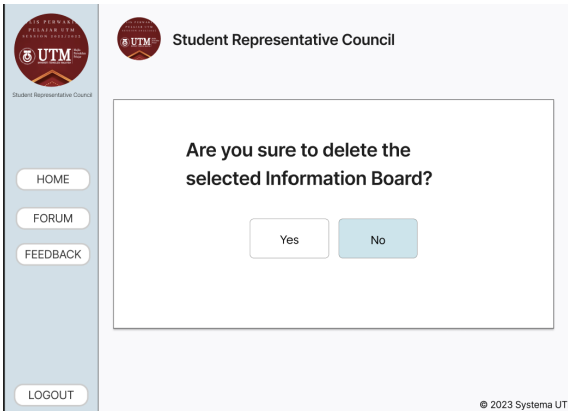


Figure 6.2.77: Interface for <Feedback been successfully submitted>

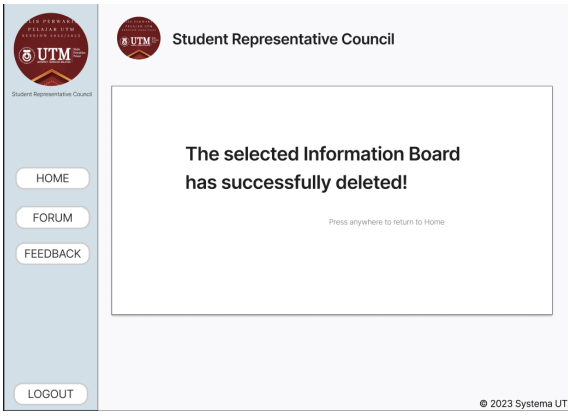


Figure 6.2.78: Interface for <Feedback been successfully submitted>

Processing Subsystem

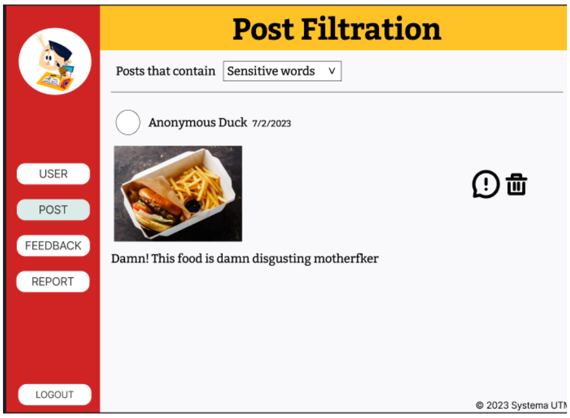


Figure 6.2.79: Interface for <Feedback been successfully submitted>

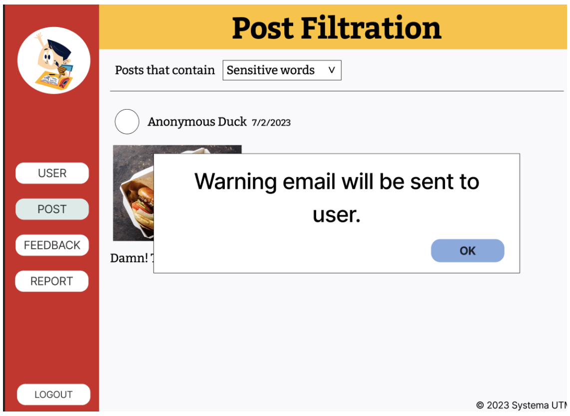


Figure 6.2.80: Interface for <Feedback been successfully submitted>

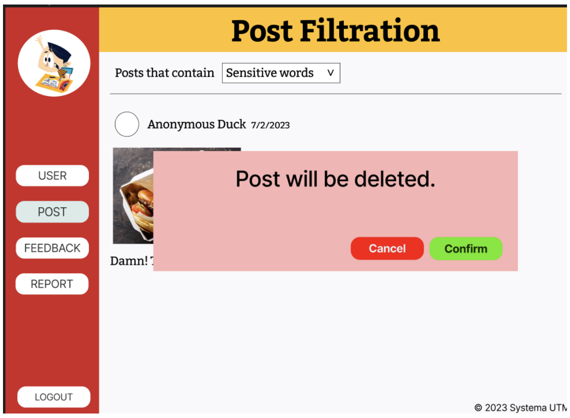


Figure 6.2.81: Interface for <Feedback been successfully submitted>

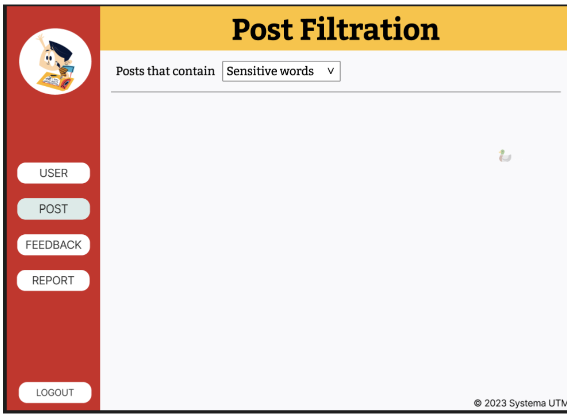


Figure 6.2.82: Interface for <Feedback been successfully submitted>

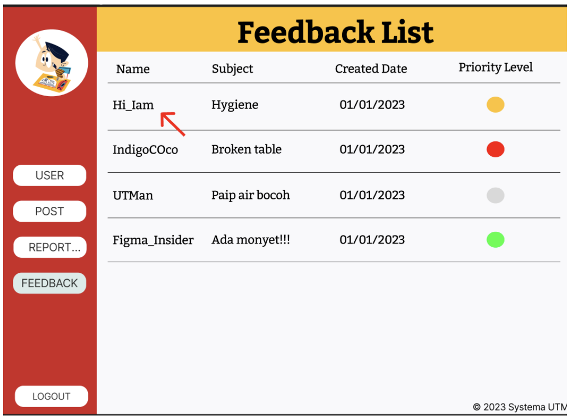


Figure 6.2.83: Interface for <Feedback been successfully submitted>

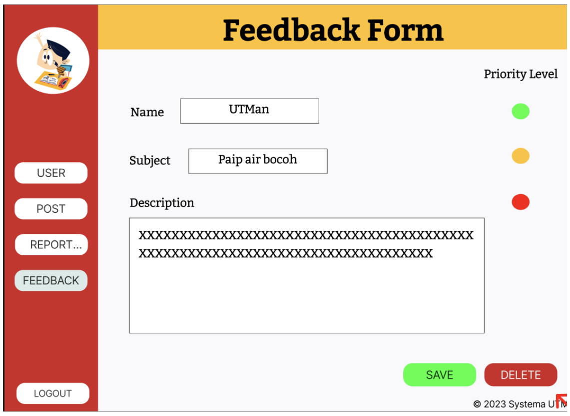


Figure 6.2.84: Interface for <Feedback been successfully submitted>

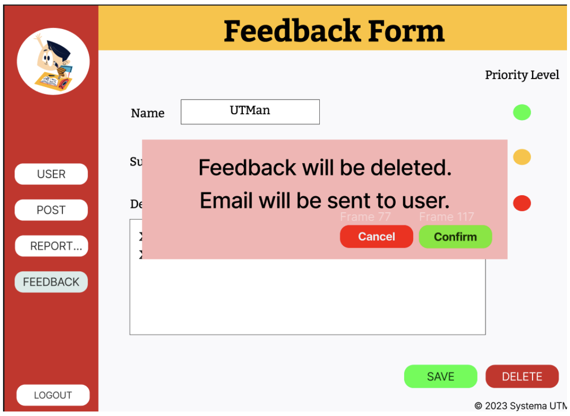


Figure 6.2.85: Interface for <Feedback been successfully submitted>

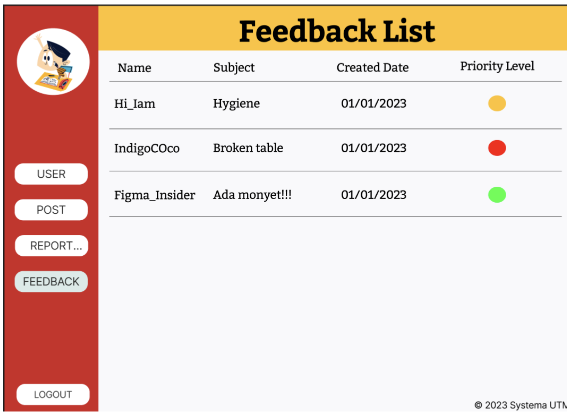


Figure 6.2.86: Interface for <Feedback been successfully submitted>

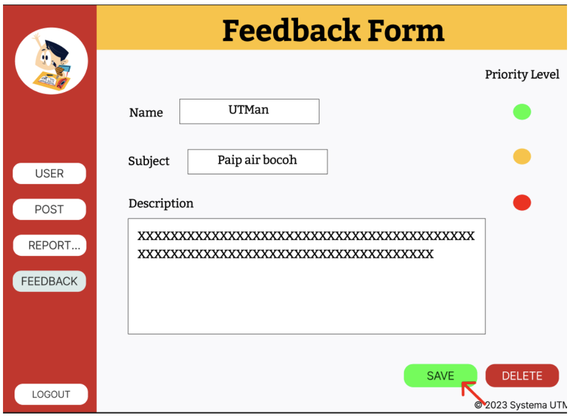


Figure 6.2.87: Interface for <Feedback been successfully submitted>

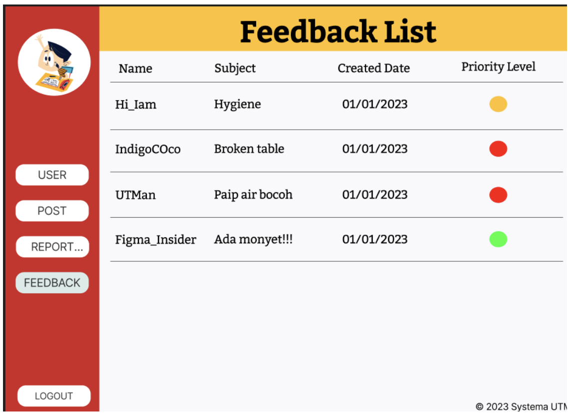


Figure 6.2.88: Interface for <Feedback been successfully submitted>

Admin and Reporting Subsystem

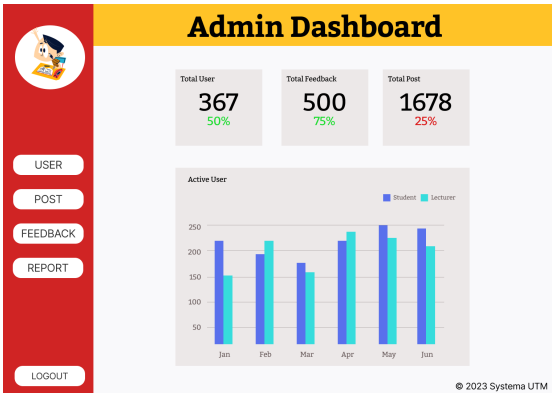


Figure 6.2.88: Interface for <Admin Dashboard>

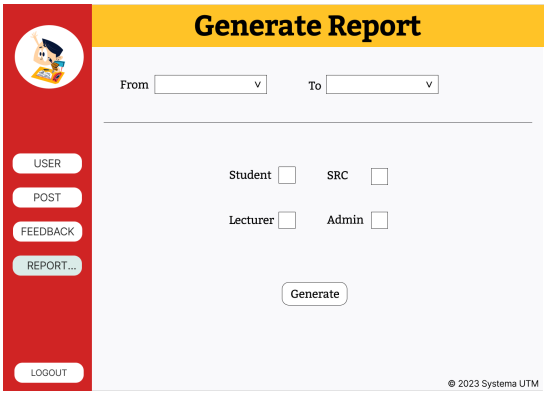


Figure 6.2.89: Interface for <Generate Report>

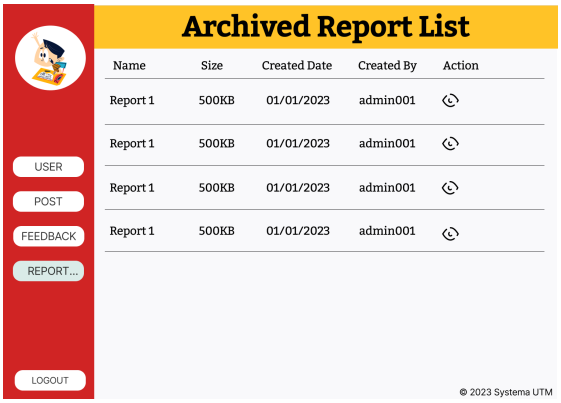


Figure 6.2.90: Interface for <Archived Report List Page>

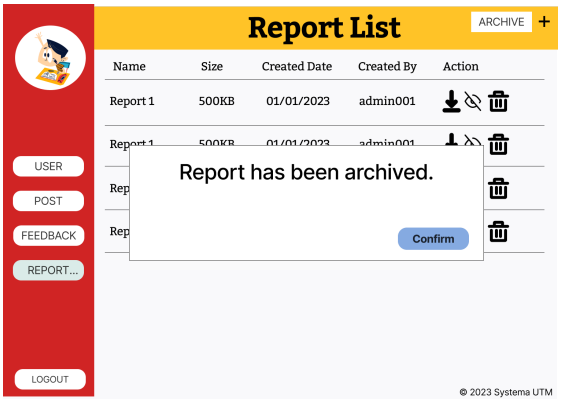


Figure 6.2.91: Interface for <Pop-up Message after Archive Report>

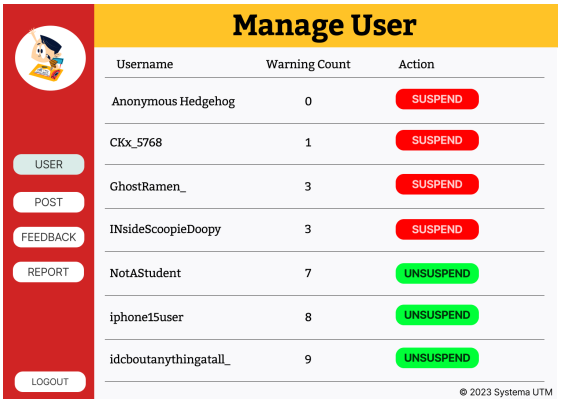


Figure 6.2.92: Interface for <Manage User Page>

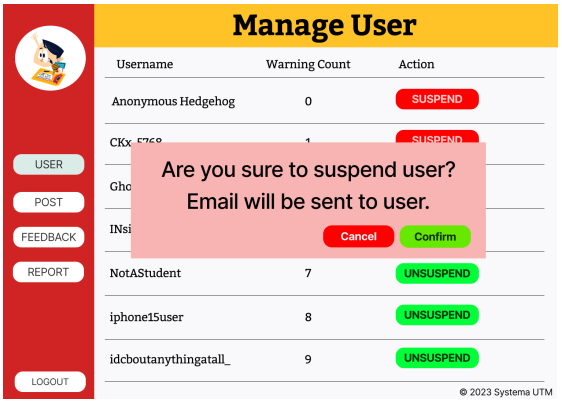


Figure 6.2.93: Interface for <Pop-up Message to Suspend User>

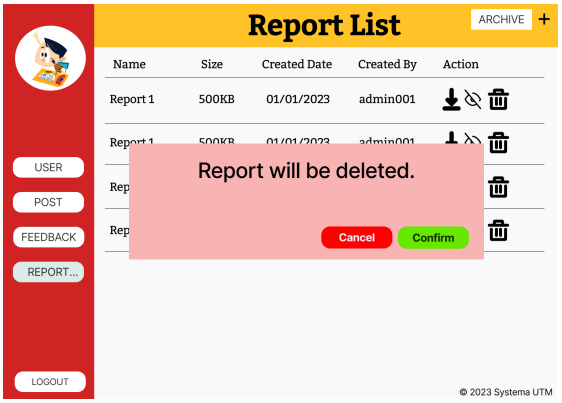


Figure 6.2.94: Interface for <Pop-up Message to Delete Report>

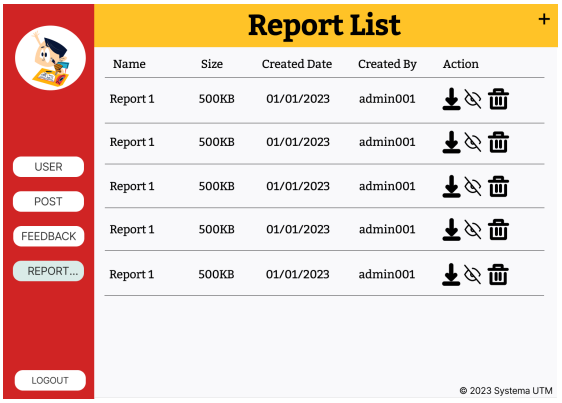


Figure 6.2.95: Interface for <Report List Page>

Personalized Dashboard Subsystem

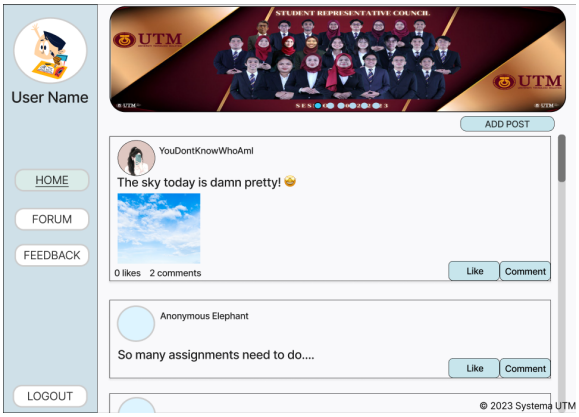


Figure 6.2.96: Interface for <Homepage>

Figure 6.2.97: Interface for <Adding Post>

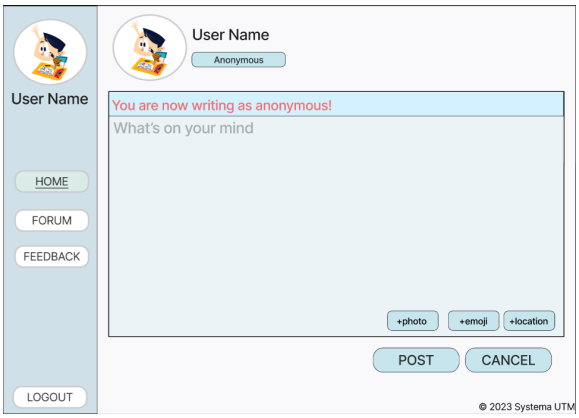


Figure 6.2.98: Interface for <Setting the post to be anonymous>

Figure 6.2.99: Interface for <Filling the post content>

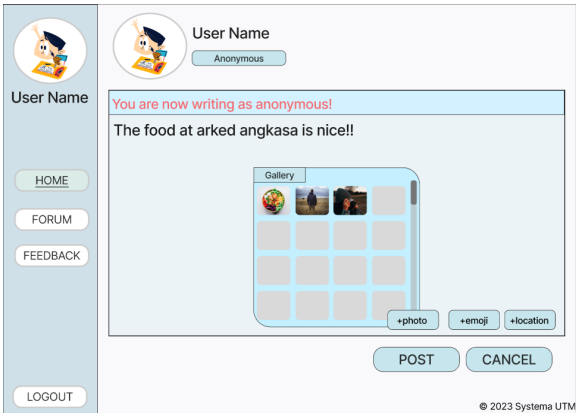


Figure 6.2.100: Interface for <Trying to add photo>

Figure 6.2.101: Interface for <Select desired photo>

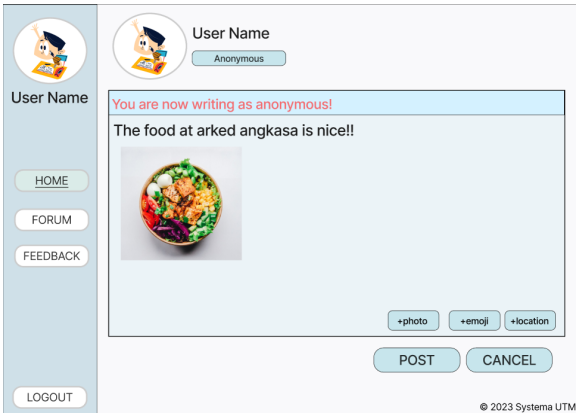


Figure 6.2.102: Interface for <Photo been selected>

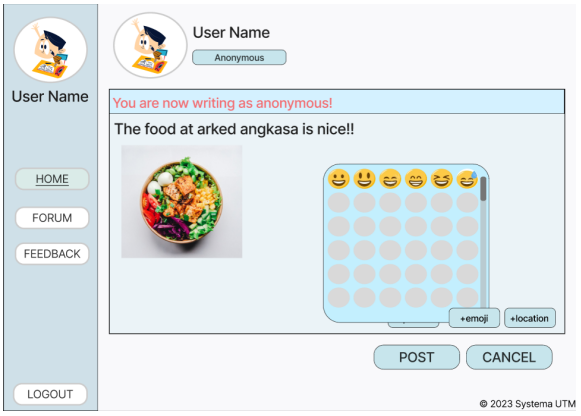


Figure 6.2.103: Interface for <Trying to add emoji>

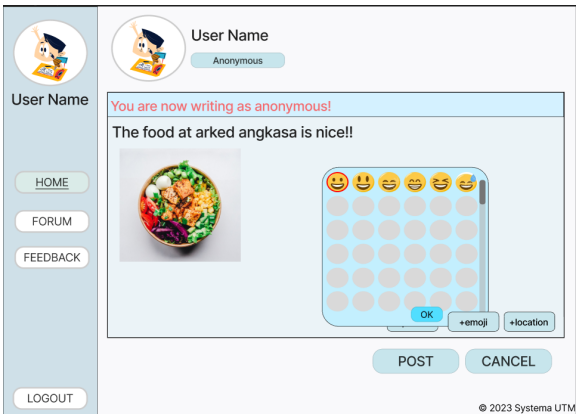


Figure 6.2.104: Interface for <Select desired emoji>

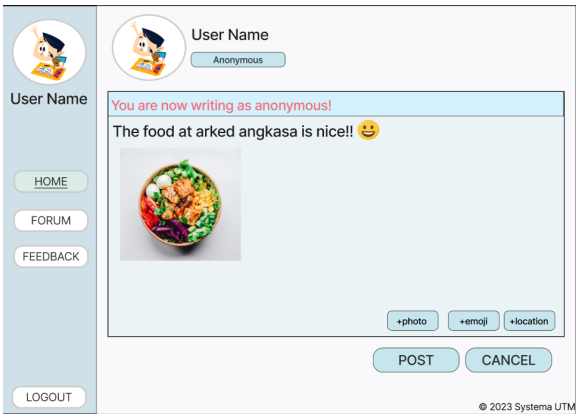


Figure 6.2.105: Interface for <Emoji been selected>

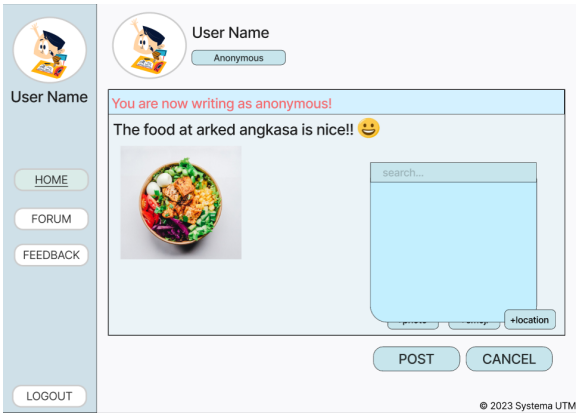


Figure 6.2.106: Interface for <Trying to add location in the post>

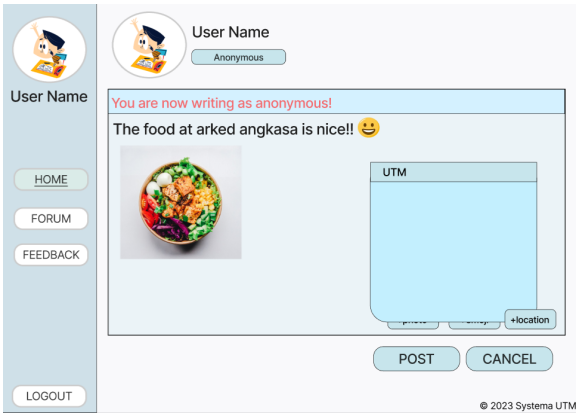


Figure 6.2.107: Interface for <Searching a location>

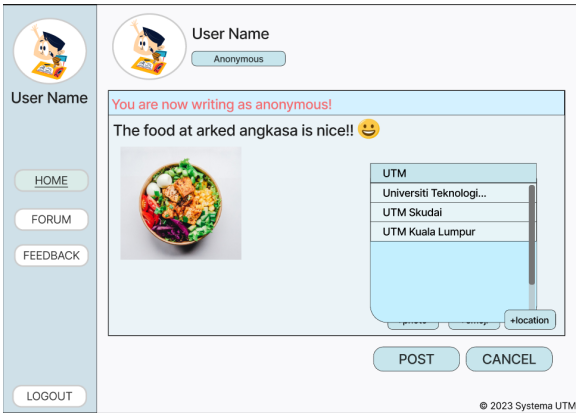


Figure 6.2.108: Interface for <Appearing of location recommendation>

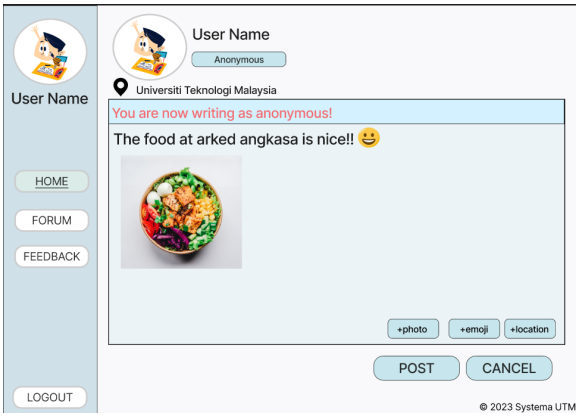


Figure 6.2.109: Interface for <Location been added>

Figure 6.2.110: Interface for <Post been posted>

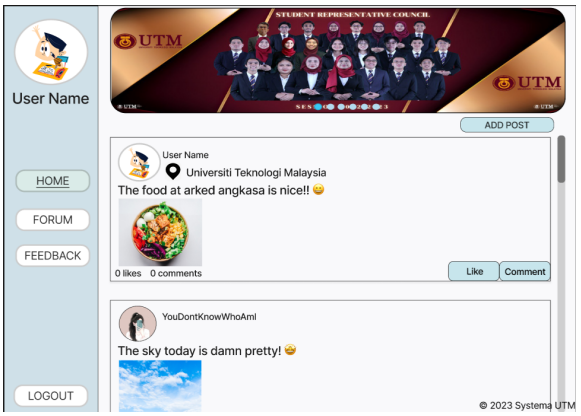


Figure 6.2.111: Interface for <Homepage showing the added post>

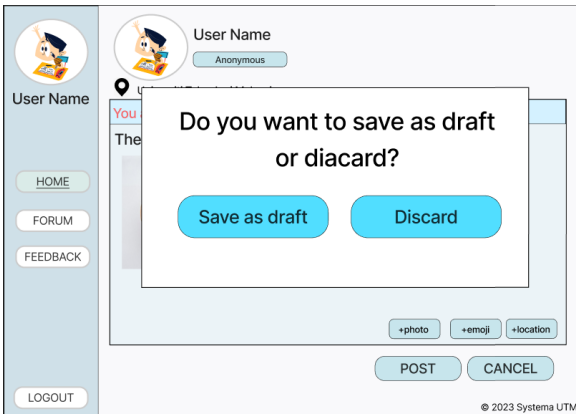


Figure 6.2.112: Interface for <Trying to canceling the post>

Figure 6.2.113: Interface for <Save post as a draft>

Figure 6.2.114: Interface for <Post is discarded>

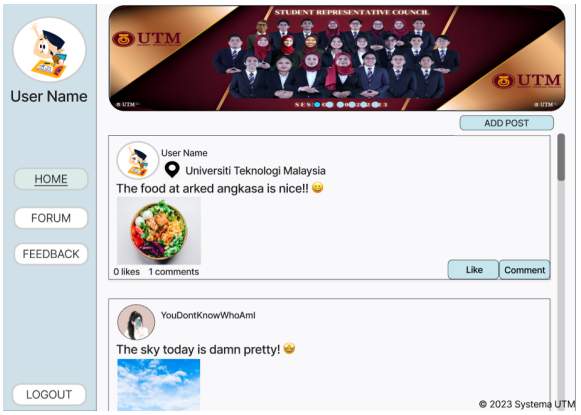


Figure 6.2.115: Interface for <Trying to like a post>

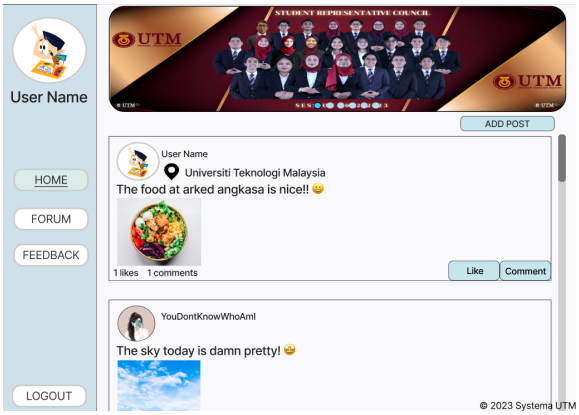


Figure 6.2.116: Interface for <The number of likes increasing>

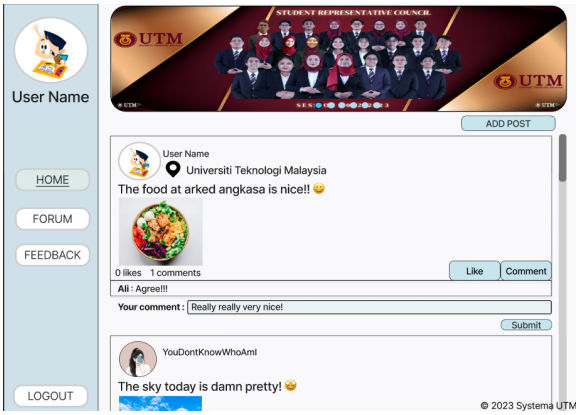


Figure 6.2.117: Interface for <Trying to add a comment>

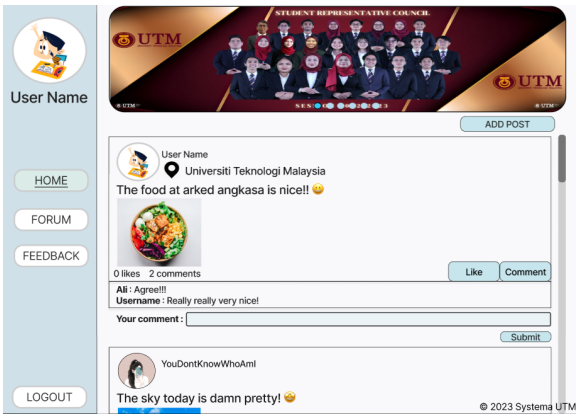


Figure 6.2.118: Interface for <Comment is successfully been added>

Feedback Subsystem

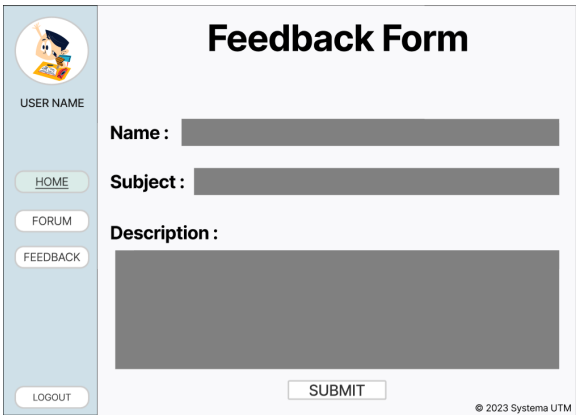


Figure 6.2.119: Interface for <Feedback form page>

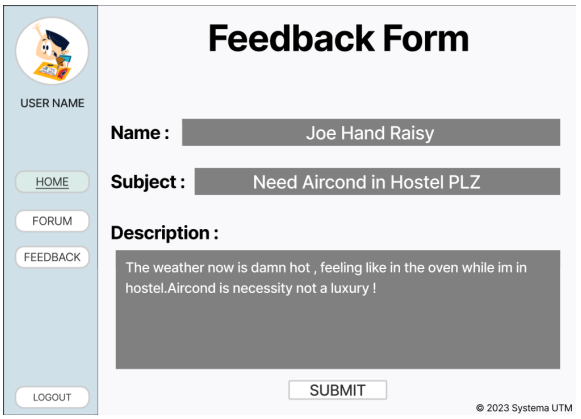


Figure 6.2.120: Interface for <Filling up the forum form>

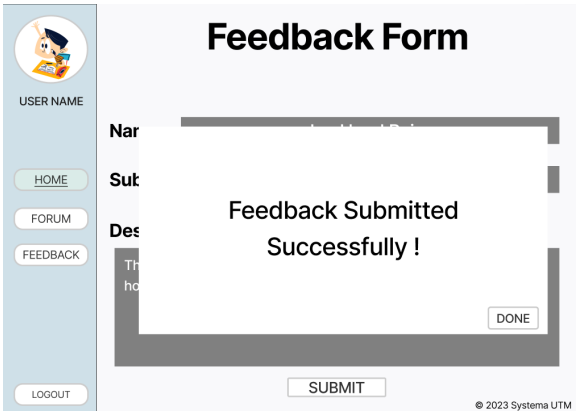


Figure 6.2.121: Interface for <Feedback been successfully submitted>

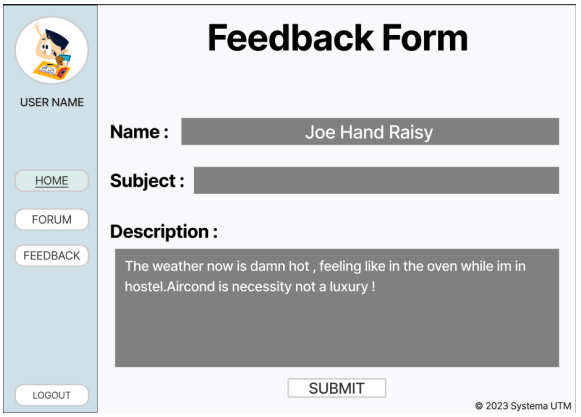


Figure 6.2.122: Interface for <Trying to submit a feedback form without subject>

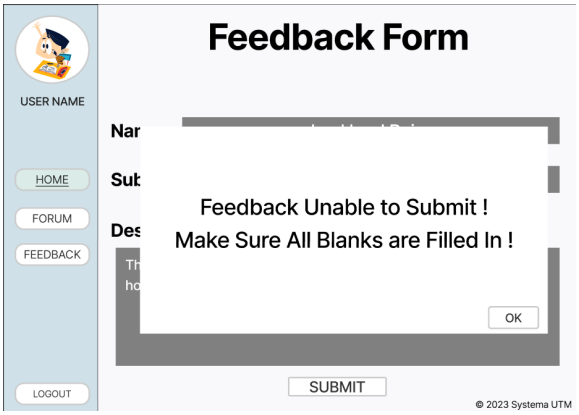


Figure 6.2.123: Interface for <Error message been pop up>

Anonymous Forum Subsystem

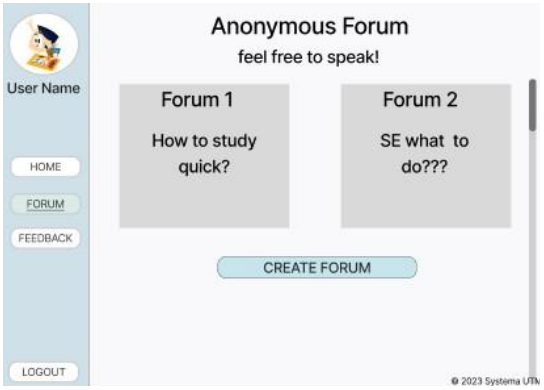


Figure 6.2.124: Interface for <Anonymous Forum>

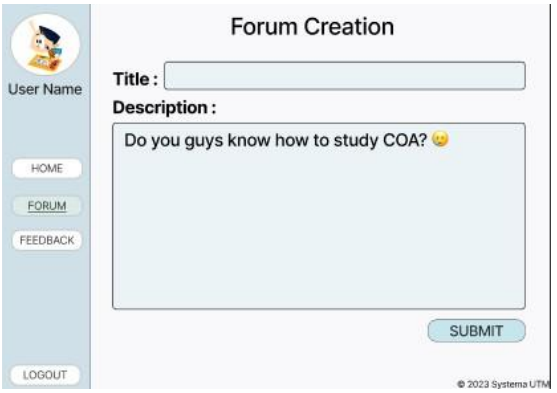


Figure 6.2.125: Interface for <Filling Forum Creation Form without Title>

Figure 6.2.126: Interface for <Pop Out Error Message>

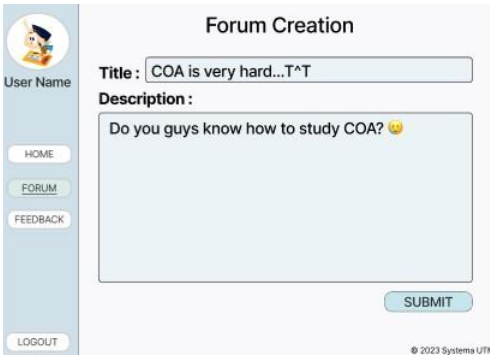


Figure 6.2.127: Interface for <Filling Forum Creation Form>

Figure 6.2.128: Interface for <Pop Out Forum Created Successfully Message>

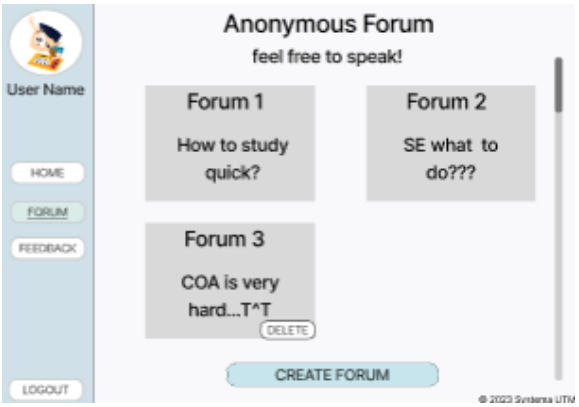


Figure 6.2.129: Interface for <Anonymous Forum Page with New Forum Created>

Figure 6.2.130: Interface for <Pop Out Message about Saving Draft or Discard Options>

Figure 6.2.131: Interface for <Pop Out Message about Forum Successfully Saved>

Figure 6.2.132: Interface for <Pop Out Message about Forum Successfully Discarded>

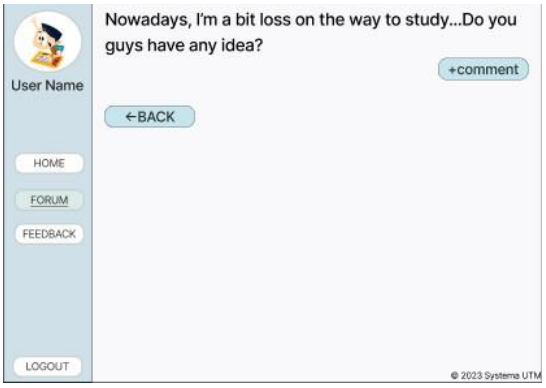


Figure 6.2.133: Interface for <See Details on Other's Forum>

Figure 6.2.134: Interface for <Adding Comments under Forum>

Figure 6.2.135: Interface for <Typing and Adding Comments under Forum>

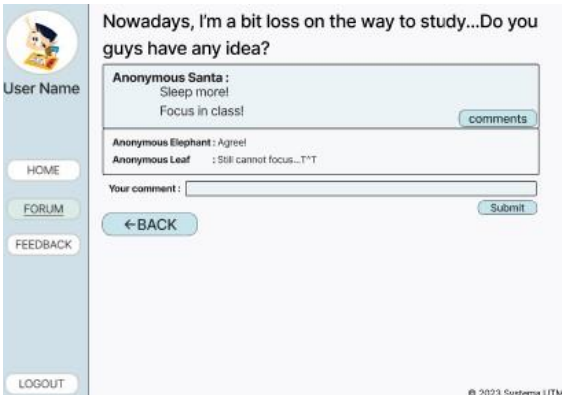


Figure 6.2.136: Interface for <Read Comments under Forum>