University of Dhaka

Department of Computer Science and Engineering

CSE-3112: Software Engineering Lab

Meramot : Comprehensive Tech Services and Supports

Software Design Document (SDD)

Submitted By:

Abdullah Ibne Hanif Arean
Roll: 12

Mahmudul Hasan
Roll: 20

Submitted To :

Dr. Saifuddin Md. Tareeq
Professor And Chairman
Department of Computer Science and Engineering
University of Dhaka

Redwan Ahmed Rizvee
Lecturer
Department of Computer Science and Engineering
University of Dhaka

Submitted On :

April 14, 2023

Contents

1 Introduction

1.1 Purpose of the System

"Meramot: Comprehensive Tech Services and Supports" aims at creating a locally operated, web-based platform that aims to provide a comprehensive and convenient solution for technology-related issues in Bangladesh. It offers an interactive interface for posting and resolving issues, live chat support, on-demand doorstep paid servicing with live tracking and a fully functioning question-answering web app. The system seeks to fill the gap in the market by catering to the needs of both technical and non-technical individuals in Bangladesh and improving the accessibility of Bangla resources. The system addresses the lack of locally operated tech support platforms and the heavy dependence on repair shops in Bangladesh that often have unqualified technicians and charge unnecessary fees to non-technical users. The project's purpose is to simplify and streamline the tech support experience, making it more convenient, reliable, and efficient for all users.

1.2 Scope of the System

The Meramot platform is intended to be an all-encompassing solution that replaces traditional applications with additional, useful features such as collaboration, advanced search and categorization capabilities, authorization controls, and more.

1.3 Objective and Success Criteria of the Project

- Increase the accessibility of Bangla resources for non-technical individuals, through the provision of the platform in both Bangla and English.
- Increase the uptake of the platform, with an increase in the number of users posting problems and engaging with the question-answering web app.
- Reduce the turnaround time for resolving technology-related issues reported on the platform.
- More users are taking paid live chat support and on-demand doorstep paid-to-service.
- Simplify and streamline the tech support experience, making it more convenient, reliable, and efficient.

1.4 Overview

Meramot is a locally operated, comprehensive web-based platform that offers one-stop solutions for all software and hardware-related issues in Bangladesh, with interactive interfaces for posting and resolving issues, and on-demand doorstep servicing with live tracking, and live chat.

1.5 Reference Material

[1] L. Chart, *Uml with lucid chart*, https://lucid.app/, [Online; accessed 21-February-2023], 2023.

1.6 Definitions, Acronyms and Abbreviations

- RAD = Requirement Analysis Document
- NPM = Node Package Manager
- WWW = World Wide Web
- URL = Uniform Resource Locator
- REST = Representational State Transfer
- JSON = JavaScript Object Notation
- UML = Unified Modeling Language, It is a standardized language used for modeling software systems and designing software architectures. UML provides a common set of notations, diagrams, and methods for describing and visualizing software systems and their components, helping to improve communication and understanding among development teams, stakeholders, and users.

2 System Description

"Meramot: Comprehensive Tech Services and Supports" is a web-based platform that aims to provide users with a collaborative problem-solving environment. This product can be thought of as the replacement for certain applications that provide a lot of similar functionalities. However, it has additional useful functionalities compared to the existing alternatives such as collaboration, advanced search and categorization features, authorization, etc. In essence, Meramot provides a one-stop solution for technical problem-solving by connecting users with technical experts. The platform is designed to be easy-to-use, efficient, and provide a personalized experience to all its users. System architecture in details as follows:

1. The system will provide a locally operated, web-based platform for comprehensive tech services and support in Bangladesh, addressing the current lack of such platforms and providing a solution to common technology-related issues.
2. The platform will be available in both Bangla and English, ensuring accessibility for non-technical individuals and increasing the availability of Bangla resources for those who may not be fluent in English.
3. The system will offer an interactive interface for posting and resolving issues.
4. On-demand doorstep/shop paid servicing with live tracking, and live chat support to provide users with comprehensive, convenient, and efficient tech support.
5. The system will allow users to post technology-related questions, issues, or problems they are facing in an interactive and user-friendly interface.
6. The interface will include fields for tags, titles, descriptions, and screenshots if necessary, providing users with an easy way to accurately describe their issue and provide necessary details.
7. A search bar will be available on the interface, enabling users to look for previously posted questions and their answers, potentially providing immediate solutions without needing to repost a question.

8. Other users and experts on the platform can collaborate to solve the issue by responding with potential solutions or asking for further clarification, providing a community approach to resolving tech problems.

9. Once the issue is resolved, the person who posted the question can mark it as resolved or choose the best answer, which provides a clear indication to others who may have similar issues.

10. The intuitive interface of the platform will enable users to navigate through the system with ease and speed, allowing them to find the information they need quickly and efficiently.

3 Design Overview

3.1 Design Rationale

Regarding the front-end development, the React library has been utilized as it provides a quicker and more efficient development cycle while providing all the essential components required for applications that do not require handling of intricate APIs of the host device operating system. This concept eradicates the necessity of employing native frameworks or codebases. React Redux provides data persistence which is very important for our application. The combination of React and JavaScript also facilitates expansion to other platforms since they are oriented towards cross-platform development right from the beginning.

On the backend, we're using Spring Boot framework to create the server. Spring Boot is a Java-based framework that provides an opinionated approach to building web applications. It is designed to simplify the development process by providing a set of pre-configured dependencies and auto-configuration options. The main design rationale behind Spring Boot is to help developers build production-ready applications quickly and easily. It does this by reducing the time and effort required to set up and configure an application from scratch. Spring Boot comes with a pre-configured Tomcat server, which means that developers can create a web application and run it immediately without having to set up an application server. Another important aspect of Spring Boot's design is its focus on convention over configuration. Spring Boot provides sensible defaults for many configuration options, which means that developers can focus on writing business logic instead of worrying about configuration details. This approach also helps to reduce the amount of boilerplate code required in an application.

We're using a combination of Firebase and Spring security for role based authentication and authorization. By combining Firebase Authentication and Spring Security, we can create a robust and scalable authentication and authorization system for our application. Firebase provides a range of authentication options and user management features, while Spring Security provides powerful authorization capabilities and integration with Spring-based web applications.

Taking all of these considerations into account, it can be inferred that the most suitable architectural pattern for the application is the Model-View-Controller (MVC) pattern. This pattern effectively separates the business logic from the presentation logic, thereby improving the application's maintainability and flexibility. Notably, the application does not depend heavily on a centralized server for its functionality, apart from user data authentication and synchronization, which are not the primary features. Therefore, the server-client pattern is not deemed appropriate. Likewise, the peer-to-peer architecture is

also unsuitable due to the high-risk nature of relying on other peers for critical services like authentication. Additionally, decentralization is not a strict requirement for the application at hand.

3.2 System Architecture

As mentioned earlier, the chosen architectural pattern for the application is the Model-View-Controller (MVC) pattern. This pattern segregates the application logic into three components: the Model, the View, and the Controller.

The Model classes are responsible for managing the application's data sources, including the Firebase database objects and the local database objects. The Model is the lowest-level component, responsible for maintaining data and connecting to the database. It responds to requests from the Controller and never communicates with the View directly. For example, if the Controller requests some data from the database, the Model fetches it and returns it to the Controller.

The Controller classes serve as a bridge between the Models and the Views. The Controller does not handle the data logic, but rather relies on the Model to do so. Once the Model fetches the required data, the Controller invokes methods in the View to display the data dynamically. The Controller is responsible for handling user input and translating it into commands for the Model and the View.

The View component generates the user interface that the user interacts with. Views are created based on the data collected by the Model, but they don't communicate directly with the Model. Instead, the View communicates with the Controller, which then interacts with the Model. In the case of this application, the majority of the built-in classes of the Flutter framework and their extensions will be the View classes.

To summarize, the MVC pattern is an effective way of decoupling the business logic, data management, and presentation logic of an application. This improves the application's maintainability and flexibility, and allows for easier testing and modification of individual components.

3.3 Assumption and Dependencies

3.3.1 Assumptions

- The device that will run the program must fulfill the minimal hardware requirements.
- To use the program without interruptions, the user must have access to a reliable internet connection.
- There will be no unexpected difficulties with the application's backend, which might cause substantial delays or even failure in development.
- There will be no schedule conflicts or time limits during the development process, which might adversely impact the project's timeframe.
- The user will have basic computer literacy and be able to navigate and use the platform's features effectively.
- Users will be able to accurately and clearly articulate their questions and problems, which will allow other users to provide helpful and relevant answers.

3.3.2 Dependencies

Front-end dependencies (React JS):

- react
- react-dom
- react-scripts
- axios (for HTTP requests)
- react-router-dom (for routing)
- material-ui (for UI components)
- material-ui/icons (for icons)
- styled-components (for CSS-in-JS)

Back-end dependencies (Spring Boot):

- spring-boot-starter-web (for building web applications)
- spring-boot-starter-security (for securing the application)
- spring-boot-starter-data-jpa (for working with databases using JPA)
- spring-boot-starter-validation (for input validation)
- spring-boot-starter-test (for testing the application)
- spring-boot-devtools (for development-time tools)
- postgresql (for connecting to a PostgreSQL database)
- lombok (for reducing boilerplate code)

These dependencies can be added to the application's build configuration files (such as package.json for React and pom.xml for Spring Boot) and installed using a package manager like npm or Maven.

4 Object Model

4.1 Object Description

4.1.1 User <<entity>>

Class Name	**User**
Brief Description	Object representing the user's information
uid	Unique identifier for the user
role	Role Authorization for the user
name	Name of the user
email	email address of the user
description	Optional description provided by the user
posts	List of posts that the user has created
comments	List of comments that the user did
reacts	List of reacts that the user did
searches	List of searches that the user did
tags	List of tags that the user used
Methods	**Description**
addPost(post)	Add a post to the user's list of posts
removePost(post)	Remove a post from the user's list of posts
addComment(comment)	Add a comment to the user's list of comments
removeComment(comment)	Remove a comment from the user's list of comments
addReact(react)	Add a react to the user's list of reacts
removeReact(react)	Remove a react from the user's list of reacts
addSearch(search)	Add a search to the user's list of searches
removeSearch(search)	Remove a search from the user's list of searches
addTag(tag)	Add a tag to the user's list of tags
removeTag(tag)	Remove a tag from the user's list of tags
getPosts()	Get a list of all posts created by the user
getComments()	Get a list of all comments made by the user
getReacts()	Get a list of all reacts made by the user
getSearches()	Get a list of all searches made by the user
getTags()	Get a list of all tags used by the user

4.1.2 Post <<entity>>

Class Name	**Post**
Brief Description	Object representing a post made by a user
id	Unique identifier for the post
user	User object representing the user who made the post
title	Title of the post
body	Body content of the post
timestamp	Date and time the post was created
comments	List of Comment objects representing comments made on the post
reacts	List of React objects representing reactions to the post
Methods	**Description**
addComment(comment)	Add a Comment object to the list of comments on the post
removecomment(comment)	Remove a Comment object from the list of comments on the post
addReact(react)	Add a React object to the list of reacts on the post
removeReact(react)	Remove a React object from the list of reacts on the post
getUser()	Get the User object representing the user who made the post

4.1.3 Comment <<entity>>

Class Name	**Comment**
Brief Description	Object representing a comment made by a user on a post
id	Unique identifier for the comment
user	User object representing the user who made the comment
post	Post object representing the post the comment was made on
body	Body content of the comment
timestamp	Date and time the comment was created
reacts	List of React objects representing reactions to the comment
Methods	**Description**
addReact(react)	Add a React object to the list of reacts on the comment
removeReact(react)	Remove a React object from the list of reacts on the comment
getUser()	Get the User object representing the user who made the comment
getPost()	Get the Post object representing the post the comment was made on

4.1.4 AuthenticationController <<controller>>

Class Name	**FirebaseAuthentication**
Brief Description	Object representing Firebase Authentication
auth	The Firebase Authentication instance
user	The currently authenticated user object
Methods	**Description**
signUpWithEmail(email: Email, password: string)	Signs up a new user with the provided email address and password
signInWithEmail(email: Email, password: string)	Signs in a user with the provided email address and password
signInWithGoogle()	Signs in a user with their Google account
signInWithFacebook()	Signs in a user with their Facebook account
signOut()	Signs out the current user
resetPassword(email: Email)	Sends a password reset email to the provided email address
verifyEmail()	Sends an email verification email to the current user's email address
getCurrentUser()	Returns the currently authenticated user object
isAuthenticated()	Returns true if a user is currently authenticated, false otherwise

4.1.5 DatabaseController <<controller>>

Class Name	**Database Connection**
Brief Description	Object representing a connection to a database
host	Host name of the database server
port	Port number on which the database server is listening
database	Name of the database to connect to
user	Username to use for authentication
password	Password to use for authentication
Methods	**Description**
connect()	Establishes a connection to the database
disconnect()	Closes the connection to the database
execute(query)	Executes a SQL query on the connected database
fetchall()	Returns all rows from the most recent query as a list of tuples
fetchone()	Returns the next row from the most recent query as a tuple
commit()	Commits the current transaction to the database
rollback()	Rolls back the current transaction

4.1.6 GuestUserController <<controller>>

Class Name	**GuestUserController**
Brief Description	Controller object handling all functionalities related to non-registered and unauthorized users.
Methods	**Description**
accessLandingPage()	Allows non-registered users to access the landing page
viewPosts()	Allows non-registered users to view posts
viewComments()	Allows non-registered users to view comments
searchProblems()	Allows non-registered users to search for problems and solutions
accessSignUp()	Allows non-registered users to access the Sign-Up option

4.1.7 AuthenticationController <<controller>>

Class Name	**AuthenticationController**
Brief Description	Controller object handling user authentication
Methods	**Description**
userLogin(email, password)	Allows registered users to log in to their accounts using their email address and password

4.1.8 UserRegistrationController <<controller>>

Class Name	**UserRegistrationController**
Brief Description	Controller object handling user registration
Methods	**Description**
userRegistration(email, password, name)	allows users to register using their email address and password and name. They can also register using their google account, gitHub account or facebook account through Firebase

4.1.9 PostController <<controller>>

Class Name	**PostController**
Brief Description	Controller object handling all functionalities related to user posts
Methods	**Description**
createPost(user, description, images)	Enables registered users to post their technical problems on the platform's home page, including a description and images of the problem
displayPosts()	Displays posts on the timeline of other registered users who are viewing the home page
suggestSolution(post)	Provides instant suggestions for the problem posted
markResolved(post)	Allows the user who posted the problem to mark it as resolved if they can solve it

4.1.10 CommentController <<controller>>

Class Name	**CommentController**
Brief Description	Controller object handling all functionalities related to commenting on user posts
Methods	**Description**
addComment(post, user, comment)	Enables users, service providers, technical experts, and administrators to comment on user posts, offering potential solutions and providing assistance

4.1.11 ChatController <<controller>>

Class Name	**Chat**
Brief Description	Object representing a chat conversation between two or more users
id	Unique identifier for the chat
participants	List of User objects representing the users participating in the chat
messages	List of Message objects representing messages sent in the chat
createdAt	Date and time the chat was created
Methods	**Description**
addParticipant(user)	Add a User object to the list of participants in the chat
removeParticipant(user)	Remove a User object from the list of participants in the chat
addMessage(message)	Add a Message object to the list of messages sent in the chat
getMessages()	Get the list of Message objects sent in the chat
getParticipants()	Get the list of User objects participating in the chat

4.1.12 AdminController <<controller>>

Class Name	**Admin**
Brief Description	Object representing an administrator with privileges to manage the system
id	Unique identifier for the admin
name	Name of the admin
email	Email address of the admin
password	Password for the admin account
Methods	**Description**
login()	Authenticate the admin and log in to the system
logout()	Log out of the system
createAccount(user)	Create a new User object representing a user account in the system
deleteAccount(user)	Delete a User object representing a user account from the system
createPost(post)	Create a new Post object representing a post made by a user
deletePost(post)	Delete a Post object representing a post made by a user

4.1.13 UserController <<controller>>

Class Name	**Admin**
Brief Description	Object representing an administrator with privileges to manage the system
id	Unique identifier for the admin
name	Name of the admin
email	Email address of the admin
password	Password for the admin account
Methods	**Description**
login()	Authenticate the admin and log in to the system
logout()	Log out of the system
createPost(post)	Create a new Post object representing a post made by a user
deletePost(post)	Delete a Post object representing a post made by a user

4.1.14 TechnicalExpertController <<controller>>

Class Name	**TechnicalExpert**
Brief Description	Object representing a technical expert with privileges to reply to the users questions, comment on their post, upvote and downvote any solution
id	Unique identifier for the Technical Expert
name	Name of the Technical Expert
email	Email address of the Technical Expert
Methods	**Description**
receiveMessage()	A technical expart can receive message from a user who is looking for solution.
sendMessage()	A technical expart can send message to a user who is looking for solution. But the user must send message first, otherwise the expert can't send message t anyone
comment()	Can comment to any post for providing solution.
vote()	Can upvote or downvote a post or any comment.

4.1.15 SearchController <<entity>>

Class Name	**Search**
Brief Description	Object representing the search functionality of the system
Methods	**Description**
searchPosts(keyword)	Method to search for Post objects containing the given keyword in their title or body
searchUsers(keyword)	Method to search for User objects with the given keyword in their username or display name
searchComments(keyword)	Method to search for Comment objects containing the given keyword in their text

4.2 Object Collaboration Diagram

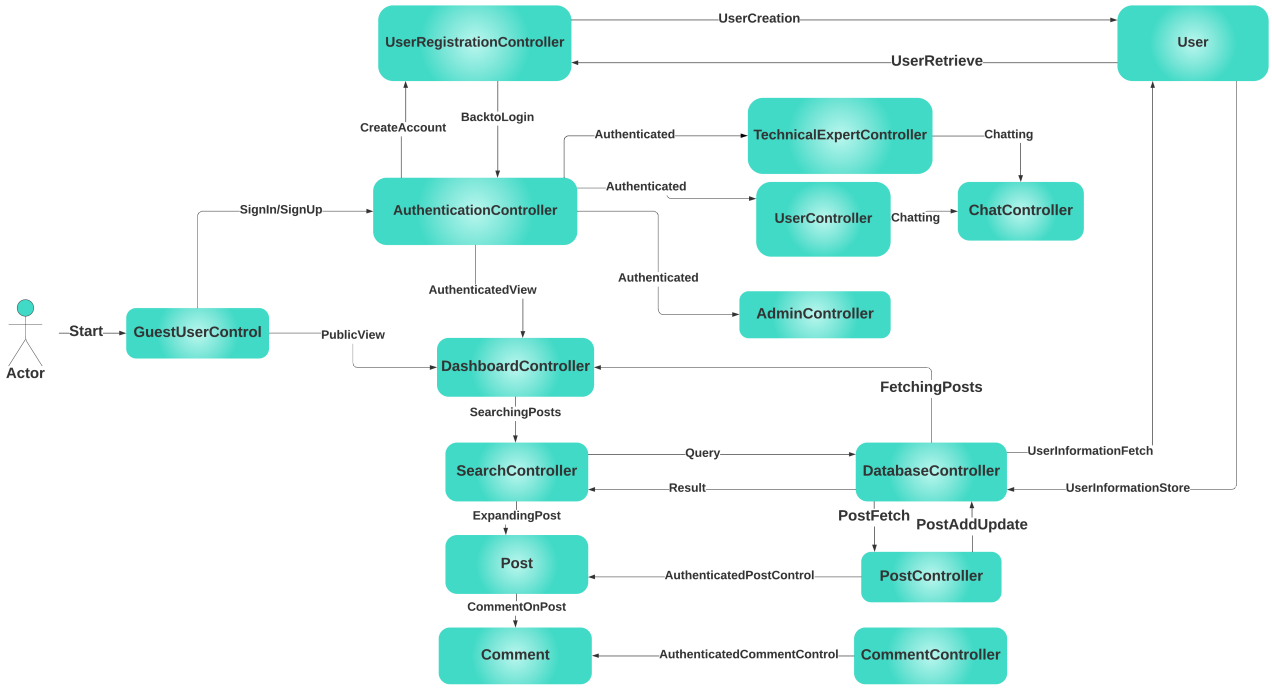

Figure 1: Object Collaboration Diagram

5 Subsystem Decomposition

5.1 Complete Package Diagram

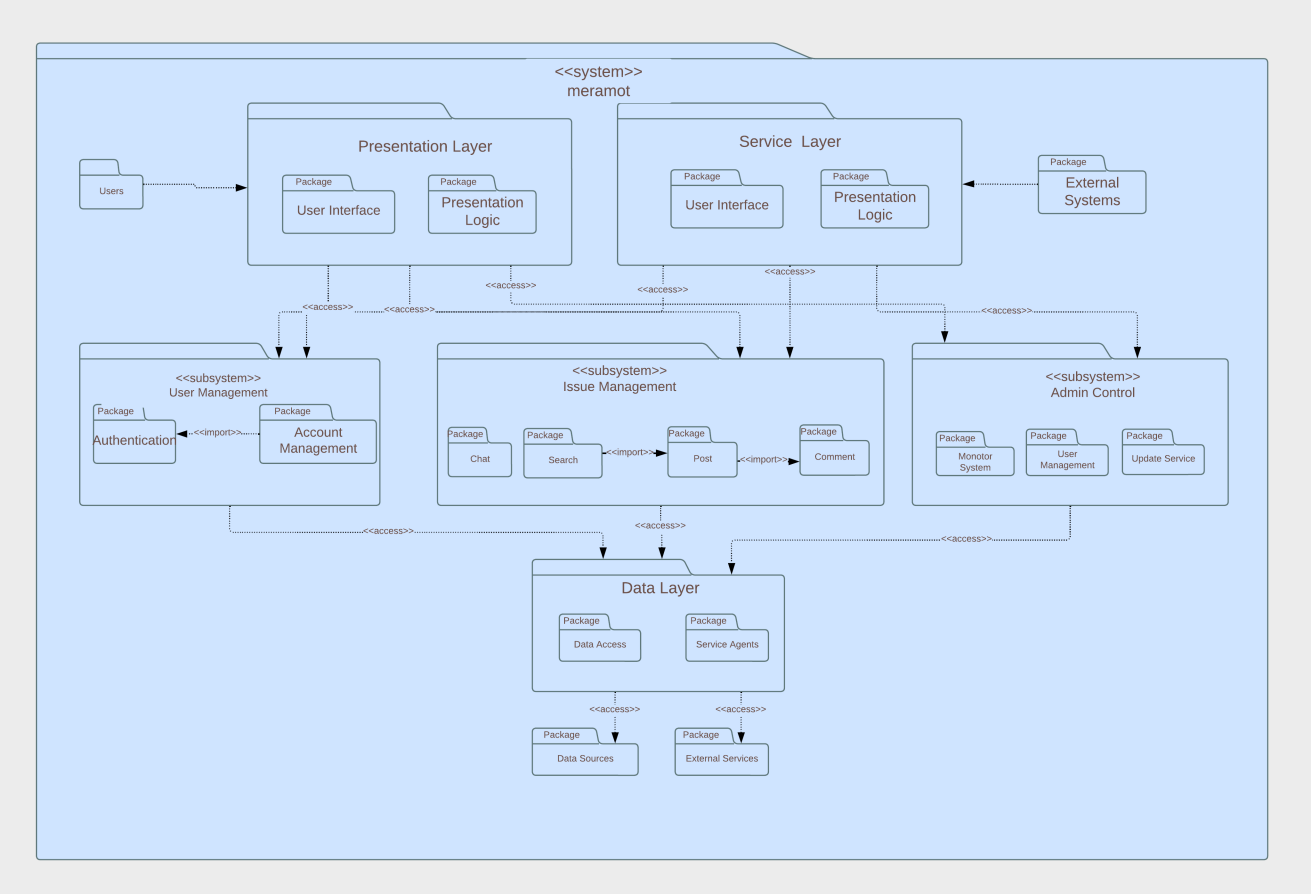

Figure 2: Package Diagram

5.2 Subsystem Detail Description

5.2.1 User Management

5.2.1.1 Module Description This module will handle user authentication, registration, and profile management. It will be responsible for user authorization and access control.

5.2.1.2 Class Diagram

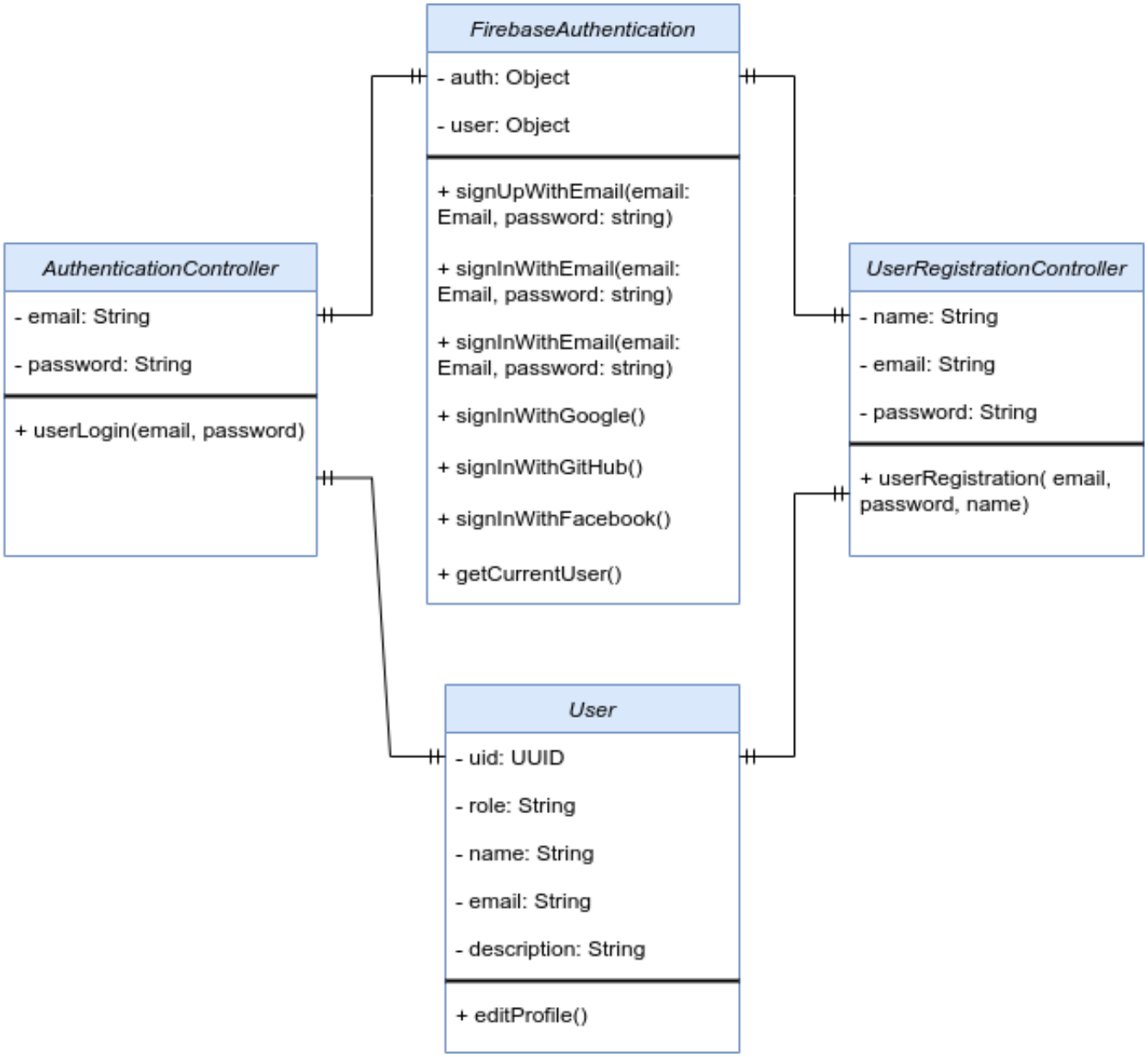

Figure 3: User Management Class Diagram

5.2.1.3 Subsystem Interfaces From the user's perspective, the User Management module will provide a seamless experience for user authentication, registration, and profile management. Users will be able to create an account, log in, and access their profile information from a single interface.

To register, users will need to provide their basic information, including their name, email address, and password. Once registered, users will be able to log in to the system using their email and password credentials. In case of incorrect login credentials, the system will display an error message to inform the user about the issue.

Users will also be able to manage their profile information, including their contact information, profile picture, and other relevant details. They will be able to update their profile information and save the changes in the system. Feedback information will be displayed to the user to confirm that their changes have been saved successfully.

The User Management module will also handle user authorization and access control. Users with appropriate permissions will be able to access certain features of the system, while others will not. Feedback information will be displayed to inform users if they don't have sufficient permissions to access specific features.

Overall, the User Management module will provide a secure and user-friendly interface for managing user authentication, registration, and profile information.

5.2.2 Issue Management

5.2.1.1 Module Description This module will provide an interface for users to post their technology-related questions, issues, or problems. It will also allow users to search for previously posted questions and their answers. Other users and experts can collaborate to solve the issue by responding with potential solutions or asking for further clarification. Once the issue is resolved, the person who posted the question can mark it as resolved or choose the best answer.

5.2.1.2 Class Diagram

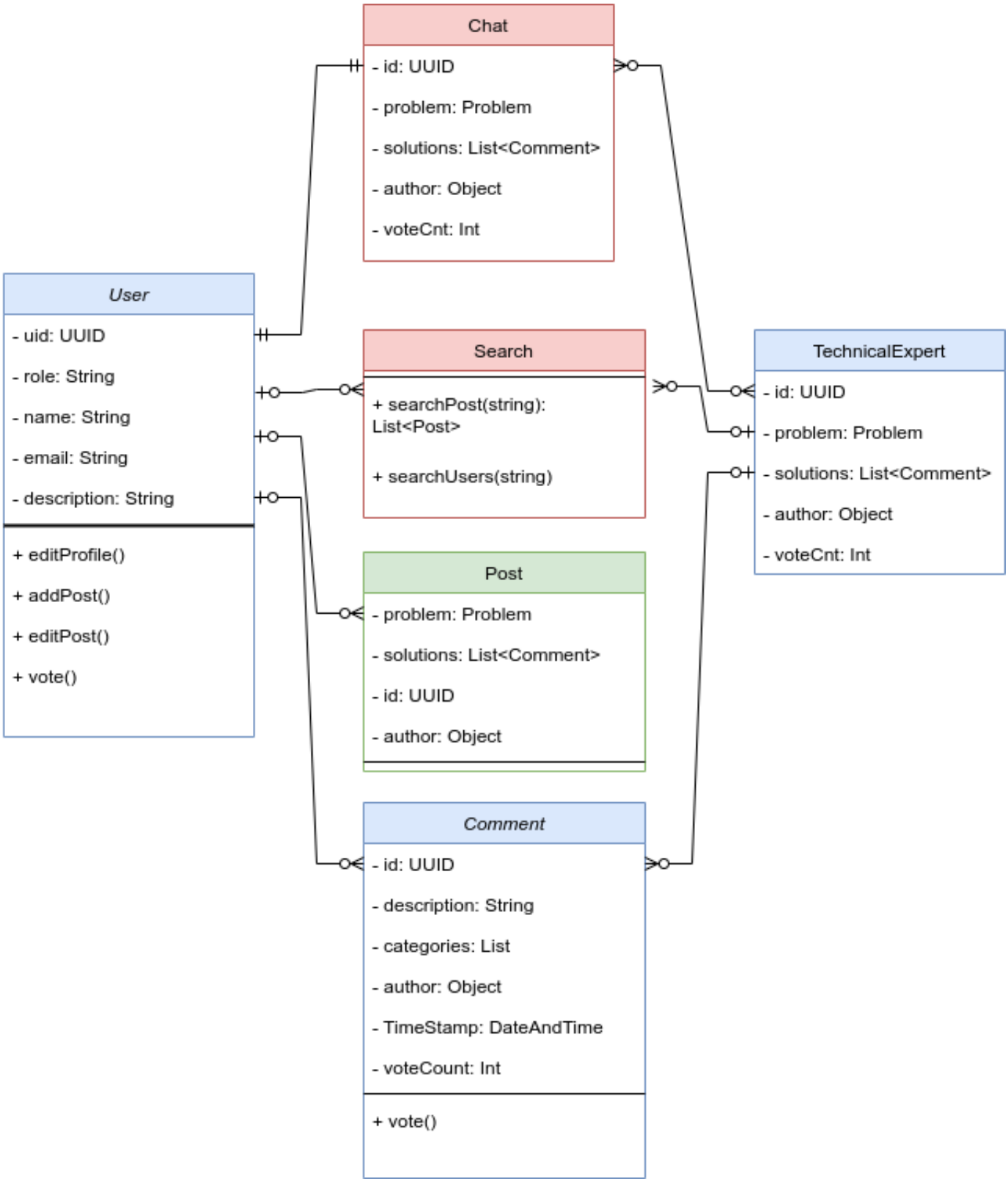

Figure 4: Issue Management Subsystem Class Diagram

5.2.1.3 Subsystem Interfaces The Issue Management subsystem will provide a user-friendly interface for users to post their technology-related questions, issues, or problems. Users can easily navigate to the issue management section of the platform and post their queries by filling out a form that includes fields for tags, titles, descriptions, and screenshots if necessary, providing users with an easy way to accurately describe their issue and provide necessary details.

Once the user posts their issue, it will be visible to other users and experts on the platform. The user can also search for previously posted questions and their answers using the search bar provided on the interface, potentially providing immediate solutions without needing to repost a question.

Other users and experts on the platform can collaborate to solve the issue by responding with potential solutions or asking for further clarification, providing a community approach to resolving tech problems. The user will be able to see these responses and choose the best answer or mark the issue as resolved once it is solved.

Throughout the process, the user will receive feedback information about the status of their issue, including any responses or updates from other users and experts. This feedback information will be displayed on the interface, allowing the user to keep track of their issue and stay up-to-date with any progress. Overall, the Issue Management subsystem will provide an efficient and effective way for users to post their tech-related issues, collaborate with others to solve them, and receive timely feedback on the status of their issues.

5.2.3 Admin Control

5.2.4.1 Module Description The module will handle all admin operations. Verifying Service provider, monitoring overall system will be handled by this module. Only admin users have the access to this subsystem.

5.2.4.2 Class Diagram

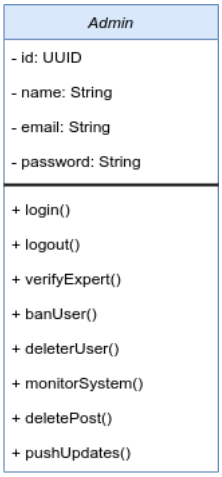

Figure 5: Admin Control Subsystem Class Diagram

5.2.4.3 Subsystem Interfaces The Admin Control subsystem is responsible for managing the overall system and ensuring that all operations are running smoothly. Only admin users have access to this subsystem, which includes features such as verifying technical experts, monitoring system performance, and managing user accounts.

To access the Admin Control subsystem, the user must log in with an admin account. Once logged in, the user will be presented with a dashboard containing various tools and features for managing the system. The dashboard will include options for managing service providers, managing user accounts, and monitoring system performance.

The user account management section will allow the admin user to manage user accounts, including adding new users, deleting accounts, and modifying user details. Admin users can also reset passwords and update user permissions.

The system monitoring section will provide the admin user with real-time information about system performance and usage. This will include metrics such as the number of active users, the number of issues posted, and response times.

The Admin Control subsystem will provide feedback information to the user through the dashboard interface. Notifications will be displayed if there are any issues with the system, such as server downtime or problems with user accounts. The dashboard will also display information about the status of service provider verification and user account management tasks.

In summary, the Admin Control subsystem will provide admin users with the necessary tools and features to manage the overall system, including verifying experts, managing user accounts, and monitoring system performance. The dashboard interface will display real-time feedback information to the user, allowing them to make informed decisions and take action quickly to resolve any issues that may arise.

6 Data Design

6.1 Data Description

- User: This class represents the user's information, including their unique identifier, authorization role, name, email address, description, and lists of posts, comments, reacts, searches, and tags that the user has created or used. It also includes methods to add or remove posts, comments, reacts, searches, and tags, as well as retrieve lists of these objects.

- Post: This class represents a post made by a user and includes a unique identifier, the user object representing the user who made the post, the post's title, body content, and timestamp, as well as lists of comments and reacts made on the post. It also includes methods to add or remove comments and reacts and retrieve the user object.

- Comment: This class represents a comment made by a user on a post and includes a unique identifier, the user object representing the user who made the comment, the post object representing the post the comment was made on, body content, timestamp, and a list of reacts made on the comment. It also includes methods to add or remove reacts and retrieve the user and post objects.

  To store data, a connection to a database server and includes the host name, port number, database name, username, and password used for authentication. It includes methods to establish and close the connection, execute SQL queries, fetch results, commit or rollback transactions.

6.2 Data Dictionary

Data Item	Data Type	Max Size (bytes)	Description
auth	Firebase Authentication	N/A	The Firebase Authentication instance.
body	string	65,536	The body content of the post.
comments	list	N/A	A list of comment objects representing comments made on the post or by the user.
description	string	65,536	An optional description provided by the user.
email	string	254	The email address of the user.
id	string	36	A unique identifier for the post or user.
name	string	255	The name of the user.
posts	list	N/A	A list of post objects that the user has created.
reacts	list	N/A	A list of react objects that the user has made or to the post.
role	string	255	Represents the authorization level of the user.
searches	list	N/A	A list of search queries made by the user.
tags	list	N/A	A list of tags used by the user.
timestamp	datetime	8	The date and time the post was created.
title	string	255	The title of the post.
uid	string	36	A unique identifier for the user.
user	User or Firebase User	N/A	The user object representing the user who made the post or currently authenticated user object.

Table 1: Data Dictionary

6.3 Entity Relationship Diagram

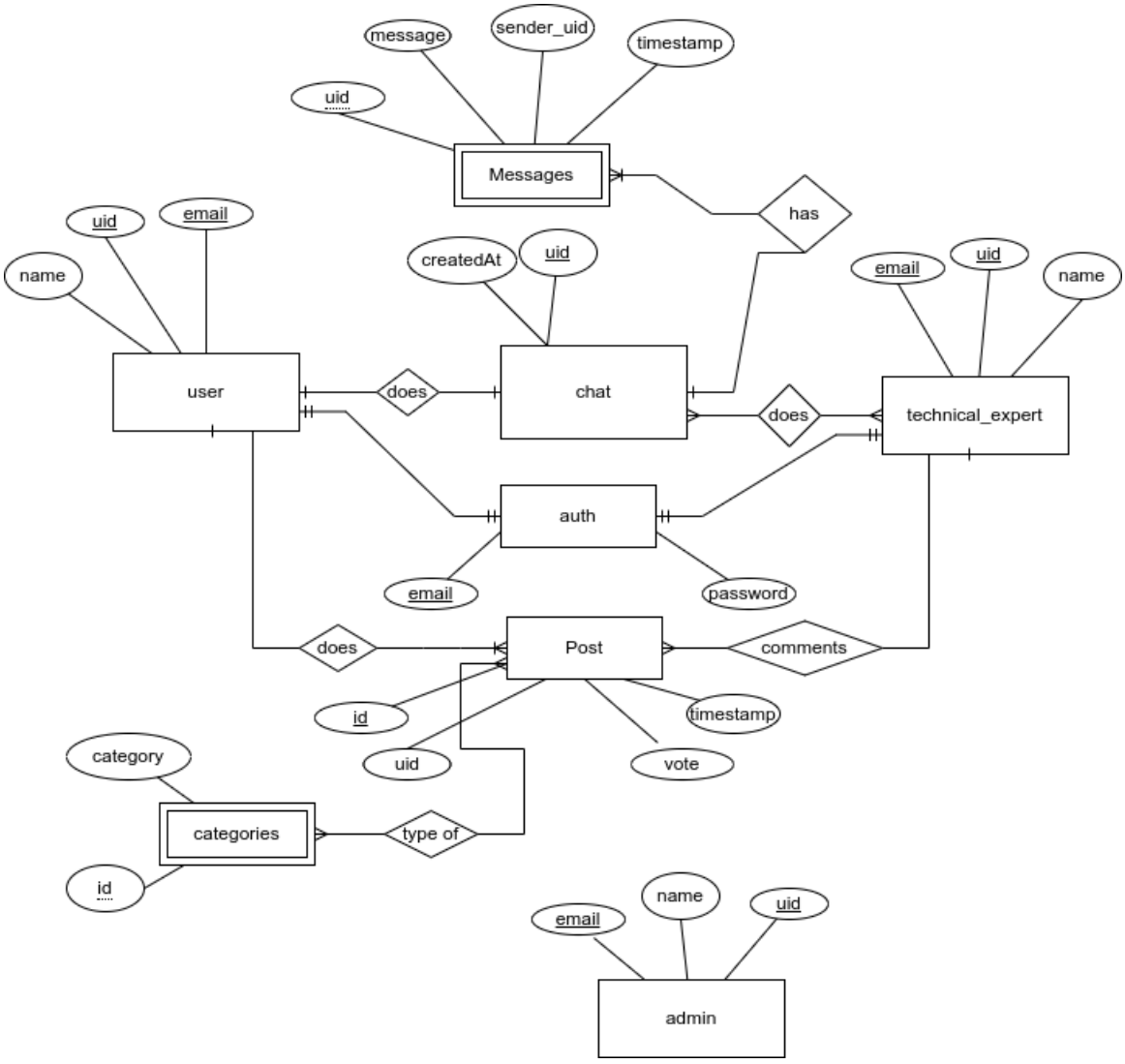

Figure 6: Entity Relationship Diagram

7 User Requirement and Component Traceability Matrix

Requirements	**User Management**	**Issue Management**	**Admin Control**
User Visits as Guest	x	x	-
User Registration and Authentication	x	-	-
User Post Functionality	-	x	-
Commenting on Posts	-	x	-
Upvoting and Downvoting Solutions	-	x	-
Live Chat Support	-	x	-
Supportability	-	-	x

Table 2: Traceability matrix

8 Supporting Information

This document does not need any supporting information from other documents. All the diagrams presented in this document were created with LucidChart [1] Online Diagram maker.