



**The Hashemite University, Zarqa, Jordan**  
**Faculty of Prince Al-Hussein Bin Abdallah II For Information**  
**Technology Software Engineering Department**

# **Two Towers**

**A project submitted**  
**in partial fulfillment of the requirements for the**  
**B.Sc. Degree in Software Engineering**  
**By**

**Hamza Murad Al-Shalabi (2039946)**  
**Supervised by**

**Supervisor: Mohammad Zarour**

**November/2023**



**TWO TOWERS**

## **CERTIFICATE**

It is hereby certified that the project titled *Two Towers* , submitted by undersigned, in partial fulfillment of the award of the degree of “bachelor’s in software engineering” embodies original work done by them under my supervision.

All the analysis, design and system development have been accomplished by the undersigned. Moreover, this project has not been submitted to any other college or university.

***Hamza Murad Al-Shalabi (2039946)***

## **ABSTRACT**

Connecting users searching for houses to rent or buy with available properties is crucial for a vibrant real estate market. Our app addresses this need by providing a user-friendly platform that streamlines the search process across the country. With rising demand for rental and sale properties.

The Two Towers app addresses the challenge of connecting users with available vacant houses for rent or sale across the country, having also many various essential functionalities. The problem of finding suitable vacant properties in a specific area is solved by providing users with a map interface, where pins represent house locations, and a post page containing detailed information about these properties. The app also offers user-to-user communication via a chat feature, making the users able to interact with each other freely.

This app aims to simplify how people find homes for rent or sale. It combines property listings, chat for direct communication, and user-generated content in one easy-to-use platform. This app could have a big impact on the real estate industry by making it easier for both buyers and sellers to connect and share information, potentially changing how the housing market works.

## Catalog

CERTIFICATE .....	II
ABSTRACT .....	III
LIST OF FIGURES .....	V
Chapter 1: Introduction .....	1
Chapter 2: Literature Review .....	10
2.1 Introduction .....	10
2.2 Existing Systems .....	10
2.3 Overall Problems of Existing Systems .....	13
Chapter 3: Requirement Engineering and Analysis .....	15
3.1 Stakeholders .....	15
3.2 Use Case Diagram .....	18
3.4 Constraints .....	28
Chapter 4: Architecture and Design .....	32
4.1.1 Logical view .....	35
4.1.2 Process view .....	35
4.1.3 Physical view .....	49
4.2 Software design .....	49
Chapter 5: Implementation Plan .....	59
5.1 Description of Implementation .....	59
5.2 Programming Language and Technology .....	62
5.3 part of implementation if possible .....	63
Chapter 6: Testing Plan .....	68
5.6 Testing automation .....	78
Chapter 7 : Conclusion and Results .....	83
5.7 Summary of accomplished project .....	83
5.8 Future Work .....	84

## LIST OF FIGURES

### Catalog

Figure 1:WBS Project 1 .....	9
Figure 2:Gantt Chart Project 1 .....	9
Figure 3:WBS Project 2 .....	10
Figure 4:Gantt Chart Project 2 .....	10
Figure 5: Old System 1 .....	12
Figure 6: Old System 2 .....	12
Figure 7: Old System 3 .....	12
Figure 8: Use Case Diagram .....	19
Figure 9:Activity Diagram 1 .....	30
Figure 10:Activity Diagram 2 .....	30
Figure 11:Activity Diagram 3 .....	30
Figure 12:Activity Diagram 4 .....	31
Figure 13:Activity Diagram 5 .....	31
Figure 14:Activity Diagram 6 .....	31
Figure 15:Activity Diagram 7 .....	32
Figure 16:Activity Diagram 8 .....	32
Figure 17:MVVM Component Diagram .....	34
Figure 18:Coneptual Diagram .....	35
Figure 19:Component Diagram .....	36
Figure 20:Sequence 1 .....	37
Figure 21:Sequence 2 .....	38
Figure 22:Sequence 3 .....	39
Figure 23:Sequence 4 .....	40
Figure 24:Sequence 5 .....	41
Figure 25:Sequence 6 .....	42
Figure 26:Sequence 7 .....	43
Figure 27:Sequence 8 .....	44
Figure 28:Sequence 9 .....	45
Figure 29:Sequence 10 .....	46
Figure 30:Sequence 11 .....	47
Figure 31:Sequence 12 .....	48
Figure 32:Sequence 13 .....	49
Figure 33:Deployment Diagram .....	50
Figure 34:Class Diagram .....	51
Figure 35:State Diagram 1 .....	52
Figure 36:State Diagram 2 .....	52
Figure 37:State Diagram 3 .....	53
Figure 38:State Diagram 4 .....	53
Figure 39:State Diagram 5 .....	54
Figure 40:State Diagram 6 .....	54
Figure 41:Map UI .....	55
Figure 42:Login UI .....	56
Figure 43:Explore UI .....	57
Figure 44:Chat UI .....	58

## LIST OF Tables

Catalog	
Table 1:Edit Profile Info .....	19
Table 2:Delete Account .....	19
Table 3:Delete Post .....	20
Table 4:Remove Saved Post .....	20
Table 5:Create Post .....	21
Table 6:Browse House .....	21
Table 7:Filter Posts .....	22
Table 8:Search User Posts .....	22
Table 9:Save House Post .....	23
Table 10:Browse Conctacts .....	23
Table 11:Send Massages .....	24
Table 12:Send Feedback .....	24
Table 13:Recieve Feedback .....	25
Table 14:Accept Post .....	25
Table 15:Reject Post .....	26
Table 16:Give Warning .....	26
Table 17:Test 1 .....	27
Table 18:Test 2 .....	71
Table 19:Test 3 .....	73
Table 20:Test 4 .....	74
Table 21:Test 5 .....	74
Table 22:Test 6 .....	75
Table 23:Test 7 .....	75
Table 24:Test 9 .....	76
Table 25:Test 10 .....	77

## CHAPTER 1: INTRODUCTION

This chapter will introduce an explanation of the project it will also introduce the main idea, problems, solutions, and reasons that led to implement this project.

**1.1 Overview:** This project introduces a Android application meticulously crafted for Android smartphones, aiming to revolutionize the way users explore and engage with the real estate market. The central focus of the app lies in providing users with a seamless and comprehensive experience in discovering available vacant houses for both rent and sale. Leveraging cutting-edge technology, enabling users to effortlessly filter through an extensive database of properties to find the ideal homes that match their specific criteria. With a user-friendly interface, the app not only presents detailed property listings but also incorporates advanced features, ensuring users can make informed decisions about their potential future homes. Additionally, the app incorporates real-time updates on property availability, price trends, and neighborhood insights, offering a holistic approach to the property discovery process.

### 1.2 Project Motivation

- The reason behind developing this project is to make the process of finding vacant houses much easier for Jordanian people, and so far, no such app was ever available for the people of Jordan.
- The goal of this project is to change experience of the user from going to search for the empty houses on his own, to looking for them online in any place in Jordan through a click of a button.
- A built-in chat that will lessen the steps needed to talk to other users.

### 1.3 Problem Statement

The current project will address several key issues in the domain of property listings and real estate services, with a focus on enhancing the user experience and streamlining the process of finding vacant houses for rent or sale. The following problem areas have been identified and targeted for improvement:

**1-Fragmented Property Listings:** Users often face the challenge of searching for vacant houses across a vast geographical area. Existing listings are fragmented across various sources, making it time-consuming to find suitable properties.

**2-Limited User Interaction:** Traditional property listing platforms lack real-time interaction features. Users have limited means of direct communication, which can hinder their ability to inquire about properties or seek additional information.

**3-Spatial Awareness:** Users may struggle to visualize the location of available properties in their desired area, making it difficult to assess proximity to important amenities or landmarks.

### 1.4 Project Aim and Objectives

- The goal of this project is to provide Android smartphone users with a comprehensive and user-friendly solution for finding vacant houses available for rent and sale in a specific geographical area across the country. The primary aim is to revolutionize the property search process by offering a single platform that seamlessly connects users with property listings, facilitates real-time communication, and encourages user contributions.

- 

#### 1. User-Centric Design:

- Implement an intuitive and user-friendly interface to enhance the overall user experience.
- Prioritize features based on user needs and feedback to ensure relevance.



## 2. Map Integration:

- Utilize the Google Maps API to display the locations of empty houses, providing users with a visual representation of available properties.
- Implement pinning functionality on the map for easy navigation and exploration.

## 3. Real-Time Communication:

- Integrate chat functionality to facilitate communication between users interested in a particular property.
- Enable users to inquire about, discuss, and share information regarding the listed houses.

## 4. Comprehensive Property Information:

- Develop a detailed property information page with images, descriptions, and relevant details to assist users in making informed decisions.

## 5. Legal and Ethical Considerations:

- Adhere to legal standards and privacy regulations to protect user data and ensure compliance.
- Implement ethical practices in content moderation and user-generated content to maintain a positive user experience.

## 1.5 Project Scope

**Property Listings:** The property listings feature serves as the cornerstone of the application, offering users a diverse array of real estate options for both rent and sale. Our listings provide comprehensive details about each property, including key features, pricing, and high-quality images. Users can easily navigate through the listings, apply filters, and customize their search parameters to find properties that

align with their unique preferences and requirements. This functionality ensures a user-friendly and efficient experience in the quest for the perfect home.

**Chat Functionality:** Facilitating seamless communication between property seekers and sellers, the app incorporates robust chat functionality. This feature empowers users to engage in real-time conversations, ask questions, and negotiate terms directly within the application. By fostering direct communication, the chat functionality enhances the user experience, expediting the decision-making process and fostering transparent interactions between all parties involved in the real estate transactions.

**Interactive Map:** The interactive map feature adds a dynamic dimension to property exploration by providing users with a visual representation of available listings in specific geographic areas. Users can explore neighborhoods, assess proximity to amenities, and gain a comprehensive understanding of the local environment. The map feature enhances the user's ability to make well-informed decisions by incorporating a spatial element into their property search, offering a holistic view beyond individual listings.

**Property Information Pages:** Each property in the app is accompanied by detailed information pages, offering a deep dive into the specifics of the listing. From architectural details to amenities and neighborhood characteristics, these pages serve as a comprehensive resource for users to evaluate the suitability of a property. Rich multimedia content, including images further enhances the informational aspect, providing users with a virtual tour of the property without leaving the app.

**Secure Authentication:** Prioritizing user security, our app employs a robust and secure authentication system. Through encrypted protocols and multi-factor authentication, we ensure that user data remains confidential and protected from unauthorized access. This emphasis on secure authentication instills trust in our users, fostering a safe and secure environment for their real estate transactions and interactions within the app.

**User-Generated Posts:** Encouraging user engagement and community building, our app allows users to create user-generated posts. Whether it's sharing personal experiences, offering insights into specific neighborhoods, or providing reviews of properties, this feature adds a social aspect to the real estate platform. User-generated posts contribute to a vibrant and dynamic community within the app, where users can share valuable information and connect with others who share similar interests or preferences in the real estate market.

## **Project Software and Hardware Requirements**

### Software Requirements:

- **Android Studio:** The primary integrated development environment (IDE) for Android app development.
- **Firebase Database:** Utilized for user authentication, storage, and real-time database functionality.
- **Android Emulator:** Used for testing the app during development.
- **Figma:** Graphics and design software for creating app layouts and assets.

### Hardware Requirements:

- **Android Smartphone:** The app is designed for Android phones, so physical testing on Android devices is essential.
- **Computer:** A development machine running Windows, macOS, or Linux with sufficient processing power and memory.
- **Internet Connection:** Required for downloading dependencies, updates, and testing cloud-based features (Firebase).

## 1.6 Project Limitations

- Technical Limitations:

The technology being used for this project may have certain limitations that could prevent certain features or functionality from being implemented.

- Development Time frame Limitations:

The project needs to adhere to a predefined schedule for design, development, testing, and deployment.

**1.7 Project Expected Output:** The Two Towers app is expected to deliver a user-friendly and feature-rich experience, providing the following key outputs:

Property Listings:

- A comprehensive list of empty houses available for rent and sale.
- Details such as price, location, and property type for each listing.

Chat Functionality:

- Real-time chat capabilities enable users to communicate with each other.
- Instant messaging for property-related inquiries and discussions.

Interactive Map:

- A dynamic map displaying the locations of empty houses across the country.
- Pins on the map representing individual properties for easy identification.

Property Information Pages:

- Individual pages for each property with detailed descriptions and images.
- Relevant information is presented in a clear and organized manner.

Authentication System:

- A fully authenticated login and sign-up system using Firebase database.
- Secure access to user accounts with personalized settings.

#### User-Generated Posts:

- Users can create and share posts featuring information about available properties.
- Posts include text descriptions, images, and other relevant details.

#### Map Integration with Post Page:

- Seamless connectivity between the interactive map and the user-generated post page.
- Clicking on a pin in the map navigates the user to the corresponding property post.

#### Current Location Initialization:

- The map initializes at the user's current location upon opening the app.
- Automatic detection of the user's geographical coordinates for an enhanced experience.

#### User Profile Management:

- The user Can update His profile information as he wishes, such as new user name or password etc.

#### Additional Functionalities:

- Intuitive UI/UX design for a smooth and visually appealing interface.
- Options for users to filter and search for properties based on specific criteria.
- Notifications for new chat messages, posts, and relevant updates.

The expected output of Two Towers app is a robust, user-centric platform that effectively connects property seekers and sellers, offering a seamless and engaging experience in the process of finding and sharing housing opportunities.

## 1.8 Report Organization

Chapter 2 introduces Literature Review of this project.

Chapter 3 introduces the requirements of this project.

Chapter 4 presents the design and architecture.

components are described in Chapter 5.

Chapter 6 presents the future work and concludes the report.

## 1.9 Work Breakdown Structure And Gantt Chart

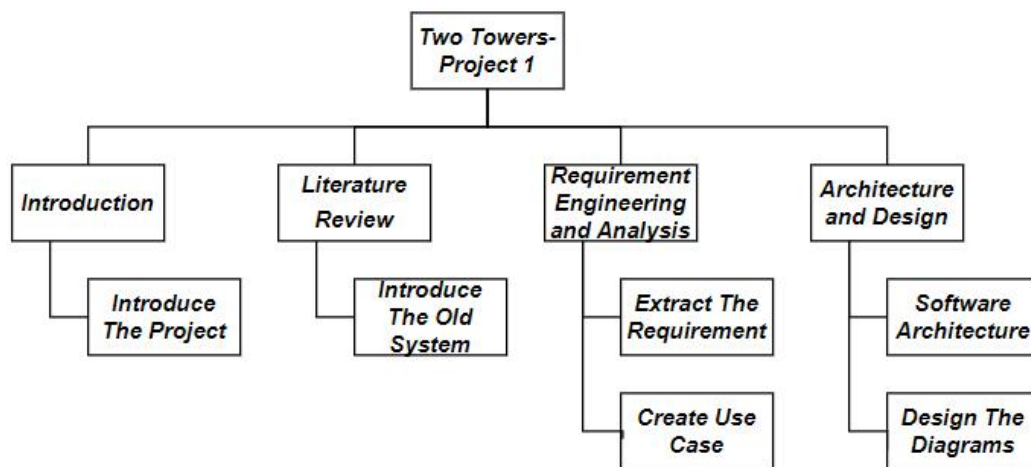
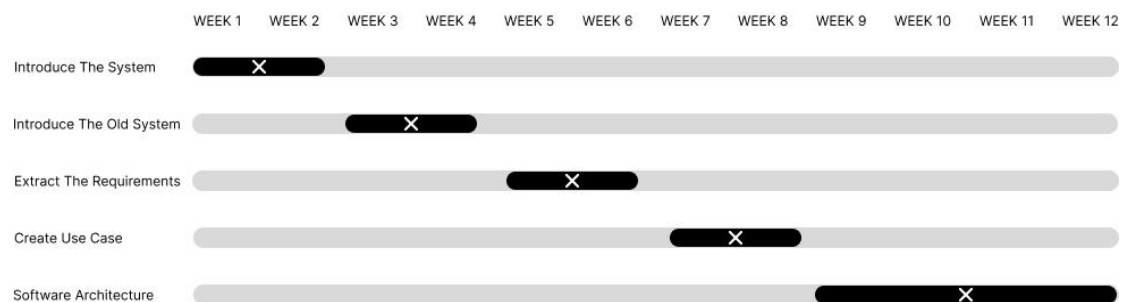
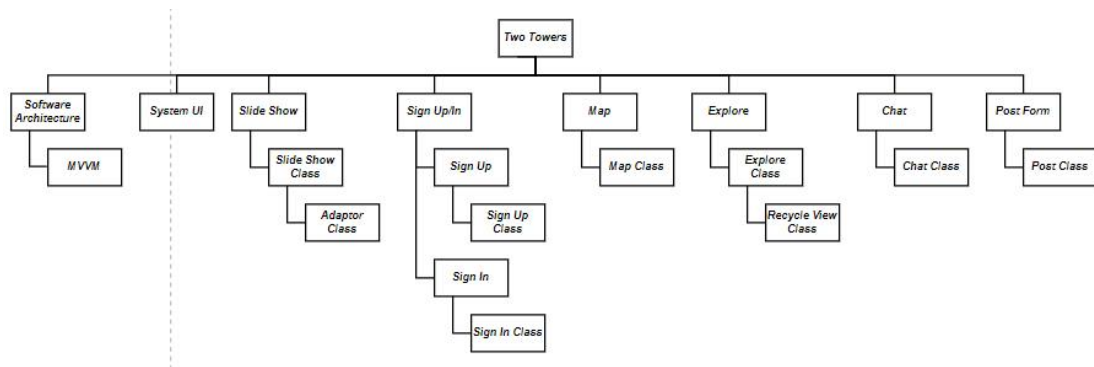


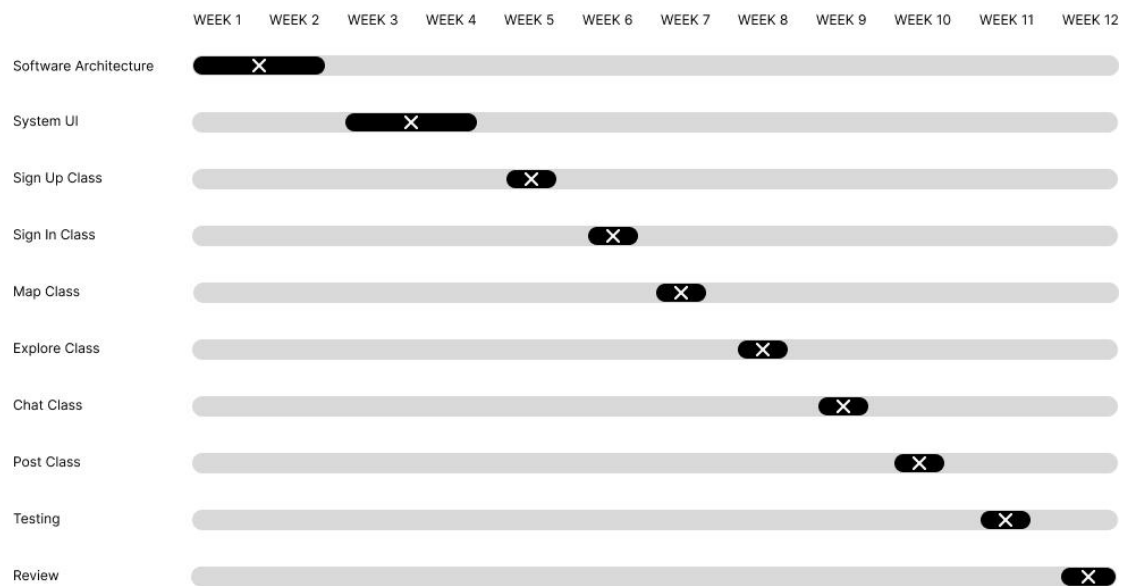
Figure 1(WBS Project 1)



**Figure 2(Gantt Chart Project 1)**



**Figure 3(WBS Project 2)**



**Figure 4(Gantt Chart Project 2)**

## **CHAPTER 2: LITERATURE REVIEW**

### **2.1 Introduction**

The literature review explores the current state of online property classifieds, focusing on OpenSooq, a well-established platform known for its diverse range of listings. This section aims to analyze OpenSooq's strengths, weaknesses, and potential areas for enhancement.

### **2.2 Existing Systems**

Overview of OpenSooq:

OpenSooq is an online marketplace that offers a comprehensive property listing functionality, catering to users seeking to buy, sell, or rent real estate. The platform provides a user-friendly interface where property owners, real estate agents, and individuals can list their properties efficiently. Users can navigate through an extensive range of property listings, including apartments, houses, commercial spaces, and more. Each property listing is detailed with essential information such as property type, size, location, price, and contact details. OpenSooq's search and filter options empower users to refine their property search based on specific criteria, ensuring a tailored and efficient browsing experience. The platform fosters communication between buyers and sellers through built-in chat functionality, allowing for direct inquiries and negotiations. With a secure authentication system in place, OpenSooq prioritizes user privacy and safety.



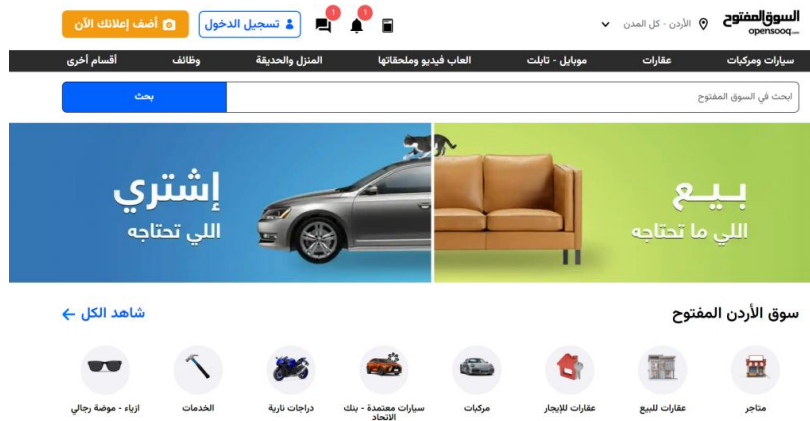


Figure 5 Main Page

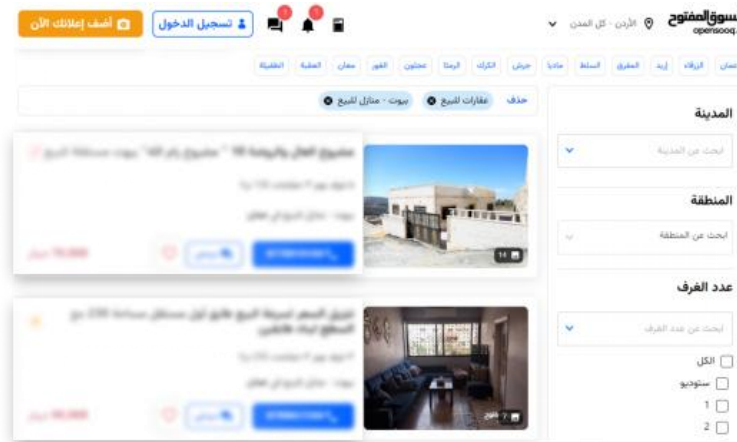


Figure 6 Real Estate Page



Figure 7 Post Page

**Key Features Contributing to OpenSooq's Popularity:**

**Extensive Property Listings:** OpenSooq offers a diverse range of property listings, covering residential and commercial spaces. Users can explore a wide variety of real estate options, making it a one-stop platform for property-related needs.

**Detailed Property Information:** Each property listing on OpenSooq provides comprehensive details, including property type, size, location, amenities, and pricing. This detailed information enables users to make informed decisions about potential properties.

**Advanced Search and Filters:** OpenSooq incorporates robust search and filter functionalities. Users can narrow down their property search based on specific criteria such as location, price range, and property type, facilitating efficient and targeted results.

**Chat Functionality:** The built-in chat feature facilitates direct communication between buyers and sellers. This real-time communication channel streamlines the negotiation process and allows users to seek additional information about listed properties.

**User-Generated Content:** OpenSooq encourages user engagement through the ability to create and post content. This user-generated aspect adds dynamism to the platform and allows individuals to actively participate in the real estate community.

## **2.3 Overall Problems of Existing Systems**

**Incomplete or Outdated Listings:** Some property listings may lack essential information or be outdated, leading to frustration for users seeking accurate and up-to-date details about available properties.

**Scams and Fraudulent Listings:** Property scams can be a concern on online platforms. Users may encounter fake listings or fraudulent schemes, impacting trust and confidence in the platform.

**Lack of High-Quality Images:** Listings with poor-quality or insufficient images can hinder users' ability to assess properties effectively.

**Technical Glitches:** Like any online platform, technical issues such as slow loading times, error messages, or unresponsive pages can frustrate users and disrupt their property search experience.

The primary issue identified in the OpenSooq lies in the broad scope of the website. Serving multiple product categories, the platform faces the issue of generalization, where the real estate segment receives less dedicated attention.

## **2.4 Overall Solution Approach.**

the TWO TOWERS app will solve some of the the problems mentioned above in the following manner

### **1. Incomplete or Outdated Listings:**

- Implement a mandatory checklist for property listings to ensure all essential information is provided.
- Set up automated notifications to remind users to update their listings regularly.

## 2. Scams and Fraudulent Listings:

- Implement a robust verification process for property listings and users to reduce the likelihood of scams.
- Encourage users to report suspicious listings and promptly investigate and remove fraudulent content.

## 3. Lack of High-Quality Images:

- Encourage users to upload high-resolution images through incentives or highlighting listings with quality visuals.
- Provide guidelines for property image submissions to ensure clarity and completeness.

## **CHAPTER 3: REQUIREMENT ENGINEERING AND ANALYSIS**

### **3.1 Stakeholders**

Primary Stakeholders:

- Users (Renters and Buyers): Individuals who use the app to find and inquire about empty houses for rent or sale.
- App Developers: The team or individuals responsible for designing, developing, and maintaining the app.

Secondary Stakeholders:

- Real Estate Agents: If the app partners with real estate agents or agencies, they become secondary stakeholders.
- Homeowners and Property Managers: Individuals who own or manage the empty houses listed on the app.
- Competitors: Other apps or services in the real estate market may perceive themselves as affected by the success of your app.
- The Supervisor: Those responsible for overseeing the development and implementation of the app.

**Use Case Scenario:**

User Shall Be able to create a new account by providing required information such as username, email, and password. Confirm account creation by clicking on "Sign Up".

User Shall Sign in to their existing account by entering their email and password. Click on "Sign In" to authenticate and access their account.

User Shall Be able to navigate to the "Manage Account" page to edit desired account information. Confirm changes to initiate system updates and receive a confirmation message. Have the option to cancel changes, causing the system to discard them and revert to the original account information.

User Shall Click on "Delete Account" to start the account deletion process. Confirm their intention to delete the account by clicking on the "Confirm" button. Be logged out of the system upon successful account deletion.

User Shall Click on "Create Post" to begin creating a new post. Enter the content of the post. Click on the "Publish" button to create and publish the post. User shall be able to save posts as drafts for later editing or publishing. User shall be able to change the status of their posts to "Sold" or "In Progress".

User Shall Click on "Delete" for a specific post to initiate deletion. Confirm deletion by clicking on "Confirm".

User shall be able to browse available properties and view their details, including location, features, and photos Through Map.

User shall be able to search for properties based on specific criteria, such as location, price range, number of bedrooms, and amenities.

User shall be able to save posts. User shall be able to view and manage their saved posts. User shall be able to delete saved posts from their lists.

User Shall Click on "Chat" next to a selected user's profile to initiate a chat conversation. Send and receive messages within the chat window.

User Shall Click on "Contacts" to view their list of contacts. Review contact information for each contact within the list.

User Shall Enter the content of the message they wish to send. Select a recipient from their contacts list. Click on "Send" to dispatch the message to the chosen recipient.

The User shall be able to submit feedback to the system administrator.

Administrators shall be able to review and approve or reject user-created posts.

The system shall notify users when their posts are rejected, providing a clear explanation of the reason for rejection. The system shall issue a warning to users who violate content guidelines. User account shall be permanently deleted after three content violations.

## 3.2 Use Case Diagram

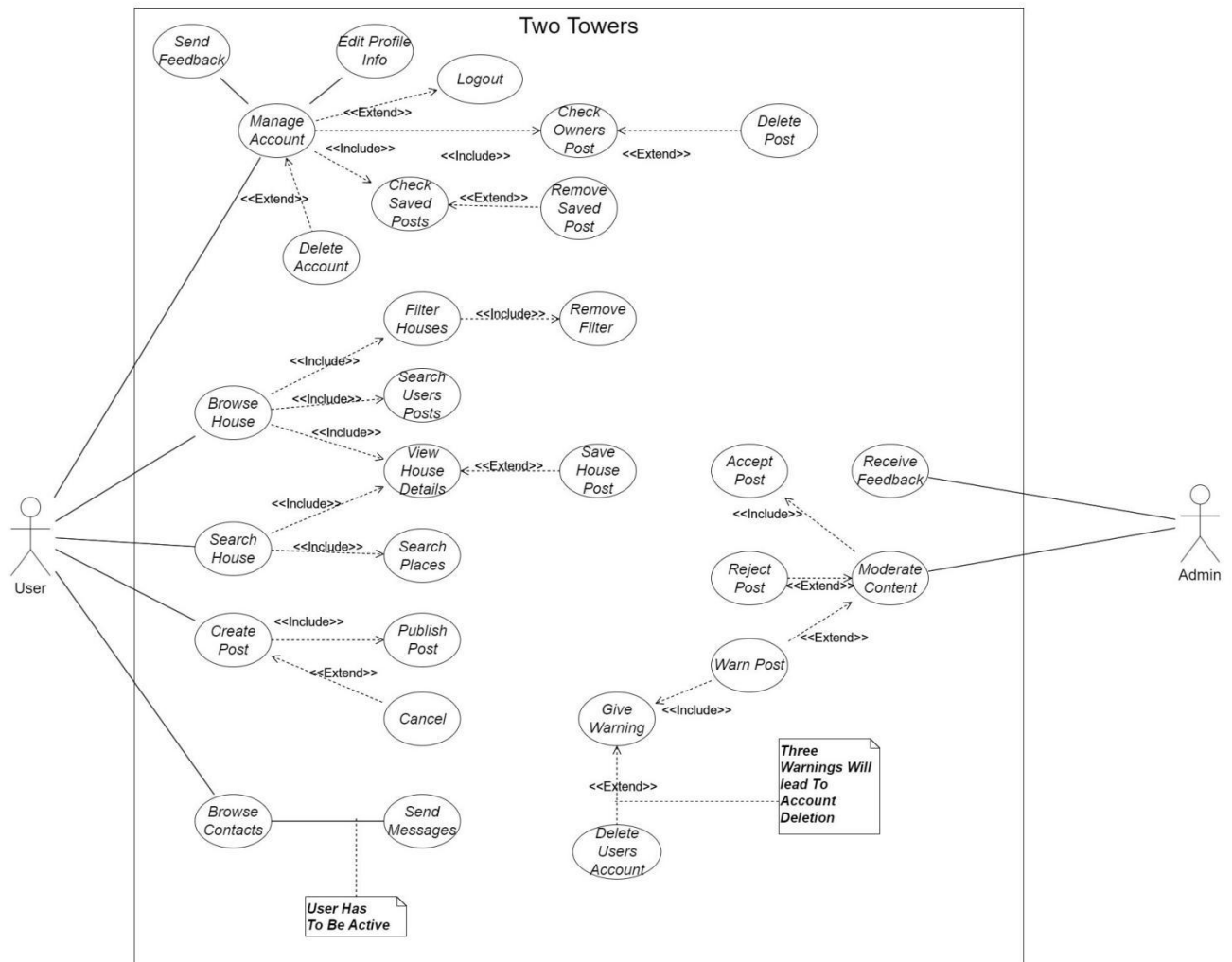


Figure 8(Use Case Diagram)



Use Case	Edit Profile Info
Actor	User
Precondition	User Is Logged In
Basic Flow	1-User navigates to the "Manage Account" page. 2-User edits desired account information. 3-User confirms changes. 4-System updates user account information. 5-System displays a confirmation message.
Alternative Flow	User cancels changes: The system discards changes and returns to the "Manage Account" page.
Post-Condition	User account information is updated.

**Table 1**

Use Case	Delete Account
Actor	User
Precondition	User Is Logged In
Basic Flow	1-User navigates to the "Manage Account" page. 2-The user clicks on the "Delete Account" Button. 3-The system asks for user info to authenticate. 4-The system displays a confirmation message. 5-The user clicks on the "Confirm" button. 6-The system deletes the user's account. 7-The system logs the user out.
Alternative Flow	1-The user clicks on the "Cancel" button.
Post-Condition	1-The user's account has been deleted. 2-The user is logged out.

**Table 2**

Use Case	Delete Post
Actor	User
Precondition	1-User Is Logged In 2-User has a published Post.
Basic Flow	1-User navigates to the "My Posts" page. 2-The user clicks on the "Delete" button for the post. 3-The system displays a confirmation message. 4-The user clicks on the "Confirm" button. 5-The system deletes the post.
Alternative Flow	The user clicks on the "Cancel" button.
Post-Condition	The post has been deleted.

**Table 3**

Use Case	Remove Saved Post
Actor	User
Precondition	1-User Is Logged In 2-User has a saved house post.
Basic Flow	1-User navigates to the "Saved Posts" page. 2-The user clicks on the "Remove" button for the post. 3-The system displays a confirmation message. 4-The user clicks on the "Confirm" button. 5-The system removes the post from saved.
Alternative Flow	The user clicks on the "Cancel" button.
Post-Condition	The post has been removed from saved.

**Table 4**

Use Case	Create Post
Actor	User
Precondition	User Is Logged In
Basic Flow	1-The user clicks on the "Create Post" button. 2-The user enters the post content. 3-The user clicks on the "Publish" button. 4-The system creates and publishes the post.
Alternative Flow	The user drafts the post.
Post-Condition	A new post has been created and published.

**Table 5**

Use Case	Browse House
Actor	User
Precondition	User Is Logged In
Basic Flow	1-User navigates to the "Map" page. 2-The system displays houses on map. 3-The user chooses a house. 4-The system displays house info.
Alternative Flow	The user does not find a house
Post-Condition	The user has found a house

**Table 6**

Use Case	Filter Post
Actor	User
Precondition	User Is Logged In
Basic Flow	1-User navigates to the "Post" page. 2-The chooses a criteria for a house and clicks "Filter" Button. 3-The user chooses a house. 4-The system displays house info.
Alternative Flow	The user does not find a house
Post-Condition	The user has found a house

**Table 7**

Use Case	Search User Posts
Actor	User
Precondition	User Is Logged In
Basic Flow	1-User navigates to the "Post" page. 2-Enters a user name in the search par 3-The system displays user posts.
Alternative Flow	The user does not find a house
Post-Condition	The user has found a house

**Table 8**

Use Case	Save House Post
Actor	User
Precondition	User Is Logged In
Basic Flow	3-The user chooses a house. 4-The system displays house info. 3-The User clicks "Save" Button.
Alternative Flow	The System does not save Post
Post-Condition	The post has been saved.

**Table 9**

Use Case	Browse Contacts
Actor	User
Precondition	User Is Logged In
Basic Flow	1-The user clicks on the "Contacts" button. 2-The system displays the user's list of contacts. 3-The user can view the contact information for each contact.
Alternative Flow	The user does not have any contacts
Post-Condition	The user has reviewed their list of contacts.

**Table 10**

Use Case	Send Messages
Actor	User
Precondition	User Is Logged In
Basic Flow	1-The user enters the message content. 2-The user clicks on the "Send" button. 3-The system sends the message to the recipient.
Alternative Flow	User erase the message
Post-Condition	A message has been sent to the selected user.

**Table 11**

Use Case	Send Feedback
Actor	User
Precondition	User Is Logged In
Basic Flow	1-The user enters the message content. 2-The user clicks on the "Send" button. 3-The system sends the message to the system admin.
Alternative Flow	The system does not send the message.
Post-Condition	The message has been sent to the system admin.

**Table 12**

Use Case	Receive Feedback
Actor	Admin
Precondition	Admin Is Logged In
Basic Flow	1-The Admin receives feedback message. 2-The Admin Saves feedback for next releases.
Alternative Flow	The Message has not been received.
Post-Condition	Feedback has been saved

**Table 13**

Use Case	Accept Post
Actor	Admin
Precondition	1-Admin Is Logged In. 2-Admin has post request.
Basic Flow	1-The Admin receives post request. 2-The Admin Checks the post content. 3-The Admin clicks on the "Accept" button. 4-The system publishes the post.
Alternative Flow	The Admin discards the post for lack of info.
Post-Condition	A new post has been published.

**Table 14**

Use Case	Reject Post
Actor	Admin
Precondition	1-Admin Is Logged In. 2-Admin has post request.
Basic Flow	1-The Admin receives post request. 2-The Admin Checks the post content. 3-The Admin clicks on the "Reject" button. 4-The system does not publish the post.
Alternative Flow	The Admin clicks "Cancel" Button.
Post-Condition	The post has been rejected.

**Table 15**

Use Case	Give Warning
Actor	Admin
Precondition	1-Admin Is Logged In. 2-Admin has post request.
Basic Flow	1-The Admin receives post request. 2-The Admin Checks the post content. 3-The Admin clicks on the "Reject" button. 4-The system does not publish the post. 5-The Admin strikes the user profile with a warning.
Alternative Flow	The Admin clicks "Cancel" Button.
Post-Condition	The User account has a warning.

**Table 16**



Use Case	Delete User Account
Actor	Admin
Precondition	1-Admin Is Logged In.
Basic Flow	1-The Admin clicks on the "Reject" button for a user post. 2-The Admin strikes the user profile with a warning. 3-The User profile has three warnings. 4-The Admin deletes the user account
Alternative Flow	The Admin clicks "Cancel" Button.
Post-Condition	The User account has been deleted.

**Table 17**

### **3.3 Non-Functional User Requirements**

#### **Execution Qualities:**

Usability: The app should have an intuitive and user-friendly interface.

Security: User authentication and authorization mechanisms should be robust.

Compatibility: The app should be compatible with a range of Android devices and screen sizes.

#### **Evolution Qualities:**

Maintainability: The app's code base should be well-organized and documented for ease of maintenance.

Testability: The app should be designed with a modular structure to facilitate unit testing.

The MVVM Design pattern we will help achieve the aforementioned non- functional attributes because it will offer a great Separation of concerns that will help us achieve our non-functional attributes especially maintainability and the testability

### **3.4 Constraints**

Platform Constraint:

The app must be developed specifically for Android devices and must be compatible with a range of Android OS versions, as specified by the target user base.

Technology Stack Constraint:

The app development should utilize Kotlin as the primary programming language and integrate with Firebase for user authentication and database management.

Data Privacy and Security Constraint:

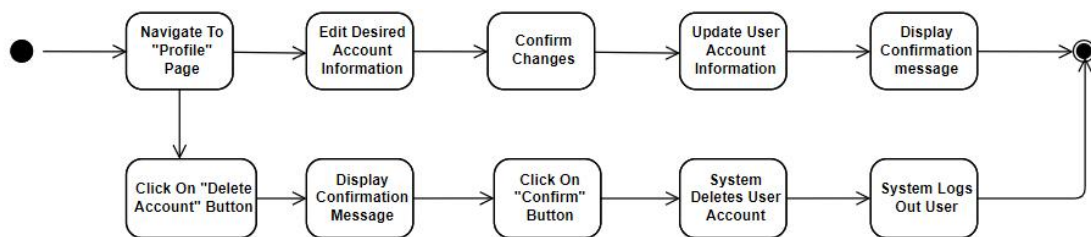
The app must adhere to data protection laws and ensure the secure storage and transmission of user data. Compliance with Firebase security standards is mandatory.

Internet Connection Constraint:

Certain features of the app, such as real-time map updates and chat functionality, may require a stable internet connection. The app should gracefully handle scenarios where the user is offline.

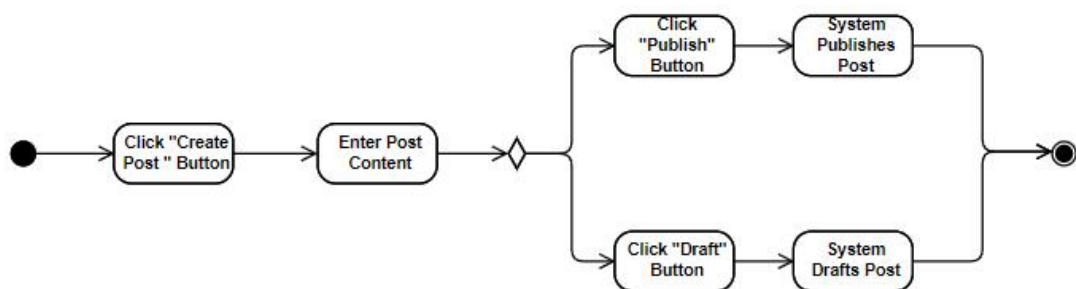
### 3.5 Activity Diagram

An activity diagram is a diagram that visually represents the workflow or activities within a system, business process, or use case. It provides a high-level view of the dynamic aspects of a system, emphasizing the flow of control and the sequence of activities. Activity diagrams are particularly useful for modeling business processes, system behaviors, and the coordination of different activities.



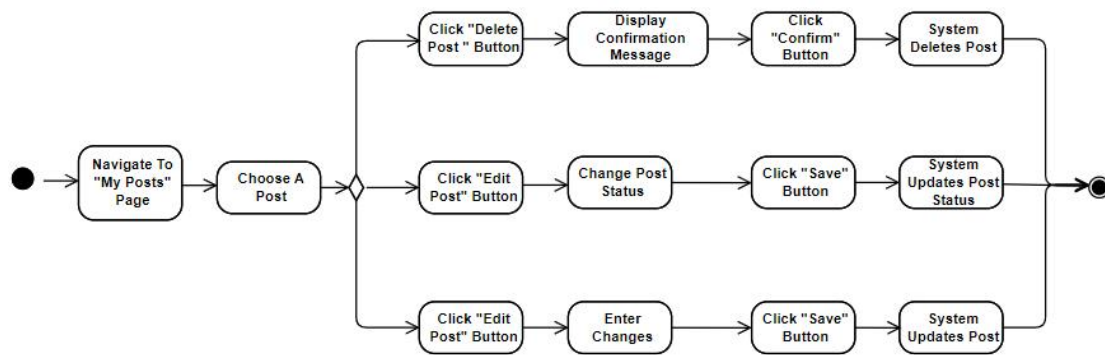
**Figure 9 (Activity 1)**

Includes Use Cases : Edit Account Info , Delete Account.



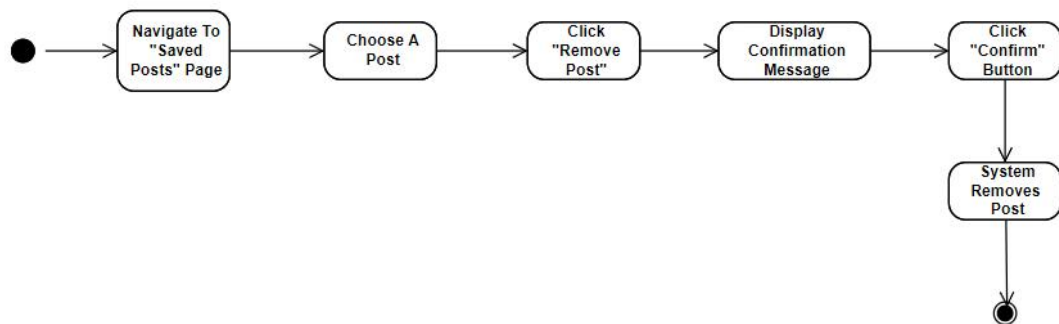
**Figure 10(Activity 2)**

Includes Use Cases : Create Post , Publish Post And , Draft Post.



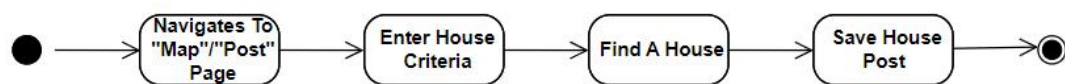
**Figure 11(Activity 3)**

Includes Use Cases : Change Post Status , Edit Post And , Delete Post.



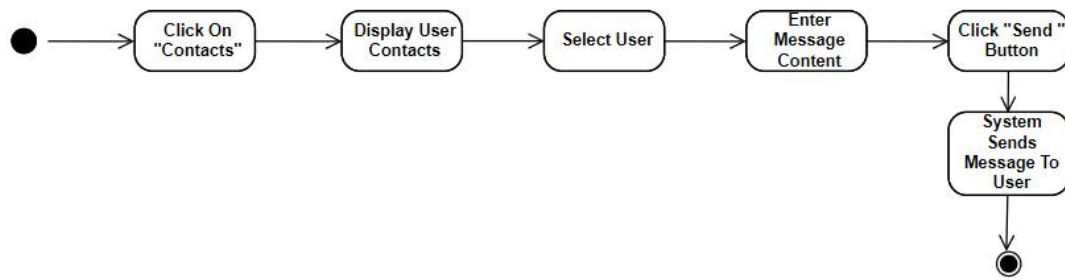
**Figure 12(Activity 4)**

Includes Use Cases : Remove Saved Posts.



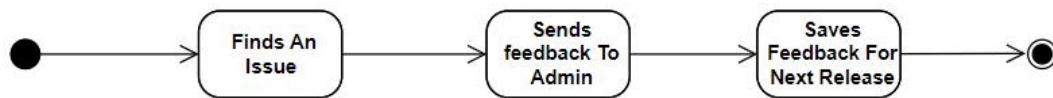
**Figure 13(Activity 5)**

Includes Use Cases : Browse House , Search House And , Save Post.



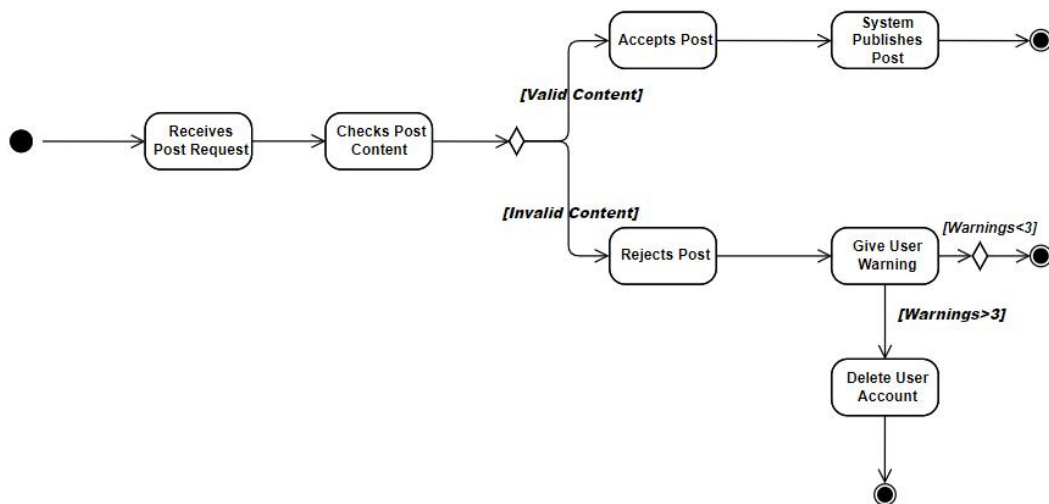
**Figure 14(Activity 6)**

Includes Use Cases : Browse Contacts And Send Message.



**Figure 15(Activity 7)**

Includes Use Cases : Send Feedback and , Receive Feedback.



**Figure 16(Activity 8)**

Includes Use Cases : Accept Post , Reject Post , Give Warning And , Delete User Account.

## CHAPTER 4: ARCHITECTURE AND DESIGN

### 4.1 Overview

Model-View-View Model (MVVM):

MVVM is a architecture that separates the user interface (View) from the application logic (View Model) and the data (Model). View Model handles the UI-related logic and manages the interaction between the View and the data layer.

Model: Represents the data and business logic.

View: Represents the UI and is responsible for displaying data and capturing user input.

View Model: Acts as an intermediary between the View and Model, handling UI-related logic and data manipulation. Software Architecture

MVVM vs. MVC

Data Binding:

MVC: In MVC, the View directly communicates with the Model, and any changes in the Model need to be manually updated in the View.

MVVM: MVVM incorporates a powerful feature called data binding, where the View is automatically updated whenever there is a change in the underlying data in the ViewModel. This reduces the boilerplate code required for manual updates.

Testability:

MVC: Testing can be challenging in MVC, as the Controller is tightly coupled with both the View and the Model.

MVVM: MVVM improves testability by isolating the UI-related logic in the ViewModel. This allows for more straightforward unit testing of the ViewModel, as it is decoupled from the actual View.

Maintainability:

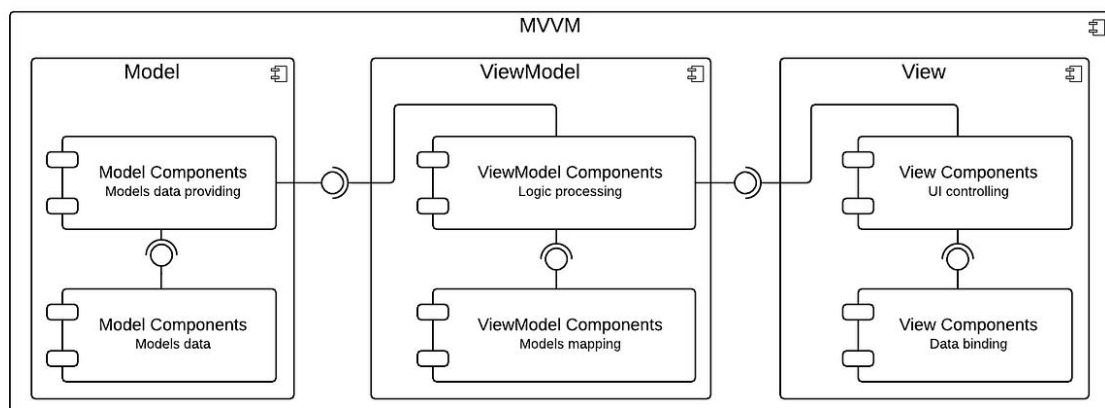
MVC: As applications grow, maintaining the MVC pattern might become complex due to potential dependencies between the View and Controller.

MVVM: The MVVM pattern promotes maintainability by clearly separating concerns. Changes in the View do not affect the ViewModel, and vice versa. This modularity simplifies maintenance and updates.

Binding Adapters:

MVVM: Android's MVVM architecture allows the use of binding adapters, enabling a more declarative UI design. Binding adapters link UI components directly to ViewModel properties, reducing the need for boilerplate code in the View.

The Following Diagram Is A Component Diagram That Represents How The MVVM Architecture Pattern Works And How Components Of This Architecture Pattern Communicates With Each Other:



**Figure 17 (MVVM Component Diagram)**

#### 4.1.1 Conceptual Diagram

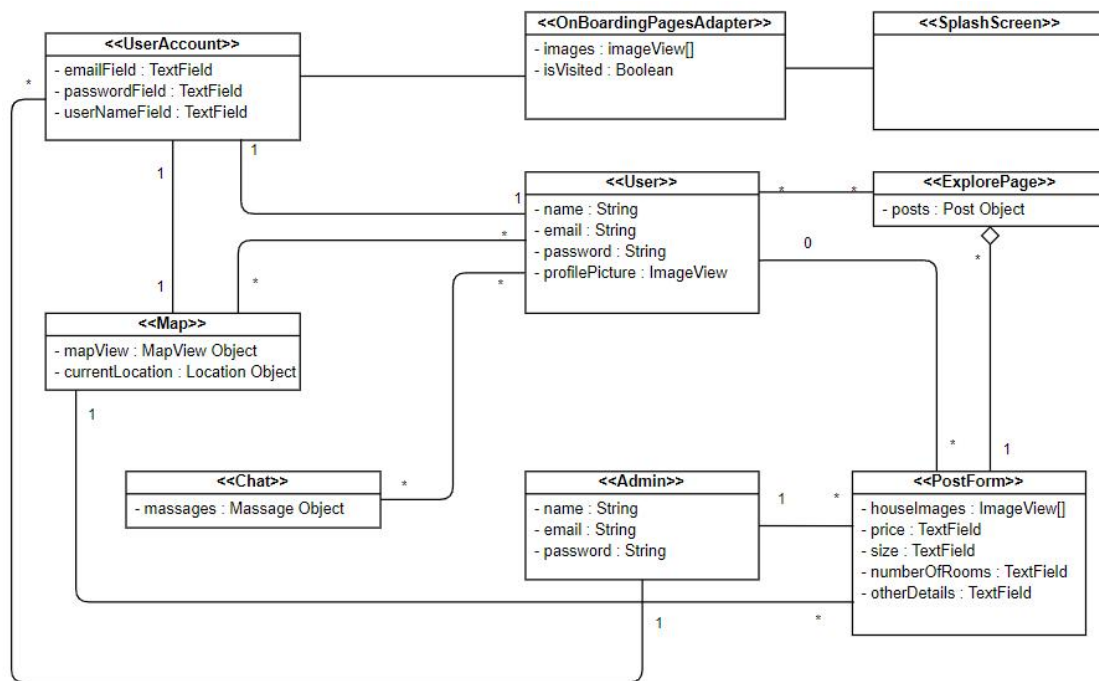
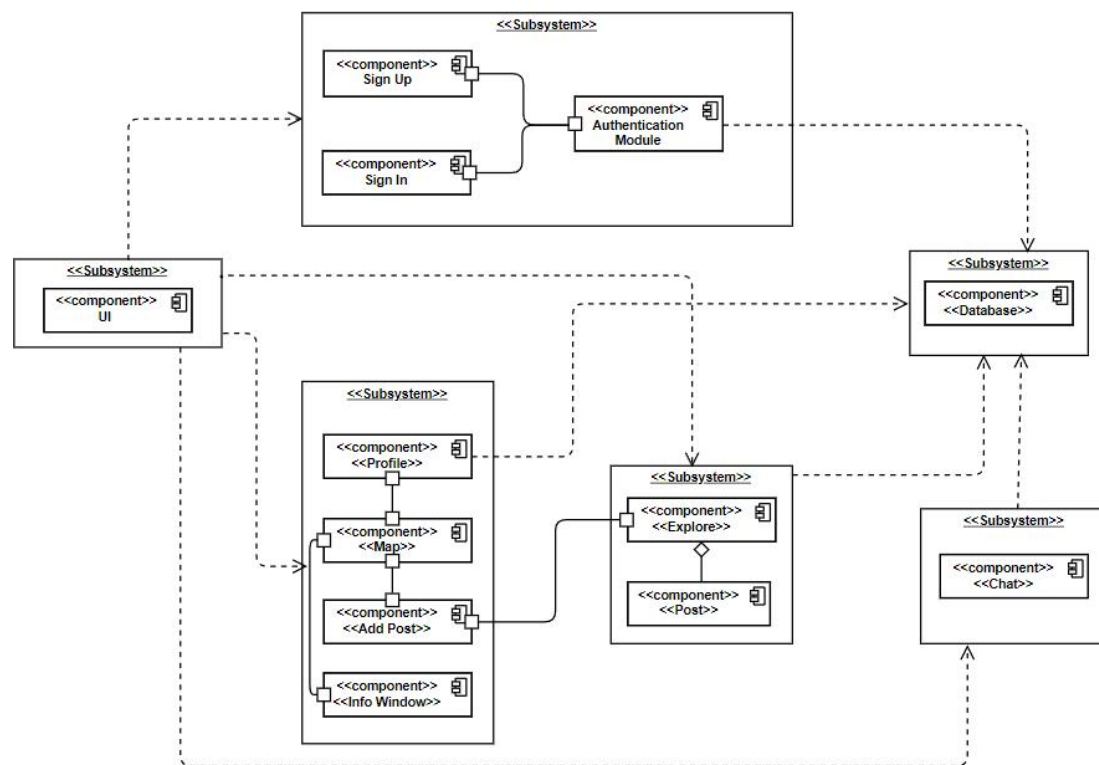


Figure 18(Conceptual Diagram)



#### 4.1.1 Logical view



**Figure 19(Component Diagram)**

#### 4.1.2 Process view

A sequence diagram is interaction diagram that illustrates how processes or objects interact in a particular scenario. It visualizes the dynamic behavior of a system over time, depicting the sequence of messages exchanged between different components or entities. Sequence diagrams are particularly useful in modeling and understanding the flow of interactions within a system, emphasizing the order and timing of these interactions.

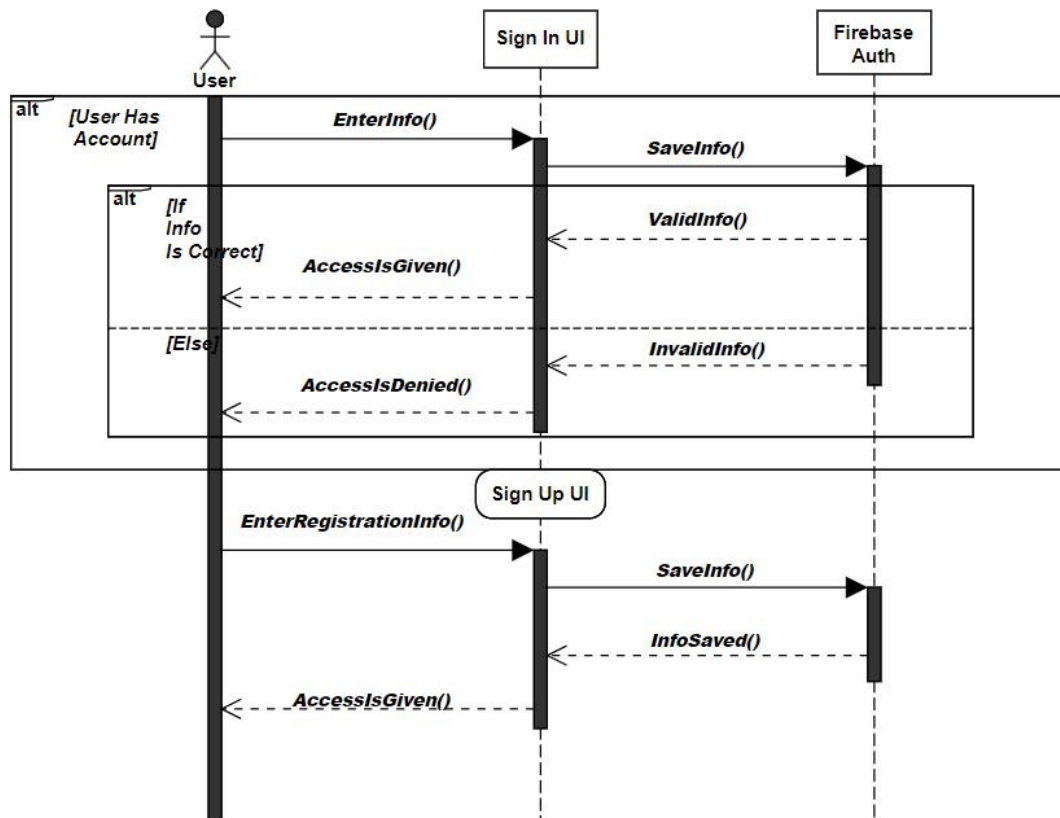


Figure 20(Sequence 1)

Includes Use Cases : Sign Up And Sign In Process.

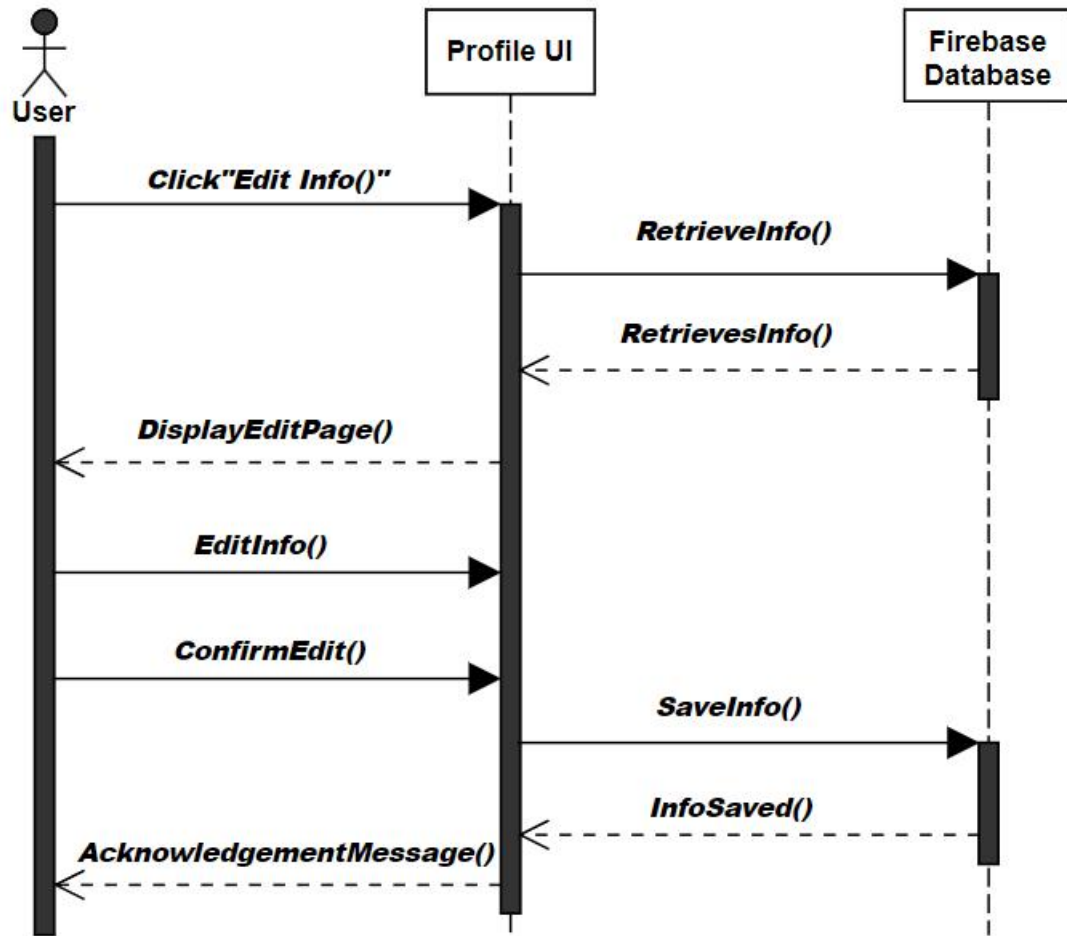


Figure 21(Sequence 2)

Includes Use Cases : Edit Info Process.

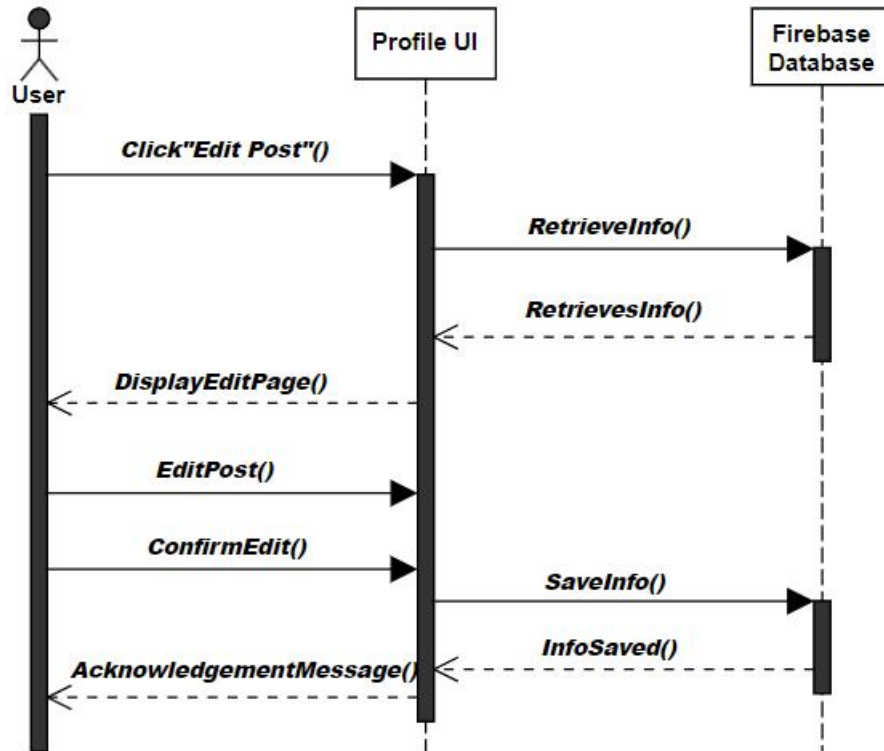


Figure 22(Sequence 3)

Includes Use Cases : Edit Post Process.

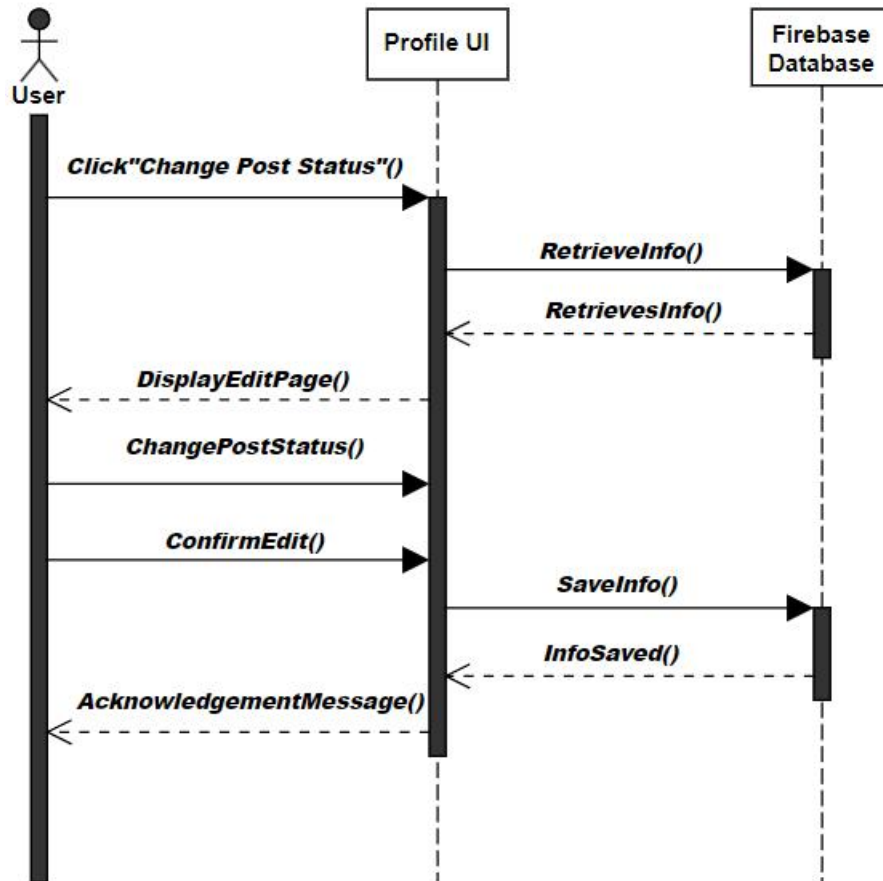


Figure 23(Sequence 4)

Includes Use Cases : Change Post Status Process.

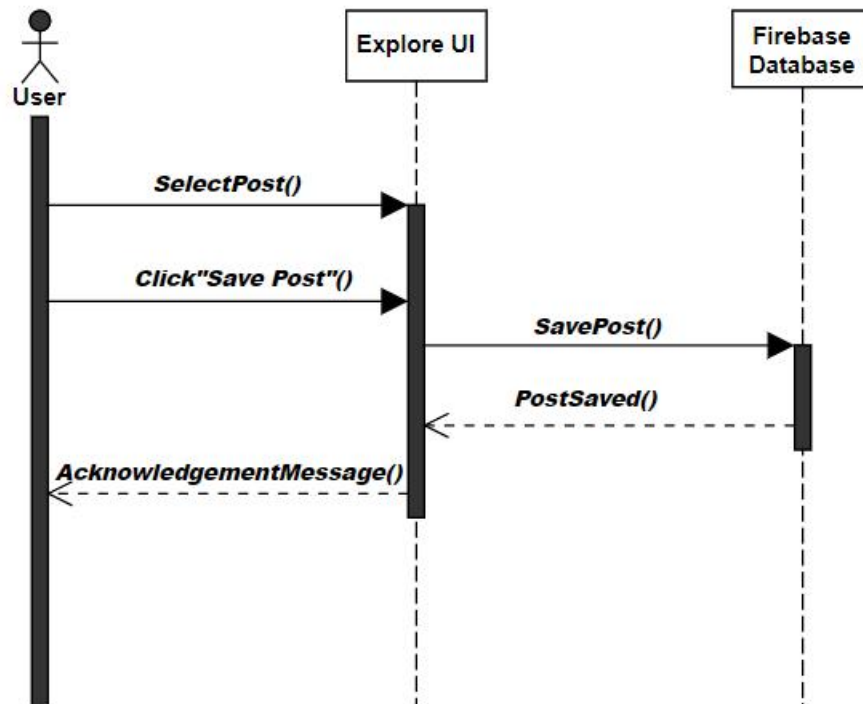
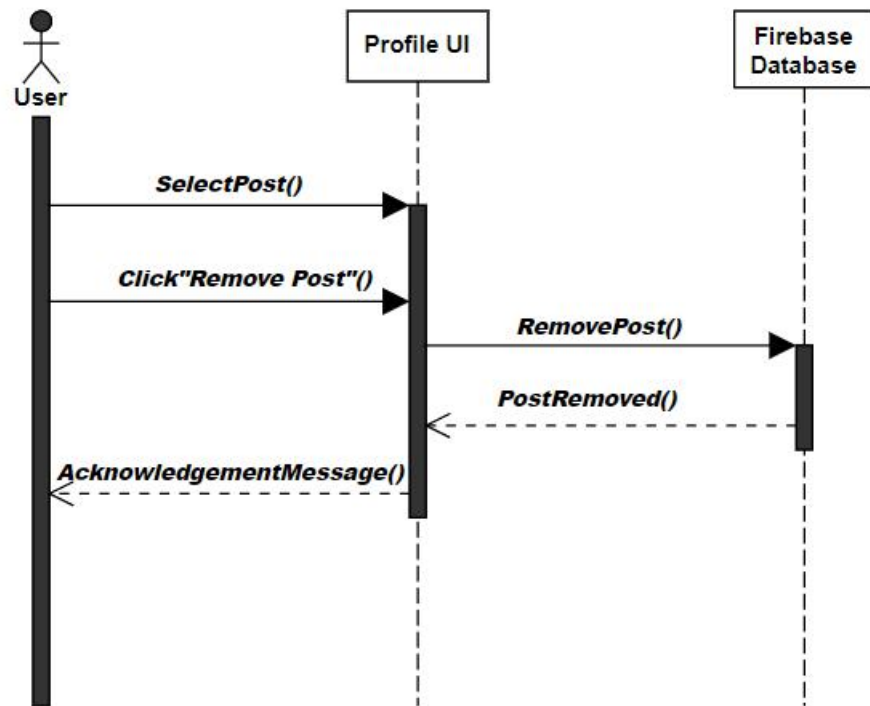


Figure 24(Sequence 5)

Includes Use Cases : Save Post Process.



**Figure 25(Sequence 6)**

Includes Use Cases : Remove Saved Post Process.

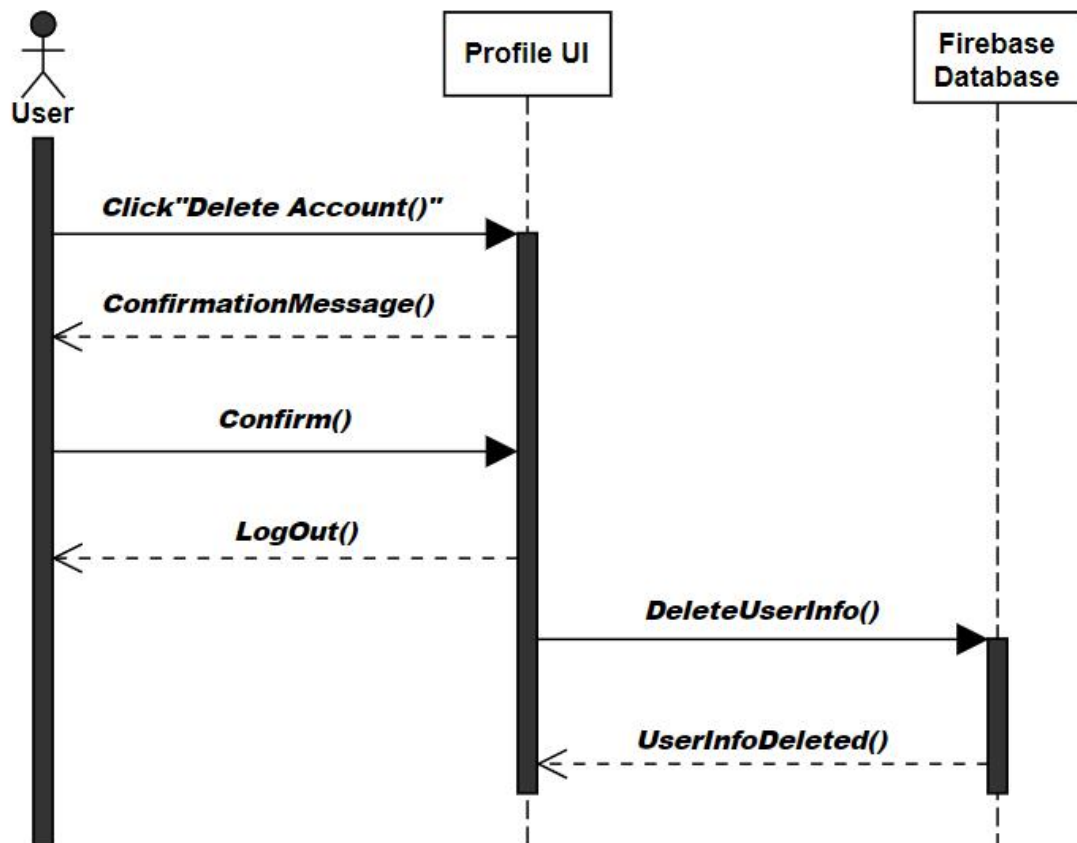


Figure 26(Sequence 7)

Includes Use Cases : Delete Account Process.



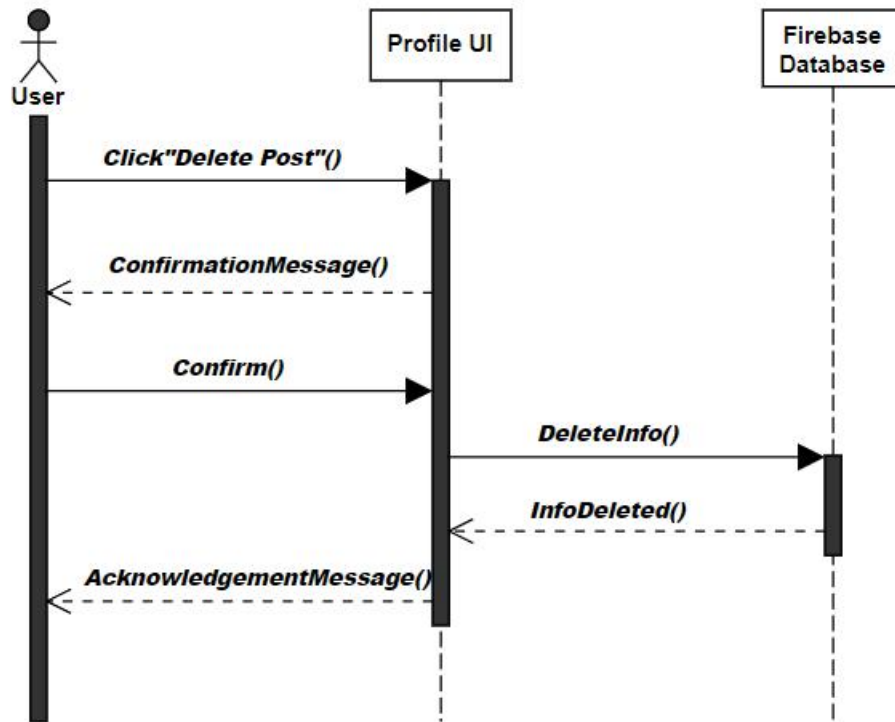


Figure 27(Sequence 8)

Includes Use Cases : Delete Post Process.

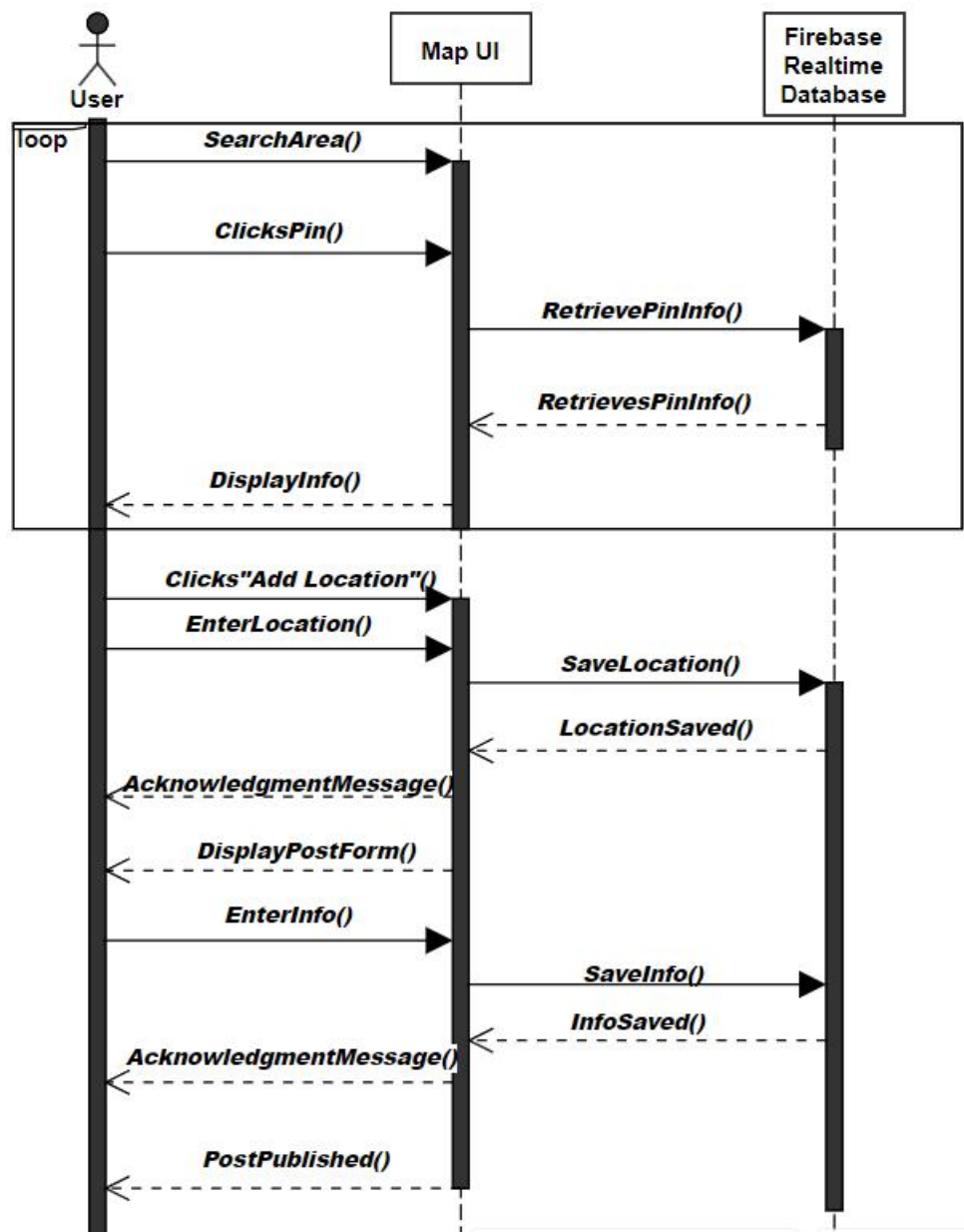


Figure 28(Sequence 9)

Includes Use Cases : Browse House , Search House , And Add Post Through Map Page.

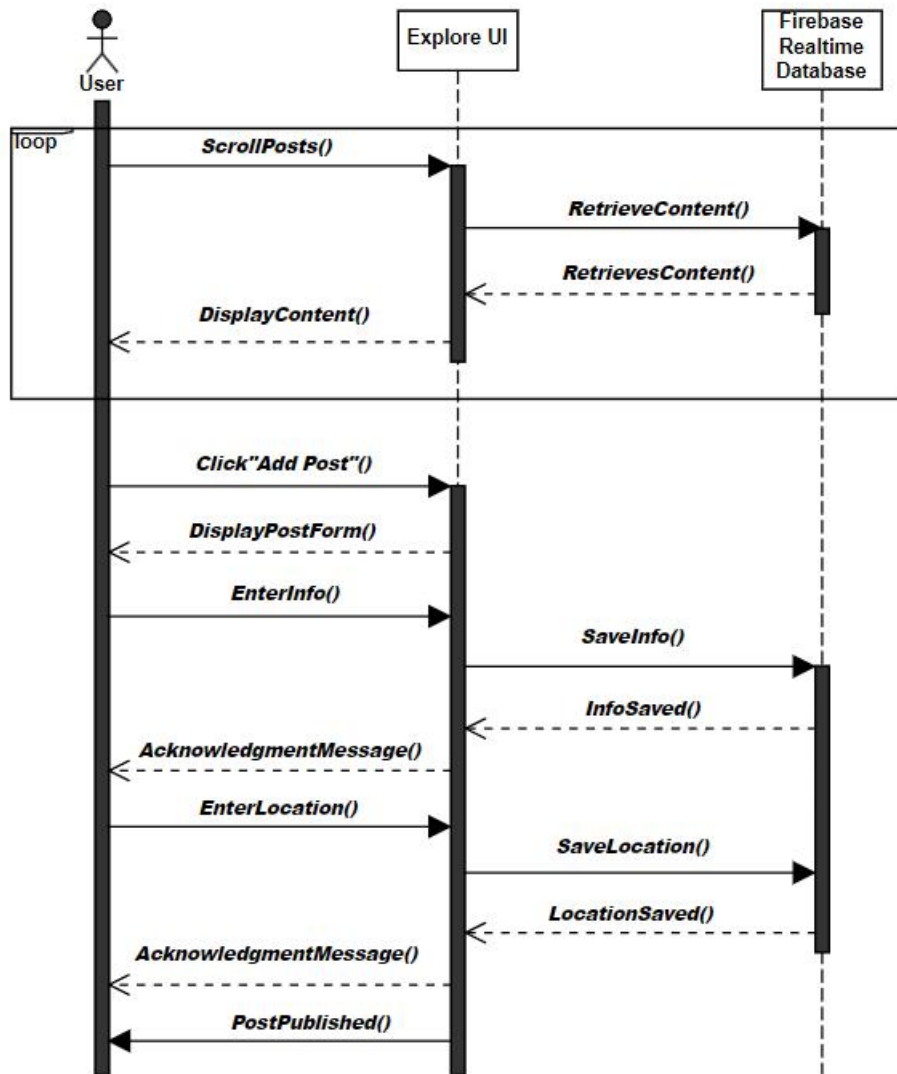


Figure 29(Sequence 10)

Includes Use Cases : Browse House , Search House , And Add Post Through Explore (Post Page) Page.

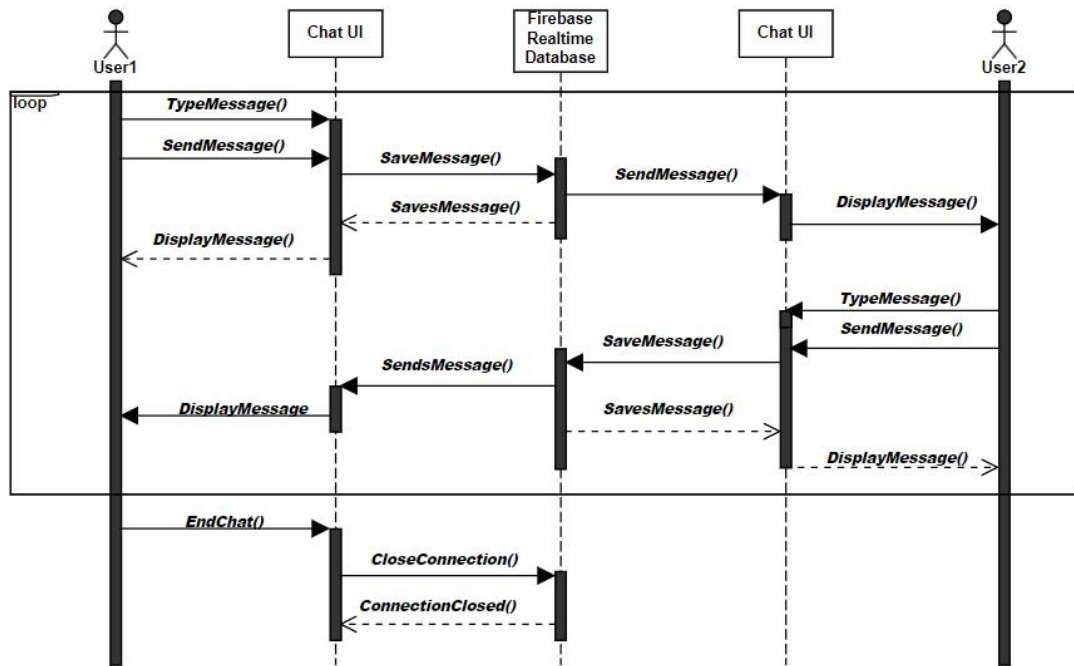


Figure 30(Sequence 11)

Includes Use Cases : Browse Contacts , Send Messages .

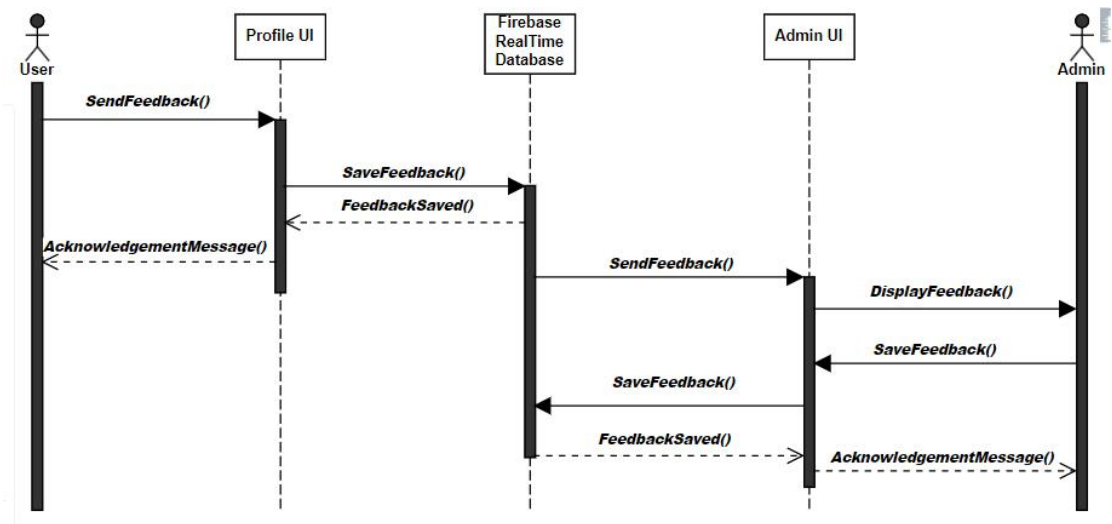
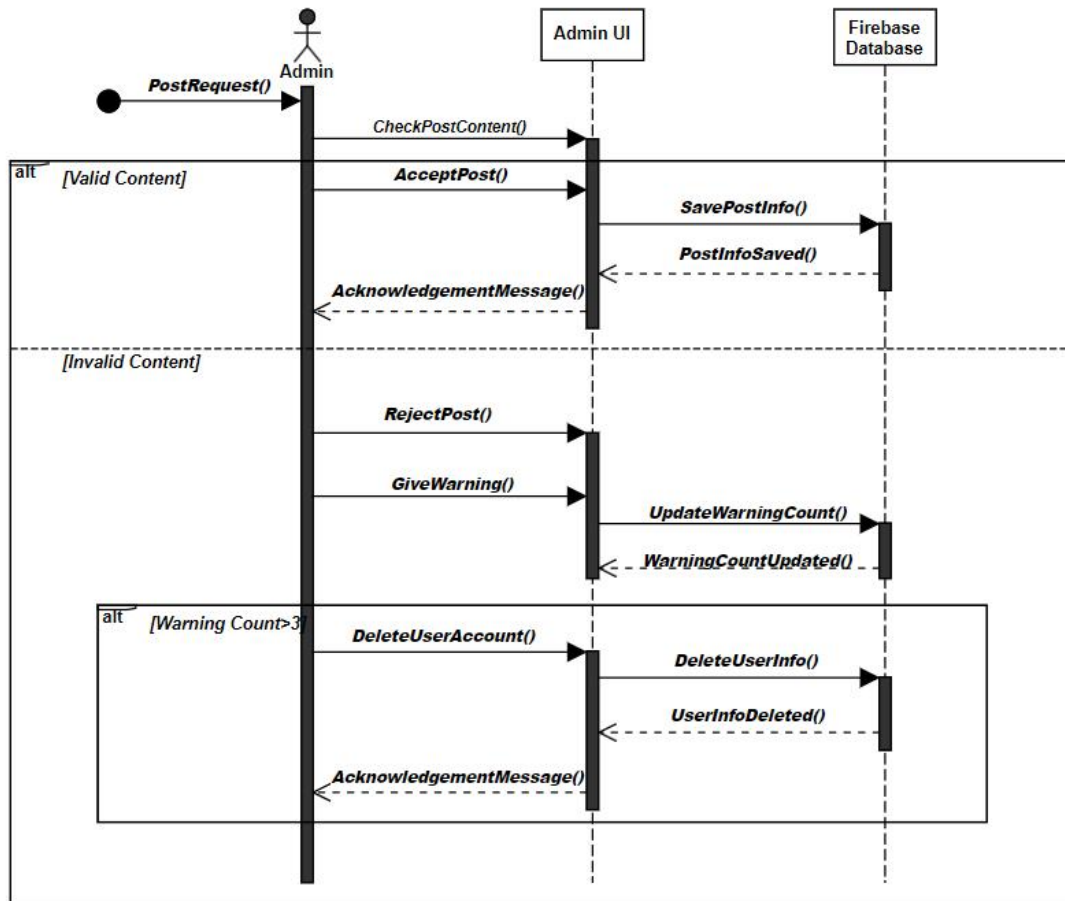


Figure 31(Sequence 12)

Includes Use Cases : Send Feedback And , Receive Feedback .



**Figure 32(Sequence 13)**

Includes Use Cases : Accept Post , Reject Post , Give Warning and , Delete User Account.

#### 4.1.3 Physical view

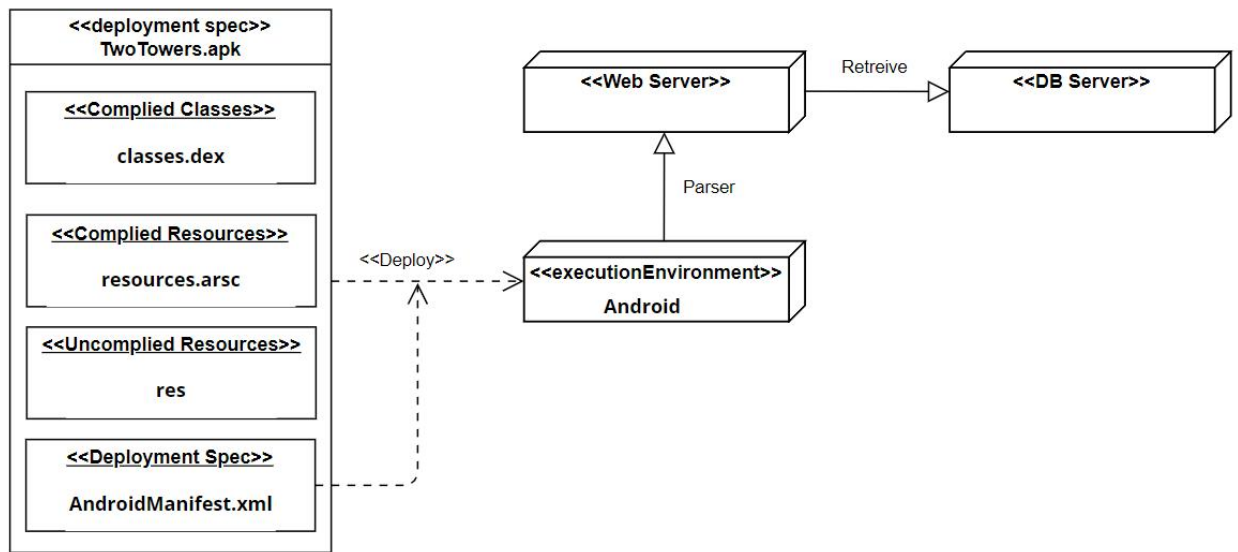
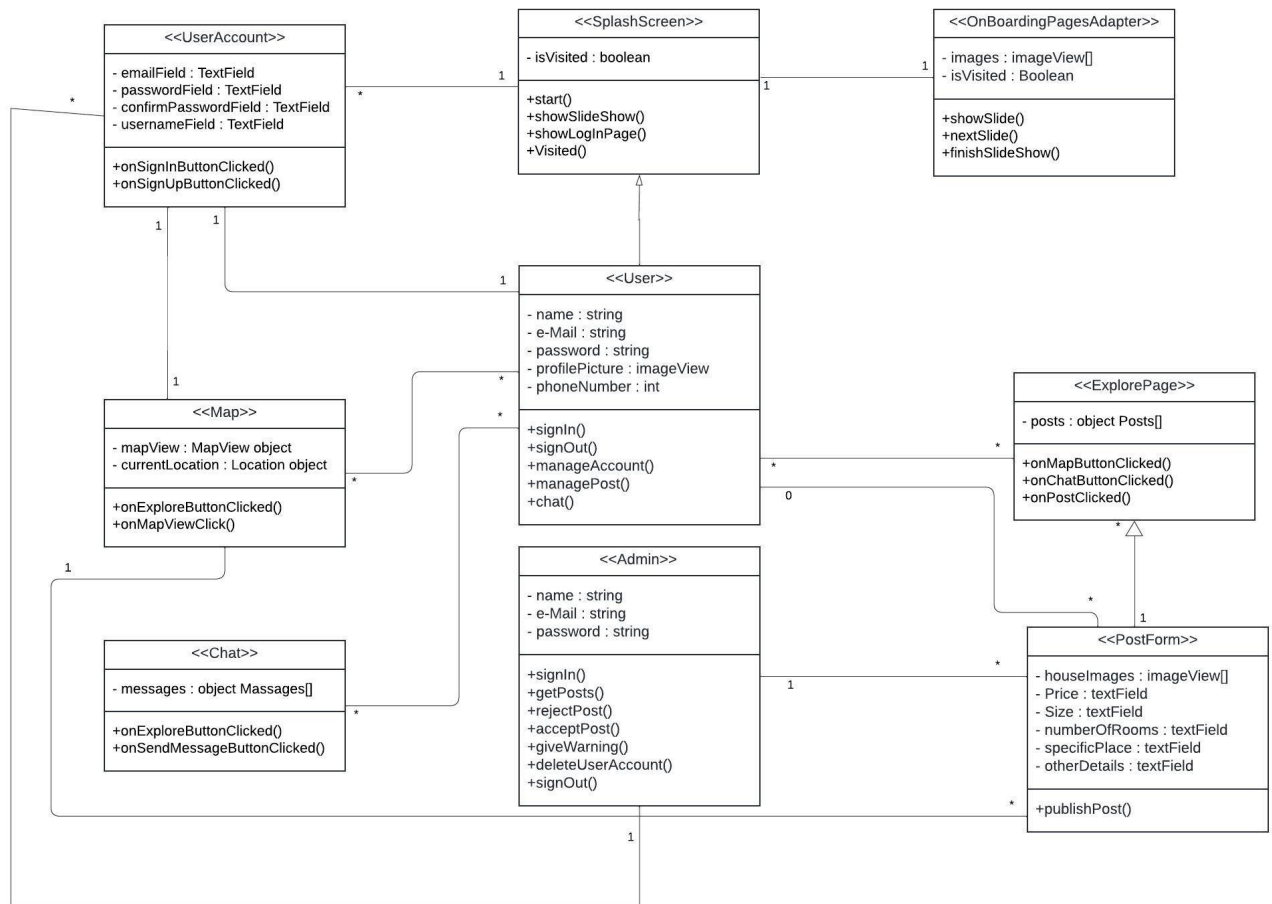


Figure 33(Deployment Diagram)

#### 4.2 Software design

### 4.2.1 Class diagram

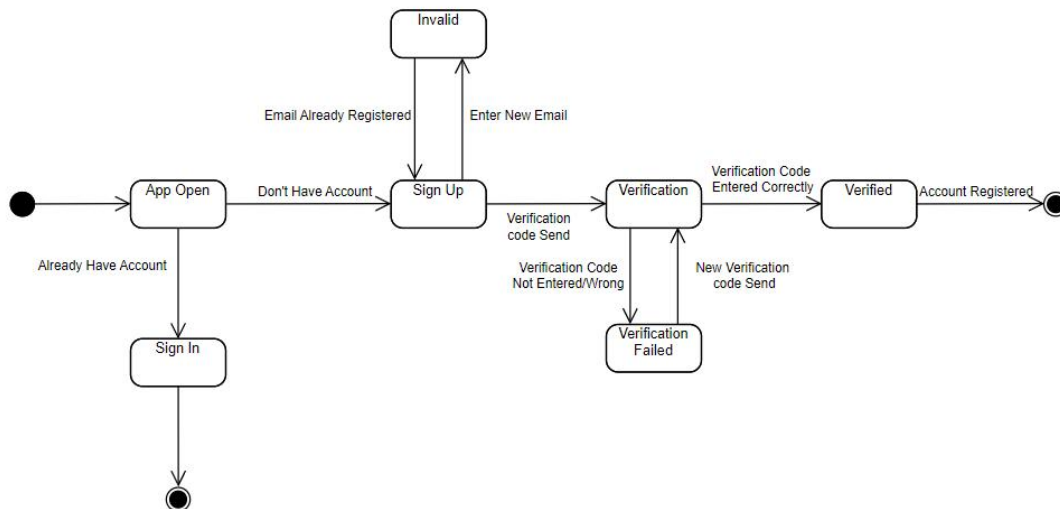


**Figure 34(Class Diagram)**

### 4.2.2 State transition diagram

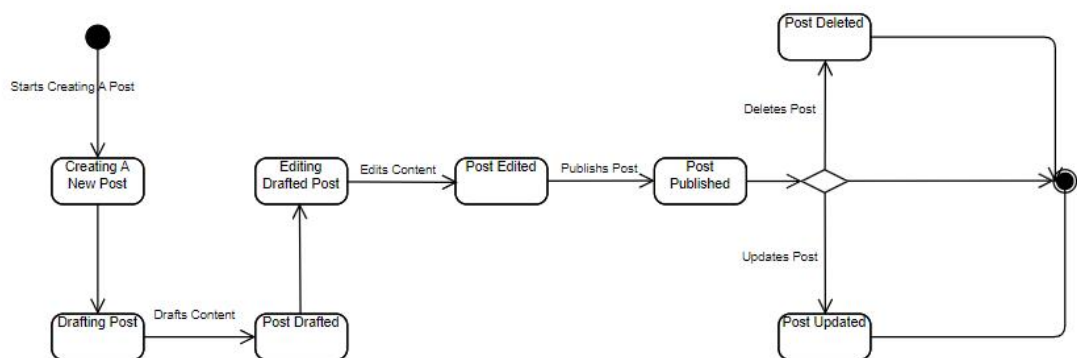
State diagram is a diagram that represents the dynamic behavior of a system or entity by depicting its states, state transitions, and events triggering those transitions. State diagrams are widely used in software engineering and systems modeling to describe the various states an object or system can exist in and how it transitions between these states in response to events.





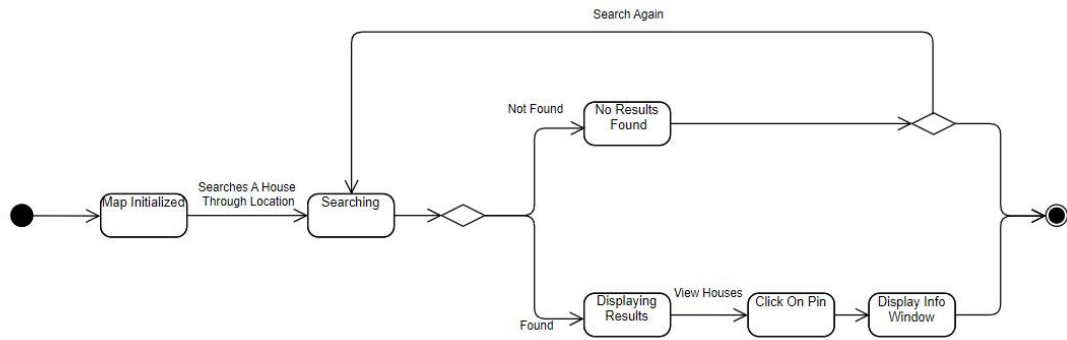
**Figure 35(State 1)**

This State Diagram Includes : Sign In And Sign Up.



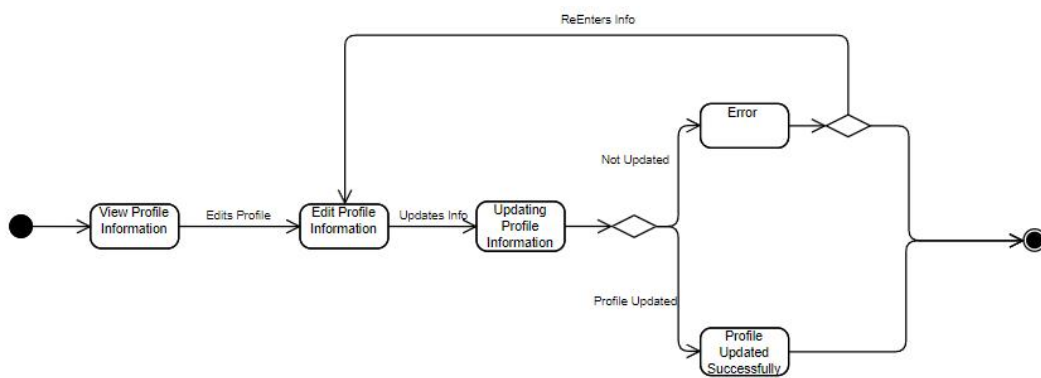
**Figure 36(State 2)**

This State Diagram Includes : Create Post , Edit Post And , Delete Post.



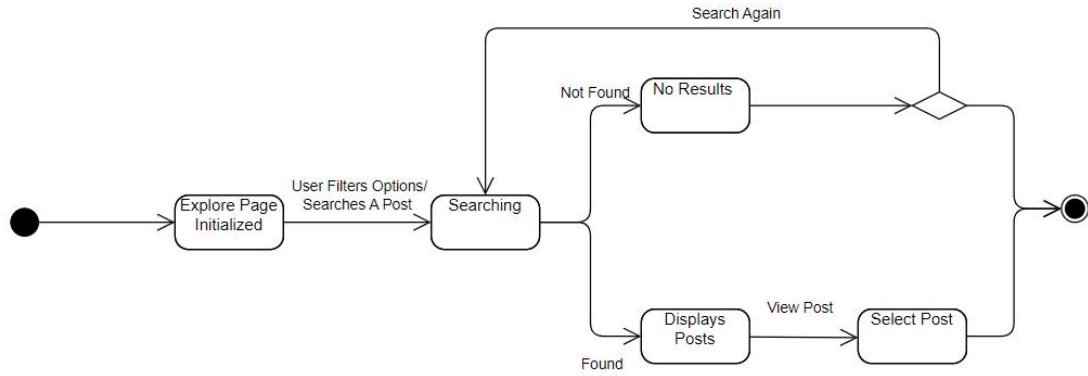
**Figure 37(State 3)**

This State Diagram Includes : Browse House And Search House Through Map Page.



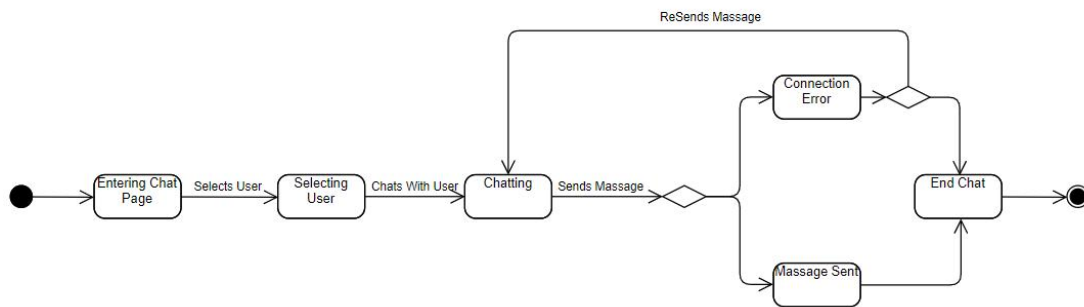
**Figure 38(State 4)**

This State Diagram Includes : Manage Account , Change Password.



**Figure 39(State 5)**

This State Diagram Includes : Browse House And Search House Through Explore Page.



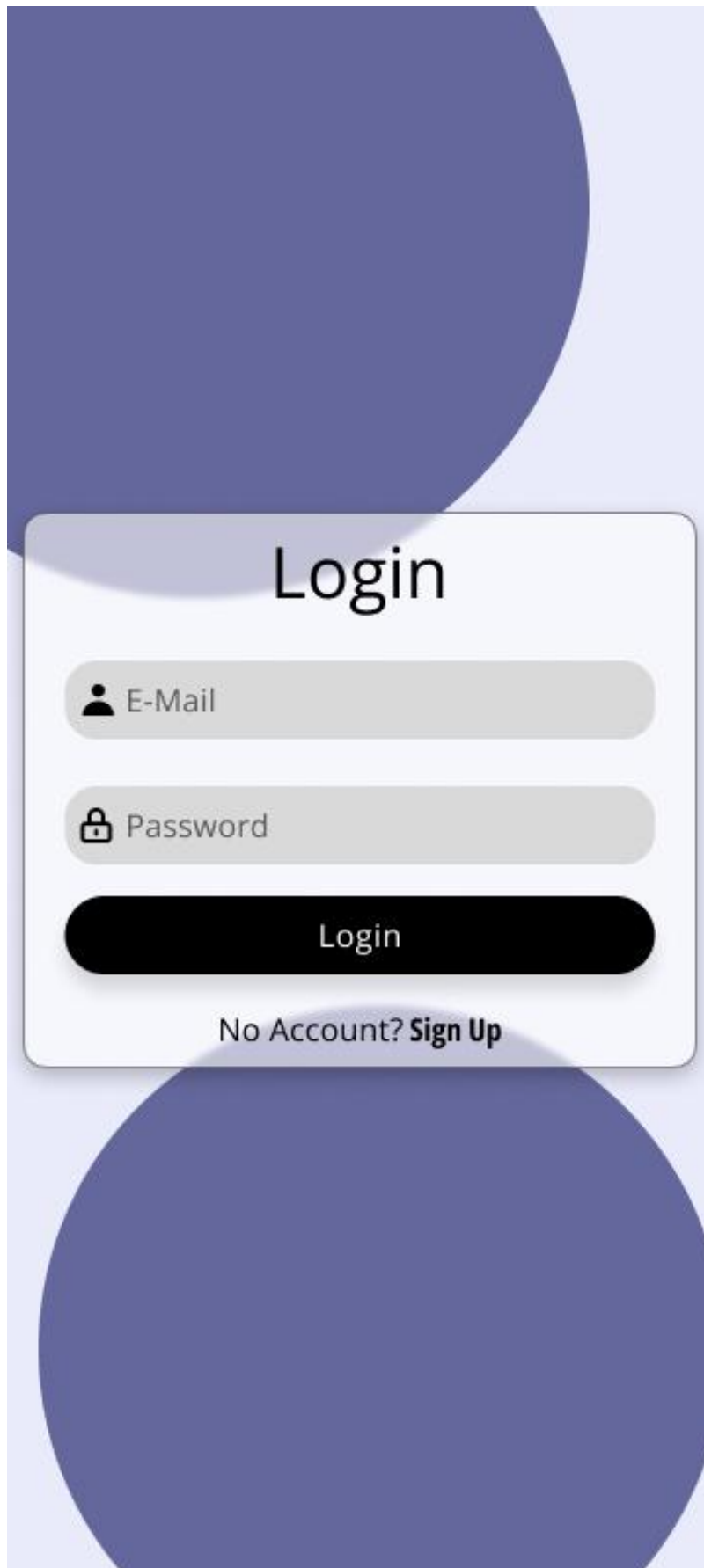
**Figure 40(State 6)**

This State Diagram Includes : Chat With Users , Browse Contacts , Send Message And , Receive Message.


#### 4.3 User interface design (prototype)




Figure 41(Map UI)

The image shows a login interface centered on a light blue background with two large, overlapping purple circles. The login form is a white rounded rectangle with a subtle drop shadow. It features the title 'Login' in a large, black, sans-serif font. Below the title are two input fields: the first is labeled 'E-Mail' with a person icon, and the second is labeled 'Password' with a lock icon. Both fields have a light gray border and a rounded bottom. Below these fields is a solid black button with the word 'Login' in white. At the bottom of the form, the text 'No Account? Sign Up' is displayed, with 'Sign Up' in a bold font.

Login

 E-Mail

 Password

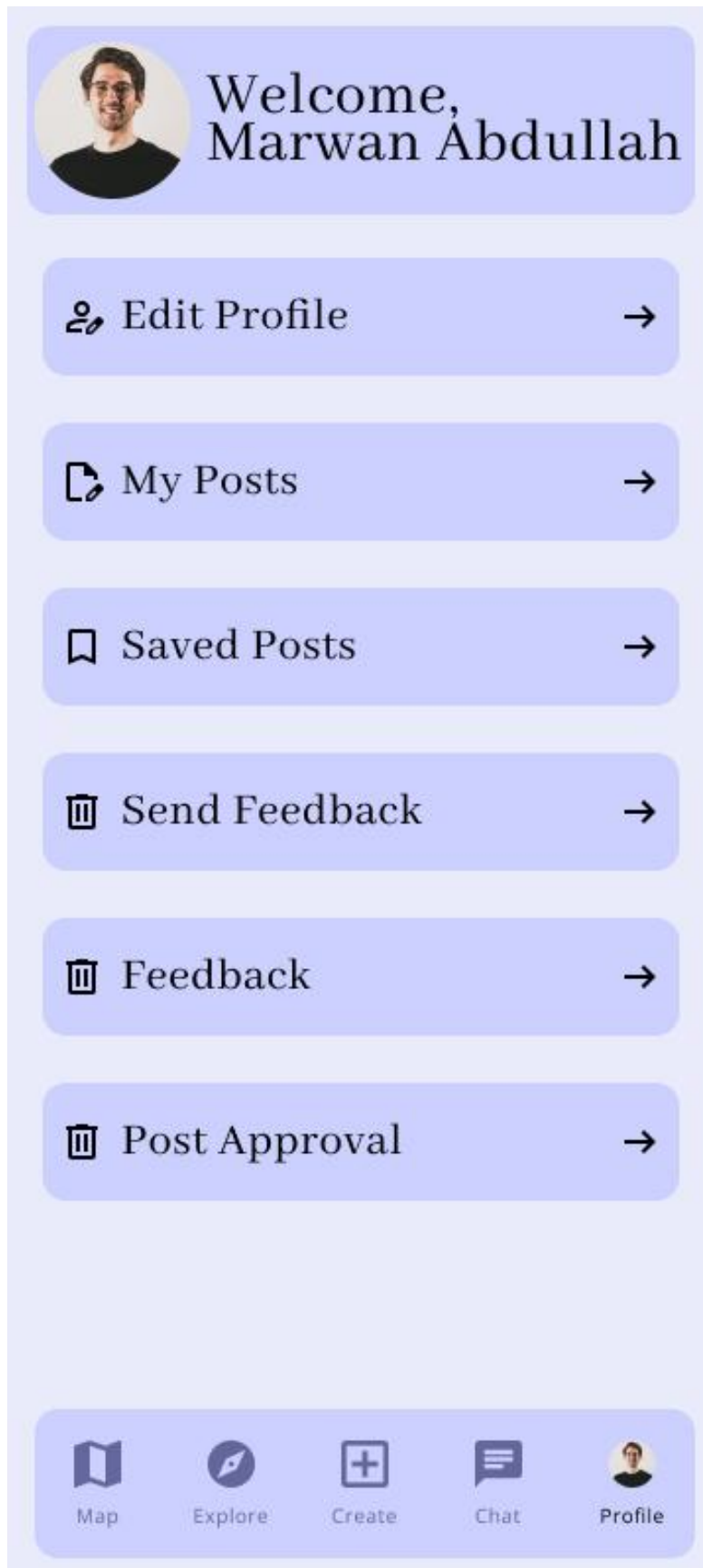
Login

No Account? **Sign Up**

Figure 42(Login UI)



Figure 43(Post UI)



**Figure 44(Chat UI Experimental)**





## **CHAPTER 5: IMPLEMENTATION PLAN**

### **5.1 Description of Implementation**

Major Tasks Involved:

**Front-end Development:** Develop the user interface components, including interactive maps, post pages, and chat functionalities, ensuring intuitive navigation and user-friendly design.

**Back-end Integration:** Integrate the front-end with the back-end server to facilitate data exchange, user authentication, and communication between different app modules.

**Database Management:** Utilize Firebase Database to store and manage user profiles, posts, house listings, chat messages, and administrative data securely.

**Authentication Setup:** Implement Firebase Authentication to enable secure user authentication and authorization processes, including login, sign up, and password management.

**Geo-location Services:** Incorporate Geo-location services to dynamically display nearby empty houses on the map, allowing users to explore available listings based on their current location.

**Administrative Controls:** Develop administrative functionalities to enable moderators to review, approve, or delete user-generated content, enforce community guidelines, and manage user strikes if necessary.

**Resource Requirements:**

Hardware: Standard development hardware (computers) for development and testing purposes.

Software: Android Studio for app development, Firebase services for back-end support, including Firebase Database and Firebase Authentication.

Facilities: Adequate workspace with internet connectivity for development and collaboration.

**Personnel:**

Android developers for front-end development.

Back-end developers proficient in Kotlin and experienced with Firebase for back-end integration.

UI/UX designers to ensure an appealing and user-friendly interface.

**Site-Specific Implementation Requirements:**

Location Services Integration: Ensure compatibility with location services to accurately identify and display nearby empty houses.

Scalability Considerations: Design the system to accommodate potential expansion to cover additional regions or scale up user base without compromising performance.

User Training and Support: Provide user documentation and support resources to assist users in navigating the app and utilizing its features effectively.

**Transition Plan:**

Transition plan objective: Ensure a smooth transition from the development phase to the operational phase.

Conduct pre-deployment testing to ensure the application functions correctly in the production environment.

- Generating a release build of Two Towers as an Android App Bundle (AAB) for optimized distribution.
- Set Two Towers pricing (free) and target geographic locations for distribution.
- Submit Two Towers for review by the Google Play team. This may take a few days.
- Address any feedback or concerns raised during the review process.
- Once approved, the app will be published on the Play Store.

Conduct post-deployment testing to validate system functionality and performance in the production environment.

Monitor app performance using Play Console analytics to track downloads, engagement, and crashes.

## 5.2 Programming Language and Technology

### **Programming Languages:**

Kotlin: Primary programming language for Android app development, offering modern features and seamless integration with Android SDK.

### **Technologies and Frameworks:**

Android SDK: Software development kit for building Android applications, providing essential tools, libraries, and APIs.

Firebase: Back-end services platform, offering various features such as Firebase Authentication, Firebase Database, and Firebase Cloud Messaging.

Firebase Authentication: Commercial off-the-shelf (COTS) for user authentication.

Firebase Database: COTS for real-time database management.

Firebase Cloud Messaging: COTS for push notifications and messaging.

Google Maps API: Integration of maps and Geo-location services, allowing for dynamic mapping and location-based functionalities.

Firebase Cloud Functions: Server-less compute service for running back-end code in response to events, facilitating back-end logic execution without managing servers.

### **Software and Databases:**

Android Studio: Integrated development environment (IDE) for Android app development, offering tools for coding, debugging, and testing.

Firebase: No-SQL cloud database for storing structured data, offering scalability and real-time synchronization capabilities.

Figma: Software for designing and prototyping user interfaces, facilitating the creation of intuitive and visually appealing app layouts.

### 5.3 part of implementation if possible

```
private val failure : Boolean = false

override fun login(email: String, password: String, callback: (success: Boolean) ->
Unit) {
    if (email.isNullOrEmpty() || password.isNullOrEmpty()) {
        callback(failure)
    } else {
        auth.signInWithEmailAndPassword(email,
password).addOnCompleteListener { task ->
            callback(task.isSuccessful)
        }
    }
}
```

#### **Login Method**

```

        override fun registration(userName: String, email: String, password: String,
callback: (success: Boolean) -> Unit) {

if(email.isNullOrEmpty()||password.isNullOrEmpty()||userName.isNullOrEmpty()){

    callback(failure)

} else {

    auth.createUserWithEmailAndPassword(email,
password).addOnCompleteListener { task ->

        callback(task.isSuccessful)

        if (task.isSuccessful) {

            val user = auth.currentUser

            user?.let { firebaseUser ->

                saveUserData(firebaseUser.uid, userName, email)

            }

        }

    }

}

}
}

```

### **Registration Method**

```

class GoogleMapsRepository (private val activity: AppCompatActivity) {

    private lateinit var googleMap : GoogleMap

    fun initializeGoogleMap(onMapReady :(GoogleMap)->Unit){

    val supportMapFragment =

        activity.supportFragmentManager.findFragmentById(R.id.mapView)      as
        SupportMapFragment

        supportMapFragment.getMapAsync { map->

            googleMap = map

            onMapReady(googleMap)

        }

    }

}

```

### **Google Maps**

```

override fun storeImages(imagesUri: List<Uri>){

    val reference = FirebaseStorage.getInstance().reference

    val userRef = reference.child("${user?.uid}")

    imagesUri.forEachIndexed { index,uri->

        val fileName = "${postId}_images_$index.jpg"

        val userPostRef = userRef.child(fileName)

        val uploadTask = userPostRef.putFile(uri)


        uploadTask.addOnSuccessListener {

            Log.e("Tag","Success")

        }

        uploadTask.addOnFailureListener{

            Log.e("Tag","Failure")

        }

    }

}

```

### **Store Images Method**



## CHAPTER 6: TESTING PLAN

### **Scope:**

The testing activities aim to ensure the reliability, functionality, and usability of the Android Kotlin app across different devices and scenarios. It covers both front-end and back-end components, including user features, administrative controls, and data management functionalities.

### **Approach:**

The testing approach follows a manual testing methodology to evaluate the app's performance. Testing tasks include functional testing, usability testing, and security testing.

### **Resources:**

#### **Personnel:**

**Testers:** Responsible for executing test cases, reporting bugs, and providing feedback.

**Developers:** Collaborate with testers to address identified issues and ensure the app meets quality standards.

#### **Tools:**

**Testing Devices:** Various Android smartphones and emulators to test compatibility across different screen sizes and hardware configurations.

**Testing Frameworks:** J-Unit for unit testing, Firebase Test Lab for automated testing on real devices.

**Schedule:**

The testing activities will be conducted throughout the development life-cycle, with specific milestones for different testing phases. Continuous integration and deployment practices will enable frequent testing cycles to ensure early detection and resolution of issues.

**Test Activities:****Test Items:**

1. User Authentication
2. User Profile Management
3. House Listing Display
4. Map Functionality
5. Chat Feature
6. Post Creation and Management
7. Administrative Controls
8. Data Security and Privacy

**Testing Tasks:**

**Develop Test Cases:** Define test scenarios and create test cases covering most of the features and functionalities.

**Execute Test Cases:** Run test cases manually and automate where feasible to validate app behavior.

**Regression Testing:** Perform regression testing to ensure new developments do not introduce regressions.

**Security Testing:** Conduct security assessments to identify and mitigate potential vulnerabilities.

**Performance Testing:** Evaluate app performance under various conditions, including network fluctuations and high user loads.

**Test Coverage:**

**Functional Coverage:** Ensuring all features and functionalities work as intended.

**UI/UX Coverage:** Assessing user interface elements for consistency, accessibility, and responsiveness.

**Security Coverage:** Identifying and mitigating potential security vulnerabilities.

ID	Requirements Description	Test Case Description	Test Status
1	Edit Profile	Update User Profile	Success
2	Delete Account	User Deletes Account	Success
3	Delete Post	User Deletes Post	success
4	Saved Posts	User UN-Saves A post	Success
5	Saved Posts	User Saves A Post	Success
6	Search Users	User Searches Another User	Success
7	Filter Posts	Users Filter Houses	Success
8	Create Post	User Adds A Post	Success
9	Send Message	User Compose A New Message	Success
10	Send Feedback	User Sends A Feedback	Success
11	Receive Feedback	Admin Receives Feedback	Success
12	Accept Post	Admin Accept Posts	Success
13	Reject Post	Admin Reject Post	Success
14	Warning Post	Admin Reject Post And Adds a Warning	Success
15	Delete User	Warning Limit Reached Admin Deletes User	Success

**Table 18**

### **Tester Independence:**

Testers will maintain a degree of independence from the development team to ensure unbiased evaluation of the app's quality. However, collaboration between testers and developers is essential for effective issue resolution and continuous improvement.

### **Test Environment:**

**Development Environment:** Android Studio for app development and debugging.

**Testing Environment:** Real Android devices and emulators for testing across different configurations.

**Production Environment:** Firebase back-end services for testing app behavior in a production-like environment.

### **Test Design Techniques:**

**SonarLint :** is a static code analysis tool that integrates directly into Android Studio, allowing the identification and fix of issues in the code as it is being written. It help identify bugs, code smells, security vulnerabilities, and potential performance optimizations

**Android Studio Built in Lint :** offers almost the same features as SonarLint but with less customization for the user making it less effective.

**Equivalence Partitioning:** Dividing input data into equivalent partitions to ensure thorough test coverage.

**Boundary Value Analysis:** Testing boundary conditions to identify potential errors at the edges of input ranges.

### **Entry and Exit Criteria:**

### **Entry Criteria:**

1. Completion of development tasks and code review.
2. Availability of test environments and necessary testing tools.
3. Test plan and test cases prepared and reviewed.

### **Exit Criteria:**

- 1) Successful execution of all test cases with minimal critical defects.
- 2) Compliance with predefined performance benchmarks and security standards.

### **Rationale for Choices:**

The chosen testing approach and activities aim to ensure comprehensive evaluation of the app's functionality, and security. By incorporating a mix of manual and automated testing techniques, the test plan strives to achieve thorough test coverage while maintaining efficiency and agility throughout the development process.

### **Black-box**

Equivalence Partitioning:

#### **Test Cases:**

##### **User Authentication:**

<b>ID</b>	<b>Test Case Description</b>	<b>Result</b>
<b>1</b>	Valid email , password , and username.	Success
<b>2</b>	Invalid username with valid password , and email.	Fail To Authenticate
<b>3</b>	Valid username , and email with invalid password.	Fail To Authenticate
<b>4</b>	Valid username , and password with invalid email	Fail To Authenticate
<b>5</b>	Empty username , password fields , and email.	Fail To Authenticate
<b>6</b>	Long usernames , passwords , and emails.	Fail To Authenticate

**Table 19**

**Explore:**

<b>ID</b>	<b>Test Case Description</b>	<b>Result</b>
<b>1</b>	Search for houses within a specific username.	Success
<b>2</b>	Filter houses by location or amenities.	Success
<b>3</b>	View houses with varying descriptions and images.	Success
<b>4</b>	Add posts to saved	Success
<b>5</b>	Move between posts in map and explore	Success

**Table 20****Chat Feature:**

<b>ID</b>	<b>Test Case Description</b>	<b>Result</b>
<b>1</b>	Send and receive text messages.	Success
<b>2</b>	Send emojis.	Success
<b>3</b>	Test message delivery under poor network conditions.	Success
<b>4</b>	Send Both Arabic and English Texts	Success

**Table 21**

### Create Post:

ID	Test Case Description	Result
1	Check edge values of price and area	Success
2	Check empty fields	Success
3	Check photos admission	Success

**Table 22**

Boundary Value Analysis:

### Test Cases:

#### Map Functionality:

ID	Test Case Description	Result
1	Test zoom levels of the map.	Success
2	Verify pin placement accuracy at map edges.	Success
3	Test map responsiveness when panning across different locations.	Success
4	Search A Place.	Success
5	View houses with varying descriptions and images.	Success

**Table 23**

### White-box

**Branch Coverage:** Branch coverage ensures that each branch or decision point in the code is evaluated both true and false during testing.



**Sample Test Case:****Login:**

ID	Test Case Description	Input	Condition	Result
1	Valid Login Credentials	Valid email and password	User attempts to log in	Verify that the login status is set to SUCCESS
2	Invalid Email	Invalid email format	User attempts to log in	Verify that the login status is set to FAILURE
3	Invalid Password	Valid email and an invalid password	User attempts to log in	Verify that the login status is set to FAILURE
4	Empty Email	Empty email field	User attempts to log in	Verify that the login status is set to FAILURE
5	Empty Password	Valid email and empty password field	User attempts to log in	Verify that the login status is set to FAILURE
6	Invalid Email and Password	Invalid email and password combination	User attempts to log in	Verify that the login status is set to FAILURE

**Table 24**

**Map:**

ID	Test Case Description	Input	Condition	Result
1	Test Saving Location to Database	LatLng, post ID, user ID	A Post Is Added	Location Is Stored
2	Test Retrieving Location	-	A Location Is Stored	A Pin Displayed In The Location
3	Test Retrieving Current User Image	-	A User Image Is Stored	A Image Is Displayed On The Nav Bar
4	Test Warning Count	-	More Than 2 Warnings	User Account Is Deleted
5	Test Zooming on Selected Place	Current User Location	Location Permission Is Given	Map Initialized With Current User Location
6	Test Retrieving Post	-	A Post Is Stored	A Post Is Displayed On The Pin

**Table 25**

## Testing automation

**Mockito:** is a open-source Java framework used for creating and working with mock objects in unit tests.

**Mocking:** Mockito allows you to create mock objects that simulate the behavior of real objects. These mock objects can be configured to return specific values or perform specific actions when their methods are called.

**Stubbing:** With Mockito, you can specify the behavior of mock objects by stubbing their methods. This means you can define what values or exceptions should be returned when specific methods are invoked.

**Annotations:** Mockito supports annotations to simplify test setup. Annotations such as `@Mock`, `@Spy`, `@InjectMocks`, and `@Captor` can be used to create mock objects, spy objects, inject dependencies, and capture method arguments for verification.

```
class FirebaseDatabaseLocationManagerTest {  
  
    @Mock  
  
    private lateinit var mockFirebaseDatabase: FirebaseDatabase  
  
    @Mock  
  
    private lateinit var mockDatabaseReference: DatabaseReference  
  
    Private lateinit var  
  
    firebaseDatabaseLocationManager: FirebaseDatabaseLocationManager  
  
    @Before  
  
    fun setUp() {  
  
        MockitoAnnotations.initMocks(this)  
  
        `when`(mockFirebaseDatabase.reference).thenReturn(mockDatabaseReference)  
  
        firebaseDatabaseLocationManager =  
        FirebaseDatabaseLocationManager(mockFirebaseDatabase)
```

```
}
```

```
@Test
```

```
fun testRetrieveLocation_Success() {
```

```
    val mockDataSnapshot = mock(DataSnapshot::class.java)
```

```
    val postId = "postId"
```

```
    val userId = "userId"
```

```
    val description = "Sample description"
```

```
    val lat = 12.34
```

```
    val lng = 56.78
```

```
    `when`(mockDatabaseReference.addValueEventListener(any())).thenAnswer {
```

```
        val listener = it.arguments[0] as ValueEventListener
```

```
        val mockPostReference = mock(DatabaseReference::class.java)
```

```
        `when`(mockPostReference.key).thenReturn(postId)
```

```
        val mockUserSnapshot = mock(DataSnapshot::class.java)
```

```
        `when`(mockUserSnapshot.key).thenReturn(userId)
```

```
        `when`(mockUserSnapshot.child("posts")).thenReturn(mockPostReference)
```

```
        val mockPostSnapshot = mock(DataSnapshot::class.java)
```

```
        `when`(mockPostSnapshot.key).thenReturn(postId)
```

```
        `when`(mockPostSnapshot.child("Post  
Status")).thenReturn(mock(DataSnapshot::class.java))
```

```
        `when`(mockPostSnapshot.child("Post  
Status")).getValue(String::class.java).thenReturn("approved")
```

```
        `when`(mockPostSnapshot.child("Description")).thenReturn(mock(DataSnapshot::class.java))
```

```
        `when`(mockPostSnapshot.child("Description")).getValue(String::class.java).thenReturn(description)
```

```

        `when`(mockPostSnapshot.child("Lat")).thenReturn(mock(DataSnapshot::class.java))

        `when`(mockPostSnapshot.child("Lat").getValue(Double::class.java)).thenReturn(lat)

        `when`(mockPostSnapshot.child("Lng")).thenReturn(mock(DataSnapshot::class.java))

        `when`(mockPostSnapshot.child("Lng").getValue(Double::class.java)).thenReturn(lng)

        listener.onDataChange(mockUserSnapshot)
    }

    val latch = CountDownLatch(1)

    latch.await(500, TimeUnit.MILLISECONDS)

    firebaseDatabaseLocationManager.retrieveLocation { postList ->

        assertEquals(1, postList.size) // Assuming only one post is retrieved

        val mapPost = postList[0]

        assertEquals(postId, mapPost?.postId)

        assertEquals(userId, mapPost?.userId)

        assertEquals(description, mapPost?.description)

        assertEquals(lat, mapPost?.location?.latitude)

        assertEquals(lng, mapPost?.location?.longitude)

    }

}

```

```

class LoginActivityTest {

    @Mock

    private lateinit var mockFirebaseAuth: FirebaseAuth

    @Mock

    private lateinit var mockFirebaseUser: FirebaseUser

    private lateinit var loginActivity: LoginActivity


    @Before

    fun setUp() {

        MockitoAnnotations.initMocks(this)

        loginActivity = LoginActivity()

        loginActivity.viewModel =
LoginViewModel(FirebaseAuthenticationManger(mockFirebaseAuth))

    }


    @Test

    fun testLogin_Success() {

        val email = "test@example.com"

        val password = "password"

        `when`(mockFirebaseAuth.signInWithEmailAndPassword(email,
password)).thenReturn(

            createMockAuthResult(true)

        )

        loginActivity.viewModel.email.value = email

        loginActivity.viewModel.password.value = password

        loginActivity.viewModel.onLoginClicked()

        assertEquals(LoginStatus.SUCCESS, loginActivity.viewModel.loginStatus.value)

```

```

    }

    @Test
    fun testLogin_Failure() {
        val email = "test@example.com"
        val password = "password"

        `when`(mockFirebaseAuth.signInWithEmailAndPassword(email,
password)).thenReturn(
            createMockAuthResult(false)
        )

        loginActivity.viewModel.email.value = email
        loginActivity.viewModel.password.value = password
        loginActivity.viewModel.onLoginClicked()
        assertEquals(LoginStatus.FAILURE, loginActivity.viewModel.loginStatus.value)
    }

    private fun createMockAuthResult(isSuccessful: Boolean): Task
    {
        val mockTask = Mockito.mock(Task::class.java)

        `when`(mockTask.isSuccessful).thenReturn(isSuccessful)

        return mockTask
    }
}

```

## **CHAPTER 7 : CONCLUSION AND RESULTS**

### **Summary of accomplished project**

The project successfully developed an Android Kotlin application aimed at assisting users in finding houses for rent and sale in Jordan. The application features functionalities such as user authentication, real-time map-based house listings, chat functionality, and user-generated posts. Additionally, an administrative module allows for post management and user warnings.

Throughout the project implementation, several challenges were encountered, including:

**Integration Complexity:** Integrating various functionalities like Firebase authentication, real-time database management, and Google Maps API posed challenges due to their complexity and the need for seamless interaction.

**Security Concerns:** Ensuring the security of user data and preventing unauthorized access to sensitive information was a critical concern throughout the development process.

Despite these challenges, the project has successfully addressed the initial objectives by providing a user-friendly and feature-rich application for finding and listing empty houses. The proposed solution offers several benefits to users, including:



Convenience: Users can easily search for empty houses based on location, view detailed information and images, and communicate with property owners or agents via chat.

Efficiency: The real-time map-based interface provides an efficient way for users to explore available properties and make informed decisions.

Community Engagement: The inclusion of user-generated posts fosters community engagement, allowing users to share information about available properties and interact with each other.

## **Future Work**

### **Enhanced User Experience (UX):**

Conduct user testing and gather feedback to identify areas for improvement in usability and user interface design.

Implement user-friendly features such as intuitive navigation, clear instructions, and responsive design for various screen sizes.

### **Performance Optimization:**

Improve app performance by optimizing code, reducing loading times, and minimizing resource usage.

**Localization and Globalization:**

Support multiple languages and currencies to cater to a diverse user base.

Customize the app experience based on the user's location, including localized property listings and relevant local information.