"GRIHA GULLY"

A MAJOR PROJECT REPORT

Submitted in partial fulfillment of the
Requirement for the award of Degree of

BACHELOR OF TECHNOLOGY

In

COMPUTER SCIENCE & ENGINEERING

Submitted by

JOSHI SAYALI SHRIPAD (20JJ1A0521)

MANIKANTA PRAMOD KUMAR REDDY (20JJ1A0530)

KOLUGOORI BHAVANA REDDY (20JJ1A0524)

CHUKKA HASINI (20JJ1A0512)

Under the Esteemed Guidance of

DR. S. VISWANADHA RAJU

Senior Professor, Department of CSE.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

(Accredited by NBA: UG(CSE))

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY

HYDERABAD UNIVERSITY COLLEGE OF ENGINEERING JAGTIAL

(Accredited By NAAC with A+ Grade)

Nachupally (Kondagattu), Jagtial Dist. -505501. T.S

2023-2024

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY

HYDERABAD UNIVERSITY COLLEGE OF ENGINEERING JAGTIAL

(Accredited By NAAC with A+ Grade)

NACHUPALLY (KONDAGATTU), JAGTIAL DIST, TELANGANA STATE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

(Accredited by NBA: UG(CSE))

CERTIFICATE

This is to certify that the major project entitled "**GRIHA GULLY**" is a bonafide work carried out by **JOSHI SAYALI SHRIPAD (20JJ1A0521), MANIKANTA PRAMOD KUMAR REDDY (20JJ1A0530), KOLUGOORI BHAVANA REDDY (20JJ1A0524), CHUKKA HASINI (20JJ1A0512)** in the partial fulfillment of the requirements for the award of the degree of **BACHELOR OF TECHNOLOGY** in **COMPUTER SCIENCE AND ENGINEERING** by the Jawaharlal Nehru Technological University Hyderabad during the academic year 2023-2024.

The results embodied in this report have not been submitted to any other University or Institution for the award of any degree.

- - - - - - - - - - - - - - - -	- -
Project Guide	**Head of the Department**
Dr. S. VISWANADHA RAJU	**Dr. B. SATEESH KUMAR**
Senior Professor, Dept of CSE	**Professor & Head, Dept of CSE**

- - - - - - - - - - - - - - -

External Examiner

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY
HYDERABAD UNIVERSITY COLLEGE OF ENGINEERING JAGTIAL

(Accredited By NAAC with A+ Grade)

NACHUPALLY (KONDAGATTU), JAGTIAL DIST, TELANGANA STATE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

(Accredited by NBA: UG(CSE))

DECLARATION

We hereby declare that the project work entitled "**GRIHA GULLY**" submitted in partial fulfillment of the requirements for the award of the degree **BACHELOR OF TECHNOLOGY** in the **COMPUTER SCIENCE AND ENGINEERING**, which was carried out under the supervision of **Dr. S. VISWANADHA RAJU** , Senior Professor, Department of CSE, JNTUH University College of Engineering, Jagtial.

Also, we declare that the matter embedded in this thesis has not been submitted by us in full or partial thereof for the award of any degree/diploma of any other institution or University previously.

JOSHI SAYALI SHRIPAD	**(20JJ1A0521)**
MANIKANTA PRAMOD KUMAR REDDY	**(20JJ1A0530)**
KOLUGOORI BHAVANA REDDY	**(20JJ1A0524)**
CHUKKA HASINI	**(20JJ1A0512)**

ACKNOWLEDGEMENT

The satisfaction that accompanies the successful completion of the task would be put incomplete without the mention of the people who made it possible, whose guidance and encouragement crown all the efforts with success.

We wish to express our deep sense of gratitude to **Dr. S. VISWANADHA RAJU** Senior Professor, department of CSE, and our Project Guide for his able, vigorous guidance and useful suggestions, which helped us in completing the project work in time.

We are extremely grateful to our HOD- **Dr. B. SATEESH KUMAR**, Professor of CSE& Head for his immense support and cooperation that contributed a lot in the completion of the task.

We show gratitude to our beloved Principal **Dr. V. KAMAKSHI PRASAD** , Professor of CSE and vice principal **Dr. T. VENUGOPAL** Sir, for providing necessary infrastructure and resources for the accomplishment of our project report at JNTUHCEJ.

We show our immense gratitude towards our college library for providing us with necessary references required for completing this project.

We also thank all the staff members of Computer Science and Engineering department and H&S department,JNTUH UCEJ for their valuable support and generous advice.

Finally, thanks to all my friends and family members for their continuous support and enthusiastic help.

JOSHI SAYALI SHRIPAD (20JJ1A0521)

MANIKANTA PRAMOD KUMAR REDDY (20JJ1A0530)

KOLUGOORI BHAVANA REDDY (20JJ1A0524)

CHUKKA HASINI (20JJ1A0512)

ABSTRACT

Griha Gully redefines the real estate experience by leveraging machine learning to personalize property suggestions based on user behavior. Unlike traditional platforms, Griha Gully uses smart technology like machine learning to understand individual preferences, suggesting homes perfectly tailored to users.

Incorporating ML algorithms, Griha Gully transforms real estate efficiency by personalizing recommendations through user behavior analysis. These algorithms refine suggestions swiftly, optimizing user time and satisfaction. Continuous learning ensures precise matches, showcasing Griha Gully's innovation and commitment to superior user experiences. ML's scalability handles growing data seamlessly, ensuring recommendation quality. Together, these features enhance efficiency, accuracy, and user-centricity, bolstering Griha Gully's competitive edge in real estate.

TABLE OF CONTENTS

ABSTRACT
1.INTRODUCTION 1
1.1 PROJECT OVERVIEW 1
1.1.1 PROBLEM STATEMENT 1
1.2 OBJECTIVES 3
1.3 EXISTING SYSTEM 3
1.3.1 Drawbacks of Existing System: 3
1.4 PROPOSED SYSTEM 4
2. SOFTWARE REQUIREMENT ANALYSIS 5
2.1 REQUIREMENT ANALYSIS 5
2.2 REQUIREMENT SPECIFICATIONS 6
2.3 FEASIBILITY STUDY 9
2.3.1 Economic Feasibility 9
2.3.2 Technical Feasibility 9
2.3.3 Operational Feasibility 10
2.3.4 Market Feasibility 10
3. SOFTWARE DESIGN 10
3.1 ARCHITECTURE 11
Fig 3.1: Architecture 11
3.2 MODULE DESCRIPTION 11
3.2.1 Presentation Layer 11
3.2.2 Business Layer 12
3.2.3 Data Layer 13
3.2.4 API - Application Programming Interface 14
3.3 UML DIAGRAMS 14
3.3.1 Data Flow Diagram 15
Fig 3.3.1: Data flow diagram of the project 16
3.3.2 Use Case Diagram 18
Fig 3.3.3: Use Case Diagram 19
3.3.3 Class Diagram 19
4. TECHNOLOGY STACK 21
4.1 FLUTTER 21
4.2 FIRE BASE 21

5.1 SOURCE CODE: 22
6. TESTING 91
6.1 TESTING STRATEGIES 91
6.2 TESTING OBJECTIVES 91
6.3 LEVELS OF TESTING (test cases) 91
6.3.1 Unit Testing 91
6.3.2 Integration Testing: 96
6.3.3 System Testing 97
6.3.4 White Box Testing 97
6.3.5 Black Box Testing 98
7. OUTPUT SCREENS 100
7.1 SPLASH AND INTRODUCTION SCREEN 100
7.2 SIGN IN SCREEN 100
7.3 REGISTER PAGE 101
7.4 AUTHENTICATION 102
7.5 SEARCH 102
7.6 SETTINGS 103
7.7 PROFILE 103
7.7 HOME 104
8.CONCLUSION 105
9. FUTURE ENHANCEMENTS 106
10. REFERENCES 107
APPENDICES: 108

LIST OF FIGURES

Fig 3.1: Architecture 12
Fig 3.3.1: Data flow diagram of the project 17
Fig 3.3.2 Login / Register 18
Fig 3.3.3: Use Case Diagram 19
Fig 3.3.4: Class Diagram 20
Fig 7.1 Splash screen 100
Fig 7.2 Introduction screen 100
Fig 7.4 Register Screen 101
Fig 7.4 Authentication screens 101
Fig 7.5 Search screen 102
Fig 7.6 Settings screen 102
Fig 7.7 Profile screen 103
Fig 7.8 Home screen 103
Fig 7.9 Favorite screen 104

1.INTRODUCTION

1.1 PROJECT OVERVIEW

Griha Gully , the real estate app with innovative use of machine learning (ML) technology. By harnessing ML algorithms, the platform can analyze user behavior to offer personalized property suggestions, a significant departure from the one-size-fits-all approach of traditional real estate platforms.

The emphasis on understanding individual preferences and tailoring recommendations accordingly is a game-changer. It not only saves users time by presenting them with relevant options but also enhances satisfaction by ensuring that the properties align closely with their needs and preferences.

The mention of continuous learning suggests that Griha Gully is committed to refining its algorithms over time, which is crucial for maintaining the accuracy and relevance of its recommendations. Additionally, the scalability of ML allows the platform to handle increasing amounts of data without compromising recommendation quality, which is essential for accommodating growth and ensuring a seamless user experience.

Overall, Griha Gully's focus on efficiency, accuracy, and user-centricity positions it as a leader in the real estate market, offering a competitive edge through its innovative use of ML technology.

1.1.1 PROBLEM STATEMENT

The gap between user preferences and available suggestions in traditional real estate platforms significantly impacts the overall user experience. Users encounter frustration due to the lack of personalized recommendations, leading to prolonged search times, dissatisfaction, and ultimately, a disconnect between what they seek and what the platform provides. This makes it hard for users to find what they're looking for, which makes them frustrated and unhappy. It also means they might stop using the website altogether.

1.1.2 MOTIVATION

Recognizing the existing challenges and limitations faced by users on traditional real estate platforms, there is a driving desire to bridge the gap between user preferences and available property suggestions. By leveraging advanced technology and innovative features, such as personalized recommendations, comprehensive property listings, and seamless communication channels,

The motivation behind building an app like Griha Gully stems from a deep-rooted commitment to revolutionize the real estate industry and enhance the property search experience for users.Griha Gully seeks to empower users with the tools and information they need to find their dream property efficiently and effectively. Moreover, the aim is to foster transparency, trust, and satisfaction among users by prioritizing their needs and delivering a user-centric platform that aligns with individual tastes and preferences.

Ultimately, the motivation behind Griha Gully is to redefine the real estate search experience, making it more personalized, transparent, and enjoyable for users while corntributing positively to the overall growth and evolution of the real estate industry.

1.2 OBJECTIVES

Griha Gully is an innovative real estate platform dedicated to transforming the property search experience by providing

- Enhance User Satisfaction: Prioritize user satisfaction by presenting highly relevant and accurately matched property options aligned with specific needs.
- Continuous Learning and Innovation: Demonstrate a commitment to innovation and superior user experiences through continuous learning, ensuring precise property matches over time.
- Guarantee High-Quality Recommendations: Ensure the high quality of property recommendations using the scalability of machine learning, effortlessly managing the expansion of data.

- Optimize Property Search Journey: Utilize technology to optimize the property search journey, reducing search time and presenting efficient, tailored recommendations.
- User-Centric Platform: Focus on providing a user-centric platform that aligns with individual tastes and needs, thereby enhancing the overall search experience.

1.3 EXISTING SYSTEM

Indeed, existing systems similar to Griha Gully include established real estate platforms such as MagicBricks and 99 acres.com, which offer property listings but may lack personalized recommendations tailored to individual user preferences. Here's a breakdown of some of these platforms:

Real estate platforms:

1. **Magicbricks**: It is one of the leading property portals in India, offering a wide range of residential and commercial property listings, along with various tools and resources for property buyers, sellers, and renters.
2. **99acres**: Similar to Magicbricks, 99acres is another prominent real estate platform providing comprehensive property listings across India. It offers a range of services including property search, listings, and market analysis.
3. **NoBroker**: NoBroker is known for its unique model that connects property owners and tenants directly, eliminating the need for brokers or intermediaries. It provides rental and resale property listings without any brokerage fees, making it a popular choice for both tenants and property owners.
4. **Square Yards**: A global real estate marketplace offering property listings, real estate advisory services, and financial assistance for buying, selling, and renting properties.

These websites and applications play a crucial role in bridging the gap between job seekers and employers globally, as well as aiding consumers in discovering and evaluating local businesses for various services.

1.3.1 Drawbacks of Existing System:

1. Limited Personalization: Many platforms lack personalized recommendations, leading to users being overwhelmed with irrelevant listings and struggling to find properties that meet their specific needs.

2. Inefficient Search Filters: The search filters on certain platforms may be limited or inefficient, making it challenging for users to narrow down their search results to find properties that match their specific criteria.
3. Poor User Experience: Cluttered interfaces, slow loading times, and complex navigation can result in a poor user experience, leading to frustration and dissatisfaction among users.
4. Lack of Integration: Integration with other services or platforms, such as financing or legal services, may be limited on some real estate platforms, making it more challenging for users to complete transactions seamlessly.

Addressing these drawbacks and improving the overall user experience is essential for real estate platforms to remain competitive and effectively serve the needs of their users.

1.4 PROPOSED SYSTEM

Griha Gully is an innovative real estate platform dedicated to transforming the property search experience.It changes how you find homes by suggesting personalized options using smart technology. Using this, it finds homes you'll love, making your search easier and more satisfying. Griha Gully aims to make finding your dream home easier and more personal than ever before

Here's an elaboration on the key features and steps of the system:

1. **User Registration and Profile Creation**:
   - Users create accounts on the platform.
   - They provide essential information, such as personal details, and preferences.
2. **Detailed Property Listings:**
   - Each property listing includes comprehensive details such as property descriptions, high-quality images, contact details of the seller or agent.
3. **Personalized Property Recommendations:**
   -The app provides tailored property recommendations based on user preferences, search history, and location.
4. **User Profile and History:**
   -Users can create profiles to save their preferences, view their search history, and access their favorite properties for easy reference.

Overall, the Griha Gully app aims to provide users with a seamless and personalized real estate experience, empowering them to find their dream property efficiently and effectively.

Griha Gully addresses the drawbacks of existing real estate platforms through several key strategies:

- Personalized Recommendations: Griha Gully's advanced algorithms provide highly tailored property recommendations, reducing the gap between user preferences and available suggestions, thereby enhancing user satisfaction.
- Comprehensive Property Listings: Griha Gully ensures that property listings are detailed and up-to-date, offering comprehensive information to users, thus minimizing the risk of incomplete or outdated property data.
- User-Centric Approach: Griha Gully focuses on providing a user-centric platform aligned with individual tastes and needs, enhancing the overall search experience and reducing user frustration.
- Seamless Communication: Griha Gully facilitates seamless communication between users and property owners or agents, ensuring that user inquiries are promptly addressed and facilitating a smoother property search process.
- Innovative Technology: Griha Gully leverages innovative technology, including machine learning and virtual property tours, to optimize the property search journey, reduce search time, and present efficient, tailored recommendations to users.

By implementing these strategies, Griha Gully aims to overcome the drawbacks of existing real estate platforms and provide users with a more satisfying and efficient property search experience.

2. SOFTWARE REQUIREMENT ANALYSIS

2.1 REQUIREMENT ANALYSIS

Project perspective

The ultimate perspective of any technology or innovation is to make the human life much easier to sustain either by providing a basic need or adding on a luxury to basic life. This project, “Griha Gully” can be used for both types.

Project Features

The features can be considered as the in-dependency, Ease-to-use, Interactive & Responsive.

a. In-dependency.

b.Ease-to-use.

c.Interactive & responsive.

2.2 REQUIREMENT SPECIFICATIONS

External Interface Requirements

External Interface requirements are the necessities that should be satisfied for successful development and working the application. Such interface requirements are basically UI requirements, Hardware and software requirements

User Interfaces:

1 Laptop/Desktop with Graphic card.

2 Ability to login and access their account.

Hardware Interfaces:

- Laptop, Desktop : Minimum i5 configuration
- RAM Capacity : 8 GB
- Hard Disk : 60 GB

Software Interfaces:

- Languages use : Dart programming language
- Technologies used : Flutter,Firebase,Android Studio
- OS : Windows/Linux
- Platform : GitHub

Functional Requirements

System requirements which specify functions that a system must be capable of performing in correct way are known as functional requirements. Following are the functional requirements of the project.

2 types of users will be using this application.

Owner:

- The registered owner should be able to add properties through their account and those properties should be visible to the users

The property details should include:

➔ Property Title
➔ Location
➔ Area covered
➔ Price of the property
➔ Property features
  1. No. of rooms
  2. Storeyed house or not
  3. Architecture etc.

- Owner can add any other parameters suitable for describing property specifications.

User:

- User should be able to view all the information available in the application.
- User should be able to search for the property's different parameters, different parameters of the property.
- User should be provided with the facility to compare different properties.

Other Requirements are :

1) **User Authentication:** Users should be able to create accounts, log in, and reset passwords securely.
2) **User Roles:** Different user roles (e.g., admin, regular user) should have specific permissions and access levels.
3) **User Profile Management:** Users should be able to update their profiles, including personal information and preferences.
4) **Content Creation and Management:** Authorized users should create, edit, and delete content.

5) **Search and Filtering:** Users should search for content and filter results based on various criteria.
6) **Data CRUD Operations:** Full CRUD (Create, Read, Update, Delete) operations for managing data entities (e.g.,properties).
7) **Notification System:** Send notifications to users (e.g., email, in-app) for various events .
8) **File Uploads:** Allow users to upload files (e.g., images, documents) and manage them.
9) **Security Features:** Implement security measures, including authorization checks, data validation, and protection against common vulnerabilities.

Non-Functional Requirements

Non-Functional Requirements are those which describe the threshold capability of the project for functioning without any breaks or errors. They are usually performance and safety requirements.

1. **Performance:** Ensure the system responds quickly and efficiently under various loads, with acceptable page load times.
2. **Scalability:** The application should handle increased user and data loads by scaling horizontally or vertically.
3. **Reliability:** The system should be available with minimal downtime and recover gracefully from failures.
4. **Security:** Protect user data, implement encryption, and follow best practices for authentication and authorization.
5. **Usability:** The user interface should be intuitive and user-friendly, with accessibility features for all users.
6. **Compatibility:** The application should work on different browsers and devices, with responsive design for mobile devices.
7. **Data Integrity:** Ensure data consistency and accuracy throughout the application.
8. **Backup and Recovery:** Implement regular data backups and disaster recovery plans.

9. **Compliance:** Comply with relevant laws and regulations (e.g., GDPR for data privacy).
10. **Monitoring and Logging:** Set up monitoring tools to track system health and log events for debugging and auditing.
11. **Availability:** Aim for high availability (e.g., 99.9% uptime) based on project requirements.
12. **Resource Efficiency:** Optimize resource usage, including memory and CPU, to reduce costs.
13. **Documentation:** Maintain comprehensive documentation for code, APIs, and system architecture.
14. **Response Times:** Define acceptable response times for various actions within the application.
15. **Data Storage and Retrieval:** Specify data storage requirements, such as database choices and data retrieval times.
16. **Error Handling:** Implement clear and informative error messages for users and detailed error logs for developers.

2.3 FEASIBILITY STUDY

A feasibility study for the provision system for region based jobs and facilities provision system project involves assessing the website, audience response, cost-effectiveness, and practicality of implementing the proposed system. Here are the key components of the feasibility study:

- Economic Feasibility
- Technical Feasibility
- Operational Feasibility
- Market Feasibility

2.3.1 Economic Feasibility

- Estimating the initial investment required to develop the app, including development costs, marketing expenses, and any other overheads.

- Projecting the revenue potential of the app based on market size, pricing strategies, and monetization models (e.g., subscription fees, in-app purchases, advertising).
- Conducting a financial analysis, including a break-even analysis, to determine the profitability of the app and the return on investment (ROI).

2.3.2 Technical Feasibility

- Evaluating the technical requirements for developing and maintaining the website.
  - Technical Requirements:
    - Frontend (Flutter):
      - Ensure compatibility with modern browsers.
      - Responsive design for various devices.
    - Backend (Firebase)
- Evaluating the availability of technology for the app development and expertise required.
  - Technology Availability and Expertise:
    - Assess the availability of Flutter and Firebase for the chosen hosting platform.
    - Evaluate the expertise of the development team in Flutter,Firebase.
- Software, hardware, and development tools needed for the development and maintenance of the website.
    - Assess server requirements based on expected traffic and database size.
    - Ensure that developers have the necessary code editors (e.g., Visual Studio Code).
- Consideration of scalability and future updates of the website.

2.3.3 Operational Feasibility

- Evaluating the operational aspects of running the app, including customer support, maintenance, and updates.
- Assessing the scalability of the app to handle increased user demand over time.
- Identifying any operational challenges and determine how they can be addressed.

2.3.4 Market Feasibility

- Identifying the target market for the app (e.g., property buyers, renters, real estate agents, property investors).
- Analysing the growth trends of the real estate market.
- Identifying existing competitors in the market and analyze their strengths, weaknesses, opportunities, and threats (SWOT analysis).
- Determining the unique selling points of the app compared to existing competitors.

3. SOFTWARE DESIGN

3.1 ARCHITECTURE

Architecture describes the overall functionality of the project with all the modules specified in it. It gives a clear picture of the internal process of the projects. The architecture along with modules can be displayed as the following.

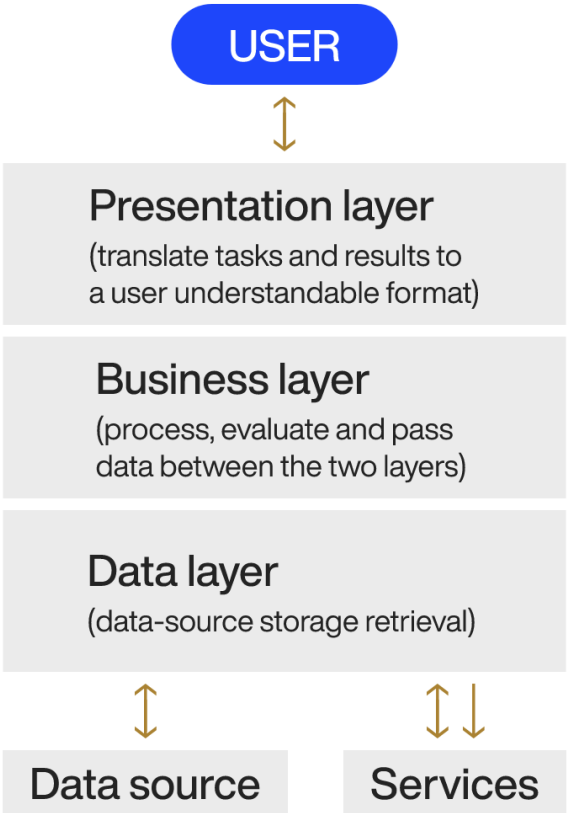


Fig 3.1: Architecture

3.2 MODULE DESCRIPTION

According to the architecture above,the modules used in this project are : front end development,backend development and database integration with the front end and backends

Modules of the project can be defined as:

3.2.1 Presentation Layer

Presentation layer in mobile applications involves creating the user interface (UI) and user experience (UX) elements that users interact with directly on their mobile devices. It focuses on designing and implementing visually appealing, intuitive, and responsive interfaces that facilitate smooth navigation and interaction within the app.

Key aspects of presentation layer in mobile applications include:

UI Design: This involves creating layouts, selecting appropriate fonts and colors, and designing interactive components such as buttons, forms, and navigation menus.

Responsive Design: Mobile devices come in various screen sizes and resolutions, so the app's UI adapts and scales appropriately across different devices and orientations. This involves implementing responsive design principles and using techniques like media queries and flexible layouts.

Performance Optimization: App's performance is optimized by minimizing load times, reducing memory usage, and optimizing rendering performance. This includes techniques like lazy loading, image optimization, and code splitting to ensure fast and smooth user experiences.

Accessibility: This ensures that the app can be used by a diverse range of users, regardless of their abilities.

3.2.2 Business Layer

Business layer in mobile app development refers to the creation and management of the server-side components that power mobile applications. While the presentation layer handles the user interface and interactions, the backend is responsible for the behind-the-scenes operations, data processing, and server communication necessary for the app to function.

Key aspects of business layer in mobile applications include:

Data Management: One of the primary functions of the backend is to manage the app's data. This includes storing and retrieving user information, application data, multimedia content, and any other relevant data required for the app's functionality.

Backend developers design databases, write queries, and create APIs to facilitate efficient data exchange between the mobile app and the server.

Business Logic: Backend developers implement the business logic of the application on the server-side. This involves coding algorithms, workflows, and rules that govern how data is processed, validated, and manipulated.

User Authentication and Authorization: Security is paramount in mobile app development. Backend developers implement authentication mechanisms to verify users' identities and authorization systems to control access to various app features and resources. This typically involves integrating authentication protocols like OAuth or JWT (JSON Web Tokens) and implementing role-based access control (RBAC) or other authorization strategies.

API Development: Mobile apps often need to communicate with external services or integrate with other systems. Backend developers create APIs (Application Programming Interfaces) that expose endpoints for the mobile app to interact with. These APIs define the protocols and data formats for exchanging information between the client (mobile app) and the server.

Performance Optimization and Scalability: Backend developers are responsible for optimizing the performance of the server-side code to ensure fast response times and efficient resource utilization. They also design the backend architecture to be scalable, capable of handling increasing numbers of users and data without sacrificing performance or reliability.

3.2.3 Data Layer

Data layer is a crucial aspect of software development, enabling applications to store, retrieve, and manage data efficiently. It’s nothing but database integration. It involves connecting the backend of an application with a database system, allowing applications to interact with and manipulate data.

Database integration includes integration of database with both the frontend and backend. Integrating the database with both frontend and backend systems facilitates a

smooth exchange of data, ensuring that user inputs are processed, stored securely, and made accessible as needed throughout the website.

In frontend database integration, user information retrieved from the database, like profile details during login, is presented to users. User-entered data on the frontend, via forms or interfaces, is transferred to the backend for processing and storage in the database. The backend manages various data operations such as registration and service requests, applying specific rules before database interaction to ensure accuracy and security. For instance, when a user submits a service request, the backend processes the data, interacts with the database to securely store the information, and sends confirmation to the frontend upon successful storage.

3.2.4 API - Application Programming Interface

An API (Application Programming Interface) refers to a set of rules, protocols, and tools that allow different software applications to communicate with each other. APIs are often used to enable communication between the app's frontend (the user interface) and backend (server-side components).

Mobile apps use APIs to perform various tasks, such as retrieving data from a server, sending user input to a server for processing, authenticating users, and integrating with third-party services like social media platforms or payment gateways. APIs define the methods and data formats that developers can use to interact with the backend systems, allowing for seamless integration and functionality within the mobile app.

3.3 UML DIAGRAMS

UML diagrams, crucial in software engineering, visually represent system components, behaviors, and relationships. They serve as blueprints, aiding system analysis, design, and communication among teams and stakeholders. These diagrams simplify complexities, acting as collaborative tools for discussing and resolving issues. They document software systems, aiding understanding, maintenance, and updates. Class, use case, sequence, activity, component, and deployment diagrams are common types. UML's value lies in providing standardized, clear representations, facilitating efficient communication, and ensuring a shared understanding of system architecture

and behavior, ultimately enhancing planning, decision-making, and system development processes.

3.3.1 Data Flow Diagram

A data flow diagram (DFD) maps out the flow of information for any process or system. It uses defined symbols like rectangles, circles and arrows, plus short text labels, to show data inputs, outputs, storage points and the routes between each destination. Data flowcharts can range from simple, even hand-drawn process overviews, to in-depth, multi-level DFDs that dig progressively deeper into how the data is handled.

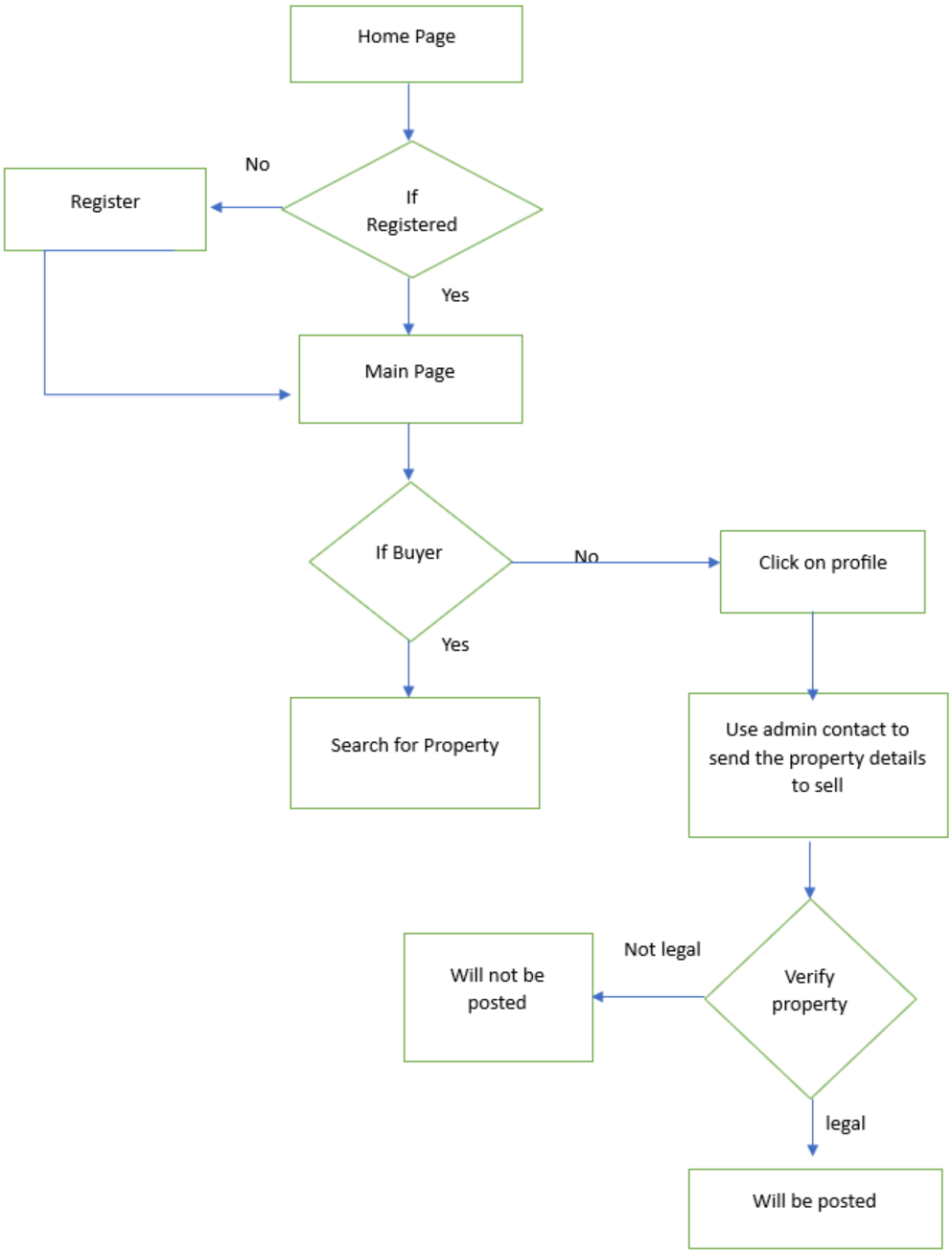


Fig 3.3.1: Data flow diagram of the project

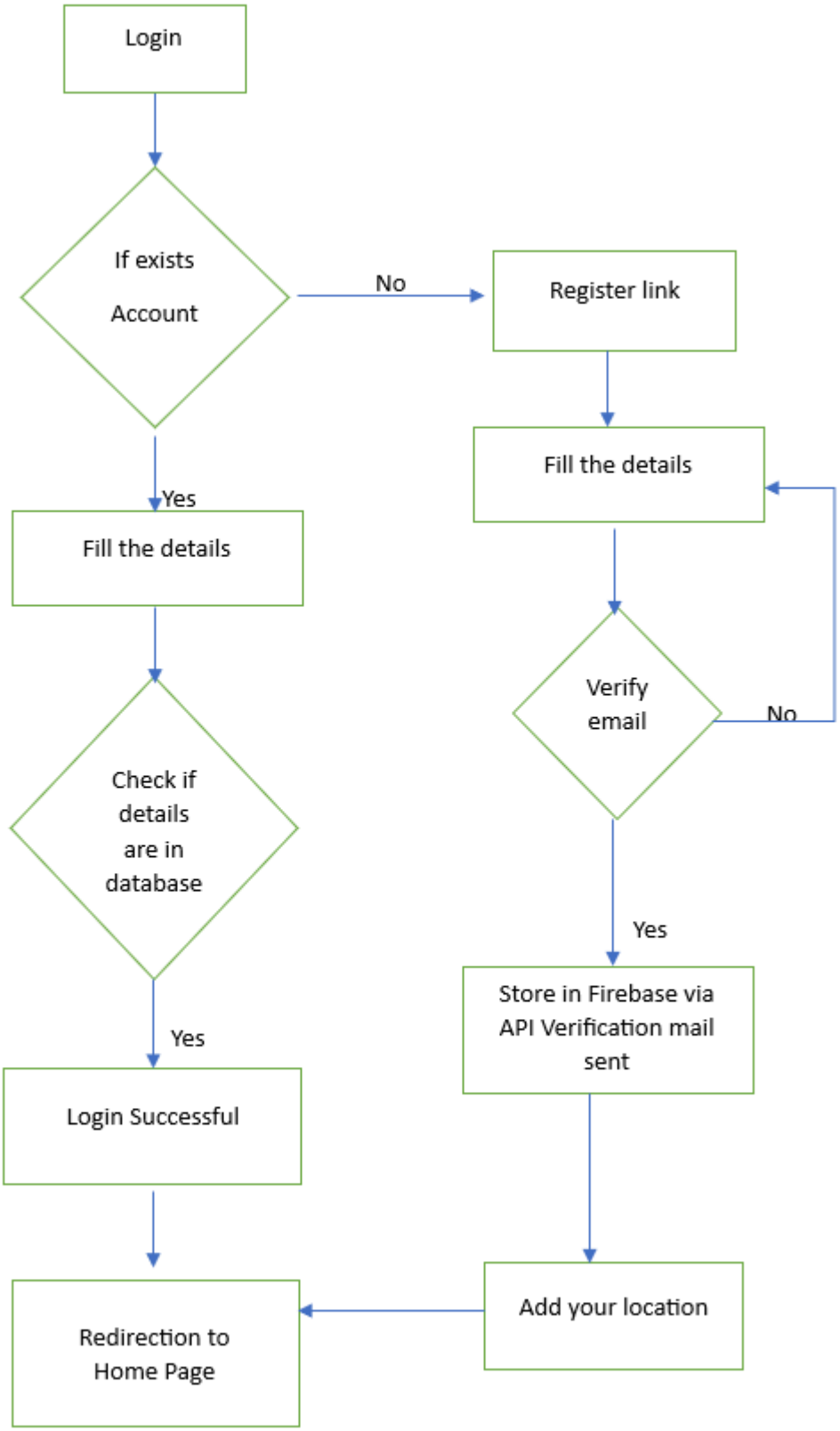


Fig 3.3.2 Login / Register

3.3.2 Use Case Diagram

Use case diagrams specify the events of a system and their flows. But use case diagram never describes how they are implemented. Use case diagram can be imagined as a black box where only the input, output, and the function of the black box is known.

These diagrams are used at a very high level of design. This high-level design is refined again and again to get a complete and practical picture of the system. A well-structured use case also describes the pre-condition, post condition, and exceptions. These extra elements are used to make test cases when performing the testing.

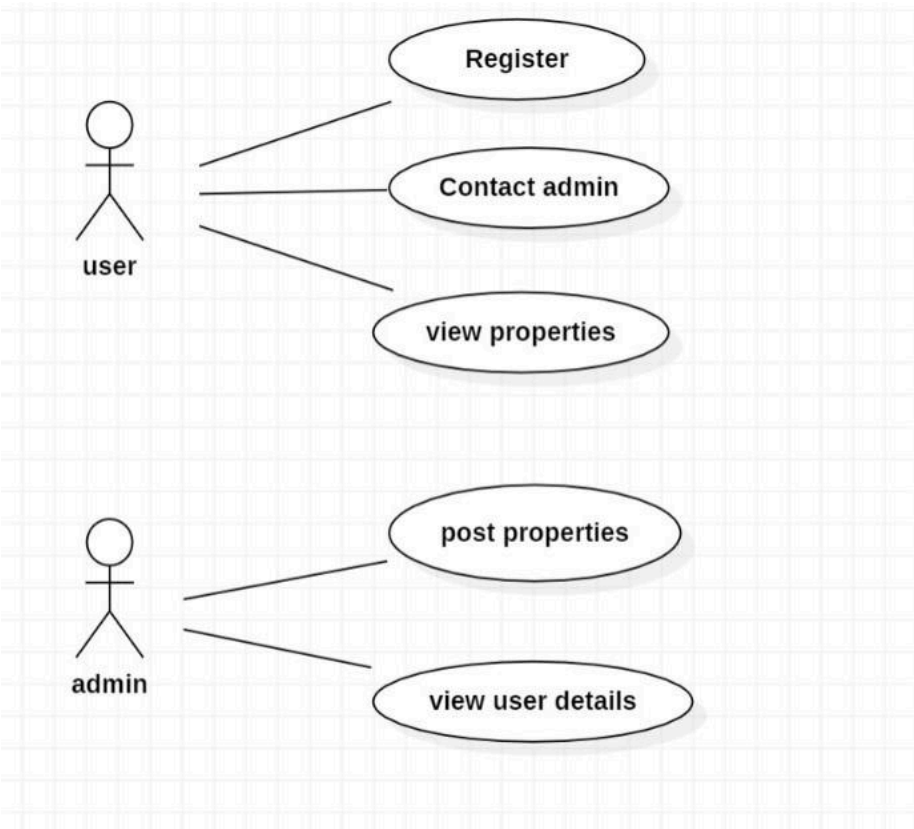


Fig 3.3.3: Use Case Diagram

3.3.3 Class Diagram

Class diagram is a static diagram and it is used to model the static view of a system. The static view describes the vocabulary of the system.Class diagram is also considered as the foundation for component and deployment diagrams.

Class diagrams are not only used to visualize the static view of the system but they are also used to construct the executable code for forward and reverse engineering of any system.Generally,UML diagrams are not directly mapped with any object - oriented programming languages but the class diagram is an exception.

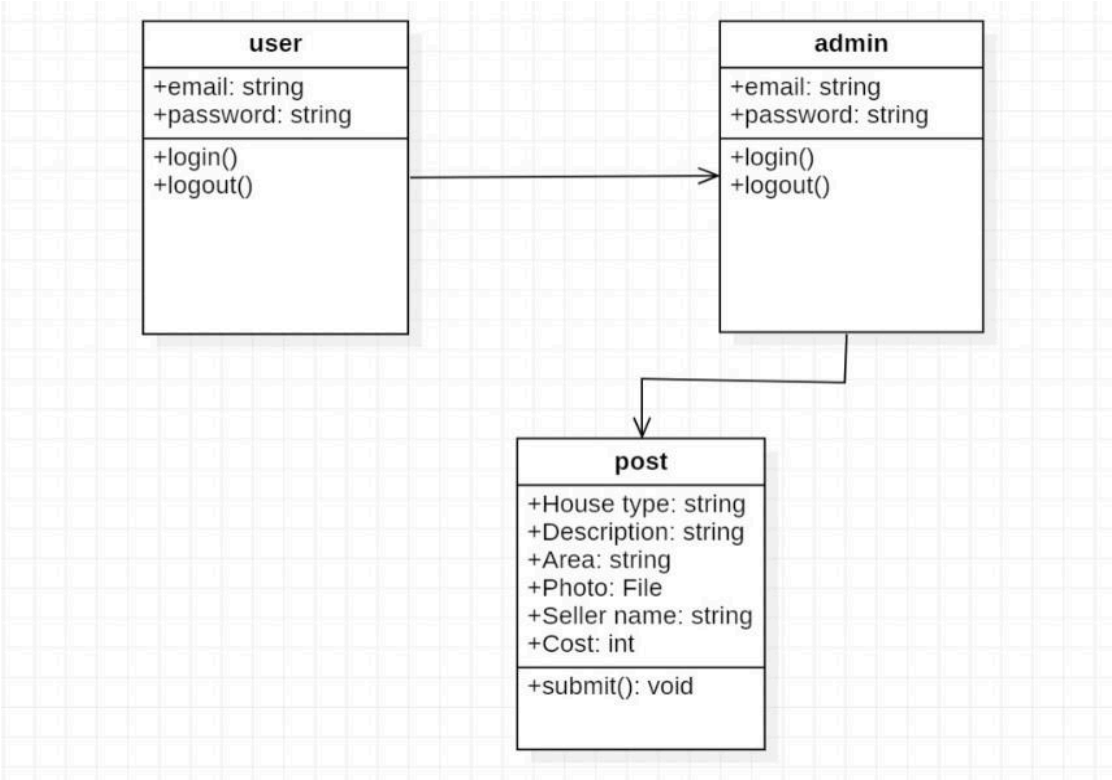


Fig 3.3.4: Class Diagram

4. TECHNOLOGY STACK

A technology stack, often referred to as a tech stack, is a collection of programming languages, software tools, frameworks, libraries, servers, databases, and other technologies used by developers to build and operate a web or software application. It's essentially the combination of technologies and tools employed to develop a particular project.

4.1 FLUTTER

Flutter is an open-source UI software development toolkit created by Google for building natively compiled applications for mobile, web, and desktop from a single codebase. It uses the Dart programming language and offers a rich set of features that make it a popular choice among developers for building cross-platform applications efficiently.

Key features of Flutter:

- Single Codebase
- Expressive UI
- Native Performance
- Hot Reload
- Access to Native Features
- Strong Community Support
- Built-in Testing Support

4.2 FIRE BASE

Firebase is a comprehensive platform developed by Google that provides a wide range of tools and services to help developers build, improve, and grow their apps. It offers features for app development, authentication, real-time database, storage, hosting, analytics, and more.

Key Features of Fire base:

- Real-time Database
- Authentication
- Cloud Firestore

- Cloud Storage
- Hosting
- Cloud Functions
- Performance Monitoring
- Analytics
- Remote Configuration
- Crashlytics

5. CODING/CODE TEMPLATES

5.1 SOURCE CODE:

1. **Importing libraries**

```
import 'package:flutter/material.dart';
import 'package:flutter/services.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
import 'package:flutter_screenutil/flutter_screenutil.dart';
import 'package:firebase_core/firebase_core.dart';
```

2. **Splash screen**

```
class SplashScreen extends StatefulWidget {
  const SplashScreen({super.key});
  @override
  State<SplashScreen> createState() => _SplashScreenState();
}
class _SplashScreenState extends State<SplashScreen> {
  @override
  void initState() {
    execute();
    super.initState();
  }
  void execute() async {
    await Future.delayed(const Duration(seconds: 4), _determinePosition);
  }
  dynamic routeToOnboard() {
    return Navigator.pushNamedAndRemoveUntil(
        context, RoutesName.setup, (route) => false);
  }
  Future<void> _determinePosition() async {
    final state = context.read<AppBloc>().state.status;
    Navigator.pushNamedAndRemoveUntil(
```

```
      context, AppRoute.onAppView(state), (route) => false);
  }
  @override
  Widget build(BuildContext context) {
    return const Scaffold(
      body: Stack(
        children: [
          BackgroundImage(),
          ContentsOfBody(),
        ], ), );
  }
}
class ContentsOfBody extends StatelessWidget {
  const ContentsOfBody({super.key});
  @override
  Widget build(BuildContext context) {
    return Align(
      alignment: Alignment.center,
      child: Column(
        children: [
          const Spacer(flex: 2),
          SvgPicture.asset(
            AssetPath.logo,
            semanticsLabel: 'App Logo',
            width: 250.w,),
          AppSpace.h10,
          Text(AppString.appname, style: AppTextStyle.wHeader),
          Text(AppString.slogan, style: AppTextStyle.wSubheader),
          const Spacer(flex: 1),
          Text(AppString.dev, style: AppTextStyle.wBody),
          Text(
            "v${AppInfo.version}",
            style: AppTextStyle.wBody.copyWith(
              fontWeight: FontWeight.bold,
            ), ), AppSpace.h50, ],),);}}
```

```
class BackgroundImage extends StatelessWidget {
  const BackgroundImage({super.key});
  @override
  Widget build(BuildContext context) {
    final height = MediaQuery.sizeOf(context).height;
    return Stack(
      children: [
        Image.asset(
          AssetPath.splash,
          height: height,
          fit: BoxFit.cover,
          semanticLabel: 'Background Image', ),
        Container(color: AppColor.fader.withOpacity(0.6)),
        Align(
          alignment: Alignment.bottomCenter,
          child: Container(
            height: height / 1.5,
            decoration: AppDecoration.fader, ), )], );
  }
}
```

3. **Home screen**

```
class HomePage extends StatelessWidget {
  const HomePage({super.key});
  @override
  Widget build(BuildContext context) {
    return OpacityTranslateAnimation(
      child: Stack(
        children: [
          _backgroundCircle(),
          const _HomePageContent(),
        ],
      ),
    );
  }

  Positioned _backgroundCircle() {
    return Positioned(
      left: -55.w,
      top: -35.w,
      child: CircleAvatar(
```

```
      radius: 150.sp,
      backgroundColor: AppColor.main.withOpacity(0.2),
    ),
  );
 }
}

class _HomePageContent extends StatelessWidget {
 const _HomePageContent();

 @override
 Widget build(BuildContext context) {
  return Positioned(
    top: kToolbarHeight + 10.h,
    left: 0,
    right: 0,
    bottom: 0,
    child: NestedScrollView(
      headerSliverBuilder: (context, _) {
        return [
          SliverPersistentHeader(
            delegate: AppBarSection(),
            pinned: true,
          ),
        ];
      },
      body: BlocBuilder<PropertyBloc, PropertyState>(
        builder: (context, state) {
          if (state is PropertyLoading) {
            return const Center(
              child: CircularProgressIndicator(),
            );
          }
          if (state is PropertyFailure) {
            return AppFailureWidget(msg: state.msg);
          }
          if (state is PropertySuccess) {
            return SingleChildScrollView(
              physics: const BouncingScrollPhysics(),
              child: Column(
                children: [
                  AppSpace.h10,
                  CategorySection(categories: state.data.categories),
                  PromotionSection(discounts: state.data.discounts),
                  FeaturedEstatesSection(features: state.data.features),
                  LocationsSection(locations: state.data.locations),
                  AgentsSection(agents: state.data.argents),
```

```
                NearbySection(nearby: state.data.nearby),
                AppSpace.h50,
              ].verticalGap(20.w),
            ),
          );
        }
        return const SizedBox();
      },
    ),
  ),
);
}
}
```

4. **Sign in screen**

```
class LoginPage extends StatelessWidget {
  const LoginPage({super.key});

  @override
  Widget build(BuildContext context) {
    final theme = Theme.of(context);
    return MultiBlocProvider(
      providers: [
        BlocProvider(create: (create) => EmailValidatorCubit()),
        BlocProvider(create: (create) => PasswordValidatorCubit()),
        BlocProvider(create: (create) => LoginCubit()),
      ],
      child: BlocConsumer<LoginCubit, LoginState>(
        listener: (context, state) {
          if (state.status.isNeedVerify) {
            Navigator.pushNamedAndRemoveUntil(
                context, RoutesName.email, (route) => false);
          }
          if (state.status.done) {
            Navigator.pushNamedAndRemoveUntil(
                context, RoutesName.home, (route) => false);
          }
        },
        builder: (context, state) {
          return Scaffold(
            body: Stack(
              children: [
                _backgroundCircle(),
                ScrollDetective(
                  list: SingleChildScrollView(
                    padding: AppPadding.h20.copyWith(top: 50.w),
```

```
child: Center(
  child: Column(
    crossAxisAlignment: CrossAxisAlignment.center,
    children: [
      AppSpace.h50,
      AppSpace.h10,
      _readyText(theme),
      AppSpace.h10,
      Align(
        alignment: Alignment.centerLeft,
        child: Text(
          AppString.slogan,
          style: AppTextStyle.wSubheader.copyWith(
            color: theme.mainText,
            fontSize: 12.sp,
          ),
        ),
      ),
      if (state.status.error)
        Container(
          height: 55.w,
          margin: AppPadding.h10
              .copyWith(bottom: 20.h, top: 20.h),
          padding: AppPadding.h20,
          decoration: BoxDecoration(
            color: AppColor.main,
            borderRadius: BorderRadius.circular(10.0),
          ),
          child: Center(
            child: Text(
              state.err,
              maxLines: 2,
              style: TextStyle(
                fontSize: 12.sp,
                color: AppColor.white,
                fontWeight: FontWeight.w400,
              ),
            ),
          ),
        )
      else
        AppSpace.h50,
      const _EmailTextFieldBox(),
      AppSpace.h10,
      const _PasswordTextFieldBox(),
      AppSpace.h20,
      Align(
```

```
                alignment: Alignment.centerLeft,
                child: ShrinkButton(
                  onPressed: () async {
                    //TODO:
                    if (state.email.isNotEmpty) {
                      AppDialogBox(
                            context: context,
                            title: 'Reset your password?',
                            descreption:
                                'are you sure you want to Reset your password? then you
click confirm and check your email.')
                          .confirmation
                          .then((value) => context
                            .read<LoginCubit>()
                            .restPassword());
                    } else {
                      AppDialogBox(
                            context: context,
                            title: 'Reset your password?',
                            descreption:
                                'please enter your email before resetting your password.')
                          .confirmation;
                    }
                  },
                  child: Padding(
                    padding: AppPadding.h10,
                    child: Text(
                      'Forgot password?',
                      style: TextStyle(
                        color: theme.mainText,
                        fontSize: 13.sp,
                        fontWeight: FontWeight.w500,
                      ),
                    ),
                  ),
                ),
              ),
              AppSpace.h20,
              const _LoginButton(),
              AppSpace.h20,
              const OrLine(),
              AppSpace.h20,
              const SocialMediaButtonsRow(),
              AppSpace.h20,
              const RegisterTextButton(),
            ],
          ),
```

```
            ),
          ),
          top: Container(
            decoration:
                AppDecoration.top(theme.scaffoldBackgroundColor),
            height: 20.w,
          ),
        ),
        Positioned(
          left: 20.w,
          top: 50.w,
          child: CircleButton(
            onPressed: () => Navigator.of(context).pop(),
            iconSize: 10.sp,
            icon: AssetPath.expandLeft,
          ),
        ),
        if (state.status.loading)
          Center(
            child: TwoRotatingArc(
              color: AppColor.fader,
              size: 50.w,
              blur: true,
            ),
          )
      ],
    ),
  );
},
),
);
}

Positioned _backgroundCircle() {
  return Positioned(
    left: -55.w,
    top: -35.w,
    child: CircleAvatar(
      radius: 150.sp,
      backgroundColor: AppColor.main.withOpacity(0.2),
    ),
  );
}

Align _readyText(ThemeData theme) {
  return Align(
    alignment: Alignment.centerLeft,
```

```
      child: RichText(
        text: TextSpan(
          style: TextStyle(
            color: theme.mainText,
            fontSize: 28.sp,
          ),
          children: const [
            TextSpan(text: 'Let’s '),
            TextSpan(
              text: 'Sign In',
              style: TextStyle(
                fontWeight: FontWeight.bold,
              ),
            ),
          ],
        ),
      ),
    );
  }
}

class _EmailTextFieldBox extends StatelessWidget {
  const _EmailTextFieldBox();

  @override
  Widget build(BuildContext context) {
    final theme = Theme.of(context);
    return BlocBuilder<EmailValidatorCubit, EmailValidatorState>(
      builder: (context, state) {
        return Column(
          crossAxisAlignment: CrossAxisAlignment.start,
          children: [
            TextFieldBoxDecoration(
              hasError: state.status.hasError,
              child: Row(
                children: [
                  SvgPicture.asset(
                    AssetPath.mail,
                    width: 18.w,
                    colorFilter: ColorFilter.mode(
                      theme.icon.withOpacity(0.5),
                      BlendMode.srcIn,
                    ),
                  ),
                  AppSpace.w10,
                  Expanded(
                    child: TextField(
```

```
                textAlignVertical: TextAlignVertical.center,
                maxLines: 1,
                keyboardType: TextInputType.emailAddress,
                onChanged: (value) {
                  context.read<EmailValidatorCubit>().validate(value);
                  context.read<LoginCubit>().email(value);
                },
                decoration: InputDecoration(
                    border: InputBorder.none,
                    hintText: 'Email',
                    hintStyle: TextStyle(
                      fontSize: 12.sp,
                      fontWeight: FontWeight.normal,
                      color: theme.icon.withOpacity(0.5),
                    )),
                ),
              ),
              AppSpace.w10,
            ],
          ),
        ),
        Padding(
          padding: AppPadding.h10,
          child: Text(
            state.status.msg,
            style: TextStyle(
              color: AppColor.red,
              fontSize: 12.sp,
              fontWeight: FontWeight.w500,
            ),
          ),
        )
      ],
    );
  },
);
}
}

class _PasswordTextFieldBox extends StatelessWidget {
  const _PasswordTextFieldBox();

  @override
  Widget build(BuildContext context) {
    final theme = Theme.of(context);
    return BlocBuilder<PasswordValidatorCubit, PassValidatorState>(
      builder: (context, state) {
```

```
return Column(
  crossAxisAlignment: CrossAxisAlignment.start,
  children: [
    TextFieldBoxDecoration(
      hasError: state.status.hasError,
      child: Row(
        children: [
          SvgPicture.asset(
            AssetPath.lock,
            width: 18.w,
            colorFilter: ColorFilter.mode(
                theme.icon.withOpacity(0.5), BlendMode.srcIn),
          ),
          AppSpace.w10,
          Expanded(
            child: TextField(
              textAlignVertical: TextAlignVertical.center,
              maxLines: 1,
              obscureText: state.obscure,
              onChanged: (value) {
                context.read<PasswordValidatorCubit>().validate(value);
                context.read<LoginCubit>().password(value);
              },
              decoration: InputDecoration(
                border: InputBorder.none,
                hintText: 'Password',
                hintStyle: TextStyle(
                  fontSize: 12.sp,
                  fontWeight: FontWeight.normal,
                  color: theme.icon.withOpacity(0.5),
                ),
              ),
            ),
          ),
          AppSpace.w10,
          ShrinkButton(
            onPressed: () =>
                context.read<PasswordValidatorCubit>().obscure(),
            child: SvgPicture.asset(
              state.obscure ? AssetPath.hide : AssetPath.show,
              width: 18.w,
              colorFilter: ColorFilter.mode(
                  theme.icon.withOpacity(0.5), BlendMode.srcIn),
            ),
          ),
        ],
      ),
```

```
          ),
          Padding(
            padding: AppPadding.h10,
            child: Text(
              state.status.msg,
              style: TextStyle(
                color: AppColor.red,
                fontSize: 12.sp,
                fontWeight: FontWeight.w500,
              ),
            ),
          )
        ],
      );
    },
  );
 }
}

class _LoginButton extends StatelessWidget {
 const _LoginButton();

 @override
 Widget build(BuildContext context) {
  final email = context.watch<EmailValidatorCubit>().state.status.canLog;
  final pass = context.watch<PasswordValidatorCubit>().state.status.canLog;

  return ShrinkButton(
    onPressed: () {
      if (email && pass) {
        context.read<LoginCubit>().login();
      }
    },
    child: Container(
      height: 55.w,
      margin: AppPadding.h10,
      padding: EdgeInsets.symmetric(horizontal: 45.w),
      decoration: BoxDecoration(
          color: AppColor.button,
          borderRadius: BorderRadius.circular(10.0),
          boxShadow: [
            BoxShadow(color: AppColor.main.withOpacity(0.2), blurRadius: 10)
          ]),
      child: Center(
        child: Text(
          'Login',
          style: TextStyle(
```

```
            fontSize: 16.sp,
            color: AppColor.white,
            fontWeight: FontWeight.bold,
          ),
        ),
      ),
    ),
  );
 }
}
```

6.Sign up screen

```
class RegisterPage extends StatelessWidget {
 const RegisterPage({super.key});

 @override
 @override
 Widget build(BuildContext context) {
  final theme = Theme.of(context);
  return MultiBlocProvider(
    providers: [
      BlocProvider(create: (create) => EmailValidatorCubit()),
      BlocProvider(create: (create) => PasswordValidatorCubit()),
      BlocProvider(create: (create) => RegisterCubit()),
    ],
    child: BlocConsumer<RegisterCubit, RegisterState>(
      listener: (context, state) {
        if (state.status.isVerified) {
          Navigator.pushNamedAndRemoveUntil(
              context, RoutesName.email, (route) => false);
        }
      },
      builder: (context, state) {
        return Scaffold(
          body: Stack(
            children: [
              _backgroundCircle(),
              ScrollDetective(
                list: SingleChildScrollView(
                  padding: AppPadding.h20.copyWith(top: 150.w),
                  child: Center(
                    child: Column(
                      crossAxisAlignment: CrossAxisAlignment.center,
                      children: [
                        _readyText(theme),
                        AppSpace.h10,
```

```
Align(
  alignment: Alignment.centerLeft,
  child: Text(
    AppString.slogan,
    style: AppTextStyle.wSubheader.copyWith(
      color: theme.mainText,
      fontSize: 12.sp,
    ),
  ),
),
// AppSpace.h50,
if (state.status.error)
  Container(
    height: 55.w,
    margin: AppPadding.h10
        .copyWith(bottom: 20.h, top: 20.h),
    padding: AppPadding.h20,
    decoration: BoxDecoration(
      color: AppColor.main,
      borderRadius: BorderRadius.circular(10.0),
    ),
    child: Center(
      child: Text(
        state.err,
        maxLines: 2,
        style: TextStyle(
          fontSize: 12.sp,
          color: AppColor.white,
          fontWeight: FontWeight.w400,
        ),
      ),
    ),
  )
else
  AppSpace.h50,
AppSpace.h10,
const _NameTextFieldBox(),
AppSpace.h20,
const _EmailTextFieldBox(),
AppSpace.h10,
const _PasswordTextFieldBox(),
AppSpace.h50,
const _RegisterButton(),
AppSpace.h50,
Align(
  alignment: Alignment.bottomCenter,
  child: ShrinkButton(
```

```
                onPressed: () {
                  //TODO:
                },
                child: Padding(
                  padding: AppPadding.h10,
                  child: Text(
                    'Terms of service',
                    style: TextStyle(
                      color: theme.mainText,
                      fontSize: 13.sp,
                      fontWeight: FontWeight.w500,
                    ),
                  ),
                ),
              ),
            ),
          ],
        ),
      ),
    ),
    top: Container(
      decoration:
          AppDecoration.top(theme.scaffoldBackgroundColor),
      height: 20.w,
    ),
  ),
  Positioned(
    left: 20.w,
    top: 50.w,
    child: CircleButton(
      onPressed: () => Navigator.of(context).pop(),
      iconSize: 10.sp,
      icon: AssetPath.expandLeft,
    ),
  ),
  if (state.status.loading)
    Center(
      child: TwoRotatingArc(
        color: AppColor.fader,
        size: 50.w,
        blur: true,
      ),
    ),
  ],
 ),
);
},
```

```
    ),
  );
}

Positioned _backgroundCircle() {
  return Positioned(
    left: -55.w,
    top: -35.w,
    child: CircleAvatar(
      radius: 150.sp,
      backgroundColor: AppColor.main.withOpacity(0.2),
    ),
  );
}

Align _readyText(ThemeData theme) {
  return Align(
    alignment: Alignment.centerLeft,
    child: RichText(
      text: TextSpan(
        style: TextStyle(
          color: theme.mainText,
          fontSize: 25.sp,
        ),
        children: const [
          TextSpan(text: 'Create your '),
          TextSpan(
            text: 'account',
            style: TextStyle(
              fontWeight: FontWeight.bold,
            ),
          ),
        ],
      ),
    ),
  );
}
}

class _NameTextFieldBox extends StatelessWidget {
  const _NameTextFieldBox();

  @override
  Widget build(BuildContext context) {
    final theme = Theme.of(context);
    return TextFieldBoxDecoration(
      child: Row(
```

```
      children: [
        SvgPicture.asset(
          AssetPath.user,
          width: 18.w,
          colorFilter: ColorFilter.mode(
            theme.icon.withOpacity(0.5),
            BlendMode.srcIn,
          ),
        ),
        AppSpace.w10,
        Expanded(
          child: TextField(
            textAlignVertical: TextAlignVertical.center,
            maxLines: 1,
            onChanged: (value) => context.read<RegisterCubit>().name(value),
            decoration: InputDecoration(
                border: InputBorder.none,
                hintText: 'Full name',
                hintStyle: TextStyle(
                  fontSize: 12.sp,
                  fontWeight: FontWeight.normal,
                  color: theme.icon.withOpacity(0.5),
                )),
          ),
        ),
        AppSpace.w10,
      ],
    ),
  );
 }
}

class _EmailTextFieldBox extends StatelessWidget {
 const _EmailTextFieldBox();

 @override
 Widget build(BuildContext context) {
  final theme = Theme.of(context);
  return BlocBuilder<EmailValidatorCubit, EmailValidatorState>(
    builder: (context, state) {
      return Column(
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
          TextFieldBoxDecoration(
            hasError: state.status.hasError,
            child: Row(
              children: [
```

```
            SvgPicture.asset(
              AssetPath.mail,
              width: 18.w,
              colorFilter: ColorFilter.mode(
                theme.icon.withOpacity(0.5),
                BlendMode.srcIn,
              ),
            ),
            AppSpace.w10,
            Expanded(
              child: TextField(
                textAlignVertical: TextAlignVertical.center,
                maxLines: 1,
                onChanged: (value) {
                  context.read<EmailValidatorCubit>().validate(value);
                  context.read<RegisterCubit>().email(value);
                },
                keyboardType: TextInputType.emailAddress,
                decoration: InputDecoration(
                    border: InputBorder.none,
                    hintText: 'Email',
                    hintStyle: TextStyle(
                      fontSize: 12.sp,
                      fontWeight: FontWeight.normal,
                      color: theme.icon.withOpacity(0.5),
                    )),
              ),
            ),
            AppSpace.w10,
          ],
        ),
      ),
      Padding(
        padding: AppPadding.h10,
        child: Text(
          state.status.msg,
          style: TextStyle(
            color: AppColor.red,
            fontSize: 12.sp,
            fontWeight: FontWeight.w500,
          ),
        ),
      )
    ],
  );
},
);
```

```
  }
}

class _PasswordTextFieldBox extends StatelessWidget {
  const _PasswordTextFieldBox();

  @override
  Widget build(BuildContext context) {
    final theme = Theme.of(context);
    return BlocBuilder<PasswordValidatorCubit, PassValidatorState>(
      builder: (context, state) {
        return Column(
          crossAxisAlignment: CrossAxisAlignment.start,
          children: [
            TextFieldBoxDecoration(
              hasError: state.status.hasError,
              child: Row(
                children: [
                  SvgPicture.asset(
                    AssetPath.lock,
                    width: 18.w,
                    colorFilter: ColorFilter.mode(
                        theme.icon.withOpacity(0.5), BlendMode.srcIn),
                  ),
                  AppSpace.w10,
                  Expanded(
                    child: TextField(
                      textAlignVertical: TextAlignVertical.center,
                      maxLines: 1,
                      obscureText: state.obscure,
                      onChanged: (value) {
                        context.read<PasswordValidatorCubit>().validate(value);
                        context.read<RegisterCubit>().password(value);
                      },
                      decoration: InputDecoration(
                        border: InputBorder.none,
                        hintText: 'Password',
                        hintStyle: TextStyle(
                          fontSize: 12.sp,
                          fontWeight: FontWeight.normal,
                          color: theme.icon.withOpacity(0.5),
                        ),
                      ),
                    ),
                  ),
                  AppSpace.w10,
                  ShrinkButton(
```

```
                onPressed: () =>
                  context.read<PasswordValidatorCubit>().obscure(),
                child: SvgPicture.asset(
                  state.obscure ? AssetPath.hide : AssetPath.show,
                  width: 18.w,
                  colorFilter: ColorFilter.mode(
                    theme.icon.withOpacity(0.5), BlendMode.srcIn),
                ),
              ),
            ],
          ),
        ),
        Padding(
          padding: AppPadding.h10,
          child: Text(
            state.status.msg,
            style: TextStyle(
              color: AppColor.red,
              fontSize: 12.sp,
              fontWeight: FontWeight.w500,
            ),
          ),
        )
      ],
    );
  },
);
}
}

class _RegisterButton extends StatelessWidget {
  const _RegisterButton();

  @override
  Widget build(BuildContext context) {
    final email = context.watch<EmailValidatorCubit>().state.status.canLog;
    final pass = context.watch<PasswordValidatorCubit>().state.status.canLog;

    return ShrinkButton(
      onPressed: () {
        if (email && pass) {
          log('onPressed');
          context.read<RegisterCubit>().register();
        }
      },
      child: Container(
        height: 55.w,
```

```
        margin: AppPadding.h10,
        padding: EdgeInsets.symmetric(horizontal: 45.w),
        decoration: BoxDecoration(
          color: AppColor.button,
          borderRadius: BorderRadius.circular(10.0),
          boxShadow: [
            BoxShadow(color: AppColor.main.withOpacity(0.2), blurRadius: 10)
          ]),
        child: Center(
          child: Text(
          'Register',
          style: TextStyle(
            fontSize: 16.sp,
            color: AppColor.white,
            fontWeight: FontWeight.bold,
          ),
        ),
      ),
    ),
  );
}
}
```

7.Authentication

```
class AuthPage extends StatelessWidget {
  const AuthPage({super.key});

  @override
  Widget build(BuildContext context) {
    final theme = Theme.of(context);
    return Scaffold(
      body: Stack(
        children: [
          _backgroundCircle(),
          Padding(
            padding: AppPadding.h20.copyWith(top: 50.w),
            child: SingleChildScrollView(
              child: Column(
                children: [
                  AppSpace.h50,
                  _logo(theme),
                  AppSpace.h10,
                  Text(
                    AppString.appname,
                    style: AppTextStyle.wHeader.copyWith(
```

```
                color: theme.mainText,
                fontSize: 30.sp,
              ),
            ),
            Text(
              AppString.slogan,
              style: AppTextStyle.wSubheader.copyWith(
                color: theme.mainText,
                fontSize: 12.sp,
              ),
            ),
            AppSpace.h50,
            _readyText(theme),
            AppSpace.h50,
            const ContinueWithEmailButton(),
            AppSpace.h50,
            const OrLine(),
            AppSpace.h50,
            const SocialMediaButtonsRow(),
            AppSpace.h50,
            const RegisterTextButton(),
          ],
        ),
      ),
    ),
  ],
),
);
}

Positioned _backgroundCircle() {
  return Positioned(
    left: -55.w,
    top: -35.w,
    child: CircleAvatar(
      radius: 150.sp,
      backgroundColor: AppColor.main.withOpacity(0.2),
    ),
  );
}

Align _readyText(ThemeData theme) {
  return Align(
    alignment: Alignment.centerLeft,
    child: RichText(
      textAlign: TextAlign.start,
      text: TextSpan(
```

```
        style: TextStyle(
          color: theme.mainText,
          fontSize: 24.sp,
        ),
        children: const [
          TextSpan(text: 'Ready to '),
          TextSpan(
            text: 'explore?',
            style: TextStyle(
              fontWeight: FontWeight.bold,
            ),
          ),
        ],
      ),
    ),
  );
}

SvgPicture _logo(ThemeData theme) {
  return SvgPicture.asset(
    AssetPath.logo,
    width: 200.w,
    colorFilter: ColorFilter.mode(theme.mainIcon, BlendMode.srcIn),
  );
}
}
```

8.Profile screen

```
class ProfilePage extends StatelessWidget {
  const ProfilePage({super.key});

  @override
  Widget build(BuildContext context) {
    return BlocProvider(
      create: (context) => ProfileTabViewCubit()..execute(),
      child: const OpacityTranslateAnimation(
        child: SafeArea(
          child: Stack(
            children: [
              ProfilePageContent(),
              ProfileAppBar(),
            ],
          ),
        ),
      ),
    );
  }
```

```
}
```

9.Edit profile screen

```
class ProfileEditPage extends StatelessWidget {
  const ProfileEditPage({super.key});

  @override
  Widget build(BuildContext context) {
    return Padding(
      padding: AppPadding.h20.copyWith(top: 80.w),
      child: const _BodyContent(),
    );
  }
}

class _BodyContent extends StatelessWidget {
  const _BodyContent();

  @override
  Widget build(BuildContext context) {
    final bottom = MediaQuery.of(context).viewInsets.bottom;
    return SingleChildScrollView(
      physics: const BouncingScrollPhysics(),
      padding: EdgeInsets.only(bottom: bottom),
      child: Column(
        children: [
          const _ProfileEditTopTextArea(),
          AppSpace.h40,
          const _ProfilePic(),
          AppSpace.h30,
          const _UserNameTextField(),
          AppSpace.h20,
          const _PhoneTextField(),
          AppSpace.h20,
          const _UserEmailAddress(),
        ],
      ),
    );
  }
}

class _ProfilePic extends StatelessWidget {
  const _ProfilePic();

  @override
  Widget build(BuildContext context) {
    final state = context.watch<ProfileEditCubit>().state;
```

```
    return Stack(
      children: [
        ClipRRect(
          borderRadius: BorderRadius.circular(100.0),
          child: SizedBox.square(
            dimension: 120.w,
            child: state.file != null
                ? Image.file(state.file!, fit: BoxFit.cover)
                : CircleAvatar(
                    backgroundColor: AppColor.softGrey,
                    child: SvgPicture.asset(AssetPath.emptyUser),
                  ),
          ),
        ),
        Positioned(
          bottom: 0,
          right: 0,
          child: CircleButton(
            onPressed: () => context.read<ProfileEditCubit>().pickImage(),
            icon: AssetPath.edit,
            bgColor: AppColor.main,
            radius: 30.w,
            color: AppColor.white,
          ),
        )
      ],
    );
  }
}

class _ProfileEditTopTextArea extends StatelessWidget {
  const _ProfileEditTopTextArea();

  @override
  Widget build(BuildContext context) {
    final theme = Theme.of(context);
    return Column(
      children: [
        AppSpace.h50,
        _readyText(theme),
        AppSpace.h10,
        Align(
          alignment: Alignment.centerLeft,
          child: Text(
            'You can edit this later on your account settings.',
            style: AppTextStyle.wSubheader.copyWith(
              color: theme.mainText,
```

```
            fontSize: 12.sp,
          ),
        ),
      ),
    ],
  );
}

Align _readyText(ThemeData theme) {
  return Align(
    alignment: Alignment.centerLeft,
    child: RichText(
      text: TextSpan(
        style: TextStyle(
          color: theme.mainText,
          fontSize: 22.sp,
        ),
        children: const [
          TextSpan(text: 'Fill your '),
          TextSpan(
            text: 'information ',
            style: TextStyle(
              fontWeight: FontWeight.bold,
            ),
          ),
          TextSpan(text: 'below '),
        ],
      ),
    ),
  );
}
}

class _UserNameTextField extends StatelessWidget {
  const _UserNameTextField();

  @override
  Widget build(BuildContext context) {
    return _TextBoxBackground(
      hintText: 'full name',
      onChanged: (value) => context.read<ProfileEditCubit>().saveName(value),
      icon: AssetPath.user,
    );
  }
}

class _PhoneTextField extends StatelessWidget {
  const _PhoneTextField();
```

```
  @override
  Widget build(BuildContext context) {
    return _TextBoxBackground(
      hintText: 'mobile number',
      onChanged: (value) =>
          context.read<ProfileEditCubit>().savePhoneNumber(value),
      icon: AssetPath.phone,
    );
  }
}

class _TextBoxBackground extends StatelessWidget {
  const _TextBoxBackground({
    required this.hintText,
    required this.icon,
    required this.onChanged,
  });

  final String hintText;
  final String icon;
  final Function(String)? onChanged;
  @override
  Widget build(BuildContext context) {
    final theme = Theme.of(context);
    return Container(
      padding: AppPadding.all10,
      decoration: BoxDecoration(
        color: theme.cardColor,
        borderRadius: BorderRadius.circular(10.0),
      ),
      child: Row(
        children: [
          Expanded(
            child: TextField(
              onChanged: onChanged,
              scrollPadding: EdgeInsets.only(
                  bottom: MediaQuery.of(context).viewInsets.bottom),
              decoration: InputDecoration(
                border: InputBorder.none,
                hintText: hintText,
                hintStyle: TextStyle(
                  color: theme.text.withOpacity(0.5),
                  fontSize: 14.sp,
                  fontWeight: FontWeight.w300,
                ),
              ),
            ),
```

```
        ),
        AppSpace.w10,
        SvgPicture.asset(
          icon,
          width: 18.sp,
          colorFilter: ColorFilter.mode(theme.text, BlendMode.srcIn),
        ),
      ],
    ),
  );
 }
}

class _UserEmailAddress extends StatelessWidget {
 const _UserEmailAddress();

 @override
 Widget build(BuildContext context) {
  final theme = Theme.of(context);
  final user = context.read<AppBloc>().state.user;
  return Container(
    padding: AppPadding.all20,
    decoration: BoxDecoration(
      color: theme.icon,
      borderRadius: BorderRadius.circular(10.0),
    ),
    child: Row(
      children: [
        Expanded(
          child: Text(
            user.email!,
            style: TextStyle(
              color: AppColor.white,
              fontSize: 14.sp,
            ),
          ),
        ),
        AppSpace.w10,
        SvgPicture.asset(
          AssetPath.mail,
          width: 18.sp,
          colorFilter:
              const ColorFilter.mode(AppColor.white, BlendMode.srcIn),
        ),
      ],
    ),
  );
 }
```

```
}
```

10.Search screen

```
class SearchPage extends StatelessWidget {
  const SearchPage({super.key});

  @override
  Widget build(BuildContext context) {
    return MultiBlocProvider(
      providers: [
        BlocProvider(create: (create) => ListTypeCubit()),
        BlocProvider(create: (create) => SearchBloc()),
      ],
      child: Scaffold(
        body: SafeArea(
          child: Padding(
            padding: AppPadding.h20,
            child: Column(
              children: [
                const _SearchPageAppBar(),
                AppSpace.h20,
                const _SearchTextField(),
                AppSpace.h20,
                const _FoundCountAndListType(),
                AppSpace.h20,
                Expanded(
                  child: BlocBuilder<SearchBloc, SearchState>(
                    builder: (context, state) {
                      if (state is SearchFound) {
                        return _SearchFoundListView(state: state);
                      }
                      if (state is SearchError) {
                        return AppFailureWidget(msg: state.msg);
                      }
                      if (state is SearchLoading) {
                        return const CircularProgressIndicator.adaptive();
                      }
                      if (state is SearchNotFound) {
                        return const _SearchNotFoundWidget();
                      }
                      return const SizedBox();
                    },
                  ),
                ),
              ],
            ),
          ),
```

```
      ),
    ),
  );
 }
}

class _FoundCountAndListType extends StatelessWidget {
 const _FoundCountAndListType();

 @override
 Widget build(BuildContext context) {
  final theme = Theme.of(context);
  return BlocBuilder<SearchBloc, SearchState>(
    builder: (context, state) {
      if (state is SearchFound) {
        return Row(
          mainAxisAlignment: MainAxisAlignment.spaceBetween,
          children: [
            Text(
              "Found ${state.properties.length} estates",
              style: TextStyle(
                color: theme.text,
                fontSize: 16.sp,
                fontWeight: FontWeight.normal,
              ),
            ),
            const _ListShortButtonRow()
          ],
        );
      }
      return const SizedBox();
    },
  );
 }
}

class _ListShortButtonRow extends StatelessWidget {
 const _ListShortButtonRow();

 @override
 Widget build(BuildContext context) {
  final theme = Theme.of(context);
  return Container(
    padding: AppPadding.all5,
    decoration: BoxDecoration(
      color: theme.cardColor,
      borderRadius: BorderRadius.circular(50.0),
    ),
```

```
      child: Row(
        children: [
          const _GridListShortButton(),
          AppSpace.w20,
          const _ListShortButton(),
        ],
      ),
    );
  }
}

class _GridListShortButton extends StatelessWidget {
  const _GridListShortButton();

  @override
  Widget build(BuildContext context) {
    final theme = Theme.of(context);
    return BlocBuilder<ListTypeCubit, ListTypeState>(
      builder: (context, state) {
        return ShrinkButton(
          onPressed: () => context.read<ListTypeCubit>().challenge(),
          child: AnimatedContainer(
            duration: const Duration(milliseconds: 1000),
            padding: AppPadding.all5,
            decoration: BoxDecoration(
              color: state.isGrid ? theme.scaffoldBackgroundColor : null,
              borderRadius: BorderRadius.circular(50.0),
            ),
            child: SvgPicture.asset(
              state.isGrid ? AssetPath.dashboardFill : AssetPath.dashboard,
              colorFilter: ColorFilter.mode(
                state.isGrid ? theme.mainIcon : theme.mainIcon.withOpacity(0.3),
                BlendMode.srcIn,
              ),
            ),
          ),
        );
      },
    );
  }
}

class _ListShortButton extends StatelessWidget {
  const _ListShortButton();

  @override
  Widget build(BuildContext context) {
    final theme = Theme.of(context);
```

```
    return BlocBuilder<ListTypeCubit, ListTypeState>(
      builder: (context, state) {
        return ShrinkButton(
          onPressed: () => context.read<ListTypeCubit>().challenge(),
          child: AnimatedContainer(
            duration: const Duration(milliseconds: 1000),
            padding: AppPadding.all5,
            decoration: BoxDecoration(
              color: state.isGrid ? null : theme.scaffoldBackgroundColor,
              borderRadius: BorderRadius.circular(50.0),
            ),
            child: SvgPicture.asset(
              state.isGrid ? AssetPath.viewDay : AssetPath.viewDayFill,
              colorFilter: ColorFilter.mode(
                state.isGrid ? theme.mainIcon.withOpacity(0.3) : theme.mainIcon,
                BlendMode.srcIn,
              ),
            ),
          ),
        );
      },
    );
  }
}

class _SearchNotFoundWidget extends StatelessWidget {
  const _SearchNotFoundWidget();

  @override
  Widget build(BuildContext context) {
    return Align(
      alignment: const Alignment(0, -0.5),
      child: Padding(
        padding: AppPadding.main,
        child: Column(
          mainAxisSize: MainAxisSize.min,
          children: [
            SvgPicture.asset(AssetPath.emptyBox),
            Row(
              mainAxisSize: MainAxisSize.min,
              children: [
                Text(
                  "Search ",
                  style: TextStyle(
                    color: AppColor.main,
                    fontSize: 22.sp,
                    fontWeight: FontWeight.normal,
                  ),
```

```
            ),
            Text(
              "not found ",
              style: TextStyle(
                color: AppColor.main,
                fontSize: 22.sp,
                fontWeight: FontWeight.bold,
              ),
            ),
          ],
        ),
        Text(
          AppString.notFound,
          textAlign: TextAlign.center,
          style: TextStyle(
            color: AppColor.main,
            fontSize: 12.sp,
            fontWeight: FontWeight.w300,
          ),
        ),
      ],
    ),
  ),
);
}
}

class _SearchFoundListView extends StatelessWidget {
  const _SearchFoundListView({required this.state});
  final SearchFound state;
  @override
  Widget build(BuildContext context) {
    final theme = Theme.of(context);
    return ScrollDetective(
      list: BlocBuilder<ListTypeCubit, ListTypeState>(
        builder: (context, type) {
          if (type.isGrid) {
            return _GridList(state: state);
          }
          return _VerticalList(state: state);
        },
      ),
      top: Container(
          decoration: AppDecoration.top(theme.scaffoldBackgroundColor),
          height: 20.w),
    );
  }
}
```

```
class _GridList extends StatelessWidget {
  const _GridList({required this.state});

  final SearchFound state;

  @override
  Widget build(BuildContext context) {
    return TranslateRightAnimation(
      duration: const Duration(milliseconds: 800),
      child: GridView.count(
        crossAxisCount: 2,
        padding: AppPadding.v5.copyWith(bottom: 50.w),
        childAspectRatio: 0.6,
        crossAxisSpacing: 10.w,
        mainAxisSpacing: 20.w,
        shrinkWrap: true,
        children: state.properties
            .map((e) => _NearbyGridItemCard(property: e))
            .toList(),
      ),
    );
  }
}

class _VerticalList extends StatelessWidget {
  const _VerticalList({required this.state});

  final SearchFound state;

  @override
  Widget build(BuildContext context) {
    return ListView.builder(
      itemCount: state.properties.length,
      padding: AppPadding.v5.copyWith(bottom: 50.w),
      shrinkWrap: true,
      itemBuilder: (context, i) => TranslateLeftAnimation(
        duration: Duration(milliseconds: 500 + (100 * i)),
        child: _FeaturedItemCard(
          property: state.properties[i],
        ),
      ),
    );
  }
}

class _FeaturedItemCard extends StatelessWidget {
  const _FeaturedItemCard({required this.property});
```

```
  final Property property;
  void goToPage(BuildContext context) {
    Navigator.pushNamed(context, RoutesName.details, arguments: property);
  }

  @override
  Widget build(BuildContext context) {
    final theme = Theme.of(context);
    return ShrinkButton(
      onPressed: () => goToPage(context),
      child: Container(
        padding: EdgeInsets.all(10.w),
        height: 120.w,
        margin: AppPadding.v5.copyWith(bottom: 20.h),
        decoration: BoxDecoration(
          color: theme.cardColor,
          borderRadius: BorderRadius.circular(25.0),
        ),
        child: Row(
          crossAxisAlignment: CrossAxisAlignment.start,
          children: [
            _FeaturedItemCardLeftSide(property: property),
            AppSpace.w10,
            _FeaturedItemCardRightSide(property: property)
          ],
        ),
      ),
    );
  }
}

class _FeaturedItemCardRightSide extends StatelessWidget {
  const _FeaturedItemCardRightSide({required this.property});

  final Property property;

  @override
  Widget build(BuildContext context) {
    final theme = Theme.of(context);
    return Expanded(
      child: Column(
        crossAxisAlignment: CrossAxisAlignment.start,
        mainAxisAlignment: MainAxisAlignment.spaceAround,
        children: [
          Text(
            property.area!,
            maxLines: 2,
            style: TextStyle(
```

```
    color: theme.text,
    fontWeight: FontWeight.bold,
  ),
),
Row(
  children: [
    SvgPicture.asset(
      AssetPath.starFill,
      width: 18.sp,
      colorFilter: const ColorFilter.mode(
        AppColor.orange,
        BlendMode.srcIn,
      ),
    ),
    AppSpace.w5,
    Text(
      property.reviewCount.toString(),
      style: TextStyle(
        color: theme.text,
        fontWeight: FontWeight.bold,
      ),
    ),
  ],
),
Row(
  children: [
    SvgPicture.asset(
      AssetPath.placeFill,
      width: 12.sp,
      colorFilter: ColorFilter.mode(
        theme.mainIcon,
        BlendMode.srcIn,
      ),
    ),
    AppSpace.w5,
    Text(
      property.city!,
      style: TextStyle(
        color: theme.text,
        fontSize: 12.sp,
        fontWeight: FontWeight.w300,
      ),
    ),
  ],
),
RichText(
  text: TextSpan(
    style: TextStyle(
```

```
            color: theme.text,
            fontSize: 16.sp,
            fontWeight: FontWeight.bold,
          ),
          children: [
            TextSpan(text: '${property.prices![0].currency} '),
            TextSpan(text: property.prices![0].amountMin.toString()),
            TextSpan(
              text: '/${property.prices![0].isSale}',
              style: TextStyle(
                fontSize: 12.sp,
                fontWeight: FontWeight.w300,
              ),
            ),
          ],
        ),
      ),
    ],
  ),
);
}
}

class _FeaturedItemCardLeftSide extends StatelessWidget {
  const _FeaturedItemCardLeftSide({required this.property});

  final Property property;

  @override
  Widget build(BuildContext context) {
    return Expanded(
      child: Stack(
        children: [
          HttpsImage(
            placeholder: Container(
              decoration: BoxDecoration(
                color: AppColor.black,
                borderRadius: BorderRadius.circular(15.0),
              ),
            ),
            imageUrl: property.image!,
            imageBuilder: (context, img) {
              return Container(
                decoration: BoxDecoration(
                  color: AppColor.softGrey,
                  borderRadius: BorderRadius.circular(15.0),
                  image: DecorationImage(image: img, fit: BoxFit.cover),
                ),
```

```
            );
          },
        ),
        Positioned(
          top: 5.w,
          left: 5.w,
          child: BlocBuilder<FavoriteCubit, FavoriteState>(
            builder: (context, state) {
              final bool hasData = state.properties.contains(property);
              return CircleButton(
                onPressed: () => context.read<FavoriteCubit>().add(property),
                icon: hasData ? AssetPath.hartFill : AssetPath.hart,
                radius: 30.sp,
                color: hasData ? AppColor.white : AppColor.radial,
                bgColor: hasData ? AppColor.button : AppColor.softGrey,
              );
            },
          ),
        ),
        Positioned(
          bottom: 5.w,
          left: 5.w,
          child: Container(
            height: 25.w,
            padding: AppPadding.h20,
            decoration: BoxDecoration(
              color: AppColor.main,
              borderRadius: BorderRadius.circular(8.0),
            ),
            child: Center(
              child: Text(
                property.propertyType!,
                maxLines: 1,
                style: TextStyle(color: AppColor.white, fontSize: 10.sp),
              ),
            ),
          ),
        )
      ],
    ),
  );
 }
}

class _SearchTextField extends StatelessWidget {
 const _SearchTextField();

 @override
```

```
  Widget build(BuildContext context) {
    final theme = Theme.of(context);
    return Container(
      // margin: AppPadding.v15,
      height: 56.w,
      padding: AppPadding.all15,
      decoration: BoxDecoration(
        color: theme.cardColor,
        borderRadius: BorderRadius.circular(10.0),
      ),
      child: Row(
        children: [
          SvgPicture.asset(
            AssetPath.search,
            colorFilter: ColorFilter.mode(
              theme.mainIcon.withOpacity(0.7),
              BlendMode.srcIn,
            ),
          ),
          Expanded(
            child: TextField(
              textAlignVertical: TextAlignVertical.center,
              onChanged: (value) =>
                  context.read<SearchBloc>().add(Search(text: value)),
              decoration: InputDecoration(
                border: InputBorder.none,
                hintText: 'Search House, Apartment, etc',
                hintStyle: TextStyle(color: AppColor.hint, fontSize: 12.sp),
              ),
            ),
          ),
        ].horizontalGap(15.w),
      ),
    );
  }
}

class _NearbyGridItemCard extends StatelessWidget {
  const _NearbyGridItemCard({required this.property});
  final Property property;
  void goToPage(BuildContext context) {
    Navigator.pushNamed(context, RoutesName.details, arguments: property);
  }

  @override
  Widget build(BuildContext context) {
    final theme = Theme.of(context);
    return ShrinkButton(
```

```
      onPressed: () => goToPage(context),
      child: Container(
        padding: EdgeInsets.all(10.w),
        decoration: BoxDecoration(
          color: theme.cardColor,
          borderRadius: BorderRadius.circular(25.0),
        ),
        child: Column(
          children: [
            _NearbyItemCardTop(property: property),
            AppSpace.h10,
            _NearbyItemCardBottom(property: property)
          ],
        ),
      ),
    );
  }
}

class _NearbyItemCardBottom extends StatelessWidget {
  const _NearbyItemCardBottom({required this.property});

  final Property property;

  @override
  Widget build(BuildContext context) {
    final theme = Theme.of(context);
    return Expanded(
      child: Column(
        mainAxisAlignment: MainAxisAlignment.spaceAround,
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
          Text(
            property.area!,
            maxLines: 1,
            style: TextStyle(
              color: theme.text,
              fontSize: 12.sp,
              fontWeight: FontWeight.bold,
            ),
          ),
          Row(
            children: [
              SvgPicture.asset(
                AssetPath.starFill,
                width: 12.sp,
                colorFilter: const ColorFilter.mode(
                  AppColor.orange,
```

```
                BlendMode.srcIn,
              ),
            ),
            AppSpace.w5,
            Text(
              property.reviewCount.toString(),
              style: TextStyle(
                color: theme.text,
                fontWeight: FontWeight.bold,
              ),
            ),
            AppSpace.w10,
            Expanded(
              child: Row(
                children: [
                  SvgPicture.asset(
                    AssetPath.placeFill,
                    width: 12.sp,
                    colorFilter: ColorFilter.mode(
                      theme.mainIcon,
                      BlendMode.srcIn,
                    ),
                  ),
                  AppSpace.w5,
                  Expanded(
                    child: Text(
                      property.city!,
                      maxLines: 1,
                      style: TextStyle(
                        color: theme.text,
                        fontSize: 10.sp,
                        fontWeight: FontWeight.w300,
                      ),
                    ),
                  ),
                ],
              ),
            ),
          ],
        ),
      ],
    ),
  );
}
}

class _NearbyItemCardTop extends StatelessWidget {
  const _NearbyItemCardTop({required this.property});
```

```
final Property property;

@override
Widget build(BuildContext context) {
  return Expanded(
    flex: 4,
    child: Stack(
      children: [
        HttpsImage(
          placeholder: Container(
            decoration: BoxDecoration(
              color: AppColor.black,
              borderRadius: BorderRadius.circular(15.0),
            ),
          ),
          imageUrl: property.image!,
          imageBuilder: (context, img) {
            return Container(
              decoration: BoxDecoration(
                color: AppColor.softGrey,
                borderRadius: BorderRadius.circular(15.0),
                image: DecorationImage(image: img, fit: BoxFit.cover),
              ),
            );
          },
        ),
        Positioned(
          top: 10.w,
          right: 10.w,
          child: BlocBuilder<FavoriteCubit, FavoriteState>(
            builder: (context, state) {
              final bool hasData = state.properties.contains(property);
              return CircleButton(
                onPressed: () => context.read<FavoriteCubit>().add(property),
                icon: hasData ? AssetPath.hartFill : AssetPath.hart,
                radius: 30.sp,
                color: hasData ? AppColor.white : AppColor.radial,
                bgColor: hasData ? AppColor.button : AppColor.softGrey,
              );
            },
          ),
        ),
        Positioned(
          bottom: 5.w,
          right: 5.w,
          child: Container(
            height: 35.w,
```

```
            padding: EdgeInsets.symmetric(horizontal: 10.w),
            decoration: BoxDecoration(
              color: AppColor.main,
              borderRadius: BorderRadius.circular(8.0),
            ),
            child: Center(
              child: RichText(
                text: TextSpan(
                  style: TextStyle(
                    color: AppColor.white,
                    fontSize: 14.sp,
                    fontWeight: FontWeight.bold,
                  ),
                  children: [
                    TextSpan(text: '${property.prices![0].currency} '),
                    TextSpan(text: property.prices![0].amountMin.toString()),
                    TextSpan(
                      text: '/${property.prices![0].isSale}',
                      style: TextStyle(
                        fontSize: 10.sp,
                        color: AppColor.white,
                        fontWeight: FontWeight.w300,
                      ),
                    ),
                  ],
                ),
              ),
            ),
          ),
        )
      ],
    ),
  );
}
}

class _SearchPageAppBar extends StatelessWidget {
  const _SearchPageAppBar();

  @override
  Widget build(BuildContext context) {
    final theme = Theme.of(context);
    return Row(
      children: [
        CircleButton(
          onPressed: () => Navigator.of(context).pop(),
          iconSize: 10.sp,
          color: theme.mainIcon,
```

```
        icon: AssetPath.expandLeft,
      ),
      Expanded(
        child: Text(
          'Search results',
          textAlign: TextAlign.center,
          style: TextStyle(
            color: theme.mainText,
            fontSize: 16.sp,
            fontWeight: FontWeight.w500,
          ),
        ),
      ),
      CircleButton(
        onPressed: () {
          //TODO:
        },
        iconSize: 10.sp,
        color: theme.mainIcon,
        icon: AssetPath.filter,
      )
    ],
  );
 }
}
```

11.Details screen

```
import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
import 'package:flutter_screenutil/flutter_screenutil.dart';
import 'package:flutter_svg/flutter_svg.dart';
import 'package:proplink/src/core/constants/constants.dart';
import 'package:proplink/src/core/extension/extension.dart';
import 'package:proplink/src/core/theme/theme.dart';
import 'package:proplink/src/core/widgets/widgets.dart';
import 'package:proplink/src/features/home/data/models/property.dart';
import '../cubit/favorite.dart';
import 'details.sections/sections.dart';

class DetailsScreen extends StatelessWidget {
  const DetailsScreen({super.key, required this.property});
  final Property property;
  @override
  Widget build(BuildContext context) {
    final theme = Theme.of(context);
    return Scaffold(
      body: SafeArea(
        child: Stack(
```

```
        children: [
          ScrollDetective(
            list: ListView(
              padding: EdgeInsets.zero,
              children: [
                ImageSection(property: property),
                TitleSection(property: property),
                Padding(
                  padding: AppPadding.h20,
                  child: Divider(thickness: 3, color: theme.cardColor),
                ),
                FacilitiesSection(property: property),
                Padding(
                  padding: AppPadding.h20,
                  child: Text(
                    property.overView!,
                    style: TextStyle(color: theme.text),
                  ),
                ),
                BrokerSection(property: property),
                LocationSection(property: property),
                ReviewSection(property: property),
                SizedBox(height: 150.h),
              ].verticalGap(12.w),
            ),
            top: Container(
                decoration: AppDecoration.top(theme.scaffoldBackgroundColor),
                height: 20.w),
          ),
          _AppBar(property: property),
          _BottomNavBar(property: property)
        ],
      ),
    ),
  );
}
}

class _BottomNavBar extends StatelessWidget {
  const _BottomNavBar({required this.property});

  final Property property;

  @override
  Widget build(BuildContext context) {
    final theme = Theme.of(context);
    return Align(
      alignment: Alignment.bottomCenter,
```

```
      child: Container(
        height: 100.w,
        decoration: AppDecoration.bottomAppBar(
          theme.appBarTheme.backgroundColor!,
        ),
        child: Row(
          mainAxisAlignment: MainAxisAlignment.spaceAround,
          children: [
            _RightSidePriceArea(property: property),
            ShrinkButton(
              onPressed: () {
                //TODO:
              },
              child: Container(
                height: 56.w,
                decoration: BoxDecoration(
                  color: AppColor.button,
                  borderRadius: BorderRadius.circular(16.0),
                ),
                padding: AppPadding.h20,
                child: Center(
                  child: Row(
                    children: [
                      Text(
                        'Booking Now',
                        style: TextStyle(
                          fontSize: 16.sp,
                          color: AppColor.white,
                          fontWeight: FontWeight.bold,
                        ),
                      ),
                      AppSpace.w10,
                      SvgPicture.asset(
                        AssetPath.arrowRight,
                        colorFilter: const ColorFilter.mode(
                            AppColor.white, BlendMode.srcIn),
                      ),
                    ],
                  ),
                ),
              ),
            )
          ],
        ),
      ),
    );
  }
}
```

```
class _RightSidePriceArea extends StatelessWidget {
  const _RightSidePriceArea({required this.property});

  final Property property;

  @override
  Widget build(BuildContext context) {
    final theme = Theme.of(context);
    return RichText(
      textAlign: TextAlign.start,
      text: TextSpan(
        style: TextStyle(
          color: theme.mainText,
          fontSize: 22.sp,
          fontWeight: FontWeight.bold,
        ),
        children: [
          TextSpan(text: '${property.prices![0].currency} '),
          TextSpan(text: property.prices![0].amountMin.toString()),
          TextSpan(
            text: '\npre ${property.prices![0].isSale}',
            style: TextStyle(
              fontSize: 12.sp,
              color: theme.mainText,
              fontWeight: FontWeight.w300,
            ),
          ),
        ],
      ),
    );
  }
}

class _AppBar extends StatelessWidget {
  const _AppBar({required this.property});
  final Property property;
  @override
  Widget build(BuildContext context) {
    final theme = Theme.of(context);
    return Positioned(
      top: 20.w,
      left: 20.w,
      right: 20.w,
      child: Row(
        children: [
          CircleButton(
            onPressed: () => Navigator.of(context).pop(),
```

```
          iconSize: 10.sp,
          color: theme.text,
          icon: AssetPath.expandLeft,
        ),
        const Spacer(),
        CircleButton(
          onPressed: () {
            //TODO:
          },
          iconSize: 10.sp,
          color: theme.text,
          icon: AssetPath.export,
        ),
        AppSpace.w10,
        BlocBuilder<FavoriteCubit, FavoriteState>(
          builder: (context, state) {
            final bool hasData = state.properties.contains(property);
            return CircleButton(
              onPressed: () => context.read<FavoriteCubit>().add(property),
              icon: hasData ? AssetPath.hartFill : AssetPath.hart,
              iconSize: 10.sp,
              color: hasData ? AppColor.white : theme.mainIcon,
              bgColor: hasData ? AppColor.button : theme.cardColor,
            );
          },
        ),
      ],
    ),
  );
 }
}
```

12.Favourites screen

```
class FavoritePage extends StatelessWidget {
 const FavoritePage({super.key});

 @override
 Widget build(BuildContext context) {
  return BlocProvider(
    create: (context) => ListTypeCubit(),
    child: OpacityTranslateAnimation(
      child: SafeArea(
        child: Padding(
          padding: AppPadding.h20,
          child: Column(
            children: [
              const _FavoriteAppBar(),
```

```
            AppSpace.h20,
            const _FavoritePageContent(),
          ],
        ),
      ),
    ),
  ),
);
}
}

class _FavoriteAppBar extends StatelessWidget {
  const _FavoriteAppBar();

  @override
  Widget build(BuildContext context) {
    final theme = Theme.of(context);
    final isEmpty = context.watch<FavoriteCubit>().state.properties.isNotEmpty;
    return Row(
      mainAxisAlignment: MainAxisAlignment.spaceBetween,
      children: [
        SizedBox(width: 45.w),
        Text(
          'My favorite',
          style: TextStyle(
            color: theme.mainText,
            fontSize: 16.sp,
            fontWeight: FontWeight.w500,
          ),
        ),
        CircleButton(
          onPressed: () {
            if (isEmpty) {
              AppDialogBox(
                    context: context,
                    title: 'delete the all favorites?',
                    descreption:
                        'are you sure you want to remove all favorites?')
                .confirmation
                .then((value) => context.read<FavoriteCubit>().removeAll());
            }
          },
          icon: AssetPath.trash,
          iconSize: 10.sp,
          color: theme.text,
          bgColor: theme.cardColor,
        ),
      ],
```

```
    );
  }
}

///

class _FavoritePageContent extends StatelessWidget {
  const _FavoritePageContent();

  @override
  Widget build(BuildContext context) {
    return BlocBuilder<FavoriteCubit, FavoriteState>(
      builder: (context, state) {
        if (state.properties.isEmpty) {
          return const _FavoriteItemsEmpty();
        }
        return _FavoriteListView(properties: state.properties);
      },
    );
  }
}

class _FavoriteListView extends StatelessWidget {
  const _FavoriteListView({required this.properties});
  final List<Property> properties;
  @override
  Widget build(BuildContext context) {
    final theme = Theme.of(context);
    return Expanded(
      child: Column(
        children: [
          const _FoundCountAndListType(),
          AppSpace.h20,
          Expanded(
            child: ScrollDetective(
              list: BlocBuilder<ListTypeCubit, ListTypeState>(
                builder: (context, type) {
                  if (type.isGrid) {
                    return _GridViewStyle(properties: properties);
                  }
                  return _VerticalList(properties: properties);
                },
              ),
              top: Container(
                  decoration: AppDecoration.top(theme.scaffoldBackgroundColor),
                  height: 20.w),
            ),
          ),
```

```
        AppSpace.h50,
      ],
    ),
  );
 }
}

class _VerticalList extends StatelessWidget {
 const _VerticalList({required this.properties});

 final List<Property> properties;

 @override
 Widget build(BuildContext context) {
  return ListView.separated(
    itemCount: properties.length,
    padding: AppPadding.v5.copyWith(bottom: 50.w),
    shrinkWrap: true,
    itemBuilder: (context, i) => TranslateLeftAnimation(
      duration: Duration(milliseconds: 500 + (100 * i)),
      child: SizedBox(
        height: 120.w,
        child: Material(
          color: AppColor.main,
          shape: RoundedRectangleBorder(
            borderRadius: BorderRadius.circular(25.0),
          ),
          child: Dismissible(
            direction: DismissDirection.endToStart,
            background: Align(
              alignment: Alignment.centerRight,
              child: Padding(
                padding: AppPadding.h20,
                child: const Icon(Icons.delete, color: AppColor.white),
              ),
            ),
            key: Key(properties[i].id!),
            onDismissed: (direction) => remove(properties[i], context),
            child: _FeaturedItemCard(property: properties[i]),
          ),
        ),
      ),
    ),
    separatorBuilder: (context, index) => AppSpace.h20,
  );
 }

 Future<void> remove(Property property, BuildContext context) async {
```

```
    {
      await context.read<FavoriteCubit>().remove(property);
      // Future.delayed(const Duration(milliseconds: 1000), () async {
      //   AppSnackBar(
      //     context: context,
      //     isError: false,
      //     title: 'The Property was successfully removed!',
      //   ).showError;
      // });
    }
  }
}

class _FeaturedItemCard extends StatelessWidget {
  const _FeaturedItemCard({required this.property});
  final Property property;
  void goToPage(BuildContext context) {
    Navigator.pushNamed(context, RoutesName.details, arguments: property);
  }

  @override
  Widget build(BuildContext context) {
    final theme = Theme.of(context);
    return ShrinkButton(
      onPressed: () => goToPage(context),
      child: Container(
        padding: EdgeInsets.all(10.w),
        decoration: BoxDecoration(
          color: theme.cardColor,
          borderRadius: BorderRadius.circular(25.0),
        ),
        child: Row(
          crossAxisAlignment: CrossAxisAlignment.start,
          children: [
            _FeaturedItemCardLeftSide(property: property),
            AppSpace.w10,
            _FeaturedItemCardRightSide(property: property)
          ],
        ),
      ),
    );
  }
}

class _FeaturedItemCardRightSide extends StatelessWidget {
  const _FeaturedItemCardRightSide({required this.property});

  final Property property;
```

```
@override
Widget build(BuildContext context) {
  final theme = Theme.of(context);
  return Expanded(
    child: Column(
      crossAxisAlignment: CrossAxisAlignment.start,
      mainAxisAlignment: MainAxisAlignment.spaceAround,
      children: [
        Text(
          property.area!,
          maxLines: 2,
          style: TextStyle(
            color: theme.text,
            fontWeight: FontWeight.bold,
          ),
        ),
        Row(
          children: [
            SvgPicture.asset(
              AssetPath.starFill,
              width: 18.sp,
              colorFilter: const ColorFilter.mode(
                AppColor.orange,
                BlendMode.srcIn,
              ),
            ),
            AppSpace.w5,
            Text(
              property.reviewCount.toString(),
              style: TextStyle(
                color: theme.text,
                fontWeight: FontWeight.bold,
              ),
            ),
          ],
        ),
        Row(
          children: [
            SvgPicture.asset(AssetPath.placeFill, width: 12.sp),
            AppSpace.w5,
            Text(
              property.city!,
              style: TextStyle(
                color: theme.text,
                fontSize: 12.sp,
                fontWeight: FontWeight.w300,
              ),
```

```
            ),
          ],
        ),
        RichText(
          text: TextSpan(
            style: TextStyle(
              color: theme.text,
              fontSize: 16.sp,
              fontWeight: FontWeight.bold,
            ),
            children: [
              TextSpan(text: '${property.prices![0].currency} '),
              TextSpan(text: property.prices![0].amountMin.toString()),
              TextSpan(
                text: '/${property.prices![0].isSale}',
                style: TextStyle(
                  fontSize: 12.sp,
                  fontWeight: FontWeight.w300,
                ),
              ),
            ],
          ),
        ),
      ],
    ),
  );
 }
}

class _FeaturedItemCardLeftSide extends StatelessWidget {
 const _FeaturedItemCardLeftSide({required this.property});

 final Property property;

 @override
 Widget build(BuildContext context) {
  return Expanded(
    child: Stack(
      children: [
        HttpsImage(
          placeholder: Container(
            decoration: BoxDecoration(
              color: AppColor.black,
              borderRadius: BorderRadius.circular(15.0),
            ),
          ),
          imageUrl: property.image!,
          imageBuilder: (context, img) {
```

```
            return Container(
              decoration: BoxDecoration(
                color: AppColor.softGrey,
                borderRadius: BorderRadius.circular(15.0),
                image: DecorationImage(image: img, fit: BoxFit.cover),
              ),
            );
          },
        ),
        Positioned(
          top: 5.w,
          left: 5.w,
          child: BlocBuilder<FavoriteCubit, FavoriteState>(
            builder: (context, state) {
              final bool hasData = state.properties.contains(property);
              return CircleButton(
                onPressed: () => context.read<FavoriteCubit>().add(property),
                icon: hasData ? AssetPath.hartFill : AssetPath.hart,
                radius: 30.sp,
                color: hasData ? AppColor.white : AppColor.radial,
                bgColor: hasData ? AppColor.button : AppColor.softGrey,
              );
            },
          ),
        ),
        Positioned(
          bottom: 5.w,
          left: 5.w,
          child: Container(
            height: 25.w,
            padding: AppPadding.h20,
            decoration: BoxDecoration(
              color: AppColor.main,
              borderRadius: BorderRadius.circular(8.0),
            ),
            child: Center(
              child: Text(
                property.propertyType!,
                maxLines: 1,
                style: TextStyle(color: AppColor.white, fontSize: 10.sp),
              ),
            ),
          ),
        )
      ],
    ),
  );
}
```

```
}

class _FoundCountAndListType extends StatelessWidget {
  const _FoundCountAndListType();

  @override
  Widget build(BuildContext context) {
    final theme = Theme.of(context);
    return BlocBuilder<FavoriteCubit, FavoriteState>(
      builder: (context, state) {
        if (state.properties.isNotEmpty) {
          return Row(
            mainAxisAlignment: MainAxisAlignment.spaceBetween,
            children: [
              Text(
                "Found ${state.properties.length} estates",
                style: TextStyle(
                  color: theme.text,
                  fontSize: 16.sp,
                  fontWeight: FontWeight.normal,
                ),
              ),
              const _ListShortButtonRow()
            ],
          );
        }
        return const SizedBox();
      },
    );
  }
}

class _ListShortButtonRow extends StatelessWidget {
  const _ListShortButtonRow();

  @override
  Widget build(BuildContext context) {
    final theme = Theme.of(context);
    return Container(
      padding: AppPadding.all5,
      decoration: BoxDecoration(
        color: theme.cardColor,
        borderRadius: BorderRadius.circular(50.0),
      ),
      child: Row(
        children: [
          const _GridListShortButton(),
          AppSpace.w20,
```

```
        const _ListShortButton(),
      ],
    ),
  );
 }
}

class _GridListShortButton extends StatelessWidget {
 const _GridListShortButton();

 @override
 Widget build(BuildContext context) {
  final theme = Theme.of(context);
  return BlocBuilder<ListTypeCubit, ListTypeState>(
    builder: (context, state) {
      return ShrinkButton(
        onPressed: () => context.read<ListTypeCubit>().challenge(),
        child: AnimatedContainer(
          duration: const Duration(milliseconds: 1000),
          padding: AppPadding.all5,
          decoration: BoxDecoration(
            color: state.isGrid ? theme.scaffoldBackgroundColor : null,
            borderRadius: BorderRadius.circular(50.0),
          ),
          child: SvgPicture.asset(
            state.isGrid ? AssetPath.dashboardFill : AssetPath.dashboard,
            colorFilter: ColorFilter.mode(
              state.isGrid ? theme.mainIcon : theme.mainIcon.withOpacity(0.3),
              BlendMode.srcIn,
            ),
          ),
        ),
      );
    },
  );
 }
}

class _ListShortButton extends StatelessWidget {
 const _ListShortButton();

 @override
 Widget build(BuildContext context) {
  final theme = Theme.of(context);
  return BlocBuilder<ListTypeCubit, ListTypeState>(
    builder: (context, state) {
      return ShrinkButton(
        onPressed: () => context.read<ListTypeCubit>().challenge(),
```

```
        child: AnimatedContainer(
          duration: const Duration(milliseconds: 1000),
          padding: AppPadding.all5,
          decoration: BoxDecoration(
            color: state.isGrid ? null : theme.scaffoldBackgroundColor,
            borderRadius: BorderRadius.circular(50.0),
          ),
          child: SvgPicture.asset(
            state.isGrid ? AssetPath.viewDay : AssetPath.viewDayFill,
            colorFilter: ColorFilter.mode(
              state.isGrid ? theme.mainIcon.withOpacity(0.3) : theme.mainIcon,
              BlendMode.srcIn,
            ),
          ),
        ),
      );
    },
  );
}
}

class _GridViewStyle extends StatelessWidget {
  const _GridViewStyle({required this.properties});

  final List<Property> properties;

  @override
  Widget build(BuildContext context) {
    return GridView.count(
      crossAxisCount: 2,
      padding: AppPadding.v5.copyWith(bottom: 50.w),
      physics: const BouncingScrollPhysics(),
      childAspectRatio: 0.6,
      crossAxisSpacing: 10.w,
      mainAxisSpacing: 20.w,
      // shrinkWrap: true,
      children: properties.map((e) => _FavoriteItemCard(property: e)).toList(),
    );
  }
}

class _FavoriteItemCard extends StatelessWidget {
  const _FavoriteItemCard({required this.property});
  final Property property;
  void goToPage(BuildContext context) {
    Navigator.pushNamed(context, RoutesName.details, arguments: property);
  }
```

```
  @override
  Widget build(BuildContext context) {
    return TranslateUpAnimation(
      duration: const Duration(milliseconds: 2500),
      child: OpacityTranslateAnimation(
        duration: const Duration(milliseconds: 4500),
        child: ShrinkButton(
          onPressed: () => goToPage(context),
          child: Container(
            padding: EdgeInsets.all(10.w),
            decoration: BoxDecoration(
              color: AppColor.softGrey,
              borderRadius: BorderRadius.circular(25.0),
            ),
            child: Column(
              children: [
                _FavoriteItemCardTop(property: property),
                AppSpace.h10,
                _FavoriteItemCardBottom(property: property)
              ],
            ),
          ),
        ),
      ),
    );
  }
}

class _FavoriteItemCardBottom extends StatelessWidget {
  const _FavoriteItemCardBottom({required this.property});

  final Property property;

  @override
  Widget build(BuildContext context) {
    return Expanded(
      child: Column(
        mainAxisAlignment: MainAxisAlignment.spaceAround,
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
          Text(
            property.area!,
            maxLines: 1,
            style: TextStyle(
              color: AppColor.darkText,
              fontSize: 12.sp,
              fontWeight: FontWeight.bold,
            ),
```

```
        ),
        Row(
          children: [
            SvgPicture.asset(
              AssetPath.starFill,
              width: 12.sp,
              colorFilter: const ColorFilter.mode(
                AppColor.orange,
                BlendMode.srcIn,
              ),
            ),
            AppSpace.w5,
            Text(
              property.reviewCount.toString(),
              style: const TextStyle(
                color: AppColor.darkText,
                fontWeight: FontWeight.bold,
              ),
            ),
            AppSpace.w10,
            Expanded(
              child: Row(
                children: [
                  SvgPicture.asset(AssetPath.placeFill, width: 12.sp),
                  AppSpace.w5,
                  Expanded(
                    child: Text(
                      property.city!,
                      maxLines: 1,
                      style: TextStyle(
                        color: AppColor.darkText,
                        fontSize: 10.sp,
                        fontWeight: FontWeight.w300,
                      ),
                    ),
                  ),
                ],
              ),
            ),
          ],
        ),
      ],
    ),
  );
}
}

class _FavoriteItemCardTop extends StatelessWidget {
```

```
const _FavoriteItemCardTop({required this.property});

  final Property property;

  @override
  Widget build(BuildContext context) {
    return Expanded(
      flex: 4,
      child: Stack(
        children: [
          HttpsImage(
            placeholder: Container(
              decoration: BoxDecoration(
                color: AppColor.black,
                borderRadius: BorderRadius.circular(15.0),
              ),
            ),
            imageUrl: property.image!,
            imageBuilder: (context, img) {
              return Container(
                decoration: BoxDecoration(
                  color: AppColor.softGrey,
                  borderRadius: BorderRadius.circular(15.0),
                  image: DecorationImage(image: img, fit: BoxFit.cover),
                ),
              );
            },
          ),
          Positioned(
            top: 10.w,
            right: 10.w,
            child: BlocBuilder<FavoriteCubit, FavoriteState>(
              builder: (context, state) {
                final bool hasData = state.properties.contains(property);
                return CircleButton(
                  onPressed: () => context.read<FavoriteCubit>().add(property),
                  icon: hasData ? AssetPath.hartFill : AssetPath.hart,
                  radius: 30.sp,
                  color: hasData ? AppColor.white : AppColor.radial,
                  bgColor: hasData ? AppColor.button : AppColor.softGrey,
                );
              },
            ),
          ),
          Positioned(
            bottom: 5.w,
            right: 5.w,
```

```
        child: Container(
            height: 35.w,
            padding: EdgeInsets.symmetric(horizontal: 10.w),
            decoration: BoxDecoration(
              color: AppColor.main,
              borderRadius: BorderRadius.circular(8.0),
            ),
            child: Center(
              child: RichText(
                text: TextSpan(
                  style: TextStyle(
                    color: AppColor.white,
                    fontSize: 14.sp,
                    fontWeight: FontWeight.bold,
                  ),
                  children: [
                    TextSpan(text: '${property.prices![0].currency} '),
                    TextSpan(text: property.prices![0].amountMin.toString()),
                    TextSpan(
                      text: '/${property.prices![0].isSale}',
                      style: TextStyle(
                        fontSize: 10.sp,
                        color: AppColor.white,
                        fontWeight: FontWeight.w300,
                      ),
                    ),
                  ],
                ),
              ),
            ),
          ),
        )
      ],
    ),
  );
}
}

class _FavoriteItemsEmpty extends StatelessWidget {
  const _FavoriteItemsEmpty();

  @override
  Widget build(BuildContext context) {
    final theme = Theme.of(context);
    return Expanded(
      child: Padding(
        padding: AppPadding.main,
```

```
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            ShrinkButton(
              onPressed: () => context.read<MainPageCubit>().click(0),
              child: SvgPicture.asset(AssetPath.dialogAdd),
            ),
            AppSpace.h20,
            RichText(
              textAlign: TextAlign.center,
              text: TextSpan(
                children: [
                  TextSpan(
                    text: "Your favorite page is\n",
                    style: TextStyle(
                      color: theme.mainText,
                      fontSize: 22.sp,
                      fontWeight: FontWeight.normal,
                    ),
                  ),
                  TextSpan(
                    text: "empty",
                    style: TextStyle(
                      color: theme.mainText,
                      fontSize: 22.sp,
                      fontWeight: FontWeight.bold,
                    ),
                  ),
                ],
              ),
            ),
            AppSpace.h10,
            Text(
              AppString.emptyFavorite,
              textAlign: TextAlign.center,
              style: TextStyle(
                color: theme.mainText,
                fontSize: 12.sp,
                fontWeight: FontWeight.w300,
              ),
            ),
          ],
        ),
      ),
    );
  }
}
```

11.Settings screen

```
class SettingsPage extends StatelessWidget {
  const SettingsPage({super.key});

  @override
  Widget build(BuildContext context) {
    return const OpacityTranslateAnimation(
      child: SafeArea(
        child: Column(
          children: [
            _SettingsAppBar(),
            _SettingsPageContent(),
          ],
        ),
      ),
    );
  }
}

class _SettingsAppBar extends StatelessWidget {
  const _SettingsAppBar();

  @override
  Widget build(BuildContext context) {
    final theme = Theme.of(context);
    return SizedBox(
      height: kToolbarHeight,
      child: Padding(
        padding: AppPadding.h20,
        child: AppBar(
          backgroundColor: AppColor.transparent,
          automaticallyImplyLeading: false,
          title: Text(
            AppLn.of(context)!.settings,
            style: TextStyle(
              color: theme.text,
              fontSize: 16.sp,
              fontWeight: FontWeight.w500,
            ),
          ),
        ),
      ),
    );
  }
}

class _SettingsPageContent extends StatelessWidget {
  const _SettingsPageContent();
```

```
@override
Widget build(BuildContext context) {
  return SingleChildScrollView(
    padding: AppPadding.h20,
    child: Column(
      children: [
        AppSpace.h50,
        _SettingsTileWithSwitch(title: AppLn.of(context)!.theme),
        _SettingsTile(
          title: AppLn.of(context)!.language,
          image: AssetPath.language,
          onPressed: () => language(context),
        ),
        _SettingsTile(
          title: AppLn.of(context)!.myAddress,
          image: AssetPath.place,
          onPressed: () {},
        ),
        _SettingsTile(
          title: AppLn.of(context)!.mailToUs,
          image: AssetPath.mail,
          onPressed: sentmail,
        ),
        _SettingsTile(
          title: AppLn.of(context)!.support,
          image: AssetPath.phone,
          onPressed: () {},
        ),
        _SettingsTile(
          title: AppLn.of(context)!.logout,
          image: AssetPath.logout,
          onPressed: () {
            context.read<AppBloc>().add(const AppLogoutRequested());
            Navigator.pushNamedAndRemoveUntil(
              context,
              RoutesName.initial,
              (route) => false,
            );
          },
        ),
      ],
    ),
  );
}

Future<void> language(context) {
  return showModalBottomSheet(
```

```
      context: context,
      backgroundColor: AppColor.transparent,
      builder: (builder) => const LanguageBottomSheet(),
    );
  }

  Future<void> sentmail() async {
    final Email email = Email(
      body: 'Email body',
      subject: 'Email subject',
      recipients: ['sudesh7788@gmail.com'],
      isHTML: false,
    );

    await FlutterEmailSender.send(email);
  }
}

class _SettingsTile extends StatelessWidget {
  const _SettingsTile({
    required this.title,
    required this.image,
    required this.onPressed,
  });
  final String title;
  final String image;
  final Function() onPressed;
  @override
  Widget build(BuildContext context) {
    final theme = Theme.of(context);
    return ShrinkButton(
      onPressed: onPressed,
      child: Container(
        margin: AppPadding.v15,
        padding: AppPadding.h10,
        child: Row(
          children: [
            SvgPicture.asset(
              image,
              width: 20.w,
              colorFilter: ColorFilter.mode(
                theme.mainIcon,
                BlendMode.srcIn,
              ),
            ),
            AppSpace.w10,
            Text(
              title,
```

```
            style: TextStyle(
              fontSize: 14.sp,
              fontWeight: FontWeight.w500,
            ),
          ),
        ],
      ),
    ),
  );
 }
}

class _SettingsTileWithSwitch extends StatelessWidget {
 const _SettingsTileWithSwitch({required this.title});
 final String title;

 @override
 Widget build(BuildContext context) {
  final theme = Theme.of(context);
  return BlocBuilder<ThemeCubit, ThemeState>(
    builder: (context, state) {
      return Container(
        margin: AppPadding.v5,
        padding: AppPadding.h10,
        child: Row(
          children: [
            SvgPicture.asset(
              AssetPath.sun,
              width: 20.w,
              colorFilter: ColorFilter.mode(
                theme.mainIcon,
                BlendMode.srcIn,
              ),
            ),
            AppSpace.w10,
            Expanded(
              child: Text(
                title,
                style: TextStyle(
                  fontSize: 14.sp,
                  fontWeight: FontWeight.w500,
                ),
              ),
            ),
            Switch.adaptive(
              value: state.isDark,
              applyCupertinoTheme: false,
              onChanged: (value) => context.read<ThemeCubit>().change(value),
```

```
            )
          ],
        ),
      );
    },
  );
}
}
```

6. TESTING

6.1 TESTING STRATEGIES

Testing is the process of detecting errors. Testing performs a very critical role for quality assurance and for ensuring the reliability of software. The results of testing are used later even during maintenance also.

6.2 TESTING OBJECTIVES

The main objective of testing is to uncover a host of errors, systematically and with minimum effort and time. Stating formally, we can say,

➢ Testing is a process of executing a program with the intent of finding an error.

➢ A successful test is one that uncovers a yet undiscovered error.

➢ A good test case is one that has a high probability of finding error, if it exists.

➢ The tests are inadequate to detect possibly present errors.

➢ The software more or less confirms the quality and reliable standards.

6.3 LEVELS OF TESTING (test cases)

6.3.1 Unit Testing

Unit testing focuses verification effort on the smallest unit of software i.e., the module.

Scope: Individual units or components (functions, methods, modules) of the system.

Purpose: Verify the correctness of each unit's functionality in isolation.

Examples: Testing login authentication functions, database interactions for user registration, validation of service request creation.

Unit testing done in our project :

1. Splash screen testing:

Simply launch the app and observe if the splash screen appears correctly.

Scenarios:

- Testing if the image or animation displayed correctly.
- Does the text (if any) appear as intended?
- Does the splash screen display for a reasonable amount of time?

  On mobile devices, you can simulate a cold launch by force quitting the app and then relaunching it. This will ensure the splash screen appears, even if the app was running in the background previously.

2. Sign up Screen:

Testing a sign-up screen is similar to testing a sign-in screen, but with an emphasis on validating user input and ensuring a smooth registration process.

Scenarios:

Testing sign-up by creating new user accounts and checking if they are stored correctly in the database.

1.Positive Test - Successful Registration:

Scenario: Test if a user can successfully register with valid credentials.

Test Steps:

- Enter valid email, enter a new password and confirm password.
- Click the sign up button.
- Verify that the user is redirected to the homepage.

Result: User successfully registered and access the homepage.

2. Negative Test - Invalid Credentials:

Scenario: Test sign up with invalid email or weak password.

Test Steps:

- Enter an invalid email or weak password.
- Click the sign up button.
- Verify the error message or prompt.

Ex: Entering details of user sayalishripad@gmail.com and weak password

Result: System displays an error message indicating invalid credentials.

3. Negative Test - Empty Fields:

Scenario: Verify handling of exceptional cases like empty input fields.

Test Steps:

- Keep the email or password field empty.
- Click the sign up button.
- Verify the error message or prompt.

Result: System shows users to fill in both email and password fields.

4. Negative Test - Existing User:

Scenario: Attempt registration with an already existing username or email address.

Test Steps:

Verify the system prevents duplicate accounts and displays informative messages.

3. Sign in Screen:

Scenarios:

- Testing if user credentials (email, password) are validated correctly during login.

Validate successful login with correct credentials and failure with incorrect ones.

1. Positive Test - Successful Login:

Scenario: Test if a user can successfully login with valid credentials.

Test Steps:

- Enter valid email and password.
- Click the login button.
- Verify that the user is redirected to the homepage.

Ex: Entering details of registered user sayalishripad@gmail.com and correct password

Result: User successfully logged in and access the homepage.

2. Negative Test - Invalid Credentials:

Scenario: Test login with invalid email or password.

Test Steps:

- Enter an invalid email or password.
- Click the login button.
- Verify the error message or prompt.

Ex: Entering details of registered user sayalishripad@gmail.com and incorrect password

Result: System displays an error message indicating invalid credentials.

3. Negative Test - Empty Fields:

Scenario: Verify handling of exceptional cases like empty input fields.

Test Steps:

- Keep the email or password field empty.
- Click the login button.
- Verify the error message or prompt.

Result: System shows users to fill in both email and password fields.

4. HomeScreen:

Scenarios:

Load and Display:

1.Positive Test - Successful Page Load:

Scenario: Ensure the home screen loads without errors.

Test Steps:

- Access the homescreen.
- Verify that the home screen loads within an acceptable time frame.
- Check for the presence of essential elements (header, navigation, footer).

Result: Home screen loads without errors, displaying essential elements.

Navigation:

1.Positive Test - Navigation Links:

Scenario: Test functionality of navigation links on the home screen.

Test Steps:

- Click on navigation links (categories).
- Verify redirection to the respective pages without errors.

Result: Navigation links direct users to the correct pages without any issues.

2.Negative Test - Broken Links:

Scenario: Test for broken or incorrect navigation links.

Test Steps:

- Attempt to access non-existent or broken URLs through navigation links.
- Verify the behavior of the system.

Result: System should handle non-existent or broken links gracefully (e.g., display a 404 error page).

5. Profile Screen:

Testing a profile screen involves ensuring it accurately displays user information, allows for editing where appropriate, and maintains user privacy.

Scenarios:

User Profile Display:

1.Positive Test - Profile Information Display:

Scenario: Ensure that user profile information displays correctly.

Test Steps:

- Access a user's profile page.
- Verify that user details (name, email, etc.) are displayed accurately.

Result: User profile information is correctly displayed without any discrepancies.

2.Negative Test - Unauthorized Access:

Scenario: Test unauthorized access to user profiles.

Test Steps:

- Attempt to access another user's profile without appropriate permissions.
- Verify system behavior or error message.

Result: System should restrict unauthorized access and display an appropriate error message.

6. Search Screen:

Testing a search screen is vital for ensuring users can find the information they need effectively.

Scenarios:

1.Positive Test: Display relevant search information.

Test Steps:

- Test with multiple words or phrases searches .
- Verify the system returns relevant results.

Result: Search screen allows filtering by categories, dates, or other criteria, test these filters thoroughly and display relevant information.

2.Negative Test: Display irrelevant information.

Test Steps:

- Test with simple, single-word searches .
- Verify the system returns relevant results.

Result: System should restrict displaying irrelevant data.

7. Favourite Screen:

A "favourite screen" can refer to a few different functionalities depending on the specific app or platform you're testing.

Scenarios:

1.Positive Test: Favouriting properties.

Test Steps:

- Test the functionality of favouriting properties.
- Verify users can successfully add properties to their favourites list from different sections of the app.

Result: Favourited properties will be displayed on Favourite Screen.

1.Positive Test: Unfavoriting properties.

Test Steps:

- Test the functionality of unfavoriting properties.
- Verify users can successfully remove properties from their favourites list from different sections of the app.

Result: Unfavorited properties will be removed on Favourite Screen.

6.3.2 Integration Testing:

After the unit testing, we have to perform integration testing. The goal here is to see if modules can be integrated properly, the emphasis being on testing interfaces between modules.

Scope: Interaction between integrated units or components.

Purpose: Verify the interaction and data flow between units to ensure they work together seamlessly.

Examples: Testing the integration between user authentication and user profile creation, verifying interactions between property listings and requests.

Integration testing done in our project :

1. Login and Registration Integration Testing:

Scenario:

- Test the interaction between the login and registration components with the database and session management.

2. Homepage Integration Testing:

Scenario:

- Verify interactions between the homepage components and the backend services fetching dynamic content or featured sections.

3. Property Listing and Service Request Integration Testing:

Scenario:

- Test the interaction between the property listing component and the buyer request feature.
- Ensure buyer requests are correctly linked and sent to the respective property owner.

4. User Profile and Service Listing Integration Testing:

Scenario:

- Verify the interaction between the user profile and service listing components.
- Test if the service requests listed in a user's profile correspond to the requests sent by the user.

Sample Integration Test Scenarios:

Login and Profile Update:

- Test if user profile details are updated after successful login or registration.

Buyer Request and Business Interaction:

- Verify buyer requests are correctly linked and sent to the respective property owner.

6.3.3 System Testing

System testing involves examining the entire software application to ensure it meets the project requirements.Both unit testing and integration testing mentioned above comprise in system testing.

6.3.4 White Box Testing

This is a unit testing method where a unit will be taken at a time and tested thoroughly at a statement level to find the maximum possible errors. Test has been done step wise for every piece of code, taking care that every statement in the code is executed at least once. The white box testing is also called Glass Box Testing.

- **Backend Functionality Testing -** User Authentication:

Objective: Validate the backend code handling user login/authentication.

Test different scenarios (valid credentials, invalid credentials, null inputs) to ensure all code paths are executed, covering conditions, loops, and error-handling statements.

- **Database Interaction** - Data Validation:

Objective: Verify how data validation is performed before storing information in the database.

Directly assess the backend code responsible for data validation, ensuring that all validation rules are rigorously tested.

- **Error Handling** - Frontend and Backend Integration:

Objective: Assess error-handling mechanisms in the frontend and backend integration.

Force errors by providing incorrect inputs or manipulating data to observe how the system responds and handles exceptions at code levels.

By applying white box testing methodologies like statement-level testing, code coverage assessments, and thorough validation of internal code pathways, the aim is to expose errors and vulnerabilities within the system, ensuring robustness and reliability in the codebase.

6.3.5 Black Box Testing

This testing method considers a module as a single unit and checks the unit at interface and communication with other modules rather getting into details at statement level. Here the module will be treated as a block box that will take one input and generate output.

Black Box Testing Examples:

- **User Interface Testing -** Login Page:

Objective: Evaluate the login page functionality without examining the backend code.

Input valid and invalid credentials, observing if correct login grants access and incorrect credentials result in denied access.

- **Business Logic** - Service Request Submission:

Objective: Test the service request submission process without examining the internal codebase.

Submit various types of service requests (different properties, details) to ensure proper acceptance and processing without knowing how the backend handles the requests.

- **Integration Testing -** Frontend and Backend Interaction:

Objective: Validate the interaction between frontend and backend systems.

Submit requests through the frontend interface and verify if corresponding actions are reflected in the user profile or database without inspecting the internal communication mechanisms.

- **Database Interaction** - Data Retrieval:

Objective: Check the correctness of displayed data without inspecting database handling code.

Validate if information retrieved from the database (like user profiles or property listings) corresponds accurately to the expected data displayed on the frontend.

- **Input Validation** - Error Handling

Objective: Test how the system handles incorrect inputs without examining the backend code.

Submit erroneous or incomplete data and observe how the system responds, ensuring it handles errors gracefully without revealing internal workings.

Black box testing focuses on the system's external behavior, testing functionality, inputs, and outputs without having visibility into the internal structure or implementation. These tests aim to ensure that the system functions as expected from a user's perspective, irrespective of the underlying code complexities.

7. OUTPUT SCREENS

7.1 SPLASH AND INTRODUCTION SCREENS

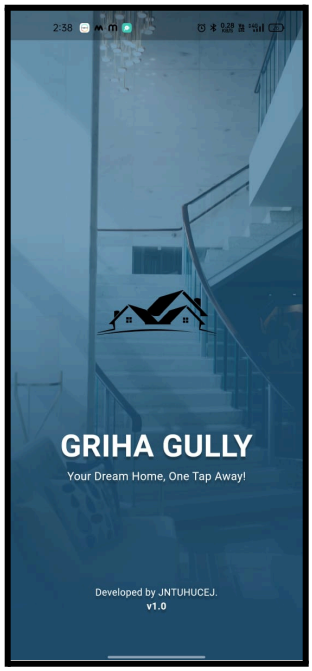


Fig 7.1 Splash screen

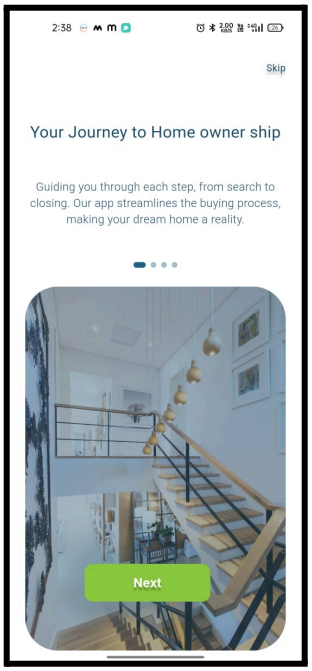
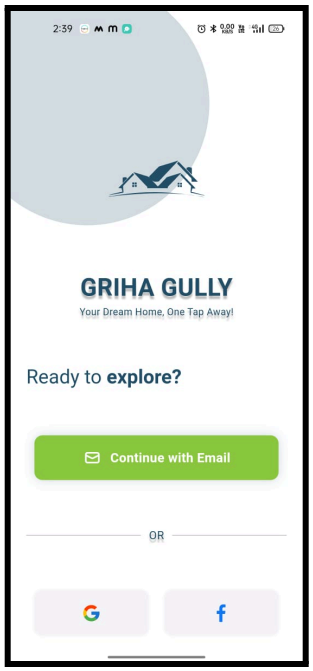


Fig 7.2 Introduction screen

7.2 SIGN IN SCREEN



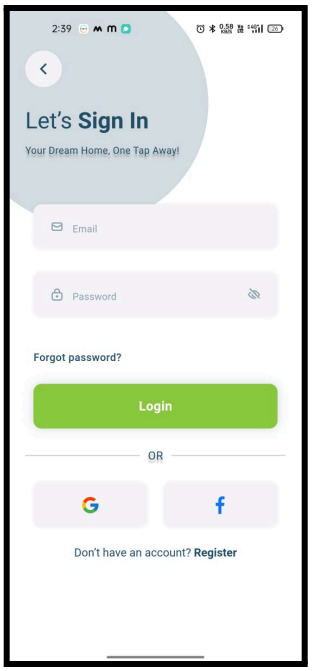


Fig 7.3 Sign in Screen

7.3 REGISTER PAGE

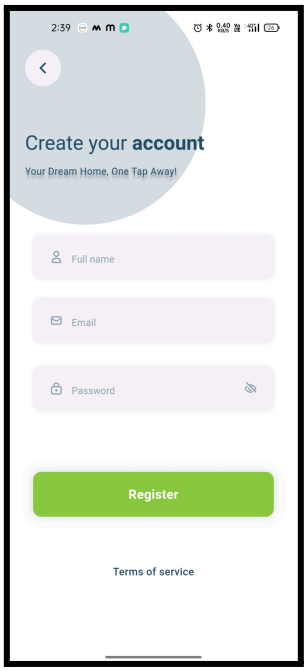


Fig 7.4 Register Screen

On the registration page, new users are required to furnish their name, email details, and create a robust password for their account. A valid email ID must be provided during this process.

7.4 AUTHENTICATION

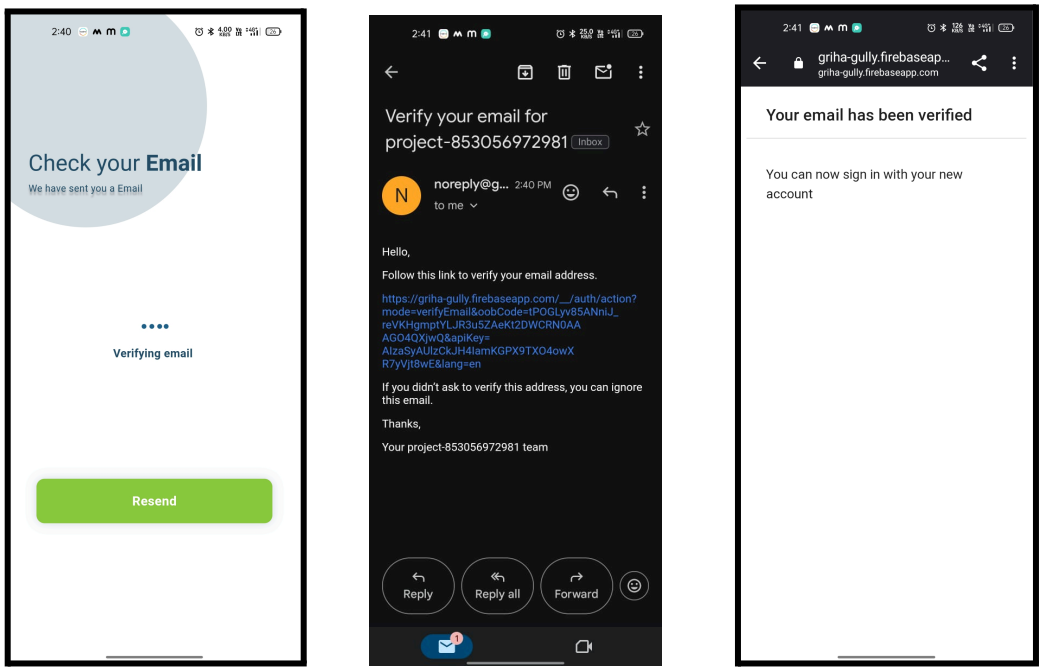


Fig 7.4 Authentication screens

7.5 SEARCH

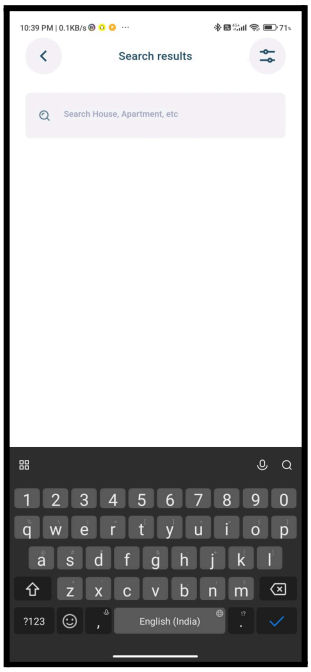


Fig 7.5 Search screen

7.6 SETTINGS

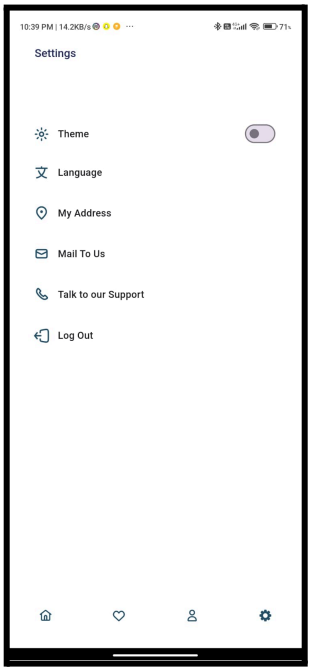


Fig 7.6 Settings screen

7.7 PROFILE

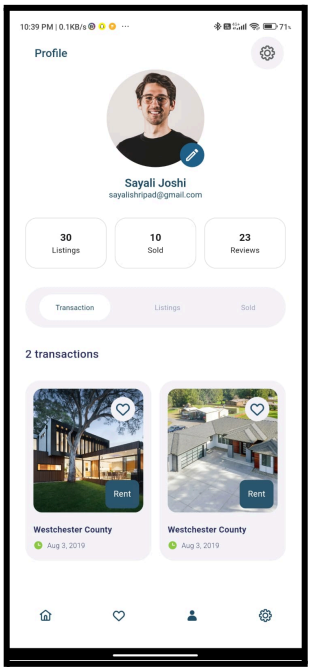


Fig 7.7 Profile screen

7.8 HOME

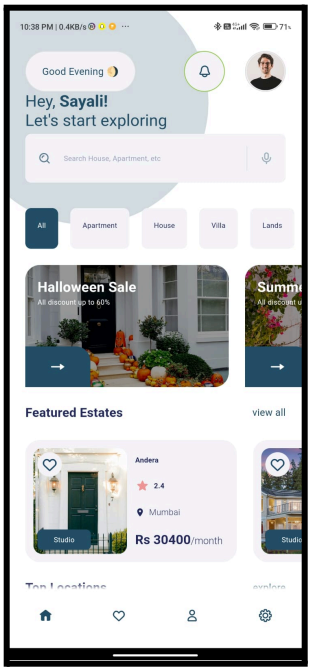


Fig 7.8 Home screen

7.9 FAVORITES

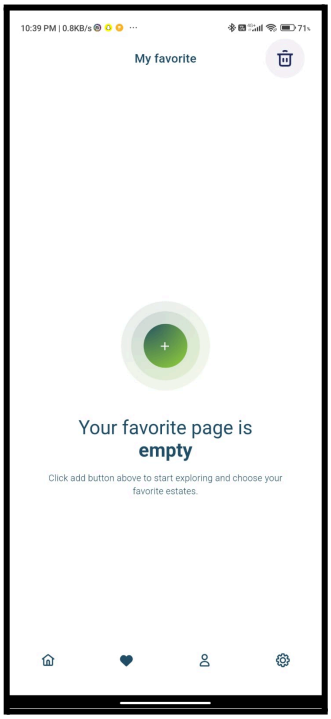


Fig 7.9 Favorite screen

8.CONCLUSION

- The creation of Griha Gully signifies a profound commitment to reshaping the landscape of the real estate industry. Recognizing the inefficiencies and frustrations that users often encounter on traditional real estate platforms, the development team behind Griha Gully embarked on a mission to revolutionize the property search experience.
- Griha Gully prioritizes transparency and trust by ensuring that property listings are comprehensive and up-to-date, empowering users to make informed decisions. Detailed property descriptions, high-quality images, and interactive map views provide users with a holistic understanding of each listing, enabling them to visualize themselves in the space.

- In conclusion, Griha Gully is more than just a real estate app; it's a testament to innovation, dedication, and a relentless pursuit of excellence. By redefining the real estate search experience and prioritizing user satisfaction, Griha Gully aims to not only meet but exceed the expectations of users, contributing to the overall growth and evolution of the real estate industry.

9. FUTURE ENHANCEMENTS

- **Social Sharing and Collaboration**: Introducing features that enable users to share property listings with friends, family, or advisors and collaborate on property searches together, fostering a sense of community and facilitating decision-making.
- **Enhanced AI-Powered Recommendations**: Continuously refining and expanding the app's AI algorithms to provide even more accurate and personalized property recommendations based on user preferences, behavior, and feedback.
- **Neighborhood Insights and Analytics**: Providing users with comprehensive insights and analytics about neighborhoods, including demographics, crime rates, schools, amenities, and future development plans, to help them make more informed decisions about where to live or invest.
- **Enhanced Customer Support**: Expanding and improving customer support services, to ensure users receive prompt assistance and resolution to their inquiries or issues.

10. REFERENCES

[1] Software Engineering A practitioner's Approach, Roger S Pressman, 6th edition. McGrawHill InternationalEdition.

[2] Building Cross-Platform Apps with Flutter and Dart: Build scalable apps for Android, iOS, and web from a single codebase,Deven Joshi

[3] The Definitive Guide to Firebase: Build Android Apps on Google's Mobile Platform,Lawrence Moroney

[4] Analysis and design for a Real Estate Inc. Agent System by using security control. Yi Mar Myint*, Khet Khet khaing Oo **

[5] REAL ESTATE MANAGEMENT SYSTEM
Akshay Patil*1, Prathamesh Patil*2, Tejas Patil*3, Farhan Mujawar*4, Prof. Aparna Kamate*5

APPENDICES:

1. Accurately matched property options / Personalized recommendations: Properties that closely align with a user's specific needs and preferences.

2. Algorithm: A set of instructions that a computer follows to perform a specific task. In Griha Gully's case, the algorithms analyze user behavior.

3. API (Application Programming Interface): A set of rules and protocols that allow different software applications to communicate with each other.

4. Backend development: The server-side development of a mobile application.

5. Black Box Testing: A software testing method that examines the functionality of a system without considering its internal structure.

6. Business Logic: The rules and processes that govern how data is processed within an application.

7. Business layer: The server-side components that power a mobile application.

8. Class Diagram: A UML diagram that shows the classes, interfaces, and their relationships in a software system.

9. Cloud Firestore: A NoSQL database for mobile, web, and server development that scales automatically with your needs.

10. Cloud Functions: Serverless functions for running code in response to events triggered by Firebase products or HTTP requests.

11. Cloud Storage: A secure, scalable object storage service for Firebase apps and web applications.

12. Collaboration (Future Enhancements): Working together on a project or task.

13. Community (Future Enhancements): A group of people living in the same area or sharing a common interest.

14. Comprehensive property listings: Listings that provide a complete and detailed overview of available properties.

15. Continuous Learning: The ability of a system to improve its performance over time by learning from new data. Griha Gully's ML algorithms are continuously learning to refine their recommendations.

16. CRUD Operations: Create, Read, Update, and Delete. These are the fundamental operations used to manage data in a software system.

17. Customer Support (Future Enhancements): The department or team that provides assistance to customers with their inquiries or issues.

18. Data Integrity: Ensuring that data within the application is consistent and accurate throughout.

19. Data Layer: The part of a software application that is responsible for storing, retrieving, and managing data.

20. Data Management: The creation and management of the systems that store and retrieve data for an application.

21. Database integration: The process of connecting a database to the frontend and backend of an application.

22. Database Interaction: The process of interacting with a database, such as reading, writing, or updating data.

23. Decision-making (Future Enhancements): The process of choosing between different options.

24. Disconnect: A lack of connection or understanding between two things. In this case, there's a disconnect between what users want and what the platform offers.

25. Driving desire: A strong motivation or urge. The creators of Griha Gully have a strong desire to improve the user experience.

26. Ease-of-use: The characteristic of being easy to use.

27. Empower: To equip someone with the knowledge or confidence to do something. Griha Gully aims to empower users with the tools they need.

28. Established real estate platforms: Well-known and reputable companies in the online real estate market.

29. Feedback (Future Enhancements): Information provided in response to an action, event, or product.

30. Frontend development: The client-side development of a mobile application, which is what users interact with directly.

31. Frustration: A feeling of annoyance or dissatisfaction caused by difficulty or by something that prevents someone from achieving something. Users experience frustration due to irrelevant suggestions.

32. Functional Requirements: Requirements that specify the functions that a system must be capable of performing.

33. Guarantee High-Quality Recommendations: Ensure that the property suggestions are consistently relevant and accurate.

34. Holistic (Griha Gully): Considering something as a whole.

35. In-dependency: The ability to function without reliance on other things.

36. Inefficiencies (Griha Gully): Ways in which a system or process is not working effectively.

37. Insights (Future Enhancements): Information that helps you understand a situation better.

38. Interactive: Involving two-way communication.

39. Intermediaries: Third parties who act as a link between buyers and sellers. In this case, real estate brokers.

40. Machine Learning (ML): A type of artificial intelligence that allows computers to learn without being explicitly programmed. Griha Gully uses ML to analyze user behavior and offer personalized property suggestions.

41. Non-Functional Requirements: Requirements that describe the overall quality attributes of the system, such as performance, security, and usability.

42. Optimize Property Search Journey: Make the process of finding a property faster and easier for users by using technology.

43. Performance Monitoring: Monitoring and measuring the performance of a web or mobile application.

44. Personalized: Tailored to the specific needs or preferences of an individual. Griha Gully personalizes property suggestions based on user behavior.

45. Personalized recommendations: Suggestions tailored to a specific user's needs or preferences. This is a key feature of Griha Gully.

46. Prolonged search times: The time it takes a user to find what they're looking for is extended due to irrelevant suggestions.

47. Property listings: Advertisements for properties that are available for sale or rent.

48. Real estate platform: An online platform that facilitates buying, selling, and renting of properties.

49. Recommendations (Future Enhancements): A suggestion about something that someone should do.

50. Registration (Login Page): The process of registering as a user on a website or application.

51. Remote Configuration: Dynamically change the behavior and appearance of your app without requiring a code push.

52. Responsive: Reacting quickly and positively.

53. Responsive Design: The ability of a user interface to adapt and scale appropriately across different devices and orientations.

54. Revolutionize: To make a radical and beneficial change. Griha Gully hopes to revolutionize the real estate industry.

55. Sample Integration Test Scenarios: Examples of test cases used to verify the interaction between different parts of a system.

56. Scalability: The ability of a system to handle increasing amounts of data or users without compromising performance.

57. Scalability of machine learning: The ability of Griha Gully's system to handle increasing amounts of data without compromising the quality of its recommendations.

58. Scope: The area or functionality that is being tested.

59. Seamless communication channels: Methods for users and relevant parties (landlords, agents) to communicate easily and efficiently.

60. Search Experience (Griha Gully): The way a user finds information on a website or application.

61. Sharing (Future Enhancements): The act of giving or showing something to others.

62. Single Codebase: The ability to write code once and use it to create applications for multiple platforms (e.g., mobile, web, desktop).

63. Social Sharing (Future Enhancements): Sharing information on social media platforms.

64. Software Requirements: Specifications that define what the software needs to do.

65. System Testing: A type of software testing that verifies the entire software application to ensure it meets the project requirements.

66. Technology Stack: The collection of programming languages, software tools, frameworks, libraries, and other technologies used to build and operate a software application.

67. Test Case: A set of inputs, expected outputs, and preconditions that are used to test a specific functionality of a software application.

68. Testing Levels: The different stages or categories of software testing.

69. Transparency: The quality of being open and honest. Griha Gully prioritizes transparency in the property search process.

70. UI Design: The process of designing the visual elements of a user interface.

71. Unit Testing: A type of software testing that focuses on the individual units of code within a system.

72. Up-to-date (Griha Gully): Current and accurate.

73. User Authentication: The process of verifying a user's identity before granting them access to the application.

74. User-centric: Designed with the user's needs and preferences as the top priority.

75. User-centric platform: A platform designed with the user's needs and preferences as the top priority.

76. User experience (UX): The overall experience a user has when interacting with a product or service. The lack of personalization creates a negative user experience.

77. Use Case Diagram: A UML diagram that shows the interaction between a system and its external actors (e.g., users).

78. Wide range: A large variety of something.