SRS

1. Introduction

1.1 Purpose of the SRS

The **Software Requirements Specification (SRS)** document provides a comprehensive description of the software's functionality, features, and requirements for the "Find Your Home" (FYH) mobile application. This document is intended to serve as a guide for developers, testers, and stakeholders involved in the project, ensuring that all system components and behaviors are understood and agreed upon.

This SRS will specify both functional and non-functional requirements, offering clear details on how the application will be developed, tested, and maintained. It will also define the system's constraints, interfaces, and user interactions, making it a key reference throughout the project lifecycle.

1.2 Intended Audience

The intended audience for this document includes:

- **Project Managers**: To understand the scope and technical requirements for planning and resourcing.
- **Developers**: To implement the system based on clear, detailed functional and technical specifications.
- **Testers**: To validate that the software meets the outlined requirements.
- **Stakeholders**: To ensure that the system will meet business needs and user expectations.

1.3 Scope of the Project

The "Find Your Home" app will be a mobile-based platform designed to assist users in finding, listing, and managing properties for rent or sale. The system will allow users to search properties, view details, and perform transactions directly via the mobile app. The app will also integrate machine learning-based price prediction and provide users with additional features like a chatbot and map integration for location tracking.

The mobile app will be available on both Android and iOS platforms, and the system will integrate with cloud-based services for secure data storage, real-time updates, and enhanced performance.

1.4 Definitions, Acronyms, and Abbreviations

- **FYH**: Find Your Home

- **UAT**: User Acceptance Testing
- **API**: Application Programming Interface
- **JWT**: JSON Web Token
- **DBMS**: Database Management System
- **UI**: User Interface
- **UX**: User Experience

---

2. System Requirements

2.1 Functional Requirements

2.1.1 User Registration and Authentication

- **Description**: Users must be able to register, log in, and manage their profiles. The app will support both traditional email/password registration and third-party authentication (e.g., Google or Facebook login).
- **Requirements**:
    - The system should validate user inputs (e.g., email format, password strength).
    - Provide a secure authentication mechanism using JWT tokens.
    - Users should be able to reset their passwords securely.
    - User data (e.g., name, email, password) must be securely stored and encrypted.

2.1.2 Property Management

- **Description**: Users should be able to list, edit, and remove properties from the app.
- **Requirements**:
    - Users should be able to add properties with details such as title, description, price, location, and images.
    - The app should allow users to edit and update their listings as needed.
    - The system should validate the data before it is stored (e.g., price should be a valid number).
    - Users should be able to delete their properties from the app.

- Properties should be associated with a unique ID to avoid duplication.

2.1.3 Property Search and Filters

- **Description**: The app should allow users to search for properties based on various criteria (location, price range, type, etc.).
- **Requirements**:
  - The search functionality should include filters for property type (e.g., apartment, house), price range, location, and size.
  - The app should display search results in a user-friendly layout.
  - Users should be able to sort results by price, newest listings, and other criteria.

2.1.4 Map Integration

- **Description**: The app should integrate with Google Maps to show property locations.
- **Requirements**:
  - Users should be able to view properties on an interactive map.
  - The system should allow users to filter properties based on proximity to their current location.
  - Markers should be displayed on the map for each property, with additional details visible upon clicking the marker.

2.1.5 Price Prediction

- **Description**: The system should provide price predictions for properties based on historical data and machine learning algorithms.
- **Requirements**:
  - The app should collect data from similar properties (e.g., size, location, amenities) to generate predictions.
  - The app should display predicted prices in a clear, user-friendly manner.
  - Users should be able to see how accurate the predictions are based on historical data.

2.1.6 Chatbot Integration

- **Description**: The app should include a chatbot to assist users in finding properties and answering common questions.

- **Requirements**:
    - The chatbot should be powered by AI or a conversational platform (e.g., Google Dialogflow).
    - Users should be able to ask the chatbot about property details, prices, and availability.
    - The chatbot should provide helpful responses, and if necessary, escalate to human support.

2.2 Non-Functional Requirements

2.2.1 Performance

- **Description**: The system must be responsive and capable of handling a large number of concurrent users without significant delays.
- **Requirements**:
    - The app should load property search results within 3 seconds.
    - The system should support up to 10,000 concurrent users.
    - Property listings and images should load efficiently, even with slow internet connections.

2.2.2 Security

- **Description**: The system must ensure the security of user data and interactions.
- **Requirements**:
    - User data, including personal information and payment details, must be encrypted both at rest and in transit.
    - Multi-factor authentication (MFA) should be available for all users.
    - The app should be protected against common vulnerabilities (e.g., SQL injection, cross-site scripting).

2.2.3 Reliability

- **Description**: The system should be highly reliable with minimal downtime.
- **Requirements**:
    - The app should have 99.9% uptime, with backup and disaster recovery mechanisms in place.
    - The system should handle server failures gracefully and ensure that users experience minimal disruptions.

- All critical data should be backed up automatically at regular intervals.

2.2.4 Scalability

- **Description**: The system must be scalable to accommodate increasing numbers of users, properties, and interactions.
- **Requirements**:
  - The app should be designed to scale horizontally (adding more servers as demand increases).
  - The database should be optimized to handle large datasets (e.g., thousands of properties and user records).

---

3. System Architecture

3.1 High-Level Architecture Diagram

- The system will have a **client-server** architecture where the mobile app (client) communicates with the backend (server) for data processing, storage, and retrieval.
  - **Mobile Client (Frontend)**: Developed using React Native to support Android and iOS.
  - **Backend**: Node.js-based server with Express.js to handle API requests.
  - **Database**: Firebase Firestore for real-time data storage and user authentication.
  - **Third-party APIs**: Google Maps API for location-based services, machine learning API for price prediction.

3.2 Database Design

- **Users Table**:
  - Fields: user_id, name, email, password_hash, phone_number, user_type (buyer/seller), etc.
- **Properties Table**:
  - Fields: property_id, title, description, price, location, owner_id (user_id), status (for sale/for rent), etc.
- **Chatbot Interactions Table**:
  - Fields: chat_id, user_id, message, timestamp.

4. Use Cases

4.1 Use Case Diagrams

Use case diagrams visually represent the interactions between users (actors) and the system. Below are the major use cases for the "Find Your Home" app:

1. **User Registration & Login**
    - **Actor**: User (Tenant/Landlord)
    - **Description**: A user must be able to register an account or log in to access the features of the application. This use case includes registration, login, password recovery, and third-party authentication (e.g., Google).
2. **Add Property**
    - **Actor**: Landlord/User
    - **Description**: A registered user can add a property to the system with details like title, price, location, description, and images.
3. **Search Property**
    - **Actor**: Tenant/User
    - **Description**: A user can search for properties based on various filters (e.g., location, price, property type) and view the details of the properties that match the search criteria.
4. **Price Prediction**
    - **Actor**: Tenant/User
    - **Description**: A user can request a price prediction for a property based on historical data and property attributes.
5. **Chatbot Interaction**
    - **Actor**: User
    - **Description**: A user interacts with a chatbot to get assistance, property recommendations, or answers to frequently asked questions.
6. **Map Integration**
    - **Actor**: User
    - **Description**: A user can view the location of properties on an interactive map.

4.2 Use Case Descriptions

Use Case 1: User Registration & Login

- **Actors**: User (Tenant/Landlord)
- **Description**: Users must register or log in to the app to access the system's features. After registration, the user's information will be stored in the database, and they can log in using their credentials.
- **Preconditions**: The user must have a valid email address or account credentials.
- **Main Flow**:
    1. User selects "Sign Up" or "Login."
    2. If "Sign Up," the user provides personal details (name, email, password).
    3. The system validates the data and stores it securely.
    4. User logs in using email/password or third-party authentication (e.g., Google).
    5. The system authenticates the user and grants access.
- **Postconditions**: The user is logged into the system and can access their dashboard.
- **Alternate Flow**: If the user forgets their password, they can reset it through a secure password recovery process.

Use Case 2: Add Property

- **Actors**: Landlord/User
- **Description**: The user (landlord) can add a new property to the system by providing detailed information, including title, description, price, location, and images.
- **Preconditions**: The user must be logged into the system.
- **Main Flow**:
    1. User clicks on "Add Property."
    2. User enters the property details: title, description, price, location (manual entry or Google Maps integration), property images.
    3. The system validates the data.
    4. The property is saved to the database and becomes available for viewing in the property listing section.

- **Postconditions**: The property is successfully listed and visible in the app.
- **Alternate Flow**: If required fields are missing or invalid, the system prompts the user to complete the information.

Use Case 3: Search Property

- **Actors**: Tenant/User
- **Description**: Users can search for properties based on various filters like price, location, type, etc.
- **Preconditions**: The user must be logged into the system.
- **Main Flow**:
    1. User selects the "Search" option.
    2. User enters filters (e.g., price range, property type).
    3. The system processes the filters and displays search results.
    4. User selects a property from the search results to view details.
- **Postconditions**: The user can view detailed information about the property they selected.
- **Alternate Flow**: If no results are found, the system notifies the user and suggests adjusting search criteria.

Use Case 4: Price Prediction

- **Actors**: Tenant/User
- **Description**: The system predicts the price of a property based on historical data and user input (e.g., property size, type).
- **Preconditions**: The user must have selected a property and provided details such as size and location.
- **Main Flow**:
    1. User inputs the property details (size, type, etc.) for prediction.
    2. The system fetches historical data related to similar properties.
    3. The system calculates and displays the predicted price.
- **Postconditions**: The predicted price is displayed to the user.
- **Alternate Flow**: If insufficient data is available for prediction, the system informs the user that predictions cannot be made at the moment.

Use Case 5: Chatbot Interaction

- **Actors**: User
- **Description**: The chatbot assists users with finding properties, answering questions, and providing recommendations.
- **Preconditions**: The user must be logged into the system.
- **Main Flow**:
    1. User clicks on the chatbot icon.
    2. User types a query or request (e.g., “Find apartments in Gulshan”).
    3. The chatbot uses AI to process the request and provide relevant answers or suggestions.
- **Postconditions**: The user receives an answer or recommendation from the chatbot.
- **Alternate Flow**: If the chatbot is unable to answer, it redirects the user to human customer support.

Use Case 6: Map Integration

- **Actors**: User
- **Description**: Users can view properties on an interactive map, making it easier to identify their location relative to the properties.
- **Preconditions**: The user must be logged into the system and have an active internet connection.
- **Main Flow**:
    1. User selects the "Map View" option.
    2. The system loads the map and pins the properties based on their locations.
    3. User can zoom in/out, click on property markers to view detailed information.
- **Postconditions**: The user can view properties on the map and navigate through them.
- **Alternate Flow**: If the map fails to load, the system notifies the user and provides troubleshooting steps.

---

5. User Interface Design

5.1 Wireframes and Mockups

The app will feature an intuitive, user-friendly interface, with the following key screens:

1. **Login/Sign Up Screen**: Simple input fields for email, password, and social media login options (Google, Facebook).
2. **Dashboard**: Once logged in, the user will land on a dashboard where they can view, add, and manage properties. The dashboard will show a list of properties, recent searches, and suggestions.
3. **Property Details**: A detailed view of each property will include images, a map, property features, and a contact button for property owners.
4. **Search Screen**: A search bar with filters (price, location, property type) for easy property discovery.
5. **Map View**: An interactive map where properties are marked for easy location tracking.

5.2 User Flows

1. **Sign-Up Flow**:
    - Start → Enter Email → Enter Password → Validate Email → Store Data → Confirm Registration → End
2. **Property Addition Flow**:
    - Start → Click "Add Property" → Enter Property Details → Validate Data → Save Property → End

---

6. System Architecture

6.1 High-Level Architecture

The "Find Your Home" app will operate on a **client-server architecture** with the following components:

- **Mobile Client (Frontend)**: Developed using **React Native** for cross-platform compatibility (Android and iOS).
- **Backend Server**: Built with **Node.js** and **Express.js** to handle API requests and data processing.
- **Database**: Firebase **Firestore** will be used for real-time data storage and user management.

- **Third-Party APIs**: Integration with **Google Maps API** for location tracking and **machine learning API** for price predictions.

6.2 Database Design

- **Users Table**:
    - user_id, name, email, password_hash, user_type (tenant/landlord), etc.
- **Properties Table**:
    - property_id, title, description, price, location, owner_id (user_id), status (for rent/for sale), etc.
- **Chatbot Interactions Table**:
    - chat_id, user_id, message, timestamp.

7. Testing Requirements

7.1 Test Plans

To ensure that the "Find Your Home" application meets its functional and non-functional requirements, we will implement comprehensive testing across various stages of the development lifecycle. Below are the key types of tests that will be conducted:

1. **Functional Testing**
    - **Objective**: Validate that all features of the application work as expected.
    - **Test Cases**: Each functional requirement listed in the system will have associated test cases. For example:
        - User registration/login: Verify that users can register, log in, and recover their passwords.
        - Property management: Ensure users can add, edit, and delete properties.
        - Property search: Test if the search functionality works with various filters.
    - **Tools**: Manual testing and automated testing using tools like **Jest** for JavaScript-based testing.
2. **Integration Testing**
    - **Objective**: Ensure that different system components (e.g., the front-end, back-end, and third-party APIs) interact correctly.

- **Test Cases**: Test how the mobile app interacts with the backend server and how it integrates with Google Maps and Firebase.
- **Tools**: Postman for API testing, Firebase Emulator for local testing.

3. **System Testing**
   - **Objective**: Test the complete and integrated system to ensure that it meets the defined requirements.
   - **Test Cases**: Test the overall functionality, including property management, search, map view, and price prediction. Check the system's ability to handle expected user loads.
   - **Tools**: Load testing tools like **Apache JMeter** for performance testing.
4. **User Acceptance Testing (UAT)**
   - **Objective**: Validate that the application meets user expectations and requirements.
   - **Test Cases**: Conduct tests with a group of end-users who will interact with the app, providing feedback on usability, functionality, and overall experience.
   - **Tools**: Feedback forms, recorded sessions, and usability testing.
5. **Security Testing**
   - **Objective**: Test the app's security features, including user authentication, data encryption, and protection against common vulnerabilities.
   - **Test Cases**:
     - Test for SQL injection vulnerabilities.
     - Verify that sensitive user data is encrypted.
     - Test multi-factor authentication (MFA) and token security.
   - **Tools**: OWASP ZAP for vulnerability scanning.

7.2 Test Cases

The following are some example test cases:

1. **Test Case: User Registration**
   - **Objective**: Verify that a new user can register with valid email and password.
   - **Preconditions**: User is on the registration page.

- **Actions**:
    1. Enter a valid email and password.
    2. Click the "Register" button.
- **Expected Results**:
    1. User is registered, and a confirmation email is sent.
    2. User is redirected to the login page.
- **Status**: Pass/Fail

2. **Test Case: Add Property**
    - **Objective**: Verify that a user can add a property with valid details.
    - **Preconditions**: User is logged in and on the "Add Property" page.
    - **Actions**:
        1. Enter property title, description, price, and location.
        2. Upload an image for the property.
        3. Click "Add Property."
    - **Expected Results**:
        1. Property is added to the database.
        2. The property is visible in the user's dashboard.
    - **Status**: Pass/Fail
3. **Test Case: Search Property**
    - **Objective**: Verify that a user can search for properties based on filters (e.g., location, price range).
    - **Preconditions**: User is logged in and on the search page.
    - **Actions**:
        1. Enter search filters (e.g., price range, location).
        2. Click the "Search" button.
    - **Expected Results**:
        1. Properties matching the search criteria are displayed.
    - **Status**: Pass/Fail

4. **Test Case: Google Maps Integration**
    - **Objective**: Verify that properties are correctly displayed on the map.
    - **Preconditions**: User is logged in and viewing the property listings.
    - **Actions**:
        1. Click the "Map View" button.
        2. View the properties on the map.
    - **Expected Results**:
        1. Each property is displayed as a marker on the map with correct details.
    - **Status**: Pass/Fail
5. **Test Case: Price Prediction**
    - **Objective**: Verify that the app correctly predicts the price of a property based on provided data.
    - **Preconditions**: User is logged in and on the "Price Prediction" page.
    - **Actions**:
        1. Enter property size and other relevant details.
        2. Click "Predict Price."
    - **Expected Results**:
        1. The predicted price is displayed.
    - **Status**: Pass/Fail

---

8. Deployment and Maintenance

8.1 Deployment Plan

The deployment plan outlines how the app will be deployed to production environments and made available for users:

1. **Pre-Deployment Checklist**:
    - Verify that all tests have passed (functional, integration, security).
    - Set up production databases and ensure that they are secure.

- Ensure that the Google Maps API and Firebase services are properly configured.
- Confirm that the app has been optimized for performance and security.

2. **Deployment Process**:
   - The app will be deployed to **Google Play Store** and **Apple App Store** for public access.
   - The backend server will be deployed to **AWS EC2** or another scalable cloud infrastructure.
   - Continuous integration and continuous deployment (CI/CD) pipelines will be set up using tools like **GitLab CI** or **Jenkins**.
3. **Post-Deployment Monitoring**:
   - Implement **real-time monitoring** of the app's performance using tools like **Datadog** or **Prometheus**.
   - Monitor server health, user activity, and potential errors or crashes.

8.2 Maintenance Plan

1. **Bug Fixes and Updates**:
   - Ensure that bugs and issues identified by users are fixed in a timely manner.
   - Regular updates will be made to the app to improve performance, security, and features based on user feedback.
2. **Performance Optimization**:
   - Continuously monitor the app's performance and make optimizations to improve speed, reduce latency, and enhance the user experience.
3. **Security Updates**:
   - Apply regular security patches and updates to keep the app and server environment secure.
   - Conduct regular security audits to ensure compliance with industry standards.

---

9. Appendix

9.1 Glossary

- **Property ID**: A unique identifier for each property in the system.
- **User Authentication**: The process of verifying a user's identity.
- **Firebase**: A platform used for real-time data storage, user authentication, and notifications.

9.2 References

- **Google Maps API Documentation**: https://developers.google.com/maps/documentation
- **Firebase Documentation**: https://firebase.google.com/docs
- **React Native Documentation**: https://reactnative.dev/docs/getting-started