RoboNav

SE4450: Software Engineering Design
Software Requirements Specification (SRS)

Faculty Advisor: **Dr. Yili (Kelly) Tang**

Email: ytang564@uwo.ca

Team 7:

Bryson Crook
bcrook4@uwo.ca
251217987

Christopher Higgins
chiggi24@uwo.ca
251245390

Mohamed El Dogdog
meldogdo@uwo.ca
251239204

Seth Langendoen
slangend@uwo.ca
251226610

October 20, 2024

GitHub Repo: https://github.com/meldogdo/RoboNav

Revision History

T – Team | C – Chris | M – Mohamed | B – Bryson | S - Seth

Date	Version	Description	Authors
October 20, 2024	0.1	Created Initial Word Document	T
October 21, 2024	0.2	Completed Introduction Section	B
October 22, 2024	0.2	Completed Specific Requirements Section	S, B, C
October 24, 2024	0.3	Completed Use Case Section	M
October 25, 2024	0.3	Completed User Interface Requirements	C
October 26, 2024	0.3	System Requirements	C
October 26, 2024	0.3	Acceptance Criteria	C
October 30, 2024	0.4	Formatted and Proof Read Document	T

Table of Contents

1. Introduction

1.1. Purpose

The purpose of this Software Requirements Specification (SRS) document is to provide a comprehensive overview of the *RoboNav* system, which focuses on enhancing backend API functionalities for autonomous robot control and creating a user-friendly mobile application. The SRS document details the technical and functional requirements needed to achieve these objectives, ensuring a shared understanding among all stakeholders and setting the foundation for subsequent phases of the project, including design, implementation, and testing.

1.2. Scope

The scope of the *RoboNav* project focuses on enhancing the existing backend API to improve its performance, scalability, and functionality, rather than creating a new backend from scratch. Key activities include optimizing the API to support real-time data handling, ensuring seamless communication between the mobile application and autonomous robots, and establishing a cloud infrastructure for secure data storage and access. Additionally, the project involves developing a user-friendly mobile application that enables users to effectively control and monitor robots, featuring real-time status updates and route assignment capabilities. The project will primarily target structured indoor environments, ensuring that the improvements address the specific needs of users interacting with autonomous robots in these settings.

1.3. Definitions, Acronyms and Abbreviations

Below are key definitions, acronyms, and abbreviations frequently referenced throughout this project.

- **Acceptance Criteria:** Conditions that a product must meet to be accepted by a user, customer, or stakeholders, often defined through test cases and expected results.
- **API (Application Programming Interface):** A set of protocols, routines, and tools that allow different software applications to communicate with each other.
- **Autonomous Robots:** Robots capable of performing tasks without human intervention, using onboard sensors, AI algorithms, and navigation techniques.
- **Backend:** The server-side component of a software application, handling logic, database interactions, and server configurations, facilitating client-server communication.
- **Bluetooth:** A wireless technology standard for short-range communication, enabling devices to exchange data.

- **CI/CD (Continuous Integration/Continuous Deployment):** Practices in software engineering for automating code integration, testing, and deployment to ensure quality and speed in software releases.
- **Cloud Infrastructure:** A network of remote servers and services hosted on the internet, providing data storage, processing power, and secure, scalable access for applications.
- **Collision Detection:** A process within robotics and autonomous navigation for identifying obstacles to avoid or respond to physical collisions.
- **Data Flow Diagram (DFD):** A visual representation of data flow within a system, showcasing data sources, destinations, and paths.
- **Dijkstra's Algorithm:** A pathfinding algorithm used to determine the shortest path between nodes in a graph, commonly applied in route optimization and navigation.
- **Error Handling:** The implementation within software to address errors, providing users with feedback or options for corrective actions.
- **IDE (Integrated Development Environment):** A software suite providing tools like editors and debuggers, such as Android Studio, for developing and testing applications.
- **Indoor Mapping:** The process of creating maps for indoor navigation, detailing floor layouts for autonomous robots.
- **Latency:** The delay between a data transmission request and the beginning of the data transfer, impacting real-time communication.
- **LIDAR (Light Detection and Ranging):** A sensor technology that measures distances using laser light, often used for 3D mapping and obstacle detection in autonomous navigation.
- **Mobile Application:** Software specifically designed for mobile devices, enabling user interaction with systems like *RoboNav* for robot monitoring and control.
- **Non-functional Requirements**: Specifications on system attributes like performance, reliability, and usability, defining how a system should operate rather than specific behaviors.
- **Obstacle Avoidance:** A function in robotics that allows systems to identify and navigate around obstacles using sensors.
- **Path Optimization:** Algorithms and techniques for determining the most efficient route or sequence in navigation or task completion.

- **Real-time Data Handling:** The system capability to process and respond to data inputs immediately, supporting timely user interactions.
- **Retrofit:** A library used in Android development for simplifying HTTP requests to backend services.
- **Robot Control System:** A system enabling direct commands to be sent to robots, managing tasks like movement, operation, and error handling.
- **ROS (Robot Operating System):** A flexible framework for writing robot software, providing libraries and tools to assist in building robot applications.
- **Scalability:** The capacity of a system to handle increased load or to expand without compromising performance, critical for growing user or data volumes.
- **SRS (Software Requirements Specification):** A document detailing both technical and functional system requirements, ensuring clear project guidelines.
- **SSL/TLS (Secure Sockets Layer/Transport Layer Security):** Protocols providing encryption for data transmitted between a client and server, enhancing security.
- **UI (User Interface):** The layout and design of an application's screens that allow users to interact with the system directly.
- **Unit Testing:** A software testing method that verifies the functionality of individual components of an application.
- **Use Case Diagram:** A visual representation of the interactions between users (actors) and a system, mapping out functionalities from the user's perspective.
- **UAT (User Acceptance Testing):** The final phase of software testing, where real users validate whether a system meets their requirements.
- **User-friendly:** Design aspects of a system or application aimed at enhancing ease of use and accessibility for end users.
- **WebSocket:** A protocol enabling real-time, bidirectional communication between client and server, often used for low-latency applications like real-time status monitoring.
- **Wireframe:** A visual guide representing the skeletal framework of a mobile or web application interface, used in the design phase.

1.4. References

- OrionStar Developer Support [Website]
- OrionStar RobotSample GitHub [GitHub]
- MAMC: A Multi-Agent Autonomous Mobility Control System [PDF]
- *RoboNav* Project Proposal Document, October 9, 2024 [PDF]
- Project Estimation Report, October 25, 2024 [PDF]

1.5. Overview

The Software Requirements Specification (SRS) for the *RoboNav* project outlines the essential requirements for a robotic navigation system that enhances backend API capabilities and provides a mobile application for user interaction. The project aims to improve backend performance, scalability, and real-time data handling for effective robot monitoring and control through the mobile app.

- **Section 1:** Introduces project objectives, scope, and terminology.
- **Section 2:** Provides a high-level description of functionalities, user classes, operating environment, and implementation assumptions.
- **Section 3:** Details functional and non-functional requirements, including real-time updates, route selection, error notifications, and cloud management, emphasizing scalability, security, and usability.
- **Section 4:** Describes user interactions and includes a use case diagram for process flows.
- **Section 5:** Specifies UI requirements, including design principles, wireframes, and navigation flows for an intuitive user experience.
- **Section 6:** Lists hardware, software, and network components essential for functionality.
- **Section 7:** Defines acceptance criteria through test cases and expected conditions.

This structured SRS guides the design, development, and testing processes, ensuring a cohesive approach to creating a reliable and scalable robotic navigation solution.

2. Specific Requirements

2.1. Functional Requirements

2.1.1. Real-Time Status Monitoring

- The mobile app shall display the robot's current location, battery level, and operational mode in real time, utilizing low-latency communication technologies like WebSocket for immediate data updates.
- The system shall provide continuous status updates, allowing users to monitor the robots' real-time conditions, such as battery levels, operational status, and current position on the indoor map.

2.1.2. Route Selection and Navigation

- The app shall allow users to select from predefined routes on the indoor map for each robot to follow autonomously.
- The backend system shall support pathfinding and route planning algorithms to ensure accurate navigation along user-selected routes.

2.1.3. Low Battery Alerts

- The system shall notify users through the mobile app when a robot's battery level drops below a predefined threshold, ensuring users are informed of critical power levels.
- The backend API shall monitor battery status and send alerts to the mobile app based on set battery level parameters.

2.1.4. Error Reporting

- The system shall detect and log any errors encountered by the robot, such as navigation issues or communication failures.
- The mobile app shall notify users of these errors in real time, providing diagnostic information to help users understand and address issues promptly.

2.1.5. Secure Real-Time Data Updates

- All data transmitted between the mobile app, backend system, and robots must be authenticated and encrypted to prevent unauthorized access.
- The backend API shall use secure protocols (e.g., HTTPS, SSL/TLS) to ensure data integrity and confidentiality during real-time updates.

2.1.6. Indoor Mapping Display

- The mobile app shall display an interactive, detailed indoor map showing each robot's current position and user-selected routes.
- Users shall be able to visualize the robots' paths on the map and observe real-time position updates as the robots move.

2.1.7. Backend API Functionality

- The backend API shall enable seamless communication between the mobile app and robots, supporting real-time status retrieval, movement commands, and logging capabilities.
- The API shall log robot activities and system events, ensuring users can access historical data and diagnostics if needed.

2.1.8. Cloud Based Data Storage and Access

- The system shall store real-time and historical robot status, including movement logs, alerts, and error reports, securely in a cloud-based environment.
- Authorized users shall be able to access this data as needed to monitor robot performance and review historical operations.

2.1.9. User Authentication and Login

- The mobile app shall require users to log in with authenticated credentials to access robot control and monitoring features.
- The system shall validate user credentials securely, ensuring only authorized users can access robot data and functionalities.

2.1.10. Robot Response Output

- The mobile app shall display responses from the robots after executing commands, indicating the status of command execution or providing feedback if an action fails.
- The system shall log each response for historical tracking, allowing users to review past interactions and troubleshoot if necessary.

2.2. Non-Functional Requirements

2.2.1. Scalability

- The backend API and cloud infrastructure should be scalable to handle additional robots if necessary.
- The cloud storage solution must accommodate an increasing volume of historical data and logs over time.

2.2.2. Availability

- The system shall automatically reconnect to the cloud and re-synchronize data in the event of a temporary network failure.

2.2.3. Usability

- The mobile application's UI shall follow best practices for non-technical users, hiding complex details and providing simple navigation and control options.
- The interface shall visually represent indoor maps clearly and provide easy-to-understand status indicators for each robot.

2.2.4. Security

- All data exchanged between the robots, cloud, and mobile app must be encrypted.

2.2.5. Resource Efficiency

- The system shall optimize data transmission to minimize bandwidth usage and cloud storage costs.

2.2.6. Performance

- The system should maintain a response time of less than one second for user actions on the mobile app to ensure a responsive user experience.
- Real-time robot status updates should refresh at intervals of no more than two seconds to provide accurate, timely information.

2.2.7. Maintainability

- The codebase should be modular and well-documented to support future updates and modifications with minimal downtime.
- Automated testing should be implemented for critical functions to facilitate reliable maintenance and quick identification of issues during development cycles.

3. Use Cases

3.1. Authenticating User Login

Actors: User, Backend API

Description: The user logs in through the mobile app. The backend API processes the login request and verifies the user's credentials.

Precondition: User must have valid login credentials, and the backend API must be reachable.

Steps:

1. User submits login information through the mobile app.
2. App sends the login request to the backend API.
3. Backend verifies the credentials.
4. If successful, the user gains access to the system.

Postcondition: User is authenticated and has access to *RoboNav's* features.

Exception: If credentials are incorrect, the backend sends an error message to the user.

3.2. Executing Robot Movement Commands

Actors: User, Backend API, Robot

Description: The user selects a destination on the app, and the backend API sends the movement command to the robot.

Precondition: Robot is online and connected to the backend API.

Steps:

1. User selects a robot and destination point.
2. App sends the movement command to the backend API.
3. Backend forwards the command to the robot.
4. Robot begins movement along the path.
5. Backend provides real-time updates on the robot's progress.

Postcondition: Robot reaches the destination successfully.

Exception: If the robot encounters an obstacle, it sends an alert to the backend for handling.

3.3. Monitoring Robot Status in Real Time

Actors: User, Backend API, Robot, Cloud Server

Description: The system provides real-time status updates (e.g., location, battery level) through the mobile app.

Precondition: Robot must send telemetry data to the backend periodically.

Steps:

1. Robot sends status updates to the backend.
2. Backend processes and synchronizes the data with the cloud.
3. User accesses the app to view robot status.
4. App displays battery levels, location, and operational status.

Postcondition: User stays informed of the robot's status.

Exception: If the robot disconnects, an error message is shown to the user.

3.4. Detecting Obstacles

Actors: Robot, Backend API, User

Description: The robot detects obstacles and sends alerts to the backend. The backend may notify the user depending on the severity.

Precondition: Robot must have operational sensors.

Steps:

1. Robot detects an obstacle.
2. Robot sends the obstacle data to the backend API.
3. Backend evaluates the severity of the alert.
4. If needed, the backend sends a notification to the user.

Postcondition: User is notified of obstacles, and the robot may adjust its path.

Exception: If the robot cannot avoid the obstacle, it stops and waits for further instructions.

3.5. Managing Battery Levels

Actors: Robot, Backend API, User

Description: The robot reports battery levels periodically. If the battery drops below a critical level, the backend triggers an alert.

Precondition: Robot must send regular battery updates to the backend.

Steps:

1. Robot sends battery data to the backend.
2. Backend checks if the battery is below a critical threshold.
3. If critical, backend sends an alert to the user.

Postcondition: User is informed and can send the robot for charging.

Exception: If the robot disconnects before sending battery data, an error message is displayed.

3.6. Notifying the User

Actors: Backend API, User

Description: The backend system sends alerts to the user via the mobile app based on important events.

Precondition: Backend API must detect significant events (e.g., low battery, obstacles, or errors).

Steps:

1. Backend evaluates the priority of an event (e.g., low battery or obstacle detected).
2. If the event requires user action, the backend sends a notification via the app.
3. User receives the alert on the app.

Postcondition: User is informed and can take action if needed.

Exception: If the notification fails, the backend retries or logs the issue.

3.7. Handling Errors

Actors: Robot, Backend API, User

Description: The backend manages errors encountered by the robot and notifies the user if needed.

Precondition: Robot must report errors to the backend system.

Steps:

1. Robot sends error data (e.g., motor failure) to the backend API.
2. Backend processes the error and records it.
3. If the error requires user intervention, the backend triggers a notification.
4. User receives the error notification on the app.

Postcondition: User can take corrective action or shut down the robot if needed.

Exception: If the error report fails to reach the backend, the robot retries sending it.

3.8. Cloud Data Synchronization

Actors: Backend API, Cloud Server

Description: The backend system stores and synchronizes robot data with the cloud.

Precondition: Cloud storage must be available and accessible.

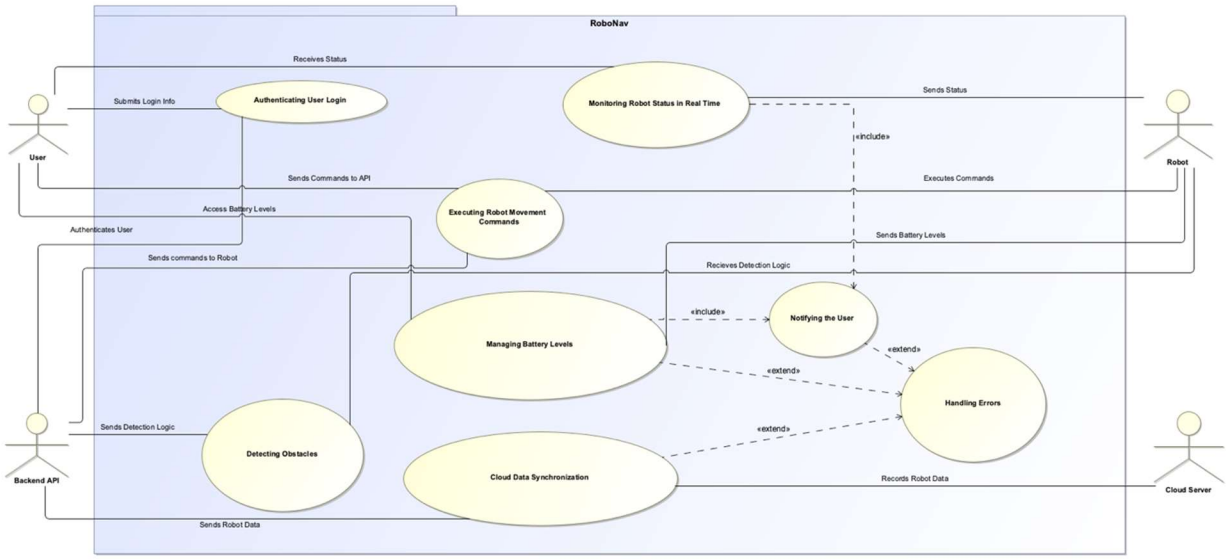
Steps:

1. Backend collects data from the robot (e.g., logs, status).
2. Backend sends the data to the cloud server.
3. Cloud server stores the data securely.

Postcondition: Data remains synchronized and accessible for future use.

Exception: If the cloud is unavailable, the backend stores the data locally and syncs it later.

3.9. Use Case Diagram



4. User Interface Requirements

4.1. UI Design Guidelines

4.1.1. General Principles

- **User-Centric Design**: Focus on the needs and preferences of users operating OrionStar robots. Conduct user research to understand their pain points and preferences.
- **Modern Aesthetic**: Use a sleek, contemporary look with a dark colour palette that enhances visibility and reduces eye strain, especially in low-light environments.

4.1.2. Layout and Structure

- **Intuitive Navigation**: Implement a bottom navigation bar for easy access to core features: Home, Map/Position, and Navigation. Ensure users can easily switch between these sections.
- **Consistent Layout**: Use a structured layout with clearly defined sections for Login, Dashboard, and Active tasks on the Home page. Group related features together for better organization.

4.1.3. Visual Design

- **Dark Palette**: Utilize a dark colour scheme that enhances contrast with text and icons, improving readability. Use accent colours for active elements and status indicators.
- **Typography**: Choose modern, legible fonts that stand out against the dark background. Maintain a clear hierarchy of information with varying font sizes and weights.

4.1.4. Interaction Design

- **Feedback Mechanisms**: Provide visual feedback (e.g., button animations, loading indicators) for user interactions to confirm actions have been registered. Use toast messages for brief confirmations and alerts.
- **Error Handling**: Design clear and informative error messages that guide users on how to resolve issues. Use inline validation for forms to enhance the user experience.

4.1.5. Performance Optimization

- **Responsive Design**: Ensure that the app adapts well to various Android device sizes and resolutions, maintaining usability across devices.
- **Loading Times**: Optimize API calls to minimize loading times. Use caching where appropriate to enhance performance and responsiveness.

4.1.6. User Customization

- **Personalization Options**: Allow users to customize settings (e.g., theme preferences, notification settings) to enhance their experience.
- **Save User Preferences**: Implement functionality to remember user preferences between sessions, improving usability and efficiency.

4.1.7. Testing and Validation

- **User Testing**: Conduct usability tests with actual users of OrionStar robots to gather feedback on the app's functionality and design.
- **Iterative Design**: Use an iterative design approach to continuously improve the app based on user feedback and analytics.

4.1.8. Documentation

- **User Manuals**: Provide clear user manuals and help sections within the app to assist users in navigating features and troubleshooting issues.
- **Onboarding Tutorials**: Implement onboarding tutorials that guide new users through the app's main features and functions, ensuring they can use the app effectively.

4.2. Screen Mockups

4.2.1. Overview

- This section provides visual representations of the user interface for *RoboNav*, illustrating how users will interact with the OrionStar robots through the app.

4.2.2. Mock-up Presentations

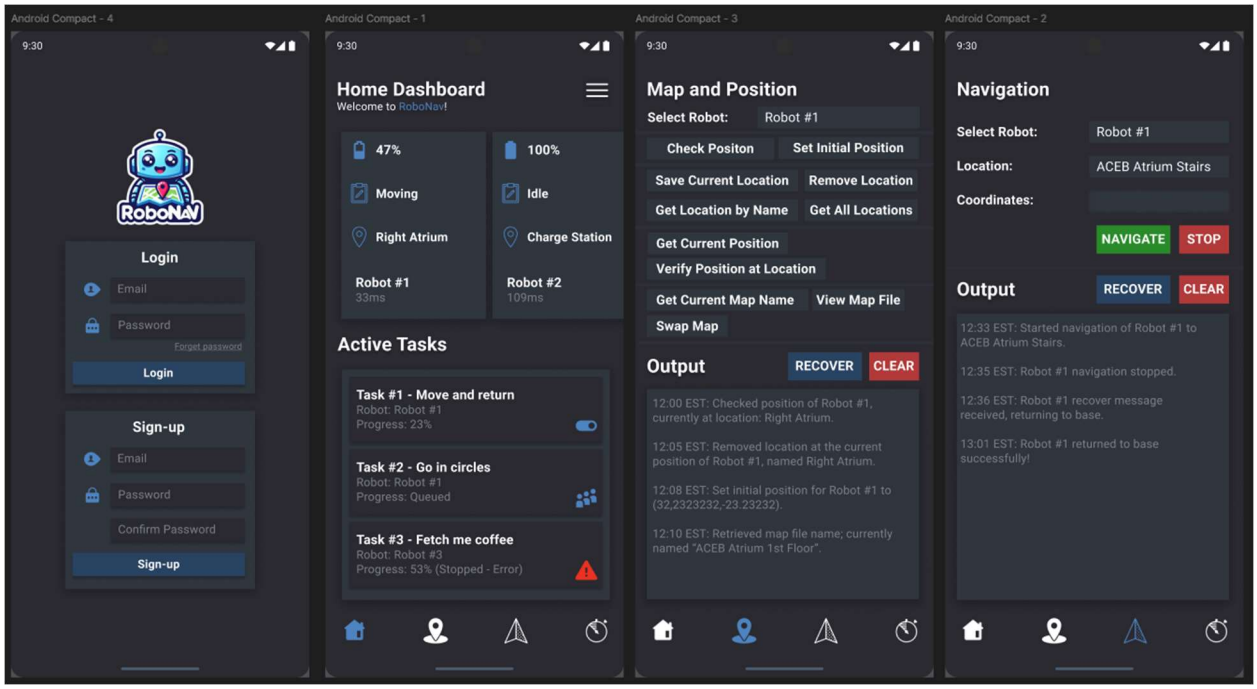
1. **Login Screen**
    - **Purpose**: This screen allows users to log in securely to access the app.
    - **Key Features**:
        - Includes fields for username and password.
        - A "Login" button for authentication.
        - Option for password recovery with a "Forgot Password?" link below the password field.
    - **User Interaction**: Users enter their credentials and click "Login" to access their dashboard. If they forget their password, they can click the "Forgot Password" link for recovery options.

2. **Home Dashboard**
    - **Purpose**: The central hub for users to view robot status and current tasks.
    - **Key Features**:
        - **Robot Overview**: Displays key information for each robot, including:
            - **Battery Level**: Current battery status.
            - **Current Task**: The task currently being executed.
            - **Location**: The robot's current position.
            - **Robot Name**: Identifier for each robot.
            - **Ping**: Connection status.
        - **Active Tasks Section**: Lists tasks being completed by various robots below the overview.
    - **User Interaction**: Users can monitor robot status and review active tasks from this screen.
3. **Map/Position Screen**
    - **Purpose**: Users can manage the robot's position and perform various position-related functions.
    - **Key Features**:
        - **Robot Selection**: Option to select a robot for position management.
        - **Functions Available**: Users can choose from various functions, including:
            - Check Position
            - Set Initial Position
            - Save Current Location
            - Remove Location
            - Get Location by Name
            - Get All Locations
            - Get Current Position
            - Verify Position at Location

        - Get Current Map Name
        - View Map File
        - Swap Map
    - **Output Terminal**: Located at the bottom, displaying results of the selected functions. It can be cleared or recovered.
- **User Interaction**: Users can select a robot, perform actions related to its position, and view results in the output terminal.

4. **Navigation Screen**
    - **Purpose**: This screen allows users to navigate robots to specific locations.
    - **Key Features**:
        - **Robot Selection**: Users can select a robot to navigate.
        - **Navigation Options**: Input options for selecting a location or coordinates.
        - **Output Terminal**: Located at the bottom, providing feedback on navigation commands and results, similar to the Map/Position screen.
    - **User Interaction**: Users can issue navigation commands, view output results, and clear or recover outputs as needed.

4.2.3. Figma Mock-up



4.2.4. Conclusion

These detailed mock-up descriptions highlight the key functionalities of *RoboNav*, and the interaction flows for users managing OrionStar robots. This is the initial version of the application, focusing on integrating the existing manufacturer API into a clean and modern interface.

Looking ahead, we plan to implement additional backend features in a new tab if time allows. We will also consider incorporating other existing API functionalities into the frontend to enhance user experience and expand the app's capabilities.

5. System Requirements

5.1. Hardware Requirements

5.1.1. Development Machine:

- **Processor:** A modern multi-core processor (Intel, AMD, or equivalent).
- **RAM:** Minimum of 8 GB RAM recommended for efficient development.
- **Storage:** Sufficient storage to accommodate the development tools, SDKs, and project files (SSD recommended for faster performance).
- **Operating System:** Windows 10 or later, macOS Mojave (10.14) or later, or a recent version of Linux (e.g., Ubuntu 20.04 or later).

5.1.2. Target Devices:

- **Android Device:**
    - Compatible with Android version 7.0 (Nougat) or higher (API Level 24).
    - Support for communication protocols required for interacting with OrionStar robots (e.g., Wi-Fi, Bluetooth).

5.1.3. Robot Communication:

- An OrionStar robot that supports the necessary communication protocols for app interaction.

5.2. Software Requirements

5.2.1. Development Environment:

- **IDE:** Android Studio (latest stable version) for Android development.
- **SDK:** Android SDK (API Level 24 or higher) to ensure compatibility with a wide range of devices.
- **Programming Languages:** Groovy/Java for app development.
- **OrionStar SDK:** Specific SDK provided by OrionStar for integrating robot functionalities within the app.

5.2.2. Libraries/Frameworks:

- **Retrofit:** For API calls to backend services.
- **Glide/Picasso:** For image loading and caching if needed.
- **RxJava:** For managing asynchronous tasks and threading (if applicable).
- **JUnit:** For unit testing.
- **Espresso:** For UI testing to ensure a smooth user experience and functionality of the app's interface.

5.3. Network Requirements

5.3.1. Communication Protocols:

- **Wi-Fi:** Required for connecting to OrionStar robots and backend services. The device should support local network connectivity.
- **Bluetooth:** Required if the robot communication involves Bluetooth connectivity.

5.3.2. Network Stability:

- A stable internet connection is essential for real-time monitoring and updates, especially when communicating with backend servers or cloud services.

5.3.3. Firewall Settings:

- Ensure that firewall settings allow communication on the necessary ports used by OrionStar robots and backend services.

5.3.4. Latency Considerations:

- Low-latency network connection is preferred to ensure quick response times during real-time operations with the robots.

6. Acceptance Criteria

6.1. Acceptance Test Cases

6.1.1. User Login

- **Test Case ID:** TC001
- **Objective:** Verify that users can successfully log in with valid credentials.
- **Preconditions:** User has a valid username and password.
- **Test Steps:**
    1. Open the login screen.
    2. Enter valid username and password.
    3. Click the "Login" button.
- **Expected Result:** User is redirected to the Home Dashboard.

6.1.2. Invalid Login

- **Test Case ID:** TC002
- **Objective:** Verify that the system displays an error message for invalid credentials.
- **Preconditions:** User enters invalid credentials.
- **Test Steps:**
    1. Open the login screen.
    2. Enter invalid username and/or password.
    3. Click the "Login" button.
- **Expected Result:** An error message is displayed indicating invalid credentials.

6.1.3. Robot Overview Display

- **Test Case ID:** TC003
- **Objective:** Ensure the Home Dashboard displays robot status correctly.
- **Preconditions:** User is logged in and has at least one robot registered.
- **Test Steps:**
    1. Navigate to the Home Dashboard.
- **Expected Result:** The dashboard shows battery level, current task, location, robot name, and connection status for each robot.

6.1.4. Position Management

- **Test Case ID:** TC004
- **Objective:** Verify the ability to check and set the robot's position.
- **Preconditions:** User is logged in and has a robot selected.
- **Test Steps:**
    1. Navigate to the Map/Position screen.
    2. Select a robot.
    3. Click "Check Position."
    4. Set an initial position.
- **Expected Result:** The current position is displayed correctly, and the initial position is set without errors.

6.1.5. Navigation Functionality

- **Test Case ID:** TC005
- **Objective:** Verify that users can navigate the robot to a specified location.
- **Preconditions:** User is logged in and has a robot selected.
- **Test Steps:**
    1. Navigate to the Navigation screen.
    2. Select a robot.
    3. Input a valid location or coordinates.
    4. Click "Navigate."
- **Expected Result:** The robot receives the navigation command, and feedback is displayed in the output terminal.

6.1.6. UI Responsiveness

- **Test Case ID:** TC006
- **Objective:** Ensure the app maintains responsive UI across various screen sizes and orientations.
- **Preconditions:** The app is installed on different Android devices with varying screen sizes.
- **Test Steps:**
    1. Open the app on devices with different screen sizes and orientations (portrait and landscape).
- **Expected Result:** The UI adapts correctly without any layout issues, maintaining usability across all devices.

6.1.7. Error Handling

- **Test Case ID:** TC007
- **Objective:** Verify the error messages displayed for invalid actions.
- **Preconditions:** User is logged in and interacts with features incorrectly (e.g., selecting an unregistered robot).
- **Test Steps:**
    1. Perform an invalid action (e.g., navigate to an unregistered robot).
- **Expected Result:** Clear and informative error messages guide the user on how to resolve the issue.

6.2. Acceptance Conditions

6.2.1. Functionality

- All core features of the app (login, dashboard, position management, navigation) must be fully functional without critical bugs or crashes.

6.2.2. Performance

- The app should load within 3 seconds on average and maintain a response time of less than 1 second for user interactions.

6.2.3. Usability

- Usability testing must yield a satisfaction score of at least 80% from actual users during testing sessions.

6.2.4. Compatibility

- The app must function correctly on all targeted Android devices (API Level 24 and above) and screen sizes.

6.2.5. Security

- User credentials and sensitive data must be securely stored and transmitted, complying with relevant security standards.

6.2.6. Documentation

- Comprehensive user manuals and onboarding tutorials must be provided, ensuring users can effectively utilize the app.