TED UNIVERSITY

CMPE 492 SENIOR PROJECT II

SEARCH AND RESCUE OPERATION PORTAL (SAROP)

Test Plan Report

Project Name: SAROP (Search and Rescue Operation Portal)

Project Url: sarop.tech

Team Members:

• Arda Gök

• Saliha Nursu Baltacı

• Ceren Özdoğan

• Mert Çıkla

Name of the supervisor: Emin Kuğu

Names of the juries: Tolga Kurtuluş Çapın - Venera Adanova

1. Introduction

This document is designed to provide a comprehensive outline of our testing strategy for both the front-end, back-end, mobile application components of the SAROP system. As SAROP aims to be a pivotal tool in improving the efficiency and effectiveness of search and rescue operations, it is important that the portal operates in a great manner under various conditions and meets all specified functional requirements.

The purpose of this document is to detail the scope, objectives, and methods for testing the SAROP application. It will cover various aspects including user authentication, integration, security, error handling, and user experience. This testing will ensure that SAROP not only meets the highest standards of operational excellence but also provides a user-friendly interface that can be relied upon in critical situations. Our test plans aim to challenge the system's functionalities, uncover potential issues, and guarantee that all user interactions are intuitive and robust.

2. Scope

2.1 Mobile:

2.1.1 In Scope:

1. User Authentication and Registration:

- Check that the screens for user registration and login work properly.
- Experiment with making a new user and login in using a working username and password.
- Make sure the appropriate error messages for unsuccessful login attempts are shown.

2. Integration Testing:

- Utilize the Read function offered by the API.
- Verify that every endpoint reacts to queries in a suitable manner.
- Verify that the API is handling authorization and authentication procedures correctly.

3. Security Testing:

- Verify data security while registering and logging in as users.
- Confirm that authorization and authentication procedures are applied securely.
- Ensure that in the event of an illegal access attempt, the proper error messages are shown.

4. Error Handling:

- Verify that the program correctly handles incorrect requests, such as those resulting from failed login attempts or missing fields.
- Verify that the API's failures are being handled appropriately.

5. User Experience:

- Confirm that the application has a user-friendly interface.
- Ensure all screens and functionalities work smoothly.
- Verify that users can easily navigate and perform desired actions.

6. Interactive Map Features:

- Testing map display.
- Evaluating map layer selection and visualization capabilities.

2.1.2 Out Scope:

1. Performance Testing:

- Assess the processing speed and reaction times of the application.
- To make sure the application functions properly, test it in various network scenarios.

2. Battery and Resource Usage:

- Verify that the app efficiently controls device resources (memory, CPU, network consumption, etc.) and battery life.

3. Different Devices and Screen Sizes:

- Verify that the app functions properly across a range of mobile devices and screen sizes.
- Make that the UI adjusts appropriately for varying screen resolutions.

4. Load Testing:

- This test plan does not involve detailed load testing to replicate high traffic scenarios.

2.2 Backend:

2.2.1 In Scope:

1. Functional Testing:

- Verify that all the CRUD(Create/Read/Update/Delete) operations work correctly for any role.
- Verify that all the endpoints return queries in a correct way.

2. Security Testing:

- Validate that the API endpoints handle authentication and authorization properly.

3. Error Handling:

- Verify that the API returns appropriate error messages for invalid requests, such as missing required fields or incorrect data types.

4. Integration Testing:

- Verify that map endpoints work correctly with GeoServer services.
- Verify that application loads data from related search and rescue team endpoints.

2.2.2 Out Scope:

1. Performance Testing:

- Evaluating the API's response time and handling of concurrent requests is not included in this test plan.

2. Compatibility Testing:

- Testing the API's compatibility across different platforms, devices, or browsers is not covered in this test plan.

3. Regression Testing:

- Comprehensive regression testing to ensure new changes do not impact existing functionalities is not part of this test plan.

4. Load Testing:

- Detailed load testing to simulate heavy traffic conditions is not included in this test plan.

2.3 Frontend:

2.3.1 In Scope:

The frontend testing scope for SAROP includes a thorough examination of the following components and functionalities:

1. **User Authentication and Authorization:**
   - Testing user login, logout, and password management functionalities.
   - Validating role-based access control (Admin vs. User) across all modules.

2. **Interactive Map Features:**
   - Testing map display, navigation controls, marker placement, and path drawing functionalities.
   - Evaluating map layer selection and visualization capabilities.

3. **Operation Management:**
    - Validating operation viewing, creation, update, and assignment functionalities.
    - Testing real-time updates and notifications for operational activities.

4. **User Interfaces:**
    - Assessing front-end responsiveness, clarity, and intuitiveness across different devices and screen sizes.
    - Verifying UI adherence to design specifications and user experience guidelines.

2.3.2 Out Scope:

Certain aspects are considered out of scope for SAROP's frontend testing, including:

- Backend services, API interactions, and database operations (covered separately in backend testing).
- Performance, load, and stress testing of backend systems.
- Hardware-specific testing (focused solely on front-end software components).

3. Quality Objective

3.1 Mobile:

3.1.1 Accuracy:

- Verify that user credentials are correctly captured and authenticated on the login and registration screens. Make sure the error messages you receive after trying to log in are clear and concise.

3.1.2 Robustness of Integration Testing:

- Verify that the API's Read function operates accurately and consistently across all endpoints.
- Verify that the permission and authentication processes are handled efficiently by the API and that it answers questions in a suitable manner.

3.1.3 Assurance of Security:

- Make sure that during the registration and login processes, user data is communicated and kept securely.
- To avoid unwanted access, make sure that permission and authentication processes are carried out securely.
- Verify that when unauthorized access attempts occur, the appropriate error messages are shown.

3.1.4 Efficiency in Handling Errors:

- Verify that the program correctly responds to erroneous requests—like unsuccessful login attempts or missing fields—without crashing or jeopardizing user information.
- Make sure the API responds to errors meaningfully and can handle failures appropriately.

3.1.5 User Satisfaction with Experience:

- Evaluate the user interface's usability and intuitiveness.
- Verify sure there are no hiccups or delays in the operation of any screens or features.
- Check that users can easily browse the program and carry out the desired actions without experiencing any confusion.

3.1.6 Assessment of Interactive Map Functionality:

- Verify that the map renders accurately and quickly by testing its display.
- Examine the map layers' selection and visualization capabilities to make sure they are accurate and responsive.

3.2 Backend:

3.2.1 Functional Correctness:

- Ensure that all the endpoints operate correctly.
- Verify that the endpoint returns the correct data when queried.

3.2.2 Security and Authorization:

- Confirm that the API endpoints handle authentication and authorization properly, allowing only authorized users to access and modify team location data.

3.2.3 Error Handling:

- Validate that the API returns appropriate and informative error messages for invalid requests, such as missing required fields or incorrect data types.
- Ensure that the API gracefully handles edge cases and unexpected inputs without crashing or exposing sensitive information.

3.2.4 Reliability:

- Verify that the API can handle a reasonable number of concurrent requests without degradation in performance or stability.

- Ensure that the API can recover from failures and maintain data integrity in the event of system or network disruptions.

3.2.5 Usability:

- Confirm that the API response format and structure are intuitive and easy to consume for client applications.
- Ensure that the API documentation is comprehensive and provides clear guidance on how to interact with the endpoints.

3.2.6 Maintainability:

- Verify that the API code follows best practices and industry standards, making it easy to understand, extend, and maintain in the future.
- Ensure that the API is designed with modularity and scalability in mind, allowing for easy integration with other system components.

3.3 Frontend:

The overarching quality objectives for SAROP's frontend testing are aligned with ensuring:

3.3.1 Functional Validation:

- All front-end features and functionalities adhere to specified requirements and acceptance criteria.
- Smooth navigation and seamless user interactions across the SAROP application.

3.3.2 Usability and User Experience:

- Frontend interfaces are assessed for clarity, consistency, and ease of use.
- Identification and resolution of usability issues to enhance overall user experience.

3.3.3 Defect Identification and Resolution:

- Thorough documentation and prioritization of front-end defects based on severity and impact.
- Collaboration with development teams to promptly address and resolve identified issues.

4. Roles and Responsibilities

PERSON	TASK
MERT ÇIKLA	Applying Backend Tests
CEREN ÖZDOĞAN	Applying Frontend User Screen Test
SALİHA NURSU BALTACI	Applying Frontend Map Screen Test
ARDA GÖK	Applying Mobile Application Tests

The success of SAROP's frontend testing relies on the collaborative efforts of the following team members:

Mert Çıkla: Backend Tester

- Create API endpoint documentation for backend tests
- Execute manual and automated tests to verify backend functionalities and user interactions.
- Document test results, track defects, and collaborate with developers for issue resolution.

Ceren Özdoğan And Saliha Nursu Baltacı: Frontend Tester

- Testing API endpoints by calling each endpoint on frontend.
- Execute manual and automated tests to verify frontend functionalities and user interactions.
- Testing map functionalities such as scaling, zoom in/out, drawing polygon, saving note.

- Document test results, track defects, and collaborate with developers for issue resolution.

Arda Gök: Mobile Tester

- Testing login/register endpoints on mobile
- Execute manual and automated tests to verify mobile functionalities and user interactions.
- Testing map functionalities such as scaling, zoom in/out, drawing polygon, saving note.
- Document test results, track defects, and collaborate with developers for issue resolution.

5. Test Methodology

In this section, we discuss our detailed testing plan used to check the performance and effectiveness of our mobile application and web application in this project. Our testing is organized into different levels: Unit Testing, Integration Testing, System Testing, and User Acceptance Testing. Each level focuses on specific parts or interactions within the system.

Unit Testing analyzes each operation in the application, both in the frontend and backend, to make sure each part works correctly by itself.

Integration Testing ensures that services of the application work well together. For the mobile and frontend part, this includes checking how frontend components interact with backend APIs. For the backend, it involves testing how different server endpoints and relation with GeoServer communicate.

System Testing checks the entire system's functionality and performance to make sure it works as expected.

User Acceptance Testing is the testing step, where we check if the application meets the needs and expectations of the users.

To determine if our testing is complete, we look at several factors, such as whether all planned test cases have been carried out, whether any serious problems have been solved, and whether the application meets the quality standards we set.

5.1 Mobile

5.1.1 Test Levels:

5.1.1.1 Unit Testing:

Validate each mobile frontend component's functionality in isolation, such as login, registration, map viewing, and dropdown menus.

- Verify that every component responds to user interactions appropriately and performs as anticipated.
- Recognize and fix any testing-related front-end-specific problems.

5.1.1.2 Integration Testing:

1. **Authentication and Authorization:**
    - Verify that user authentication in mobile applications is handled effectively by the API endpoints.

2. **Security Checks:**
    - Test the mobile application for potential security vulnerabilities.

3. **Test the Other Integration:**
    - Validate that map viewing functionality retrieves and displays data from the backend API accurately. Verify interactions and integrations between mobile frontend modules/components and backend APIs.
    - Test data flow between the mobile frontend and backend systems, ensuring seamless communication and proper handling of data.

- Test dropdown menus' functionality to select options and fetch data from the backend as required.

5.1.1.3 System Testing:

1. Error Handling:

- Validate that the mobile app returns appropriate and informative error messages for invalid API fetch’s, such as missing required fields or incorrect data types.
- Ensure that the mobile application gracefully handles edge cases and unexpected inputs without crashing or exposing sensitive information.

2. Usability:

- Test the entire user flow, including login, registration, map viewing, and data retrieval from backend via dropdown menus.
- Perform end-to-end validation of SAROP's mobile frontend to ensure seamless functionality and usability.

5.1.1.4 User Acceptance Testing:

Involve such as stakeholders end users to verify that the mobile frontend satisfies user needs and acceptance standards.

- Gather feedback on the general operation and design of the mobile application's user interface.
- Confirm that stakeholders are satisfied with the mobile application's functionality and suitability for their needs.
- To make sure that the finished product lives up to user expectations, address any issues or difficulties that come up during user acceptability testing.

5.1.2 Test Completeness:

5.1.2.1 User Authentication and Registration Testing:

- Check that the login and registration screens work properly.
- Verify that newly registered users are logged in using legitimate credentials.
- Make sure that when a login attempt fails, error messages are displayed correctly.
- Verify the security of the data while logging in and registering.

5.1.2.2 Integration Testing:

- Make use of the API Read method for every endpoint.
- Confirm that authorization and authentication are handled correctly.
- Examine how the front-end and back-end systems interact.
- Verify that dropdown menus correctly retrieve data from the backend.

5.1.2.3 Security Testing:

- Verify data security when logging in and registering.
- Verify that authorization and authentication are handled securely.
- Check that the right error messages are displayed in response to unauthorized access attempts.

5.1.2.4 Error Handling:

- Assure proper handling of erroneous requests, such as unsuccessful login attempts.
- Check how the API handles failures and errors.

5.1.2.5 User Experience Testing:

- Assess the usability of the user interface.
- Verify that all screens and functionalities operate without a hitch.
- Guarantee simple operation and navigation.

5.1.2.6 Interactive Map Features Testing:

- Verify that the map renders quickly and accurately.
- Examine the map layers' selecting and visualization capabilities.

5.1.2.7 Unit Testing:

- Examine each front-end component's functionality separately.
- Ensure that user interactions receive suitable answers.

5.1.2.8 System Testing:

- Verify error messages pertaining to incorrect API fetches.
- Examine the entire user experience, including registration, login, and map browsing.

5.1.2.9 User Acceptance Testing:

- Consult stakeholders to confirm that users are satisfied.
- Get input on the general usability and design of the interface.

5.2 Backend

5.2.1 Test Levels:

5.2.1.1 Unit Testing:

1. **User Registration:**

   - Verify that the /auth/register endpoint correctly processes the input data (name, email, password) and creates a new user account.
   - Validate that the generated access_token and refresh_token are correctly formatted and contain the expected user information.

2. **User Login:**

   - Ensure that the /auth/login endpoint correctly authenticates users with valid credentials (email and password).
   - Test the endpoint with both valid and invalid credentials and verify that the API returns the expected access_token and refresh_token (for valid credentials) or appropriate error messages (for invalid credentials).

3. **Admin Login:**

   - Verify that the /auth/login endpoint correctly authenticates administrators with valid credentials (email and password).
   - Test the endpoint with both valid and invalid admin credentials and ensure that the API returns the expected access_token and refresh_token (for valid credentials) or appropriate error messages (for invalid credentials).

4. **Creation of All the Entities**

   - Verify that entity create endpoints works correctly based on the authenticated role. Test it with the credentials of users and see whether the function operates correctly or returns an error message.

- Verify that creation of the entities works correctly with one-to-one, many-to-one, many-to-many relationships.
- Test the endpoint with missing arguments in request body to see how it operates.

5. Update of All the Entities

- Verify that entity update endpoints works correctly based on the authenticated role. Test it with the credentials of users and see whether the function operates correctly or returns an error message.
- Verify that edit of the entities works correctly with one-to-one, many-to-one, many-to-many relationships.
- Test the endpoint with missing arguments in request body to see how it operates.

6. Deletion of All the Entities

- Verify that entity delete endpoints works correctly based on the authenticated role. Test it with the credentials of users and see whether the function operates correctly or returns an error message.
- Verify that deletion of the entities works correctly with one-to-one, many-to-one, many-to-many relationships.

5.2.1.2 Integration Testing:

1. **Authentication and Authorization:**

   - Ensure that the API endpoints correctly handle authentication and authorization, allowing only authorized users to access and perform actions.
   - Verify that the access_token and refresh_token are properly validated and used for subsequent requests.

2. **Security Checks:**

   - Test the API endpoints for potential security vulnerabilities.
   - Ensure that the API properly sanitizes and validates all input data to prevent such vulnerabilities.

3. **GeoServer Post and Delete Operations:**

   - Ensure that API creates a relationship with GeoServer API and correctly post and delete the map from the server.

4. **Loading Data from Related Endpoints on Launch:**

   - Ensure that the API creates a relationship with related endpoints on launch of application and gets data from these APIs.
   - Ensure that the application operates in this job in an applicable performance.

5.2.1.3 System Testing:

1) **Error Handling:**

   - Validate that the API returns appropriate and informative error messages for invalid requests, such as missing required fields or incorrect data types.
   - Ensure that the API gracefully handles edge cases and unexpected inputs without crashing or exposing sensitive information.

2) **Usability:**

- Verify that the API response format and structure are intuitive and easy to consume for client applications.
- Ensure that the API documentation is comprehensive and provides clear guidance on how to interact with the endpoints.

5.2.1.4 User Acceptance Testing:

3) **Functional Correctness:**

- Ensure that all the endpoints work correctly with their acceptance criteria such as creation of entity, update of entity, deletion of entity in a correct way as well as operating in relationship side too by ensuring data consistency.
- Confirm that return values of all the endpoints are correct and valid.

4) **Security and Authorization:**

- Verify that the API endpoints handle authentication and authorization properly, allowing only authorized users to access and perform actions.

5) **Reliability:**

- Validate that the API can handle a reasonable number of concurrent requests without degradation in performance or stability.
- Ensure that the API can recover from failures and maintain data integrity in the event of system or network disruptions.

5.2.2 Test Completeness:

5.1.2.1 Unit Testing:

- Ensure that the individual components of the endpoints function as expected, including input validation, operating correctly, and returning the correct data.
- Validate edge cases and error handling at the unit level.

5.1.2.2 Integration Testing:

- Verify the integration of the authentication-related endpoints with other system components, such as the user management and authorization modules.
- Confirm that the access_token and refresh_token are properly validated and used for subsequent requests.
- Confirm that the relation and integration with other services works in a specified time.

5.1.2.3 System Testing:

- Evaluate the overall system behavior, including the handling of concurrent requests, error recovery, and data integrity.
- Ensure that the API meets the expected performance and reliability requirements.

5.1.2.4 User Acceptance Testing:

- Confirm that the API endpoints meet the functional, security, and usability requirements from the end-user's perspective.
- Validate the API's ability to handle real-world usage scenarios and edge cases.

5.3 Frontend

5.3.1 Test Levels:

5.3.1.1 Unit Testing:

- **Objective**: To test individual components of the frontend in isolation to ensure that each part functions correctly by itself.
- **Approach:** Developers write unit tests for each component, such as buttons, input fields, and visual elements, to check their behavior under various scenarios. Tools like Jest or Mocha can be used for this purpose.
- **Coverage:** Focus on testing the logic of JavaScript functions, the rendering of components without external dependencies, and the correct application of styles.

5.3.1.2 Integration Testing:

- **Objective:** To verify that different components of the frontend interact correctly with each other and with backend APIs.
- **Approach:** Conduct tests that simulate user interaction with the application that involves multiple components working together. This includes testing form submissions, navigation between pages, and any client-side logic that integrates different modules.
- **Coverage:** Ensure that API calls fetch and post data accurately, components update dynamically in response to user inputs, and transitions between application states are smooth.

5.3.1.3 System Testing:

- **Objective:** To validate the complete and integrated software product to ensure it meets the defined requirements.
- **Approach:** Execute comprehensive tests covering the full functionality of the frontend as it interacts with the backend and other systems. This level checks the complete flow of the application from start to finish.

- **Coverage:** Test the application in environments that mimic real-world operating conditions. Assess the overall system behavior, including loading times, data accuracy, and responsiveness.

5.3.1.5 Acceptance Testing:

- **Objective:** To confirm that the front end meets the business requirements and is ready for deployment and use by end-users.
- **Approach:** This testing can include User Acceptance Testing (UAT), where actual users test the system in a production-like environment to validate the user experience against their expectations and needs.
- **Coverage:** Focus on user-driven scenarios and use cases that reflect real-life usage of the application. Evaluate the application's usability, content, and overall performance to ensure it aligns with user expectations and business objectives.

5.3.1.6 Usability Testing:

- **Objective:** To determine if the application is intuitive, easy to use, and configured logically for end-users.
- **Approach:** Sessions where testers or actual users perform typical tasks using the application. Observations and feedback are gathered to identify areas of confusion or difficulty.
- **Coverage:** Key aspects like navigation flow, clarity of content, interaction simplicity, and visual design are evaluated. This testing helps in refining the user interface for better user satisfaction and reduced error rates.

5.3.1.6 Regression Testing:

- **Objective:** To ensure that new code changes have not adversely affected existing functionality of the application.

- **Approach:** Automated tests are run every time there's a change in the codebase to catch any regressions early in the development cycle.
- **Coverage:** Includes all critical paths and functionalities that might be affected by recent changes or updates to ensure that previous functionalities still perform as expected.

Each of these test levels plays a crucial role in validating different aspects of the frontend, ensuring the software is robust, user-friendly, and ready for deployment. This structured approach helps in identifying issues at early stages and guarantees that the application meets all performance, usability, and reliability requirements.

5.3.2 Test Completeness:

Criteria for determining test completeness for SAROP's frontend testing include:

- Execution of all planned test cases covering functional and non-functional requirements.
- Resolution of critical and major defects identified during testing.
- Validation against acceptance criteria and achievement of predefined quality metrics.

6. Project Tasks, Estimations, and Schedule

TEST	PERFORMED MEMBER	PASSED/FAILED/FUTURE TEST
BACKEND- Authorization tests(Register/Login)	Mert Çıkla	PASSED
BACKEND- Team Location endpoints(Create/Read/Update/Delete)	Mert Çıkla	PASSED
BACKEND- Team Endpoints(Create/Read/Update/Delete)	Mert Çıkla	PASSED
BACKEND- Map Endpoints(Create/Read/Delete)	Mert Çıkla	PASSED
BACKEND- Geoserver Post/Delete Relation	Mert Çıkla	PASSED
BACKEND- BACKEND- Polygon Endpoints(Create/Read/Delete)	Mert Çıkla	PASSED
BACKEND- Note Endpoints(Create/Read/Delete)	Mert Çıkla	PASSED
BACKEND- Operation Endpoints(Create/Read/Update/Delete)	Mert Çıkla	PASSED
BACKEND- Category Endpoints(Create/Read/Update/Delete)	Mert Çıkla	PASSED
BACKEND- Exception Handling – Crashing case when there is an error	Mert Çıkla	PASSED
BACKEND -Exception Handling – Error messages	Mert Çıkla	Future Test
BACKEND- Loading Data From API	Mert Çıkla	PASSED
BACKEND- Performance test while loading data from API	Mert Çıkla	FAILED
BACKEND- Performance test for the relation with Geoserver API	Mert Çıkla	PASSED
MOBILE- User Authentication and Registration	Arda Gök	Passed
MOBILE- Registration screen works properly	Arda Gök	Passed
MOBILE Login with valid credentials	Arda Gök	Passed
MOBILE- Display of appropriate error messages	Arda Gök	Passed
MOBILE - Data security during registration/login	Arda Gök	Passed
MOBILE- Utilization of API Read function	Arda Gök	Passed
MOBILE- Proper handling of authentication	Arda Gök	Passed
MOBILE- Interaction between frontend and backend	Arda Gök	Passed
MOBILE- Accuracy of dropdown menu data fetching	Arda Gök	Passed
MOBILE- Data security during registration/login	Arda Gök	Passed
MOBILE- Display of appropriate error messages	Arda Gök	Passed

MOBILE- Proper response to incorrect requests	Arda Gök	Passed
MOBILE- API's response to errors and failures	Arda Gök	Passed
MOBILE- Usability of the user interface	Arda Gök	Passed
MOBILE- Smooth operation of all features	Arda Gök	FAILED
MOBILE- Easy navigation and execution of actions	Arda Gök	Passed
MOBILE- Accurate and quick map rendering	Arda Gök	FAILED(NOT QUICK)
MOBILE- Selection and visualization capabilities	Arda Gök	Passed
MOBILE- Functionality of each frontend component	Arda Gök	Passed
MOBILE- Response to user interactions	Arda Gök	Passed
MOBILE- Validation of error messages	Arda Gök	Passed
MOBILE- End-to-end user flow validation	Arda Gök	Passed
FRONTEND- User Authentication and Registration	Ceren Özdoğan	Passed
FRONTEND- Role-Based Access Control Validation	Ceren Özdoğan	Passed
FRONTEND- Interactive Map Display and Controls	Nursu Baltacı	Passed
FRONTEND- Map Marker Placement and Path Drawing	Nursu Baltacı	Future Test
FRONTEND- Operation Management Functions	Nursu Baltacı	Passed
FRONTEND- Real-Time Updates and Notifications	Ceren Özdoğan	Passed
FRONTEND- Cross-Browser Compatibility	Ceren Özdoğan	Passed - Tested on Chrome, Firefox, Safari
FRONTEND- Responsive Design on Various Devices	Ceren Özdoğan	Passed- Includes tests on desktops,tablets,smartphones
FRONTEND- User Interface Consistency	Ceren Özdoğan	Passed
FRONTEND- Accessibility Compliance Testing	Ceren Özdoğan	Future Test
FRONTEND- Security Measures for User Data	Nursu Baltacı	Passed
FRONTEND- Error Message Handling	Nursu Baltacı	Passed
FRONTEND- Session Management Testing	Nursu Baltacı	Passed
FRONTEND-Load Times and Performance Optimization	Nursu Baltacı	Failed - Needs optimization for faster load times
FRONTEND-Usability Testing	Ceren Özdoğan	Passed- Scheduled to gather user feedback on interface usability

7. Resource & Environment Needs

7.1 Testing Tools

7.1.1 Backend:

1. **Postman:**
    - Postman is a popular API testing tool that will be used to execute the test cases for the endpoints.
    - Postman allows creating and running API requests, validating responses, and automating the testing process.
2. **Wiremock:**
    - Wiremock is a tool for stubbing and mocking HTTP services, which will be used to simulate the backend API endpoints.
    - Wiremock allows defining the expected API responses and behaviors, enabling isolated testing of the frontend application.

7.1.2 Mobile:

1. Emulator:

In Android studio we have emulator's that can exactly same with the mobile phone. And ı try the whole test with Android studio's emulator.

- Pixel 2 API
- Pixel 3 API

2. Local:

The other test area is my local computer and my phone. Xiaomi Redmi Note 8 pro.

7.2 Testing Environment

TESTING ENVIRONMENT	COMPUTER	RAM	CPU	MEMORY
Arda Gök	Hp Pavilion	16 GB	Amd Ryzen 4800H 2.90 GHz	256 GB SSD/1 TB HDD
Mert Çıkla	MSI Bravo 15 B5DD	16 GB	AMD Ryzen 5800H 3.2 GHz	1 TB SSD
Saliha Nursu Baltacı	Casper Excalibur G770	16 GB	Intel Core I5 2.5 GHz	500 GB SSD/1 TB HDD
Ceren Özdoğan	HP l15s-fq2xxx	8 GB	Intel Core i5 2.5 GHz	256 GB SSD/1 TB HDD

7.2.1 Web Application:

7.2.1.1 Development Environment

- The testing will be conducted in a local development environment, where the front-end application and the Wiremock server are running.
- This environment will be used for initial unit and integration testing of the front-end components interacting with the mocked API endpoints.

7.2.2 Staging Environment

- The front-end application will be deployed to a staging environment, where it will be tested against the actual backend API endpoints. This is a future plan for the time project is ready for related search and rescue teams' use.
- This environment will be used for end-to-end testing, ensuring the frontend application integrates correctly with the live API.

7.2.3 Mobile Application:

Testing Environment	Description
Development Environment	Testing the mobile application in a local development environment. Tests are carried out using Android Studio's emulators. In this environment, tests are carried out on Pixel 2 API and Pixel 3 API emulators.
Staging Environment	One of the testing areas is my local computer and my Xiaomi Redmi Note 8 Pro phone. In this environment, the application is tested on a real device.

8. Conclusion

This testing document lays out our careful plan to check and improve the SAROP system thoroughly. We've applied for a series of detailed tests. Our goal is to spot any errors and fix them, ensuring that SAROP works better than ever before. This testing isn't just a routine check; it's a crucial step toward providing a tool that goes beyond what’s expected by everyone who depends on it for search and rescue missions.

Moving forward, we’ll use what we learn from these tests to make SAROP even stronger and more reliable. We’re really focused on making improvements continuously and adapting to new feedback to make sure SAROP is both easy to use and effective. The success of these tests is key to our commitment to help save lives through a dependable and efficient SAROP system. We're dedicated to making sure that SAROP stands out as a valuable resource in critical rescue operations all over the world.

This conclusion aims to connect more personally with the reader, emphasizing the ongoing commitment to enhancement and the direct impact of these efforts on real-world rescue operations.