# **Test Plan: User Management Application**

## **Application Overview**

The User Management Application consists of a React frontend communicating with a Node.js backend API using authentication and JSON file-based storage. The application was created by Claude (AI tool) as I didn't manage to find a suitable application to test.

## **Testing Scope**

### **UI Testing**

* Admin login functionality
* User creation, editing, and deletion capabilities

### **API Testing**

* User retrieval, creation, updating, and deletion through API endpoints

## **Test Coverage Areas**

### **Frontend**

* **Functional Testing**: Core application functionality
* **Security Testing**: Authentication mechanisms

### **Backend**

* **Functional Testing**: API endpoint functionality
* **Schema Testing**: Data structure validation
* **Security Testing**: Authentication and authorization controls

## **Testing Tools**

### **Playwright**

**Reason**: Provides fast execution, cross-browser testing support, reliable auto-waiting features and easy to develop.

### **REST Assured**

**Reason**: Relied on previous experience with the framework, offers ease of use, and is one of the robust and popular frameworks for REST API testing.

### **Postman**

**Reason**: For quicker validation of CRUD operations for validation.

## **Setup and Execution Instructions**

### **Initial Setup**

1. Extract the compressed file to your desired location
2. Review the folder structure:
   * **Backend**: Contains server.js, package.json, and auto-generated files
   * **Frontend**: Contains Login.js, UserApp.js, and supporting application files
   * **Tests**:
     + **UI**: Configuration files, test data, and Playwright tests
     + **API**: Constants, utilities, models, helpers, test data, configuration files, and REST Assured tests

### **Prerequisites**

* Java 8 or higher
* Maven
* Node.js (latest version)
* Dependencies as specified in pom.xml

### **Backend Server Setup**

Navigate to the backend directory and execute:

npm install

npm run dev

The server will start at http://localhost:5000

### **Frontend Server Setup**

Navigate to the frontend directory and execute:

npm install

npm start

The application will launch at http://localhost:3000

### **Test Execution**

Navigate to the tests directory and run:

# Execute all tests

mvn clean test

# Execute specific test class

mvn test -Dtest="ClassName"

# Example: mvn test -Dtest=TestGetUser.java

## **Limitations**

### **Test Coverage Constraints**

Due to time limitations, the following was covered:

* **Frontend**: Positive scenarios
* **Backend**: Both positive and negative scenarios

### **Validation Gaps**

The current UI and API implementation lacks field-level validation. For example, the email field in UI/API does not enforce the mandatory presence of "@" followed by "." format validation.

### **Data Storage**

Due to time constraints, JSON file-based storage was selected over database implementation for this project's scope.

## **Additional Features**

### **CI/CD Pipeline**

While time constraints prevented the implementation of a GitHub Actions CI pipeline, the testing framework supports single-command execution of all tests.

### **Reporting and Documentation**

* **Allure Reports**: Results are stored in tests/allure-results and can be viewed using:  
   allure serve tests/allure-results
* **Screenshot Capture**: Playwright automatically captures screenshots after each action, saving them to the targets/screenshots folder for debugging and documentation purposes.