CULater – Test Plan

Table of contents

1. Scope and Objectives

1.1 Scope:

This test plan focuses on ensuring the functionality, performance, and security of the CULater web application. The following areas will be tested:

* User authentication (login, logout, signup)
* Group management (creation, editing, leaving groups)
* Member management (viewing members)
* Task management (not fully covered in provided tests)
* API endpoints functionality
* Error handling and validation
* Integration between frontend and middleware

1.2 Out of Scope:

* Cross-browser compatibility testing
* Mobile responsiveness testing
* Localization testing
* Third-party integrations beyond basic authentication

1.3 Objectives:

* Verify that all functional requirements from the SRS are properly implemented
* Ensure the application handles errors gracefully
* Validate that frontend components interact correctly with backend APIs
* Confirm that security measures (authentication, authorization) are properly implemented
* Ensure data integrity across frontend and backend

2. Test Cases and Scenarios

2.1 Example Test Cases for Functional Requirements:

Authentication:

1. **Login Functionality**
   - Steps: Enter valid credentials and submit
   - Expected Result: User is authenticated and redirected to dashboard
   - Test File: `Login.test.jsx`

- Related Requirement: FR01

2. **Login with Invalid Credentials**
   - Steps: Enter invalid credentials and submit
   - Expected Result: Error message is displayed, user stays on login page
   - Test File: `Login.test.jsx`

3. **Logout Functionality**
   - Steps: Click logout button while authenticated
   - Expected Result: Session is terminated, user is redirected to login page
   - Test File: `Logout.test.jsx`

4. **User Registration**
   - Steps: Fill out signup form with valid information and submit
   - Expected Result: New user account is created
   - Test File: `SignUp.test.jsx`
   - Related Requirement: FR01, FR02

Group Management:

5. **Group Creation**
   - Steps: Fill out group creation form and submit
   - Expected Result: New group appears in group list
   - Test File: `Group.test.jsx`
   - Related Requirement: FR04

6. **Group Editing**
   - Steps: Edit group details and save changes
   - Expected Result: Group details are updated in the UI
   - Test File: `Group.test.jsx`

7. **Leaving a Group**
   - Steps: Click "Leave" button on a group
   - Expected Result: Group is removed from user's group list
   - Test File: `Group.test.jsx`

8. **Viewing Group Members**
   - Steps: Click "Members" button on a group
   - Expected Result: List of group members is displayed
   - Test File: `Group.test.jsx`
   - Related Requirement: FR08

API Testing:

Authentication API Tests

1. **Successful Login with Valid Credentials**
   - **Endpoint**: `POST /api/auth/login`
   - **Test Case**: Verify system returns valid JWT token for correct credentials
   - **Input**: Valid email and password
   - **Expected Response**:
     - Status: 200 OK
     - Body: Contains `access_token` and user details
     - Token should be verifiable and contain correct claims

2. **Login with Invalid Password**
   - **Test Case**: Verify system rejects incorrect passwords
   - **Input**: Valid email but wrong password
   - **Expected Response**:
     - Status: 401 Unauthorized
     - Body: Clear error message about invalid credentials

3. **Login with Nonexistent Email**
   - **Test Case**: Verify system handles unknown users properly
   - **Input**: Email not in database
   - **Expected Response**:
     - Status: 401 Unauthorized
     - Body: Generic error message (security best practice)

4. **Login with Malformed Request**
   - **Test Case**: Verify validation of request format
   - **Input**: Missing email or password field
   - **Expected Response**:
     - Status: 400 Bad Request
     - Body: Specific validation error messages

User Registration API Tests

5. **Successful User Registration**
   - **Endpoint**: `POST /api/users/signup`
   - **Test Case**: Verify new user creation with valid data

- **Input**: Valid email, strong password, and unused license key
- **Expected Response**:
  - Status: 201 Created
  - Body: Contains created user email and timestamp
  - Verify user exists in database with hashed password

6. **Registration with Existing Email**
- **Test Case**: Prevent duplicate account creation
- **Input**: Email already registered in system
- **Expected Response**:
  - Status: 400 Bad Request
  - Body: Clear error about email already in use

7. **Registration with Invalid License Key**
- **Test Case**: Enforce license key validation
- **Input**: Nonexistent or malformed license key
- **Expected Response**:
  - Status: 400 Bad Request
  - Body: Specific error about invalid license

8. **Registration with Weak Password**
- **Test Case**: Enforce password policy
- **Input**: Password that doesn't meet complexity requirements
- **Expected Response**:
  - Status: 400 Bad Request
  - Body: Detailed password requirements error

Security-Focused Tests

9. **Token Expiration Verification**
- **Test Case**: Verify tokens expire as configured
- **Steps**:
  1. Obtain valid token
  2. Wait until after expiration time
  3. Attempt to use token
- **Expected Response**:
  - Status: 401 Unauthorized
  - Body: Error about expired token

10. **Brute Force Protection**

- **Test Case**: Verify rate limiting on auth endpoints
- **Steps**:
  1. Send multiple failed login attempts rapidly
  2. Verify system throttles requests
- **Expected Response**:
  - After threshold: 429 Too Many Requests
  - Clear message about rate limiting

2.2 Example Test Cases for Non-Functional Requirements

Performance Testing:

1. **API Response Time**
   - Objective: Verify API endpoints respond within acceptable time limits
   - Measurement: Response time < 500ms for all critical endpoints

Security Testing:

2. **Authentication Token Validation**
   - Objective: Verify tokens are properly validated and revoked
   - Test File: `test_auth.py`
   - Related Requirement: SSR1, SSR2

3. **Password Policy Enforcement**
   - Objective: Verify weak passwords are rejected
   - Test File: `test_user.py`
   - Related Requirement: SSR1

Reliability Testing:

4. **Error Handling**
   - Objective: Verify application handles errors gracefully
   - Covered in all test files checking error cases

3. Resource Allocation

3.1 Team Roles and Responsibilities:

Role	Responsibilities	Assigned To
Test Lead	Oversees testing process, creates test plan	Simon

Frontend Tester	Executes frontend tests, reports issues	Addison
Backend Tester	Executes API tests, validates business logic	Omer, Shen, Kai
Developer	Fixes identified issues, writes unit tests	Omer, Shen, Kai

3.2 Tools and Software:

- Jest/Vitest for frontend unit testing
- Pytest for backend testing
- React Testing Library for component testing
- Axios mocking for API interaction tests
- PostgreSQL for test database
- Redis for session management testing

3.3 Testing Environments:

1. **Development Environment**
   - Local machines with Node.js and Python
   - Used for unit and component testing

2. **Staging Environment**
   - Mirrors production setup
   - Used for manual integration testing

3.4 Time Allocation:

Activity	**Time(%)**
Test Planning	15%
Test Case Creation	20%
Test Execution	40%
Bug Reporting/Retesting	20%

Documentation	5%

4. Testing Approach

4.1 Types of Testing:

- **Unit Testing**: Individual components and functions (e.g., `Login.test.jsx`, `test_auth.py`)
- **Integration Testing**: Interaction between frontend and backend (e.g., API calls in React components)
- **System Testing**: End-to-end user flows (covered by component tests)
- **Regression Testing**: Re-running tests after changes

4.2 Methodologies:

- Test-Driven Development (TDD) for critical components
- Behavior-Driven Development (BDD) for user flows
- Manual testing for UI/UX validation
- Automated testing for regression and unit tests

5. Timeline and Schedule

5.1 Agile Model (2-week sprint cycle):

Day	**Activities**
1	Sprint planning, test case review
2-3	Write new test cases for sprint features
4-7	Execute test cases, report bugs
8-9	Regression testing, bug verification
10	Sprint review, test summary

6. Risk Assessment and Mitigation

6.1 Potential Risks:

1. **Incomplete Test Coverage**
   - Mitigation: Regular test case reviews, requirement traceability matrix

2. **Environment Configuration Issues**
   - Mitigation: Use containerized environments, documented setup procedures

3. **Time Constraints**
   - Mitigation: Prioritize critical path testing, automate repetitive tests

7. Success Criteria

- All critical test cases pass
- No high-priority bugs remain open
- All functional requirements from SRS are verified
- Code coverage > 80% for critical components
- Positive feedback from user acceptance testing

8. Reporting Requirements

8.1 Documentation:

- Test case execution reports
- Bug reports with reproduction steps
- Code coverage reports
- Test summary at end of each sprint

8.2 Communication:

- Daily standups for test progress
- Bug triage meetings with developers

- Sprint review presentations

This test plan provides a structured approach to ensure CULater is thoroughly tested and meets all requirements before release. The provided test files demonstrate good coverage of authentication and group management functionality, which should be expanded to cover all system features.