WeTravel - Testing Document

Group D3; May 7, 2024

Version: 1.0

Shuyang Song 1155173859, Department of Statistics
Yanze Yang 1155210986, Department of Computer Science and Engineering
Sriroth Poonyapat 1155205059, Department of Computer Science and Engineering
Ziqin Wei 1155173761, Department of Computer Science and Engineering
Thuwanontha Kittiphan 1155190410, Department of Computer Science and Engineering

Contents

1 Version History

Version	Modified by	Date	Comments
0.1	All members	May 1, 2025	Initial draft of Test Plan outline
0.2	All members	May 5, 2025	Expanded Representative Test Cases and merged team input
1.0	All members	May 7, 2025	Final review and formatting adjustments

2 Testing Plan

This section first presents the components tested to validate core functionality, security, and data integrity in the local notebook-based development environment. Then, the exclusions, which are out-of-scope features, are introduced.

2.1 Tested Components

- User Authentication
  - Test account creation, login, and session management.
- Trip Posting & Management (CRUD Operations)
  - Test data injection/display, image uploads, and public feed filtering.
- AI-Driven Recommendations
  - Test OpenAI API integration, prompt formatting, and response parsing.
- Expense Tracking
  - Test expense logging.
- Social Interactions
  - Test comment/like functionality and user references.
- Data Integration (MongoDB)
  - Test data injection and correct references (e.g., between users, posts, and recommendations).
- Security (Encryption)
  - Test encryption of stored data (e.g., passwords).
- Local Deployment Performance
  - Test responsiveness in the local laptop-based frontend-backend environment.

2.2 Excluded Components

- Cross-Platform Compatibility
  - Reasons:
    * Testing is limited to the local laptop-based development environment (frontend and backend running locally).
    * The application is not deployed to a live server, making it technically infeasible to validate compatibility across platforms (e.g., mobile devices, different browsers).
    * Focus remains on core functionality validation within the controlled local setup, as cross-platform testing requires external deployment and device/browser diversity beyond the current scope.

3 Representative Test Cases

3.1 Test Case Design Approach

Test cases are designed to evaluate key functionalities using:

- Typical Scenarios: Validate standard user workflows.
- Exceptional Scenarios: Test error conditions (e.g., invalid data).

3.2 Test Cases

3.2.1 User Sign-Up with Valid Credentials and Password Encryption Verification

- Input:
  - Username: example_user
  - Password: example_user
- Expected result:
  - The account is created successfully.
  - The encrypted password is stored in MongoDB.
- Reason for design:
  - To validate the function of secure user registration and password encryption.
- Testing approach:
  - Submit the sign-up form via the frontend.

  - Verify that MongoDB stores the user information with the encrypted password.
  - Verify whether the registered account can log in successfully

3.2.2 User Login with Invalid Credentials

- Input:
  - Email: user@example.com
  - Password: WrongPassword
- Expected result:
  - User cannot log in clicking submit button.
- Reason for design:
  - To test the authentication failure handling.
- Testing approach:
  - Attempt login via frontend.
  - Verify the backend error returned.

3.2.3 Create Trip Post with Valid Data and Image

- Input:
  - Location: ”Paris”
  - Text: ”Amazing cultural experience in Paris!”,
  - Dates: ”2024-10-01 to 2024-10-07”
  - Budget: $1500
  - Image: a 1MB JPEG file.
  - Activities: ”Museum tours, dining”
- Expected result:
  - Post appears in the public feed with correct details and image.
  - Related data is stored in the MongoDB database with the correct reference.
- Reason for design:
  - To validate the trip posting functionality under normal conditions.

- Testing approach:
  - If a user doesn't log in, it will redirect to the login page.
  - Post the trip after logging into an account.
  - Verify that MongoDB stores the trip data with the image as a buffer type.
  - Check the frontend feed displays the post correctly.

3.2.4 Upload Oversized Image

- Input:
  - Location: "Paris"
  - Text: "Amazing cultural experience in Paris!",
  - Dates: "2024-10-01 to 2024-10-07"
  - Budget: $1500
  - Image: a 7MB JPEG file.
  - Activities: "Museum tours, dining"
- Expected result:
  - Error: "Image size is too large."
- Reason for design:
  - To test file size validation and error handling.
- Testing approach:
  - Attempt to post a trip with a 7MB image via the frontend.
  - Verify the system rejects the request and returns an error.

3.2.5 Empty Required Fields in Trip Posting

- Input
  - Text: "Amazing cultural experience in Paris!",
  - Dates: "2024-10-01 to 2024-10-07"
  - Image: a 1MB JPEG file.
  - Activities: "Museum tours, dining"

  - Omit location and budget.

- Expected result:
  - Error: ”Missing required fields”
- Reason for design:
  - To ensure required field validation.
- Testing approach:
  - Attempt to post a trip with empty required fields via the frontend.
  - Verify the system rejects the request and returns an error.

3.2.6 AI-Driven Recommendation Generation

- Input:
  - Location: ”Barcelona”
  - Budget: $1800
  - Duration: 5 days
  - Activity: football
- Expected result:
  - OpenAI API successfully returns reasonable travel recommendations.
  - Suggestions are parsed and stored in a structured manner.
- Reason for design:
  - To test OpenAI API integration and proper parsing/storage of recommendations.
- Testing approach:
  - Submit preferences via the frontend.
  - Capture API request logs to validate prompt formatting.
  - Check that the recommendations can be correctly displayed in the frontend and stored in MongoDB in a structured manner

3.2.7 Like and Comment on a Trip Post

- Input:

  - Comment: ”This looks amazing! A great trip!”
  - Like: Click the ”Like” button.
- Expected result:
  - Comment and liked appear under the post.
- Reason for design:
  - To test user interaction features.
- Testing approach:
  - Log in and click on a trip post.
  - Like the post, and add a comment.
  - Check MongoDB for updated like count and comment references.

3.2.8 Group Chat Functionality

- Input:
  - Message: ”Let’s meet at the train station at 9 AM!”
- Expected result:
  - Message appears in the group chat for all members.
- Reason for design:
  - Tests real-time communication within a group.
- Testing approach:
  - User A and User B join the group.
  - Send a message from User A’s account.
  - Check if User B’s chat interface updates.
  - Try to delete some sent messages.
  - Verify MongoDB stores/deletes the message with correct group/user references.

3.2.9 Add and Delete Expense with Default Payer

- Input:
  - Description: test expense

  - Amount: $200
  - Category: Food
  - Date: 2025/5/8
- Expected result:
  - Payer is automatically set to the user who added the expense.
  - The expense can be successfully displayed in the group page and stored in MongoDB
  - If expense is deleted, it shouldn't be seen in the group page anymore.
- Reason for design:
  - Validates expense function and default payer assignment.
- Testing approach:
  - Log in as User A and add an expense in one group page.
  - Check MongoDB and group page display updates.
  - Log in as User A and delete an expense in one group page.
  - Check MongoDB and group page display updates.

3.2.10 Testing Incorrect License

- Input:
  - An incorrect license before logged in.
- Expected result:
  - Will prompt to input license again.
- Reason for design:
  - Tests license management
- Testing approach:
  - When prompted to input a license, input "1" which is incorrect

4 Resources & Schedule

This section defines the human resources, their responsibilities, and the testing schedule.

4.1 Roles & Responsibilities

Role	Name(s)	Responsibilities
Test Lead (Plan author & integrator)	Sriroth Poonyapat	• Produce and maintain the Test Plan document • Coordinate integration and system tests • Final sign-off on test completion
Front-end Tester	Yanze Yang, Sriroth Poonyapat	• Execute manual UI and API tests for front-end components • Peer-review front-end test results
Back-end Tester	Shuyang Song, Ziqin Wei, Thuwanontha Kittiphan	• Execute manual API tests in Postman • Verify database state in MongoDB Compass • Peer-review back-end test results

4.2 Testing Schedule

Phase	Timeline	Activities
Unit Testing During Implementation	Weeks 1–7	• Developers write and run unit tests as features are completed • Test Lead reviews coverage and results
Integration & System Testing	Weeks 8-10	• Execute all representative test cases in Postman • Verify end-to-end flows (Auth, Posts, Recommendations, Expenses, Groups, Licenses) • Log defects and coordinate fixes
Regression & Final Sign-off	Week 11-13	• Re-execute all test cases to confirm fixes • Produce Test Execution Report and defect summary • Test Lead performs final sign-off

5 Risk Assessment & Mitigation

This section identifies potential risks to our testing process and the strategies we will employ to mitigate them.

5.1 Risk Identification & Assessment

We have identified the following key risks, rated by likelihood and impact:

- **Missing Environment Variables:** (Likelihood: Medium; Impact: High): If `OPENAI_API_KEY`, `MONGODB_URI` or `JWT_SECRET` are absent, API endpoints will fail to start or authenticate.

- **Invalid Fixture IDs in Dummy Data:** (Likelihood: Medium; Impact: Medium) Malformed ObjectId values (for example, '"7"') cause the seeding endpoints to error out.
- **Image Upload Edge Cases:** (Likelihood: Low; Impact: Medium) Oversized files or unsupported formats may be rejected or cause runtime errors in the Multer middleware.
- **Authentication Token Expiry During Testing:** (Likelihood: Low; Impact: Medium) Short-lived JWTs could expire mid-workflow, resulting in unexpected 401 Unauthorized responses.
- **API Contract Changes:** (Likelihood: Low; Impact: High) Updates to request or response schemas risk breaking existing test cases and test scripts.

5.2 Mitigation Strategies

Risk	Impact	Mitigation Actions
Missing Environment Variables	API and DB tests fail	• Provide `.env.example` with placeholders • Document required vars in Test Plan • Fallback to stubs/mocks for OpenAI calls when key missing
Invalid Fixture IDs in Dummy Data	Seeder routes crash, blocking test setup	• Add `Types.ObjectId.isValid(...)` guards in seeders • Skip or log invalid entries rather than throwing • Validate JSON fixtures before running tests
Image Upload Edge Cases	Unexpected errors on file handling	• Enforce size/type limits in Multer config • Include test cases for boundary sizes (0 bytes, max limit) • Document allowed formats and sizes
Auth Token Expiry During Testing	401 errors mid-test	• Extend JWT TTL in development environment • Include token-refresh step in Postman pre-scripts • Log out/in as part of long-running tests
API Contract Changes	Broken test cases, false failures	• Version Postman collection alongside API changes • Include schema validation in tests (e.g. JSON Schema) • Update Test Plan and cases immediately after any API update

6 Reporting & Deliverables

This section describes the artifacts and reports that will be produced during and after testing.

6.1 Deliverables

6.1.1 Postman Collection

- All API requests for Auth, Users, Posts, Recommendations, Expenses, Trips, Groups, and Licenses
- Pre-request scripts to handle authentication and token refresh

6.1.2 Test Execution Log

- A table (spreadsheet or Markdown) listing:
  - Test case name/ID and description
  - Date executed and tester name
  - Pass/Fail status
- Attachments (screenshots, response dumps) for any failed tests

6.1.3 Test Summary Report

- Charts or tables summarizing results by module or feature area
- Key observations and recommendations for future testing cycles

6.1.4 Final Sign-Off

- Formal confirmation (email or meeting minutes) that all critical and high-severity defects are resolved
- Approval signatures from the Test Lead (Sriroth Poonyapat) and Project Lead