CS673 Software Engineering

Team 3 - PlanningJam 

Software Test Document

Team Member	Role(s)	Signature	Date
Jason	QA	*JasonLee*	09/21/2025
David Metraux	Team Leader	*David Metraux*	*9/21/2025*
Ashley Sachdeva	Requirements Leader	*Ashley Sachdeva*	9/21/2025
Donjay Barit	Configuration Leader	*Donjay Barit*	9/21/2025
Haolin	Design and Implementation Leader	*Haolin Yang*	*9/21/2025*

Revision history

Version	**Author**	**Date**	**Change**
0.0.0	**Ashley Sachdeva**	**09/21/2025**	**Added additional frontend testing steps**
1.0.0	**Ashley Sachdeva**	**10/5/2025**	**Added automated test section and extended frontend**

			manual testing section
1.0.1	**David Metraux**	**10/6/2025**	**Added manual tests for friends page**
1.0.2	**Jason Lee**	**10/5/2025**	**Added to testing summary, backend automated testing report, and testing metrics**
1.0.3	**Donjay Barit**	**10/6/2025**	**Added manual test for plans api query**
1.0.4	**Haolin Yang**	**10/9/2025**	**Added manual tests for user and rsvp api**
1.0.5	**David Metraux**	**10/11/2025**	**Added manual tests for user pages, profile, and editing**

- # Testing Summary

  In this section, you will summarize what was tested, who is involved in testing, when to test, testing techniques used, and testing result. You may have the following tests
  - Unit Testing
  - Integration testing
  - System Testing
  - Acceptance Testing
  - Regression Testing

    On elements in the frontend, you often have to check if other

What was tested

- Authentication & User APIs (registration, login, profile, user listing)
- Friends API (send/respond/remove friend requests, reciprocal auto-accept logic)
- Plans API (CRUD operations for plans in MongoDB)
- Frontend authentication flows (login, signup, logout, token expiration)
- Frontend plans functionality (create, edit, delete)
- UI responsiveness (cross-browser and viewport testing)

Who is involved

- Backend testing: Team members responsible for Django REST Framework API
- Frontend testing: Team members responsible for React + Vite application

- Integration/system tests: Performed jointly via Postman and browser

When testing occurs

- Unit and integration tests: Continuous, as features are developed
- System/acceptance tests: Before sprint demos and release candidates
- Regression tests: Run after significant refactors, migrations, or merges

Testing techniques used

- Unit Testing: Django pytest for backend, Vitest + React Testing Library for frontend
- Integration Testing: Postman API collection tests, end-to-end flows
- System Testing: Browser/manual exploratory testing
- Acceptance Testing: Full workflow scenarios (signup → login → add friend → create plan)
- Regression Testing: Automated test suites rerun on feature updates

Test Environment

- Backend: Django, DRF, SimpleJWT, django-mongodb-backend
- Database: PostgreSQL (users/friends), MongoDB (plans collection)
- Frontend: React + Vite + TypeScript
- Tools: Postman, Vitest, pytest, React Testing Library

Manual Testing Report

In this section, you will give a detailed description of each manual test case performed and the result. If this is a previous You shall list what are existing tests developed in the previous semester and what are new tests developed currently.

Here is a sample template that can be used for each test case. For system tests or acceptance tests, you may also include some screenshots.

- Test case ID, name

- New or old:
- Test items: (what do you test)
- Test priority (high/medium/low)
- Dependencies (to other test case/requirement if any):
- Preconditions: (if any)
- input data:
- Test steps:
- Postconditions:
- Expected output:
- Actual output:
- Pass or Fail:
- Bug id/link: (this should link to your github issue id)
- Additional notes:

(You can use an additional spreadsheet for this section as well)

- Test case 1: **Browser testing**
- Test items: Using Chrome or other devtools, verify that items are displayed correctly on different browsers and on different viewpoint sizes..
- Test priority (high)
- Dependencies (to other test case/requirement if any): N/A
- Preconditions: (if any)
- input data: Depends on the item
- Test steps:

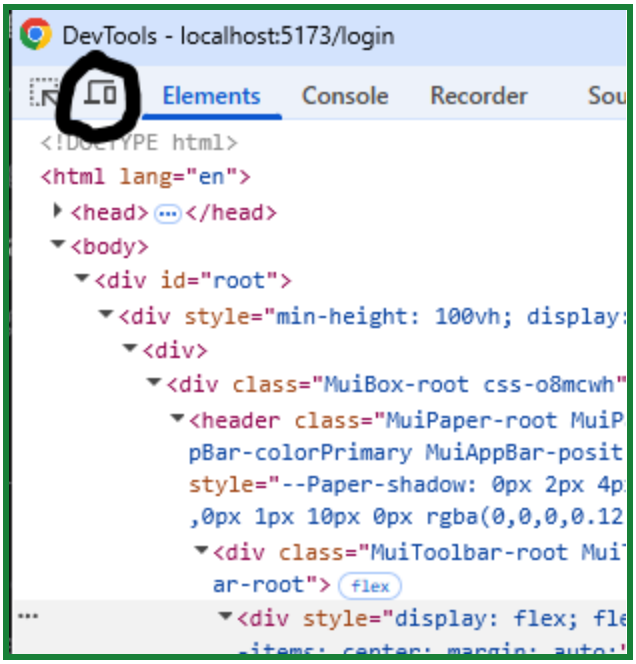
  1. Load location where element will be viewable on a browser

  2. Determine it is still can be viewed and interacted with at a variety of viewpoint sizes (via toggle device toolbar)

  3. Attempt again on a different browser
- Postconditions:
- Expected output:

  The element to look as desired
- Actual output: N/A
- Pass or Fail: N/A
- Bug id/link: N/A
- Additional notes:

  This is how I can check various viewport sizes



- Test case 2: **Login**
- Test items: Login flow for user
- Test priority: High
- Dependencies (to other test case/requirement if any): Must have an existing account
- Preconditions: (if any): user is registered
- input data: email: username: asach8888 pwd: mimic123
- Test steps:
    - Navigate to login
    - Enter username
    - Enter password
    - Click login
    - Clicking signup will redirect you to the signup page
- Postconditions: Successfully login, redirect to home page
- Expected output:
- Actual output:
- Pass or Fail:
- Bug id/link: (this should link to your github issue id)
- Additional notes:
    - If a user enters incorrect username/password correct error message is shown
    - If a user tries to access login/signup from there they get redirected to homepage until they logout

- Test case 3: **Signup**
- Test items: Signup flow for user
- Test priority: High
- Dependencies (to other test case/requirement if any): none
- Preconditions: (if any):
- input data: email: username: xxx, email: xxx@gmail.com, pwd: mimic12345, confirm pws
- Test steps:
    - Navigate to signup
    - Fill out form
    - Click sign up
    - Clicking Login will redirect you to login
- Postconditions: Successfully registered and logged in, redirect to home page
- Expected output:
- Actual output:
- Pass or Fail:
- Bug id/link: (this should link to your github issue id)
- Additional notes:
    - Filling out an existing username will fail validation
    - Incorrect email format will fail validation
    - Passwords being too short, common, or all numerical will fail validation
    - If a user tries to access login/signup from there they get redirected to homepage until they logout

- Test case 4: **Logout**
- Test items: Logout flow for user
- Test priority: medium
- Dependencies (to other test case/requirement if any): none
- Preconditions: (if any):
- input data: None
- Test steps:
    - Navigate to /logout
- Postconditions: Successfully logged out, redirected to login page
- Expected output:
- Actual output:
- Pass or Fail:

- Bug id/link: (this should link to your github issue id)
- Additional notes:
    - Navigating to home page redirects them back to login

- Test case 5: **Token Expiration**
- Test items: Expired token for user authentication
- Test priority: high
- Dependencies (to other test case/requirement if any): Logged in user
- Preconditions: (if any): User is logged in for 60 minutes
- input data: None
- Test steps:
    - If the user is logged in for 60+ minutes on the application they will be logged out
    - Login for 1 hour or change JWT in settings.py to something shorter like 10 seconds
    - Refresh screen, should be redirected to login
- Postconditions: Successfully logged out, redirected to login page
- Expected output:
- Actual output:
- Pass or Fail:
- Bug id/link: (this should link to your github issue id)
- Additional notes:
    - Navigating to home page redirects them back to login

Test Case 6: **Add Plan**

- Test items: Add new plan functionality
- Test priority: High
- Dependencies (to other test case/requirement if any): User must be logged in
- Preconditions (if any): Valid authenticated session, user on /plans/add
- Input data: plan title, description, location name, address, city, state, zipcode, start and end time
- Test steps:
    - Click “Add Plan” button on home page
    - Fill in only some of the data
    - Click “Save”
    - Ensure error message shows to fill out other input data
    - Click cancel button
    - Ensure it redirects to home page
    - Fill rest of input data

    - Click Save button
- Postconditions:
    - New plan is created and visible on the home page
- Expected output: The new plan appears immediately in the plans list with correct information
- Actual output: N/A
- Pass or Fail: N/A
- Bug id/link: N/A
- Additional notes: UI refreshes automatically after adding a plan; no manual reload needed.

Test Case 7: **Edit Plan**

- Test items: Edit new plan functionality
- Test priority: High
- Dependencies (to other test case/requirement if any): User must be logged in
- Preconditions (if any): Valid authenticated session, user has plan already created
- Input data: plan title, description, location name, address, city, state, zipcode, start and end time
- Test steps:
    - Click “Edit Plan” button on plan created by user
    - The plan inputs are pre-filled
    - Click cancel button
    - Ensure it redirects to home page
    - Edit some input data
    - Click Save button
- Postconditions:
    - Edited plan is created and visible on the home page
- Expected output: The edited plan appears immediately in the plans list with correct information
- Actual output: N/A
- Pass or Fail: N/A
- Bug id/link: N/A
- Additional notes: UI refreshes automatically after editing a plan; no manual reload needed.

Test Case 8: **Delete Plan**

- Test items: delete new plan functionality
- Test priority: High
- Dependencies (to other test case/requirement if any): User must be logged in

- Preconditions (if any): Valid authenticated session, user has plan already created
- Input data: none
- Test steps:
    - Click “Delete Plan” button on plan created by user
    - Ensure a confirm deletion modal shows
    - Click cancel button
    - Ensure modal is closed
    - Click delete plan button on plan again
    - Click confirm delete button
- Postconditions:
    - Plan is no longer visible on home page
- Expected output: The plan disappears immediately in the plans list
- Actual output: N/A
- Pass or Fail: N/A
- Bug id/link: N/A
- Additional notes: UI refreshes automatically after deleting a plan; no manual reload needed.

Test Case 9: **Send Friend Request**

- Test items: Friend Request initiation
- Test priority: High
- Dependencies (to other test case/requirement if any): Two registered users
- Preconditions (if any): 1st user logged in
- Input data: none
- Test steps:
    - POST /api/friends/request/<bob_id>/
- Postconditions:
    - A friend request is sent
- Expected output: Success 201 Created status = pending
- Actual output: N/A
- Pass or Fail: N/A
- Bug id/link: N/A
- Additional notes:

Test Case 10: **Respond to Friend Request**

- Test items: Accepting Friend Request
- Test priority: High
- Dependencies (to other test case/requirement if any): Pending request exists
- Preconditions (if any): 2nd user logged in
- Input data: none
- Test steps:
    - POST /api/friends/respond/<request_id>/ with {"action":"accept"}
- Postconditions:
    - A friendship is bloomed
- Expected output: 200 OK, status = accepted
- Actual output: N/A
- Pass or Fail: N/A
- Bug id/link: N/A
- Additional notes:

Test Case 11: **Remove Friend**

- Test items: Friend Removal
- Test priority: High
- Dependencies (to other test case/requirement if any):
- Preconditions (if any): A souring friendship exists
- Input data: none
- Test steps:
    - DELETE /api/friends/remove/<request_id>/
- Postconditions:
    - A broken heart
- Expected output: 204 No Content
- Actual output: N/A
- Pass or Fail: N/A
- Bug id/link: N/A
- Additional notes:

Test Case 12: **Friends page**

- Test items: Friend functions
- Test priority: High

- Dependencies (to other test case/requirement if any):
- Preconditions (if any): At least two users in the program
- Input data: Two users
- Test steps:
    - Sign in two users, each in a different browser
    - have a user send a friend request to another user
    - Have the receiving user delete the friend request
    - Have the first user send a friend request again
    - Have the receiving user accept the friend request
    - Have either user delete the other as a friend.
- Postconditions:
- Expected output: The UI to change smoothly
- Actual output: So far, I often need to click twice for the UI to update
- Pass or Fail: Sort of fail
- Bug id/link: N/A
- Additional notes:

Test Case 13: **Plans API filter query**

- Test items: Query get plans api
- Test priority: High
- Dependencies (to other test case/requirement if any):
- Preconditions (if any): Plans created by two or more users and having a friends relationship with each other or none
- Input data: Set optional values for query filters: start time, end time, friends
- Test steps:
    - Send a GET request to /api/plans?start_time=<stime>&end_time=<etime>&friends=1
- Postconditions: None
- Expected output: A list of plans that is created by the user or their friends with a date range
- Actual output: The list of expected plans for the given user and their friends between the date ranges
- Pass or Fail: Pass
- Bug id/link: N/A
- Additional notes:

Test case 14: **User Registration Success**

- Test items: Successful user registration with all profile fields
- Test priority: high

- Dependencies: None
- Preconditions: Valid API client available
- Input data: {"username": "newuser", "email": "newuser@example.com", "first_name": "New", "last_name": "User", "date_of_birth": "1990-01-01", "bio": "This is my bio", "password": "testpass123", "password_confirm": "testpass123"}
- Test steps:
    - Send POST request to /api/register/ with complete user data
    - Verify response status is 201 CREATED
    - Verify response contains message, access, and refresh tokens
    - Verify user was created in database with correct fields
    - Verify UserProfile was created with correct date_of_birth and bio
- Postconditions: User and profile created successfully
- Expected output: HTTP 201, success message, JWT tokens, user and profile in database
- Actual output:
- Pass or Fail:
- Bug id/link:
- Additional notes: Tests complete registration flow including profile creation

Test case 15: **User Registration Minimal Data**

- Test items: User registration with only required fields
- Test priority: high
- Dependencies: None
- Preconditions: Valid API client available
- Input data: {"username": "minimaluser", "email": "minimal@example.com", "password": "testpass123", "password_confirm": "testpass123"}
- Test steps:
    - Send POST request to /api/register/ with minimal required data
    - Verify response status is 201 CREATED
    - Verify user was created in database
    - Verify UserProfile was created with default values (None for date_of_birth, empty string for bio)
- Postconditions: User and profile created with defaults
- Expected output: HTTP 201, user created, profile with default values
- Actual output:
- Pass or Fail:

- Bug id/link:
- Additional notes: Tests registration with minimal required fields

Test case 16: **User Registration Password Mismatch**

- Test items: Registration validation with mismatched passwords
- Test priority: high
- Dependencies: None
- Preconditions: Valid API client available
- Input data: {"username": "testuser", "email": "test@example.com", "password": "testpass123", "password_confirm": "differentpass"}
- Test steps:
    - Send POST request to /api/register/ with mismatched passwords
    - Verify response status is 400 BAD REQUEST
    - Verify error message contains "Passwords don't match"
- Postconditions: User registration fails
- Expected output: HTTP 400, password mismatch error
- Actual output:
- Pass or Fail:
- Bug id/link:
- Additional notes: Tests password validation

Test case 17: **User Registration Duplicate Username**

- Test items: Registration validation with duplicate username
- Test priority: high
- Dependencies: Existing user with same username
- Preconditions: User with username "existinguser" already exists
- Input data: {"username": "existinguser", "email": "new@example.com", "password": "testpass123", "password_confirm": "testpass123"}
- Test steps:
    - Create existing user with username "existinguser"
    - Send POST request to /api/register/ with same username
    - Verify response status is 400 BAD REQUEST
    - Verify error response contains "username" field
- Postconditions: Registration fails due to duplicate username
- Expected output: HTTP 400, username validation error
- Actual output:

- Pass or Fail:
- Bug id/link:
- Additional notes: Tests username uniqueness validation

Test case 18: **User Registration Duplicate Email**

- Test items: Registration validation with duplicate email
- Test priority: high
- Dependencies: Existing user with same email
- Preconditions: User with email "existing@example.com" already exists
- Input data: {"username": "user2", "email": "existing@example.com", "password": "testpass123", "password_confirm": "testpass123"}
- Test steps:
    - Create existing user with email "existing@example.com"
    - Send POST request to /api/register/ with same email
    - Verify response status is 400 BAD REQUEST
    - Verify error response contains "email" field
- Postconditions: Registration fails due to duplicate email
- Expected output: HTTP 400, email validation error
- Actual output:
- Pass or Fail:
- Bug id/link:
- Additional notes: Tests email uniqueness validation

Test case 19: **User Registration Invalid Date**

- Test items: Registration validation with invalid date format
- Test priority: medium
- Dependencies: None
- Preconditions: Valid API client available
- Input data: {"username": "testuser", "email": "test@example.com", "date_of_birth": "invalid-date", "password": "testpass123", "password_confirm": "testpass123"}
- Test steps:
    - Send POST request to /api/register/ with invalid date format
    - Verify response status is 400 BAD REQUEST
- Postconditions: Registration fails due to invalid date
- Expected output: HTTP 400, date validation error
- Actual output:

- Pass or Fail:
- Bug id/link:
- Additional notes: Tests date format validation

Test case 20: **User Registration Bio Too Long**

- Test items: Registration validation with bio exceeding max length
- Test priority: medium
- Dependencies: None
- Preconditions: Valid API client available
- Input data: {"username": "testuser", "email": "test@example.com", "bio": "x" * 501, "password": "testpass123", "password_confirm": "testpass123"}
- Test steps:
    - Send POST request to /api/register/ with bio exceeding 500 characters
    - Verify response status is 400 BAD REQUEST
- Postconditions: Registration fails due to bio length validation
- Expected output: HTTP 400, bio length validation error
- Actual output:
- Pass or Fail:
- Bug id/link:
- Additional notes: Tests bio field length validation (500 character limit)

Test case 21: **Get User Profile Success**

- Test items: Successful profile retrieval for authenticated user
- Test priority: high
- Dependencies: Authenticated user with existing profile
- Preconditions: User is authenticated and has a UserProfile
- Input data: None
- Test steps:
    - Create UserProfile for authenticated user with date_of_birth and bio
    - Send GET request to /api/profile/
    - Verify response status is 200 OK
    - Verify response contains all user and profile fields
    - Verify date_of_birth and bio are correctly returned
- Postconditions: Profile data retrieved successfully
- Expected output: HTTP 200, complete user and profile data
- Actual output:

- Pass or Fail:
- Bug id/link:
- Additional notes: Tests profile retrieval functionality

Test case 22: **Get User Profile No Profile**

- Test items: Profile retrieval when user has no profile
- Test priority: medium
- Dependencies: Authenticated user without profile
- Preconditions: User is authenticated but has no UserProfile
- Input data: None
- Test steps:
    - Ensure user has no UserProfile
    - Send GET request to /api/profile/
    - Verify response status is 200 OK
    - Verify response contains user data with None/empty profile fields
- Postconditions: User data returned with default profile values
- Expected output: HTTP 200, user data with None/empty profile fields
- Actual output:
- Pass or Fail:
- Bug id/link:
- Additional notes: Tests graceful handling of missing profile

Test case 23: **Get User Profile Unauthorized**

- Test items: Profile retrieval without authentication
- Test priority: high
- Dependencies: None
- Preconditions: No authentication
- Input data: None
- Test steps:
    - Send GET request to /api/profile/ without authentication
    - Verify response status is 401 UNAUTHORIZED
- Postconditions: Access denied
- Expected output: HTTP 401, unauthorized error
- Actual output:
- Pass or Fail:
- Bug id/link:

- Additional notes: Tests authentication requirement

Test case 24: **Update User Profile Full Update**

- Test items: Complete profile update using PUT method
- Test priority: high
- Dependencies: Authenticated user with existing profile
- Preconditions: User is authenticated and has a UserProfile
- Input data: {"first_name": "Updated First", "last_name": "Updated Last", "date_of_birth": "1985-05-15", "bio": "Updated bio content"}
- Test steps:
    - Create initial UserProfile for user
    - Send PUT request to /api/profile/update/ with complete data
    - Verify response status is 200 OK
    - Verify response contains success message
    - Verify user fields were updated in database
    - Verify profile fields were updated in database
- Postconditions: Profile and user data updated successfully
- Expected output: HTTP 200, success message, updated data in database
- Actual output:
- Pass or Fail:
- Bug id/link:
- Additional notes: Tests complete profile update functionality

Test case 25: **Update User Profile Partial Update**

- Test items: Partial profile update using PATCH method
- Test priority: high
- Dependencies: Authenticated user with existing profile
- Preconditions: User is authenticated and has a UserProfile
- Input data: {"bio": "Only updating bio"}
- Test steps:
    - Create initial UserProfile with date_of_birth and bio
    - Send PATCH request to /api/profile/update/ with only bio field
    - Verify response status is 200 OK
    - Verify only bio was updated, date_of_birth remains unchanged
- Postconditions: Only specified fields updated
- Expected output: HTTP 200, only bio updated

- Actual output:
- Pass or Fail:
- Bug id/link:
- Additional notes: Tests partial update functionality

Test case 26: **Update User Profile Create Profile**

- Test items: Profile update when user has no existing profile
- Test priority: medium
- Dependencies: Authenticated user without profile
- Preconditions: User is authenticated but has no UserProfile
- Input data: {"first_name": "New First", "bio": "Creating new profile"}
- Test steps:
    - Ensure user has no UserProfile
    - Send PATCH request to /api/profile/update/ with profile data
    - Verify response status is 200 OK
    - Verify UserProfile was created with specified data
    - Verify user field was updated
- Postconditions: UserProfile created and user data updated
- Expected output: HTTP 200, profile created and user updated
- Actual output:
- Pass or Fail:
- Bug id/link:
- Additional notes: Tests automatic profile creation during update

Test case 27: **Update User Profile Invalid Date**

- Test items: Profile update validation with invalid date
- Test priority: medium
- Dependencies: Authenticated user
- Preconditions: User is authenticated
- Input data: {"date_of_birth": "invalid-date"}
- Test steps:
    - Send PATCH request to /api/profile/update/ with invalid date
    - Verify response status is 400 BAD REQUEST
- Postconditions: Update fails due to invalid date
- Expected output: HTTP 400, date validation error
- Actual output:

- Pass or Fail:
- Bug id/link:
- Additional notes: Tests date validation in profile updates

Test case 28: **Update User Profile Bio Too Long**

- Test items: Profile update validation with bio exceeding max length
- Test priority: medium
- Dependencies: Authenticated user
- Preconditions: User is authenticated
- Input data: {"bio": "x" * 501}
- Test steps:
    - Send PATCH request to /api/profile/update/ with bio exceeding 500 characters
    - Verify response status is 400 BAD REQUEST
- Postconditions: Update fails due to bio length validation
- Expected output: HTTP 400, bio length validation error
- Actual output:
- Pass or Fail:
- Bug id/link:
- Additional notes: Tests bio length validation in profile updates

Test case 29: **Update User Profile Unauthorized**

- Test items: Profile update without authentication
- Test priority: high
- Dependencies: None
- Preconditions: No authentication
- Input data: {"bio": "Unauthorized update"}
- Test steps:
    - Send PATCH request to /api/profile/update/ without authentication
    - Verify response status is 401 UNAUTHORIZED
- Postconditions: Access denied
- Expected output: HTTP 401, unauthorized error
- Actual output:
- Pass or Fail:
- Bug id/link:
- Additional notes: Tests authentication requirement for profile updates

Test case 30: **List Users Success**

- Test items: Successful user listing for authenticated user
- Test priority: high
- Dependencies: Multiple users with profiles
- Preconditions: Authenticated user and multiple other users exist
- Input data: None
- Test steps:
    - Create additional users with UserProfiles
    - Send GET request to /api/users/
    - Verify response status is 200 OK
    - Verify response contains all users
    - Verify bio fields are included in response
- Postconditions: All users listed with profile data
- Expected output: HTTP 200, list of all users with profile information
- Actual output:
- Pass or Fail:
- Bug id/link:
- Additional notes: Tests user listing functionality

Test case 31: **List Users With Profiles**

- Test items: User listing includes profile information
- Test priority: medium
- Dependencies: Users with different profile configurations
- Preconditions: Multiple users with varying profile completeness
- Input data: None
- Test steps:
    - Create users with different profile setups
    - Send GET request to /api/users/
    - Verify response status is 200 OK
    - Verify profile data is included for each user
    - Verify users without profiles show default values
- Postconditions: Users listed with appropriate profile data
- Expected output: HTTP 200, users with profile information
- Actual output:
- Pass or Fail:
- Bug id/link:

- Additional notes: Tests profile data inclusion in user listings

Test case 32: **List Users Unauthorized**

- Test items: User listing without authentication
- Test priority: high
- Dependencies: None
- Preconditions: No authentication
- Input data: None
- Test steps:
    - Send GET request to /api/users/ without authentication
    - Verify response status is 401 UNAUTHORIZED
- Postconditions: Access denied
- Expected output: HTTP 401, unauthorized error
- Actual output:
- Pass or Fail:
- Bug id/link:
- Additional notes: Tests authentication requirement for user listing

Test case 33: **Get User Success**

- Test items: Successful individual user retrieval
- Test priority: high
- Dependencies: Target user with profile
- Preconditions: Target user exists with UserProfile
- Input data: User ID
- Test steps:
    - Create target user with UserProfile
    - Send GET request to /api/users/{user_id}/
    - Verify response status is 200 OK
    - Verify response contains user data and profile information
    - Verify user ID matches requested ID
- Postconditions: Individual user data retrieved
- Expected output: HTTP 200, complete user and profile data
- Actual output:
- Pass or Fail:
- Bug id/link:

- Additional notes: Tests individual user retrieval

Test case 34: **Get User Not Found**

- Test items: Individual user retrieval with non-existent user
- Test priority: medium
- Dependencies: None
- Preconditions: Valid ObjectId format but non-existent user
- Input data: Valid ObjectId string for non-existent user
- Test steps:
    - Generate valid ObjectId for non-existent user
    - Send GET request to /api/users/{fake_id}/
    - Verify response status is 404 NOT FOUND
    - Verify error message contains "User not found"
- Postconditions: User not found error returned
- Expected output: HTTP 404, "User not found" error
- Actual output:
- Pass or Fail:
- Bug id/link:
- Additional notes: Tests handling of non-existent users

Test case 35: **Get User Unauthorized**

- Test items: Individual user retrieval without authentication
- Test priority: high
- Dependencies: None
- Preconditions: No authentication
- Input data: Any user ID
- Test steps:
    - Send GET request to /api/users/some-id/ without authentication
    - Verify response status is 401 UNAUTHORIZED
- Postconditions: Access denied
- Expected output: HTTP 401, unauthorized error
- Actual output:
- Pass or Fail:
- Bug id/link:
- Additional notes: Tests authentication requirement for individual user retrieval

Test case 36: **User Profile Creation On Registration**

- Test items: UserProfile automatically created during registration
- Test priority: high
- Dependencies: User registration functionality
- Preconditions: Valid API client available
- Input data: Complete registration data with profile fields
- Test steps:
    - Send POST request to /api/register/ with profile data
    - Verify response status is 201 CREATED
    - Verify user was created in database
    - Verify UserProfile was automatically created with correct data
- Postconditions: User and profile created during registration
- Expected output: HTTP 201, user and profile created
- Actual output:
- Pass or Fail:
- Bug id/link:
- Additional notes: Tests automatic profile creation during registration

Test case 37: **Profile Serialization In User Data**

- Test items: Profile data properly serialized in user responses
- Test priority: medium
- Dependencies: User with profile
- Preconditions: User has UserProfile with data
- Input data: None
- Test steps:
    - Create UserProfile for user with date_of_birth and bio
    - Send GET request to /api/profile/
    - Verify profile data is correctly serialized
    - Send GET request to /api/users/
    - Verify profile data is included in user listing
- Postconditions: Profile data correctly serialized in all responses
- Expected output: Profile data in both profile and user list endpoints
- Actual output:
- Pass or Fail:

- Bug id/link:
- Additional notes: Tests profile data serialization consistency

Test case 38: **Profile Update Creates Profile If Missing**

- Test items: Profile update creates UserProfile if it doesn't exist
- Test priority: medium
- Dependencies: Authenticated user without profile
- Preconditions: User is authenticated but has no UserProfile
- Input data: {"bio": "Creating profile via update"}
- Test steps:
    - Ensure user has no UserProfile
    - Send PATCH request to /api/profile/update/ with profile data
    - Verify response status is 200 OK
    - Verify UserProfile was created with specified data
- Postconditions: UserProfile created during update
- Expected output: HTTP 200, profile created
- Actual output:
- Pass or Fail:
- Bug id/link:
- Additional notes: Tests automatic profile creation during updates

Test case 39: **Multiple Users With Profiles**

- Test items: Multiple users with different profile configurations
- Test priority: medium
- Dependencies: Multiple users with varying profiles
- Preconditions: Multiple users with different profile setups
- Input data: None
- Test steps:
    - Create users with full profile, partial profile, and no profile
    - Send GET request to /api/users/
    - Verify response status is 200 OK
    - Verify each user shows appropriate profile data
    - Verify users without profiles show default values
- Postconditions: All users listed with appropriate profile data
- Expected output: HTTP 200, users with varying profile completeness
- Actual output:

- Pass or Fail:
- Bug id/link:
- Additional notes: Tests handling of different profile configurations

Test case 40: **Create RSVP Success**

- Test items: Successful RSVP creation
- Test priority: high
- Dependencies: Authenticated user, valid plan ID
- Preconditions: User is authenticated, MongoDB mocked
- Input data: {"plan_id": "valid_plan_id"}
- Test steps:
    - Mock MongoDB insert_one and find_one methods
    - Send POST request to create-rsvp endpoint
    - Verify response status is 201 CREATED
    - Verify response contains RSVP ID, message, and data
    - Verify data contains plan_id, user_id, and created_at
- Postconditions: RSVP created in database
- Expected output: HTTP 201, RSVP created with complete data
- Actual output:
- Pass or Fail:
- Bug id/link:
- Additional notes: Tests basic RSVP creation functionality

Test case 41: **Create RSVP Missing Plan ID**

- Test items: RSVP creation with missing plan_id
- Test priority: high
- Dependencies: Authenticated user
- Preconditions: User is authenticated
- Input data: {}
- Test steps:
    - Send POST request to create-rsvp endpoint without plan_id
    - Verify response status is 400 BAD REQUEST
    - Verify error response contains "plan_id" field
- Postconditions: RSVP creation fails
- Expected output: HTTP 400, plan_id validation error
- Actual output:

- Pass or Fail:
- Bug id/link:
- Additional notes: Tests required field validation

Test case 42: **Create RSVP Invalid Plan ID**

- Test items: RSVP creation with invalid plan_id format
- Test priority: medium
- Dependencies: Authenticated user, MongoDB mocked
- Preconditions: User is authenticated, MongoDB mocked
- Input data: {"plan_id": "invalid_id"}
- Test steps:
    - Mock MongoDB methods
    - Send POST request with invalid plan_id format
    - Verify response status is 201 CREATED (no format validation)
- Postconditions: RSVP created despite invalid format
- Expected output: HTTP 201, RSVP created
- Actual output:
- Pass or Fail:
- Bug id/link:
- Additional notes: Tests handling of invalid plan_id format

Test case 43: **Get RSVP By Plan ID Success**

- Test items: Successful retrieval of RSVPs by plan_id
- Test priority: high
- Dependencies: Authenticated user, MongoDB mocked
- Preconditions: User is authenticated, MongoDB mocked with RSVP data
- Input data: Valid plan_id
- Test steps:
    - Mock MongoDB find method to return RSVP list
    - Send GET request to get-rsvp-by-plan-id endpoint
    - Verify response status is 200 OK
    - Verify response contains data array with RSVPs
    - Verify RSVPs contain correct plan_id and user_id
- Postconditions: RSVPs retrieved successfully
- Expected output: HTTP 200, list of RSVPs for plan
- Actual output:

- Pass or Fail:
- Bug id/link:
- Additional notes: Tests RSVP retrieval by plan

Test case 44: **Get RSVP By Plan ID Not Found**

- Test items: Retrieval of non-existent RSVP by plan_id
- Test priority: medium
- Dependencies: Authenticated user, MongoDB mocked
- Preconditions: User is authenticated, MongoDB returns empty list
- Input data: Valid plan_id
- Test steps:
    - Mock MongoDB find method to return empty list
    - Send GET request to get-rsvp-by-plan-id endpoint
    - Verify response status is 404 NOT FOUND
    - Verify error message is "RSVP not found"
- Postconditions: No RSVPs found
- Expected output: HTTP 404, "RSVP not found" error
- Actual output:
- Pass or Fail:
- Bug id/link:
- Additional notes: Tests handling of no RSVPs found

Test case 45: **Get RSVP By User ID Success**

- Test items: Successful retrieval of RSVPs by user_id
- Test priority: high
- Dependencies: Authenticated user, MongoDB mocked
- Preconditions: User is authenticated, MongoDB mocked with user RSVPs
- Input data: None (uses authenticated user's ID)
- Test steps:
    - Mock MongoDB find method to return user's RSVPs
    - Send GET request to get-rsvp-by-user-id endpoint
    - Verify response status is 200 OK
    - Verify response contains data array with user's RSVPs
    - Verify RSVPs contain correct user_id
- Postconditions: User's RSVPs retrieved successfully
- Expected output: HTTP 200, list of user's RSVPs

- Actual output:
- Pass or Fail:
- Bug id/link:
- Additional notes: Tests RSVP retrieval by user

Test case 46: **Get RSVP By User ID Not Found**

- Test items: Retrieval of non-existent RSVP by user_id
- Test priority: medium
- Dependencies: Authenticated user, MongoDB mocked
- Preconditions: User is authenticated, MongoDB returns empty list
- Input data: None
- Test steps:
    - Mock MongoDB find method to return empty list
    - Send GET request to get-rsvp-by-user-id endpoint
    - Verify response status is 404 NOT FOUND
    - Verify error message is "RSVP not found"
- Postconditions: No RSVPs found for user
- Expected output: HTTP 404, "RSVP not found" error
- Actual output:
- Pass or Fail:
- Bug id/link:
- Additional notes: Tests handling of no user RSVPs found

Test case 47: **Delete RSVP By ID Success**

- Test items: Successful deletion of RSVP by ID
- Test priority: high
- Dependencies: Authenticated user, MongoDB mocked
- Preconditions: User is authenticated, MongoDB mocked
- Input data: Valid RSVP ID
- Test steps:
    - Mock MongoDB delete_one method to return deleted_count = 1
    - Send DELETE request to delete-rsvp-by-id endpoint
    - Verify response status is 200 OK
    - Verify success message contains RSVP ID
- Postconditions: RSVP deleted from database
- Expected output: HTTP 200, deletion success message

- Actual output:
- Pass or Fail:
- Bug id/link:
- Additional notes: Tests RSVP deletion by ID

Test case 48: **Delete RSVP By ID Not Found**

- Test items: Deletion of non-existent RSVP by ID
- Test priority: medium
- Dependencies: Authenticated user, MongoDB mocked
- Preconditions: User is authenticated, MongoDB returns deleted_count = 0
- Input data: Valid but non-existent RSVP ID
- Test steps:
    - Mock MongoDB delete_one method to return deleted_count = 0
    - Send DELETE request to delete-rsvp-by-id endpoint
    - Verify response status is 404 NOT FOUND
    - Verify error message is "RSVP not found"
- Postconditions: No RSVP deleted
- Expected output: HTTP 404, "RSVP not found" error
- Actual output:
- Pass or Fail:
- Bug id/link:
- Additional notes: Tests handling of non-existent RSVP deletion

Test case 49: **Delete RSVP By ID Invalid ID**

- Test items: Deletion with invalid RSVP ID format
- Test priority: medium
- Dependencies: Authenticated user
- Preconditions: User is authenticated
- Input data: Invalid ID format
- Test steps:
    - Send DELETE request with invalid ID format
    - Verify response status is 400 BAD REQUEST
    - Verify error message is "Invalid ID"
- Postconditions: Deletion fails due to invalid ID
- Expected output: HTTP 400, "Invalid ID" error
- Actual output:

- Pass or Fail:
- Bug id/link:
- Additional notes: Tests ID format validation

Test case 50: **Delete RSVP By Plan ID Success**

- Test items: Successful deletion of RSVP by plan_id
- Test priority: high
- Dependencies: Authenticated user, MongoDB mocked
- Preconditions: User is authenticated, MongoDB mocked
- Input data: Valid plan_id
- Test steps:
    - Mock MongoDB delete_one method to return deleted_count = 1
    - Send DELETE request to delete-rsvp-by-plan-id endpoint
    - Verify response status is 200 OK
    - Verify success message contains plan_id
- Postconditions: RSVP deleted from database
- Expected output: HTTP 200, deletion success message
- Actual output:
- Pass or Fail:
- Bug id/link:
- Additional notes: Tests RSVP deletion by plan

Test case 51: **Delete RSVP By Plan ID Not Found**

- Test items: Deletion of non-existent RSVP by plan_id
- Test priority: medium
- Dependencies: Authenticated user, MongoDB mocked
- Preconditions: User is authenticated, MongoDB returns deleted_count = 0
- Input data: Valid but non-existent plan_id
- Test steps:
    - Mock MongoDB delete_one method to return deleted_count = 0
    - Send DELETE request to delete-rsvp-by-plan-id endpoint
    - Verify response status is 404 NOT FOUND
    - Verify error message is "RSVP not found"
- Postconditions: No RSVP deleted
- Expected output: HTTP 404, "RSVP not found" error
- Actual output:

- Pass or Fail:
- Bug id/link:
- Additional notes: Tests handling of non-existent plan RSVP deletion

Test case 52: **Delete RSVP By Plan ID Invalid ID**

- Test items: Deletion with invalid plan ID format
- Test priority: medium
- Dependencies: Authenticated user
- Preconditions: User is authenticated
- Input data: Invalid plan ID format
- Test steps:
    - Send DELETE request with invalid plan ID format
    - Verify response status is 400 BAD REQUEST
    - Verify error message is "Invalid ID"
- Postconditions: Deletion fails due to invalid ID
- Expected output: HTTP 400, "Invalid ID" error
- Actual output:
- Pass or Fail:
- Bug id/link:
- Additional notes: Tests plan ID format validation

Test case 53: **RSVP Authentication Required**

- Test items: RSVP endpoints require authentication
- Test priority: high
- Dependencies: None
- Preconditions: No authentication
- Input data: Various RSVP data
- Test steps:
    - Remove authentication from client
    - Test all RSVP endpoints (create, get by plan, get by user, delete by ID, delete by plan)
    - Verify all endpoints return 401 UNAUTHORIZED
- Postconditions: All RSVP endpoints require authentication
- Expected output: HTTP 401 for all RSVP endpoints
- Actual output:
- Pass or Fail:
- Bug id/link:

- Additional notes: Tests authentication requirement for all RSVP operations

Test case 54: **Multiple RSVPs Same Plan**

- Test items: Multiple users can RSVP to the same plan
- Test priority: medium
- Dependencies: Multiple authenticated users, MongoDB mocked
- Preconditions: Multiple users authenticated, MongoDB mocked
- Input data: Same plan_id for different users
- Test steps:
    - Create two different users
    - Mock MongoDB methods for both users
    - Create RSVP for first user
    - Create RSVP for second user with same plan_id
    - Verify both RSVPs are created successfully
- Postconditions: Multiple RSVPs created for same plan
- Expected output: HTTP 201 for both RSVPs
- Actual output:
- Pass or Fail:
- Bug id/link:
- Additional notes: Tests multiple users RSVPing to same plan

Test case 55: **Duplicate RSVP Prevention**

- Test items: Users cannot RSVP to the same plan multiple times
- Test priority: high
- Dependencies: Authenticated user, MongoDB mocked
- Preconditions: User is authenticated, existing RSVP exists
- Input data: {"plan_id": "existing_plan_id"}
- Test steps:
    - Mock MongoDB find_one to return existing RSVP
    - Send POST request to create-rsvp with same plan_id
    - Verify response status is 400 BAD REQUEST
    - Verify error message contains "already RSVP'd"
    - Verify insert_one was not called
- Postconditions: Duplicate RSVP prevented
- Expected output: HTTP 400, duplicate RSVP error
- Actual output:

- Pass or Fail:
- Bug id/link:
- Additional notes: Tests duplicate RSVP prevention

Test case 56: **Create RSVP Duplicate Prevention**

- Test items: Duplicate RSVP creation is prevented
- Test priority: high
- Dependencies: Authenticated user, MongoDB mocked
- Preconditions: User is authenticated, existing RSVP exists
- Input data: {"plan_id": "existing_plan_id"}
- Test steps:
    - Mock MongoDB find_one to return existing RSVP
    - Send POST request to create-rsvp with same plan_id
    - Verify response status is 400 BAD REQUEST
    - Verify error message contains "already RSVP'd"
    - Verify insert_one was not called
- Postconditions: Duplicate RSVP creation prevented
- Expected output: HTTP 400, duplicate RSVP error
- Actual output:
- Pass or Fail:
- Bug id/link:
- Additional notes: Tests duplicate RSVP creation prevention

Test case 57: **RSVP Data Integrity**

- Test items: RSVP data is stored correctly
- Test priority: high
- Dependencies: Authenticated user, MongoDB mocked
- Preconditions: User is authenticated, MongoDB mocked
- Input data: {"plan_id": "valid_plan_id"}
- Test steps:
    - Mock MongoDB insert_one and find_one methods
    - Send POST request to create-rsvp
    - Verify response status is 201 CREATED
    - Verify insert_one was called with correct data
    - Verify data contains plan_id, user_id, and created_at
- Postconditions: RSVP data stored correctly

- Expected output: HTTP 201, correct data passed to database
- Actual output:
- Pass or Fail:
- Bug id/link:
- Additional notes: Tests RSVP data integrity and storage

Test case 58: **User Bio Test**

- Test items: User Bios can be obtained
- Test priority: moderate
- Dependencies: Authenticated user, plans working,
- Preconditions: User is logged in, user has friends, outgoing friend requests, incoming friend requests, and people they aren't friends with. User has friends with plans, and User has a plan that a friend has RSVP'd too
- Test steps:
    - Click on usernames in the following locations:
        - Friends list:
            - That they're friends with
            - That they aren't friends with
            - That they sent a friend request
            - That they received a friend request
        - Plans/Home page:
            - On a friend who has created a plan
            - On a friend who has RSVP'd to a plan
- Expected output: Sent to user BIO
- Actual output: Sent to user BIO
- Pass or Fail: Pass

Test case 59: **Profile Test**

- Test items: You can get to your own Profile page
- Test priority: moderate
- Dependencies: Authenticated user, plans working
- Preconditions: User is logged in, user has created a plan
- Test steps:
    - Do the following items:
        - Click on Profile icon in the banner
        - Click on own username on a plan you created

- Expected output: Should be sent to own Profile
- Actual output: Should be sent to own Profile
- Pass or Fail: Pass

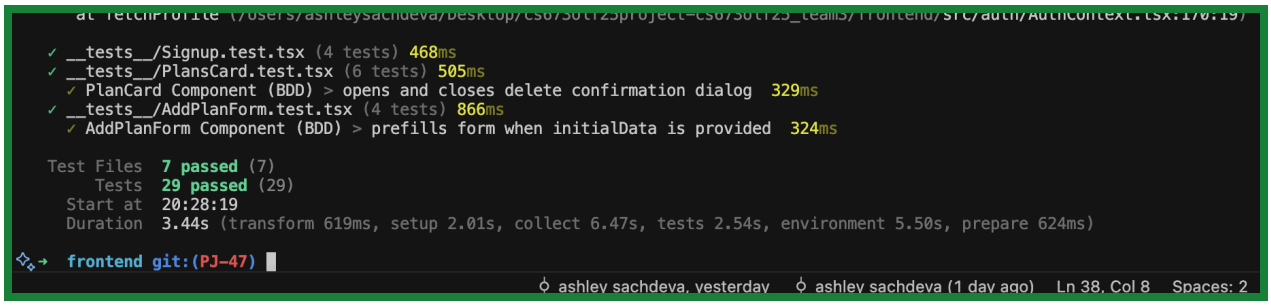
Test case 60: Edit **Profile Test**

- Test items: You can edit your profile page
- Test priority: moderate
- Dependencies: Authenticated user
- Preconditions: User is logged in, user is on Profile page
- Test steps:
    - Click “Edit”
    - Change all details
- Postconditions: You are sent back to your profile
- Expected output: Your profile items should change
- Actual output: Your profile items should change
- Pass or Fail: Pass

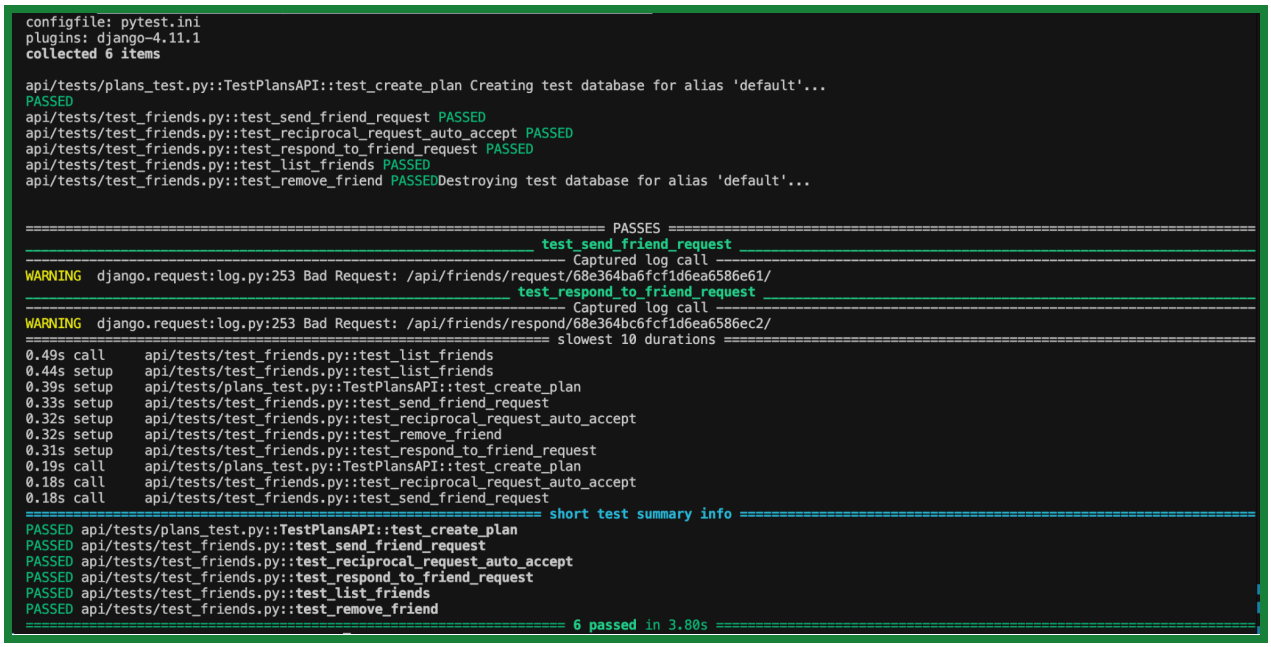
Automated Testing Report

Describe briefly the automated testing you have done, including where the test code resides in your code repository, what test frameworks are used, and the screen shots or generated testing report.

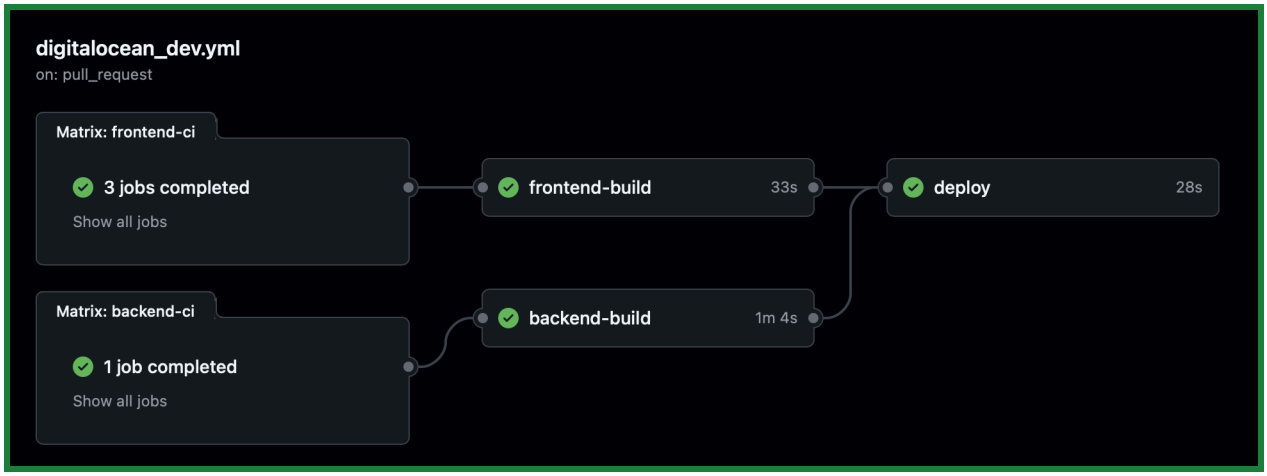
For the frontend of the application, all test files are stored under the __tests__/ directory. Each component and page (e.g., SignUp, PlansHeader, PlanCard, AddPlanForm, AddPlanPage, Home) has its corresponding test file (e.g., SignUp.test.tsx). We use the Vitest test runner and react testing library to both mock and assert. These tools allow realistic user interaction testing for React components while maintaining isolation from backend dependencies.



Backend and frontend tests validate core API and UI behavior: backend tests (backend/api/tests/, e.g., test_friends.py, plans_test.py) exercise friend relationships and plans CRUD using pytest + pytest-django with the DRF test client, while frontend tests cover React components and user interactions using Vitest + React Testing Library. The Postman collection was updated (postman/planning-jam-api-postman.json) to reflect API examples, and optional machine readable reports can be generated with pytest in the terminal



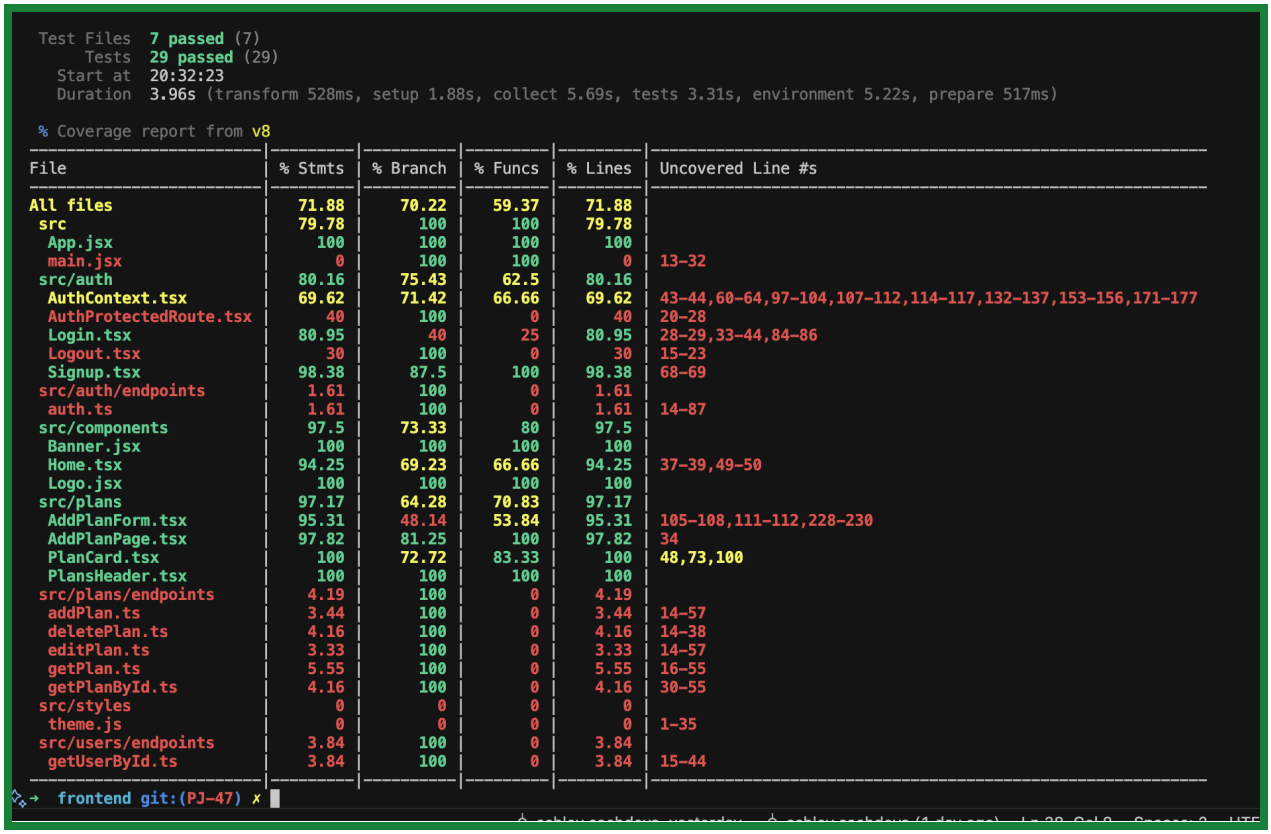
In addition, automated testing and deployment are integrated using GitHub Actions and DigitalOcean workflows (`digitalocean.yml`, `digitalocean_dev.yml`). On every PR, the CI pipeline runs `pytest` for backend tests and `vitest` for frontend unit tests. Only after all tests pass does the workflow trigger deployment to the DigitalOcean development environment (via `digitalocean_dev.yml`) or production environment (`digitalocean.yml`), ensuring no broken code reaches production.



- Testing Metrics

  In this section, you shall report any metrics used for the evaluation, e.g. # of test cases, test coverage, defects rate, etc.

  For the frontend we have above 70% coverage on statements, branches and lines and slightly below 60% coverage on functions.



```
 Test Files  7 passed (7)
      Tests  29 passed (29)
   Start at  20:32:23
   Duration  3.96s (transform 528ms, setup 1.88s, collect 5.69s, tests 3.31s, environment 5.22s, prepare 517ms)

 % Coverage report from v8
-------------------------|---------|----------|---------|---------|-------------------------------------------------------------------
File                     | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
-------------------------|---------|----------|---------|---------|-------------------------------------------------------------------
All files                |   71.88 |    70.22 |   59.37 |   71.88 |
 src                     |   79.78 |      100 |     100 |   79.78 |
  App.jsx                |     100 |      100 |     100 |     100 |
  main.jsx               |       0 |      100 |     100 |       0 | 13-32
 src/auth                |   80.16 |    75.43 |    62.5 |   80.16 |
  AuthContext.tsx        |   69.62 |    71.42 |   66.66 |   69.62 | 43-44,60-64,97-104,107-112,114-117,132-137,153-156,171-177
  AuthProtectedRoute.tsx |      40 |      100 |       0 |      40 | 20-28
  Login.tsx              |   80.95 |       40 |      25 |   80.95 | 28-29,33-44,84-86
  Logout.tsx             |      30 |      100 |       0 |      30 | 15-23
  Signup.tsx             |   98.38 |     87.5 |     100 |   98.38 | 68-69
 src/auth/endpoints      |    1.61 |      100 |       0 |    1.61 |
  auth.ts                |    1.61 |      100 |       0 |    1.61 | 14-87
 src/components          |    97.5 |    73.33 |      80 |    97.5 |
  Banner.jsx             |     100 |      100 |     100 |     100 |
  Home.tsx               |   94.25 |    69.23 |   66.66 |   94.25 | 37-39,49-50
  Logo.jsx               |     100 |      100 |     100 |     100 |
 src/plans               |   97.17 |    64.28 |   70.83 |   97.17 |
  AddPlanForm.tsx        |   95.31 |    48.14 |   53.84 |   95.31 | 105-108,111-112,228-230
  AddPlanPage.tsx        |   97.82 |    81.25 |     100 |   97.82 | 34
  PlanCard.tsx           |     100 |    72.72 |   83.33 |     100 | 48,73,100
  PlansHeader.tsx        |     100 |      100 |     100 |     100 |
 src/plans/endpoints     |    4.19 |      100 |       0 |    4.19 |
  addPlan.ts             |    3.44 |      100 |       0 |    3.44 | 14-57
  deletePlan.ts          |    4.16 |      100 |       0 |    4.16 | 14-38
  editPlan.ts            |    3.33 |      100 |       0 |    3.33 | 14-57
  getPlan.ts             |    5.55 |      100 |       0 |    5.55 | 16-55
  getPlanById.ts         |    4.16 |      100 |       0 |    4.16 | 30-55
 src/styles              |       0 |        0 |       0 |       0 |
  theme.js               |       0 |        0 |       0 |       0 | 1-35
 src/users/endpoints     |    3.84 |      100 |       0 |    3.84 |
  getUserById.ts         |    3.84 |      100 |       0 |    3.84 | 15-44
-------------------------|---------|----------|---------|---------|-------------------------------------------------------------------
➜  frontend git:(PJ-47) ✗
```

There is around 80% coverage on backend statements currently:

```
Name                                                                                    Stmts   Miss  Cover   Missing
---------------------------------------------------------------------------------------------------------------------
__init__.py                                                                                 0      0   100%
api/__init__.py                                                                             0      0   100%
api/admin.py                                                                                1      0   100%
api/apps.py                                                                                 4      0   100%
api/management/__init__.py                                                                  0      0   100%
api/migrations/0001_initial.py                                                              9      0   100%
api/migrations/0002_remove_friend_api_friend_sender__dfedbd_idx_and_more.py                 4      0   100%
api/migrations/0003_remove_friend_unique_sender_receiver_and_more.py                        6      0   100%
api/migrations/__init__.py                                                                  0      0   100%
api/models/__init__.py                                                                      2      0   100%
api/models/friends_models.py                                                               21      5    76%   61, 64-65, 68-69
api/serializers/auth_serializer.py                                                         32     10    69%   25-27, 30-32, 35-38
api/serializers/plans_serializer.py                                                        15      0   100%
api/tests/__init__.py                                                                       0      0   100%
api/tests/plans_test.py                                                                    16      0   100%
api/tests/test_friends.py                                                                  58      0   100%
api/urls.py                                                                                 5      0   100%
api/utils/mongo.py                                                                         23     10    57%   50-61, 65
api/views/__init__.py                                                                       0      0   100%
api/views/friends.py                                                                       83     19    77%   37, 44, 76, 152, 157-183
api/views/plans.py                                                                         78     34    56%   54-62, 69-75, 82-93, 100-116, 123-133
api/views/users.py                                                                         46     21    54%   39-52, 63-65, 83-101, 113-128
app/__init__.py                                                                             0      0   100%
app/apps.py                                                                                 9      0   100%
app/settings.py                                                                            29      0   100%
app/urls.py                                                                                 3      0   100%
conftest.py                                                                                20      0   100%
mongo_migrations/__init__.py                                                                0      0   100%
mongo_migrations/admin/0001_initial.py                                                     10      0   100%
mongo_migrations/admin/__init__.py                                                          0      0   100%
mongo_migrations/auth/0001_initial.py                                                      10      0   100%
mongo_migrations/auth/__init__.py                                                           0      0   100%
mongo_migrations/contenttypes/0001_initial.py                                               7      0   100%
mongo_migrations/contenttypes/__init__.py                                                   0      0   100%
---------------------------------------------------------------------------------------------------------------------
TOTAL                                                                                     491     99    80%
```

Along with 23 passed tests (not including the 6 automated CI pipeline tests run to each Pull Request ticket)

```
=================================================== short test summary info ====================================================
PASSED api/tests/plans_test.py::TestPlansAPI::test_create_plan
PASSED api/tests/rsvp_test.py::TestRSVPAPI::test_create_rsvp_success
PASSED api/tests/rsvp_test.py::TestRSVPAPI::test_create_rsvp_missing_plan_id
PASSED api/tests/rsvp_test.py::TestRSVPAPI::test_create_rsvp_invalid_plan_id
PASSED api/tests/rsvp_test.py::TestRSVPAPI::test_get_rsvp_by_plan_id_success
PASSED api/tests/rsvp_test.py::TestRSVPAPI::test_get_rsvp_by_plan_id_not_found
PASSED api/tests/rsvp_test.py::TestRSVPAPI::test_get_rsvp_by_user_id_success
PASSED api/tests/rsvp_test.py::TestRSVPAPI::test_get_rsvp_by_user_id_not_found
PASSED api/tests/rsvp_test.py::TestRSVPAPI::test_delete_rsvp_by_id_success
PASSED api/tests/rsvp_test.py::TestRSVPAPI::test_delete_rsvp_by_id_not_found
PASSED api/tests/rsvp_test.py::TestRSVPAPI::test_delete_rsvp_by_id_invalid_id
PASSED api/tests/rsvp_test.py::TestRSVPAPI::test_delete_rsvp_by_plan_id_success
PASSED api/tests/rsvp_test.py::TestRSVPAPI::test_delete_rsvp_by_plan_id_not_found
PASSED api/tests/rsvp_test.py::TestRSVPAPI::test_delete_rsvp_by_plan_id_invalid_id
PASSED api/tests/rsvp_test.py::TestRSVPAPI::test_rsvp_authentication_required
PASSED api/tests/rsvp_test.py::TestRSVPAPI::test_multiple_rsvps_same_plan
PASSED api/tests/rsvp_test.py::TestRSVPAPI::test_duplicate_rsvp_prevention
PASSED api/tests/rsvp_test.py::TestRSVPAPI::test_rsvp_data_integrity
PASSED api/tests/test_friends.py::test_send_friend_request
PASSED api/tests/test_friends.py::test_reciprocal_request_auto_accept
PASSED api/tests/test_friends.py::test_respond_to_friend_request
PASSED api/tests/test_friends.py::test_list_friends
PASSED api/tests/test_friends.py::test_remove_friend
==================================================== 23 passed in 7.61s ====================================================
```

Defect counts are inconsistent, we are unable to compute a defect density. Currently however, our LOC is = 1,416,730. And Average Cyclomatic Complexity is: A (3.08)

```
backend/api/models.py
    C 19:0 Friend - A (2)
    M 54:4 Friend.__str__ - A (1)
    M 57:4 Friend.accept - A (1)
    M 61:4 Friend.reject - A (1)
backend/api/apps.py
    C 4:0 ApiConfig - A (1)
backend/api/migrations/0002_remove_friend_api_friend_sender__dfedbd_idx_and_more.py
    C 6:0 Migration - A (1)
backend/api/migrations/0003_remove_friend_unique_sender_receiver_and_more.py
    C 8:0 Migration - A (1)
backend/api/migrations/0001_initial.py
    C 10:0 Migration - A (1)
backend/api/tests/test_friends.py
    F 70:0 test_list_friends - B (9)
    F 21:0 test_send_friend_request - A (4)
    F 35:0 test_reciprocal_request_auto_accept - A (4)
    F 54:0 test_respond_to_friend_request - A (4)
    F 97:0 test_remove_friend - A (3)
backend/api/tests/rsvp_test.py
    M 29:4 TestRSVPAPI.test_create_rsvp_success - B (7)
    M 86:4 TestRSVPAPI.test_get_rsvp_by_plan_id_success - B (7)
    M 128:4 TestRSVPAPI.test_get_rsvp_by_user_id_success - B (7)
    M 308:4 TestRSVPAPI.test_rsvp_data_integrity - A (5)
    C 17:0 TestRSVPAPI - A (4)
    M 53:4 TestRSVPAPI.test_create_rsvp_missing_plan_id - A (3)
    M 116:4 TestRSVPAPI.test_get_rsvp_by_plan_id_not_found - A (3)
    M 158:4 TestRSVPAPI.test_get_rsvp_by_user_id_not_found - A (3)
    M 170:4 TestRSVPAPI.test_delete_rsvp_by_id_success - A (3)
    M 185:4 TestRSVPAPI.test_delete_rsvp_by_id_not_found - A (3)
    M 199:4 TestRSVPAPI.test_delete_rsvp_by_id_invalid_id - A (3)
    M 208:4 TestRSVPAPI.test_delete_rsvp_by_plan_id_success - A (3)
    M 222:4 TestRSVPAPI.test_delete_rsvp_by_plan_id_not_found - A (3)
    M 235:4 TestRSVPAPI.test_delete_rsvp_by_plan_id_invalid_id - A (3)
    M 257:4 TestRSVPAPI.test_multiple_rsvps_same_plan - A (3)
    M 284:4 TestRSVPAPI.test_duplicate_rsvp_prevention - A (3)
    M 65:4 TestRSVPAPI.test_create_rsvp_invalid_plan_id - A (2)
    M 243:4 TestRSVPAPI.test_rsvp_authentication_required - A (2)
    M 19:4 TestRSVPAPI.setup_method - A (1)
backend/api/tests/plans_test.py
    C 15:0 TestPlansAPI - A (5)
    M 16:4 TestPlansAPI.test_create_plan - A (4)
backend/api/management/commands/seed_dev_users.py
    C 5:0 Command - A (5)
    M 8:4 Command.handle - A (4)
backend/api/utils/mongo.py
    F 31:0 get_collection - B (6)
    F 14:0 _unwrap_db - A (3)
```

```
backend/api/models/friends_models.py
    C 20:0 Friend - A (2)
    M 59:4 Friend.__str__ - A (1)
    M 63:4 Friend.accept - A (1)
    M 67:4 Friend.reject - A (1)
backend/api/serializers/rsvp_serializer.py
    C 8:0 RSVPSerializer - A (1)
backend/api/serializers/friends_serializer.py
    C 20:0 FriendSerializer - A (2)
    M 30:4 FriendSerializer.create - A (1)
    M 33:4 FriendSerializer.update - A (1)
backend/api/serializers/fields.py
    C 17:0 ObjectIdField - A (3)
    M 18:4 ObjectIdField.to_representation - A (2)
    M 24:4 ObjectIdField.to_internal_value - A (2)
backend/api/serializers/auth_serializer.py
    C 13:0 UserRegistrationSerializer - A (3)
    M 21:4 UserRegistrationSerializer.validate_email - A (2)
    M 29:4 UserRegistrationSerializer.validate - A (2)
    M 34:4 UserRegistrationSerializer.create - A (1)
    C 40:0 UserSerializer - A (1)
backend/api/serializers/plans_serializer.py
    C 7:0 LocationSerializer - A (1)
    C 15:0 PlansSerializer - A (1)
backend/api/views/friends.py
    F 140:0 remove_friend - B (10)
    F 35:0 send_friend_request - B (6)
    F 87:0 list_friends - A (5)
    F 67:0 respond_to_friend_request - A (4)
backend/api/views/users.py
    F 72:0 list_users - A (5)
    F 107:0 get_user - A (5)
    F 33:0 register_user - A (2)
    F 58:0 get_user_profile - A (1)
backend/api/views/rsvp.py
    F 25:0 create_rsvp - B (6)
    F 68:0 get_rsvp_by_plan_id - A (4)
    F 87:0 get_rsvp_by_user_id - A (4)
    F 106:0 delete_rsvp_by_id - A (3)
    F 122:0 delete_rsvp_by_plan_id - A (3)
backend/api/views/plans.py
    F 28:0 create_plan - A (5)
    F 81:0 get_plans_by_id - A (3)
    F 99:0 update_plan - A (3)
    F 122:0 delete_plan - A (3)
    F 68:0 get_plans - A (2)
    F 20:0 get_plans_collection - A (1)

75 blocks (classes, functions, methods) analyzed.
Average complexity: A (3.08)
```

- References

- Glossary