CS673 Software Engineering 

Team 1 - GetActive

Software Test Document

| Team Member | Role(s) | Signature | Date |
| --- | --- | --- | --- |
| Jin Hao Li | Design and Implementation Lead, Team Lead | JHL | 5/26/2025 |
| Arshdeep Dhillon | Security Lead  Design and Implementations  Team Lead | *AD* | 5/17/2025 |
| Shaohua Yue | Configuration Lead | *Shaohua Yue* | 5/26/2025 |
| Jie Shi | Requirements Lead | *JS* | 5/26/2025 |
| Xiaoyao Yin | Design and Implementation | *Xiaoyao Yin* | 5/26/2025 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**R**evision history

| Version | Author | Date | Change |
| --- | --- | --- | --- |
| 0.1 | Jin Hao Li | 5/26/2025 | Filled out Test Summary |
| 0.2 | Shaohua Yue | 5/26/2025 | Filled out part of frontend test summary |
| 1 | Arshdeep Dhillon | 5/26/2025 | Finalized for Iteration 1 |
| 2.0 | Jin Hao Li | 6/8 | Updated Unit Testing |
| 2.1 | Shaohua Yue | 6/9 | Updated Frontend Unit Testing, E2E Testing and Manual Testing |
| 2.2 | Jin Hao Li | 6/9 | Updated Testing Summary and Manual Testing Report |
| 2.3 | Xiaoyao Yin | 6/9 | Update backend implementation of activity discovery |
| 3.0 | Jin Hao Li | 6/15 | Updated Testing Summary and Manual Testing Report |
| 3.1 | Xiaoyao Yin | 6/15 | Implement activity discovery page testing |
| 3.2 | Shaohua Yue | 6/16 | Updated Manual testing |

[● Testing Summary 3](#_heading=h.j9qa868oc4gb)

[● Manual Testing Report 4](#_heading=h.b07hj7so1vs7)

[● Automated Testing Report 8](#_heading=h.udxhdtgov95)

[Frontend Automated Testing 8](#_heading=h.an9xs7fs4gqx)

[Backend Automated Testing 9](#_heading=h.5mnre12vr6e2)

[● Testing Metrics 10](#_heading=h.8xt8kndm5zas)

[● References 13](#_heading=h.381cbpgkvzcb)

[● Glossary 13](#_heading=h.dm4bbqyaypoq)

# 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
    - JUnit tests for the following APIs:
      * GET /v1/activities
      * GET /v1/activity/{name}
      * POST /v1/login
      * POST /v1/register
      * POST /v1/register/confirmation
      * POST /v1/register/confirmation/resend
      * DELETE /v1/activity/{id}
      * PUT /v1/activity/{id}
      * POST /v1/activity/participants
      * DELETE /v1/activity/participants
      * GET /v1/activity/participants
      * GET /v1/activity/{id}/participants
      * POST /v1/activity/{id}/comment
      * GET /v1/activity/{id}/comments
    - JUnit tests for the following APIs:
      * JWTService

Unit tests for the frontend:

Pages: Home, Registration, Login, Registration confirmation

Components: ProtectedRoute, AuthContext, AvatarUpload

Services: activityService, authService

* + Integration testing
    - Using SpringBootTest and TestRestTemplate to test each endpoint without any mocks
  + System Testing
    - Testing the application manually as whole to ensure the application meets functional and non-functional requirements.
  + Acceptance Testing -QA Testing - feature works as expected
    - Acceptance Testing is done manually by the QA of each ticket to ensure the acceptance criteria are met and the feature is working as expected.
  + Regression Testing - Manually
    - This is done manually on each change before submitting a PR to ensure the existing features are still working as expected.

End to end Testing:

* + - Cypress tests for the following pages:

User’s registration and login process

User’s join/leave activity process on the home page

# 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 name: (iteration3)

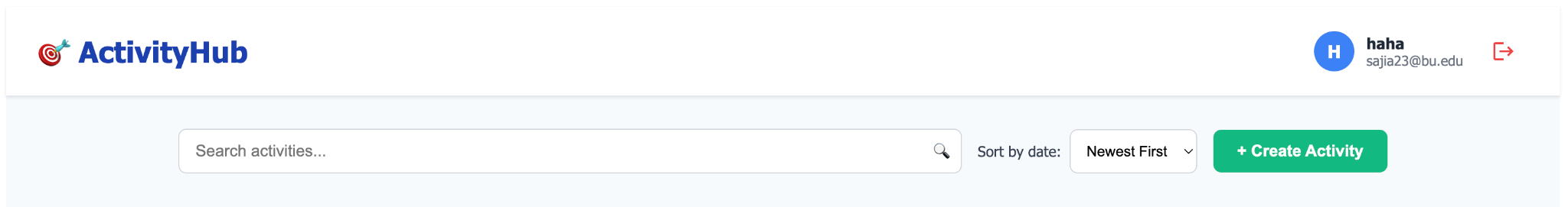
Allow user to change their avatar

* New
* Test steps:

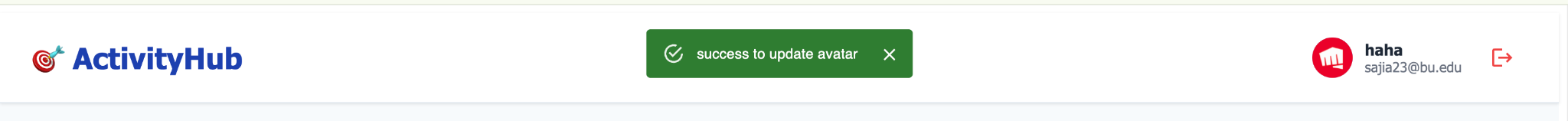
1. Register an account and login to the home page.
2. Upload a new avatar
3. Users should see the new avatar on the home page.

* Expected output: Users should see the new avatar on the home page.
* Actual output: Users could see the new avatar on the home page.
* Pass

Before uploading the avatar:



After uploading the avatar:



* Test case name: (iteration2)

System should lead user to login page after token expired

* New
* Test steps:

1. Register an account and login to the home page.
2. Wait until the token expires(30 mins).
3. Click any button or component of the home page.
4. Users should be redirected to the login page.

* Expected output: Users should be redirected to the login page.
* Actual output: Users are redirected to the login page.
* Pass
* Test case name: (iteration1)

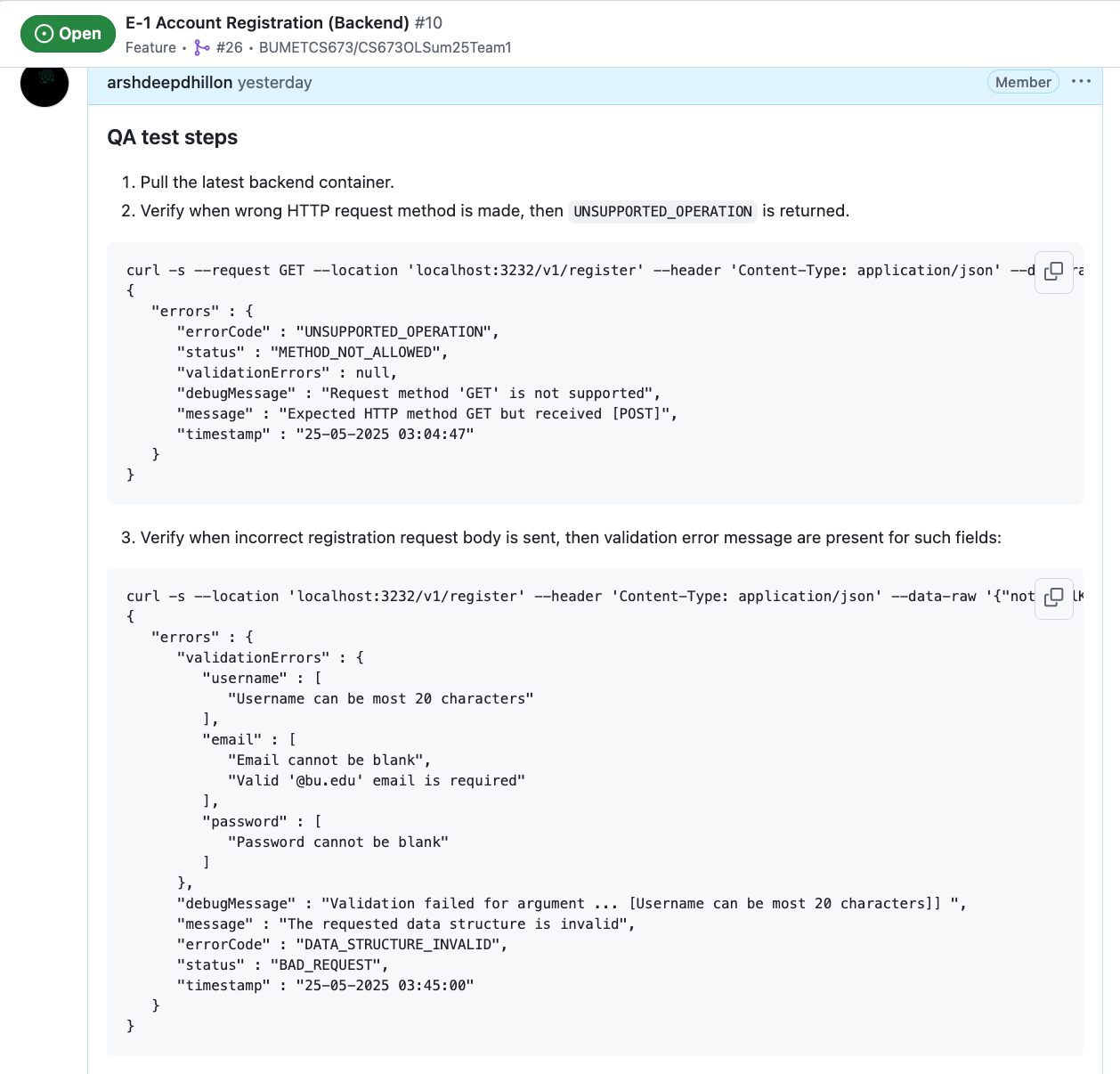
System should stay in home page after user refresh the home page

* New
* Test steps:

1. Register an account and login to the home page.
2. Refresh the page.

* Expected output: Users should be on the home page.
* Actual output: Users can’t see the content of the home page.
* Fail
* Bug link: <https://github.com/BUMETCS673/CS673OLSum25Team1/pull/49>
* Addition Notes: Fixed

A simple format for manual testing is used where each ticket will contain test steps for verifying the functionality of the changes and that the whole system continues to work as before:



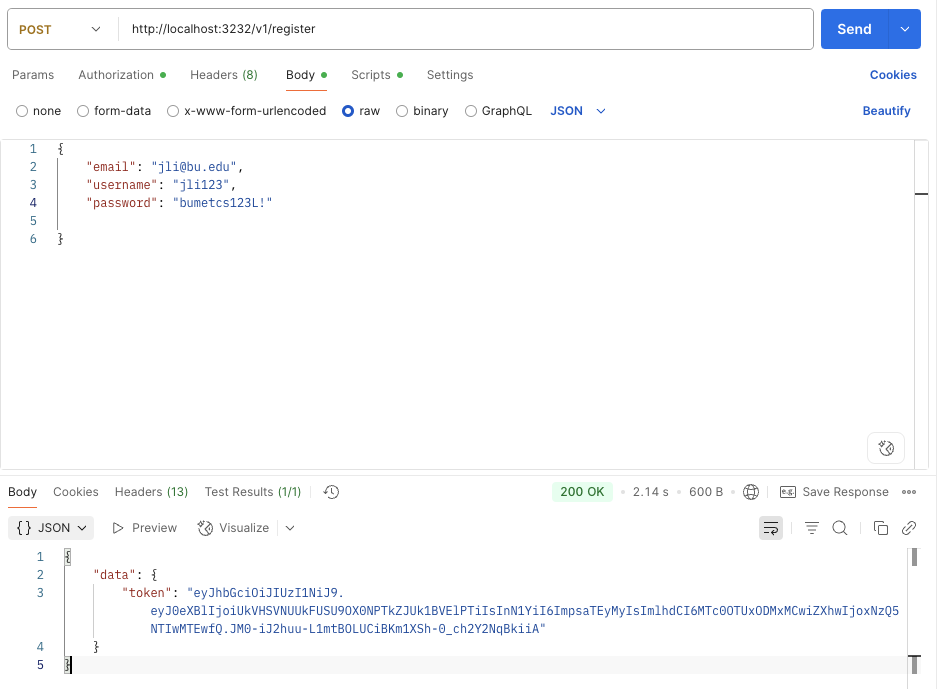
* Test case name: (iteration2)

Happy Path Backend Test for Account Registration API

* New
* Input Data:
  + Email: Valid Bu email address
  + Password:
    - Password must contain at least one uppercase letter
    - Password must contain at least one special character
    - Password must contain at least one digit
    - Password must contain at least 8 characters long
  + Username:
    - Username must contain at least 2 characters long
* Test steps:

1. Enter a valid email, password and username in the request body

* Expected output: A token is returned in the response body.
* Actual output: A token is returned in the response body.
* Pass



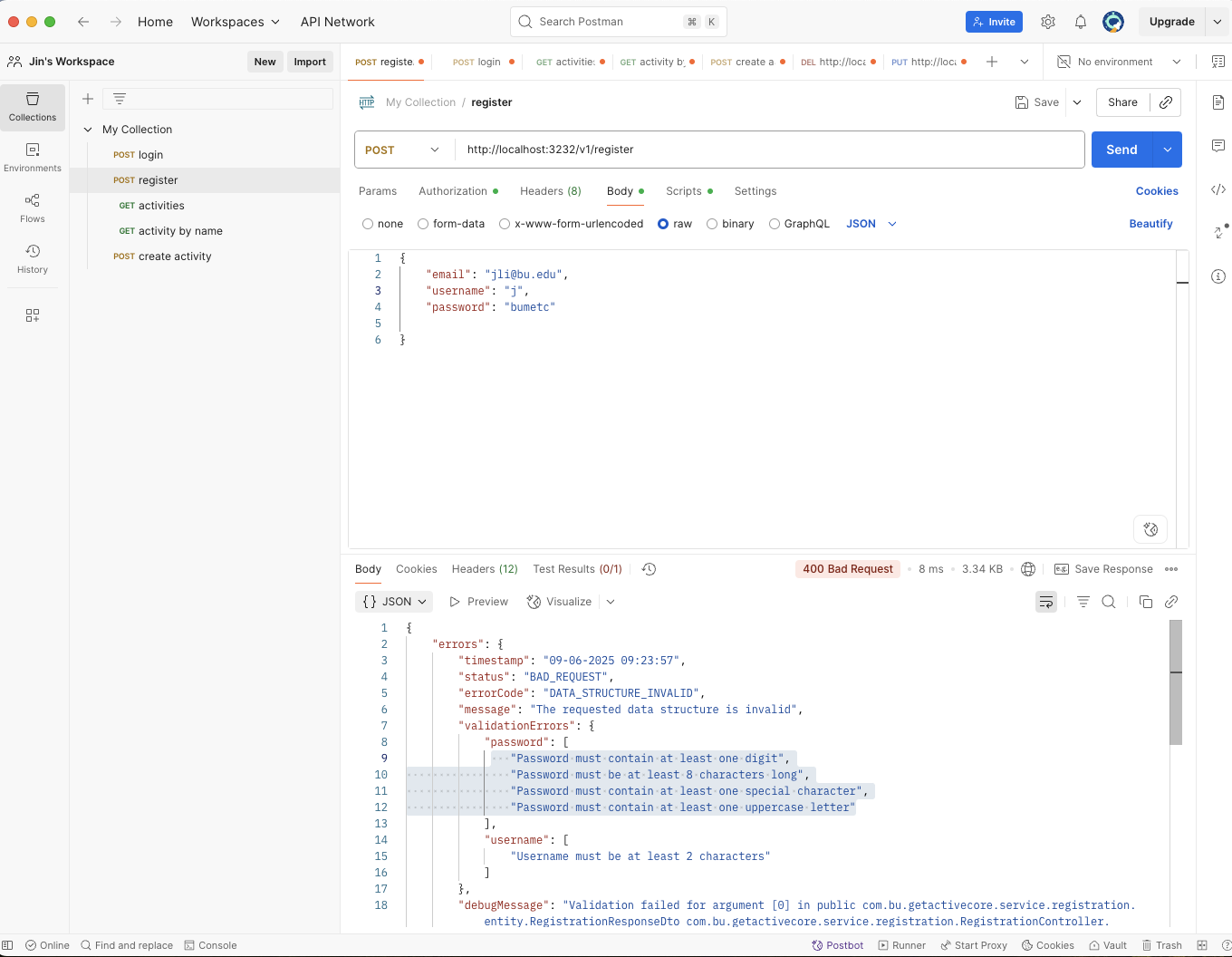
* Test case name: (iteration2)

Validation Backend Test for Account Registration API

* New
* Input Data:
  + Email: Valid Bu email address
  + Password:
    - Password must contain at least one uppercase letter
    - Password must contain at least one special character
    - Password must contain at least one digit
    - Password must contain at least 8 characters long
  + Username:
    - Username must contain at least 2 characters long
* Test steps:

1. Enter a invalid email with valid password and username in the request body and then call /v1/register
2. Enter a invalid email with valid password and email in the request body and then call /v1/register
3. Enter a invalid password with valid email and username in the request body and then call /v1/register

* Expected output: Validation errors are returned in the response body.
* Actual output: Validation errors are returned in the response body.
* Pass



* Test case name: (iteration3)

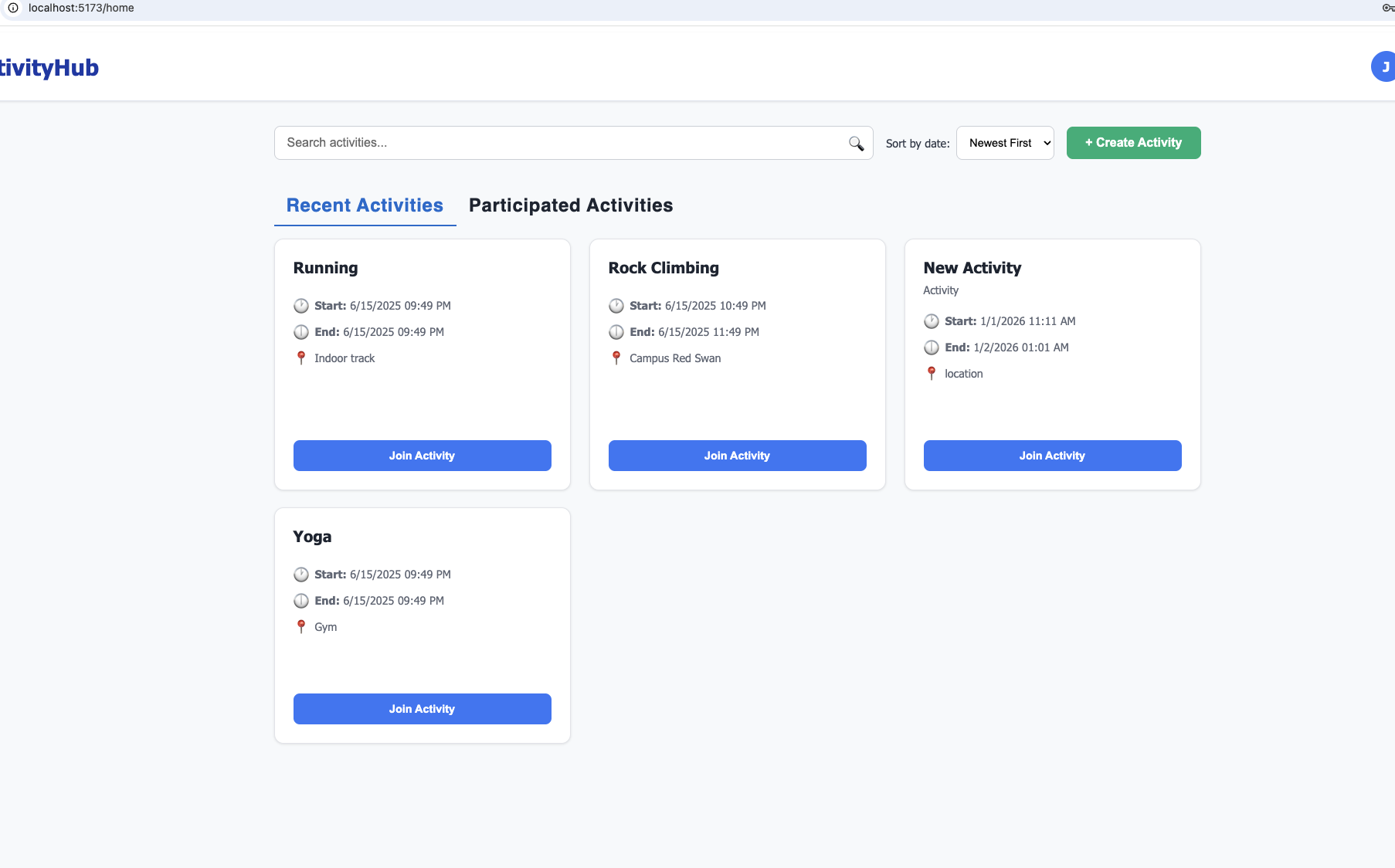
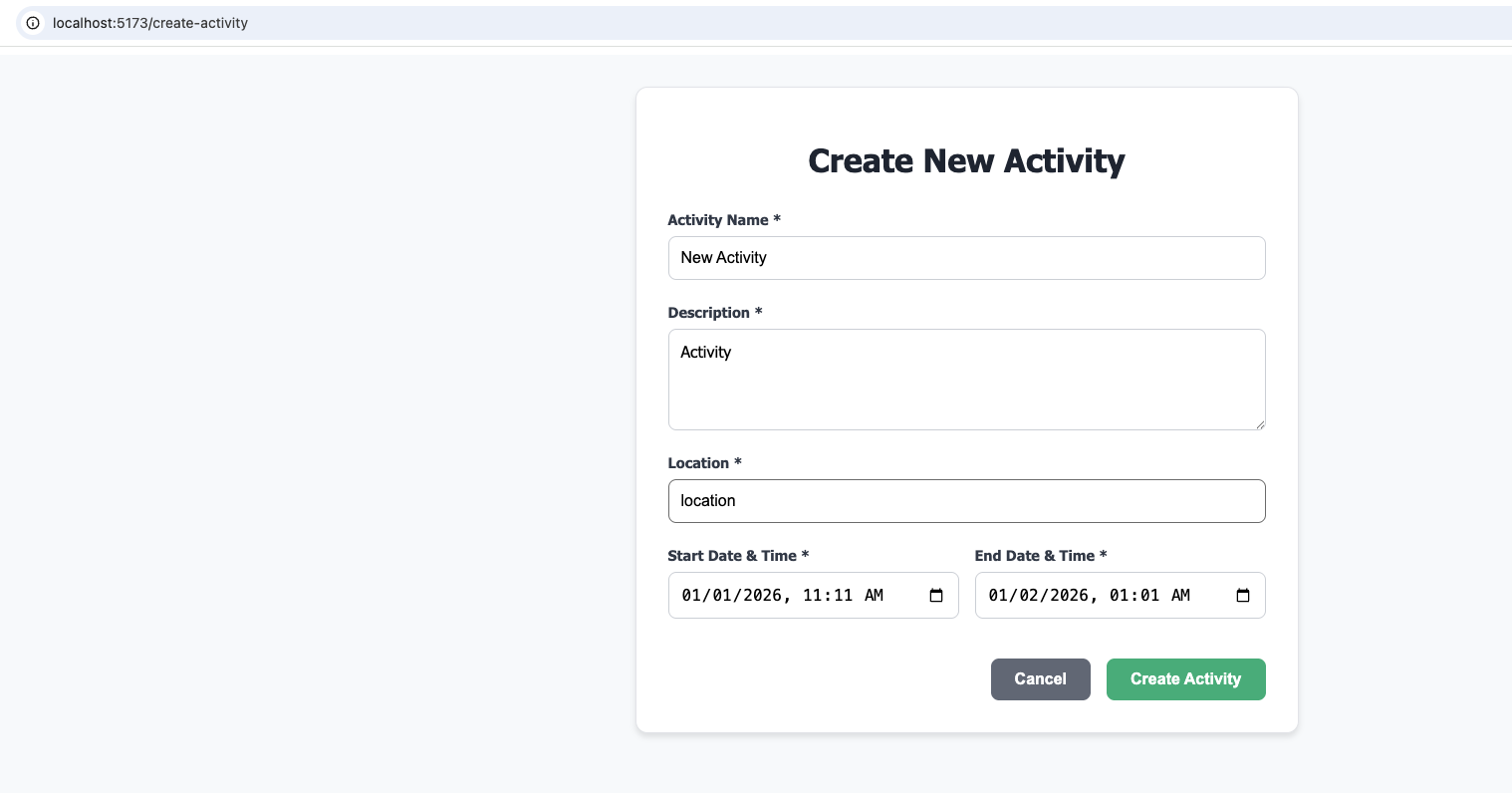
Happy Path for Frontend Test for Create Activity

* New
* Input Data:
  + Name:
    - Name cannot be blank
    - The length of name must be less than or equal to 250.
  + Description:
    - Description cannot be blank
    - The length of description must be less than or equal to 250.
  + Location:
    - Location cannot be blank
    - The length of location must be less than or equal to 250.
  + Start DateTime:
    - Start DateTime cannot be blank
  + End DateTime:
    - End DateTime cannot be blank
* Test steps:

1. Enter a valid name, description, location, start datetime and end datetime

2. Hit create activity button

* Expected output: Activity is successfully created and displayed on the Recent Activity Page
* Actual output: Activity is successfully created and displayed on the Recent Activity Page
* Pass



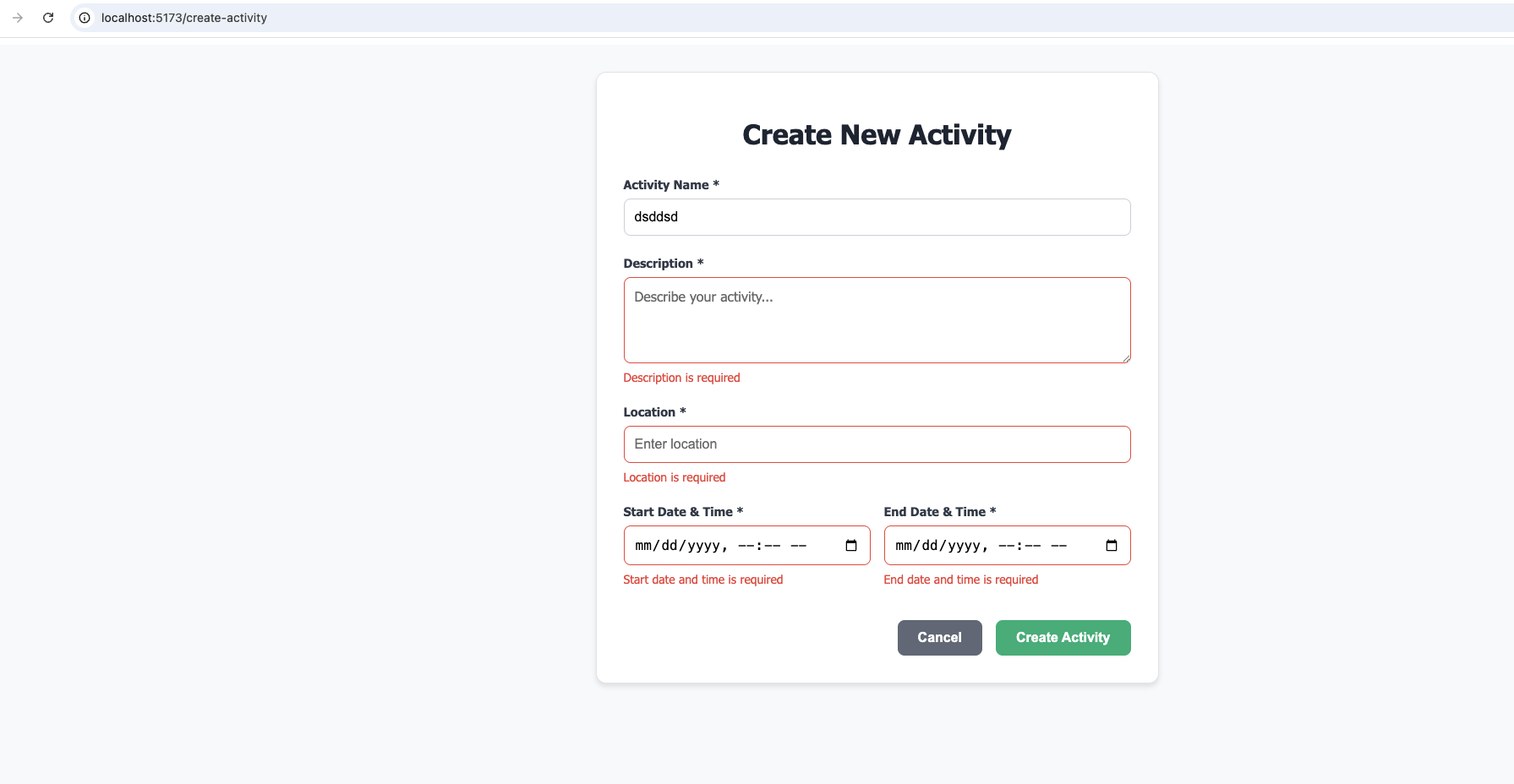
* Test case name: (iteration3)

Validation Path for Frontend Test for Create Activity

* New
* Input Data:
  + Name:
    - Name cannot be blank
    - The length of name must be less than or equal to 250.
  + Description:
    - Description cannot be blank
    - The length of description must be less than or equal to 250.
  + Location:
    - Location cannot be blank
    - The length of location must be less than or equal to 250.
  + Start DateTime:
    - Start DateTime cannot be blank
  + End DateTime:
    - End DateTime cannot be blank
* Test steps:

1. Enter a valid name
2. Hit create activity button

* Expected output: Errors are displayed on the Create Activity Page
* Actual output: Errors are displayed on the Create Activity Page
* 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.

## Frontend Automated Testing

The project incorporates a comprehensive automated testing strategy for both the frontend and backend to ensure system reliability, early detection of defects, and ease of maintenance. On the frontend, unit and integration tests are implemented using **Vitest** in conjunction with **React Testing Library**. These tests validate the behavior of key UI components, including the login and registration forms, by simulating user interactions through *userEvent*. The tests focus on verifying correct handling of both valid and invalid user input, error messaging, and navigation flow upon authentication. To isolate component behavior, dependencies such as *useAuth* and *useNavigate* are mocked. All unit and integration test cases are located in the *frontend/test/* directory, and test coverage reports are generated using the *vitest --coverage* command, which is integrated into the development workflow.

In addition to unit tests, end-to-end (E2E) testing is conducted using **Cypress**. These tests cover full user workflows, primarily focusing on the user registration and login processes. They validate interactions from the user's perspective, ensuring that data flow and UI updates occur as expected throughout the registration and authentication journey. The E2E tests reside in the *frontend/cypress/* directory and are configured to run automatically within the continuous integration (CI) pipeline, allowing for consistent validation during each build and deployment cycle.

## Backend Automated Testing

On the backend, automated testing is implemented using **JUnit 5**, **Spring Boot Test**, and **MockMvc**. These tests ensure that the RESTful API endpoints under */v1/* behave correctly under a variety of scenarios, including valid requests, invalid inputs, missing data, and improper content types. Each endpoint is rigorously tested to confirm that appropriate HTTP status codes are returned and that the response bodies contain informative error messages when needed. External dependencies such as the email service and Gmail configuration are mocked using *@MockBean* to isolate the logic and prevent unintended side effects during testing. These backend tests are located under the *code/backend/getactivecore/src/test/java/com/bu/getactivecore* directory and are executed using standard Gradle commands as part of the CI process. Overall, the automated test suites across the frontend and backend provide a solid foundation for continuous quality assurance. Unit, integration, and E2E tests together ensure that individual components function as intended, modules interact correctly, and full workflows operate seamlessly from the user’s perspective. This robust testing strategy not only improves our code quality but also facilitates rapid development and confidence in the deployment.

For CI/CD:

In CI workflow, we make unit tests of frontend and backend, and also E2E tests for the whole project.

Process:

We first run **unit tests for the backend**.

Then, we **build and start the backend service**.

Next, we run **unit tests for the frontend**, followed by **starting the frontend project**.

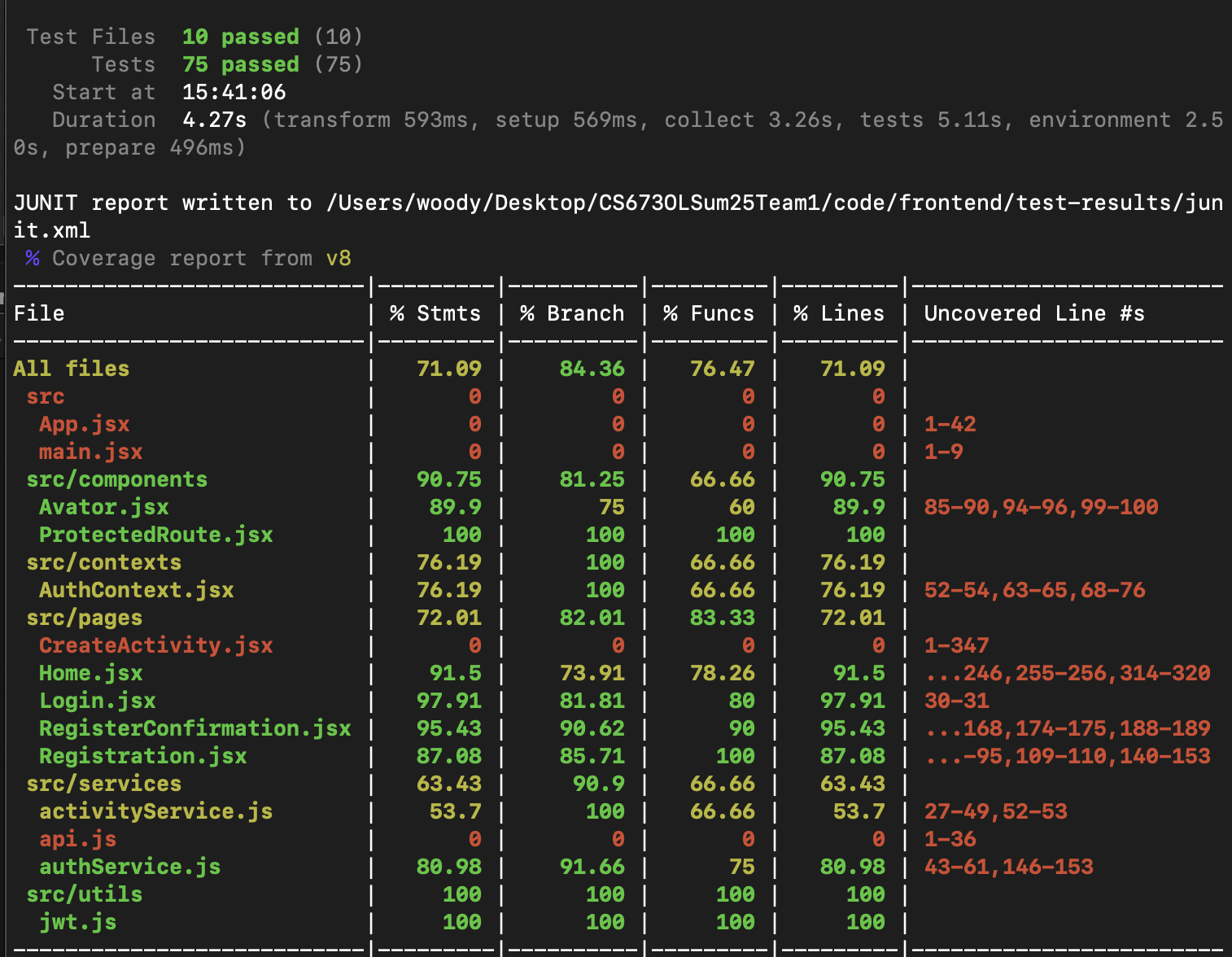
Finally, with both frontend and backend services running, we execute the **end-to-end (E2E) tests**.

In the CD workflow, the project is automatically deployed to the production environment on AWS whenever a Git tag is created.

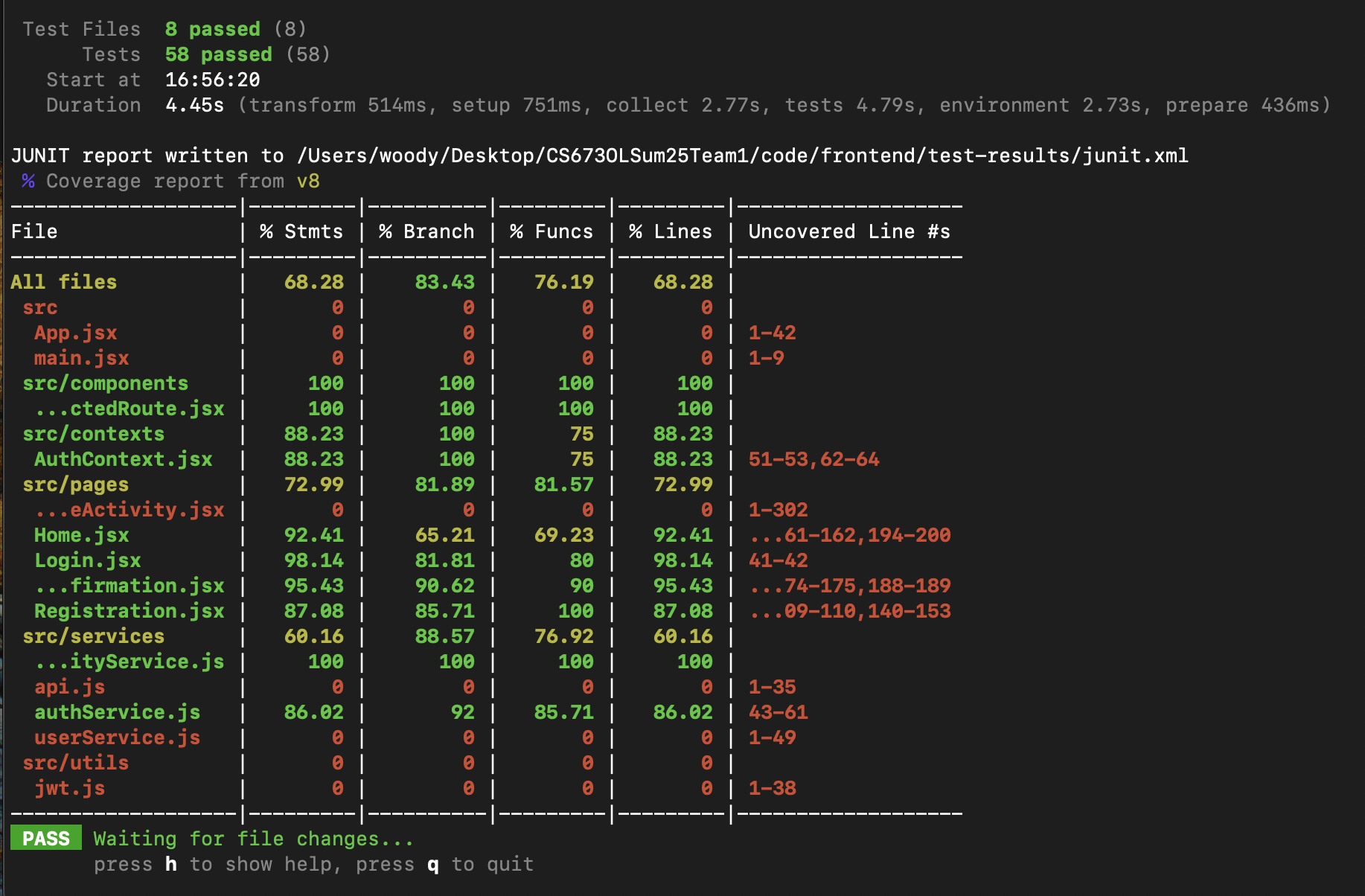
# Testing Metrics

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

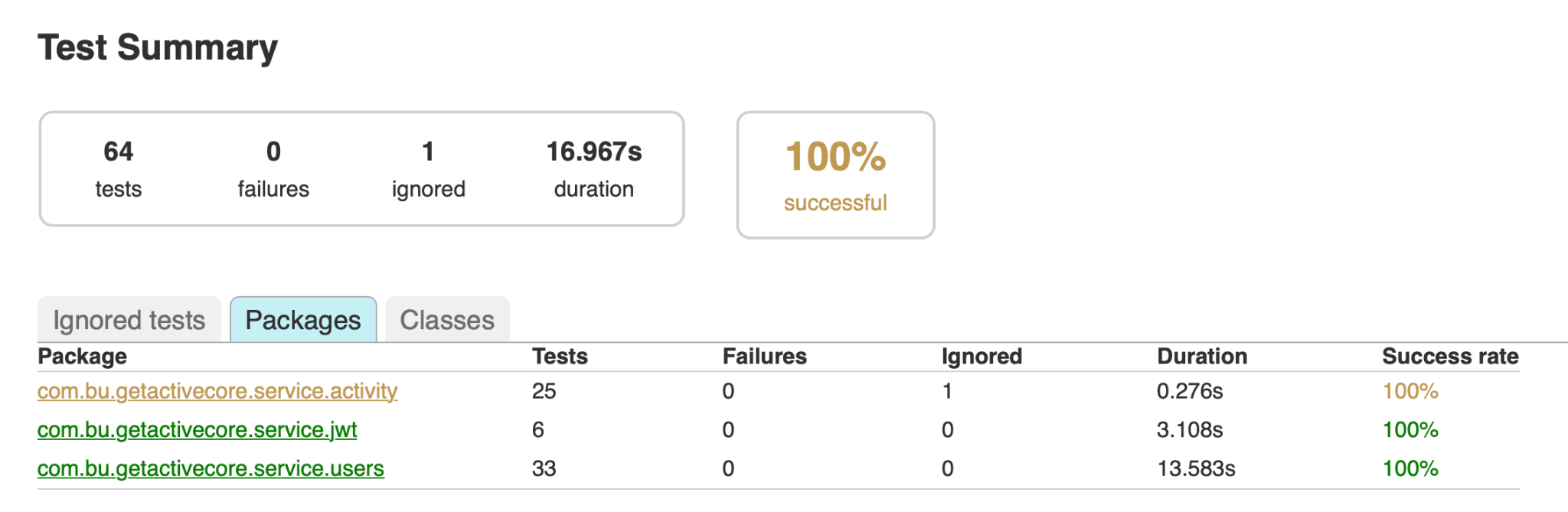
Unit test report for frontend(iteration3):



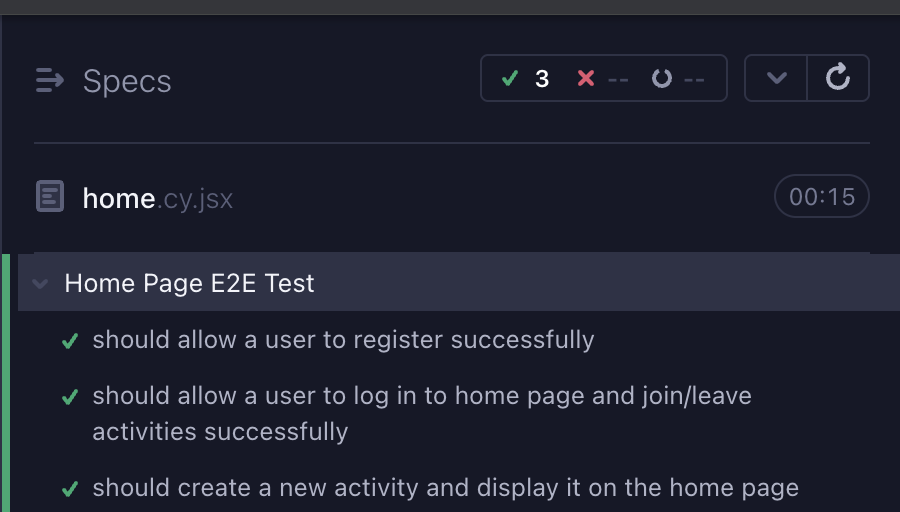
Unit test report for frontend(iteration2):



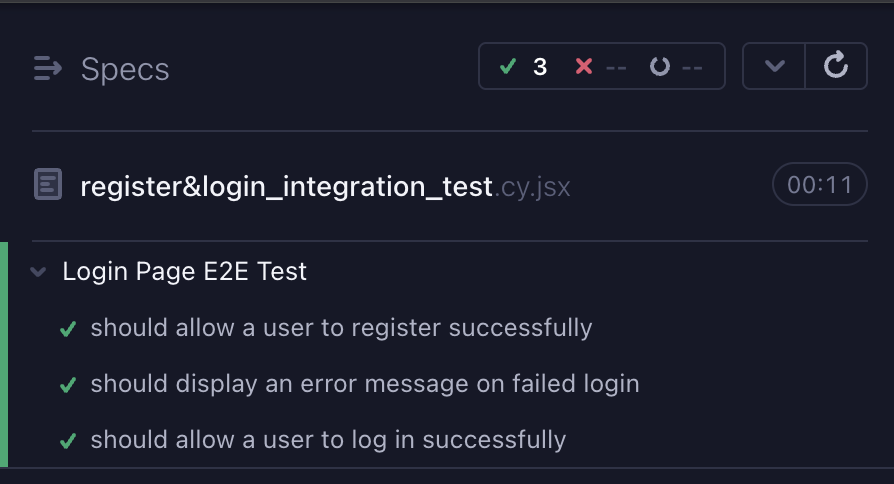
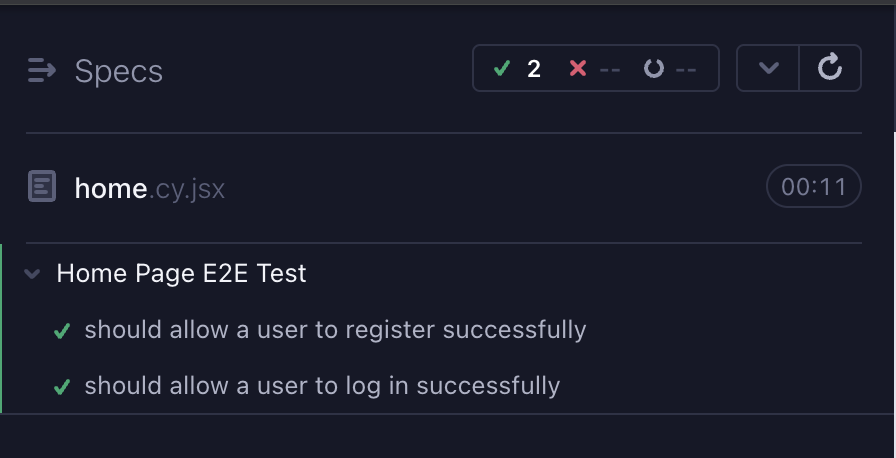
Unit test report for backend(iteration2):



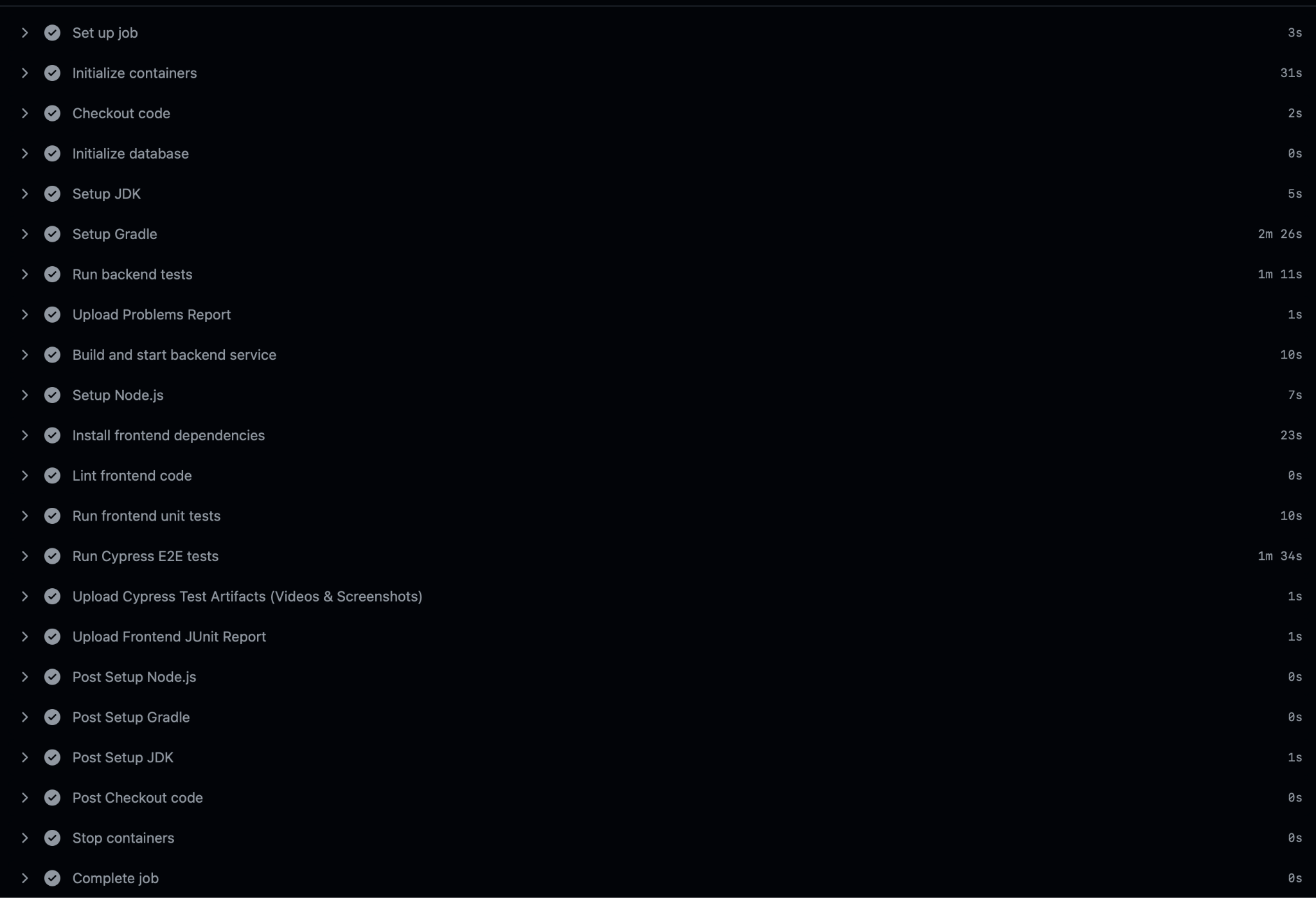
E2E test(iteration3):



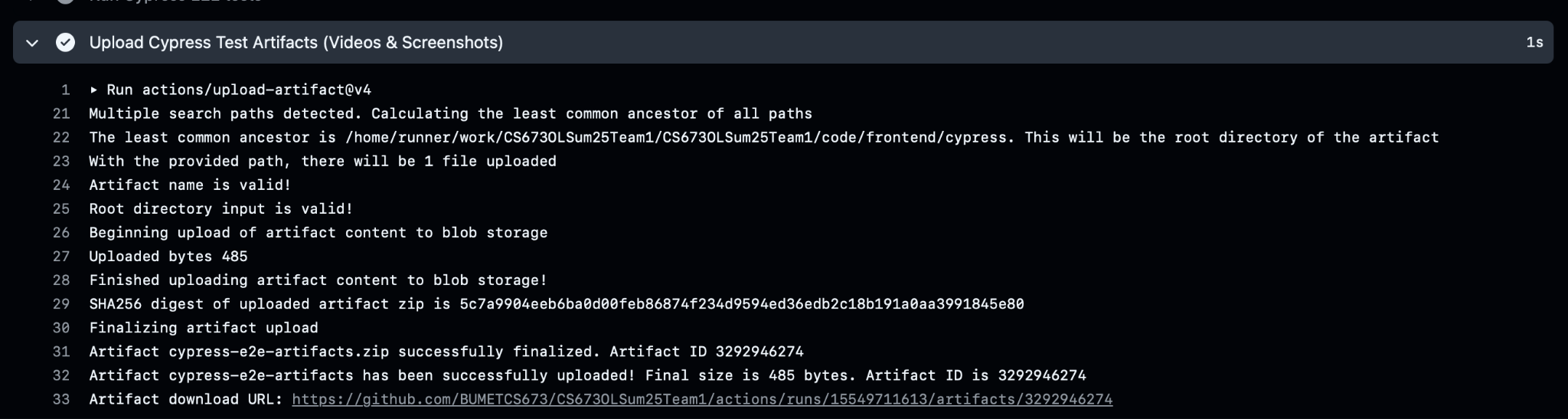
E2E test(iteration2):



CI workflow:



After each workflow, the system will generate test reports and user could download them:



# References

# Glossary