Test Plan

Introduction

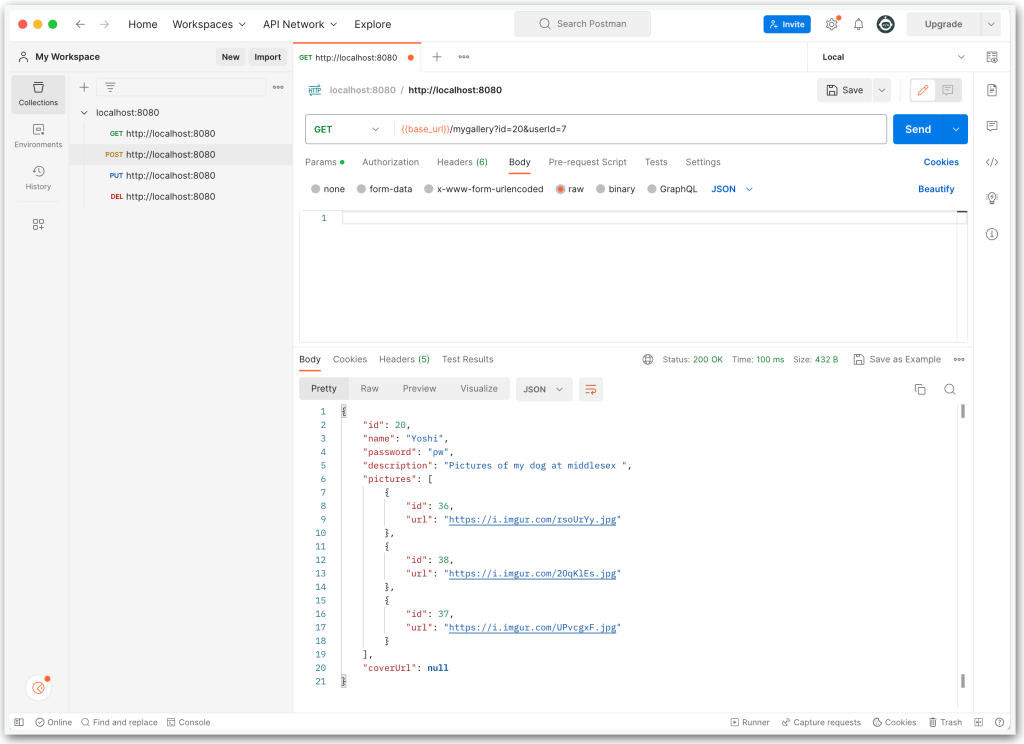
This test plan document provides an overview of the testing approach and activities carried out during the development of VisionVault.

Test Plan

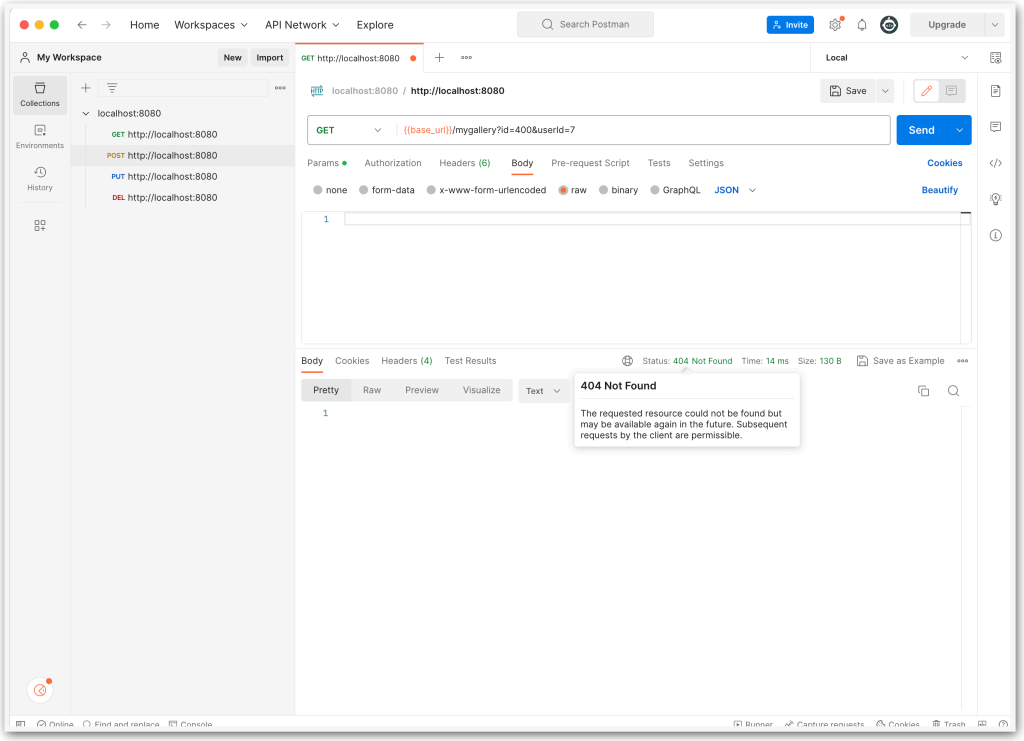
One of the core features of VisionVault is the ability to generate comprehensive reports. It was crucial to thoroughly test the functionality manually and with unit tests to ensure the reports would be generated correctly.
Report generation functionality was implemented in methods of the **ReportServiceImpl** class. The unit test plan for the **ReportServiceImpl** class covers its public methods and possible scenarios that might arise during the operation. The following scenarios were covered:

1. **ReportServiceImpl.getAllUserReports()** returns a list of reports when there are galleries for a user and the galleries have associated events.
2. **ReportServiceImpl.getAllUserReports()** returns an empty list when there are no galleries for a user.
3. **ReportServiceImpl.getAllUserReports()** returns multiple reports when there are multiple galleries with associated events for a user.
4. **ReportServiceImpl.getAllUserReports()** returns reports with zero events when there are no events for a user's gallery.

In addition to automated unit testing, manual testing was conducted at various development stages. It included testing the REST API endpoints responsible for creating, viewing, updating, and sharing photo galleries. I utilized Postman, an API testing tool with a user-friendly interface, to send requests to the API endpoints and inspect the responses.
Here is an example of the approach I took to test the Get My Gallery endpoint: `/mygallery`
I sent a GET request to the `/mygallery` endpoint with a valid id and userId as request parameters. I verified that the response contained the correct gallery and had a HttpStatus.OK status, or a HttpStatus.NOT_FOUND if the gallery was not found.



GET request to the `/mygallery` returns a Gallery JSON object with HttpStatus.OK



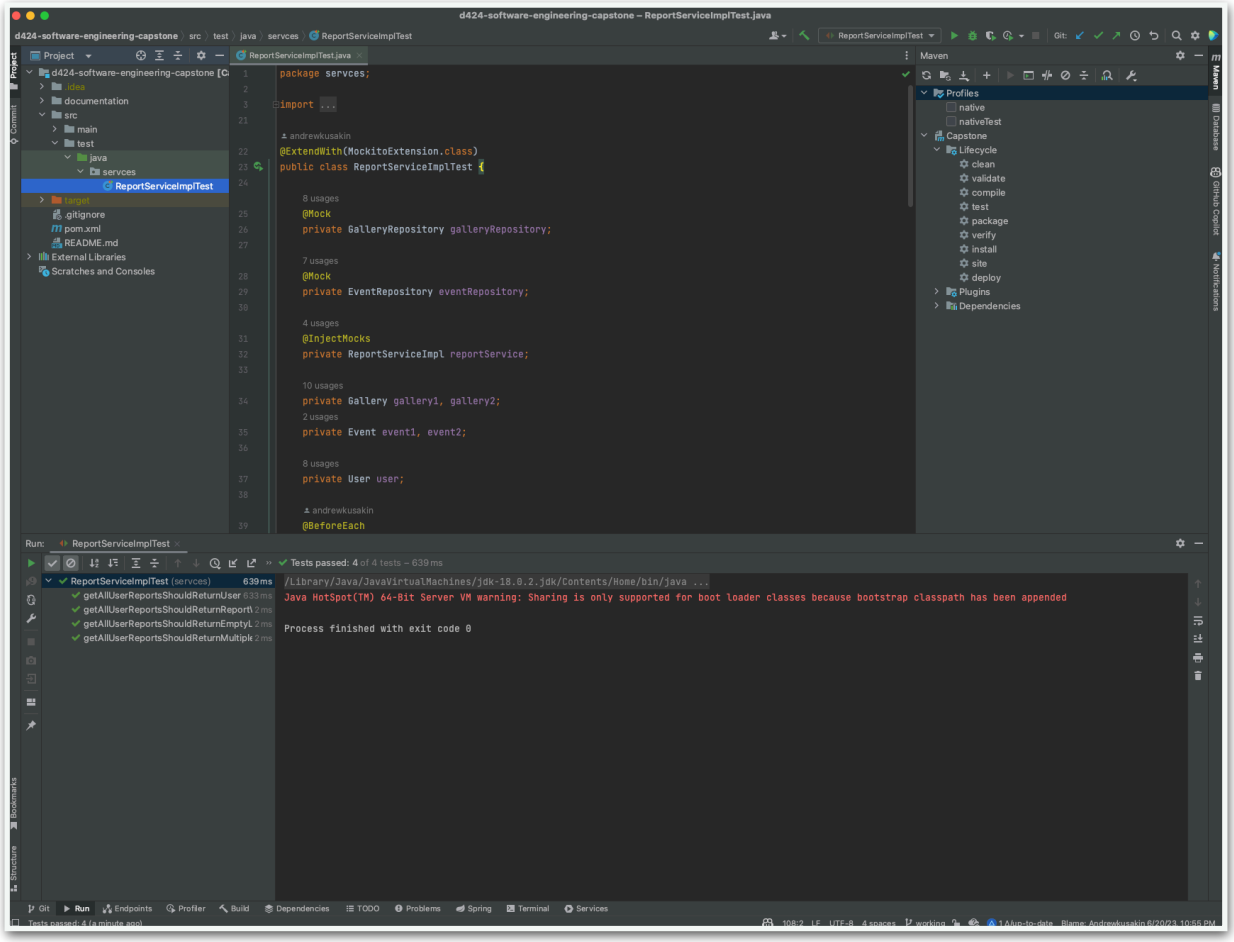
GET request to the `/mygallery` with an invalid gallery id returns an empty response object with HttpStatus.NOT_FOUND

Unit Test Scripts

The test scripts were written in Java using the JUnit 5 and Mockito frameworks. It allowed me to create a mock environment and verify that the service's methods behaved as expected. Please find the code with the scripts for four test scenarios, described in section a of this document, here: https://gitlab.com/wgu-gitlab-environment/student-repos/andreikusakin/d424-software-engineering-capstone/-/blob/working/src/test/java/servces/ReportServiceImplTest.java

Unit Test Results

JUnit provides a report outlining which tests passed and failed each time a test is run. All four out of four test cases passed successfully. Please find the screenshots of the results for each test case below:



The result of the unit tests based on the provided test plan.

Changes Resulting from Completed Tests

All unit tests passed successfully on the first attempt. It indicated that the getAllUserReports() method of the ReportServiceImpl class performed as expected under all listed scenarios. Therefore, it was unnecessary to make any changes to the existing code based on the results of these tests.

Conclusion

The combination of automated unit tests and manual testing ensured that the application behaved as expected and provided a user-friendly and efficient platform for photographers and their clients.