Hackademia
Emily Bever, Ryan Connolly, Robin Heft, Molly Vongsakhamphouy

Overall Test Plan

Our plan encompasses three primary concerns that were central to our testing approach: back-end functionality and security, front-end functionality and responsiveness, and the overall performance/coverage of both the application and the associated tests.

For the back-end testing, our tests conduct an examination of the application's server-side processes, database interactions, and data integrity. This will be done by calling methods used by the back-end application using predetermined input values and checking the output against the expected results. New functions will be written to check the application's resilience against common cybersecurity threats, such as SQL injection and Cross-Site Scripting.

Front-end functionality and responsiveness is addressed by evaluating the UI/UX. Test cases focus on verifying that all UI elements operate as expected. Responsiveness testing involves assessing the application's performance on different web browsers and screen sizes in an effort to provide a consistent user experience.

Performance testing will be done using stress tests, assessing the application's durability under different loads. Coverage of the code by tests will also be evaluated for quality assurance and so that we can improve our test suite in the future.

Test Case Descriptions

1.1 **Homepage Load Test.**
1.2 To verify that the homepage of the Hackademia website loads correctly with all necessary elements.
1.3 This test involves accessing the platform's URL and checking if the homepage loads fully with all interactive elements, images, and texts.
1.4 Inputs: Hackademia URL
1.5 Outputs: A fully loaded homepage with all elements displayed correctly and interactive components functioning.
1.6 Normal
1.7 Blackbox
1.8 Functional
1.9 Unit

2.1 **Database Integration Test.**
2.2 To ensure that the platform's components can establish a connection and interact with the Postgres database successfully.
2.3 This test involves multiple components of the platform, such as user registration and data retrieval, interacting with the database. The test checks if data flows correctly between the application and the database, ensuring data is accurately read from and written to the database.

2.4 Inputs: User registration details for testing the registration process, requests for data retrieval to evaluate data fetching
2.5 Outputs: Successful registration of new users with details accurately stored in the database, correct and timely retrieval of data upon user requests
2.6 Normal
2.7 Blackbox
2.8 Functional
2.9 Integration

3.1 **System Security Test.**
3.2 To assess the platform's security mechanisms against common cybersecurity threats.
3.3 This test involves performing various security checks, such as SQL injection, Cross-Site Scripting (XSS), and password cracking attempts, to ensure the platform's resilience against these threats.
3.4 Inputs: Attack vectors specific to SQL injections, XSS, and other common web application vulnerabilities.
3.5 Outputs: The platform should not break from these attacks, demonstrating effective security measures.
3.6 Abnormal
3.7 Whitebox
3.8 Functional
3.9 Unit

4.1 **User Registration Test**
4.2 To ensure the use is able to create an account in the user interface and that account is stored in the database
4.3 This test involves mocking the account registration process by entering a username in the user interface and then checking that the database has saved the username and then checks that the user is able to login with the newly registered accounts.
4.4 Inputs: User registration information
4.5 Outputs: Server will validate the user's details are in the database and let the user login to the application
4.6 Normal
4.7 Whitebox
4.8 Functional
4.9 Integration

5.1 **Full End-to-End Test**
5.2 To validate the entirety of the application works and a user can go through a normal flow of the application.
5.3 This test involves loading the application and then simulating a real world flow that the user would take and making sure elements of the application are loaded, visible, clickable/ can be interacted with and no errors or unexpected behavior occurs.

5.4 Inputs: Login information and clicking on buttons to take the user through the application as a whole.
5.4 Outputs: A fully loaded application with no unexpected behavior to go through an entire, live flow.
5.6 Normal
5.7 Whitebox
5.8 Functional
5.9 Integration

6.1 **Web Browser Compatibility Test**
6.2: To ensure all browsers load the application the same, correct way.
6.3 This test involves loading the application and checking that all elements of the application load and appear correctly on all main browsers such as Chrome, Firefox, Edge, Safari, ect.
6.4 Inputs: User login data, application requests.
6.5 Outputs: A fully loaded application with the correct functionality and appearance.
6.6 Normal
6.7 Blackbox
6.8 Functional
6.9 Integration

7.1 **Animation Load Test**
7.2 To verify that animations are displayed successfully.
7.3 This test will load the application and determine if the module animations are correctly loaded and displayed on module pages for different browsers and sizes.
7.4 Inputs: URLs of module pages with animations.
7.5 Outputs: Fully and successfully loaded animations.
7.6 Normal
7.7 Blackbox
7.8 Functional
7.9 Unit

8.1 **Animation Interaction Test**
8.2 To assess the application’s responses to anticipated user input.
8.3 This test will load the application at its module pages containing animations. It will simulate user interactions via cursor/keyboard by calling the methods written to respond to such user inputs and evaluate the results against the intended behavior.
8.4 Inputs: URLS of module pages with animations, user interaction response methods
8.5 Outputs: Reactions displayed in the animation as according to user input.
8.6 Normal
8.7 Whitebox
8.8 Functional
8.9 Integrated

Hackademia
Emily Bever, Ryan Connolly, Robin Heft, Molly Vongsakhamphouy

9.1 **Performance Stress Test**
9.2 To determine if the application can handle a large amount of traffic at once
9.3 This test will utilize different tools to flood the application with web traffic. We have not decided on a testing platform yet, but the options include: ApacheBench, k6, and loader.ai.
9.4 Input: Web application URL into the load testing tool
9.5 Output: Results provided through the tool indicating how the application handled the stress
9.6 Normal
9.7 Whitebox
9.8 Performance
9.9 Integration

10.1 **Code Coverage Test**
10.2 To quantify how much of the codebase is covered by unit tests
10.3 This test will determine how much of the code is tested. It will highlight any untested code, which allows the development team to adjust the testing strategy to become more robust. Tools such as Istanbul or Jest are options to conduct this test.
10.4 Input: code base
10.5 Output: percentages of code covered by unit tests and other analytics
10.6 Normal
10.7 Blackbox
10.8 Performance
10.9 Unit

Test ID	Normal/ Abnormal	Blackbox/ Whitebox	Functional/ Performance	Unit/ Integration
1	Normal	Blackbox	Functional	Unit
2	Normal	Blackbox	Functional	Integration
3	Abnormal	Whitebox	Functional	Unit
4	Normal	Whitebox	Functional	Integration
5	Normal	Whitebox	Functional	Integration
6	Normal	Blackbox	Functional	Integration
7	Normal	Blackbox	Functional	Unit
8	Normal	Whitebox	Functional	Integration
9	Normal	Whitebox	Performance	Integration
10	Normal	Blackbox	Performance	Unit