COSC 310 Milestone 4- The Learning Layers

Group members:
- Oluwadabira Omotoso (84518448)
- Afua Frempong (90434176)
- Karim Khalil (38485272)
- Peter Idoko (89385496)
- Atharva Jagtap (39283783)

Testing Implementation:

For our project built on the MERN tech stack, we've employed Jest as our primary testing framework. Jest has proven to be invaluable in testing and mocking various functionalities across our application. Here are the approaches we've used during our testing phase, along with examples of how we've employed them:

Mocking Fetch Requests:
- Whenever we need to interact with our database, particularly through HTTP requests like POST, we mock the fetch calls in our tests. This ensures that we can simulate different scenarios and responses without actually hitting the server.

Mocking Changes in Window Location:
- For pages that don't have forms but require redirection or navigation, we mock changes in the window's location. This allows us to simulate page transitions without actually navigating away from the test environment.

Mocking Clear:
- In scenarios where we need to test functionalities that involve clearing certain data or state, we use mocking to simulate the clearing process.

Spying On Functions:
- We use Jest's spying capabilities to monitor function calls, allowing us to assert whether certain functions were called and with what arguments.

Button Click Testing:
- We ensure that buttons behave as expected by testing their click events and associated functionalities.

Aside from only creating jest tests, we have included automated testing in our GitHub. Every time a pull request is made to the main, there is a constant check that ensures that with any new changes, the tests pass.

Steps in the next weeks to improve testing:

- **Increase Test Coverage**:
  - Identify areas of our application with low test coverage and prioritise writing additional tests for these areas.
  - Aim to achieve a higher percentage of code coverage to ensure that critical parts of your application are thoroughly tested.
- **Refactor and Simplify Tests:**
  - Review existing test suites for redundancy, complexity, and duplication.
  - Refactor and simplify tests to improve readability, maintainability, and execution speed.
  - Consider breaking down large test cases into smaller, more focused tests to isolate and identify issues more efficiently.
- **Review and Learn from Test Results**:
  - Regularly review test results and analyze any failures or issues encountered during testing.
  - Use these insights to identify patterns, root causes, and areas for improvement in both the application code and test suites.

Update on functionalities tested:

Our team has conducted thorough testing on several key functionalities, such as account creation and login procedures. We have also performed targeted testing on specific features like the course creation page to ensure that administrators can create courses successfully. Furthermore, we have tested the pending enrollments page to verify that students are either accepted or rejected from enrolling in a course. Additionally, we have ensured that teachers can create new assignments and view the courses they are teaching. We have also tested the functionality of students enrolling in a course and viewing the course information for the courses they are enrolled in. Finally, we have confirmed that a teacher can create an assignment.

Automation Implemented:

In our project, we have implemented automation for unit tests to streamline our development process and ensure code quality. We have integrated a robust unit testing framework into our CI/CD pipeline, which automatically runs tests whenever changes are made to the codebase. This ensures that any new code adheres to our standards and does not introduce regressions.

Additionally, we have automated the process of reviewing and merging pull requests. Whenever a team member creates a pull request, our automation system kicks in, running the unit tests against the proposed changes. This helps us quickly identify any potential issues or conflicts early in the development cycle. Overall, the

automation initiatives have significantly improved our development workflow, reducing manual intervention, and allowing our team to focus more on coding and less on administrative tasks. The automated testing and review processes have also enhanced the reliability and stability of our software releases.

Summary

So far we are making good progress in terms of development and teamwork. However, a few weeks into our first sprint cycle, we realised that our project timeline did not allow us enough time for troubleshooting, and we were not on track as we had hoped. To address this, we decided to create a weekly project timeline document in which we adjusted weekly marking off completed functionalities and highlighting ongoing issues that need to be carried into the following weeks. We also divided our group into front-end and back-end development to help accelerate our issue completion time. As a result, we can complete the majority of our weekly assigned issues, adjust our timeline accordingly, and focus on having the front and back end of our site cohesive.

As we continue, we have recognized the importance of thoroughly testing our code and addressing potential issues. Currently, we ensure the quality of our code by using Jest, reviewing pull requests in person during our weekly scrum meetings, creating new branches when necessary, and documenting our code thoroughly. Code documentation is especially important for us because it ensures that team members from different areas, like front-end and back-end, can understand each other's thought processes when reviewing code. Additionally, Jest plays a crucial role in maintaining the quality of our code by ensuring that only clean code that has passed our implementation standards can be merged into the main branch. Moving forward, we aim to continue addressing potential issues in our code logic, communicating in person, documenting our code, and carefully reviewing each other's pull requests.