Testing Update Report

Github: https://github.com/cartercsm9/codeCrafters/

Team Members:

Omar Hemed (39938816)

Karam Hejazin (70825294)

Tyler Cummings (14814578)

Rhythm Trivedi (41908864)

Carter Meekison (90445313)

Update on Testing Implementation

- Approaches Employed:
    - Unit Testing for `getCityName` Function:
        - Method: Stubbing axios.get calls, utilizing Sinon to simulate API responses. This approach is essential for testing asynchronous operations and ensures that the unit tests are focused and isolated, preventing external API call dependencies.
    - Integration Testing for User Authentication Endpoints:
        - Method: Employing Chai-HTTP for interaction with server endpoints.
        - Features tested: `/users/signup` for new user creation and `/users/login` for existing user authentication, with a cleanup process for dummy users.
    - Integration Testing for Weather API Endpoints:
        - Method: Utilizing Chai-HTTP for testing various weather-related endpoints.
        - Features tested: Endpoint functionality for retrieving and inserting weather data based on city names.

Future Improvements:

- Expansion of test coverage to include more edge cases, boundary conditions, and error scenarios.
- Implementation of parameterized tests to optimize the testing process for similar scenarios.
- Enhancement of test documentation for better clarity and maintenance.

Update on Functionality Testing

- Current Status:
    - Successful API response testing in unit testing phase.
    - User authentication processes (signup/login) validated through integration testing.
    - Weather API endpoints tested for functionality related to data retrieval and forecast insertion.
- Areas for Improvement:
    - Focus on improving error handling mechanisms within tests.
    - Extend testing scope to cover more functionalities of the project.

Automation Implemented

As part of our efforts to streamline the development and testing process, we have set up a GitHub Actions workflow named "Node.js CI with Docker". This automation is triggered by pull requests or pushes to the `main` branch, facilitating continuous integration by building and testing the project in an isolated Docker environment.

Here's an overview of the workflow steps implemented:

- Trigger Conditions:
    - The workflow is activated on `push` and `pull_request` events targeting the `main` branch, ensuring code changes are continuously integrated and validated.
- Build Environment:
    - Runs on `ubuntu-latest`, utilizing Docker services to create a consistent testing environment.

    - Docker version 19.03.12 is used, with ports 3001 and 3306 mapped to ensure web and database services are accessible.
- Workflow Steps:
    - Code Checkout: Utilizes `actions/checkout@v2` to fetch the repository code.
    - Docker Build Setup: Establishes Docker Build using `docker/setup-buildx-action@v1` for advanced build capabilities.
    - Docker Layers Caching: Caches Docker layers with `actions/cache@v2`, optimizing build times by reusing data.
    - DockerHub Login: Authenticates to DockerHub using credentials stored in GitHub secrets, enabling image push/pull capabilities.
    - Build and Run Containers: Executes `docker-compose up -d` to build and start the project containers, setting up the runtime environment.
    - Database Readiness Check: Ensures the database is fully operational before proceeding, avoiding connection errors during testing.
    - Execute Tests: Runs the project tests within the Docker container to validate functionality.
    - Cleanup Process: Tears down the environment using `docker-compose down` to ensure a clean state for subsequent runs.

This automated process not only ensures that each build is consistent but also validates the integrity of the code and its functionality before it merges into the main codebase, thus maintaining a high standard of quality and reliability in our development cycle.

Summary of the project progress:

The project is progressing smoothly with a well-organized team structure. Omar and Rhythm are responsible for the front end, focusing on HTML and CSS, while Tyler and Carter handle the back end, specifically database operations and PHP development. Karam is actively involved in testing, managing alerts, and providing support to team members as needed.

Process Evaluation:

The selected team process appears to be working effectively, with clear distribution of responsibilities between front-end and back-end tasks.
Regular communication and coordination among team members are crucial to ensuring a smooth workflow. It's important to maintain open channels for discussion and updates on progress.
Continuous refinement of the process may be beneficial, such as implementing regular check-ins or status updates to track progress and address any issues promptly.

Source Code Quality Assurance:

Ensuring the quality of source code is vital for the project's success. The team can adopt several strategies to maintain code quality:
Conducting code reviews: Regularly reviewing each other's code helps identify bugs, improve coding standards, and share knowledge among team members.
Automated testing: Implementing automated testing, such as unit tests for PHP scripts and integration tests for database interactions, can catch errors early in the development cycle.
Code documentation: Writing clear and comprehensive comments within the codebase helps other team members understand the functionality and purpose of different components.
Code standards and guidelines: Establishing and following coding standards and guidelines ensures consistency and readability across the codebase.
Improvement Initiatives:

The team can schedule regular code review sessions to discuss code improvements, address any technical debt, and share best practices.
Implementing continuous integration and deployment (CI/CD) pipelines can automate code testing, integration, and deployment processes, reducing manual errors and ensuring code stability.

Encouraging knowledge sharing sessions or workshops within the team can enhance skills and promote a collaborative environment.

Overall, the project is progressing well, and by incorporating these quality assurance measures and improvement initiatives, the team can further enhance the efficiency and quality of the project deliverables.

Use Cases and Requirements Backlog

The application allows users to view current weather conditions, access historical weather data, compare weather conditions, receive severe weather alerts, and customize display settings.

- Setup and configure the web server. **COMPLETED**
- Develop the database schema for storing weather data. **COMPLETED**
- Implement API requests for fetching current weather data. **COMPLETED**
- Design and implement the front-end interface. **COMPLETED**
- Implement User and Admin Login. **COMPLETED**
- Integrate SQL database connection to server. **COMPLETED**
- Integrate historical data retrieval and display functionality. **COMPLETED**
- Add weather alert system for emergencies
- Add user customization options for weather data display. **COMPLETED**
- Test the application for usability, performance, and security. **IN-PROGRESS**
- Deploy the application and monitor for issues.