Software Implementation and Testing Document

For

Group 4

Version 3.0

Authors:
Amanda Orama
Matthew Cegala
Nicholas Holguin
Matthew Hummel
Ashton Singpradith

1. Programming Languages (5 points)

Frontend

- JavaScript (React) - Used for building the user interface, pages, and reusable components. Chosen for its efficiency in creating interactive UIs and managing state.

Backend

- C# - This is used for the class libraries and interacting with the ASP.NET API services that are being utilized.

Database

- SQL – This is the most common and foundational language used with MySQL databases.

2. Platforms, APIs, Databases, and other technologies used (5 points)

Frontend

- React- For building the UI and reusable components. (Workout, biometric, admin pages)
- Tailwind CSS- for styling components.
- Browser local storage (for now)- Temporarily storing biometric and placeholder data until API integration
- Figma- For building low and high-fidelity prototypes for UXUI

Backend

- ASP.NET Core Web REST API – For communication between the frontend and database in a consistent JSON packet format.

Database

- MySQL – Database management system chosen to help store and retrieve data.

Source Control

- Git – Source code management and team collaboration.

3. Execution-based Functional Testing (10 points)

- The frontend was tested using mock data to verify that all pages and components functioned correctly. This included forms for login, registration, forgot password, admin homepage and admin management pages ensuring buttons, dropdowns, and navigation flows behaved as expected.
- The backends execution-based functional testing was done by entering mock data into the front-end and verifying via Network, that the requests were being sent successfully to the backend. Once this was confirmed, the database was checked for the correct front-end data being stored, updated, retrieved, and deleted properly through the CRUD endpoints. After this verification, it was clear that there was functionality between the front-end, backend, and database.
- The database functional testing performed was by creating a simple test console program that was responsible for calling each function in the respective database libraries at least once to ensure proper functionality when sending and receiving data from the database. There was a DML script that loads test data and performs tests that allow for calls to each function to ensure proper requests on multiple entries. This program was created using the white box testing methodology for knowing the internal workings of the classes and libraires that allow for proper database communication. This successfully tested the working of all CRUD, create, read, update, and delete functionality required by the database to meet the functional requirements outlined. These test programs can be found in the ./MySQL.SmartGymTracker/TestDB/* file path.

4. Execution-based Non-Functional Testing (10 points)

- The frontend was checked for consistent layout, proper alignment of components, and adherence to UX principles such as clear navigation, error messages, and user control. Mock data allows testing of interactions and flows without a fully functional backend.
- The backend execution-based non-functional testing involved verifying performance, reliability, and error handling. For the CRUD operations, the response times were nearly instant, showing the performance was highly efficient. The reliability was tested by sending multiple requests mixed with invalid inputs to confirm whether there were proper error responses.
- The database was tested for non-functional requirements by ensuring that scalability and security were met. The database ensured that no passwords were ever sent outside the database during the testing of the test program created. The database also ensures reliability by performing local and manual testing of duplicate entries of primary keys and invalid entries based on the low-level data validation. This was done to ensure that nothing would cause complete database corruption and that reliability was maintained. There were also dummy scripts and manual spam entries created to ensure that multiple updates and gets would not break the database. Due to the database being hosted on MySQL systems, it has strong reliability built in to support the non-functional testing goals outlined.

5. Non-Execution-based Testing (10 points)

During the initial planning phases of the design process, we outlined the clear goals and outcomes expected for the code to be accepted. We determined that in order for code to be merged into the respective increment branch, it would require at least one other person to perform static code review and verify that the code meets the outlined goals and standards for the ticket. These static code reviews were intended to be checking for anything that might have interfered with another person's code or to verify that everyone is aware of what is being added to the codebase. By utilizing GitHub's pull request feature and branch rules being in place to ensure at least one user is reviewing before merging to increment or master branches upstream, we have ensured that proper code validation techniques are being applied. On top of code validation being performed, there were also reviews of all documentation and front-end wireframes to ensure that everyone was satisfied with the design elements.