Software Implementation and Testing Document

- Increment 1 -
Group #2

Authors:

Daivion Brooks
Brooks Berry
Jason Lee
Nadav Peper
Nandagopal Nair

1. Programming Languages (5 points)

TypeScript (Frontend)*: Used with React to build the user interface. We chose TypeScript for its type safety and ability to reduce runtime errors, making development more manageable for a group project.*

Python (Backend): *Used with Flask to build the API. Python was chosen because it is beginner-friendly, flexible, and integrates well with Flask for web application development.*

SQL (Database): *Used mainly through SQLAlchemy to manage the Postgres database and handle queries. SQLAlchemy made it easier to work with structured data and store user, symptom, and treatment information.*

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

React*: JavaScript/TypeScript library used for building the frontend.*

Flask*: Lightweight Python framework used for the backend and API development.*

PostgreSQL*: Relational database used for persistent storage of user accounts, symptoms, and treatments.*

Docker*: Used for containerization to ensure consistent environments across team members' machines and deployment.*

JWT (JSON Web Tokens)*: Used for secure authentication and session management.*

Marshmallow: *Used for serialization and validation of data in the backend.*

Postman*: Used for testing backend API routes during development.*

GitHub*: Used for version control and collaboration between team members.*

3. Execution-based Functional Testing (10 points)

We tested functionality directly against the requirements defined in our RD document. Postman was used to test API endpoints such as registration, login, adding symptoms, retrieving symptoms, adding treatments, and retrieving treatments. Each endpoint was tested with both valid and invalid inputs to ensure proper error handling. On the frontend, we don't have an api call implemented currently. Database initialization was also tested by starting a fresh local environment and applying migrations with flask db upgrade to confirm tables were created correctly and data could be stored and retrieved.

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

Security: *Verified that passwords were hashed in the database and that JWT authentication prevented unauthorized access when tokens expired.*

Performance: *Measured response times using Postman and browser developer tools. All API responses returned within 1–2 seconds, meeting our performance goals.*

Usability: *Informal testing was performed by navigating the frontend to ensure pages were accessible, readable, and the interface was clear.*

Reliability: *Database migrations were tested in a new environment to ensure that tables were created correctly and data could be stored and retrieved without issues.*

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

Code Reviews: *Team members reviewed pull requests on GitHub before merging to ensure code quality and reduce errors.*

Walkthroughs: *During meetings, we walked through key features such as login and symptom tracking to confirm group understanding and gather feedback.*

Static Checking: *TypeScript's compiler caught many potential type errors in the frontend. Python linters and IDE warnings helped us maintain backend code quality.*

Manual inspection: *Team members periodically inspected the codebase to catch typos, syntax issues, and simple mistakes, and refactored parts of the code where improvements were needed.*