Software Implementation and Testing

Document

For

Group <25>

Version 1.0

Authors:

Erik P, Noah L, and Keene M

**1. Programming Languages:**

- We are using Python for our backend development: Python is utilized to develop the server-side logic of the application using the Flask web framework. It handles routing, data processing, user authentication, and integration with the PostgreSQL database. We chose python because of its simplicity and readability make it the perfect choice for backend development with Flask. Flask allows flexibility and scalability. Python offers a myriad of libraries and extensions (e.g., Flask-Login, SQLAlchemy, Bcrypt, flask-migrate) that we will utilize to help set up that database and the user authentication for login.

- We are using HTML for the front-end development. HTML is used to structure the content and layout of the web application's frontend, defining elements such as headers, footers, navigation bars, and content sections. HTML is the foundational language for creating web pages, ensuring compatibility and accessibility across all browsers and devices. We also will utilize CSS3, in conjunction with Bootstrap. We want to style the web application's frontend with a more modern look. CSS3 provides extensive styling capabilities, allowing for a visually appealing and responsive user interface. We are utilizing Bootstrap's pre-defined CSS classes to enhance development speed and ensure a consistent design language.

**2. Platforms, APIs, Databases, and other technologies used:**

- Flask (Web Framework):

-Backend Development: Flask serves as the core web framework for handling HTTP requests, routing, session management, and integrating with other backend components. Utilized with extensions like Flask-Login for user authentication and Flask-Migrate for database migrations. We chose Flask since it is lightweight and flexible: Flask's minimalistic approach allows for customized development tailored to the project's specific needs. Also, it is easily scalable to accommodate future features and increased user load.

- PostgreSQL (Database):

- PostgreSQL is used as the primary relational database for storing user information, race data, strategies, and other persistent data. Integrated with SQLAlchemy for object-relational mapping, facilitating seamless database interactions.

- Bootstrap (CSS Framework)

- Primary purpose is front-end design. Bootstrap is employed to design a responsive and aesthetically pleasing user interface, leveraging its grid system, pre-defined components, and utility classes.

- Flask-Migrate (Database Migrations)

- Flask-Migrate will be used to handle database schema changes, allowing for version-controlled migrations and seamless updates to the PostgreSQL database.

-GitHub (Version Control and Collaboration)

- GitHub is used to manage the project's source code, track changes, collaborate among team members, and host the project's repository. It is also beneficial for issue tracking.

-Flask-Login (User Authentication)

- We are using Flask-Login to manage user sessions, handling login, logout, and session persistence within the Flask application.

- SQLAlchemy (ORM)

- SQLAlchemy facilitates interactions with the PostgreSQL database through object-relational mapping, allowing us to work with Python objects instead of writing raw SQL queries for every database request/update.

- BCrypt:

- BCrypt is used to hash passwords for securely storing user passwords. This will be necessary for user accounts and administrators accounts.

- OpenF1 API

- The OpenF1 API is the service used to compile all of the necessary information into one place. The API allows us to filter through their database and extract what is needed for our simulation,

**3. Execution-based Functional Testing:**

- We have not done any functional testing yet but in the future we plan to by performing Unit tests. We plan on testing the integrity and behavior of database models, ensuring that validations, relationships, and CRUD operations function correctly. We also will verify the correctness of utility functions and helper methods used within the application. Some examples of future unit testing would be testing model classes such as Race and User model. We would do this by creating a user with valid data that succeeds and test invalid data (e.g., missing email) that raises appropriate exceptions/errors.

**4. Execution-based Non-Functional Testing:**

- We have not done the test yet but for the future we will do Performance Testing and evaluate how the application performs under expected user loads, ensuring responsiveness and stability. Also, we will need to run response time tests to ensure the time taken to process requests and render pages, aiming for minimal delays. We will also utilize tools like Flask-Profiler to monitor request processing times

**5. Non-Execution-based Testing:**

- We have not performed non-execution based testing yet but for the future we will by utilizing peer reviews where each team member reviews code to identify potential issues, ensure coding standards are met, and make sure the app is on track. We will also utilize standardized checklists focusing on readability, efficiency, security, and adherence to project guidelines. We can monitor these by adding issues to Github and completing those issues throughout the development process.