Software Implementation and Testing Document For Group <25> Version 3.0

Authors:

Erik P, Noah L, and Keene M

1. Programming Languages:

- In this increment, we have deepened our utilization of Python for backend development. Leveraging the Flask web framework, we have successfully established routing mechanisms, implemented user authentication processes using Flask-Login, and integrated the PostgreSQL database through SQLAlchemy ORM. Python's simplicity and readability have facilitated efficient development, allowing us to create maintainable and scalable server-side logic. Additionally, we have incorporated essential libraries such as Bcrypt for secure password hashing and Flask-Migrate for streamlined database migrations.

- Our frontend development has progressed with the creation of structured and responsive web pages using HTML. We have enhanced the visual appeal and user experience by applying CSS3 alongside Bootstrap's predefined classes. This combination has enabled us to design a modern and consistent user interface across different devices and browsers. Although certain frontend features like badges and detailed race listings are not yet implemented, we have established the foundational layout and styling, ensuring readiness for future enhancement.

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

- Flask (Web Framework):

  Flask remains the core of our backend development, handling HTTP requests, session management, and routing. With extensions like Flask-Login for managing user sessions and Flask-Migrate for database migrations, Flask provides the flexibility and scalability necessary for our application's growth. Its lightweight nature has allowed us to customize our development process to align with project-specific needs effectively.

- PostgreSQL (Database):

  PostgreSQL has been successfully integrated as our primary relational database. Utilizing SQLAlchemy for object-relational mapping, we have established robust interactions between our application and the database. This setup ensures efficient storage and retrieval of user information, race data, and other persistent entities critical to the application's functionality.

- Bootstrap (CSS Framework):

  Bootstrap has been instrumental in designing a responsive and aesthetically pleasing frontend. By leveraging its grid system, pre-defined components, and utility classes, we have accelerated the development process while maintaining a consistent design language throughout the application.

- Flask-Migrate (Database Migrations):

  Implementing Flask-Migrate has allowed us to manage database schema changes systematically. This tool ensures that migrations are version-controlled and can be applied seamlessly, preserving data integrity during updates and modifications.

- Flask-Login (User Authentication):

  Flask-Login has been integrated to manage user authentication processes, including login, logout, and session persistence. This extension ensures secure handling of user sessions, enhancing the overall security of the application.

- SQLAlchemy (ORM):

  SQLAlchemy has been pivotal in simplifying database interactions through object-relational mapping. It allows us to work with Python objects instead of writing raw SQL queries, thereby increasing development efficiency and reducing potential errors.

- BCrypt:

  We have employed BCrypt for hashing user passwords, ensuring that sensitive information is stored securely. This implementation is crucial for protecting user data and maintaining the application's integrity.

- OpenF1 API:
  The OpenF1 API has been integrated to compile and retrieve necessary race information for our simulations. This API enables us to filter and extract relevant data, providing the backbone for our race prediction algorithms.

3. Execution-based Functional Testing:

We have begun functional testing in this increment, and have performed unit tests that ensure the accuracy of our database models and API endpoint scraping. We have tested the API's data against our hand-gathered work, ensuring that it matches within the 1000ms period we expect, and have run our simulation against real-world F1 race results and found it to be accurate within ~2% of real race times, a significant milestone in terms of accuracy.

4. Execution-based Non-Functional Testing:

Non-functional testing has been a focus of this increment. We have ensured that the database connection persists for extended periods of use by testing connections at various points throughout the day. The API scraping has been accurate to the second for each endpoint we have looked at. We have timed the responsiveness of the page requests and rendering, and the slowest has been around 9000ms on average to scrape all data required and run the simulation.

5. Non-Execution-based Testing:

We have begun the peer review process of each others' code, ensuring that bugs are eliminated. This process has led to several minor issues being caught, from improper datetime formats, to more minor bugs like variables converting from strings instead of integers, leading to false inequality assertions.