Software Implementation and Testing

Document

For

Group <25>

Version 2.0

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 pre-defined 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:**

- While functional testing remains pending in this increment, we have laid the groundwork for comprehensive testing in future phases. Our plans include that **w**e intend to perform unit tests to verify the integrity and behavior of our database models, ensuring that validations, relationships, and CRUD operations function as expected. For instance, we will test model classes such as Race and User by creating instances with both valid and invalid data to confirm that appropriate exceptions and errors are raised when necessary.

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

- We have not done a lot of Non-functional testing yet in this increment. However, we have outlined plans to undertake the following in the next phase: **-** Evaluating how the application performs under expected user loads to ensure responsiveness and stability. **-** Measuring the time taken to process requests and render pages, aiming for minimal delays to enhance user experience. Utilizing tools like Flask-Profiler to monitor and optimize request processing times, ensuring efficient backend performance.  
 - So far we have done small tests on how the database persists but that is it so far.  
 - Scraping the API and ensuring the data accurately represents the very moment in time it is supposed to fetch data from has been tested across several points in time, which has revealed bugs in the code or demonstrated accurate functionality.

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

- In this increment, we have actively begun utilizing GitHub's issue tracker to manage and monitor project tasks and potential issues. This system allows us to organize tasks effectively, prioritize work, and ensure that identified problems are addressed promptly. Additionally, our team is preparing to implement peer reviews in the near future. Peer reviews will involve structured code evaluations by team members to identify potential issues, ensure adherence to coding standards, and maintain high-quality code across the project. This practice will enhance collaboration, foster knowledge sharing, and contribute to the overall robustness of the application.