**Software Implementation and Testing Document**

**For**

**Group 9**

Version 1.0

**Authors**:

Kaitlyn Krause

Tara Kerstetter

Megan Cole

Brandon Pina

# Programming Languages (5 points)

*List the programming languages use in your project, where you use them (what components of your project) and your reason for choosing them (whatever that may be).*

* Python
  + We’re using Python for the backend server and data processing
  + We chose Python due to its simplicity and extensive libraries. It is a good choice for rapid development.
* JavaScript
  + We’re using javascript for game logic and frontend game development
  + We chose Phaser specifically for 2D game development for our typing minigames
* HTML/CSS
  + We’re using HTML/CSS for frontend design
  + HTML and CSS will be used to structure and style the non-game aspects of the web interface.

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

*List all the platforms, APIs, Databases, and any other technologies you use in your project and where you use them (in what components of your project).*

* PostgreSQL
  + We are using PostgreSQL as our database for user profiles, typing statistics, and game data
* Phaser
  + We are using Phaser for our typing games and game logic
* Django
  + We are using Django as our web framework
* EC2 (AWS):
  + The platform that is hosting our project provides the virtual server on which all the project components are running.
* Gunicorn
  + It acts as the Python WSGI HTTP server that interfaces with our application. It manages all incoming HTTP requests and passes them on to the app.
* Nginx
  + Reverse proxy server in front of Gunicorn. It handles load balancing and the service of static files.

# Execution-based Functional Testing (10 points)

*Describe how/if you performed functional testing for your project (i.e., tested for the* ***functional requirements*** *listed in your RD).*

* Tracking user typing
  + We conducted manual tests by performing typing tests and reviewing the data. We used this to verify that the application tracks all keys pressed, and mistakes, and calculates Words Per Minute accurately.
* Sentence generation
  + We manually validated that the generated sentences met the expected complexity requirements per difficulty level and didn’t repeat.
* Statistical calculations
  + We checked the accuracy of calculated statistics like Words Per Minute and accuracy percentage over multiple sessions. We did this by comparing the output with the expected output.
* User information and statistics storage
  + We verified that user data was stored correctly by testing save and retrieval operations on our database. We also used negative testing to make sure the system responded appropriately to errors.
* Host multiple mini-games
  + We confirmed that all mini-games launched and operated correctly by performing manual tests by playing each game to completion multiple times.
* Different practice modes in type test
  + We tested each practice mode individually and compared the modes to each other to ensure that the modes were achieving the specific difficulty we wanted.
* Battle Pass feature
  + We have no testing for this feature because it has not yet been implemented
* Multi-player
  + We have no testing for this feature because it has not yet been implemented

# Execution-based Non-Functional Testing (10 points)

*Describe how/if you performed non-functional testing for your project (i.e., tested for the* ***non-functional requirements*** *listed in your RD).*

* Password Encryption
  + We examined our database to ensure that no passwords were stored in plain text and confirmed that our encryption algorithm is applied to both registration and login processes. We also implemented negative testing to ensure that attempting to access user accounts without correct credentials to make sure that encryption will not be bypassed.
* Ability to handle multiple users at once
  + We conducted load testing using tools like JMeter to simulate multiple users accessing the application simultaneously. Stress tests were performed to identify the maximum number of concurrent users the system could support without significant lag or crashes. We observed server performance, response times, and error rates during peak load times to ensure smooth operation for multiple users.
* Reliability through maintaining a system uptime of 99%
  + We monitored system uptime using automated monitoring tools, like AWS CloudWatch to track system availability over an extended period. We conducted failover testing to ensure the system could recover from unexpected failures.

# Non-Execution-based Testing (10 points)

*Describe how/if you performed non-execution-based testing (such as code reviews/inspections/walkthroughs).*

Code Reviews:

We conducted structured code reviews within our team in order to catch bugs and improve the overall quality of our code. We did this by having each team member review code written by others focusing more on logic and readability than style guidelines. We did this during our weekly meetings. We were able to use our GitHub commits to track this.

Walkthroughs:

We conducted informal walkthroughs where we would explain our code and functionality to each other. These were focused on explaining the purpose, logic, and flow of the code and opening up discussion on possible areas for improvement. This helped the entire team with a better understanding of the codebase.