Software Implementation and Testing Document

For

Group 1

Version 3.0

Authors:
Andrew Augustine
Anthony Ingle
Tristan Morris
Cole Warren
Dawson Stoller

1. Programming Languages (5 points)

- Python
    - Usage: Backend
    - Reasoning: Python (+ it's libraries) are easy for everyone to pick up, meaning less time spent training.
- Typescript
    - Usage: Frontend
    - Reasoning: Standard frontend designing language. Has type safety compared to Javascript.
- Bash
    - Usage: Inside Docker Images
    - Reasoning: Image bootstrapping.

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

- Flask - Backend HTTP Framework
- Pytest - Python testing library
- Solidjs - Frontend Javascript Framework
- Docker - Container runtime; used in development and production server
- MariaDB - Database of choice; Integrates with backend to store user data
- DigitalOcean - VPS that hosts our production server
- Doppler - Shared secrets manager
- Github - Shared Codebase
- Auth0 - Authentication service provider
- Nginx - Web server used in production to serve frontend
- Gunicorn - Web server used in production to serve backend
- Stripe - API for payment handling
- Figma - Mockup tool for designing frontend views

3. Execution-based Functional Testing (10 points)

We used testing libraries to make sure that a lot of our functional requirements gave the proper output or result that we expected.

For the backend, we used Pytest alongside Flask's testing suite to ensure that API responses were proper. On top of this, we also used Postman to ensure correct API responses. We're also planning on expanding this to include authentication verification with Auth0.

For the frontend, we used Vitest alongside a custom testing library from SolidJS to ensure that the DOM responds properly when we click on certain buttons, and that the frontend can parse all kinds of responses from the API.

We don't have any tools set up to automate integration testing, and we do the testing manually every release. It's a known flawed procedure.

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

For non-functional testing, we all sat in and walked through the project to make sure that it has a concise and logical flow. We tested the full breadth of user experience and made sure that the backend was prompt and effective in serving the frontend.

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

Code Reviews: Code reviews were an essential part of our non-execution-based testing process. Whenever a developer completed a piece of code, they would create a pull request (PR) and assign it to one to three individuals for review. The reviewers would thoroughly examine the code, focusing on aspects such as code logic, adherence to coding standards, potential bugs, and

overall design quality. They would provide feedback, suggestions for improvement, and identify any potential issues.

Walkthroughs: Walkthroughs involved a collaborative session where the developer would present their code to a group of reviewers. The walkthroughs provided an opportunity for in-depth discussions about the code, its functionality, and potential improvements. The reviewers would ask questions, provide suggestions, and offer insights based on their diverse perspectives. This process facilitated knowledge sharing, identified potential issues, and improved the overall code quality.