**Software Implementation and Testing Document**

**Group <2>**

Version 2.0

**Authors**:

Lucas Albano

Jason Santos

Nick Watts

Chris Santos

Spencer Denis

# 1. Programming Languages

FRONTEND:

We are using React Native, a javascript framework, to create everything on the front end.

BACKEND:

We are using Python through Django’s REST API to interact with our database.

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

FRONTEND:

The main platform we are using is Expo. We are using a managed workflow on expo to simplify the process of designing an IOS and Android app simultaneously. As for API’s, the main ones in use include React Navigation and Native Base. React Navigation allows us to develop the navigation of our app including the drawer menu. From Native Base, we are using some premade components to style the app. React Navigation is embedded throughout the app whereas Native Base is only used when a premade component has a better design than the traditional react native components.

BACKEND:

The framework we are using is Django, and the Django REST Framework. With these two, we are able to to create our api that will interact with a POSTgres database, saving the information of each user, events, and also be able to associate events and users based on which events a user has been to. This backend is intended to reference Classy’s API to authenticate login information and to retrieve user’s fundraising information. If the Classy API is unable to be implemented, then the backend will need to go web scraping for this information.

# 3. Execution-based Functional Testing

FRONTEND:

The Expo CLI allows us to deploy the app and interact with it on our phones through the scanning of a barcode. While the app is running, we can test functionality and detect errors that arise in the code. Because there is a console log, we are also able to debug efficiently as we output messages to detect what is going on.

BACKEND:

We created a couple of dummy users for the frontend so that they can test their logging in functions.

For sign-in/sign-out/check-in/and registration we will implementing unit tests for each of these features. Since they can be isolated as their own models we can test each feature separately to make sure everything is working correctly. the unit test would consist of importing the model, creating an instance of the model, and trying various tasks to see if the ORM is working correctly.

# 4. Execution-based Non-Functional Testing

FRONTEND:

    We had to be able to access data from our backend so we did testing such as logging/checking that the front end was accessing the back end information correctly. Example would be prior to check in verification we had to verify the front end was in fact grabbing the check in code from the back end.

BACKEND:

We can create tests that makes sure when a request comes through, it comes from an actual user and not someone just trying to access the api by having each request need a token to authenticate with. That way, we are not having to send user information every time we need to fetch data from the backend.

# 5. Non-Execution-based Testing

FRONTEND:

    Front end is working on separate pages so we didn’t do code verification for each other, however we had to test several different components to see what worked best for each page. Having to test different input boxes, modals, layouts etc. to see what was best for the UI.

BACKEND:

Since each of us are working on separate modules we don’t have to worry too much about coding on top of each other. But we when we do push code, we ask the other to look it over to see it it makes since and if enough documentation was provided. If need be, other back end members are free to change code to improve functionality.