**Software Requirements and Design Document**

**For**

**Group 2**

Version 2.0

**Authors**:

Lucas Albano

Spencer Dennis

Chris Santos

Jason Santos

Nick Watts

# Overview

**We are developing a mobile application for Dance Marathon here at FSU. It will contain information for upcoming events, such as the time/place/location of each, the points associated with each event that users can add to their profile, and a “check in” feature that users can use to verify they have attended the event. It will also contain profile information including name, email, gathered points, position, and organization.**

**It will also contain an “about” page that lists general information about Dance Marathon. We will also include a page for “Classy”, which is an API that Dance Marathon uses to show users’ fundraising progress. Our page will allow users to hopefully see their fundraising progress directly from the application.**

# Functional Requirements

**Sign in: Once users are already signed up, they have the ability to sign in again as their login information will be stored in our back end database. High priority.**

**Sign up: If a user has no already created an account with the application, they are given the option to create an account with our sign up feature. High priority.**

**Sign out: If a user has already used the sign up feature to create an account, and the sign in feature to login to the application, they will have the ability to sign out of their account within the application. High priority.**

**Check in: Users are encouraged to attend events associated with Dance Marathon to earn “points” to add to their account. Each event will have a “check in” feature that allows them to confirm and verify they have attended that event. On successful check in, the points associated with the event will be added to the user’s account. High priority.**

**View Upcoming Events: Users should be able to land on a dashboard-like feed of events that are upcoming in Dance Marathon. These events will list the event’s name, time, date, location, and how many points the user will receive for attending. High priority.**

**View User Profile: Users should be able to see their own profile, which contains a picture of themself, their name, their position, their points, the money they have raised, and their organization. Mid priority.**

**View User’s Events: Users should be able to see their own events that they have signed up for that have passed or are upcoming. Mid priority.**

**Side Navigation Bar: The user will have access to a navigation bar that will contain links to DM’s About page, the current leader boards, links to DM’s facebook, and links to Classy--the website that students use to make donations.**

**Message Other Users: Dance Marathon staff should be able to send messages to other users on the app. As of now, it is unclear whether only DM staff will have this ability or if it is ubiquitous. Low priority.**

**View Leaderboards: Users will be able to see their team’s raised funds in ranked order compared to other teams, which will be in the form of a list, a bar graph, and a pie graph. Low priority.**

# Non-functional Requirements

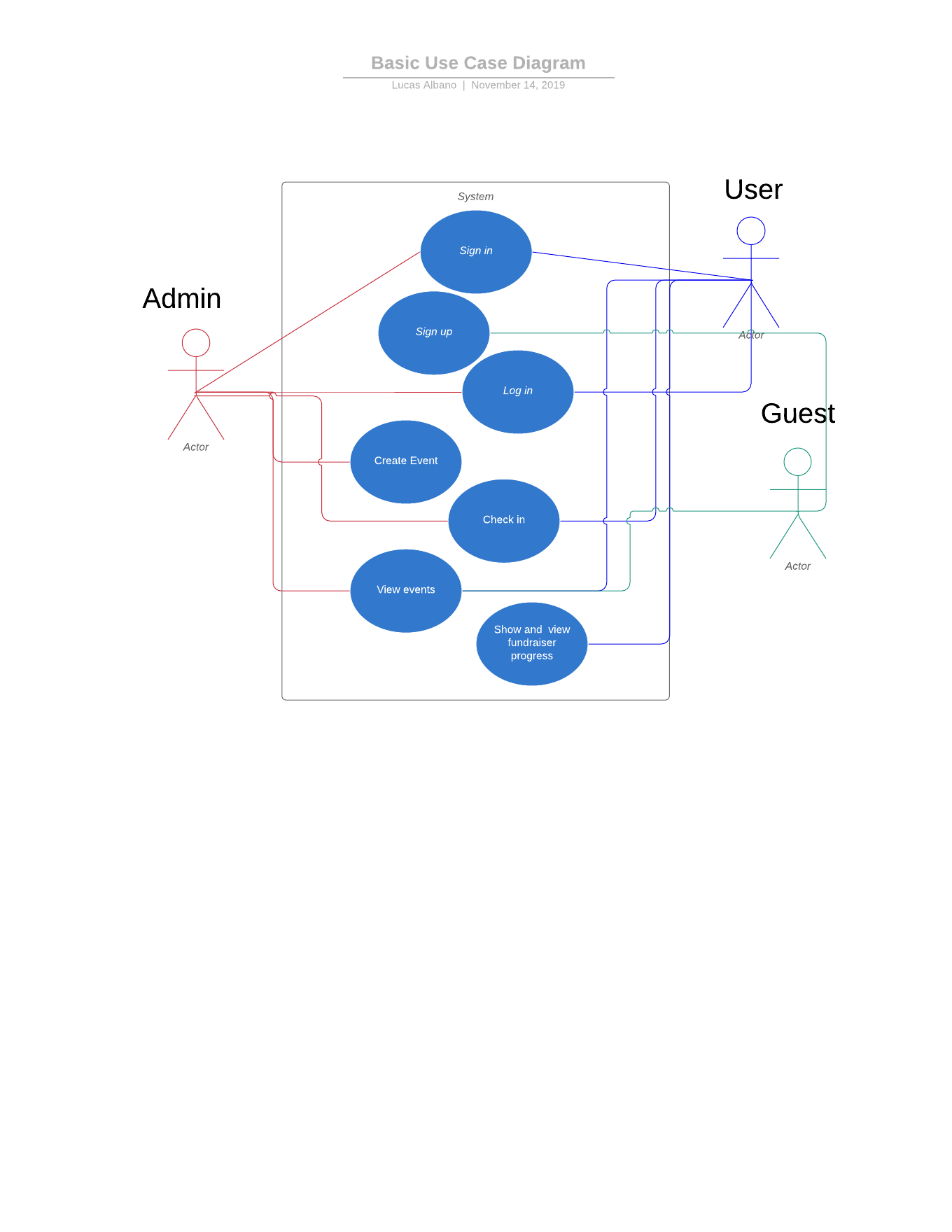
**In the case of security. We will require all API calls to be verified with a session key. Even guests who don’t have accounts will be assigned session keys that can verify they are actually a user of some sorts. The idea for this is so keep unauthorized individuals from accessing the API outside of the app with the intention of malicious wrong-doings.**

**The app has a garnet and gold color scheme with three buttons at the bottom to take them to each section of the app. One section will have their profile, one section will have a list of all upcoming events, and one section will have a list of the user’s events. The look and feel will be simple and uncomplicated, with each screen on the app having a defined purpose with clearly labelled menus.**

**The app is quick enough to load any of its pages within a second of accessing it. However, this is through Expo, and the final app’s performance will be determined by server quality.**

**The app should be reliable enough to accommodate up to one thousand signed in users at once. This is because the app is intended to be used up to the end of Dance Marathon, an event where thousands of people attend and will be checking for updates and upcoming events. The performance of the app will be largely determined by the quality of server, which depends on the money that DM is willing to spend on server space. As of yet, it is unclear how efficiently the app will run with multiple users.**

# Use Case Diagram

**

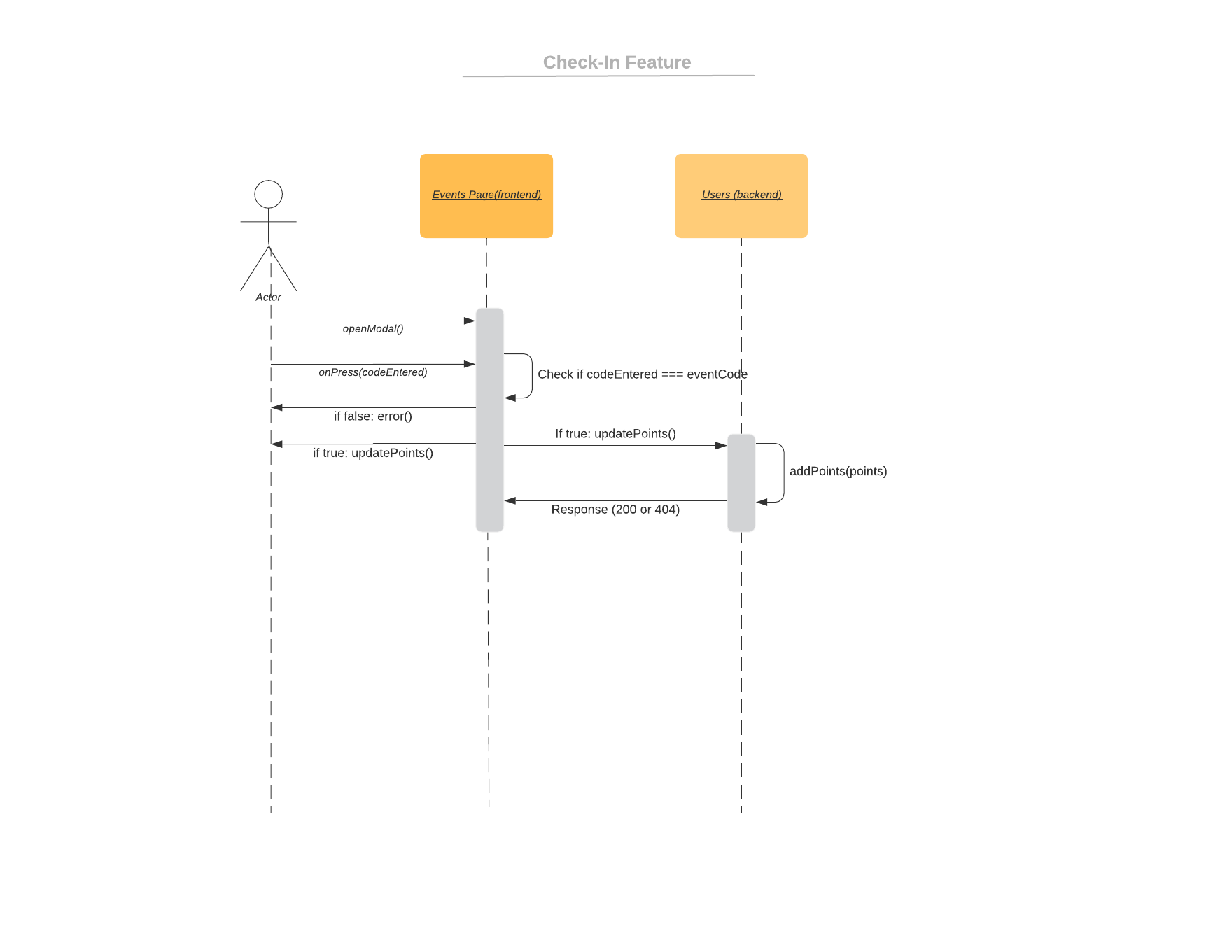
# Class Diagram and/or Sequence Diagrams

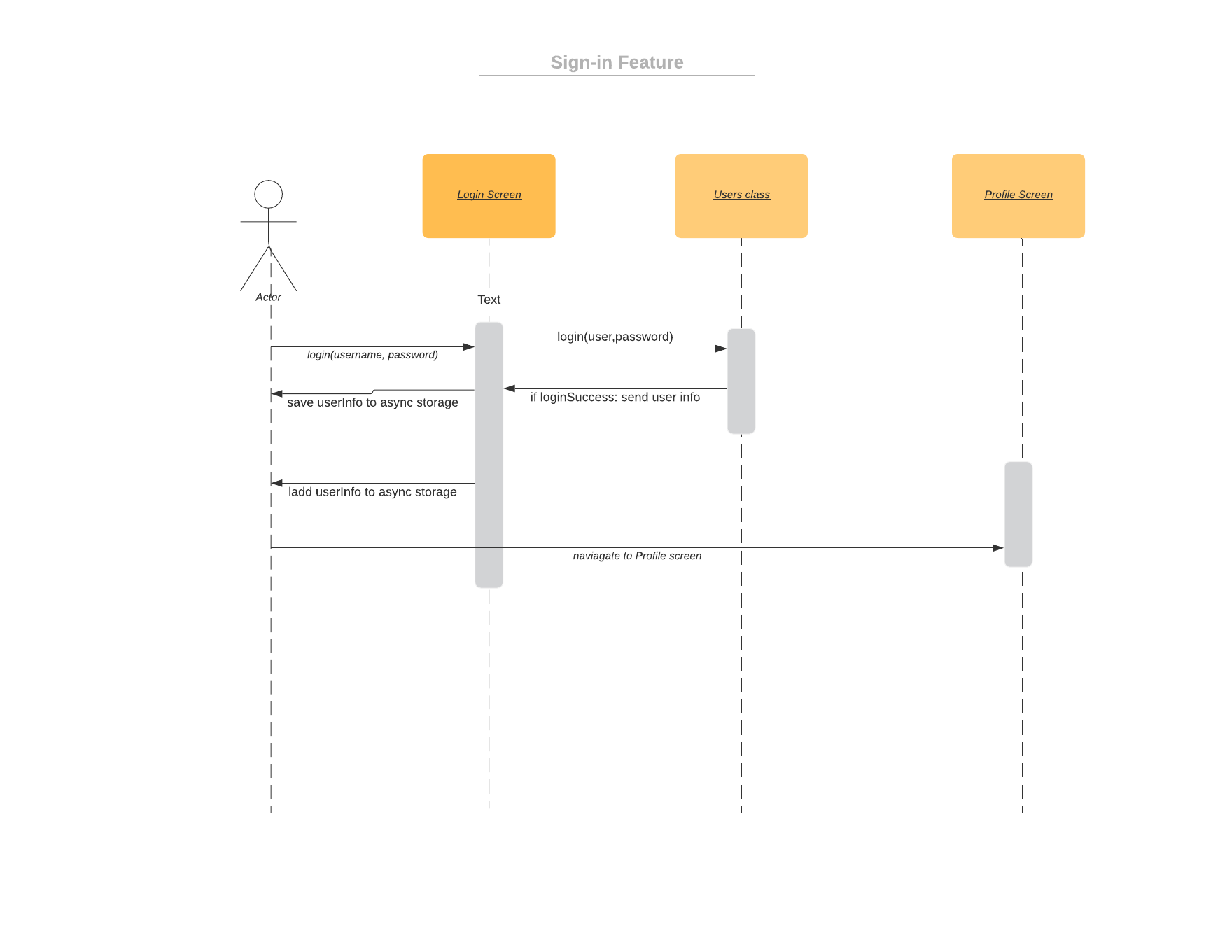
**Class Diagram:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Profile** | **Events** | **About** | **Classy** | **Login** | **Setting** |
| Email  Events  Organization  Points  Id  Signedin | Id  Name  Time  Place  Points  Code  User\_list  codeEntered | description | signedIn  Name |  |  |
| **Constructor()**  **componentDidMount()**  **showGuest()**  **decideScreen()**  **render()** | **Constructor()**  **componentDidMount()**  **lapsList()**  **openModal()**  **closeModal()**  **render()** | **showDescription()**  **render()** | **showProgress()**  **showLeaderboard()**  **render()** | **Render()**  **\_signInAsync()** | **Render()**  **changeName()**  **deleteAccount()**  **changePassword()** |

|  |  |
| --- | --- |
| Signing In | Signing Out |
|  |  |
| Render()  ComponentDidMount() | Render()  ComponentDidMount()  \_signOut() |

**Sequence Diagram:**

****

****

# Operating Environment

**The operating environment is limited to mobile. It will only have a mobile application implementation. However, we decided not to limit our application to just iOS or just Android, we are using a framework (React Native) that allows our application to be cross-platform. Additionally, other software components that must coexist with our application is Classy API which will be used to show user’s fundraising progress.**

# Assumptions and Dependencies

**We are assuming Classy API will be functional and be well documented so we are able to implement easily in our application. Else we are assuming that we will be able to web scrape for the user’s data on Classy’s browser view. We are assuming our client will not change any requirements. We are also assuming async storage does not become fully deprecated (it is currently deprecated but still usable, there is a community version and a non-community version. Expo does not support the community version which is why we are currently using the deprecated non community version).**

**Our dependencies include React Native, React Navigation, Expo, Native Base, Classy API, Django, REST framework.**