OneUp

Design Document

Team 13

Harris Christiansen

Jeremy Craven

Rajan Dalal

Marty Kausas

Adam Loeb

Nicky Semenza

Table of contents

1. Purpose

Social people crave new outlets for interaction in order to share their lives with friends and strangers. Building off the successful components of products like Instagram, Yik Yak, and Yelp, we hope to provide a novel new way for friends and rivals alike to show their competitive spirit in a fun, supportive environment. OneUp strives to appeal to humans' innate desire for competition and victory. The app will allow users to post challenges, break records, rate content, and show off all their accomplishments to friends and strangers.

1.1. Functional Requirements

The functional side of OneUp will be split into 3 main components.

Newsfeed

The newsfeed is a scrollable list of records that can be filtered by local, global, new, and popular. The material here will be highly customized for entertainment purposes and to encourage users to engage with the content.

1. As a user, I would like...
    a. to view challenges that are recent, popular, or global
    b. to be able to see records around me on a map
    c. to express opinion on past records
    d. to be able to see a list of currently running timed events
    e. to be able to quickly discern popular challenges
    f. to be able to see what category a record falls in
    g. to see more graphic content then text while scrolling through a feed
    h. to be able to quickly distinguish challenges I've topped in the feed
    i. to see records that are made in other areas
    j. to be able to report inappropriate challenges and fake users (if time allows)
    k. to be able to use force touch to preview submissions (if time allows)
    l. to be able to have video previews begin playing when I look at them (if time allows)
    m. to be able to carry out all the same actions on the web (if time allows)

2. As a developer, I would like…
    a. to make a web splash page for the app (if time allows)

Profile

Each user will have a semi anonymous profile identifiable by a chosen username. From the profile page, a user will be able to see all kinds of information about himself and other users.

1. As a user, I would like…
    a. to be able to create an account
    b. to be able to link my Facebook account to the app
    c. to save my history and challenges
    d. to be identified by a username
    e. to be able to view my, and others', history
    f. to be able to view a list of my records held
    g. to be notified if a challenge I watch or have broken is broken
    h. to be able to link my Google Plus account to the app (if time allows)

Challenge Details and Creation

Every challenge will have a bunch of information associated with it ranging from a history of record holders, all the way to media proof and user ratings and comments. There will be detailed pages for each challenge as well as a detailed form to create challenges.

1. As a user, I would like…
    a. to be able to view all details about a challenge
    b. to be able to see who held the challenge before me
    c. to know how long each record was held for
    d. to be able to see where challenges have been completed
    e. to be able to save challenges for reference
    f. to be able to place money on a challenge being broken
    g. to be able to put a timer filter on my media to help with content verification
    h. to know about community events regarding certain challenges (if time allows)

2. As a developer, I would like…
    a. user-inputted challenges to follow a structure

1.2. Non-Functional Requirements

In addition to the actions the user will be able to take, they will also expect a smooth experience when inside the app as described by our non-functional requirements.

1. As a user, …
    a. I would like to be able to easily view and upload videos
    b. I don't want to have to wait for more records to load
    c. I don't want to have to wait more than 10 seconds for a video upload
    d. I want a seamless experience without any lag
    e. I want the app to be stable and not experience crashes
    f. I want my account to be secure
    g. I want to be able to use this on iOS
    h. I want to be able to use this on Android
    i. I would like the design style to be consistent between platforms
    j. I would like to be able to use the app without an internet connection (if time allows)

2. As a developer, …
    a. I would like to be able to track app usage
    b. I want the our servers to keep up with load
    c. I want the ability to push to staging and production servers
    d. I want it to be easy to connect to different API servers (staging, production, etc.)

3. As an administrator, …
    a. I want to be able to reject submissions, and review reported submissions

2. Design Outline

2.1. High Level Overview of the System

Our system consists of two mobile applications, for iOS and Android, and will use the Client-Server architecture. We believe that the Client-Server architecture will be most suited to our system because our design requires that multiple users will connect to a server to access and modify stored data. The server will connect and process requests from multiple clients and will access the database where all media and user profiles are stored. We would like to separate user design aspects from the back-end server in order to make them as independent of each other as possible.

1. Client
    a. The client will serve as the user's interaction with our system.
    b. The client will display a newsfeed, a map, user profile information, and saved user data.
    c. The client will present features for the user to create, rate, save, and view content.

2. Server
    a. The server will handle all traffic between the application and the database.
    b. The server will generate requested media files when the client requests them.
    c. The server retrieves and stores database information.

3. Database
    a. The database will store all of the media files and their details for this application.
    b. Typical data will be image and video files, comments, ratings, location data, bookmarks, content category, and user profile information.

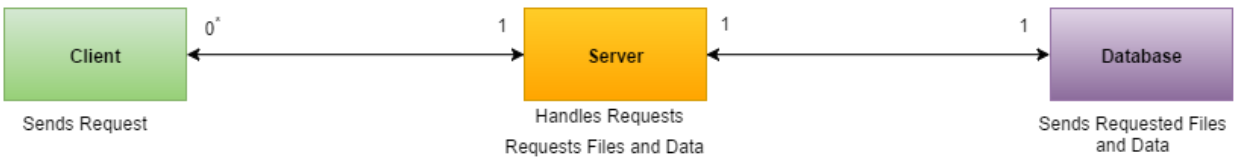

Figure 2.1

We will have two primary user types each with a different set of privileges. The user type will be able to access all the other functionality of the system, but will only be able to control other users behaviors through a rating system. The moderator user will have more privileges to control other users' behaviors.

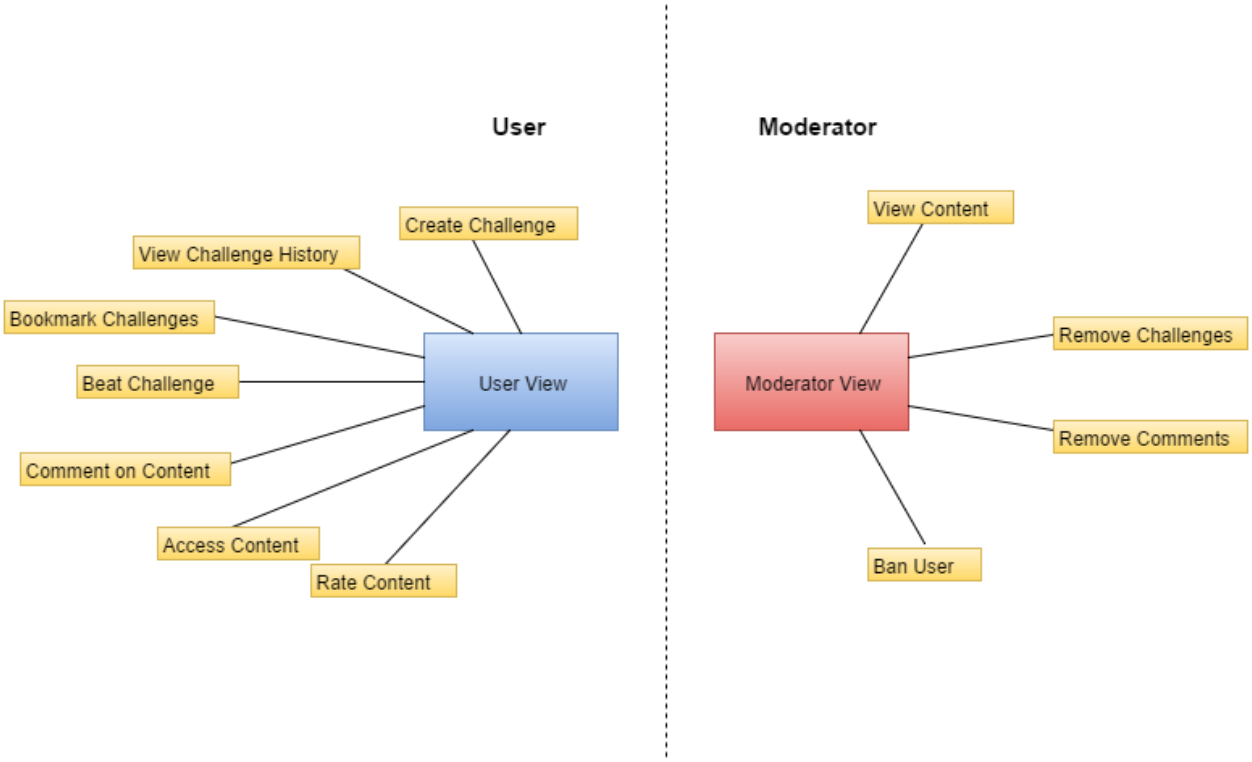

Figure 2.2

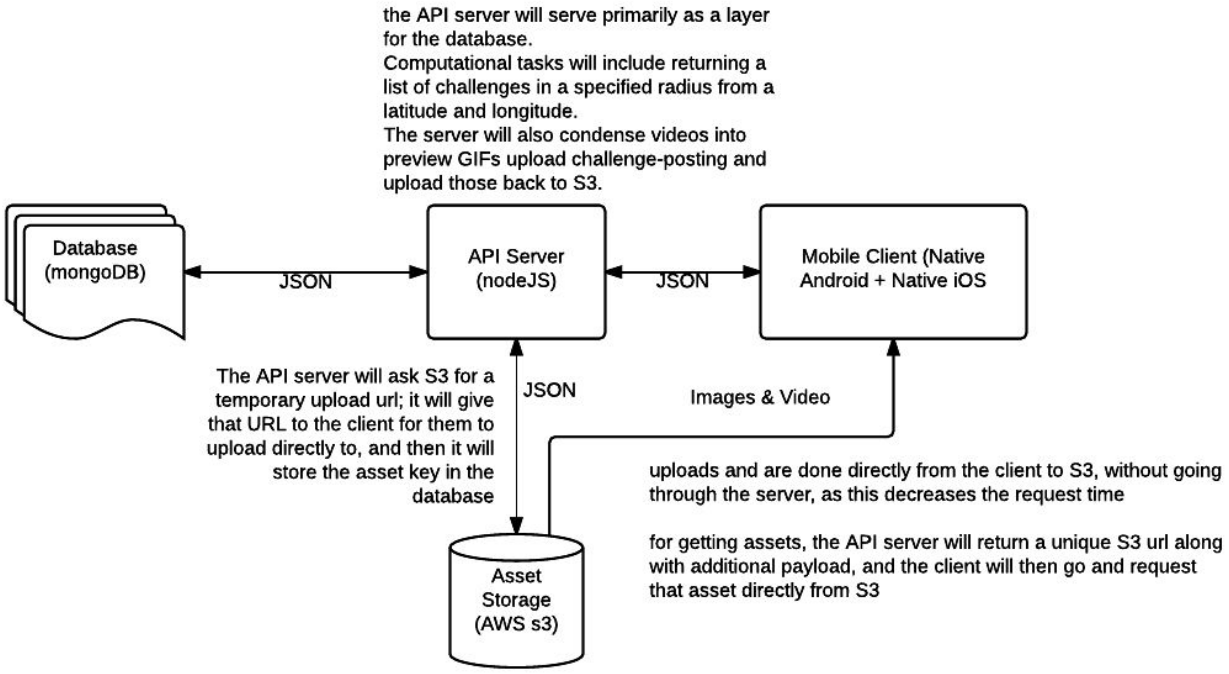

Figure 2.3

3. Design Issues

When planning our design, we ran into a lot of design issues that we had to talk through and figure out the best options for us, the developers, and the users.

3.1. Functional Issues

Issue 1: How should we display data in the newsfeed?

Since the newsfeed is the most visible and, debatably, the most important part of our app, we needed to decide how we wanted to display lots of content to the user without being overwhelming. See mock ups for this in design details.

- Option A: Half width squares, big focus on image
- Option B: Full width squares, big focus on image
- Option C: Full width rectangles, equal focus on image and information
- Decision: Option C:
    - Enough space for extra challenge information
    - Didn't want videos/images to be as prevalent as Option B

Issue 2: How should we handle user feedback on challenges?

In order to make sure users only view high quality content, we decided we needed some sort of user rating system so we can automatically remove fake/dangerous/boring challenges. However, we needed to decide how exactly we want to let users rate content.

- Option A: Use a "five star" system, somewhat like Yelp
- Option B: Use "upvotes" and "downvotes" like Yik Yak or Reddit
- Option C: Use "likes" like Facebook or Instagram
- Decision: Option B:
    - Didn't want it as complicated as Option A
    - Wanted some way to have negative feedback, unlike Option C

Issue 3: How should we handle offline challenge creation/viewing?

We would like our app to be usable in almost every situation, so an immediate problem we are going to face is what to do when a user isn't connected to the internet. Most social media sites won't let you do anything without internet, but we were wondering if we could do better.

- Option A: Don't allow the user to do anything without an internet connection
- Option B: Only allow the user to view cached content without internet
- Option C: Allow the user to view cached content and create content to be automatically uploaded when internet connection is acquired
- Decision: Option A
    - Caching is pretty tricky and isn't often done well
    - We can push caching off into an if time allows category

Issue 4: User/password passed and/or social login?

Creating an account and logging in is the first hurdle a user faces when they open up any social media app. We need to make sure the experience is as easy for users as possible.

- Option A: Have the user create an account, password and all
- Option B: Simply use Facebook/Google Plus to login
- Decision: Option B:
    - Simpler onboarding process for users
    - Don't have to worry about passwords

Issue 5: How do we serve graphic content without overloading user devices?

In our challenge news feeds, we hope to have eye-catching graphical content with each challenge in the form of video, gifs, or photos.

- Option A: Play a video in each section
- Option B: Play a gif in each section
- Option C: Display a static photo in each section
- Decision: Play a gif in each section
    - not as intensive as Option A
    - more engaging than Option C

3.2. Non-Functional Issues

Issue 1: React Native vs Native iOS and Android

One of the very first design issues to come up was the decision to write native apps in Java and Objective-C/Swift, or try to write both apps at the same time using Javascript with React Native.

- Option A: React Native
- Option B: Native iOS and Android
- Decision: Option B:
    - Much more familiar with Native apps
    - More progress possible in sprint one

Issue 2: How do we make sure uploading and downloading videos doesn't take too long?

When uploading and viewing videos, one of the big pain points for a user is waiting for the upload or download process to take place. We wanted to make sure this wouldn't take too long and interrupt the user's flow.

- Option A: Allow users to upload any length/quality video
- Option B: Set a max quality on a video
- Option C: Set a max length on a video
- Option D: Set a max file size on a video upload with minimum quality
- Option E: Provide tools for the user to trim/edit videos
- Decision: Option D and E
    - High quality videos are going to be important
    - Still need videos to convey all the information they can

Issue 3: What database to use?

Based on preliminary research, our two main contenders came down to MySQL vs mongodb.

- Option A: mongodb
    - open source
    - document based storage (structured json)
    - no foreign keys or relational data, everything is a tree
    - can be more efficient as you can nest things when you save them instead of having to join them
    - has built in geospatial processing!
- Option B: mysql
    - has proven its worth through the years
    - all team members have varying levels of knowledge on it

- Decision: Ultimately, we decided to go with mongodb since:
    - It interfaces well with node.js via *mongoose*
    - It has built in geospatial analysis (for finding challenges within a given radius of the user)
    - The database is interfaced via an ORM wrapper, so our team members can learn a new technology without wasting too much time, as it is somewhat abstracted via the wrapper.
    - Our data is not very complex, so we can structure the document tree easily

Issue 4: Where to run our servers?

For our app to operate and share content between different mobile devices, we will need to have servers that handle creating, uploading, and viewing content. We need to decide on how and where to host our server.

- Option A: Amazon Web Services (AWS)
- Option B: Digital Ocean
- Option C: Self-hosted Servers
- Decision: Ultimately, we decided to go with mongo self-hosted servers since:
    - we already have them
    - they are free
    - our server could be ported to a more powerful option later, AWS or Digital Ocean, if we find that we have too many users

4. Design Details

4.1. Class Design

4.1.1. Android

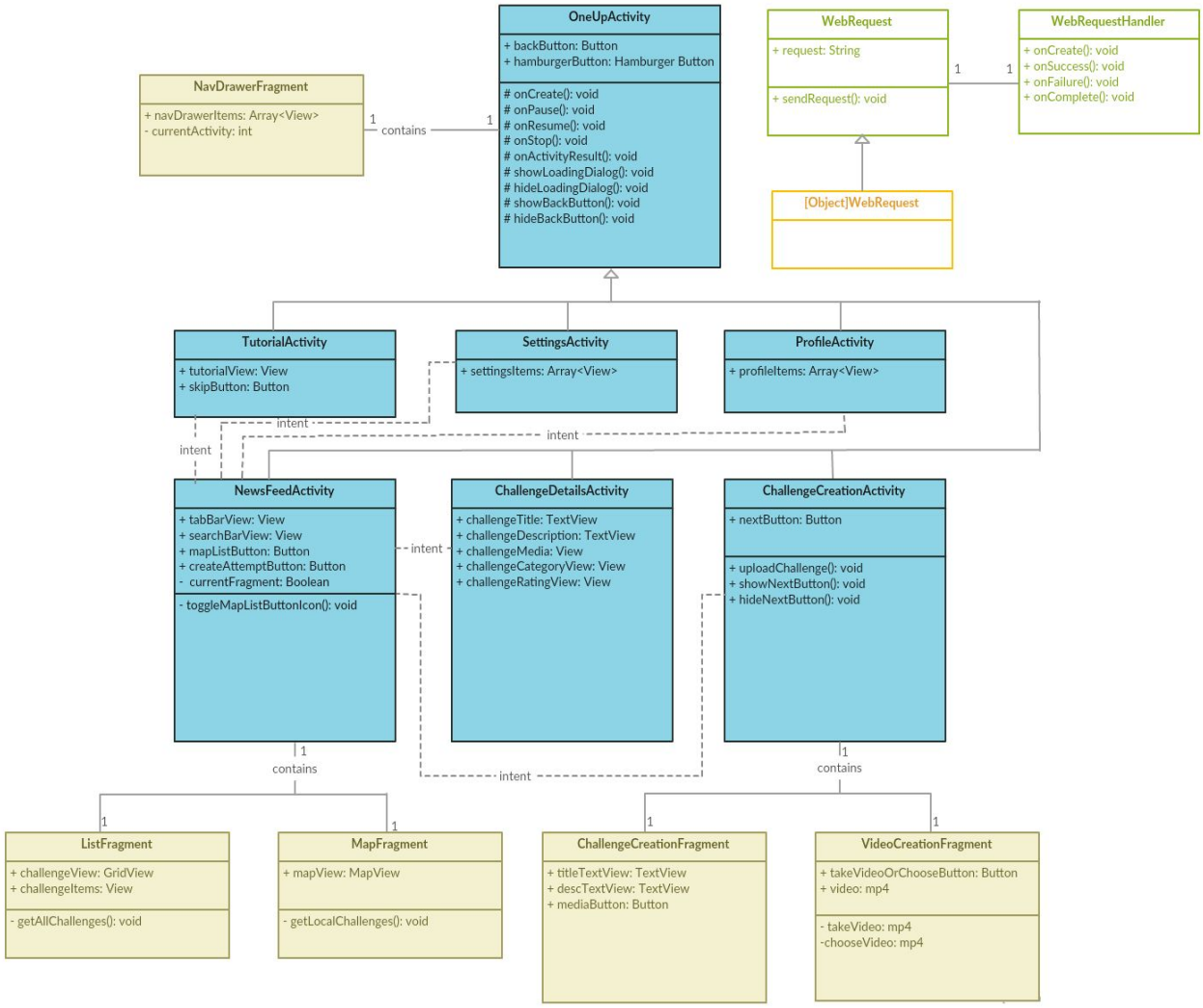

Figure 4.1

Descriptions

We have a bunch of activities and fragments to handle navigation throughout the various pages of the app. Everything we need is as follows:

- *OneUpActivity*: A subclass of Activity, this is a superclass to be used for all of our activities. We'll have things like the hamburger and basic navigation code in here to take care of the actions that every activity in our app needs to be able to do.
    - *TutorialActivity*: A subclass of OneUpActivity, this is where we will handle all the tutorial stuff associated with onboarding new users. We'll have a nice paged layout to showcase a bunch of information the user might need to know.
    - *NewsfeedActivity*: A subclass of OneUpActivity, this is where we will display all the challenges a user will see when they enter the app to browse. From here they will also be able to search.
        - *ListFragment*: A subclass of Fragment, this will be one of the fragments we use to display challenges to the user in the newsfeed
        - *MapFragment*: A subclass of Fragment, this will be one of the fragments we use to display challenges to the user in the newsfeed
    - *SettingsActivity*: A subclass of OneUpActivity, this is where we will let the user change settings. It should be accessible from the nav drawer and the profile page.
    - *ChallengeDetailsActivity:* A subclass of OneUpActivity, this is where a user can view all the details and history for a challenge.
    - *ChallengeCreationActivity*: A subclass of OneUpActivity, this is where a user can create a challenge to be put on other's newsfeeds.
        - *ChallengeCreationFragment*: A subclass of Fragment, this is the main fragment for the challenge creation process. It's where the user will enter all the basic information and get to the video upload fragment.
        - *VideoCreationFragment*: A subclass of Fragment, this is the fragment for creating and sending videos/images with a challenge.
    - *ProfileActivity*: A subclass of OneUpActivity, this is where a user can view their profile. From here they can see their challenge history, username, followed OneUper's, and more.

    - *NavDrawerFragment*: A subclass of fragment, this is the nav drawer that will appear in every activity when the hamburger button is pressed. It will allow navigation between the various activities.

We'll also need some sort of framework to handle web requests to our server. The required classes are as follows:

- *WebRequest*: This is the superclass for all web requests. It will handle making asynchronous calls to our api.
    - *[Object]WebRequest*: This is a subclass of WebRequest and encompasses multiple classes. We'll have one of these for each request we'll have to make.
    - *WebRequestHandler*: This is a private abstract class within WebRequest. It exists so that callers can tell when asynchronous requests come back.

4.1.2. iOS

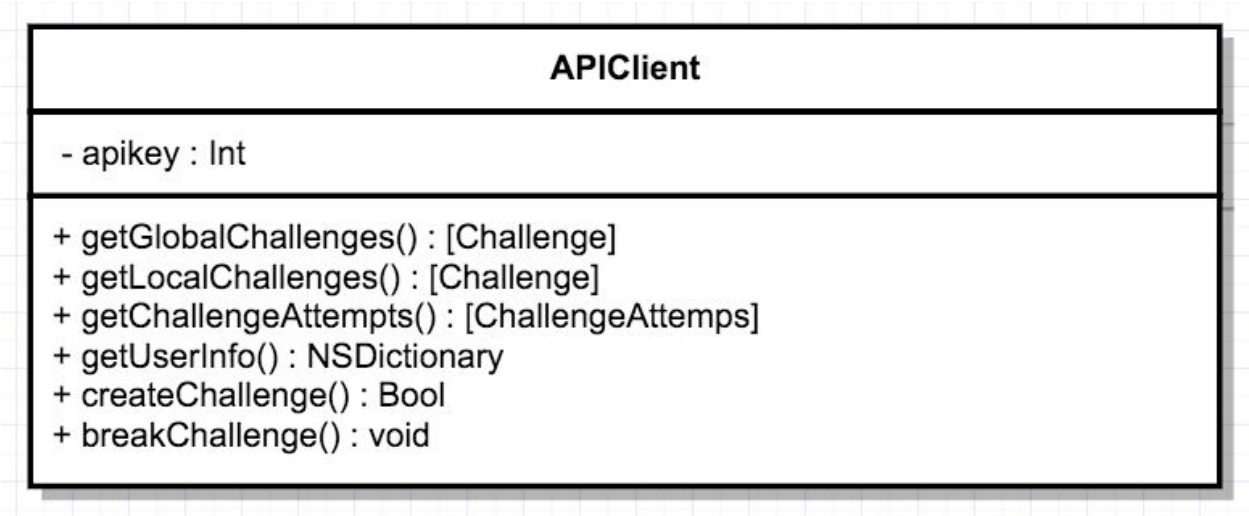

Figure 4.2

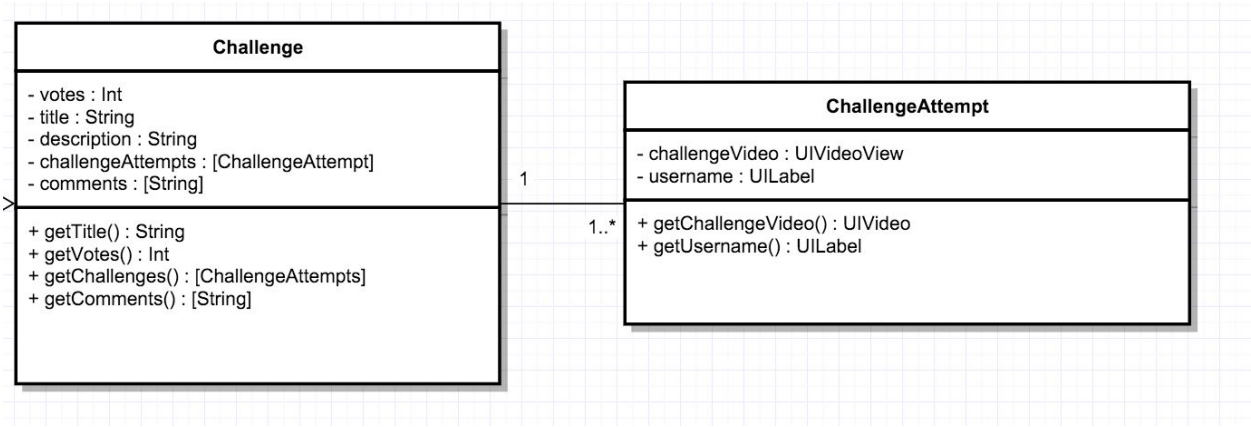

Figure 4.3

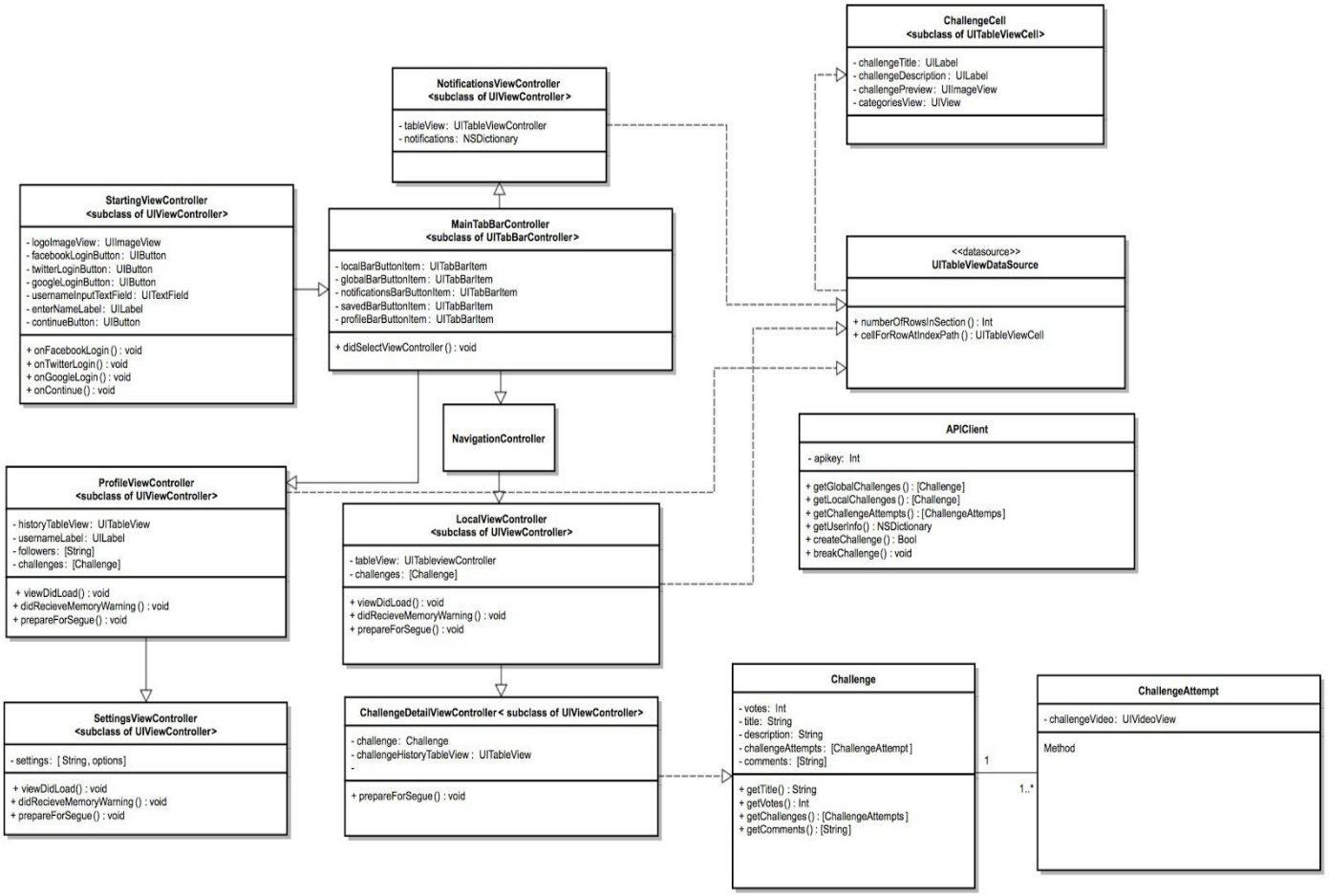

Figure 4.4

Descriptions

- ***StartingViewController*:** The controller for the “intro and login” view, which presents our logo, the options to login with Facebook or Google+, as well as allow users to set their username upon signup.

- ***MainTabBarController*:** The controller for the application’s tab bar, which users will use to quickly navigate between the “Local”, “Global”, “Notifications”, “Saved”, and “Profile” views in the app.

- ***NavigationController*:** The containing navigation controller for the challenges section of the app, which presents a header bar with titles, action buttons, and a back button(when applicable).

- ***LocalViewController*:** The controller for the “local challenges” view, which presents a table view of challenge cells containing selected challenges.

- ***ChallengeDetailViewController*:** The controller for the “challenge detail” view, which presents all information relevant to a specific challenge, including: information on record, current holder, challenge history, and comments.

- ***Challenge*:** A class that represents a “challenge”, and holds information including “votes”, “title”, “description”, “attempts”, and “comments”.

- ***ChallengeAttempt*:** A class that represents a “challenge attempt”, and holds information including “user”, “date”, and “video/media”.

- ***ProfileViewController*:** The controller for the “profile” view, which presents profile information including username, followers, and challenges the user has participated in, using a table view when applicable.

- ***SettingsViewController*:** The controller for the “settings” view, which presents fields for the user to manage their account and app.

- ***NotificationsViewController*:** The controller for the “notifications” view, which presents a table view of notifications sent to the user (for example, when a record is beat or someone leaves a comment on their challenge).

- ***UITableViewDataSource:*** A reusable class for presenting data from the API in a tabular manner, and is used for the “challenge” view, “profile” view, and “settings” view.

- ***ChallengeCell*:** A class used with a table view when presenting a list of challenges. The challenge cell formats and displays challenge data in a consistent, elegant fashion.

- ***APIClient*:** A class used for getting data from the API server, and contains public methods for performing common requests, such as getting “global challenges”, “local challenges”, “challenge attempts”, “user info”, creating “challenges”, and breaking “challenges”.

4.1.3. Backend

Database Schema

- *User*
    - facebook_id
    - nickname
    - settings[]
    - bookmarks[] = array of challenge ids

- *Location*
    - latitude
    - longitude
    - foursquare_id
    - parent
    - type

- *ChallengeGroup*
    - description
    - schema
    - categories[]
    - Challenges []
        - location_id
        - Attempts[]
            - user_id
            - score
            - assets[]
                - url
                - type
                - status
            - Comments[]
                - user_id
                - comment
            - Votes[]
                - user_id
                - vote

4.2. Sequence Diagrams

4.2.1. Sequence when a user opens the app for the first time

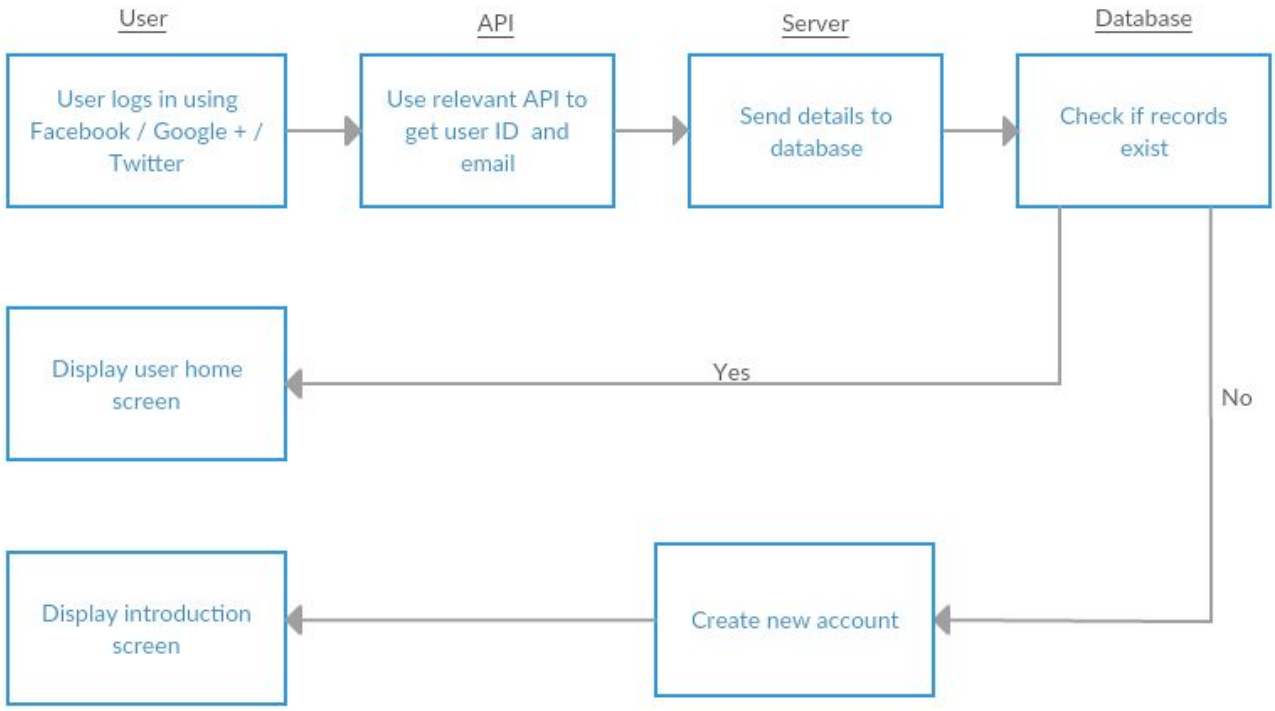

Figure 4.5

4.2.2. Sequence when a user opens the app any other time

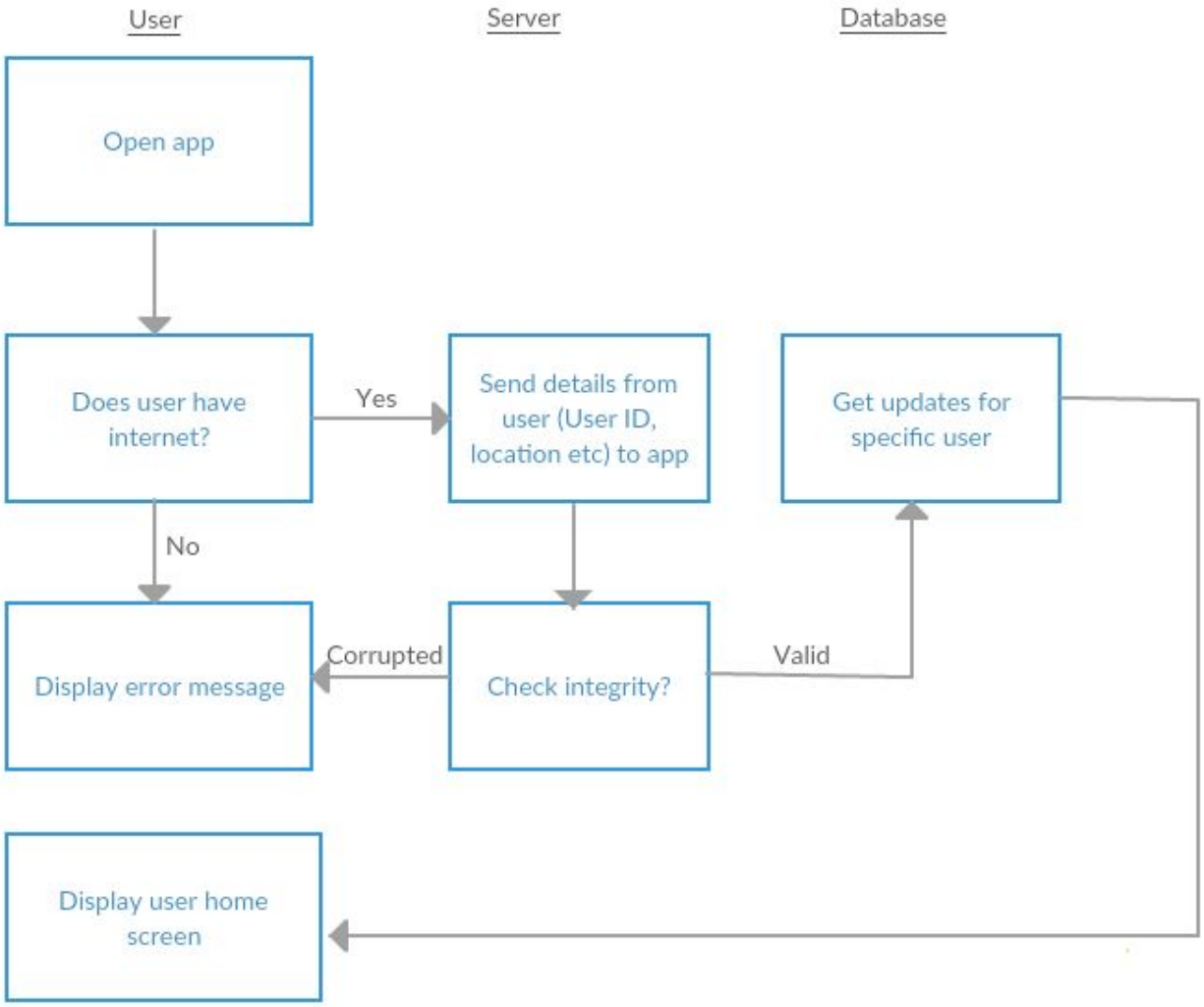

Figure 4.6

4.2.3. Sequence when a user creates a challenge

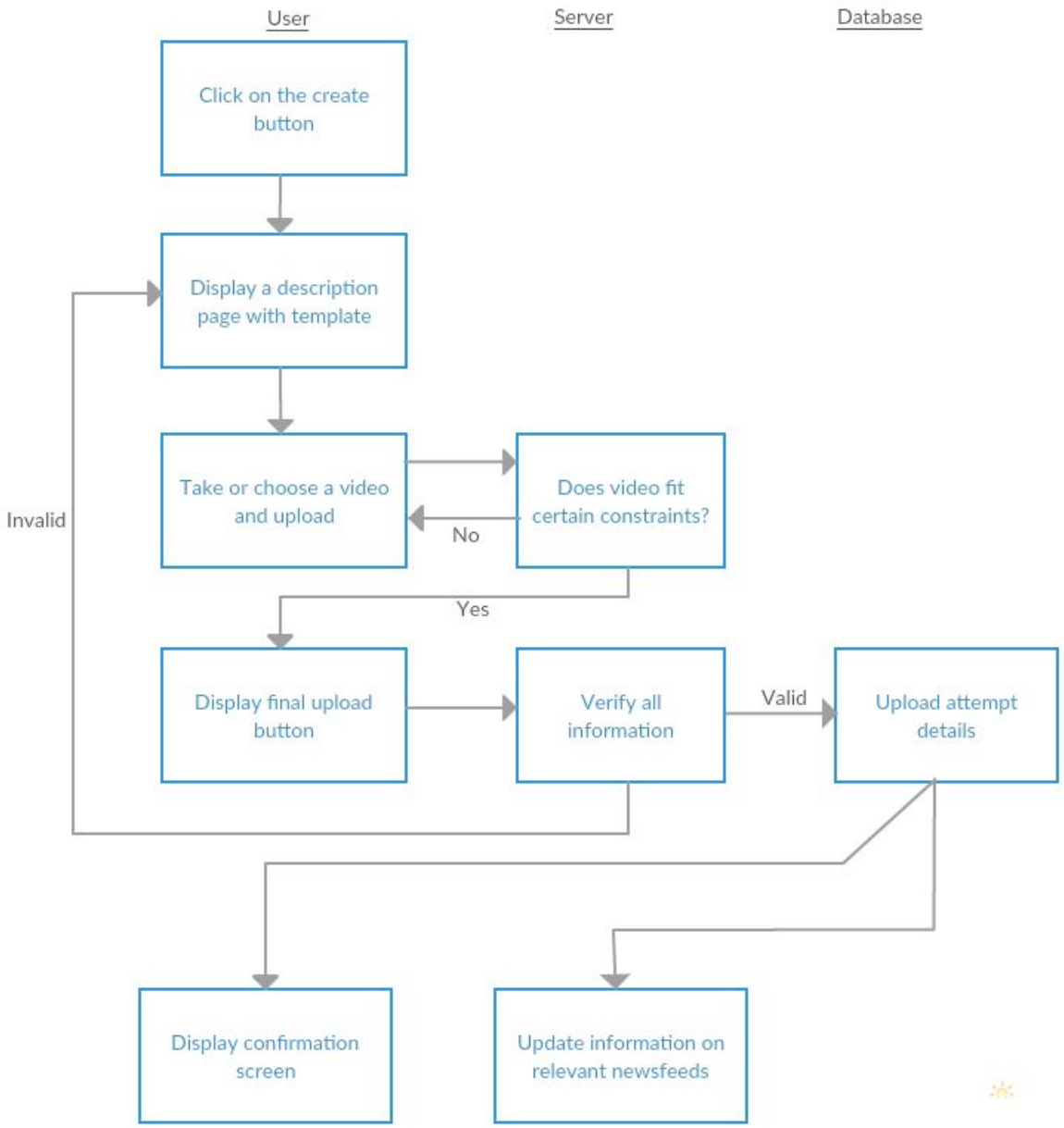

Figure 4.7

4.3. State Diagrams and Mockups

4.3.1. State diagram for entire application

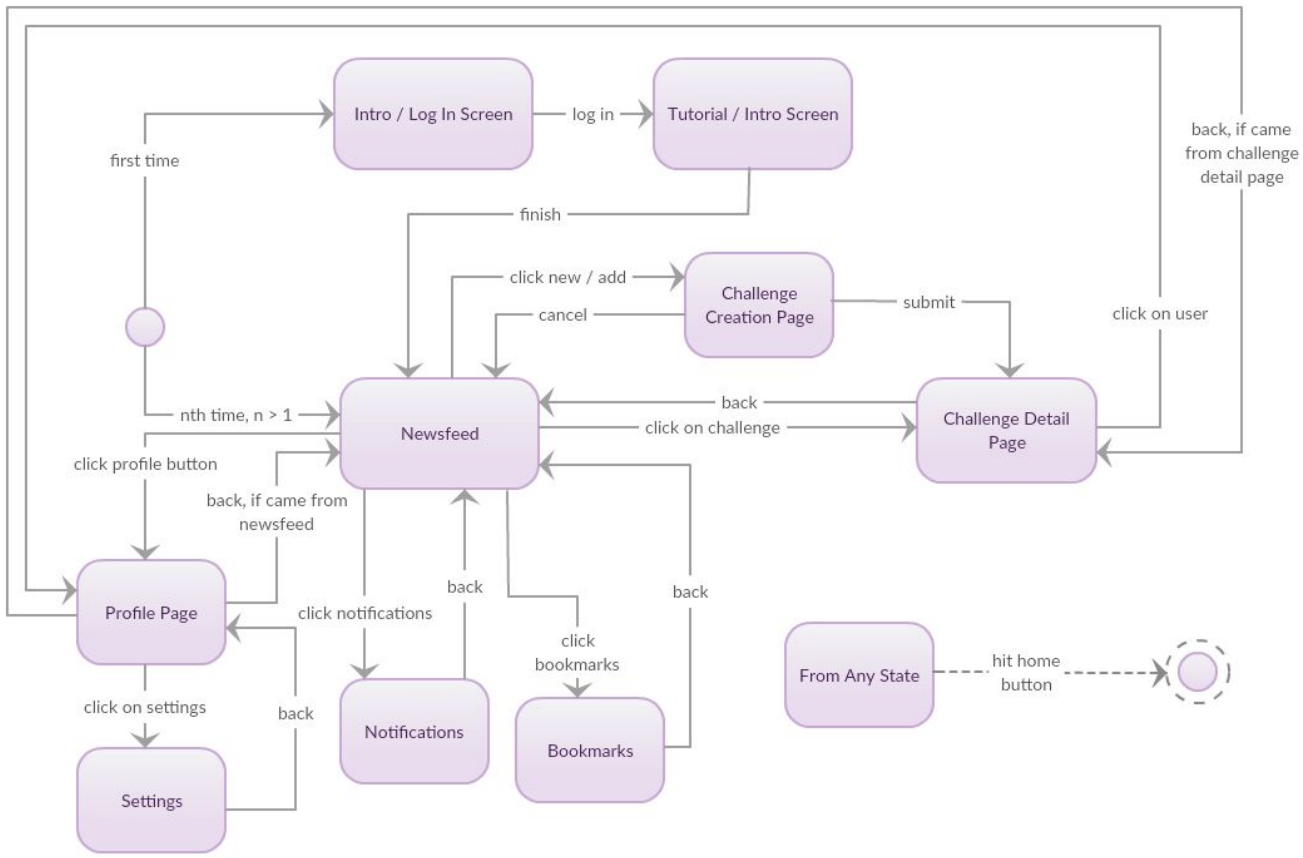

Figure 4.8

4.3.2. Mockups for Newsfeed items

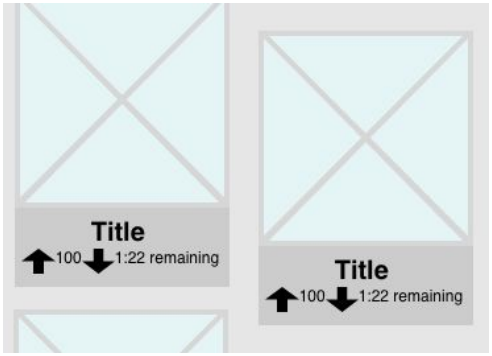

Figure 4.9

Figure 4.10

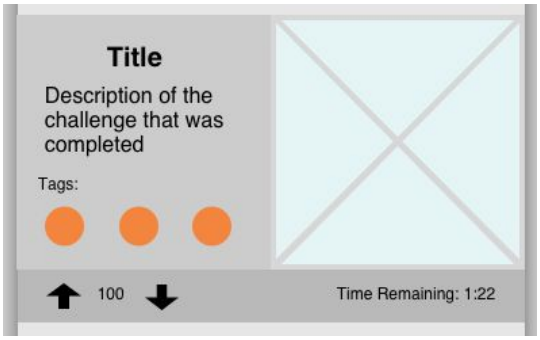

Figure 4.11