|  |
| --- |
| **Fans For Fans**  CSCC01 Final Project  Redacted  Redacted  Redacted  Redacted  Redacted  Redacted  October 22, 2019 |
|  |

# Table of Contents

[**CRC Cards**](#_g44bw6q72ulm) **3**

[Front-End CRC Cards](#_s5pxi2c1hbkd) 3

[Back-End CRC Cards](#_iznb2qing3s) 5

[**System Design**](#_8h3vial7pbux) **6**

[System Interaction with Environment](#_or2hzcxxajfc) 6

[Architecture of the System](#_6wnxp95s2fos) 6

[System Decomposition](#_mzuij8qkd5m8) 7

[Firebase Authentication](#_yqwlsexxtb7p) 7

[Firebase Storage](#_w5ylqf8m1w36) 7

[Firebase Realtime Database](#_k77ghovpdm5l) 7

[Mobile Device (React Native App)](#_80szky81niu0) 7

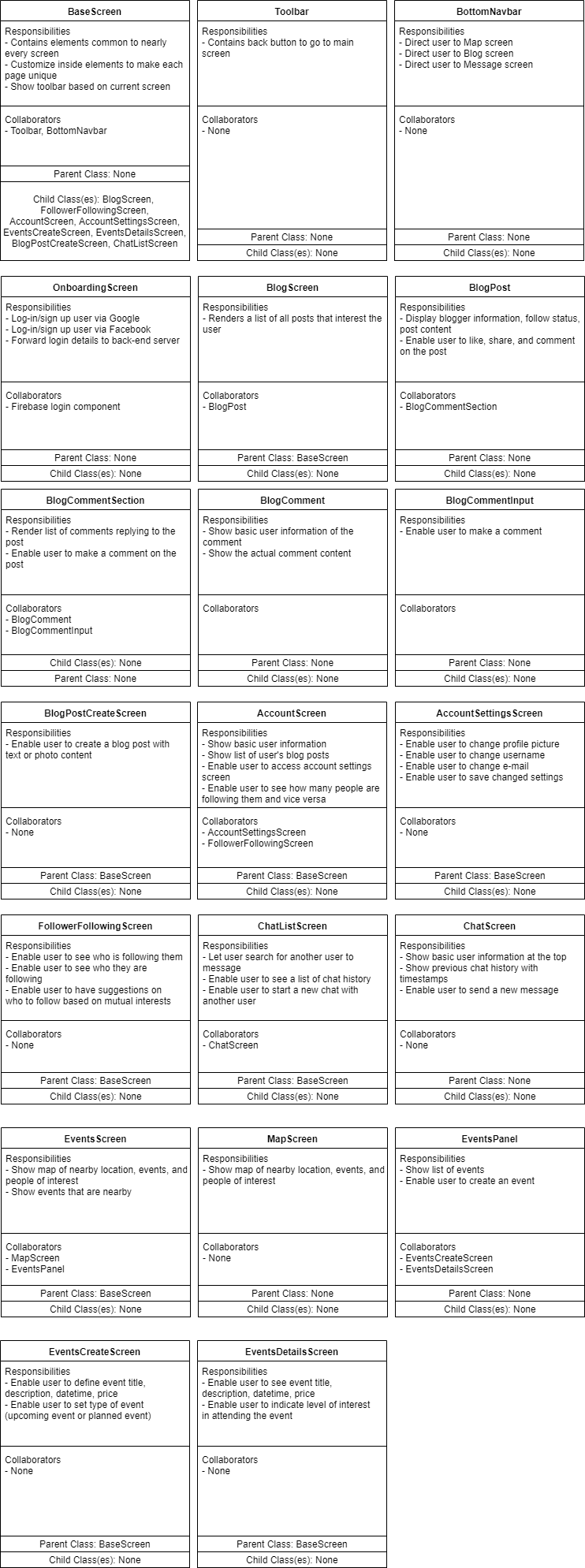
[Neo4j Storage](#_linr02r4e1ec) 8

# CRC Cards

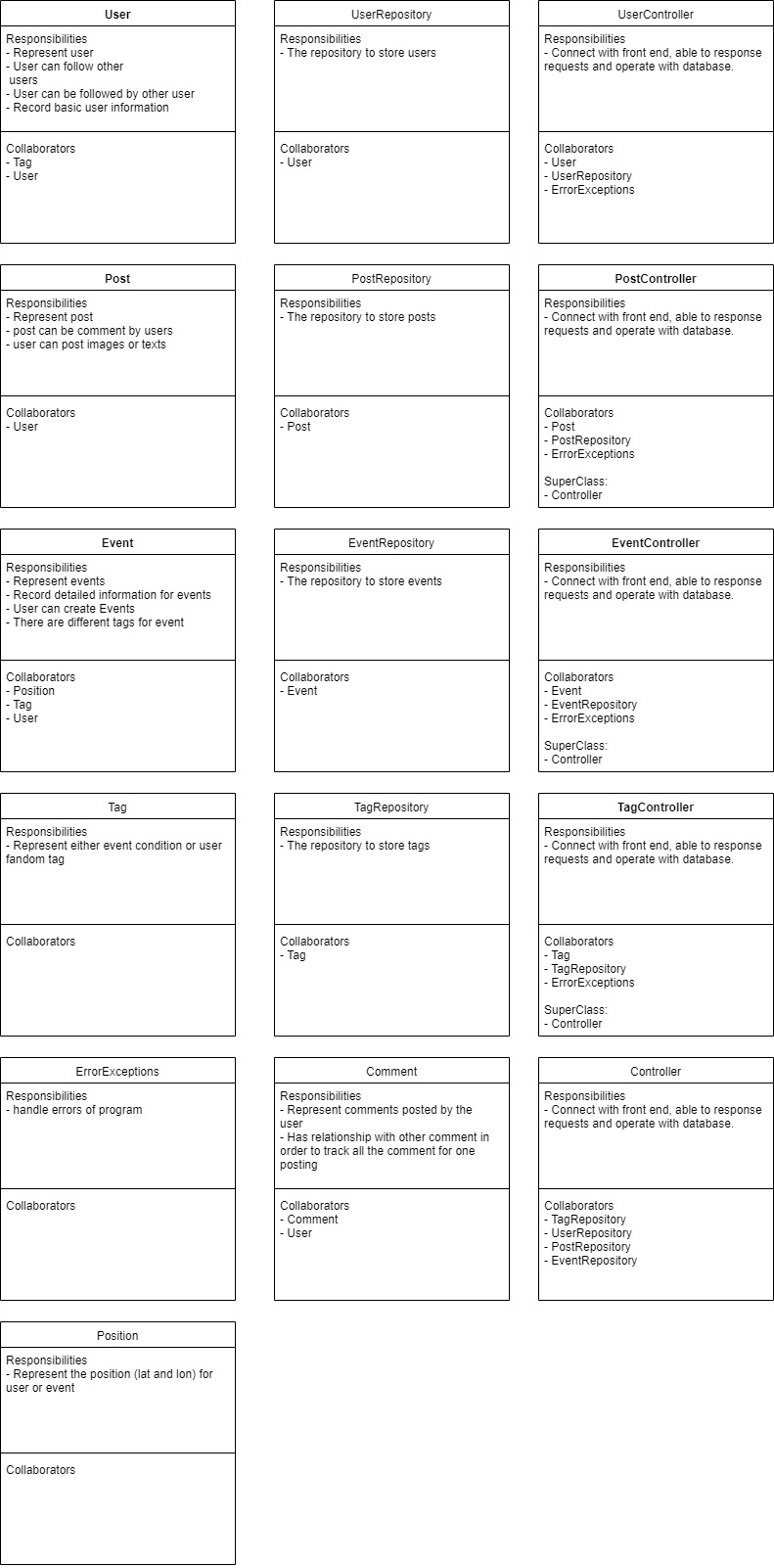
## Front-End CRC Cards

## 

## 



## Back-End CRC Cards



# 

# 

# 

# 

# 

# 

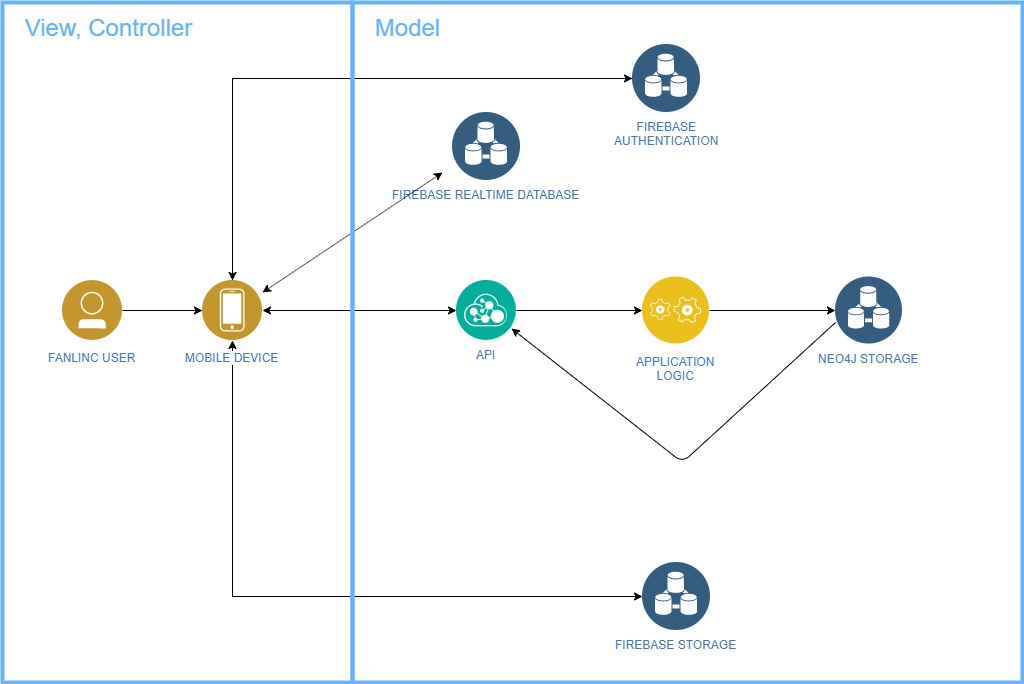
# 

# System Design

## System Interaction with Environment

* The system works both for IOS and Android devices (To start the app, clone the github repo and run `npm install` and then `expo start`, if run on Mac, just choose “Run on IOS simulator”, if run on Windows, first start an Android emulator and then choose “Run on Android device/emulator”).
* Users should have a Google account ready because the system needs to authenticate user account through Google.
* Users need to keep GPS tracker on since the system has a map to show the locations of nearby events.
* Users need to have Internet access whenever they are using the system since we provide real chats between users and they need the Internet to fetch all the posts and events information.
* Users should give the system access to their albums if they want to make some posts with images or choose their own user icons.

## Architecture of the System



Components:

* Failing user: The user that is interacting with the React native app (view, controller) to orchestrate actions and observe data provided by the backend (model)
* Mobile device: Hosts the React native app (view, controller)
* Firebase authentication: The authentication features of Firebase used to sign-up and sign-in users through email or Google. Stores registered user data.
* Firebase realtime database: A cloud database built for clients that need real-time notifications in response to changes to data (used for the chat feature)
* Firebase storage: A cloud database used to store and retrieve picture information relating to blog posts
* API: The network endpoints accessed via various request methods (PUT, GET, POST) to interact with the model
* Application logic: The backend code that handles the modification and reception of data from the backend database
* Neo4j storage: The backend database responsible for holding persistent data

## System Decomposition

### Firebase Authentication

Firebase Authentication is done using the Oauth2 protocol and will be used to authenticate users. Users will be able to sign in with their Google account or create a Google account if they do not have one. The authentication token will then be forwarded from Firebase to the mobile app which will then forward it to the API to be stored with a generated user profile. If any errors occur in login, the user will be prompted with a generic error and be asked to attempt to login again. Users should not be able to get past the login screen without being successfully authorized.

### Firebase Storage

Firebase Storage will be used to store images that users want to attach to their blog posts. When making a new blog post, users will be prompted to upload their photo to the Fans For Fans app which will send the photo and store it in Firebase Storage. A link to the photo will be returned from Firebase Storage to the app which will then be forwarded to the API when the blog post is made. This same link will be called each time a user wants to view that blog post.

### Firebase Realtime Database

Firebase Realtime database will be used to set up the environment for the chat section. This feature has a dependency on the ‘follower and followee’ feature. Once the backend (Spring boot) returns the ids of the users who want to chat. A Firebase Realtime database will be set up in order to support the chatting feature. After the chat session ends, the chat log will be sent to the backend and will be stored in the database.

### Mobile Device (React Native App)

The mobile device and React Native app will serve as both the view and controller, as well as handle a large majority of the errors that occur. The app will have the responsibility of taking in user input and showing information retrieved from the models in the MVC architecture as defined above. The app will act as the controller by orchestrating calls between the model (Firebase, API) and the view (React Native).

In the case of invalid user input, the user will be prompted to type invalid user input with some guidance on what input is expected. In the case of network or external system failure, the user will be prompted to try again later. These different error cases can be determined by error status codes returned by the controller and model in the API, Firebase, and other components that interact with the mobile device and React Native app.

### Neo4j Storage

Neo4j will be used to store user account information and will handle the relationship between users (follower and followee), as well as all the information of events. It will be part of the model in the MVC architecture and will enable persistent storage by storing any information that is not able to be stored in Firebase.