Arthur Silverstein, Hanqing Wang, Margaret Schmit

CS 184 Final Project Documentation: Face Scanner App

**Problem Statement**

As members of clubs at UCSB, we need a quick, easy way to keep track of attendance at club meetings. Members want to be able to check in easily, and club administrators want to check attendance easily.

We have tried several solutions to this problem, including having a sign-in sheet at each meeting, having a sign-in Google form at every meeting, and having one club administrator take attendance. All of these solutions have flaws and have been ineffective at taking attendance at our club meetings. Therefore, we decided to create an Android app that could solve this problem.

**Design Process**

*User Stories*

* As a club member, I want to be able to check into an event quickly and easily.
* As a club administrator, I want to easily check who is attending my events.
* As a club administrator for a large club, I want to check multiple people in at the same time, so I can have multiple entrances for my club event.

We used these user stories to determine what the important use-cases are for our app. These helped us decide what features we should focus on.

*Design Decisions*

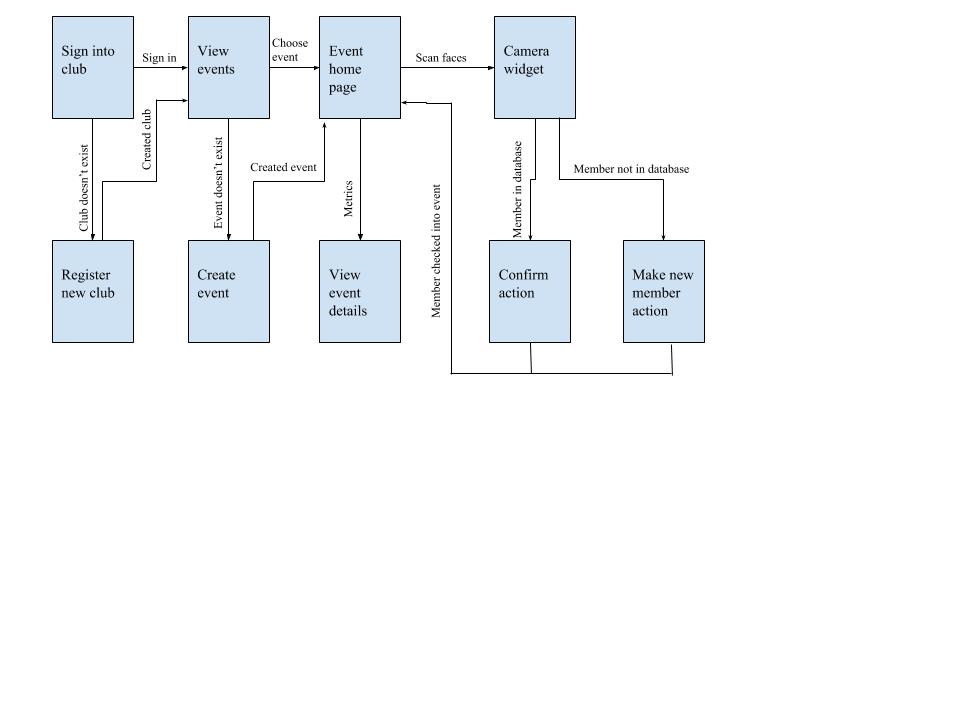
Our brainstorming focused around the above user stories.

For the first user story, we came up with two ways to easily hash members to their unique information: fingerprint identification and face scanning. Both of these messages are quick ways to identify people quickly, and both have existing APIs in Android that allow developers to implement these features. We decided to use facial recognition for two reasons: firstly, we thought that facial recognition would be quicker/more sanitary, as it does not require each user to touch the phone, and, secondly, we had security concerns about fingerprint recognition. Specifically, fingerprints are used for security, but most people have at least one photo of themselves online, so we thought people would be more willing to allow a club to keep their picture.

For the second user story, we decided to create a view in the application that would allow admins to see event attendance. We considered creating a web view for admins but, as this is an Android development class, we decided to create an Android view for club admins.

Lastly, for the third user story, we decided to create an external server to store club information. This allowed us to have multiple phones all signing people into an event at once, because th phones would not be storing the objects internally. The phones all communicate with one server, so they can all update that server and pull new information from the server in real time, thus updating every simultaneously.

*Application Flow*

******

The above figure shows the brainstormed flow for the application. Before we wrote any code, we designed this UI so we could have a framework for how the users would interact with the application.

**Resources Used**

In addition to the Android SDK, we used a number of third party services. We used PostgreSQL and Redis for data storage, NodeJS and ExpressJS to run the server, AWS S3 for image storage, and AWS Rekognition for facial recognition. We also used countless open-source NPM packages. Some notable ones include bcrypt (for hashing passwords), pg (for connecting the server to the SQL database), and the AWS Node SDK.

We host the server and PostgreSQL databases on Heroku. The Redis instance is hosted by Redis Labs.

In terms of documentation, we used the standard source of documentation for each software. For example, for Android we used <https://developer.android.com/reference/>. We also used stack overflow occasionally, example https://stackoverflow.com/questions/11766878/sending-files-using-post-with-httpurlconnection.

**Implementation**

Our app is a user-friendly client for our custom restful API, running on Express/Node. The server itself (in addition to boilerplate code), has its functionality encapsulated in endpoints (such as checkFace, getEvents, addMember, etc). We make heavy use of Javascript promises and async/await to make dealing with numerous network requests easy.

We chose to use Node and Express simply because they are easy, and we needed to allocate time carefully. We chose a server-centric design because it improves portability, easy development, and performance. Because business logic is on the server, we can add clients for other platforms without duplicating much code. The fact that the client and server are coupled only by API endpoints, we could work on them in parallel without stepping on each others’ toes. And because several pieces of functionality require multiple requests to other services in sequence, moving these requests to a server speeds up the user experience immensely -- datacenters have faster internet than a mobile phone.

The client uses the API to send and retrieve important information and display it to the user. Classes such as HTTPUrlConnection and Volley were used to make these API calls. Each part of the client is an activity, and each activity interacts with other activities minimally. We used Intents to connect the activities. We were able to easily connect the frontend, even the parts created by different people. In terms of the actual “Face Scanning,” the camera is used to take a photo of that person, which is then sent to the server.

**Difficulties Encountered**

*HTTP Requests:*

We had to send the JPEG pictures of club members to the server to run them through the facial recognition algorithm. Unfortunately, that meant we had to learn how to make HTTP requests from Android that could encapsulate a file. First, we tried to create a Volley request with the file, but couldn’t find a reliable way to send a file through Volley. Thus, we ended up writing the HTTP request ourselves. This caused a lot of errors while testing, and it took awhile to fully debug this and reliably transfer files to the server.

*Data Structuring:*

We needed a structure that was scalable, performant, and worked well with relational data. Since no single database solves this problem out of the box, we decided to combine other solutions. We ended up using a relational database for almost everything, except checking members in.

*Error Handling:*

When we encountered errors from the HTTP request, these errors would manifest as *FileNotFound* requests on the Android side, which did not give us information about the actual error that the server was encountering. As we worked on these project parts in parallel, the server logs were not always available to the people working on the Android side; this created difficulty as people attempted to debug without knowing what the bug was.

This was solved when every person was given access to the server logs.

**Evaluation Results**

We were pleasantly surprised by the stability and performance of our app, especially the facial recognition parts. Across over 20 trials, the app only failed to recognize a face once. That was when we intentionally tried to break it by taking a blurry picture of the side of someone’s head. Response times for the facial recognition were almost always below 5 seconds. Ultimately, between the app’s speed and the fact that scanning can be done from a distance (meaning people don’t need to crowd around a sign-in shee), we think that this app could actually be used in the real world.