BadgerConnect

Che John
Department of Electrical and Computer Engineering
UW-Madison
Madison, U.S.
cbfu@wisc.edu

Sahas Gelli
Department of Electrical and Computer Engineering
UW-Madison
Madison, U.S.
gelli@wisc.edu

Khyati Gupta
Department of Electrical and Computer Engineering
UW-Madison
Madison, U.S.
kgupta44@wisc.edu

Christopher Su
Department of Electrical and Computer Engineering
UW-Madison
Madison, U.S.
cjsu@wisc.edu

Divyangam Dutta
Department of Electrical and Computer Engineering
UW-Madison
Madison, U.S.
ddutta5@wisc.edu

I. INTRODUCTION

Sarah, a nervous freshman at the University of Wisconsin-Madison, feels isolated on her first day of college. Determined to find her place, she tirelessly searches online for a solution to her freshman loneliness. That is when she discovers BadgerConnect, a tailor-made mobile app for UW-Madison students. Through BadgerConnect, Sarah connects with like-minded peers, finds a mentor who guides her through her major's complexities, and effortlessly organizes study groups and study buddies. The app even helps her discover classmates studying in the same on-campus library using location services, providing invaluable advice, and transforming her college experience. Through a few taps on her phone, she can bridge the gap and become empowered with friendships and academic success.

II. RELATED WORK & BACKGROUND

In the realm of remote learning, students have encountered a significant challenge in connecting with their peers. Existing applications like Demic, which aim to bridge this gap, have fallen short in several key areas. Demic's basic 1:1 messaging functionality proves inadequate for effective collaboration and resource sharing among students. Furthermore, the lack of location-based matching further inhibits students from connecting with fellow learners in their vicinity, particularly freshmen seeking to establish connections in a new environment. Moreover, Demic's potential UI limitations can impede user experience, making it challenging for students to locate features and navigate the application efficiently. These shortcomings highlight the pressing need for a more comprehensive and student-centric solution like BadgerConnect, which aims to overcome these limitations and provide an enriching and effective platform for student connections in the remote learning era. Recognizing these gaps, BadgerConnect steps forward, driven by the mission to empower students, enhance collaboration, and provide a holistic learning experience. With its help, students can break through the barriers of freshman loneliness, create meaningful connections that will last throughout their college journey, and, most importantly, thrive academically and socially.

III. HIGH LEVEL OVERVIEW

A. Diagram of Main Components

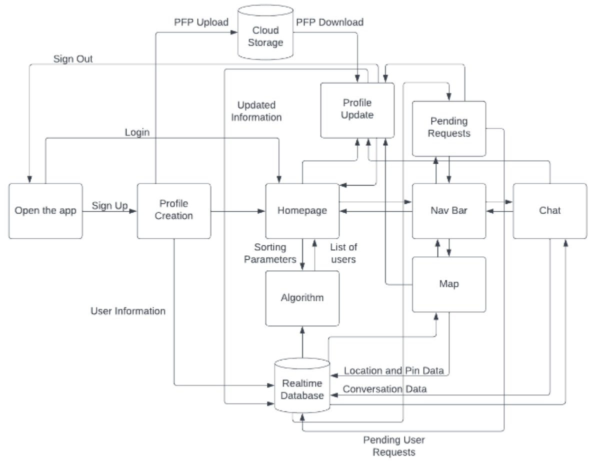

Figure 1. Diagram highlighting the main processes and activities that take place in the system

B. System Description

The diagram in Fig. 1 highlights the core components of our system architecture and some critical connections and operations. The rectangles indicate activities, fragments, and functions while the cylinders are database components. The arrows indicate changes in the ongoing process with some labeled according to what must be done to make the transition. While the design may initially look complex, the app flow is simple with ample connections between the pages to allow the user to go to the intended feature with minimal steps.

While the inner workings of the processes will be described in Section V, Detailed Design, this will serve as a run through of our applications lifecycle. The user begins by opening the app from where they can either create a new account or login. If they chose to login then they will be taken to the homepage, otherwise, if they chose to sign-up then they will go through the profile creation process before reaching the homepage. Once at the homepage the user can view recommended users which they can click on to view more information or connect with. They can also refresh or change filter options to get a new list of recommended users from the algorithm. From the homepage the user can navigate to other fragments such as the chat fragment, map fragment and the pending request fragment. The chat fragment displays the most current list of connections and the respective chats, the map fragment that displays a map of your area and allows you to make or find events by dropping a marker and the pending request which is a list of users that have sent requests which can either be accepted or denied. From any of these fragments the user can press on the top right profile icon to view their profile and update it. The user can either go back to the previous page or choose to log out which will take them back to the login/sign-up page which ends the app's lifecycle as they start from the beginning.

Our architecture was made to be modular and flexible to allow for new features to be implemented effortlessly through the use of fragments for the main components of our app. Using a wrapper class and a nav bar, we can extend our system to have more feature fragments. We used this architecture to allow each feature to be changed and improved separately while not affecting other components but remaining interconnected through the database functions. At the heart of our system is the Firebase Realtime database which stores all our essential information, from user data, conversations, map markers, and connection requests. All our components heavily use the database, and so we have developed reliable methods to communicate with the database to allow information to be shared throughout the fragments. As our schema for the data base is meticulously organized, the data is very accessible which enables fast CRUD operations while ensuring that our system is scalable and maintains integrity.

IV. Design Constraints

While modern android phones have a variety of features and impressive hardware there were some cases where we had obstacles that could not be solved with just the inbuilt hardware. Some of the constraints we faced are the following:

1. Limited to non-relational databases
2. Insufficient processing power of android phones and slow database communication
3. Deprecated android functionality from Android 12 to 13

As we were already planning to integrate other Firebase functionalities into our app, we surmised that it would be ideal if we could also maintain and communicate with a database built within Firebase. As Firebase and Android are both made by Google, connecting and interfacing with a Firebase Realtime database was smooth. However, due to this choice we were limited to working with a non-relational database design despite our app and experience being more suited towards relational databases.

Another important constraint that we faced was the processing power of the android phone. While the current android phones are quite powerful in terms of processing power, our algorithm had to find and recommend users efficiently and allow users to set criteria by which to find users. This meant that if we wanted to go through all the users and process them to find which ones to recommend, the app would be far too slow. Along with this we also found that sometimes the database was slow, unresponsive, and so if the app were to wait for the necessary data before continuing it would take a very long time.

Finally, another constraint we had to deal with when implementing the app was the supported android versions. Due to the changes between Android 12 and 13 we found that some functionalities were deprecated and as most of our team had different versions of Android it became difficult to identify and debug issues. We found that the phones we were given had a limit on their Android version as they were no longer supported and so would not receive updates. This meant that despite having Android 12, we needed to make an app that could be supported by the latest version of Android.

How We Addressed the constraints

The solutions to the design constraints we came across include:

1. Despite being limited to non-relational databases, we were able to replicate the reliability of sequel queries through complex nested queries and the use of listeners to get specific information. We also used good organization techniques to ensure that all the information could be easily accessed.
2. As we had insufficient processing power to handle the bulk of sorting users within the phone, we made use of a series of queries that would help us narrow down user recommendations without using the phone's resources. As we were using complex time-taking queries we also employed the use of asynchronous threads that could handle database calls in parallel with normal operations. With the help of CompletableFuture objects we were able to close threads and populate data within the app after the query was completed.
3. We ran into issues when accessing data within the phones for setting profile pictures. Due to differences between Android 12 and 13 there were changes in obtaining permissions for external data. To solve this issue, we decided to develop our app for the latest version of the android as most users would prefer to run the latest version of android rather than older versions. We also let users keep empty placeholder images if they are on Android 12 and below so they can continue to use the app.

Ethical/Social Issues

Ethical and social issues are a serious topic when working with social media apps that store and use user data. Our app allows users to build a profile with the intention that all the information will be viewable by other users and so we do not create a profile with any private or sensitive information. The only sensitive piece of information we store is the email of the user which is not revealed to other users whatsoever. As we have decided to use the Firebase Realtime database, we have the added benefit of Google's encryption system to ensure that the security and integrity of user information is maintained.

As an added measure of security within the database we also store user information by their security UID rather than their name or email. This will make sure that no user can be found through a normal search as they require the other user's security UID which is not easily available. Communicating with the database is also abstracted and so while the recommended userID's are cached, they will not

be able to access the methods required to gain more information about the users.
Finally, we have conformed to COPPA, the Children's Online Privacy Protection Rule, which means we check for users' age during the sign-up process to ensure they are over 12.

V. Detailed Design

The development of our app has remained consistent with the four key features we proposed in the phase 1 report (the connections algorithm, a messaging system, interactive GUI, and a Map-based collaboration system). This section will be dedicated to describing our design for these features, highlighting the key elements and methods we used.

A) The Data on Firebase

The data currently stored on Firebase can be categorized into 3 main sections; the Realtime Database, the cloud storage, and the authentication database.

The Realtime Database: This storage container on the Firebase platform houses our main JSON data structure which we call "Data". "Data" comprises of a few sub-data structures. "Conversations", being the first, holds all currently existing message threads within the app. Each conversation marked by a unique ID contains timestamped messages alongside the sender's name and a list of participants in the conversation. The list of participants is essential in the creation of the Users fragments as the UserAdapter class uses it to parse and display existing conversations on the Users Fragment.
Pending_Connection_Request JSON structure holds sent connection requests awaiting a response from the recipient. Connection requests are generated once the current user clicks connect on another user's profile tile on the homepage fragment. Each pending request is tracked with the recipient's user id as key and the sender's user id as values. For each current user, the pending request fragment parses through the JSON structure, reads the key that matches the current user id and uses the list of sender ids to pull user data on the corresponding users. Once a request is accepted, the sender user id is removed from the Pending_Connection_Request list and a new conversation is created with both the recipient and sender as participants.
The "UserData" JSON structure holds all the profile related information. Each instance has the user's id as key, contains the username, a bio, connection type, Date of Birth, Email address, Major, Meeting preference, StudyBuddy course and academic standing. The links to the profile pictures of the users stored in the cloud are stored in the adjacent structure "Users". "Maps" stores the user location and any event created by the user. Lastly, "markers" holds key coordinates for events already set up on the maps feature (user_longitude and user_latitude), the meeting description, title, and user id of whoever set up the meeting. This data is used in populating the Maps fragment.

Cloud Storage: The app uses the cloud storage space on Firebase to hold the profile images for all users on the app. Each image folder bearing the user id as name contains the profile image and an access token which is used by the app functions to access, parse, and display the user profile pictures.

Authentication: This storage holds the credentials of all signed up users (email, password) as well as an autogenerated user.

B) Interactive UI

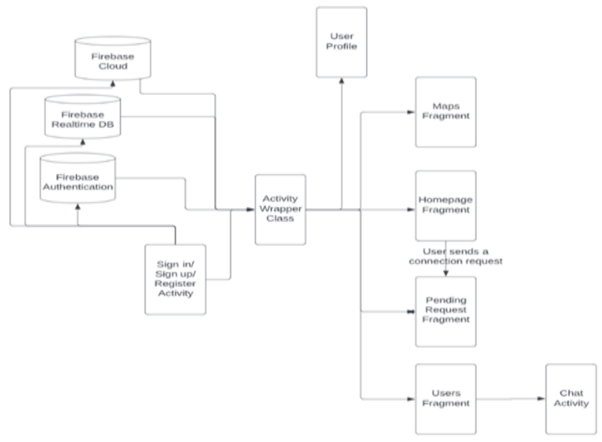

Figure 2. Schematic of BadgerConnect GUI

Figure 2 shows a simplified version of how the front-end components of our app are connected. Once the user signs up using the signup/register activity, they are taken to the ActivityWrapper class. The user is automatically logged in with the help of the auto login feature each time they open the app after signing up. ActivityWrapperClass houses the 4 fragments of the app. As can be seen in the flow chart, the only interdependency between fragments exists between the homepage fragment and the pending request fragment. Edits in the user profile also make changes to the data stored on the real-time database.

C) The Authentication, Signup feature and Connection Algorithm

The Authentication feature leverages the privileges provided by Firebase Auth to validate, authorize, and manage the login credentials of the users of the app. The Signup feature, in line with the authentication feature, leverages Firebase's inbuilt signup configuration which supports anonymous, email/password and google sign-in methods. The connections algorithm finds potential matches for the current user based on provided information during the sign-up process. The primary factors used to determine potential matches are based on the user's declared major, academic standing, their choice of either being a mentor or mentee and whether they intend to participate as a study buddy user on the app. The secondary metric for establishing connections is based on what we call the course similarity index which is used for study buddy searches to indicate the number of similar courses the current user shares with potential connections. The algorithm interfaces with the database using complex queries and returns a list of users that match the criteria. The list of recommended users can then be clicked on to reveal more information about them. By pressing on the connect button a request will be sent to the respective user. There is also the caching of User ids to facilitate showcasing connections without necessarily reparsing all potential connections from the firebase Realtime database which decreases long wait times.

D) Messaging communication system

The User fragment parses the list of active conversations containing the current user by iterating through the "Conversations" JSON structure on the Firebase real-time database and returns the list of conversations containing the current user. The useradapter class filters and displays the relevant user in a recyclerview layout. Once a connection is established, a new conversation instance is created between the two users. If the connection is broken, the delete conversation method is designed to remove the existing conversation.

The user fragment navigates the user to the chat activity once a specified conversation in the recyclerview layout is chosen. The existing messages between both users are formatted and displayed based on their timestamps. Both users can continue the conversation within the activity. New messages are added to the real time database via the sendMessage method in the MessageActivity.

E) Map-based collaboration system

The map feature is handled and implemented within the maps fragment. This fragment navigates to the user's current location once it gains focus. The user's current location is tracked with the help of a red location marker. The map feature continuously requests location updates from the location manager class after a certain defined time or distance interval. To set up a meeting, users can long-click on the map and enter event details, which will be displayed in a yellow hue marker. Users can also remove the markers they have created if they are no longer needed. Other users can see these yellow hue markers and see the details of the event by clicking on the marker which brings up an info window.

VI. Implementation

During implementation, a major decision we had to make was how we were going to represent users. We deliberated on whether to use a localized storage system or a centralized system. We decided to choose Firebase, a centralized system, due to the inbuilt features and extensive documentation. Next, we had to decide how we were going to represent the data in the database of our choice. Initially we opted to work with a single user definition that contained profile information. When implementing the Private & Secure Messaging functionality, we realized that creating two definitions for a user would keep the data being transmitted pertinent to what was requesting/updating it. In the case of messaging, user IDs, names and chat IDs were the only data required to construct and display the real time messaging system. However, for displaying users to each other as potential connections, more information is required so we used our initial User definition.

Another aspect that we did not initially account for was how to ensure that users did not show up to each other under distinct categories when already connected. Specifically, users who opted to seek out Mentor-Mentee relationships while also seeking out StudyBuddies should be displayed to each other until they connect. Upon connection, we noticed that they would still appear in each other's potential connection feeds despite already connecting. Luckily, we had code written to handle a "blocklist" in case users wanted to remove someone from their feed. Given they were already connected, we repurposed this latent functionality to remove the duplicate appearance.

A major challenge we hit was reworking our database halfway through which caused redundant code and work. This was also evident when we chose to create our application homepage after we had finished our core features. Features that were implemented as Android Activities needed to be converted into Fragments which created overhead. As a result, time that could have been spent implementing design goals became time spent on porting existing components.

VII. Evaluation

A. Testing Approaches

To ensure the functionality of our application, the core features: Private Messaging, Location Based Meeting Scheduling, and Connections were the focus of testing. This involved validating user input and storing the data in the Firebase RealTime database. Since the main function of the app is to display a matching userbase to the user. The backend functions that populated user data was the first testing that was performed. After validating the CRUD operations, testing data could then be created without the need to run through the entire authentication and sign-up process. Following this each component could then be tested in-hand using the test data created and a running instance of the application. The goal at this stage was to identify corner cases and unexpected behavior while performing simple tasks in the app. Additionally, any inconsistencies in data formatting were resolved at this stage. When individual components were fully tested in a standalone environment, integration testing was then performed to ensure continuity of existing components.

Table 1User Study

Person	Year	OS	Task 1 Time	Task 2 Time	Task 3 Time
1	2	iOS	4	2	1
2	4	Both	3	3	2
3	4	iOS	3	2	1
4	1	iOS	3	3	2
5	3	Both	4	2	2
6	1	iOS	5	4	1

Table 2 User Study

Person	Avg Odd	Avg Even	SUS Score
1	4	1.2	85
2	3.8	2	72.5
3	4	1.8	77.5
4	4.4	1.6	85
5	4.4	1.4	87.5
6	2.8	2.4	55

B. *User Study*

Our user study consisted of 6 individuals with a distribution across our target audience of college students. Our general questionnaire will be attached in the final submission. Across our participants we found that on average, students are comfortable approaching 1-3 classmates for course assistance. Additionally, in their freshman year, students felt comfortable approaching 0-1 upperclassmen for academic guidance for their field of study.

While only a few of our participants had experience with Android, we found that it did not have a considerable influence on time to completion for the assigned tasks. For the 1st task, the participant was asked to create a profile as a studybuddy, entering a specific set of courses. This was to place the focus on the application flow for sign-up and profile creation. Among all participants, we found that they were all successful with an expected amount of time taken. However, we did notice that when asked about the step with the most friction for the assigned task, course entry needed to be improved. Users did not like the fact that input needed to be from the autocomplete, but that they were still allowed to try to enter custom text. Task 2 was to then edit the profile, seek out a new connection type, and message the new connection. This task was designed to expose a new connection type on the homepage, test profile editing and private messaging. Like task 1, all participants were able to successfully complete the task. With this task, we found that the major pain points were the lack of clearer signposting on how to view the new connection type as well as where profile editing was located. Task 3 was focused on gathering feedback on our location-based collaboration feature. As such, we asked users to create a meeting point with a desired time and place of study. On completion, we then asked them to delete the said meeting point. Positive user feedback was positive across all participants with complaints solely being UI focused. 2 of our participants directly complained that while the workflow was intuitive, the placed marker was the same color as everyone else's. Because of this, they were not sure which was theirs.

In the second table of our user study, we collected the results of our post-study questionnaire. We adopted the System Usability Scale and calculated our participant scores, respectively. A higher average score on odd questions indicated satisfaction while even numbered questions were to identify participant dissatisfaction.

VIII. Lessons & Insights

A. *Scrum & Version Control*

During this project we used JIRA and Bitbucket respectively for Scrum & VCS. We created our JIRA roadmap initially outlining our core features (Connectivity, Private & Secure Messaging, Map Feature, and GUI) into epics. Each general epic was broken down into a more granular level with Stories. Child-issues that had smaller targets that added up to the full feature were then added to the Stories. We chose to use Bitbucket so we could create branches linked to the overall Epics we had scheduled. Initially, we created branches for each respective epic. As the codebase for core features grew, we shifted into creating branches for the accommodating child issue. Once a core feature became functional, we merged our production branch(main) into the feature's branch to run integration testing before merging the new feature into production.

A screenshot of our repository activity is shown in Figure 3.

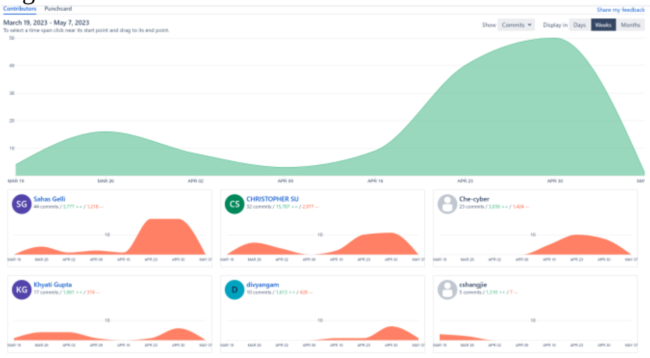
Contributors
Punchcard
Share my feedback
March 19, 2023 - May 7, 2023
To select a time span click near its start point and drag to its end point.
Show
Commits
Display in
Days
Weeks
Months
50
40
30
20
10
MAR 19
MAR 26
APR 02
APR 09
APR 16
APR 23
APR 30
MAY
Sahas Gelli
44 commits / 3,777 ++ / 1,218 --
CHRISTOPHER SU
32 commits / 15,787 ++ / 2,977 --
Che-cyber
23 commits / 3,036 ++ / 1,424 --
Khyati Gupta
17 commits / 1,061 ++ / 374 --
divyangam
10 commits / 1,613 ++ / 428 --
cshangjie
5 commits / 1,310 ++ / 7 --

Figure:3 Screenshot of commit history

B. *Individual Contributions*

1) Che John (CJ): My role as team lead invloved coordinating the project, checking in with team members on their progress, setting up meetings and making sure we were meeting required deadlines. My initial technical responsibilities involved developing the messaging system for the app. The early stages of my work on this feature invloved developing an activity which could create, delete, parse and display existing conversations in recycler view. I converted the chat activity into a fragment which included changing library dependencies and extensions when we decided pivoting to a fragment-based application would be best. I also developed the pending requests fragment and the implementation of its front and backend features. Due to its relationship to the homepage fragment, I collaborated with Sahas to link the connections. I also supported Divy with converting the Maps activity into a fragment.

2) Khyati: My role as the Frontend Developer for this project included shaping the platform into a user-friendly and visually appealing application. I was responsible for the implementation of GUI for User Authentication, Profile Creation and Homepage among other major functionalities. For the User Authentication GUI, I focused on designing a secure and straightforward login and signup process. I paid attention to user interface design principles, incorporating intuitive form validations, error handling, and clear messaging to guide users through the authentication process seamlessly. Similarly, I worked on translating our lo-fi wireframes and mockups into functional profile creation interfaces. I ensured that users could easily input and manage their personal information, select their academic interests, and define their preferences for potential connections. Finally, I worked with Sahas on creating a visually appealing layout for the Homepage UI that displayed and organized information effectively with smooth transitions to enhance the user experience.

3) Divyangam: I was responsible for implementing the map feature of the app. My initial focus was tracking and updating user location within a specific time or distance using

Google Map API services like location data and location manager. I then used setOnMapLongClickListener to help create markers, added a dialogue box to set meeting details and designed the info window to display the details. I ensured that yellow hue markers were visible to all app users by using Firebase to load all markers every time a user refreshed (close or pause) the map fragment. In addition, I worked on implementing the functionality to create a remove marker button, which eliminates the map and the created event from both map and firebase. I then took CJ's help to convert the Maps activity into a fragment.

4) Chris: I committed under the aliases (cjsu-UW & cshangjie). I was responsible for the implementation of User Authentication and Profile Creation & Editing. For the Profile Creation, I implemented two scripts to pull UW course information and all the BS degrees currently offered by the institution. The two scripts gave me string arrays that I added into the strings.xml of our app and utilized them for the autofill when creating/editing profiles. I also made sure that when creating/editing a profile's courses, duplicates would be handled accordingly. Finally, I implemented the Activity that held each feature's fragment to allow navigation between them.

5) Sahas: I committed under the aliases (SahasG). I was responsible for the general backend, algorithm and homepage. The general backend includes developing methods to communicate with the database such as CRUD functions, organizing the database and modeling a schema, creating a user class along with enums to abstract user data. I also designed and implemented the algorithm and tested it rigorously so that it could function reliably and return a list of users based on the current users information. Finally, I created the Homepage UI along with Khyati and also developed all the code for the functioning of the fragment.

C. Retrospective Insights

1) CJ: This project provided a great opportunity for me to nurture and develop leadership skills. I learned the importance of setting team goals and communicating them effectively to my peers. In addition, I learned how to partition and prioritize key tasks within a project. A significant part of our team's experience involved transforming abstract ideas into tangible solutions which could be developed in Android Studios. From a technical standpoint, programming the app exposed me to essential app components such as fragments, adapters, and models. Working with Firebase was also quite educative.

2) Khyati: This project offered me several valuable insights in communication, teamwork, and project structure. In a project of this scale, where multiple team members with diverse skill sets and responsibilities are involved, effective communication is essential to stay aligned and address any roadblocks. Furthermore, striking the right balance between functionality and simplicity was a learning curve but an important lesson in developing my existing skills as a Frontend Developer. In hindsight, conducting thorough user research and usability testing at the outset would have provided valuable insights, allowing us to make informed decisions about feature prioritization and interface design.

3) Divyangam: Working on the project helped me learn the importance of setting clear goals and deadlines. Examining code snippets, testing different approaches, and researching potential solutions ultimately led me to discover new programming techniques and concepts. The technical challenges we faced also allowed me to hone my problem-solving and critical-thinking skills and gain practical experience with a range of app development tools. It also taught me to appreciate the value of timely collaboration and seeking help from others to arrive at the best possible solution.

4) Chris: Looking over how we initially approached the project and our current standing; we definitely undervalued the importance of laying out a comprehensive plan that encompassed all implementation details. Had we spent more time outlining our backend and app flow, time spent on transitioning features could have been spent on implementing the features from our original design goals.

5) Sahas: I believe that using pull requests may have been the best course of action when working on version control with a large group of people. At first our features were disjointed, and it took time and effort at the end to integrate them with some features requiring larger changes to work with the whole. With PR requests, there would have been more effective collaboration between teammates and more code base knowledge which could have led to better design decisions during implementation.