# Golf Player Time Management

Adam Horle, Alyssa Beeker, Bailey Carlin, Daniel Teel, Elijah Hunt

# Table of Contents

# 1. Project vision

## 1.1. Background

Golf Player Time Manager, or GPTM, is the title of our time saving golf course companion. GPTM has two components, a cloud-based management application and an interactive mobile phone application. The issue we are aiming to solve with golf is the player frustration due to the lack of easy coordination between each player on a course, as well as communication between golf course staff and players. With the nature of golf being played on a large field, players cannot reach each other to determine wait times themselves. This is also an issue that golf course staff cannot currently alleviate effectively due to the lack of online communicative technology, as they still rely on manual solutions to inform players. Not only does this lack of technology affect the issue of wait times, but players also cannot easily request assistance or services. Golf course players and staff both lack effective means of communication involving modern online technology.

## 1.2. Socio-economic Impact, Business Objectives, and Gap Analysis

Our app will improve the down time between rounds for players. The optimization we plan to develop will make playing golf at a participating location more favorable. This means better business for a golf course that utilizes our app, which means they can host more players and games. It also gives players easier access to a golf course's services, boosting sales for refreshments and additional rounds. This could potentially create new jobs on golf courses who use our app.

## 1.3. Security and ethical concerns

Since GPTM relies on GPS location tracking, location privacy is the largest security concern. To avoid attacks, we will be using two secure GPS services, Smart Location and the Google Maps API. Another concern is account information privacy, because we require our players and admins to log in to use our services.

The Administrator app has a high security concern since it is through this service that data is entered and a golf course is monitored. We do not want anyone to be able to gain access to this information especially that regarding the players' location and

behaviours. To combat this an administrator must be email verified to access the service. Password recovery can take place outside of the app but the process sends an email to the account holder only. Account editing capabilities are locked within the app on the support page so outside parties can not gain access to those tools. Administrator authentication status only persists for the current session so navigating away from the page will force users to re-enter their credentials to continue using the service.

The biggest ethical concern of our project is that an outside party can gain access to player locations and behaviours. Another ethical concern is that it introduces more work for golf course employees. This can heighten stress levels for employees and thus it is important for us to make our app highly functional.

## 1.4. Glossary of Key Terms

- Admin: An abbreviation for administrator.
- Golf Player Time Manager (GPTM): This is the name of our project, abbreviated as GPTM.
- Golf Player/Player: This refers to the person participating in a game of golf in real life. They do not work for the golf course and are using our app as a standard user. This type of user is referred to as a “Player” in our design.
- Golf Course Employee: This refers to the person who is employed at a golf course and will be involved in our app through golf course management, tending to services and requests.
- Golf Course Manager/Admin: Refers to the person that sets a request to have their golf course be geofenced as well as us (GPTM team)
- Application Programming Interface (API): a set of functions and procedures allowing the creation of applications that access the features or data of an operating system, application, or other service. More specifically, in this project, things like Firebase, Google Maps, etc.
- Global Position System (GPS): Navigational system that tells the player where they are and if they are in close proximity to or at a golf course
- Geofence: Geographical location  in which some arbitrary code will execute / player is given special permissions, i.e. in this case access specific golf course pages

# 2. Project Execution and Planning

## 2.1. Team Information

The members in our team are Alyssa Beeker, Bailey Carlin, Daniel Teel, Elijah Hunt, and Adam Horle. Together our team has a strong background in mobile and web app development, as well as user interface design. As a team we are adept at Java, Typescript, HTML, and CSS which are required for GPTM.

## 2.2. Tools and Technology

The primary languages we will be using are Java for the mobile app, and Typescript and HTML with CSS for the web app. We will be using Android Studio as our IDE for mobile development and the text editor of each team members choice for web development.

We chose Android Studio because it is Android's official development software and our team has experience using it. We will primarily use Microsoft Visio and Lucidchart for flowchart design. We chose these because they are accessible to each teammate and they are simple diagramming tools everyone is familiar with.

For GPS services, we will be using Smart Location and the Google Maps API. Smart Location is an Android library project which will simplify the usage of location providers and activity recognition. Google Maps API offers a map for us to use in our application for our location features.

Our database will be Firebase Realtime Database. Firebase is a backend service that allows us to share data easily on both our mobile and web application, and offers account authentication.

Angular 9 is being utilized as the framework for the web app. Angular is a fantastic framework because the module and component based design is easy to manage and, if done correctly, is quite scalable.

For collaboration and communication, we will be using Github for version control, Google Drive for all of our written and visual documents, and Trello for planning and delegation. For further communication we have set up an accessible chat group for the team to use. Each tool was specifically chosen because it was compatible, easy to understand, and modern.

## 2.3. Project Plan

| Sprint | Task | Due Date |
|---|---|---|
| 1 | User Auth UI/UX | 1/20/20 |
| 1 | User Auth User Stories | 1/20/20 |
| 1 | Develop Mobile User Auth | 1/20/20 |
| 1 | Mobile User Auth QA | 1/22/20 |
| 1 | Golf Course Registration User Stories | 1/22/20 |
| 1 | Continuous Integration Tests | 1/22/20 |
| 1 | Golf Course Registration UI/UX | 1/22/20 |
| 1 | Sprint 1 Presentation | 1/22/20 |
| 2 | Admin Site UI/UX | 1/24/20 |
| 2 | Admin Site User Auth User Stories | 1/24/20 |
| 2 | Develop Admin User Auth | 1/26/20 |
| 2 | Admin Site Nav User Stories | 1/31/20 |
| 2 | Develop Admin Nav/Layout | 1/29/20 |
| 2 | Set Up Geofencing | 2/3/20 |
| 2 | Admin Site Auth, Nav QA | 2/2/20 |
| 2 | Game Update User Stories | 2/2/20 |
| 2 | Admin Course Overview User Stories | 2/3/20 |
| 2 | Game Update UI/UX | 2/3/20 |
| 2 | Sprint 2 Presentation | 2/3/20 |
| 3 | Requests User Stories | 2/16/20 |
| 3 | Develop Course Registration | 2/14/20 |
| 3 | Develop Admin Course Overview | 2/15/20 |
| 3 | Golf Course Registration QA | 2/16/20 |
| 3 | Requests UI/UX | 2/16/20 |
| 3 | Sprint 3 Presentation | 2/17/20 |

| | | |
|---|---|---|
| 4 | Rework Course Registration and Overview | 2/29/20 |
| 4 | Play Speed Prompt UI/UX | 3/1/20 |
| 4 | Develop Geofencing in the Admin Site | 3/1/20 |
| 4 | Play Speed Prompt User Stories | 3/1/20 |
| 4 | Develop Requests | 3/1/20 |
| 4 | Requests QA | 3/1/20 |
| 4 | Assistance User Stories | 3/1/20 |
| 4 | Assistance UI/UX | 3/1/20 |
| 4 | Player Overview User Stories | 3/2/20 |
| 4 | Player Overview UI/UX | 3/2/20 |
| 4 | Sprint 4 Presentation | 3/2/20 |
| 5 | Develop Assistance | 3/15/20 |
| 5 | Develop Play Speed Prompt | 3/15/20 |
| 5 | Develop Admin Player Overview | 3/15/20 |
| 5 | Develop Location Service | 3/15/20 |
| 5 | Assistance QA | 3/15/20 |
| 5 | Play Speed Prompt QA | 3/15/20 |
| 5 | Admin Player Overview QA | 3/15/20 |
| 5 | UI Style Mockups for Mobile App | 3/15/20 |
| 5 | Sprint 5 Presentation | 3/16/20 |
| 6 | Admin Support Page | 3/29/20 |
| 6 | Mobile App UI Work | 3/29/20 |
| 6 | Admin Site UI Work, Course Congestion | 3/29/20 |
| 6 | Sprint 6 Presentation | 3/30/20 |
| Final | Final Project Poster | 4/13/20 |
| Final | Dry Run Presentation | 4/13/20 |
| Final | Admin Dashboard QA | 4/21/20 |
| Final | Mobile App QA | 4/21/20 |

| | | |
|---|---|---|
| Final | Bug Bash | 4/21/20 |
| Final | Final Presentation | 4/22/20 |

## 2.4. Best standards and Practices

We decided on multiple standards that we have learned in our previous in-major courses. Rather than repeating code in multiple places, make a function instead. Functions for common operations in the app will enable scalability and most importantly, make things more manageable. When working in Angular, linting before a push will promote standardization of code. Leaving comments on code where the purpose or flow is apparent is necessary to enable other team members to start working on the code quicker.

In general to keep our process flowing, we will let only one person work on one component at a time. This will decrease Github merge conflicts to zero. Also on Github, merging master into the current working branch will help with the prevention of merge conflicts. These practices will ensure we can all keep working efficiently.

# 3. System Requirement Analysis

## 3.1. Functional Requirements

Administrators need to be able to register a new course, and declare the geographical location of their course. They must be able to edit the information for each hole, as well as mark the boundaries. Administrators also need to be able to see current players and groups, and what hole they're playing on. They must be able to see requests from players with sufficient options to respond to them. They should also be able to see the wait times for the current games.

Players need to be able to select a course, with a search bar and a list sorted by closest to furthest. They should be able to see the course details along with the wait times. They should be able to use a digital scorecard. They should be able to move their location to the next hole. They should have an option to submit a request to the admin account.

## 3.2. Non-functional Requirements

The application will have two separate components with a simple and intuitive user interface for each. Information needs to be displayed in an easy to read manner.

This means appropriately large buttons with clear and concise language to depict where they take the user. All text in the application must be understandable by the target audience, and also be written to accommodate users with less understanding of the English language.

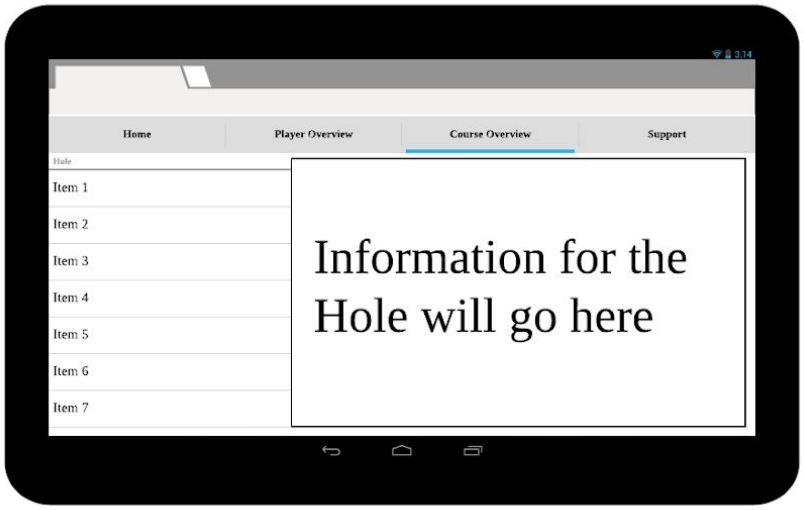
## 3.3. On-Screen Appearance of landing and other pages requirements

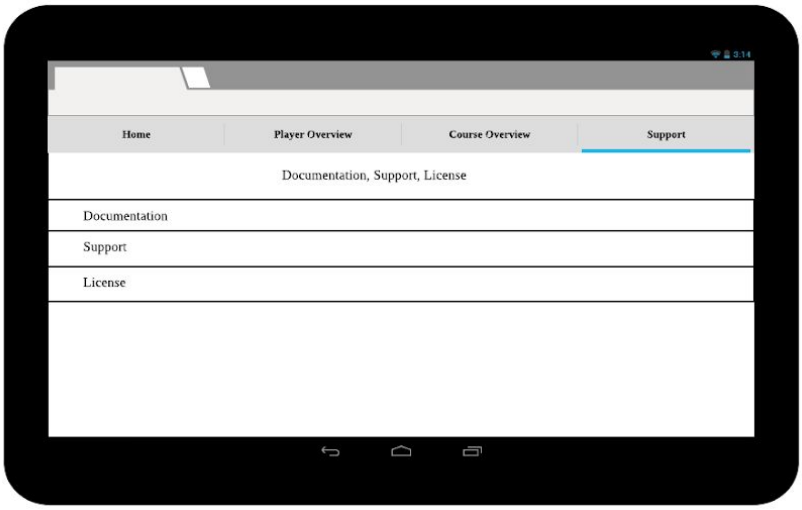
The administrator will land on the homepage of the website, which will have tabs for Home, Player Overview, Course Overview, and Support. These tabs are aptly named to allow for easy navigation for administrators. The Player Overview page will immediately open to two sections, active games and player requests. The Course Overview page will have a sidenav where the administrator can select each hole to edit that hole, which will plainly show each bit of information necessary for players to play on that hole.

The mobile process takes the player from login to immediately choosing a course, as to not slow down the process. After choosing a course they will land on a page where they can start a game or form a group. While a game is active, the player can see their score card, and if they gesture to slide the screen down, a map of the active hole will be shown.

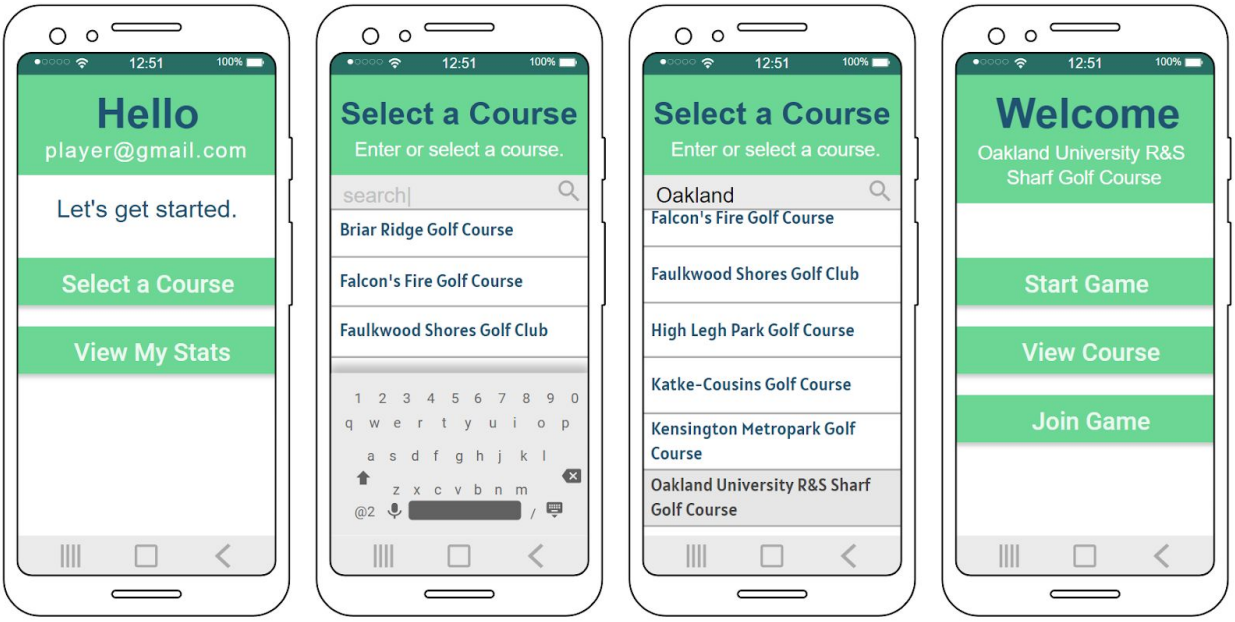
## 3.4. Wireframe designs

Administrator Website wireframe design



Mobile App Wireframe Design




# 4. Functional Requirements Specification

## 4.1. Stakeholders

*Golf players* are the primary users for our application. They operate on the mobile app. As primary users, they will have the biggest part in the design of our app. Firstly, they will want to be able to select a golf course to play at. They will then want the app to assist them throughout playing, this includes directions around the course, tips for playing a certain hole, the time they are taking as well as other players time, player queue information, accurate estimations, and easy access to requesting assistance or refreshments from employees. Other features include general account creation and management.

*Golf Course Managers* are the admin users for our application. They operate on the web app. On the web app, they will want to be able to register their admin account with their golf course, as well as be able to add more golf courses after initial creation if desired. After creation, they will want to be able to edit their golf course's information, add holes and corresponding information for those holes, receive and manage requests from players, and general account management similar to player account management.

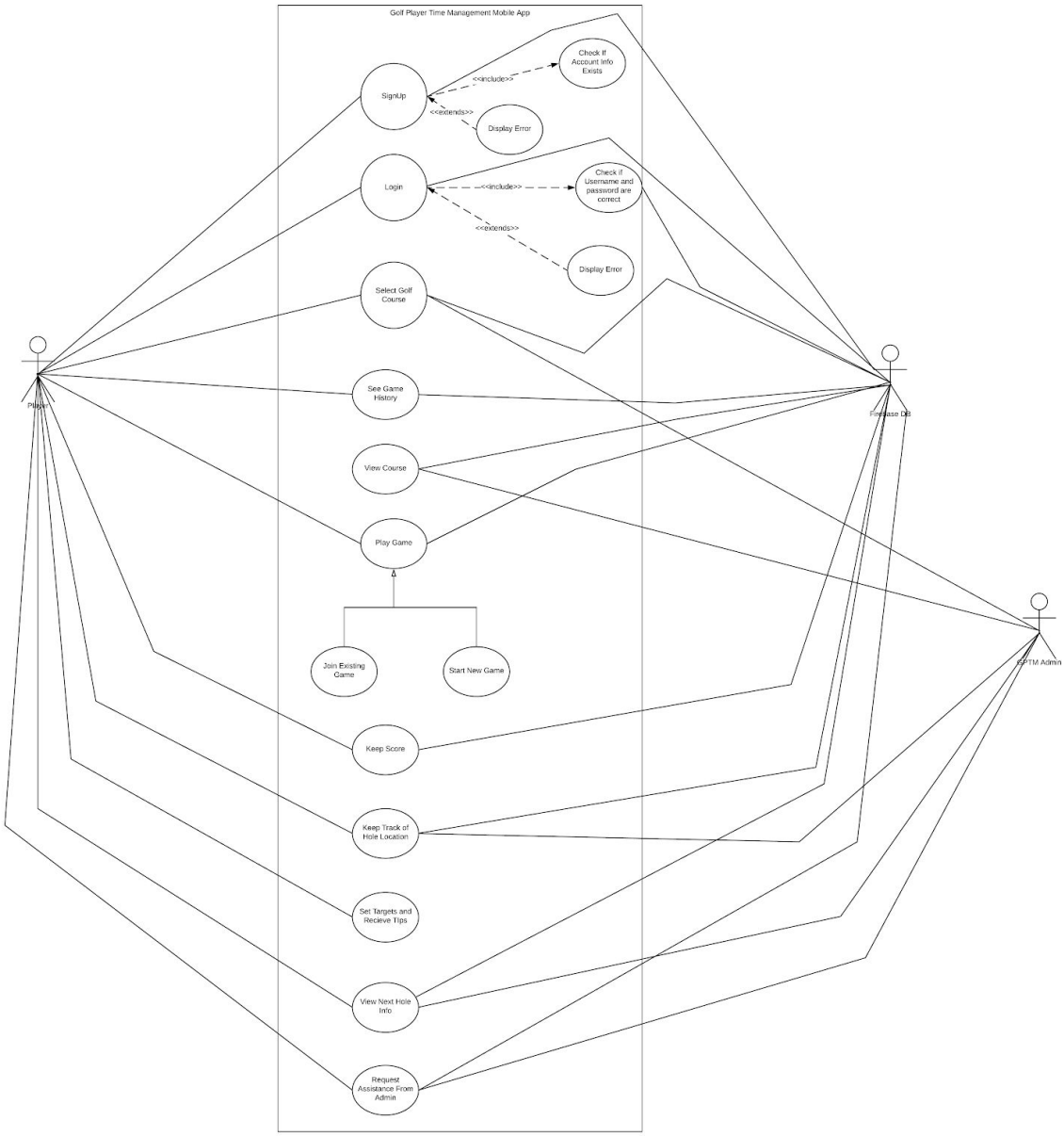
Golf Course Employees (Caddy) are a minor user for our application. Their role is accepting or declining the player requests, as well as fulfilling them. The Golf Course

Manager will handle how this is done, but the request should be available to the Caddy no matter how it is done. The Caddy's do not have accounts on our application of their own.

## 4.2. Actors and Goals

- Player: When "player" is used as an actor, it's referring to the person using the mobile application to participate in a game of golf.
- Admin: When "admin" is used as an actor, it's referring to the person using the web application to manage one or more golf courses.
- Database: Our database is managed through Firebase. When it's used as an actor, it's to demonstrate how Firebase interacts with our application.
- Cloud: The cloud service we use is also from Firebase. The cloud is an important part of the functionality of our app and therefore it's important to differentiate it from the Database when diagramming.
- Web App: "Web App" refers to the website that is used by admin accounts. It's used in diagrams to specify where the activity or sequence is taking place.
- Mobile App: "Mobile App" refers to the mobile application that is used by player accounts. It's used in diagrams to specify where the activity or sequence is taking place.

## 4.3. User stories, scenarios and Use Cases




### Mobile App User Authentication User Stories

“As a Golfer, I want to securely log in to the application so that my history, requests, and preferences are saved.”

- When I launch the app, then I should be directed to the authentication screen on start up.
- If I do not have an account, then the following events will happen:

    - If I do not have an account already, then there should be a selectable option for creating one.
    - If I select the “Sign Up” option, then I should be prompted to either sign in with an existing Google account or create a new one.
    - If I do not sign in with an existing Google account, then I will be asked to provide an email and password to sign up.
    - If I input a password, then I should be asked to confirm and type the same password twice.
    - If I do provide a password, then the field will have validators to ensure that it meets minimum security requirements.
    - If I leave any of the fields blank, then the “Sign Up” button will be disabled and I will not be able to proceed.
    - If I provide valid information, then I will be logged in and redirected to the “Check In” screen to register at a golf course.
    - If an email that I want to register is already in use I would like the system to tell me to use a new email
    - If I am registering an account and I forget to put information into one of the fields (email or password) I would like the app to disallow me from registering.
- If I do have an account, then the following events will happen:
    - If I do have an existing account, then I will select “Sign In” on the startup screen.
    - If the email and password match an existing email and password combination, then I will be able to log in.
    - If I provide the correct credentials, then the “Sign In” button will be active and I will be able to select it.
    - If the “Sign In” button is selected, then I will be redirected to the “Check In” screen to register at a golf course.
    - As an admin I would like to have a separate login page.
    - As an admin, to stop spam accounts, I would like that all emails be legitimate, i.e. test@test.com will NOT work.
    - If I am a returning user and I forgot my password I would like to be able to create a new one through email

## Golf Course Registration User Stories

“As a player, I want to view a searchable list of golf courses and choose which one I want to
Play.”

- After securely logging into the app, I should be directed to the Golf Course Registration screen.
- As a player, I can see a list of golf courses sorted by distance, with closest at the top of the list and furthest at the bottom.
- As a player, I can type in a search field at the top of the screen that will allow me to search for a golf course.
- As a player, after selecting a golf course from the list I will be taken to another screen with more information about the corresponding golf course, including address, hours, website URL, and phone number.
- As a player, on the second screen I can select “Yes” or “No”.
- As a player, If I select “Yes” I will be directed to the Main Game screen.
- As a player, If I select “No” I will be returned to the Golf Course Registration list screen.

“As an admin, I want to be able to register my golf course so that players can use it on the app.”

After securely logging into the app, I should be directed to the Golf Course Registration screen.

- As an admin, I can click on a button on the Admin Dashboard page that will redirect me to the Golf Course Registration page.
- As an admin, I will be prompted to fill out a form asking for my golf course’s name, address, state, zip code, description, website, and phone number.
- As an admin, I can press submit after filling in every form, which then adds the information to the database.
- As an admin, if I do not want to submit the form, there will be a return button so that I can return to the Admin Dashboard.
- As an admin, I will be redirected to the next screen after pressing submit, which will have a message according to the success or failure of the submission.
- As an admin, on the success/failure screen there will be a return button so that I can return to the Admin Dashboard.

## Admin Site Login User Stories

“As an administrator, I want to securely login to a dashboard that provides information on current players.”

- When an administrator launches the website, then he will be automatically directed to the sign in page.
- When the administrator reaches the sign in page, two fields (email and password) must be entered.
- When an already created email and password is entered, then the administrator is redirected to the dashboard page.
- When invalid credentials are entered, then the following events will happen:
    - When an administrator enters an invalid email/password, then an error message will be displayed alerting them that they have entered an invalid username and/or password.
    - When an administrator enters no information on one or both of the fields, then an error message will be displayed saying that the field is required.

NOTE: Since this is an administrator dashboard that only they will have access to, there is no registration option on this page.

## Admin Site Navigation User Stories

“As a logged in administrator, I’d like to be able to navigate to multiple tabs corresponding with major functions to keep the layout easier to read.”

- When an administrator logs in, then they will be redirected to the home page.
- When an administrator logs in, there will be a toolbar at the top of the page with the app name and a logout button.
    - If an administrator selects the log out button, then the user’s session will expire and they will be redirected to the login page.
- When an administrator is on the home page, then there will be a navigation bar on the top of the screen under the toolbar.
    - If an administrator is logged in, then the navigation bar will be split into 5 categories.
        - The categories are: home, player overview, course overview, register, and support.
            - If an administrator selects the home tab, then they will be redirected to the home page.

- If an administrator selects the player overview tab, then they will be redirected to the player overview page with data pertaining to the current users.
    - The data will be split into player name, status, and the course that they're playing at.
- If an administrator selects the course overview tab, then they will be redirected to the course overview page with data pertaining to all the courses enrolled.
- If an administrator selects the register tab, then they will be redirected to the register page and can create a new user account.
    - When invalid credentials are entered, then the following events will happen:
        - When an administrator enters an invalid email/password, then an error message will be displayed alerting them that they have entered an invalid username and/or password.
        - When an administrator enters no information on one or both of the fields, then an error message will be displayed saying that the field is required.
    - If valid credentials are entered and the register button is clicked, then the account will be created.
    - If the Register with Google button is clicked, then an account can be created from an already existing Google account.
- If an administrator selects the support tab, then they will be redirected to the support page with helpful hints.
    - If an administrator is on the support page, then they will have access to a walkthrough on how to register a new player, a link to download the app from the Google Play store, a contact email, and the app's license.

## Player Overview User Stories (Admin Dashboard)

- As an authenticated administrator, when I select the Player Overview tab then I will be redirected to the Player Overview tab that houses player data for active users at my course.

- As an authenticated administrator who is on the Player Overview tab there are two headers: Player Requests and Active Players.
    - As an administrator, I would like the Player Requests accordion to house a table of requests that shows:
        - The type of request
        - The player making the request
        - The player's location
        - An action button to acknowledge the request to let the player know that the request is in the process of being fulfilled
    - As an administrator, I would like the Active Players tab to have data pertaining to who is playing and where. The table should include:
        - Player name(s)
        - Player locations
        - Running clock of play time

## Requests User Stories

- As a golf course owner, I would like to be able to register my golf course.
- As a golf course owner, I would like to be able to register multiple golf courses under my franchise.
- As a player, I would like to be able to use the GPTM app without having to log into some account.
- As a player, I would like to know what the hole map looks like using the GPTM app.
- As a player, I would like to be able to know what the par score is for a given hole.
- As a player, I would like to be able to keep track of what my score currently is using GPTM.
- As a player, I would like to be able to add to my own score using something like a button on GPTM. i.e. pressing = score +1
- As a player, I would like to be able to subtract my own score using something like a button on GPTM incase I make a mistake. i.e. pressing = score -1
- As a player, I would like to be able to subtract my own score using something like a button on GPTM incase I make a mistake. i.e. pressing = score -1
- As a player, I would like GPTM to tell me if I was above par.
- As a player, I would like GPTM to tell me how much above par I was.
- As a player, I would like GPTM to tell me if I was below par.
- As a player, I would like GPTM to tell me how much below par I was.
- As a player, I would like to know which hole I am currently on using GPTM.

- As a golf course owner, I would like more functionality on the GPTM site.

---

- As a player I would like to be able to make a request to the front desk for a food product, a different club, etc.
- As the golf course worker I would like to know which hole a player is at when I go to deliver his request.
- As a golf course owner I would like to be able to decline a request if I don't want to fulfill the request.
- As a player I would like to receive some sort of notification if the request went through.
- As a player I would like to receive some sort of notification if the request is denied.
- As a player I would like to receive some sort of notification if the request is accepted.
- As a golf course owner I would like some notification for receiving a request.
- As a golf course owner I would like to be able to see all of my requests on some web page.
- As a golf course owner I would like to see what user sent me the request.
- As a golf course owner I think that only registered users will be able to make requests.

## Assistance User Stories

"As a Player I should be able to request assistance from the golf course I'm playing at."

- As a Player I should see a "Request Assistance" button on my player dashboard.
- As a Player on the Assistance screen I should see radio buttons that allow me to choose what request I have. The options are:
    - Refreshments
    - Directions and/or Visibility Assistance
    - Ball Retrieval Assistance
    - Maintenance, Obstacle, and/or Cleanliness issue
- As a Player, after selecting a button I can press submit.
- After submitting the request will be shown on the Player Overview tab of the admin dashboard.
- As a Player, I should see a notification when an admin accepts my request and is in progress.

As an Admin I should be able to see the requests my players have made and be able to accept or decline the request.

- As an Admin I should see a “Player Requests” tab in the Player Overview page.
- As an Admin I should be able to see the type of request, the player who is making the request, the location of the player, and an accept button.
- As an Admin, if I press to accept the player’s request they should be sent a notification accordingly.

## Game Speed User Stories

- As a player I would like to know the estimated time on this hole.
- As a golf course owner I would like to set an upper limit on play time on certain holes so I can get more people through my golf holes.
- As a player I would like to know how long it would approximately take me to get to the next hole. (i.e. shows next player in lines current time)
- As a player I would like to know what position I am in in the virtual line.
- As a player I would like to know how long I have been currently playing on a hole.

## Admin Support and UI User Stories

- As a player I would like a clean, simple looking and easy to read UI
- As an admin/developer I would a clean and simple UI so more users enjoy and use the app
- As a player I do not want too many buttons that perform very different functions on the same screen incase I press the wrong button
- As an admin I don’t want that many buttons with different functions on the same screen so there is less user error

---

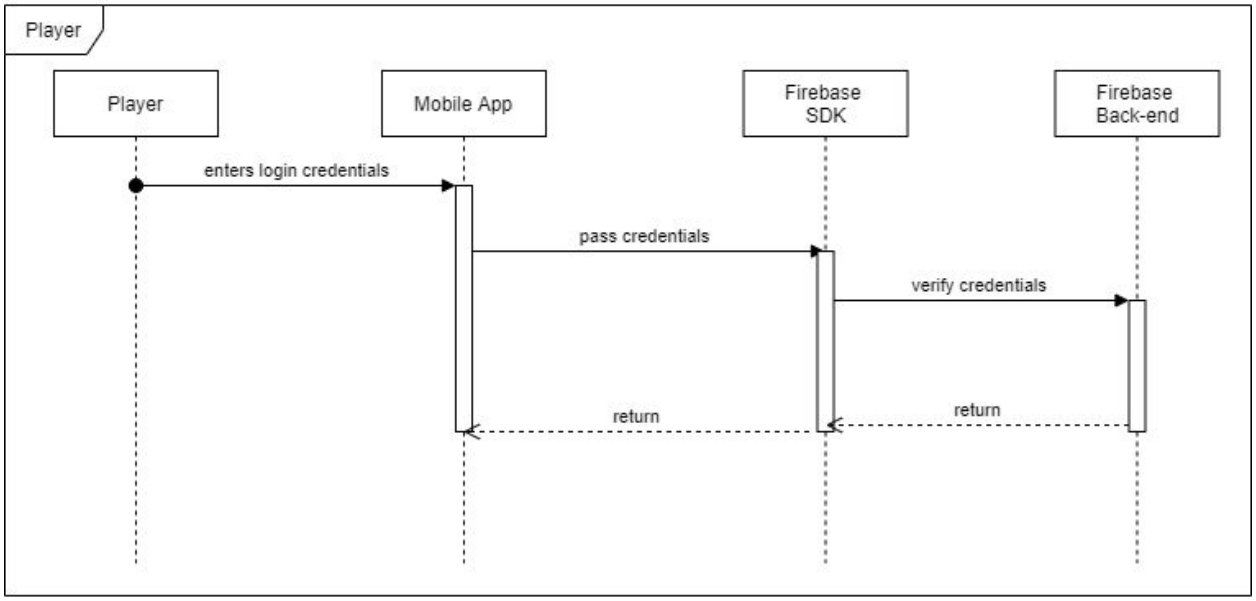
- As an admin I would like to be able to submit a support ticket if I find an issue with the app
- As a developer I would like to know if there is an issue with the app so it can be fixed as soon as possible
- As an admin I would like to be able to submit a support ticket if I feel like there should be a function in the app if it is not there
- As a developer I would like to know how I can improve the app and add functions so more golf courses use the app

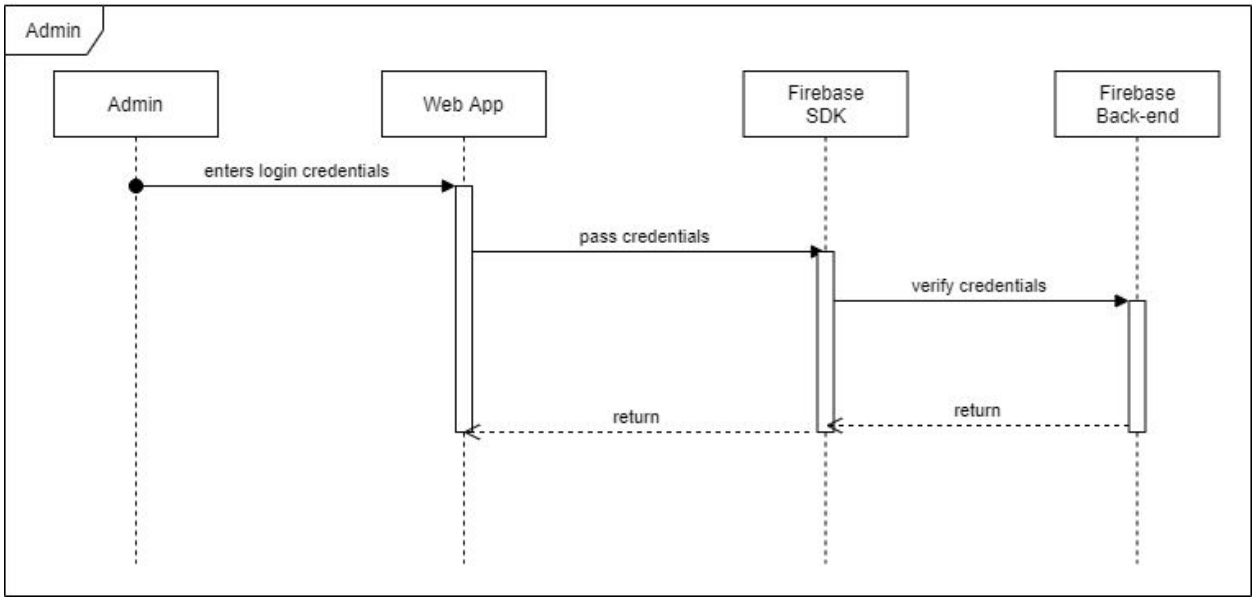
## Admin Site Course Congestion User Stories

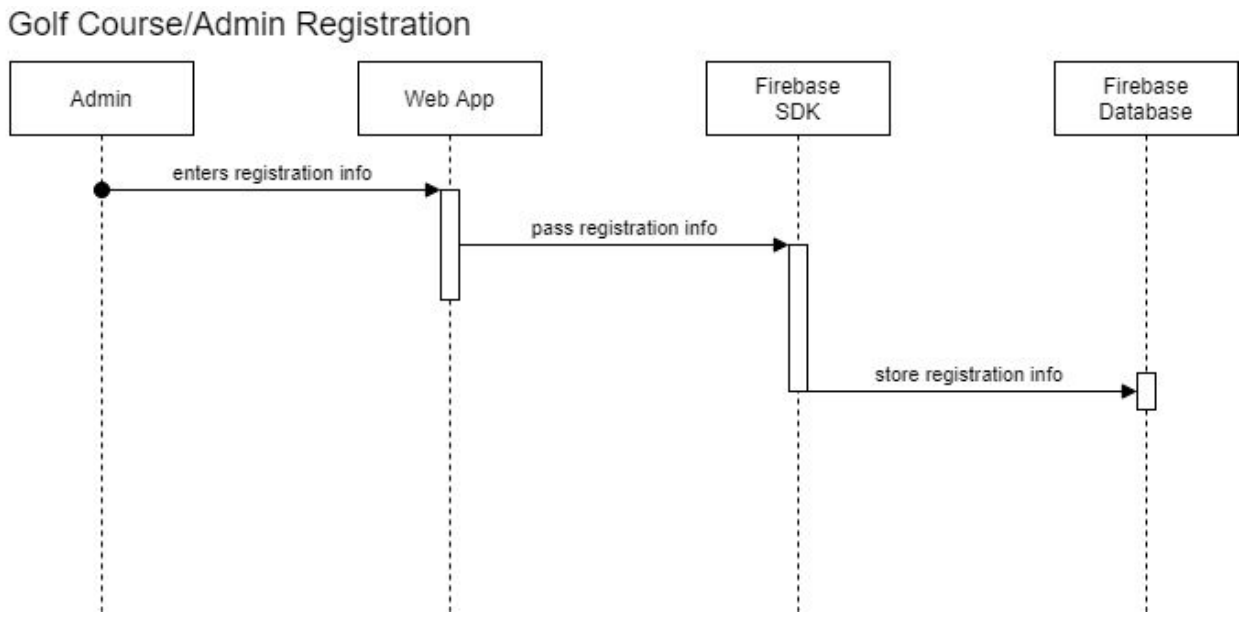
- As an admin, I would like fairly accurate predictions for wait times and accurate counts of groups at each hole on my course
- As an admin, I would like to see this information presented in an easy to read and focused manner
- As an admin. I would like this information to accurately reflect the games on my course at any given time
    - I need this information to be updated constantly so that I can determine which holes may need staff members sent over.

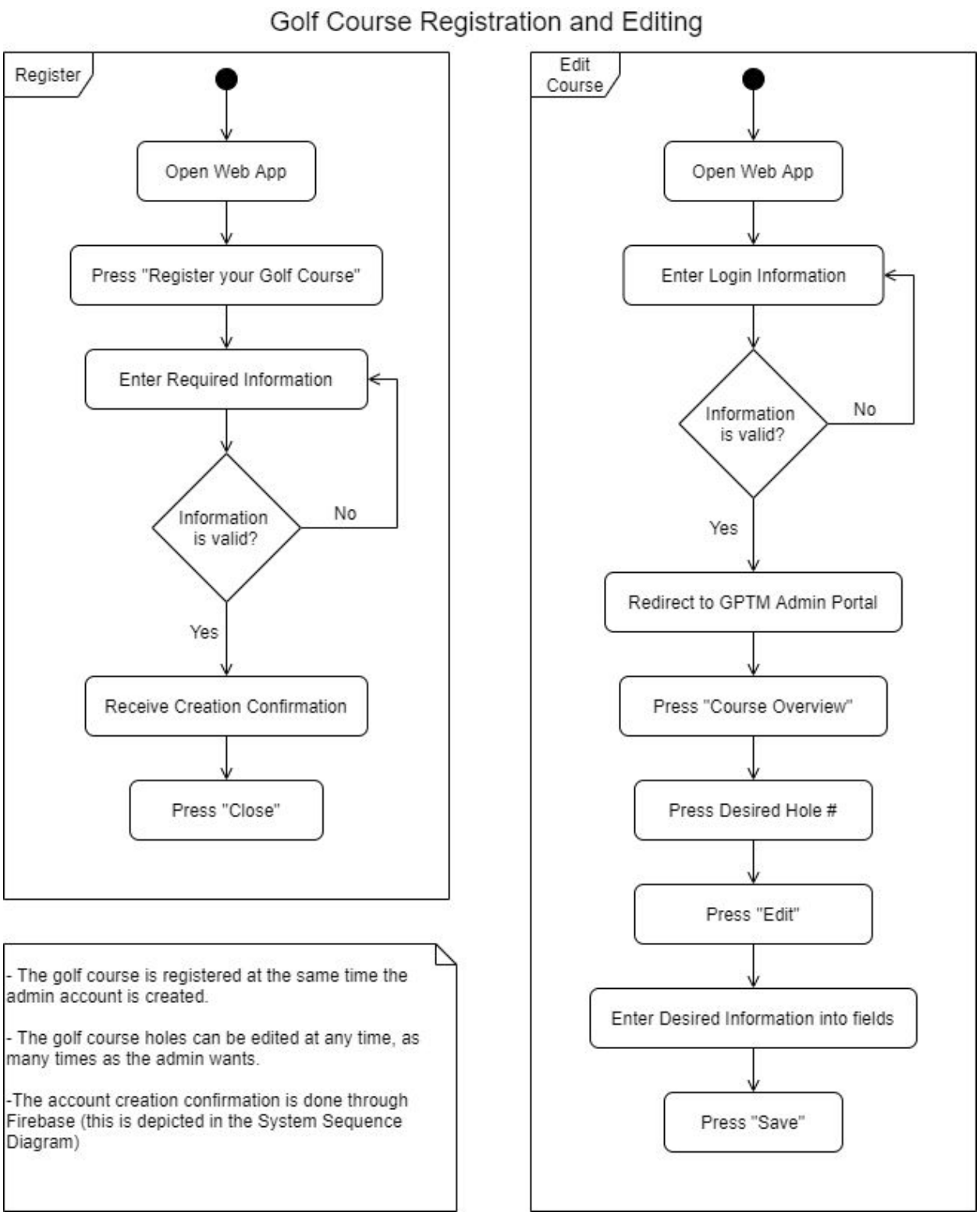
# 4.4. System Sequence / Activity Diagrams

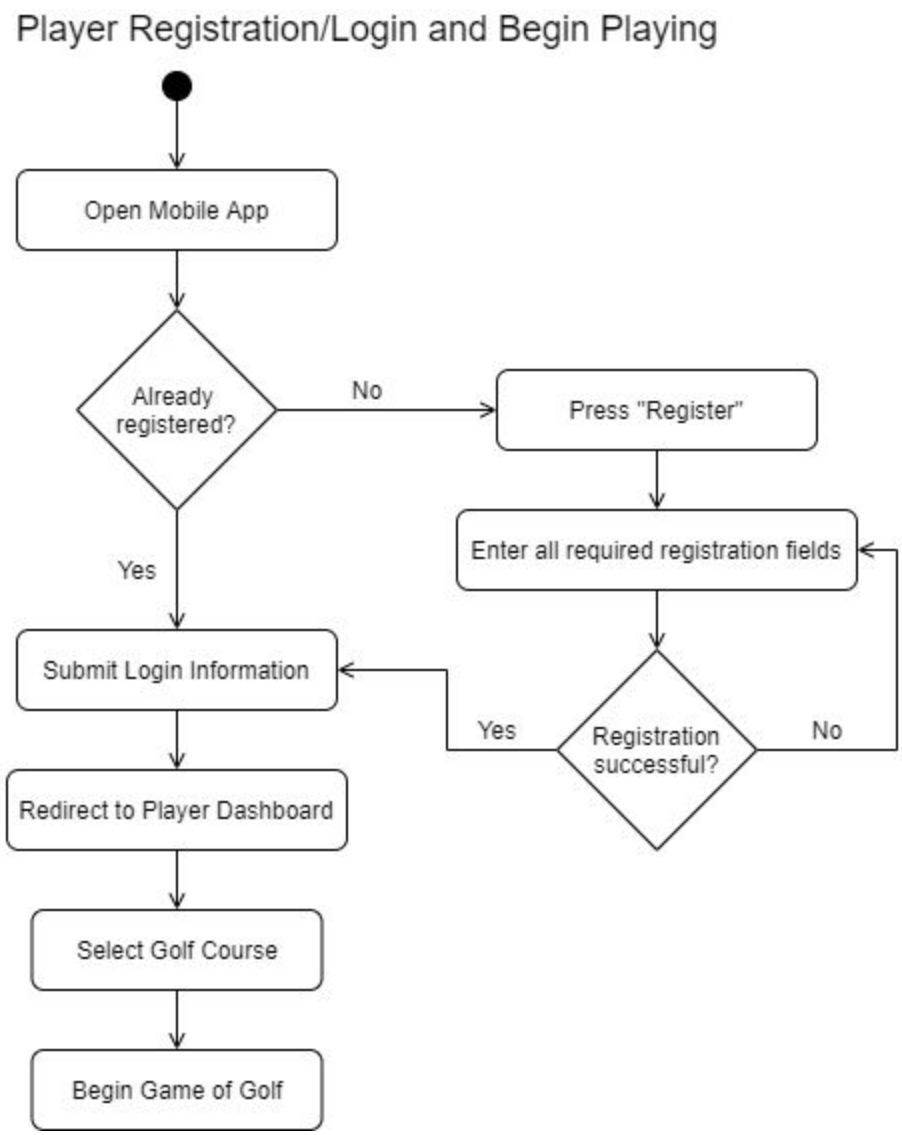
Player and Admin Login Sequences

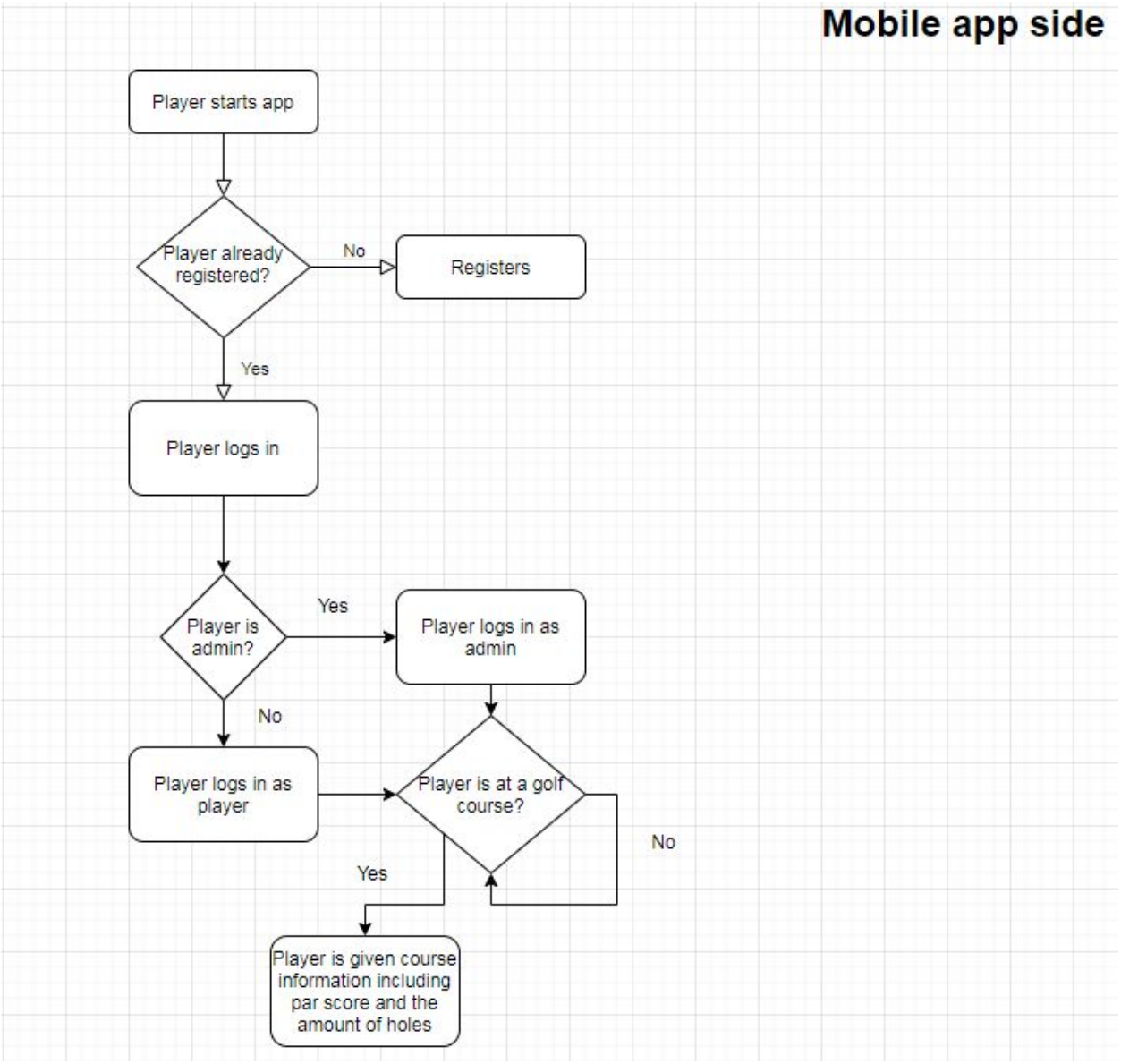



Golf Course/Admin Registration
Admin
Web App
Firebase SDK
Firebase Database
enters registration info
pass registration info
store registration info


Golf Course Registration and Editing
Register
Open Web App
Press "Register your Golf Course"
Enter Required Information
Information is valid?
No
Yes
Receive Creation Confirmation
Press "Close"
- The golf course is registered at the same time the admin account is created.
- The golf course holes can be edited at any time, as many times as the admin wants.
-The account creation confirmation is done through Firebase (this is depicted in the System Sequence Diagram)
Edit Course
Open Web App
Enter Login Information
Information is valid?
No
Yes
Redirect to GPTM Admin Portal
Press "Course Overview"
Press Desired Hole #
Press "Edit"
Enter Desired Information into fields
Press "Save"


Player Registration/Login and Begin Playing
Open Mobile App
Already registered?
No
Press "Register"
Enter all required registration fields
Yes
Submit Login Information
Yes
Registration successful?
No
Redirect to Player Dashboard
Select Golf Course
Begin Game of Golf




(Old mobile app activity diagram above)

# 5. User Interface Specifications

## 5.1. Preliminary Design

Player and Admin Registration Page:

For our preliminary registration page design, we will include fields for the user to enter an email and password. There will also be a button for the user to use their google account to register. After the user registers they will be taken to the dashboard page for their respective account type.

Admin Dashboard Page:

The Admin Dashboard Page is a webapp only, there is no mobile application for admins as it is unnecessary. On this page the admin can view tabs for their Home, Course Overview, Player Overview, Requests, and Support.

- On the Home page
- The Course Overview page will have a navbar to select the hole and a 5 tile format to show the hole's data.
- On the Player Overview page the admin can view the players currently playing on their course, including where they are, as well as any player requests.
- On the Support page the admin can view helpful information and tutorials about the webapp. They can also get help with their account here.

Player Dashboard Page:

The Player Dashboard Page is a mobile app only. This is where players are directed after logging in. After selecting a golf course, there is where the player will be able to manage their game. They will also be able to request assistance from the golf course from a button on this page. The buttons we expect to include on this page are as following:

- Join Game: This button will allow the player to find a location nearby to select and then join.
- Next Hole: We need the player to tell us when they are done with their hole. After pressing this the app will put them in queue for the next hole.
- Add/Remove Stroke: Simple buttons we will include to help players keep track of their game. There will be a counter next to these buttons.
- Overview: Displays all users in current group and their current scores
- Next hole: Displays information about the next hole -- expected wait time

Request Assistance Page:

This page will have a version for admins and players. The admin webapp version is where admins can go to view requests, including the requesting players name, location, and request description. Admins can accept or deny these requests. Players can find the request assistance button on their player dashboard. When that button is pressed, they will be taken to a screen where they fill in their request description. They will then receive a notification when the admin accepts or declines their request.

Error/Exceptions:

Since golf courses can have varying wifi or service strength it is important for the sake of user enjoyment that we secure their space in a fair way. In the event a player abruptly disconnects from the app, whether by loss of Internet connection or their phone

shutting off from low battery, we plan to save their space in their selected queue or course for 5 minutes. If the player does not log back in before 5 minutes, they will be removed from the queue or course they were registered in. This amount of time allows the player to make adjustments, but also doesn't aggravate other players who may be waiting behind them.

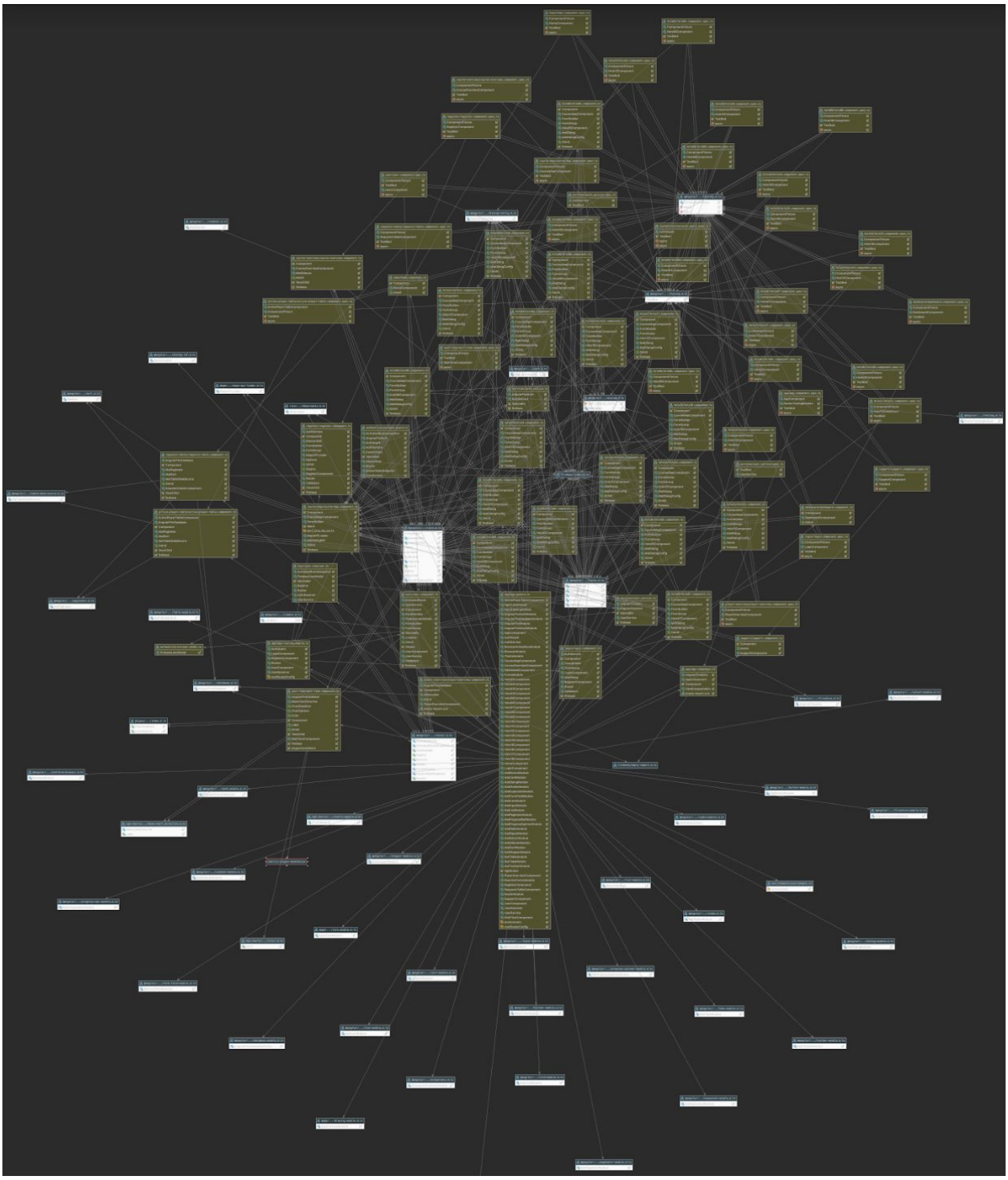
## 5.2. User Effort Estimation

Ultimately, we do not want our users to expend much effort using our apps. Our web and mobile applications should be quick, intuitive, and easy to make swift actions. The user should not spend more than a few minutes using our app at a time. For example, players should be able to quickly open their phone, add a stroke, and put it back in their pocket. They should not be expected to log back in during their game, nor should they have to navigate through multiple screens or menus to achieve their goal. For admins, their primary function during long term usage of the app is to be able to check requests. All related information to player requests should be upfront and on one page. When they accept or decline a request, a message of confirmation should pop up on the screen but not a dialog box. The request should then disappear automatically as well. This interaction, like the example player one, should only take a few seconds.

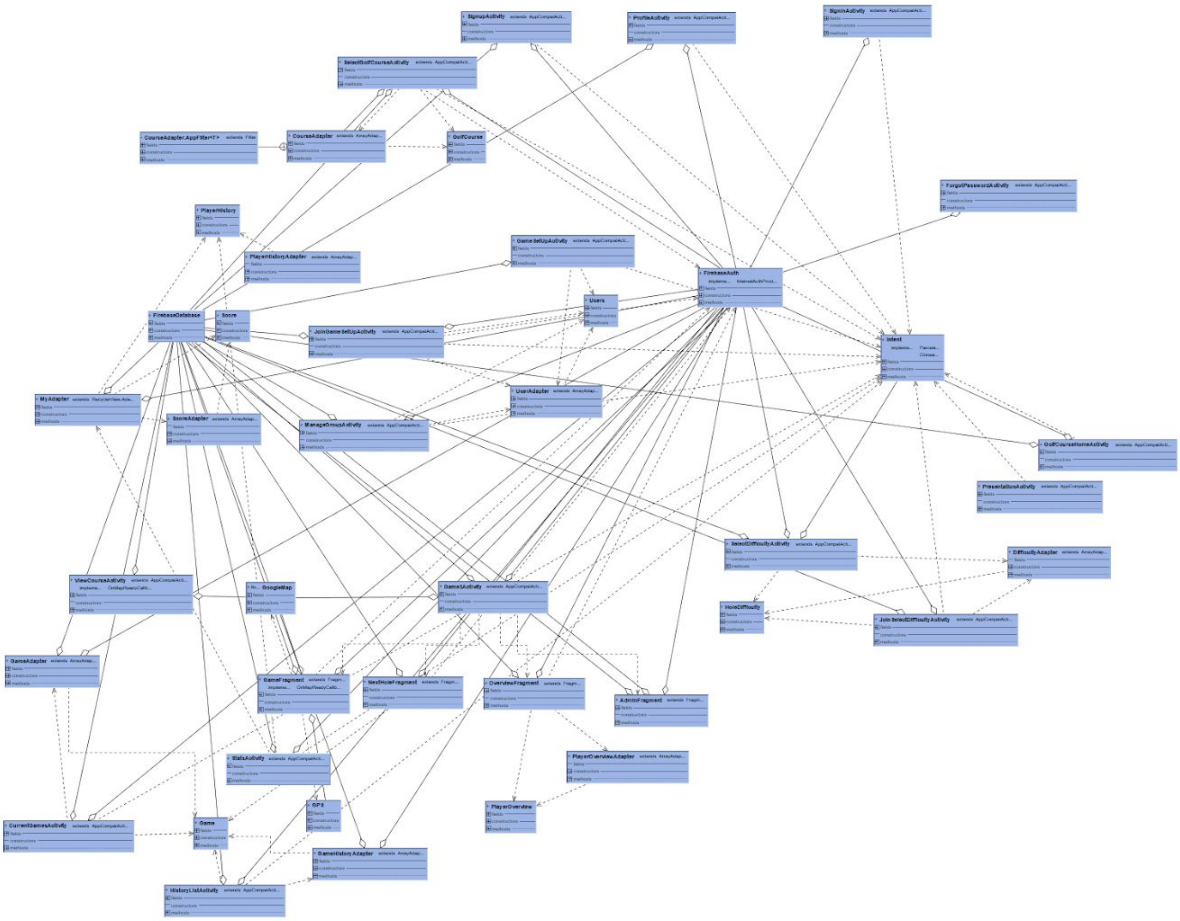
# 6. Static Design

## 6.1. Class Model

Administrator Website Module Diagram



Mobile App Module Diagram



## 6.2. System Operation Contracts

Administrator Website

*Course Registration:*

Precondition: There is a golf course that wants to be registered to use the GPTM application.

Postcondition: A check is run to ensure that the golf course is unique, an actual golf course, the course is not already registered, and the user's email and password combination is unique. After the check is performed a course is created in firebase with placeholder values in a predefined structure to create a new golf course in the system.

*User Authentication:*

Precondition: There is an already registered user that wishes to login to access their courses dashboard.

Postcondition: The user's credentials are checked. First, the email is checked to determine if the account already exists. Next, the password is checked to see if it is the right combination for that user. If it is the right combination, the user is redirected to the dashboard. If the combination is not correct, the system reads the error message and formats it into a user friendly message so that they can see what went wrong.

*Player Overview:*

Precondition: There is an already authenticated user that is on the dashboard and on the "Player Overview" tab.

Postcondition: A query is sent out to firebase that pulls the information for the "Player Requests" table. The query looks at the requests that are specific to the course and sets the table datasource with this reference as an observable to constantly pull in new entries. The data is sorted by time. When the complete icon is selected then the entry in Firebase is removed.

For the "Active Games" table the same process is implemented except with the reference made to the Games table in Firebase. The data is sorted by the hole number that each game is on.

*Course Overview:*

Precondition: There is already an authenticated user that is on the dashboard and on the "Course Overview" tab.

Postcondition:

1. If on the "Congestion" page then there is a query sent out that counts the number of games that are on each hole. This is displayed in the "Groups at Hole" table. This number is taken and multiplied by an average time factor which is multiplied by the number of strokes for a par on that hole. This information is displayed on the "Estimated Wait Time in Minutes" table in both a standard table and line graph format.
2. If on any of the holes' pages then each area on the tile is populated by the data that is in Firebase. Each hole is queried and the table's contents are filled with this returned information. When the edit button is pressed then the data entered is pushed and inserted into Firebase if the user hits save. The geofencing pop up is a shared component that looks for one point for the hole and one polygon to be drawn. If save is clicked then this information is pushed to Firebase. The field below will tell users if the

geofence has already been drawn by seeing if the node actually exists in Firebase for the hole that the user is viewing.

*Support:*

Precondition: There is already an authenticated user that is on the dashboard and on the “Support” tab.

Postcondition: There are pre-populated messages for walkthroughs on how to use the various features on the website. If the user clicks on the email hyperlink then the user is directed to a new tab and an email is generated using the users already logged in email account.

## Mobile App

*User Registration:*

Precondition: User that wants to register needs to have an email

Postcondition: Email is checked against firebase records if the account exists or not, if it does an error is thrown, otherwise the user is now a registered user and can use the app to its full potential (can see history, other user’s emails on hole, add other users to your group, etc.).

*Course Selection:*

Precondition: There is a golf course that has already been registered through the GPTM site.

Postcondition: The desired course is selected and the appropriate information for each hole is pulled and formed to fit the game fragment module.

*Group Selection:*

Precondition: There must be an ongoing game at that specific golf course

Postcondition: After either joining the game through simply selecting it through the list, or by joining it through the invite that was sent to you, you are sent to the game setup screen.

*Game Setup:*

Precondition: A golf course has been selected

Postcondition: The user is able to add groups if so desired, the user must then select their hole difficulty (i.e. blue, green, pink).

*Game Fragment:*

Precondition: The game setup has been completed, users may have been added to the group, difficulty MUST have then also been selected
Postcondition: The user is then free to play the game, add strokes to their score which will then be logged by firebase, likewise going to the next hole will also be logged.

*Next Hole Fragment:*

Precondition: The user is playing a game and has selected the next hole fragment at the bottom of the screen
Postcondition: The user can see if there is a game being played on the next hole

*Overview Fragment:*

Precondition: The user is playing a game and has selected the overview fragment at the bottom of the screen
Postcondition: The user can see who is playing on the same hole (user's group partners) and their scores

*Admin Fragment:*

Precondition: The user is playing a game and has selected the admin fragment at the bottom of the screen
Postcondition: Users can make a selection of their choice of assistance, more golf balls, food, drinks, etc.

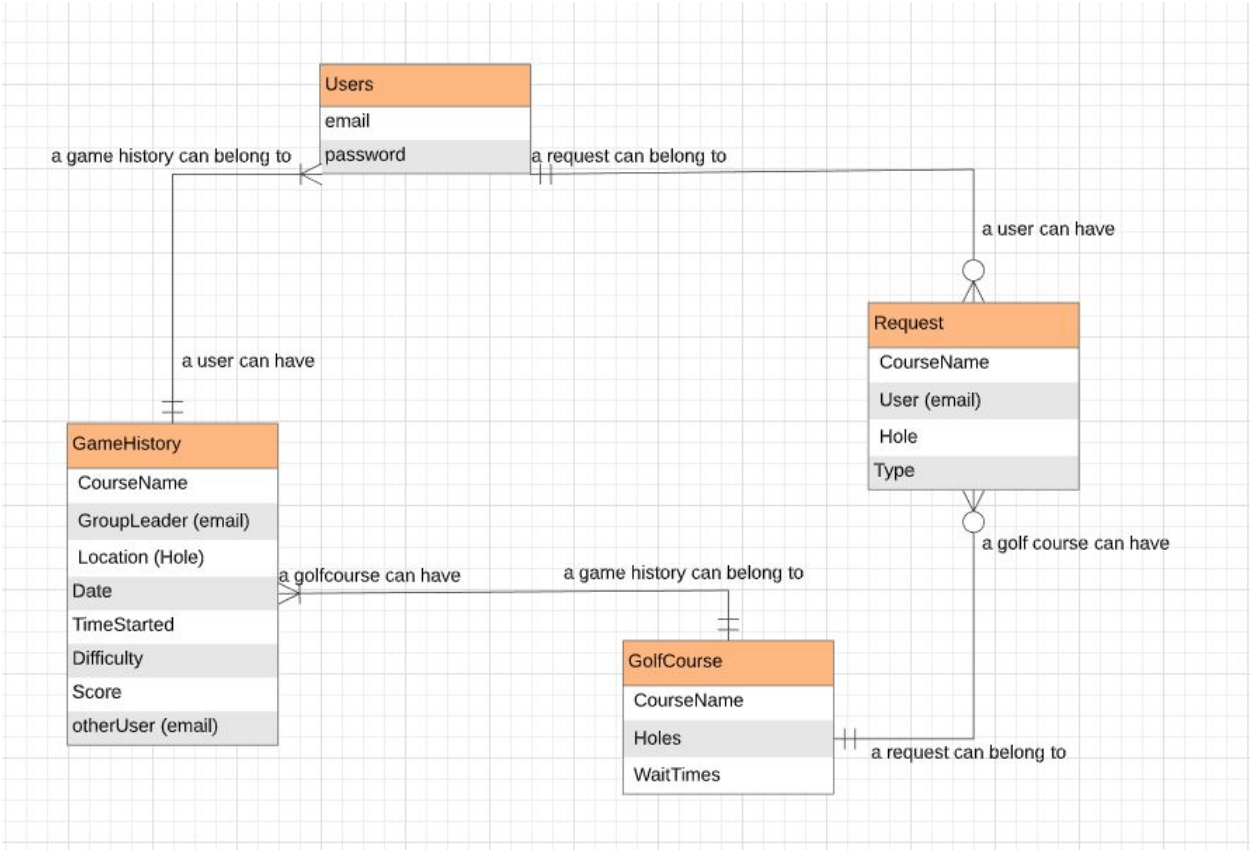
## 6.3. Mathematical Model

The only real mathematical model used in GPTM is the model used to calculate hole wait times.

WaitTime = (holePar * 3.2 * holeQueue)

This model is created based on the assumption that it should take 10 minutes for a 3 par hole, 13 minutes for a 4 par hole, and 15 minutes for a 5 par hole. This data comes from an article from Golf News Net. Using these averages we can deduce that it will take approximately 3.2 minutes for each par stroke on a hole. This can be used to calculate a wait time by multiplying this average by the actual par of the hole and then by the number of groups that are currently on the hole.

## 6.4. Entity Relation




**Relations:**

**Users:** A user's email is passed to Request and GameHistory

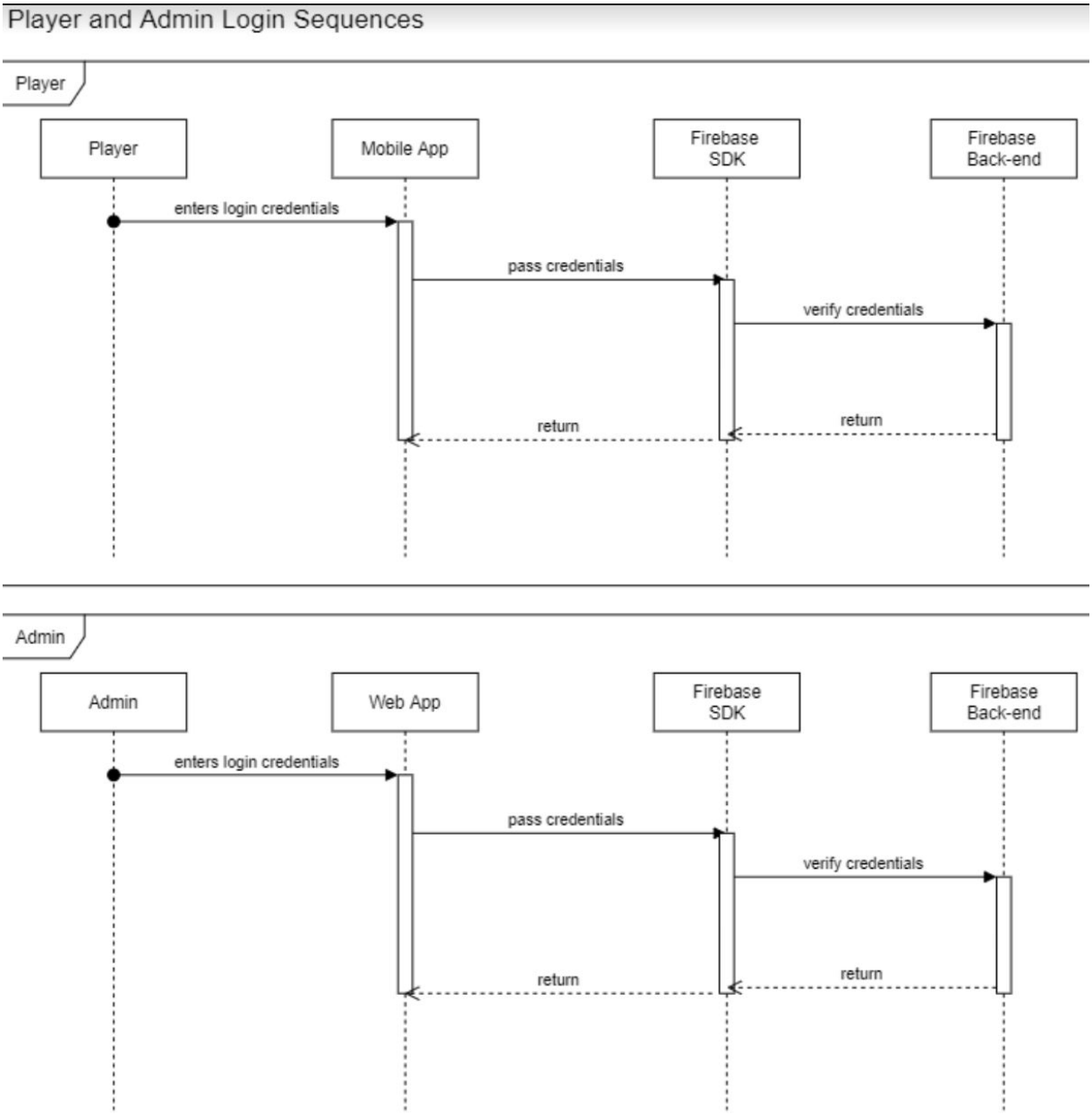
**GameHistory:** Takes User's email and makes a new entry to a specified GolfCourse

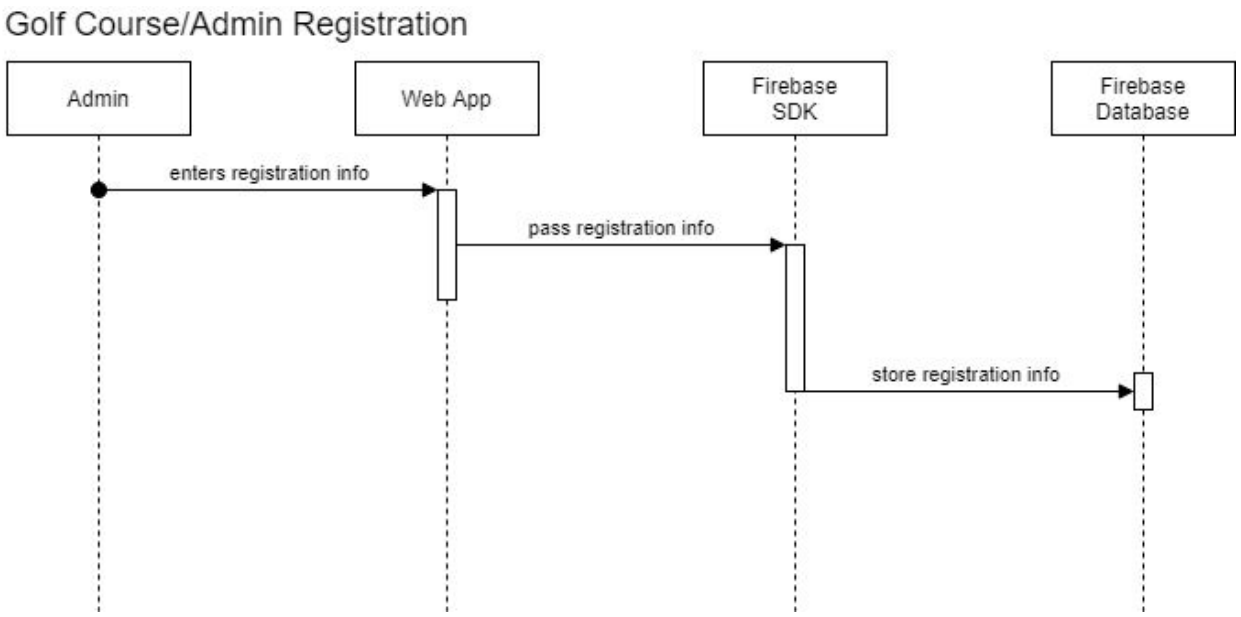
**GolfCourse:** Holds all course information, shares info such as the name of the course and hole information to GameHistory

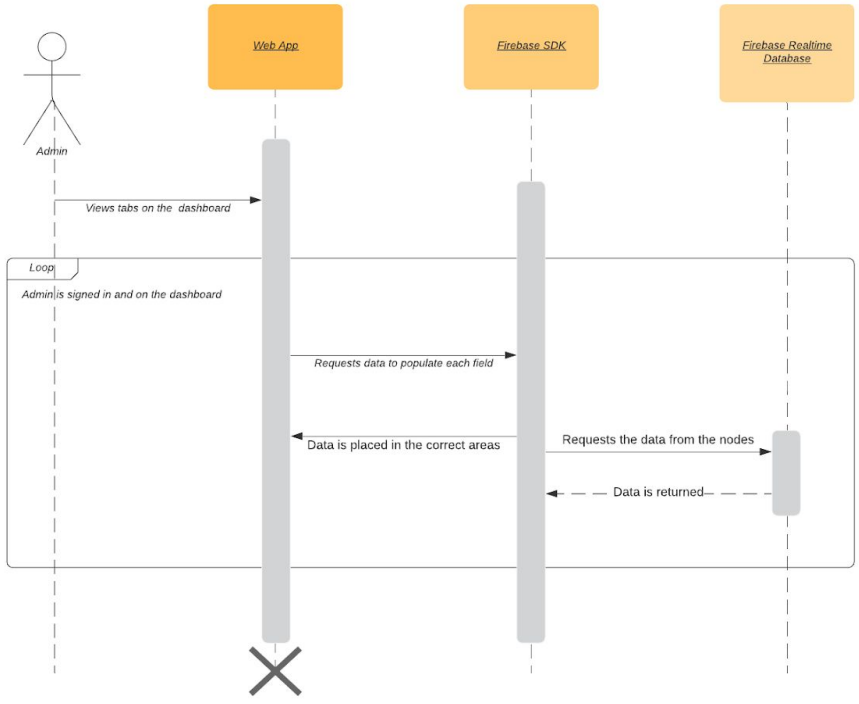
**Request:** Takes user's email and current hole information and makes creates them for the specified golf course

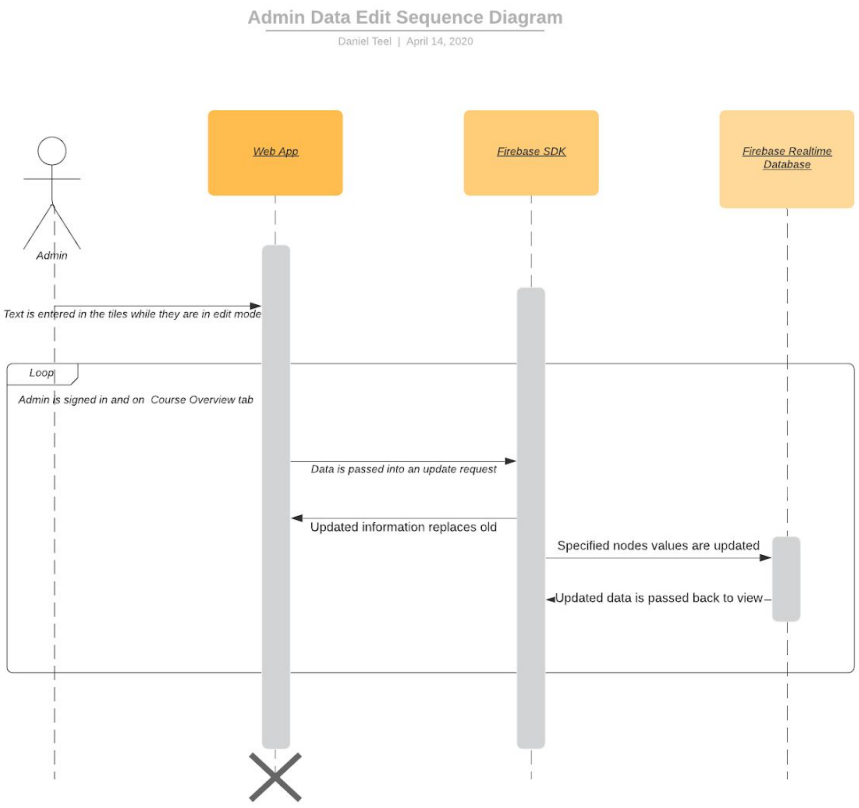
# 7. Dynamic Design

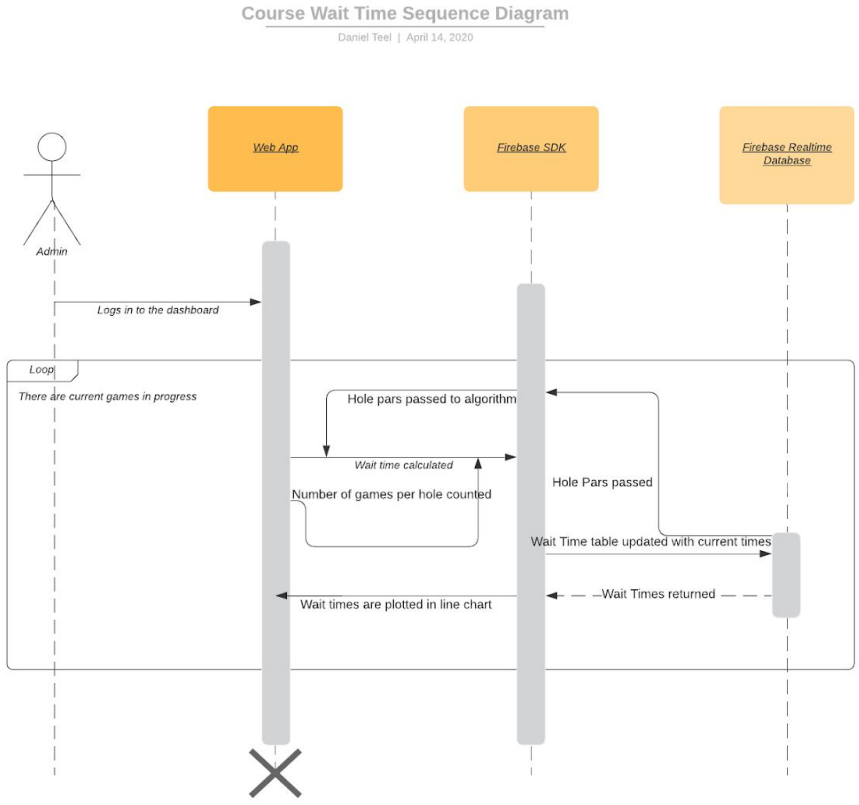
## 7.1. Sequence Diagrams

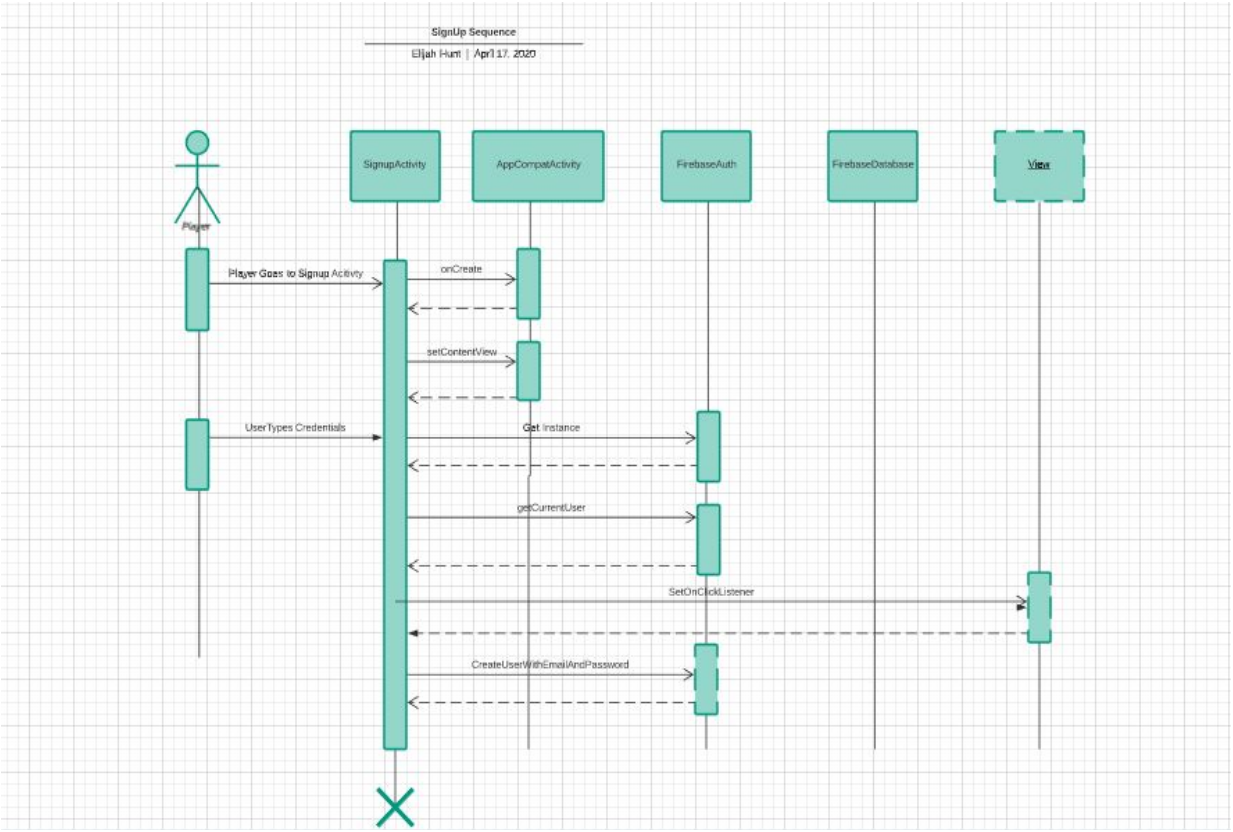


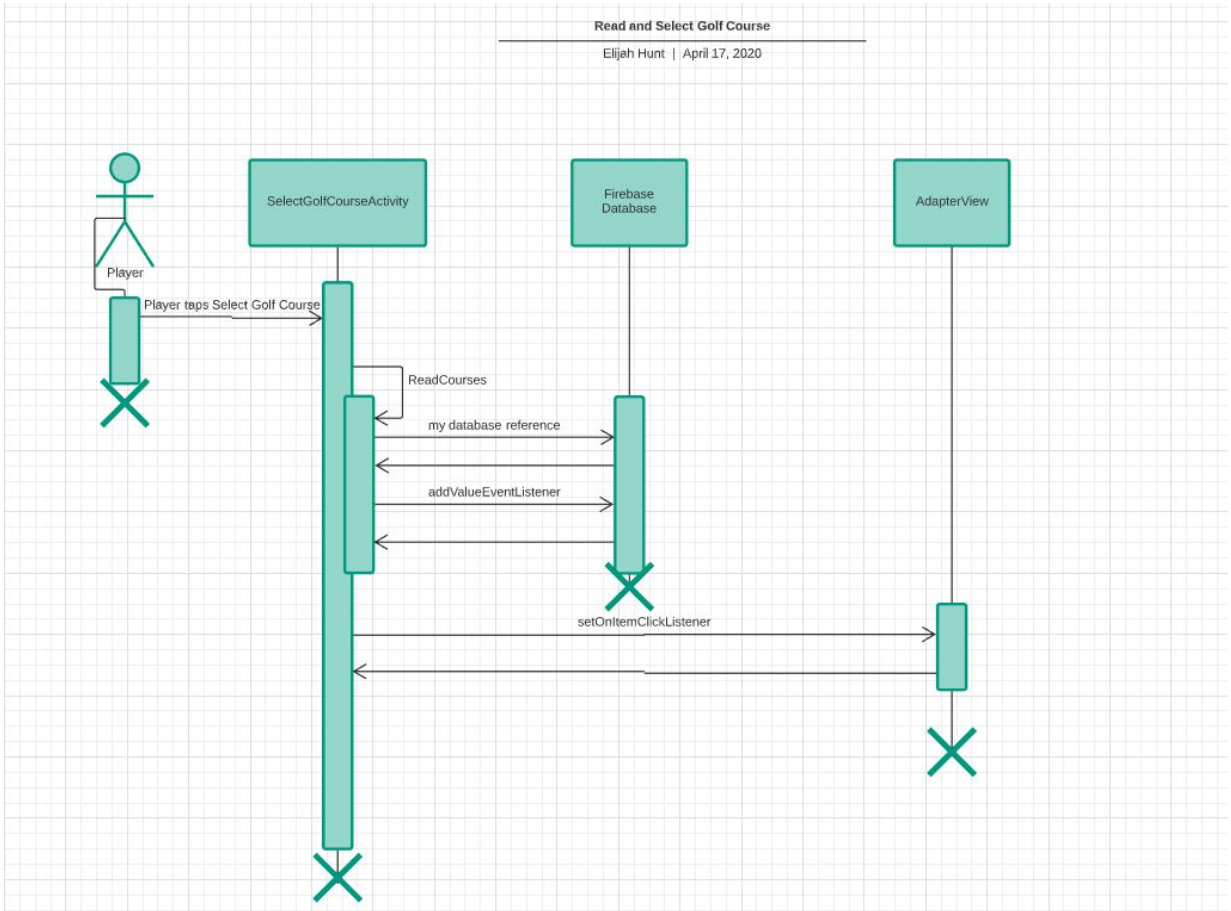

Golf Course/Admin Registration
Admin
Web App
Firebase SDK
Firebase Database
enters registration info
pass registration info
store registration info

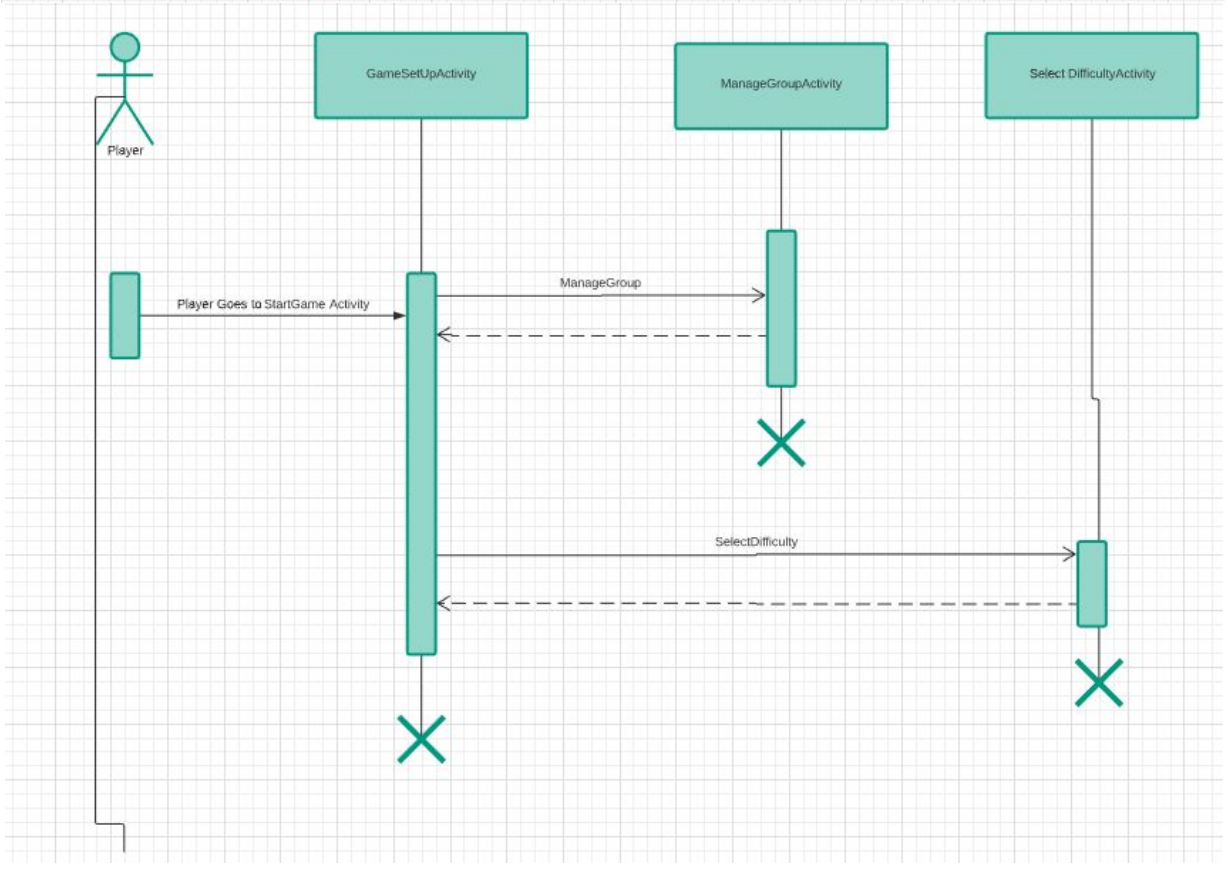

Admin Data View Sequence Diagram
Daniel Teel | April 14, 2020
Web App
Firebase SDK
Firebase Realtime Database
Admin
Views tabs on the dashboard
Loop
Admin is signed in and on the dashboard
Requests data to populate each field
Data is placed in the correct areas
Requests the data from the nodes
Data is returned

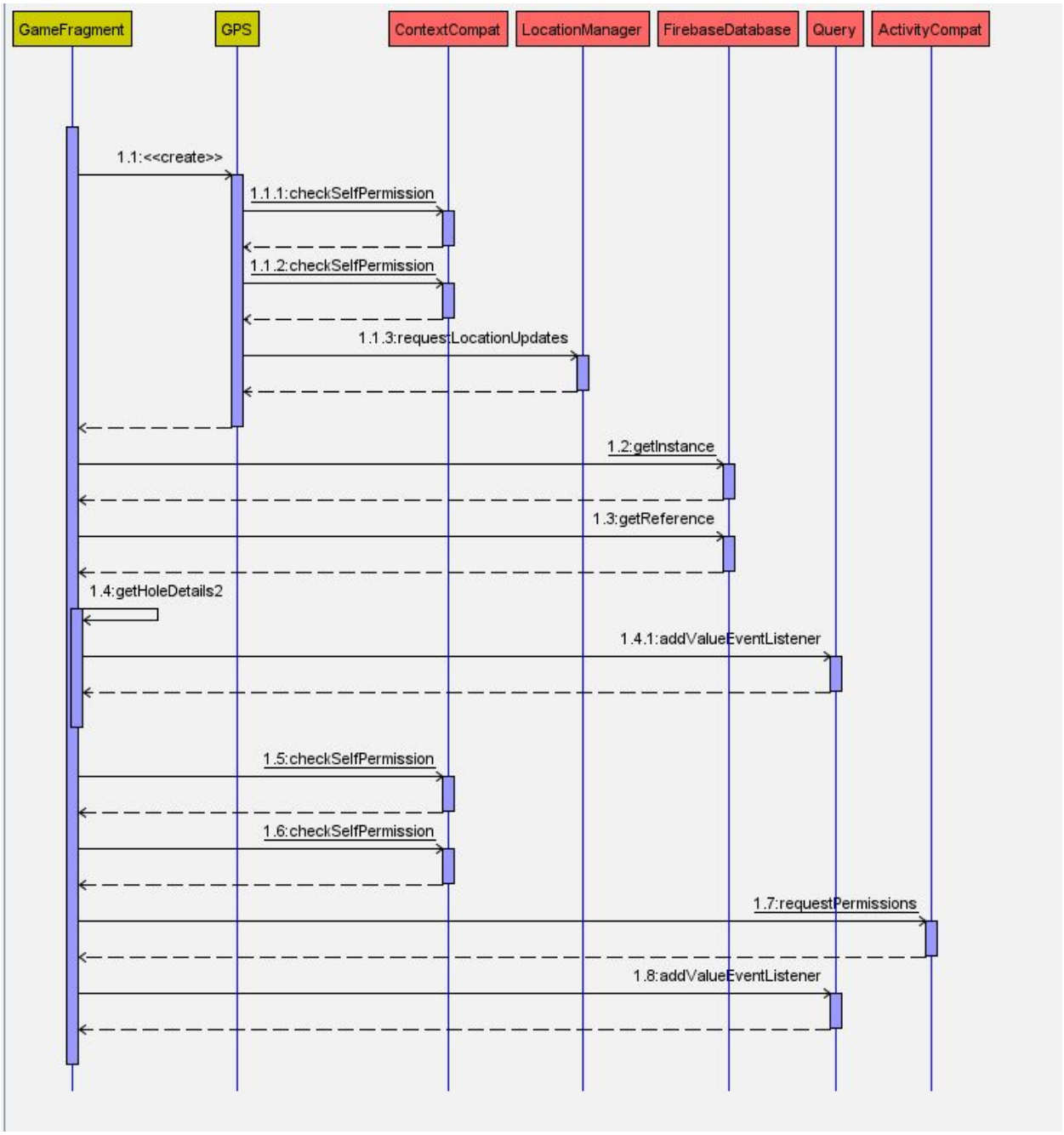

Admin Data Edit Sequence Diagram
Daniel Teel | April 14, 2020
Web App
Firebase SDK
Firebase Realtime Database
Admin
Text is entered in the tiles while they are in edit mode
Loop
Admin is signed in and on Course Overview tab
Data is passed into an update request
Updated information replaces old
Specified nodes values are updated
Updated data is passed back to view

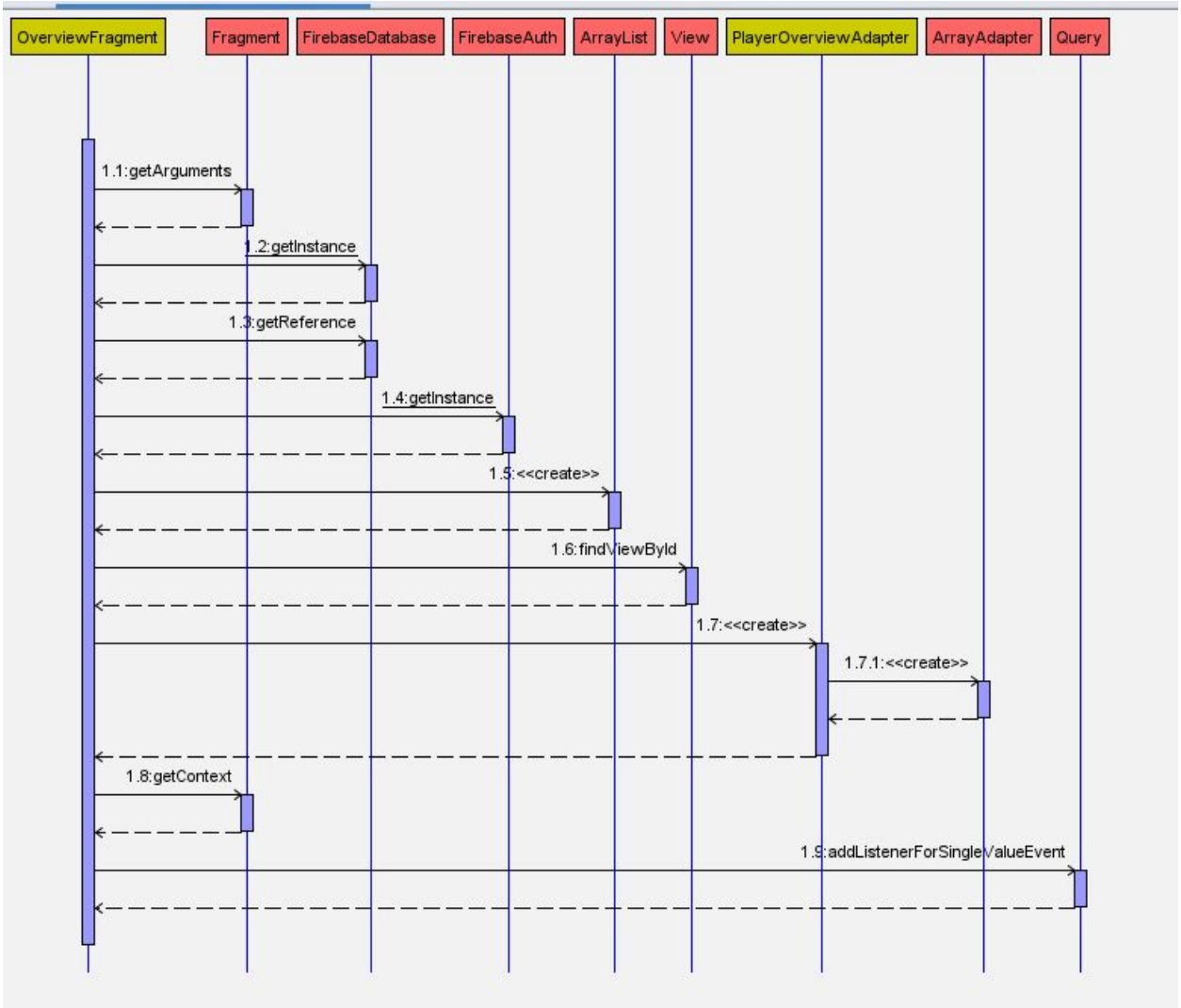

Course Wait Time Sequence Diagram
Daniel Teel | April 14, 2020
Web App
Firebase SDK
Firebase Realtime Database
Admin
Logs in to the dashboard
Loop
There are current games in progress
Hole pars passed to algorithm
Wait time calculated
Number of games per hole counted
Hole Pars passed
Wait Time table updated with current times
Wait Times returned
Wait times are plotted in line chart

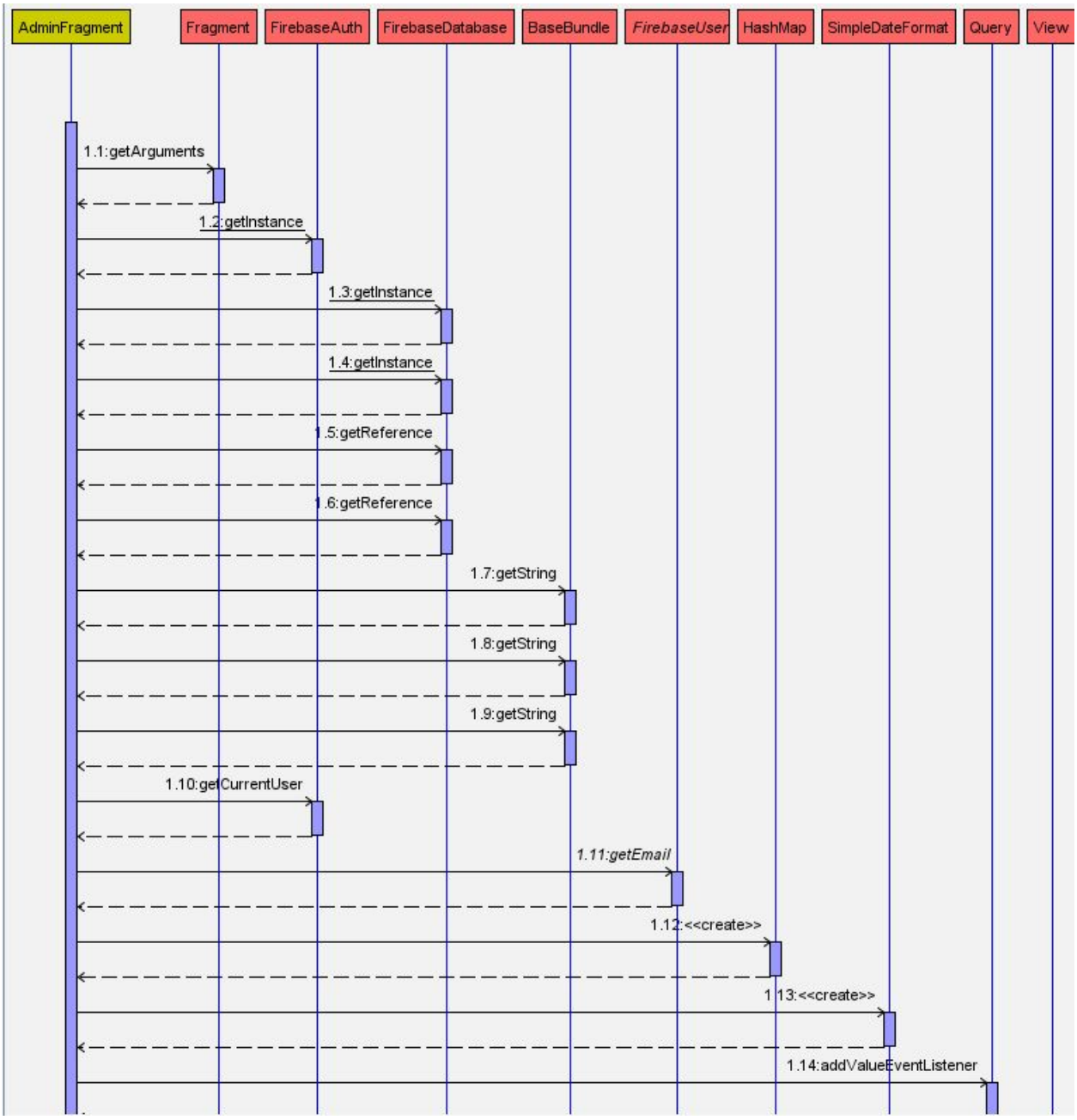

SignUp Sequence
Elijah Hunt | April 17, 2020
Player
SignupActivity
AppCompatActivity
FirebaseAuth
FirebaseDatabase
View
Player Goes to Signup Activity
onCreate
setContentView
UserTypes Credentials
Get Instance
getCurrentUser
SetOnClickListener
CreateUserWithEmailAndPassword

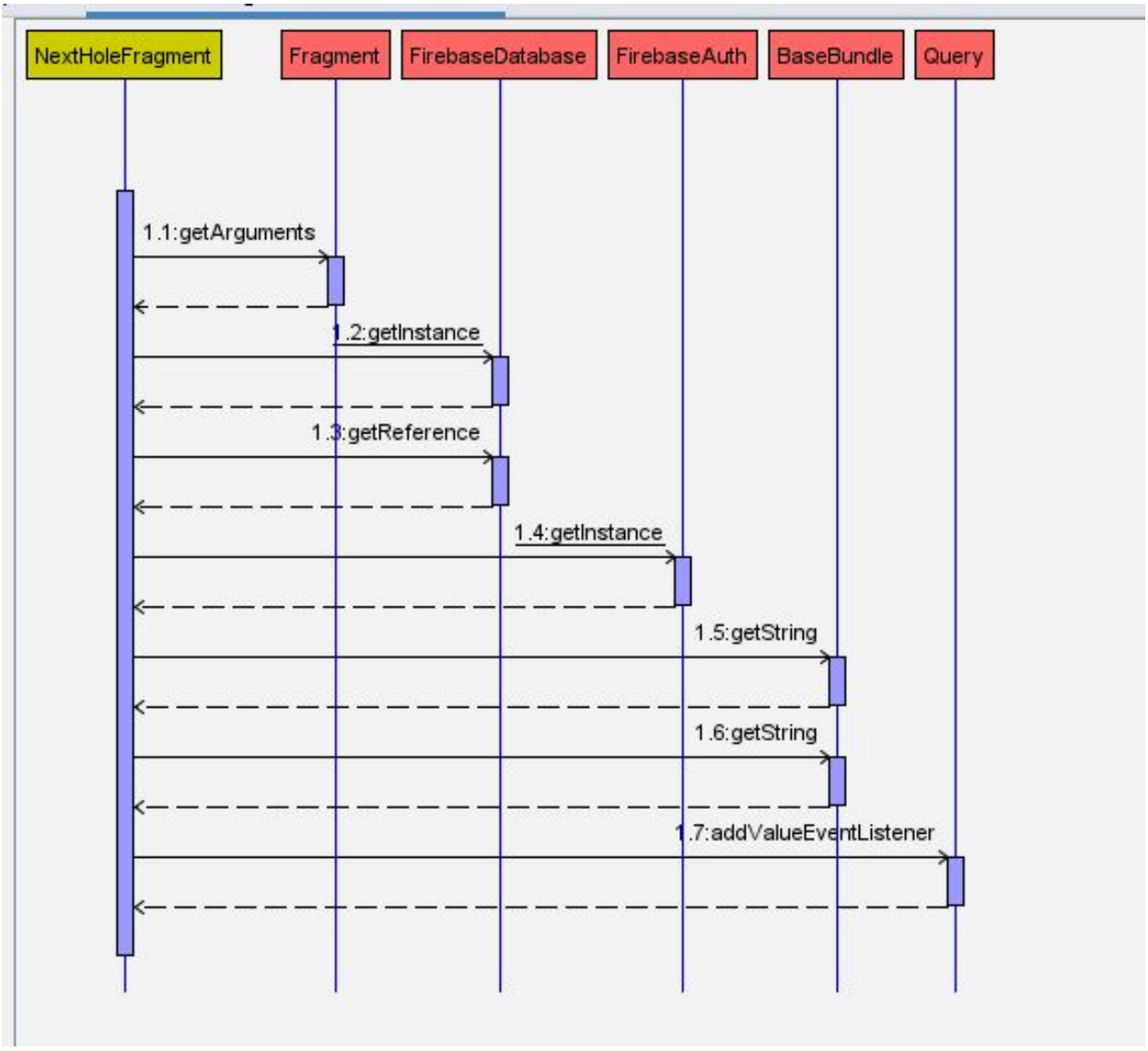

Read and Select Golf Course
Elijah Hunt | April 17, 2020
Player
SelectGolfCourseActivity
Firebase Database
AdapterView
Player taps Select Golf Course
ReadCourses
my database reference
addValueEventListener
setOnItemClickListener


Player
GameSetUpActivity
ManageGroupActivity
Select DifficultyActivity
Player Goes to StartGame Activity
ManageGroup
SelectDifficulty


GameFragment
GPS
ContextCompat
LocationManager
FirebaseDatabase
Query
ActivityCompat
1.1:<<create>>
1.1.1:checkSelfPermission
1.1.2:checkSelfPermission
1.1.3:requestLocationUpdates
1.2:getInstance
1.3:getReference
1.4:getHoleDetails2
1.4.1:addValueEventListener
1.5:checkSelfPermission
1.6:checkSelfPermission
1.7:requestPermissions
1.8:addValueEventListener


OverviewFragment
Fragment
FirebaseDatabase
FirebaseAuth
ArrayList
View
PlayerOverviewAdapter
ArrayAdapter
Query
1.1:getArguments
1.2:getInstance
1.3:getReference
1.4:getInstance
1.5:<<create>>
1.6:findViewById
1.7:<<create>>
1.7.1:<<create>>
1.8:getContext
1.9:addListenerForSingleValueEvent


AdminFragment
Fragment
FirebaseAuth
FirebaseDatabase
BaseBundle
FirebaseUser
HashMap
SimpleDateFormat
Query
View
1.1:getArguments
1.2:getInstance
1.3:getInstance
1.4:getInstance
1.5:getReference
1.6:getReference
1.7:getString
1.8:getString
1.9:getString
1.10:getCurrentUser
1.11:getEmail
1.12:<<create>>
1.13:<<create>>
1.14:addValueEventListener




## 7.2. Interface Specification

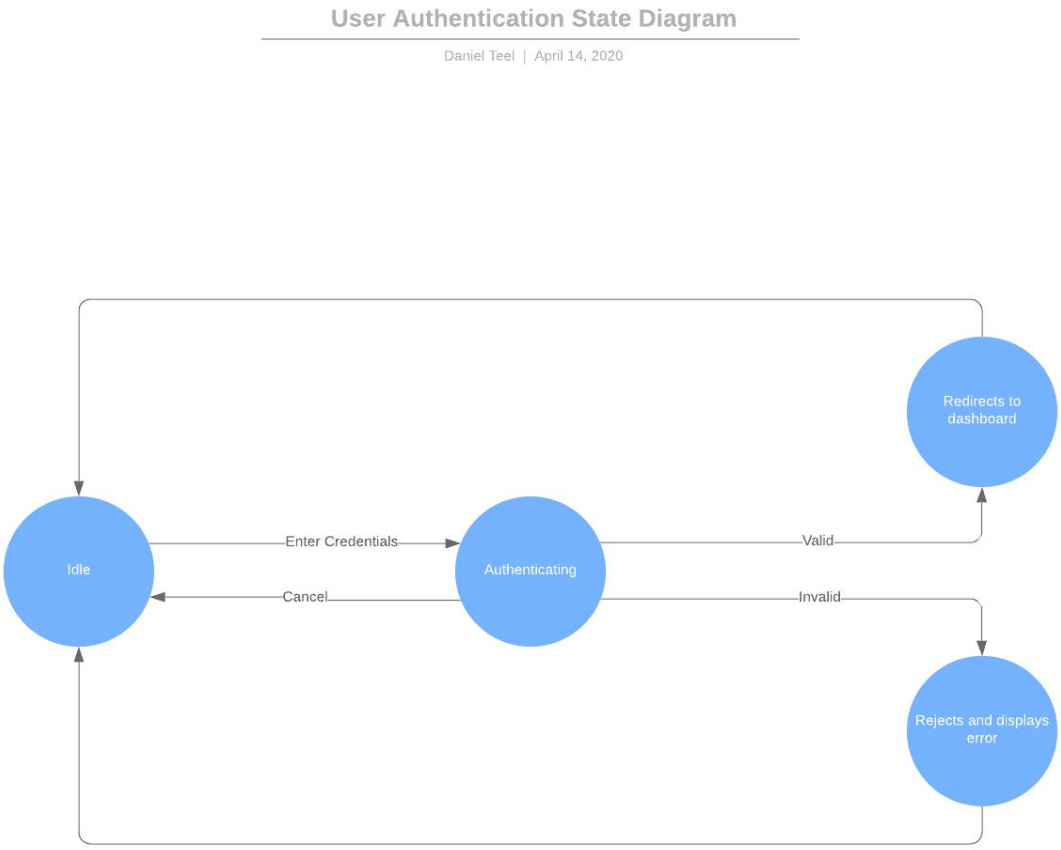
Within the applications, both of them, an interface must be able to be clearly identified as a general use interface or a specific/singular use specification. If it is a specific/singular use interface then the name should contain the component or case that it is intended for.

While specific/singular use interfaces are alright we should strive to generalize as much as possible so that code can be recycled without too much effort. Parameters should be named in a descriptive way so that it is clear to read what the interface is trying to achieve. Interfaces with specific endpoints that are not entirely unique (ex. Hole nodes) should have a parameter to broaden the use cases of that interface.

Creating interfaces increases efficiency and efficiency of the application so as a rule of thumb they should be created if the function(s) are used multiple times across multiple files.

Another interface that should be standardized is arrays for collections of data. Since Firebase is being utilized, the easiest way to collect data and send that collection to the JSON structure of Firebase's Realtime Database is via arrays. Firebase is able to convert these arrays easily into the children of a node.

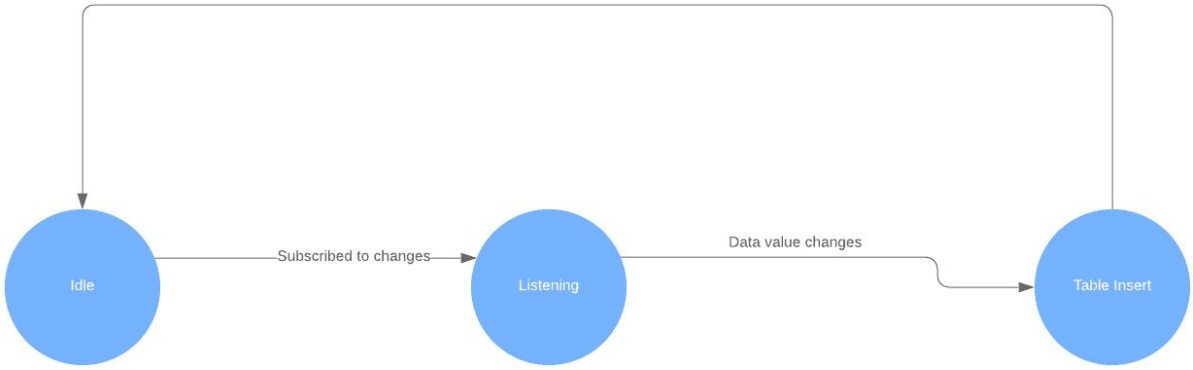
# 7.3. State Diagrams



# Administrator Website State Diagrams

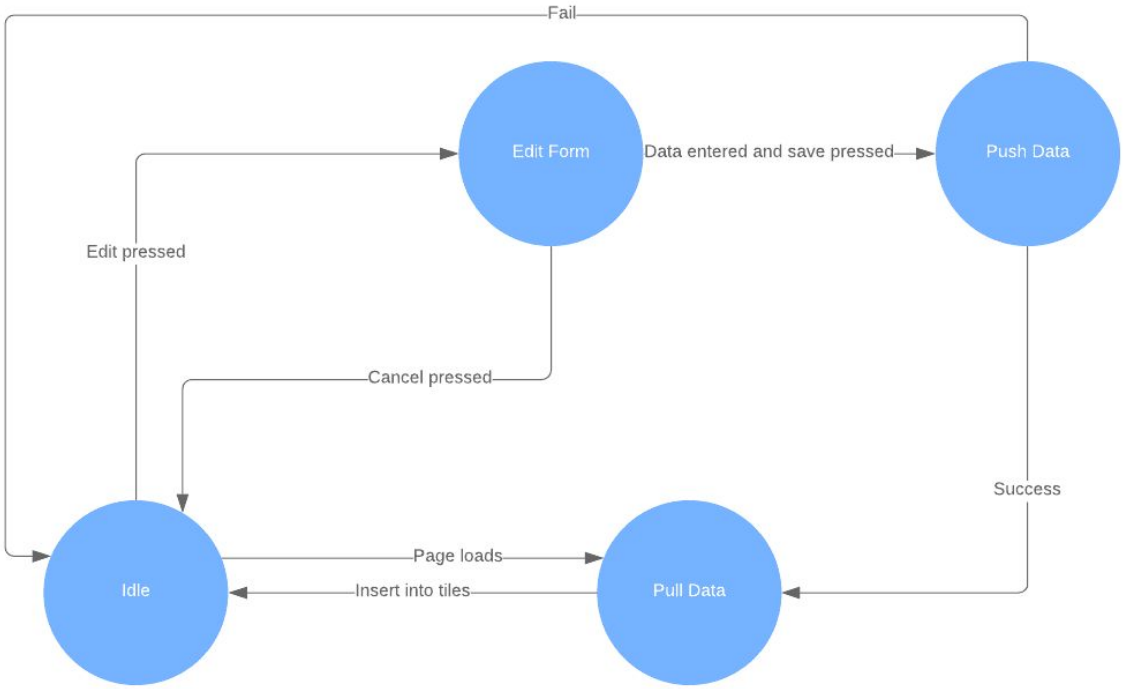
## Admin Data Table State Diagram

Daniel Teel | April 14, 2020

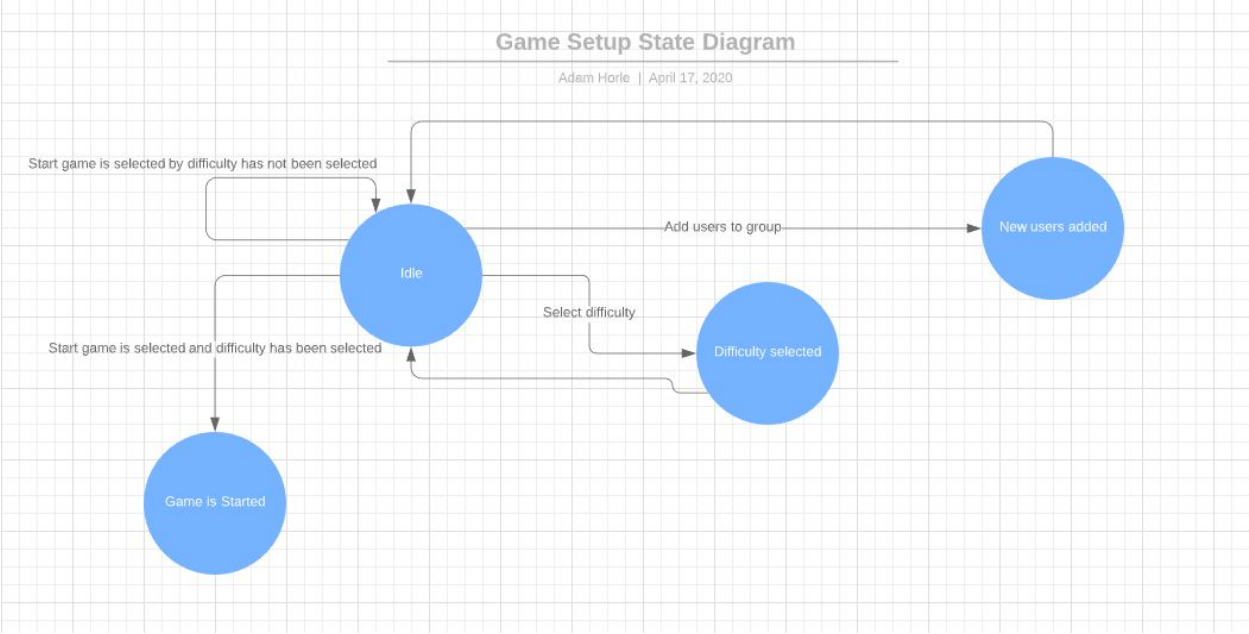


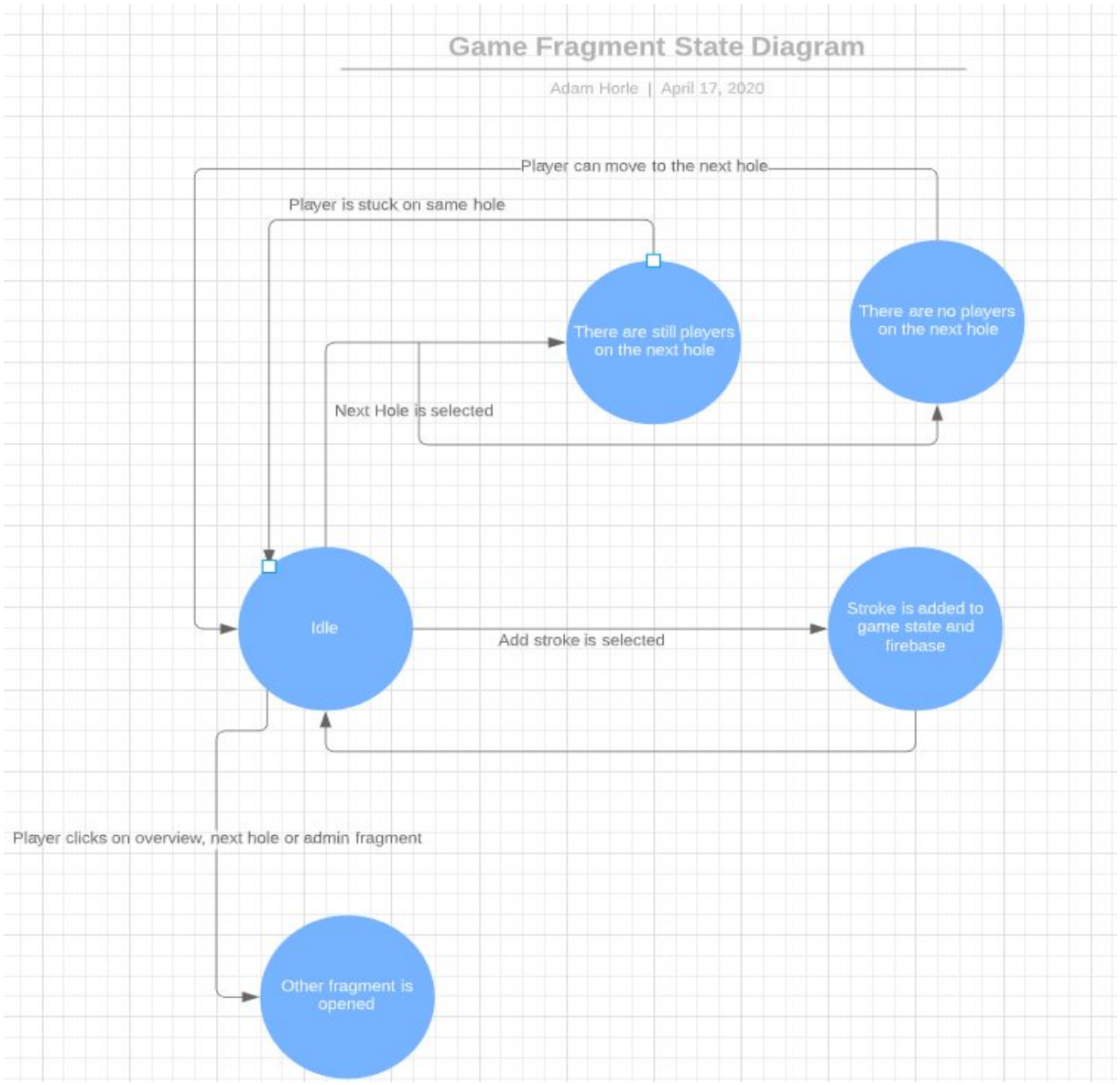

## Course Overview State Diagram

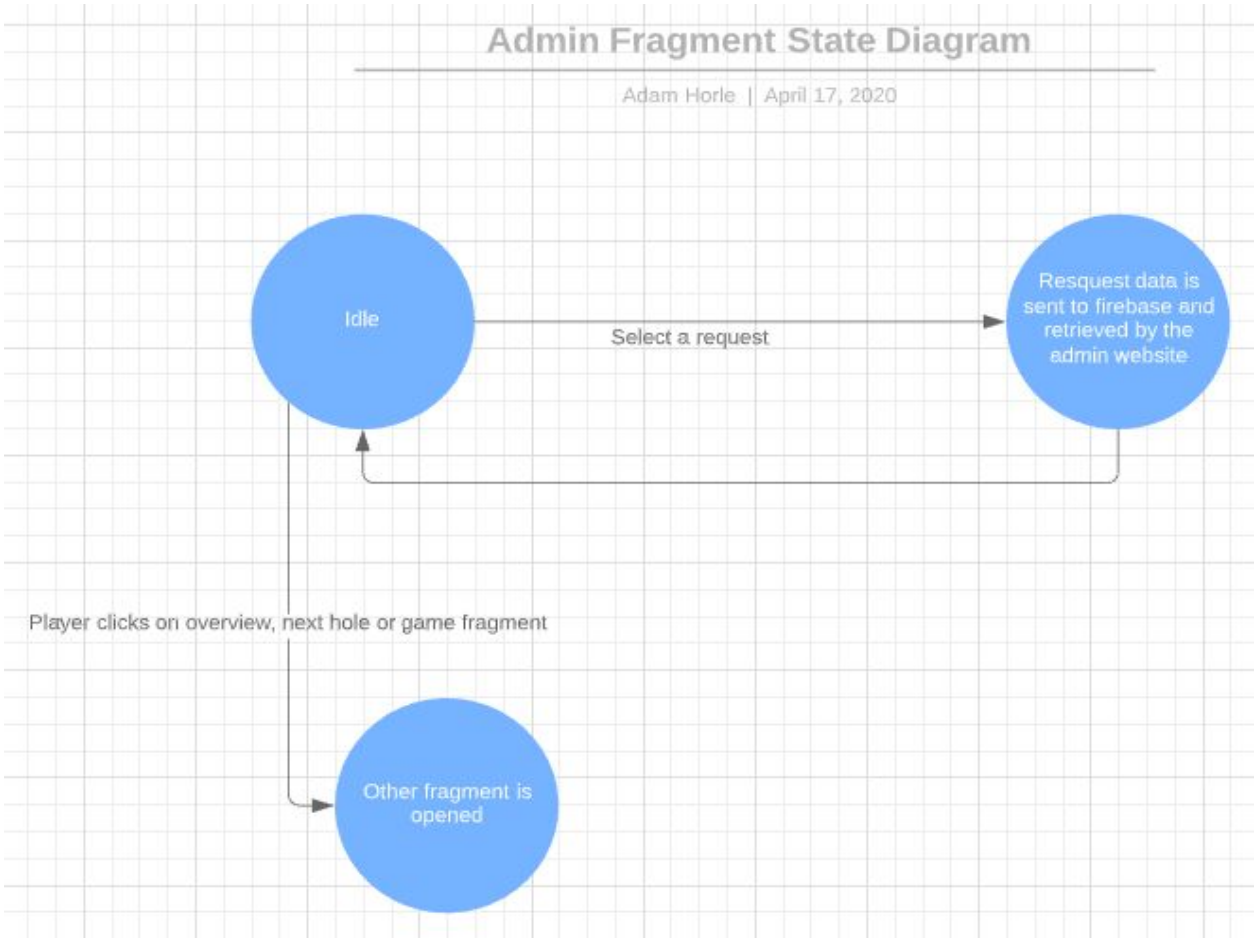
Daniel Teel | April 14, 2020



# Mobile App State Diagrams




Game Fragment State Diagram
Adam Horle | April 17, 2020
Player can move to the next hole
Player is stuck on same hole
There are still players on the next hole
There are no players on the next hole
Next Hole is selected
Idle
Add stroke is selected
Stroke is added to game state and firebase
Player clicks on overview, next hole or admin fragment
Other fragment is opened




# 8. System Architecture and System Design

## 8.1. Subsystems / Component / Design Pattern Identification

Golf Player Time Management has a data read and write design pattern for the Administrator website and a data view pattern for the Mobile application. To accomplish this there are a variety of patterns that were put into play for this project.

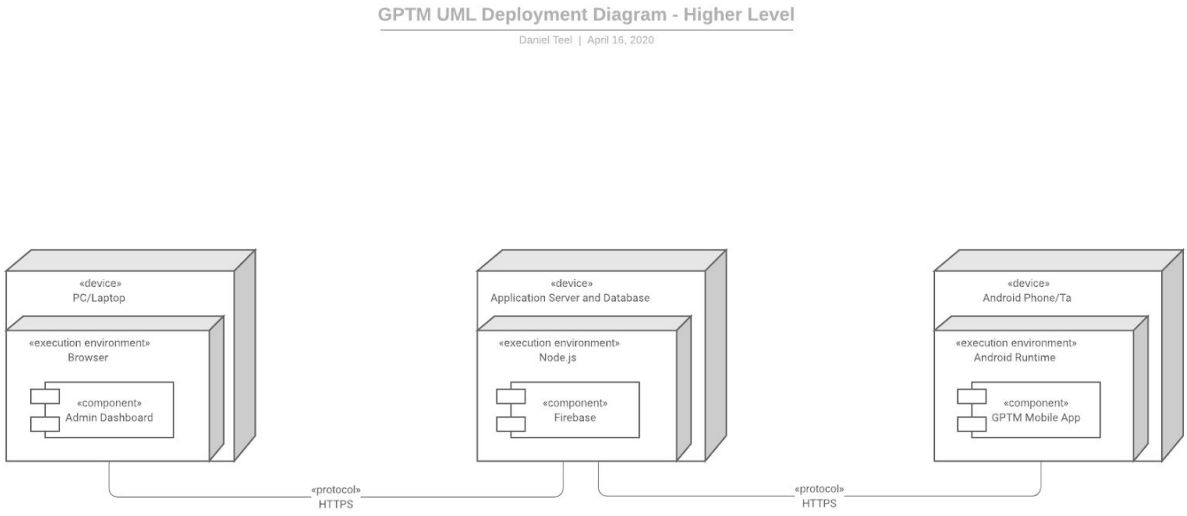
The Administrator Website utilizes the patterns provided by the Angular framework. The Angular framework breaks things down into modules, components, and services. For GPTM there is only one module needed since it is a single page dashboard application. It uses a multitude of components, however. The Hole components are all modeled off of each other and operate independently to try to reduce the size of requests to the Firebase Realtime Database. Each table has its own component as well. By having many components we can have requests tailored to pull and update data that is relevant exclusively to that hole rather than pulling the whole course each time. In order to manage users the application utilizes an Auth Service that

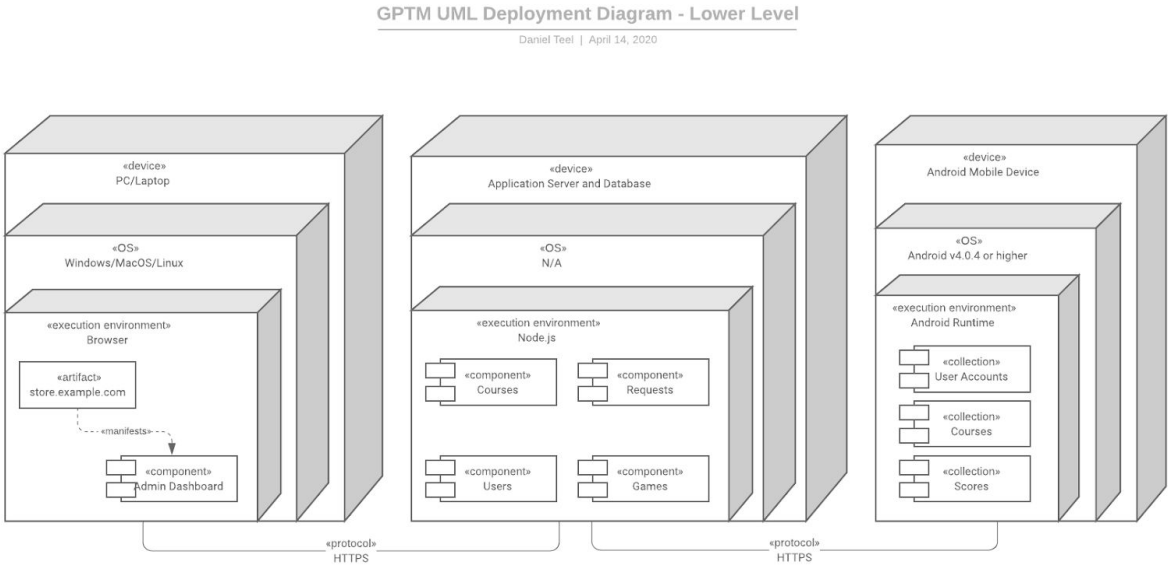
handles user account login, updates, registration, and sign outs. This service is accessible app-wide. Another service that is commonly used in the app is the User Service that is used to get account data for the already authenticated user. Lastly, the app module is used to import any modules that are intended to be used across the app.

The Mobile Application uses the standard activity design. Each activity is a new view and each view accomplishes a different function within the app. Links/buttons transport users to different views and each one (intent) ends up creating an eventual circular pattern that leads back to the MainActivity/PresentationActivity also known as the landing page.

In terms of system architecture, the Administrator Website utilizes a layered architecture pattern and the Mobile Application takes advantage of an event-bus pattern. The layered architecture pattern is used for the Administrator Website because the 4 layers (user interface, service, domain, and persistence) align with the end goal that the application was meant to meet (managing a golf course). This architecture pattern is very common for web applications. The event-bus pattern was chosen for the Mobile Application because it is the standard architecture pattern used for Android development. It has 4 components (event source, event listener, channel, and event bus) which are closely followed in the Activity and Intent design of an Android Application.

## 8.2. Mapping Subsystems to Hardware (Deployment Diagram)




## 8.3. Persistent Data Storage

Persistent data storage for Golf Player Time Management is provided by Firebase. Firebase is a service built, maintained, and managed by Google. Firebase has different services for data storage but for the scope of our app we are only using Firebase's Realtime Database.

The Realtime Database is a database built using JSON trees. The data is stored on a server managed by Google so we know that our data is secure and will not be

erased if one of our machines fails. The data will persist and be preserved in its current structure until the service is either terminated or our project is manually deleted by one of our team members.

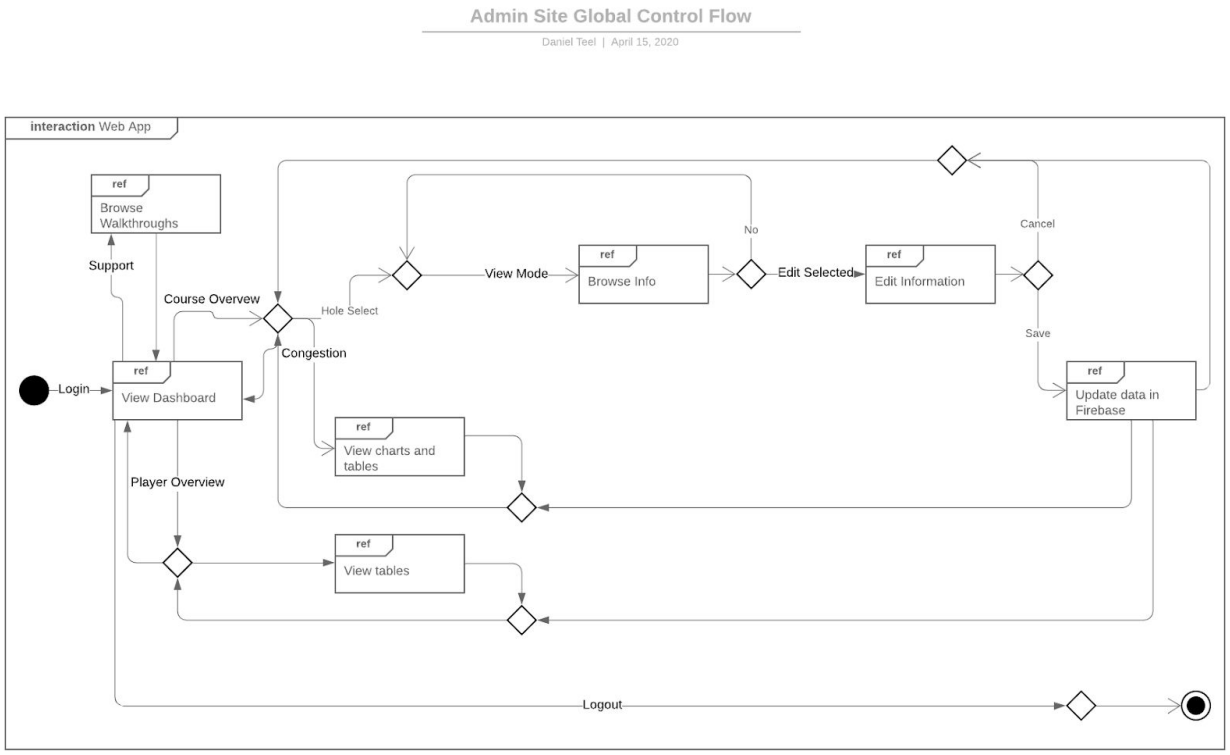
Firebase provides the convenience of not having to use a backend and a physical server that we have to manage and maintain ourselves. Google employees instead manage our data in a separate location according to Google's industry standards.

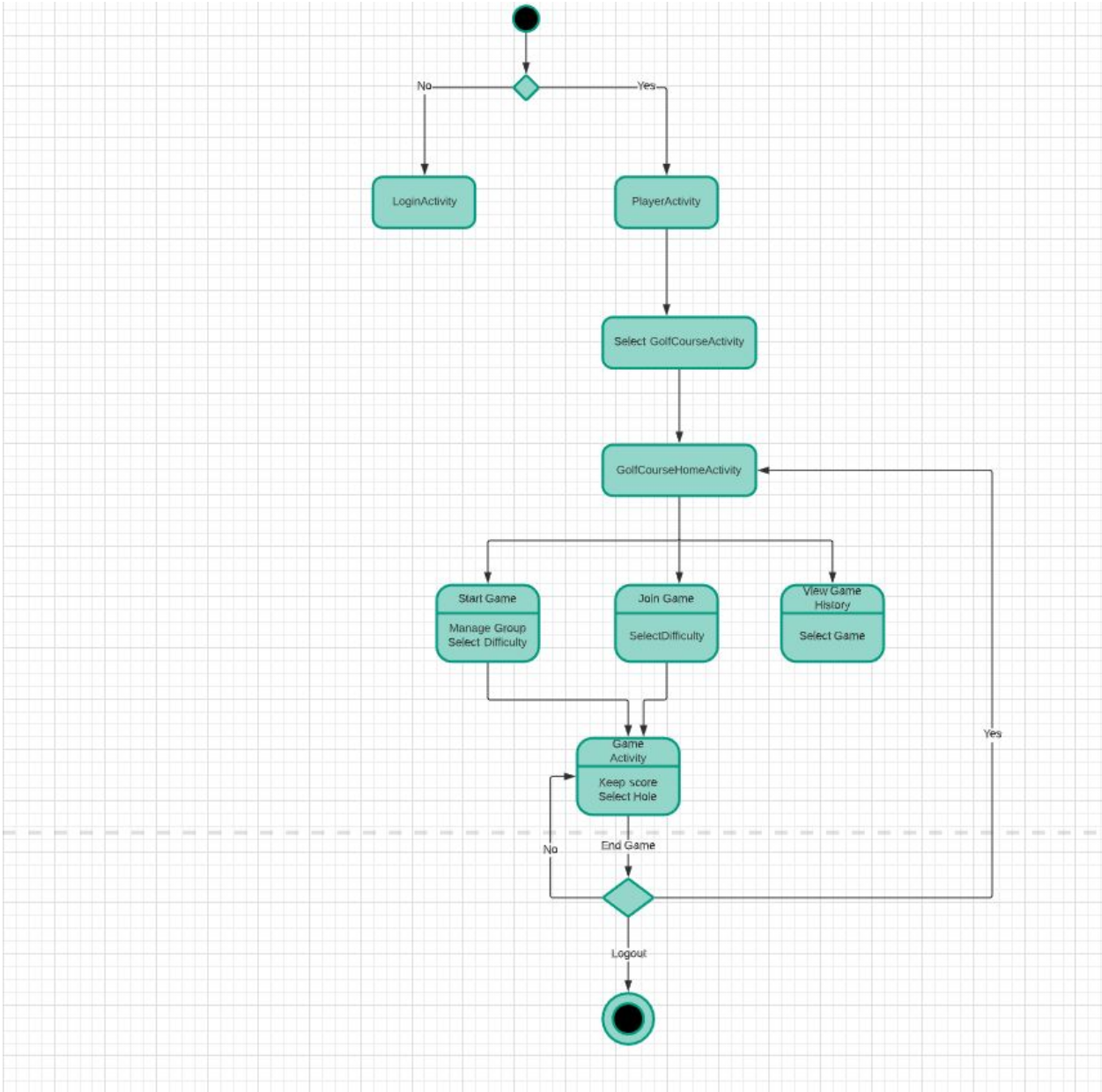
## 8.4. Network Protocol

Golf Player Time Management utilizes standard internet protocols. In other words, it uses HTTP as well as TCP/IP network protocols to send and receive information across the world wide web.

At the internet layer IP (Internet Protocol) is used for delivering packets from the source host to the destination host by looking at the IP addresses in the packet headers. At the host-to-host later TCP (Transmission Control Protocol) is used for reliable communication between end systems by performing sequencing and segmentation of data. At the application layer HTTP (Hypertext transfer protocol) is used to manage communications between web browsers and servers and is the standard of the world wide web.

# 8.5. Global Control Flow






## 8.6. Hardware Requirement

Our apps require minimal hardware, our apps are accessible on platforms that our target audience is expected to own. Both admins and players need a device that has access to the internet. Admins will need a current web browser, such as Google Chrome or Microsoft Edge. Players will need an Android device that is running version 4.0.3 Ice Cream Sandwich or above and able to download and run apps from the Google Play Store.

# 9. Algorithms and Data Structures

## 9.1. Algorithms

Administrator Website Algorithms (TypeScript)

Getting course details associated with users:

```
     getCourseName() {
        const userId = firebase.auth().currentUser.uid;
        return firebase.database().ref('/Users/' +
userId).once('value').then(function(snapshot) {
          const golfCourse = (snapshot.val() &&
snapshot.val().golfCourse || 'No Associated Course');
          return golfCourse;
        });
      }
     getCourseDetails() {
        // use the golfCourse value to match it to the
GolfCourse table and get the hole info
        return firebase.database().ref('/GolfCourse/' +
this.courseName +
'/Holes/Hole1').once('value').then(function(snapshot) {
          // All the data is being pulled here. Assign it a
value then it can be shown in the front end.
          const data = snapshot.val();
          return data;
        });
      }
```

Standard boilerplate algorithm for saving new data:

```
    saveGeneral() {
       const courseRef = firebase.database().ref('/GolfCourse/'
+ this.courseName + '/Holes/Hole1');
       courseRef.update({
         Description: this.generalDescription,
         Tips: this.generalTips
       });
       this.initData();
       this.isEdit1 = false;
     }
```

Calculating wait times for holes:

```
    countHoleQueue(courseName, holeNum) {
       let i = 1;
       firebase.database().ref('GolfCourse/' + courseName +
'/Holes/Hole' + holeNum +
'/Blue_Square').once('value').then(function(snapshot) {
         holePar = snapshot.val().Par;
         const holeref = firebase.database().ref('Games/' +
courseName);
         // Update on new games

holeref.orderByChild('Location').equalTo(holeNum).on('child_adde
d', function() {
           holeQueue = i;
           i++;
           const queueRef =
firebase.database().ref('GolfCourse/' + courseName +
'/WaitTimes/Hole' + holeNum);
           queueRef.update({
```

```
              Queue: holeQueue,
              WaitTime: (holePar * 3.2 * holeQueue).toFixed(1)
            });
          });
          // Update on changes to in progress games

holeref.orderByChild('Location').equalTo(holeNum).on('child_chan
ged', function() {
            holeQueue = i;
            i++;
            const queueRef =
firebase.database().ref('GolfCourse/' + courseName +
'/WaitTimes/Hole' + holeNum);
            queueRef.update({
              Queue: holeQueue,
              WaitTime: (holePar * 3.2 * holeQueue).toFixed(1)
            });
          });
          // Update on deletion

holeref.orderByChild('Location').equalTo(holeNum).on('child_remo
ved', function() {
            holeQueue = i;
            i++;
            const queueRef =
firebase.database().ref('GolfCourse/' + courseName +
'/WaitTimes/Hole' + holeNum);
            queueRef.update({
              Queue: holeQueue,
```

```
              WaitTime: (holePar * 3.2 * holeQueue).toFixed(1)
            });
          });
        });
      }
```

Using the above function in tandem with a datasource:

```
// This is what is fed into the table, listen for new games
        this.db.list('Games/' +
this.courseName).valueChanges().subscribe(res => {
          this.playerData = res;
          this.playerDataSource.data = this.playerData;

          // Clear previous calculations
          this.clearQueue(this.courseName, '1');
          for (let i = 2; i <= 9; i++) {
            this.clearQueue(this.courseName, i);
          }
          // Estimate wait times and set the waiting queue
          this.countHoleQueue(this.courseName, '1');
          for (let i = 2; i <= 9; i++) {
            this.countHoleQueue(this.courseName, i);
          }
        });
        // Recalculate wait times if there are updates to the
hole info (par value may change)
        this.db.list('GolfCourse/' + this.courseName +
'/Holes').valueChanges().subscribe(() => {
          // Clear previous calculations
          this.clearQueue(this.courseName, '1');
```

```
            for (let i = 2; i <= 9; i++) {
              this.clearQueue(this.courseName, i);
            }
            // Estimate wait times and set the waiting queue
            this.countHoleQueue(this.courseName, '1');
            for (let i = 2; i <= 9; i++) {
              this.countHoleQueue(this.courseName, i);
            }
          });
```

Boilerplate function to read data and listen for changes for a data table:

```
this.db.list('Request/' +
this.courseName).valueChanges().subscribe(res => {
       this.requestsData = res;
       this.requestsDataSource.data = this.requestsData;
     });
```

Boilerplate function to edit an entry in a table (modified in other areas to set status and also delete nodes):

```
removeRequest(i: number) {
   const ref = firebase.database().ref('Request/' +
this.courseName);
   ref.once('value').then(function(snapshot) {
     // Get the key and then remove the data belonging to that
key
     const key = Object.keys(snapshot.val())[i];
     ref.child(key).remove();
   });
 }
acknowledgeRequest(i: number) {
```

```
   const ref = firebase.database().ref('Request/' +
this.courseName);
   ref.once('value').then(function(snapshot) {
     // Get the key and then update the data belonging to that
key
     const key = Object.keys(snapshot.val())[i];
     ref.child(key).update({
       Status: 'In Progress'
     });
   });
 }
completeRequest(i: number) {
   const ref = firebase.database().ref('Request/' +
this.courseName);
   ref.once('value').then(function(snapshot) {
     // Get the key and then update the data belonging to that
key
     const key = Object.keys(snapshot.val())[i];
     ref.child(key).update({
       Status: 'Complete'
     });
   });
 }
```

Functions for Firebase authentication:

```
 // Standard Firebase auth registration
 doRegister(value) {
   return new Promise<any>((resolve, reject) => {
     firebase.auth().createUserWithEmailAndPassword(value.email,
value.password)
```

```
      .then(res => {
        resolve(res);
        res.user.sendEmailVerification();
      }, err => reject(err));
    });
  }

  // Standard Firebase auth login
  doLogin(value) {
    return new Promise<any>((resolve, reject) => {
      firebase.auth().signInWithEmailAndPassword(value.email,
value.password)
      .then(res => {
        resolve(res);
      }, err => reject(err));
    });
  }

  // Standard Firebase auth logout
  doLogout() {
    return new Promise ((resolve, reject) => {
      if (firebase.auth().currentUser) {
        this.afAuth.auth.signOut();
        resolve();
      }
      if (!firebase.auth().currentUser) {
        reject();
      }
    });
```

```
 }
```

Changing an email address or password with confirmation:

```
 // Asks the user to make sure that they want to change their
email and checks to make sure that it is actually valid
 confirmChange(confirm) {
   this.confirm = confirm;
   if (this.emailForm.valid) {
     this.confirm = confirm;
   }
   if (!this.email || !this.emailForm.valid) {
     this.confirm = false;
   }
 }

   // Asks the user to make sure that they want to change their
password and checks to make sure that it is actually valid
   confirmChangePassword(confirm) {
     this.confirmP = confirm;
     if (this.resetForm.valid) {
       this.confirmP = confirm;
     }
     if (!this.password || !this.password2 ||
!this.resetForm.valid) {
       this.confirmP = false;
     }
   }

 /* tslint:disable:quotemark */
 // Change email address
```

```
 changeEmail() {
   this.emailChanged = false;
   this.isLoading = true;
   const user = firebase.auth().currentUser;
   if (this.email === this.email2) {
     user.updateEmail(this.email).then(() => {
       // Update successful
       const userId = firebase.auth().currentUser.uid;
       const userRef = firebase.database().ref('/Users/' +
userId);
       userRef.update({
         email: this.email,
         verified: false
       });
       user.sendEmailVerification();
       this.noMatchEmail = null;
       this.confirm = false;
       this.errorMessage = null;
       this.emailChanged = true;
       this.isLoading = false;
     }, err => {
       // An error happened
       this.noMatchEmail = null;
       this.confirm = false;
       this.emailChanged = false;
       this.errorMessage = err.message;
       this.isLoading = false;
     });
   }
```

```
  if (this.email !== this.email2) {
    this.confirm = false;
    this.noMatchEmail = "Emails don't match";
    this.emailChanged = false;
    this.errorMessage = null;
    this.isLoading = false;
  }
}

// Change password
changePassword() {
  this.passwordChanged = false;
  this.isLoadingP = true;
  const user = firebase.auth().currentUser;
  if (this.password === this.password2) {
    user.updatePassword(this.password).then(() => {
      // Update successful
      this.noMatch = null;
      this.passwordChanged = true;
      this.confirmP = false;
      this.errorMessageP = null;
      this.isLoadingP = false;
    }, err => {
        // An error happened
        this.noMatch = null;
        this.passwordChanged = false;
        this.confirmP = false;
        this.errorMessageP = err.message;
        this.isLoadingP = false;
```

```
    });
  }
  if (this.password !== this.password2) {
    this.confirmP = false;
    this.noMatch = "Passwords don't match";
    this.errorMessageP = null;
    this.isLoadingP = false;
  }
}
/* tslint:enable:quotemark */
```

An example of using the auth.service in other components:

```
logout() {
  this.authService.doLogout()
  .then(() => {
    this.router.navigate(['/login']);
  }, (error) => {
    console.log('Logout error', error);
  });
}
```

Setting form validation in the typeScript:

```
createForm() {
  this.loginForm = this.fb.group({
    email: ['', Validators.email],
    password: ['', Validators.minLength(6)]
  });
}
```

Maintaining security in authentication and creating new golf courses in the database if the login is a first time login:

```
tryLogin(value) {
  this.isLoading = true;
  this.unverified = false;
  this.errorMessage = null;
  this.noAdmin = false;
  this.tryAgain = false;
  this.authService.doLogin(value)
  .then(() => {
    // If email verified
    if (firebase.auth().currentUser.emailVerified) {
      this.getCourseStatus()
      .then(val => {
        this.golfCourse = val;
        this.checkIfExists(this.golfCourse)
        .then(val => {
          this.courseExists = val;
          // If not a first time sign in
          if (this.courseExists !== null) {
            this.getPendingInfo()
            .then(data => {
              if (!data.numberOfHoles) {
                this.evalAdmin()
                .then(val => {
                  this.isAdmin = val;
                  this.checkAdmin(this.isAdmin);
                });
              }
              if (data.numberOfHoles) {
                this.removeUser();
```

```
                this.isLoading = false;
                this.tryAgain = true;
              }
            });
          }
          // If first time sign in init a default golf course
in Firebase
          if (this.courseExists === null) {
            this.getPendingInfo()
            .then(data => {
              this.golfCourse = data.golfCourse;
              this.numberOfHoles = data.numberOfHoles;
              this.latitude = data.lat;
              this.longitude = data.long;
              this.writeHoleData(this.golfCourse,
this.latitude, this.longitude, this.numberOfHoles);
              this.removePending();
              this.evalAdmin()
              .then(val => {
                this.isAdmin = val;
                this.checkAdmin(this.isAdmin);
              });
            });
          }
        });
      });
    }
    // If email is not verified
    if (!firebase.auth().currentUser.emailVerified) {
```

```
        this.unverified = true;
        this.errorMessage = null;
        this.isLoading = false;
      }
    }, err => {
      this.tryAgain = false;
      this.unverified = false;
      this.noAdmin = false;
      this.errorMessage = 'Invalid username/password';
      this.errorMessage = err.message;
      this.isLoading = false;
    });
  }

  // Pull in admin data
  evalAdmin() {
    const userId = firebase.auth().currentUser.uid;
    return firebase.database().ref('/Users/' +
userId).once('value').then(function(snapshot) {
      const isAdmin = (snapshot.val() && snapshot.val().isAdmin)
|| false;
      return isAdmin;
      });
  }

  // Determine if the user is an actual admin
  checkAdmin(isAdmin) {
    if (this.isAdmin === true) {
      this.noAdmin = false;
```

```
      // Set to verified if this is a case of an email change
      const userId = firebase.auth().currentUser.uid;
      const userRef = firebase.database().ref('/Users/' +
userId);
      userRef.update({
        verified: true
      });
      this.router.navigate(['/dashboard']);
      this.isLoading = false;
    }
    if (this.isAdmin === false) {
      this.noAdmin = true;
      this.isLoading = false;
    }
  }

  // Grab course for user
  getCourseStatus() {
    const userId = firebase.auth().currentUser.uid;
    return firebase.database().ref('/Users/' +
userId).once('value').then(function(snapshot) {
      const golfCourse = snapshot.val().golfCourse;
      return golfCourse;
    });
  }

  // See if the course is in use already
  checkIfExists(courseName) {
```

```
    return firebase.database().ref('GolfCourse/' +
courseName).once('value').then(function(snapshot) {
      const courseName = snapshot.val();
      return courseName;
    });
  }

  // Get the necessary data to make a new course
  getPendingInfo() {
    const userId = firebase.auth().currentUser.uid;
    return firebase.database().ref('/Users/' +
userId).once('value').then(function(snapshot) {
      const data = snapshot.val();
      return data;
    });
  }

    /* tslint:disable:quotemark */
    /* tslint:disable:object-literal-shorthand */
  // Initialize the GolfCourse Firebase structure
  writeHoleData(courseName, latitude, longitude, selectedNumber)
{
    firebase.database().ref('GolfCourse/' + courseName).set({
      golfCourse: courseName,
      latitude: latitude,
      longitude: longitude
    });
    for (this.i = 1; this.i <= selectedNumber; this.i++) {
```

```
    firebase.database().ref('GolfCourse/' + courseName +
'/Holes' + '/Hole' + this.i).set({
      Description: 'No description set',
      Tips: 'No tips set',
    });
    firebase.database().ref('GolfCourse/' + courseName +
'/Holes' + '/Hole' + this.i + '/Red_Circle').set({
      Description: "Men's professional tee",
      Tips: 'No tips set',
      Yards: 0,
      Par: 0,
    });
    firebase.database().ref('GolfCourse/' + courseName +
'/Holes' + '/Hole' + this.i + '/Blue_Square').set({
      Description: "Men's average tee",
      Tips: 'No tips set',
      Yards: 0,
      Par: 0,
    });
    firebase.database().ref('GolfCourse/' + courseName +
'/Holes' + '/Hole' + this.i + '/Yellow_Triangle').set({
      Description: "Women's professional tee",
      Tips: 'No tips set',
      Yards: 0,
      Par: 0,
    });
    firebase.database().ref('GolfCourse/' + courseName +
'/Holes' + '/Hole' + this.i + '/Pink_Diamond').set({
      Description: "Women's average tee",
```

```
      Tips: 'No tips set',
      Yards: 0,
      Par: 0,
    });
    firebase.database().ref('GolfCourse/' + courseName +
'/WaitTimes/Hole' + this.i).set({
      Queue: 0,
      WaitTime: 0
    });
  }
}
/* tslint:enable:quotemark */
/* tslint:enable:object-literal-shorthand */

// Get rid of the unnecessary nodes
removePending() {
  const userId = firebase.auth().currentUser.uid;
  const userRef = firebase.database().ref('/Users/' + userId);
  userRef.child('numberOfHoles').remove();
  userRef.child('lat').remove();
  userRef.child('long').remove();
  userRef.update({
    verified: true
  });
}

removeUser() {
  const user = firebase.auth().currentUser;
  const userId = firebase.auth().currentUser.uid;
```

```
    firebase.database().ref('/Users/' + userId).remove();
    user.delete();
  }

  // Pop-up register form
  tryRegister(): void {
    this.dialog.open(RegisterComponent, {
      disableClose: true,
      width: '800px',
      height: '550px'
    });
  }

  // Open email submission
  resetPassword(): void {
    this.dialog.open(PasswordResetComponent, {
      disableClose: true,
      width: '400px',
      height: '300px'
    });
  }

  // For golf course search autocomplete
  geolocate() {
    if (navigator.geolocation) {
      navigator.geolocation.getCurrentPosition(function(position)
{
        const geolocation = {
          lat: position.coords.latitude,
```

```
          lng: position.coords.longitude
        };
        const circle = new google.maps.Circle({
          center: geolocation,
          radius: position.coords.accuracy
        });
        this.autocomplete.setBounds(circle.getBounds());
      });
    }
  }
```

Opening a page in a new window (webpage and email using mailto):

```
// Open new window with endpoint set to Google Play Store
appDownload() {
  window.open(
    'https://play.google.com/store/apps', '_blank');
}

// Open new window and use current user's email to open up a
blank draft email to the support account
mailTo() {
  window.open(
    'mailto:glfmngmnt@gmail.com', '_blank');
}
```

Map methods used:

```
onMapReady(map) {
 this.initDrawingManager(map);
}
```

```
initDrawingManager(map: any) {
 const options = {
   drawingControl: true,
   drawingControlOptions: {
   drawingModes: ['polygon']
   },
   polygonOptions: {
     draggable: false,
     editable: false,
     clickable: false,
     fillColor: 'green',
     strokeColor: 'yellow'
   },
   drawingMode: google.maps.drawing.OverlayType.POLYGON
 };

 const drawingManager = new
google.maps.drawing.DrawingManager(options);
 drawingManager.setDrawingMode(null);

 // Place Marker
 let marker;
 google.maps.event.addListener(map, 'click', function(event) {
   placeMarker(event.latLng);
   const lat = marker.getPosition().lat();
   const lng = marker.getPosition().lng();
   const holeLocation = [];
   /* tslint:disable:object-literal-shorthand */
   holeLocation.push({
```

```
      lat: lat,
      lng: lng
    });
    markerData = holeLocation[0];
  });

  // Marker image
  const image =
'https://img.icons8.com/color/48/000000/map-pin.png';
  function placeMarker(location) {
    if (marker == null) {
      marker = new google.maps.Marker({
          position: location,
          map: map,
          animation: google.maps.Animation.DROP,
          icon: image
      });
    }
    /* tslint:enable:object-literal-shorthand */
    if (marker != null) {
      marker.setPosition(location);
    }
  }


  // Draw Polygon
  drawingManager.setMap(map);
  google.maps.event.addListener(drawingManager,
'polygoncomplete', function(polygon) {
```

```
    myPolygon = polygon;
    const path = polygon.getPath();
    const coordinates = [];

    for (let i = 0 ; i < path.length ; i++) {
      coordinates.push({
        lat: path.getAt(i).lat(),
        lng: path.getAt(i).lng()
      });
    }
    coordinateData = coordinates;

    // To disable after 1 shape is drawn
    drawingManager.setDrawingMode(null);

    // To hide controls
    drawingManager.setOptions({
      drawingControl: false,
    });
  });


google.maps.event.addDomListener(document.getElementById('delete
-button'), 'click', function() {
    // Bring back controls to allow the user to try again
    drawingManager.setOptions({
      drawingControl: true,
    });
  });
```

```
}

// Push coordinates to Firebase
savePoly() {
 const geoFence = [];
 geoFence.push({
   Hole: markerData,
   CourseOutline: coordinateData
 });
 this.dialogRef.close(geoFence);
}

close() {
 this.dialogRef.close();
}

// Clear overlay
resetMap() {
 myPolygon.setMap(null);
}
```

## Mobile App Algorithms (Java)

Registration:

```
String Email = email.getText().toString();
String Password1 = password1.getText().toString();
String Password2 = password2.getText().toString();
if(Email.isEmpty()){
   email.setError("Please enter an email");
   Toast.makeText(getApplicationContext(), "REGISTRATION
FAILED", Toast.LENGTH_SHORT).show();
```

```
}
else if (Password1.isEmpty() || Password2.isEmpty()){
   password1.setError("Enter a password");
   Toast.makeText(getApplicationContext(), "REGISTRATION
FAILED", Toast.LENGTH_SHORT).show();
}
else if (!(Password1.equals(Password2))){
   password1.setError("Your passwords do not match");
   Toast.makeText(getApplicationContext(), "REGISTRATION
FAILED", Toast.LENGTH_SHORT).show();
}

else if (!(Email.isEmpty() && Password1.isEmpty() &&
Password2.isEmpty())){
   mAuth.createUserWithEmailAndPassword(Email,
Password1).addOnCompleteListener(new
OnCompleteListener<AuthResult>() {
       @Override
       public void onComplete(@NonNull Task<AuthResult> task) {
           if (task.isSuccessful()) {

               Toast.makeText(getApplicationContext(),
"REGISTRATION SUCCESSFUL", Toast.LENGTH_SHORT).show();

myRef.child(mAuth.getUid()).child("email").setValue(Email);

myRef.child(mAuth.getUid()).child("isAdmin").setValue("false");
               ProfileActivityIntent();
           }
```

```
            else if(task.getException() instanceof
FirebaseAuthUserCollisionException){
                Toast.makeText(getApplicationContext(), "USER
ALREADY EXISTS", Toast.LENGTH_SHORT).show();
            }
            else{
                Toast.makeText(getApplicationContext(),
"REGISTRATION FAILED", Toast.LENGTH_SHORT).show();
            }
        }
    });
```

Sign In:

```
String Email = email.getText().toString();
String Password = password.getText().toString();

if(Email.isEmpty()){
    email.setError("Please enter an email");
    Toast.makeText(getApplicationContext(), "SIGN-IN FAILED",
Toast.LENGTH_SHORT).show();
}
else if (Password.isEmpty() ){
    password.setError("Enter a password");
    Toast.makeText(getApplicationContext(), "SIGNIN FAILED",
Toast.LENGTH_SHORT).show();
}
else if (!(Email.isEmpty() && Password.isEmpty())){
    mAuth.signInWithEmailAndPassword(Email,
Password).addOnCompleteListener(new
OnCompleteListener<AuthResult>() {
```

```
        @Override
        public void onComplete(@NonNull Task<AuthResult> task) {
            if(task.isSuccessful()){
                String uid = mAuth.getUid();
                Toast.makeText(getApplicationContext(), "LOGGED
IN", Toast.LENGTH_SHORT).show();
                Intent intent = new Intent(SigninActivity.this,
ProfileActivity.class);
                intent.putExtra("uid", uid);
                intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
                startActivity(intent);
            }
            else{
                Toast.makeText(getApplicationContext(), "LOGIN
FAILED", Toast.LENGTH_SHORT).show();
            }
        }
    });
}
```

Finding and displaying registered courses:

```
public void ReadCourse(){
    myRef =
database.getInstance().getReference().child("GolfCourse");
    myRef.addValueEventListener(new ValueEventListener() {
        @Override
        public void onDataChange(@NonNull DataSnapshot
dataSnapshot) {

            if(dataSnapshot.exists()) {
```

```
                for (DataSnapshot ds :
dataSnapshot.getChildren()) {
                    String ID = ds.getKey();
                    String name =
ds.child("Name").getValue(String.class);
                    GolfCourse course = new GolfCourse(ID, name);
                    courses.add(course);
                }

                courseAdapter = new
CourseAdapter(getApplicationContext(), courses);
                listView.setAdapter(courseAdapter);
                listView.setTextFilterEnabled(true);
                search.addTextChangedListener(new TextWatcher() {
                    @Override
                    public void beforeTextChanged(CharSequence
charSequence, int i, int i1, int i2) {

                    }

                    @Override
                    public void onTextChanged(CharSequence
charSequence, int i, int i1, int i2) {

courseAdapter.getFilter().filter(charSequence);

                    }

                    @Override
```

```
                    public void afterTextChanged(Editable
editable) {

                    }
                });



            }
        }

        @Override
        public void onCancelled(@NonNull DatabaseError
databaseError) {

        }
    });
}
```

Signout:

```
if(item.getItemId() == R.id.signout){
    Toast.makeText(this, "Signout pressed",
Toast.LENGTH_SHORT).show();
    mAuth.signOut();
    Intent intent = new Intent(CurrentGamesActivity.this,
PresentationActivity.class );
    startActivity(intent);
    finish();
    return true;
}
```

Join games: Searches if invited (registered) searches all games in current course (anon)

```
myRef.addValueEventListener(new ValueEventListener() {
    @Override
    public void onDataChange(@NonNull DataSnapshot dataSnapshot)
{
        int Location = 1;
        for(DataSnapshot ds:
dataSnapshot.child("Games").child(GolfCourse).getChildren()){
            String gameID = ds.getKey();
            if(ds.child(Uid).exists()) {
                String playerID =
ds.child(Uid).getKey().toString();
                String GroupLeader =
ds.child("GroupLeader").getValue().toString();
                if(ds.child("Location").exists()) {
                    Location =
Integer.parseInt(ds.child("Location").getValue().toString());
                }
                String TimeStarted =
ds.child("TimeStarted").getValue().toString();
                Game game = new Game(gameID, GolfCourse,
playerID, GroupLeader, Location, TimeStarted, 0, "");
                games.add(game);
                Log.e("CurrentGamesActivity", game.toString());
            }else{
                String GroupLeader =
ds.child("GroupLeader").getValue().toString();
                if(ds.child("Location").exists()) {
```

```
                    Location =
Integer.parseInt(ds.child("Location").getValue().toString());
                }
                else{
                    Location = 0;
                }
                String TimeStarted =
ds.child("TimeStarted").getValue().toString();
                //Toast.makeText(CurrentGamesActivity.this,
TimeStarted, Toast.LENGTH_SHORT).show();
                Game game = new Game(gameID, GolfCourse, anonNum,
GroupLeader, Location, TimeStarted, 0, "");
                games.add(game);

            }
        }

        GameAdapter gameAdapter = new
GameAdapter(getApplicationContext(), games);
        listView.setAdapter(gameAdapter);
    }
    @Override
    public void onCancelled(@NonNull DatabaseError databaseError)
{ }});
```

Finding users through inviting and adding them to current group:

```
myRef.child("Users").addValueEventListener(new
ValueEventListener() {
    @Override
```

```
   public void onDataChange(@NonNull DataSnapshot dataSnapshot)
{
       if(dataSnapshot.exists()){
           users = new ArrayList<Users>();
           for(DataSnapshot ds: dataSnapshot.getChildren()){
               String userId = ds.getKey();
               String email =
dataSnapshot.child(userId).child("email").getValue().toString();
               if(userId.equals(Uid)){
                   Log.e("Not adding", "Not adding my ID");
               }else {
                   Users user = new Users();
                   user.setEmail(email);
                   user.setUid(userId);
                   users.add(user);
               }
           }

           Log.e("Users", users.toString());
           icon = findViewById(R.id.addIcon);
           if(extras.containsKey("group")){
               Group = extras.getStringArrayList("group");
               UserAdapter  userAdapter = new
UserAdapter(getApplicationContext(), users);
               listView.setAdapter(userAdapter);
               listView.setTextFilterEnabled(true);
               Log.e("Group", Group.toString());
               for(int i = 0; i < userAdapter.getCount();i++){
```

```
if(Group.contains(userAdapter.getItem(i).Uid)){

                    }
                }
            }else{
                UserAdapter  userAdapter = new
UserAdapter(getApplicationContext(), users);
                listView.setAdapter(userAdapter);
                listView.setTextFilterEnabled(true);
            }
        }else{
            Log.e("ManageGroupActivity", "Datasnapshot doesnt
exist");
        }
    }

    @Override
    public void onCancelled(@NonNull DatabaseError databaseError)
{

    }
});
```

Setting values for game and checking for anon and if user has added others to group:

```
String gameID = myRef.child("Games").push().getKey();
if(currentFirebaseUser != null) {

myRef.child("Games").child(GolfCourse).child(gameID).child("Grou
pLeader").setValue(currentFirebaseUser.getEmail());
```

```
myRef.child("Games").child(GolfCourse).child(gameID).child(curre
ntFirebaseUser.getUid()).child("Difficulty").setValue(Difficulty
);

myRef.child("Games").child(GolfCourse).child(gameID).child(curre
ntFirebaseUser.getUid()).child("score").child("holes").child("ho
le1").setValue(0);

myRef.child("Games").child(GolfCourse).child(gameID).child("Loca
tion").setValue("1");

myRef.child("Games").child(GolfCourse).child(gameID).child("Time
Started").setValue(currentTime());
}
else{

myRef.child("Games").child(GolfCourse).child(gameID).child("Grou
pLeader").setValue("Anon");

myRef.child("Games").child(GolfCourse).child(gameID).child(anonN
um).child("Difficulty").setValue(Difficulty);

myRef.child("Games").child(GolfCourse).child(gameID).child(anonN
um).child("score").child("holes").child("hole1").setValue(0);

myRef.child("Games").child(GolfCourse).child(gameID).child("Loca
tion").setValue("1");
```

```
myRef.child("Games").child(GolfCourse).child(gameID).child("Time
Started").setValue(currentTime());
}
if(groupIDs!=null) {
   for (int i = 0; i < groupIDs.size(); i++) {

myRef.child("Games").child(GolfCourse).child(gameID).child(group
IDs.get(i)).child("score").child("holes").
               child("hole1").setValue(0);
   }
}
```

Grabbing arguments for game to function properly (course name, user info, etc.):

```
if(mAuth.getCurrentUser() == null){
   email = anonNum;
}
else{
   email = mAuth.getCurrentUser().getEmail();
}

myRef.child("Games").child(CourseName).child(gameID).addValueEve
ntListener(new ValueEventListener() {
   @Override
   public void onDataChange(@NonNull DataSnapshot dataSnapshot)
{
       holenum =
dataSnapshot.child("Location").getValue().toString();
   }
```

```
    @Override
    public void onCancelled(@NonNull DatabaseError databaseError)
{

    }
});

Log.e("Game2Activity", bundle.toString());

mAuth = FirebaseAuth.getInstance();
if(mAuth.getCurrentUser() == null){
    Uid = anonNum;
}
else{
    Uid = mAuth.getUid();
}
```

Updating score to firebase:

```
myRef.child("Games").child(CourseName).child(GameID).addValueEve
ntListener(new ValueEventListener() {
    @Override
    public void onDataChange(@NonNull DataSnapshot dataSnapshot)
{
        if(dataSnapshot.exists()){
            int location =
Integer.parseInt(dataSnapshot.child("Location").getValue().toStr
ing());

if(dataSnapshot.child(Uid).child("score").child("holes").child("
hole"+location).exists()) {
```

```
                playerPar =
Integer.parseInt(dataSnapshot.child(Uid).child("score").child("h
oles").child("hole" + location).getValue().toString());
            }else{
                playerPar = 0;
            }
            score.setText("Score: "+playerPar);
            if(playerPar== 0){
                shot.setText("Take Tee Shot");
            }else{
                shot.setText("Take A Stroke");
            }
            shot.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View view) {
                    playerPar +=1;

myRef.child("Games").child(CourseName).child(GameID).child(Uid).
child("score").child("holes").child("hole"+holeNum).setValue(pla
yerPar)
                            .addOnCompleteListener(new
OnCompleteListener<Void>() {
                        @Override
                        public void onComplete(@NonNull
Task<Void> task) {
                            if(task.isSuccessful()){
                                score.setText("Score: " +
playerPar);
                                shot.setText("Take A Stroke");
```

```
                            }
                        }
                    });
                }
            });
        }
    }
    @Override
    public void onCancelled(@NonNull DatabaseError databaseError)
{ }});
```

Updating current hole info:

```
myRef.child("Games").child(CourseName).child(GameID).child("Loca
tion").addValueEventListener(new ValueEventListener() {
    @Override
    public void onDataChange(@NonNull DataSnapshot dataSnapshot)
{
        if(dataSnapshot.exists()){
            holeNum =
Integer.parseInt(dataSnapshot.getValue().toString());
            Log.e("Hole Num", String.valueOf(holeNum));
            getHoleDetails2();

            NextHole.setOnClickListener(new
View.OnClickListener() {
                @Override
                public void onClick(View view) {
                    holeNum +=1;
                    //Hole Number cant be greater than 18
                    if (holeNum>18){
```

```
                        holeNum = 18;
                    }

myRef.child("Games").child(CourseName).child(GameID).child("Loca
tion").setValue(holeNum).addOnCompleteListener(new
OnCompleteListener<Void>() {
                        @Override
                        public void onComplete(@NonNull
Task<Void> task) {
                            if(task.isSuccessful()){
                                getHoleDetails2();

myRef.child("Games").child(CourseName).child(GameID).child(Uid).
child("score").child("holes").child("hole"+holeNum).addValueEven
tListener(new ValueEventListener() {
                                    @Override
                                    public void
onDataChange(@NonNull DataSnapshot dataSnapshot) {
                                        if(dataSnapshot.exists())
{
                                            playerPar =
Integer.parseInt(dataSnapshot.getValue().toString());
                                        }else{
                                            playerPar = 0;
                                        }
                                        if(playerPar== 0){
                                            shot.setText("Take
Tee Shot");
                                        }else{
```

```
                                        shot.setText("Take A
Stroke");
                                    }
                                    score.setText("Score:" +
playerPar);
                                    mMap.clear();
                                    Log.e("Hole Num",
String.valueOf(holeNum));

                                }

                                @Override
                                public void
onCancelled(@NonNull DatabaseError databaseError) {

                                }
                            });
                        }
                    }
                });
            }
        });

        BackHole.setOnClickListener(new
View.OnClickListener() {
            @Override
            public void onClick(View view) {
                holeNum -=1;
                //Hole number cant be less than 1
```

```
                    if(holeNum == 0 || holeNum < 0){
                        holeNum =1;
                    }
                    //Hole Number cant be greater than 18
                    if (holeNum>18){
                        holeNum = 18;
                    }


myRef.child("Games").child(CourseName).child(GameID).child("Loca
tion").setValue(holeNum).addOnCompleteListener(new
OnCompleteListener<Void>() {
                        @Override
                        public void onComplete(@NonNull
Task<Void> task) {
                            if(task.isSuccessful()){
                                getHoleDetails2();

myRef.child("Games").child(CourseName).child(GameID).child(Uid).
child("score").child("holes").child("hole"+holeNum).addValueEven
tListener(new ValueEventListener() {
                                    @Override
                                    public void
onDataChange(@NonNull DataSnapshot dataSnapshot) {
                                        if(dataSnapshot.exists())
{
                                            playerPar =
Integer.parseInt(dataSnapshot.getValue().toString());
                                        }else{
```

```
                                        playerPar = 0;
                                    }
                                    if(playerPar== 0){
                                        shot.setText("Take
Tee Shot");
                                    }else{
                                        shot.setText("Take A
Stroke");
                                    }
                                    score.setText("Score:" +
playerPar);
                                    mMap.clear();
                                    Log.e("Hole Num",
String.valueOf(holeNum));

                                }

                                @Override
                                public void
onCancelled(@NonNull DatabaseError databaseError) {

                                }
                            });
                        }
                    }
                });
            }
        });
    }
```

```
    }

    @Override
    public void onCancelled(@NonNull DatabaseError databaseError)
{

    }
});
```

Map methods used:

```
myRef.child("GolfCourse").child(CourseName).child("Holes").child
("Hole"+Holenum).addValueEventListener(new ValueEventListener()
{
    @Override
    public void onDataChange(@NonNull DataSnapshot dataSnapshot)
{
        if (dataSnapshot.exists()) {
            if (dataSnapshot.child("Geofence").exists()) {
                List<LatLng> lstLatLngRoute = new LinkedList<>();
                ArrayList<String> points = new ArrayList<>();

if(dataSnapshot.child("Geofence").child("CourseOutline").exists(
)) {
                    int i = 0;
                    for (DataSnapshot ds :
dataSnapshot.child("Geofence").child("CourseOutline").getChildre
n()) {
                        if (dataSnapshot.exists()) {
                            points.add(String.valueOf(i));
```

```
                            i++;
                        }
                    }
                    Log.e("Number of points", points.toString());
                    for (int j = 0; j < points.size(); j++) {
                        double lat =
Double.parseDouble(dataSnapshot.child("Geofence").child("CourseO
utline").child(points.get(j)).child("lat").getValue().toString()
);
                        double lng =
Double.parseDouble(dataSnapshot.child("Geofence").child("CourseO
utline").child(points.get(j)).child("lng").getValue().toString()
);
                        LatLng latLng = new LatLng(lat, lng);
                        lstLatLngRoute.add(latLng);
                    }
                    Log.e("long lat", lstLatLngRoute.toString());

                    Polyline polyline =
GameFragment.this.mMap.addPolyline(new
PolylineOptions().clickable(true).addAll(lstLatLngRoute));
                    polyline.setTag("Hole" + holeNum);
                    getHolebounds((LinkedList) lstLatLngRoute);
                    mapTap(mMap, (LinkedList) lstLatLngRoute);

                    MapMyLocation(mMap,(LinkedList)
lstLatLngRoute);
                }
```

```
            }
        }
    }

    @Override
    public void onCancelled(@NonNull DatabaseError databaseError)
{

    }
});
}

private void getHolebounds(LinkedList lstLatLngRoute){
LatLngBounds.Builder build = new LatLngBounds.Builder();
for(int l = 0; l< lstLatLngRoute.size();l++){
    build.include((LatLng) lstLatLngRoute.get(l));
}
LatLngBounds bounds = build.build();
mMap.animateCamera(CameraUpdateFactory.newLatLngBounds(bounds,50
));
}



//Being called in the MapHole2 method
private void mapTap(GoogleMap mMap, LinkedList lstLatLngRoute){
mMap.setOnMapClickListener(new GoogleMap.OnMapClickListener() {
    @Override
```

```
   public void onMapClick(LatLng latLng) {
        LatLngBounds.Builder build = new LatLngBounds.Builder();
        for(int l = 0; l< lstLatLngRoute.size();l++){
            build.include((LatLng) lstLatLngRoute.get(l));
        }
        LatLngBounds bounds = build.build();
        if(bounds.contains(latLng)){

myRef.child("GolfCourse").child(CourseName).child("Holes").child
("Hole"+holeNum).child("Geofence").child("Hole").addValueEventLi
stener(new ValueEventListener() {
                @Override
                public void onDataChange(@NonNull DataSnapshot
dataSnapshot) {
                    if(dataSnapshot.exists()){
                        double lat =
Double.valueOf(dataSnapshot.child("lat").getValue().toString());
                        double lng =
Double.valueOf(dataSnapshot.child("lng").getValue().toString());
                        LatLng holelatlng = new LatLng(lat,lng);
                        double yards =
CalculationByDistance(MyLocation(), latLng);
                        if(yards <=65) {
                            Toast.makeText(getContext(), yards +
" yards. Try a Lob wedge", Toast.LENGTH_LONG).show();
                        }else if(yards >= 90 && yards < 110){
                            Toast.makeText(getContext(), yards +
" yards. Try a Sand Wedge", Toast.LENGTH_LONG).show();
                        }
```

```
                    else if(yards >= 110 && yards < 120){
                        Toast.makeText(getContext(), yards +
" yards. Try a 9-iron", Toast.LENGTH_LONG).show();
                    } else if(yards > 120 && yards <= 130){
                        Toast.makeText(getContext(), yards +
" yards. Try a 8-iron", Toast.LENGTH_LONG).show();
                    }else if(yards > 130 && yards <= 140){
                        Toast.makeText(getContext(), yards +
" yards. Try a 7-iron ", Toast.LENGTH_LONG).show();
                    }else if(yards > 140 && yards <= 150){
                        Toast.makeText(getContext(), yards +
" yards. Try a 6-iron", Toast.LENGTH_LONG).show();
                    } else if(yards > 150 && yards <= 160){
                        Toast.makeText(getContext(), yards +
" yards. Try a 5-iron", Toast.LENGTH_LONG).show();
                    } else if(yards > 160 && yards <= 170){
                        Toast.makeText(getContext(), yards +
" yards. Try a 4-iron", Toast.LENGTH_LONG).show();
                    }else if(yards > 170 && yards <= 180){
                        Toast.makeText(getContext(), yards +
" yards. Try a 3-iron", Toast.LENGTH_LONG).show();
                    } else if(yards > 180 && yards <= 190){
                        Toast.makeText(getContext(), yards +
" yards. Try a 2-iron", Toast.LENGTH_LONG).show();
                    }else if(yards > 190 && yards <= 210){
                        Toast.makeText(getContext(), yards +
" yards. Try a 3-wood", Toast.LENGTH_LONG).show();
                    }else if(yards >=230){
```

```
                            Toast.makeText(getContext(), yards +
" yards. Try a Driver", Toast.LENGTH_LONG).show();
                        }
                    }
                }

                @Override
                public void onCancelled(@NonNull DatabaseError
databaseError) {

                }
            });

        }else{
            Toast.makeText(getContext(), "That is not in the
correct bounds", Toast.LENGTH_SHORT).show();
        }


    }
});
}

private BitmapDescriptor bitmapDescriptorFromVector(Context
context,@DrawableRes int vectorDrawableResourceId) {
Drawable background = ContextCompat.getDrawable(context,
vectorDrawableResourceId);
background.setBounds(0, 0, background.getIntrinsicWidth(),
background.getIntrinsicHeight());
```

```
Drawable vectorDrawable = ContextCompat.getDrawable(context,
vectorDrawableResourceId);
vectorDrawable.setBounds(40, 20,
vectorDrawable.getIntrinsicWidth() + 40,
vectorDrawable.getIntrinsicHeight() + 20);
Bitmap bitmap =
Bitmap.createBitmap(background.getIntrinsicWidth(),
background.getIntrinsicHeight(), Bitmap.Config.ARGB_8888);
Canvas canvas = new Canvas(bitmap);
background.draw(canvas);
vectorDrawable.draw(canvas);
return BitmapDescriptorFactory.fromBitmap(bitmap);
}

private void MapCurrentHole(GoogleMap mMap, int holeNum){
myRef.child("GolfCourse").child(CourseName).child("Holes").child
("Hole"+holeNum).child("Geofence").child("Hole").addValueEventLi
stener(new ValueEventListener() {
   @Override
   public void onDataChange(@NonNull DataSnapshot dataSnapshot)
{
       if(dataSnapshot.exists()){
           Log.e("ButtonClicked", "DataSnapShotExists");
           List<LatLng> lstLatLngRoute = new LinkedList<>();
           double startLat = 0;
           double startLng = 0;

           LatLng Start = new LatLng(startLat, startLng);
```

```
            Log.e("Geofence", "Exist");
            startLat =
Double.parseDouble(String.valueOf(dataSnapshot.child("lat").getV
alue().toString()));
            startLng =
Double.parseDouble(String.valueOf(dataSnapshot.child("lng").getV
alue().toString()));

            Start = new LatLng(startLat, startLng);
            Log.e("StartDetails", "Start = " + Start.toString());
            if(getContext() != null) {
                mMap.addMarker(new
MarkerOptions().position(Start).title("Hole" +
holeNum).icon(bitmapDescriptorFromVector(getContext(),
R.drawable.ic_golf_course_black_24dp)));
            }


//
//                        double yards =
CalculationByDistance(Start, End);
//                        Log.e("Yards", String.valueOf(yards));
//                        float zoomLevel = getZoomLevel(yards);
//                        Log.e("Location", startLat + "   " +
startLng + " " + endLat + " " + endLng);
//
//                        lstLatLngRoute.add(Start);
//                        lstLatLngRoute.add(End);
//
```

```
//                          zoomRoute(mMap, lstLatLngRoute);



        }else{
            Log.e("SnapShot", "DoesNot Exist");
        }
    }

    @Override
    public void onCancelled(@NonNull DatabaseError databaseError)
{

    }
});
}



private int getZoomLevel(double radius){
double scale = radius / 2000;
return ((int) (16 - Math.log(scale) / Math.log(2)));
}

public void zoomRoute(GoogleMap googleMap, List<LatLng>
lstLatLngRoute) {

if (googleMap == null || lstLatLngRoute == null ||
lstLatLngRoute.isEmpty()) return;
```

```
LatLngBounds.Builder boundsBuilder = new LatLngBounds.Builder();
for (LatLng latLngPoint : lstLatLngRoute)
   boundsBuilder.include(latLngPoint);

int routePadding = 100;
LatLngBounds latLngBounds = boundsBuilder.build();

googleMap.moveCamera(CameraUpdateFactory.newLatLngBounds(latLngB
ounds, routePadding));

}


public double CalculationByDistance(LatLng StartP, LatLng EndP)
{
int Radius =6967488;// radius of earth in yards
double lat1 = StartP.latitude;
double lat2 = EndP.latitude;
double lon1 = StartP.longitude;
double lon2 = EndP.longitude;
double dLat = Math.toRadians(lat2 - lat1);
double dLon = Math.toRadians(lon2 - lon1);
double a = Math.sin(dLat / 2) * Math.sin(dLat / 2)
       + Math.cos(Math.toRadians(lat1))
       * Math.cos(Math.toRadians(lat2)) * Math.sin(dLon / 2)
       * Math.sin(dLon / 2);
double c = 2 * Math.asin(Math.sqrt(a));
double yards = Radius * c;
double feet = yards*3;
```

```
Log.e("Radius Value", String.valueOf(yards) + " " +
String.valueOf(feet));

return yards;
}


private LatLng MyLocation(){
LatLng mylatlng = null;
if(getContext() !=null) {
   myGPS = new GPS(getContext());
   Location location = myGPS.getMylocation();
   mylatlng = new LatLng(location.getLatitude(),
location.getLongitude());
   Log.e("MyLocation", mylatlng.toString());

}
return mylatlng;
}


private void MapMyLocation(GoogleMap mMap, LinkedList
lstLatLngRoute){
LatLngBounds.Builder build = new LatLngBounds.Builder();
for(int l = 0; l< lstLatLngRoute.size();l++){
   build.include((LatLng) lstLatLngRoute.get(l));
}
LatLngBounds bounds = build.build();
```

```
if(getContext() != null) {
   if (bounds.contains(MyLocation())) {
       Log.e("MapMyLocation", MyLocation().toString());
       mMap.setOnMyLocationClickListener(new
GoogleMap.OnMyLocationClickListener() {
           @Override
           public void onMyLocationClick(@NonNull Location
location) {
               LatLng mylocation = new
LatLng(location.getLatitude(), location.getLongitude());
               mMap.addMarker(new
MarkerOptions().position(mylocation).title("me"));
           }
       });
   }
}
}
```

Returning overview of player scores and names on current hole:

```
@Override
public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
   ArrayList<String> playerIDs = new ArrayList<>();
   int holeNum =
Integer.parseInt(dataSnapshot.child("Games").child(Golfcourse).c
hild(gameID).child("Location").getValue().toString());
   Log.e("OverviewCurrent hole", String.valueOf(holeNum));
   String numberOfHoles = "";
```

```
   for(DataSnapshot ds:
dataSnapshot.child("Games").child(Golfcourse).child(gameID).getC
hildren()){
       String key = ds.getKey();
       if(key.equals("Location") ||key.equals("TimeStarted") ||
key.equals("GroupLeader") || key.equals("Date")){
           Log.e("key", "Not what we are looking for");
       }else{
           Log.e("Key", key);
           playerIDs.add(key);
       }
   }
   Log.e("playerIDs", playerIDs.toString());

   for(int i = 0; i<playerIDs.size(); i++){
       int totalscore = 0;
       int currentholescore = 0;
       String email = "";
       if(
dataSnapshot.child("Games").child(Golfcourse).child(gameID).chil
d(playerIDs.get(i)).child("score").child("holes").child("hole"+h
oleNum).exists()) {
           currentholescore =
Integer.parseInt(dataSnapshot.child("Games").child(Golfcourse).c
hild(gameID).child(playerIDs.get(i)).child("score").child("holes
").child("hole" + holeNum).getValue().toString());
       }
       Log.e("currentholescore",
String.valueOf(currentholescore));
```

```
        for(DataSnapshot ds:
dataSnapshot.child("Games").child(Golfcourse).child(gameID).chil
d(playerIDs.get(i)).child("score").child("holes").getChildren())
{
            int score =
Integer.parseInt(ds.getValue().toString());
            totalscore += score;
        }

        if(
dataSnapshot.child("Users").child(playerIDs.get(i)).child("email
").exists()){
            email =
dataSnapshot.child("Users").child(playerIDs.get(i)).child("email
").getValue().toString();
        }else{
            email = "Unknown email";
        }
        Log.e("Total score", String.valueOf(totalscore));
        PlayerOverview playerOverview1 = new
PlayerOverview(email, currentholescore, totalscore);
        playerOverview.add(playerOverview1);
    }
    Log.e("Player Overview", playerOverview.toString());
    list.setAdapter(adapter);

}
```

Info for next hole:

```
public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
```

```
                ArrayList<Game> games = new ArrayList<>();
                ArrayList<String> gameIDs = new ArrayList<>();
                long numberOfPlayersonHole = 0;
                long numberOfGamesOnHole = 0;
                double waittime = 0;
                int hole =
Integer.parseInt(dataSnapshot.child("Games").child(GolfCourse).c
hild(gameID).child("Location").getValue().toString());
                int nexthole = hole+1;
                //get all hole locations
                for(DataSnapshot ds:
dataSnapshot.child("Games").child(GolfCourse).getChildren()){
                    String gameID = ds.getKey();
                    gameIDs.add(gameID);
                }
                Log.e("Games", gameIDs.toString());
                String groupleader = "";
                int location = 0;
                String timeStarted = "";
                int playerNum;
                for(int i = 0; i<gameIDs.size(); i++){
                    if(!gameIDs.get(i).equals(gameID)) {
                        groupleader =
dataSnapshot.child("Games").child(GolfCourse).child(gameIDs.get(
i)).child("GroupLeader").getValue().toString();
                        location =
Integer.parseInt(dataSnapshot.child("Games").child(GolfCourse).c
hild(gameIDs.get(i)).child("Location").getValue().toString());
```

```
                        if
(dataSnapshot.child("Games").child(GolfCourse).child(gameIDs.get
(i)).child("TimeStarted").exists()) {
                            timeStarted =
dataSnapshot.child("Games").child(GolfCourse).child(gameIDs.get(
i)).child("TimeStarted").getValue().toString();
                        } else {
                            timeStarted = "unknown";
                        }
                        long playernum =
dataSnapshot.child("Games").child(GolfCourse).child(gameIDs.get(
i)).getChildrenCount() - 3;
                        Game game = new Game(gameIDs.get(i),
GolfCourse, "", groupleader, location, timeStarted, playernum,
"");
                        games.add(game);
                    }else{
                        Log.e("NextHoleFragment", "GameID equals
my game ID");
                    }
                }
                Log.e("Game Info", games.toString());
                for(int i = 0; i<games.size();i++){
                    if(games.get(i).getLocation() == nexthole){
                        numberOfPlayersonHole =
numberOfPlayersonHole+ games.get(i).numOfPlayers;
                        numberOfGamesOnHole++;
                    }
                }
```

```
if(dataSnapshot.child("GolfCourse").child(GolfCourse).child("Wai
tTimes").child("Hole"+location).child("WaitTime").exists()) {
                    waittime =
Double.parseDouble(dataSnapshot.child("GolfCourse").child(GolfCo
urse).child("WaitTimes").child("Hole" +
nexthole).child("WaitTime").getValue().toString());
                    Log.e("Exists", "Wait time " + waittime);

                }
                Log.e("Next Hole Info", "Number Of Game:" +
numberOfGamesOnHole + "nmber Of Players " +
numberOfPlayersonHole);
                holenum.setText("Hole " + nexthole);
                gamestext.setText(numberOfGamesOnHole +  " Games
On This Hole");
                players.setText(numberOfPlayersonHole + " Players
On This Hole");
                timewaiting.setText(waittime + " Minutes For This
Hole");


            }
```

Boilerplate template for requests:

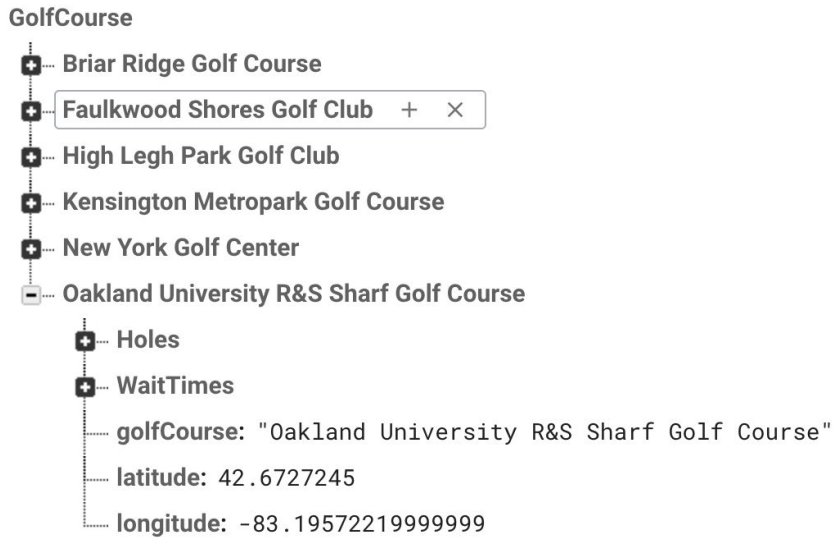
```
@Override

public void onDataChange(@NonNull DataSnapshot dataSnapshot) {


```

```
    //String email =
dataSnapshot.child(Uid).child("email").getValue(String.class);


    currentTime1 = df.format(Calendar.getInstance().getTime());


    // String emailTrun = email.split("@")[0];
    if(email!=anonNum){
        emailTrun = email.split("@")[0];
    }
    else{
        emailTrun = email;
    }

    holeNum = Integer.decode(holenum);

    taskMap.put("User", email);
    taskMap.put("Request", type2);
    taskMap.put("Location", "Hole " + holeNum );
    taskMap.put("Time", currentTime1);
    taskMap.put("Status", "Pending");
    //taskMap.put("Location", tLocation);



myRef.child("Request").child(CourseName).push().setValue(taskMap
);
```

```
   Toast.makeText(getActivity(), "Request Sent!",
Toast.LENGTH_SHORT).show();

   resetButtons();
}
```

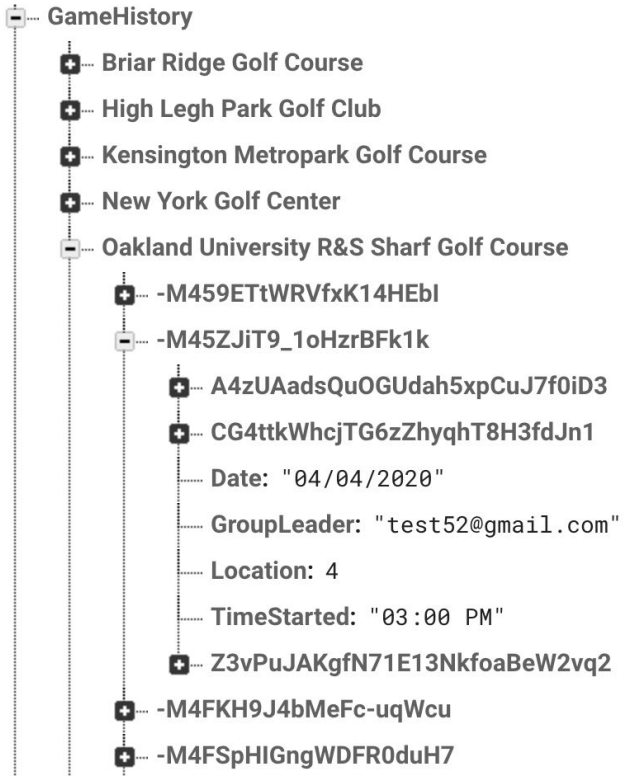
## 9.2. Data Structures

Firebase (database)
Golf Course Table:

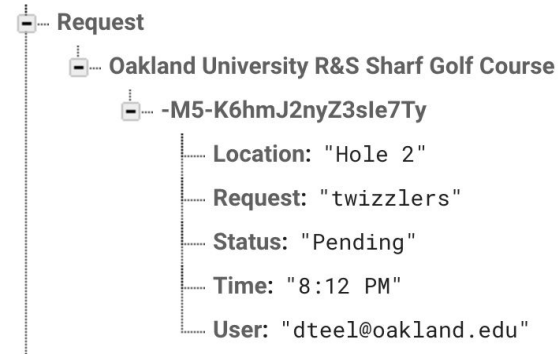



Games Table:

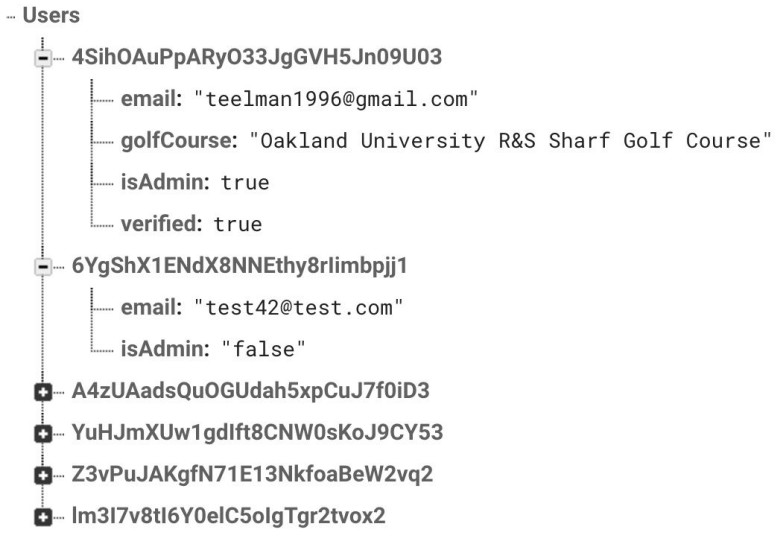
Game History Table:




Request Table:




Users table:




On the Admin Website arrays are utilized when a polygon is drawn. Each point is pushed to the aray and after the “polygoncomplete” listener is triggered the array is finalized. Once the user clicks save this array is pushed to Firebase where it resides in the given hole’s JSON structured tree. All of the entered data is pushed to Firebase as children of nodes. Nodes are created to structure the tree. When data needs to be imported then there is a transaction against the JSON structure of Firebase where the nodes need to be transversed to read the desired data.

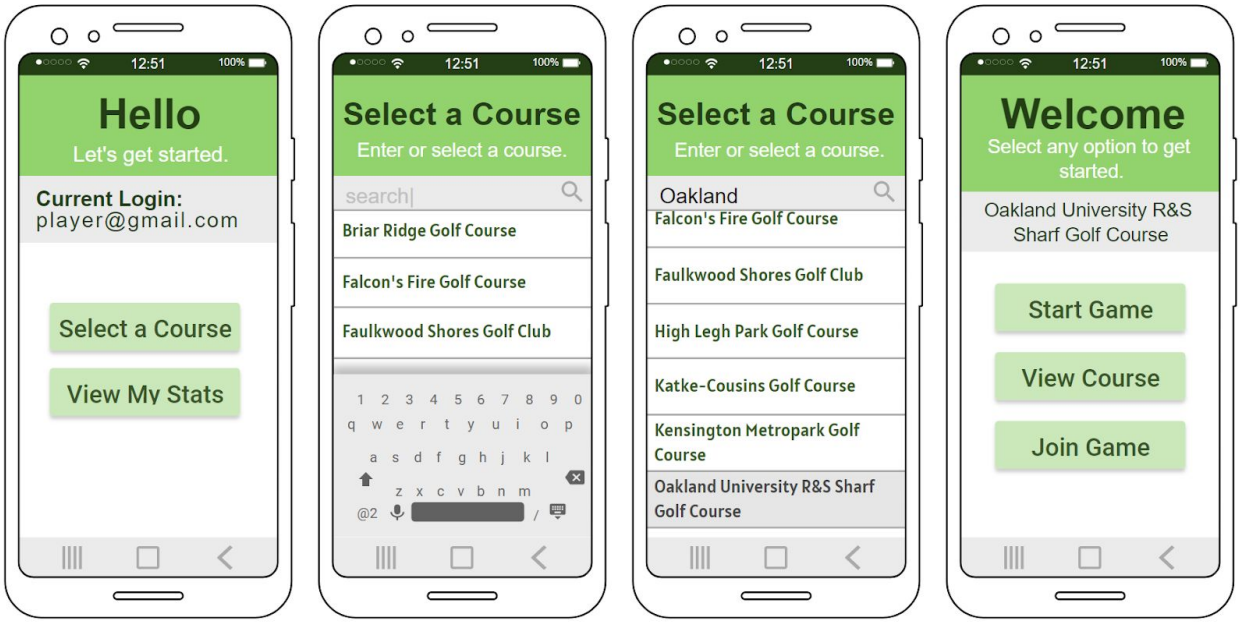
# 10. User Interface Design and Implementation

## 10.1. User Interface Design

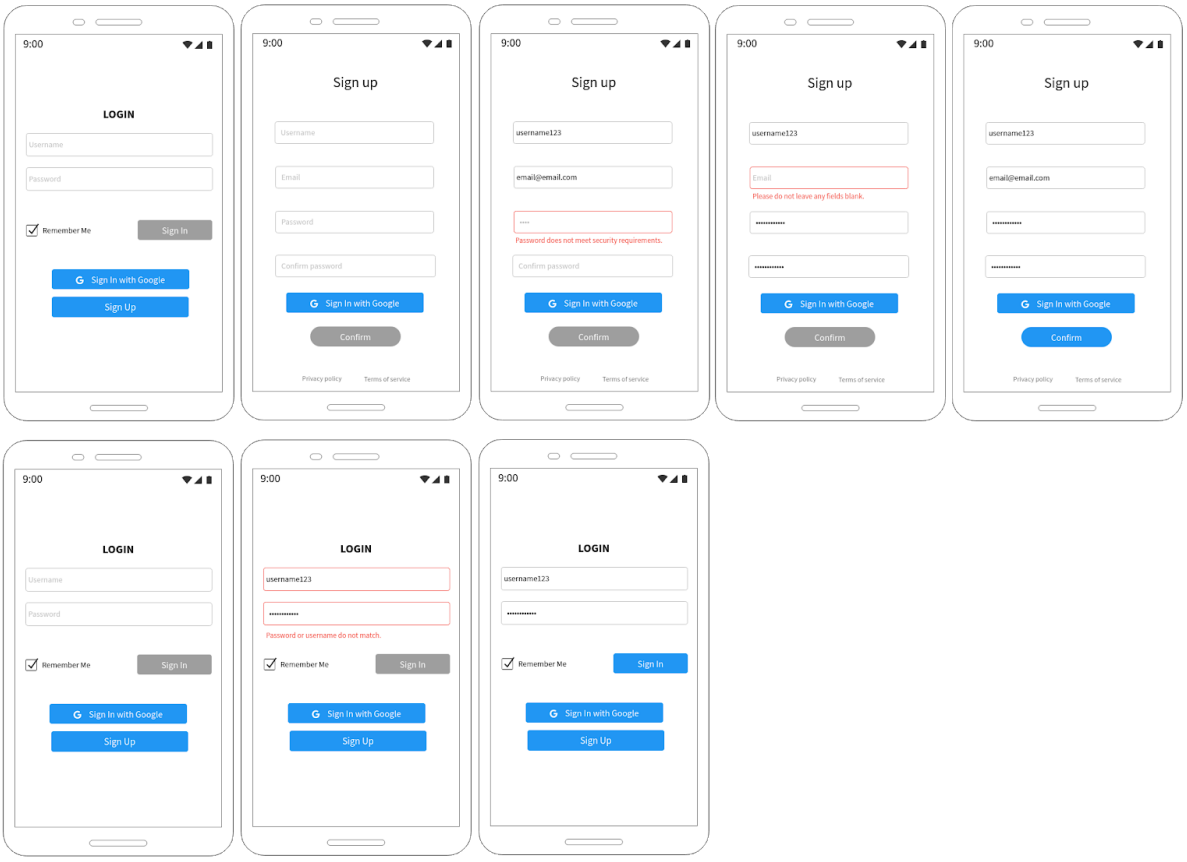
Before every development task, we have multiple tasks for design. We first start with a user story to describe what we want a particular feature to do. This is broken up by the user and events involved. Then, we design a mockup based on the user story. This mockup isn’t an exact representation of what the app will look like but it provides the developer a visual aid to work off of.
Iterations of User Interface Design:

## Mobile App

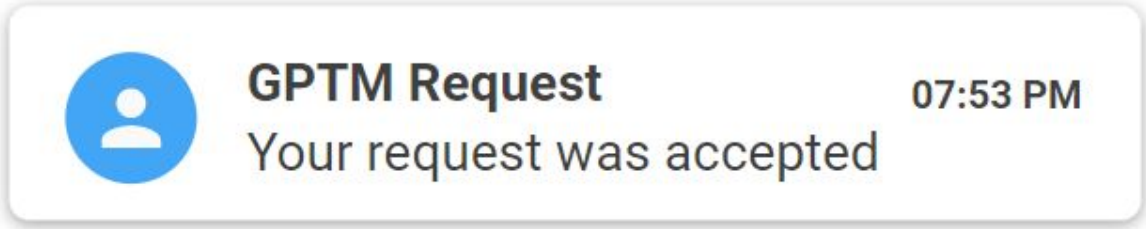

12:51
100%
Hello
player@gmail.com
Let's get started.
Select a Course
View My Stats
Select a Course
Enter or select a course.
search
Briar Ridge Golf Course
Falcon's Fire Golf Course
Faulkwood Shores Golf Club
Oakland
Falcon's Fire Golf Course
Faulkwood Shores Golf Club
High Legh Park Golf Course
Katke-Cousins Golf Course
Kensington Metropark Golf Course
Oakland University R&S Sharf Golf Course
Welcome
Oakland University R&S Sharf Golf Course
Start Game
View Course
Join Game
Hello
Let's get started.
Current Login:
player@gmail.com
Select a Course
View My Stats
Select a Course
Enter or select a course.
search
Briar Ridge Golf Course
Falcon's Fire Golf Course
Faulkwood Shores Golf Club
Oakland
Falcon's Fire Golf Course
Faulkwood Shores Golf Club
High Legh Park Golf Course
Katke-Cousins Golf Course
Kensington Metropark Golf Course
Oakland University R&S Sharf Golf Course
Welcome
Select any option to get started.
Oakland University R&S Sharf Golf Course
Start Game
View Course
Join Game

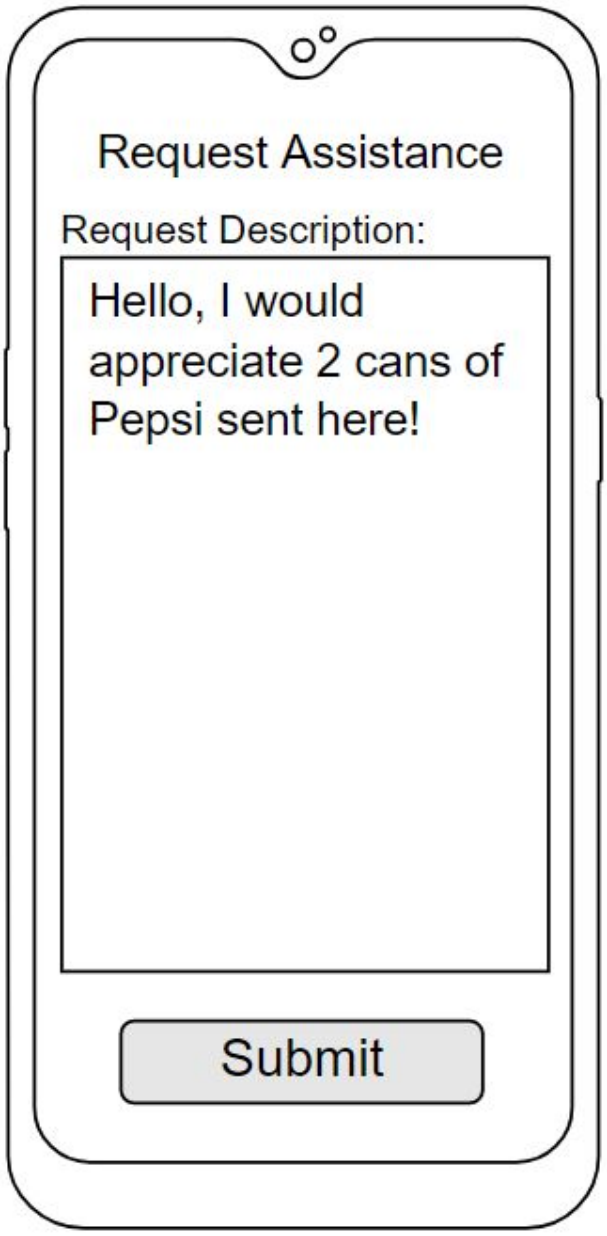



Mobile Login (with error messages)

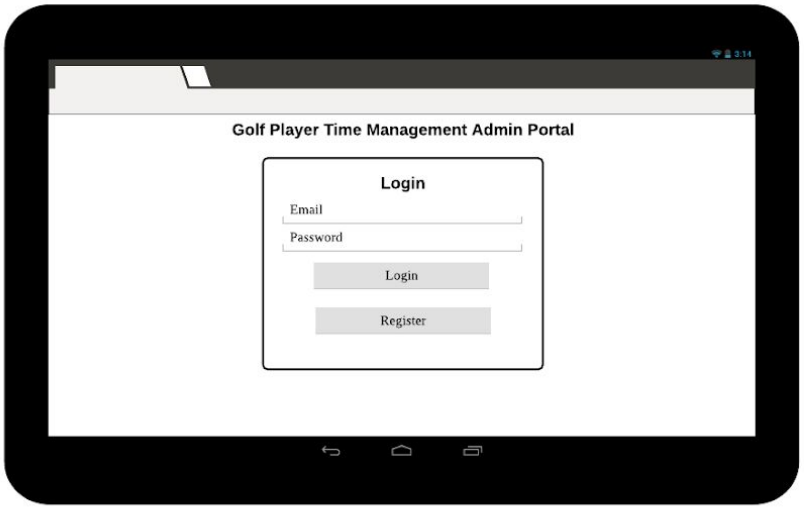


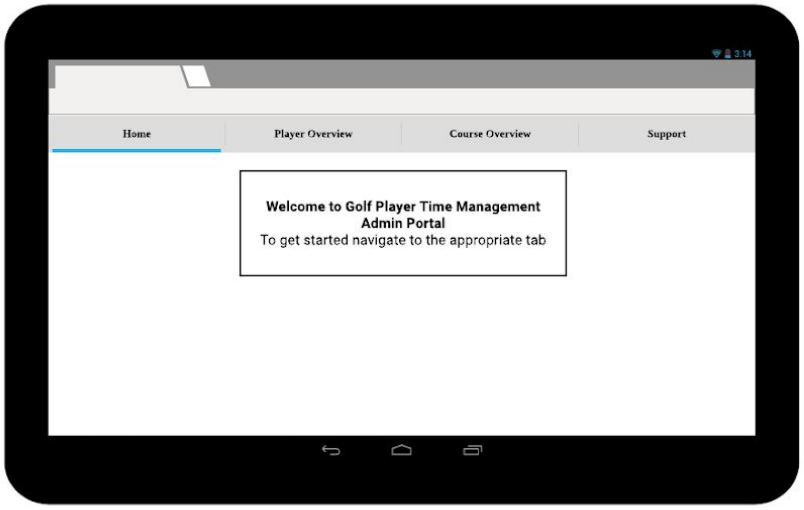

Player Request Assistance (with example notification):

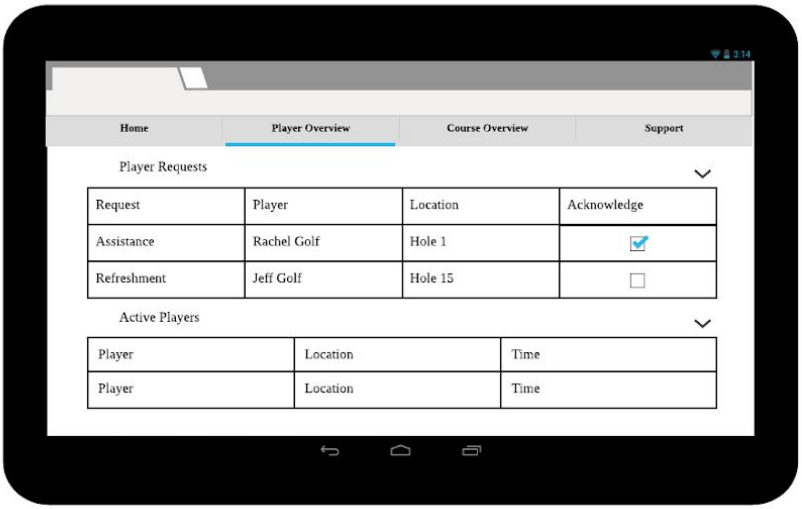

GPTM Request
07:53 PM
Your request was accepted

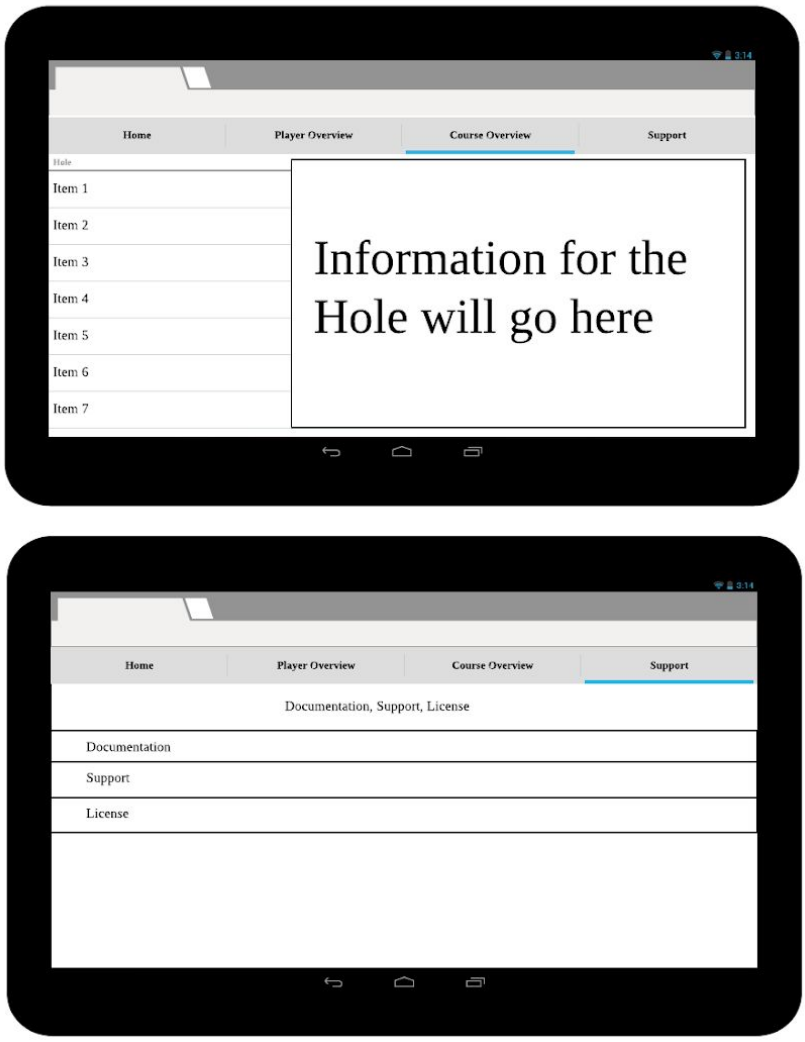

Request Assistance
Request Description:
Hello, I would appreciate 2 cans of Pepsi sent here!
Submit

# Administrator Website






## 10.2. User Interface Implementation

For implementing the user interface for all of our app and website pages we had to take many scenarios and requirements into consideration. For the mockups, we only needed to make a vague outline of the interface. But when implementing, there are more things to consider, such as visual styles, support for standard user input, as well as testing for the best positioning and functionality. Through research, what we've learned along the way in school, and our instructor's input, we found that a simple and reduced UI worked best. A decluttered and fluid design is modern and works the best for our project. As always during user design implementation optimizing is key. So for implementing, we often had to return to older areas and rework them to this new vision. We want the player to be able to use the app quickly, without thinking much or get distracted, and enjoy a sense of consistency.

In order to get the user through each step quickly and efficiently, we had to test multiple layouts of our player dashboard. We also had to have multiple iterations of language used to shorten the learning curve, and to expand our reach to a larger audience. We were especially critiqued on a distracting layout, so we made sure to start focusing on usability. Instead of going through multiple registration processes for an admin account and golf course, we streamlined it to just one. We also modified the admin dashboard to rely less on accordion menus. Consistency is also important in any app design, as a user on the app you want to see consistent language and locations of entities. We struggled with this during our developmental phases as multiple people were working on implementation, however, time was spent on ensuring this consistency. For example, if we have a submit button, the word used on the button should be "Submit" throughout the entire app or web app. It should not say "OK", "Enter", etc., anywhere. The submit button should also be in the same or similar intuitive location everywhere it appears.

# 11. Testing

## 11.1. Unit Test Architecture and Strategy/Framework

1. Look at the User Stories to determine what the expected behaviour is.
2. Test the scenarios that will yield the expected results and record these scenarios.
3. Test fringe scenarios that could potentially occur with the validation and functionality in mind and record these scenarios.
4. If there are any scenarios that yield unexpected results file a bug ticket with the scenario and the environment in the description.
5. Reguarily test the scenarios that have filed bugs to see if they were fixed. If so, mark the issue (bug) as closed. If not, keep the issue open.

## 11.2. Unit test definition, test data selection

Unit Testing for GPTM is the process of identifying the requirements that the software should meet, coding the functions to achieve the requirements, testing the implemented software in a demo build and keeping a log of testing scenarios with their accompanying outcomes, and then taking that testing one step further and testing it in a real world environment (in our case, a real golf course).

Test Data is an important task in application design and should be identified ealy. Since GPTM is geared towards golf courses we know that there is a lot of variation in

the data that can be entered. In order to be able to test things efficiently test data should be data from golf courses that are within driving distance.

The main test course that we have chosen in Oakland University's own R&S Sharf Golf Course. This course is close and within driving or walking distance for everybody on the team. As new features were rolled out it proved quite easy to head over to the course and test them in a real world environment. The course was an ideal candidate for its length (18 holes) and the multiple difficulty settings.

## 11.3. System Test Specification

### Administrator Website System Test Specifications

Login

- Input an email that is not in proper email form, input a password in less than 6 characters, input an email and password that is registered and is an administrator account, input an email and password that is registered but not an administrator account, input an email and password that has not been verified via email, input an email and password that is not registered, input a registered email with a wrong password, input a non-registered email with a registered password - errors show saying the cause
- Loading bar - only shows when login/registration is in progress

Registration

- Input a course name that already exists, input an email that's not in the proper format, input a password that's less than 6 characters, input an existing username and password combination with an unregistered course, input an unregistered course name with an unregistered email and password combination - errors show saying the cause
- Enter course name and number of holes - next button is disabled and the only option is verify
- Choose an establishment from the autocomplete that is not a golf course and hit verify - next button is greyed out and an error message is shown

Player Overview

- When switching to the player overview tab, the requests and active games tabs are expanded
- In requests, when clicking the yellow check icon button - request status is changed to 'In Progress'

- In requests, when clicking the purple check icon button - request status is changed to ‘Complete’
- In requests, when clicking the red minus icon button - request is removed
- All data in the table only pertains to the current course
- Clicking collapse all - all accordions collapse
- Clicking expand all - all accordions expand

Course Overview

- The initial display is the Congestion tab
- The Congestion tab shows the wait times in both a line graph and table format
- The Congestion tab has the number of groups at each hole in a table format
- Clicking on the hole from the nav list - goes to correct page
- Clicking the edit button - card switched to edit
- Inputting information and clicking the save button - data saved
- Clicking the cancel button in edit mode - switches to read only
- Clicking edit on another tile while another is currently in edit mode

Geofencing (in Course Overview)

- Drawing a polygon to trace the hole - lines connected and overlay is shaded
- Clicking close - closes the window and changes are not saved
- Clicking undo - the overlay is deleted
- Clicking save - data is saved and feedback is given

Support Tab

- Clicking expand all - all accordions expanded
- Clicking collapse all - all accordions collapse
- Changing emails - confirmation and email validators appear
- Clicking hyperlinks - new tab is opened on the appropriate page

## Mobile App System Test Specifications

Login

- User must input their appropriate credentials - if the incorrect credentials are put in toast is thrown telling user that login has failed

Register

- User CANNOT register under an email already recognized by GPTM - Toast is thrown telling user that that email is already in use

- User does NOT input the same password correctly twice - Toast is thrown telling user that they must get their password correct

Anonymous Sign In

- Player can join games without an invite - bypasses checking checking for invited names in GameHistory table
- Players can send requests - players given random names (i.e Anon42, Anon 90, etc.)
- Player can see other user's scores
- Player CANNOT see game history button - it is simply not displayed for them
- Player CANNOT host games - Toast is thrown telling user that that the feature cannot be used by anonymous users
- Player can make shots and go to the next hole

Course Select

- All registered courses are shown
- Courses can be searched by name - correct courses show up
- Upon course selection the corresponding course info is retrieved through firebase

Start Game

- Game Home
    - Correct hole info is shown through the drag down map screen as are the correct yards of the course and par score
- Game Overview
    - Correct current score, hole and user names are displayed on registered accounts through the listview
- Next Hole Info
    - Accurate hole info is pulled from firebase
- Admin
    - Correct request type and username is displayed to firebase

View Course

- Correct geolocation info is taken from admin site

Join Game

- Correct username is displayed when invite is sent - can only accept when user is at specified course
- GameHistory is updated to include registered user to game

See Game History

- Correct game history including course name, time taken and score are displayed - pulled from firebase --uses listview

## 11.4. Test Reports per Sprint

### Administrator Website Quality Assurance

Login

- Input an email that is not in proper email form
    - Validators provides error - expected result
- Input a password in less than 6 characters
    - Validators provides error - expected result
- Input an email and password that is registered and is an administrator account
    - Redirects to dashboard page - expected result
- Input an email and password that is registered but not an administrator account
    - Error displayed saying that the account in not an administrator account - expected result
- Input an email and password that has not been verified via email
    - Error displayed saying “Not yet verified” - expected result
- Input an email and password that is not registered
    - Error displayed saying that the email/password combination is invalid - expected result
- Input a registered email with a wrong password
    - Error displayed saying that the email/password combination is invalid - expected result
- Input a non-registered email with a registered password
    - Error displayed saying that the email/password combination is invalid - expected result
- Loading bar
    - Starts
        - After the button is pressed - expected result
    - Stops
        - When a validation error is returned - expected result
        - When the login attempt is a success - expected result

Registration

- Input a course name that already exists

    - Error displayed saying that the course name already exists - expected result
- Input an email that's not in the proper format
    - Validator provides error - expected result
- Input a password that's less than 6 characters
    - Validator provides error - expected result
- Input an existing username and password combination with an unregistered course
    - Error displayed saying that the account already exists - expected result
- Input an unregistered course name with an unregistered email and password combination
    - Prompt displays saying that the verification email has been sent - expected result
- Enter course name and number of holes
    - Next button is disabled and the only option is verify - expected result
- Choose an establishment from the autocomplete that is not a golf course and hit verify
    - Next button is greyed out and an error message saying that it is not a golf course is displayed - expected result

<u>Player Overview</u>

- When switching to the player overview tab, the requests and active games tabs are expanded
    - Expected result
- In requests, when clicking the yellow check icon button
    - Request status is changed to 'In Progress' - expected result
- In requests, when clicking the purple check icon button
    - Request status is changed to 'Complete' - expected result
- In requests, when clicking the red minus icon button
    - Request is removed - expected result
- All data in the table only pertains to the current course
    - Expected result
- Clicking collapse all
    - All accordions collapse - expected result
- Clicking expand all
    - All accordions expand - expected result

<u>Course Overview</u>

| STEP | TEST STEP/INPUT | EXPECTED RESULTS | ACTUAL RESULTS | Requirements Validated | PASS/FAIL |
|---|---|---|---|---|---|
| **Define holes: https://golf-player-time-management.firebaseapp.com/dashboard** | | | | | |
| **1.** | edit information for all holes | Information is added into firebase. | information is added into firebase | good | pass |
| **2.** | leave one blank | information is added except for blank one | information is added except for blank one | good | pass |

- The initial display is the Congestion tab
    - Expected result
- The Congestion tab shows the wait times in both a line graph and table format
    - Expected result
- The Congestion tab has the number of groups at each hole in a table format
    - Expected result
- The congestion tab updates as players move through the course
    - Expected result
- On the Congestion tab there are loading spinners while the data is loading
    - Expected result
- Course name is displayed as the header in the general description card
    - Expected result
- Expanding the accordions has proper hole data - DESIGN CHANGE TO NAV-LIST WITH CARDS
    - Description, strokes to par, and yards to hole is accurate - expected result
- Clicking on the hole from the navbar
    - The page content is the data pertaining to that hole - expected result
- Clicking the edit button
    - Expanded accordion switches to a form to input new data - expected result
- Clicking on the field
    - Current value is shown - expected result
- Inputting information into all fields and clicking the save button
    - All fields update and the tab is switched to non-edit mode - expected result
- Inputting information into description and yards fields and clicking the save button
    - Only description and yards fields update and the tab is switched to non-edit mode - expected result
- Inputting information into description and par fields and clicking the save button

    - Description and par fields update and the tab is switched to non-edit mode - expected result
- Inputting information into yards and par fields and clicking the save button
    - Yards and par fields update and the tab is switched to non-edit mode - expected result
- Inputting information into description field only and clicking the save button
    - description field updates and the tab is switched to non-edit mode - expected result
- Inputting information into yards field only and clicking the save button
    - Yards field updates and the tab is switched to non-edit mode - expected result
- Inputting information into par field only and clicking the save button
    - Par field updates and the tab is switched to non-edit mode - expected result
- Clicking the cancel button in edit mode
    - Tab switches back to non-edit mode - expected result
- Clicking edit on another tile while another is currently in edit mode
    - The tab switches to edit and the old one switches to read only - expected result

<u>Geofencing (in Course Overview)</u>

- Selecting the map icon in the general information tile
    - Dialog window appears - expected result
- Drawing a polygon to trace the hole
    - Works - expected result
- Clicking close
    - Closes the window and changes are not saved - expected result
- Clicking undo
    - The overlay is deleted - expected result
- Clicking save
    - The polygon is saved to the database - expected result
- When geofencing is not complete the field in the general tile in the hole reads ‘Not Complete’ in red
    - Expected result
- When geofencing is complete for the hole the field in the general tab reads ‘Complete’ in green
    - Expected result

<u>Support Tab</u>

- Clicking expand all with all tabs collapsed
    - All accordions expand - expected result
- Clicking expand all with 1 tab expanded
    - All accordions expand - expected result
- Clicking expand all with 2 tabs expanded
    - All accordions expand - expected result
- Clicking expand all with 3 tabs expanded
    - All accordions expand - expected result
- Clicking expand all with 4 tabs expanded
    - All accordions expanded - expected result
- Clicking expand all with 5 tabs expanded
    - All accordions expanded - expected result
- Clicking expand all with 6 tabs expanded
    - All accordions expanded - expected result
- Clicking expand all with 7 tabs expanded
    - All accordions expanded - expected result
- Clicking expand all with 8 tabs expanded
    - All accordions expanded - expected result
- Clicking collapse all with all accordions expanded
    - All accordions collapse - expected result
- Clicking collapse all with 1 tab collapsed
    - All accordions collapse - expected result
- Clicking collapse all with 2 tabs collapsed
    - All accordions collapse - expected result
- Clicking collapse all with 3 tabs collapsed
    - All accordions collapse - expected result
- Clicking collapse all with 4 tabs collapsed
    - All accordions collapse - expected result
- Clicking collapse all with 5 tabs collapsed
    - All accordions collapse - expected result
- Clicking collapse all with 6 tabs collapsed
    - All accordions collapse - expected result
- Clicking collapse all with 7 tabs collapsed
    - All accordions collapse - expected result
- Clicking collapse all with 8 tabs collapsed
    - All accordions collapse - expected result
- Clicking ‘Change email’ with ‘Change your account’s email address’ expanded
    - If blank validation error and no change - expected result
    - If not blank but not in an email format - no change - expected result

    - If in a correct email format
        - Buttons change to confirm and cancel - expected result
            - If cancel is pressed it switches button back to the original - expected result
            - If confirm is pressed it gives a confirmation message that the email is sent and explains that they won't be able to log in again until the new email has been verified - expected result
- Clicking 'Google Play' hyperlink with the 'Download the Mobile App' expanded
    - Reroutes to the Google Play App Store in a new tab - expected result
- Clicking 'email' hyperlink with the 'Contact us'' expanded
    - Reroutes to an email draft in a new tab - expected result

## Mobile App Quality Assurance

User Authentication

| STEP | TEST STEP/INPUT | EXPECTED RESULTS | ACTUAL RESULTS | Requirements Validated | PASS/FAIL |
|---|---|---|---|---|---|
| **Register a specific course** | | | | | |
| **1.** | Registering for a course. Filling in all fields and pressing register. | I am logged in and my course and account is created. | I stayed on the signin page but my account was created. User file and Course were created. | This should log the user in and take them to the dashboard. | Pass |
| **2.** | Press Register again after it didn't leave the page from the following test | It doesn't register me again. | a warning for already registered users. | good | pass |
| **3.** | left email blank | doesn't register me | doesn't register me | good | pass |
| **4.** | without filling in password | doesn't register me | doesn't register me | good | pass |
| **5.** | Left password blank | doesn't register me | doesn't register me | good | pass |

Requests

- Basic test to see if working ✔

- Original string used to display data too long
    - Fixed ✔
- Reformatting admin pages to work like the standalone page would work
    - Kept getting firebase error since I used the wrong variable for finding the path ✔
- Reinserting chronometer to game3activity
    - Attempted to use chrono in individual fragments, just put it in the fragment controller ✔
- Formatting said chronometer
    - Can’t be unified with current format since admin tab takes up most of the screen, chose to keep it in the background instead ✔

Game Activity

- Player is sent to game activity when in location of geofenced zone - ✓
- When removing strokes number will never go past 0 - ✓
- Each map shows the correct hole - ✓
- Each hole will tell you your correct par score - ✓
- Search function will find the correct player - ✓
- Strokes taken resets to zero each time “next hole” is pressed - ✓
- When group is joined correct email is displayed - ✓
- When group is joined search feature is hidden - ✓
- Correct par score is displayed for each hole - ✓
- If more than 4 players are in a group, user may not join it - ✓
- Reformatting admin pages to work like the standalone page would work
    - Kept getting firebase error since I used the wrong variable for finding the path ✔
- Reinserting chronometer to game3activity
    - Attempted to use chrono in individual fragments, just put it in the fragment controller ✔
- Formatting said chronometer
    - Can’t be unified with current format since admin tab takes up most of the screen, chose to keep it in the background instead ✔

# 12. Project Management

## 12.1. Project Plan

| Sprint | Task | Due Date |
|---|---|---|

| | | |
|---|---|---|
| 1 | User Auth UI/UX | 1/20/20 |
| 1 | User Auth User Stories | 1/20/20 |
| 1 | Develop Mobile User Auth | 1/20/20 |
| 1 | Mobile User Auth QA | 1/22/20 |
| 1 | Golf Course Registration User Stories | 1/22/20 |
| 1 | Continuous Integration Tests | 1/22/20 |
| 1 | Golf Course Registration UI/UX | 1/22/20 |
| 1 | Sprint 1 Presentation | 1/22/20 |
| 2 | Admin Site UI/UX | 1/24/20 |
| 2 | Admin Site User Auth User Stories | 1/24/20 |
| 2 | Develop Admin User Auth | 1/26/20 |
| 2 | Admin Site Nav User Stories | 1/31/20 |
| 2 | Develop Admin Nav/Layout | 1/29/20 |
| 2 | Set Up Geofencing | 2/3/20 |
| 2 | Admin Site Auth, Nav QA | 2/2/20 |
| 2 | Game Update User Stories | 2/2/20 |
| 2 | Admin Course Overview User Stories | 2/3/20 |
| 2 | Game Update UI/UX | 2/3/20 |
| 2 | Sprint 2 Presentation | 2/3/20 |
| 3 | Requests User Stories | 2/16/20 |
| 3 | Develop Course Registration | 2/14/20 |
| 3 | Develop Admin Course Overview | 2/15/20 |
| 3 | Golf Course Registration QA | 2/16/20 |
| 3 | Requests UI/UX | 2/16/20 |
| 3 | Sprint 3 Presentation | 2/17/20 |
| 4 | Rework Course Registration and Overview | 2/29/20 |
| 4 | Play Speed Prompt UI/UX | 3/1/20 |
| 4 | Develop Geofencing in the Admin Site | 3/1/20 |

| | | |
|---|---|---|
| 4 | Play Speed Prompt User Stories | 3/1/20 |
| 4 | Develop Requests | 3/1/20 |
| 4 | Requests QA | 3/1/20 |
| 4 | Assistance User Stories | 3/1/20 |
| 4 | Assistance UI/UX | 3/1/20 |
| 4 | Player Overview User Stories | 3/2/20 |
| 4 | Player Overview UI/UX | 3/2/20 |
| 4 | Sprint 4 Presentation | 3/2/20 |
| 5 | Develop Assistance | 3/15/20 |
| 5 | Develop Play Speed Prompt | 3/15/20 |
| 5 | Develop Admin Player Overview | 3/15/20 |
| 5 | Develop Location Service | 3/15/20 |
| 5 | Assistance QA | 3/15/20 |
| 5 | Play Speed Prompt QA | 3/15/20 |
| 5 | Admin Player Overview QA | 3/15/20 |
| 5 | UI Style Mockups for Mobile App | 3/15/20 |
| 5 | Sprint 5 Presentation | 3/16/20 |
| 6 | Admin Support Page | 3/29/20 |
| 6 | Mobile App UI Work | 3/29/20 |
| 6 | Admin Site UI Work, Course Congestion | 3/29/20 |
| 6 | Sprint 6 Presentation | 3/30/20 |
| Final | Final Project Poster | 4/13/20 |
| Final | Dry Run Presentation | 4/13/20 |
| Final | Admin Dashboard QA | 4/21/20 |
| Final | Mobile App QA | 4/21/20 |
| Final | Bug Bash | 4/21/20 |
| Final | Final Presentation | 4/22/20 |

## 12.2. Risk management

In order to manage risks we needed to come up with multiple plans and implementations to be able to handle any problems that arise.

Potential Database Issues
We selected a service that is known to be reliable and we should not encounter any issues with its performance. However, there are issues with the service itself.

Risk mitigation strategy: Firebase is known to have limited capability when scaled up. As well as being costly over time because of the overcharge fees. Due to these known issues, we will limit our potential clients and offered golf courses for our initial roll out and look towards merging our software/app into a larger developer cloud that would be able to handle a larger quantity of traffic for less cost.

Mobile/Web App Communication Issues
Information about golf courses is entered from an Admin on the web application. This has to display on the mobile app for players who are viewing that golf course.

Risk mitigation strategy: If the web and mobile apps do not communicate with one another, a player will be unable to continue his game. In the development of this application, we will utilize services that are known to be reliable and have minimal downtime. Although there will always be a small risk of a player's game ending prematurely due to a connection error, by using well regarded services, we can minimize the frequency of these occurrences.

Location Service Issues
Parts of our app rely on the Google Maps API to not only track location but to display and interact with it in the app.

Risk mitigation strategy: In the event that the Google Maps services API goes down for whatever reason, the app will save the player's current position on the map and give them a grace period to reconnect to the Google Maps services in order to continue the game.

# 13. References

Android Developers Documentation:

https://developer.android.com/docs

Angular Documentation:
https://angular.io/docs

Article for average play time:
https://thegolfnewsnet.com/golfnewsnetteam/2020/01/30/how-much-time-play-18-holes-9-holes-golf-74399/

Demo Golf Course Model:
https://oakland.edu/golf/courses/sharf/

Firebase Documentation:
https://firebase.google.com/docs/cli/auth#authexport
https://firebase.google.com/docs/auth/admin/manage-users
https://firebase.google.com/docs/database/web/read-and-write
https://firebase.google.com/docs/reference/js/firebase.database.Query

Google Maps API Documentation:
https://developers.google.com/maps/documentation/android-sdk/start
https://developers.google.com/maps/documentation/javascript/drawinglayer
https://developers.google.com/maps/documentation/javascript/geometry
https://developers.google.com/maps/documentation/javascript/places
https://developers.google.com/maps/documentation/javascript/controls

Material UI Documentation:
https://material.angular.io/