VirtuCards

CS 30700 Design Document

TEAM 8

June Seo

Ryan Hawks

Aryan Wadhwani

Kade Boltjes

Shayne Marques

Umang Sharma

Index

Purpose

In response to the COVID-19 pandemic, social distancing policies were implemented in public areas, which limited physical interaction and some social activities. One of these activities is playing card games with friends. In real-world situations, the card deck and other surfaces become points of contact between players. VirtuCards can replace this, as a mobile game with a common external screen that allows for in-person contactless play with a similar experience as before the pandemic.

The purpose of this project is to develop an app for mobile devices to join games and an app for laptops/desktops to host them. With both of these, we will have a seamless digital environment in order to play various different card games of your choosing in compliance with COVID-19 social distancing guidelines. There are other gaming platforms similar to what we are creating, but none will be as seamless and easy to use as VirtuCards. We hope VirtuCards will be able to make playing cards easy and safe again.

Functional Requirements

1. **Game Host**
   As a user hosting the game,
   a. I would like to be able to create a game lobby on my laptop/desktop as a common external screen, acting as a table.
   b. I would like to create a game lobby easily.
   c. I would like to be able to allow my friends to join my game using a generated join code.
   d. I would also like the option to invite my friends to join my game.
   e. I would like to have the cards played by the game players on their mobile appear on the shared screen.
   f. I would like to have the option to play another game with the same players at the end of a game.
   g. I would like to have the option to play games whose rules aren't in the library of VirtuCards.
   h. I would like to declare a winner for games whose rules aren't in the library of VirtuCards.
   i. I would like to be able to choose my preferred game easily.

j. I would like to be able to kick players from the game, when necessary.
k. I would like to be able to mute the chat if needed.
l. I would like to have songs being played from a queue from the common external screen if time allows.
m. I would like to be able to remove songs from the queue played by the shared screen if time allows.
n. I would like to be able to shuffle the songs currently in the queue if time allows.
o. I would like to be able to set a timer for each player to play their move.
p. I would like to choose custom backgrounds for the table.
q. I would like to be able to join the game lobby through my phone as a game player.
r. I would like to be able to shuffle the deck of cards easily.
s. I would like to be able to disable the chat option, if it may be detrimental to the game experience

2. Account Management

As a user,

a. I would like to be able to register for an account for VirtuCards.
b. I would like to be able to login into my account for VirtuCards.
c. I would like to reset my password, in the event I forget.
d. I would like to login with my Google account.
e. I would like to login with my Facebook account if time allows.
f. I would like the option to play as a guest, not having to make an account.
g. I would like to select an avatar from a given list to represent me
h. I would like to upload a custom photo as my avatar
i. I would like to have my avatar and other details linked to my account, allowing me to sign in to different devices with the same account
j. I would like to be able to change my username.
k. I would like to search for my friends on VirtuCards with their username.

3. Gameplay

As a user,

a. I would like to join the game host's common screen on my mobile device through the join code the host provides.
b. I would like to view the actions of other players in real-time and without any delay.
c. I would like to be able to accept invites from a game host to join their game.
d. I would like to view the cards that I have been dealt at the beginning of the round from my mobile device.
e. I would like to play the cards I have been dealt with on the table, from my mobile device.
f. I would like to draw cards from the table using my mobile device.
g. I would like to be able to pass my turn.

h. I would like to be able to fold my hand.
i. I would like to be able to see the number of games I have won and lost if time allows.
j. I would like for the cards held by each player to be visible on the shared screen, hidden face down.

4. Messaging

As a user,

a. I would like to be able to send and receive private messages to others in the same game lobby.
b. I would like to be able to send public messages to the game lobby.
c. I would like to be able to view public messages on the common screen shared between players.
d. I would like to use a default messaging system to say something very fast.
e. I would like to have the option of censoring profanity in the chat
f. I would like to have default animated reactions I can use, like “Boiler Up!”, if time allows.
g. I would like to have the ability to turn chat on and off for a group of players.

5. Social Interactions

As a user,

a. I would like to have an indication of a winner, allowing us to end a game.
b. I would like to have the games I play have the rules related to the game enforced, ensuring only legal moves are allowed.
c. I would like to be able to see where I stand compared to my peers if time allows.
d. I would like to choose a custom card sleeve for my group or myself.
e. I would like to listen to the lobby’s song playlist if time allows.
f. I would like to add songs to the queue played by the shared screen if time allows.
g. I would like to be able to hide my cards on my mobile device if I need to set my device in view of other players if time allows.

6. Miscellaneous

As a user,

a. As a game player on Android, I would like to have achievements unlocked on Google Play Games for certain milestones if time allows.
b. As a game player on iOS, I would like to have achievements unlocked on Game Center for certain milestones if time allows.
c. I would like to be able to change game settings easily.
d. I would like to be able to exit a game without disrupting the flow of play.
e. I would like sound effects when an action happens on the screen if time allows.
f. I would like haptic feedback when it is my turn to play or I make an illegal move.

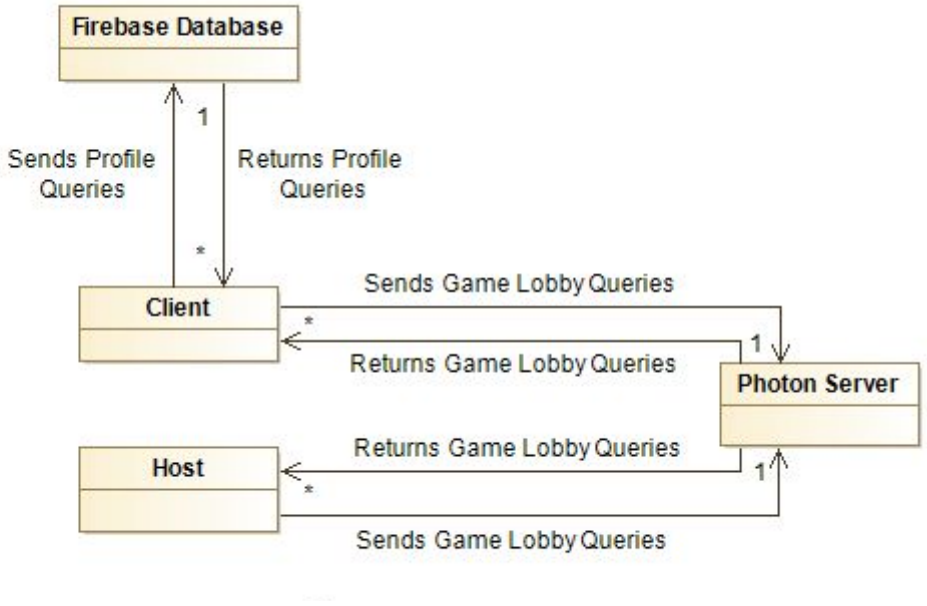
Non-Functional Requirements

- **Security:**
  As a developer,
  - I would like to set up a secure Google Firebase authentication system to allow users to register and sign in using their email and password.
  - I would like to set up alternative ways of signing in without passwords, such as Google, Facebook, and Apple accounts.
  - I would like to ensure the only identification players can see regarding other players is their username and avatar.
- **Response Time**
  As a user,
  - I would like to be able to cycle through my cards with no visible lag.
  - I would like my actions on my client game to appear on the host's screen in less than 500 ms.
  - I would like the game to take less than 5 seconds to start on my mobile device.
- **Usability**
  As a user,
  - I would like the game to be easy to navigate.
  - I would like the acts of playing and drawing cards to be simple.
  - I would like all the options that I can perform visible in an uncluttered manner
- **Hosting/Development**
  As a developer,
  - I would like to use Unity to develop both the client-side and server-side.
  - I would like to have the client-side and server-side developed as independently as possible to reduce dependency conflicts.
  - I would like the game to be deployed to both Android and iOS if time allows.
  - I would like to use Firebase's Realtime Database to store related information for each user such as their username, account avatar, list of friends, games played, games won and games lost, and a unique ID for that player.

Design Outline

High-Level Overview

This project derives from the client-server architecture and will consist of two main applications: the client and the host, which acts as the master client. The host is started as a desktop application. The client is started as a mobile game. When the host sets up a lobby, it queries the Photon server to set up the game. That allows the client to join the lobby using a join code that is generated. Once a number of clients have joined the lobby, the host can start the game. The game logic is then handled on the host to synchronize the clients. It is relayed between the clients and the host through the Photon server.



1. **Client:**
    a. The client provides the user an interface to interact with the game and track their hand of cards.
    b. The client validates and sends player input to the photon server.
    c. The client sends a request to validate authentication information to the database along with requests to retrieve profile information.
    d. The client sends queries to the Photon server to join a lobby and relay gameplay inputs to the host such as drawing or playing cards.
2. **Host:**
    a. The host provides the users with a shared interface to track the progress of the game.
    b. The host receives and interprets inputs from the client through the Photon server and renders them in its interface.
    c. It also uses the Photon server to set up new game lobbies.
    d. The host specifies which client's turn it is.

3. **Database:**
    a. A Firebase NoSQL database stores all user data such as usernames, passwords, and player statistics.
    b. Additionally, the database validates user authentications like OAuth.
4. **Photon Server:**
    a. The Photon server hosts all the different game lobbies.
    b. It allows the clients to act as players and the host to act as the game controller and common screen.
    c. Once a game has started, it relays information between clients and the host such as client inputs and which client's turn it is.

Sequence of Events

The sequence of events given below outlines the primary interactions between the client, host, database (Firebase) and server (Photon Server).

This process is initiated by the players logging into the mobile app. After using their credentials to log in, a request is sent from the client to Firebase. The app then retrieves their user details, including username and their avatar photo.

Simultaneously, the host launches the desktop app and creates a lobby with a unique game code. The host app then sends a request to the Photon Server to create a lobby with the given game code. Upon creation, the host displays the table.

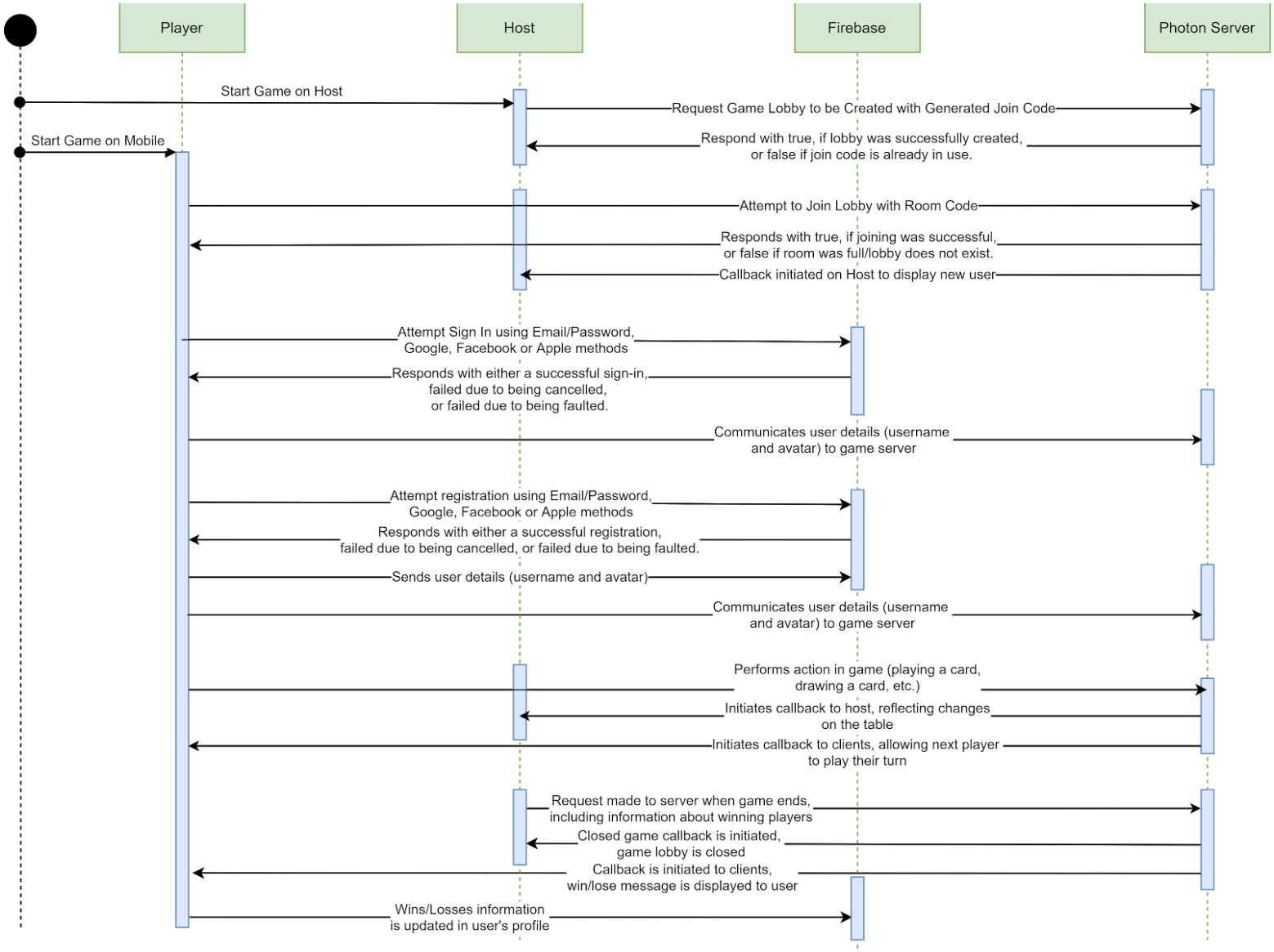
Following a successful login operation, the client can further request the server for more functionality such as creating a profile and viewing their current game statistics. In response to these requests, Firebase handles the requests made by the client, either responding with information about the user or updating information about the user.

When the player enacts a move, the move is sent to the Photon server, which initiates a callback on the host's side to reflect the changes. Another callback is sent allowing the next player to play their move.

When the host determines that the game has ended, due to a player winning, the host sends a request to the Photon Server to end the game, with information about the winners of the game. The Photon Server then initiates a callback on the player's devices indicating whether they have won or lost. After this, the player's app updates their win and loss statistics on Firebase.

An overview of the Sequence of Events is shown on the following page.

Sequence Diagram



Design Issues

Functional Issues

1. Do users need to sign in to play our games?
    a. Option 1: Have the users log in using username/password
    b. Option 2: Have the users log in using OAuth (Google, Facebook, or Apple)
    c. Option 3: No login (Play as a guest)

    Choice: All of the above

    Justification: We decided to allow users to play within all of the options. This would allow user information to be stored in the database in a more organized manner and it would allow users to view the statistics from previous games, which fosters a more competitive gaming environment. However, if people don't want to keep track of their statistics, they have the option to play as a guest.

2. What happens if the chat is getting profane?
    a. Option 1: Add a choice to censor out inappropriate words
    b. Option 2: Ban players that are being profane

    Choice: Option 1

    Justification: Banning a player for inappropriate words seems too extreme. Sometimes, when people play with their friends, swearing naturally comes and they want to speak how they want, so it would be better to give people the choice to censor inappropriate words if needed.

3. How will the game resume if I disconnect during the game?
    a. Option 1: You can't join back and have to wait until the next game
    b. Option 2: You can join back the game with your previous cards back depending on the game
    c. Option 3: Kicks everybody else out of the game to start a new game

    Choice: Option 2

    Justification: We want to make sure the game flow isn't disrupted just because one person is disconnected from the game. In certain games, we will hold the player's cards for them until they join back. In other games, we will return the player's cards to the deck when they disconnect then make the player wait until the next round to be dealt back in.

Non-Functional Issues

1. Which game engine is best suited for cross-platform building?
    a. Option 1: Unity Game Engine
    b. Option 2: Unreal Engine

    Choice: Option 1

    Justification: Unity is better for cross-platform development. It has a relatively simple interface and works best for mobile games and apps. Unity includes a huge number of assets and plugins, which would allow us to focus more on functionality. Unity also has a large online community of developers, so it would be easier to fix issues and get support during the development process. Unreal is known for more realistic graphics, but our game is not graphics heavy and is mostly two-dimensional.

2. Which programming languages are appropriate for implementing our frontend and backend service?
    a. Option 1: C#
    b. Option 2: C++

    Choice: Option 1

    Justification: C# is the main programming language used in Unity. It is also easier to use for the development of games, as it is a more recent programming language and it is a high-level language. With C++, we also need to worry about memory management, which is not a concern with C# as it runs in a virtual machine.

3. What frameworks can we use to enable multiplayer?
    a. Option 1: Photon
    b. Option 2: SmartFoxServer
    c. Option 3: Google Play Games Services

    Choice: Option 1

    Justification: Photon is a package in the Asset Store in Unity. It is a popular package for developing multiplayer games with Unity. It has simple predefined callbacks and functions that model common multiplayer game operations such as creating and joining rooms, acquiring a list of available rooms, etc. Furthermore, our team is more familiar with Photon.

4. What platform is most appropriate for our database and login services?
    a. Option 1: MongoDB
    b. Option 2: Google Firebase
    c. Option 3: Microsoft Azure
    d. Option 4: Amazon Web Services

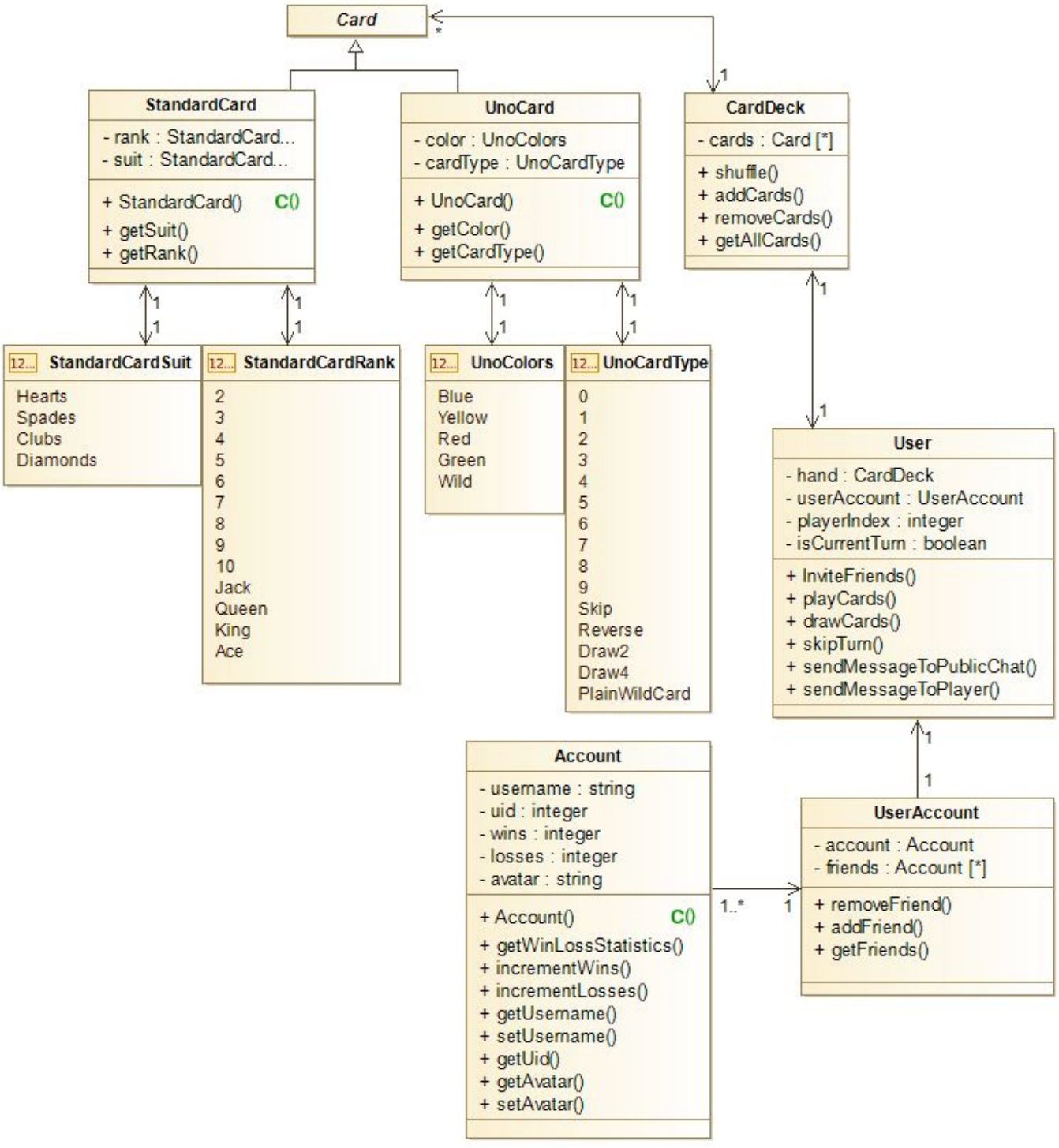
    Choice: Option 2

    Justification: Firstly, we are interested in having authentication through email/password, Google, Facebook, and Apple sign-ins enabled through the same platform as the database, to minimize the number of external services used, which is not available from MongoDB. Also, the tools we wish to use in Firebase are free, which is not the case for Azure and AWS. Firebase also has an integration directly in Unity, allowing user logins and database accesses to be more simple.

Design Details

Being the client and host will be two separate applications, they each have their own class level designs.

Client Class Level Design



Description of Client Classes and their Interactions

The classes in our client application are based on a blueprint laid out for the objects in our game.

StandardCardSuit

- Enum that defines the suits of the cards.
- Four Types: Clubs, Diamonds, Hearts, and Spades.

StandardCardRank

- Enum that defines the ranks of the cards.
- It contains all the numbered cards along with the face cards.

StandardCard

- Each StandardCard represents an individual card in an ordinary deck of 52 cards.
- The attribute rank stores a CardRank.
- The attribute suit stores a CardSuit.
- The constructor requires the rank and suit parameters to prevent undefined cards.

UnoColors

- Enum that defines the colors of the cards for the game Uno.
- Five Types: Red, Yellow, Green, Blue, and Wild.

UnoCardType

- Enum that defines the different types of the cards for the game Uno.
- It contains all the numbered cards along with the special cards like Draw2 and Skip.

UnoCard

- Each UnoCard represents an individual card in a deck for the game Uno.
- The attribute color stores an UnoColor.
- The attribute cardType stores an UnoCardType.
- The constructor requires the color and cardType parameters to prevent undefined cards.

Card

- The Card class is used as a level of abstraction to generalize the different types of cards.
- It allows the CardDeck class's card attribute to be any of the classes generalized by Card.

CardDeck

- Each User receives an object of type CardDeck when the game begins. It holds their current hand.
- The cards attribute stores a list of the objects of type Card. It is a generalization of different types of cards like the classes StandardCard and UnoCard. This allows it to have either StandardCards or UnoCards assigned to the card attribute.
- It allows Card objects to be added and removed from each player's deck.

User

- The User class stores essential data about the players in the game that is required for gameplay to proceed in a structured manner like the attributes isCurrentTurn and playerIndex.
- The attribute hand stores an object of type CardDeck which holds the hand of each individual player.
- Has the methods playCards and drawCards, which provide a level of abstraction for the CardDeck methods removeCards and addCards.
- The final attribute called userAccount contains a reference to an object of type UserAccount which stores data not relevant to the gameplay specifically and is more database and login purposes.

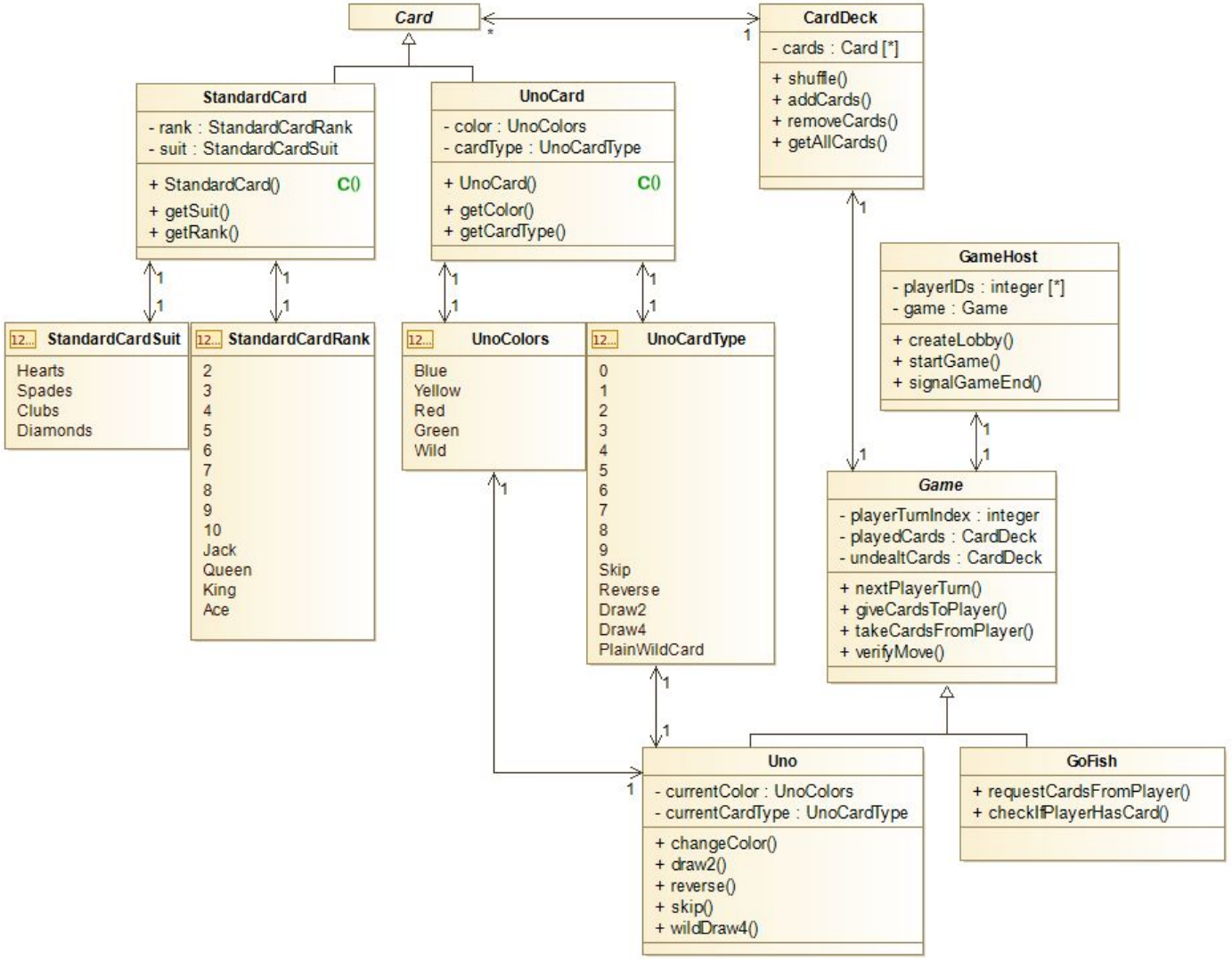
UserAccount

- The UserAccount class stores information associated with the player's account.
- The attribute account is an Account object that contains all the information for the current player.
- The attribute friends contain all the account information for the player's friends within Account objects.
- It contains methods to get, add, and remove friends.

Account

- These class models the data required for each User.
- It has attributes such as username, user ID, avatar, wins, and losses.
- It also contains methods to access game statistics for a particular player.

Host Class Level Design



Description of Host Classes and their Interactions

The classes in our host application are based on a blueprint laid out for the objects in our game.

StandardCardSuit

- Enum that defines the suits of the cards.
- Four Types: Clubs, Diamonds, Hearts, and Spades.

StandardCardRank

- Enum that defines the ranks of the cards.
- It contains all the numbered cards along with the face cards.

StandardCard

- Each StandardCard represents an individual card in an ordinary deck of 52 cards.
- The attribute rank stores a CardRank.
- The attribute suit stores a CardSuit.
- The constructor requires the rank and suit parameters to prevent undefined cards.

UnoColors

- Enum that defines the colors of the cards for the game Uno.
- Five Types: Red, Yellow, Green, Blue, and Wild.

UnoCardType

- Enum that defines the different types of the cards for the game Uno.
- It contains all the numbered cards along with the special cards like Draw2 and Skip.

UnoCard

- Each UnoCard represents an individual card in a deck for the game Uno.
- The attribute color stores an UnoColor.
- The attribute cardType stores an UnoCardType.
- The constructor requires the color and cardType parameters to prevent undefined cards.

Card

- The Card class is used as a level of abstraction to generalize the different types of cards.
- It allows the CardDeck class's card attribute to be any of the classes generalized by Card.

CardDeck

- The CardDeck keeps track of 2 main things relevant to the game; it stores all the cards that have not been dealt out yet, and it stores all the cards that all the players have dealt.
- The cards attribute stores a list of the objects of type Card. It is a generalization of different types of cards like the classes StandardCard and UnoCard. This allows the host to use different types of cards for different games.

Game

- Game is an abstract class that allows for the generalization of different card games.
- It keeps track of which player's turn it is.
- The attribute playedCards is a CardDeck that contains all the cards that all the players have played
- The attribute undealtCards is a CardDeck that contains all the cards that have not been given out to players.
- It handles the logic to deal and remove cards from players as well.

Uno

- Inherits from Game.
- It handles the game logic for Uno.
- The attributes currentColor and currentCardType keep track of the top (most recently played) card, which are used to determine valid plays.

GoFish

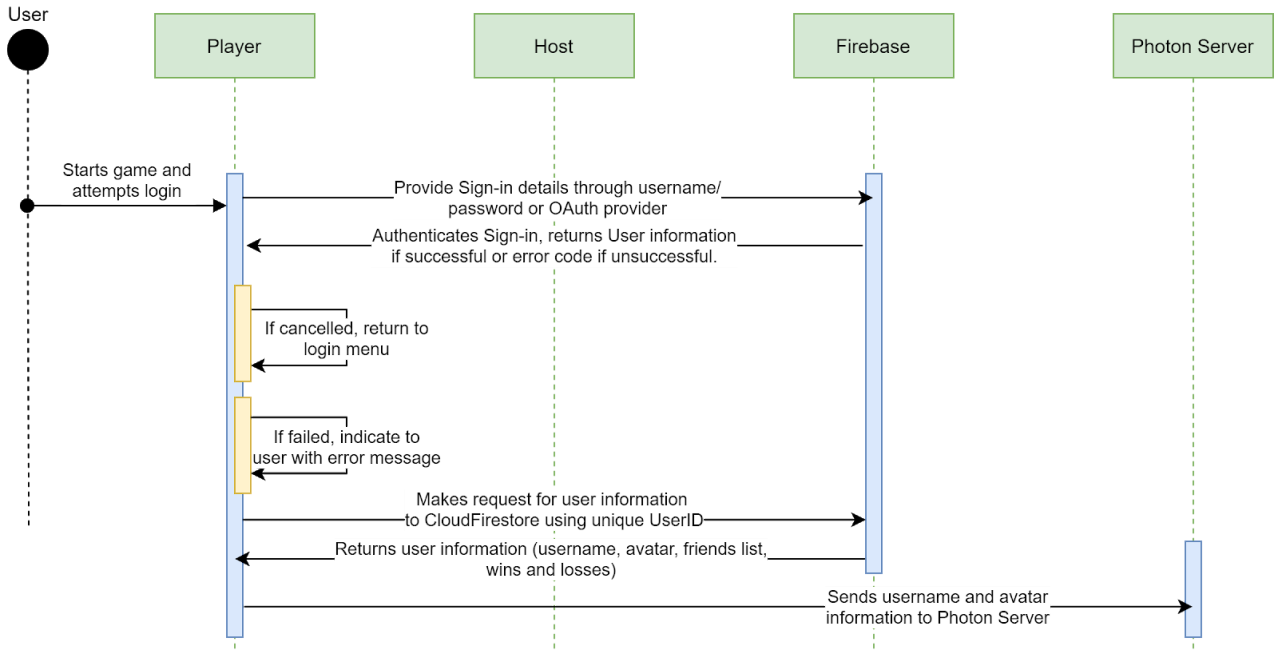
- Inherits from Game.
- It handles the game logic for GoFish
- The method checkIfPlayerHasCard is used to determine if a player can get cards from another. If they can, the method requestCardsFromPlayer will transfer the cards of a certain type to that player from the other's hand.

GameHost

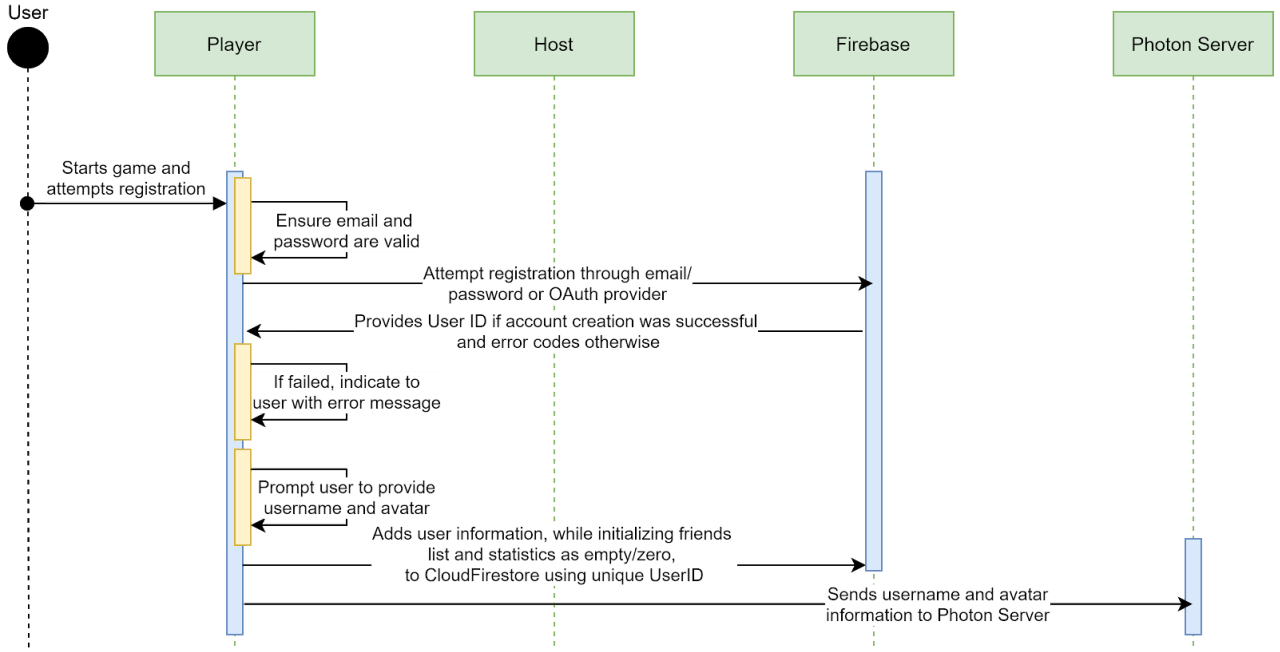
- The GameHost class keeps track of the game lobby and the players in a game.
- It creates lobbies for games, and designates which game will be played.
- The attribute playerIDs keeps a list of all the IDs of connected players.

Sequence Diagrams

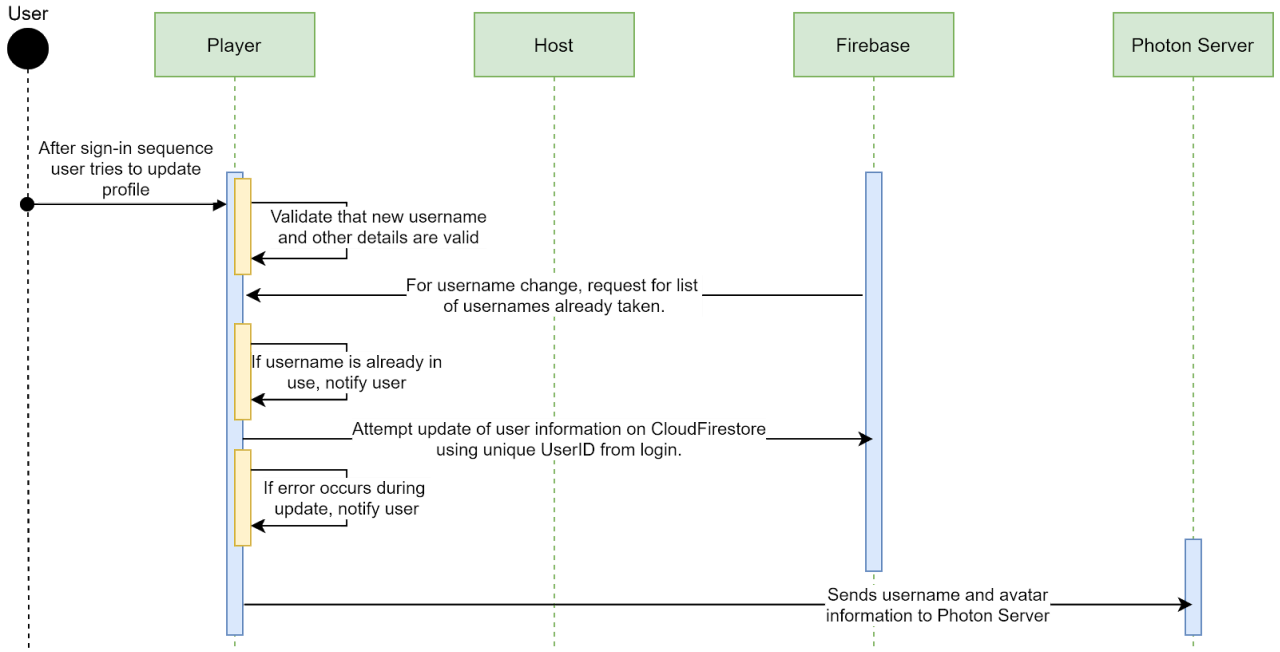
Signing in on the mobile app



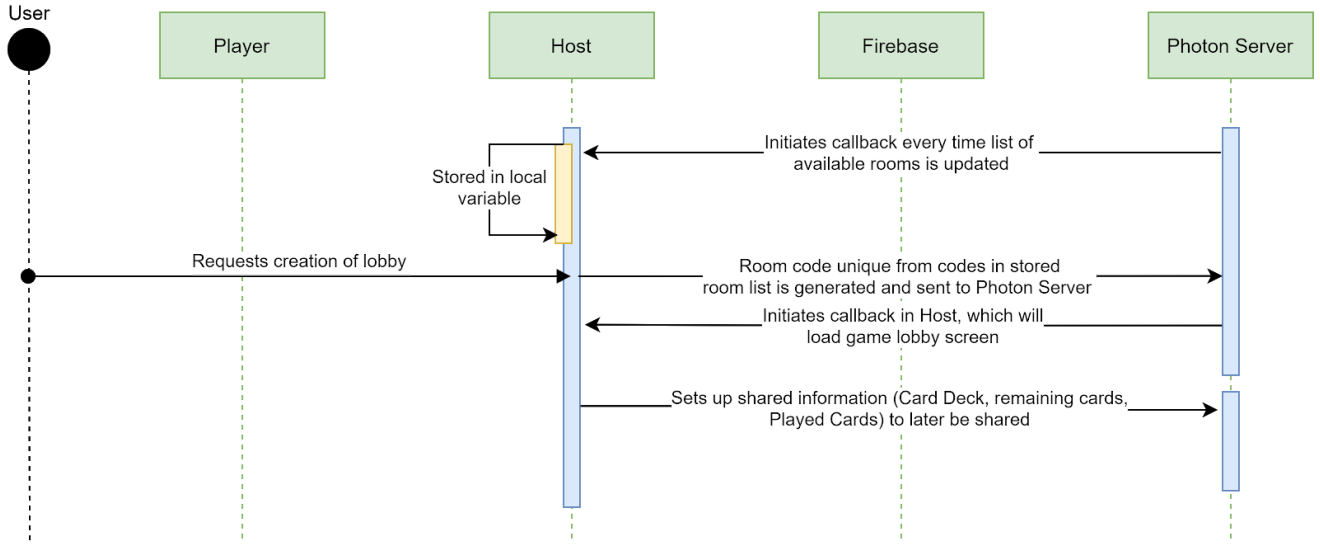
Registering on the mobile app



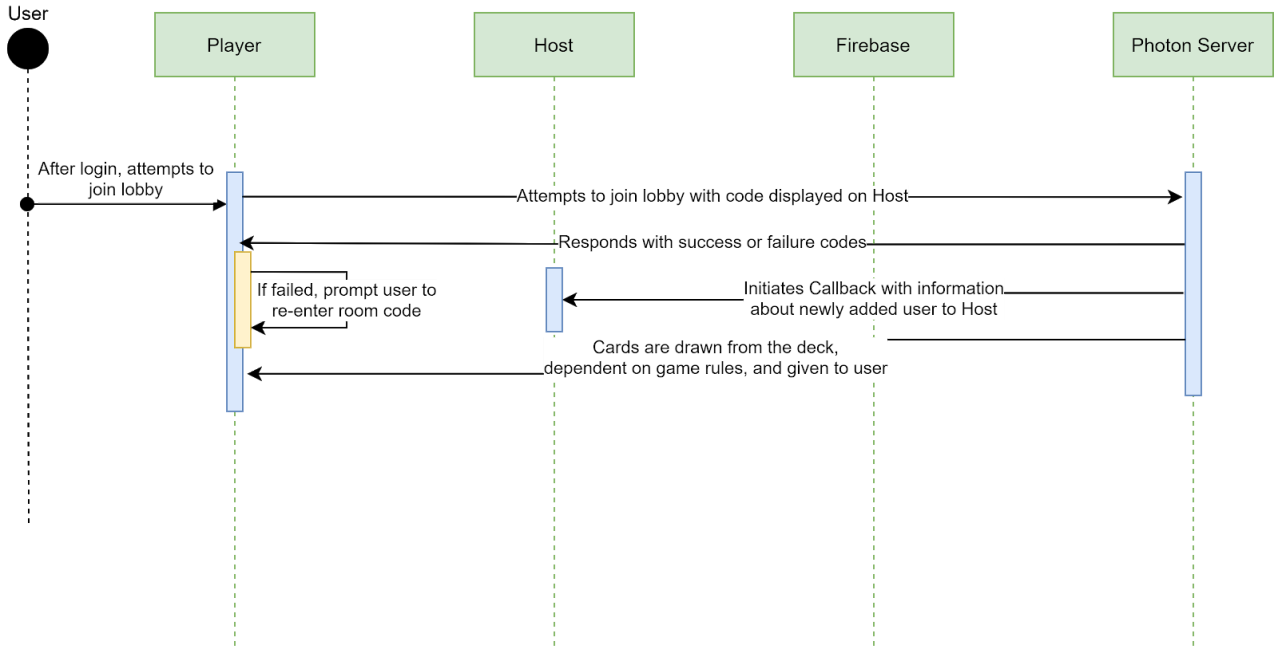
Updating user information



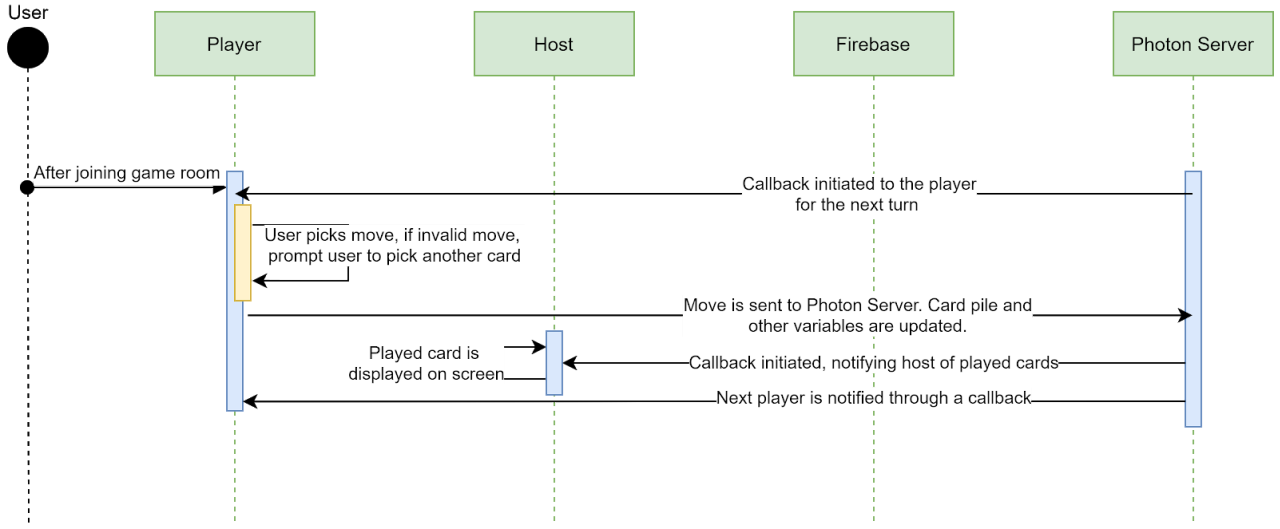
Creating Lobby



Players joining game



Players playing a Card



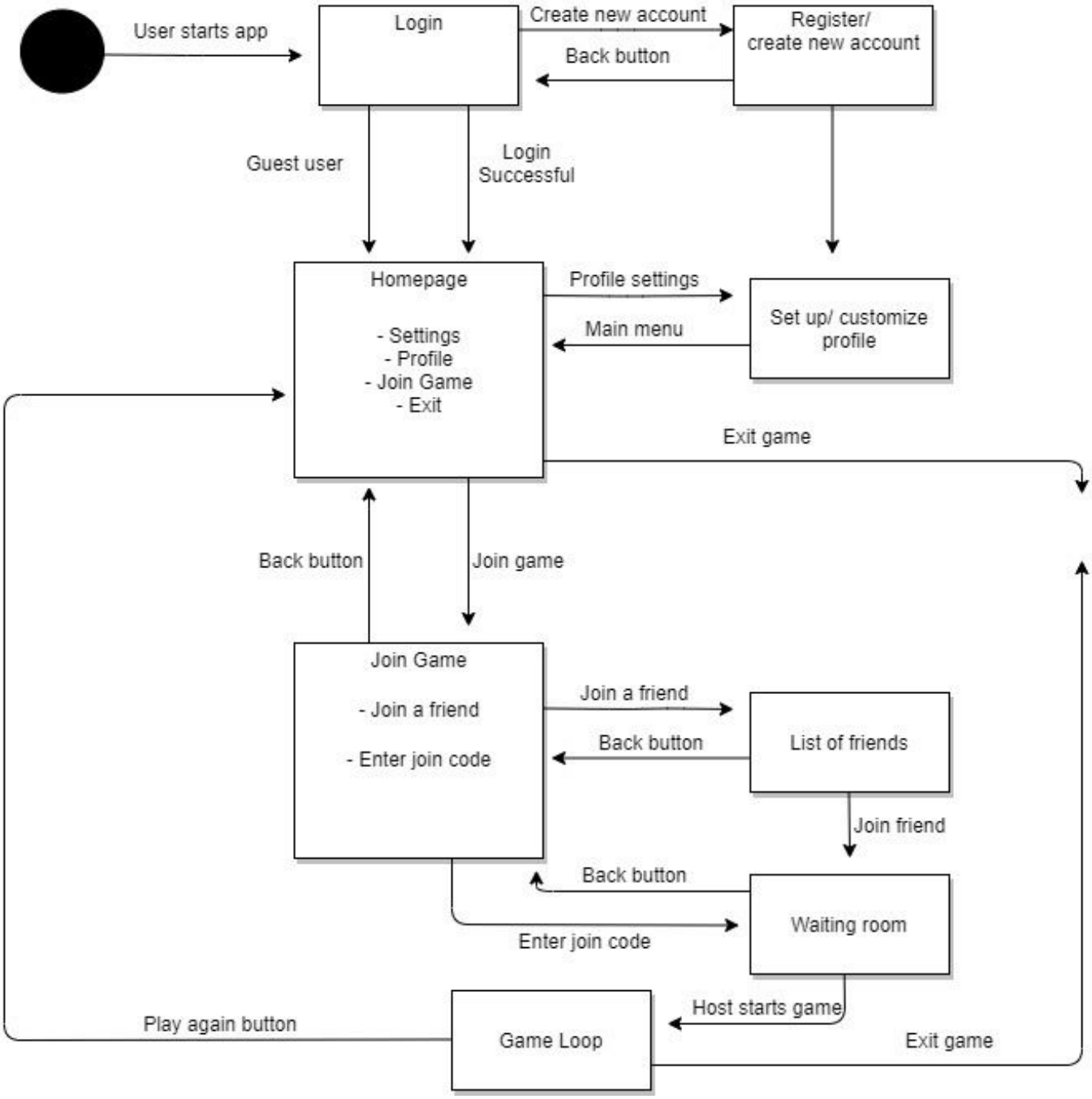
Navigation Flow Map

In order to make VirtuCards accessible to everyone, we believe it must be easy to use by everyone. The mobile and desktop app will be very intuitive and easy to use for anyone who downloads and plays VirtuCards. Since we have separate apps with fundamentally different features, they will both have different Navigation Maps in order to present the flow with as much clarity as we can. The mobile app will start by prompting the user to either log in to their account or register for a new one. After that, it will take them to the home page where they will be met with all of the options that they will need to have access to. This includes an exit button, a play button, a box to enter a code, etc.

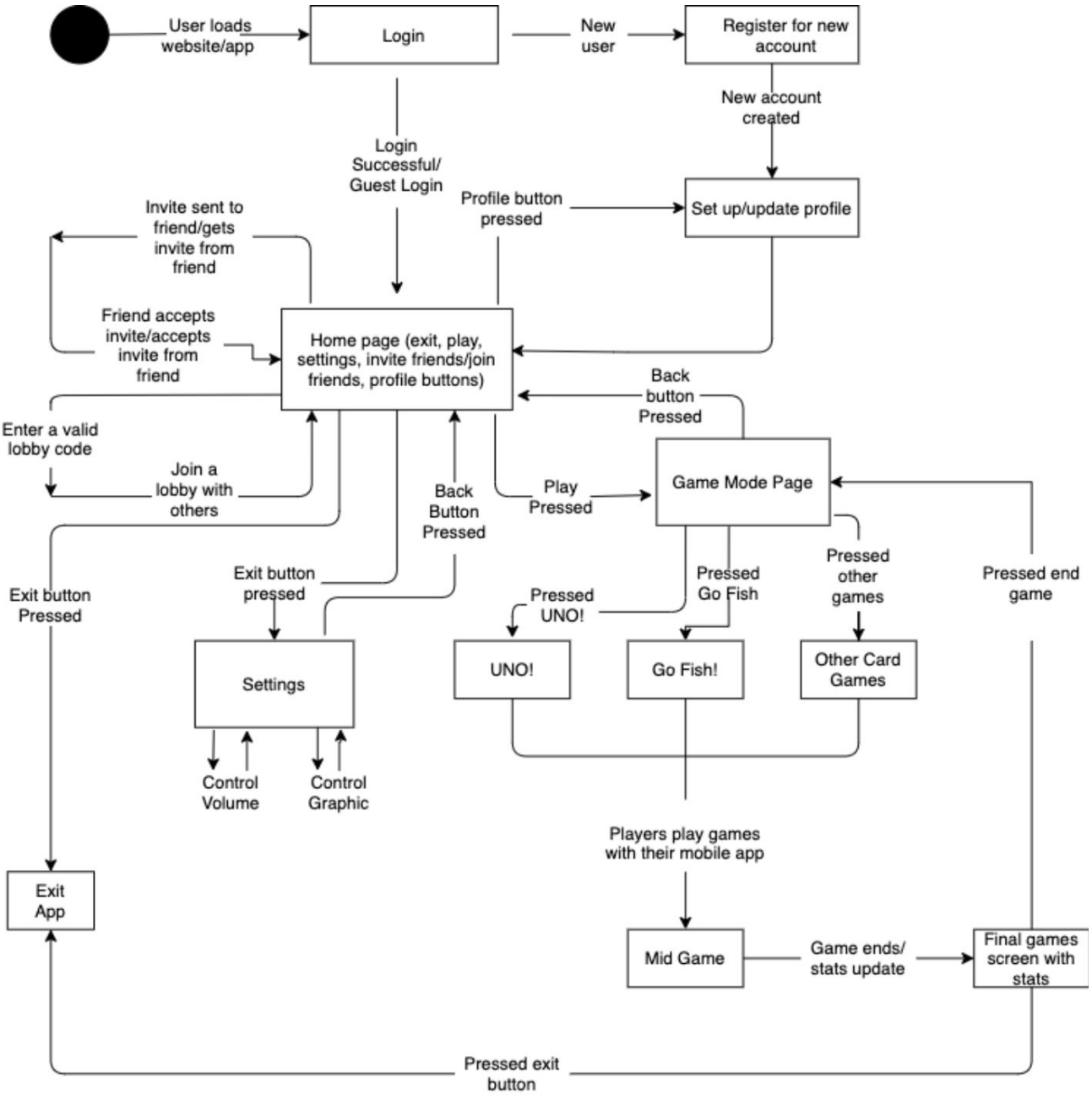
The host app (for desktop/laptop) will function a little differently. When started the user will still have the option to log in or create an account just as the mobile version does. After this, the user will be able to create a room and invite friends in order to start a game. They will have the options to choose settings such as turn time limits, custom card backs, custom table designs, and the card game mode itself.

The purpose of this project is to develop an app for mobile devices to join games and an app for laptops/desktops to host them. With both of these, we will have a seamless digital environment in order to play various different card games of your choosing in compliance with COVID-19 social distancing guidelines. There are other gaming platforms similar to what we are creating, but none will be as seamless and easy to use as VirtuCards. We hope VirtuCards will be able to make playing cards easy and safe again.

Mobile Flow Map

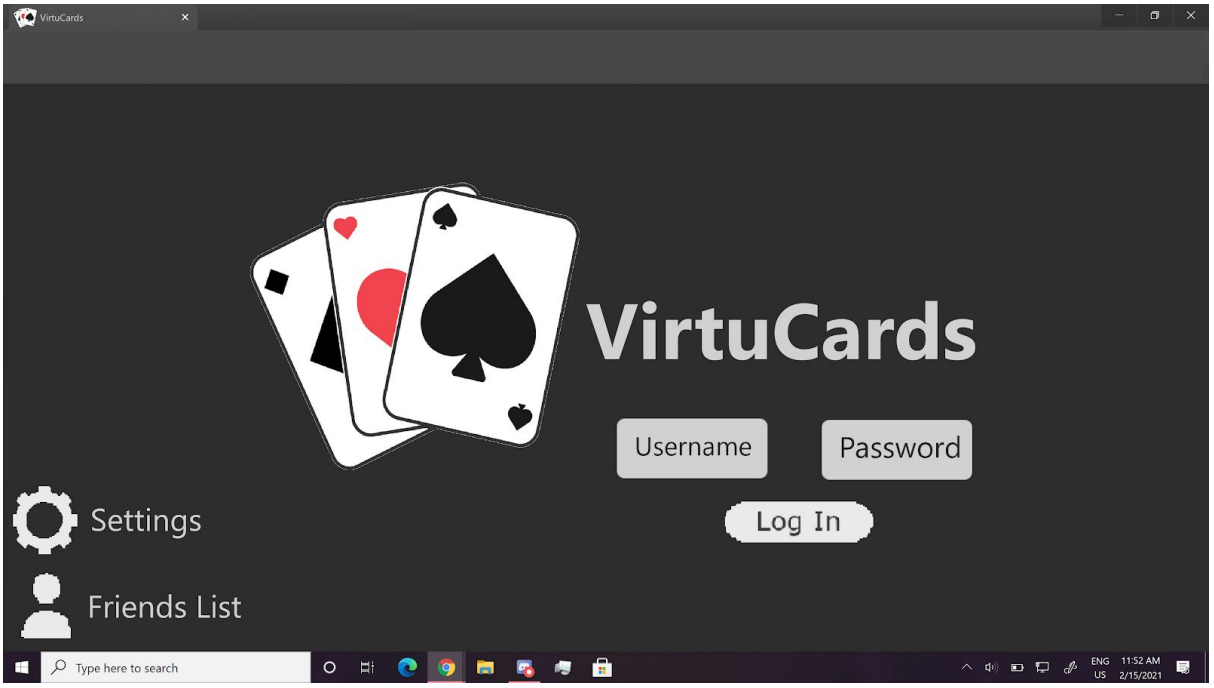


Desktop Flow Map

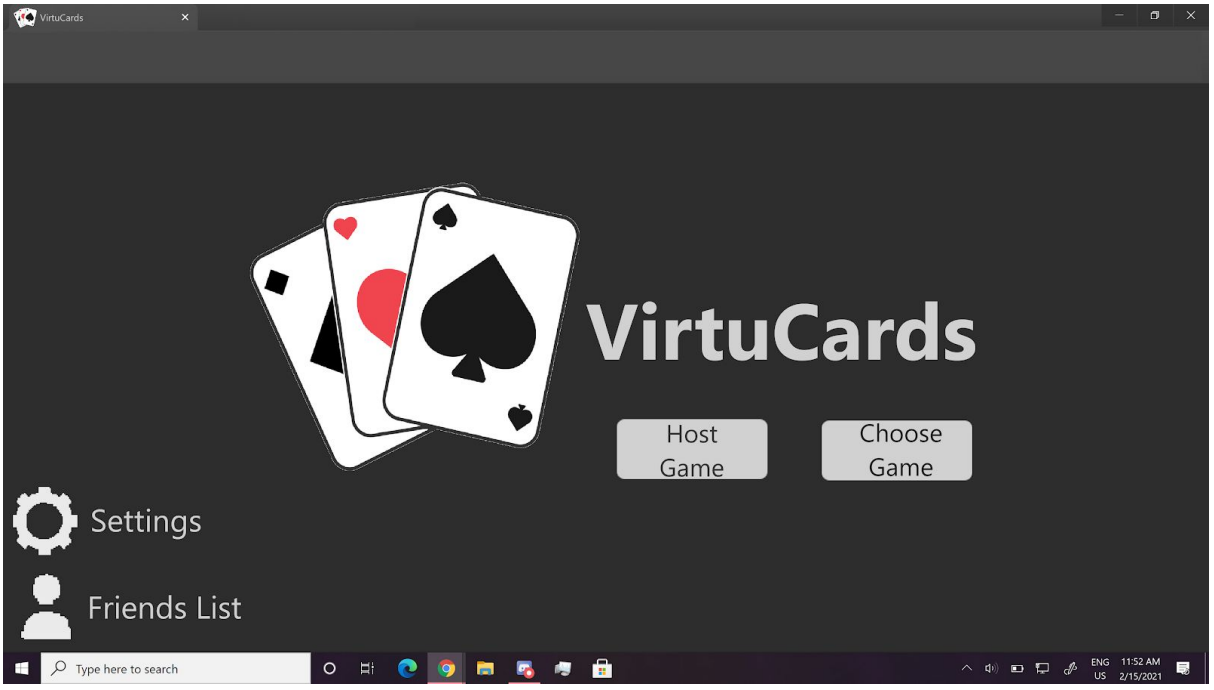


UI Mockup

Desktop Menu Interface before Login:

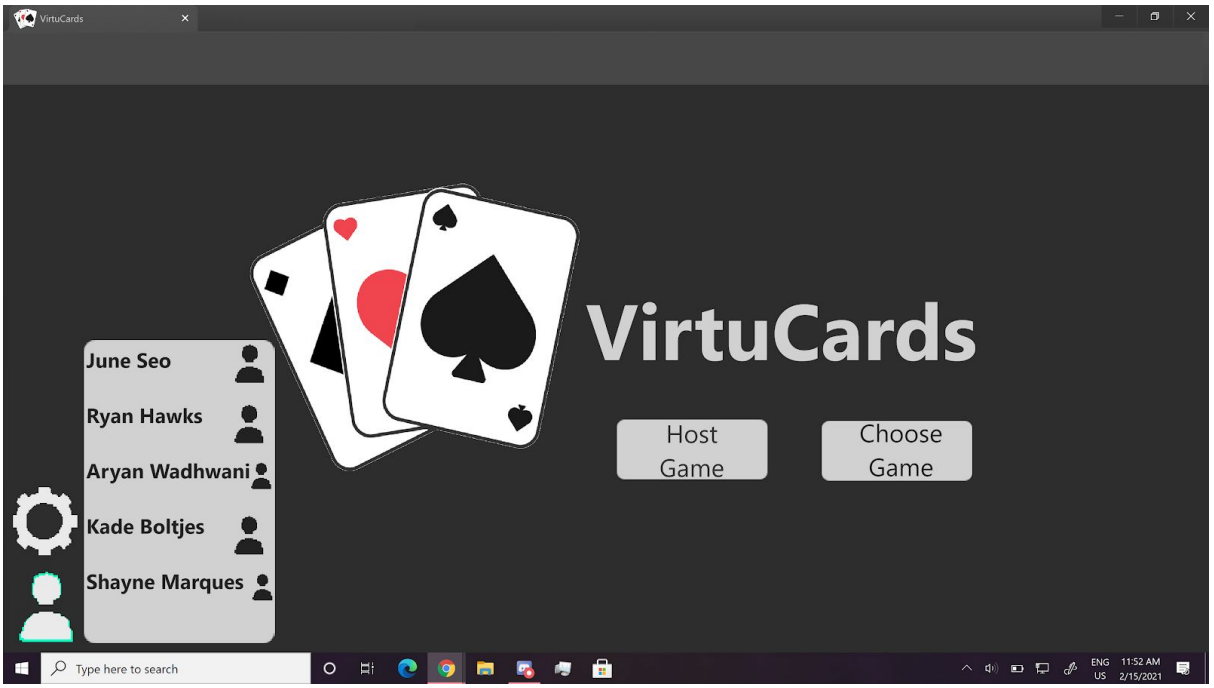


Desktop Menu Interface after Login:



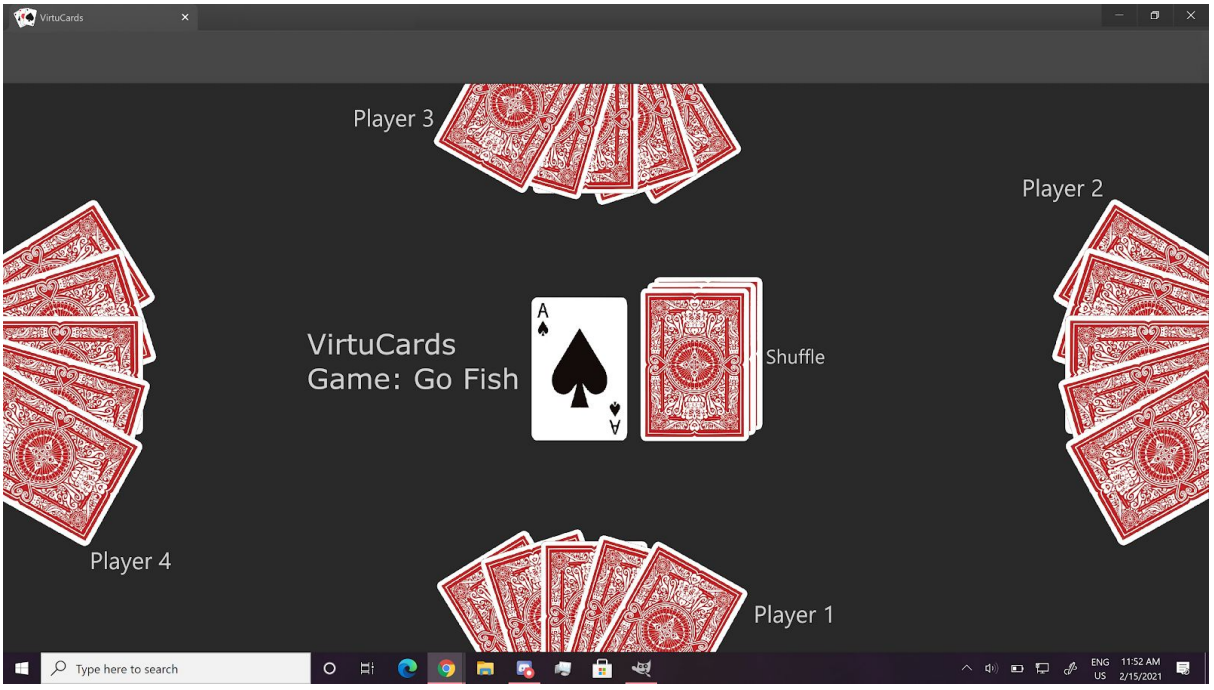
This UI mockup depicts the Menu seen by users on start-up of the application, after log-in. The Settings button allows users to customize their in-game cosmetic items such as the card sleeves and the background of the application. The Friends List button would display the list of accounts the user has marked as a friend. This information would be stored in Google Firebase, and accessed at the time of login.

Desktop Interface with Friends List open:



The person icons are placeholders for avatars that users can choose or upload images for.

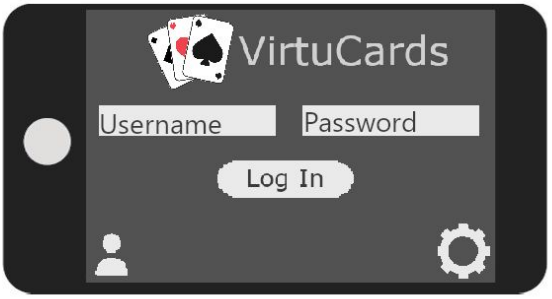
Desktop Interface during the Game session:



Using this web interface and a shared screen, all players can view the current game statistics, check whose turn it is and the music that is being played (if the playlist feature is implemented). The web interface will update in realtime along with the progression of the game.

Mobile Interface outlining joining a game:

Portrait Orientation
Depicting Menu on Mobile Device after Log-In.



Landscape Orientation
Depicting the screen on start-up of the app with Log-In Requirements

The UI mockup above displays some of the menu screens that users will encounter as they navigate to joining a game. The menus can be viewed in both Portrait or Landscape orientations. The Friends List button is depicted by a Person shaped icon in the bottom-left corner, and the Settings button is depicted by a Cog shaped icon in the bottom-right corner.

Mobile Interface during Game Session:

Portrait Orientation

Landscape Orientation

This interface will be visible to all players on their individual devices after a host creates their game lobby. The interface can be viewed using a Portrait orientation or a Landscape orientation. The Lobby Code is displayed on all users' devices in the event that if one user disconnects, the other members of the lobby can communicate it to them.