



Barbu Online

Cole Compton

Noah Johnson

Morgan Kaehr

Julian Ng

Matt Whipple

Index

- 1. Purpose**
 - a. Non-functional Requirements**
 - b. Functional Requirements**
- 2. Design Outline**
 - a. Components**
 - b. High-level Overview**
 - c. In-depth Overview**
- 3. Design Issues**
 - a. Non-functional Issues**
 - b. Functional Issues**
- 4. Design Details**
 - a. Class Design**
 - b. Class Interactions**
 - c. Sequence Diagrams**
 - d. UI Mockup**

(1.) Purpose

Barbu is a 4-player card game that is very popular amongst Bridge players. It involves players rotating turns selecting the rule set to be played within the game. There are 7 different sets a player can choose, and the strategy of the game is to choose the rule set that best fits the cards that players have been dealt. Current online platforms of this game are not very social, extremely fast paced, and difficult to play.

The purpose of this project is to provide a friendly, social environment that allows beginning players to learn and practice while maintaining an intense, competitive aspect of online play.

(1a.) Non-functional Requirements

1. Client: As a developer,
 - a. I would like the client to be used in a web browser on a computer.
 - b. I would like the client to provide a clean, user friendly experience.
2. Server: As a developer,
 - a. I would like the server to store user data into a database.
 - b. I would like the server to quickly respond to client requests.
 - c. I would like the server to only respond to authorized client requests.
3. Performance: As a developer,
 - a. I would like the application to run consistently without crashing.
 - b. I would like the backend server to be error-free.
 - c. I would like the frontend to be ready for any unexpected user inputs.
 - d. I would like the server to be able to support hundreds of simultaneous Barbu games.

(1b.) Functional Requirements

1. User account: As a user,
 - a. I would like to be able to register and create an account.

- b. I would like to be able to log in and edit my account preferences.
 - c. I would like to be able to check my game statistics.
 - d. I would like to compare my skills to those of others via a ranking system.
2. Social features: As a user,
- a. I would like to be able to create and easily access a list of friends.
 - b. I would like to be able to chat with my friends both during and outside of a game.
3. Gameplay requirements: As a player,
- a. I would like to be able to see my wins and losses.
 - b. I would like to be able to join both private lobbies and public games.
 - c. I would like to be able to see a beginner tutorial on how Barbu is played.
 - d. I would like to be able to play competitively against other like-minded players.
 - e. I would like to be able to pick the game type when it is my deal.
 - f. I would like to be able to double other players before a hand is played.
 - g. if I attempt to make a forbidden play, I would like to receive a message that explains why I can't make the move that I want to.
 - h. (If time allows) I would like to be able to play a single player mode that challenges me to get better.
4. Game interaction requirements: As a player,
- a. I would like to see a visual list of the cards in my hand.
 - b. I would like to be able to easily select a card when it is my turn.
 - c. I would like other players' plays to be displayed clearly during the game.
 - d. I would like the score to be calculated automatically at the end of each game.

(2.) Design Outline

We are choosing to use a client-server-database model, as we feel like this is the simplest way to abstract the needs of our application. The client will be a relatively light client. This will be optimal for the user as this is a web application, meaning they have to reload the client every time they want to use it. The client will be responsible for displaying the information relayed by the server to the user in a nice manner. It will also promote easy browsing through all of the application's features. The server will be responsible for handling the game-state. Whenever a user makes a move through the client, for instance, this information will be sent to the server, and the server will calculate and display the outcome of each move. The server will also be responsible for communicating with our database. Whenever the user visits a profile, for instance, the server will need to poll for the appropriate profile data from the database. Similarly, the server will update the database with the results of each game played. A full breakdown of the model is presented below:

(2a.) Components

1. Client
 - a. During gameplay, the client will:
 - i. Send data to the server regarding the choice of card, game, doubles, and other user decisions.
 - ii. Receive data regarding the cards in the user's hand, plays by other players, updates to the score, and other changes to the current game state.
 - iii. Change the user interface accordingly.
 - b. During browsing of the site, the client will:
 - i. Send HTTP requests to the server
 - ii. Update the user interface accordingly

2. Server

- a. During gameplay, the server will:
 - i. Spawn a new thread for each lobby/game
 - ii. At the beginning of a hand, “shuffle” the cards and send each client the cards in their hand
 - iii. During the middle of the hand, receive data from clients about what card they want to play
 - iv. Upon receiving information about the user’s play, send that information to all the clients so that they can update their user interfaces accordingly
 - v. At the end of the hand, calculate the scores and:
 1. Write the updated scores to the database
 2. Send the updated scores to the clients
 - vi. At the end of the game:
 1. Update the statistics in the database for each player in the game
 2. Redirect clients back to homepage
- b. For browsing the site, the server will:
 - i. Receive HTTP requests from clients
 - ii. Send clients the corresponding HTTP response
 - iii. When a user adds or removes a friend, update the database accordingly
 - iv. When a request is sent to view game history, record, etc. query the database to find the relevant information for the user

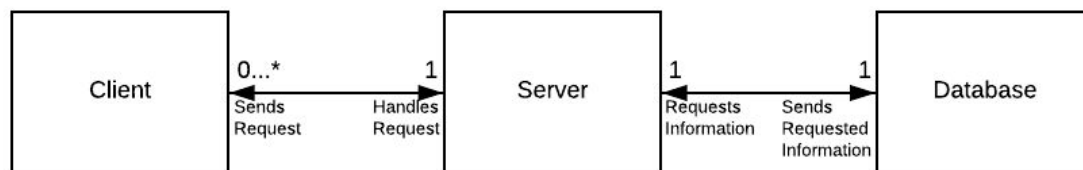
3. Database

- a. The database will be structured as a relational database and will store information about each user, including
 - i. Game record (W-L)
 - ii. Average score

- iii. High/low score
 - iv. Friends
- b. When a client requests this information (by the user visiting their profile), this data will be sent to the server, which will pass it along to the client.

(2b.) High-level Overview

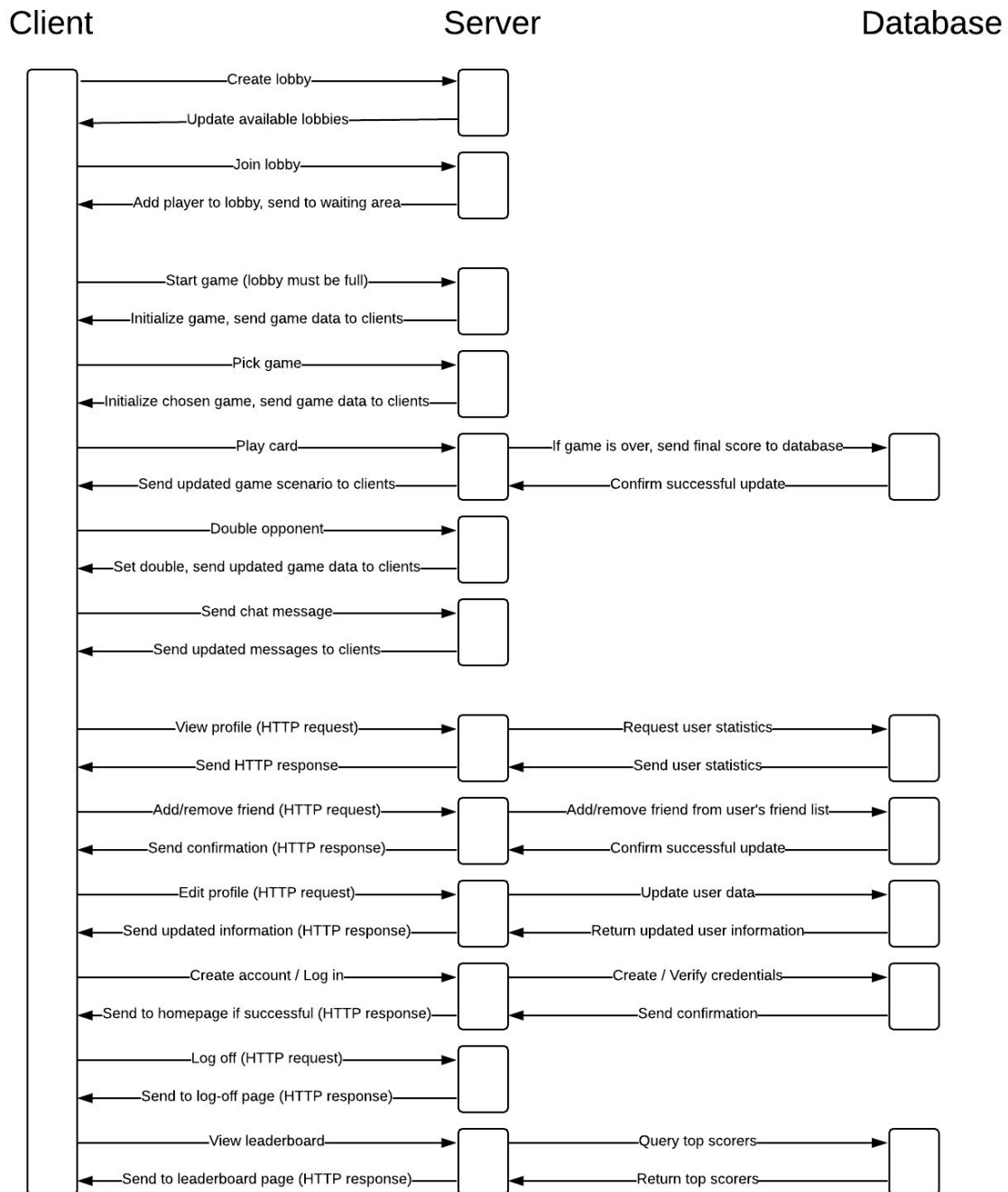
We are making a web application which will be able to handle multiple clients sending requests to the server whether that be game moves, requests to see statistics, profile information, or managing who is on their friends list. The server will then request any required information from the database that is stored. This could be user account information, leaderboard statistics, or even win-loss numbers.



(2c.) In-depth Overview

The sequence diagram below illustrates the interactions between the client, server, and database. Client actions can be divided into three categories: pre-game, gameplay, and site browsing. For the pre-game and gameplay categories, the information is stored in the server. Clients send information when it is their turn to make a play or some other action, and the server sends the updated game scenario to all the clients in the game. The database is involved only in updating user statistics at the completion of the game. During normal site browsing, the interaction between the client and server is in the form of HTTP requests and responses. The database is needed for many of these actions in order to display relevant information - for example, when a

user views their profile, the server needs to query the database in order to retrieve the information that goes on the user's page.



(3.) Design Issues

(3a.) Functional Issues:

1. *Should users need an account to play the game?*

Option 1: Yes, users need to be logged in to play a game.

Option 2: No, users can play a game as a guest.

Decision: We decided that Option 1 was most in line with our goals for the application. Since one of the main goals of the app is for a community based experience, we felt that it was a must for users' games to be attached to their profile, and thus that an account would be necessary to play a game.

Furthermore, if a guest is in a game with other registered users, the guest could lose the game without facing any consequences (i.e. they wouldn't have a loss saved to their profile and lose a spot on the leaderboards).

2. *How should competitive players' rankings be decided?*

Option 1: Use the average score of the player.

Option 2: Use the players' wins and losses.

Option 3: Use an elo system.

Decision: We decided to use the players' average score, Option 1. Wins and losses seem to not be as indicative of skill. For instance, a beginning player who really enjoys Barbu and plays a lot might accumulate more wins than an advanced player who plays only occasionally. We also considered using an elo system, but we thought that it wouldn't suit the game as well as average score would, especially considering this is a 4 person game. Users will still be able to access player leaderboards and see such information as wins, losses, win percentage, and average score. Users will also have the ability to sort by any of these categories if they wish, but the default ranking system will use Option 1 as the determining factor.

3. *Should players be able to login on two or more platforms simultaneously?*

Option 1: Prevent an account that is already logged in from logging in on a different platform.

Option 2: Log out the account that is already logged in.

Option 3: Allow one account to be logged in on multiple platforms.

Decision: We decided to go with option 2 because there could be problems if we didn't let someone log in if they were logged in elsewhere. The person might have left a tab open on another device that they cannot access and essentially locking out their account. Having one account logged in multiple times would be a headache to deal with and could lead to unnecessary issues.

4. *How should we handle players leaving mid-game?*

Option 1: End the game, return remaining players to the home page, and punish the player who left.

Option 2: Have an A.I. player take place of the player that left the game

Decision: We decided that option 1 would work best with the amount of time we have for the project. Adding an A.I. player would take a lot of our time out of developing other important features. Option 1 would take significantly less time to develop. Players leaving mid-match could face a penalty (such as a temporary ban or a loss on their record) for leaving while the remaining players would be unaffected.

(3b.) Non-functional Issues:

1. *Should game information be stored in the server or in the database?*

Option 1: Store the information for each live game in the database.

Option 2: Store the information for each live game in the server.

Decision: Ultimately we decided Option 2 was more practical. We felt that the server storing game information (such as card positions, running score, etc)

would run faster than if the server had to update the database after every turn. Furthermore, this significantly reduces the complexity of the database, as it will only need to store user profile information. As a trade off, the code for the server will be more complex as it has to manage all of the game information. The server will also need to use more memory for each game as a result. This may impact efficiency if an exceptional amount of games were taking place at once, since each game will require its own thread/process.

2. *What language will we use to implement our backend?*

Option 1: C#

Option 2: Python

Option 3: Java

Decision: We decided to go with Java for our implementation of the backend. It's a language we are all familiar and comfortable with. On top of that Java has functions already made for cards and classes that will make development much smoother and more efficient.

3. *What language will we use to implement our frontend?*

Option 1: HTML and Javascript

Option 2: React

Option 3: Angular

Decision: We decided to go with React since it is an easy framework to learn and also has great documentation on how to use it. React can have multiple components and each can be controlled separately when we need to use them.

4. *What database will we use?*

Option 1: MySQL

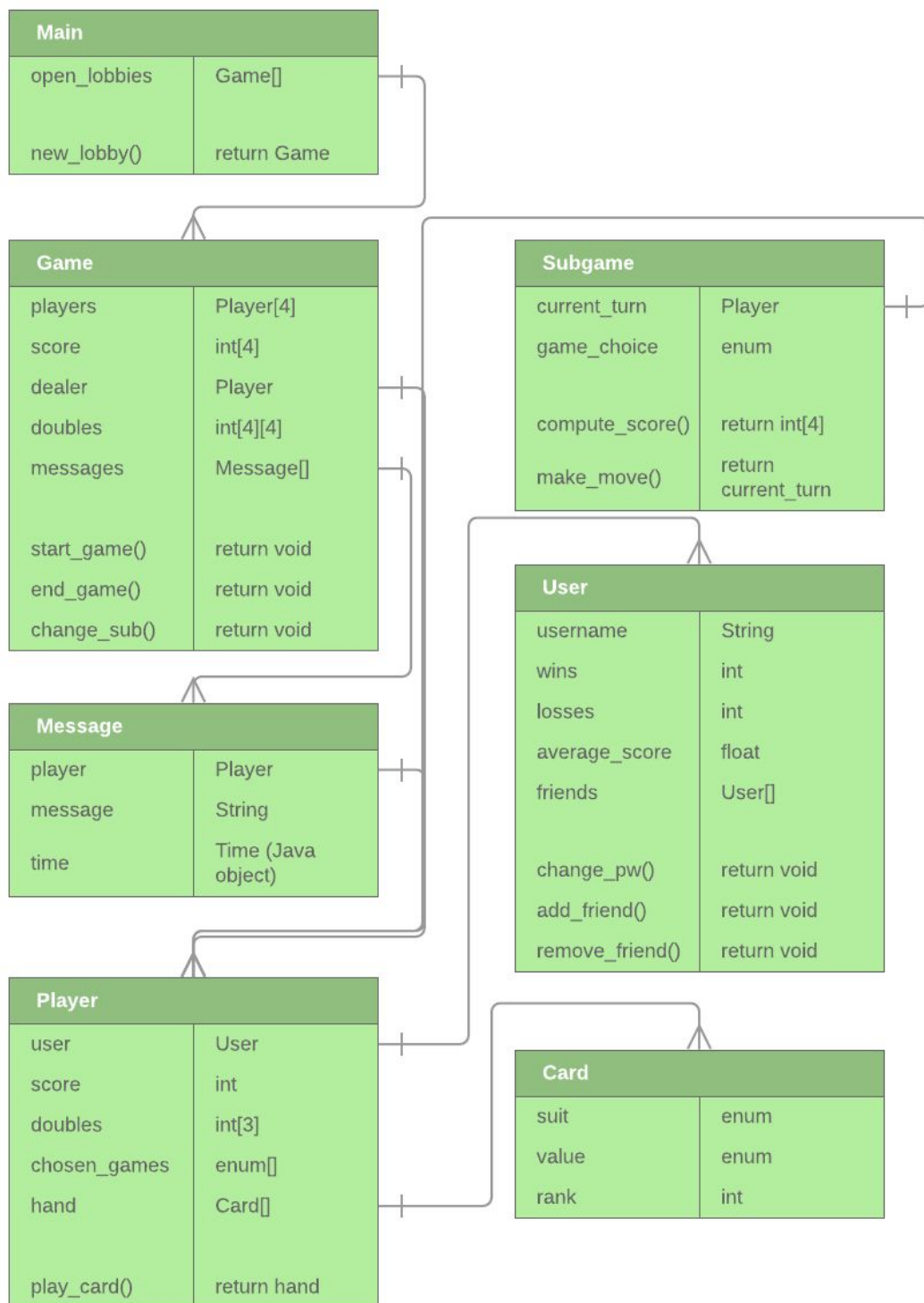
Option 2: Firebase

Option 3: MongoDB

Decision: We have decided to go with Firebase for our database. Firebase provides storage, authentication, and hosting. It's also free and is provided with tutorials/guides and examples.

(4.) Design Details

(4a.) Class Design



Classes list:

- Main
 - The main class will be responsible for facilitating the operation of the system.
 - It will run in its own thread on the server and handle things like initializing new games and website navigation.
 - It will also maintain a list of open lobbies that players who are looking for a game can join.
- Game
 - The game class will store some of the higher-level data pertaining to a particular Barbu game.
 - Some of the data stored in this class will be game settings, players in the lobby, score, chat messages, dealer, and doubles.
 - There will be one instance of this for every game being played, and each instance will also require its own thread.
 - This class will also include helper methods that set up games and also clean up after a game concludes, as well as `change_sub()`, which changes the sub-game that is being played.
- Subgame
 - The subgame class will represent the state of the subgame currently being played.
 - The most important thing it will store is an enum called `game_choice` that indicates which of the possible sub-games in Barbu is currently being played.
 - It will also store things like `current_turn`, and provide the methods `make_move` and `compute_score()`.
 - There will generally be one instance of one of the 7 subgame sub-classes for each Game object. The Game object will always keep a reference to the current subgame being played.
- Card
 - The card class will just store information about a card, and there will be one instance of it per card in the game (52 total per game).
 - The specific information stored will include the card's rank, suit, and value
 - Value differs from rank in that a card's rank may be "Jack" but the useful value would be 11.
- User
 - The user class will store information concerning each user.
 - Information stored will include username, average score and wins/losses, and friends.

- It will also include methods to add/remove friends and change the user's password.
- **Player**
 - The player class will be used to store information pertinent to an individual during the course of a game.
 - Since Barbu is a 4-player game, there will be 4 Player instances per Game instance.
 - This class will contain information like the player's hand (which will be an array of Card objects), the player's score, the subgames they have played (chosen_games), their doubles, and a reference to the corresponding User object.
 - It will also include a play_card() method which runs when a user selects the card they wish to play.
- **Message**
 - The message class will hold information about messages sent in the game's chat feature. It will include fields such as a reference to the Player who sent the message, message content, and time of message.

(4b.) Class Interactions

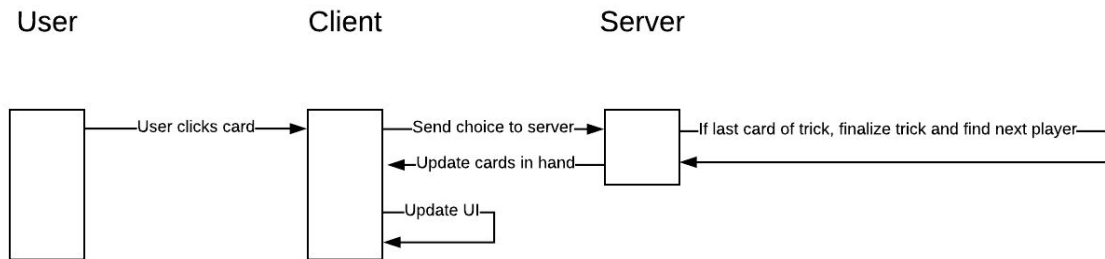
The Main class is where everything is started from an interactions standpoint. It will create User instances whenever users log into their accounts. It will also create Game instances whenever a User creates a new lobby. When this happens, the Main class will create a new thread for that particular Game instance, starting at the Game.start_game() method. It will also pass User objects along to the Game as users join the public lobbies.

When the game actually starts, the Game class will take the 4 Users it has and create corresponding Player class instances. After the dealer picks out the subgame that he/she wishes to play, the Game instance will create an instance of the appropriate subgame class. The Game object will also maintain a reference to an array of Message objects, which will represent the game's chat.

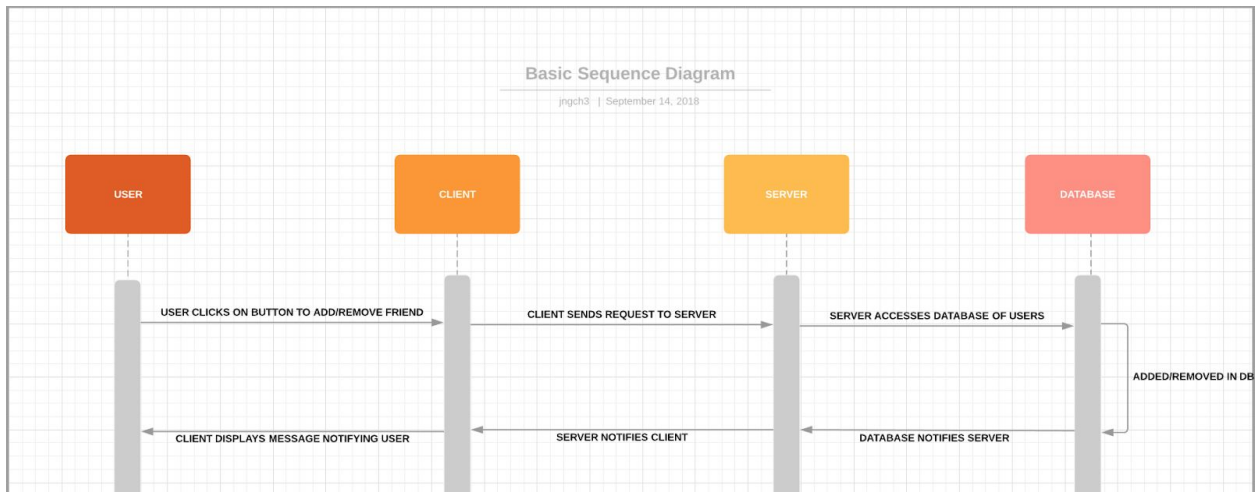
The Player objects will also hold a sorted array of Card objects representing the player's hand.

(4c.) Sequence Diagrams

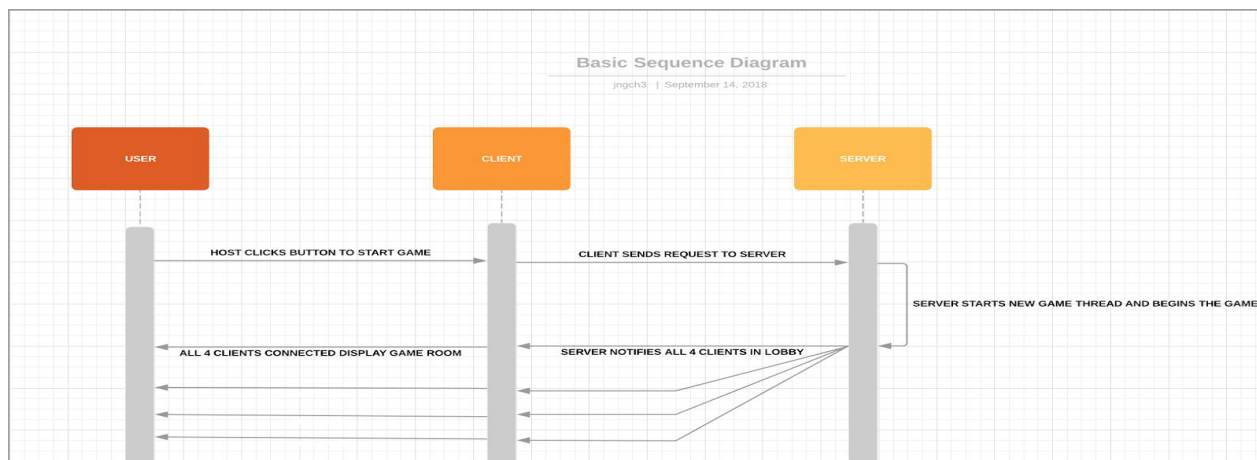
1. Sequence diagram for playing a card



2. Sequence diagram for adding/removing friends



3. Sequence diagram for starting a game

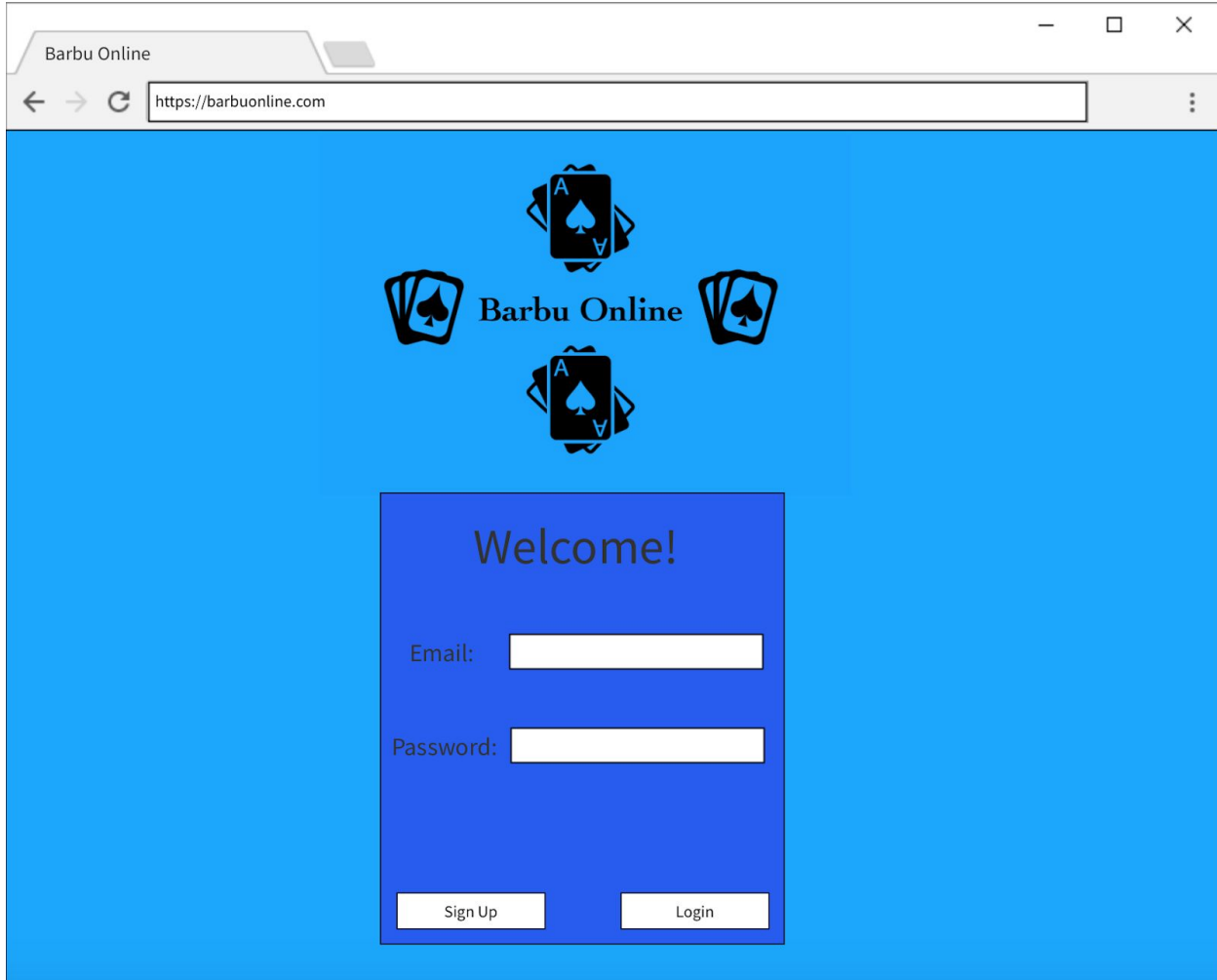


4. Sequence diagram for sending global chat message



(4d.) UI Mockups

Login Page:



The image shows a web browser window with the title "Barbu Online" and the URL "https://barbuonline.com". The page has a blue background. In the center, there is a logo consisting of four playing cards (Ace of Spades, King of Spades, Queen of Spades, and Jack of Spades) arranged in a square, with the text "Barbu Online" in the middle. Below the logo, there is a blue rectangular box containing the text "Welcome!". Inside this box, there are two input fields: "Email:" and "Password:". Below the input fields, there are two buttons: "Sign Up" and "Login".

Barbu Online

https://barbuonline.com

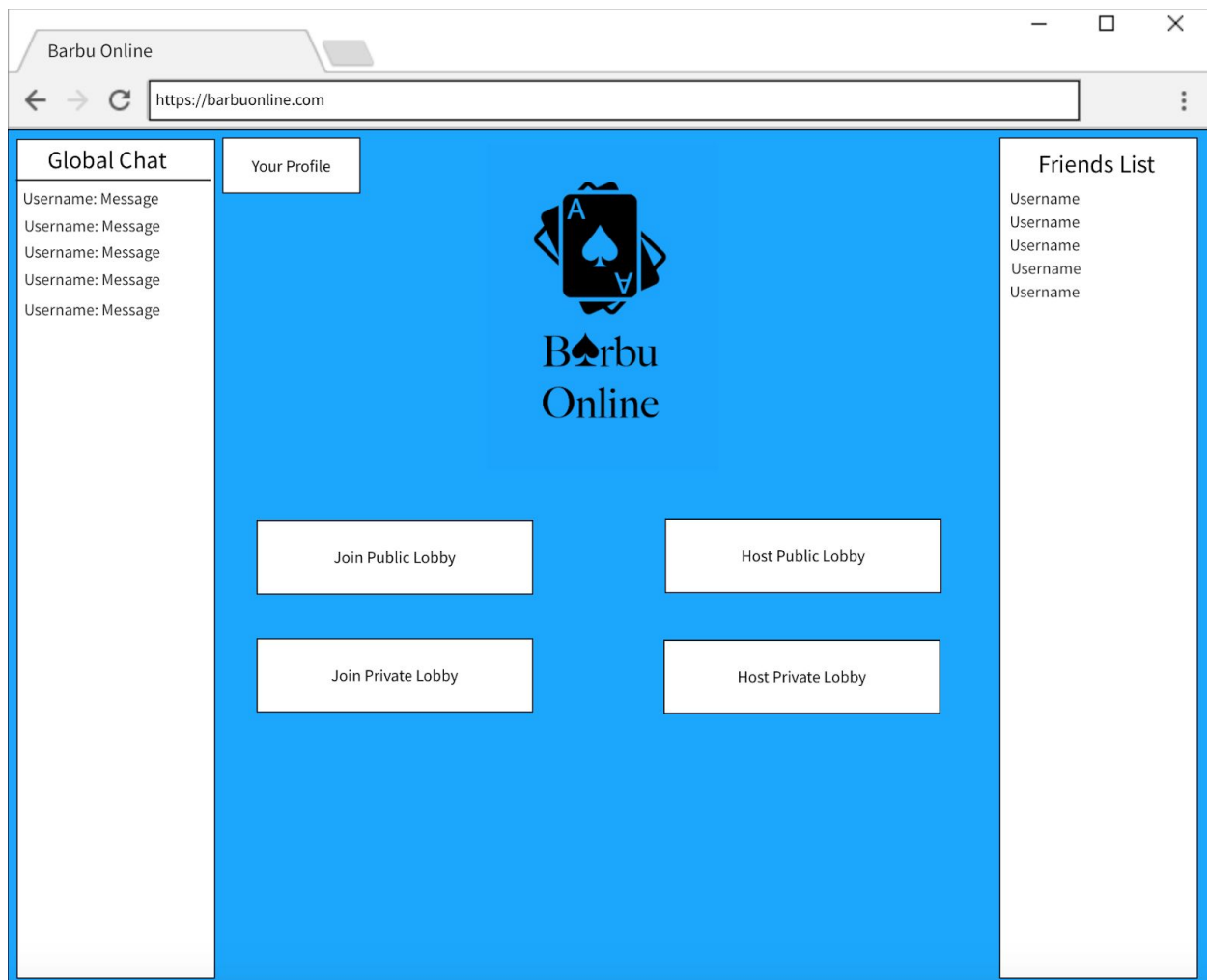
Welcome!

Email:

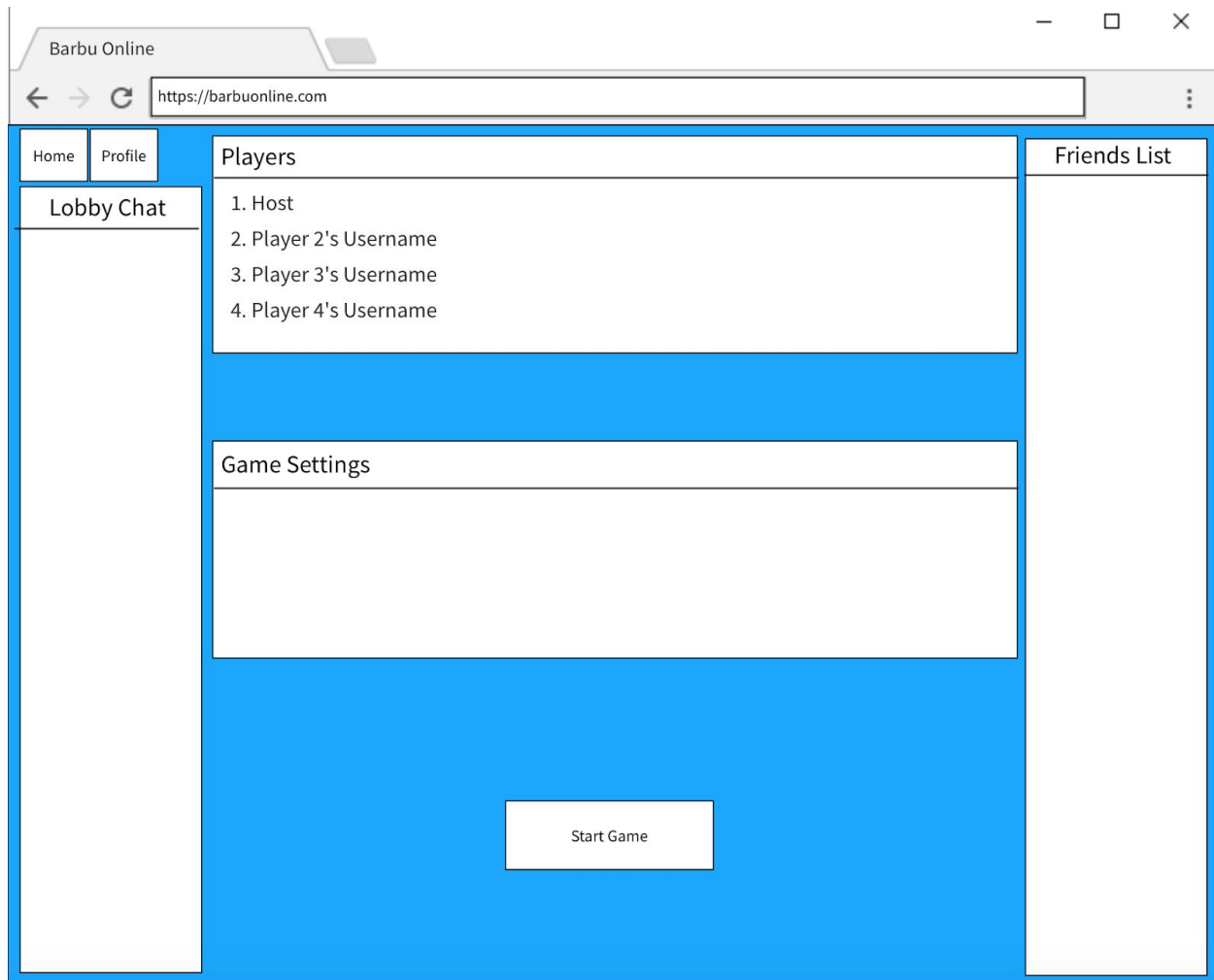
Password:

Sign Up Login

Homepage:



Lobby page (Host's perspective)



Lobby Page (Joining player's perspective)

