

wordblast.io

CS307: Design Document

Team 19

Emilio Barradas · Andrew Zheng · David Long · Inbang Seo

Purpose

21% of American adults are illiterate/functionally illiterate. Our goal is to build “wordblast.io”, a turn-based word typing game where players must type a word that contains a given word combination. We believe that it is a fun and simple word game to develop our citizen’s spelling, reading, and typing abilities with a high potential to be popular.

Existing implementations do not use the easy-to-use .io game approach, and also have limited functionalities. Some examples are that users are not able to choose which type of words (i.e. verbs, nouns, adjectives) to place into the word pool, the definitions of submitted words are not given, there are no leaderboards, and there is no XP/ranking system. We will build this implementation.

Functional Requirements

1. Players can log in, join in, and start a game without issues.

- I would like a username input form, so that I can login anonymously and immediately play the game.
- I would like the option to change my name while in the game lobby.
- I would like to enter a game lobby before the game starts so I can see who is ready to play.
- I would like the game to start once everyone in the game lobby has readied up.
- In a private lobby, I would like to have a custom URL to give to my friends so they can easily join my game.
- I would like to be able to create an account so that I can track my progress.

2. The application has a straightforward UI.

- I would like to be able to view my game statistics and see a summary of my games, including my average word length, most used words, and average typing speed.
- I would like to be able to see my user level.
- I would like to be able to view the leaderboards page which hosts a list of the top performing players.
- I would like to be able to see the number of players currently online.
- I would like to chat with other players in the same game.

3. Players can customize the game.

- I would like to have a tutorial/walkthrough option.
- I would like to choose background themes.
- I would like to be able to customize the settings of a game.
- I would like to have background music and sound effects.

- I would like the option to set my custom game as public, so that random players can join.
- I would like to be able to view the game settings while in game.
- I would like the option to toggle a profanity filter in the actual game and in the chatroom.
- I would like to have the ability to toggle between light and dark mode.

4. The gameplay should be balanced and fair.

- I would like a “play with friends” option so that I can join a game lobby with friends.
- I would like to have the bomb indicating whose turn it is, by pointing it to that player.
- I would like the specified letter combination to be highlighted so that it is easier to visualize.
- I would like to lose a life whenever I am unable to come up with a valid word.
- I would like to gain a life when I use all letters of the alphabet at least once.
- I would like a visual representation of which letters I have used so that I can track my progress towards an extra life.
- I would like to be shown the definition of a word after it has been submitted.
- I would like a single player mode so that I can hone my skills.
- I would like to be able to play in languages other than English.
- I would like to gain more XP after each successful word I type.
- I would like copy/pasting to be disabled so that players do not cheat.
- I would like to be able to input custom words into the word pool.
- I would like the option to play with players who are the same level as me.

Non Functional Requirements

1. Performance

- I would like web requests to our frontend and backend to be cached within the user's browser to make load times much quicker.
- I would like to get response time below 200ms.
- I would like the website to be online 24/7 and to only go down for less than 10 minutes per week.
- I would like the website to be able to hold 500 concurrent connections.

2. Appearance

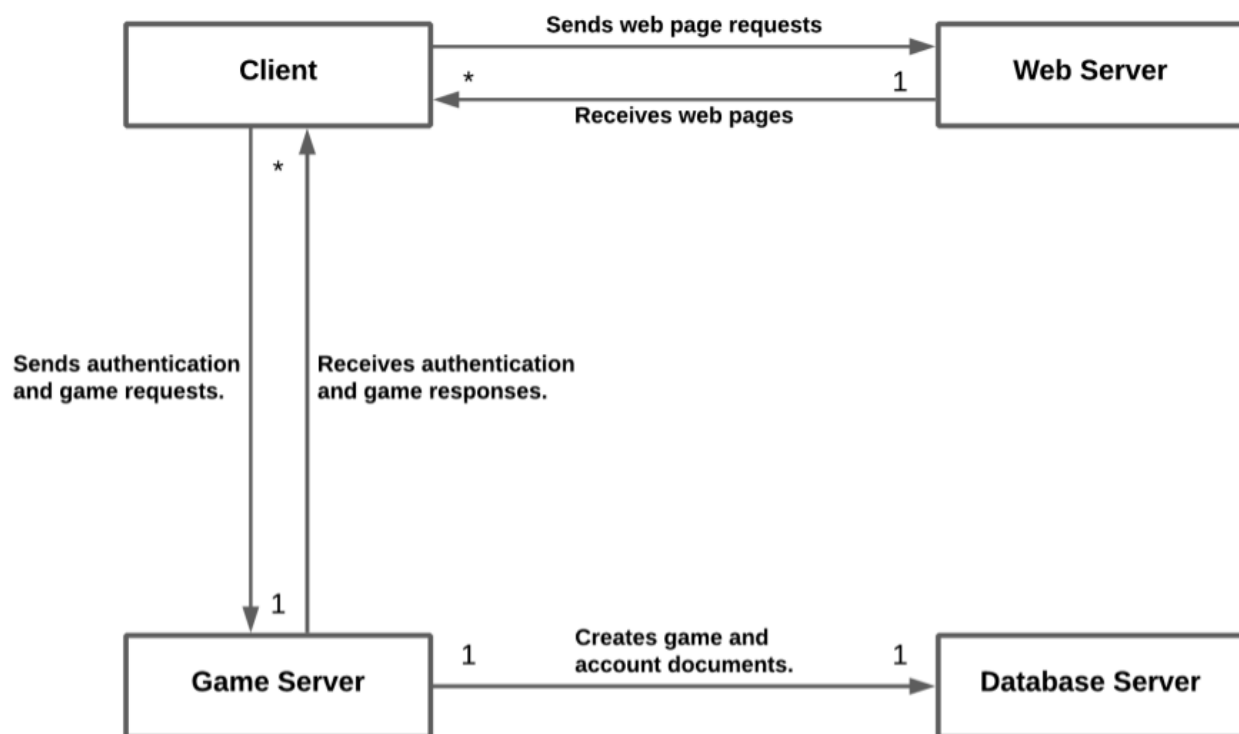
- I would like the UI to be easy to navigate, and the game to be easy to understand.
- I would like as few clicks as possible to enter a game.

3. Usability

- I would like the game to support players of all ages.

Design Outline

Using the client-server model, the application will be split up into two high-level components, the frontend and backend. The frontend is the user interface of our application, which will be represented using a website. On the backend, we have three sub-components, the web server, game server, and database server. The web server will handle web page requests from the frontend clients. The game server will handle authentication/session management and will host on-going games. The database server will store account information, player statistics, and data on completed games. The implementation of these components will follow the model-view-controller pattern, where the frontend is the view, and classes within the backend will correspond to the model and controller.



Client

- The client sends requests to the web server for the user interface of our application.
- The client sends account creation/login requests to the game server.
- The client opens a connection to the game socket on the game server.
- The client modifies its application state based on the responses from the requests to the backend.

Web Server

- The web server receives HTTP web page requests from the client.
- The web server responds with the requested web page from the HTTP requests.

Game Server

- The game server has various modules, some of which include, the authentication module, word module, game module, and database module.
- The authentication module will create accounts and sessions, handle logins, and validate sessions.
- The word module will parse word data (words and whether they are verbs, adjectives, nouns, etc.) and provide an interface for the game module to randomly select words. This module will support multiple languages.
- The game module will create games and modify their data as game events occur.
- The database module will create and modify account and game documents.

Database Server

- The database server will be running MongoDB.
- The game server will create account and game documents on the database server.
- The database server will have various MongoDB views corresponding to statistical data, such as the leaderboard page.

Design Issues

Functional Issues

1. What form of login should be available to our users?
 - a. Sign in with third-party accounts (Google, Facebook, Twitter, etc.)
 - b. Email and password
 - c. Anonymous login (no password)
 - d. Phone number
 - e. A mixture of the above

Our pick: A mixture of the above (a, b, and c)

Justification: We decided that it would be most convenient for players to be able to use their email or a third-party account to create an account for our game to track statistics. If players do not want to sign up and simply want to play, they can enter a username and immediately jump into a game.

2. What game mechanics should contribute towards player XP gain?
 - a. The overall length of the match
 - b. End result of a match (win or loss)
 - c. Quality of words entered mid-match
 - d. All of the above

Our pick: All of the above

Justification: Players should be rewarded according to their performance in the game. The better a player performs in a match, the higher their XP gain will be. Players will be rewarded with more XP based on whether they won or lost, and how many lives they maintained if they won. Higher quality words will result in more experience than lower quality words. However, players will also get experience from their time invested in the game.

3. How should we progress the difficulty after each round?
 - a. Shorten the amount of time players are given to type a word
 - b. Present more difficult letter combinations
 - c. Increase the required letter count for a valid word
 - d. A mixture of the above

Our pick: A mixture of the above

Justification: We chose to use all the options listed to create more ways to increase difficulty. Shortening the amount of time and increasing the required letter count will require a player to enter words faster while presenting more difficult letter combinations will require the player to consider uncommon words and finger movements.

4. What statistics will be continuously tracked and displayed starting from account creation?
 - a. Total games
 - b. Average word length
 - c. Average words per minute
 - d. Most frequently used words
 - e. Account level
 - f. ELO rating
 - g. All of the above

Our pick: All of the above

Justification: Total games and account level are general statistics that represent how long a player has been playing while the other statistics show a player's traits and skill level.

5. How will private lobbies be shared and/or displayed to players?
- a. All private lobbies will be shown and players will be able to filter by username or lobby ID
 - b. Users will enter a lobby ID and password to attempt to join
 - c. Lobby hosts will share an invite link for players to click and join

Our pick: C, a shareable invite link

Justification: We plan to allow lobby hosts to copy and share a link for players to use as an invite for simplicity. We found no need to show all lobbies to players or use an ID and password system.

Non-Functional Issues

6. What frontend framework should be used?
- a. React
 - b. Next.js
 - c. Svelte
 - d. Vue.js

Our pick: Next.js

Justification: Next.js is a framework which is built on React. React is a javascript library that makes creating and debugging interactive websites easier than using vanilla javascript. Next.js provides all of what React provides, and much more, such as file-system routing, fast refresh live-editing, Typescript support, code-splitting, and building. Most of these features we were going to reach out for anyway, so it's best for us to start with them from the get-go.

7. What backend framework should be used?

- a. Spring w/ Java
- b. Django w/ Python
- c. NestJS w/ Node.js
- d. Express w/ Node.js

Our pick: Spring w/ Java

Justification: Spring is a Java framework for building Java applications. The framework has multiple modules which aid in setting up HTTP Rest APIs, web sockets, and database integration. Most of the other options do not provide this type of foundation that we can work off of.

8. What database technology should be used?

- a. MySQL
- b. PostgreSQL
- c. MariaDB
- d. MongoDB

Our pick: MongoDB

Justification: MongoDB stores most of its data in RAM, which allows for quick performance when executing queries. This makes MongoDB much quicker than other relational databases. In addition, it uses a simple query syntax that is much easier to understand than SQL.

9. What infrastructure host should be used?

- a. Google Cloud Platform
- b. Amazon Web Services
- c. Microsoft Azure
- d. Heroku

Our pick: Google Cloud Platform

Justification: Our team has the most experience using Google Cloud platform and Amazon Web Services, therefore we decided to pick from these two. The prices for both providers were practically equivalent, therefore it came down

to the fact that Google Cloud Provider has an easier to use/read UI and documentation.

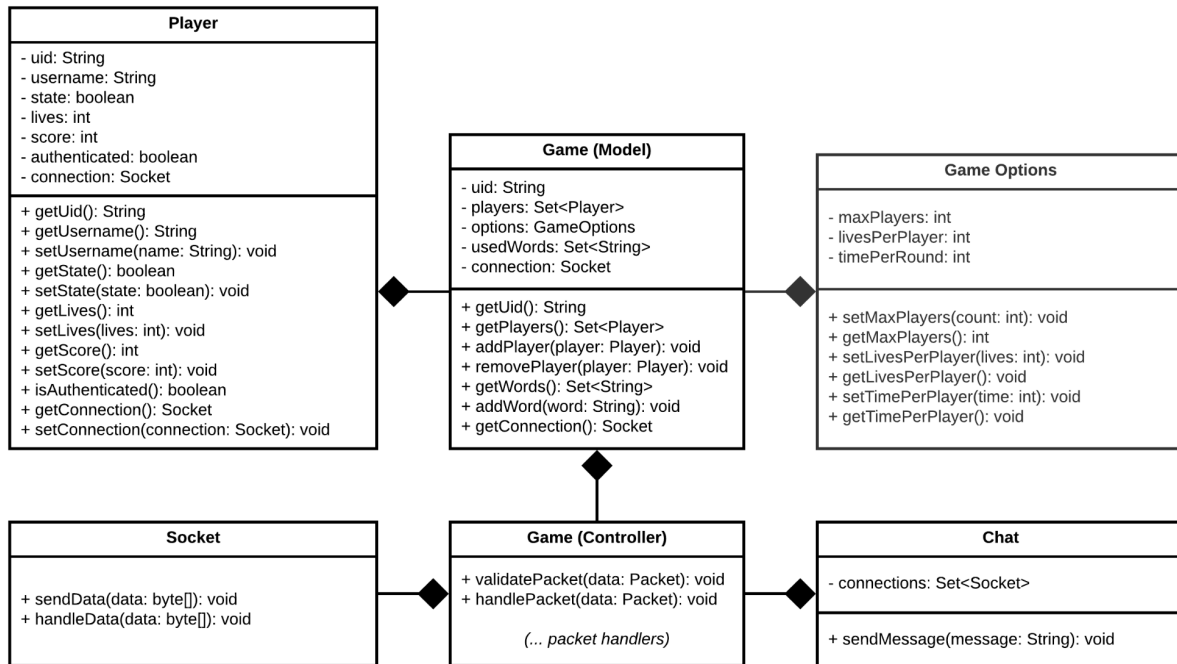
10. What type of service management should be used?

- a. Docker on a managed kubernetes service
- b. Docker on a VM instance
- c. Serverless functions
- d. Firebase

Our pick: Docker on a VM instance

Justification: The providers which host firebase, or support serverless functions, have a limited variety of languages and frameworks which can be used, so instead we chose to use Docker to encapsulate our backend. This gives us greater flexibility on what backend operating systems, languages, and frameworks we can use. This option also allows us to be platform independent, meaning we can switch from one provider to another with little friction, which the other options would have not. This option is also the most cost-effective as we can host all of our services on the same VM instance.

Design Details



Player

- When a user joins a game, a player object is created for the game using the player socket connection.
- If the user is authenticated, the object's authenticated field will be true. The uid field and username field will be set to the uid and username of the user, respectively.
- Otherwise, the authenticated field will be false and the uid and username field will be randomly generated.
- The default values will be used for the score and lives field.
- This player object will be stored inside the game object.

Game (Model)

- A game object will be created whenever a user joins a random game, and no game is available, and when a user is creating a lobby for their friend group.
- This newly created game object is added to the game manager.

- Whenever a user joins a game, their player object is added to the game object.
- Once a word has been used inside the game, that word is added to the used words field, so that it no longer is used again within the same game.
- A web socket will be created for this game, and the game object's connection field is set to it.

Game Options

- Every game object will have a game options object associated with it. The game options object will store the different settings of the game.
- On the frontend, the owner of the game is allowed to change the settings of the game within the lobby. The game options object should reflect these settings.

Game (Controller)

- Each game controller object will have a game model object associated with it.
- Game socket packets will be sent into and from the game controller.
- When a packet is received, the game controller will analyze its contents and depending on the context, it may validate the packet.
- Once the packet has been validated, if it was necessary, the packet will then be sent into some handler within the game controller.
- These handlers will directly interact with the game model object, and/or may send packets to the game socket.

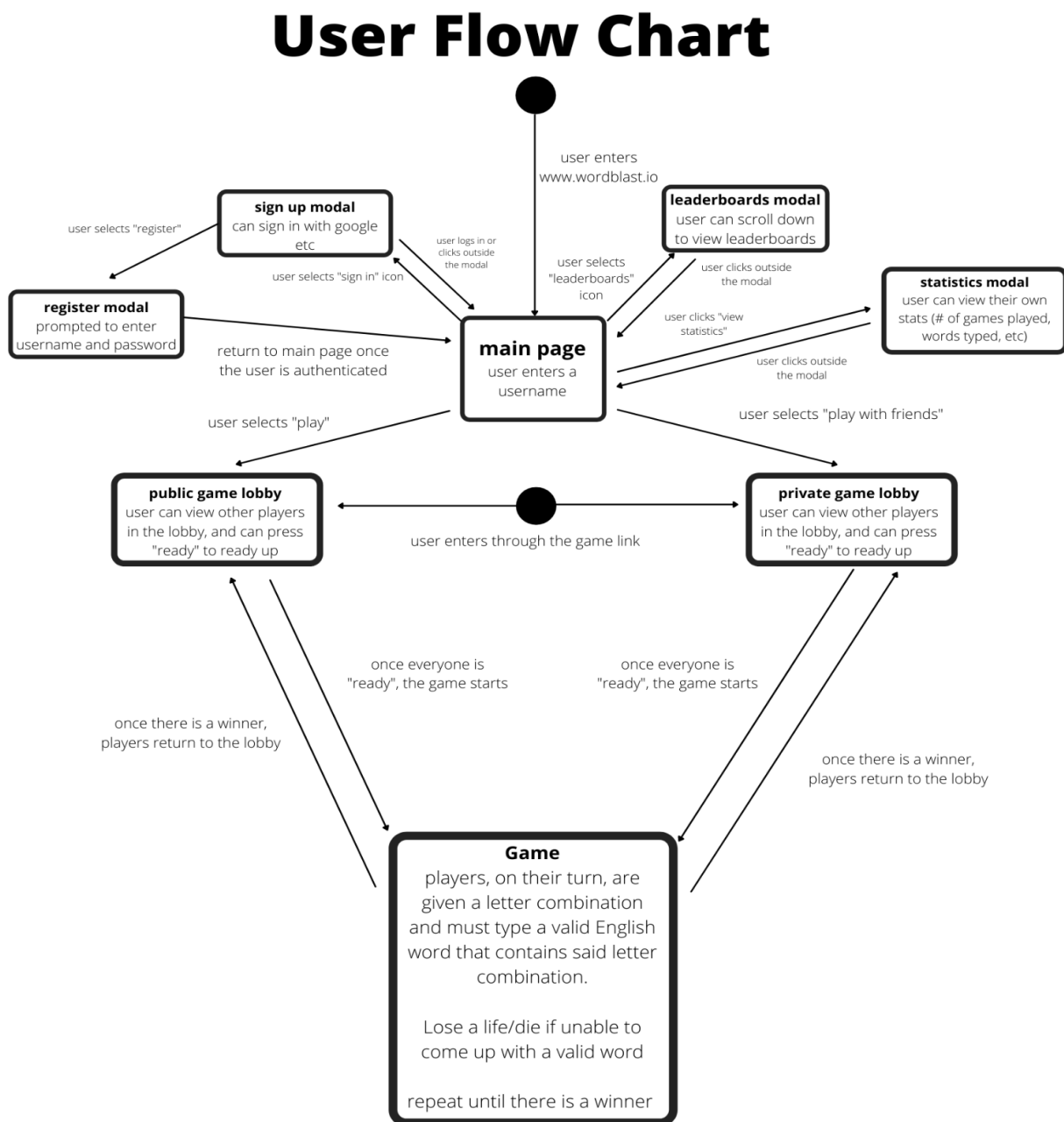
Chat

- Every game controller will have a chat object associated with it.
- The game controller will include some handlers that will interact with the game's chat system.
- Whenever a player sends a message to the chatbox, the packet will be sent to the socket, into the game controller, and then the send message method will be called in the chat object.

Socket

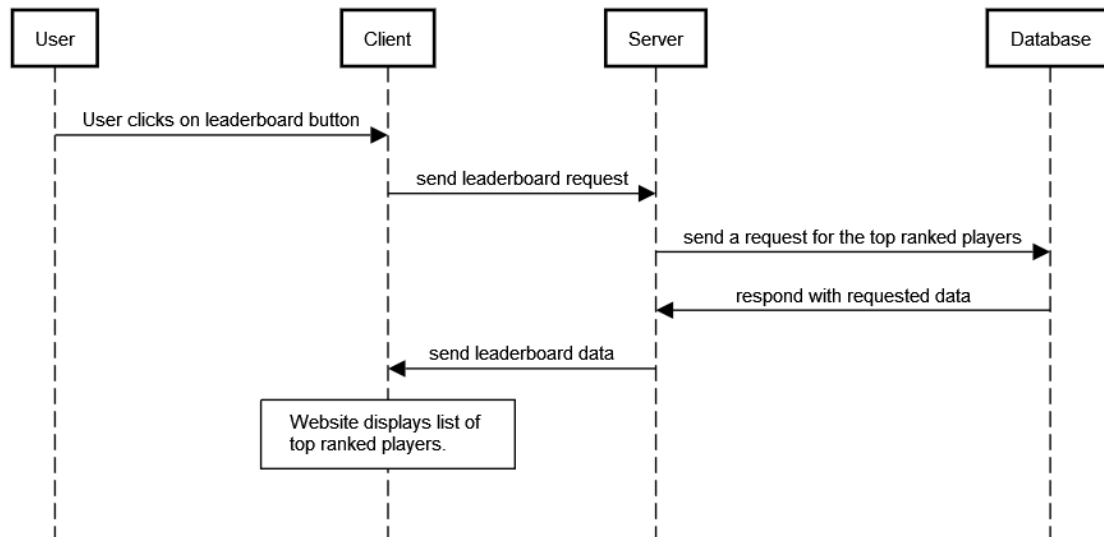
- The game socket object will be created whenever a game is created.
- The socket object will read incoming byte data and deserialize it into packets following a standard which we will form later in development.
- Outgoing packets will be serialized into byte data before sending it out to the clients.
- All game events will be sent through the socket in the form of packets.

Activity/State Diagram

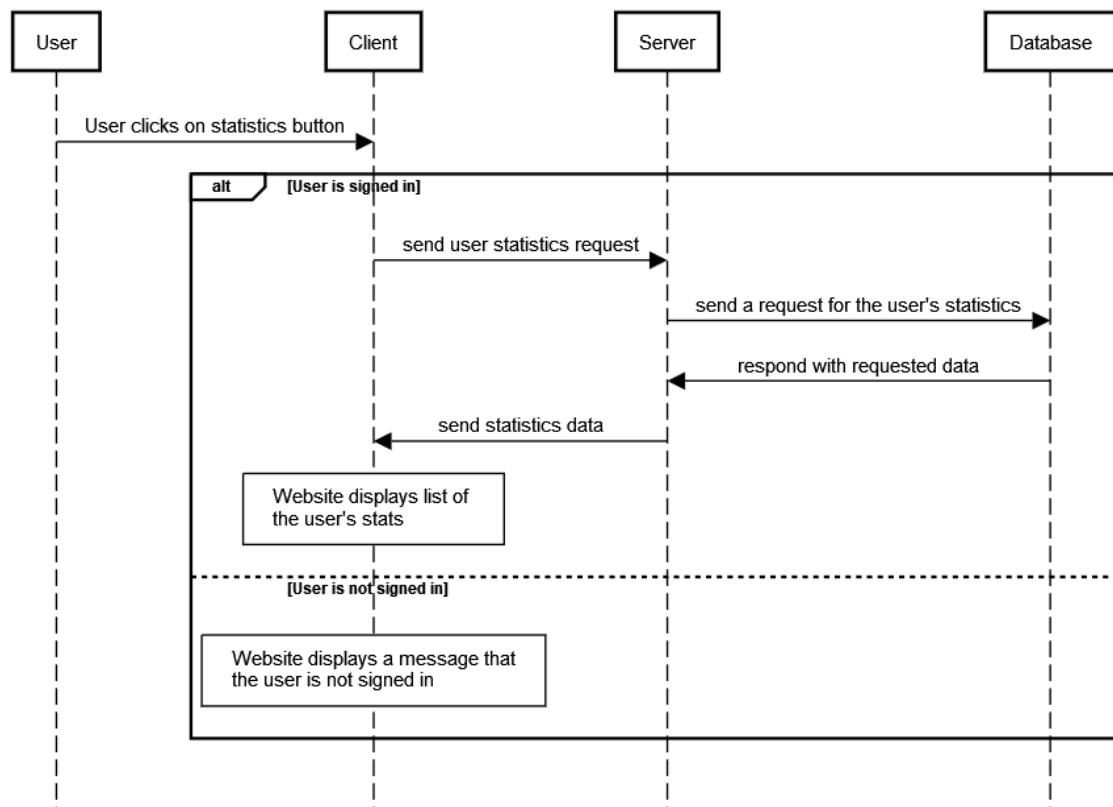


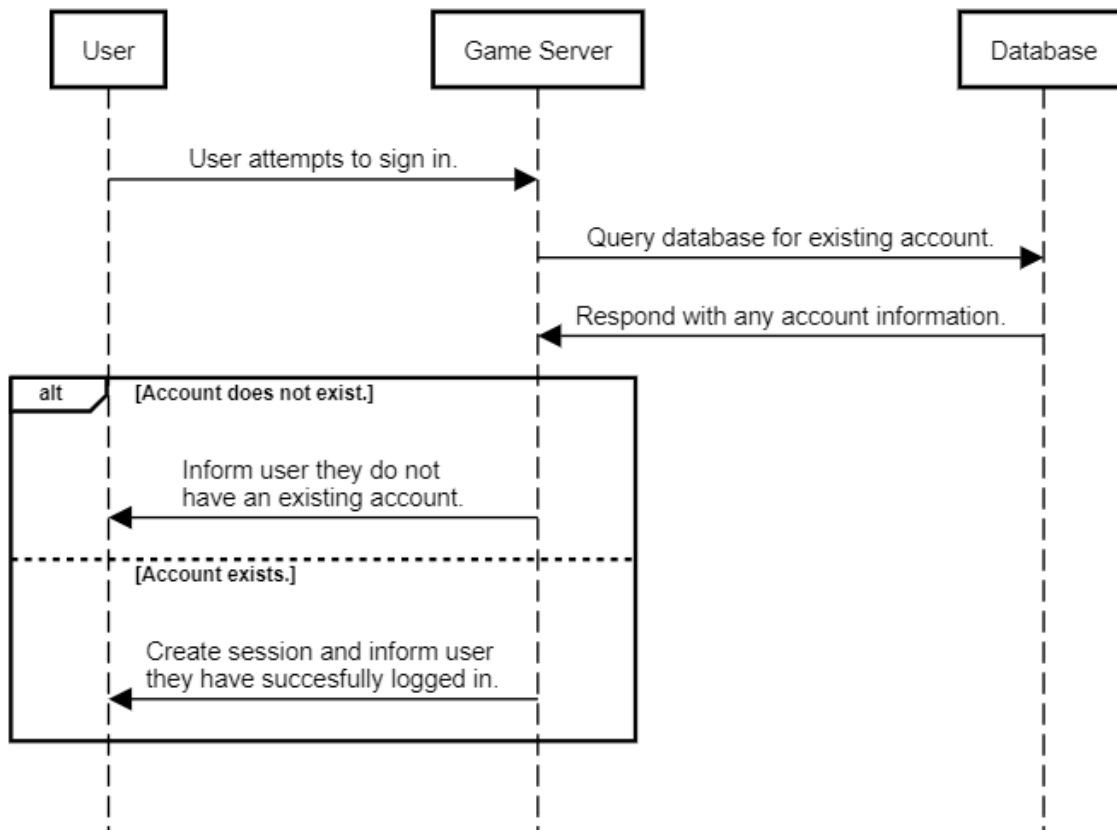
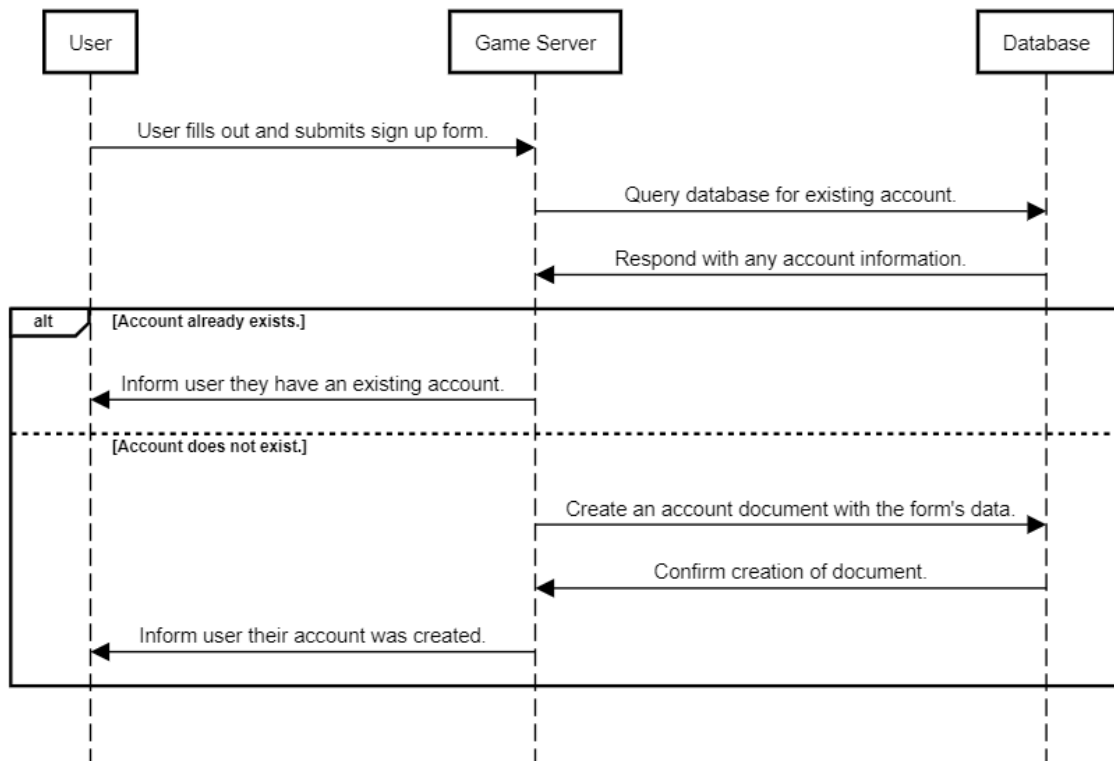
Sequence Diagrams

Sequence of events when user checks the leaderboard

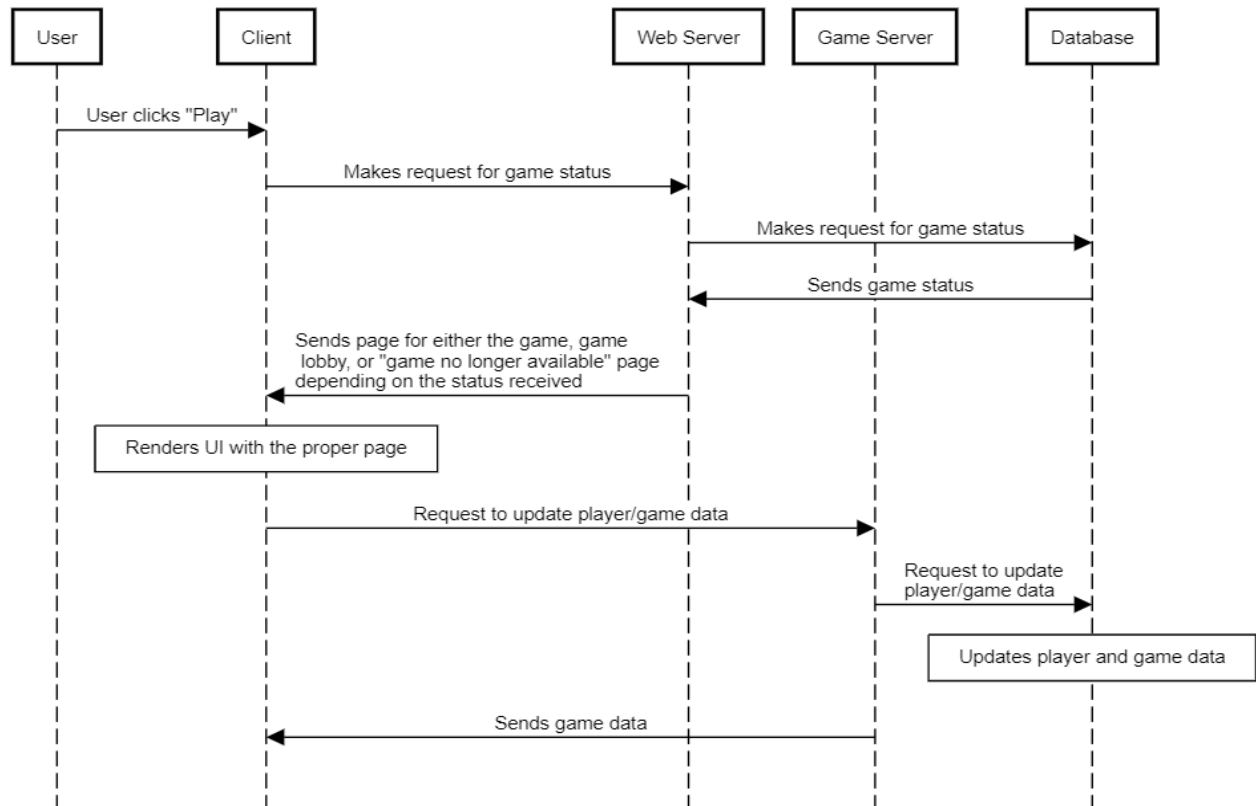


Sequence of events when user checks their statistics

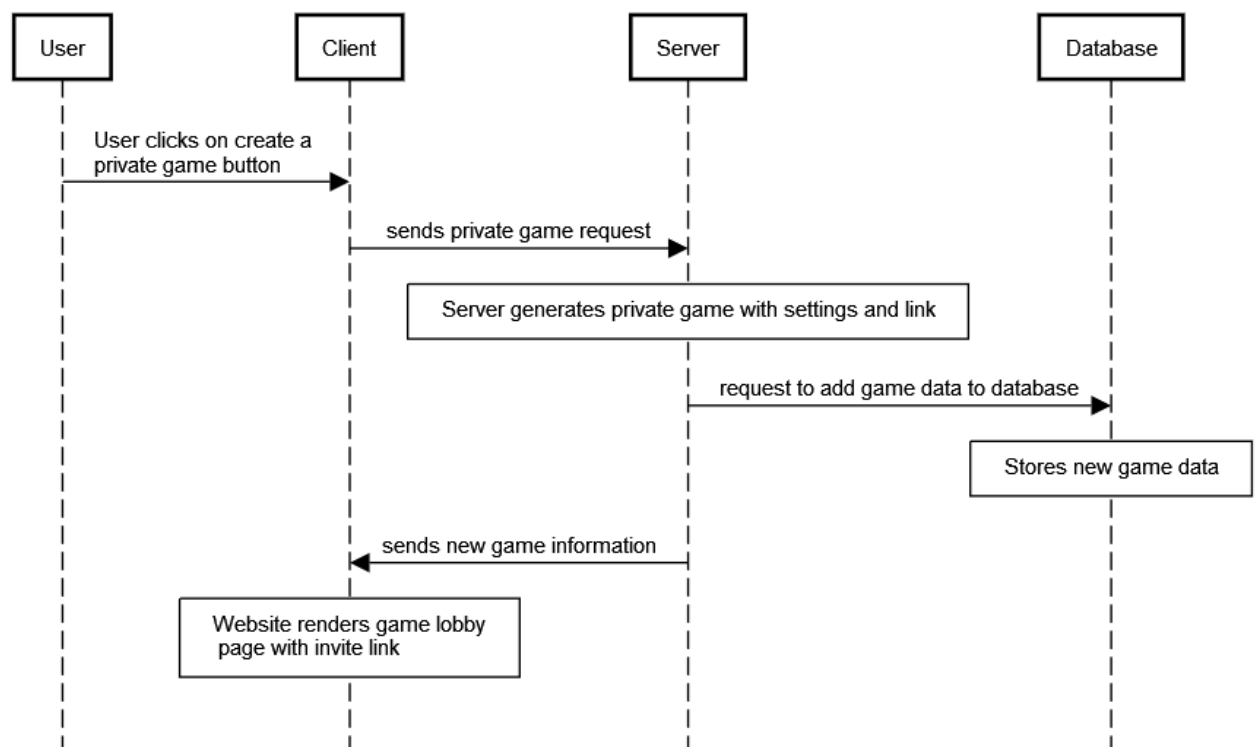


Sequence of events when user attempts to sign up or log in

Sequence of events when user makes a request to join a game



Sequence of events when user makes a request to create a game

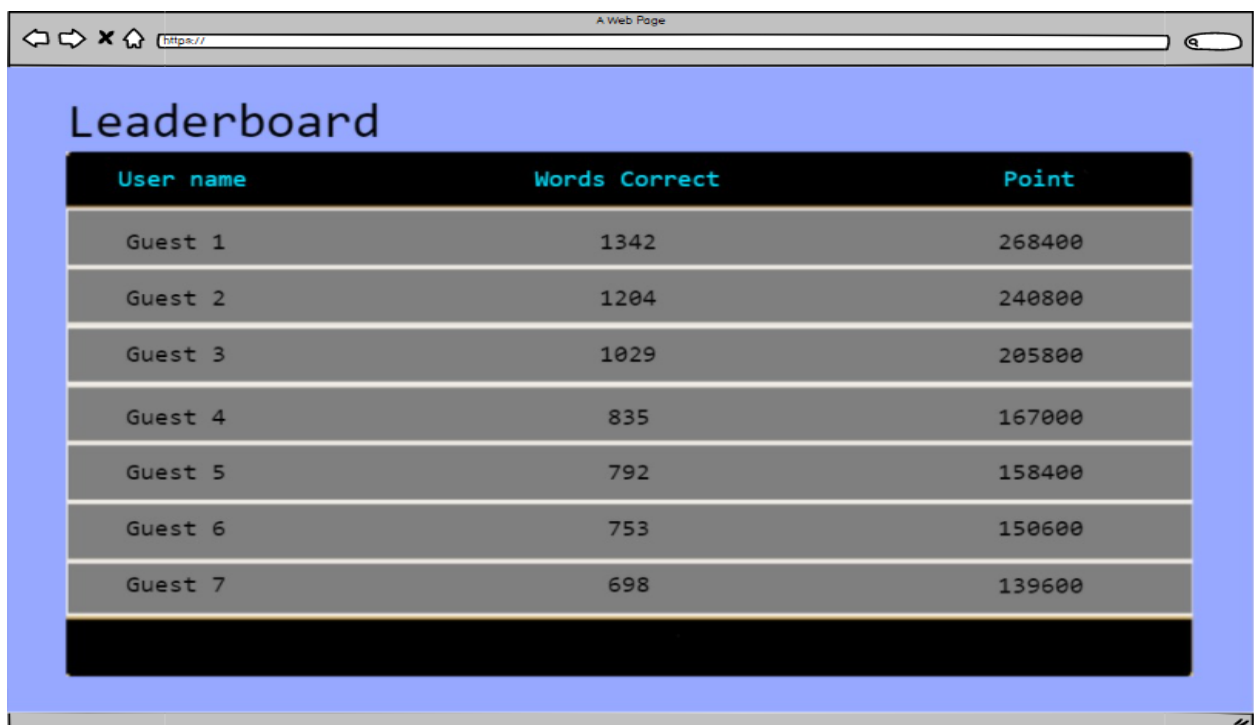


UI Mockups

Main Page



Leaderboard



A browser window titled "A Web Page" showing the leaderboard page. The page has a light blue background. The title "Leaderboard" is displayed in a large, dark blue, monospace font. Below the title is a table with three columns: "User name", "Words Correct", and "Point". The table contains seven rows of data, each representing a guest user and their score.

User name	Words Correct	Point
Guest 1	1342	268400
Guest 2	1204	240800
Guest 3	1029	205800
Guest 4	835	167000
Guest 5	792	158400
Guest 6	753	150600
Guest 7	698	139600

In Game Screen

