CS 30700
Design Document

Team 15

Joe Ruhe: jruhe@purdue.edu
Arielle Dong: dong344@purdue.edu
Nicole Bain: nbain@purdue.edu
Nathan Lytle: lytlen@purdue.edu

Purpose

Crossword puzzles are a regular staple in newspapers, puzzle books, and magazines, simultaneously offering entertainment and mental stimulation. However, crossword puzzle solvers often run into challenges and get stuck when trying to solve these puzzles all by themselves. When you get really stuck on a crossword, it's typical to ask nearby friends/family to take a look and give a second opinion, but what about times when you have nobody around to ask? Our web application, Puzzle Party, aims to address this by giving our users a collaborative platform where they can work with their friends/peers in real-time to solve crossword puzzles, enhancing the overall experience.

The domain our project belongs to is best described as "collaborative puzzle-solving applications." While this domain isn't entirely novel, we plan on focusing on providing a seamless user experience without requiring our users to purchase/obtain some kind of membership, which is a common barrier that collaborative puzzle-solving applications' users are forced to jump through.

Our targeted users consist of puzzle enthusiasts, students looking for educational/mental stimulation, casual gamers, and groups of friends/peers/family looking for engaging, interactive activities to enjoy with each other. By catering to a diverse target user-base and providing the potential for expansion into additional puzzle game types, Puzzle Party aims to provide a collaborative solution to the question of "what do you guys want to do?"

Functional Requirements:

Joining Rooms:

- As a user, I would like to be able to create a room for people to join attached to a 6 digit code
- As a user, I would like to be able to join another person's room when I enter the 6 digit code
- As a user, I would like to be able to kick people from my room and/or ban them from my room
- As a user, I would like to be able to set my room to public or private
- As a user, I would like to be able to join a random public room
- As a user, I would like to be able to invite friends I have on my account
- As a user, I would like to be able to see public rooms that my friends are in
- As a user, I would like to be able to exit a room
- As a user, I would like to be able to see who all is in my current room

Chat:

- As a user, I would like to be able to chat with people in my room
- As a user, I would like to be able to distinguish between people chatting by assigning their chat messages a color
- As a user, I would like to be able to turn on a chat filter
- As a user, I would like to be able to edit the chat filter
- As a user, I would like to be able to adjust the size of the chat box

---

Puzzle Generation:

- As a user, I would like to be able to select the size of the puzzle
- As a user, I would like to be able to be able to generate a puzzle using a seed
- As a user, I would like the generator to use a random seed by default unless I specify
- As a user, I would like the option to randomize the seed again with a reroll button

---

Gameplay:

- As a user, I would like to be able to see where other player's cursors are at (and mine) on the crossword assorted by color highlighting
- As a user, I would like the whole word (all boxes) to be highlighted in a fainter color than where my cursor is so that I know which direction I am typing
- As a user, I would like to be able to select the color my cursor shows as
- As a user, I would like to be able to interact with the puzzle and type where my cursor is at
- As a user, I would like to see descriptions for each number on the crossword separated by "down" and "across"
- As a user, I would like to be able to select a description to have my cursor moved to that number on the puzzle
- As a user, I would like to be able to change my typing direction by pressing space (down or across)
- As a user, I would like to be able to check my current guesses
- As a user, I would like to be able to receive a hint in the form of an extra letter on my current word
- As a user, I would like to be able to set a timer for any crossword I do
- As a user, I would like to be able to play more game modes than crossword, such as sudoku (if time allows)

Log-in:

- As a user, I would like to be able to play as a guest
- As a user, I would like to be able to create an account with my email and a password

- As a user, I would like to be able to save settings, such as my color for highlighting my cursor/current word, timer on/off, and chat filter on/off
- As a user, I would like to be able to set my display name
- As a user, I would like to be able to add friends
- As a user, I would like to be able to check my statistics
- As a user, I would like to be able to customize my avatar (If time allows)

Quality of Life:

- As a user, I would like to have a landing page with options to create a room, join a room, access my account, and generate a puzzle
- As a user, I would like to be able to click a home button to bring me back to the landing page
- As a user, I would like to be able to access Puzzle Party as a website
- As a user, I would like to be able to play on both my computer and mobile devices
- As a user, I would like to be able to refresh the page and remain in my current room
- As a user, I would like this program to be tested extensively for bugs and flaws so that I receive a polished product
- As a user, I would like a well designed UI that is colorful and pleasing
- As a user, I would like to be able to suggest words/descriptions to be added to the game (If time allows)

Nonfunctional Requirements:

Client:

- As a developer, I want the application to be able to be used on a phone or computer browser

---

Server:

- As a developer, I would like the server to save player statistics to a central database
- As a developer, I would like to have both a central server to manage rooms, and do the rest of the computing with distributed servers (host's computers)
- As a developer, I want to be able to test the client during all stages of development with the server

---

Design:

- As a developer, I want both the client and server applications to use a shared set of classes to make development more streamlined

---

Performance:

- As a developer, I want a strong enough central server to be able to handle several hundred rooms and a few thousand players
- As a developer, I want to limit the number of players per room to 6 so that the host's computer does not have a high latency
- As a developer, I would like to cache recently generated puzzles and set the default seeds to these cached puzzles as to not waste resources
- As a developer, I would like to have the first few randomizations of a seed align with the cached puzzles
- As a developer, I want both the client and server to handle any errors that might occur without crashing
- As a developer, I want to design a system that scales gracefully

---

Appearance:

- As a developer, I want the application to be enjoyable and pleasing to look at

---

Security:

- As a developer, I want to make sure that user's information is stored securely
- As a developer, I want to make sure that user's information is transmitted securely
- As a developer, I want to make sure that the user's information is not used in unethical ways

Design Outline

Overview:

We will be using a client-server model, with the server ultimately communicating with our database to store user information (outlined in our user stories), including things such as friends, statistics, and display name. We will be using a Javascript Library called Ably to handle socket interactions and allow for multiplayer collaboration and Axios to handle client-server communication.

Breakdown:

Client:

1. Client provides user interface and provides interactive, responsive UI to allow users to view other users
2. Client sends user requests (actions and inputs) to the server - Axios
3. Client gets and interprets the Axios responses from the server that indicate what the client should render in the UI

Server:

1. Server gets and interprets the Axios responses from the client and appropriately updates information
2. Server sends requests to the database when data needs to be modified (user information, profile information)
3. Server allows for cross-communication and collaboration between multiple users

---

Database:

1. Store user information (used for user profile and statistics)
2. Relational database schema, as there is highly structured data

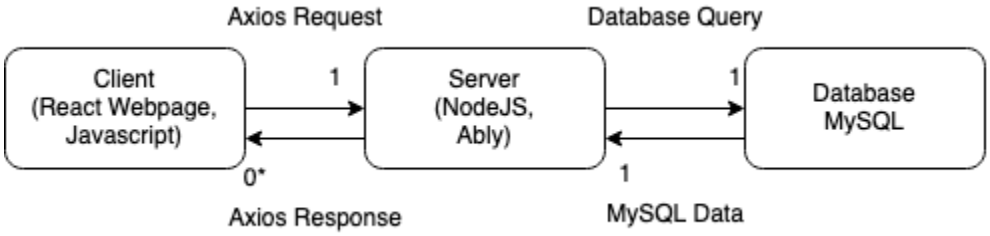
---

Interactions Between System Components:

First, the user opens the webpage, and is greeted by the landing page, rendered by the client. If they choose to sign in, they will be prompted to enter their information, which will then be communicated to the server and eventually the database to verify their credentials. After logging in, the user can choose to create rooms to play with other users, play puzzles, or modify their account information. When clients perform actions, the actions and responses are sent over to the server, and vice versa. The server only communicates with the database when modifying account information and updating statistics. Otherwise, all actions are localized to the client and server, and additional

actions are not stored.

UML Diagram:



Design Issues

Functional Issues:

1. Should users need to login to use our service?
    - **A.** Login not required, but optional
    - **B.** Login with email, username, password, and phone number
    - **C.** Login with email, username, and password

Decision: Option A and C. We decided to make logging in optional. Logging in would provide users with game statistics and the feature where they can view games that friends are currently in, which also gives them the option to join the game. We made logging in optional for users who want to try out our website or join a friends game. These users are not invested enough to care about game statistics or maintain a friends list. We decided on simply using email, username, and password as our login credentials. We chose to include emails for users that forget their other login credentials. Since we have user emails as a second form of verification, it is unnecessary to ask for their phone number.

---

2. How should lobbies work?
    - **A.** A host creates a lobby for others to join, if the host leaves, the game ends
    - **B.** A host creates a lobby for others to join, if the host leaves, the game migrates to another player who becomes the new host.

Decision: Option A. We want the host of the game to be able save puzzles that they're working on and resume playing as desired. Thus, the multiplayer session would end when the host leaves, allowing the host ownership of the puzzle. If we implemented migration, the host wouldn't have the feature of picking up where they left off. Overall, this decision was made to give users flexibility in when they want to resume play.

---

3. How should users join a private lobby?
    - **A.** A short 6 digit code is given to the host, who can then share it with users they would like to invite
    - **B.** A link is given to the host, who can then share it with users they would like to invite
    - **C.** Both

Decision: Option A. We chose to base our game joining feature on apps like Kahoot and Among Us, where hosts can share a 6 digit code with team members. Since hosts can share a code similarly to how links are shared, there would be no need to

include both abilities. Furthermore, codes can more easily be verbally and visually shared for players who want to collaborate on a puzzle in person. Overall, a code provides simplicity and ease to users when connecting to their team.

---

4. How should we get our puzzles?
    - **A.** We create our own puzzle generator
    - **B.** We legally take existing puzzles on the internet

Decision: Option A. We decided to create our own puzzle generator to give our users a unique experience. There are no random crossword puzzles on the web as of now, and using our seed-based system users could generate many puzzles and not worry about running out. With taking puzzles from online, we have to worry about limited supply and possibly legal troubles even where there appear to be none. Also, our team enjoys creating and solving puzzles, so generating our own would be enriching.

Non-Functional Issues:

---

1. What database is most appropriate for our data?
    - **A.** MySQL
    - **B.** NoSQL
    - **C.** MongoDB

Decision: Option A. We chose mySQL since it is a relational database and is free and open source. Since it is widely used, there are many tutorials, and it has extensive documentation. MySQL also provides security features and provides structured and consistent data management. Furthermore, some of our team members have prior experience with mySQL, which makes managing the backend of our product easier.

---

2. Which backend service are we going to use?
    - **A.** AWS
    - **B.** Ably
    - **C.** Google cloud
    - **D.** Azure

Decision: Option B. We chose to use Ably as a backend service since it provides APIs to connect data between devices and the backend. With these data streams we can support multiplayer collaboration, which is the primary feature of our website. Although the other backend service options are more popular, this service is free

and provides us with guides on how to integrate this specific feature in our web application.

---

3. What frontend language/framework should we use?
    - **A.** HTML + JavaScript
    - **B.** React
    - **C.** Angular

Decision: Option B. We chose React because it is a widely used framework, so there exists many guides and solutions to common problems online. It also uses components, which are organized and easy to maintain. React uses the virtual DOM, which provides a higher performance because it minimizes the number of DOM operations. Overall, React provides a clean and efficient way to build the UI of our project, and it has plenty of online resources for beginners like us.

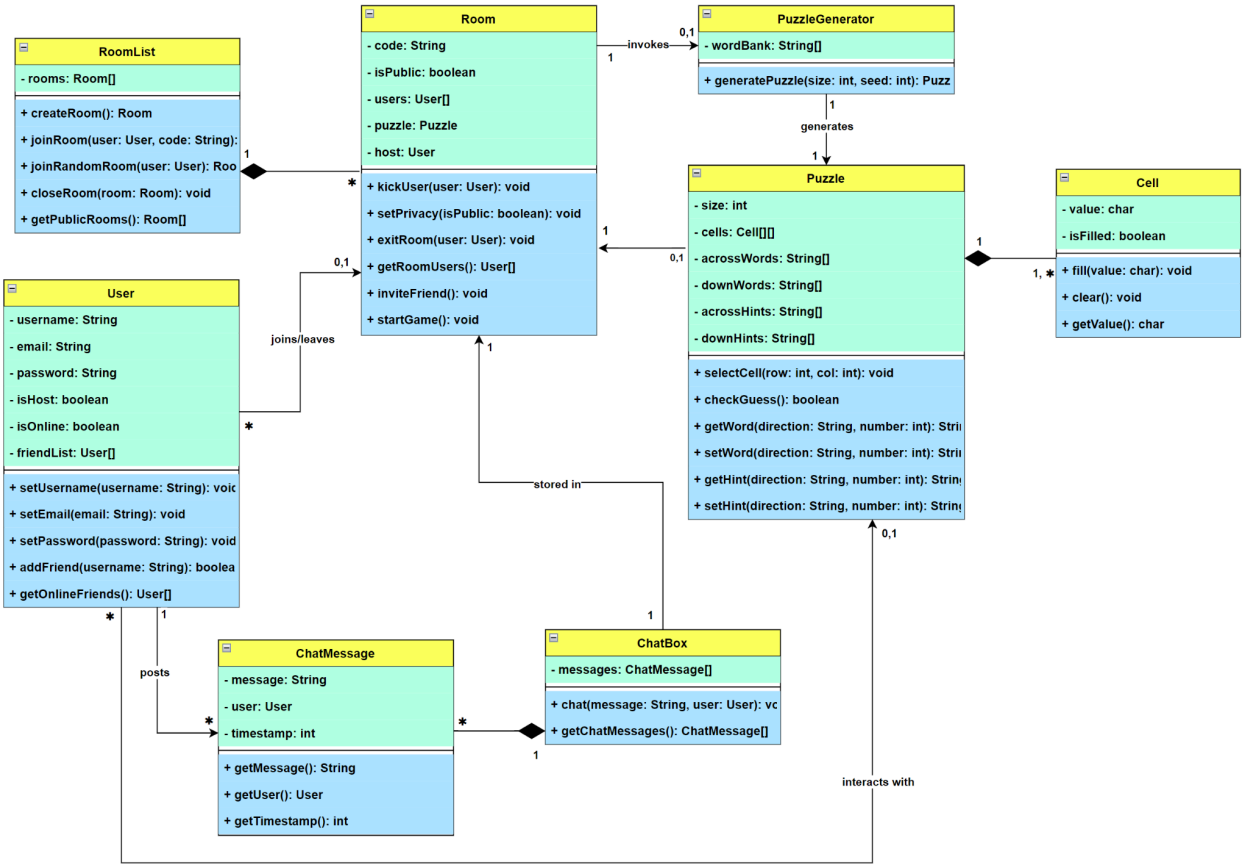
---

4. What backend language/framework should we use?
    - **A.** Node.js
    - **B.** Django
    - **C.** PHP

Decision: Option A. We chose to use Node.js because our product requires real-time updates for our chat and puzzle features. Node.js uses an event-driven architecture, which would support our websites' need for high concurrency. Node.js is good with handling a large number of requests and connections. It also provides consistency with React, which we chose as our frontend technology, since they are both JavaScript based.

Design Details

Class Diagram:



Class Descriptions / Class Interactions

Classes are assigned on the basis of grouping similar things together. When attributes (light blue) or methods (dark blue) can fall under one umbrella term, we attempt to group them into a class (yellow). Below is a description of each class and how it interacts with other classes.

User:

The User class is used primarily to keep track of users and their credentials, as seen with the attributes username, email, and password. Users can also become a host of a room, as shown with the isHost attribute. Users will have an online status that can be used to determine if they are eligible for an invite when a host has created a room. Users can also add other people to their friend list and invite them when they are online. Users can edit their username, email, and password.

Attributes:

- username
- email
- password
- isHost
- isOnline
- friendList

Methods:

- setUsername()
- setEmail()
- setPassword()
- addFriend
- getOnlineFriends()

RoomList:

The RoomList class is a housekeeping class that manages all available rooms. It is also the class that will connect users together by putting them in rooms, or close rooms when they are no longer in use. Users can request to join a random room, and the RoomList class will put them in a random public room. Users can also request to see a list of all available public rooms so that they may choose which one to join.

Attributes:

- rooms

Methods:

- createRoom()
- joinRoom()
- joinRandomRoom()
- closeRoom()
- getPublicRooms()

Room:

The Room class is a child class of RoomList. This class represents an individual room. The code attribute is used to identify and assign rooms by RoomList, and someone is only able to join a room if isPublic is true. The room has a list of users, and 1 or 0 active puzzles per room. The host is the user that can control the room and do things such as start the game, kick users, or set the room to public or private. All users in the room can invite friends, exit the room, and see who is currently in the room.

Attributes:

- code
- isPublic
- users
- puzzle

- host

Methods:

- kickUser()
- setPrivacy()
- exitRoom()
- getRoomUsers()
- inviteFriend()
- startGame()

PuzzleGenerator:

The PuzzleGenerator class will be used solely to generate puzzles, as the name implies. It will have access to a wordBank, which will be located in the database as it will eventually become very large. The structure of the wordbank is unknown as of the moment, as we need an efficient way to search and sort for words that can be added to the puzzle (work in progress). This class will also be able to generate a puzzle given a seed, and that seed will be the basis for generation. Random seeds will be used if the seed section is left blank.

Attributes:

- wordBank

Methods:

- generatePuzzle()

Puzzle:

The Puzzle class holds all the information about a single puzzle and will be used to manage the puzzle as it is completed. Basic attributes include the size of the puzzle and cells. The across words and down words will be matched with the across hints and down hints using the appropriate attributes. When a hint is selected, then the user will be taken to the corresponding cell on the puzzle. This class is also able to check guesses as players are working on the puzzle.

Attributes:

- size
- cells
- acrossWords
- downWords
- acrossHints
- downHints

Methods:

- selectCell()
- checkGuess()
- getWord()

- setWord()
- getHint()
- setHint()

Cell:

The Cell class is a child class of the Puzzle class. The cells make up the puzzle, and each cell holds an individual character denoted by value. The isFilled attribute determines whether there is a letter in the cell or not. Users can fill cells with a value, erase cells, and check their guesses using the Puzzle class.

Attributes:

- value
- isFilled

Methods:

- fill()
- clear()
- getValue()

ChatBox:

The ChatBox class is used to store all of the current messages up to a limit that users in the same room send. There is one chat box per room, and so the Room class and ChatBox class have a one to one relationship. Users can send messages.

Attributes:

- messages

Methods:

- chat()
- getChatMessages()

ChatMessage:

The ChatMessage class is a child of the ChatBox class. This class stores one individual message that a user sends and all of the associated information, such as the message itself and the timestamp. The ChatBox class can use the timestamp to delete old messages if need be.

Attributes:

- message
- user
- timestamp

Methods:

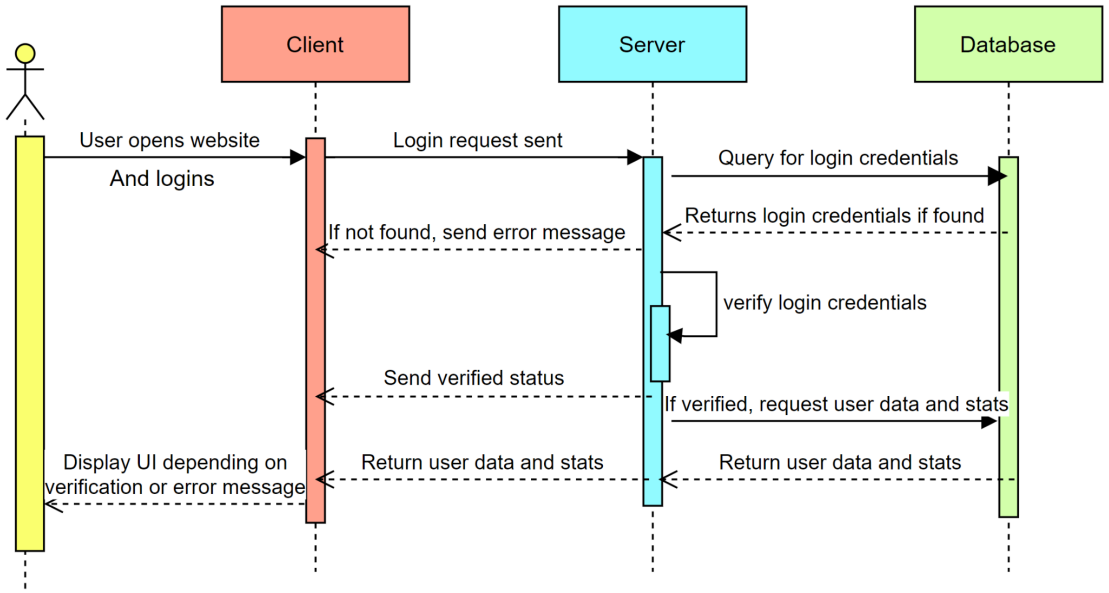
- getMessage()
- getUser()
- getTimestamp()

Sequence Diagrams:

These diagrams show what might occur when the user attempts to the the following:

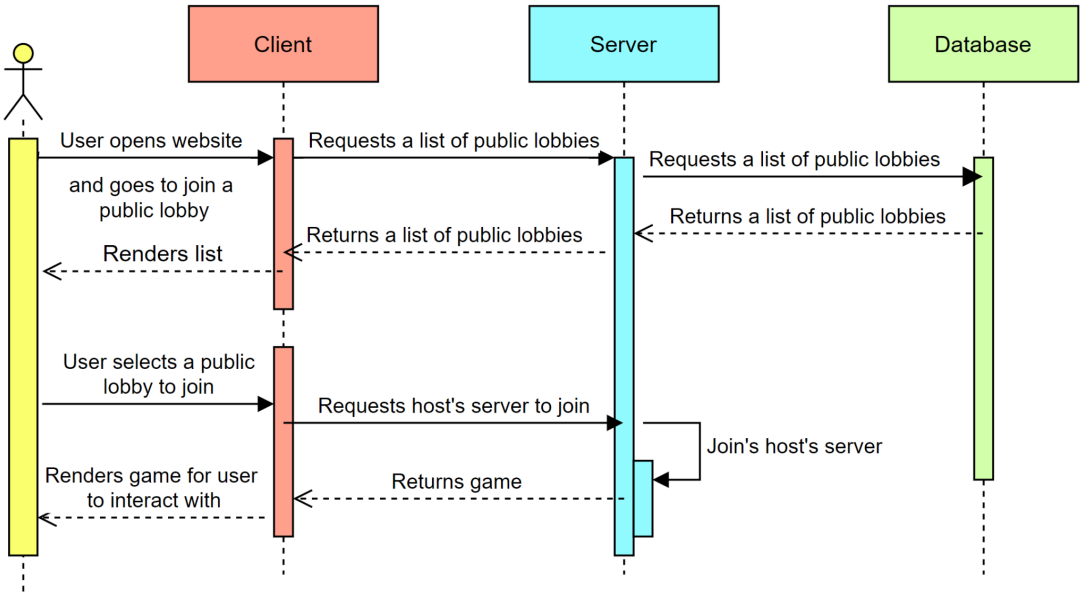
Log In to the Website:

This Diagram shows the process for logging into the website. The user first opens the website and enters their credentials. The request is sent to the server, and the server grabs the user's actual credentials from the database (if found, if not send an error). The server then checks the credentials that were sent against the actual login credentials, and if they match the user will be notified and logged in. The server will then fetch user statistics to display on the UI for that specific user and return them to the client.



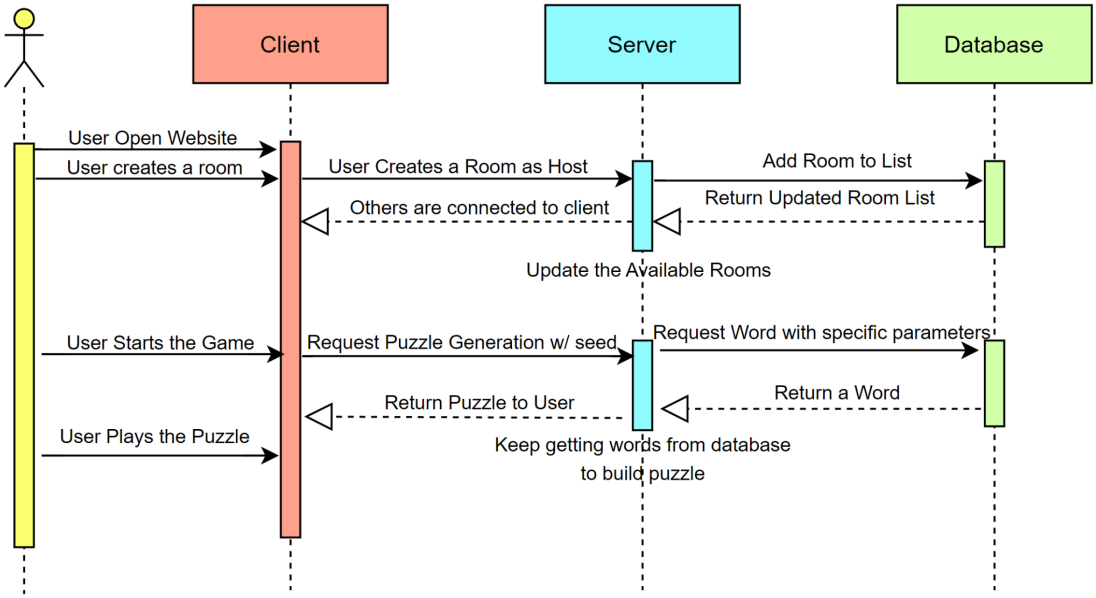
Join a Public Lobby:

This diagram shows what happens when a user attempts to join a public lobby. First they open the website and can choose to login as shown above. Then, a request for public lobbies is sent to the server, which the server then requests from the database. This list is then passed back to the server, and then the client where it renders for the user. The user then selects the lobby/room they want to join, and the request is sent back to the server. The central server then connects the user with the host's computer (peer to peer server) and allows them to play the game without the need for a central server.



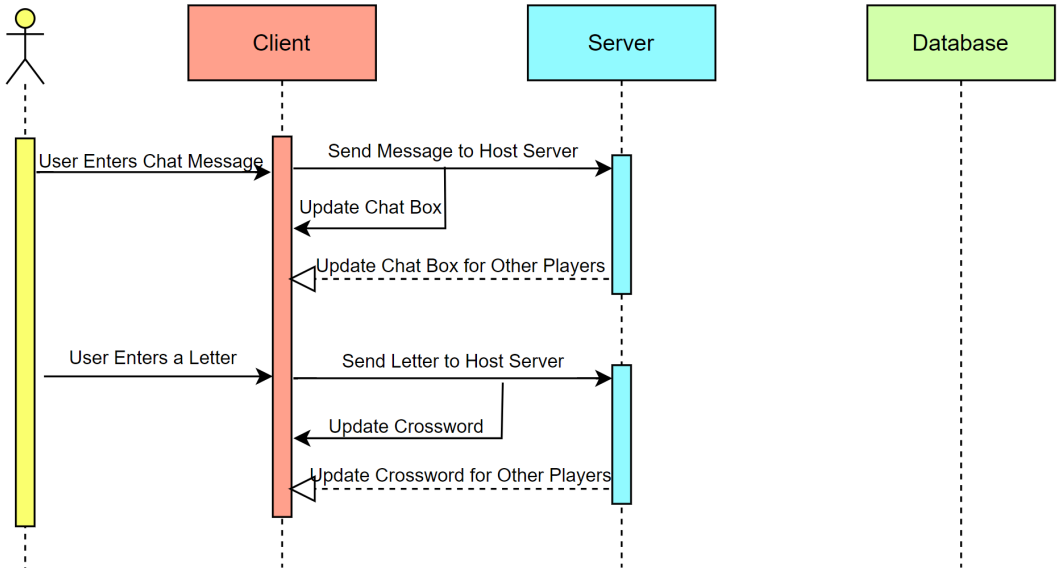
Generate a Puzzle:

This Diagram shows the process for generating a puzzle. The user must first create a room as a host, as only hosts have the ability to start a game/generate a puzzle. When the room is created, it is managed by the RoomList class and added to a list of rooms in the database. The host can then allow users to join if wanted. After that, the user starts the game and a request for a puzzle is sent to the server with a default or custom seed. The server repeatedly pulls words with specific parameters from the database and builds a crossword puzzle. When the puzzle is finished, it is sent back to the client where no further contact from the server should be required to play the game.



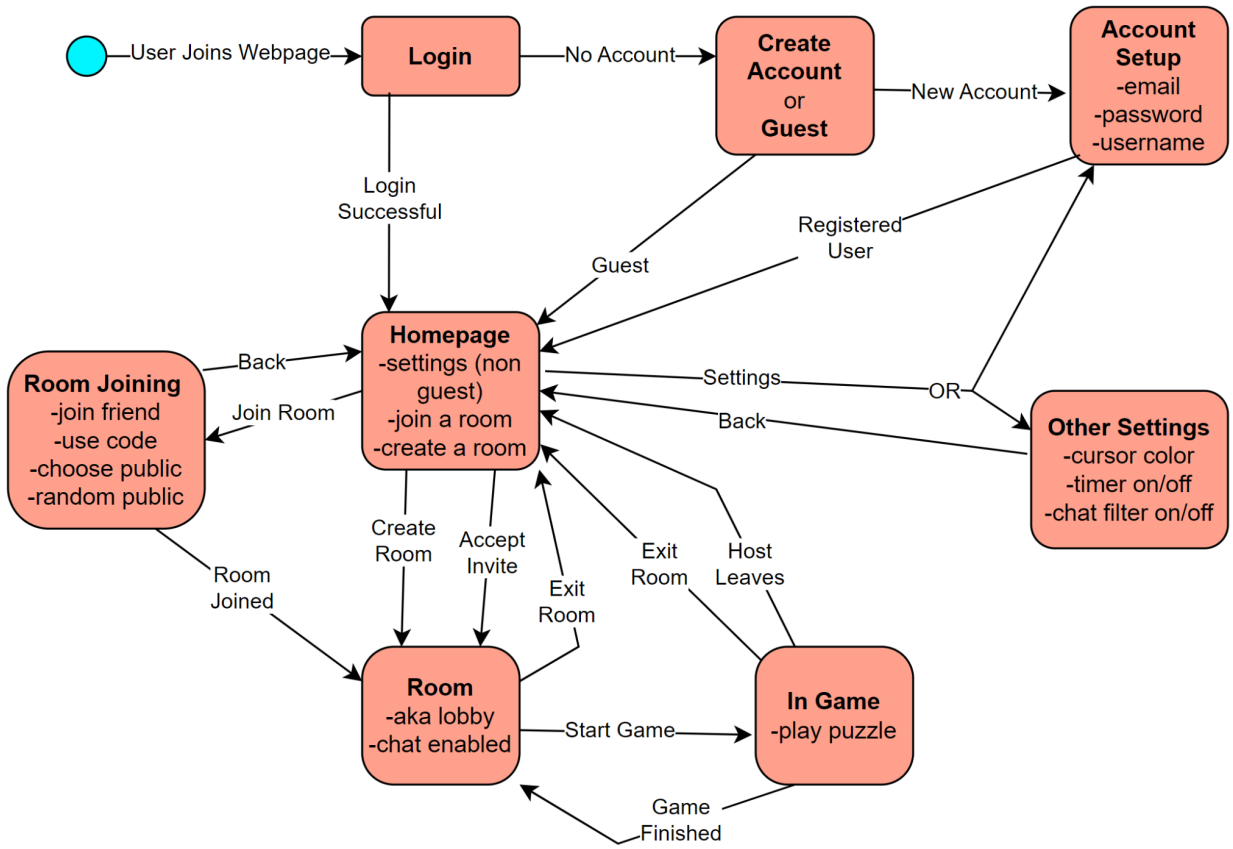
Send a Chat Message or Enter a Letter:

This Diagram shows the process for sending a message when in a room. The user first enters the message they want to send in the chatbox, and then presses enter. The message is sent to the server, and the new message is also updated on the user's computer/client instantly. When the server (host's computer) receives the message, it sends it to all of the other players so that their clients can render it in their chat boxes. The same principle applies to typing a letter into the crossword puzzle. It will update instantly for the user, and then the letter typed will be sent to the host's computer and back to the other players so they can render it on their crossword.

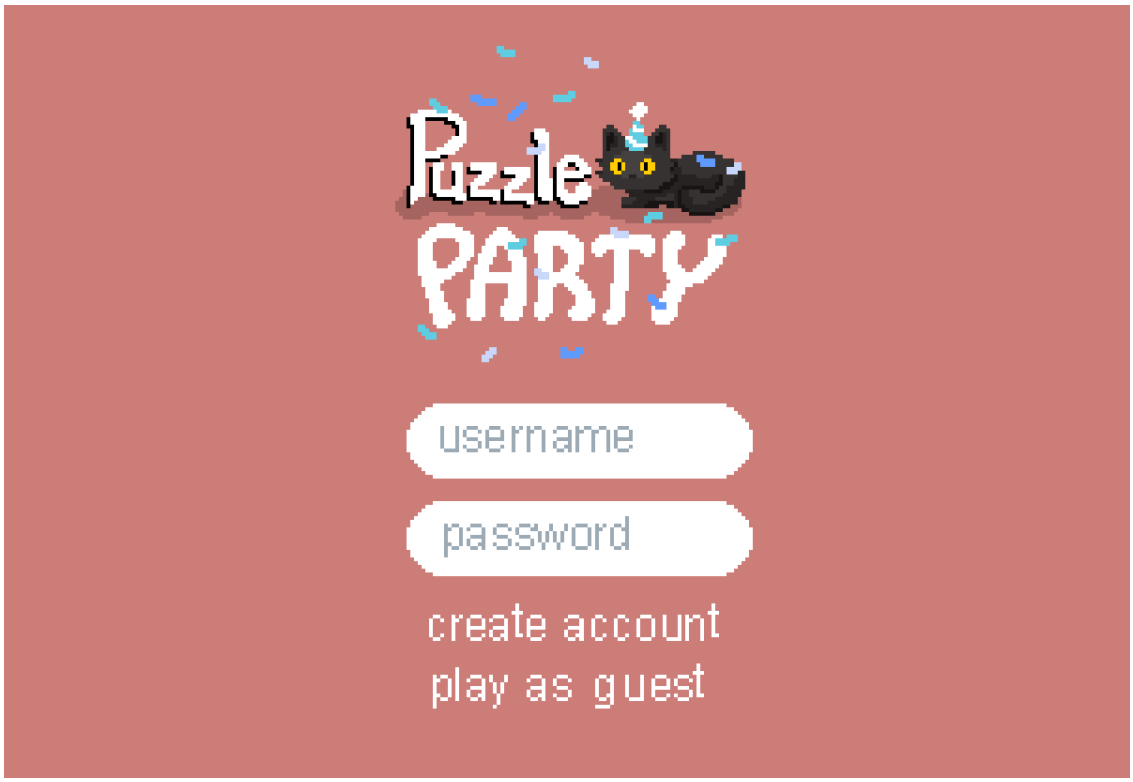


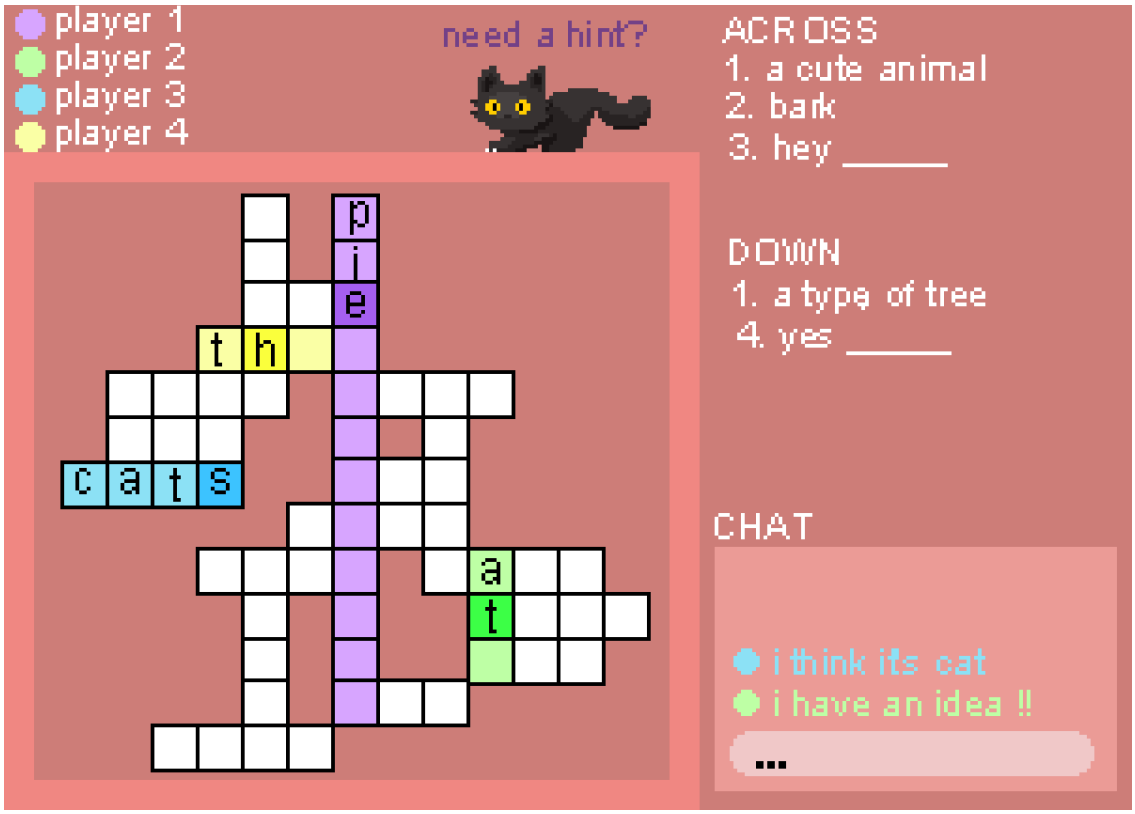
Navigation Flow Diagram:

This diagram represents the flow that a user might experience when taking certain actions. They are first required to log in. If they can't, then they are given the option to create an account or continue as a guest. All three of these paths lead to the homepage. From here, users can go to settings (account settings or game settings). They can also create a room or join a room. Creating a room lands the user in a room/lobby. Joining a room gives the user several options, but all lead to a room being joined. Finally, the host of the room can start the game and the users can play the game or leave. Finishing a game will put the users back into the room/lobby. Users are always able to go back or exit if need be.



UI Mockups:


Puzzle
PARTY
username
password
create account
play as guest


player 1
player 2
player 3
player 4
need a hint?
ACROSS
1. a cute animal
2. bark
3. hey _____
DOWN
1. a type of tree
4. yes _____
CHAT
i think its cat
i have an idea !!
...