# StairCase

## Game Application

**Brendan Ly and Ashkan Nikfarjam**

## 2. Introduction

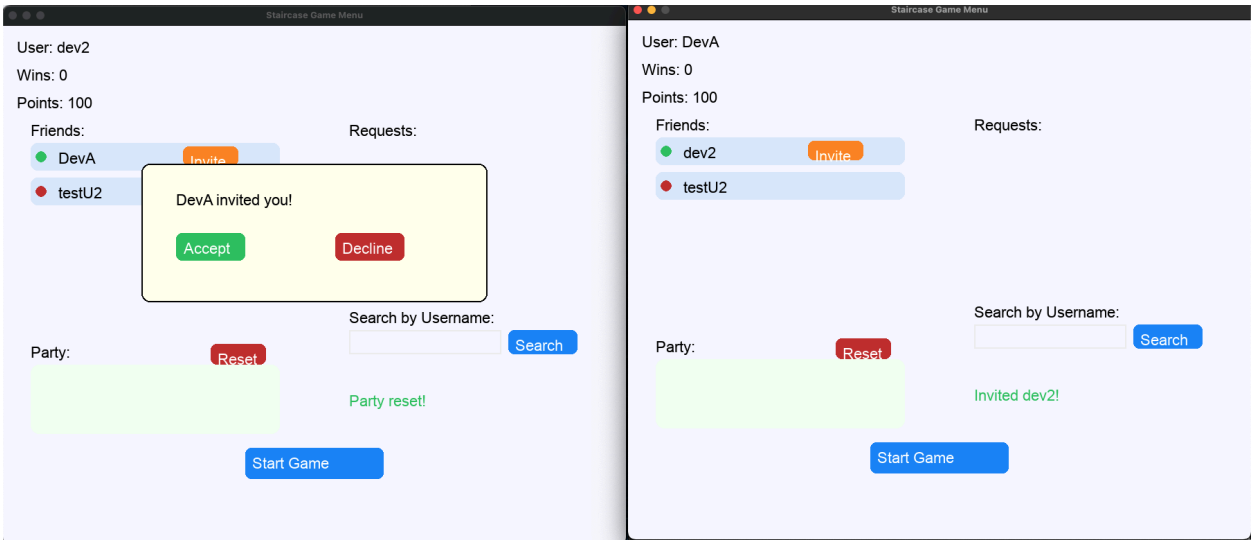
StairCase, is a rendition of a famous Persian game called Snakes and Ladders. We decided to develop this game for PC and Mac to allow people of all ages to play together and bring back old memories of playing this game with their friends and family. We are excited to see how we can tie all of our skills together to make a full stack game, and get to experience other people's joy of the game. We hope to develop our full stack game development utilizing **langchain** to leverage LLMs to develop our game application with.

In the game, there can be up to 2 players playing with each other from different clients, as there is multiplayer support. Each player takes turns rolling a dice that moves them 1-6 squares forward on the gameboard which is a 10x10 grid. In random places there will be ladders that take you ahead and snakes that push you back. There are also Trivia and Hangman minigames that are triggered by landing on certain squares. Correctly solving the minigames moves the player forward, while giving the wrong answer moves you backward.

## 3. Final Features & Functionality

### User Perspective:

The user must first sign up on the designated website using their email and create a stand alone user name for security purposes to be granted access to the download link and run the game client on their device. Then, upon starting the game, the user must sign in with the same credentials they used to sign up on the website. Then, they will be directed to a screen where they can invite other players, search other other users and send game party or friend requests just by their username. After inviting each other, the two players can start the game with each other. The person who made the invitation has to start the game first for correct player assignment.



Upon starting the game, one player is Player 1, represented by the red circle on the board, and another is Player 2, represented by the blue circle. A small message on the bottom left of the game indicates which player's turn it is. All players start at position 1 on the board, and the first player to make it to 100 wins the game. Player one gets to go first, and they can click the “Roll Dice” button to move 1-6 squares forward. Then, the players alternate turns rolling the dice.

There are obstacles scattered throughout the board though, which include:

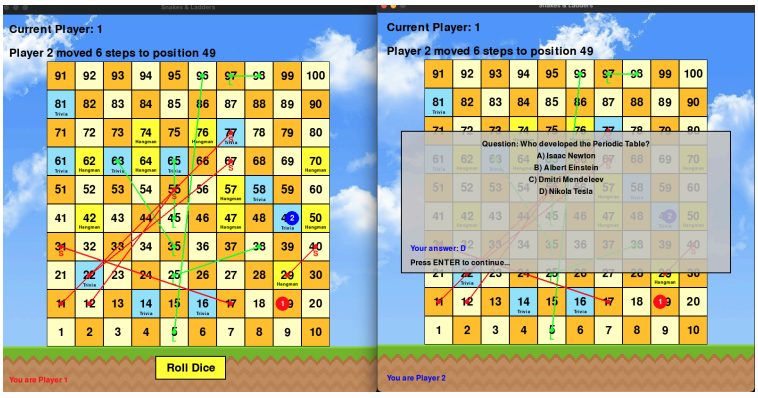
- Snakes: Take you back to its base if you land on its head
- Ladder: Take you to its top if you land on its base
- Trivia: Prompts you to answer a trivia question, successful answers move you forward up to 10 steps, otherwise move back 10 steps
- Hangman: Prompts you to answer a hangman word, and you have 3 attempts to guess. Missing an attempt reveals one more letter. If you solve it, you get to move up to 10 steps forward, otherwise you move back 10 steps

Each time a player rolls the dice, a message is displayed on the top to give better indication of what happened:

Current Player: 1 or 2
Player _ moved _ steps to position _
Player _ climbed a ladder from _ to _ (if landed on base position of ladder)
Player _ was bitten by a snake and moved from _ to _ (if landed on head position of snake)

This can keep going on until one player reaches the end at position 100 first and wins the game. Once that happens, the winner is displayed, and a restart button appears on the bottom to replay another game.

As for the final features compared between our final submission and the progress report, things were the same except for some features that didn’t make it into the final project. For example, features such as a shop to buy new background themes or more minigames such as the wheel of fortune (moves you a random amount of steps) were not included due to time constraints.



### 4. Design, Implementation and Deployment

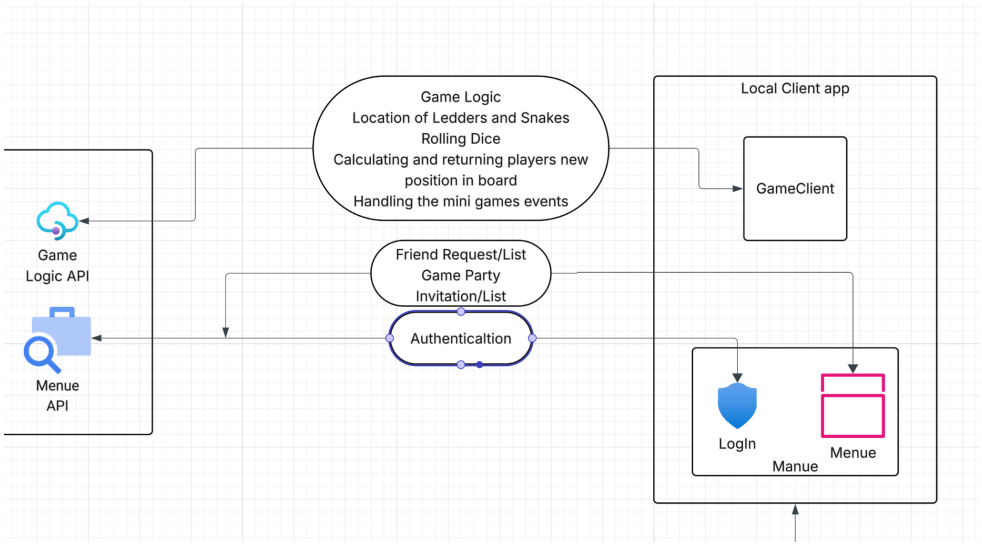
StairCase is implemented using a three-layer architecture: the Front-End Client (developed with Pygame), the Back-End Server (built with Flask API), and the Database Layer (utilizing Firebase Authentication and Firestore DB). This layered design ensures secure, scalable, and multi-device compatibility while maintaining a clear separation of tasks across the system.

To support user registration and get access to download links, a lightweight web application was also developed using ReactJS. This portal allows users to sign up, which is the only method implemented for users to get added to the database. Currently, the front-end client is compiled and distributed for both Windows systems and Apple devices powered by M-series silicon chips.

#### Front-End

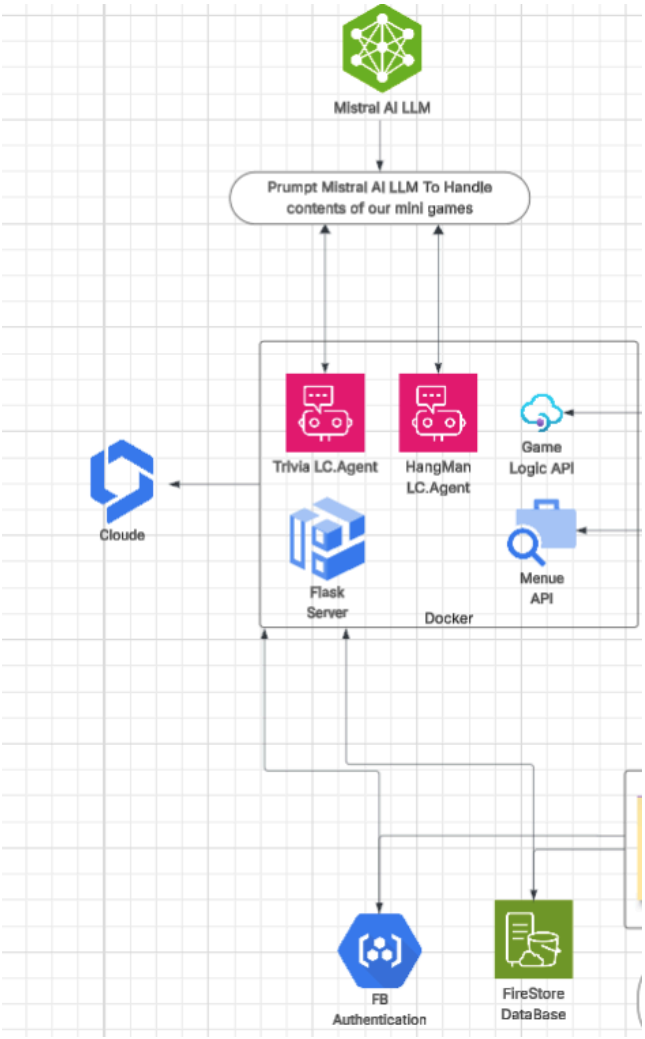
The front-end is responsible for user interaction and graphical rendering. It consists of two primary components:

- **Menu Module:** This module manages user authentication and social features. It consists of two components, Log-In and Menu. Once authenticated, users can search for other players, send friend requests, and accept invitations to games or friends requests. While the visual interface is handled here, the core logic and data processing are delegated to the back-end server through API requests.
- **Game Client Module:** This component handles the gameplay interface, player interactions, and visual feedback during a game session. It communicates with the back-end server to retrieve game states, update moves, and synchronize multi-device sessions in real time. This mainly draws all the Pygame GUI components of the game, such as the 10x10 grid, dice rolling buttons, player animations, etc.



### Back-End

The Back is implemented using Flask Server, where it provides APIs for front end clients. Also the minigames contents are handled by two LangChain agents where it's wrapped by Flask server and it is accessible through the APIs. This layer also communicates with our database for user authentication as well as maintaining user information.




- **Utilizing Flask Blueprint**

A Flask blueprint is a way to organize a Flask application into reusable components. It helps manage routes and functionality for different parts of an application, making the code modular and maintainable, especially for larger projects. Blueprints can contain routes, templates, static files, and other resources, similar to a Flask app, but are not a complete application on their own. They are registered with the main application to become functional.

To make our Back End more modular, all the Api routes are organized and placed in two main BluePrint containers.

**Menu BP:**

- /set_statu**s:** This route sends a request to Firebase DB, To change the value of the status to True or False, this status is used to determine whether a user is Online or Offline and it reflects the friends list in the Menu. When a user logs in and exits the application it sends a request to update this.
- /get_username_by_id: This api receives requests from the front end during user search. It sends a request to the FireStore to search each user's documents to find a matching user id and then sends back a json containing the user's real firebase id, that later is used to send a friend request.
- /login_auth: Authenticates the user using Firebase ID token. Returns UID and email upon success.
- /get_username_by_id: Fetches a username given the document ID. Helps identify friends via Firestore references.
- /get_use_info: Returns a user's **username**, **points**, and **number of wins** based on their username.
- /get_friends_list: Retrieves the list of a user's friends along with their status and stats.
- /search_user_by_username: Searches for a user document by exact match on Username.
- /send_friend_request: Sends a friend request from one user to another. Updates sender's and receiver's Firestore data.
- /accept_friend_request: Sends a friend request from one user to another. Updates sender's and receiver's Firestore data.
- /accept_friend_request: Accepts a pending friend request
- /send_invite: Sends a game invitation. Sets the receiver’s pending_invite field and initializes the party list with the sender.
- /check_invite: Checks if a user has a pending game invite (returns who sent it).
- /accept_invite: Accepts an invite: adds both players to each other’s Party and appends the receiver to the global GameStart.party document.
- /decline_invite: Declines a game invitation by removing pending_invite field.
- /get_party: Fetches the local Party list of the requesting user (i.e., who they’re grouped with).
- /reset_party: Clears a user’s Party field in Firestore. Used to reset the party state manually or on logout.
- /shared_party: Retrieves the shared party from GameStart.party. Used to know who should join the game globally.
- /start_game_signal: Signals game start by setting a global start_for field and updating the user's GameStarted flag.
- /check_start: Checks if the game has started for the specific user by comparing start_for.
- /get_party_status : Checks if a specific user has started the game (GameStarted = True).

**StartGame BP: These are the endpoints of our StartingGameRout.py**

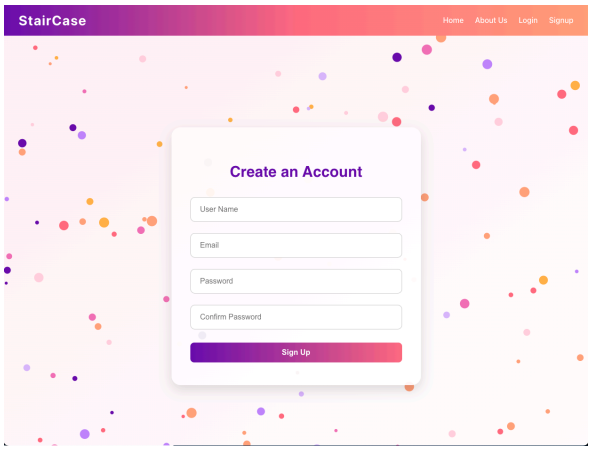
- **/join:** Handles joining the game for 2 player support
- **/roll:** Handles rolling dice logic, returning new position of player, triggering game events such as snakes/ladders/minigames.
- **/state:** Returns the game state which contains the player, snakes/ladders, minigames positions, and other important state variables such as player's turn.
- /**submit_answer**: Handles player's answers to the trivia and hangman minigames
- /**restart_game**: Clears all game data so joining players start fresh
- /**new_game**: Restarts the game so another can be played

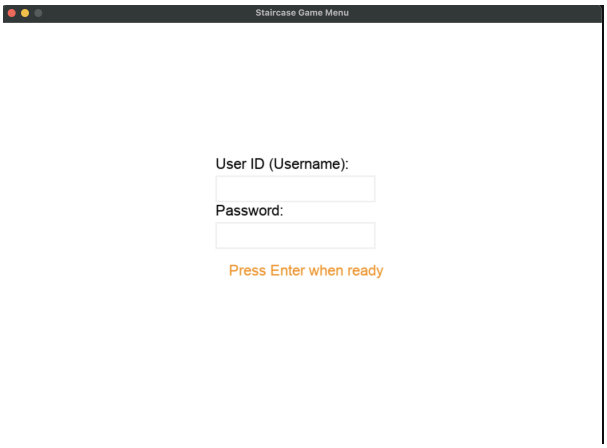
**LangChain Agents:** LangChain is a framework for developing applications powered by large language models, in our case Misteral AI into applications by managing how these models interact with data, tools, and users. In our project, LangChain acts as the intelligent engine behind the minigames, Trivia and Hangman, by generating dynamic, context-aware questions and puzzles that respond to user input. Rather than hard coding questions or answers, we used LangChain to build agents that tap into the power of language models, allowing for more natural, varied, and engaging gameplay.

StairCase LangChain agents are wrapped inside the Back End layer and our flask apis call these agents. There are many implementation methods but for purpose of our project we used creating Prompt template. During the course of this project we learn the nuances of prompt engineering.

**Firebase Authentication**

Firebase is a service that is a part of the Google Cloud platform, which handles user sign-up and sign-in. The benefit of using Firebase is that Google's API client libraries securely handle user information and passwords within the application. Users can sign up for the service within the website, which uses the NodeJS client library. When signing into the game, the Python SDK is used.

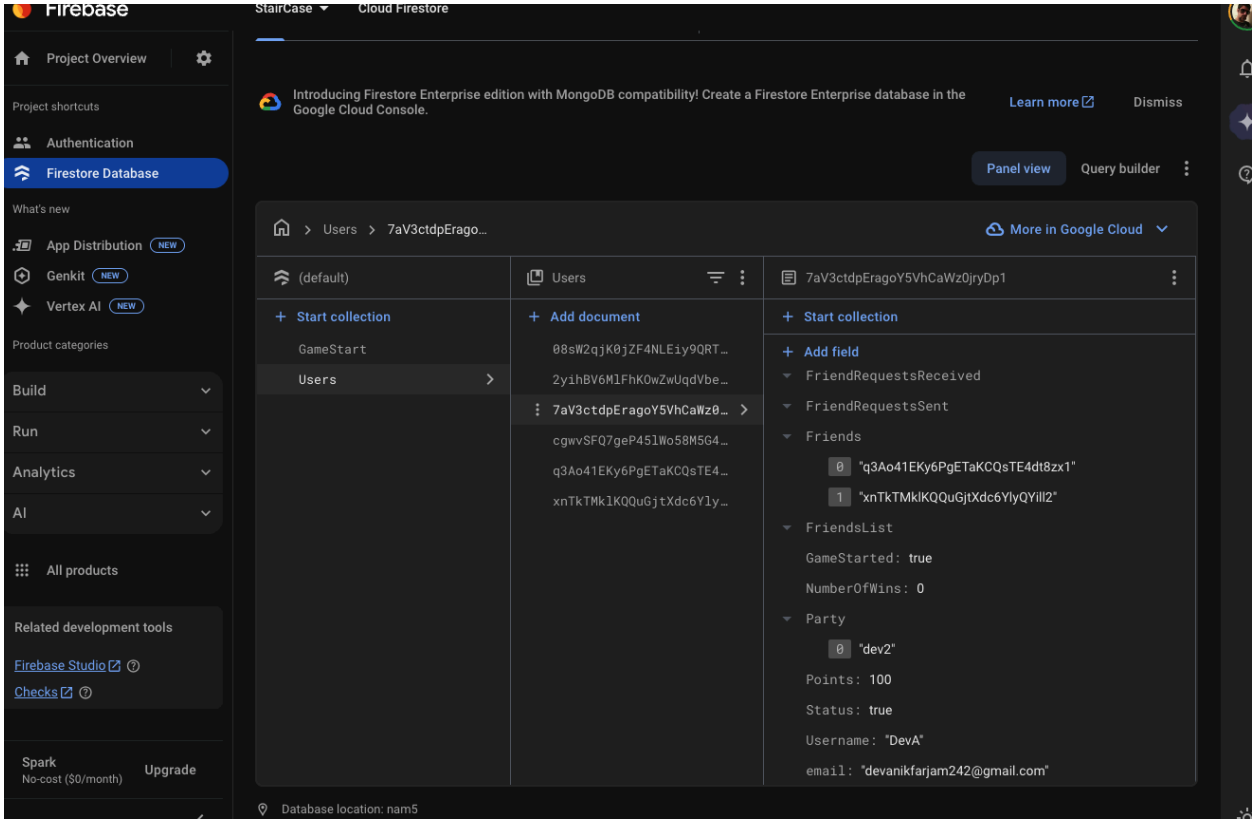


### Firestore DB

Firestore is a NoSQL database service within Google Cloud platform. Data is organized into alternating hierarchical sets of collections and documents.

In our database, our top-level collection is Users. Then, each document uses the Firebase ID as the document ID for each user's document. The user document contains information about the user, such as their online status, username, friends, game status, and global game statistics.

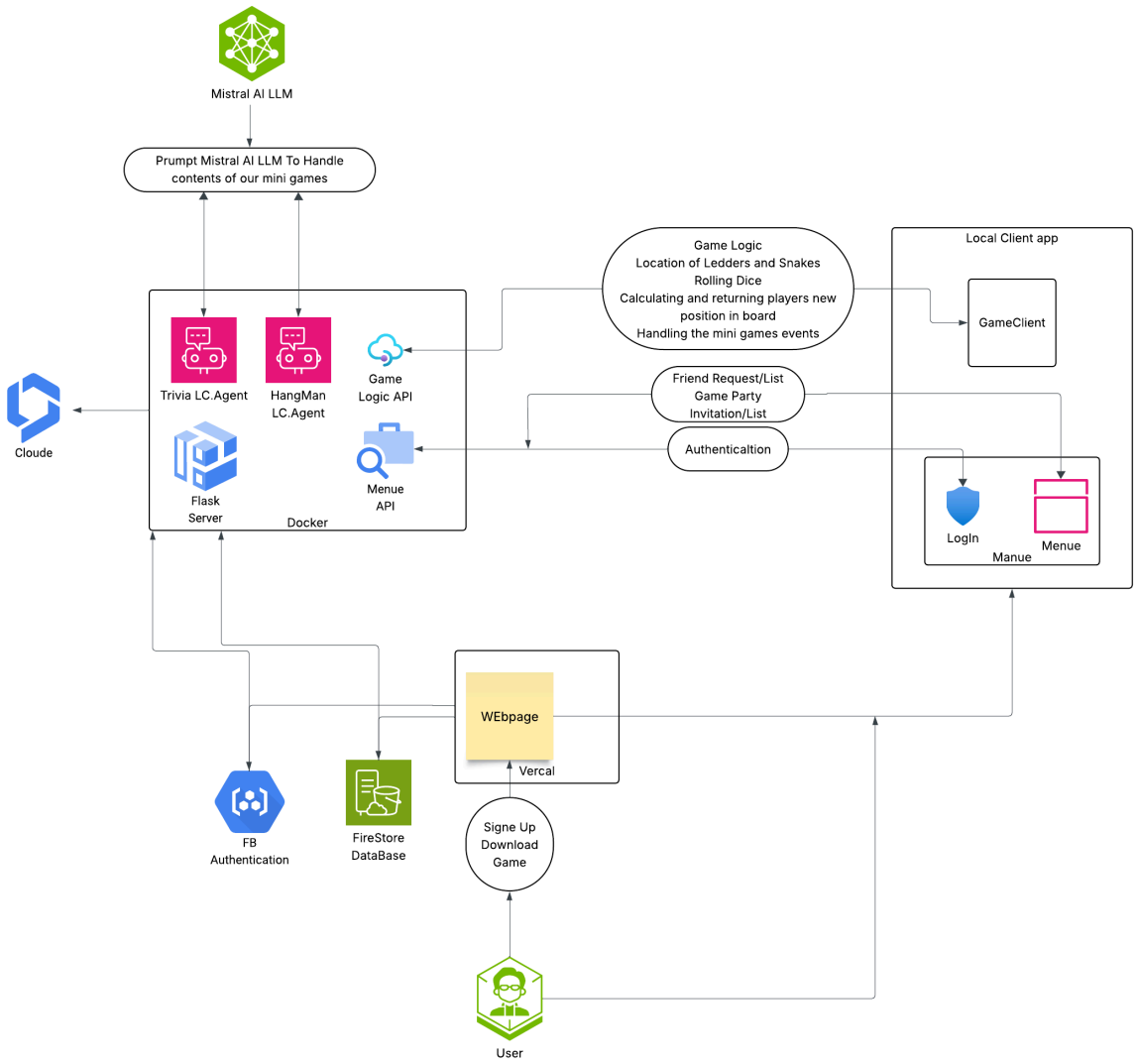



### Deployment

Our frontend website and the application server were deployed separately.

The frontend website was deployed using Vercel, which is a platform for quickly launching websites built with modern web frameworks such as React. We connected our Github repository, so that every push will automatically build and re-deploy the frontend server and keep the site up to date.

The application server was deployed as a Docker container to a service called NorthFlank. Our Docker container contains all necessary application scripts and exposes the port for the Flask server, and then runs the server. The built Docker image was pushed to DockerHub. NorthFlank is a deployment platform that is able to run Docker containers, so by pointing it to our DockerHub image, we are able to deploy our application server.




## 5. Testing

A local server was created on the computer to test the game between two clients. This was ran through the backend file by calling:

python3 -m BackEnd.stairCase_Server

Then the command below was run twice, on two different terminals. This simulates running two different clients and simulates players playing with each other in the game.

python3 -m Front_Game_Client.Menue

As soon as everything was operating as expected, we were good to go forward and deploy our game.

To test the Langchain, this was run on the test_server.ipynb notebook file. Prompt engineering was conducted to find the prompt that made sure the trivia minigame was returning the question in a proper format.

There is a unittest file in a separate folder in Front_Game_Client/tests/test_game_client.py. This file tests that our game board is calculating the coordinates of each position on the board correctly, so that the 10x10 board is drawn correctly. It tests various positions, such as position 1, 50, and 100, and checks that the correct coordinates are being returned.

**6. Challenges and Solutions**

One challenge we faced during this project was learning how to work with Pygame's coordinate system to draw the 10 x 10 board, calculate the coordinate positions for each grid, draw players on the board, and handle the animation between positions. Learning the coordinate system was very counterintuitive at first, for example, the y-axis for coordinates is inverted when working with Pygame, so a y-coordinate at 0 would be at the top of the screen instead of the bottom.

We addressed this problem by watching several Pygame tutorials and reading the documentation to familiarize ourselves with the library. Then, we drew a 'blueprint' of the Pygame GUI for our game on paper, labelling the coordinate distances between each button, text, component, etc. For example, for planning out the 10 x 10 grid, we drew the grid and labelled the coordinate lengths of each cell, the margin from the screen, etc. We then used this diagram to plan out the code from there.

**7. Individual Contributions**

| **Ashkan** | **Brendan** |
|---|---|
| Created Flas Server and registered the BP within the Flask app | Created Pygame GUI that draws the visuals for the game |
| Created Menu component for Front End Client | Drawing 10x10 game board using Pygame |
| Handled the logic of friend list and friends requests | Draw snakes, ladders, trivia cells, hangman cells |
| Handled the logic of Party invitation and signaling the start of the game | Drawing player movement and animations |
| Created Lanchain Agents For Hangman and Trivia game | Roll Dice, Restart Game buttons + logic |

| | |
|---|---|
| Implemented all the  FireBase Authentication API calls | Displaying game informative messages (player 1 rolled _ to position _, etc) |
| Created a FireStore DB for managing users friends list and stats | Draw minigame modals when player lands on one |
| Implemented all the api calls to database | Implement proper /roll endpoint logic that handles rolling dice |
| Created Front End web app for user sign up and download the game | Implement /new_game endpoint for Restart button that restarts game |
| Wrapping Back end into Docker container for deployment | Implement /init_board endpoint defining the positions of snakes, ladders, minigames that the state will store. |
| Hosted deployed backend container in NorthFlank | /submit_answer endpoint that handles submission of player's answers to minigame questions |
| Minor adjustments to the final code so it could run in the Deployed server | |
| Deployed Website to Vercel | |
| Created Test python Notebooks to test initial ideas of the game and agents before final implementations | |

**8. Conclusion & Future Work**

We have mostly implemented all the major features we planned out for this game, except for a few extra features. In the future, we could improve the logic of the Hangman Langchain agent so the gameplay is more intuitive and user-friendly. We have implemented the shop in the database, but due to time constraints, we didn't fully integrate it with our Pygame GUI, so it is inaccessible for now. We could definitely finish implementing the shop for players to purchase new game backgrounds, etc. in the future.