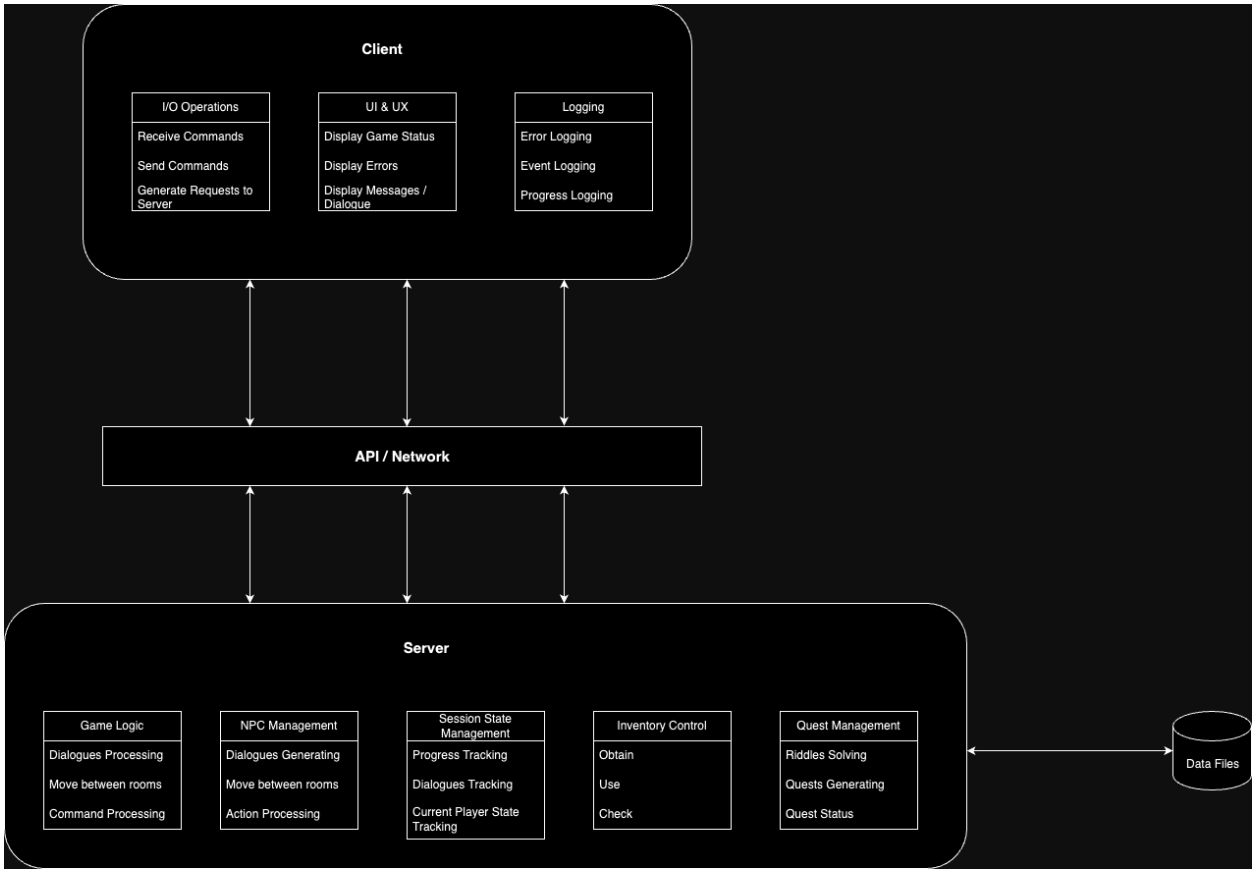
Group members:

- Vi Nguyen - 501240732
- Leo Lai - 500414796
- Jeffry Lien - 500638398
- Alireza Iman - 500901122
- Viri Nguyen - 501132181

Client - Server UML Architecture Diagram:

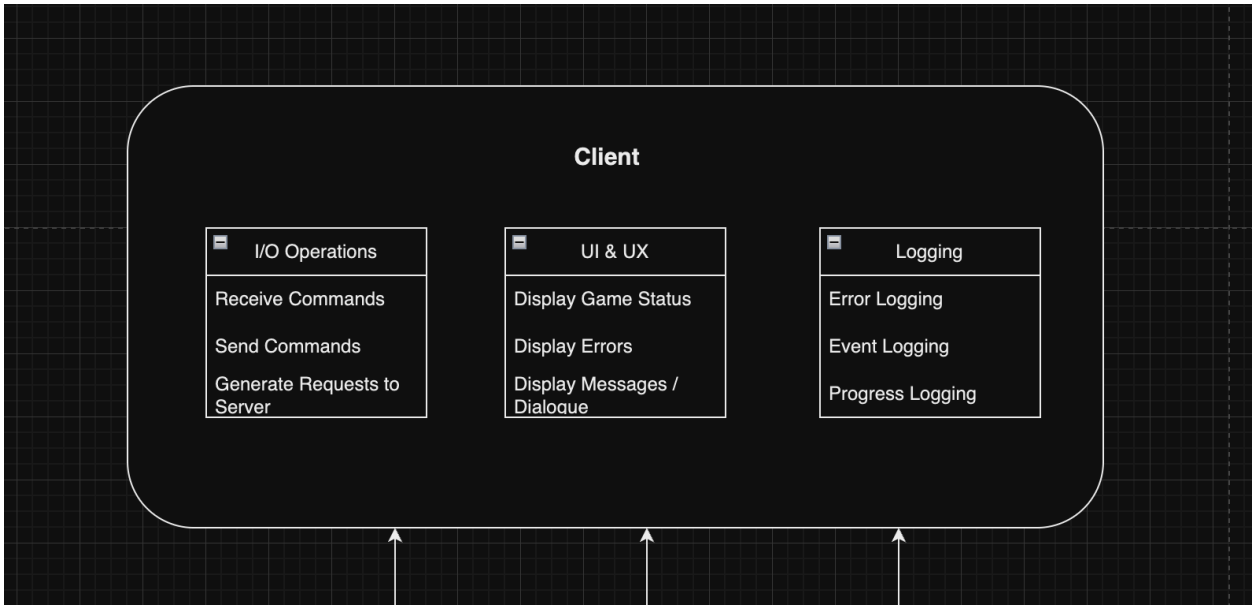


Following Neighbourhood Crossing's game view and logic, Client - Server is the most suitable architecture that provides both support and ease of usage and maintenance. Some of advantages of this approach include:

- Clear delegation of responsibilities: with the game's components separated by Client and Server side, the code will be more organized and easy to document.
- Flexibility in maintaining and scaling: bug fixes for servers won't affect clients, or adding more players or characters to the game won't break current players or characters.

There are 4 layers to our Client - Server system, we will be going through each layer one by one to demonstrate their responsibilities and outline their connections to each other to build the game world.

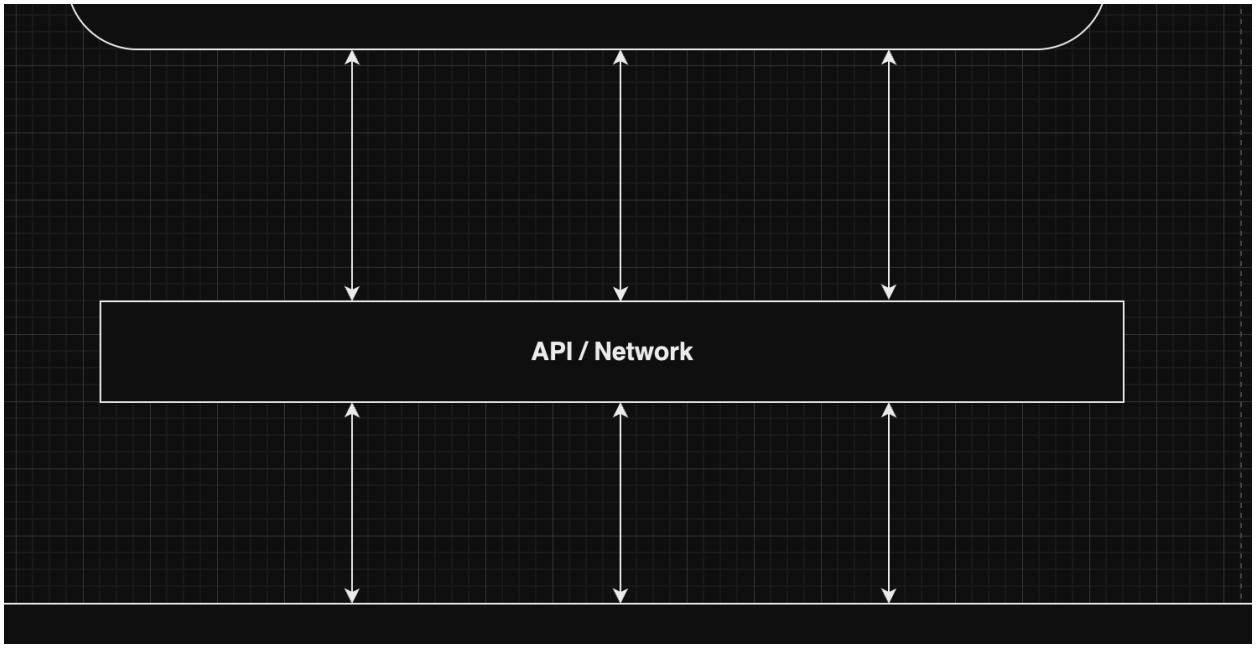
The first layer is the Client side:



There are 3 main components in the Client side, they all share the different responsibilities of the front-end logic of the game:

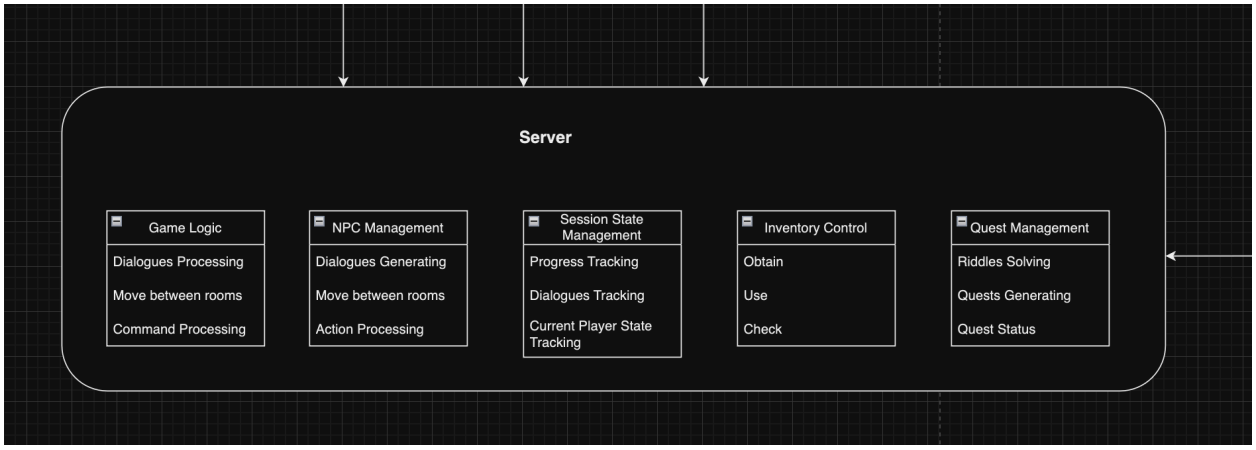
- I/O Operations: This component is responsible for all Input / Output operations of the game, including receiving commands from players, processing the command inputs, and sending those commands / outputs to the Server through the API layer.
- UI & UX: This component is responsible for all user interfaces and experiences, including displaying messages, tasks, quests, statuses, or errors. This component decides where all those parts of the UI go in order to create the best experience for the players.
- Logging: We decided to have a logging file in the front-end so that the player can use it to check their records for each play. This component keeps track of error logs, event logs and process logs throughout a player’s journey.

The second layer is the API / Network:



This layer's sole responsibility is to act as the communicator between the Client and Server layers, holding all API connections to communicate data back-and-forth between front and back ends. The front-end will make an API call to get the processed data from the back-end, and the back-end will send an API to the front-end that returns that data that's being asked for.

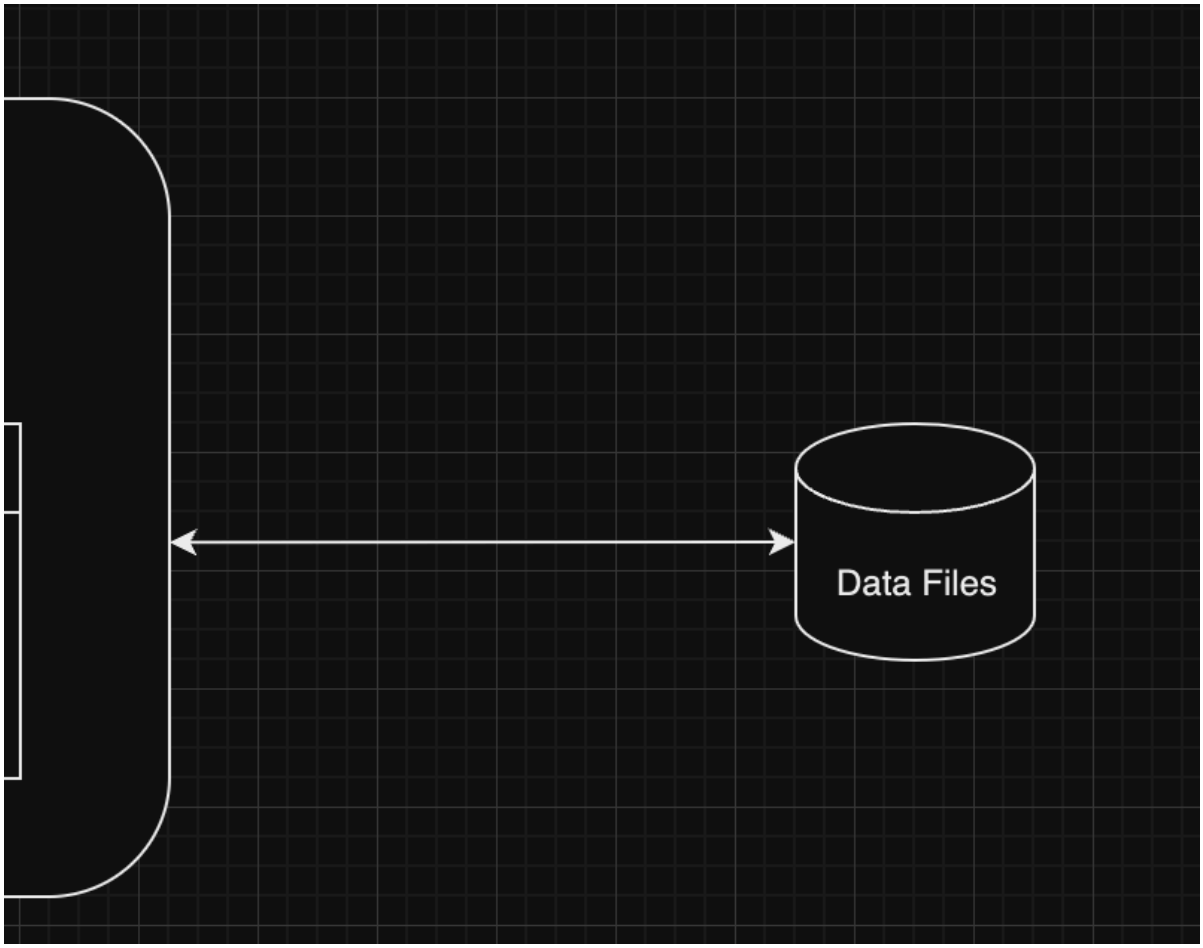
The third layer is the Server side:



This is the biggest layer of the game, where all of the game's logic and data processing acts happen. We tried to break it down as much as possible so that not any one component is too big or holds too much logic, that way it would be easier to maintain and scale moving forward.

- Game Logic: This component is responsible for all the basic game's logic, such as processing the dialogues, moving between rooms, and processing commands received from the Client side in order to send the commands to be appropriately handled.
- Session State Management: This component handles all game's states, such as progress tracking, dialogues tracking, or current player's state tracking. Essentially this component was considered to also be the game's logic, because it is, but tracking and updating states on its own is already a big part of the logic, it would be too much to have one component handle it all, so we broke it down into smaller components so that each can just focus on smaller tasks and produces more precise results.
- Inventory Control: This component is also a part of the game logic broken down, where here it only controls all inventory / items logic. It will update and read the items data file when the payer needs to obtain, use or check an item.
- Quest Management: Same thing here, this component is also a part of the game logic, here it just controls the game's tasks / quests. That includes activities such as solving riddles, generating the quests, or storing and updating quests' statuses.
- NPM Management: This component is a stand alone that will only handle logic for the 2 NPCs of the game. Since the NPCs don't do much in term of game logic, at most they will just be moving between rooms, having dialogues, or do some simple actions that would affect the story line but not much with the core logic, they can be handled neatly within a component. The component can easily be updated if more NPCs were to be added.

The last layer is the database, or in this case the json data files:



This layer’s core duty is to store data for the game, which is why it’s connected to the Server side as the back-end will be the one handling all game logic. The back-end will read the data files, get data from the files and do its processing responsibilities. Since we are storing all tracking logic in a log file, there won’t be any data return to the database. The files will just be a separate static component that gets updated independently following its original data structure design, which won’t affect the server code when making the request for data.