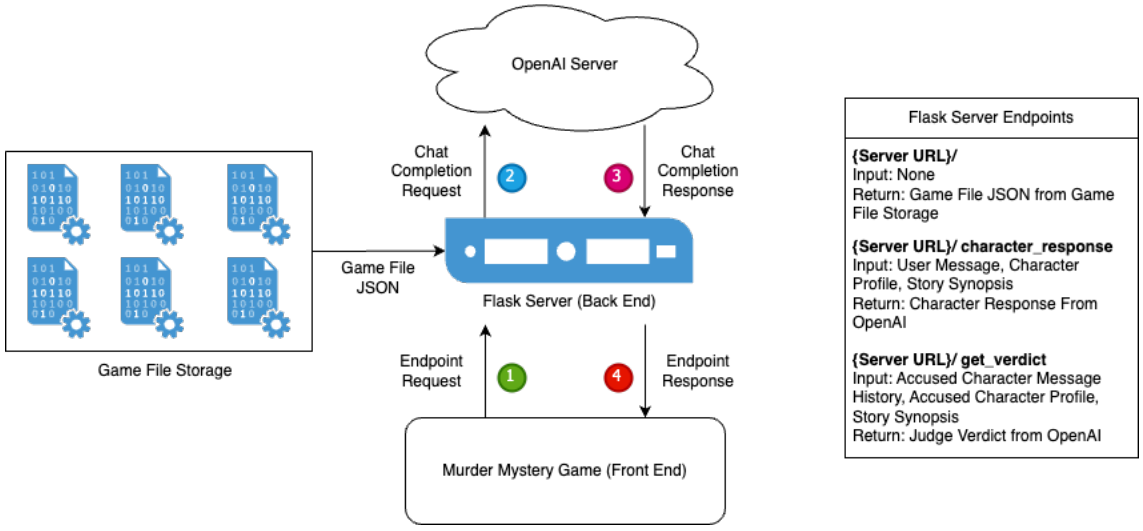
Tyler Webster – PUI Final Project

Project Overview:

Website Link: https://tmwebste.github.io/pui_project/
Website Github Repo: https://github.com/tmwebste/pui_project
Backend Server Github Repo: https://github.com/tmwebste/mystery_server



1. Project System Diagram

NOTE: This website is designed for desktop use. It should scale with a variety of monitor sizes.

Project Summary:

The purpose of the website is to provide a dynamic and unique experience each time a user plays a murder mystery game. Users are presented with an AI generated situation including a unique story and set of characters that they can interact with. The user interrogates each character via a text chat, where character responses are generated by OpenAI's ChatGPT API. This is intended to be a proof of concept demonstrating AI's ability to dynamically generate user guided content in the context of games. It is primarily aimed at users who have some interest in the possible uses of generative AI.

User Tasks:

- Start the game
  - Click the "See Instructions" or the "Skip Instructions" button
  - After reading or skipping the instructions the story summary is presented
    - Click the "Start Game" button to proceed to the main game
- Navigate the game
  - Click the "View Case" button on the bottom left to see the story summary again
    - Click the same button again to go back – It will now say "Back"
  - Click the "Menu" button on the bottom left to open the menu
    - Click the same button again to go back – It will now say "Back"
  - Click any of the character cards in the far-left section to see further details for the selected character
    - This will display the selected characters details in the main center section

- Interrogate characters
    - When a character is selected, the chat history and text input box and button where the user can send new messages will be shown on the right side of the screen
        - To send a message to the selected character, type your message in the box and click the send button
- Accuse a character
    - After a user has messaged a character, a button allowing them to accuse the selected character will appear in the character details in the center of the display.
    - Click the “Accuse” button to accuse them
        - This will disable the ability to further correspond with this character
        - The character details in the center will be replaced with the results of your accusation
        - Users can continue to interact with the other characters
            - If the user wishes to quit the game click the “Quit Game” button in the accusation result area in the center

Tools Used:

- Python
    - Python was chosen to program the backend because I am most familiar with this language and it integrates with the OpenAI Library very simply. It also supports very easy to create backend servers via the use of the Flask library.
    - Python is used to generate game stories via a JSON template whose fields are completed by responses from the OpenAI API. It is also used to handle communication between the web app, server, and OpenAI API.
    - Python adds the ability to easily produce new game stories and reliably retrieve data from the OpenAI API. This is the most optimal language to support this functionality based on my past experience.

- Flask
    - Flask was chosen because it is a well documented, simple and lightweight tool to create a backend server to be used for web based applications. A backend server was a critical component of this project because of the need to hide my OpenAI API key from a public facing repo.
    - This tool is responsible for serving pre-generated story files in the form of JSON objects to the client web app. It also handles queries from the client and returns new data generated by OpenAI like character messages and judge verdicts.
    - This tool is necessary to isolate my API key from the internet. It effectively acts as an API for interfacing with OpenAI within the context of my application, creating a more modular and maintainable system.

- OpenAI API
    - OpenAI was chosen because it is currently the most popular and well documented generative AI model available to the public. A generative AI model was necessary to support the fundamental concept of my project.
    - This tool is used to generate the information used to fill out a JSON story template I created to hold all the game information. It also generates responses to users messages to characters, as well as verdicts from the judge in the game.
    - This tool adds the ability to dynamically generate stories as well as responses to user inputs on the client side. It enables much of the interaction I wanted to achieve through this project.

- FlyIO
    - FlyIO was chosen because it is a free hosting service for simple web applications. It was the most straight forward option I found that would allow me to host my server with minimal configuration and tinkering.
    - FlyIO is used to host the Flask server and game story files remotely. The server can be queried through a basic URL and the server remains running 24/7.
    - This tool enables the ability to host the backend server in a simple way without exposing the API key. It also allows the server to be queried from anywhere in the world without crazy network configuration.

- React VITE
    - Vite was used because it has some efficiency and implementation benefits over vanilla React. It allows me to compile and test quicker than normal React.
    - Vite is used in place of React. It has a similar structure and syntax but has some quality of life improvements.
    - This tool allows me to reuse components like react. It also give me the ability to utilize Github actions to automatically deploy my site whenever I push to the main branch.

Development Process:

Early on in my project, I focused heavily on implementing the OpenAI API functionality into the backend. This process involved a lot of prompt engineering, it took a lot of iterating to get the API to return responses with expected values and consistent formatting. I felt that this was the most critical to complete early as it would determine a lot of the implementation needed for the front end to interface with the server and utilize the data returned. After the API functionality was implemented, I turned my attention to connecting the front end to the server by displaying the raw story information and a character response to a hard coded question. I then built out the interaction for one character (messaging and character information display), with the plan to replicate that format for the other characters in the game. I made a series of reusable components that would allow to display and interact with different characters based what the user selected. After this main page was completed, I added additional pages for the game description, instructions, story introduction and menu. The final part I completed was the end screen users would see when they accused a character and were told if they guessed the correct person.

Challenges:

The biggest challenge was getting the OpenAI API to behave in a consistent manner, especially when generating a story. Originally, I was asking the AI to fill in all the blanks in the story template at one time. This often resulted in an incomplete story, frivolous comments from the AI, or a JSON file with different key values that would break the game when the front end attempted to use the data. To resolve this, I created a series of smaller queries that would complete individual components of the game file, resulting in consistently formatted game files which could be generated in batches with minimal need for manual corrections.

Appendix: Screenshots from WAVE