Project 12

Team Member: Taii Simon-Brown

To start off with, let's talk about the API used for this scenario. Open AI API is what allows you to connect your apps and/or projects to OpenAI’s AI models. AKA ChatPGT models. You can send text, code, or images to the AI and get smart responses back automatically.

Note these responses depend on the Open AI model that you use. For this project, I used GPT-4.1 for the advanced responses. (Also because it was the only one within my budget plan.)

(YES. You have to pay to use OpenAI.)

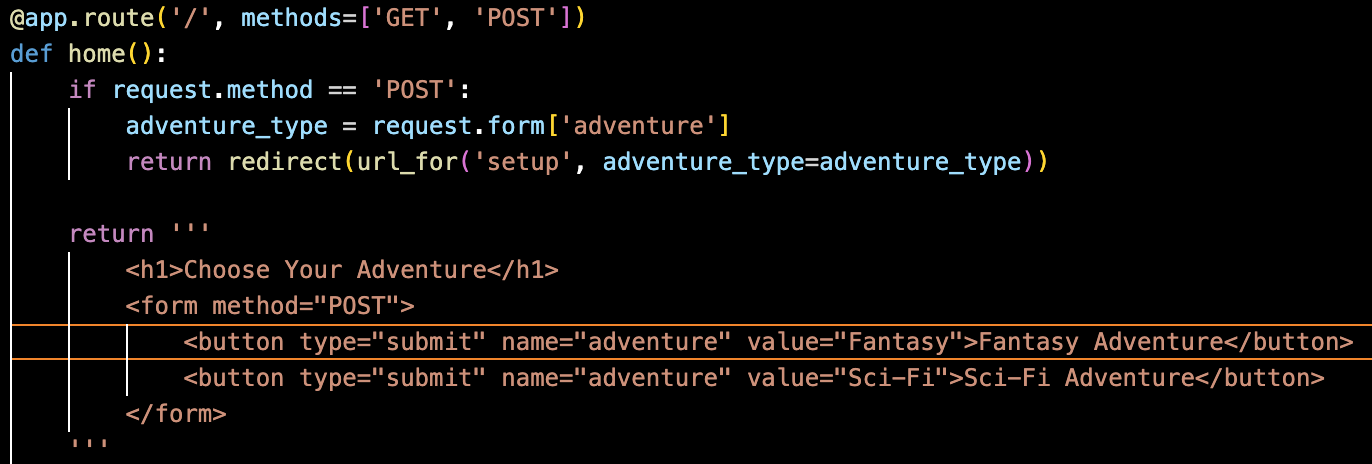
This Flask-based web application is a text adventure game powered by OpenAI's GPT-4. Users begin by choosing an adventure type (Fantasy or Sci-Fi) and then set up their character by providing a name, special skill, and goal. The game stores each session in memory and progresses through a /play route where GPT narrates the unfolding story. Based on the player’s actions, GPT continues the narrative and offers three possible choices for what to do next. The user interacts via a simple HTML form, and the game can be exited at any time through a dedicated /exit route that ends the session with a thank-you message.

from flask import Flask, request, redirect, url\_for

import openai

app = Flask(\_\_name\_\_)

Since this project is to make a game, I decided to use Flask to turn into an app/web application for fun.   
Flask creates the app.   
Request is to handle incoming HTTP data.   
Redirect and url\_for is to navigate between routes



Route to Home Page with function home()

Purpose:

Displays a simple form for users to choose an adventure type and redirects them based on their selection.

How it works:

If the request is a POST (form submitted):

Retrieves the selected adventure type from the form.

Redirects to the /setup route, passing the adventure type as a URL parameter.

If the request is a GET:

Displays an HTML form with two buttons: “Fantasy Adventure” and “Sci-Fi Adventure”.



Route to Setup Page with function setup()

Purpose:

Collects details from the user to set up a personalized adventure game.

How it works:

GET request: Displays a form asking for the hero's name, special skill, and goal, based on the selected adventure\_type.

POST request:Retrieves user input from the form.

Creates a new game entry with a user's name, the special skill they wish to use for the quest, and the goal of this adventure. As well saving the game history.

Redirects to the play route/ game page to begin the adventure.

@app.route('/play', methods=['GET', 'POST'])

def play():

game\_id = request.args.get('game\_id')

game = games.get(game\_id)

if not game:

return "<h1>Game not found. Please start a new adventure.</h1>"

if request.method == 'POST':

player\_action = request.form['action']

game['history'] += f"\nPlayer: {player\_action}\n"

# Build the prompt

prompt = f"""

You are narrating a {game['adventure\_type']} text-based adventure game.

The hero's name is {game['hero\_name']}.

Their special skill is {game['special\_skill']}.

Their ultimate goal is {game['goal']}.

Story so far:

{game['history']}

Describe what happens next and ask the player what they want to do.

"""

#Calling GPT to respond to player has chosen

response = client.chat.completions.create(

model="gpt-4",

messages=[

{"role": "user", "content": prompt}

]

)

story = response.choices[0].message.content

game['history'] += f"Narrator: {story}\n"

return f'''

<h1>Adventure: {game['adventure\_type']}</h1>

<h2>{game['hero\_name']} - {game['special\_skill']}</h2>

<p><b>Goal:</b> {game['goal']}</p>

<hr>

<div style="white-space: pre-wrap;">{story}</div>

<hr>

<form method="POST">

Your Action: <input type="text" name="action">

<button type="submit">Continue</button>

</form>

<br>

<form action="/exit" method="GET">

<button type="submit">Quit Adventure</button>

</form>

'''

Route to Setup Page with function play()

Purpose:

Continues the user’s text-based adventure by generating story updates using GPT based on the player's actions. The text-based adventure will continue on until the goal is reached. Or the user quits/exits the app/program.

How it works:

Retrieves the current game session using game\_id.

If no game is found, show an error message.

On POST:

User submits an action in the text bar.

Adds the player’s input to the game’s history.

Builds a prompt using the adventure type, hero name, skill, goal, and story so far.

Sends the prompt to GPT-4 to generate the next part of the story.

Updates the history with GPT’s narration and displays it along with an input form for the next action.

@app.route('/exit')

def exit\_game():

return '''

<h1>Thanks for playing!</h1>

<p>Your adventure has ended.</p>

<a href="/">Return to Home</a>

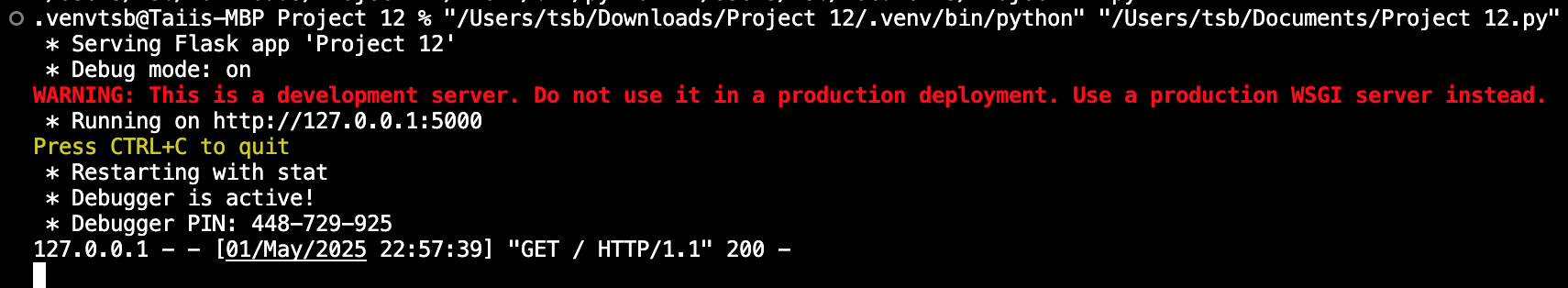
''

Purpose:   
Exiting the game. Users may end their adventure whenever they feel like it.

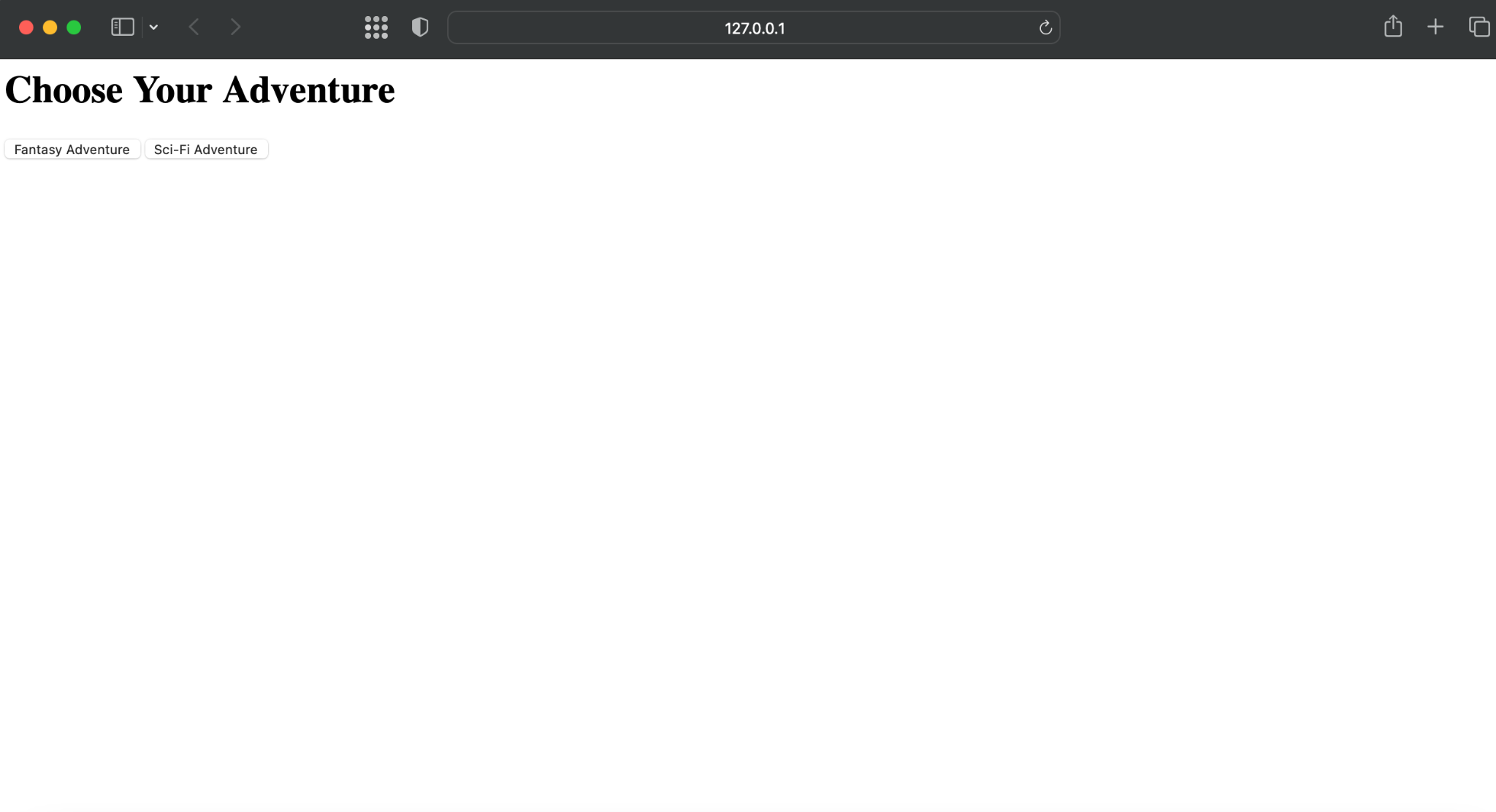
How it works:

Exiting also allows the user to return to the homepage and start their adventure again. The game history is only erased when you select a new adventure.

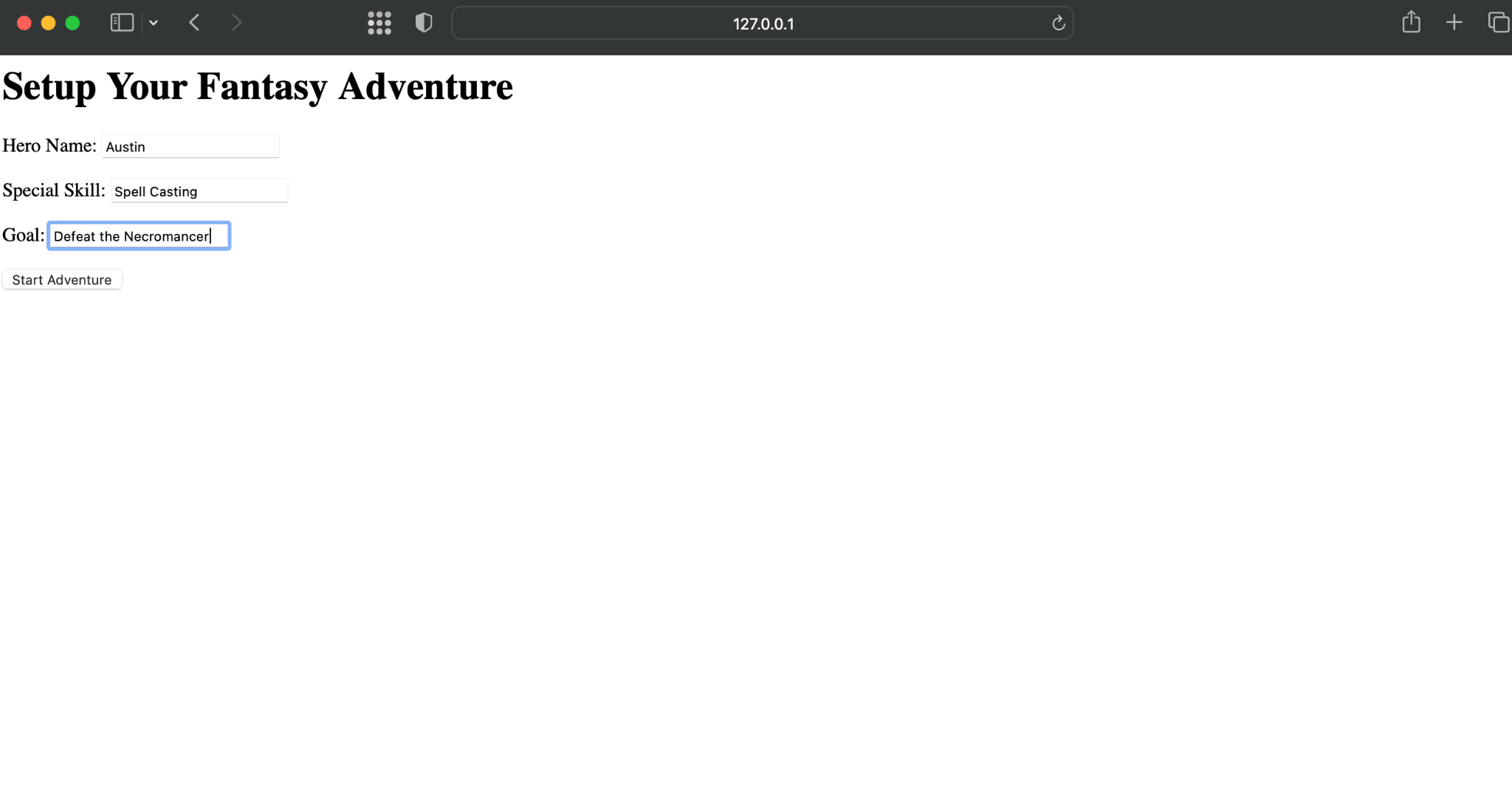
Demonstration



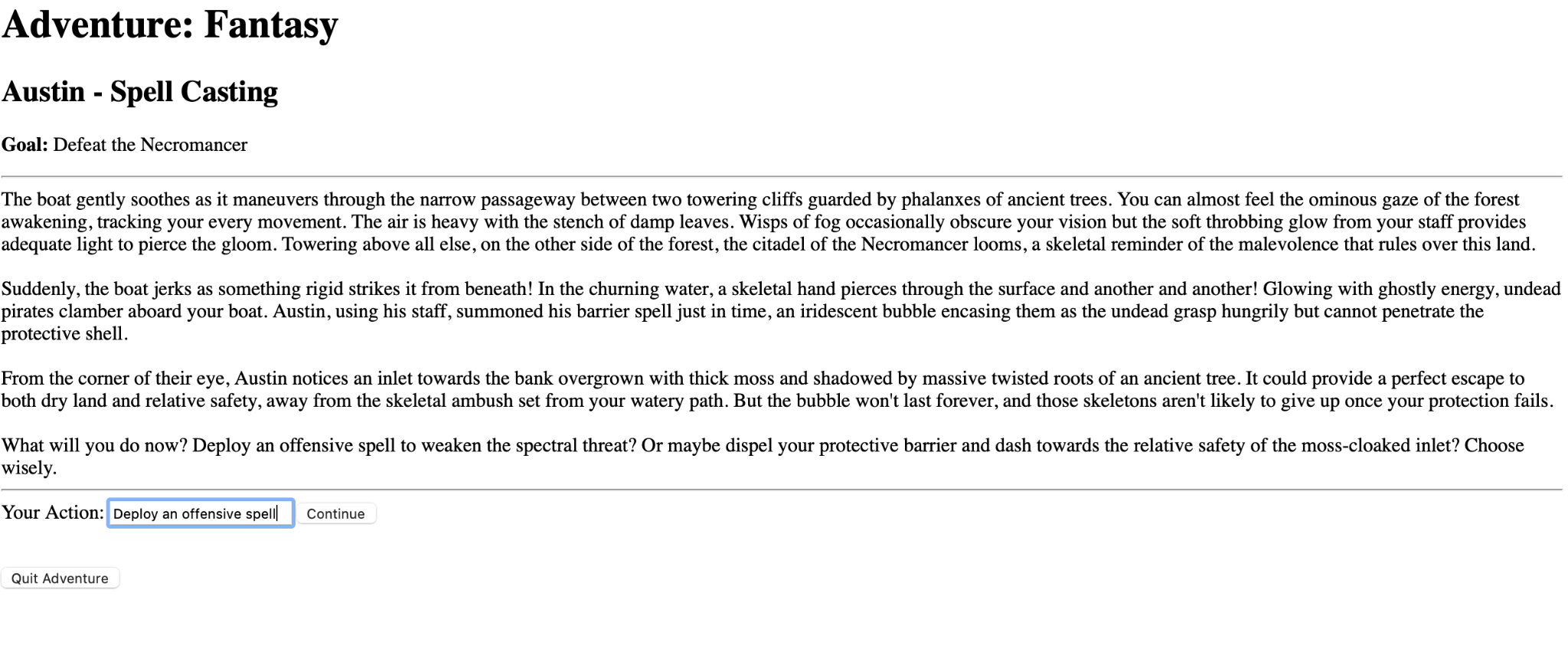
When I initizale the program



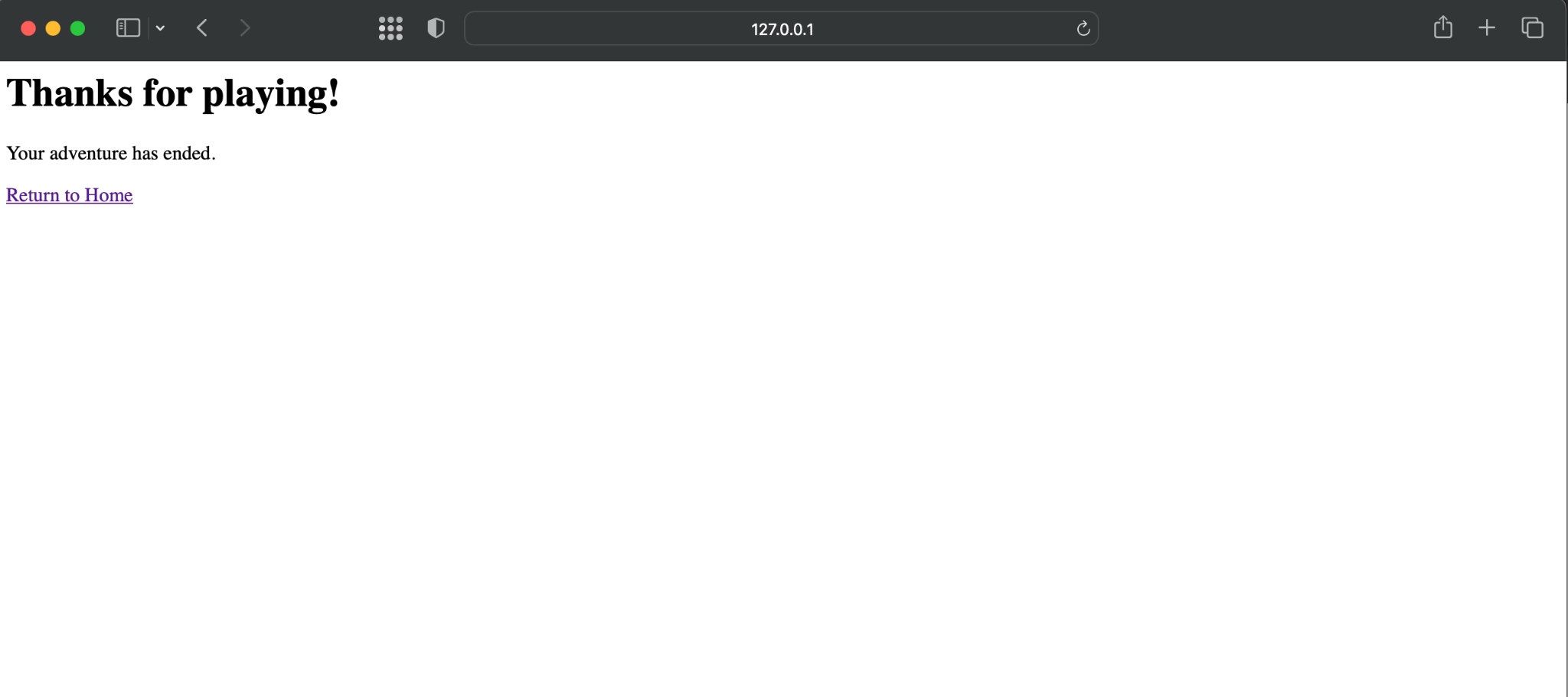
The Homepage



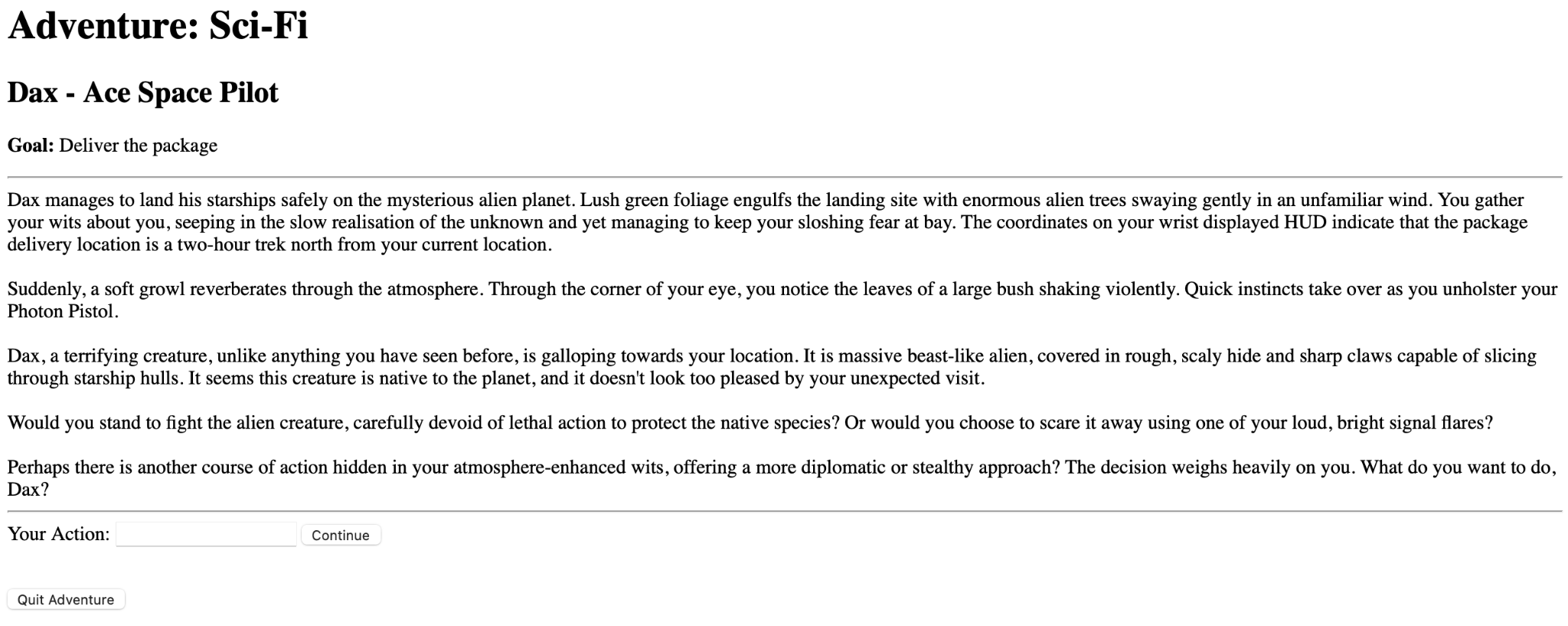
The Setup Page



The Play Page



The exit/end page.



A Sci-Fi adventure playthrough.