

Building an AI Doctor API using chatGPT's Language Model.



In the ever-evolving landscape of artificial intelligence, developers are continually seeking innovative ways to enhance user experiences, automate tasks, and provide valuable solutions. The emergence of powerful language models has opened up new horizons in AI development, and one such model that has been at the forefront of this revolution is GPT-3, developed by OpenAI. GPT-3 stands as a testament to the potential of AI to transform our digital interactions, and in this technical article, we'll explore how you can leverage this remarkable language model to create a simple chatbot tailored exclusively to medical health.

We will delve into the technical details of setting up a Node.js application, making POST requests to an API, and customizing prompts to ensure that the chatbot stays on topic, providing accurate and contextually relevant responses to medical health inquiries. This article aims to provide you with the tools and knowledge to create a specialized chatbot powered by GPT-3.

Prerequisites

This tutorial focuses on explaining the Node.js API component, thus you will need a basic understanding of HTML, CSS and JavaScript.

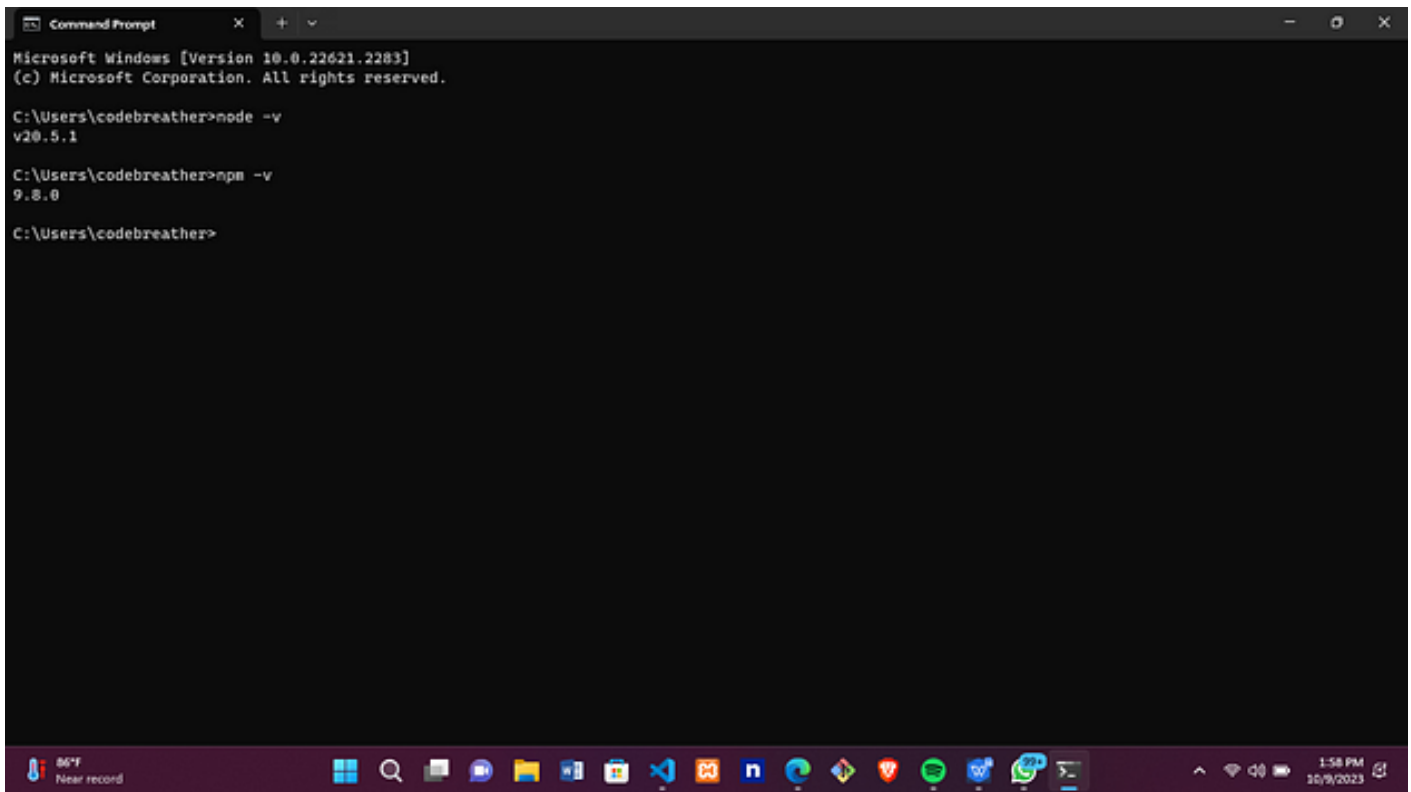
Before we dive into the code, we need to ensure that we have the following tools:

- Node.js
- npm
- VSCode

If you don't have Node.js installed, you can download it from the [official Node.js website](#). Installing node.js also installs npm. Confirm that Node.js and npm is installed using the following commands:

```
node -v  
npm -v
```

In my case I'm running the following versions of Node.js and npm:

A screenshot of a Windows Command Prompt window. The title bar says "Command Prompt". The window content shows the following text: "Microsoft Windows [Version 10.0.22621.2283] (c) Microsoft Corporation. All rights reserved. C:\Users\codebreather>node -v v20.5.1 C:\Users\codebreather>npm -v 9.8.0 C:\Users\codebreather>". The taskbar at the bottom shows various application icons and the system clock indicating 1:58 PM on 10/9/2023.

```
Microsoft Windows [Version 10.0.22621.2283]
(c) Microsoft Corporation. All rights reserved.

C:\Users\codebreather>node -v
v20.5.1

C:\Users\codebreather>npm -v
9.8.0

C:\Users\codebreather>
```

Node.js enables us to run JavaScript on the server while npm (Node Package Manager) is primarily used for managing and distributing JavaScript libraries and packages that developers can use in their web and Node.js applications.

To set up the project, we will:

- Create a new folder for our project and create .env and app.js files inside:

```
API
| --- .env
| --- app.js
```

The two files will be used later to set up our API.

- Navigate to that folder in the terminal.
- Run the following command to initialize a new Node.js project:

```
npm init -y
```

This will create a package.json file in our project directory. This file will store information about our project, such as its name, version, and dependencies.

Once we have our project set up, we can start coding!

The first step is to install the required packages. On your terminal, still in your project folder, run the following command:

```
npm install openai express body-parser dotenv cors
```

This will install the necessary packages that we will use for our API. We then have to import those packages.

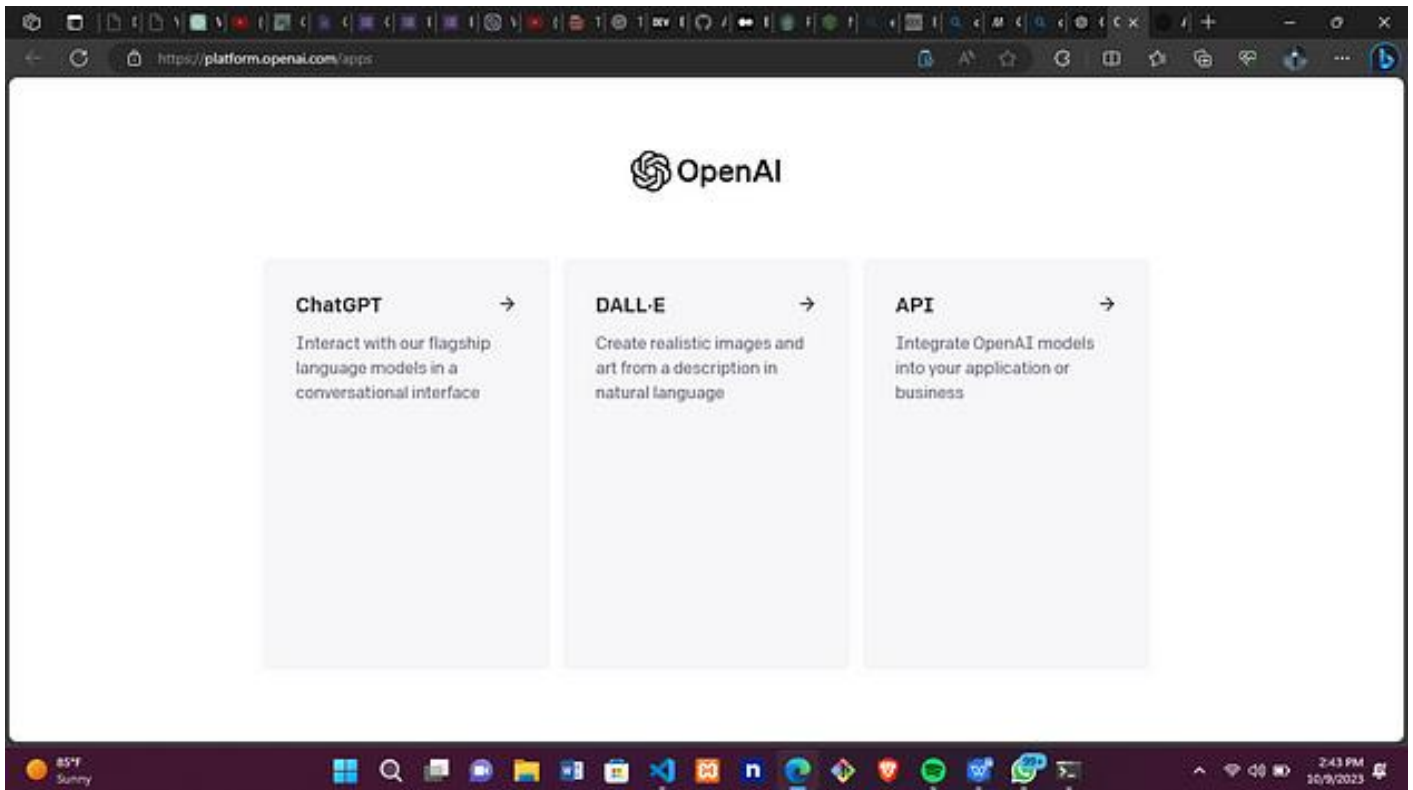
```
const { Configuration, OpenAIApi } = require('openai');  
const express = require('express');  
const bodyParser = require('body-parser');  
const cors = require('cors');  
require('dotenv').config();
```

We will be using the `openai` package to communicate with the OpenAI API. We also need `express` to create our API. Express is a popular lightweight framework that makes server-side programming in Node.js easier with its robust set of features and utilities. `body-parser` module parses incoming HTTP requests particularly when dealing with POST, PUT and PATCH requests, `cors` module controls how our chat box interface can request and interact with resources from our API which is running on a different port locally. Lastly, we use `dotenv` to load our environment variables.

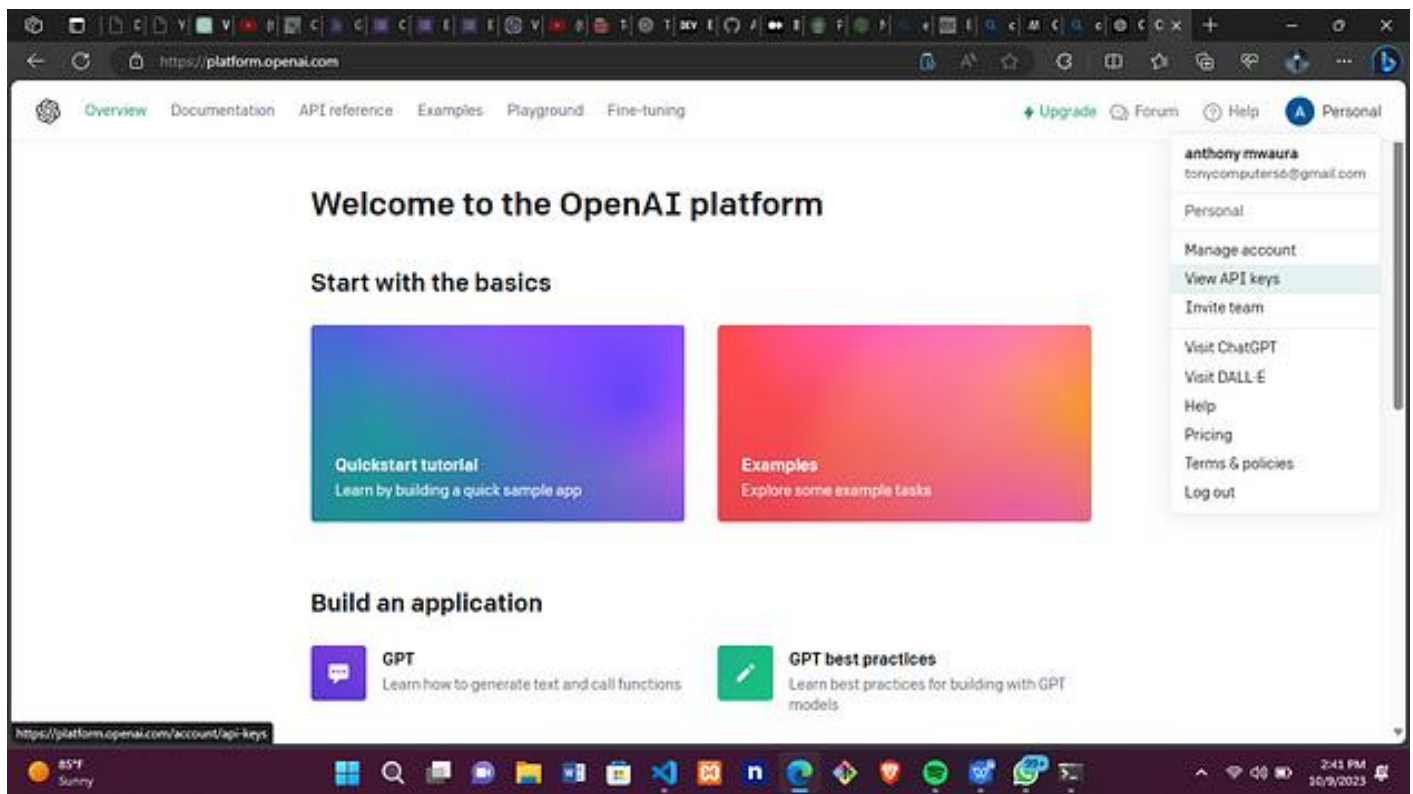
Navigate to .env file that we created earlier and write the following:

```
OPENAI_KEY=your-open-api-key-goes-here
```

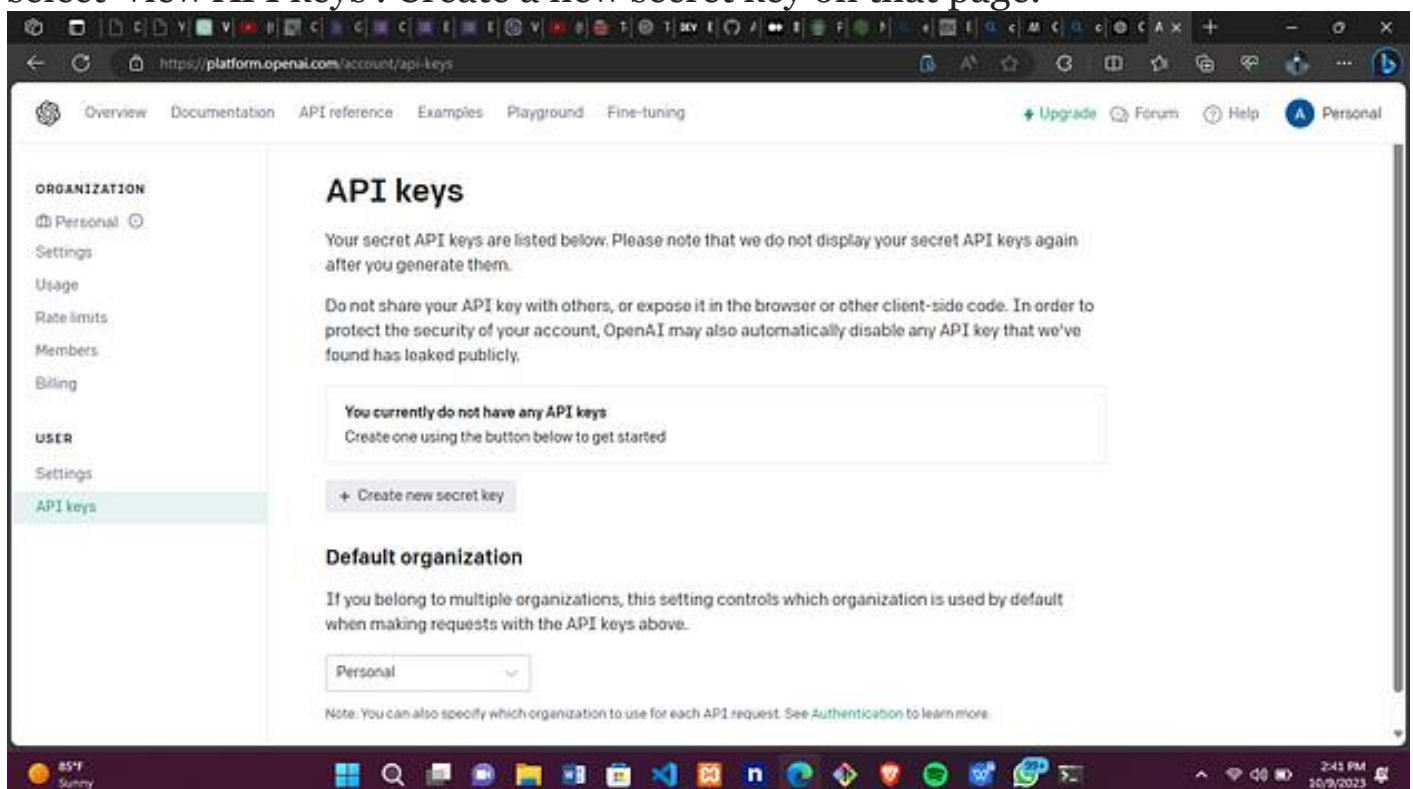
To obtain your openai key, you must first register with [openai](https://openai.com). After you sign up, login and you will be offered with the following options, of which you should select API.

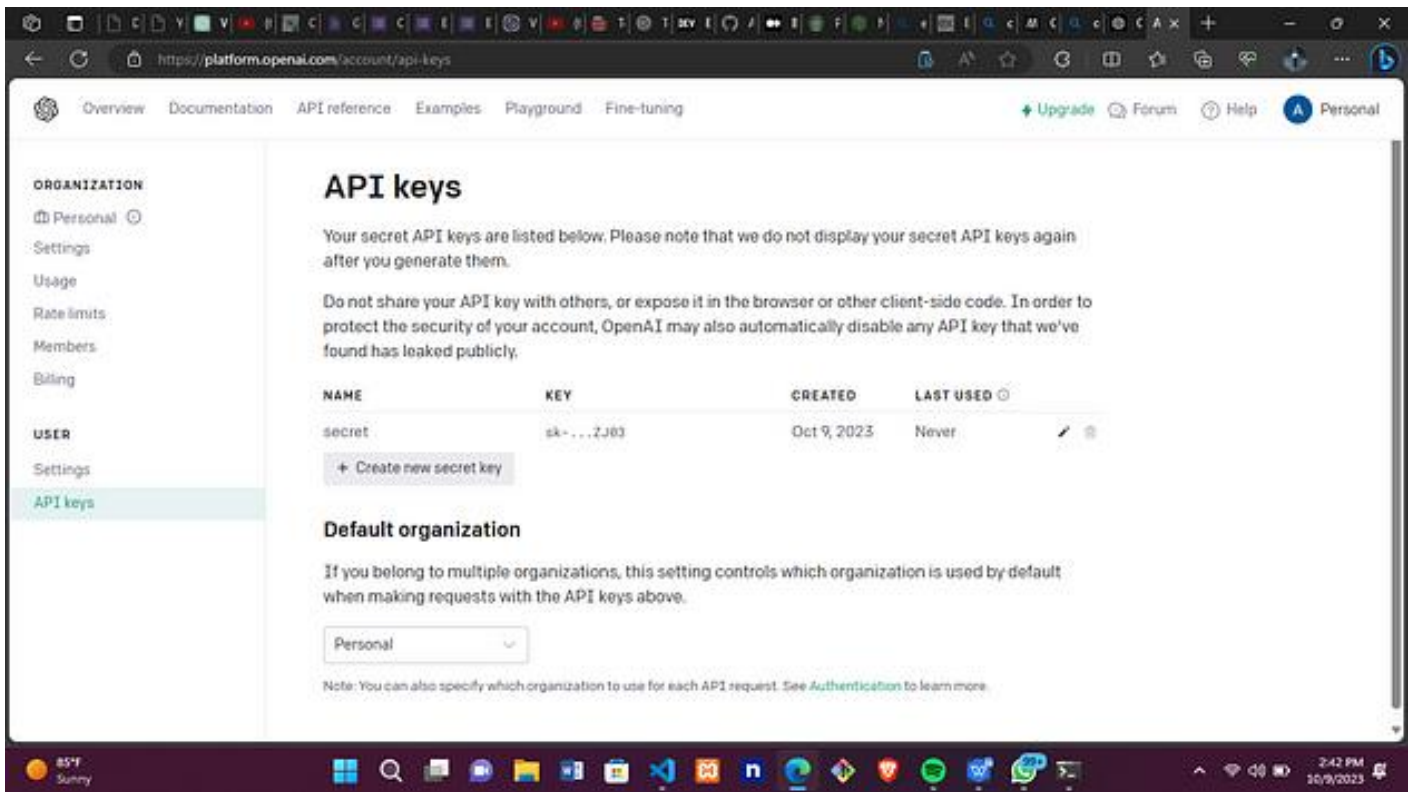


Your screen should look something like this:



Navigate to your profile on the top right and click to be presented with options; select 'view API keys'. Create a new secret key on that page:





Copy that key and paste it on your .env file in the OPENAI_KEY variable.

Next, we create our express app and set up the middleware with the help of express:

```
const app = express();
app.use(bodyParser.json());
app.use(cors());
```

We are using `body-parser` to parse incoming requests with JSON payloads and `cors` to enable [Cross-Origin Resource](#) Sharing. Now, let's set up our OpenAI API credentials:

```
const config = new Configuration({
  apiKey: process.env.OPENAI_KEY
});

const openai = new OpenAIApi(config);
```


We create a new instance of Configuration with our OpenAI API key, which is loaded from our environment variables using `process.env.OPENAI_KEY`. We then pass this configuration to create a new instance of the OpenAIApi class.

Now that we've set up our middleware and API credentials, let's create our endpoints. We start with the root endpoint that simply returns a welcome message:

```
app.get('/', (req, res) => {  
  res.send('Welcome to the Medical Health Assistant API with GPT-3 language model');  
});
```

This endpoint returns a simple welcome message when a GET request is made to the root URL.

Next, we define our prompt, which will be used as a base prompt for every message:

```
const prompt = `You are an AI assistant that is an expert in medical health and is  
part of a hospital system called medicare AI  
You know about symptoms and signs of various types of illnesses.  
You can provide expert advice on self-diagnosis options in the case where an illness  
can be treated using a home remedy.  
If a query requires serious medical attention with a doctor, recommend them to book an  
appointment with our doctors  
If you are asked a question that is not related to medical health respond with "Im  
sorry but your question is beyond my functionalities".  
Do not use external URLs or blogs to refer  
Format any lists on individual lines with a dash and a space in front of each line.  
>`;
```

This defines a constant that contains the prompt that will be used to generate responses from the GPT-3 language model. You can tailor this prompt to your liking.

Now that we have our prompt defined, let's move on to handling the message processing. We define a POST endpoint `/message` to receive incoming messages and generate responses using the GPT-3 model:

```
app.post('/message', (req, res) => {
  const response = openai.createCompletion({
    model: 'text-davinci-003',
    prompt: prompt + req.body.message,
    temperature: 0.5,
    max_tokens: 1024,
    top_p: 1,
    frequency_penalty: 0,
    presence_penalty: 0
  });

  response
    .then((data) => {
      const message = { message: data.data.choices[0].text };
      res.send(message);
    })
    .catch((err) => {
      res.send(err);
    });
});
```

This endpoint receives a POST request containing a message in the request body. We use the `openai.createCompletion()` method to generate a response from the GPT-3 model. The method takes various parameters such as the model, prompt, temperature, max_tokens, top_p, frequency_penalty, and presence_penalty. These parameters influence the behavior of the response generation process.

Once we receive the response from the GPT-3 model, we extract the generated message and send it back as a response to the client.

Lastly, we start the server by listening on a specific port, in this case, port 3000:

```
app.listen(3000, () => console.log('Listening on port 3000'));
```

While you are still in your project's directory in the terminal, enter the following command to start the server:

```
node app.js
```

```
anthony@CodeBreather:~/Documents/programs/nodeGPT-3_FINAL/api$ node index.js
Listening on port 3000
```

Chatbot Interface

The interface is made up of HTML, CSS, and JavaScript code that creates the chat box. The code for this interface is kept separate from the API folder.

HTML:

```
<html>
  <head>
    <title>GPT-3</title>
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <div class="chat-box">
      <!-- Chat Box Header -->
      <div class="chat-box-header"> AI chat</div>

      <!-- Chat Box Body -->
      <div class="chat-box-body">
        <div class="response">
          <span>Hello there, how can I help you today?</span>
        </div>
      </div>

      <!-- Chat Box Footer -->
      <div class="chat-box-footer">
        <input type="text" class="input-box" placeholder="Ask a question..." />
        <button>Send</button>
      </div>
    </div>

    <script src="index.js"></script>
  </body>
</html>
```

CSS:

```
@import
url('https://fonts.googleapis.com/css2?family=Roboto:wght@400;700&display=swap');
/* Add a background color to the chat screen */
body {
    font-family: 'Roboto', sans-serif;
    background: #E8EBF5;
    background-size: cover;
    background-position: center center;
}

#loading {
    font-size: 30px;
}

/* Style the chat box */
.chat-box {
    margin: 80px auto;
    width: 400px;
    max-width: 100%;
    background-color: #ffffff;
    border-radius: 8px;
    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
    overflow: hidden;
    height: 600px;
}

/* Style the chat box header */
.chat-box-header {
    background-color: #846c38;
    color: #ffffff;
    padding: 16px;
    font-size: 20px;
    font-weight: bold;
    border-top-left-radius: 8px;
    border-top-right-radius: 8px;
}

/* Style the chat box body */
.chat-box-body {
    padding: 16px;
    overflow-y: auto;
    height: calc(100% - 145px);
}

/* Style the chat messages */
.chat-box-body .message,
.chat-box-body .response {
    margin-bottom: 16px;
}

.message {
    color: rgb(44, 41, 41);
    font-family: Helvetica;
    font-size: 16px;
    font-weight: normal;
    text-align: right;
    clear: both;
}
```

```
.message span {
  line-height: 1.5em;
  display: inline-block;
  background: #F0EEED;
  padding: 20px;
  border-radius: 8px;
  border-bottom-right-radius: 2px;
  max-width: 80%;
  margin-right: 10px;
  animation: floatup .5s forwards
}

/* Style the chatbot's responses */
.response {
  color: white;
  font-family: Helvetica;
  font-weight: normal;
  font-size: 16px;
  text-align: left;
}

.response {
  line-height: 1.5em;
  display: inline-block;
  background: #0084ff;
  padding: 10px;
  border-radius: 8px;
  border-bottom-left-radius: 2px;
  max-width: 80%;
  margin-left: 10px;
}

/* Style the chat box footer */
.chat-box-footer {
  bottom: 0;
  display: flex;
  align-items: center;
  background-color: #846c38;
  border-top: 1px solid #846c38;
  padding: 8px 16px;
}

/* Style the input field */
.chat-box-footer input[type="text"] {
  flex: 1;
  padding: 8px;
  border: 1px solid #846c38;
  border-radius: 8px;
  font-size: 16px;
  outline: none;
}

/* Style the button */
.chat-box-footer button {
  margin-left: 8px;
  padding: 8px 16px;
  background-color: #0084ff;
  color: #ffffff;
  font-size: 16px;
  font-weight: bold;
  border: none;
  border-radius: 8px;
  cursor: pointer;
}
```

```

/* Add some hover effects to the button */
.chat-box-footer button:hover {
  background-color: #0073e6;
}

.chat-box-footer button:active {
  background-color: #005bbf;
}

```

The CSS code provided in the style.css file adds styles to the chatbox and its components. It defines the layout, colors, fonts, and animations for the chat box UI:

The JavaScript code handles the chat box functionality. It listens for user interactions, such as clicking the send button or pressing the enter key, and sends the user's message to the server for processing:

```

const chatBox = document.querySelector(".chat-box");
const inputField = chatBox.querySelector("input[type='text']");
const button = chatBox.querySelector("button");
const chatBoxBody = chatBox.querySelector(".chat-box-body");

// Send message on button click
button.addEventListener("click", sendMessage);

// Send message on enter key press
inputField.addEventListener("keypress", function(event) {
  if (event.key === "Enter") {
    sendMessage();
  }
});

// Function to send message to the server
function sendMessage() {
  const message = inputField.value;
  inputField.value = "";
  chatBoxBody.innerHTML += `<div class="message"><span>${message}</span></div>`;
  chatBoxBody.innerHTML += `<div id="loading" class="response loading">.</div>`;
  scrollToBottom();

  // Simulating loading animation with dots
  window.dotsGoingUp = true;
  var dots = window.setInterval(function() {
    var wait = document.getElementById("loading");
    if (window.dotsGoingUp)
      wait.innerHTML += ".";
    else {
      wait.innerHTML = wait.innerHTML.substring(1, wait.innerHTML.length);
      if (wait.innerHTML.length < 2)
        window.dotsGoingUp = true;
    }
  }, 100);
  if (wait.innerHTML.length > 3)
    window.dotsGoingUp = false;
}

```

```

    }, 250);

    // Send message to the server
    fetch('http://localhost:3000/message', {
      method: 'POST',
      headers: {
        accept: 'application/json',
        'Content-Type': 'application/json'
      },
      body: JSON.stringify({message})
    }).then(response => {
      return response.json();
    }).then(data => {
      document.getElementById("loading").remove();
      chatBoxBody.innerHTML += `<div class="response"><p>${data.message}</p></div>`;
      scrollToBottom();
    })
  }

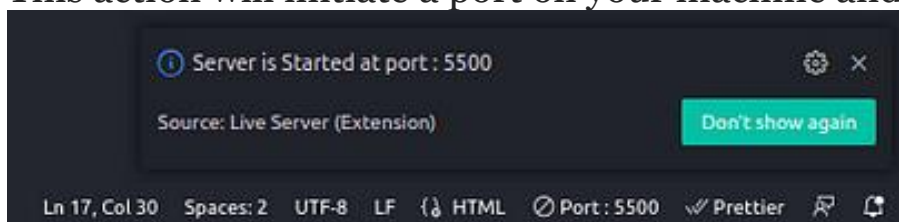
  // Function to scroll to the bottom of the chatbox
  function scrollToBottom() {
    chatBoxBody.scrollTop = chatBoxBody.scrollHeight;
  }
}

```

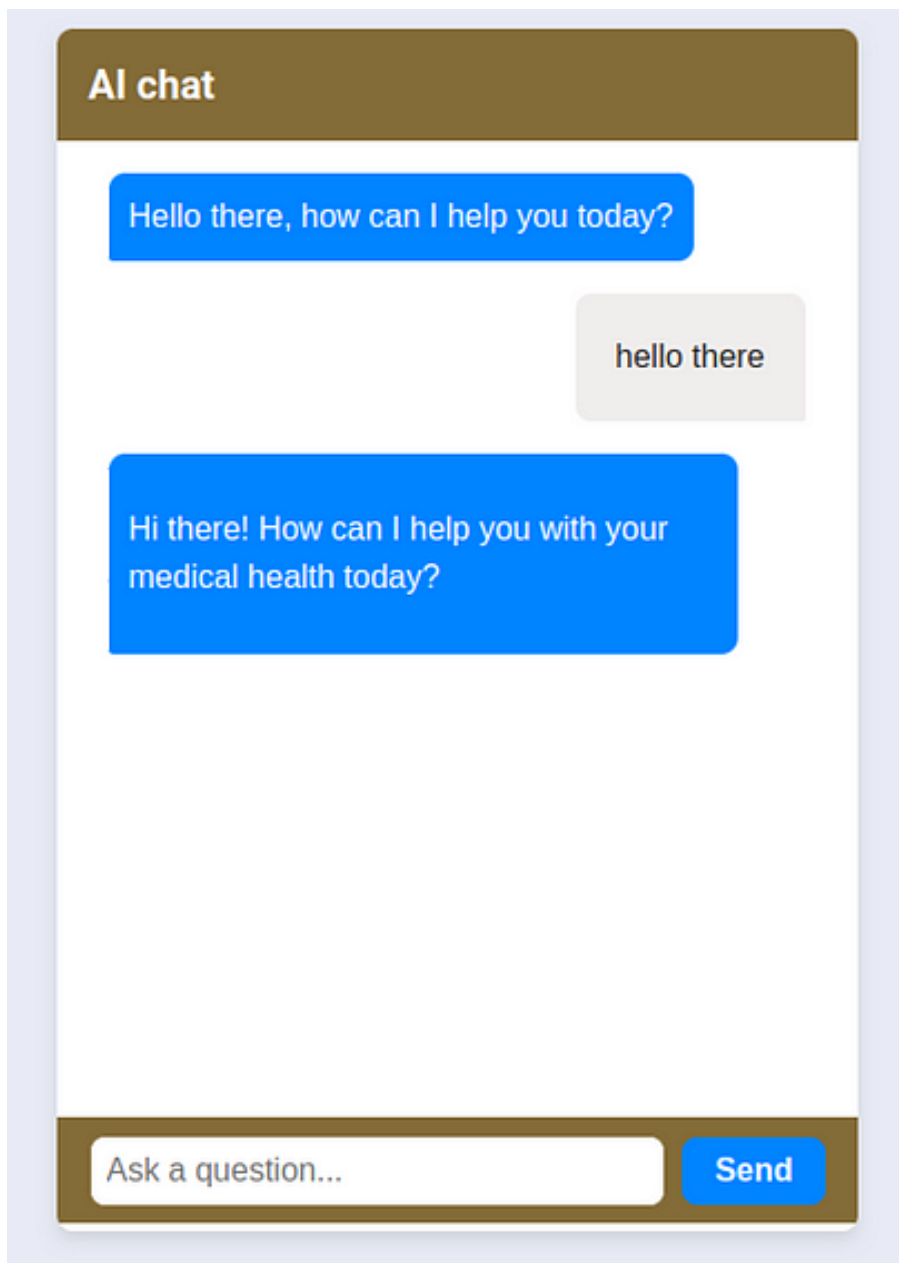
To run the chatbot follow these steps:

1. Navigate to the chatbot Interface folder.
2. Open it with a live server by clicking go live.

This action will initiate a port on your machine and serve the chatbot Interface.



Opening the above port on a browser launches our chatbot.



And that's it! We have our API set up to handle incoming messages, generate responses using the GPT-3 language model, and send the generated response back to the client.