ChatGPT integration to Web Applications

Theory :

About ChatGPT:

ChatGPT is an artificial intelligence chatbot developed by OpenAI and released in November 2022. The name "ChatGPT" combines "Chat", referring to its chatbot functionality, and "GPT", which stands for generative pre-trained transformer.
It basically works like a conversational bot, it answers that amount that the AI thinks should be able to satisfy the need of the customer.
The AI which we think is a boon can have certain functionalities that might be not satisfying the need of certain customer, the following being some of the reasons which I think:

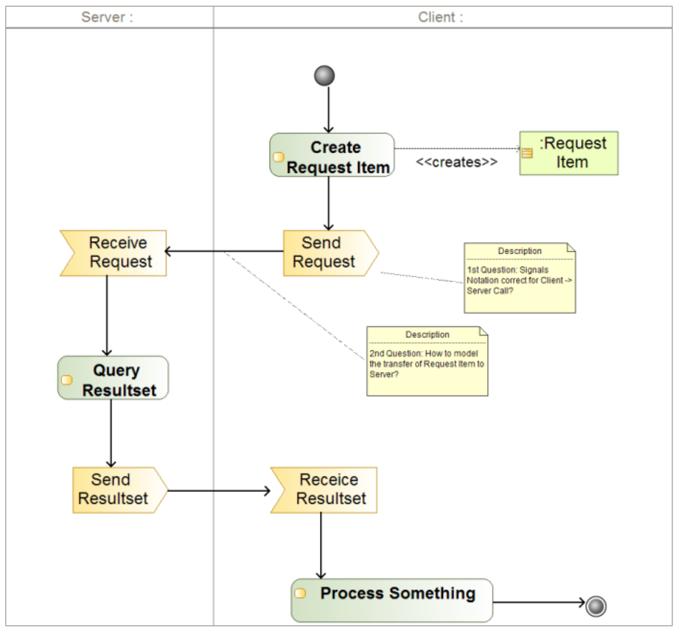
1. Since its a bot, and the data all over the world keeps increasing by every second, the bot is trained well with the data uptill 2021, and information after which may not be accurate or erroneous information hence it needs to be trained(it has the ability to self learn if it provides any erroneous responses or incorrect responses)
2. Since it's a conversational bot, it provides only the amount of information that it thinks is relevant for that matter, whereas websites like google,bing etc use information retrieval and provides the documentation of everything that is sort of linked with the object that is searched about.(Will be working on it to improve it).

API - Application Programming Interface

APIs are mechanisms that allow two software components to communicate with each other using a set of definitions and protocols. A few common examples that can be listed are:

- Weather API
- Maps API
- Bots

In this project we will be using the API of ChatGPT. The current version of ChatGPT that is in use is GPT -4, whereas OpenAI has put the older version, ie., GPT - 3.5 as an open API to be used by anyone.



Website:

We have created the basic framework of our website using the most basic, ie, using HTML, CSS and Javascript.
HTML - for the basic outline of the website
CSS - for the design
Javascript - for the functionality of the website.

Procedure:

Step1: Getting the API :

- Create an account in ChatGPT using google or any other mail id
- Go to the profile on the upper right corner and then go to the user.
- We will be able to see an option of creating API's, once we create the API, we need to copy the API key that is generated and use it in our code.
  Link for getting the API Key

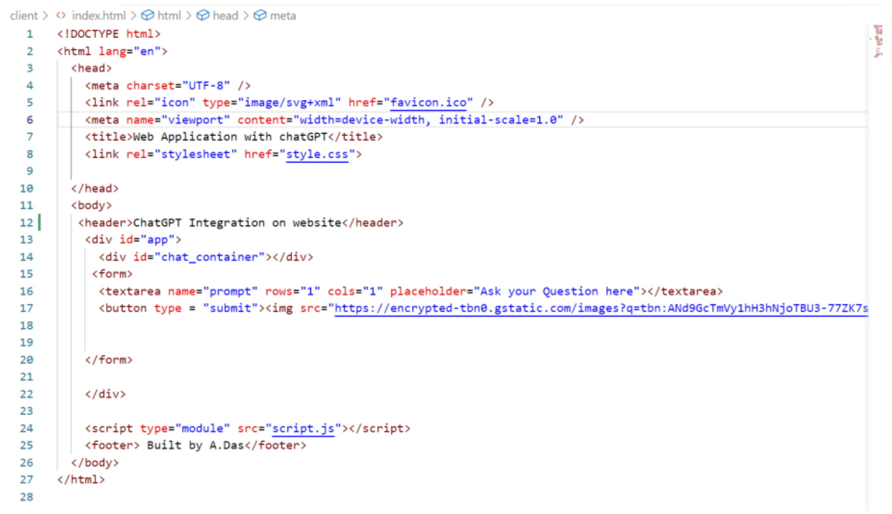
Step2: Starting with the website

- **HTML:**

  Ideally we have created 2 HTML pages, the first being the front page/home page or landing page. The second page being our main chat gpt integrated page.

Since we are using API here, the communication is between the client and server, we will be coding it out separately.

Step3 : Setting up the client side:

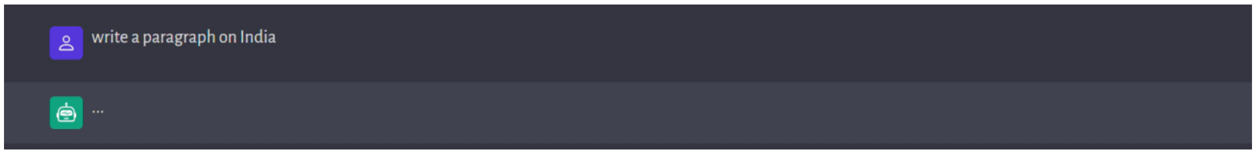
- npm init -y -> for creating the package.json file containing information about your project, its dependencies, scripts, and other metadata.
- Creating the HTML page, we have tried to make it similar to that of the original chatGPT hence we will be using it like a form, the css file and the javascript file has been linked from this page rather than making it all in 1 doc.
- 
client > index.html > html > head > meta
```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <link rel="icon" type="image/svg+xml" href="favicon.ico" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Web Application with chatGPT</title>
    <link rel="stylesheet" href="style.css">

  </head>
  <body>
   <header>ChatGPT Integration on website</header>
    <div id="app">
      <div id="chat_container"></div>
     <form>
      <textarea name="prompt" rows="1" cols="1" placeholder="Ask your Question here"></textarea>
      <button type = "submit"><img src="https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcTmVy1hH3hNjoTBU3-77ZK7s


    </form>

    </div>

    <script type="module" src="script.js"></script>
    <footer> Built by A.Das</footer>
  </body>
</html>

```
- The javascript file:
- Since it's more like a conversational bot, it is important that we are able to differentiate between who has written the message or the query and response, for that we have used 2 icons 1 for the bot and 1 for the user.
- The content that we type in the form is selected using **document.querySelector** and assigns them as “form” or “chatContainer”.
- **`Loadinterval`** It is used for storing the interval ID
- This is basically for the loading time- the time taken for the answer to be generated from the server’s end. - this is done with the help of the loader function.
- 
write a paragraph on India

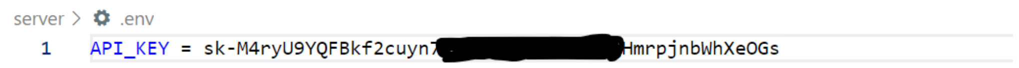
...

- typetext is used for the amount of time taken to type or in other words the typing effect such that it doesn’t simply present the whole thing at once but types every word of it.

- **ChatStrip** function is the one to differentiate it as 2 stripes for the user and the server and can be seen as 2 different shades as black and grey respectively.
- It has 3 parameters in it -
- 1. isAi - which identifies whether its a user or the bot,
- 2. Value - which is the message
- 3. UniqueID - its a uniqueID provided to every message
- The handleSubmit function is defined as an async function to handle form submission events. It takes an event object as a parameter. It prevents the default form submission behavior and extracts the form data using FormData.
- An HTTP POST request is made to 'http://localhost:5500' with the user's prompt as the request body in JSON format.
- If the entire execution is correct, the status is 200 otherwise displays an error

Step4: Setting up the server:

- Create a ".env" file where we paste the API key that we have acquired



```
API_KEY = sk-M4ryU9YQFBkf2cuyn7[redacted]HmrpjnbWhXeOGs
```

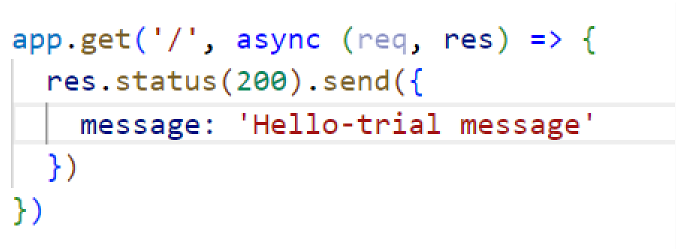
-
-
- Download the dependencies :
  npm install dotenv openai cors express nodemon

  dotenv package is used for loading the environment variables from the ".env" file
  Openai provides convenient access to the OpenAI API from Node.js applications
  CORS (Cross-Origin Resource Sharing) is a mechanism that allows web browsers to make cross-origin HTTP requests
  Express simplifies the process of handling HTTP requests, routing, middleware management, and response generation - req, res, GET, POST, DELETE etc are implemented using this package.
  nodemon - This package is used for restarting the server whenever any change is detected, there is no requirement of restarting the server after any small or big change that is made.

- npm init -y -> for creating the package.json file containing information about your project, its dependencies, scripts, and other metadata.
- An instance of **Configuration** is created for passing the API key retrieved from the environment variable(.env file)
- An instance of **OpenAIApi** is created to pass the configuration object
- An instance of **Express** is assigned to the “app” variable which is further used for initiating the cors and the express.json function

The next thing that comes about is the GET and POST request

GET: When a GET request is made to this route, it sends a JSON response with a message 'Hello-trial message' and a 200 status code.

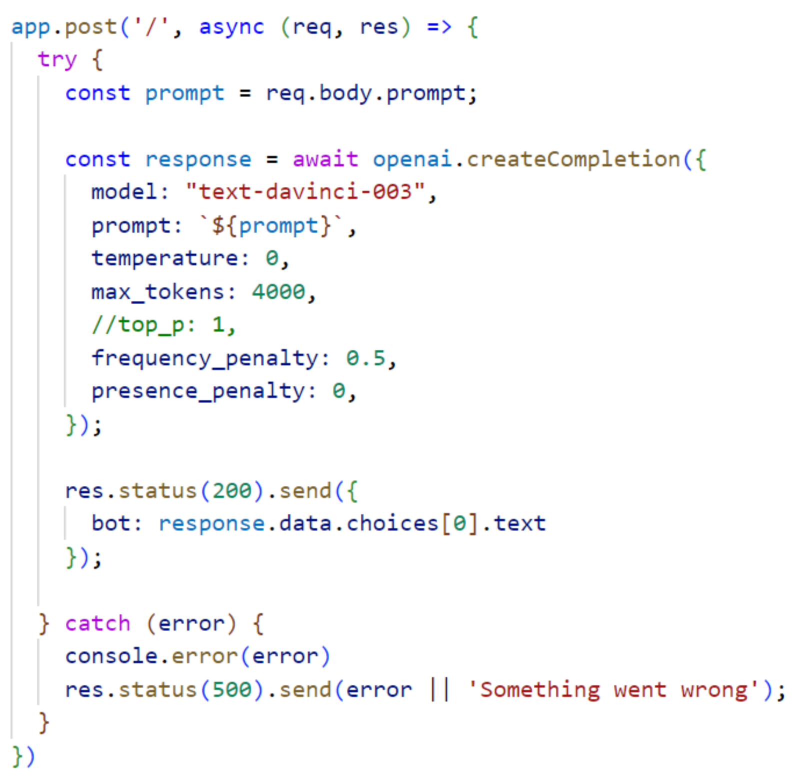


```
app.get('/', async (req, res) => {
  res.status(200).send({
    message: 'Hello-trial message'
  })
})
```

Here the use of async function is done as it returns a promise by default
Eg:

```
async function myFunction() {
  return "Hello";
}
myFunction().then(
  function(value) {myDisplayer(value);},
  function(error) {myDisplayer(error);}
);
```

POST :
For its implementation we take help of the OpenAI itself,
[Link provided here]
We go to Natural Language to openAI API, there we are able to see a few demo codes that are available that are provided by OpenAI for our ease.
Here we select **text-davinci-003** and view the code.
We copy the response part of the code from the code provided to us.



```
app.post('/', async (req, res) => {
  try {
    const prompt = req.body.prompt;

    const response = await openai.createCompletion({
      model: "text-davinci-003",
      prompt: `${prompt}`,
      temperature: 0,
      max_tokens: 4000,
      //top_p: 1,
      frequency_penalty: 0.5,
      presence_penalty: 0,
    });

    res.status(200).send({
      bot: response.data.choices[0].text
    });

  } catch (error) {
    console.error(error)
    res.status(500).send(error || 'Something went wrong');
  }
})
```

Temperature signifies the risk parameter, hence we equate it to 0.
Max tokens signify the number of tokens it can present out in the response, for keeping an upper range we have taken it to be as 4000.
Frequency penalty basically tells us that in case if a similar question is asked by the user, it is less likely to give out the same response.
Presence Penalty basically tells to use different words and phrases that are not recurring
Prompt is basically an input text or message that is sent to the OpenAI API to generate the response and it is extracted from the "request" body.
The function "openai.createCompletion()" is called for making the API call to ChatGPT -3.5 and it receives or retrieves the relevant response from the model, this is received in the form of an array which is then extracted using the bot's property.

When the answer is received, the server sends the response (JSON response) along with a status 200 code, which tells that everything is fine.
In case if anything doesn't work as per the requirement or we are unable to get the response from the API, an error message is generated with a status 500 code.

The server starts running at the local port 5500 and a log message is printed (The port can be changed manually, we can replace it with an actual url once we have deployed it).

Future work that remains pending:

1. Get “more” information on the desired search on ChatGPT in json format
2. Currently we are simply able to get the response, to view the past responses after refreshing,i.e, linking it with an email id or phone number will be required,ie, working on the backend.
3. Make the UI more responsive, create a diversion to view the response in json format and normal text format.