Creating a ChatGPT-based Chatbot with context memory in Vanilla JS

Aug. 27th, 2023
Alessandro Demontis
Vivacity Design Web Agency

Welcome to this tutorial on creating an OpenAI ChatGPT-based chatbot using pure DHTML code. In this tutorial, I will guide you through the necessary steps to create an interactive chatbot using the ChatGPT3.5 Turbo Model. The tutorial is divided into different progressive steps, using the creation of my personal Chatbot as a study case: VivacityGPT Online.

Introduction to VivacityGPT Online

VivacityGPT Online is a clone of ChatGPT developed by Vivacity Design Web Agency. This chatbot specializes in generating code and has 2 distinct personalities and contextual memory. The chatbot allows users to interact with it by entering messages and receiving contextually coherent responses.

Prerequisites

To follow this tutorial, you should have a basic understanding of HTML and JavaScript. The CSS section will be left out of the scope, nevertheless the HTML code contains all the appropriate styling classes and IDs so that the user can develop its own style. Make sure you have a text editor or development environment ready to get started.

Step 1: Prepare the Files

Firstly, ensure you have the HTML file, `script.js`, and `vdgptstyle.css` files present in the same directory as your project.

Step 2: HTML Structure

Open the HTML file and create the basic structure. We need a section for user input and a section for the chat history. Additionally, there is a script that handles sending messages when the user presses the Enter key. In the sudy case, a simple Navbar is provided.

```
<div class="fixedhead">
  <nav>
    <img src="../img/coollogo_com-108841550.png" width=300 alt="Vivacity
Logo">
    <div style="display: flex; flex-direction: column; align-items: center;">
      <h1 class="paddedheader">VivacityGPT Online</h1>
      <span class="typing-span">Now with context memory!</span>
    </div>
  </nav>
  <section class="userinput">
    <input type="text" id="user-input" placeholder="Entyer a request..."
onkeydown="verifyEnter(event)" />
    <button onclick="sendMessage()">Send</button>
  </section>
</div>
<section class="chathistory">
  <div id="chat-container"></div>
</section>
<footer>Courtesy of: Vivacity Design, June, 2023</footer>
<script>
  function verifyEnter(event) {
    // Check if the Enter key is pressed
    if (event.keyCode === 13) {
      event.preventDefault();
      // call the sendMessage function
      sendMessage();
    }
  }
</script>
```

Step 3: Adding the JavaScript Code to customize the Chatbot

In the `script.js` file, there is already a basic implementation to handle sending messages when the user presses the Enter key. This code calls the `sendMessage()` function.

We'll be implementing the response logic within the `sendMessage()` function. This function should take the text entered by the user, process it, and generate an appropriate response based on the context.

```
async function sendMessage() {
    const inputElement = document.getElementById('user-input');
    const userInput = inputElement.value.trim();

    if (userInput !== '') {
        showMessage("Guest", userInput);
        chatMemory = await getChatGPTResponse(userInput, chatMemory);
        inputElement.value = '';
    }
}
```

You will notice that we have two main components to develop.
- the showMessage function
- the chat memory context

Implementing the Chatbot Memory Context

We suggest to add this step at the top of the script. The concept is to save previouslyl received requests and answers in an array which will constitute the input for subsequent answers.

```
function createMemory(messages) {
  const memory = [];
  for (const msg of messages) {
    memory.push({ role: msg.role, content: msg.content });
  }
  return memory;
}
```

We will now define an initial state for the memory, where we will include the system prompt for the Chatbot personality:

```
let chatMemory = createMemory([
  { role: 'system', content: "You are a full stack developer working for Vivacity Design Web Agency. You are specialized in producing code for websites and you will always provide clean and effective code. When asked for some code, you will also provide the description for  different approaches to achieve the same goal." }
]);
```

Implementing the showMessage() to create the Chatbot history

I this section we'll use the basic references given by OpenAI for the management of the messages in and out. We'll intercept the messages and roles and place them in different divs to create the chat history structure:

```
function showMessage(sender, message) {
  const chatContainer = document.getElementById('chat-container');
  const chatSection = document.querySelector('.chathistory');
  const typingIndicator = document.getElementById('typing-indicator');

  // Removes "Typing..." when the chatbot starts answering
  if (typingIndicator && sender === 'VivacityGPT') {
    chatContainer.removeChild(typingIndicator);
  }

  // Create a new message element
  const messageElement = document.createElement('div');
  messageElement.innerText = `${sender}: ${message}`;

  // attributes the correct styling class according to the message source
  if (sender === 'Guest') {
    messageElement.classList.add('user-message');
  } else if (sender === 'VivacityGPT') {
    messageElement.classList.add('chatgpt-message');

    // Adds a function to copy the answer
    const copyLink = document.createElement('button');
    copyLink.innerText = 'Copia';
    copyLink.style.float = 'right';
    copyLink.addEventListener('click', function (event) {
        event.preventDefault();
        const text = message;
        const input = document.createElement('input');
        input.value = text;
        document.body.appendChild(input);
        input.select();
        document.execCommand('copy');
        document.body.removeChild(input);
    });

    messageElement.appendChild(copyLink);
  }
  //appends the message and makes sure to scroll to bottom
  chatContainer.appendChild(messageElement);
  chatSection.scrollTop = chatSection.scrollHeight;
}
```

Implementing the Requests to OpenAI and gets responses back, with filtering

This is the longest passage where we will implement the requests to the API, providing our KEY (is left blank in the code to insert your own); the concept is to send the request and filter the returned content the way we want. In the study case, a simple manipulation si provided to erase the language identification back-ticks. This feature can be expanded to intercept certain terms and replace them with others.

```
async function getChatGPTResponse(userInput, chatMemory = []) {
    const chatContainer = document.getElementById('chat-container');

    const typingIndicator = document.createElement('p');
    typingIndicator.id = 'typing-indicator';
    typingIndicator.textContent = 'Typing...';
    chatContainer.appendChild(typingIndicator);

    try {
        const response = await
fetch('https://api.openai.com/v1/chat/completions', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json',
            'Authorization': 'Bearer placeyourkeyhere'
          },
          body: JSON.stringify({
            "model": "gpt-3.5-turbo",
            "messages": [
              ...chatMemory,
              {"role": "user", "content": userInput}
            ]
          })
        });

        if (!response.ok) {
          throw new Error('An error occurred in the request to the \'API');
        }

        const data = await response.json();

        if (!data.choices || !data.choices.length || !data.choices[0].message
    || !data.choices[0].message.content) {
          throw new Error('Invalid API response');
        }

        const chatGPTResponse = data.choices[0].message.content.trim();
        var cleanResponse =
    chatGPTResponse.replace(/(```html|```css|```javascript|```php|```python)(.*?)/g
    s, '$2');
        cleanResponse = cleanResponse.replace(/```/g, "");
        showMessage("VivacityGPT", cleanResponse);

        // Place the current response in the context memory array
        chatMemory.push({ role: 'user', content: userInput });
        chatMemory.push({ role: 'assistant', content: cleanResponse });

        // Return the updated context memory
        return chatMemory;
      } catch (error) {
        console.error(error);
        // Here we can put some code to properly manage the errors.
      }
    }
```

By following these steps and understanding the provided code, you can create a JavaScript-based chatbot with context memory that enhances user interactions and provides meaningful responses. Remember to adapt and customize the chatbot according to your application's requirements and user experience goals.

The full JS working code of my version (italian) is reported below for the sake of completeness:

```
// Definizione della funzione per creare la memoria del contesto
function createMemory(messages) {
  const memory = [];
  for (const msg of messages) {
    memory.push({ role: msg.role, content: msg.content });
  }
  return memory;
}

// Funzione per inviare il messaggio
async function sendMessage() {
  const inputElement = document.getElementById('user-input');
  const userInput = inputElement.value.trim();

  if (userInput !== '') {
    showMessage("Guest", userInput);
    chatMemory = await getChatGPTResponse(userInput, chatMemory);
    inputElement.value = '';
  }
}

// Funzione per mostrare il messaggio nella chat
function showMessage(sender, message) {
  const chatContainer = document.getElementById('chat-container');
  const chatSection = document.querySelector('.chathistory');
  const typingIndicator = document.getElementById('typing-indicator');

  // Rimuove il messaggio "Digitazione in corso..." quando ChatGPT scrive la risposta
  if (typingIndicator && sender === 'VivacityGPT') {
    chatContainer.removeChild(typingIndicator);
  }

  // Crea il nuovo elemento del messaggio
  const messageElement = document.createElement('div');
  messageElement.innerText = `${sender}: ${message}`;

  // Aggiunge la classe corretta a seconda del mittente
  if (sender === 'Guest') {
    messageElement.classList.add('user-message');
  } else if (sender === 'VivacityGPT') {
    messageElement.classList.add('chatgpt-message');

    // Aggiunge link per copiare il contenuto del messaggio
    const copyLink = document.createElement('button');
```

```
        copyLink.innerText = 'Copia';
        copyLink.style.float = 'right';
        copyLink.addEventListener('click', function (event) {
            event.preventDefault();
            const text = message;
            const input = document.createElement('input');
            input.value = text;
            document.body.appendChild(input);
            input.select();
            document.execCommand('copy');
            document.body.removeChild(input);
        });

        messageElement.appendChild(copyLink);
        // Fine aggiunta link copia
    }

    chatContainer.appendChild(messageElement);
    chatSection.scrollTop = chatSection.scrollHeight;
}

// Funzione per ottenere la risposta da ChatGPT
async function getChatGPTResponse(userInput, chatMemory = []) {
    const chatContainer = document.getElementById('chat-container');

    const typingIndicator = document.createElement('p');
    typingIndicator.id = 'typing-indicator';
    typingIndicator.textContent = 'Digitazione in corso...';
    chatContainer.appendChild(typingIndicator);

    try {
        const response = await fetch('https://api.openai.com/v1/chat/completions', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json',
                'Authorization': 'Bearer yourapikeygoeshere'
            },
            body: JSON.stringify({
                "model": "gpt-3.5-turbo",
                "messages": [
                    ...chatMemory,
                    {"role": "user", "content": userInput}
                ]
            })
        });

        if (!response.ok) {
            throw new Error('Errore nella richiesta all\'API');
        }

        const data = await response.json();

        if (!data.choices || !data.choices.length || !data.choices[0].message
|| !data.choices[0].message.content) {
            throw new Error('Risposta API non valida');
        }

        const chatGPTResponse = data.choices[0].message.content.trim();
        var cleanResponse =
chatGPTResponse.replace(/(```html|```css|```javascript|```php|```python)(.*?)/gs, '$2');
        cleanResponse = cleanResponse.replace(/```/g, "");
        showMessage("VivacityGPT", cleanResponse);

        // Aggiungi la risposta corrente alla memoria del contesto
        chatMemory.push({ role: 'user', content: userInput });
        chatMemory.push({ role: 'assistant', content: cleanResponse });

        // Ritorna la memoria del contesto aggiornata
        return chatMemory;
    } catch (error) {
        console.error(error);
        // Gestisci l'errore appropriatamente, ad esempio mostrando un messaggio di
errore all'utente.
    }
}



// Esempio di utilizzo iniziale con memoria di contesto vuota
let chatMemory = createMemory([
    { role: 'system', content: "You are a full stack developer working for Vivacity Design
Web Agency. You are specialized in producing code for websites and you will always
provide clean and effective code. When asked for some code, you will also provide
the description for  different approaches to achieve the same goal." }
]);
```

The system can be tested here: http://www.vivacitydesign.net/vdgpt/index.html