Creating a Custom Chat Bot with OpenAI APIs

Blazor and .NET MAUI Integration

Table of Contents

[Creating a Chat Bot using the Blazor server app 3](#_Toc175225130)

[Project Setup in Visual Studio 3](#_Toc175225131)

[Adding NuGet Packages 6](#_Toc175225132)

[Updating the Program.cs File 7](#_Toc175225133)

[Modifying Imports.razor File 7](#_Toc175225134)

[Designing the UI 7](#_Toc175225135)

[Creating a Chat Bot engine 22](#_Toc175225136)

[Creating a Custom Chat Bot 30](#_Toc175225137)

[Creating an Assistant 31](#_Toc175225138)

[Create a Vector Store and get the Vector Store ID 33](#_Toc175225139)

[Upload the file and get the file ID 33](#_Toc175225140)

[Adding the file to the vector store 35](#_Toc175225141)

[Update the assistant with the vector store ID 35](#_Toc175225142)

[Creating a thread to ask a question 36](#_Toc175225143)

[Create a run to execute the thread and retrieve the answers 37](#_Toc175225144)

[Retrieve the message from the thread 39](#_Toc175225145)

[Asking questions to the AI 41](#_Toc175225146)

[Creating a Hybrid app using MAUI 43](#_Toc175225147)

[Uncovering the textbox UI in Android 70](#_Toc175225148)

Creating a Chat Bot using the Blazor server app

Let’s see the step-by-step guide for creating a Chat Bot using the **Blazor server app**.

### Project Setup in Visual Studio

Step 1: Open Visual Studio and click *Create a new project*.

A screenshot of a computer

Description automatically generated

Step 2: Search for “*Blazor Server App*” and select it, then click next.

A screenshot of a computer

Description automatically generated

Step 3: Name the project “*ChatBot Server App*”, click ‘*Next’* and click ‘*Create’*.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A project will be created.

A screenshot of a computer

Description automatically generated

### Adding NuGet Packages

Step 4: Right-click on the “*Dependencies*”, click “*Manage NuGet Packages*” and add the following NuGet packages to the project.

*Syncfusion.Blazor.Buttons*

*Syncfusion.Blazor.Inputs*

*Syncfusion.Blazor.Lists*

*Syncfusion.Blazor.Notifications*

*Syncfusion.Blazor.Spinner*

*Syncfusion.Blazor.SplitButtons*

*Syncfusion.Blazor.Themes*

A screenshot of a computer

Description automatically generated

### Updating the Program.cs File

Step 5: Open ***Program.cs*** file and add the below highlighted code. Copy and paste the required namespace at the top.

Note: Replace your valid Syncfusion license key in the **syncfusionkey** variable.

|  |
| --- |
| using Syncfusion.Blazor;  .  .  .  .  .  var builder = WebApplication.CreateBuilder(args);  builder.Services.AddSyncfusionBlazor();  // Add the required keys  string syncfusionkey = "Your synckey";  Syncfusion.Licensing.SyncfusionLicenseProvider.RegisterLicense(syncfusionkey); |

Modifying Imports.razor File

Step 6: Open ***\_Imports.razor*** file and add the following namespace.

|  |
| --- |
| @using Syncfusion.Blazor  @using Syncfusion.Blazor.Lists  @using Syncfusion.Blazor.Inputs  @using System.Collections.ObjectModel  @using Syncfusion.Blazor.Spinner  @using Syncfusion.Blazor.SplitButtons  @using Syncfusion.Blazor.Notifications  @using Syncfusion.Blazor.Buttons |

### Designing the UI

Step 7: Open the ***Pages/Index.razor*** page, remove the existing code and replace it with the following code.

|  |
| --- |
| @page "/"  @inject HttpClient HttpClient  <PageTitle>AI Chat Bot</PageTitle>  @using ChatBot\_Server\_App.Data  @if (VisibleFAB)  {  <SfFab @ref="FabObject" Target="#container" IconCss="demo-icons chat-icon" OnClick="EventClick" CssClass="custom-fab"></SfFab>  }  <div id="container" >    <div id="sample" style="visibility:@ContainerVisibility">  <div style="color:white; Background:deepskyblue; width:100%; height:50px; padding:10px; font-size:23px">  AI Chat Bot  </div>  <div>  <SfSpinner @bind-Visible="@VisibleProperty">  </SfSpinner>  </div>  <form method="post" @onsubmit="OnSend" >  <SfListView ID="list" class="flex flex\_\_direction"  DataSource="@DataSource" Height="89vh" >  <ListViewFieldSettings TValue="ListDataModel" Id="Id" Text="Text"></ListViewFieldSettings>  <ListViewTemplates TValue="ListDataModel">  <Template>  @{  ListDataModel currentData = context as ListDataModel;  <div class="flex item\_\_container">  <div class="flex flex\_\_1 vertical\_\_center flex\_\_center @(currentData.Chat == "sender" ? "flex\_\_order\_\_2" : "")">  @if (currentData.Avatar != "")  {  <span class="e-avatar e-avatar-circle">@currentData.Avatar</span>  }  else  {  <span class="@currentData.Pic e-avatar e-avatar-circle"></span>  }  </div>    <div class="flex content\_\_container flex\_\_8 vertical padding @(currentData.Chat == "sender" ? "right\_\_align" : "left\_\_align")">  <div class="bold">@currentData.Text</div>  <div class="small\_\_font">@((MarkupString)@currentData.Contact)</div>  </div>  </div>  }  </Template>  </ListViewTemplates>  </SfListView>  <div class="flex">  <div class="flex\_\_8 padding">  <SfTextBox Placeholder="Type your message"  @ref="@ChatTextBox"  @oninput="@(e=>OnTyping(e))"></SfTextBox>  </div>  <div class="flex">  <button class="e-btn" ></button>  </div>  <div>  <SfToast @ref="ToastObj" ID="toast\_type">  <ToastPosition X="@ToastPosition"></ToastPosition>  </SfToast>  </div>  </div>  </form>  </div>  </div>  @code  {  SfToast ToastObj;  SfFab FabObject;  SfTextBox ChatTextBox;  string textBoxValue = "";  private string ToastPosition = "Right";  ObservableCollection<ListDataModel> DataSource = new ObservableCollection<ListDataModel>();  public string result = "";  public async Task EventClick()  {  this.VisibleFAB = false;  StateHasChanged();  this.ContainerVisibility = "visible";  }    async Task OnSend()  {  if (ChatTextBox.Value != null)  {  textBoxValue = ChatTextBox.Value;  ChatTextBox.Value = "";  DataSource.Add(new ListDataModel  {  Text = "Mike",  Contact = textBoxValue,  Id = new Random().Next(300, 900).ToString(),  Avatar = "",  Pic = "pic02",  Chat = "receiver"  });  this.VisibleProperty = true;  // result = await GetResult(textBoxValue);  this.VisibleProperty = false;  DataSource.Add(new ListDataModel  {  Text = "AI Bot",  Contact = result,  Id = "1",  Avatar = "",  Pic = "pic01",  Chat = "sender"  });  }  else  {  DataSource.Add(new ListDataModel  {  Text = "Mike",  Contact = textBoxValue,  Id = new Random().Next(300, 900).ToString(),  Avatar = "",  Pic = "pic02",  Chat = "receiver"  });  DataSource.Add(new ListDataModel  {  Text = "AI Bot",  Contact = "Please enter a valid query",  Id = "1",  Avatar = "",  Pic = "pic01",  Chat = "sender"  });  }  }  async Task OnTyping(ChangeEventArgs args)  {  if (args.Value != "")  {  textBoxValue = args.Value.ToString();  }  }  public class ListDataModel  {  public string Id  {  get;  set;  }  public string Chat  {  get;  set;  }  public string Pic  {  get;  set;  }  public string Avatar  {  get;  set;  }  public string Text  {  get;  set;  }  public string Contact  {  get;  set;  }  }  private bool VisibleProperty { get; set; } = false;  private bool VisibleFAB { get; set; } = true;  private string ContainerVisibility { get; set; } = "hidden";  }  <style>  #list {  box-shadow: 0 1px 4px #ddd;  border-bottom: 1px solid #ddd;  }  #list {  margin: 0 auto;  border: 1px solid #ccc;  }  #list .e-list-item {  height: auto;  cursor: pointer;  line-height: 22px;  padding: 8px;  }  #list.e-listview{  overflow-y:scroll;  position:sticky;  scrollbar-width:none;  }  #list.e-listview .e-list-container {  overflow:unset;  position:unset;  }  #list.e-listview .e-list-header {  background-color: #0278d7;  color: white;  }  #list .e-list-item.e-active {  background-color: transparent;  }  #list .e-list-item.e-hover {  background-color: transparent;  }  .padding {  padding: 4px;  }  .right\_\_align {  text-align: right;  margin-right: 8px;  padding-right: 8px;  }  .left\_\_align {  margin-left: 8px;  padding-left: 8px;  }  .content\_\_container {  background-color: aliceblue;  }  .e-btn{  background-image: url("../send.png");  background-repeat: no-repeat;  background-color :white;  height: 30px;  width:50px;  background-size: 100%;  margin: 5px;    }  .e-btn:hover{    background-image: url("../send.png");  background-repeat: no-repeat;  background-size: 100%;  }  .e-btn:focus{  background-image: url("../send.png");  background-repeat: no-repeat;  background-color: white;  height: 30px;  width: 50px;  background-size: 100%;  margin: 5px;  }  .flex {  display: flex;  }  .flex\_\_direction{  flex-direction: column-reverse;  flex-grow: 1;  }  .flex\_\_center {  justify-content: center;  }  .vertical\_\_center {  align-items: center;  }  .vertical {  flex-direction: column;  }  .flex\_\_order\_\_1 {  order: 1;  }  .flex\_\_order\_\_2 {  order: 2;  }  .flex\_\_1 {  flex: 1;  }  .flex\_\_2 {  flex: 2;  }  .flex\_\_3 {  flex: 3;  }  .flex\_\_5 {  flex: 5;  }  .flex\_\_8 {  flex: 8;  }  .bold {  font-weight: bold;  color:green;  }  .margin {  margin: 10px;  }  .small\_\_font {  font-size: 13px;  margin: 2px 0;  }  .pic01 {  background-image: url("../cody.png");  }  .pic02 {  background-image: url("../mike.jpg");  }  .pic03 {  background-image: url("https://ej2.syncfusion.com/demos/src/grid/images/5.png");  }  .pic04 {  background-image: url("https://ej2.syncfusion.com/demos/src/grid/images/2.png");  }  @@font-face {  font-family: 'Toast\_icons';  src: url(data:application/x-font-ttf;charset=utf-8;base64,) format('truetype');  font-weight: normal;  font-style: normal;  }  #toast\_types button {  margin: 5px;  min-width: 160px;  max-width: 160px;  }  .toast-icons {  font-family: 'Toast\_icons' !important;  speak: none;  font-size: 55px;  font-style: normal;  font-weight: normal;  font-variant: normal;  text-transform: none;  line-height: 1;  -webkit-font-smoothing: antialiased;  -moz-osx-font-smoothing: grayscale;  }  #toast\_type .e-toast-icon.e-icons {  height: auto;  font-size: 30px;  }  .bootstrap5 #toast\_type .e-toast-icon.e-icons,  .bootstrap5-dark #toast\_type .e-toast-icon.e-icons {  height: 25px;  }  .toast-icons.e-success::before {  content: "\e701";  }  .toast-icons.e-error::before {  content: "\e700";  }  .toast-icons.e-info::before {  content: "\e704";  }  .toast-icons.e-warning::before {  content: "\e703";  }  #toast\_types {  text-align: center;  }  @@font-face {  font-family: 'demo-icons';  src:  url(data:application/x-font-ttf;charset=utf-8;base64,) format('truetype');  font-weight: normal;  font-style: normal;  }  [class^="demo-icons"], [class\*=" demo-icons"] {  font-family: 'demo-icons' !important;  speak: none;  font-size: 55px;  font-style: normal;  font-weight: normal;  font-variant: normal;  text-transform: none;  line-height: 1;  -webkit-font-smoothing: antialiased;  -moz-osx-font-smoothing: grayscale;  }  .chat-icon:before { content: "\e700"; }  custom-fab.e-fab.e-btn .e-btn-icon {  font-size: 30px;  }  </style> |

Step 8: Open ***Pages/\_Host.cshtml*** file and add the <head> tag with Syncfusion scripts.

|  |
| --- |
| <head>  <link href="\_content/Syncfusion.Blazor.Themes/bootstrap5.css" rel="stylesheet" />  <script src="\_content/Syncfusion.Blazor.Core/scripts/syncfusion-blazor.min.js" type="text/javascript"></script>  </head> |

Step 9: Now run the application to see the following output.

A screenshot of a computer

Description automatically generated

You’ll see a chat icon at the bottom right corner. Click the chat Icon to open the Chat interface.

A screenshot of a computer

Description automatically generated

Step 10: To remove the default server app layout, navigate to ***Shared/MainLayout.razor*** file, and remove the exact below code.

|  |
| --- |
| <div class="sidebar">  <**NavMenu** />  </div>  <div class="top-row px-4">  <a href="https://docs.microsoft.com/aspnet/" target="\_blank">About</a>  </div> |

After removing the default layout, you will see the updated user interface below. A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Step 11: Click on the **wwwroot** folder and add the png files using **Add-> Add Existing Items.**

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Launch the application again and click on the text box to send "hi."

A computer screen with a white screen

Description automatically generated

The AI bot will not respond as we have only completed the UI/UX so far.

Creating a Chat Bot engine  
Let’s dive into steps for creating a Chat Bot engine.

Step 1: Inside the ***Data*** Folder, create a new file called ***ChatGPTEngine.cs*** file.

Step 2: Then, add these required namespaces to the same file.

|  |
| --- |
| using Microsoft.AspNetCore.Identity;  using System.Net.Http;  using System.Net.Http.Headers;  using System.Text;  using System.Text.Json;  using System.Text.Json.Serialization;  using System.Text.RegularExpressions;  using System.Threading.Tasks; |

Step 3: Add the following code inside the class: Include your openai key, the model you are going to use, the api endpoint you are going to call, and the HTTP client to raise this request.

|  |
| --- |
| private HttpClient httpClient { get; set; }  private string OPENAI\_KEY = ""; // Add a valid OpenAI key here.  private string OPENAI\_MODEL = "gpt-3.5-turbo-instruct";  private string API\_Chat\_ENDPOINT = "https://api.openai.com/v1/chat/completions";  private string API\_ENDPOINT = "https://api.openai.com/v1/completions"; |

Note:Assign your valid **OpenAI key** to the **OPENAI\_KEY** variable used in the above code.

Step 4: Initialize the HttpClient inside the constructor and place the below code inside the ChatGPTEngine class.

|  |
| --- |
| public ChatGPTEngine()  {  httpClient = new HttpClient();  } |

Step 5: Create a method to handle the user input. This method should be an asynchronous task, with the prompt from the user passed as a string.

|  |
| --- |
| internal async Task<string> ProcessTheUserInput(string prompt)  {  var val = new AuthenticationHeaderValue("Bearer", OPENAI\_KEY);  httpClient.DefaultRequestHeaders.Authorization = val;  var openAIPrompt = new  {  model = OPENAI\_MODEL,  prompt,  temperature = 0.5,  max\_tokens = 1500,  top\_p = 1,  frequency\_penalty = 0,  presence\_penalty = 0  };  var content = new StringContent(JsonSerializer.Serialize(openAIPrompt), Encoding.UTF8, "application/json");  var response = await httpClient.PostAsync(API\_ENDPOINT, content);  var jsonContent = await response.Content.ReadAsStringAsync();  var choices = JsonDocument.Parse(jsonContent).RootElement.GetProperty("choices").GetRawText();  var result = JsonDocument.Parse(Regex.Replace(choices, @"[\[\]]", string.Empty)).RootElement;  return result.GetProperty("text").GetString();  } |

Step 6: In the ***Pages/index.razor*** file, add **OnInitializedAsync()** method to set the visible property of the spinner with the below code:

|  |
| --- |
| private ChatGPTEngine gptEngine = new ChatGPTEngine();  public string reply = "";  protected override async Task OnInitializedAsync()  {  this.VisibleProperty = true;  } |

Also, in the same page, add the following highlighted code inside the **EventClick**() method:

|  |
| --- |
| public async Task EventClick()  {  this.VisibleFAB = false;  StateHasChanged();  this.ContainerVisibility = "visible";  reply = await gptEngine.ProcessTheUserInput("You are an AI chatbot and you are here to answer my question");  await Task.Delay(3500);  this.VisibleProperty = false;  DataSource.Add(new ListDataModel  {  Text = "AI Bot",  Contact = reply,  Id = "1",  Avatar = "",  Pic = "pic01",  Chat = "sender"  });  } |

This will call the prompt “You are an AI chatbot and you are here to answer my question” using the **ProcessTheUserInput** method. We will get a reply from the ChatBot as shown below.

A screenshot of a computer

Description automatically generated

Step 7: To make this conversational, go to the **OnSend()** method in the ***Pages/index.razor*** file, and **uncomment** this below line.

|  |
| --- |
| result = await GetResult(textBoxValue); |

Step 8: Add the **GetResult** method as shown below in the same ***Pages/index.razor*** file.

|  |
| --- |
| private async Task<string> GetResult(string query)  {  #region Request simple user query  result = await gptEngine.ProcessTheUserInput(query);  #endregion  return result;  } |

Now, if we ask any questions in the textbox and click the send button, the OnSend() method will be called, and our question will be sent as a prompt. We will get a response, which is stored as a result and added to the datasource of the listview to show it as chat.

A screenshot of a computer

Description automatically generated

This is just a basic chatbot that can answer questions but cannot provide contextual answers or remember the previous questions.

A screenshot of a computer

Description automatically generated

To get contextual answers, we need to use different API endpoint. This endpoint provides the ChatGPT with the user role and prompt. For e.g., you have to provide the role as **user** when asking a question and store the answer received form GPT as system.

Step 9: Go to the ***Data/ChatGPTEngine.cs*** file and add a new class called **message** to store role and content.

|  |
| --- |
| public class Message  {  public string role { get; set; }  public string content { get; set; }  } |

Step 10: Inside the **ChatGPTEngine** class, add the following code.

|  |
| --- |
| public List<Message> myMessage = new List<Message>();  internal async Task<string> ProcessTheGivenInfoWithContext(string aiAnswer, string userQuestion)  {  var val = new AuthenticationHeaderValue("Bearer", OPENAI\_KEY);  httpClient.DefaultRequestHeaders.Authorization = val;  myMessage.Add(new Message { role = "system", content = aiAnswer });  myMessage.Add(new Message { role = "user", content = userQuestion });  var openAIPrompt = new  {  model = "gpt-3.5-turbo",  messages = myMessage,  temperature = 0.5,  max\_tokens = 1500,  top\_p = 1,  frequency\_penalty = 0,  presence\_penalty = 0  };  var content = new StringContent(JsonSerializer.Serialize(openAIPrompt), Encoding.UTF8, "application/json");  var response = await httpClient.PostAsync(API\_Chat\_ENDPOINT, content);  var jsonContent = await response.Content.ReadAsStringAsync();  var choices = JsonDocument.Parse(jsonContent).RootElement.GetProperty("choices")[0].GetProperty("message").GetRawText();  var result = JsonDocument.Parse(Regex.Replace(choices, @"[\[\]]", string.Empty)).RootElement;  var result1 = result.ToString().Replace("\n", "<br>");  return result.GetProperty("content").GetString();  } |

Step 11: Go to ***Pages/index.razor*** file and locate **EventClick()** method. Delete the below highlighted single line of code and then add the next 2 lines of code in that place.

|  |
| --- |
| reply = await gptEngine.ProcessTheUserInput("You are an AI chatbot and you are here to answer my question");  gptEngine.myMessage.Add(new Message { role = "user", content = "Remember my name as Michael" });  reply = await gptEngine.ProcessTheGivenInfoWithContext(result, "You are an AI chatbot and you are here to answer my question"); |

Step 12: Then, go to **GetResult** method and replace its code with the following code.

|  |
| --- |
| result = await gptEngine.ProcessTheGivenInfoWithContext(result, query); |

Once these changes are made, the conversational AI will reply with context.

A screenshot of a computer

Description automatically generated

To store a large conversation, this method will take a lot of memory bytes, but if you want your bot to answer from a small file, like a restaurant menu that can be done.

Step 1: In **wwwroot** folder, create a new sub-folder and name it as **files**. Then, add the **restaurant\_menu.txt** file to it. A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Step 2: Declare a variable called **fileContent** in the **index.razor** file before **EventClick()** method.

|  |
| --- |
| private string? fileContent = ""; |

Step 3: Inside the **EventClick**() method, assign the menu content to the fileContent variable as a user message, and set the role as "user."

|  |
| --- |
| this.ContainerVisibility = "visible";  fileContent = await HttpClient.GetStringAsync("https://localhost:**7256**/files/restaurant\_menu.txt");  gptEngine.myMessage.Add(new Message { role = "user", content = fileContent });  gptEngine.myMessage.Add(new Message { role = "user", content = "Remember my name as Michael" });  reply = await gptEngine.ProcessTheGivenInfoWithContext(result, "You are an AI chatbot and you are here to answer my question"); |

In the above code, I am using my localhost port which is available when I run this application. You should use your application port number.

A screenshot of a computer

Description automatically generated

A close-up of a computer screen

Description automatically generated

But this is not the right way to do a file search. To perform the file search, we must take a different approach.

## Creating a Custom Chat Bot

The correct way to do this is to create a custom chat bot, how can we do that? We need to create an assistant that understands our specific requirements.

We must follow the below steps.

* Create an Assistant and get the Assistant ID
* Create a Vector Store and get the Vector Store ID
* Upload the file and get the file ID
* Add the file to the vector store
* Update the assistant with the vector store id
* Create a thread to ask a question
* Create a run to execute the thread and retrieve the answers
* Retrieve the message from the thread

### Creating an Assistant

Step 1: Go to **Data** Folder and create a new class called **AssistantFileSearch.cs** and add these namespaces.

|  |
| --- |
| using System;  using System.IO;  using System.Net.Http;  using System.Net.Http.Headers;  using System.Numerics;  using System.Runtime.Intrinsics;  using System.Text;  using System.Text.Json;  using System.Text.Json.Serialization;  using System.Threading;  using System.Threading.Tasks;  using Microsoft.Extensions.Configuration;  using Syncfusion.Blazor.Notifications;  using static System.Net.WebRequestMethods; |

Step 2: Declare these variables.

|  |
| --- |
| private static readonly HttpClient httpClient = new HttpClient();  private string OPENAI\_KEY="";  private const string API\_URL = "https://api.openai.com/v1";  public string threadId = "";  public string assistantId = ""; |

Note:Assign your valid **OpenAI key** to the **OPENAI\_KEY** variable used in the above code.

Step 3: Inside the **AssistantFileSearch** class, define a **Search** method like below, to search the file using the above steps. I will use SfToast component to generate notifications for each request from the OpenAI.

|  |
| --- |
| public async Task Search(SfToast toast)  {  httpClient.DefaultRequestHeaders.Clear();  httpClient.DefaultRequestHeaders.Add("Authorization", $"Bearer {OPENAI\_KEY}");  httpClient.DefaultRequestHeaders.Add("OpenAI-Beta", "assistants=v2");  } |

Step 4: Add the below code inside the **Search** method.

|  |
| --- |
| // Create Assistant  var assistant = await CreateAssistant();  await toast.ShowAsync(new ToastModel { Title = "Assistant Created", Content = "Created a new Assistant with id"+assistant.Id, CssClass = "e-toast-success", Icon = "e-info toast-icons" });  assistantId = assistant.Id; |

Step 5: Create a new method called **CreateAssistant()**, as like below.

|  |
| --- |
| private static async Task<AssistantResponse> CreateAssistant()  {  var openAIPrompt = new  {  name = "Restaurant take away manager",  instructions = "You are an expert in Restaurant take away and answer anything related to the menu, also always keep the conversation about the restaurant menu even if user asks any non related questions divert them back to ask menu related questions.",  model = "gpt-4-turbo",  tools = new[] { new { type = "file\_search" } }  };  var content = new StringContent(JsonSerializer.Serialize(openAIPrompt), Encoding.UTF8, "application/json");  var response = await httpClient.PostAsync($"{API\_URL}/assistants", content);  response.EnsureSuccessStatusCode();  var jsonContent = await response.Content.ReadAsStringAsync();  return JsonSerializer.Deserialize<AssistantResponse>(jsonContent);  } |

Step 6: For this method to work, we need a new class called **AssistanceResonpse** with some properties to deserialize the response from OpenAI.

|  |
| --- |
| public class AssistantResponse  {  [JsonPropertyName("id")]  public string Id { get; set; }  } |

### Create a Vector Store and get the Vector Store ID

Step 1: We need to create a **vector store** after the **assistant ID** like below in the **Search** method.

|  |
| --- |
| assistantId = assistant.Id;  // Create Vector Store  var vectorStore = await CreateVectorStore();  await toast.ShowAsync(new ToastModel { Title = "Vector Store Created", Content = "Created a new Vector store with id" + vectorStore.Id, CssClass = "e-toast-success", Icon = "e-info toast-icons" }); |

Step 2: Create a new method called **CreateVectorStore()** like below**.**

|  |
| --- |
| private static async Task<VectorStoreResponse> CreateVectorStore()  {  var vectorStoreData = new { name = "Syncfusion Blazor" };  var content = new StringContent(JsonSerializer.Serialize(vectorStoreData), Encoding.UTF8, "application/json");  var response = await httpClient.PostAsync($"{API\_URL}/vector\_stores", content);  response.EnsureSuccessStatusCode();  var jsonContent = await response.Content.ReadAsStringAsync();  return JsonSerializer.Deserialize<VectorStoreResponse>(jsonContent);  } |

Step 3: For this method to work, we need a new class called **VectorStoreResponse** with some properties to deserialize the response from the OpenAI.

|  |
| --- |
| public class VectorStoreResponse  {  [JsonPropertyName("id")]  public string Id { get; set; }  } |

### Upload the file and get the file ID

Step 1: Create a new folder to your project and name it as **DataFiles** and add the menu text file to it.

A screenshot of a computer

Description automatically generated

Step 2: Upload the text file using the code below after creating the vector store in the search method.

|  |
| --- |
| // Upload the file  var fileId = await UploadFile("DataFiles/restaurant\_menu.txt", "assistants");  await toast.ShowAsync(new ToastModel { Title = "File Uploaded", Content = "File uploaded with id" + fileId, CssClass = "e-toast-success", Icon = "e-info toast-icons" }); |

Step 3: Create the method **UploadFile** like below.

|  |
| --- |
| private static async Task<string> UploadFile(string filePath, string purpose)  {  using var fileStream = new FileStream(filePath, FileMode.Open);  var formData = new MultipartFormDataContent();  formData.Add(new StreamContent(fileStream), "file", Path.GetFileName(filePath));  formData.Add(new StringContent(purpose), "purpose");  var response = await httpClient.PostAsync($"{API\_URL}/files", formData);  response.EnsureSuccessStatusCode();  var jsonContent = await response.Content.ReadAsStringAsync();  var fileResponse = JsonSerializer.Deserialize<FileResponse>(jsonContent);  return fileResponse.Id;  } |

Step 4: For this method to work, we need a new class called **FileResponse** with some properties to deserialize the response from the OpenAI.

|  |
| --- |
| public class FileResponse  {  [JsonPropertyName("id")]  public string Id { get; set; }  [JsonPropertyName("filename")]  public string Filename { get; set; }  } |

### Adding the file to the vector store

Step 1: Add the text file to the vector store using the file ID, like the code below, after uploading the text file in the search method.

|  |
| --- |
| // Add the file to the vector store  await AddFileToVectorStore(vectorStore.Id, fileId);  await toast.ShowAsync(new ToastModel { Title = "File added to Vector store", Content = "File linked with vector store", CssClass = "e-toast-success", Icon = "e-info toast-icons" }); |

Step 2: Create the method **AddFileToVectorStore** as like below.

|  |
| --- |
| static async Task AddFileToVectorStore(string vectorid,string fileId)  {  var VECTOR\_STORE\_ID = vectorid;  var requestUrl = $"{API\_URL}/vector\_stores/{VECTOR\_STORE\_ID}/files";  var requestData = new { file\_id = fileId };  var content = new StringContent(JsonSerializer.Serialize(requestData), Encoding.UTF8, "application/json");  var response = await httpClient.PostAsync(requestUrl, content);  response.EnsureSuccessStatusCode();  var jsonContent = await response.Content.ReadAsStringAsync();  } |

### Update the assistant with the vector store ID

Step 1: The next step is to update the assistant with the vector store using the vector store ID, inside the **Search** method, as shown below.

|  |
| --- |
| //Update Assistant with Vector Store  var newAssist = await UpdateAssistant(assistant.Id,vectorStore.Id);  await toast.ShowAsync(new ToastModel { Title = "Link Vector", Content = "Linked Vector store to assistant", CssClass = "e-toast-success", Icon = "e-info toast-icons" }); |

Step 2: Create the method **UpdateAssistant** like below.

|  |
| --- |
| private static async Task<AssistantResponse> UpdateAssistant(string assistantID, string vectoreStoreId)  {  var openAIPrompt = new  {  tool\_resources = new  {  file\_search = new  {  vector\_store\_ids= new[] { vectoreStoreId }  }  }  };  var content = new StringContent(JsonSerializer.Serialize(openAIPrompt), Encoding.UTF8, "application/json");  var response = await httpClient.PostAsync($"{API\_URL}/assistants/{assistantID}", content);  response.EnsureSuccessStatusCode();  var jsonContent = await response.Content.ReadAsStringAsync();  return JsonSerializer.Deserialize<AssistantResponse>(jsonContent);  } |

### Creating a thread to ask a question

Step 1: Add the below code inside the **Search** method to create a new Thread and assign that thread to the Assistant by assigning the vector store id like below.

|  |
| --- |
| // Create a thread  threadId = await CreateThread(fileId, vectorStore.Id);  await toast.ShowAsync(new ToastModel { Title = "Create Thread", Content = "Create a new Thread for the assistant" , CssClass = "e-toast-success", Icon = "e-info toast-icons" }); |

Step 2: Define a **CreateThread** method like below.

|  |
| --- |
| private static async Task<string> CreateThread(string fileId, string vectorID)  {  var thread = new  {    tool\_resources = new  {  file\_search = new  {  vector\_store\_ids = new[] { vectorID },  }  }  };  var content = new StringContent(JsonSerializer.Serialize(thread), Encoding.UTF8, "application/json");  var response = await httpClient.PostAsync($"{API\_URL}/threads", content);  response.EnsureSuccessStatusCode();  var jsonContent = await response.Content.ReadAsStringAsync();  var threadResponse = JsonSerializer.Deserialize<ThreadResponse>(jsonContent);  return threadResponse.Id;  } |

Step 3: For this method to work, we need a new class called **ThreadResponse** with some properties to deserialize the response from the OpenAI.

|  |
| --- |
| public class ThreadResponse  {  [JsonPropertyName("id")]  public string Id { get; set; }  } |

### Create a run to execute the thread and retrieve the answers

Step 1: To create a run for the thread, add below code inside the **Search** method.

|  |
| --- |
| // Create a Run for the thread  var runId =await CreateRun(assistantId, threadId,toast);  await toast.ShowAsync(new ToastModel { Title = "Create a Run", Content = "Run the thread on the assistant", CssClass = "e-toast-success", Icon = "e-info toast-icons" }); |

Step 2: Define the **CreateRun** method like below.

|  |
| --- |
| private static async Task<String> CreateRun(string assistantId, string threadId, SfToast toast)  {  var runContent = new { assistant\_id = assistantId };  var content = new StringContent(JsonSerializer.Serialize(runContent), Encoding.UTF8, "application/json");  var response = await httpClient.PostAsync($"{API\_URL}/threads/{threadId}/runs", content);  response.EnsureSuccessStatusCode();  var jsonContent = await response.Content.ReadAsStringAsync();  var runResponse = JsonSerializer.Deserialize<RunResponse>(jsonContent);  var runID = runResponse.Id;  while(runResponse.Status!="completed")  {  response = await httpClient.GetAsync($"{API\_URL}/threads/{threadId}/runs/{runID}");  response.EnsureSuccessStatusCode();  jsonContent = await response.Content.ReadAsStringAsync();  runResponse = JsonSerializer.Deserialize<RunResponse>(jsonContent);  if(toast!=null)  {  await toast.ShowAsync(new ToastModel { Title = "Waiting!", Content = "Waiting for the run to complete", CssClass = "e-toast-info", Icon = "e-info toast-icons" });  }  await Task.Delay(1000);  }  return runID;  } |

Step 3: For this method to work, we need new classes called **RunResponse**, **Message1**, **Content**, **Annotation**, **FileCitation** with some properties to deserialize the response from the OpenAI.

|  |
| --- |
| public class RunResponse  {  [JsonPropertyName("id")]  public string Id { get; set; }  [JsonPropertyName("status")]  public string Status { get; set; }  [JsonPropertyName("messages")]  public System.Collections.Generic.List<Message1> Messages { get; set; }  }  public class Message1  {  [JsonPropertyName("content")]  public Content[] Content { get; set; }  }  public class Content  {  [JsonPropertyName("type")]  public string Type { get; set; }  [JsonPropertyName("text")]  public string Text { get; set; }  [JsonPropertyName("annotations")]  public Annotation[] Annotations { get; set; }  }  public class Annotation  {  [JsonPropertyName("text")]  public string Text { get; set; }  [JsonPropertyName("filecitation")]  public FileCitation FileCitation { get; set; }  }  public class FileCitation  {  [JsonPropertyName("fileid")]  public string FileId { get; set; }  } |

### Retrieve the message from the thread

Step 1: Inside the same AssistantFileSearch class, create a **GetMessages** method to retrieve the message from the thread.

|  |
| --- |
| public async Task<string> GetMessages(SfToast toast)  {  var response = await httpClient.GetAsync($"{API\_URL}/threads/{threadId}/messages");  response.EnsureSuccessStatusCode();  var jsonContent = await response.Content.ReadAsStringAsync();  var threadResponse = JsonSerializer.Deserialize<ThreadMessageResponse>(jsonContent);  var messages = threadResponse.Data;  var message = messages[0];  var text1 = "";  if (message.Content[0].Type == "text")  {  text1 = message.Content[0].Text.Value;  }  if (toast != null) {  await toast.ShowAsync(new ToastModel { Title = "Success!", Content = "Retrieved message from the thread", CssClass = "e-toast-success", Icon = "e-success toast-icons" });  }  return text1;  } |

Step 2: For this method to work, we need new classes called **ThreadMessageResponse**, **MessageData** , **ContentItem**, **TextItem** with some properties to deserialize the response from the OpenAI.

|  |
| --- |
| public class ThreadMessageResponse  {  [JsonPropertyName("object")]  public string Object { get; set; }  [JsonPropertyName("data")]  public List<MessageData> Data { get; set; }  [JsonPropertyName("firstid")]  public string FirstId { get; set; }  [JsonPropertyName("lastid")]  public string LastId { get; set; }  [JsonPropertyName("hasmore")]  public bool HasMore { get; set; }  }  public class MessageData  {  [JsonPropertyName("id")]  public string Id { get; set; }  [JsonPropertyName("object")]  public string Object { get; set; }  [JsonPropertyName("createdat")]  public long CreatedAt { get; set; }  [JsonPropertyName("assistantid")]  public string AssistantId { get; set; }  [JsonPropertyName("threadid")]  public string ThreadId { get; set; }  [JsonPropertyName("runid")]  public string RunId { get; set; }  [JsonPropertyName("role")]  public string Role { get; set; }  [JsonPropertyName("content")]  public List<ContentItem> Content { get; set; }  [JsonPropertyName("attachments")]  public List<object> Attachments { get; set; }  //public Dictionary<string, object> Metadata { get; set; }  }  public class ContentItem  {  [JsonPropertyName("type")]  public string Type { get; set; }  [JsonPropertyName("text")]  public TextItem Text { get; set; }  }  public class TextItem  {  [JsonPropertyName("value")]  public string Value { get; set; }  [JsonPropertyName("annotations")]  public List<Annotation> Annotations { get; set; }  } |

### Asking questions to the AI

Step 1: After getting the messages, we need to ask questions to the AI using the new method **AskQuestionToAssistant** like below. Add this method inside the **AssistantFileSearch** class.

|  |
| --- |
| public async Task AskQuestionToAssistant(string query,SfToast toast)  {  var message = new  {  role = "user",  content = query,  };  var content = new StringContent(JsonSerializer.Serialize(message), Encoding.UTF8, "application/json");  var response = await httpClient.PostAsync($"{API\_URL}/threads/{threadId}/messages",content);  response.EnsureSuccessStatusCode();  await toast.ShowAsync(new ToastModel { Title = "Success!", Content = "User Message added to the thread", CssClass = "e-toast-success", Icon = "e-success toast-icons" });  var runId = await CreateRun(assistantId, threadId,toast);  } |

Step 2: Initialize the **AssistantFileSearch** class like below inside the **@code** section of ***Pages/index.razor*** file.

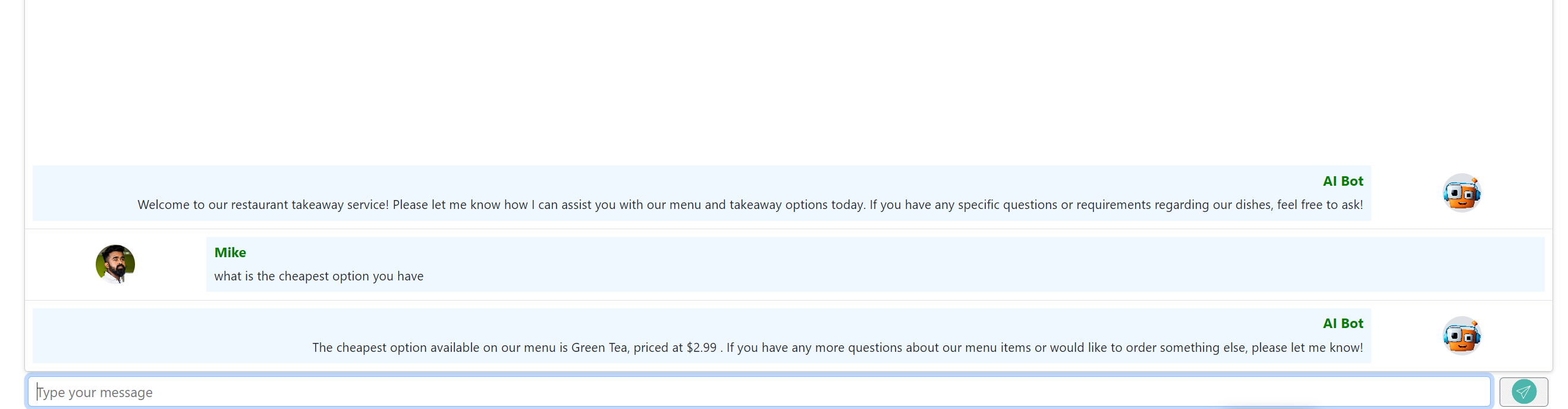
|  |
| --- |
| private AssistantFileSearch assistantFileSearch = new AssistantFileSearch(); |

Step 3: Now that we have completed all our methods to create and communicate with the assistant we need to call the functions on the **EventClick** method in the **index.razor** file by removing the previous local file search code.

|  |
| --- |
| await assistantFileSearch.Search(this.ToastObj);  reply = await assistantFileSearch.GetMessages(this.ToastObj); |

Step 4: Delete the previous code inside the **GetResult** method and add the below code.

|  |
| --- |
| #region Request user query to an AI Assistant via a dedicated user thread.  await assistantFileSearch.AskQuestionToAssistant(query, this.ToastObj);  result = await assistantFileSearch.GetMessages(this.ToastObj);  #endregion  return result; |

Now when we rerun the application, we can ask questions related to the menu, and it will provide the appropriate response. 

A screenshot of a computer

Description automatically generated

## Creating a Hybrid app using MAUI

Let us create a hybrid app using MAUI and deploy this Web application into Windows forms, Android, IOS, etc.

Step 1: In Visual Studio and click “Create a new project”.

A screenshot of a computer program

Description automatically generated

Step 2: Search **.Net Maui Blazor Hybrid App** in the search bar, choose the project, and click next.

A screenshot of a computer

Description automatically generated

Step 3: Name the project **Chat Bot** and click next.

A screenshot of a computer

Description automatically generated

Step 4: Choose **.Net 8 Long Term Support** and click **Create**.

A screenshot of a computer

Description automatically generated

We need to use the exact code from the **Blazor Server App** here, so that everything works in other platforms.

Step 1: Install the required Syncfusion NuGet packages.

*Syncfusion.Blazor.Buttons*

*Syncfusion.Blazor.Inputs*

*Syncfusion.Blazor.Lists*

*Syncfusion.Blazor.Notifications*

*Syncfusion.Blazor.Spinner*

*Syncfusion.Blazor.SplitButtons*

*Syncfusion.Blazor.Themes*

Step 2: Go to the **wwwroot** folder and open the **index.html** file to add the Syncfusion’s theme and script files inside the head tag.

|  |
| --- |
| <link href="\_content/Syncfusion.Blazor.Themes/bootstrap5.css" rel="stylesheet" />  <script src="\_content/Syncfusion.Blazor.Core/scripts/syncfusion-blazor.min.js" type="text/javascript"></script> |

Step 3: Create a folder named **Data** inside the **Components** folder.

A screenshot of a computer

Description automatically generated

Step 4: Inside the **Data** folder, create the **AssistantFileSearch.cs** class file and copy-paste the below code we used in the blazor server file.

|  |
| --- |
| using System;  using System.IO;  using System.IO.Pipes;  using System.Net.Http;  using System.Net.Http.Headers;  using System.Numerics;  using System.Runtime.Intrinsics;  using System.Text;  using System.Text.Json;  using System.Text.Json.Serialization;  using System.Threading;  using System.Threading.Tasks;  using Microsoft.Extensions.Configuration;  using Syncfusion.Blazor.Notifications;  using static System.Net.WebRequestMethods; |

Step 5: Inside your **AssistantFileSearch** namespace, replace the below code with the existing class.

|  |
| --- |
| public class AssistantFileSearch  {  private static readonly HttpClient httpClient = new HttpClient();  private string OPENAI\_KEY = "your open ai key";  private const string API\_URL = "https://api.openai.com/v1";  public string threadId = "";  public string assistantId = "";  public async Task Search(SfToast toast)  {  httpClient.DefaultRequestHeaders.Clear();  httpClient.DefaultRequestHeaders.Add("Authorization", $"Bearer {OPENAI\_KEY}");  httpClient.DefaultRequestHeaders.Add("OpenAI-Beta", "assistants=v2");  // Create Assistant  var assistant = await CreateAssistant();  await toast.ShowAsync(new ToastModel { Title = "Assistant Created", Content = "Created a new Assistant with id" + assistant.Id, CssClass = "e-toast-success", Icon = "e-info toast-icons" });  assistantId = assistant.Id;  // Create Vector Store  var vectorStore = await CreateVectorStore();  await toast.ShowAsync(new ToastModel { Title = "Vector Store Created", Content = "Created a new Vector store with id" + vectorStore.Id, CssClass = "e-toast-success", Icon = "e-info toast-icons" });  // Upload the file  var fileId = await UploadFile("DataFiles/restaurant\_menu.txt", "assistants");  await toast.ShowAsync(new ToastModel { Title = "File Uploaded", Content = "File uploaded with id" + fileId, CssClass = "e-toast-success", Icon = "e-info toast-icons" });  // Add the file to the vector store  await AddFileToVectorStore(vectorStore.Id, fileId);  await toast.ShowAsync(new ToastModel { Title = "File added to Vector store", Content = "File linked with vector store", CssClass = "e-toast-success", Icon = "e-info toast-icons" });  //Update Assistant with Vector Store  var newAssist = await UpdateAssistant(assistant.Id, vectorStore.Id);  await toast.ShowAsync(new ToastModel { Title = "Link Vector", Content = "Linked Vector store to assistant", CssClass = "e-toast-success", Icon = "e-info toast-icons" });  // Create a thread  threadId = await CreateThread(fileId, vectorStore.Id);  await toast.ShowAsync(new ToastModel { Title = "Create Thread", Content = "Create a new Thread for the assistant", CssClass = "e-toast-success", Icon = "e-info toast-icons" });  // Create a Run for the thread  var runId = await CreateRun(assistantId, threadId, toast);  await toast.ShowAsync(new ToastModel { Title = "Create a Run", Content = "Run the thread on the assistant", CssClass = "e-toast-success", Icon = "e-info toast-icons" });  }  public async Task AskQuestionToAssistant(string query, SfToast toast)  {  var message = new  {  role = "user",  content = query,  };  var content = new StringContent(JsonSerializer.Serialize(message), Encoding.UTF8, "application/json");  var response = await httpClient.PostAsync($"{API\_URL}/threads/{threadId}/messages", content);  response.EnsureSuccessStatusCode();  await toast.ShowAsync(new ToastModel { Title = "Success!", Content = "User Message added to the thread", CssClass = "e-toast-success", Icon = "e-success toast-icons" });  var runId = await CreateRun(assistantId, threadId, toast);  }  public async Task<string> GetMessages(SfToast toast)  {  var response = await httpClient.GetAsync($"{API\_URL}/threads/{threadId}/messages");  response.EnsureSuccessStatusCode();  var jsonContent = await response.Content.ReadAsStringAsync();  var threadResponse = JsonSerializer.Deserialize<ThreadMessageResponse>(jsonContent);  var messages = threadResponse.Data;  var message = messages[0];  var text1 = "";  if (message.Content[0].Type == "text")  {  text1 = message.Content[0].Text.Value;  }  if (toast != null)  {  await toast.ShowAsync(new ToastModel { Title = "Success!", Content = "Retrieved message from the thread", CssClass = "e-toast-success", Icon = "e-success toast-icons" });  }  return text1;  }  private static async Task<String> CreateRun(string assistantId, string threadId, SfToast toast)  {  var runContent = new { assistant\_id = assistantId };  var content = new StringContent(JsonSerializer.Serialize(runContent), Encoding.UTF8, "application/json");  var response = await httpClient.PostAsync($"{API\_URL}/threads/{threadId}/runs", content);  response.EnsureSuccessStatusCode();  var jsonContent = await response.Content.ReadAsStringAsync();  var runResponse = JsonSerializer.Deserialize<RunResponse>(jsonContent);  var runID = runResponse.Id;  while (runResponse.Status != "completed")  {  response = await httpClient.GetAsync($"{API\_URL}/threads/{threadId}/runs/{runID}");  response.EnsureSuccessStatusCode();  jsonContent = await response.Content.ReadAsStringAsync();  runResponse = JsonSerializer.Deserialize<RunResponse>(jsonContent);  if (toast != null)  {  await toast.ShowAsync(new ToastModel { Title = "Waiting!", Content = "Waiting for the run to complete", CssClass = "e-toast-info", Icon = "e-info toast-icons" });  }  await Task.Delay(1000);  }  return runID;  }  private static async Task<string> CreateThread(string fileId, string vectorID)  {  var thread = new  {  tool\_resources = new  {  file\_search = new  {  vector\_store\_ids = new[] { vectorID },  }  }  };  var content = new StringContent(JsonSerializer.Serialize(thread), Encoding.UTF8, "application/json");  var response = await httpClient.PostAsync($"{API\_URL}/threads", content);  response.EnsureSuccessStatusCode();  var jsonContent = await response.Content.ReadAsStringAsync();  var threadResponse = JsonSerializer.Deserialize<ThreadResponse>(jsonContent);  return threadResponse.Id;  }  private static async Task<AssistantResponse> UpdateAssistant(string assistantID, string vectoreStoreId)  {  var openAIPrompt = new  {  tool\_resources = new  {  file\_search = new  {  vector\_store\_ids = new[] { vectoreStoreId }  }  }  };  var content = new StringContent(JsonSerializer.Serialize(openAIPrompt), Encoding.UTF8, "application/json");  var response = await httpClient.PostAsync($"{API\_URL}/assistants/{assistantID}", content);  response.EnsureSuccessStatusCode();  var jsonContent = await response.Content.ReadAsStringAsync();  return JsonSerializer.Deserialize<AssistantResponse>(jsonContent);  }  static async Task AddFileToVectorStore(string vectorid, string fileId)  {  var VECTOR\_STORE\_ID = vectorid;  var requestUrl = $"{API\_URL}/vector\_stores/{VECTOR\_STORE\_ID}/files";  var requestData = new { file\_id = fileId };  var content = new StringContent(JsonSerializer.Serialize(requestData), Encoding.UTF8, "application/json");  var response = await httpClient.PostAsync(requestUrl, content);  response.EnsureSuccessStatusCode();  var jsonContent = await response.Content.ReadAsStringAsync();  }  private static async Task<string> UploadFile(string filePath, string purpose)  {  // using var fileStream = new FileStream(filePath, FileMode.Open);  using var fileStream =  await FileSystem.OpenAppPackageFileAsync("restaurant\_menu.txt");  using var reader = new StreamReader(fileStream);  var formData = new MultipartFormDataContent();  formData.Add(new StreamContent(fileStream), "file", Path.GetFileName(filePath));  formData.Add(new StringContent(purpose), "purpose");  var response = await httpClient.PostAsync($"{API\_URL}/files", formData);  response.EnsureSuccessStatusCode();  var jsonContent = await response.Content.ReadAsStringAsync();  var fileResponse = JsonSerializer.Deserialize<FileResponse>(jsonContent);  return fileResponse.Id;  }  private static async Task<VectorStoreResponse> CreateVectorStore()  {  var vectorStoreData = new { name = "Syncfusion Blazor" };  var content = new StringContent(JsonSerializer.Serialize(vectorStoreData), Encoding.UTF8, "application/json");  var response = await httpClient.PostAsync($"{API\_URL}/vector\_stores", content);  response.EnsureSuccessStatusCode();  var jsonContent = await response.Content.ReadAsStringAsync();  return JsonSerializer.Deserialize<VectorStoreResponse>(jsonContent);  }  private static async Task<AssistantResponse> CreateAssistant()  {  var openAIPrompt = new  {  name = "Restaurant take away manager",  instructions = "You are an expert in Restaurant take away and answer anything related to the menu, also always keep the conversation about the restaurant menu even if user asks any non related questions divert back them to ask menu related questions.",  model = "gpt-4-turbo",  tools = new[] { new { type = "file\_search" } }  };  var content = new StringContent(JsonSerializer.Serialize(openAIPrompt), Encoding.UTF8, "application/json");  var response = await httpClient.PostAsync($"{API\_URL}/assistants", content);  response.EnsureSuccessStatusCode();  var jsonContent = await response.Content.ReadAsStringAsync();  return JsonSerializer.Deserialize<AssistantResponse>(jsonContent);  }  }  public class AssistantResponse  {  [JsonPropertyName("id")]  public string Id { get; set; }  }  public class VectorStoreResponse  {  [JsonPropertyName("id")]  public string Id { get; set; }  }  public class FileResponse  {  [JsonPropertyName("id")]  public string Id { get; set; }  [JsonPropertyName("filename")]  public string Filename { get; set; }  }  public class ThreadResponse  {  [JsonPropertyName("id")]  public string Id { get; set; }  }  public class RunResponse  {  [JsonPropertyName("id")]  public string Id { get; set; }  [JsonPropertyName("status")]  public string Status { get; set; }  [JsonPropertyName("messages")]  public System.Collections.Generic.List<Message1> Messages { get; set; }  }  public class Message1  {  [JsonPropertyName("content")]  public Content[] Content { get; set; }  }  public class Content  {  [JsonPropertyName("type")]  public string Type { get; set; }  [JsonPropertyName("text")]  public string Text { get; set; }  [JsonPropertyName("annotations")]  public Annotation[] Annotations { get; set; }  }  public class Annotation  {  [JsonPropertyName("text")]  public string Text { get; set; }  [JsonPropertyName("filecitation")]  public FileCitation FileCitation { get; set; }  }  public class FileCitation  {  [JsonPropertyName("fileid")]  public string FileId { get; set; }  }  public class ThreadMessageResponse  {  [JsonPropertyName("object")]  public string Object { get; set; }  [JsonPropertyName("data")]  public List<MessageData> Data { get; set; }  [JsonPropertyName("firstid")]  public string FirstId { get; set; }  [JsonPropertyName("lastid")]  public string LastId { get; set; }  [JsonPropertyName("hasmore")]  public bool HasMore { get; set; }  }  public class MessageData  {  [JsonPropertyName("id")]  public string Id { get; set; }  [JsonPropertyName("object")]  public string Object { get; set; }  [JsonPropertyName("createdat")]  public long CreatedAt { get; set; }  [JsonPropertyName("assistantid")]  public string AssistantId { get; set; }  [JsonPropertyName("threadid")]  public string ThreadId { get; set; }  [JsonPropertyName("runid")]  public string RunId { get; set; }  [JsonPropertyName("role")]  public string Role { get; set; }  [JsonPropertyName("content")]  public List<ContentItem> Content { get; set; }  [JsonPropertyName("attachments")]  public List<object> Attachments { get; set; }  //public Dictionary<string, object> Metadata { get; set; }  }  public class ContentItem  {  [JsonPropertyName("type")]  public string Type { get; set; }  [JsonPropertyName("text")]  public TextItem Text { get; set; }  }  public class TextItem  {  [JsonPropertyName("value")]  public string Value { get; set; }  [JsonPropertyName("annotations")]  public List<Annotation> Annotations { get; set; }  } |

Step 6: Goto ***Components/Layout/MainLayout.razor*** page, delete the **div** element, and add the below code.

|  |
| --- |
| <div class="page">  <main>  @Body  </main>  </div> |

Step 7: Go to ***Components/\_imports.razor*** file, add the Syncfusion references.

|  |
| --- |
| @using Syncfusion.Blazor  @using Syncfusion.Blazor.Lists  @using Syncfusion.Blazor.Inputs  @using System.Collections.ObjectModel  @using Syncfusion.Blazor.Spinner  @using Syncfusion.Blazor.SplitButtons  @using Syncfusion.Blazor.Notifications  @using Syncfusion.Blazor.Buttons |

Step 8: Also add the same **restaurant\_menu.txt** file under the **Resources-> Raw** folder.

A screenshot of a computer menu

Description automatically generated

Step 9: Go to ***Components/Pages*** folder, and open ***Home.razor*** file and replace the existing code with the below code.

|  |
| --- |
| @page "/"  @inject HttpClient HttpClient  <PageTitle>AI Chat Bot</PageTitle>  @using Chat\_Bot.Components.Data  @if (VisibleFAB)  {  <SfFab @ref="FabObject" Target="#container" IconCss="demo-icons chat-icon" OnClick="EventClick" CssClass="custom-fab"></SfFab>  }  <div id="container" >    <div id="sample" style="visibility:@ContainerVisibility">  <div style="color:white; Background:deepskyblue; width:100%; height:50px; padding:10px; font-size:23px">  AI Chat Bot  </div>  <div>  <SfSpinner @bind-Visible="@VisibleProperty">  </SfSpinner>  </div>  <form method="post" @onsubmit="OnSend" >  <SfListView ID="list" class="flex flex\_\_direction"  DataSource="@DataSource" Height="89vh" >  <ListViewFieldSettings TValue="ListDataModel" Id="Id" Text="Text"></ListViewFieldSettings>  <ListViewTemplates TValue="ListDataModel">  <Template>  @{  ListDataModel currentData = context as ListDataModel;  <div class="flex item\_\_container">  <div class="flex flex\_\_1 vertical\_\_center flex\_\_center @(currentData.Chat == "sender" ? "flex\_\_order\_\_2" : "")">  @if (currentData.Avatar != "")  {  <span class="e-avatar e-avatar-circle">@currentData.Avatar</span>  }  else  {  <span class="@currentData.Pic e-avatar e-avatar-circle"></span>  }  </div>    <div class="flex content\_\_container flex\_\_8 vertical padding @(currentData.Chat == "sender" ? "right\_\_align" : "left\_\_align")">  <div class="bold">@currentData.Text</div>  <div class="small\_\_font">@((MarkupString)@currentData.Contact)</div>  </div>  </div>  }  </Template>  </ListViewTemplates>  </SfListView>  <div class="flex">  <div class="flex\_\_8 padding">  <SfTextBox Placeholder="Type your message"  @ref="@ChatTextBox"  @oninput="@(e=>OnTyping(e))"></SfTextBox>  </div>  <div class="flex">  <button class="e-btn" ></button>  </div>  <div>  <SfToast @ref="ToastObj" ID="toast\_type">  <ToastPosition X="@ToastPosition"></ToastPosition>  </SfToast>  </div>  </div>  </form>  </div>  </div>  @code  {  SfToast ToastObj;  SfFab FabObject;  SfTextBox ChatTextBox;  string textBoxValue = "";  private string ToastPosition = "Right";  ObservableCollection<ListDataModel> DataSource = new ObservableCollection<ListDataModel>();  public string result = "";  public string reply = "";  private string? fileContent = "";  private AssistantFileSearch assistantFileSearch = new AssistantFileSearch();  public async Task EventClick()  {  this.VisibleFAB = false;  StateHasChanged();  this.ContainerVisibility = "visible";  await assistantFileSearch.Search(this.ToastObj);  reply = await assistantFileSearch.GetMessages(this.ToastObj);    await Task.Delay(3500);  this.VisibleProperty = false;  DataSource.Add(new ListDataModel  {  Text = "AI Bot",  Contact = reply,  Id = "1",  Avatar = "",  Pic = "pic01",  Chat = "sender"  });  }  protected override async Task OnInitializedAsync()  {  this.VisibleProperty = true;  }  private async Task<string> GetResult(string query)  {  #region Request user query to an AI Assistant via a dedicated user thread.  await assistantFileSearch.AskQuestionToAssistant(query, this.ToastObj);  result = await assistantFileSearch.GetMessages(this.ToastObj);  #endregion  return result;  }    async Task OnSend()  {  if (ChatTextBox.Value != null)  {  textBoxValue = ChatTextBox.Value;  ChatTextBox.Value = "";  DataSource.Add(new ListDataModel  {  Text = "Mike",  Contact = textBoxValue,  Id = new Random().Next(300, 900).ToString(),  Avatar = "",  Pic = "pic02",  Chat = "receiver"  });  this.VisibleProperty = true;  result = await GetResult(textBoxValue);  this.VisibleProperty = false;  DataSource.Add(new ListDataModel  {  Text = "AI Bot",  Contact = result,  Id = "1",  Avatar = "",  Pic = "pic01",  Chat = "sender"  });  }  else  {  DataSource.Add(new ListDataModel  {  Text = "Mike",  Contact = textBoxValue,  Id = new Random().Next(300, 900).ToString(),  Avatar = "",  Pic = "pic02",  Chat = "receiver"  });  DataSource.Add(new ListDataModel  {  Text = "AI Bot",  Contact = "Please enter a valid query",  Id = "1",  Avatar = "",  Pic = "pic01",  Chat = "sender"  });  }  }  async Task OnTyping(ChangeEventArgs args)  {  if (args.Value != "")  {  textBoxValue = args.Value.ToString();  }  }  public class ListDataModel  {  public string Id  {  get;  set;  }  public string Chat  {  get;  set;  }  public string Pic  {  get;  set;  }  public string Avatar  {  get;  set;  }  public string Text  {  get;  set;  }  public string Contact  {  get;  set;  }  }  private bool VisibleProperty { get; set; } = false;  private bool VisibleFAB { get; set; } = true;  private string ContainerVisibility { get; set; } = "hidden";  }  <style>  #list {  box-shadow: 0 1px 4px #ddd;  border-bottom: 1px solid #ddd;  }  #list {  margin: 0 auto;  border: 1px solid #ccc;  }  #list .e-list-item {  height: auto;  cursor: pointer;  line-height: 22px;  padding: 8px;  }  #list.e-listview{  overflow-y:scroll;  position:sticky;  scrollbar-width:none;  }  #list.e-listview .e-list-container {  overflow:unset;  position:unset;  }  #list.e-listview .e-list-header {  background-color: #0278d7;  color: white;  }  #list .e-list-item.e-active {  background-color: transparent;  }  #list .e-list-item.e-hover {  background-color: transparent;  }  .padding {  padding: 4px;  }  .right\_\_align {  text-align: right;  margin-right: 8px;  padding-right: 8px;  }  .left\_\_align {  margin-left: 8px;  padding-left: 8px;  }  .content\_\_container {  background-color: aliceblue;  }  .e-btn{  background-image: url("../send.png");  background-repeat: no-repeat;  background-color :white;  height: 30px;  width:50px;  background-size: 100%;  margin: 5px;    }  .e-btn:hover{    background-image: url("../send.png");  background-repeat: no-repeat;  background-size: 100%;  }  .e-btn:focus{  background-image: url("../send.png");  background-repeat: no-repeat;  background-color: white;  height: 30px;  width: 50px;  background-size: 100%;  margin: 5px;  }  .flex {  display: flex;  }  .flex\_\_direction{  flex-direction: column-reverse;  flex-grow: 1;  }  .flex\_\_center {  justify-content: center;  }  .vertical\_\_center {  align-items: center;  }  .vertical {  flex-direction: column;  }  .flex\_\_order\_\_1 {  order: 1;  }  .flex\_\_order\_\_2 {  order: 2;  }  .flex\_\_1 {  flex: 1;  }  .flex\_\_2 {  flex: 2;  }  .flex\_\_3 {  flex: 3;  }  .flex\_\_5 {  flex: 5;  }  .flex\_\_8 {  flex: 8;  }  .bold {  font-weight: bold;  color:green;  }  .margin {  margin: 10px;  }  .small\_\_font {  font-size: 13px;  margin: 2px 0;  }  .pic01 {  background-image: url("../cody.png");  }  .pic02 {  background-image: url("../mike.jpg");  }  .pic03 {  background-image: url("https://ej2.syncfusion.com/demos/src/grid/images/5.png");  }  .pic04 {  background-image: url("https://ej2.syncfusion.com/demos/src/grid/images/2.png");  }  @@font-face {  font-family: 'Toast\_icons';  src: url(data:application/x-font-ttf;charset=utf-8;base64,) format('truetype');  font-weight: normal;  font-style: normal;  }  #toast\_types button {  margin: 5px;  min-width: 160px;  max-width: 160px;  }  .toast-icons {  font-family: 'Toast\_icons' !important;  speak: none;  font-size: 55px;  font-style: normal;  font-weight: normal;  font-variant: normal;  text-transform: none;  line-height: 1;  -webkit-font-smoothing: antialiased;  -moz-osx-font-smoothing: grayscale;  }  #toast\_type .e-toast-icon.e-icons {  height: auto;  font-size: 30px;  }  .bootstrap5 #toast\_type .e-toast-icon.e-icons,  .bootstrap5-dark #toast\_type .e-toast-icon.e-icons {  height: 25px;  }  .toast-icons.e-success::before {  content: "\e701";  }  .toast-icons.e-error::before {  content: "\e700";  }  .toast-icons.e-info::before {  content: "\e704";  }  .toast-icons.e-warning::before {  content: "\e703";  }  #toast\_types {  text-align: center;  }  @@font-face {  font-family: 'demo-icons';  src:  url(data:application/x-font-ttf;charset=utf-8;base64,) format('truetype');  font-weight: normal;  font-style: normal;  }  [class^="demo-icons"], [class\*=" demo-icons"] {  font-family: 'demo-icons' !important;  speak: none;  font-size: 55px;  font-style: normal;  font-weight: normal;  font-variant: normal;  text-transform: none;  line-height: 1;  -webkit-font-smoothing: antialiased;  -moz-osx-font-smoothing: grayscale;  }  .chat-icon:before { content: "\e700"; }  custom-fab.e-fab.e-btn .e-btn-icon {  font-size: 30px;  }  </style> |

Step 10: Add all the required images inside the **wwwroot folder** for the send button, user profile pic, and the chat bot.

A screenshot of a computer

Description automatically generated

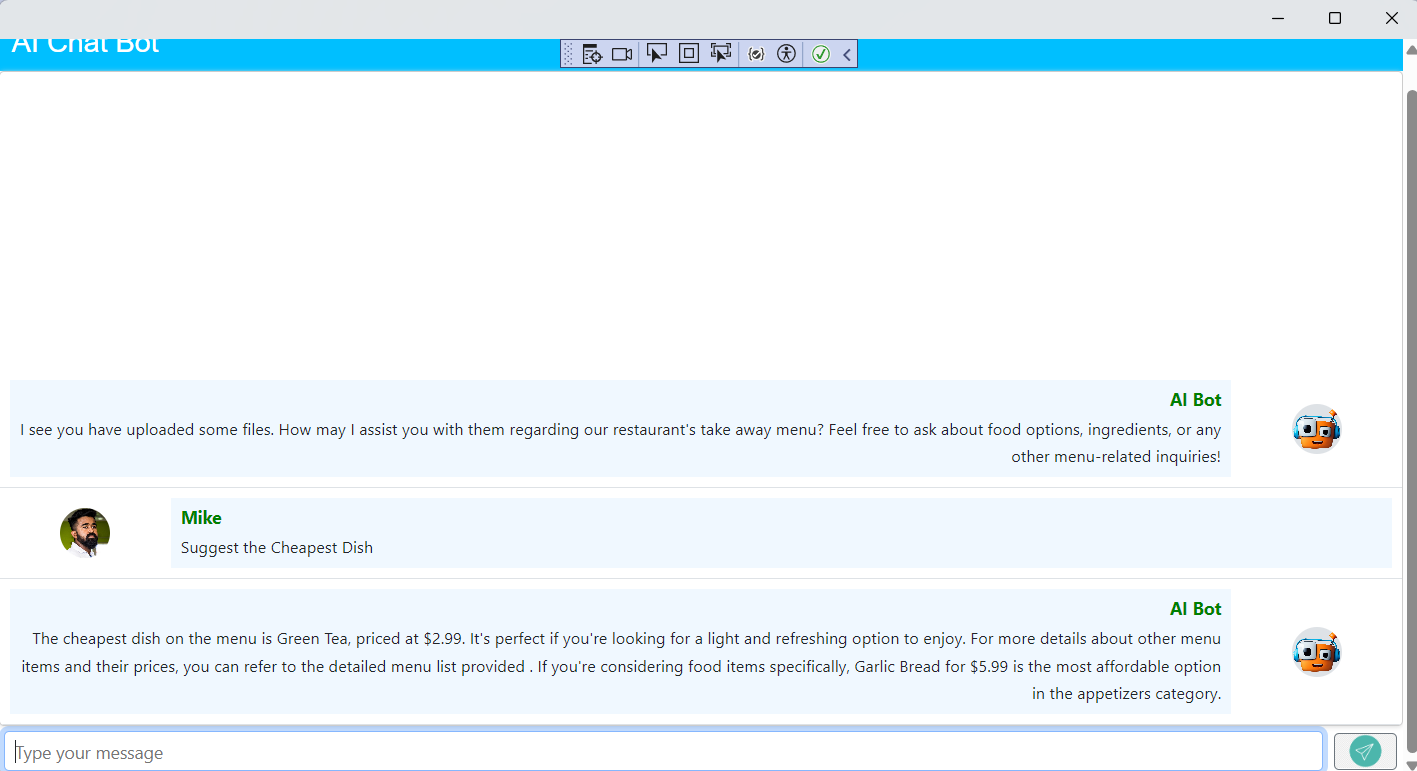
Step 11: Open the **MauiProgram.cs** file and add the **Syncfusion Blazor** namespace.

|  |
| --- |
| using Syncfusion.Blazor; |

Then add the below highlighted code for **Syncfusion license** key registration.

|  |
| --- |
| Syncfusion.Licensing.SyncfusionLicenseProvider.RegisterLicense("Your license key");  builder.Services.AddSyncfusionBlazor();  builder.Services.AddMauiBlazorWebView();  #if DEBUG    builder.Services.AddBlazorWebViewDeveloperTools();  builder.Logging.AddDebug();  #endif |

Run the application, check the Chat Bot in the Windows forms.



Chat Bot in Android looks like below.

A white rectangular frame with colorful borders

Description automatically generated

When I try to click the text box, the keyboard will cover the textbox as shown below.

A screenshot of a computer

Description automatically generated

Uncovering the textbox UI in Android  
To avoid this scenario, we must add few codes. Go to **app.xaml.cs** file and add the following code after **mainpage** initialization**.**

|  |
| --- |
| Current.On<Microsoft.Maui.Controls.PlatformConfiguration.Android>().UseWindowSoftInputModeAdjust(WindowSoftInputModeAdjust.Resize); |

And inherit **Microsoft.Maui.Controls.Application** instead of **Application.**

A screenshot of a chat

Description automatically generated

That’s it. We have created a simple chatbot for all the platforms.