# Create a generative AI chat app

In this exercise, you use the Azure AI Foundry SDK to create a simple chat app that connects to a project and chats with a language model.

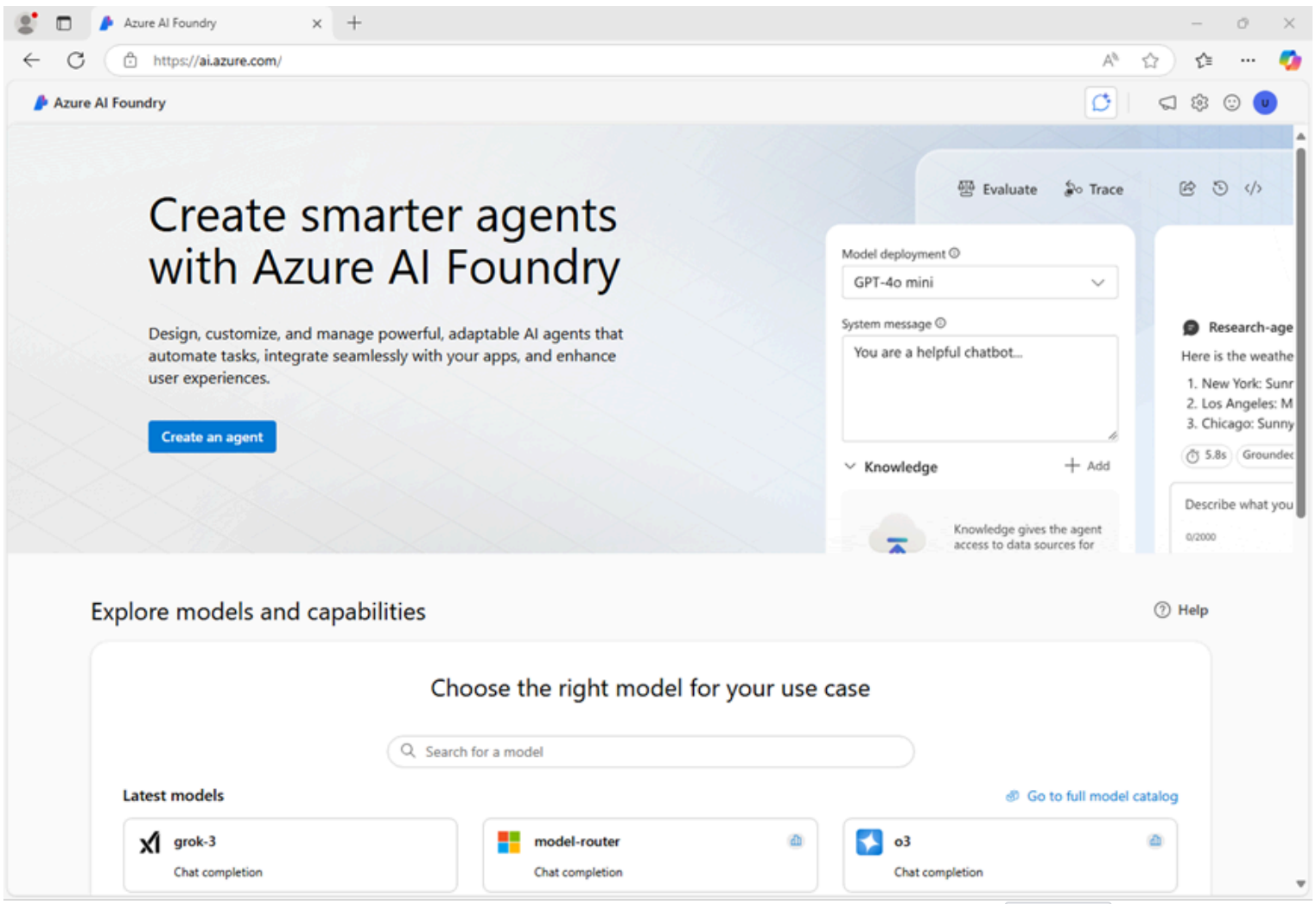
This exercise takes approximately **40** minutes.

> **Note**: This exercise is based on pre-release SDKs, which may be subject to change. Where necessary, we've used specific versions of packages; which may not reflect the latest available versions. You may experience some unexpected behavior, warnings, or errors.

## Deploy a model in an Azure AI Foundry project

Let's start by deploying a model in an Azure AI Foundry project.

1. In a web browser, open the Azure AI Foundry portal at `https://ai.azure.com` and sign in using your Azure credentials. Close any tips or quick start panes that are opened the first time you sign in, and if necessary use the **Azure AI Foundry** logo at the top left to navigate to the home page, which looks similar to the following image (close the **Help** pane if it's open):

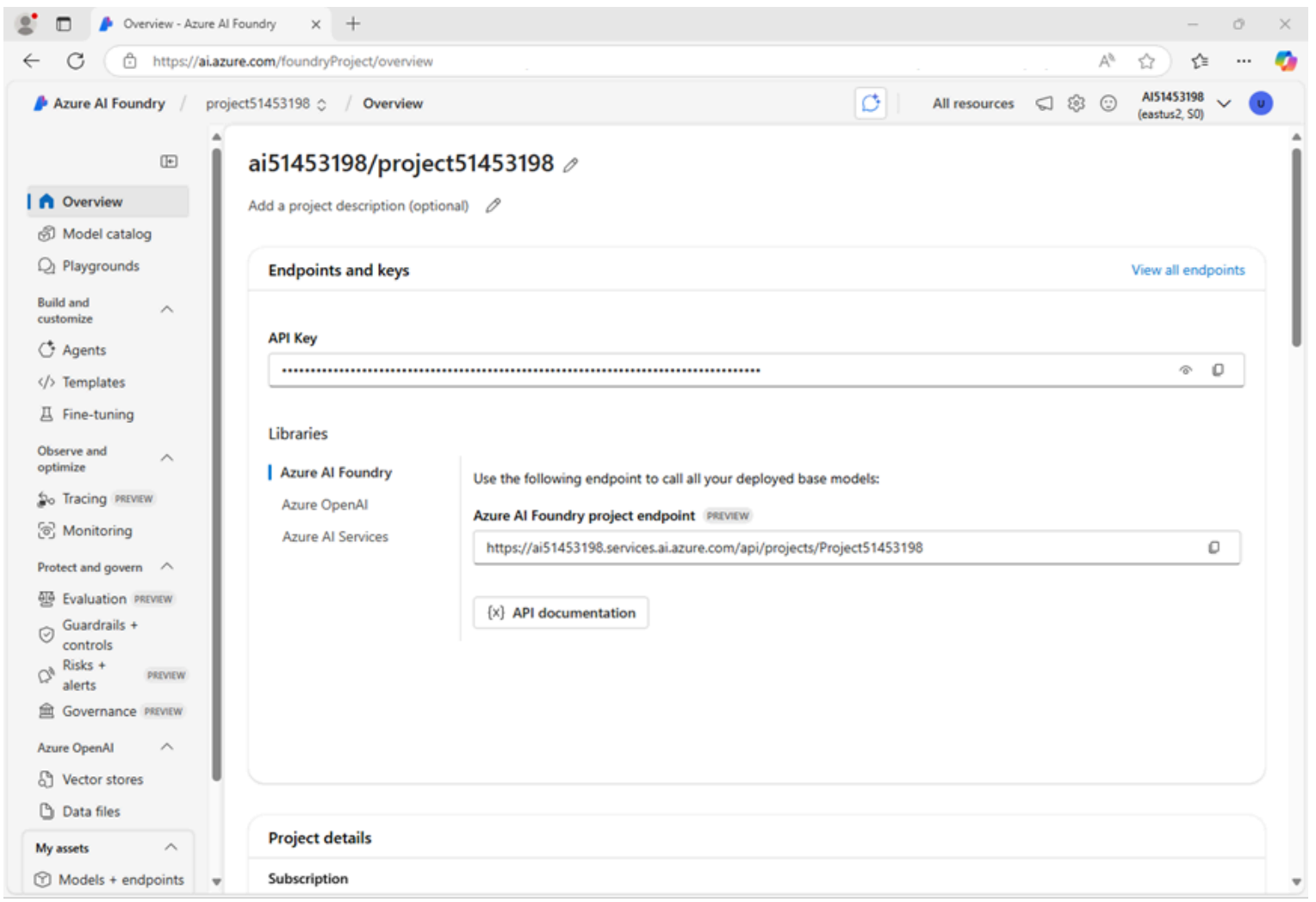



2. In the home page, in the **Explore models and capabilities** section, search for the `gpt-4o` model; which we'll use in our project.
3. In the search results, select the **gpt-4o** model to see its details, and then at the top of the page for the model, select **Use this model**.
4. When prompted to create a project, enter a valid name for your project and expand **Advanced options**.
5. Select **Customize** and specify the following settings for your project:
   - **Azure AI Foundry resource**: *A valid name for your Azure AI Foundry resource*
   - **Subscription**: *Your Azure subscription*
   - **Resource group**: *Create or select a resource group*
   - **Region**: *Select any* ***AI Services supported location****

> * Some Azure AI resources are constrained by regional model quotas. In the event of a quota limit being exceeded later in the exercise, there's a possibility you may need to create another resource in a different region.

6. Select **Create** and wait for your project, including the gpt-4 model deployment you selected, to be created.
7. When your project is created, the chat playground will be opened automatically.
8. In the **Setup** pane, note the name of your model deployment; which should be **gpt-4o**. You can confirm this by viewing the deployment in the **Models and endpoints** page (just open that page in the navigation pane on the left).

9. In the navigation pane on the left, select **Overview** to see the main page for your project; which looks like this:




## Create a client application to chat with the model

Now that you have deployed a model, you can use the Azure AI Foundry and Azure OpenAI SDKs to develop an application that chats with it.

> **Tip**: You can choose to develop your solution using Python or Microsoft C#. Follow the instructions in the appropriate section for your chosen language.

### Prepare the application configuration

1. In the Azure AI Foundry portal, view the **Overview** page for your project.
2. In the **Endpoints and keys** area, ensure that the **Azure AI Foundry** library is selected and view the **Azure AI Foundry project endpoint**. You'll use this endpoint to connect to your project and model in a client application.

   > **Note**: You can also use the Azure OpenAI endpoint!

3. Open a new browser tab (keeping the Azure AI Foundry portal open in the existing tab). Then in the new tab, browse to the Azure portal at `https://portal.azure.com`; signing in with your Azure credentials if prompted.

   Close any welcome notifications to see the Azure portal home page.

4. Use the **[>_]** button to the right of the search bar at the top of the page to create a new Cloud Shell in the Azure portal, selecting a ***PowerShell*** environment with no storage in your subscription.

The cloud shell provides a command-line interface in a pane at the bottom of the Azure portal. You can resize or maximize this pane to make it easier to work in.

> **Note**: If you have previously created a cloud shell that uses a *Bash* environment, switch it to ***PowerShell***.

5. In the cloud shell toolbar, in the **Settings** menu, select **Go to Classic version** (this is required to use the code editor).

   **Ensure you've switched to the classic version of the cloud shell before continuing.**

6. In the cloud shell pane, enter the following commands to clone the GitHub repo containing the code files for this exercise (type the command, or copy it to the clipboard and then right-click in the command line and paste as plain text):

   Code Copy

   ```
   rm -r mslearn-ai-foundry -f
   git clone https://github.com/microsoftlearning/mslearn-ai-studio mslearn-ai-foundry
   ```

   > **Tip**: As you enter commands into the cloudshell, the output may take up a large amount of the screen buffer. You can clear the screen by entering the `cls` command to make it easier to focus on each task.

7. After the repo has been cloned, navigate to the folder containing the chat application code files and view them:

   Use the commands below depending on your choice of programming language.

   **Python**

   Code Copy

   ```
   cd mslearn-ai-foundry/labfiles/chat-app/python
   ls -a -l
   ```

   **C#**

   Code Copy

   ```
   cd mslearn-ai-foundry/labfiles/chat-app/c-sharp
   ls -a -l
   ```

   The folder contains a code file as well as a configuration file for application settings and a file defining the project runtime and package requrirements.

8. In the cloud shell command-line pane, enter the following command to install the libraries you'll use:

   **Python**

   Code Copy

   ```
   python -m venv labenv
   ./labenv/bin/Activate.ps1
   pip install -r requirements.txt azure-identity azure-ai-projects openai
   ```

   **C#**

   Code Copy

```
dotnet add package Azure.Identity --prerelease
dotnet add package Azure.AI.Projects --prerelease
dotnet add package Azure.AI.OpenAI --prerelease
```

9. Enter the following command to edit the configuration file that has been provided:

**Python**

Code Copy

```
code .env
```

**C#**

Code Copy

```
code appsettings.json
```

The file is opened in a code editor.

10. In the code file, replace the **your_project_endpoint** placeholder with the **Azure AI Foundry project endpoint** for your project (copied from the **Overview** page in the Azure AI Foundry portal); and the **your_model_deployment** placeholder with the name of your gpt-4 model deployment.
11. After you've replaced the placeholders, within the code editor, use the **CTRL+S** command or **Right-click > Save** to save your changes and then use the **CTRL+Q** command or **Right-click > Quit** to close the code editor while keeping the cloud shell command line open.

## Write code to connect to your project and chat with your model

> **Tip**: As you add code, be sure to maintain the correct indentation.

1. Enter the following command to edit the code file that has been provided:

**Python**

Code Copy

```
code chat-app.py
```

**C#**

Code Copy

```
code Program.cs
```

2. In the code file, note the existing statements that have been added at the top of the file to import the necessary SDK namespaces. Then, find the comment **Add references**, and add the following code to reference the namespaces in the libraries you installed previously:

**Python**

Code Copy

```
# Add references
from azure.identity import DefaultAzureCredential
from azure.ai.projects import AIProjectClient
from openai import AzureOpenAI
```

**C#**

C# Copy

```
// Add references
using Azure.Identity;
using Azure.AI.Projects;
using Azure.AI.OpenAI;
using OpenAI.Chat;
```

3. In the **main** function, under the comment **Get configuration settings**, note that the code loads the project connection string and model deployment name values you defined in the configuration file.
4. Find the comment **Initialize the project client**, and add the following code to connect to your Azure AI Foundry project:

> **Tip**: Be careful to maintain the correct indentation level for your code.

**Python**

Code Copy

```
# Initialize the project client
project_client = AIProjectClient(
         credential=DefaultAzureCredential(
             exclude_environment_credential=True,
             exclude_managed_identity_credential=True
         ),
         endpoint=project_endpoint,
     )
```

**C#**

C# Copy

```
// Initialize the project client
DefaultAzureCredentialOptions options = new()
        { ExcludeEnvironmentCredential = true,
          ExcludeManagedIdentityCredential = true };
var projectClient = new AIProjectClient(
          new Uri(project_connection),
          new DefaultAzureCredential(options));
```

5. Find the comment **Get a chat client**, and add the following code to create a client object for chatting with a model:

**Python**

Code Copy

```
# Get a chat client
openai_client = project_client.inference.get_azure_openai_client(api_version="2024-10-21")
```

**C#**

C# Copy

```
// Get a chat client
ChatClient openaiClient = projectClient.GetAzureOpenAIChatClient(deployment:
model_deployment, connectionName: null, apiVersion: "2024-10-21");
```

6. Find the comment **Initialize prompt with system message**, and add the following code to initialize a collection of messages with a system prompt.

**Python**

Code Copy

```
# Initialize prompt with system message
prompt = [
        {"role": "system", "content": "You are a helpful AI assistant that answers
questions."}
    ]
```

**C#**

C# Copy

```
// Initialize prompt with system message
var prompt = new List<ChatMessage>(){
                new SystemChatMessage("You are a helpful AI assistant that answers
questions.")
            };
```

7. Note that the code includes a loop to allow a user to input a prompt until they enter “quit”. Then in the loop section, find the comment **Get a chat completion** and add the following code to add the user input to the prompt, retrieve the completion from your model, and add the completion to the prompt (so that you retain chat history for future iterations):

**Python**

Code Copy

```
# Get a chat completion
prompt.append({"role": "user", "content": input_text})
response = openai_client.chat.completions.create(
        model=model_deployment,
        messages=prompt)
completion = response.choices[0].message.content
print(completion)
prompt.append({"role": "assistant", "content": completion})
```

**C#**

C# Copy

```
// Get a chat completion
prompt.Add(new UserChatMessage(input_text));
ChatCompletion completion = openaiClient.CompleteChat(prompt);
var completionText = completion.Content[0].Text;
Console.WriteLine(completionText);
prompt.Add(new AssistantChatMessage(completionText));
```

8. Use the **CTRL+S** command to save your changes to the code file.

## Sign into Azure and run the app

1. In the cloud shell command-line pane, enter the following command to sign into Azure.

Code Copy

```
az login
```

**You must sign into Azure - even though the cloud shell session is already authenticated.**

> **Note**: In most scenarios, just using *az login* will be sufficient. However, if you have subscriptions in multiple tenants, you may need to specify the tenant by using the *–tenant* parameter. See Sign into Azure interactively using the Azure CLI for details.

2. When prompted, follow the instructions to open the sign-in page in a new tab and enter the authentication code provided and your Azure credentials. Then complete the sign in process in the command line, selecting the subscription containing your Azure AI Foundry hub if prompted.
3. After you have signed in, enter the following command to run the application:

**Python**

Code Copy

```
python chat-app.py
```

**C#**

Code Copy

```
dotnet run
```

> **Tip**: If a compilation error occurs because .NET version 9.0 is not installed, use the `dotnet --version` command to determine the version of .NET installed in your environment and then edit the **chat_app.csproj** file in the code folder to update the **TargetFramework** setting accordingly.

4. When prompted, enter a question, such as `What is the fastest animal on Earth?` and review the response from your generative AI model.
5. Try some follow-up questions, like `Where can I see one?` or `Are they endangered?`. The conversation should continue, using the chat history as context for each iteration.
6. When you're finished, enter `quit` to exit the program.

> **Tip**: If the app fails because the rate limit is exceeded. Wait a few seconds and try again. If there is insufficient quota available in your subscription, the model may not be able to respond.

## Summary

In this exercise, you used the Azure AI Foundry SDK to create a client application for a generative AI model that you deployed in an Azure AI Foundry project.

## Clean up

If you've finished exploring Azure AI Foundry portal, you should delete the resources you have created in this exercise to avoid incurring unnecessary Azure costs.

1. Open the Azure portal and view the contents of the resource group where you deployed the resources used in this exercise.
2. On the toolbar, select **Delete resource group**.
3. Enter the resource group name and confirm that you want to delete it.