Lesson 05 Demo 05

Using a Pretrained Model to Create an Application Structure

Objective: To create a pre-trained model for generating an application structure based on the type of application

Tools required: GitHub Copilot

Prerequisites: Basic knowledge of programming languages

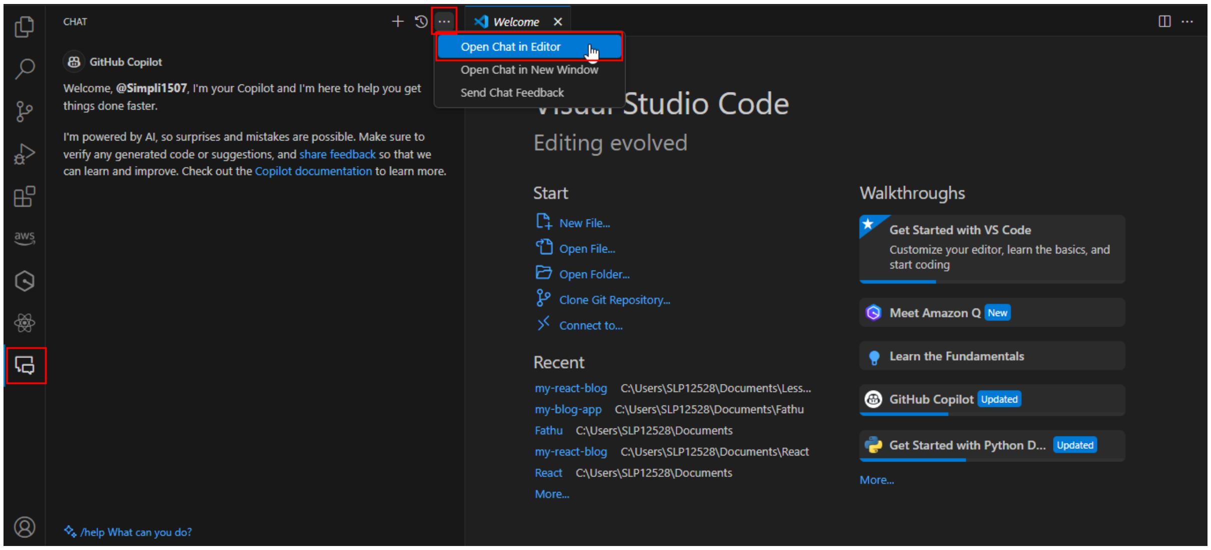
Steps to be followed:

1. Create a pretrained model using GitHub Copilot
2. Run the code using GitHub Copilot

Note: Please be advised that GitHub Copilot is an artificial intelligence tool that can produce varied outputs even when presented with similar prompts. Thus, you may get different code snippets for the same prompt.

Step 1: Create a pretrained model using GitHub Copilot

1.1 Click on the **Chat** icon, click on the **three dots** icon, and select the **Open Chat in Editor** option



GitHub Copilot

Welcome, **@Simpli1507**, I'm your Copilot and I'm here to help you get things done faster.

I'm powered by AI, so surprises and mistakes are possible. Make sure to verify any generated code or suggestions, and share feedback so that we can learn and improve. Check out the Copilot documentation to learn more.

1.2 Use the following prompt to create a pretrained model class **MyStructureGenModel** that generates an application structure:

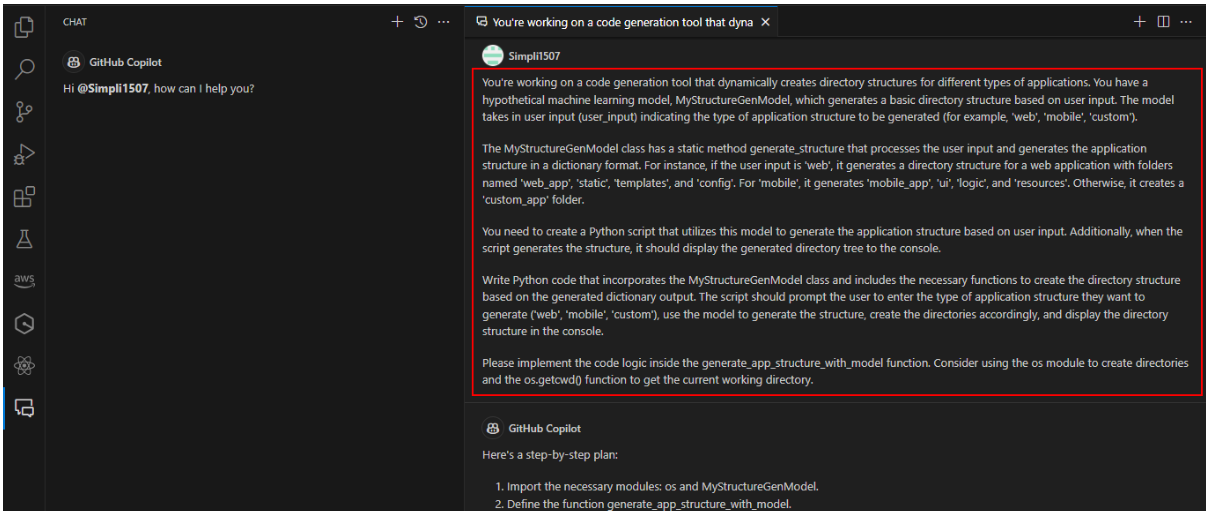
You're working on a code generation tool that dynamically creates directory structures for different types of applications. You have a hypothetical machine learning model, MyStructureGenModel, which generates a basic directory structure based on user input. The model takes in user input (user_input) indicating the type of application structure to be generated (for example, 'web', 'mobile', 'custom').

The MyStructureGenModel class has a static method generate_structure that processes the user input and generates the application structure in a dictionary format. For instance, if the user input is 'web', it generates a directory structure for a web application with folders named 'web_app', 'static', 'templates', and 'config'. For 'mobile', it generates 'mobile_app', 'ui', 'logic', and 'resources'. Otherwise, it creates a 'custom_app' folder.

You need to create a Python script that utilizes this model to generate the application structure based on user input. Additionally, when the script generates the structure, it should display the generated directory tree to the console.

Write Python code that incorporates the MyStructureGenModel class and includes the necessary functions to create the directory structure based on the generated dictionary output. The script should prompt the user to enter the type of application structure they want to generate ('web', 'mobile', 'custom'), use the model to generate the structure, create the directories accordingly, and display the directory structure in the console.

Please implement the code logic inside the generate_app_structure_with_model function. Consider using the os module to create directories and the os.getcwd() function to get the current working directory.



GitHub Copilot

Hi **@Simpli1507**, how can I help you?

Simpli1507

You're working on a code generation tool that dynamically creates directory structures for different types of applications. You have a hypothetical machine learning model, MyStructureGenModel, which generates a basic directory structure based on user input. The model takes in user input (user_input) indicating the type of application structure to be generated (for example, 'web', 'mobile', 'custom').

The MyStructureGenModel class has a static method generate_structure that processes the user input and generates the application structure in a dictionary format. For instance, if the user input is 'web', it generates a directory structure for a web application with folders named 'web_app', 'static', 'templates', and 'config'. For 'mobile', it generates 'mobile_app', 'ui', 'logic', and 'resources'. Otherwise, it creates a 'custom_app' folder.

You need to create a Python script that utilizes this model to generate the application structure based on user input. Additionally, when the script generates the structure, it should display the generated directory tree to the console.

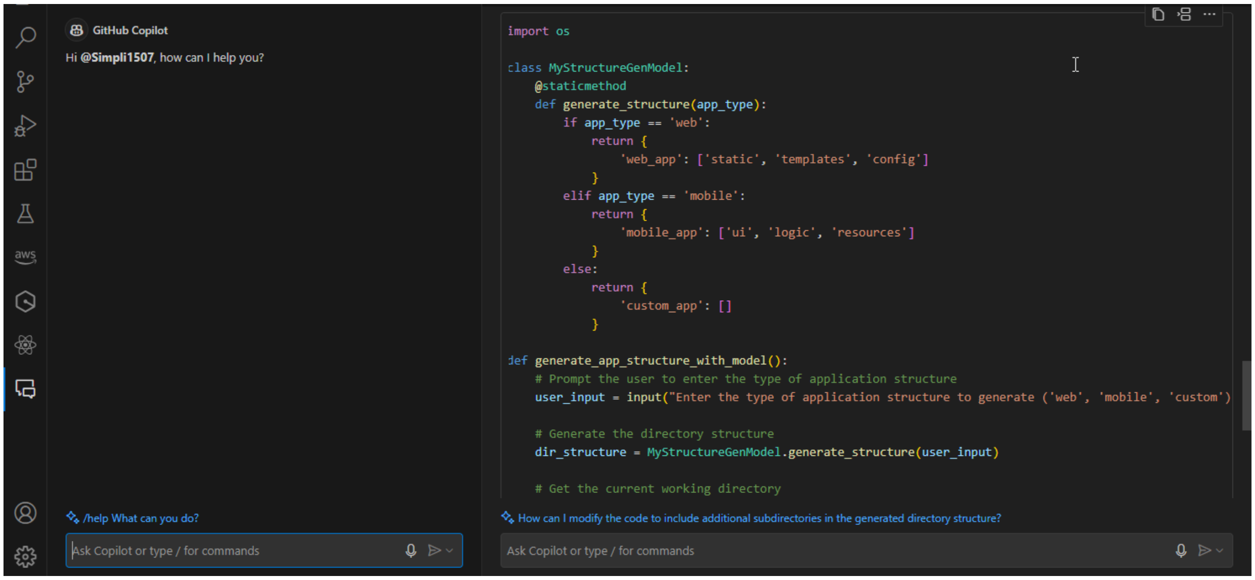
Write Python code that incorporates the MyStructureGenModel class and includes the necessary functions to create the directory structure based on the generated dictionary output. The script should prompt the user to enter the type of application structure they want to generate ('web', 'mobile', 'custom'), use the model to generate the structure, create the directories accordingly, and display the directory structure in the console.

Please implement the code logic inside the generate_app_structure_with_model function. Consider using the os module to create directories and the os.getcwd() function to get the current working directory.

GitHub Copilot

Here's a step-by-step plan:

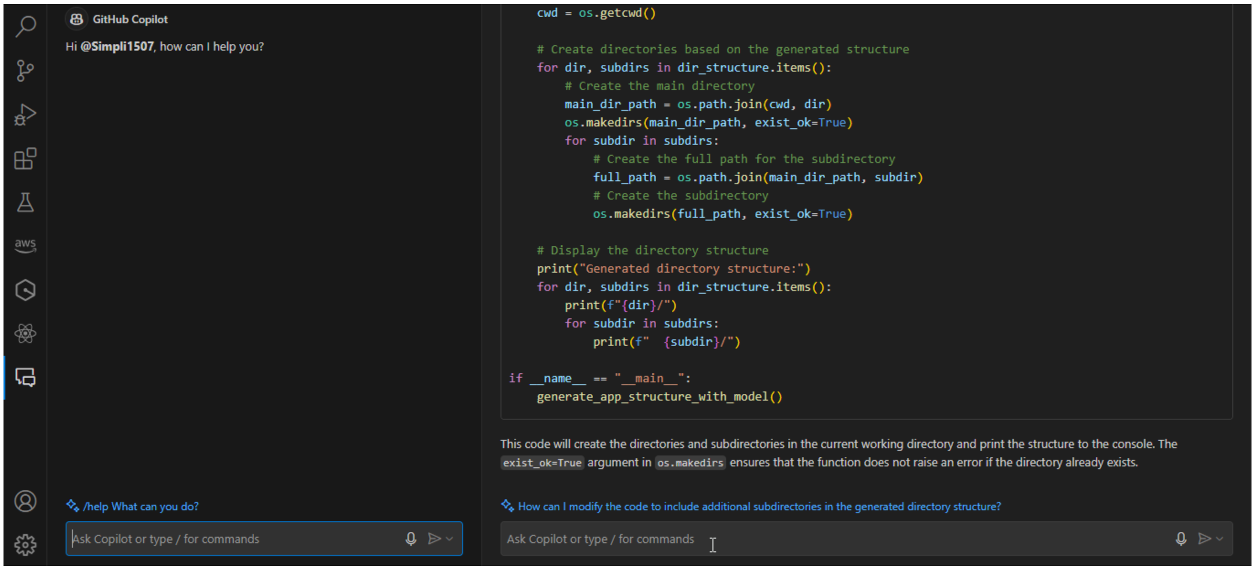
1. Import the necessary modules: os and MyStructureGenModel.
2. Define the function generate_app_structure_with_model.



Hi **@Simpli1507**, how can I help you?

```
import os

class MyStructureGenModel:
    @staticmethod
    def generate_structure(app_type):
        if app_type == 'web':
            return {
                'web_app': ['static', 'templates', 'config']
            }
        elif app_type == 'mobile':
            return {
                'mobile_app': ['ui', 'logic', 'resources']
            }
        else:
            return {
                'custom_app': []
            }

def generate_app_structure_with_model():
    # Prompt the user to enter the type of application structure
    user_input = input("Enter the type of application structure to generate ('web', 'mobile', 'custom')

    # Generate the directory structure
    dir_structure = MyStructureGenModel.generate_structure(user_input)

    # Get the current working directory
```

/help What can you do?
How can I modify the code to include additional subdirectories in the generated directory structure?
Ask Copilot or type / for commands
Ask Copilot or type / for commands



Hi **@Simpli1507**, how can I help you?

```
    cwd = os.getcwd()

    # Create directories based on the generated structure
    for dir, subdirs in dir_structure.items():
        # Create the main directory
        main_dir_path = os.path.join(cwd, dir)
        os.makedirs(main_dir_path, exist_ok=True)
        for subdir in subdirs:
            # Create the full path for the subdirectory
            full_path = os.path.join(main_dir_path, subdir)
            # Create the subdirectory
            os.makedirs(full_path, exist_ok=True)

    # Display the directory structure
    print("Generated directory structure:")
    for dir, subdirs in dir_structure.items():
        print(f"{dir}/")
        for subdir in subdirs:
            print(f"  {subdir}/")

if __name__ == "__main__":
    generate_app_structure_with_model()
```

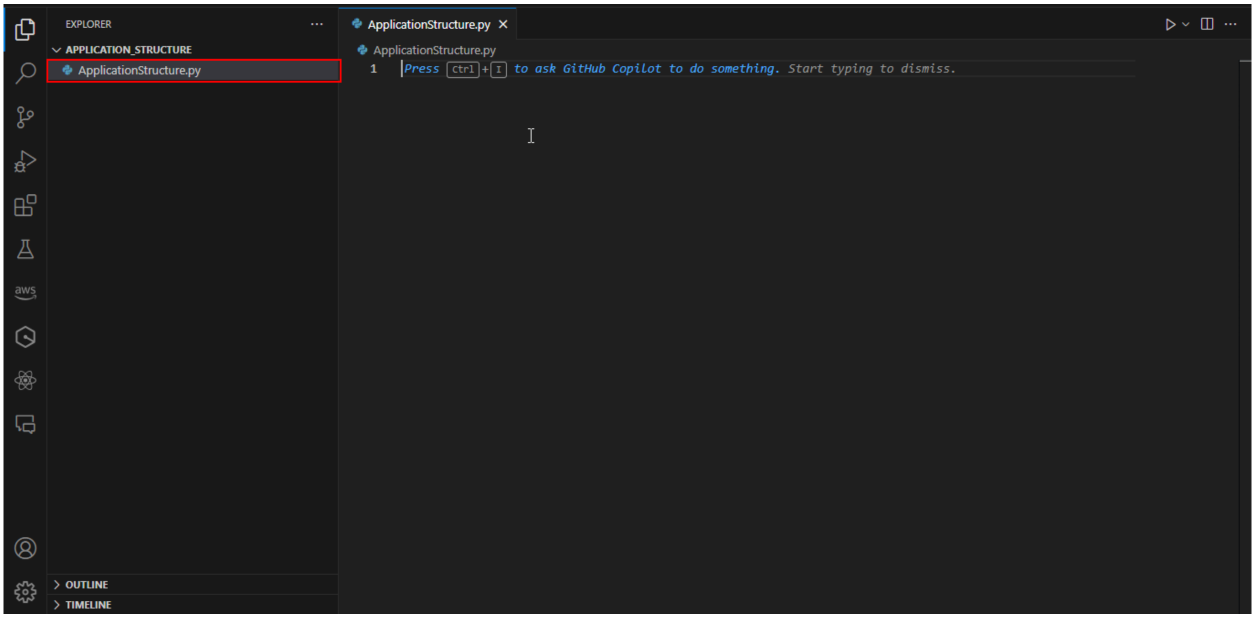
This code will create the directories and subdirectories in the current working directory and print the structure to the console. The `exist_ok=True` argument in `os.makedirs` ensures that the function does not raise an error if the directory already exists.

/help What can you do?
How can I modify the code to include additional subdirectories in the generated directory structure?
Ask Copilot or type / for commands
Ask Copilot or type / for commands

Note: Copy the generated code and save it in the Notepad for the next steps

Step 2: Run the code using GitHub Copilot

2.1 Create a new file named **ApplicationStructure.py**, as shown in the screenshot below:

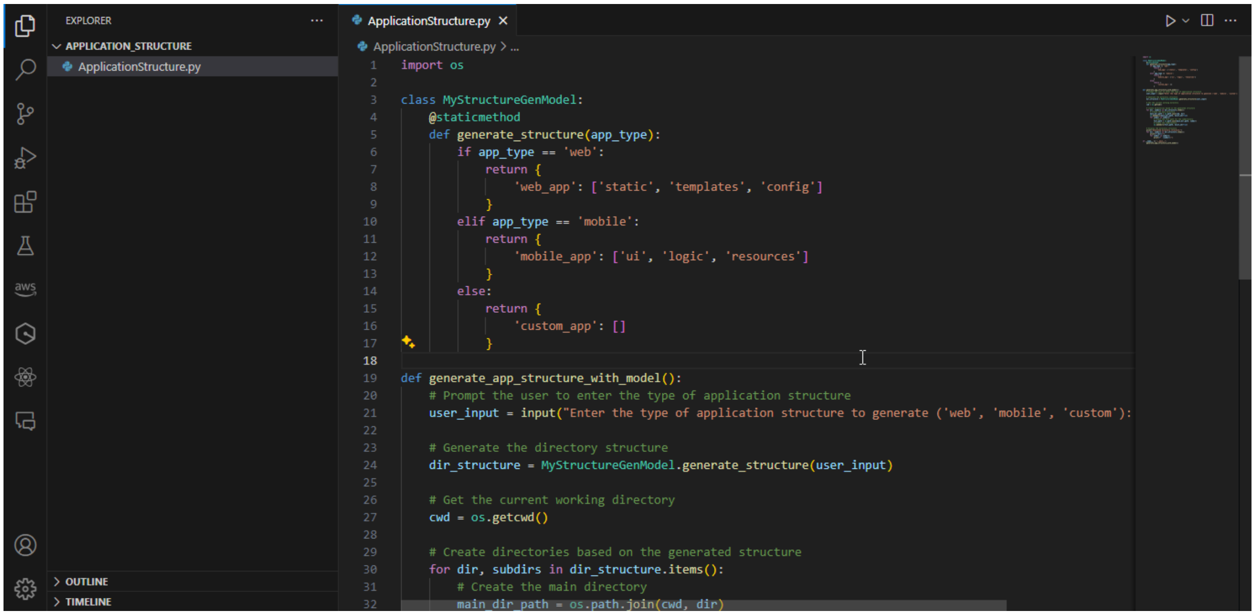


Press ctrl+I to ask GitHub Copilot to do something. Start typing to dismiss.

2.2 Paste the following generated code in the **ApplicationStructure.py** file, as shown in the screenshot below:

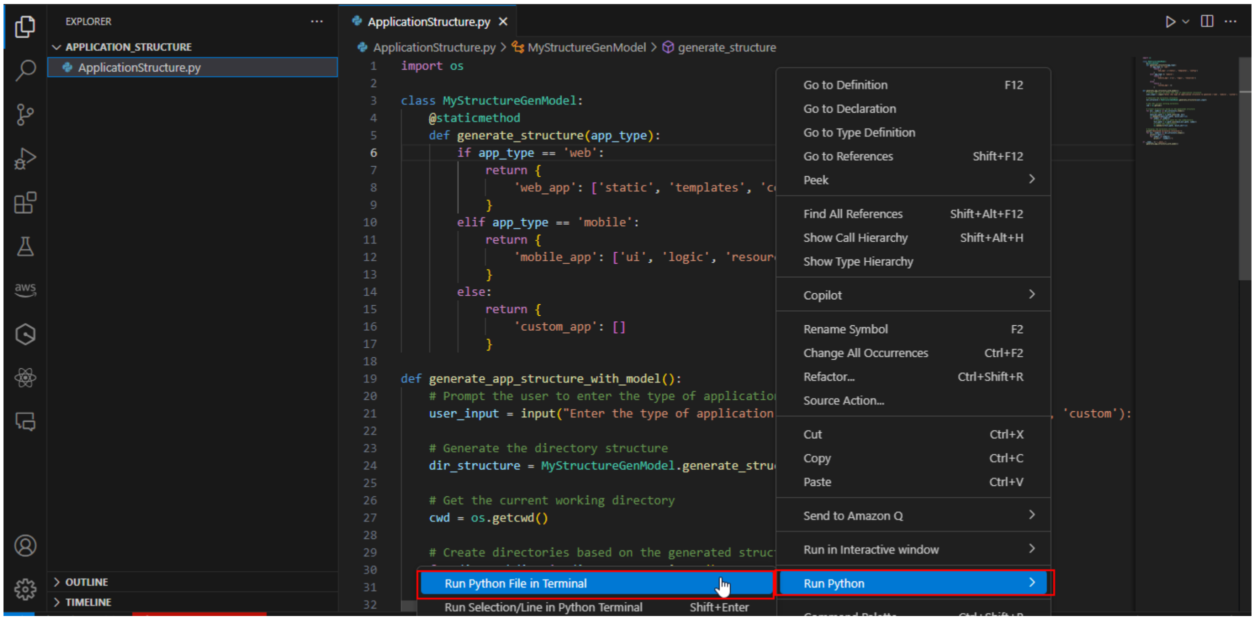
```
import os

class MyStructureGenModel:
    @staticmethod
    def generate_structure(app_type):
        if app_type == 'web':
            return {
                'web_app': ['static', 'templates', 'config']
            }
        elif app_type == 'mobile':
            return {
                'mobile_app': ['ui', 'logic', 'resources']
            }
        else:
            return {
                'custom_app': []
            }
```

```python
def generate_app_structure_with_model():
    # Prompt the user to enter the type of application structure
    user_input = input("Enter the type of application structure to generate ('web', 'mobile', 'custom'): ").strip().lower()

    # Generate the directory structure
    dir_structure = MyStructureGenModel.generate_structure(user_input)

    # Get the current working directory
    cwd = os.getcwd()

    # Create directories based on the generated structure
    for dir, subdirs in dir_structure.items():
        # Create the main directory
        main_dir_path = os.path.join(cwd, dir)
        os.makedirs(main_dir_path, exist_ok=True)
        for subdir in subdirs:
            # Create the full path for the subdirectory
            full_path = os.path.join(main_dir_path, subdir)
            # Create the subdirectory
            os.makedirs(full_path, exist_ok=True)

    # Display the directory structure
    print("Generated directory structure:")
    for dir, subdirs in dir_structure.items():
        print(f"{dir}/")
        for subdir in subdirs:
            print(f"  {subdir}/")

if __name__ == "__main__":
    generate_app_structure_with_model()
```



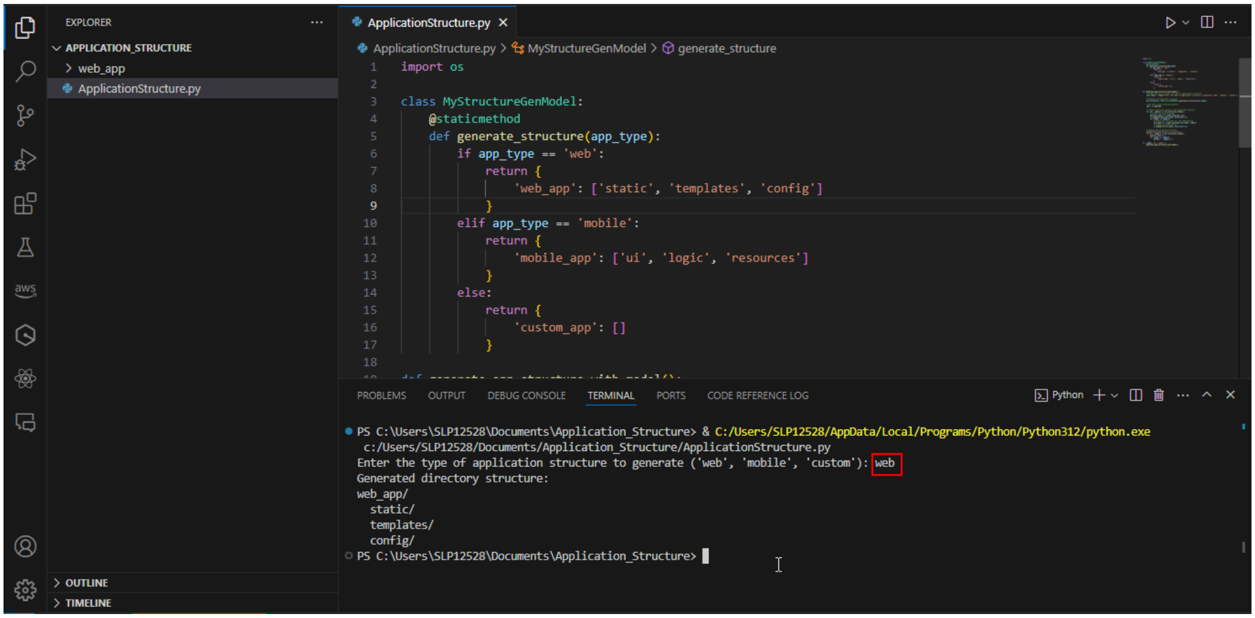
```python
import os

class MyStructureGenModel:
    @staticmethod
    def generate_structure(app_type):
        if app_type == 'web':
            return {
                'web_app': ['static', 'templates', 'config']
            }
        elif app_type == 'mobile':
            return {
                'mobile_app': ['ui', 'logic', 'resources']
            }
        else:
            return {
                'custom_app': []
            }

def generate_app_structure_with_model():
    # Prompt the user to enter the type of application structure
    user_input = input("Enter the type of application structure to generate ('web', 'mobile', 'custom'):

    # Generate the directory structure
    dir_structure = MyStructureGenModel.generate_structure(user_input)

    # Get the current working directory
    cwd = os.getcwd()

    # Create directories based on the generated structure
    for dir, subdirs in dir_structure.items():
        # Create the main directory
        main_dir_path = os.path.join(cwd, dir)
```

2.3 Save the file, right-click on the editor, click on **Run Python**, and then click on **Run Python File in Terminal**, as shown in the screenshot below:



```python
import os

class MyStructureGenModel:
    @staticmethod
    def generate_structure(app_type):
        if app_type == 'web':
            return {
                'web_app': ['static', 'templates', 'c
            }
        elif app_type == 'mobile':
            return {
                'mobile_app': ['ui', 'logic', 'resour
            }
        else:
            return {
                'custom_app': []
            }

def generate_app_structure_with_model():
    # Prompt the user to enter the type of applicatio
    user_input = input("Enter the type of application                                         , 'custom'):

    # Generate the directory structure
    dir_structure = MyStructureGenModel.generate_stru

    # Get the current working directory
    cwd = os.getcwd()

    # Create directories based on the generated struc
```

2.4 Enter **web** to specify the type of application structure you want to generate, as shown in the screenshot below:

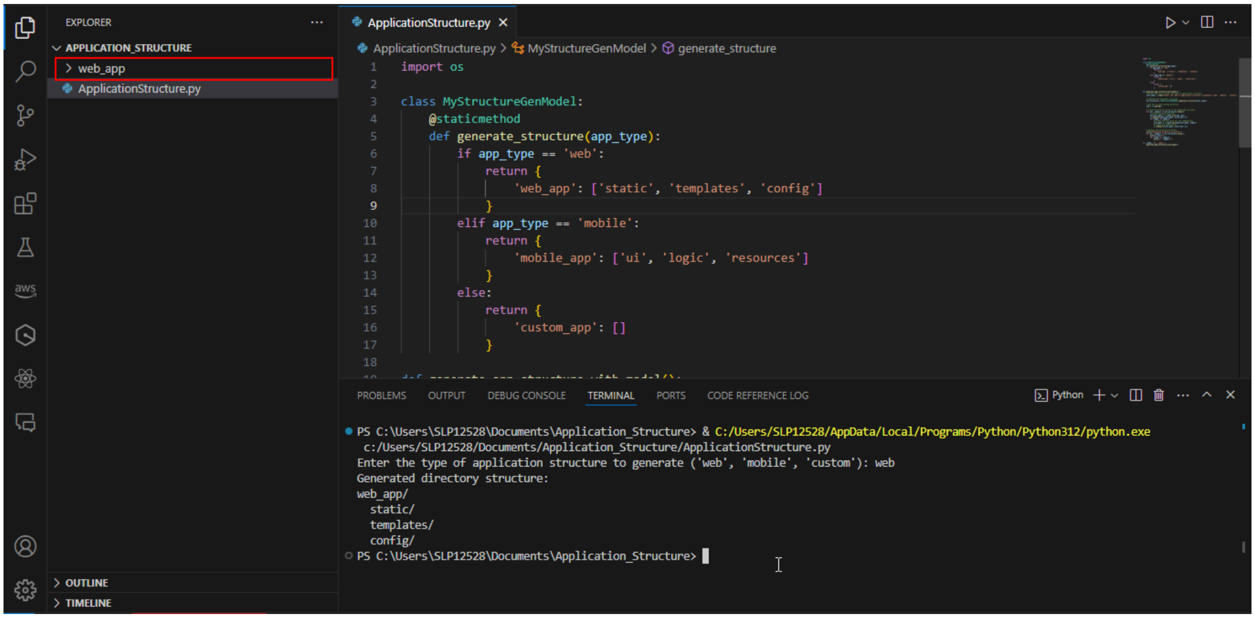


```python
import os

class MyStructureGenModel:
    @staticmethod
    def generate_structure(app_type):
        if app_type == 'web':
            return {
                'web_app': ['static', 'templates', 'config']
            }
        elif app_type == 'mobile':
            return {
                'mobile_app': ['ui', 'logic', 'resources']
            }
        else:
            return {
                'custom_app': []
            }

```

```
PS C:\Users\SLP12528\Documents\Application_Structure> & C:/Users/SLP12528/AppData/Local/Programs/Python/Python312/python.exe
 c:/Users/SLP12528/Documents/Application_Structure/ApplicationStructure.py
Enter the type of application structure to generate ('web', 'mobile', 'custom'): web
Generated directory structure:
web_app/
  static/
  templates/
  config/
PS C:\Users\SLP12528\Documents\Application_Structure>
```

The application structure will be generated in the folder list.

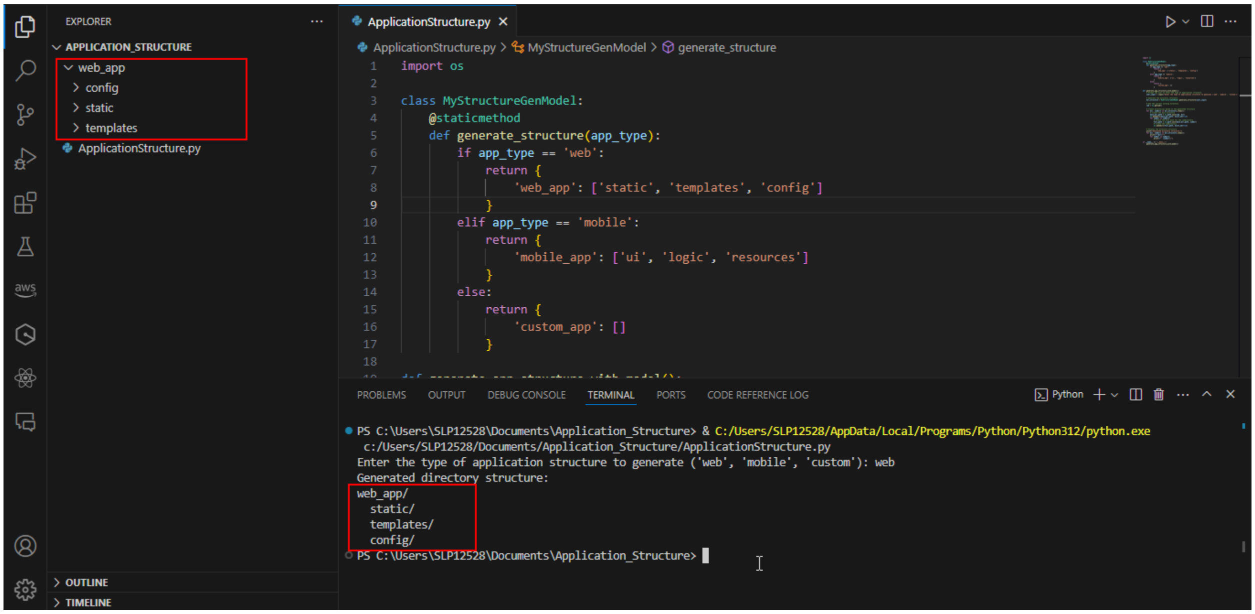
2.5 Click on the **web_app** to view the application structure, as shown in the screenshot below:



```python
import os

class MyStructureGenModel:
    @staticmethod
    def generate_structure(app_type):
        if app_type == 'web':
            return {
                'web_app': ['static', 'templates', 'config']
            }
        elif app_type == 'mobile':
            return {
                'mobile_app': ['ui', 'logic', 'resources']
            }
        else:
            return {
                'custom_app': []
            }

```

```
PS C:\Users\SLP12528\Documents\Application_Structure> & C:/Users/SLP12528/AppData/Local/Programs/Python/Python312/python.exe
 c:/Users/SLP12528/Documents/Application_Structure/ApplicationStructure.py
Enter the type of application structure to generate ('web', 'mobile', 'custom'): web
Generated directory structure:
web_app/
  static/
  templates/
  config/
PS C:\Users\SLP12528\Documents\Application_Structure>
```

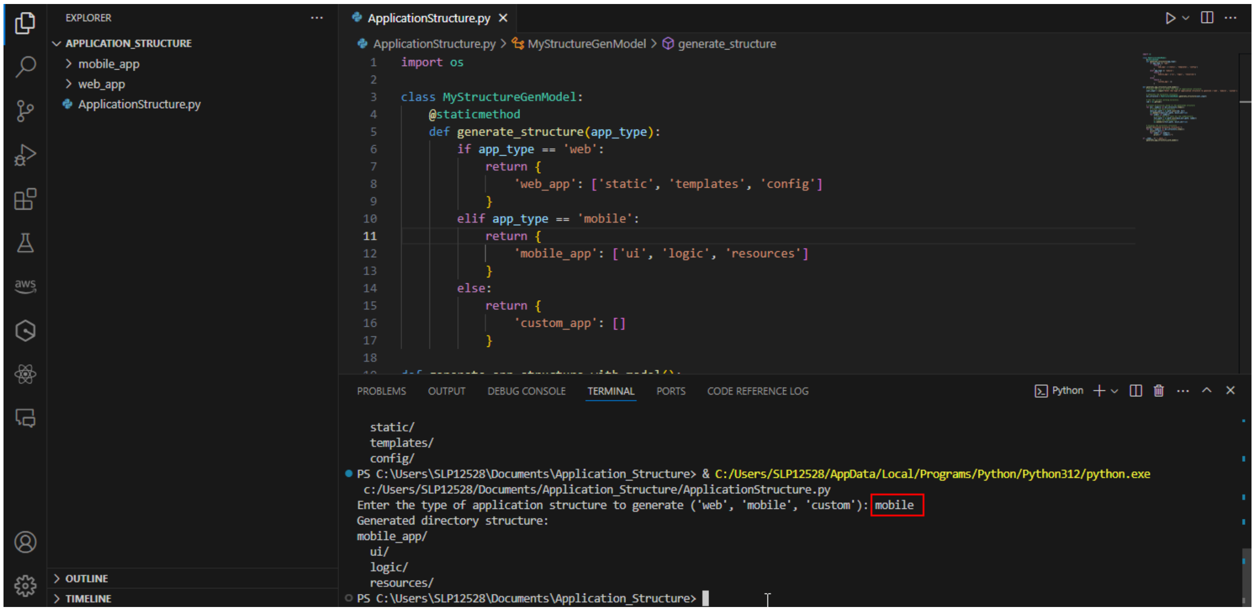
You will see the following interface:



```python
import os

class MyStructureGenModel:
    @staticmethod
    def generate_structure(app_type):
        if app_type == 'web':
            return {
                'web_app': ['static', 'templates', 'config']
            }
        elif app_type == 'mobile':
            return {
                'mobile_app': ['ui', 'logic', 'resources']
            }
        else:
            return {
                'custom_app': []
            }

```

```
PS C:\Users\SLP12528\Documents\Application_Structure> & C:/Users/SLP12528/AppData/Local/Programs/Python/Python312/python.exe
 c:/Users/SLP12528/Documents/Application_Structure/ApplicationStructure.py
Enter the type of application structure to generate ('web', 'mobile', 'custom'): web
Generated directory structure:
web_app/
  static/
  templates/
  config/
PS C:\Users\SLP12528\Documents\Application_Structure> 
```

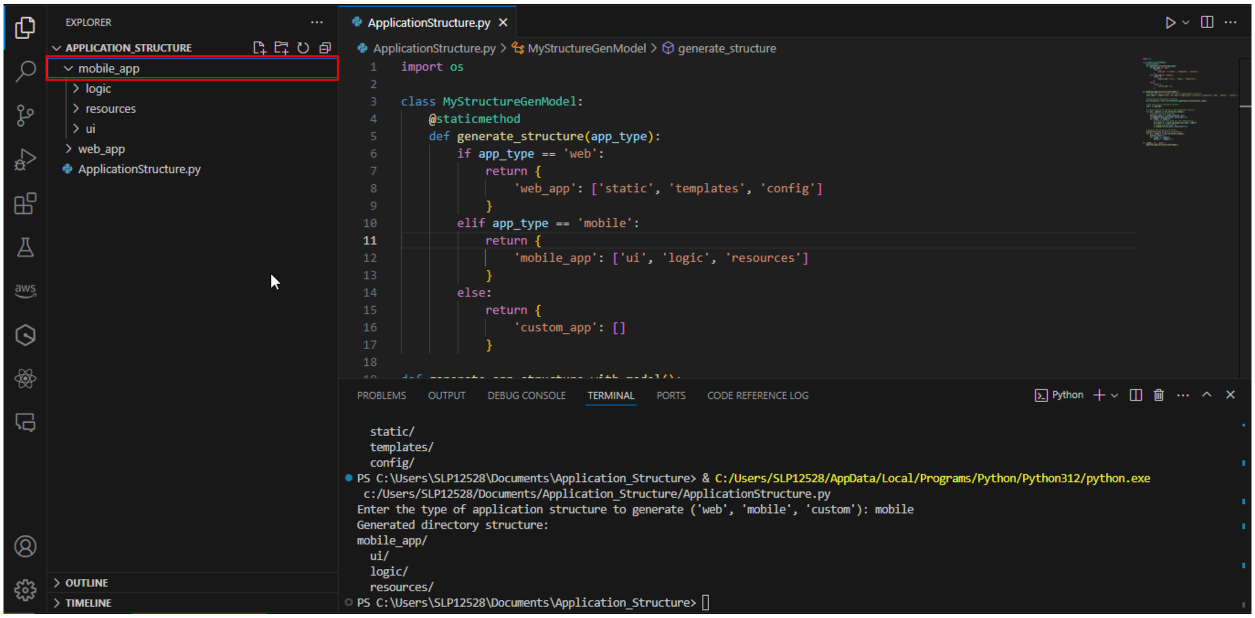
2.6 Execute the **ApplicationStructure.py file** and input **mobile** to select the type of application structure you wish to generate, as depicted in the screenshot below:



```python
import os

class MyStructureGenModel:
    @staticmethod
    def generate_structure(app_type):
        if app_type == 'web':
            return {
                'web_app': ['static', 'templates', 'config']
            }
        elif app_type == 'mobile':
            return {
                'mobile_app': ['ui', 'logic', 'resources']
            }
        else:
            return {
                'custom_app': []
            }

```

```
  static/
  templates/
  config/
PS C:\Users\SLP12528\Documents\Application_Structure> & C:/Users/SLP12528/AppData/Local/Programs/Python/Python312/python.exe
 c:/Users/SLP12528/Documents/Application_Structure/ApplicationStructure.py
Enter the type of application structure to generate ('web', 'mobile', 'custom'): mobile
Generated directory structure:
mobile_app/
  ui/
  logic/
  resources/
PS C:\Users\SLP12528\Documents\Application_Structure> 
```

2.7 Click on **mobile_app** to view the application structure, as shown in the screenshot below:



```python
import os

class MyStructureGenModel:
    @staticmethod
    def generate_structure(app_type):
        if app_type == 'web':
            return {
                'web_app': ['static', 'templates', 'config']
            }
        elif app_type == 'mobile':
            return {
                'mobile_app': ['ui', 'logic', 'resources']
            }
        else:
            return {
                'custom_app': []
            }

```

```
  static/
  templates/
  config/
PS C:\Users\SLP12528\Documents\Application_Structure> & C:/Users/SLP12528/AppData/Local/Programs/Python/Python312/python.exe
 c:/Users/SLP12528/Documents/Application_Structure/ApplicationStructure.py
Enter the type of application structure to generate ('web', 'mobile', 'custom'): mobile
Generated directory structure:
mobile_app/
  ui/
  logic/
  resources/
PS C:\Users\SLP12528\Documents\Application_Structure> 
```

By following these steps, you have successfully created a pretrained model for generating an application structure based on the type of application.