GitHub Copilot, Flask, OpenAPI and Swagger Editor

Prerequisites

* Swagger Editor: <https://editor.swagger.io>
  + Just make sure you can access this tool, this is a workflow step in the process that converts APIFlask generated JSON to YAML. Yaml is consumed by the OpenAPI VS Code extensions
* Python/Pip
  + $ pip install apiflask
* API Flask
  + Website: <https://apiflask.com>
  + Documentation: <https://apiflask.com/docs>
  + PyPI Releases: <https://pypi.python.org/pypi/APIFlask>

API Flask is a lightweight Python web API framework based on [Flask](https://github.com/pallets/flask) and [marshmallow-code](https://github.com/marshmallow-code) projects. It's easy to use, highly customizable, ORM/ODM-agnostic, and 100% compatible with the Flask ecosystem.

* + With APIFlask, you will have:
    - More sugars for view function (@app.input(), @app.output(), @app.get(), @app.post() and more)
    - Automatic request validation and deserialization
    - Automatic response formatting and serialization
    - Automatic [OpenAPI Specification](https://github.com/OAI/OpenAPI-Specification) (OAS, formerly Swagger Specification) document generation
    - Automatic interactive API documentation
    - API authentication support (with [Flask-HTTPAuth](https://github.com/miguelgrinberg/flask-httpauth))
    - Automatic JSON response for HTTP errors
* VSCode: <https://code.visualstudio.com/download>
* Install these VS Code extensions as they will be used in this lab:  
  + Python VS Code ExtensionA screenshot of a computer

    Description automatically generated
  + OpenAPI VS Code Extension

A screenshot of a computer

Description automatically generated

## Requirements

* Python 3.8+
* Flask 2.0+

## Use Prompting to Get Started

1. Create a new file called **app.py** in VS Code
2. In your **app.py** file, ask Github Copilot to generate an API Flask app and add a “GET” method. Do this by typing “#” followed by the prompt you wish to pass to Copilot. There is an example provided below, but feel free to experiment with your own.

|  |
| --- |
| # Generate APIFlask app  # Add a GET method called “SayHello” that accepts input as a query string using a string as a parameter for the name to say a greeting |

1. Press **CTRL – [Return].** This should open a Copilot editor to the right that provides Github Copilot suggestions for app.py.  
     
   A screenshot of a computer screen

   Description automatically generated

1. Select the code solution of your choice from the list of suggestions by clicking “**Accept Solution**”. This should add the suggested code to your app.py file.
2. After you have used Copilot to generate the Imports and application Object, prompt it for Schema Objects. Type **“# Schema for Input**” followed by “**Return**” and Copilot should suggest a class for you. Hit “**Tab**” to accept the suggestion and autocomplete the schema.

|  |
| --- |
| # Schema for input  # Schema for output |

1. From here it will generate the Input and Output Schema objects and look something like this:

A computer screen with text

Description automatically generated

1. Add another prompt for an entry point (below) and repeat the process of clicking “**Return**” > “**Tab**” to autocomplete

|  |
| --- |
| # GET method – Use Query string parameters |

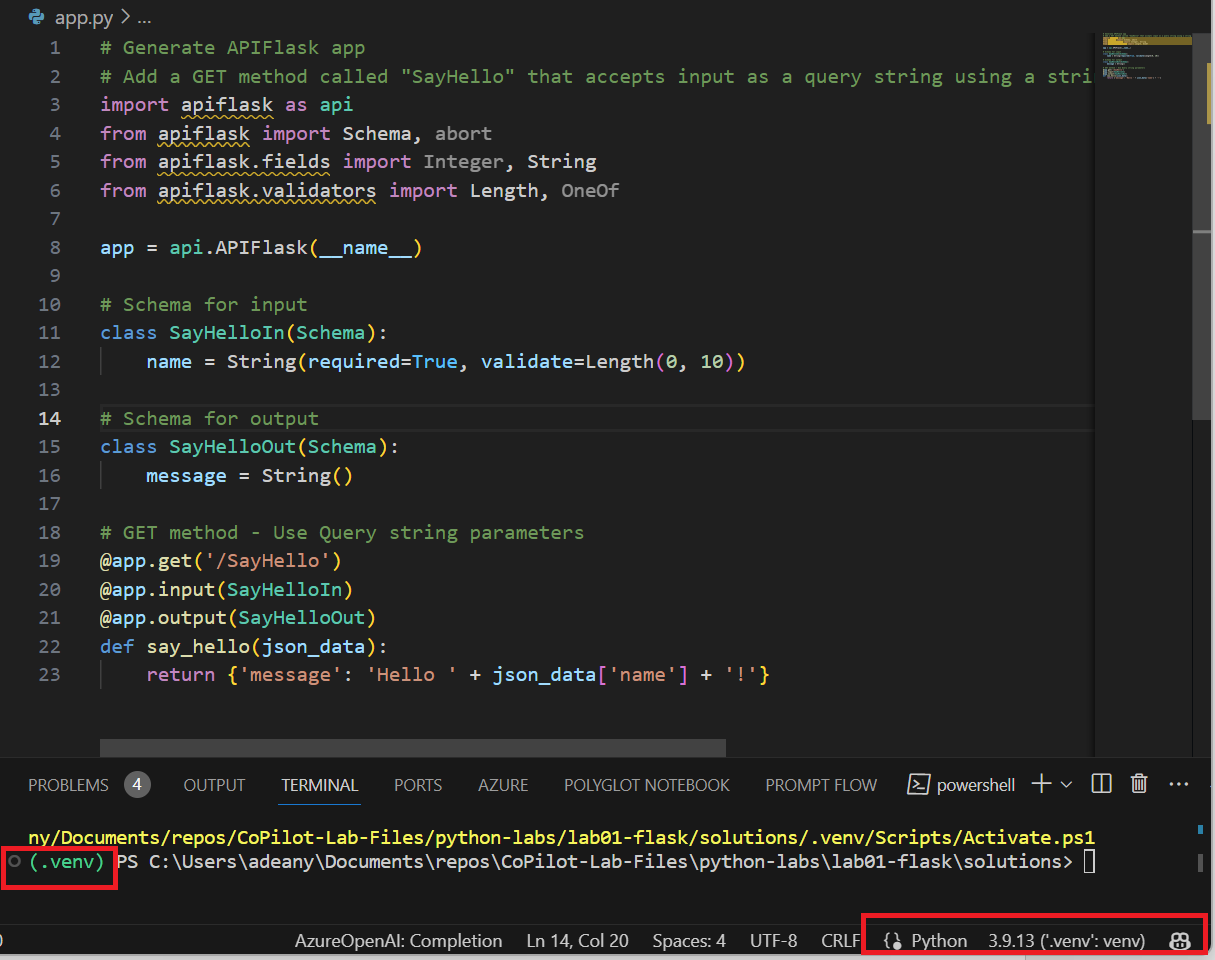
1. You should have something that resembles:  
     
   A screen shot of a computer program

   Description automatically generated
2. This should match your virtual environment. Save your progress: **[CTRL] + s or [CMD] + s**

## Running Your Application

1. Set up your virtual environment. In your project’s folder, use VSCode to create a new environment (**CTRL + Shift +P: Python: Create Environment**). The command presents a list of two types: select **Venv**.  
     
   A close-up of a black and white sign

   Description automatically generated
2. You will be prompted to select an interpreter to be used as a base for the new virtual environment. Select the desired interpreter.   
     
   A screenshot of a computer program

   Description automatically generated
3. Open a terminal the selected interpreter should show on the right side of the status bar.  
     
   
4. In your venv, install requirements (**pip install apiflask**).
5. From your **app.py** file, click the Debug Icon  > **Debug Python File.** A “Run and Debug” pane should open on the left side that looks like this:  
      
    A screenshot of a computer

   Description automatically generated
6. Click “**Create a launch.json file**”. VSCode will open a new file under .vscode/launch.json.
7. Choose **“Add Configuration”**

A screenshot of a computer program

Description automatically generated

1. Add this configuration to your **.vscode/launch.json**

|  |
| --- |
| {  "name": "Python: Flask",  "type": "python",  "request": "launch",  "module": "flask",  "env": {  "FLASK\_APP": "app.py",  "FLASK\_DEBUG": "1"  },  "args": [  "run",  "--no-debugger",  "--no-reload"  ],  "jinja": true,  "justMyCode": true  } |

1. Save changes. Open the file: **app.py**
2. To enter a breakpoint, click in the margin, left of the line numbers. It will look like this:  
     
   A screen shot of a computer program

   Description automatically generated
3. From the VS Code “Run and Debug” Menu, click the **Start Debugging** icon**.  
     
   A black screen with white text and red box

   Description automatically generated**
4. You should see your app is now running on http://127.0.0.1:5000.A computer screen shot of text

   Description automatically generated
5. Bring up a browser: <http://127.0.0.1:5000/SayHello?name=John>
6. You will see an error handler response that looks like this:  
     
   A screenshot of a computer error

   Description automatically generated

## Debugging Generated Code

1. We need to figure out why we are getting a Validation error. Let us take advantage of OpenAPI and use the executable Documentation found here: <http://127.0.0.1:5000/docs>  
     
   A screenshot of a computer

   Description automatically generated
2. The problem is that the data is in the request body (which you can’t do with a GET) and is not being passed as query parameters.  
     
     
   A screenshot of a computer

   Description automatically generated
3. We will need to modify the generated code. We can reference the APIFlask documentation to see what changes we need to make: <https://apiflask.com/request/>
4. From the documentation, we see that we must change from the body to query string. After making some modifications, our new GET method should look something like this:  
     
   A screen shot of a computer code

   Description automatically generated
5. Run your app again. Bring up your browser to confirm that your changes worked and the app is running successfully: <http://127.0.0.1:5000/SayHello?name=John>.  
     
   A screenshot of a computer

   Description automatically generated
6. Add a prompt to add one more method to add 2 numbers and return the sum. Repeat the process of clicking “Return” > “Tab” to autocomplete

|  |
| --- |
| # GET method – Add two query string parameters and return the sum |

1. CoPilot learned from the last method and from appearances did this method properly. Your new method should look something like this:

A screen shot of a computer program

Description automatically generated

1. Let’s go back to our OpenAPI documentation to test this: <http://127.0.0.1:5000/docs>. Click “**Try It Out**” and enter two number inputs. We see this works!  
     
   A screenshot of a computer

   Description automatically generated

## Pulling Existing OpenAPI Docs into VSCode

1. So far, this has been a Code First API approach, now we will pull our design documentation into VS Code. Navigate to <http://127.0.0.1:5000/docs> and click the Blue **“/openapi.json**” link.

A close up of a logo

Description automatically generated

1. You will see the openapi.json file in the browser. **Copy the JSON**.
2. Open a new tab in your browser and open the Swagger URL: <https://editor.swagger.io>. From the menu: **File | clear editor**A screenshot of a computer

   Description automatically generated
3. Next paste the JSON into the editor, then you will see:  
     
   A screenshot of a computer

   Description automatically generated
4. Select **OK**. You will see:  
     
   A screenshot of a computer

   Description automatically generated
5. Navigate back to VS Code. The VS Code OpenAPI extension works from the command palette, and all commands are prefixed with: OpenAPI. Click **CTRL + Shift + P** > **OpenAPI: Create new OpenAPI file**  
     
   A screenshot of a computer

   Description automatically generated
6. **Paste** in our API specification in JSON and **Save**.  
     
   A screenshot of a computer

   Description automatically generated
7. We will want to preview our OpenAPI SwaggerUI. Click **CTRL + Shift + P** > **OpenAPI: Show preview using the default renderer.**  
     
   A screenshot of a computer program

   Description automatically generated
8. You will now see an OpenAPI SwaggerUI in your VS Code! (Note: To make the Swagger/OpenAPI editor work, you may need to run the Flask API in another terminal and set your default directory where your python/flask app is located)

A screenshot of a computer

Description automatically generated