Lab #3: Introduction to APIs and Flask

COMP 395 – AI and Learning Technologies

Week 3

Today's Learning Objectives

By the end of this lab, you will be able to:

- Explain what an API is and why they matter
- Distinguish between **building** APIs vs. **consuming** APIs
- Set up a Python environment for web development
- Create and run a minimal Flask application
- Write a client script that communicates with your API

What is an API?

API = **A**pplication **P**rogramming **I**nterface

Think of it as a **contract** between two pieces of software:

- "If you send me *this* request in *this* format..."
- "...I'll send you back *this* response in *this* format."

Analogy: A restaurant menu is like an API.

- You (the client) look at the menu (the API documentation)
- You place an order (send a request)
- The kitchen (the server) prepares and delivers your food (the response)

Why Do APIs Matter?

APIs enable **modularity** and **interoperability**:

- Your weather app doesn't have satellites—it calls a weather API
- ChatGPT's interface doesn't contain the model—it calls OpenAI's API
- Your learning app can combine multiple services without building everything from scratch

For this course: APIs let us **separate concerns**—your frontend can talk to your backend, and your backend can talk to AI services.

Two Types of API Work

Building APIs

Creating endpoints that others (or your own frontend) can call.

Web Frameworks:

- Flask
- FastAPI
- Django REST Framework

Consuming APIs

Calling someone else's endpoints to use their services.

Third-Party Services:

- OpenAI API
- Anthropic API
- Hugging Face API

Today: We focus on building. Later: we'll consume AI APIs.

Python Web Frameworks Compared

	Flask	**FastAPI**	**Django REST**
Learning Curve	Easy	Easy-Medium	Steeper
Speed	Good	Excellent	Good
Built-in Features	Minimal	Moderate	Extensive
Async Support	Add-on	Native	Add-on
Best For	Learning, small projects	Modern APIs, high performance	Large apps, full websites

Why Flask for this course?

- Minimal boilerplate—you see exactly what's happening
- Perfect for learning the fundamentals
- Easy to extend when you need more

Third-Party AI APIs (Preview)

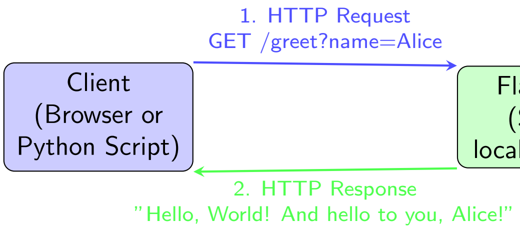
These are services where *someone else* built and hosts the API:

Provider	**What They Offer**	**You Send/Receive**
OpenAI	GPT models, DALL-E	Text prompts / Completions
Anthropic	Claude models	Text prompts / Completions
Hugging Face	500k+ models	Varies by model

The pattern: Your Flask app can *receive* requests from users, then *send* requests to these AI services, and return the results.

We'll integrate these in upcoming labs!

How a Client Talks to Flask



```
@app.route("/greet")
def greet():
    return ...
```

1. **Request:** Client sends a request to a URL (endpoint)
2. **Processing:** Flask matches the URL to a function via `@app.route`
3. **Response:** Flask returns the function's output to the client

Step 1: Create Your Conda Environment

Open your terminal (or Anaconda Prompt on Windows).

Create a new environment:

```
conda create -n flask-env python=3.11
```

Activate the environment:

```
conda activate flask-env
```

You should see `(flask-env)` at the start of your terminal prompt.

Why a separate environment? Keeps dependencies isolated—your Flask project won't conflict with other Python projects.

Step 2: Install Flask and Requests

With your environment activated, install the packages:

```
pip install flask requests
```

- `flask` – the web framework for building our API
- `requests` – a library for making HTTP requests (our client will use this)

Verify installation:

```
python -c  import flask; print(flask.__version__)
```

Step 3: Create the Flask App

Create a new file called `app.py`:

```
from flask import Flask, request

app = Flask(__name__)

@app.route( / )
def hello_world():
    return  Hello, World!

@app.route( /greet )
def greet():
    name = request.args.get( name ,  Friend )
    return f Hello, World! And hello to you, {name}!

if __name__ ==  __main__ :
    app.run(debug=True)
```

Understanding the Flask Code

- `Flask(__name__)` – Creates the application instance
- `@app.route("/")` – A **decorator** that maps a URL to a function
- `request.args.get("name", "Friend")` – Gets the `name` parameter from the URL query string; defaults to “Friend” if not provided
- `app.run(debug=True)` – Starts the development server with auto-reload

Two endpoints:

- `/` → Returns “Hello, World!”
- `/greet?name=Alice` → Returns “Hello, World! And hello to you, Alice!”

Step 4: Run the Flask App

In your terminal (with `flask-env` activated):

```
python app.py
```

You should see output like:

```
 * Running on http://127.0.0.1:5000
 * Debug mode: on
```

Test in your browser:

- Visit: `http://127.0.0.1:5000/`
- Visit: `http://127.0.0.1:5000/greet?name=YourName`

Keep this terminal running! Open a new terminal for the next step.

Step 5: Create the Client Script

In a **new file** called client.py:

```
import requests

BASE_URL =  http://127.0.0.1:5000

# First, get the basic Hello World
response = requests.get(f {BASE_URL}/ )
print(response.text)

# Now, ask for the user's name and greet them
name = input( Enter your name:  )
response = requests.get(f {BASE_URL}/greet , params={ name : name})
print(response.text)
```

Understanding the Client Code

- `requests.get(url)` – Sends an HTTP GET request to the URL
- `params={"name": name}` – Adds query parameters to the URL
  - This turns into `/greet?name=Alice`
- `response.text` – The body of the response (what Flask returned)

The flow:

1. Client sends request $\rightarrow$ Flask receives it
2. Flask processes and returns response
3. Client receives and displays response

Step 6: Run the Client

Open a **new terminal window** (keep Flask running in the first one).

Activate your environment and run:

```
conda activate flask-env
python client.py
```

Expected output:

```
Hello, World!
Enter your name: Alice
Hello, World! And hello to you, Alice!
```

Congratulations! You've built your first API and client.

Step 7: Stopping the Flask Server

When you're done, stop the Flask server:

In the terminal running Flask:

```
Press Ctrl+C
```

You'll see something like:

```
^C
 * Shutting down...
```

To deactivate your conda environment (optional):

```
conda deactivate
```

Troubleshooting Common Issues

“Connection refused” error in client:

- Is Flask still running? Check the other terminal.
- Is it running on port 5000? Check the Flask output.

“Address already in use”:

- Another process is using port 5000
- Find and stop it, or change Flask’s port: `app.run(port=5001)`

“ModuleNotFoundError: No module named flask”:

- Did you activate the environment? Check for `(flask-env)`
- Did you install Flask in this environment?

Recap: What You Learned

1. **APIs** are contracts between software components
2. **Web frameworks** (Flask, FastAPI, Django) let you *build* APIs
3. **Third-party APIs** (OpenAI, Anthropic) let you *consume* AI services
4. **Flask basics:** routes, request parameters, running the server
5. **The requests library** lets Python scripts talk to APIs

Next time: We'll extend this to return JSON and integrate with AI APIs!