LESSON 55

Handling Requests

80 minutes (2 periods) | Python Web Backend | Outcome 1.22 | Outcome 1.23 | Outcome 2.15 | Outcome 2.7 | Outcome 2.20 | Outcome 3.3 | Outcome 1.3

WHAT THIS LESSON DOES

In this lesson, you'll explore how to manage GET and POST requests in Flask, crucial for creating interactive web applications. Follow step-by-step instructions to set up a project, build routes, handle forms, and process user data effectively.

CURRICULUM OUTCOMES

Outcome	What the specification asks for
1.22	read, write, test, and modify computer programs
1.23	reflect and communicate on the design and development process
2.15	explain what is meant by the World Wide Web (WWW) and the Internet, including the client server model, hardware components and communication protocols
2.7	implement algorithms using a programming language to solve a range of problems
2.20	identify and fix/debug warnings and errors in computer code and modify as required
3.3	use appropriate programming languages to develop an interactive website that can display information from a database that meets a set of users' needs
1.3	solve problems by deconstructing them into smaller units using a systematic approach in an iterative fashion

WHAT STUDENTS SHOULD BE ABLE TO DO

- Understand the fundamental differences between GET and POST requests in web development.
- Develop skills to create and manage GET routes for retrieving data in a Flask application.
- Acquire the ability to handle POST requests for processing user-submitted data via forms.
- Learn to utilise query parameters in GET requests to customise server responses.
- Build competence in combining multiple HTTP methods within a single route for efficient code structure.

BEFORE THE LESSON

- Run through the whole build yourself once: creating the venv, pip installing Flask, and confirming flask run serves at 127.0.0.1:5000. School filters sometimes block pip, so test install on a lab machine, not just your own laptop.
- Confirm the Python extension is installed in VS Code so Python: Create Environment appears in the Command Palette.
- Have a clean copy of app.py and form.html saved somewhere the class can reach, so students who fall behind on the environment steps can still work on requests.

Handling Requests

HOW THE LESSON RUNS

- Introduction
- Create a Project Folder
- Create the Environment
- Create a New Terminal and Install Flask
- Creating a Basic Flask App with GET
- Setting Up Templates for Forms
- Handling GET for the Form
- Handling POST Requests
- Combining GET and POST in One Route
- Query Parameters
- Challenge
- Wrap Up

TEACHING MOVES

- 2. Create a Project Folder. Insist everyone names the folder FlaskRequests exactly and opens it with File > Open Folder, not by opening a single file. Flask and templates only resolve correctly when VS Code has the folder as its workspace root.
- 3. Create the Environment. Watch for the .venv folder appearing and the interpreter switching in the status bar. Students who skip activation will pip install Flask globally and get confusing import errors later.
- 4. Create a New Terminal and Install Flask. Check the terminal prompt shows (.venv) before running pip install flask. If the install stalls behind the filter, hand out your pre-installed copy rather than losing the period to network troubleshooting.
- 5. Creating a Basic Flask App with GET. Stress that visiting a URL in a browser is itself a GET request. Get the app running and the welcome message showing before adding anything else, so the baseline works.
- 6. Setting Up Templates for Forms. The folder must be named templates exactly, lowercase, inside FlaskRequests. render_template looks there by convention. A misnamed folder produces a TemplateNotFound error that puzzles students.
- 7. Handling GET for the Form. Point out that methods=['GET'] just displays the form. Nothing is processed yet. Import request now even though it is unused, and say plainly it is for the next step.
- 8. Handling POST Requests. Show that request.form['name'] reads the value by the input's name attribute, not its id. Match the name in the HTML to the key in Python side by side on screen.
- 9. Combining GET and POST in One Route. Explain the pattern: one route, methods=['GET','POST'], and if request.method == 'POST' decides what happens. Change the form action to /form first, or submissions hit the old /submit route.
- 10. Query Parameters. Type a URL with ?query=flask live and show the value reaching the server through request.args.get. Contrast this with form data: query parameters sit in the URL and belong to GET, not POST.
- 11. Challenge. Have students add one extra field, matching name attribute in HTML and key in Python, before attempting query parameters. Circulate and ask them to predict the output before running.

WHAT TO WATCH FOR

- Students think a page that displays without error means the request worked correctly. A form that submits but reads the wrong field still looks fine in the browser. Have them deliberately mistype a name attribute and observe the KeyError or blank output. Correct behaviour is proved by the data appearing, not by the page loading.
- Confusing GET query parameters with POST form data, and expecting request.form to hold URL values or request.args to hold submitted form fields. Put a two-column note on the board: form data goes through request.form on POST, URL parameters go through request.args on GET. Point to which the current route uses.
- Reading form values by the input id instead of its name attribute, so request.form returns nothing. State the rule once clearly: Flask matches on name, not id. Have them check every input has a name before submitting.

DIFFERENTIATION

- Emerging: Pair with a student whose environment is working and give them your pre-built app.py and form.html so they focus on requests rather than losing the period to venv setup. Get the basic GET welcome route running first and confirm it in the browser before touching forms.
- Developing: Ask them to explain aloud what happens when they click Submit, tracing the request from form to route to response, before adding query parameters.
- Proficient: Set the combined GET and POST route as the goal, then have them add validation that rejects an empty name and returns a helpful message. Ask them to note how this request handling could feature as evidence in an Applied Learning Task on web development, distinguishing client and server responsibilities.