INTRO APP WEB

Politecnico di Torino

Introduction to Web Applications

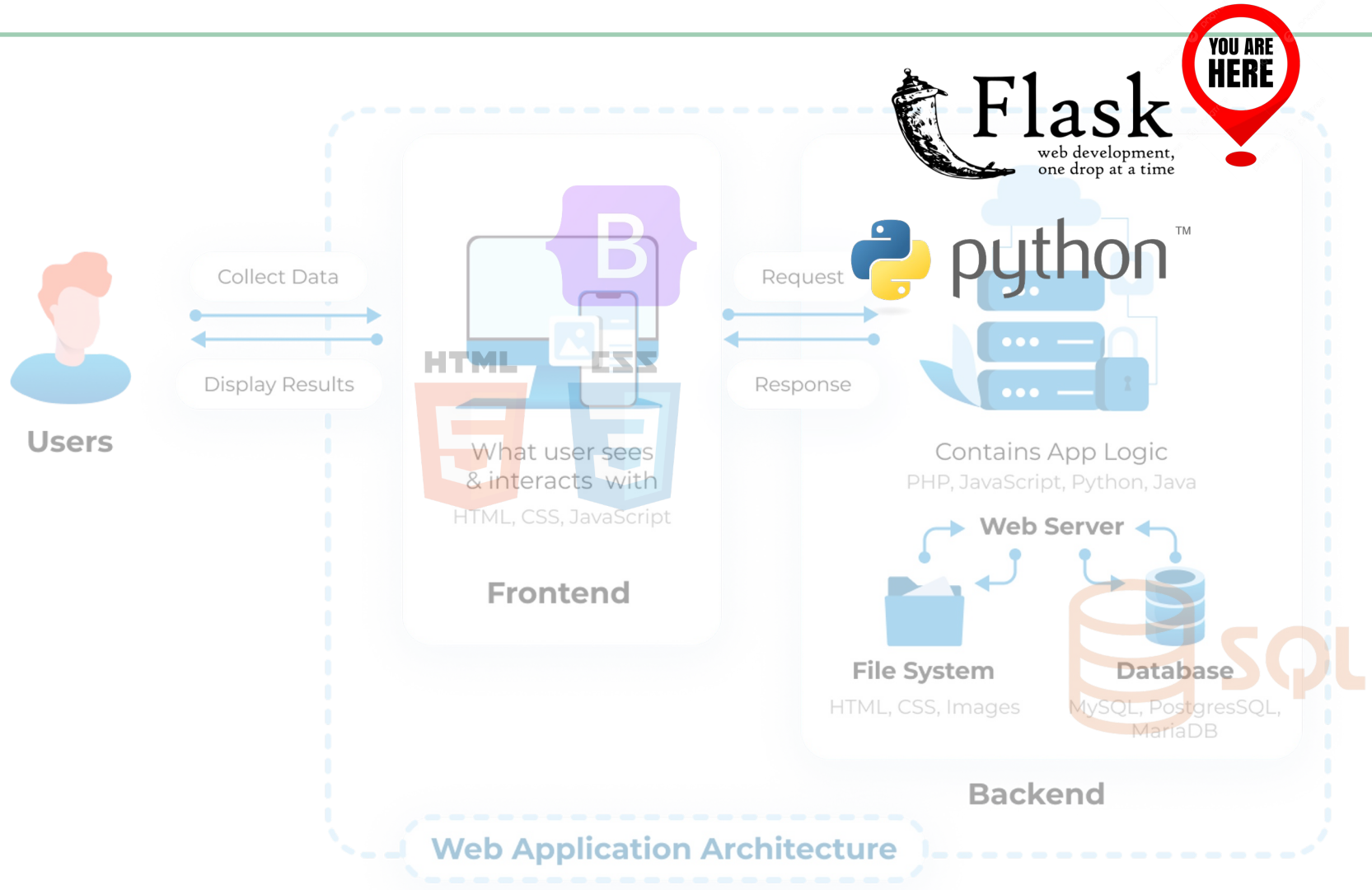
Flask

Juan Pablo Sáenz

Goals

- Understand **what Flask is** and where it fits within the **web architecture**
- **Install** and **set up** Flask in our development environment
- Create a basic Flask application and **handle routes**
- **Integrate Flask** with the webpages we have developed so far

Flask: where are we?



🗺️ Web architecture components: Backend

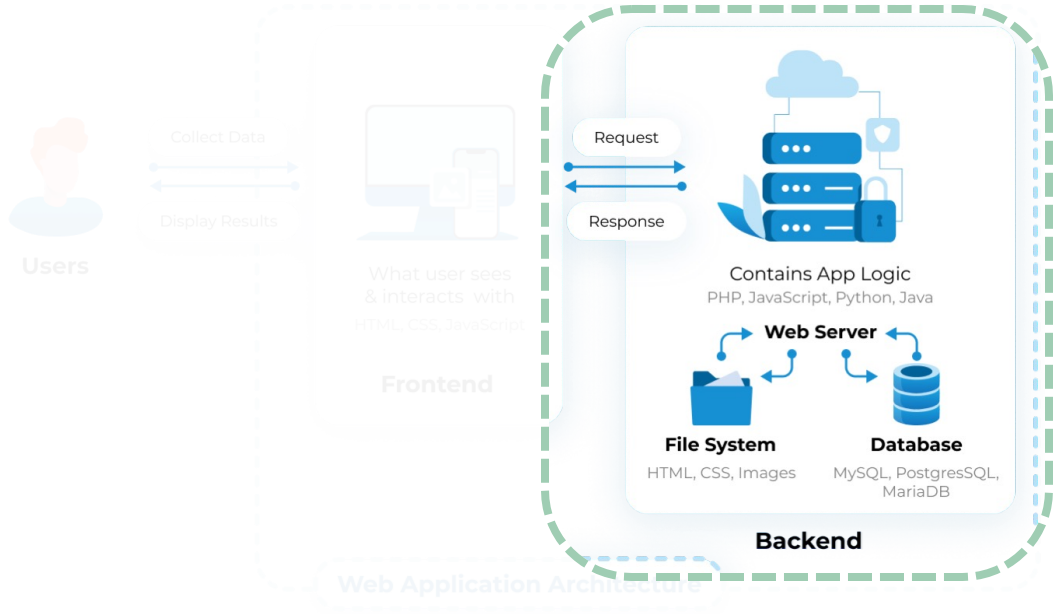
Backend: The part of a web application that works behind the scenes, handling **data**, **logic**, and server **communication**.

🧩 **Components**:

Server: a computer or system that provides **resources**, **data**, **services**, or **programs** to other computers, known as clients, over a network.

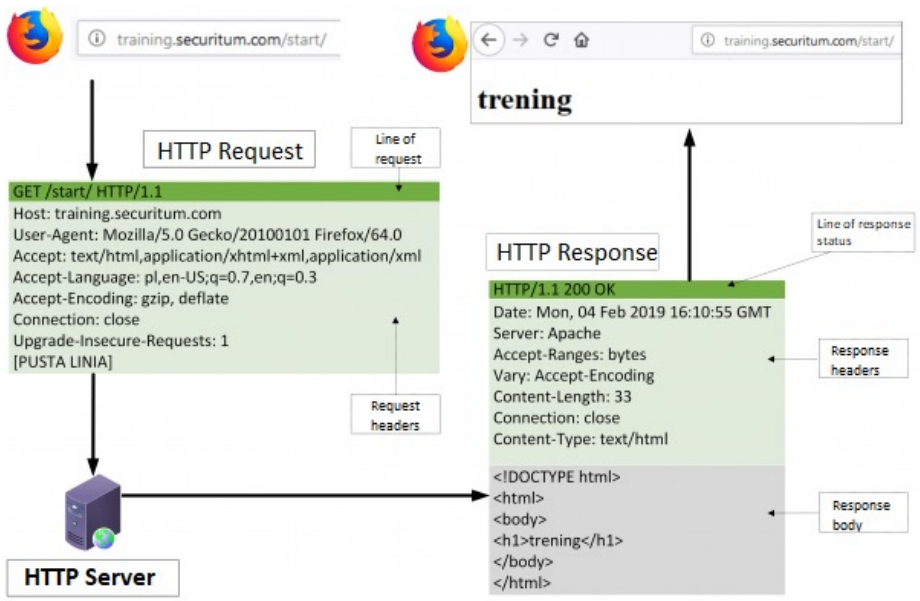
Web server: **hosting websites**, **handling HTTP requests** from clients (like browsers), and **delivering web pages**, images, and other content to the client's device.

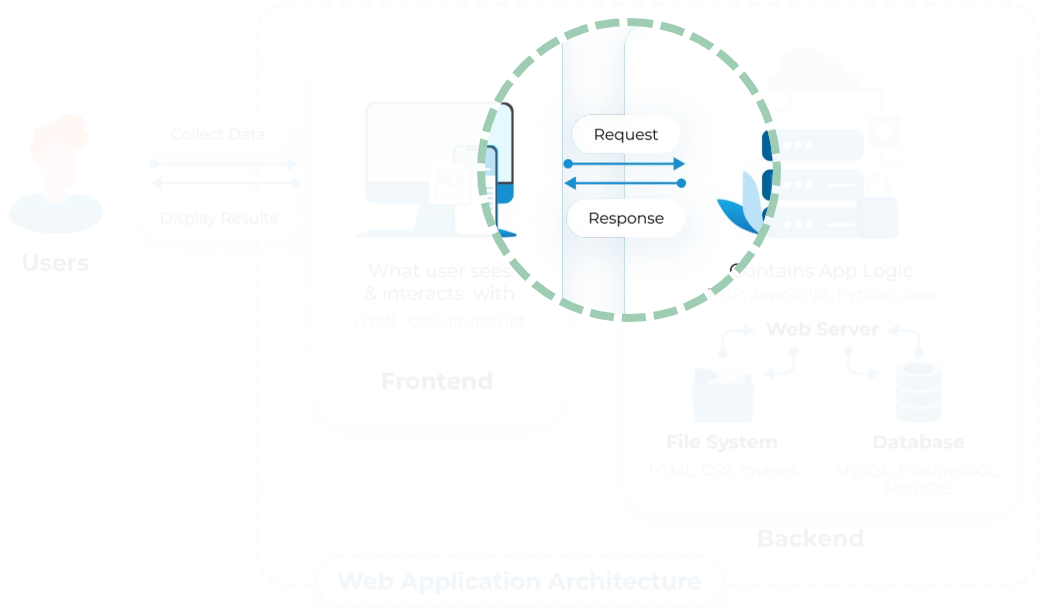
Intermediary between the client and the **database**



Web architecture components: HTTP Protocol

HTTP Protocol: the protocol used for **transferring data over the web**, allowing communication between **clients** (like browsers) and **servers** by sending requests and receiving responses

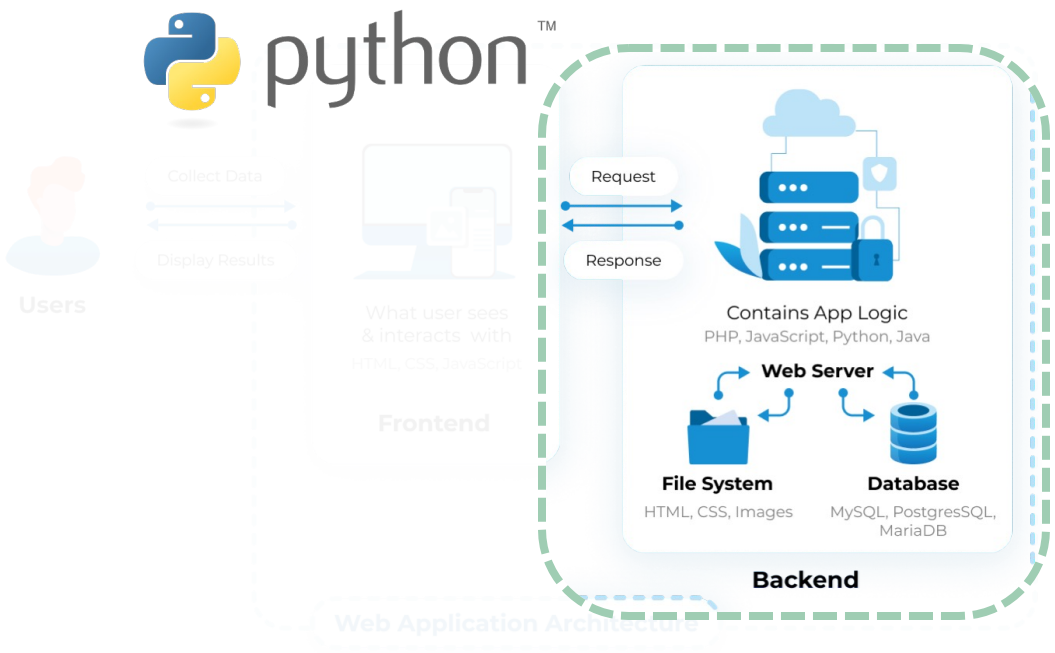




Why Flask?

Python includes **`SimpleHTTPServer`** to activate a **basic web server**

- it is **low-level** and **NOT** very **developer-friendly**



Flask

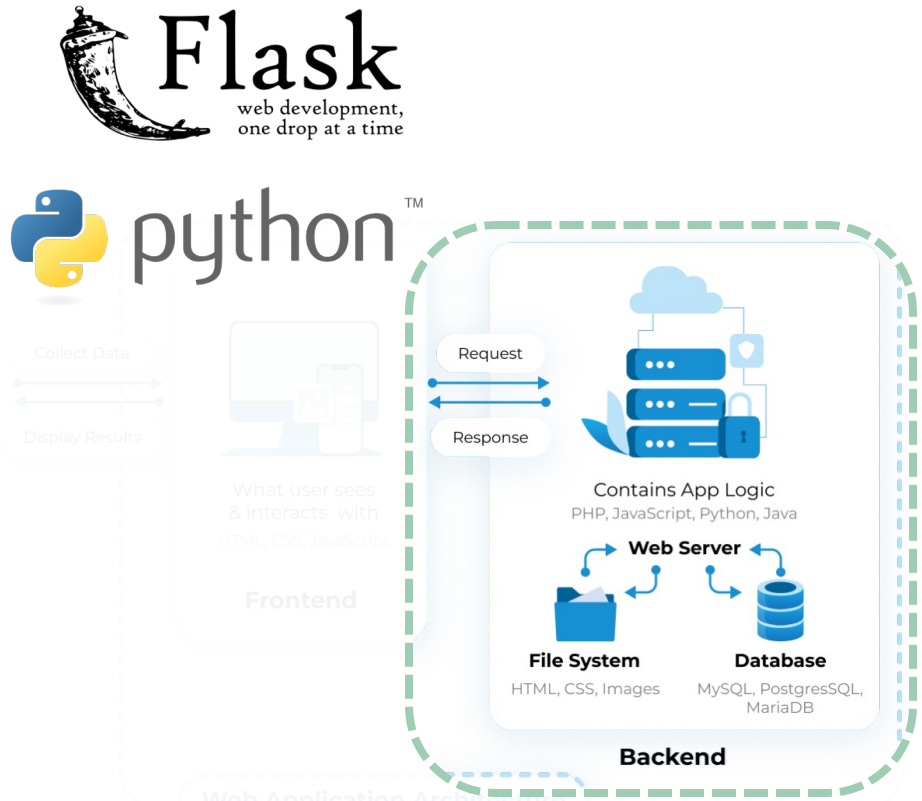
Flask is a lightweight and flexible **web framework** for **Python**

Designed to be **simple** and **minimal**

🔗 **https://flask.palletsprojects.com/en/stable/**

It provides tools to:

- Handle **HTTP requests**
- **Define routes** to map **URLs** to **functions**
- **Render templates** for dynamic content
- Manage **sessions** and **user data**
- Integrate with **databases**



Flask Installation and Setting Up

📌 Prerequisite

Ensure **Python is installed** (recommended: Python 3.7 or later)

- 🔗 **https://www.python.org/downloads/**

Check **Python version** (running the command in the terminal)

> 👉 If the '**python**' command is not found, use '**python3**' instead

```
python --version

python3 --version
```

Flask Installation and Setting Up

Step 1: Install Flask using `pip` (by running a terminal command)

- **`pip`** is a **package manager** for **Python**, used to install and manage **external libraries** and **dependencies** for Python projects

Step 2: Verify installation

```
pip install Flask

pip3 install Flask


python -m flask --version

python3 -m flask --version
```

Flask Installation and Setting Up

Step 3: Create a basic Flask application

- Create a Python file (**app.py**)

```python
from flask import Flask

app = Flask(__name__)

@app.route("/")
def home():
    return "Hello, Flask!"
```

Flask Installation and Setting Up

Step 4: Run the application

- If the Python file is named **`app.py`**, you can run it with the command **`flask run`**
- In the first option, "**`main`**" is the name (with the path, and without extension) of the Python file

🎉 **The web server starts and listens for incoming HTTP requests!**

- It is accessible via a web browser at **`http://127.0.0.1:5000`** by default

```
flask --app main run

flask run
```

Let's see it in practice

Understanding Flask

Importing and initializing Flask:

The **Flask class** is used to define the app instance that handles incoming requests and manages the routing

Defining a route:

`@app.route("/")` is a **decorator**

A function that modifies the behavior of another function or method

`@app.route("/")` creates a **route** for the homepage (URL `/`), which **triggers** the **`home()`** function

```python
# Import the Flask class from the
flask module
from flask import Flask

# Initialize the Flask application
app = Flask(__name__)

# Define a route for the homepage
(URL "/")
@app.route("/")
def home():
    return "Hello, Flask!"
```

Understanding Flask

Defining a route:

Each web page is represented by a **route decorator** and its corresponding **function**

Returning a response:

the **`home()`** function returns **"Hello, Flask!"** to be displayed in the browser

HTML is the default content type

```
# Import the Flask class from the
flask module
from flask import Flask

# Initialize the Flask application
app = Flask(__name__)

# Define a route for the homepage
(URL "/")
@app.route("/")
def home():
    return "Hello, Flask!"
```

Re-running it

Run the application

- If the Python file is named **`app.py`**, you can run it with the command **`flask run`**
- In the first option, "**`main`**" is the name (with the path, and without extension) of the Python file

🎉 **The web server starts and listens for incoming HTTP requests!**

- It is accessible via a web browser at **`http://127.0.0.1:5000`** by default

```
flask --app main run

flask run

flask run --debug
```

Flask Routes Returning HTML Content as Strings 🤢

The **Flask app** defines two **routes**:

- The **homepage** «**/**»
- The **About** page «**/about**»

Each route returns **a complete HTML page** as a response using triple-quoted **strings**

```
@app.route('/')
def index():
    return """<html>
    <head><title>IAW App Home</title></head>
    <body><h1>My IAW App</h1>
    <p>Welcome</p>
    </body>
    </html>
    """
@app.route('/about')
def about():
    return """<html>
    <head><title>IAW App About</title></head>
    <body><h1>Who are we?</h1>
    </body>
    </html>
    """
```

Flask Routes Returning HTML Content Using Templates 🤩

Flask supports **Jinja** out of the box.
https://palletsprojects.com/projects/jinja/

- A templating engine for rendering **dynamic content** in HTML
- Templates are **HTML files**, with **`.html`** extension
- To render a template you can use the **`render_template()`** method

⚠️ **Flask** will look for templates in the «**`templates`**» subfolder

```
@app.route('/')
def index():
    return render_template('index.html')

@app.route('/about')
def about():
    return render_template('about.html')
```

Flask Routes Returning HTML Content Using Templates 🤩

Jinja allows you to embed **Python-like expressions** in HTML files

Supports features like:

- **loops**, **conditionals**, **filters**, and template **inheritance**,

making it powerful for generating **dynamic** web pages

Jinja features: Loops

`{% statement %}`

- **controls** the template's execution flow

`{{ expression }}`

- **evaluates** a variable or expression and outputs the result in the HTML
- **`|  e`**: to escape **>**, **<**, &, and "

```python
# app.py
@app.route("/")
def index():
  items = ["Apple", "Banana", "Cherry"]
  return render_template("index.html", p_items=items)
```

```jinja
<!-- templates/index.html -->
<ul>
  {% for item in p_items %}
    <li>{{ item | e}}</li>
  {% endfor %}
</ul>
```

Jinja features: Conditionals

Welcome, John!

- If **`user=None`**, it displays «**Welcome, Guest!**» instead

`{% elif another_condition %}` is also possible

```
# app.py
@app.route('/')
def index():
  return render_template('index.html', user='John')
```

```
{% if user %}
  <p>Welcome, {{ user }}!</p>
{% else %}
  <p>Welcome, Guest!</p>
{% endif %}
```

Jinja features: Inheritance (extending a base template)

```
<!-- templates/base.html -->
<!DOCTYPE html>
<html>
  <head>
    <title>My Website - {% block title %}{%
endblock %}</title>
  </head>
  <body>
    <header>
      <h1>Welcome to My Website</h1>
    </header>
    <main>{% block content %}{% endblock
%}</main>
  </body>
</html>
```

```
<!-- templates/index.html -->
{% extends 'base.html' %}

{% block title %}Home{% endblock %}

{% block content %}
<p>This is the homepage.</p>
{% endblock %}
```

Jinja features: Inheritance (extending a base template)

In more complex web applications, multiple pages share **common elements** such as:

- Navigation bars
- Footers
- Page layouts

Jinja provides **blocks** and **template inheritance** to handle this efficiently

Jinja features: Inheritance (extending a base template)

{% block <block_name> %}…{% endblock %}

- Defines a reusable HTML block in a template
- **<block_name>** must be unique within the template

{% extends "filename.html" %}

- Extends a parent template for reuse in a child template

```
<!-- templates/about.html -->
{% extends 'base.html' %}

{% block title %}About{% endblock %}

{% block content %}
<p>Who we are?</p>
{% endblock %}
```

Jinja features: `url_for`

- **`url_for('route_name')`**: generates a URL for a Flask route
- **`url_for('static', filename='path/to/file')`**: links to static assets like images, CSS, and JS

```
<!DOCTYPE html>
<head>
<title>Example</title>
</head>
<body>
<!-- Link to another page -->
<a href="{{ url_for('about') }}">Go to
About Page</a>
<!-- Display an image from the static
folder -->
<img src="{{ url_for('static',
filename='images/logo.png') }}" alt="Logo">
</body>
</html>
```

Folder Structure for a Flask Project

- 📦 project_root
  - 📁 static
    - 📁 images
      - 🌟 logo.png
  - 📁 templates
    - 📄 index.html
    - 📄 about.html
  - 🐍 app.py

```
<!DOCTYPE html>
<head>
<title>Example</title>
</head>
<body>
<!-- Link to another page -->
<a href="{{ url_for('about') }}">Go to
About Page</a>
<!-- Display an image from the static
folder -->
<img src="{{ url_for('static',
filename='images/logo.png') }}" alt="Logo">
</body>
</html>
```

Let's see it in practice

Flask Dynamic Routes

- **Dynamic routes** capture values **from the URL** and pass them to **view functions**
- They provide flexibility for creating **reusable routes** in web applications

```
@app.route("/user/<username>")
def show_user_profile(username):
    return f"User {username}"
```

Flask Dynamic Routes

- **Dynamic routes** capture values **from the URL** and pass them to **view functions**
- They provide flexibility for creating **reusable routes** in web applications
- The **`int:`** in **`<int:post_id>`** specifies the type of value expected in the URL

  `string:` – Expects a string (default behavior)

  `float:` – Expects a floating-point number

```
@app.route('/post/<int:post_id>’)
def show_post(post_id):
    return f'Post {post_id}'
```

Flask Dynamic Routes

- The **`url_for`** function generates a URL for a specific view function based on its name and any dynamic parameters.
- This would generate a URL like **`/user/jpsaenzmo`**, which can be used to navigate to the user's profile.

```
url_for('show_user_profile',
username='jpsaenzmo')
```

Licenza

- These slides are distributed under a Creative Commons license “**Attribution-NonCommercial-ShareAlike 4.0 International (CC BY-NC-SA 4.0)**”
- **You are free to:**
  - **Share** — copy and redistribute the material in any medium or format
  - **Adapt** — remix, transform, and build upon the material
  - The licensor cannot revoke these freedoms as long as you follow the license terms.
- **Under the following terms:**
  - **Attribution** — You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.
  - **NonCommercial** — You may not use the material for commercial purposes.
  - **ShareAlike** — If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original.
  - **No additional restrictions** — You may not apply legal terms or technological measures that legally restrict others from doing anything the license permits.
- https://creativecommons.org/licenses/by-nc-sa/4.0/