Search

Flask Workshop

MLDA

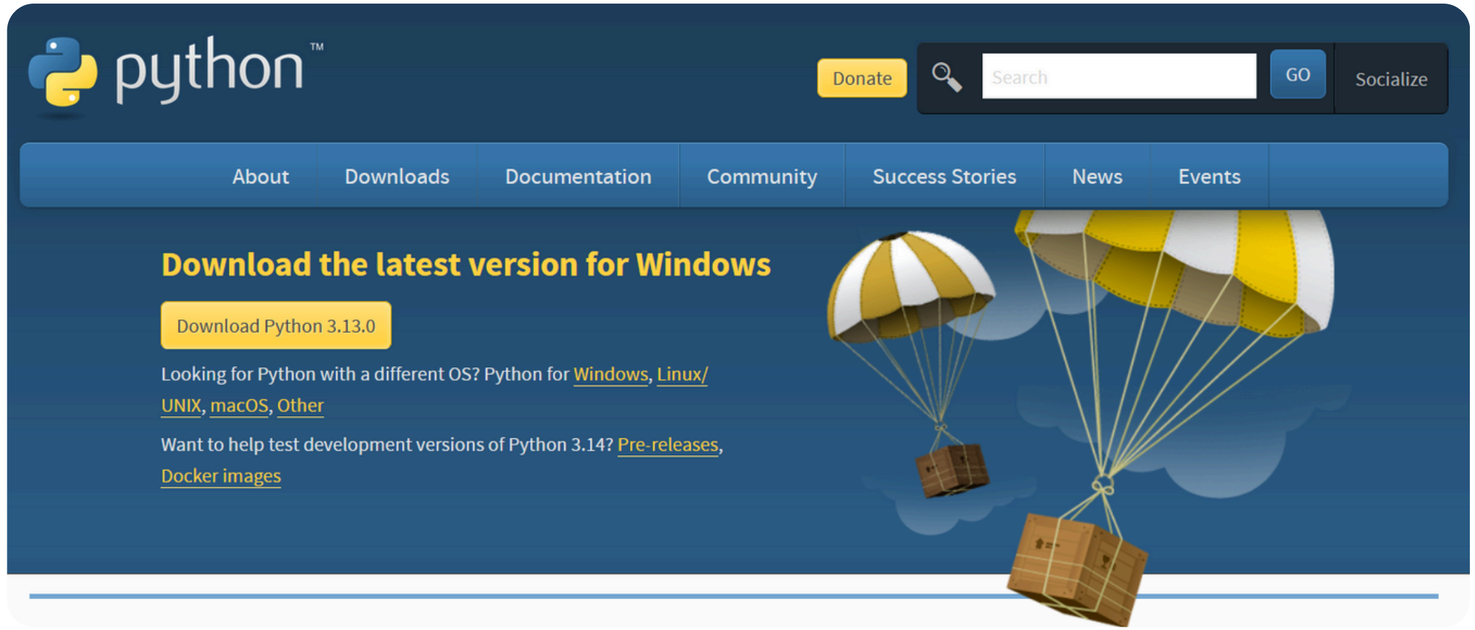
Setup

Preparing the environment for web development.

Installing Python

Must Have!

Head over to: https://www.python.org/downloads/
and click on Download Python



Download

Install

Installing VS Code

Must Have!

Head over to: https://code.visualstudio.com/download
and choose the installer for your OS

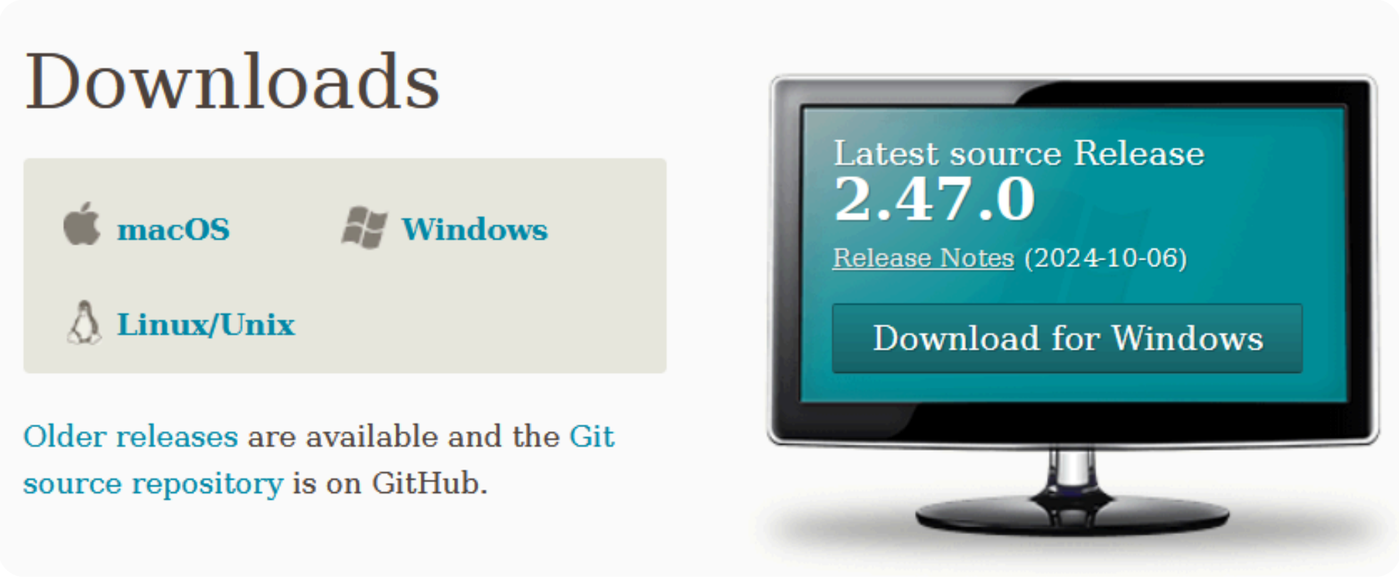
Download

Install

Installing Git

Optional

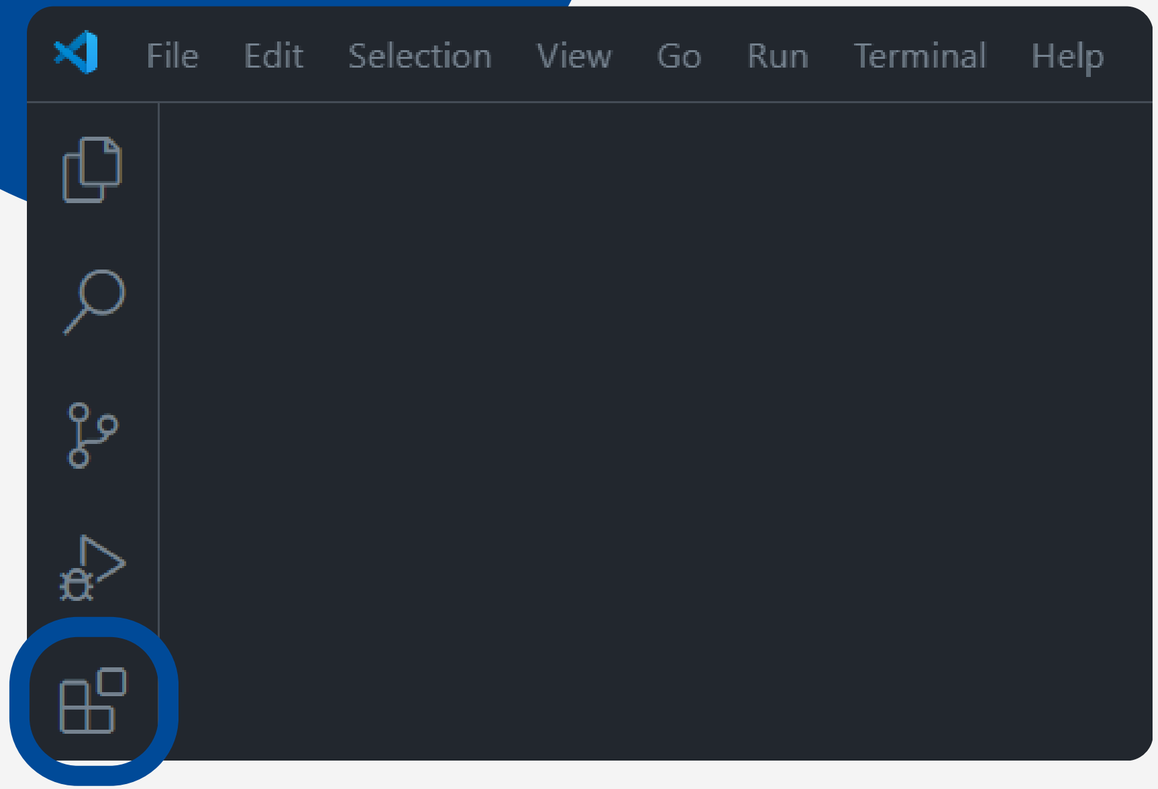
Head over to: https://git-scm.com/downloads
and choose your OS and follow the instructions

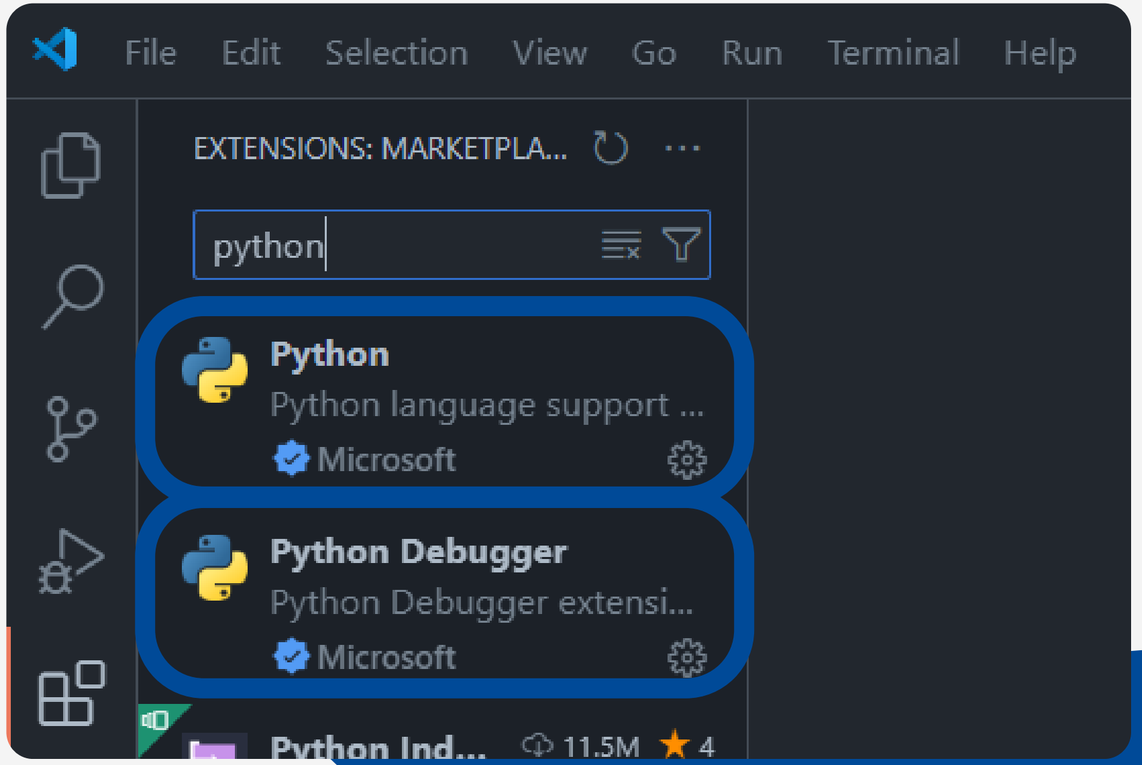


Download

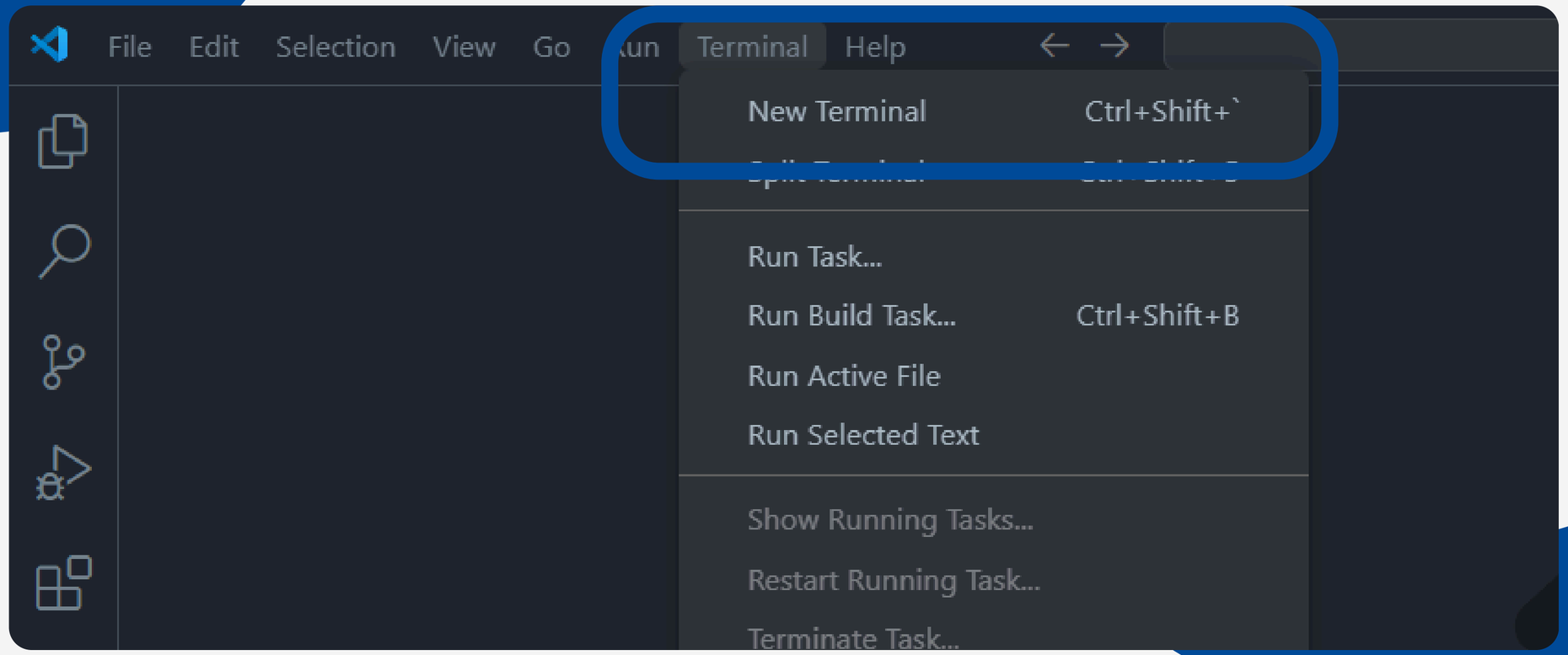
Install

Setting up VS code for python

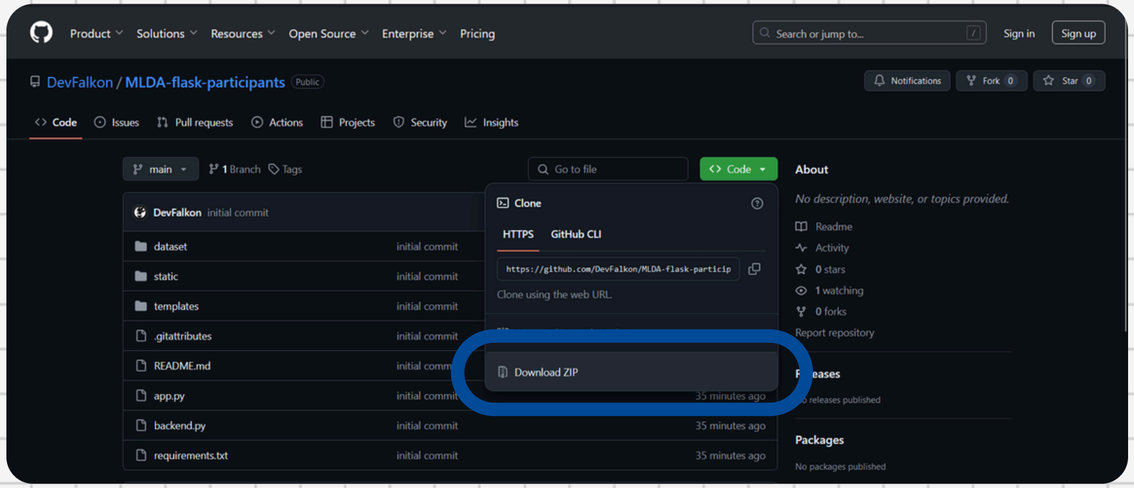




Setting up python environment



Clone the Workshop's GitHub repository.
If you don't have git installed, install the zip file from:
https://github.com/DevFalkon/MLDA-flask-participants)



For mac and linux:
source ./env/bin/activate

Each virtual environment can have its own set of dependencies, allowing us to manage packages for different projects separately. This prevents version conflicts.

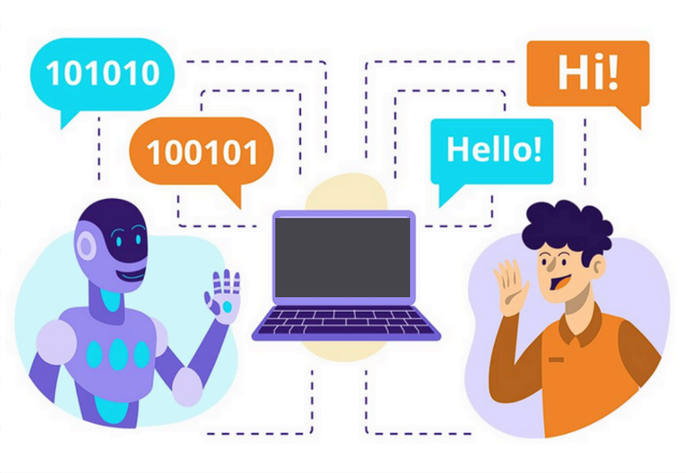
All the dependencies (python packages) for this project have been listed in requirements.txt

ML Model

- NLP
- TF-IDF
- Cosine Similarity

NLP

Using AI and ML to allow computers to understand and communicate with human language.



TF-IDF

Term Frequency-Inverse Document Frequency:
Vector that helps to determine the importance of the word in a given document

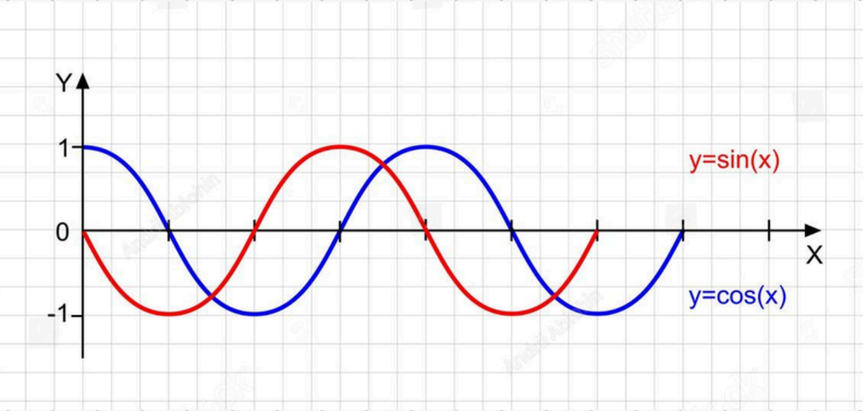
Term Frequency

Inverse document Frequency

TF-IDF score calculated

Cosine similarity

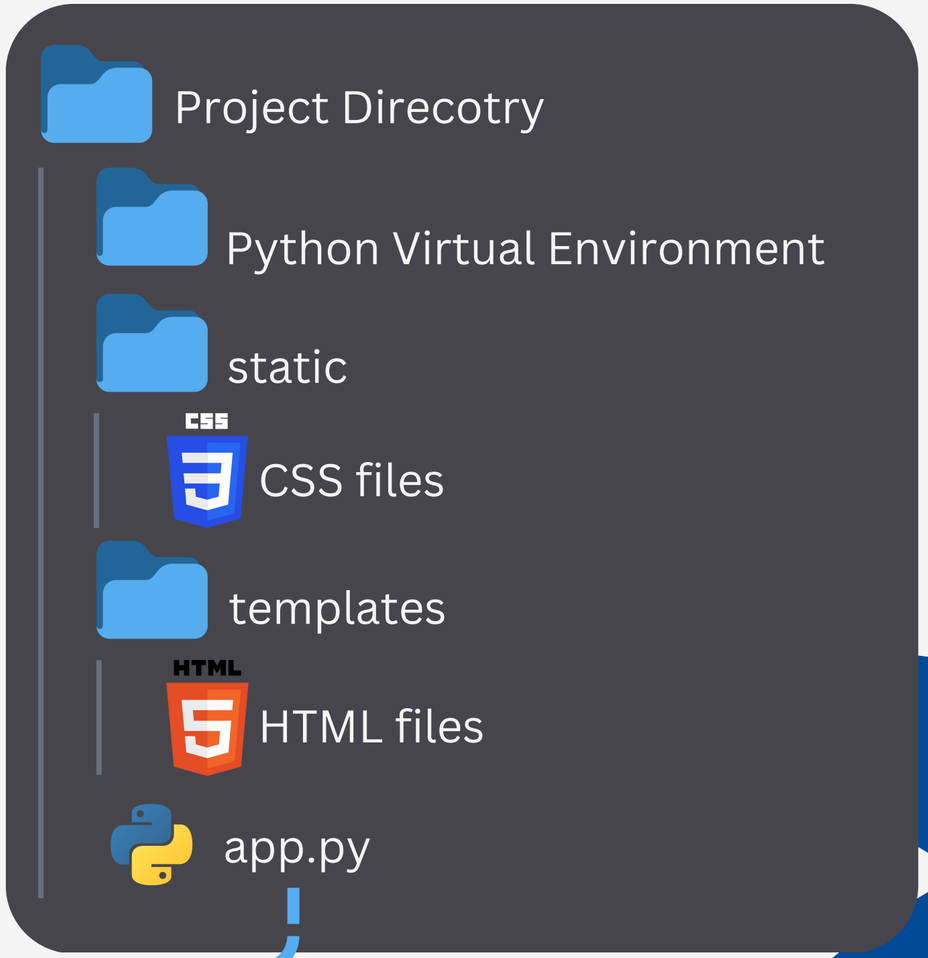
Similarity between two vectors based on the angles between them



Flask

Flask is a lightweight web framework for Python, ideal for small to medium-sized applications, prototypes, and services, providing a good balance between simplicity and functionality.

Project Directory Structure



While you can name your main Flask file anything you like, app.py is a widely recognized choice

Python Decorators

```
def greet(func):
    def wrapper():
        print("Hello")
        func()
    return wrapper

@greet
def func1():
    print("world!")

func1()
```

Decorators are functions that modify the behavior of another function. Think of them as wrappers that you can apply to functions to extend or change what they do without modifying their actual code.

A decorator is a function that takes another function as an argument.

Inside the decorator, you define a new function (often called a "wrapper") that adds some behavior before or after the original function runs.

OUTPUT:

```
Hello
world!
```

Routes

Routes are the URL patterns that the application responds to. Each route is associated with a specific function (using the @app.route() decorator) that gets executed when the URL is accessed.

Endpoints

Specific function that handles a request for a particular route. Each route corresponds to an endpoint, which is the callable function that gets executed when the route is accessed.

HTTP methods

Standardized ways that clients (like web browsers) communicate with servers over the Internet. Each method specifies a different action that the client wants to perform on the resource identified by the URL

GET

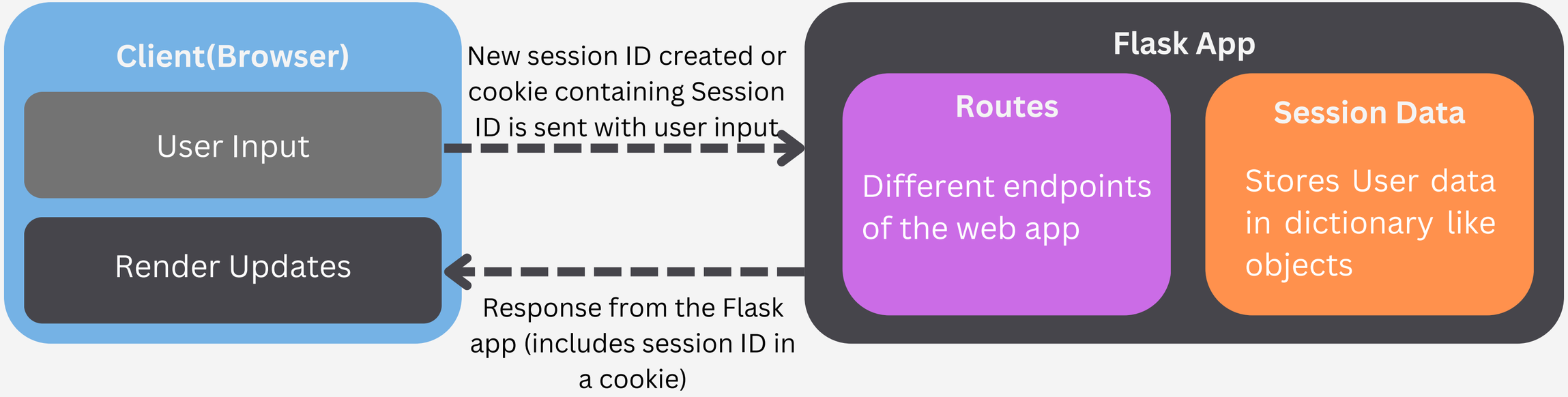
GET is used to request data from a specified resource. It is a read-only operation and should not modify any server state.
Use Case: Fetching a webpage or retrieving data from an API.

POST

POST is used to Send data to the server to create or update a resource. It often modifies the server state.
Use Case: Submitting form data, uploading files, or creating a new resource (e.g., a new user account).

Flask Sessions

1. **A way to store data for a user as they navigate the web application. Unlike cookies, which are stored on the client side, session data is stored on the server, and a session ID is sent to the client in a cookie. This ID is used to retrieve the corresponding session data on the server.**
2. **A cookie containing the session ID is sent to the client. When the client makes subsequent requests, this cookie is sent back to the server, allowing Flask to retrieve the session data.**



Importring modules from flask

Initializing flask

Using __name__ helps Flask behave correctly depending on whether your application is run directly or imported as a module in another script.
If the app is run directly, __name__ will be set to "__main__"

The secret_key is used to sign the session cookie, ensuring that the session data is secure.

The app.route() decorator is used to define the URL paths that the flask application will respond to. It associates a specific URL with a function (endpoint) , allowing us to handle incoming web requests for that URL.

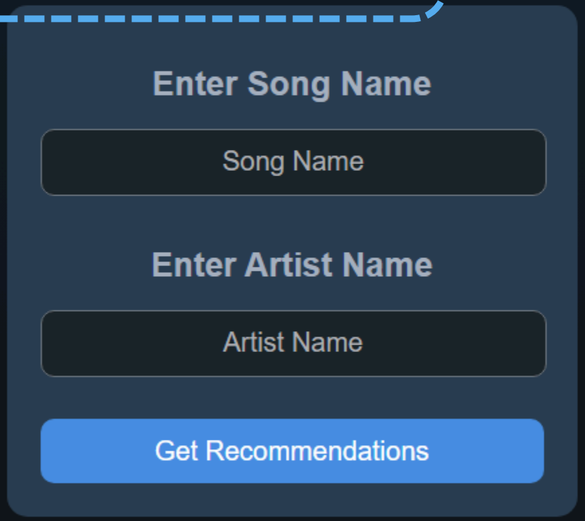
```python
from flask import Flask, render_template, request, flash, redirect, url_for, session


app = Flask(__name__)
app.secret_key = "BAD_SECRET_KEY"

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

The primary purpose of render_template() is to generate HTML dynamically by combining a template file (usually an HTML file) with data passed from your Flask application. This helps create more dynamic and interactive web pages.

http://127.0.0.1:5000

- Root/home URL: The URL of the home page of the website.
- it is represented by a forward slash (/)
- In this case, the root URL is http://127.0.0.1:5000/

```python
@app.route("/")
def home():
    return render_template("index.html")
```

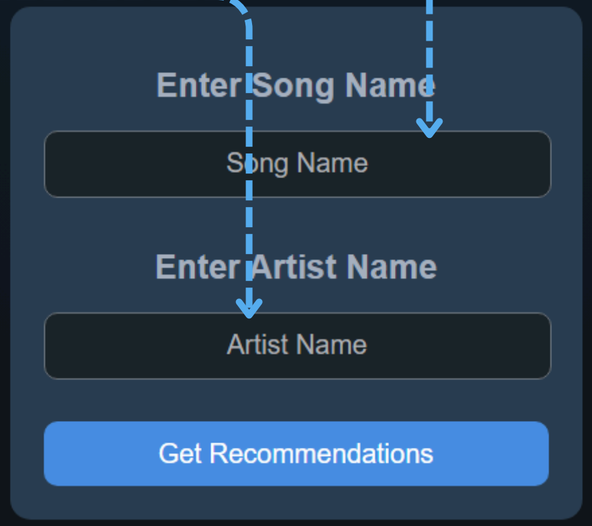


User Input

- Create a simple HTML form that allows users to submit data
- Create a route to handle the form submission.
- You can also get user input directly from the URL using query parameters. This is useful for simpler inputs.
- It's important to validate input data to avoid issues such as SQL injection or unexpected errors. Libraries such as Flask-WTF can be used for form validation.

```html
<form method="post">
    <h2>Enter Song Name</h2>
    <input type="text" name="song_name" placeholder="Song Name">
    <h2>Enter Artist Name</h2>
    <input type="text" name="artist_name" placeholder="Artist Name">
    <button type="submit">Get Recommendations</button>
</form>
```

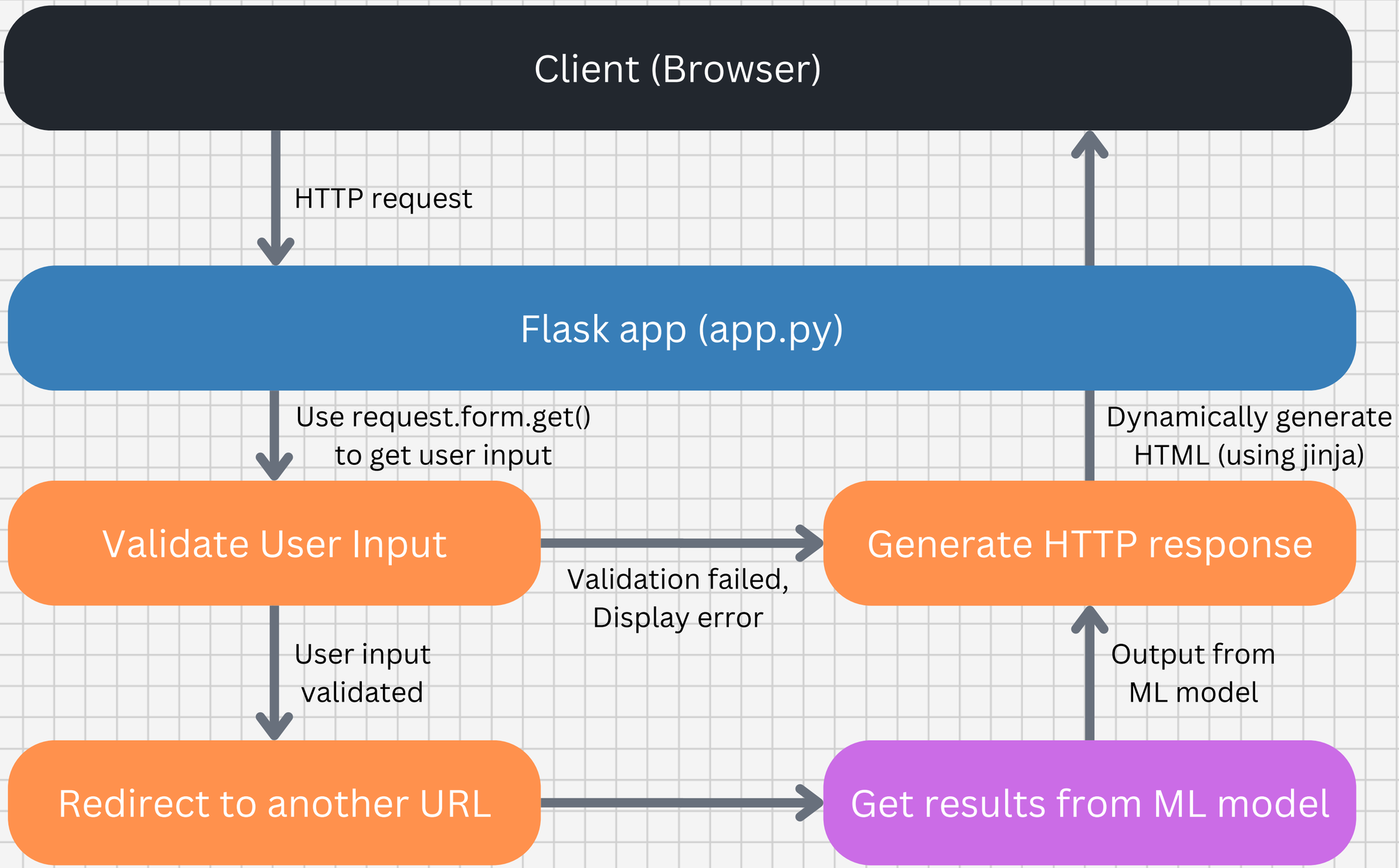
- POST method is used to send the user data from Clinet to Flask



- By default, routes in Flask respond to GET requests.
- If a method is not specified, a route may unintentionally respond to requests it shouldn't.

request.form.get() used to access form data submitted in a POST request. It retrieves the value associated with a specific key from the incoming form data.

```python
@app.route("/", methods=["POST"]
def get_user_input():
    if request.method == "POST":
        song_name = request.form.get("song_name")
        artist_name = request.form.get("artist_name")
```


Client (Browser)
HTTP request
Flask app (app.py)
Use request.form.get()
to get user input
Dynamically generate
HTML (using jinja)
Validate User Input
Generate HTTP response
Validation failed,
Display error
User input
validated
Output from
ML model
Redirect to another URL
Get results from ML model

Validate User Input

Validating user input

```
if not song_name:
    flash("Song name can't be blank", "error")
elif not artist_name:
    flash("Artist name can't be blank", "error")
else:
    song_id = backend.get_song_id(song_name, artist_name)
    if song_id:
        session['song_id'] = song_id
        return redirect(url_for("machine_learning", song_name=song_name, artist_name=artist_name))
    else:
        flash(f"{song_name} by {artist_name} not in data set", "error")
return render_template("index.html")
```

Check if he song name exists in the dataset
Get the song ID if it exists

Store song id in Flask session for accessing later

Redirect the Client to another URL if the Input has been validated and song exists in the dataset

If the validation fails or song is not present in the dataset, render index.html again as every endpoint must return a valid HTTP response

Flash an error if the the song id does not exist (***Flash***: used to send messages from your application to the user, typically to provide feedback after an action)

Get results from ML model

```
return redirect(url_for("machine_learning", song_name=song_name, artist_name=artist_name))
```

url_for() is a utility function that generates dynamic URLs for a given endpoint.

Utility function: A function that performs a specific, often repetitive, task that can be reused throughout the codebase.

Route URL is dynamically generated using song_name and artist_name passed as the arguments

```
@app.route("/recom/<song_name>_by_<artist_name>")
def machine_learning(song_name, artist_name):
    recom = backend.get_recom(session['song_id'])
    return render_template("recom.html", song_name=song_name.title(), artist_name=artist_name.title(), results=recom)
```

Return HTTP response

Get results from the ML model

Get results from ML model

Song id

Get similar songs based on song id

Return 5 similar songs (list)

```
[('In my feelingss', "['Drake']", 'https://images.genius.com/a32bad54a4d1048bf8ddd0add72b5bd9.300×300×1.jpg'), ('Take Care', "['Jennifer
[('Hotline bleeing', "['Drake']", 'https://images.genius.com/a32bad54a4d1048bf8ddd0add72b5bd9.300×300×1.jpg'), ('Hotline Bling', "['Kehl
[('Hennessy', "['Juice WRLD']", 'https://images.genius.com/42b0e9b115b685c8dd7e7f32e3f1ce9c.300×300×1.jpg'), ('Super Powers', "['Lil Dur
[('Liberation', "['Christina Aguilera']", 'https://images.genius.com/f6b6c4d1a85b142d997be0de22c9142b.300×300×1.png'), ('Jus Know', "['P
[('The Hills', "['Dua Lipa']", 'https://images.genius.com/87b285166213403fabe64d0440d6b583.300×300×1.png'), ('Fucking Fucked Up', "['Mil
[('Shape of You', "['Ed Sheeran']", 'https://images.genius.com/3710c6f28a35a0635d3e54ea30a8fc5c.300×300×1.png'), ('Feel Like', "['Queen'
[('Happy the Man', "['The Cure']", 'https://images.genius.com/d2eb6f2aafeb9ed7a6b0b3c78aed11ac.300×306×1.jpg'), ('Book of Love', "['Flee
[('Fortnight', "['Taylor Swift']", 'https://images.genius.com/9e602eb80c004d0302187f908fa5afea.300×300×1.png'), ('Fortnight', "['Taylor
[('Good Bye', "['Jeremy Zucker']", 'https://images.genius.com/777c12150937611df2e4049da88dd3b0.300×300×1.jpg'), ('I'm The Shit', "['Juic
[('Perfect', "['Ed Sheeran', 'New Kids In The Flock']", 'https://images.genius.com/643a3ac8828118a439649ae8c920fe1a.300×300×1.jpg'), ('P
[('Cardigan', "['Taylor Swift']", 'https://images.genius.com/df4a816f593b08bc8a361ad58a848640.300×300×1.jpg'), ('\u200bcardigan', "['Tay
[('I'm Still', "['Chris Brown']", 'https://images.genius.com/34b5eb355ff9bc8e966bd8aaf92fe73e.300×300×1.jpg'), ('Why U Look Mad', "['Lil
```

Song name

Artist(s) - list

Cover art

Generate HTTP response/ redirect to another URL

```
return render_template("recom.html", song_name=song_name.title(), artist_name=artist_name.title(), results=recom)
```

```
{% block content %}
    <div class="container">
        <h1>Recommendations based on "{{song_name}}" by {{artist_name}}</h1>
        <div id="recom_list">
            {% for name, artist, url in results %}
            <a href="https://www.google.com/search?q={{name}}+{{artist[0]}}" target="_blank">
                <div class="recom">
                    <img src="{{url}}" alt="{{name}} cover" class="cover-image">
                    <div class="info">
                        <p class="track-name">{{name}}</p>
                        <p class="artist-name">by {{artist[0]}}</p>
                    </div>
                </div>
            </a>
            {% endfor %}
        </div>
    </div>
{% endblock %}
```

Jinja

A templating engine for Python, which allows developers to generate dynamic HTML content by embedding Python-like expressions and logic directly within HTML files.

Create a base template with a consistent layout (like headers, footers, and navigation) that can be extended by child templates.

```
{% block content %}
    <div class="container">
        <h1>Recommendations based on "{{song_name}}" by {{artist_name}}</h1>
        <div id="recom_list">
            {% for name, artist, url in results %}
            <a href="https://www.google.com/search?q={{name}}+{{artist[0]}}" target="_blank">
                <div class="recom">
                    <img src="{{url}}" alt="{{name}} cover" class="cover-image">
                    <div class="info">
                        <p class="track-name">{{name}}</p>
                        <p class="artist-name">by {{artist[0]}}</p>
                    </div>
                </div>
            </a>
            {% endfor %}
        </div>
    </div>
{% endblock %}
```

{% %} Can be used to generate HTML using loops and conditional statements.

Double curly braces: {{ }} can be used to dynamically add variables to the HTML

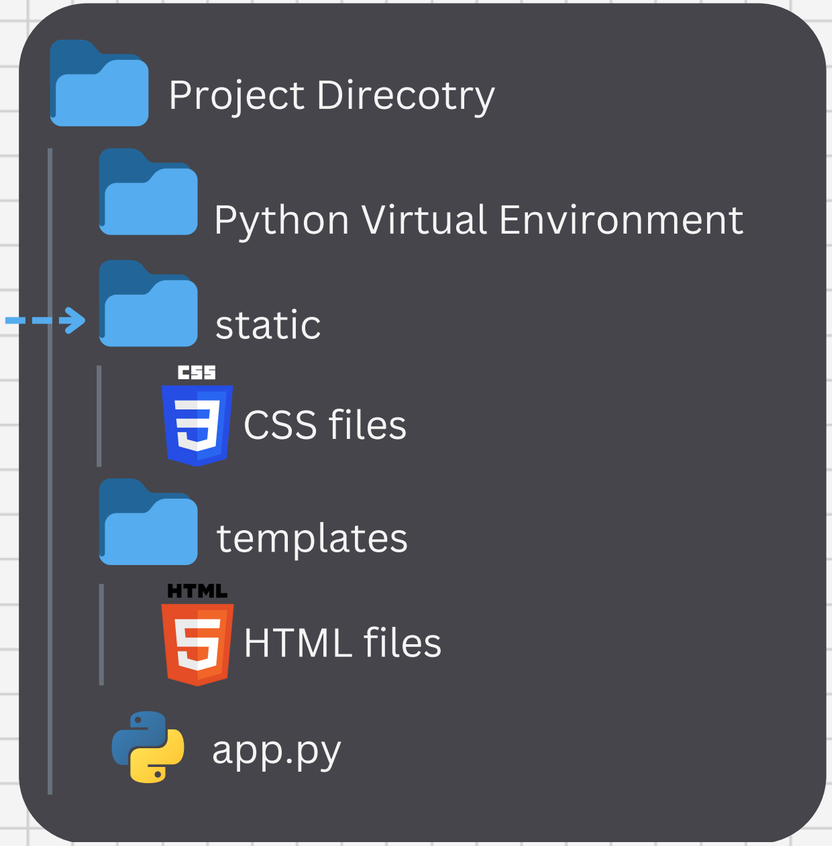
Display flash messages

```
{% block content %}
    {% with messages = get_flashed_messages() %}
        {% if messages %}
            {% for msg in messages %}
                <p class="error-message">{{msg}}</p>
            {% endfor %}
        {% endif %}
    {% endwith %}
{% endblock %}
```

Front-end

part of a web application or website that users interact with directly.

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="{{ url_for('static', filename='styles.css') }}">
    <title>Home</title>
</head>
```



Flask Demo

Your Turn!

Create a simple sign-up page and display Hello “username” after the user signs up

app.py > ...

```python
from flask import Flask, request, render_template, session, redirect, url_for

app = Flask(__name__)
app.secret_key = "BAD_SECRET_KEY"

@app.route("/", methods=["GET", "POST"])
def home():
    if request.method == "POST":
        user_name = request.form.get("u_name")
        pswd = request.form.get("pswd")
        if user_name and pswd:
            session['user_name'] = user_name
            return redirect(url_for("login"))
    return render_template("index.html")


@app.route("/login")
def login():
    user_name = session['user_name']
    return render_template("hello.html", user_name=user_name)
```

templates > index.html > ...

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="{{url_for('static', filename='styles.css')}}">
    <title>Home</title>
</head>
<body>
    <form method="post">
        <h1>Username</h1>
        <input type="text" name="u_name" placeholder="username" required>
        <h1>Password</h1>
        <input type="password" name="pswd" placeholder="passwword" required>
        <button type="submit">Sign-up</button>
    </form>
</body>
</html>
```

templates > hello.html > ...

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="{{url_for('static', filename='styles.css')}}">
    <title>Welcome</title>
</head>
<body>
    <h1>Hello {{user_name}}</h1>
</body>
</html>
```

static > styles.css > ...

```css
body {
    font-family: Arial, sans-serif;
    background-color: #1e1e1e;
    color: #f0f0f0;
    display: flex;
    justify-content: center;
    align-items: center;
    height: 100vh;
    margin: 0;
}

form {
    background-color: #2a2a2a;
    padding: 20px;
    border-radius: 8px;
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.5);
    width: 40%;
    text-align: center;
}

h1{
    font-size: 1.5rem;
}
input{
    padding: 10px;
    margin-top: 10px;
    margin-bottom: 10px;
    border: 0px;
    border-radius: 100px;
    background-color: #3a3a3a;
    color: #f0f0f0;
    width: 95%;
    text-align: center;
}


button {
    background-color: #007bff;
    color: #ffffff;
    border: none;
    padding: 10px;
    border-radius: 4px;
    cursor: pointer;
    width: 100%;
    margin-top: 20px;
}

button:hover {
    background-color: #0056b3;
}
```

Feedback for the Tech Workshop 2

Thank you!