## Web application – Tutorial

### Step 1: Set up the project

1. Login to GitHub and open your ‘Programming for the web’ repository.
2. Create a new directory called ‘file-web-app’ with a file called ‘app.py’.
3. Select **Commit changes** to create the directory and file.
4. Open a **codespace** and then open the new ‘app.py’ file.
5. When prompted, install the ‘Python extension…’. It is also a good idea to install ‘Pylint’ from the ‘Extensions’ menu.
6. Finally, install the Flask (web application framework) by entering the following into the terminal:

pip install flask

### Step 2: Set up the backend service

1. Begin by importing the required Flask module and the OS library.

import os

from flask import Flask, request, render\_template

**Explainer**

* os: Handles file paths and folder creation.
* Flask: The main web framework.
* request: Handles incoming HTTP requests (such as file uploads).
* render\_template: Renders HTML templates.

1. Next, we initialise the Flask app and define a folder to store file uploads.

app = Flask(\_\_name\_\_)

UPLOAD\_FOLDER = "uploads"

os.makedirs(UPLOAD\_FOLDER, exist\_ok=True)

**Explainer**

* app = Flask(\_\_name\_\_): Creates the Flask application.
* UPLOAD\_FOLDER = "uploads": Defines the folder where uploaded files will be stored.
* os.makedirs(UPLOAD\_FOLDER, exist\_ok=True): Ensures the upload folder exists (creates it if missing).

1. Now define the route (location of the main page) and begin a function to handle file uploads

@app.route("/", methods=["GET", "POST"])

def upload\_file():

**Explainer**

* @app.route("/"): Defines the main route (/).
* methods=["GET", "POST"]: Allows both viewing the upload form (GET) and submitting a file (POST).
* def upload\_file(): Creates a user-defined function to hold our code.

1. The function will hold the instructions to handle the file uploads.

if request.method == "POST":

file = request.files["file"]

file\_path = os.path.join(UPLOAD\_FOLDER, file.filename)

file.save(file\_path)

**Explainer**

* if request.method == "POST":: Runs when a file is submitted.
* file = request.files["file"]: Retrieves the uploaded file.
* file\_path = os.path.join(UPLOAD\_FOLDER, file.filename): Defines where to save the file.
* file.save(file\_path): Saves the uploaded file to the server.

1. With the file uploaded, we can now read and display the file’s content.

with open(file\_path, "r", encoding='utf-8') as f:

content = f.read()

return f"<h3>File Content:</h3><pre>{content}</pre>"

**Explainer**

* Opens the saved file in read mode ("r").
* Reads the content and displays it in the browser inside <pre> tags.

1. Finally, let’s consider if no file is uploaded for rendering.

return render\_template("index.html")

**Explainer**

* If no file is uploaded, it renders index.html, which contains the file upload form.

1. All that remains is to call the app.

if \_\_name\_\_ == "\_\_main\_\_":

app.run(debug=True)

**Explainer**

* if \_\_name\_\_ == "\_\_main\_\_":: Ensures the script runs only when executed directly.
* app.run(debug=True): Runs the server in debug mode, allowing automatic reloading on changes.

### Step 3: Set up the frontend page

1. The Flash framework app expects the HTML documents for the frontend to be in a folder called ‘templates’:
   1. Select the ‘file-web-app’ folder and then **New Folder…**
   2. Name the folder ‘templates’
2. For simplicity, students can copy the following code from the class OLE page into a new file called ‘index.html’.

<!DOCTYPE html>

<html lang="en">

<head>

<title>Upload file</title>

</head>

<body>

<h2>Upload a text file (.txt)</h2>

<form action="/" method="post" enctype="multipart/form-data">

<input type="file" name="file">

<button type="submit">Upload</button>

</form>

</body>

</html>

**Explainer**

* <h2>: Displays a heading instructing the user to upload a text file.
* <form>: Defines an HTML form where users can submit data.
* action="/": Specifies that the form data will be sent to the root URL (/) of the server.
* method="post": Specifies that the form will use POST (not GET) to submit data.
* enctype="multipart/form-data": Required for uploading files. It ensures that files are sent properly instead of being treated as plain text.
* <input>: Creates an input field.
* type="file": Allows users to browse and select a file from their device.
* name="file": Assigns a name to the uploaded file, which the Flask back-end will use to reference it (request.files["file"]).
* <button>: A clickable button that submits the form.
* type="submit": Ensures the button submits the form when clicked.

### Finished code

**Python**

"""Basic web app to upload and process a file"""

import os

from flask import Flask, request, render\_template

app = Flask(\_\_name\_\_)

UPLOAD\_FOLDER = "uploads"

os.makedirs(UPLOAD\_FOLDER, exist\_ok=True)

@app.route("/", methods=["GET", "POST"])

def upload\_file():

"""Handles file uploads"""

if request.method == "POST":

file = request.files["file"]

file\_path = os.path.join(UPLOAD\_FOLDER, file.filename)

file.save(file\_path)

with open(file\_path, "r", encoding='utf-8') as f:

content = f.read()

return f"<h3>File Content:</h3><pre>{content}</pre>"

return render\_template("index.html")

if \_\_name\_\_ == "\_\_main\_\_":

app.run(debug=True)

**HTML**

Note: This code must be in the templates sub-folder.

<!DOCTYPE html>

<html lang="en">

<head>

<title>Upload file</title>

</head>

<body>

<h2>Upload a text file (.txt)</h2>

<form action="/" method="post" enctype="multipart/form-data">

<input type="file" name="file">

<button type="submit">Upload</button>

</form>

</body>

</html>

## Bibliography

* OpenAI. (2024). *ChatGPT* (GPT-4o) [Large language model]. <https://chat.openai.com>