



Temasek Junior College

2024 JC2 H2 Computing

Web Applications 10 – Handling File and Image Uploads

Section	4	Computer Networks
Unit	4.2	Web Applications
Objectives	4.2.3	Use HTML, CSS (for clients) and Python (for the server) to create a web application that is able to: <ul style="list-style-type: none"> - accept user input (text and image file uploads) - process the input on the local server - store and retrieve data using an SQL database - display the output (as formatted text/ images/table)
	4.2.4	Test a web application on a local server.

1 File Uploading

Recall the following methods of the **request** object:

Attribute	request.args	request.form	request.files
Contents	Dictionary of field names and their associated values from query portion of URL	Dictionary of field names and their associated values	Dictionary of file upload names and their associated FileStorage objects
Applicable HTTP request	Usually GET ¹	POST only	POST only
Typical Use	Access form data submitted using GET	Access form data submitted using POST	Handle files submitted using POST
Encoding Enforcement			enctype attribute of <form> tag must be specified as "multipart/form-data"

Observe from the table that for file uploads to work properly, the **method** attribute of the **<form>** tag must be configured to **POST**. In addition, an **enctype** attribute that is set to **multipart/form-data** must also be include. This attribute indicates that one or more sets of data are combined in a single body.

By setting the **type** attribute of the **<input>** tag to **file**, a HTML form can be made to receive file uploads for submission.

¹ Also works with **POST** requests if the URL has query portion

Exercise 1A

Create the HTML template, `form_with_file_upload.html` to create a form for uploading photos as well as a link for seeing all the photos that have been uploaded

```

1  <!DOCTYPE html>
2  <html>
3
4      <head><title>Photo Upload</title></head>
5
6      <body>
7
8          <form method="post" enctype="multipart/form-data">
9              <p>Photo: <input name="photo" type="file"></p>
10             <p><input type="submit"></p>
11         </form>
12
13         <p><a href="{{ url_for('view') }}">View photos</a></p>
14
15     </body>
16
17 </html>

```

Exercise 1B

Create the HTML template, `view_file_uploads.html` for a page to view the photos uploaded when the link to view these photos in **Exercise 1A** is clicked.

```

1  <!DOCTYPE html>
2  <html>
3
4      <head>
5          <title>View Photos</title>
6      </head>
7
8      <body>
9
10         {% for photo in photos %}
11             
13         {% endfor %}
14
15         <p><a href="{{ url_for('home') }}">Home</a></p>
16
17     </body>
18
19 </html>

```

For the main Flask server application, we shall create three routes:

- one for uploading photos,
- one for seeing all the uploaded photos
- one for retrieving the image data of an uploaded photo given its filename.

We shall also initialize and use a simple SQLite database to keep track of the photos uploaded:

Exercise 1C

Create a Flask code `server.py` for the photo uploading and viewing application.

```

1  import flask, os, sqlite3
2  from flask import render_template, request
3  from flask import send_from_directory
4  from werkzeug.utils import secure_filename
5
6  if not os.path.isfile('db.sqlite3'):
7      db = sqlite3.connect('db.sqlite3')
8      db.execute('CREATE TABLE photos(photo TEXT)')
9      db.commit()
10     db.close()
11
12 app = flask.Flask(__name__)
13
14 @app.route('/', methods=['GET', 'POST'])
15 def home():
16     if request.method == 'POST' and \
17         request.files and 'photo' in request.files:
18         # Save file
19         photo = request.files['photo']
20         filename = secure_filename(photo.filename)
21         path = os.path.join('uploads', filename)
22         photo.save(path)
23         # Add filename to database
24         db = sqlite3.connect('db.sqlite3')
25         db.execute('INSERT INTO photos(photo) VALUES(?)',
26                     (filename,))
27         db.commit()
28         db.close()
29         return render_template('form_with_file_upload.html')
30
31 @app.route('/view')
32 def view():
33     db = sqlite3.connect('db.sqlite3')
34     cur = db.execute('SELECT photo FROM photos')
35     photos = []
36     for row in cur:
37         photos.append(row[0])
38     db.close()
39     return render_template('view_file_uploads.html',
40                             photos=photos)
41
42 @app.route('/photos/<filename>')
43 def get_file(filename):
44     return send_from_directory('uploads', filename)
45
46 if __name__ == '__main__':
47     app.run()

```

Before running the program, create an empty **uploads** subfolder (same level as **templates** and **static**) to store the uploaded photos.

Next, run the program and visit **`http://127.0.0.1:5000/`** to upload some photos (i.e., GIF, JPG or PNG files). When done, click on the **"View"** link to see the uploaded photos.

Important Notes on File Uploading

- If you try to upload non-photo files such as PDF files, the file will still be uploaded but will not be displayed properly.
- Unlike normal form data, submitted files are not accessed from **`request.form`** but from a separate **`request.files`** dictionary (e.g. line **19**).
 - Each value in this dictionary is a file object with a **`filename`** attribute as well as a **`save()`** method that accepts a file path and writes the submitted file onto the server's file system using the provided file path.
- Be careful when letting users specify filenames for reading or writing files on the server's file system as file paths can use special folder names such as **`..`** to access parent folders that contain source code or your server's configuration files.
 - To prevent this from happening, pass the filename through the **`secure_filename()`** function provided in the **`werkzeug.utils`** module first (e.g. line **20**). This function returns a modified filename with all special characters replaced so it can be safely treated like a normal filename. This modified filename is then used to form a file path that is guaranteed to be in the **`uploads`** subfolder (e.g. line **21**).
- To actually view uploaded photos, two routes are needed:
 - one for generating the HTML document that will display all the photos on a single page.
 - one for accessing each photo's file data.
- For the second route, use **`send_from_directory()`** (e.g. line **44**) to avoid the same security issues that come from using paths or filenames provided by users.
 - To use **`send_from_directory()`**, call it with the name of a subfolder that the requested file must be stored in as well as the file's filename. Then return the result of this call directly.
- As an alternative to configuring a new route to access each uploaded photo, the uploads can be saved instead in the **`static`** subfolder.
 - The existing **`'static'`** route that Flask provides by default can then be used.
 - This however requires care to not let users overwrite files erroneously.

[Challenge] Can you edit the program in **Exercise 1C** to only allow files with extensions GIF, JPG or PNG to be uploaded?