Capítulo 2

Framework Flask

Flask

- Framework liviano para desarrollo de aplicaciones web WSGI
- ¿WSGI?
  - Web Server Gateway Interface:
    - https://peps.python.org/pep-3333/
  - Especificación de interfaces, define como se comunica una aplicación web Python y un servidor web
- Flask:
  - Diseñado para comenzar desarrollos de forma rápida y fácil
  - Puede escalar hasta aplicaciones complejas
  - https://palletsprojects.com/p/flask/

Flask

- No impone dependencias
  - Desarrollador puede elegir bibliotecas y herramientas que desea usar
  - Existen muchas extensiones provistas por la comunidad
- Documentación
  - https://flask.palletsprojects.com/

Flask

- Instalación
  - Flask soporta Python versión 3.7+
  - Incluye estas dependencias:
    - Werkzeug: implementa WSGI (interfaz estándar de Python entre aplicaciones y servidores)
    - Jinja: lenguaje para plantillas que “dibuja” las páginas de la aplicación
    - MarkupSafe: viene con Jinja, limpia entradas de información para evitar ataques de inyección
    - ItsDangerous: firma segura sobre los datos para asegurar la integridad
    - Click: (Command Line Interface Creation Kit) framework para escribir aplicaciones de línea de comandos

Flask

- Instalación
  - Usar un ambiente virtual para la instalación

```
$ mkdir myproject
$ cd myproject
$ python3 -m venv venv
```

  - Activar e instalar:

```
$ . venv/bin/activate
```

```
$ pip install Flask
```

Flask

- Hello world:

```python
from flask import Flask

app = Flask(__name__)

@app.route("/")
def hello_world():
    return "<p>Hello, World!</p>"
```

  - Almacenar el archivo como `hello.py` y ejecutar:

```
$ flask --app hello run
 * Serving Flask app 'hello'
 * Running on http://127.0.0.1:5000 (Press CTRL+C to quit)
```

Flask

```python
# save this as app.py
from flask import Flask, request
from markupsafe import escape

app = Flask(__name__)

@app.route('/')
def hello():
    name = request.args.get("name", "World")
    return f'Hello, {escape(name)}!'
```

```
$ flask run
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Flask

- Variables
  - Permite agregar variables con `<nombre variable>`
    - La función recibe el nombre de variable como argumento
  - Se pueden usar “convertidores” para especificar el tipo de variable

`string`	(default) accepts any text without a slash
`int`	accepts positive integers
`float`	accepts positive floating point values
`path`	like `string` but also accepts slashes
`uuid`	accepts UUID strings

Flask

```python
from markupsafe import escape

@app.route('/user/<username>')
def show_user_profile(username):
    # show the user profile for that user
    return f'User {escape(username)}'

@app.route('/post/<int:post_id>')
def show_post(post_id):
    # show the post with the given id, the id is an integer
    return f'Post {post_id}'

@app.route('/path/<path:subpath>')
def show_subpath(subpath):
    # show the subpath after /path/
    return f'Subpath {escape(subpath)}'
```

Flask

- URL únicas y redireccionamiento

```
@app.route('/projects/')
def projects():
    return 'The project page'

@app.route('/about')
def about():
    return 'The about page'
```

  - Permite acceder a `/projects` y será redirigido a URL `/projects/`
  - `/about` no tiene "/" final, si se accede con "`/about/`" generará error 404 (Not Found)

Métodos HTTP

- Las aplicaciones web pueden usar distintos métodos:
  - De forma predeterminada `@app.route` responde solo solicitudes `GET`
  - Se puede agregar el argumento `methods` para soportar otros tipos de solicitudes

```python
from flask import request

@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        return do_the_login()
    else:
        return show_the_login_form()
```

Métodos HTTP

- Se puede separar las vistas para diferentes métodos en funciones:
  - Flask provee rutas con `get()`, `post()`, etcétera.

```python
@app.get('/login')
def login_get():
    return show_the_login_form()

@app.post('/login')
def login_post():
    return do_the_login()
```

  - Si usa `GET`, se agrega soporte automático para `HEAD`. También se agrega `OPTIONS`

Archivos estáticos

- Normalmente tenemos archivos estáticos en las aplicaciones web:
  - Hojas de estilo (CSS)
  - Javascript
  - Imágenes
  - Documentos
  - Etcétera.
- Al usar Flask solo se deben dejar en un directorio llamado `static/`

Plantillas

- Jinja2 ofrece manejo de plantillas
- Plantillas pueden generar diversos tipos de texto:
  - Documentos HTML
  - Texto de emails
- Para usar una plantilla se usa el método `render_template()`