Flask

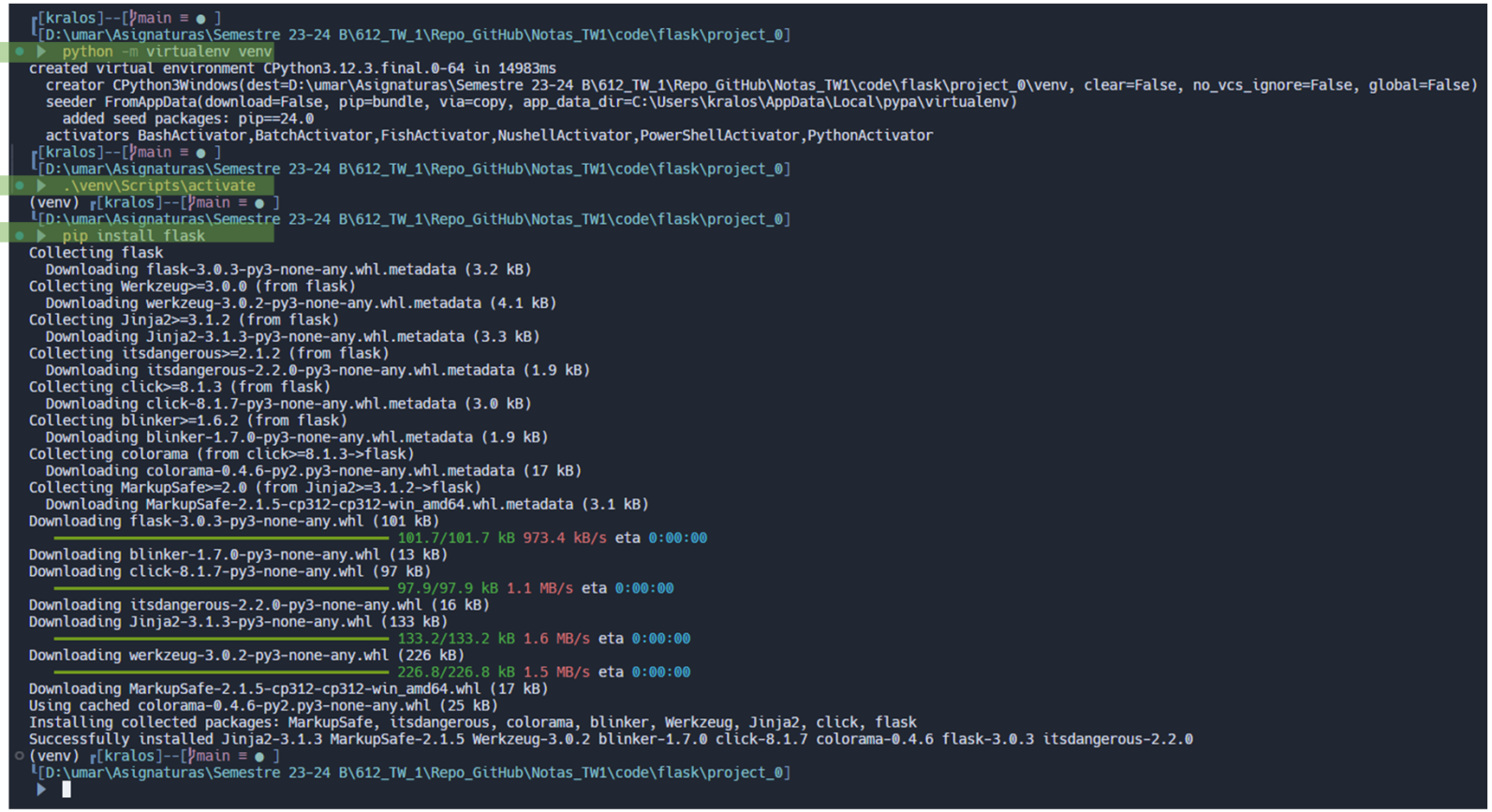
Framework de Python

¿Qué es Flask?

- Framework web creado por el desarrollador austriaco Armin Ronacher en 2010.
  - Flask es un microframework para Python basado en Werkzeug que permite crear aplicaciones web de todo tipo rápidamente.
- Flask solo incluye el motor de plantillas Jinja y una biblioteca llamada “tool”. Sin embargo, ofrece la posibilidad de integrar funciones de terceros.
- Flask está bajo una licencia BSD. Es gratuito y de código abierto.

Primera aplicación Flask

- Preparando el entorno de programación



Primera aplicación Flask

- Preparando el entorno de programación

```
pip freeze
blinker==1.7.0
click==8.1.7
colorama==0.4.6
Flask==3.0.3
itsdangerous==2.2.0
Jinja2==3.1.3
MarkupSafe==2.1.5
Werkzeug==3.0.2
```

Primera aplicación Flask

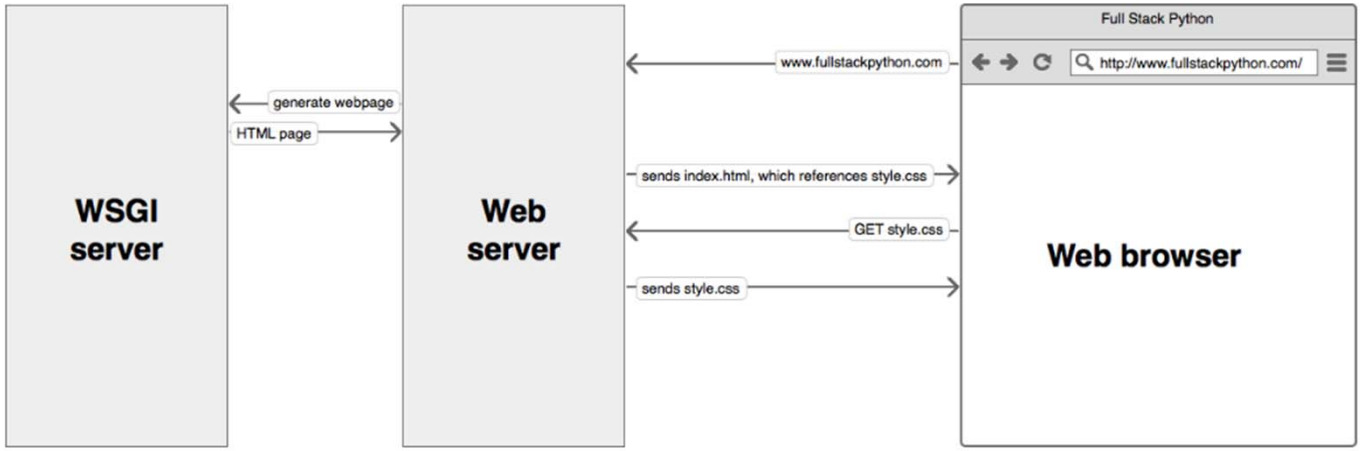
- Creando la primera aplicación Flask

```
1.  from flask import Flask
2.  app = Flask(__name__)
3.
4.  @app.route('/')
5.  def hello_world():
6.      return 'Hello, World!'
```

- Toda aplicación Flask es una instancia WSGI de la clase Flask. Por tanto, importamos dicha clase y creamos una instancia que en este caso he llamado app.

Primera aplicación Flask

- ¿WSGI?
- Significa Web Server Gateway Interface, y es una especificación estándar sobre cómo deben interactuar los servidores web y las aplicaciones web de Python



Primera aplicación Flask

- Una característica de Flask es que tendremos métodos asociados a las distintas URLs que componen nuestra aplicación.
- Es en estos métodos donde ocurre toda la lógica que queramos implementar. Dentro del patrón MVC, esta parte del código se correspondería con el controlador.
- Flask se encarga de hacernos transparente el cómo a partir de una petición a una URL se ejecuta finalmente nuestra rutina.
- Lo único que tendremos que hacer nosotros será añadir un decorador a nuestra función. En nuestro caso, hemos llamado a nuestra función hello_world que será invocada cada vez que se haga una petición a la URL raíz de nuestra aplicación.

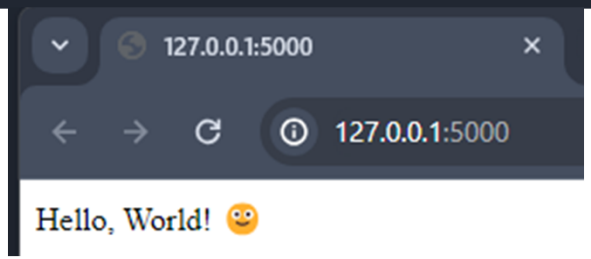
Primera aplicación Flask

- El decorador route de la aplicación (app) es el encargado de decirle a Flask qué URL debe ejecutar su correspondiente función.
- El nombre que le demos a nuestra función será usado para generar internamente URLs a partir de dicha función.
- La función debe devolver la respuesta que será mostrada en el navegador del usuario.

Primera aplicación Flask

- Probando la aplicación Flask
- Flask viene con un servidor interno que nos facilita mucho la fase de desarrollo. **No debemos usar este servidor en un entorno de producción ya que no es este su objetivo.**

```
(venv) ┌[kralos]--[ main ≡ ● ]
└[D:\umar\Asignaturas\Semestre 23-24 B\612_TW_1\Repo_GitHub\Notas_TW1\code\flask\project_0]
  ▶  python -m flask run
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
127.0.0.1 - - [25/Apr/2024 10:17:27] "GET / HTTP/1.1" 200 -
```



Uso de plantillas para las páginas HTML

- Routing
- La manera muy rápida cómo Flask asocia una URL con un método de nuestro código. Para ello, simplemente tenemos que añadir el decorador route() a la función que queramos ejecutar cuando se hace una petición a una determinada URL.
- En Flask, por convención, a las funciones que están asociadas a una URL se les llama «vistas».

Uso de plantillas para las páginas HTML

- Creando la vista de la página home

```
1.  posts = []
2.
3.  @app.route("/")
4.  def index():
5.      return "{} posts".format(len(posts))
```

- La variable posts es una lista que almacenará los posts que vayamos creando.
- En segundo lugar hemos creado la función index(), que es la responsable de mostrar los posts. Lo único que hace es mostrar en el navegador el número de posts que contiene la variable posts.

Uso de plantillas para las páginas HTML

- Creando la vista que muestra el detalle de un post

- Para hacer esto, a una URL le podemos añadir secciones variables o parametrizadas con <param>.

- La vista recibirá <param> como un parámetro con ese mismo nombre. Opcionalmente se puede indicar un conversor (converter) para especificar el tipo de dicho parámetro así <converter:param>.

Uso de plantillas para las páginas HTML

- Por defecto, en Flask existen los siguientes conversores:

  - **string**: Es el conversor por defecto. Acepta cualquier cadena que no contenga el carácter ‘/’.
  - **int**: Acepta números enteros positivos.
  - **float**: Acepta números en punto flotante positivos.
  - **path**: Es como string pero acepta cadenas con el carácter ‘/’.
  - **uuid**: Acepta cadenas con formato UUID.

Uso de plantillas para las páginas HTML

- **Crear una vista para mostrar un post a partir del slug del título del mismo**. Un slug es una cadena de caracteres alfanuméricos (más el carácter '-'), sin espacios, tildes ni signos de puntuación.

```
1.  @app.route("/p/<string:slug>/")
2.  def show_post(slug):
3.      return "Mostrando el post {}".format(slug)
```

- En esta vista se definió el parámetro slug en la URL y dicho parámetro se toma como argumento de la función show_post(slug).

Uso de plantillas para las páginas HTML

- Creando la vista para crear/modificar un post
  - Prácticamente lo que hay que hacer en ambos casos es lo mismo: recuperar los datos de un formulario, crear un objeto post, asignarle los valores de los campos del formulario y guardar el objeto post. La única diferencia es que modificar un post supone recuperarlo previamente de la base de datos.

```
1.  @app.route("/admin/post/")
2.  @app.route("/admin/post/<int:post_id>/")
3.  def post_form(post_id=None):
4.      return "post_form {}".format(post_id)
```

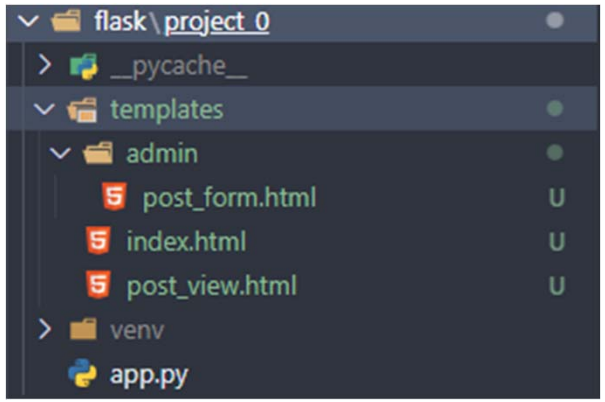
  - Para asociar dos URLs diferentes a una sola vista. Simplemente añadiendo dos decoradores a la misma función.

Uso de plantillas para las páginas HTML

- **Renderizando una página HTML**
- Flask trae por defecto un motor de renderizado de plantillas llamado Jinja2 que te ayudará a crear las páginas dinámicas de tu aplicación web.
- Para renderizar una plantilla creada con Jinja2 simplemente hay que hacer uso del método render_template(). A este método debemos pasarle el nombre de nuestra plantilla y las variables necesarias para su renderizado como parámetros clave-valor.

Uso de plantillas para las páginas HTML

- **Renderizando una página HTML**
- Flask buscará las plantillas en el directorio templates de nuestro proyecto. En el sistema de ficheros, este directorio se debe encontrar en el mismo nivel en el que hayamos definido nuestra aplicación.



Uso de plantillas para las páginas HTML

- **Renderizando una página HTML**
- Ahora modifiquemos el cuerpo de las vistas index(), show_post() y post_form() para que muestren el resultado de renderizar las respectivas plantillas. Pero antes recuerda importar el método render_template() del módulo flask:

```
from flask import render_template
```

Uso de plantillas para las páginas HTML

```python
@app.route("/")
def index():
    return render_template("index.html", num_posts=len(posts))


@app.route("/p/<string:slug>/")
def show_post(slug):
    return render_template("post_view.html", slug_title=slug)


@app.route("/admin/post/")
@app.route("/admin/post/<int:post_id>/")
def post_form(post_id=None):
    return render_template("admin/post_form.html", post_id=post_id)
```

index.html

```html
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>Tutorial Flask: Miniblog</title>
</head>
<body>
    {{ num_posts }} posts
</body>
</html>
```

post_view.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>{{ slug_title }}</title>
</head>
<body>
    Mostrando el post {{ slug_title }}
</body>
</html>
```

post_form.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>
        {% if post_id %}
            Modificando el post {{ post_id }}
        {% else %}
            Nuevo post
        {% endif %}
    </title>
</head>
<body>
{% if post_id %}
    Modificando el post {{ post_id }}
{% else %}
    Nuevo post
{% endif %}
</body>
</html>
```

Uso de plantillas para las páginas HTML

- La excepción de {{ num_posts }} y {{ slug_title }} y los caracteres {% y %}. Dentro de las llaves se usan los parámetros que se pasaron al método render_template(). El resultado de ello es que durante el renderizado se sustituirán las llaves por el valor de los parámetros. De este modo podemos generar contenido dinámico en nuestras páginas.

Uso de plantillas para las páginas HTML

- Jinja2 está basado en Python, así que casi todo lo que conoces de este lenguaje es aplicable al lenguaje utilizado en Jinja2.

  - Cualquier expresión contenida entre llaves dobles se mostrará como salida al renderizarse la página.

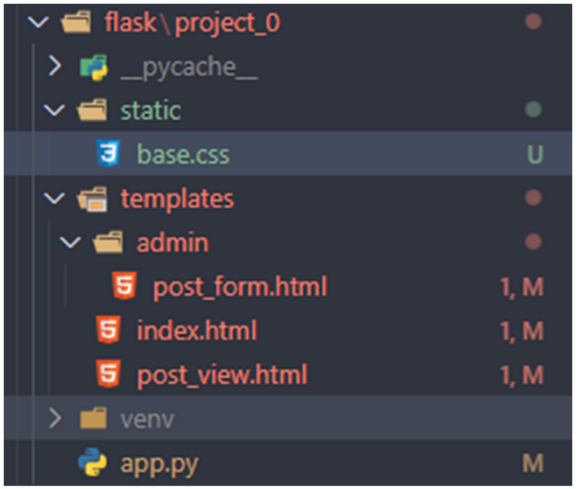
  - Es posible usar estructuras de control, como sentencias if o bucles for, entre los caracteres {% y %}

Ficheros estáticos

- Una página web también se compone de ficheros CSS para definir estilos, imágenes y código javascript. Todo este tipo de ficheros se conocen como recursos estáticos, ya que su contenido no cambia a lo largo del ciclo de ejecución de una aplicación web.
- Normalmente, en un entorno real, estos ficheros deben ser servidos por un servidor web como puede ser Apache o Nginx y no por un servidor de aplicaciones.

Ficheros estáticos

- Para que el código CSS o Javascript sea servido, debemos ubicar estos ficheros en un directorio llamado static situado al mismo nivel que el directorio templates. Este directorio estará accesible en la URL /static .



Ficheros estáticos

- Dentro del directorio templates crea un fichero llamado base_template.html y pega el siguiente código:

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>{% block title %}{% endblock %}</title>
    <link rel="stylesheet" href="{{ url_for("static", filename="base.css") }}">
</head>
<body>
{% block content %}{% endblock %}
</body>
</html>
```

- El método url_for() para componer una URL a partir del nombre de una vista.

Ficheros estáticos

index.html

```
{% extends "base_template.html" %}

{% block title %}Tutorial Flask: Miniblog{% endblock %}

{% block content %}
    {{ num_posts }} posts
{% endblock %}
```

post_view.html

```
{% extends "base_template.html" %}

{% block title %}{{ slug_title }}{% endblock %}

{% block content %}
    Mostrando el post {{ slug_title }}
{% endblock %}
```

post_form.html

```
{% extends "base_template.html" %}

{% block title %}
    {% if post_id %}
        Modificando el post {{ post_id }}
    {% else %}
        Nuevo post
    {% endif %}
{% endblock %}

{% block content %}
    {% if post_id %}
        Modificando el post {{ post_id }}
    {% else %}
        Nuevo post
    {% endif %}
{% endblock %}
```

Referencias

- j2logo. (2019, February 25). *Tutorial de Flask en español: Desarrollando una aplicación web en Python*. J2LOGO. https://j2logo.com/tutorial-flask-espanol/
- *¿Qué es Flask Python? Un breve tutorial sobre este microframework*. (2023, March 1). IONOS Digital Guide; IONOS. https://www.ionos.mx/digitalguide/paginas-web/desarrollo-web/flask/