Práctica 3: Desarrollo de una aplicación Web con Flask (I)

Preparando tu entorno de desarrollo

Para desarrollar esta práctica y las siguientes, debes preparar tu entorno de trabajo. Necesitarás instalar en tu ordenador:

> Un intérprete de Python reciente.
>
> Un entorno de desarrollo como, por ejemplo, Visual Studio Code.
>
> Un gestor de bases de datos MySQL o MariaDB.

Todos ellos se distribuyen con licencia de código libre, son gratuitos y están disponibles para Windows, Linux y MacOS.

En los enunciados se proporcionan por defecto los comandos para el intérprete de línea de comandos BASH (Linux y MacOS). Adicionalmente, se proporcionan los comandos equivalentes para el intérprete de línea de comandos CMD de Windows.

Ejercicio 1
Desplegando la aplicación base

La aplicación que desarrollarás en los siguientes laboratorios funcionará sobre el framework de aplicaciones web Flask. Descarga y descomprime el fichero **`microblog-app.tgz`**, que contiene el esqueleto de la aplicación de microblogging que desarrollaremos en los siguientes laboratorios.

El primer paso será instalar Flask y otra biblioteca dentro de un entorno virtual de Python local. Abre un terminal de línea de comandos y entra en el directorio **`microblog-app`** que acabas de descomprimir. Ejecuta en él los siguientes comandos:

```
python3 -m venv venv
. venv/bin/activate
```

```
pip install -U pip
pip install Flask python-dateutil
```

Si estás en Windows:

```
py -3 -m venv venv
venv\Scripts\activate
pip install -U pip
pip install Flask python-dateutil
```

A partir de ahora, cada vez que necesites trabajar en este proyecto, deberás abrir un terminal de nuevo, asegurarte de que estás ejecutando BASH y activar el entorno virtual desde el directorio **`microblog-app`**:

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

En Windows:

```
venv\Scripts\activate
```

Ahora, ejecuta la aplicación dentro del servidor de desarrollo que incluye Flask:

```
export FLASK_APP=microblog
flask --debug run
```

En Windows:

```
set FLASK_APP=microblog
flask --debug run
```

Ya puedes acceder a la aplicación abriendo la URL local http://localhost:5000/ en un navegador web. Deberías ver una página muy sencilla que sólo muestra dos mensajes de prueba.

Puedes detener el servidor en cualquier momento con la combinación de teclas Control+C en el terminal donde estás ejecutando el comando **`flask --debug run`**.

Dado que has lanzado el servidor en modo *desarrollo*, la aplicación se recargará cada vez que cambies algo en el código fuente de tu aplicación.

Ejercicio 2
Analizando la aplicación

Abre el directorio **`microblog-app`** en tu entorno de desarrollo. Por ejemplo, si usas Visual Studio Code, entra en ese directorio y ejecuta (o simplemente ejecuta Visual Studio Code y selecciona la entrada de menú *File / Open Folder*):

```
code . &
```

Encontrarás el código fuente de la aplicación en el subdirectorio **`microblog`**. El primer fichero de la aplicación que puedes mirar es **`microblog/__init__.py`**:

```
from flask import Flask

from . import main

def create_app(test_config=None):
    app = Flask(__name__)
    app.config["SECRET_KEY"] = b"\x8c\xa5\x04\xb3\x8f\xa1<\xef\x9bY\xca/*\xff\x12\xfb"
    app.register_blueprint(main.bp)
    return app
```

Esta clase crea la aplicación web y registra un *blueprint*. Los *blueprints* de Flask son una forma de dividir una aplicación en módulos, para que esta sea más fácil de mantener.

El código de ese *blueprint* está en el módulo **`microblog/main.py`**. Como puedes ver, define un controlador para la ruta raíz (es decir, para la ruta **`/`**). Cuando llega una petición para la ruta **`/`** se llamará a la función llamada **`index`**:

```
@bp.route("/")
def index():
    user = model.User(1, "mary@example.com", "mary")
    posts = [
        model.Message(
            1, user, "Test post", datetime.datetime.now(dateutil.tz.tzlocal())
        ),
        model.Message(
            2, user, "Another post", datetime.datetime.now(dateutil.tz.tzlocal())
        ),
    ]
    return render_template("main/index.html", posts=posts)
```

Este controlador crea, a modo de demostración porque la aplicación aún no está conectada a una base de datos, un nuevo usuario y dos mensajes (mira el código de las clases **`User`** y **`Message`** en el módulo **`microblog.model`**). Después, el programa llama a la plantilla **`main/index.html`** y pasa como parámetro una lista con estos dos mensajes.

La respuesta HTTP contendrá el resultado de aplicar la plantilla. Echa un vistazo a esta plantilla, que está almacenada en **`microblog/templates/main/index.html`**:

```
{% extends 'base.html' %}

{% block content %}
    <section>
        <h2>Latest posts from the people you follow</h2>
        {% include 'posts_template.html' %}
    </section>
{% endblock %}
```

Esta plantilla se basa en el lenguaje de plantillas Jinja. Como puedes ver, no contiene demasiado código HTML. La razón es que se basa en otras dos plantillas. Por una parte, extiende **`base.html`**:

```
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Microblogging app</title>
        <link rel="stylesheet"
            href="{{ url_for('static', filename='microblog.css') }}">
    </head>
    <body>
        <header>
            <h1>Microblogging</h1>
            <a href="{{ url_for('main.index') }}">Home</a>
        </header>
        <section class="content">
            {% block content %}{% endblock %}
        </section>
        <footer><small>Developed for Web applications</small></footer>
    </body>
</html>
```

Cada vista de nuestra aplicación extenderá esta plantilla, de forma que todas ellas compartan la misma estructura, sin necesidad de repetir el mismo código una y otra vez.

Por otra parte, dado que son varias las vistas que necesitan mostrar una lista de mensajes, dicho código se ha movido a la plantilla **`posts_template.html`**:

```
<div class="messages">
    {% for post in posts %}
        {% include 'post_template.html' %}
    {% endfor %}
</div>
```

Contiene un bucle que muestra los mensajes uno a uno, cada uno siguiendo la plantilla **`post_template.html`**:

```
<div class="message">
    <div class="text">{{ post.text}}</div>
    <div class="metadata">
        <span class="author">{{ post.user.name }}</span>
        <span class="date">{{ post.timestamp }}</span>
    </div>
</div>
```

Modifica el controlador para crear y añadir a la lista un tercer mensaje. La aplicación debería reiniciarse automáticamente cuando guardes el código modificado. Si no es así, simplemente para la aplicación (Control+C en el terminal donde la estés ejecutando) y vuelve a iniciarla. Recarga la página en el navegador y comprueba que ahora también se muestra el mensaje que acabas de crear.

Ejercicio 3

Plantilla de la vista principal

Integra la vista principal que has creado en el laboratorio 2, junto con su hoja de estilos CSS, en las plantillas del ejercicio anterior, de forma que los mismos tres mensajes se presenten con la apariencia que tendrán en tu aplicación.

Almacena los ficheros estáticos, como las hojas de estilos CSS o las imágenes, bajo el directorio **`microblog/static`**. Su ruta comenzará por **`/static`** (mira, por ejemplo, cómo ya se está incluyendo la hoja de estilos CSS de la aplicación base en **`base.html`**).

Ejercicio 4
Plantilla de la vista del perfil público de usuario

De forma similar a la vista principal, crea un controlador y una plantilla para la vista que muestra el perfil público de un usuario.

Primero, debes crear una nueva función dentro de **`main.py`** que cree un usuario de prueba, así como una lista de mensajes con un par de mensajes de prueba creados por ese usuario.

A continuación, debes crear una nueva plantilla que reciba el objeto del usuario y los mensajes, y los muestre. Reutiliza las plantillas **`base.html`** y **`posts_template.html`** de los ejercicios anteriores.

Ejercicio 5
Plantilla de la vista de mensajes

De forma similar a la vista principal, crea un controlador y una plantilla para la vista que muestra un mensaje concreto junto con las respuestas al mismo.

Primero, debes crear una nueva función dentro de **`main.py`** que cree un mensaje de prueba, así como una lista con mensajes de respuesta a dicho mensaje.

Después, debes crear una nueva plantilla que muestre dichos mensajes. De nuevo, reutiliza las plantillas **`base.html`** y **`posts_template.html`**, y reutiliza o adapta las otras plantillas comunes.

uc3m | Universidad Carlos III de Madrid
Departamento de Ingeniería Telemática