Guión de prácticas 2: Microframework Flask

Sergio Alonso (zerjioi@ugr.es) y José María Guirao (jmguirao@ugr.es)

Entrega: 25-Octubre

`Flask` es un micro framework web para Python de uso sencillo pero suficientemente potente que permite desarrollar una aplicación web en poco tiempo.

En esta sesión trataremos de construir algunas pequeñas aplicaciones web usando las opciones más básicas de `Flask`. De hecho en esta sesión no prestaremos aún atención a buenas prácticas de desarrollo (como el uso del patrón MVC). En futuras sesiones se practicarán otros conceptos como el manejo de sesiones, plantillas, el acceso a bases de datos, etc.

Un `Hola Mundo!` en `Flask`

En la página web oficial de `Flask` podemos encontrar un ejemplo minimalista ("Hola Mundo") en el que se utiliza el framework para crear un aplicación web extremadamente sencilla que saluda al usuario. Copie dicho código, ejecútelo, compruebe que funciona e intente entender cada parte de dicho programa. Es posible que necesite consultar la "Guia de usuario" o la `API` de la biblioteca.

Tenga en cuenta que para que funcione `Flask` en nuestro contenedor necesitamos instalarlo (no viene de serie en las instalaciones básicas de `Python`). Para ello vamos a crear una imagen de `Docker` que contenga la distribución de python y la librería `flask`. Cuando necesitemos otras librerías iremos actualizando la imagen. Las imágenes de docker se crean con un archivo **Dockerfile**, donde se especifica una imagen de la que se parte, ordenes para instalar librerías, variables de entorno, etc.

La instalación de las librerías las haremos siempre con el instalador pip, para asegurarnos de que las versiones sean las mismas durante todo la fase de desarrollo.

En nuestro caso el archivo **Dokerfile**:

```
# Dockerfile
FROM python:3.7-alpine

# directorio dentro del contenedor para el código
WORKDIR /app
COPY . /app
RUN pip install -r requirements.txt

ENV FLASK_APP app.py
ENV FLASK_RUN_HOST 0.0.0.0
ENV FLASK_ENV development

CMD ["flask", "run"]
```

El archivo **requirements.txt** contendrá las librerías y versiones para `pip`

```
# requirements.txt
flask==1.1
```

El archivo **docker-compose-yml** para usar esta imagen:

```
# docker-compose.yml
version: '3.7'

  services:
    app:
      build: .
      ports:
        - 5000:5000
      volumes:
        - ./app:/app
```

Donde especificamos que una imagen creada desde el archivo **Dockerfile** en el *buildcontext de docker* (nuestro directorio), se comunicará por el puerto 5000, y tendrá montada la carpeta **app**, con el código de la aplicación:

```
#./app/app.py
from flask import Flask
app = Flask(__name__)

@app.route('/')
  def hello_world():
    return 'Hello, World!'
```

Al estar montado el directorio **app** podemos modificar el código desde fuera del contenedor, pero ejecutarlos dentro. Si en algún momento se cambia el **Dockerfile** hay que volver a hacer la imágen con:

```
> docker-compose build
```

La aplicación se pone en marcha con:

```
> docker-compose up
```

La estructura de archivos queda:

```
.
├── app
|   └── app.py
├── docker-compose.yml
├── Dockerfile
├── README.md
└── requirements.txt
```

Es muy conveniente usar git para cualquier proyecto de software, podemos hacerlo desde un repositorio local con

```
> git init .
```

o crear un repositorio en github y clonarlo. En Iniciar repositorio Git y primer commit hay una pequeña introducción.

Además git se integra con Visual Studio Code, Atom y otros editores.

Ejercicios:

1. Hacer un interface web para los ejercicios 2-6 de la práctica anterior. La entrada se hará por URL p.e.

   ```
   http://localhost:5000/ordena/5,2,7,3
   ```

   En Routing y Variable Rules está la documentación para ello.

2. Crear una página servida desde un directorio **static**, con las direcciones del ejercicio anterior

3. Crear una página para el caso en que una URL no esté definida (error `HTTP 404, not found`)

En la documemntación de flask encontrará como hacer esto.

Para Nota Crear Imágenes Dinámicas [Vectoriales]

Desarrolle una aplicación web sencilla que nos permita crear una imagen `SVG` dinámica (que cambie cada vez que visitemos la página) y aleatoria. Por ejemplo, que cada vez que se visite la página dibuje elipses, rectángulos, etc. de colores y posiciones distintas.