PRÁCTICA – 1 DWES 2º TRIMESTRE

Índice

Las siguientes prácticas se harán en grupo de **2 personas máximo**. Si el número de alumnos es impar se admitirá un grupo de 3 personas y nada más que uno.

Se entregará un documento en formato **Word** o, *preferiblemente,* en **Pdf**. En cada una de ellas se indicará el procedimiento empleado para su resolución mediante **texto** y **capturas de pantalla**. Es importante que en las capturas se *aprecie perfectamente nombres de usuarios, textos significativos, hosts, urls, dominios*, etc. De esta forma se diferencia claramente el trabajo de un grupo de alumnos con respecto al de otro, así como de personas ajenas.

En el caso de las aplicaciones ubicadas en un servidor de hosting, se debe además indicar claramente la **URL** de acceso a las mismas.

Estas prácticas se **entregarán antes** del **Jueves, 25 de Enero de 2024, en el Aula Virtual**.

Solamente uno de los alumnos realizará la entrega en dicho Aula Virtual, lo que implica que en el documento a entregar deben aparecer claramente en la portada los nombres de todos los participantes en la práctica.

1. Aplicación Web de tipo CRUD en Python/Flask

Esta aplicación se desarrollará en Windows y consistirá en un **CRUD** (Create-Read-Update-Delete) y con arquitectura **MVC** (Modelo-Vista-Controlador).

La base de datos tendrá una o dos tablas. La tabla(s) se creará(n) utilizando el **ORM** de Flask (paquete SQLAlchemy y otros), así como todos los accesos a las tablas necesarios para realizar el CRUD. Esto implica necesariamente crear un **modelo de datos**.

En el caso de dos tablas, estarán interrelacionadas entre sí de la forma 1:N. Como ejemplos, podrían ser marcas/coches, ciclos/estudiantes, categorías/productos, etc.

Ambas tablas tendrán un campo auto-incremental *id* y otro campo será de tipo String para albergar una descripción.

Además, al menos una tabla tendrá dos columnas dedicadas a las imágenes. Una de estas dos columnas será un String que contendrá el nombre del archivo imagen. Y el archivo imagen se almacenará en la carpeta *static/img* o similar. La otra columna dedicada a imágenes será de tipo BLOB. También existirá una columna de tipo entero o float y otra de tipo de fecha.

En la aplicación se mostrarán claramente las vistas necesarias para listar los datos, darlos de alta y modificación mediante formulario, así como la opción de eliminación y la de detalle.

Los formularios de alta y modificación tendrán validación de campos. Validación tal que, por ejemplo, se obligue a introducir los campos, se valide un rango en el caso de un entero, si es un email, etc. Esta validación puede hacerse con JavaScript y/o con la librería WT_FORMS de Flask.

Se subirá a un servidor de hosting (Vercel u otro que se encuentre). Se pondrá en el documento en **formato texto** (no captura de pantalla) la URL completa de acceso a la misma. No será necesario explicar el procedimiento de subida al hosting en este documento, dado que esto se estará haciendo con otra aplicación en el módulo de Despliegue de Aplicaciones Web.

*Nota: para una mayor uniformidad de las vistas de esta aplicación se aconseja utilizar el motor de plantillas **Jinja2** de Flask. Habrá una vista maestra que se puede enlazar con Bootstrap y/o con los archivos CSS que utilicéis. Esta vista también tendrá la estructura base para la maquetación del resto de vistas. Y estas vistas heredarán y reutilizarán a conveniencia a la vista maestra.*

2. Servicio Web de tipo API REST en Python/Flask y MySQL

Esta aplicación se desarrollará en Windows y consistirá en un **API REST** que implemente los cuatro verbos del protocolo HTTP: GET, POST, PUT, DELETE.

Para esta aplicación ya no se utilizará la librería Flask, sino **FastAPI**.

La tabla sobre la que actuará será la misma del ejercicio anterior, así como el modelo anterior. Y en el caso de haber utilizado dos tablas, será aquella que contenga las columnas imagen.

Como API REST que es esta aplicación tendrá al menos estas rutas:

- **/** o **/usuarios** para listar el contenido (suponiendo que la aplicación sea de usuarios)
- **/** o **/usuario** para dar de alta un elemento (método **POST**. En este caso será necesario incorporar en el cuerpo de solicitud del mensaje HTTP los datos en formato JSON con los valores a añadir)
- **/usuario/id** para editar el contenido del elemento con id=id (método **GET**)
- **/usuario/id** para eliminar el elemento con id=id (método **DELETE**)
- **/usuario/id** para modificar el elemento con id=id (método **PUT**. En este caso será necesario incorporar en el cuerpo de solicitud del mensaje HTTP los datos en formato JSON con los valores a modificar).

Después de realizar la operación, la aplicación devolverá un mensaje en formato JSON. Puede ser de error. Y en el caso de éxito, los datos que correspondan. Si es listar, el contendido en formato JSON de todos los datos de la tabla. Si es dar alta, modificación o eliminación, los datos en formato JSON del elemento implicado en la operación.

Para probar la aplicación, se utilizará la utilidad **Thunder Client** de *Visual Studio Code* o la herramienta gratuita **Postman**.

También sería posible hacerlo mediante llamadas **fetch** desde *JavaScript*.

Incluso, se puede invocar desde una aplicación React.js. En este caso, la aplicación Python/Flask será el *Backend* **productor** de datos y React,js el *Frontend* y **consumidor** de dichos datos.

Se subirá a un servidor de hosting (Vercel u otro que se encuentre). Se pondrá en el documento en **formato texto** (no captura de pantalla) la URL completa de acceso a la misma. No será necesario explicar el procedimiento de subida al hosting en este documento, dado que esto se estará haciendo con otra aplicación en el módulo de Despliegue de Aplicaciones Web.

3. Servicio Web de tipo API REST en Python/Flask y MongoDB

Esta aplicación tendrá exactamente la misma funcionalidad que la del ejercicio nº 2, pero accediendo a la base de datos *NoSQL* **MongoDB**.