

Práctica 06. API Rest en Flask

Administracion y Diseño de Bases de Datos

Cheuk Kelly Ng Pante (alu0101364544@ull.edu.es)

8 de diciembre de 2023

Índice general

1. Introducción	1
2. Actividad 1	1
2.1. Instalación del framework <i>Flask</i> y la biblioteca <i>psycpg2-binary</i>	1
2.2. Despliegue de la aplicación web	2
2.3. Creación de la base de datos e inserción de datos	2
2.4. Personalizar la referencia <i>About</i>	2
2.5. Verificar funcionamiento de la operación de visualizar los registros	3
2.6. Verificar operacion de inserción de registros	4
2.7. Construcción de una REST API para la operación de borrado	5
2.8. Construcción de una REST API para la operación de actualización	6
3. Actividad 2	9
4. Bibliografía	10

1. Introducción

La API RESTful es una interfaz que dos sistemas de computación utilizan para intercambiar información de manera segura a través de Internet. La mayoría de las aplicaciones para empresas deben comunicarse con otras aplicaciones internas o de terceros para llevar a cabo varias tareas

Flask es un framework para desarrollo web escrito en Python. Se puede utilizar para diversos tipos de aplicación, entre ellas, desarrollo de APIs. Existen muchas maneras de implementar un API REST en Flask. Desde usar el framework con lo que ofrece de base, o con la ayuda de extensiones con diferentes configuraciones.

2. Actividad 1

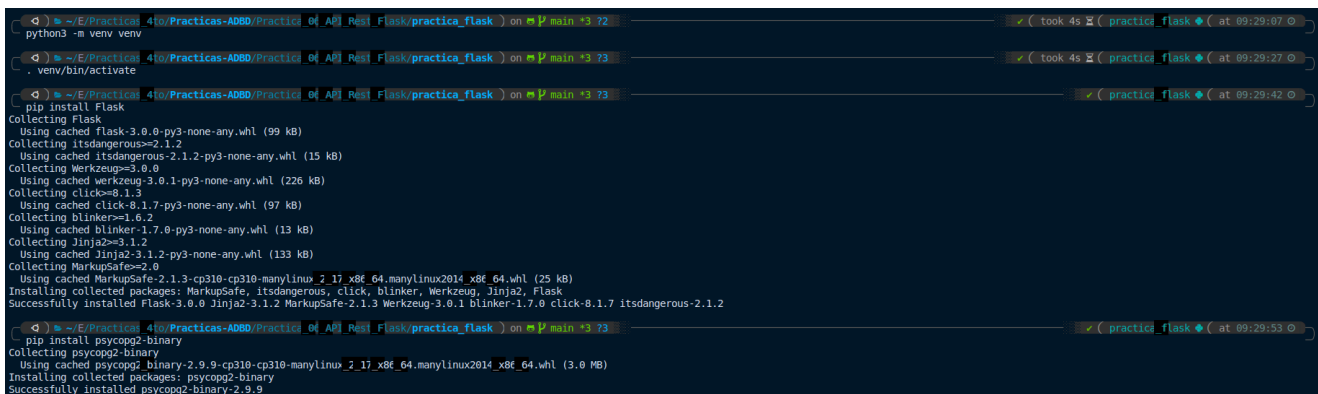
2.1. Instalación del framework *Flask* y la biblioteca *psycpg2-binary*

Para la instalación de *Flask* y *psycpg2-binary*, así como el desarrollo de la práctica se va a crear un entorno virtual con *virtualenv*. Primero se crea un directorio para el entorno virtual y se accede a él.

```
$ mkdir practica_flask
$ cd practica_flask
```

Dentro del directorio se crea el entorno virtual con *virtualenv* y se instalará *Flask* y *psycpg2-binary* dentro de él. Para ello, se ejecutarán los siguientes comandos:

```
$ sudo apt install python3.10-venv
$ python3 -m venv venv
$ source venv/bin/activate
$ pip install Flask
$ pip install psycpg2-binary
```

A screenshot of a terminal window with a dark background and light-colored text. The terminal shows the execution of several commands to set up a virtual environment and install dependencies. The commands are: 'python3 -m venv venv', '. venv/bin/activate', 'pip install Flask', and 'pip install psycpg2-binary'. The output for 'pip install Flask' shows the collection and installation of various packages including Flask, Werkzeug, Jinja2, and others. The output for 'pip install psycpg2-binary' shows the collection and installation of the psycpg2-binary package. The terminal window has a title bar at the top with icons and text indicating the current directory and environment.

```
practica_flask ~$ python3 -m venv venv
practica_flask ~$ . venv/bin/activate
practica_flask ~$ pip install Flask
Collecting Flask
  Using cached flask-3.0.0-py3-none-any.whl (99 kB)
Collecting itsdangerous<=2.1.2
  Using cached itsdangerous-2.1.2-py3-none-any.whl (15 kB)
Collecting Werkzeug>=3.0.0
  Using cached werkzeug-3.0.1-py3-none-any.whl (226 kB)
Collecting click>=8.1.3
  Using cached click-8.1.7-py3-none-any.whl (97 kB)
Collecting blinker>=1.6.2
  Using cached blinker-1.7.0-py3-none-any.whl (13 kB)
Collecting Jinja2>=3.1.2
  Using cached Jinja2-3.1.2-py3-none-any.whl (133 kB)
Collecting MarkupSafe>=2.0
  Using cached MarkupSafe-2.1.3-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (25 kB)
Installing collected packages: MarkupSafe, itsdangerous, click, blinker, Werkzeug, Jinja2, Flask
Successfully installed Flask-3.0.0 Jinja2-3.1.2 MarkupSafe-2.1.3 Werkzeug-3.0.1 blinker-1.7.0 click-8.1.7 itsdangerous-2.1.2
practica_flask ~$ pip install psycpg2-binary
Collecting psycpg2-binary
  Using cached psycpg2_binary-2.9.9-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (3.0 MB)
Installing collected packages: psycpg2-binary
Successfully installed psycpg2-binary-2.9.9
```

Figura 2.1: Instalación de Flask y psycpg2-binary

2.2. Despliegue de la aplicación web

Para desplegar la aplicación web se va a crear una base de datos en *PostgreSQL* con el nombre de *flask_db*. Para ello, ejecutamos dentro de *psql* el siguiente comando:

```
postgres=# CREATE DATABASE flask_db;
```

Una vez creada la base de datos, sobre el directorio *practica_flask* se ponen los archivos que se van a utilizar para el desarrollo de la práctica. Estos archivos son: *app.py* e *init.py* y dentro de estos ficheros añadimos el usuario y contraseña de *postgres* para poder acceder a la base de datos.

Ya modificados los archivos, se ejecuta el siguiente comando para desplegar la aplicación web:

```
$ python3 app.py
```

También se puede ejecutar el siguiente comando para desplegar la aplicación web:

```
$ flask --app app.py run --host 0.0.0.0 --port=8080
```

Con los comandos anteriores desplegamos la aplicación en local en el puerto 5000, pero va a fallar ya que antes necesita la inicialización de la base de datos, lo cual se hace con el siguiente comando:

```
$ python3 init_db.py
```

2.3. Creación de la base de datos e inserción de datos

La creación de la base de datos se hizo en el apartado anterior y la inserción de datos se crean en el script *init_db.py*.

2.4. Personalizar la referencia *About*

Para personalizar la referencia del *about* creamos un nuevo archivo *html* en el directorio *templates* *about.html* en donde añadimos los nombres y apellidos de los integrantes del grupo. Luego modificamos *base.html* y el fichero *app.py* para poder acceder a la nueva sección.

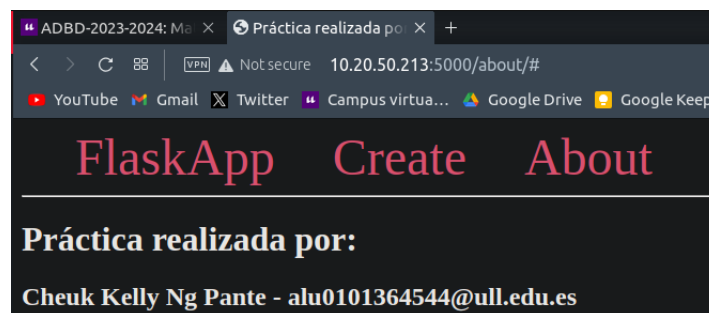


Figura 2.2: About personalizado

2.5. Verificar funcionamiento de la operación de visualizar los registros

Se puede verificar que se pueden visualizar los registros de la base de datos gracias a la función *index()* que se encuentra en el fichero *app.py*, en el que se realiza la siguiente consulta para conseguir todas las entradas de la tabla *books*:

```
1 SELECT *  
2 FROM books;
```

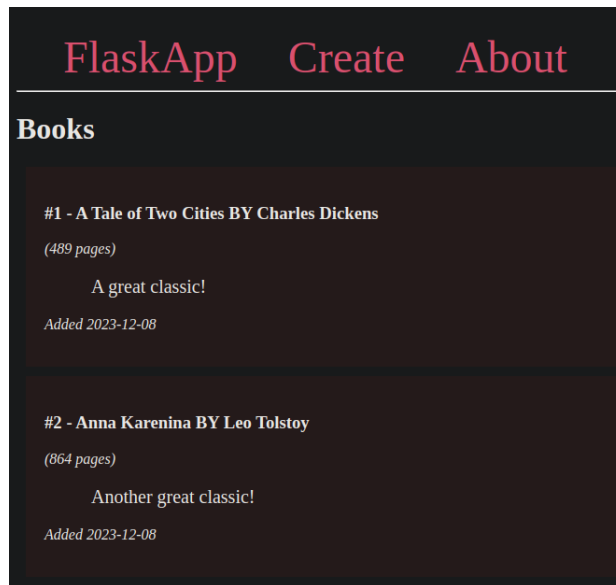
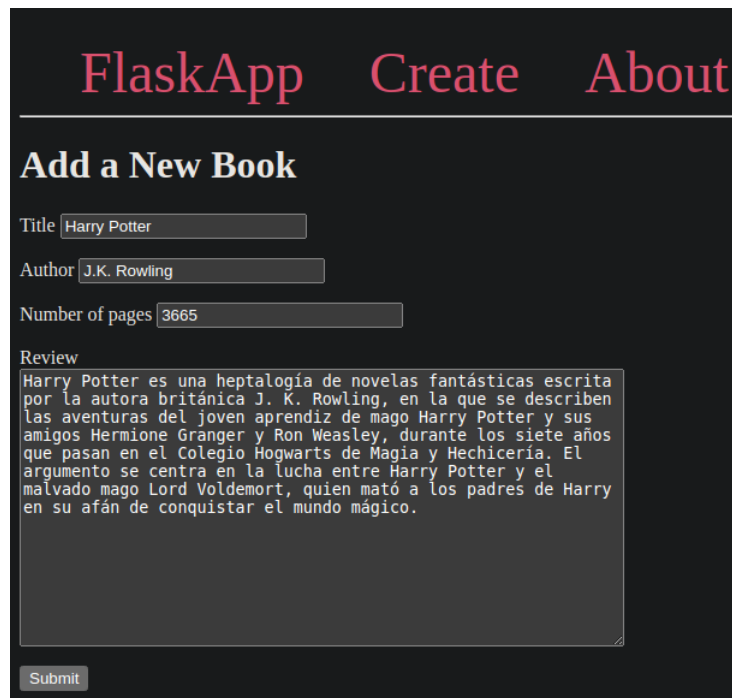


Figura 2.3: Visualizar registros

2.6. Verificar operacion de inserción de registros

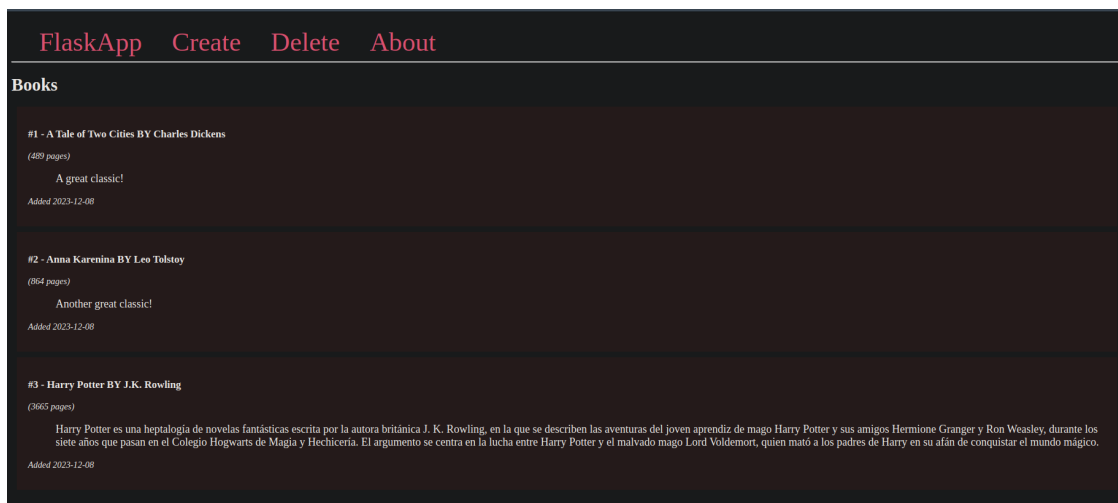
Se puede verificar que se pueden insertar registros en la base de datos gracias a la función `create()` que se encuentra en el fichero `app.py`, en el que se realiza la siguiente consulta para insertar una entrada en la tabla `books`:

```
1 INSERT INTO books (title, author, pages_num, review)
2 VALUES (%s, %s, %s, %s);
```



The screenshot shows a web application interface with a dark background. At the top, there are navigation links: 'FlaskApp', 'Create' (which is highlighted), and 'About'. Below the navigation bar, the heading 'Add a New Book' is displayed. The form contains four input fields: 'Title' with the value 'Harry Potter', 'Author' with 'J.K. Rowling', 'Number of pages' with '3665', and a 'Review' text area containing a paragraph about the Harry Potter series. A 'Submit' button is located at the bottom left of the form.

Figura 2.4: Insertar registros



The screenshot shows the 'Create' page of the FlaskApp, which now includes a 'Delete' link in the navigation bar. The main content area is titled 'Books' and displays a list of three books. Each book entry includes its rank, title, author, page count, a short description, and the date it was added.

Rank	Title	Author	Pages	Description	Added
#1	A Tale of Two Cities	BY Charles Dickens	(489 pages)	A great classic!	Added 2023-12-08
#2	Anna Karenina	BY Leo Tolstoy	(864 pages)	Another great classic!	Added 2023-12-08
#3	Harry Potter	BY J.K. Rowling	(3665 pages)	Harry Potter es una heptalogía de novelas fantásticas escrita por la autora británica J. K. Rowling, en la que se describen las aventuras del joven aprendiz de mago Harry Potter y sus amigos Hermione Granger y Ron Weasley, durante los siete años que pasan en el Colegio Hogwarts de Magia y Hechicería. El argumento se centra en la lucha entre Harry Potter y el malvado mago Lord Voldemort, quien mató a los padres de Harry en su afán de conquistar el mundo mágico.	Added 2023-12-08

Figura 2.5: Registro insertado

2.7. Construcción de una REST API para la operación de borrado

Para construir una REST API para la operación de borrado se crea una nueva función en el fichero *app.py* llamada *delete()*, aquí el código en python:

```
1 @app.route('/delete/', methods=['GET', 'POST'])
2 def delete():
3     if request.method == 'POST':
4         id = request.form['ID']
5
6         conn = get_db_connection()
7         cur = conn.cursor()
8         cur.execute('DELETE FROM books WHERE id = %s',
9                     (id))
10        conn.commit()
11        cur.close()
12        conn.close()
13        return redirect(url_for('index'))
14
15    return render_template('delete.html')
```

Este código en python lo que hace es una consulta en sql para borrar una entrada de la tabla *books*, aquí el código en sql:

```
1 DELETE FROM books
2 WHERE id = %s;
```

Además, hay que modificar el fichero *base.html* para crear un apartado en el que se pueda borrar un registro de la base de datos. Creamos fichero *delete.html* en el directorio *templates* y lo codificamos para que se pueda borrar un registro de la base de datos con un formulario, aquí el código en html:

```
1 {% extends 'base.html' %}
2
3 {% block content %}
4 <h1>{% block title %} Delete a Book {% endblock %}</h1>
5 <form method="POST">
6     <p>
7         <label for="ID">ID</label>
8         <input type="text" name="ID" placeholder="Book ID">
9     </input>
10 </p>
11 <p>
12     <button type="submit">Submit</button>
13 </p>
14 </form>
15 {% endblock %}
```

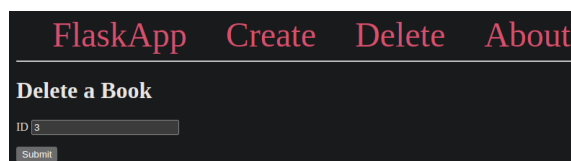


Figura 2.6: Borrar registros

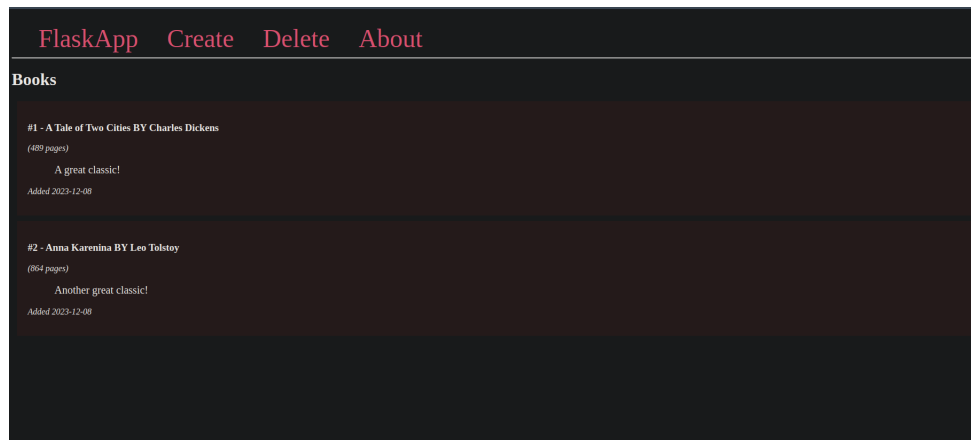


Figura 2.7: Registro borrado

2.8. Construcción de una REST API para la operación de actualización

Para construir una REST API para la operación de actualización se crea una nueva función en el fichero *app.py* llamada *update()*, en el que se actualizará un registro indicando su ID, aquí el código en python:

```

1 @app.route('/update/', methods=['GET', 'POST'])
2 def update():
3     if request.method == 'POST':
4         id = request.form['ID']
5         title = request.form['title']
6         author = request.form['author']
7         review = request.form['review']
8
9         conn = get_db_connection()
10        cur = conn.cursor()
11        cur.execute('UPDATE books SET title = %s, author = %s, review = %s WHERE id = %s
12                    (title, author, review, id))
13        conn.commit()
14        cur.close()
15        conn.close()
16        return redirect(url_for('index'))
17
18    return render_template('update.html')

```

Este código en python lo que hace es una consulta en sql para actualizar una entrada de la tabla *books*, aquí el código en sql:

```

1 UPDATE books
2 SET title = %s, author = %s, review = %s WHERE id = %s;

```

Luego, hay que modificar el fichero *base.html* para crear un apartado en el que se pueda actualizar un registro de la base de datos. Creamos fichero *update.html* en el directorio *templates* y lo codificamos para que se pueda actualizar un registro de la base de datos con un formulario, aquí el código en html:


```

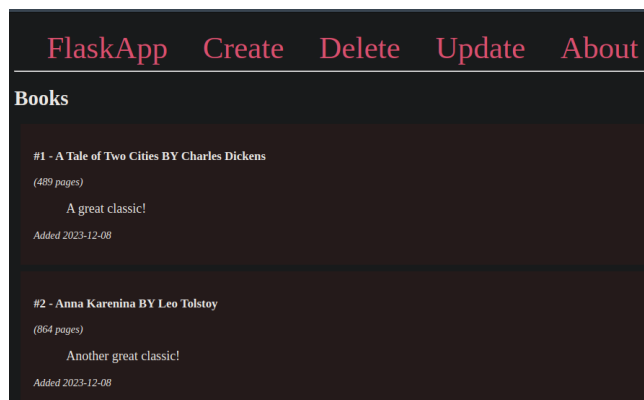
1 {% extends 'base.html' %}
2
3 {% block content %}
4 <h1>{% block title %} Update a Book {% endblock %}</h1>
5 <form method="post">
6   <p>
7     <label for="ID">ID</label>
8     <input type="text" name="ID" placeholder="Book ID">
9   </input>
10 </p>
11
12 <p>
13   <label for="title">Title</label>
14   <input type="text" name="title" placeholder="Book title">
15 </input>
16 </p>
17
18 <p>
19   <label for="author">Author</label>
20   <input type="text" name="author" placeholder="Book author">
21 </input>
22 </p>
23
24 <p>
25   <label for="pages_num">Number of pages</label>
26   <input type="number" name="pages_num" placeholder="Number of pages">
27 </input>
28 </p>
29
30 <p>
31   <label for="review">Review</label>
32   <br>
33   <textarea name="review" placeholder="Review" rows="15" cols="60"></textarea>
34 </p>
35 <p>
36   <button type="submit">Submit</button>
37 </p>
38 </form>
39 {% endblock %}

```

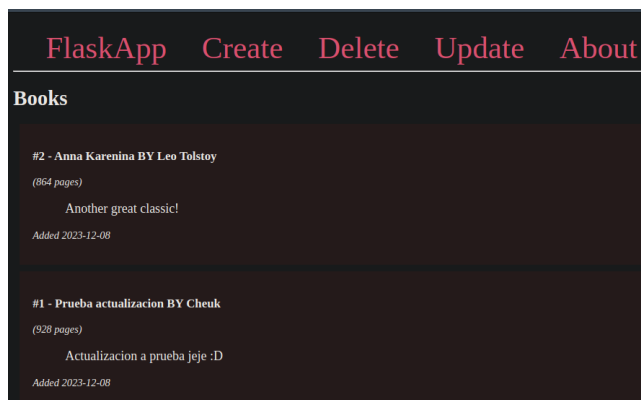
The screenshot shows a web application with a dark background. At the top, there is a navigation bar with links: 'FlaskApp', 'Create', 'Delete', 'Update', and 'About'. Below this, the main heading is 'Update a Book'. The form contains the following elements:

- ID:** A text input field with the value '1'.
- Title:** A text input field with the value 'Prueba actualizacion'.
- Author:** A text input field with the value 'Cheuk'.
- Number of pages:** A text input field with the value '928'.
- Review:** A large text area containing the text 'Actualizacion a prueba jeje :D'.
- Submit:** A button at the bottom of the form.

Figura 2.8: Actualizar registros



(a) Libro no actualizado



(b) Libro actualizado

Figura 2.9: Actualización de registros

3. Actividad 2

3.1.

4. Bibliografía

- 1.