

Clase 05

Rumbo al Proyecto final Full Stack Python con objetos y API REST

Quinta etapa:

Desplegar el proyecto en PythonAnywhere

PythonAnywhere es una empresa de hosting para aplicaciones web escritas en Python. Al crear un usuario, conseguimos de forma gratuita una especie de máquina virtual Linux con varios intérpretes de Python instalados, múltiples módulos y paquetes de terceros y la capacidad de instalar nuevos vía pip, una base de datos MySQL lista para usar, un servidor web plenamente configurado, un sistema de archivos con 512 MB de capacidad y muchas otras cosas interesantes. No solamente se trata de una solución ideal para llevar por primera vez una aplicación web a producción, sino también para proyectos profesionales. PythonAnywhere permite seleccionar únicamente los recursos que queremos usar y pagar en consecuencia; y, en los planes pagos, configurar nuestras aplicaciones con un dominio propio.

Registro y configuración

Para comenzar a utilizar PythonAnywhere, nos dirigimos a <https://www.pythonanywhere.com/pricing/> y presionamos el botón Create a Beginner account:

Plans and pricing

Beginner: Free!

A **limited account** with one web app at `your-username.pythonanywhere.com`, restricted outbound Internet access from your apps, low CPU/bandwidth, no IPython/Jupyter notebook support.

It works and it's a great way to get started!

Create a Beginner account

Luego nos registramos completando el formulario con un nombre de usuario, una dirección de correo electrónico y una contraseña. Esto enviará un correo electrónico a la dirección indicada para confirmar el registro.

Create your account

Username:

Email:

Password:

Password (again):

☐ I agree to the [Terms and Conditions](#) and the [Privacy and Cookies Policy](#), and confirm that I am at least 13 years old.

Register

We promise not to spam or pass your details on to anyone else.

Como siempre, nos aseguramos de tener los datos (username, pass, etc) a mano porque los vamos a necesitar luego. Esto enviará un correo electrónico a la dirección indicada para confirmar el registro. Ahora, al ingresar a <https://www.pythonanywhere.com/> deberíamos ver algo así:

Dashboard

Welcome, [arielcodo](#)CPU Usage: 0% used – 0.00s of 100s. Resets in 23 hours, 58 minutes [More Info](#)File storage: 0% full – 48.0 KB of your 512.0 MB quota [More Info](#)[Upgrade Account](#)Recent
Consoles

+ 5 -

You have no recent consoles.

New console:

\$ Bash

>>> Python ▾

[More...](#)Recent
Files

+ 5 -

You have no recently edited files.[+ Open another file](#)[Browse files](#)Recent
Notebooks

+ 5 -

Your account does not support Jupyter Notebooks. [Upgrade your account](#) to get access!

Tenemos que indicarle a PythonAnywhere que queremos subir una aplicación de Flask. En el menú superior derecho, vamos a dirigirnos a la opción Web. Una vez allí, a la izquierda, presionamos el botón "+ Add a new web app":

[+ Add a new web app](#)

You have no web apps

To create a PythonAnywhere-hosted web app, click the "Add a new web app" button to the left.

Se iniciará un asistente que nos consultará el framework que queremos usar, la versión de Python y el nombre del proyecto. Completaremos esos datos con las opciones «Flask», «Python 3.10» (o la versión que esté

utilizando) y «mysite» (o el nombre de la carpeta que contenga tu archivo `app_flask.py`), como se ilustra a continuación.


Create new web app

Quickstart new Flask project


Enter a path for a Python file you wish to use to hold your Flask app. If this file already exists, its contents will be overwritten with the new app.

Path

`/home/arielcodo/mysite/flask_app.py`



En este punto nos encontramos con la configuración básica realizada:

 **python**anywhere
by ANACONDA.

Dashboard Consoles Files **Web** Tasks Databases

All done! Your web app is now set up. Details below.

arielcodo.pythonanywhere.com

+ Add a new web app

Configuration for arielcodo.pythonanywhere.com

Reload:

Reload arielcodo.pythonanywhere.com

Y vemos cual es la URL de nuestro sitio. Si dirigimos nuestro navegador a esa dirección, vemos:



Ahora necesitamos cambiar la `flask_app.py` que el sitio ha puesto por defecto por nuestro código. Hay muchas maneras de hacer eso, una sencilla es simplemente editar `flask_app.py` y copiar encima nuestro código. Para ello, bajamos y hacemos click en el siguiente enlace:

Code:

What your site is running.

[/home/arielcodo/mysite](#)
➔ Go to directory


En la pantalla siguiente vemos los archivos que se encuentran en nuestro directorio, en "Files":



pythonanywhere

by ANACONDA.

[Dashboard](#) [Consoles](#) **Files** [Web](#) [Tasks](#) [Databases](#)

/home/arielcodo/  [mysite](#) [Open Bash console here](#)

0% full – 432.0 KB of your 512.0 MB quota [More Info](#)

Directories

[New directory](#)

[__pycache__/](#)





Files

[New file](#)

 [flask_app.py](#)



  2023-06-10 18:23 8.8 KB

 [inventario.db](#)

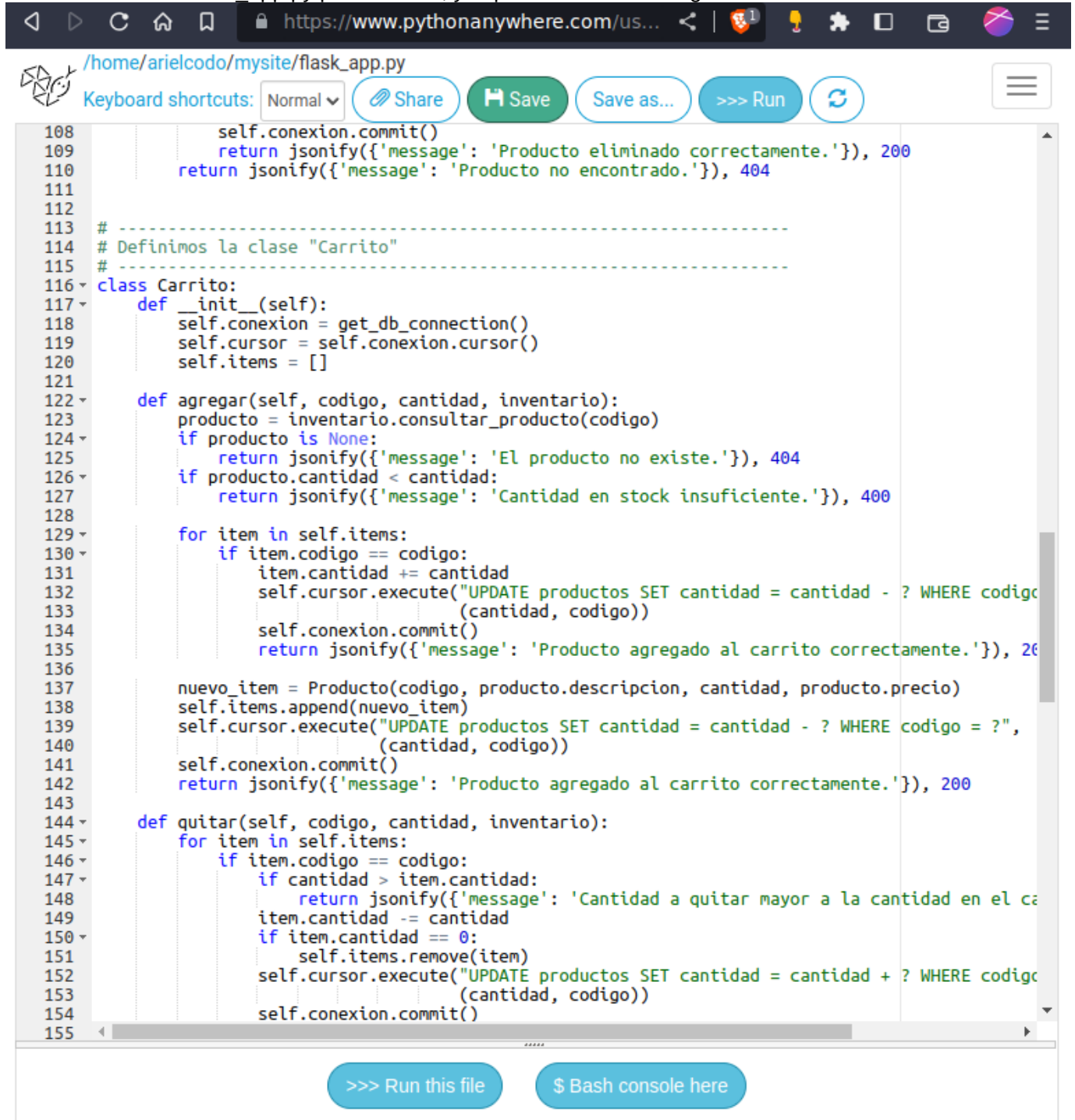


 2023-06-10 17:34 8.0 KB

[Upload a file](#)

100MiB maximum size

Hacemos click en Flask_app.py para editarlo, y copiamos nuestro código:



```
108         self.conexion.commit()
109         return jsonify({'message': 'Producto eliminado correctamente.'}), 200
110     return jsonify({'message': 'Producto no encontrado.'}), 404
111
112     # -----
113     # Definimos la clase "Carrito"
114     # -----
115
116     class Carrito:
117     def __init__(self):
118         self.conexion = get_db_connection()
119         self.cursor = self.conexion.cursor()
120         self.items = []
121
122     def agregar(self, codigo, cantidad, inventario):
123         producto = inventario.consultar_producto(codigo)
124         if producto is None:
125             return jsonify({'message': 'El producto no existe.'}), 404
126         if producto.cantidad < cantidad:
127             return jsonify({'message': 'Cantidad en stock insuficiente.'}), 400
128
129         for item in self.items:
130             if item.codigo == codigo:
131                 item.cantidad += cantidad
132                 self.cursor.execute("UPDATE productos SET cantidad = cantidad - ? WHERE codigo = ?",
133                                     (cantidad, codigo))
134                 self.conexion.commit()
135                 return jsonify({'message': 'Producto agregado al carrito correctamente.'}), 200
136
137         nuevo_item = Producto(codigo, producto.descripcion, cantidad, producto.precio)
138         self.items.append(nuevo_item)
139         self.cursor.execute("UPDATE productos SET cantidad = cantidad - ? WHERE codigo = ?",
140                             (cantidad, codigo))
141         self.conexion.commit()
142         return jsonify({'message': 'Producto agregado al carrito correctamente.'}), 200
143
144     def quitar(self, codigo, cantidad, inventario):
145         for item in self.items:
146             if item.codigo == codigo:
147                 if cantidad > item.cantidad:
148                     return jsonify({'message': 'Cantidad a quitar mayor a la cantidad en el carrito.'}), 400
149                 item.cantidad -= cantidad
150                 if item.cantidad == 0:
151                     self.items.remove(item)
152                 self.cursor.execute("UPDATE productos SET cantidad = cantidad + ? WHERE codigo = ?",
153                                     (cantidad, codigo))
154                 self.conexion.commit()
```

Antes de salir, guardamos (Save) y recargamos el sitio con el ultimo botón de la barra:



Con esto tendríamos lista nuestra API, corriendo en la dirección web (URL) que nos ha proporcionado Pythonanywhere. En este texto, es <https://arielcodo.pythonanywhere.com>

Podemos probar si funciona escribiendo en el navegador <https://arielcodo.pythonanywhere.com/productos> y deberíamos recibir como respuesta un arreglo vacío ([]), ya que no hemos cargado aún productos.

Prueba rápida de la API

Podemos hacer una prueba rápida usando este archivo HTML. Lo ejecutamos en nuestra computadora con liveServer, completamos el formulario y enviamos su contenido:

pruebarapida.html:

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Agregar producto</title>
</head>

<body>
  <h1>Agregar Producto al Inventario</h1>
  <form id="formulario">
    <label for="codigo">Código:</label>
    <input type="text" id="codigo" name="codigo" required><br>

    <label for="descripcion">Descripción:</label>
    <input type="text" id="descripcion" name="descripcion" required><br>

    <label for="cantidad">Cantidad:</label>
    <input type="number" id="cantidad" name="cantidad" required><br>

    <label for="precio">Precio:</label>
    <input type="number" step="0.01" id="precio" name="precio" required><br>

    <button type="submit">Agregar Producto</button>
  </form>

  <script>
    // Capturamos el evento de envío del formulario
    document.getElementById('formulario').addEventListener('submit', function (event) {
      event.preventDefault(); // Evitamos que se recargue la página

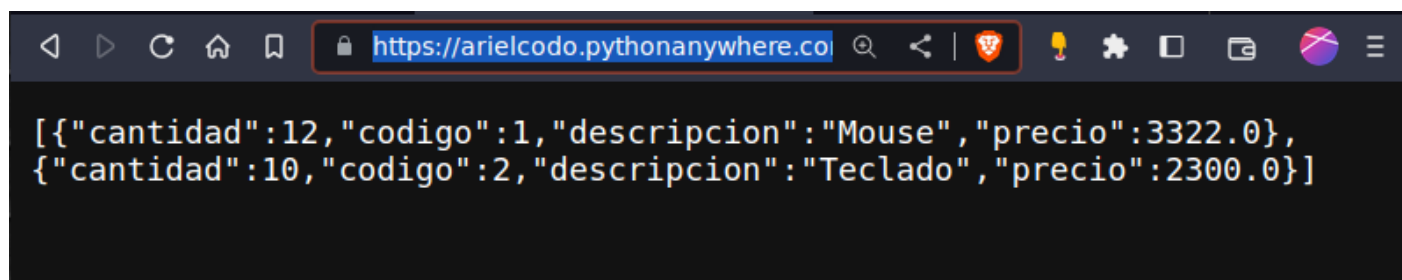
      // Obtenemos los valores del formulario
      var codigo = document.getElementById('codigo').value;
      var descripcion = document.getElementById('descripcion').value;
      var cantidad = document.getElementById('cantidad').value;
      var precio = document.getElementById('precio').value;

      // Creamos un objeto con los datos del producto
      var producto = {
        codigo: codigo,
        descripcion: descripcion,
        cantidad: cantidad,
        precio: precio
      };
      console.log(producto)
      // Realizamos la solicitud POST al servidor
      fetch('https://xxxxxx.pythonanywhere.com/productos', {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json'
        },
        body: JSON.stringify(producto)
      })
        .then(function (response) {
          if (response.ok) {
            return response.json(); // Parseamos la respuesta JSON
          } else {
            throw new Error('Error al agregar el producto.');
```

</html>

IMPORTANTE: Recuerda cambiar la URL <https://xxxxxx.pythonanywhere.com/productos> por la de TU implementación.

Si todo ha funcionado, al apuntar el navegador a <https://arielcodo.pythonanywhere.com/productos> nuevamente, en lugar del arreglo vacío deberías ver algo como esto:

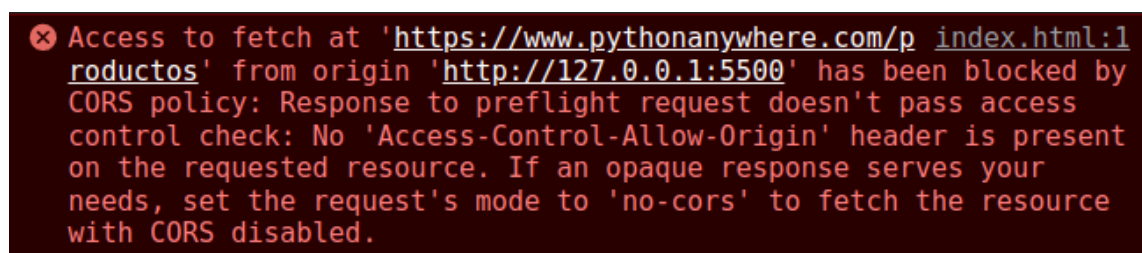


Es decir, un arreglo con los datos de los productos de la base de datos.

Pero puede que no haya funcionado. Vemos porqué:

CORS

Es probable que al intentar ingresar un producto en el inventario se obtenga un error similar a este:



El error se debe a una política de seguridad denominada **Same-Origin Policy** (Política del mismo origen) implementada por los navegadores web. Esta política restringe las solicitudes HTTP realizadas desde un origen (dominio, protocolo y puerto) a otro origen diferente.

Ocurre, por ejemplo, se estás realizando una solicitud desde el origen <http://127.0.0.1:5500> a <https://www.pythonanywhere.com>, lo cual no cumple con la política de mismo origen y, por lo tanto, se bloquea.

Para solucionar este problema, necesitas habilitar el intercambio de recursos de origen cruzado (Cross-Origin Resource Sharing, CORS) en el servidor de PythonAnywhere para permitir que tu sitio web acceda a la API.

En el código de Flask, debes agregar los encabezados de respuesta adecuados que permitan las solicitudes desde el origen deseado. Puedes hacerlo utilizando la extensión Flask-CORS, que debemos instalar así:

1) Instalar Flask-CORS en el entorno de PythonAnywhere. Para ello, abrimos una terminal bash desde el dashboard de PythonAnywhere y en ella ejecutamos:

```
pip install flask-cors
```

2) Importar y configurar Flask-CORS en la aplicación Flask, modificando nuestro código Python para que se agregue al principio la línea **from flask_cors import CORS**


```
from flask import Flask
from flask_cors import CORS
```

Y cerca del final, luego de crear la app Flask agregamos **CORS(app)** :

```
app = Flask(__name__)
CORS(app)
```

Con estos cambios, el error desaparece.

TODO list

Está claro que esta API, y el proyecto en general, está lejos de ser una aplicación web completa. Este ejercicio puede ser mejorado, por ejemplo en los siguientes aspectos:

1) Utilizar una base de datos SQL en Pythonanywhere

Para ello, se debe acceder a la sección "databases" y configurar allí nuestra base de datos:

[Send feedback](#) [Forums](#) [Help](#) [Blog](#) [Account](#) [Log out](#)



pythonanywhere
by ANACONDA.

[Dashboard](#) [Consoles](#) [Files](#) [Web](#) [Tasks](#) **Databases**

MySQL

[Postgres](#)

Initialize MySQL

Let's get started! The first thing to do is to initialize a MySQL server:

Enter a new password in the form below, and note it down: you'll need it to access the databases once you've created them. You will only need to do this once.

New password:

Confirm password:

Initialize MySQL

This should be different to your main PythonAnywhere password, because it is likely to appear in plain text in any web applications you write.

Y, por supuesto, el código Python desarrollado debe modificarse. Hay que utilizar el módulo **pymysql** en lugar de **sqlite3**, y modificar la forma en que se accede a la base de datos.

Se deja a disposición del alumno un ejemplo de código en **Clase 03_pymysql.py**. Recuerda que para utilizar ese código, necesitas tener funcionando el servidor SQL (si lo vas a utilizar en forma local) o la base de datos correctamente configurada en Pythonanywhere.

2) No es difícil incorporar mas campos al inventario, o mas tablas a nuestra base de datos. El código desarrollado sirve de ejemplo de como implementar estas mejoras. Incluso se pueden agregar imágenes, implementando un mecanismo que permite seleccionar una imagen de la computadora del usuario y que las suba al servidor. En la base de datos se guarda el nombre del archivo unicamente. En ese caso, debemos utilizar algun servidor que permita almacenar un mayor número de archivos (Pythonanywhere, en su versión gratuita, solo permite 5 archivos por sitio).

3) El carrito de compras es el mismo para todos los usuarios. Para que cada usuario tenga su propio carrito de compras es necesario, por supuesto, definir la clase *Cliente* (o similar) y crear una tabla en la que se almace el contenido de los carritos de compras de los usuarios con un campo que sirva de clave externa para vincular cada uno de sus registros con el usuario correspondiente.

Ahora, solo queda implementar el frontend.