



PROCESO DE GESTIÓN DE FORMACIÓN PROFESIONAL INTEGRAL
FORMATO MATERIAL DE APOYO

Material de Apoyo de Aplicación web en Python con Flask

1. Realizar las tareas del CRUD a la siguiente base de datos tipo SQL: Puede ser en mysql o sqlite

v	tiendaorm	productos
		idProducto : int(11)
		proCodigo : int(11)
		proNombre : varchar(50)
		proPrecio : int(11)
		proFoto : varchar(50)
		proCategoria : int(11)

2. Realizar las tareas del CRUD a la base de datos anterior pero en MongoDB

Colección:

```
Productos: {  
    proCodigo,  
    proNombre,  
    proPrecio,  
    proFoto,  
    proCategoria  
}
```

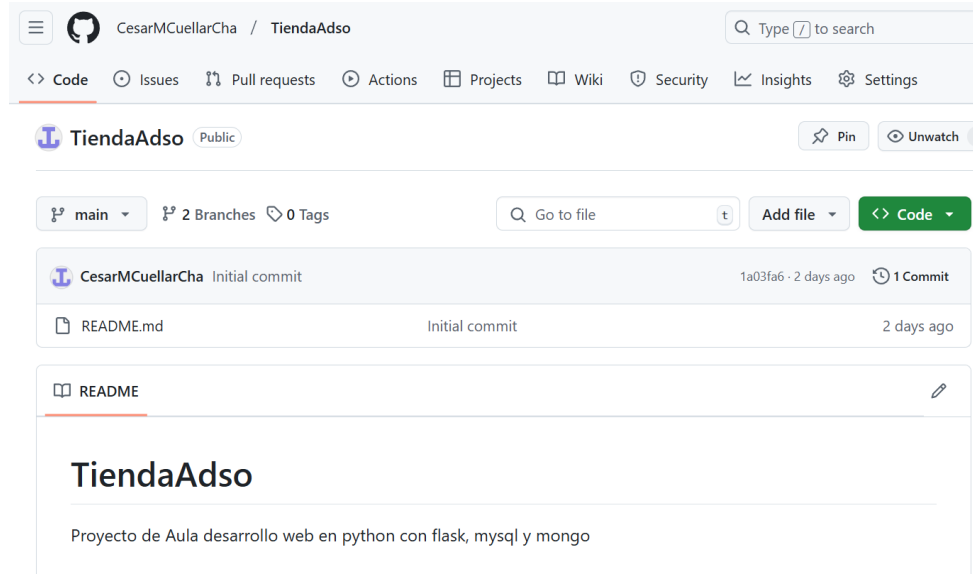
Tareas:

- Agregar un producto
- Consultar producto
- Listar los productos
- Actualizar un producto
- Eliminar un producto



Pasos

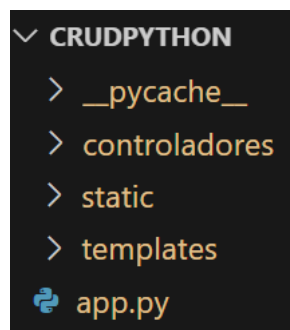
1. Crear repositorio en Github



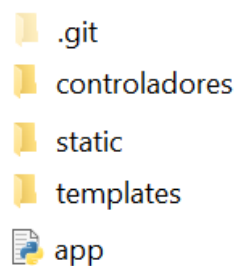
2. Crear Carpeta del proyecto

3. Abrir **Visual Studio Code** con carpeta de trabajo la del proyecto

4. Crear estructura de carpetas del proyecto incluyendo el archivo que arranca la aplicación llamado app.py

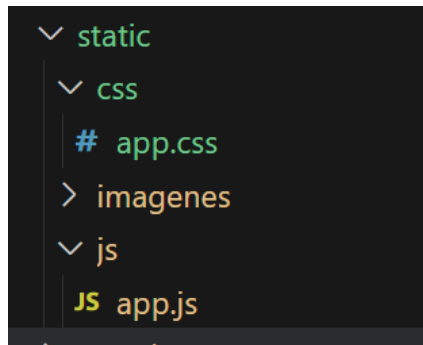


5. Desde visual studio en una terminal crear repositorio local con **git init**.





- Dentro de la carpeta **static** crear una carpeta llamada **css**, otra **js** y otra llamada **imágenes**. Dentro de **css** crear un archivo **app.css** para configurar su hoja de estilos; dentro de la carpeta **js** crear un archivo llamado **app.js** para ahí colocar código javascript que necesite; y en la carpeta **imágenes** colocar imágenes que necesite en el proyecto y además se va a utilizar para ahí guardar las imágenes de los productos.



- Crear un entorno de trabajo para guardar las librerías utilizadas en el proyecto

En la raíz del proyecto y en una terminal ejecutar comando **Python -m venv entorno**

```
C:\Users\AdminSena\Documents\SENA2025\GRUPOS\ADS02874057\Actividades\CONSTRUCCIÓN DEL SOFTWARE\PROYECTOS PYTHON\CRUDPYTHON> python -m venv entorno
```

- Activar el entorno. Para activar el entorno ubicarse en la carpeta **scripts** que se encuentra dentro de la carpeta **entorno** y ejecutar el archivo **activte.bat**

```
C:\Users\AdminSena\Documents\SENA2025\GRUPOS\ADS02874057\PYTHON\CRUDPYTHON\entorno\Scripts>activate.bat
```

- Después en la parte izquierda del prompt de la consola parece entre paréntesis el nombre del entorno que se encuentra activo:

```
(entorno) C:\Users\AdminSena\Documents\SENA2025\GRUPOS\PYTHON\CRUDPYTHON\entorno\Scripts>
```



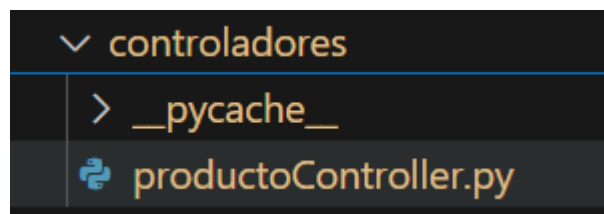
10. Instalar las librerías requeridas para el proyecto: Para nuestro proyecto se requiere instalar **Flask** como framework de desarrollo web en Python y **Pymysql** como conector a bases de datos mysql.

```
(entorno) C:\Users\AdminSena\Documents\SENA2025\GRUPOS\ADS02874
\PROYECTOS PYTHON\CRUDPYTHON>pip install Flask Pymysql
```

11. Verificar si se instalaron las librerías en el entorno: ejecutar comando **pip freeze**

```
(entorno) C:\Users\AdminSena\Documents\SENA
\PROYECTOS PYTHON\CRUDPYTHON>pip freeze
blinker==1.9.0
click==8.1.8
colorama==0.4.6
Flask==3.1.0
itsdangerous==2.2.0
Jinja2==3.1.6
MarkupSafe==3.0.2
PyMySQL==1.1.1
Werkzeug==3.1.3
```

12. Crear en la carpeta controladores un archivo llamado **productoController.py**, el cual se va a contener las rutas a las peticiones del **CRUD**.





13. Editar el archivo **app.py** para arrancar la aplicación. Dicho archivo debe importar el archivo **productoController.py** donde se van a encontrar todas las rutas.

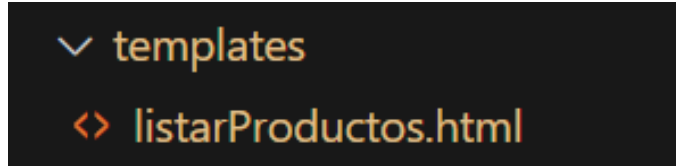
```
app.py > ...
1  from flask import Flask
2  import pymysql as mysql
3
4  # crear el objeto de tipo Flask
5  app = Flask(__name__)
6
7  #configurar carpeta donde se van a guardar las fotos de los productos
8  app.config["UPLOAD_FOLDER"]="./static/imagenes/"
9
10 #variables para conectarse a mysql
11 host="localhost"
12 user="root"
13 password=""
14 baseDatos="tienda"
15
16 #obtener objeto conexión de tipo mysql
17 miConexion=mysql.connect(host=host, user=user,
18 | | | | | password=password, database=baseDatos)
19
20
21 #arrancar la aplicación
22 if __name__ == '__main__':
23 | #importar el controlador de producto
24 | from controladores.productoController import *
25 | app.run(port=5000, debug=True)
```

14. Editar el archivo **productoController.py** con la ruta principal o de inicio la cual debe mostrar la interfaz donde se listan los productos que tenga la base de datos.

```
controladores > productoController.py > inicio
1  from flask import Flask, render_template, jsonify, request, redirect
2  #importar objetos del archivo app.py
3  from app import app, miConexion
4  #libreria para obtener nombre de archivo
5  from werkzeug.utils import secure_filename
6  #libreria para guardar archivos en servidor
7  import os
8
9
10 Tabnine | Edit | Test | Explain | Document
11 @app.route("/")
12 def inicio():
13 | """_summary_
14 |     Obtiene todos los productos de la base de datos
15 |     y los retorna a la vista listarProductos
16 | Returns:
17 |     _type_: Tupla de Tuplas, con los productos
18 | """
19 | try:
20 |     productos=None
21 |     consulta="select * from productos"
22 |     cursor = miConexion.cursor()
23 |     cursor.execute(consulta)
24 |     productos = cursor.fetchall()
25 | except miConexion.Error as error:
26 |     mensaje=str(error)
27 | return render_template("listarProductos.html", listaproductos=productos)
```



15. Como la ruta principal retorna a una vista llamada **listarProductos.html**, vamos a crear el archivo editándolo para que muestre los productos. El archivo debe ser creado en la carpeta **templates**.



Código del documento html

```
templates > <> listarProductos.html > html > head > link
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css"
7      rel="stylesheet" integrity="sha384-QWTKZyjpPEjISv5WaRU90FeRpok6YctnYmDr5pNlyT2bRjXh0JMhY6hW+ALEwIH"
8      crossorigin="anonymous">
9      <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js"
10     integrity="sha384-YvpcrYf0tY3lHB60NNkmXc5s9fDVZLESaAA55NDz0xhy9GkcIdslK1eN7N6jIeHz" crossorigin="anonymous"></script>
11     <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/sweetalert2@11"></script>
12     <script src="https://cdn.jsdelivr.net/npm/sweetalert2@11"></script>
13     <title>Tienda ADSO</title>
14 </head>
```

En la imagen anterior se muestra el bloque head del documento html donde se han importado las librerías para usar como **Bootstrap**, **Font-awesome** y **sweetalert** para los estilos. A continuación se muestra el encabezado de la tabla html que va a contener los productos así como un botón en la parte superior para ir a agregar un producto.

```
15 <body>
16     <div class="container w-50">
17         <h2 class="text-center fw-bold">GESTIÓN PRODUCTOS</h2>
18         <a href="/agregar">
19             <button class="btn btn-secondary"> Agregar </button>
20         </a>
21         <table class="table table-bordered mt-2">
22             <thead class="thead-dark">
23                 <tr class="text-center">
24                     <th>Código</th>
25                     <th>Nombre</th>
26                     <th>Precio</th>
27                     <th>Categoría</th>
28                     <th>Foto</th>
29                     <th>Acción</th>
30                 </tr>
31             </thead>
```



El siguiente código incorpora código Python para recorrer la lista o tupla que retorna el servidor con los productos. Aquí se utiliza la librería **jinja2** la cuál es la que permite incorporar código Python en documentos html. En la última columna se incorporan dos **iconos** mediante la librería **Font-awesome** para que respondan a las acciones de editar o eliminar un producto.

```
32         <tbody>
33             {% for producto in listaproductos %}
34                 <tr>
35                     <td>{{producto[1]}}</td>
36                     <td>{{producto[2]}}</td>
37                     <td>{{producto[3]}}</td>
38                     <td>{{producto[5]}}</td>
39                     <td></td>
40                     <td class="text-center">
41                         <i class="fa fa-edit text-warning"></i>
42                         <i class="fa fa-trash text-danger"></i>
43                     </td>
44                 </tr>
45             {%endfor%}
46         </tbody>
47     </table>
48 </div>
49 <script src="../../static/js/app.js"></script>
50 </body>
51 </html>
```

16. Ejecutar el servidor para verificar que se encuentre hasta el momento funcionando bien.

```
PS C:\Users\AdminSena\Documents\SENA2025\GRUPOS\ADS02874057\Actividades\CONSTRUCCIÓN DEL SOFTWARE\PROYECTOS PYTHON\CRUDPYTHON\ts/python.exe" "c:/Users/AdminSena/Documents/SENA2025/GRUPOS/ADS02874057/Actividades/CONSTRUCCIÓN DEL SOFTWARE/PROYECTOS PYTHON/app.py"
* Serving Flask app 'app'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 990-526-079
```

Y la interfaz:

127.0.0.1:5000

Nueva versión de Chrome

GESTIÓN PRODUCTOS

Agregar

Código	Nombre	Precio	Categoría	Foto	Acción
--------	--------	--------	-----------	------	--------



17. Ahora vamos a realizar el proceso de Agregar un producto a la base de datos. Inicialmente vamos a crear una vista con el nombre **frmAgregar.html** que contiene un formulario donde se ingresan los datos del producto. Recuerden que el documento debe ser creado en la carpeta templates.

No olvidar, que como se va a enviar una imagen es necesario agregar el atributo **enctype** del formulario como **multipart/form-data**. Desde este archivo vamos a llamar al archivo app.js el cual va a contener una función que permite mostrar la imagen que se selecciona en el campo de la foto.

Código de la función en el archivo app.js

```
static > js > JS app.js > eliminar
1
  Tabnine | Edit | Test | Explain | Document
2 function mostrarImagen(evento){
3   const archivos = evento.target.files
4   const archivo = archivos[0]
5   const url = URL.createObjectURL(archivo)
6   document.getElementById("imagenProducto").src=url
7 }
8
```




```
13 <body>
14   <div class="container w-50" style="margin: 0 auto">
15     <h2 class="text-center fw-bold">REGISTRAR PRODUCTOS</h2>
16     <form action="/agregar" enctype="multipart/form-data" method="post" class="was-validated">
17       <div class="mt-2">
18         <label for="txtCodigo" class="fw-bold">Código:</label>
19         <input type="number" name="txtCodigo" id="txtCodigo"
20           class="form-control" value="{{producto[0]}}" required>
21       </div>
22       <div class="mt-2">
23         <label for="txtNombre" class="fw-bold">Nombre:</label>
24         <input type="text" name="txtNombre" id="txtNombre"
25           class="form-control" value="{{producto[1]}}" required>
26       </div>
27       <div class="mt-2">
28         <label for="txtPrecio" class="fw-bold">Precio:</label>
29         <input type="number" name="txtPrecio" id="txtPrecio"
30           class="form-control" value="{{producto[2]}}" required>
31       </div>
32       <div class="mt-2">
33         <label for="txtPrecio" class="fw-bold">Precio:</label>
34         <select name="cbCategoria" id="cbCategoria"
35           class="form-select" value="{{producto[3]}}" required>
36           <option value="">Selecione</option>
37           <option value="Electrodomestico">Electrodomestico</option>
38           <option value="Ropa">Ropa</option>
39           <option value="Calzado">Calzado</option>
40         </select>
41       </div>
42       <div class="mt-2">
43         <label for="fileFoto" class="fw-bold">Foto:</label>
44         <input type="file" name="fileFoto" id="fileFoto"
45           class="form-control" onchange="mostrarImagen(event)" required>
46       </div>
47       <div class="mt-2 text-center">
48         <img id="imagenProducto" src="" alt="" width="80" height="80">
49       </div>
50       <div class="mt-2">
51         <button class="btn btn-success">Agregar</button>
52         <button class="btn btn-warning">Cancelar</button>
53       </div>
54     </form>
55   </div>
56
57   <script src="../static/js/app.js"></script>
58   <script> document.getElementById('cbCategoria').value='{{producto[3]}}' </script>
59
60   {% if mensaje %}
61     <script>
62       swal.fire("Actualizar Producto", '{{mensaje}}','info')
63     </script>
64   {% endif %}
65 </body>
66 </html>
```

En el código anterior podemos darnos cuenta en la **línea 45**, como se agrega al input del control de tipo file un evento llamado **onchange**, el cual llama a la función **mostrarImagen** que se encuentra en el archivo **app.js**.



En la propiedad action del formulario se ha colocado **action="/agregar"**, ruta que vamos a crear en Python para recibir la petición **post** para agregar el formulario.

Los campos del formulario tienen por defecto un **value=producto[posición]**. Dicho valor se carga cuando se presentan problemas al agregar, el servidor retorna el producto que se pretende agregar y dichos valores se utilizan para volver a colocar los datos en el formulario.

18. Crear la ruta "/agregar" en el archivo productoController.py para que reciba la petición del cliente.

```
Tabnine | Edit | Test | Explain | Document
28 @app.route("/agregar", methods=['GET', 'POST'])
29 def agregar():
30     """_summary_
31         Recibe peticiones del front-end de tipo GET y
32         Post para agregar un Producto a la base de datos.
33         Cuando es GET muestra el formulario y cuando es
34         POST recibe los datos del formulario para
35         agregar el producto
36     Returns:
37         _type_: Si se agrega bien el producto, redirecciona
38         a la ruta "/" y sino retorna a la vista de
39         agregar el producto con una tupla (producto) con el producto
40         para poder visualizar lo que se había ingresado
41         en el formulario.
42     """
43     producto=None

44     if request.method=='GET':
45         return render_template("frmAgregar.html", producto=producto)
46     elif(request.method=='POST'):
47         #obtener los datos que vienen del formulario
48         codigo = int(request.form['txtCodigo'])
49         nombre = request.form['txtNombre']
50         precio = int(request.form['txtPrecio'])
51         categoria = request.form['cbCategoria']
52         foto = request.files['fileFoto']
53         #obtener el nombre del campo de tipo file
54         nombreArchivo = secure_filename(foto.filename)
55         listaNombreArchivo = nombreArchivo.rsplit('.', 1)
56         #crear la extensión del archivo en minúscula
57         extension = listaNombreArchivo[1].lower()
58         #crear el nombre del archivo como se va a guardar en el servidor
59         # se utiliza codigo.extension, ya que el código no se repite
60         nuevoNombre = str(codigo) + "." + extension
61         rutaFoto = os.path.join(app.config['UPLOAD_FOLDER'], nuevoNombre)
```



```
62 try:
63     producto = (codigo,nombre,precio,nuevoNombre,categoria)
64     cursor = miConexion.cursor()
65     consulta="insert into productos values(null, %s, %s, %s, %s,%s)"
66     cursor.execute(consulta, producto)
67     miConexion.commit()
68     if cursor.rowcount==1:
69         foto.save(rutaFoto) #subimos la foto del producto al servidor
70         return redirect("/")
71 except miConexion.Error as error:
72     miConexion.rollback()
73     mensaje="Ya existe producto con ese código"
74     return render_template("frmAgregar.html",producto=producto, mensaje=mensaje)
```

19. Prueba funcionalidad agregar un producto

127.0.0.1:5000/agregar

REGISTRAR PRODUCTOS


Código:

Nombre:

Precio:


Precio:

Foto:



127.0.0.1:5000

GESTIÓN PRODUCTOS

Código	Nombre	Precio	Categoría	Foto	Acción
11	TV	2500000	Electrodomestico		 



20. **Proceso Editar un producto.** Para ello vamos a crear un formulario muy parecido al de agregar el cual va a recibir los datos al consultar el producto que se quiere editar. Este formulario también va a tener una petición de tipo **POST** para actualizar. El formulario se va a llamar **frmEditar.html** y debe ser creado en la carpeta templates.html. Para iniciar el proceso, vamos a agregar una etiqueta a al icono de editar de la vista listarProductos.html así:

```
<a href="/editar/{{producto[0]}}" > <i class="fa fa-edit text-warning"></i></a>
```

Ahora el código de la ruta **"/editar/id"** en el archivo productoController.py

```
Tabnine | Edit | Test | Explain | Document
77 @app.route("/editar/<int:id>", methods=['GET','POST'])
78 def editar(id):
79     """_summary_
80     Recibe una petición de Tipo GET o POST del
81     front-end para editar un producto de acuerdo
82     a su id, el cual lo recibe en la url. Si es
83     de Tipo GET lo consulta y lo retorna a la vista
84     que permite editar el producto. Si la Petición
85     es de tipo POST recibe de la vista los datos
86     del producto para ser actualizado
87     Args:
88         id (_type_): int, id del producto
89
90     Returns:
91         _type_: LO redirecciona a la vista inicial "/"
92         si se actualiza bien, de lo contrario retorna a
93         la misma vista con el producto para ser visualizado
94         y con un mensaje informando el error.
95     """
96
97     if request.method=="GET":
98         try:
99             datos=(id,)
100             cursor=miConexion.cursor()
101             consulta="select * from productos where idProducto=%s"
102             cursor.execute(consulta,datos)
103             producto = cursor.fetchone()
104             if(producto):
105                 return render_template("frmEditar.html", producto=producto)
106             except miConexion.Error as error:
107                 mensaje="Problemas de conexión a la base de datos"
108                 return render_template("frmEditar.html", producto=producto, mensaje=mensaje)
```



```
108 elif(request.method=='POST'):
109     mensaje=None
110     codigo = int(request.form['txtCodigo'])
111     nombre = request.form['txtNombre']
112     precio = int(request.form['txtPrecio'])
113     categoria = request.form['cbCategoria']
114     foto = request.files['fileFoto']
115     nombreArchivo = secure_filename(foto.filename)
116     try:
117         cursor = miConexion.cursor()
118         #si no llega nada en foto no se actualiza el campo de la foto
119         if nombreArchivo == "":
120             #tupla con los datos del producto
121             producto = (codigo,nombre,precio,categoria, id)
122             consulta = "update productos set proCodigo=%s, proNombre=%s, \
123             proPrecio=%s, proCategoria=%s where idProducto=%s"
124             cursor.execute(consulta, producto)
125             miConexion.commit()
126             if cursor.rowcount==1:
127                 return redirect("/")
128             else:
129                 mensaje="Problemas al actualizar sin foto"
130
131     else:
132         #se actualiza el campo foto
133         listaNombreArchivo = nombreArchivo.rsplit('.', 1)
134         extension = listaNombreArchivo[1].lower()
135         nuevoNombre = str(codigo) + "." + extension
136         #tupla con los datos del producto a actualizar
137         producto = (codigo,nombre,precio,categoria,nuevoNombre, id)
138         rutaFoto = os.path.join(app.config['UPLOAD_FOLDER'], nuevoNombre)
139         consulta = "update productos set proCodigo=%s, proNombre=%s, \
140         proPrecio=%s, proCategoria=%s, proFoto=%s where idProducto=%s"
141         cursor.execute(consulta, producto)
142         miConexion.commit()
143         if cursor.rowcount==1:
144             #subimos la foto del producto al servidor
145             foto.save(rutaFoto)
146             return redirect("/")
147         else:
148             mensaje="problemas al actualizar con foto"
149     except miConexion.Error as error:
150         miConexion.rollback()
151         mensaje="Problemas al actualizar. Revisar \
152         código del producto, puede estar repetido"
153
154     #retorno a la misma vista cuando hay problemas al actualizar
155     producto=(id, codigo, nombre,precio,nombreArchivo,categoria)
156     return render_template("frmEditar.html",producto=producto, mensaje=mensaje)
```







21. Prueba funcionalidad **editar un producto**



GESTIÓN PRODUCTOS

Agregar




Código	Nombre	Precio	Categoría	Foto	Acción
11	TV	2500000	Electrodomestico		  

Colocamos mouse encima icono para editar y vemos como en la parte inferior nos muestra la url a la que direcciona si damos clic.



GESTIÓN PRODUCTOS

Agregar

Código	Nombre	Precio	Categoría	Foto	Acción
11	TV	2500000	Electrodomestico		 

27.0.0.1:5000/editar/1



Damos clic en el icono y nos muestra el formulario con los datos del producto para que se editen lo que se requiera. Como ejercicio vamos a cambiar el precio de 2500000 a 3500000



EDITAR PRODUCTO

Código:
11 ✓

Nombre:
TV ✓

Precio:
2500000 ✓

Categoría:
Electrodomestico ✓ ▾

Foto:
Seleccionar archivo Sin archi...cionados ✓



Editar

Cancelar

EDITAR PRODUCTO

Código:
11 ✓

Nombre:
TV ✓

Precio:
3500000 ✓

Categoría:
Electrodomestico ✓ ▾

Foto:
Seleccionar archivo Sin archi...cionados ✓



Editar

Cancelar



Damos clic en Editar y nos debe llevar a la lista de los productos donde se pueda visualizar los cambios.



GESTIÓN PRODUCTOS

Agregar

Código	Nombre	Precio	Categoría	Foto	Acción
11	TV	3500000	Electrodomestico		



22. **Proceso Eliminar un producto:** Para realizar el proceso partimos del archivo **listarProductos.html** donde le agregamos el **evento click** al icono de eliminar, el cual llama a una función javascript llamada **eliminar**, donde se pasa el **id del producto** a eliminar. Dicha función muestra una ventana modal creada por sweetalert donde verifica si de verdad se desea eliminar.

```
<i class="fa fa-trash text-danger" onclick="eliminar('{{producto[0]}}')"></i>
```

```
function eliminar(id){
  Swal.fire({
    title: 'Eliminar Producto',
    text: "¿Está seguro de eliminar?",
    icon: 'warning',
    showCancelButton: true,
    confirmButtonColor: '#3085d6',
    cancelButtonColor: '#d33',
    cancelButtonText: 'NO',
    confirmButtonText: 'SI'
  }).then((result) => {
    if (result.isConfirmed) {
      location.href="/eliminar/"+id
    }
  })
}
```



La función javascript llama a la ruta **“eliminar/id”** si de verdad se desea eliminar.

Código de la ruta **eliminar/id** en el archivo **productoController.py**

```
@app.route("/eliminar/<int:id>", methods=['GET'])
def eliminar(id):
    """_summary_
        Elimina un producto de acuerdo a su id que
        recibe mediante petición GET
    Args:
        id (_type_): int, id del producto
    Returns:
        _type_: lo redirecciona a la ruta raíz
    """
    if request.method == 'GET':
        try:
            productoEliminar=(id,)
            cursor = miConexion.cursor()
            #datos para eliminar foto
            consulta="select proFoto from productos where idProducto=%s"
            cursor.execute(consulta,productoEliminar)
            foto=cursor.fetchone()[0] #obtenemos el contenido de la posición 0
            rutaFoto = os.path.join(app.config['UPLOAD_FOLDER'], foto)
            consulta="delete from productos where idProducto=%s"
            cursor.execute(consulta, productoEliminar)
            miConexion.commit()
            if cursor.rowcount == 1:
                #eliminar el archivo fisicamente
                os.remove(rutaFoto)
            except miConexion.Error as error:
                miConexion.rollback()
        return redirect("/")
```




23. Prueba **Funcionalidad Eliminar**: Vamos a eliminar la camisa de acuerdo al listado de los productos.

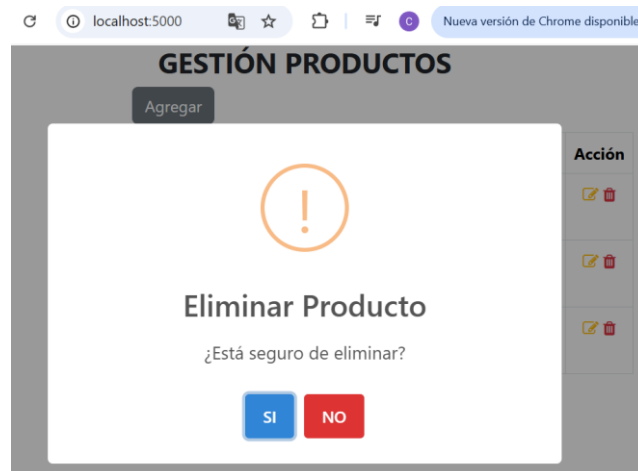
localhost:5000 Nueva versión de Chrome disponible

GESTIÓN PRODUCTOS

Agregar

Código	Nombre	Precio	Categoría	Foto	Acción
11	TV	3500000	Electrodomestico		
12	Camisa	123200	Ropa		
13	Botas	344000	Calzado		

Clic en el icono de eliminar de la camisa. Damos clic en **SI**



Y nos muestra el listado de productos ya actualizado.

localhost:5000 Nueva versión de Chrome disponible

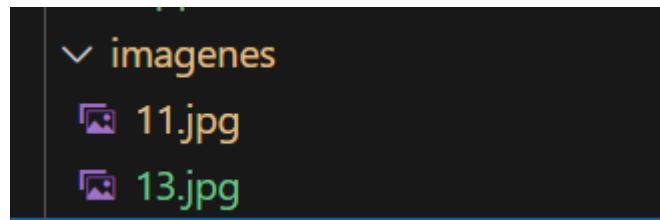
GESTIÓN PRODUCTOS

Agregar

Código	Nombre	Precio	Categoría	Foto	Acción
11	TV	3500000	Electrodomestico		
13	Botas	344000	Calzado		



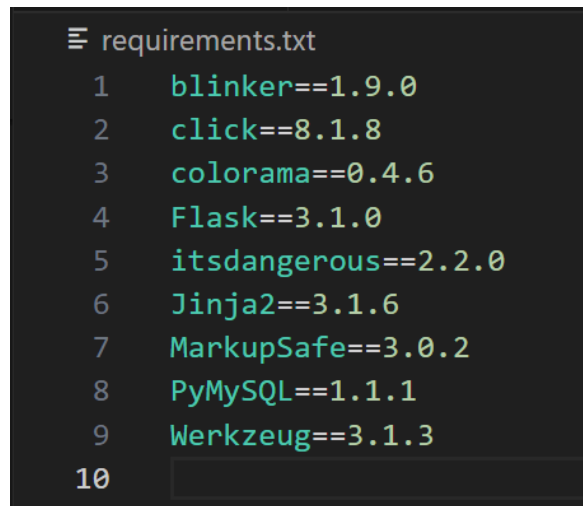
Se verifica en la carpeta de las imágenes que ya no existe el archivo llamado **12.extensión** del producto que acabamos de eliminar.



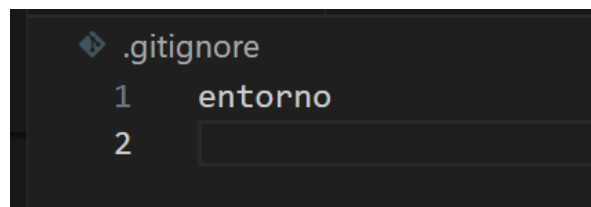
24. Crear un archivo llamado **requirements.txt** en la carpeta raíz, para copiar ahí el nombre de las librerías utilizadas en mi proyecto. Para copiar el nombre de las librerías ejecutar el siguiente comando en la terminal: **pip freeze > requirements.txt**

```
(entorno) C:\Users\AdminSena\Documents\SENA2025\GRUPOS\ADS02874057\YTHON>pip freeze > requirements.txt
```

25. Verificamos el contenido del archivo **requirements.txt**



26. Vamos a crear un archivo en la carpeta raíz del proyecto con el nombre de **.gitignore** para aquí colocar lo que no quiero sincronizar con el repositorio remoto. En nuestro caso vamos a excluir la carpeta entorno. Nuestro archivo **.gitignore** quedaría así:





27. Sincronizar el repositorio local con el remoto. Vamos a ejecutar los siguientes comandos:

git add .

git commit -m "Primera carga de la aplicación"

git remote add crudPython <https://github.com/CesarMCuellarCha/TiendaAdso.git>

git push crudPython master

```
(entorno) C:\Users\AdminSena\Documents\SENA2025\GRUPOS\ADS02874057\
\CRUDPYTHON>git add .
```

```
(entorno) C:\Users\AdminSena\Documents\SENA2025\GRUPOS\ADS02874057\A
ON\CRUDPYTHON>git commit -m "Actualización del CRUD"
[master 340ba2e] Actualización del CRUD
13 files changed, 182 insertions(+), 32 deletions(-)
create mode 100644 static/css/app.css
delete mode 100644 static/imagenes/10.jpg
delete mode 100644 static/imagenes/12.png
create mode 100644 static/imagenes/13.jpg
```

```
(entorno) C:\Users\AdminSena\Documents\SENA2025\GRUPOS\ADS02874057\Actividades\CONSTRUCCIÓN
ON\CRUDPYTHON>git remote add crudPython https://github.com/CesarMCuellarCha/TiendaAdso.git
```

```
(entorno) C:\Users\AdminSena\Documents\SENA2025\GRUPOS\ADS02874057\Actividades\O
ON\CRUDPYTHON>git push crudPython master
Enumerating objects: 34, done.
Counting objects: 100% (34/34), done.
Delta compression using up to 8 threads
Compressing objects: 100% (15/15), done.
Writing objects: 100% (19/19), 8.90 KiB | 911.00 KiB/s, done.
Total 19 (delta 3), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (3/3), completed with 3 local objects.
To https://github.com/CesarMCuellarCha/TiendaAdso.git
46c2f5b..340ba2e master -> master
```



Verificamos el repositorio remoto

 CesarMCuellarCha Actualización del CRUD		340ba2e · 3 minutes ago	 2 Commits
 __pycache__	Actualización del CRUD	3 minutes ago	
 controladores	Actualización del CRUD	3 minutes ago	
 static	Actualización del CRUD	3 minutes ago	
 templates	Actualización del CRUD	3 minutes ago	
 .gitignore	primer cargue de la aplicación	2 days ago	
 app.py	Actualización del CRUD	3 minutes ago	
 requirements.txt	primer cargue de la aplicación	2 days ago	

CONTROL DEL DOCUMENTO

	Nombre	Cargo	Dependencia	Fecha
Autor (es)	César Marino Cuéllar Chacón	Instructor	CIES-NEIVA	09-03-2025