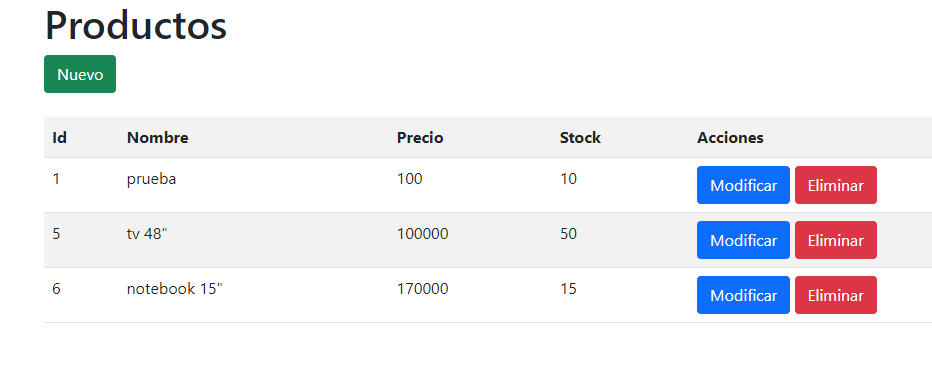
C.R.U.D. de Productos

Create Read Update Delete

Instructivo para proyecto API REST de CRUD de productos con las siguientes tecnologías:

HTML5, CSS3, Bootstrap 5, VUE3, MySQL, Python 3.11 y Flask. con Arquitectura de Software: Modelo Vista Controlador. Las tres partes del patrón de diseño de software **MVC** se pueden describir de la siguiente manera: **Modelo**: Maneja datos y lógica de negocios. Vista: Se encarga del diseño y presentación. Controlador: Enruta comandos a los **modelos** y vistas.



Sitio: <https://frontendapirest.netlify.app/productos.html>

Api: <https://mcerda.pythonanywhere.com/productos>

-

Video explicativo del BackEnd para generar la Api que permita los métodos GET,

DELETE, POST y PUT:

GitHub ( backend): <https://github.com/MarcelaCerda/backend_productos.git>

1- Antes que nada, en la consola de Windows, git bash o en la terminal VSC probar:

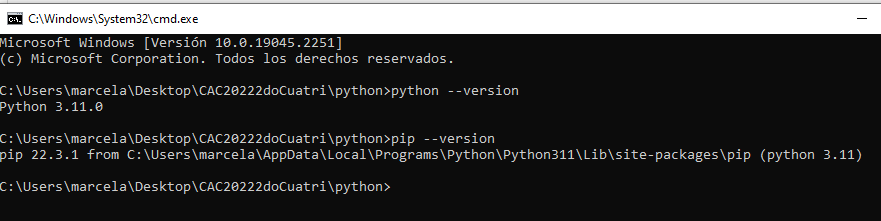
python - -version

pip - -version

Para Mac o linux

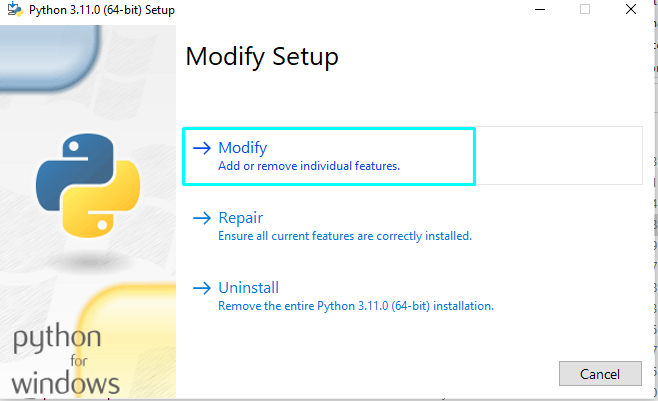
python3 - -version

pip3 - -version

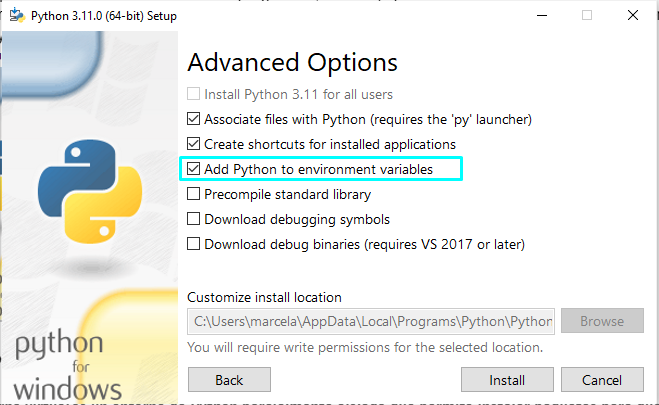


y tiene que mostrar la versión de ambos programas, sino hay que solucionarlo !!!

**Si da error, “No se reconoce como un comando….. “** no van a poder instalar lo necesario para el proyecto, van a tener que ejecutar el instalador de Python y hacer click en Modify



y tildar Add Python to environment variables



y volver al punto 1 y si sigue sin funcionar, en el instalador. presionar Reparar

2- Back End con Python y MySQL que luego subiremos al Hosting PythonAnyWhere

Vamos a crear un entorno virtual de desarrollo para instalar los paquetes del proyecto

Un *entorno virtual* es un entorno de Python parcialmente aislado que permite instalar paquetes para que los use una aplicación en particular, en lugar de instalarlos en todo el sistema

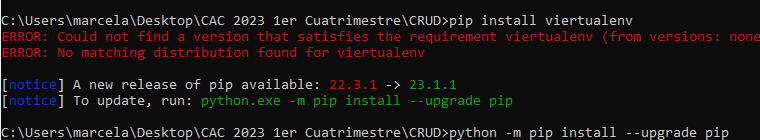
Si quieren seguir leyendo:<https://omes-va.com/virtualenv-python/>

**Si no tienen instalado virtualenv:**

En la consola escribir:

pip install virtualenv

Si da el siguiente error:



ejecutar la última línea

**Vamos a crear el entorno de trabajo**

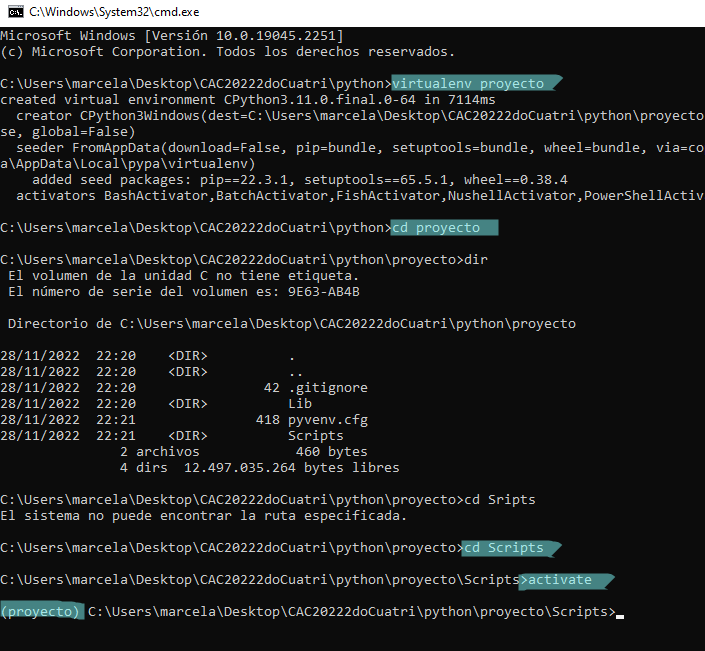
En la consola **nos movernos a la carpeta de trabajo** y si queremos crear un entorno de trabajo virtual en la carpeta que se llame por ejemplo “proyecto” escribimos en la consola del sistema

virtualenv proyecto

notamos que nos creó muchos archivos y carpetas, entre ellas una carpeta “proyecto”. luego

hay que activar el entorno virtual, ejecutando “activate” que se encuentra dentro de la carpeta Scripts en window

Si diera error ejecutar en la carpeta de trabajo: python -m virtualenv proyecto



luego de activar el entorno de trabajo nos lo indica en el recuadro celeste

(proyecto)

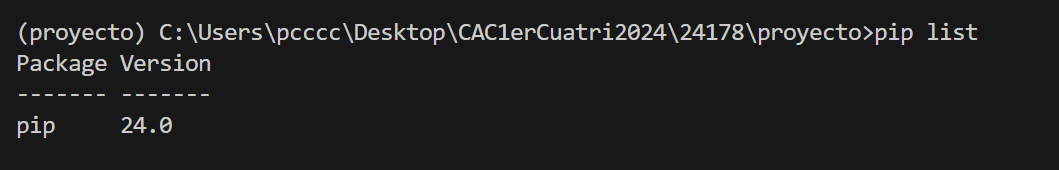
—--------------------------------------------------------------------------------------------------------------------------

**Arranquemos con el BackEnd del proyecto**

Desde VSC abrimos la carpeta que acabamos de crear “proyecto”

Vamos a instalar Flask

Pero antes ejecuta : pip list para ver que hay instalado en esa carpeta



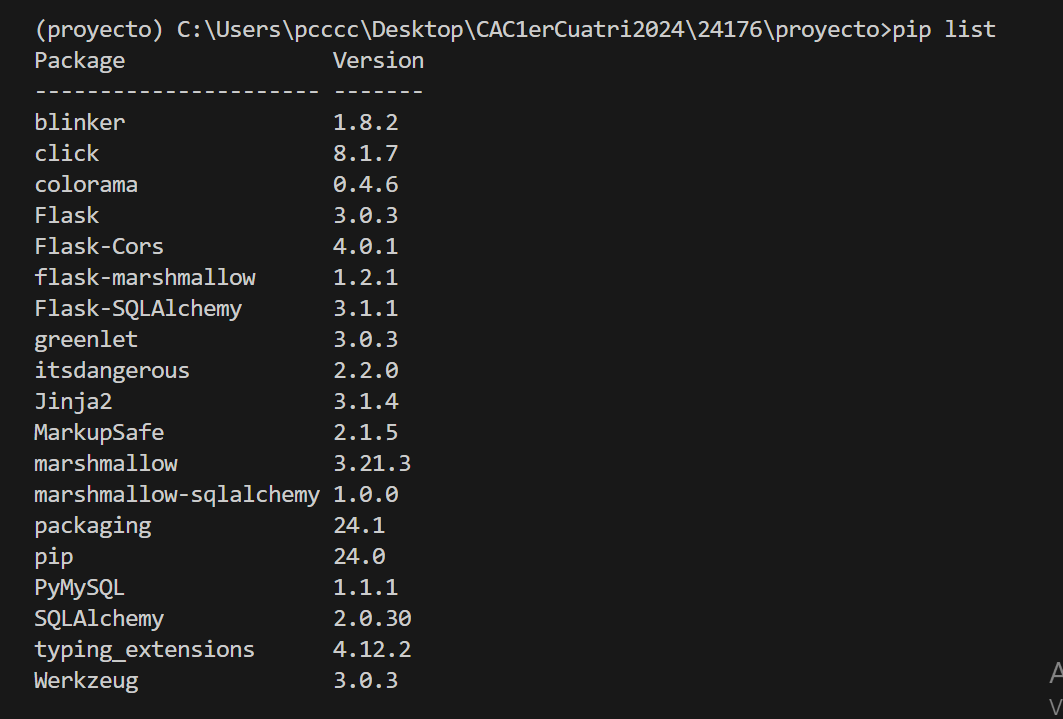
Ir al menú de VSC, abrir la terminal e instalar segun el sistema operativo:

* para windows: pip install flask …….
* para mac: pip3 install flask ……………
* para linux sudo apt install python3-pip ………….

Instalar en la terminal (segun el sistema operativo):

pip install flask flask-sqlalchemy flask-marshmallow marshmallow-sqlalchemy pymysql -U flask-cors

volver a mirar que tienen instalado en esa carpeta



**Flask** es un framework que nos ayuda a ser más productivos con python , bases de datos y la creracion de la api para mas informacion : <https://flask-es.readthedocs.io/>

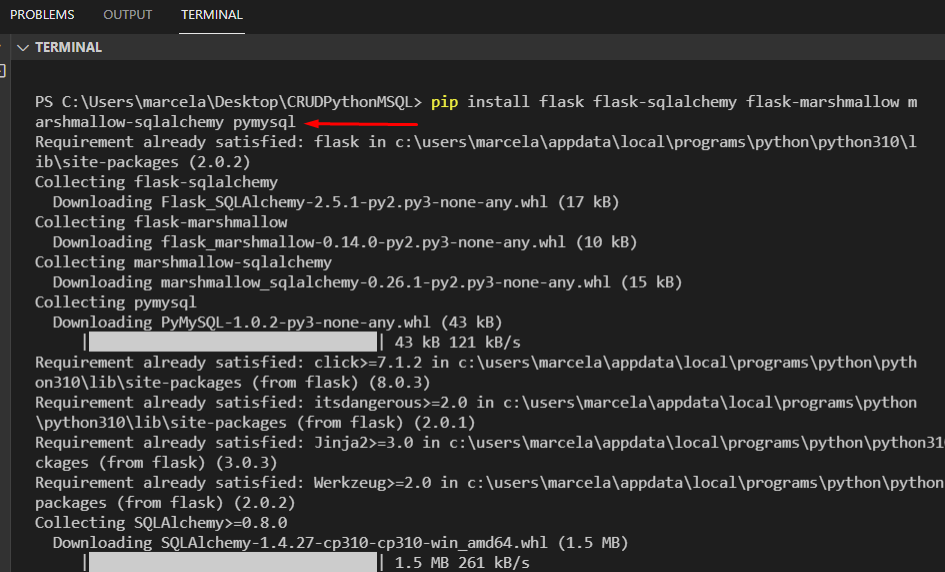
**Marshmallow** es una biblioteca independiente de ORM/ODM/framework para convertir tipos de datos complejos, como objetos, hacia y desde tipos de datos nativos de Python.

para mas informacion <https://marshmallow.readthedocs.io/en/stable/>

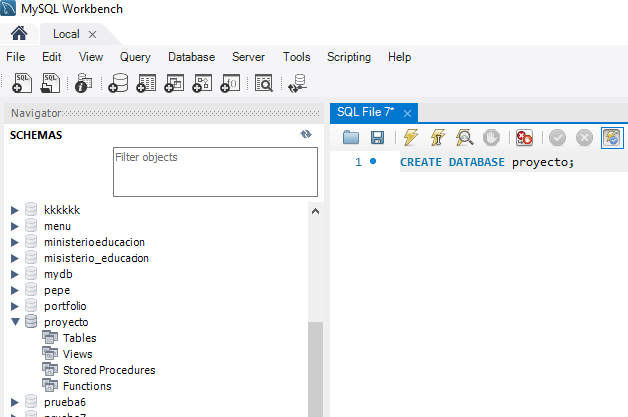
**SQL Alchemy** es un kit de herramientas SQL y un ORM (Object Relational Mapper) que permite a las clases en Python asociarlas a tablas en bases de datos relacionales

para mas informacion: <https://flask-es.readthedocs.io/patterns/sqlalchemy/>

**Cors** me va a permitir acceder desde el frontend al JSON que genera el backend



-Crear una base de datos que se llame por ejemplo proyecto, refrescar y verificar que se haya creado



Volver al proyecto

Dentro de la carpeta “proyecto”, crear un archivo app.py

dentro del archivo app.py copiar lo siguiente:

from flask import Flask ,jsonify ,request

# del modulo flask importar la clase Flask y los métodos jsonify,request

from flask\_cors import CORS # del modulo flask\_cors importar CORS

from flask\_sqlalchemy import SQLAlchemy

from flask\_marshmallow import Marshmallow

app=Flask(\_\_name\_\_) # crear el objeto app de la clase Flask

CORS(app) #modulo cors es para que me permita acceder desde el frontend al backend

# configuro la base de datos, con el nombre el usuario y la clave

app.config['SQLALCHEMY\_DATABASE\_URI']='mysql+pymysql://root:root@localhost/proyecto'

# URI de la BBDD driver de la BD user:clave@URLBBDD/nombreBBDD

app.config['SQLALCHEMY\_TRACK\_MODIFICATIONS']=False #none

db= SQLAlchemy(app) #crea el objeto db de la clase SQLAlquemy

ma=Marshmallow(app) #crea el objeto ma de de la clase Marshmallow

# defino las tablas

class Producto(db.Model): # la clase Producto hereda de db.Model de SQLAlquemy

id=db.Column(db.Integer, primary\_key=True) #define los campos de la tabla

nombre=db.Column(db.String(100))

precio=db.Column(db.Integer)

stock=db.Column(db.Integer)

imagen=db.Column(db.String(400))

def \_\_init\_\_(self,nombre,precio,stock,imagen): #crea el constructor de la clase

self.nombre=nombre # no hace falta el id porque lo crea sola mysql por ser auto\_incremento

self.precio=precio

self.stock=stock

self.imagen=imagen

# si hay que crear mas tablas , se hace aqui

with app.app\_context():

db.create\_all() # aqui crea todas las tablas si es que no estan creadas

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

class ProductoSchema(ma.Schema):

class Meta:

fields=('id','nombre','precio','stock','imagen')

producto\_schema=ProductoSchema() # El objeto producto\_schema es para traer un producto

productos\_schema=ProductoSchema(many=True) # El objeto productos\_schema es para traer multiples registros de producto

# crea los endpoint o rutas (json)

@app.route('/productos',methods=['GET'])

def get\_Productos():

all\_productos=Producto.query.all() # el metodo query.all() lo hereda de db.Model

result=productos\_schema.dump(all\_productos) #el metodo dump() lo hereda de ma.schema y

# trae todos los registros de la tabla

return jsonify(result) # retorna un JSON de todos los registros de la tabla

@app.route('/productos/<id>',methods=['GET'])

def get\_producto(id):

producto=Producto.query.get(id)

return producto\_schema.jsonify(producto) # retorna el JSON de un producto recibido como parametro

@app.route('/productos/<id>',methods=['DELETE'])

def delete\_producto(id):

producto=Producto.query.get(id)

db.session.delete(producto)

db.session.commit() # confirma el delete

return producto\_schema.jsonify(producto) # me devuelve un json con el registro eliminado

@app.route('/productos', methods=['POST']) # crea ruta o endpoint

def create\_producto():

#print(request.json) # request.json contiene el json que envio el cliente

nombre=request.json['nombre']

precio=request.json['precio']

stock=request.json['stock']

imagen=request.json['imagen']

new\_producto=Producto(nombre,precio,stock,imagen)

db.session.add(new\_producto)

db.session.commit() # confirma el alta

return producto\_schema.jsonify(new\_producto)

@app.route('/productos/<id>' ,methods=['PUT'])

def update\_producto(id):

producto=Producto.query.get(id)

producto.nombre=request.json['nombre']

producto.precio=request.json['precio']

producto.stock=request.json['stock']

producto.imagen=request.json['imagen']

db.session.commit() # confirma el cambio

return producto\_schema.jsonify(producto) # y retorna un json con el producto

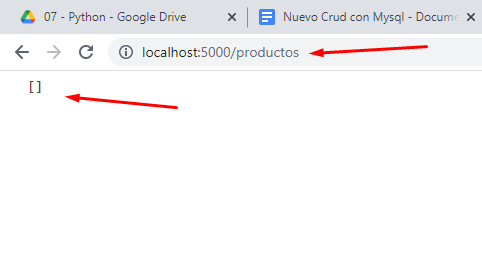
# programa principal \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

if \_\_name\_\_=='\_\_main\_\_':

app.run(debug=True, port=5000) # ejecuta el servidor Flask en el puerto 5000

y correr en la terminal: python app.py

probar en el navegador <http://localhost:5000/productos> , si devuelve [ ] es que esta’ correcto, es el json que contiene una lista vacía donde trajo los datos la tabla productos de la base de datos, que como la tabla fue recién creada, está vacío



cargar en la base de datos 2 o 3 productos y volver a probar en el navegador <http://localhost:5000/productos> y tiene que mostrar el json con los datos que acaban de crear

En mysql para cargar un producto en la base de datos proyecto:

use proyecto;

insert into producto (nombre, precio, stock ,imagen) values ("mouse bluetooth",16000,15,"https://http2.mlstatic.com/D\_NQ\_NP\_610024-MLA44840092138\_022021-O.webp");

select \* from producto;

Probar en el navegador <http://localhost:5000/productos>/1 , y devuelve un Json con el registro con id=1.

Probar en Visual Studio en la extensión Thunder Client o en el programa de escritorio Postman los métodos GET, DELETE, POST y PUT

Video explicativo de como subir el backend al servidor de PythonAnyWhere: <https://youtu.be/GVVjEpuu2d0>

Tutorial de FrontEnd del CRUD en el siguiente enlace: <https://docs.google.com/document/d/14SerlwTDVlL9DX5yMJiNHVhPb84-iFEU/edit?usp=drive_link&ouid=109850938251135584181&rtpof=true&sd=true>

Video explicativo proyecto de Backend con muchas tablas son arquitectura MVC(modelo vista controlador) y como subirlo a PythonAnyWhere: <https://youtu.be/vM3XdRHm8w0>

GitHub BackEnd MVC: