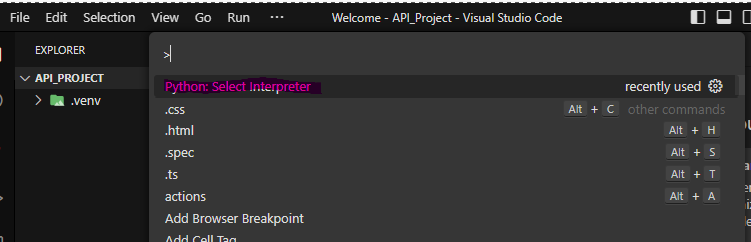
REST API CON FLASK

# Qué se puede hacer con esta REST API en FLASK

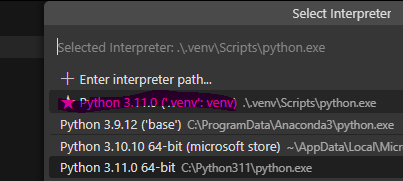
* Crear y sacar información sobre tiendas
* Crear, sacar, buscar, actualizar y eliminar ítems, relacionados en tablas con en esas tiendas
* Guardar los datos con **SQLAlchemy**
* Validar entrada de datos
* Alojar aplicación en Docker para desarrollo

# Instalar Virtual Environment en Visual Studio Code

* Abierta la carpeta donde vamos a trabajar con el proyecto API, abrir terminal CMD y escribir: > py -3.11 -m venv .venv
* Ctrl + Shift + P



* Seleccionar el que está dentro de la venv que hemos creado:



* Reiniciar la terminal (ctrl + J)
* Debería ahora abrirse el CMD con el virtual environment, poner pip install flask en terminal.

# Requerimientos / Funcionalidades añadidas:

* Extensiones de Flask
* **Insomnia**
* Docker para correr la API de forma mas estable. **Es necesario para hacer peticiones con Insomnia correctamene (redirige desde puerto 5005)**
* Flask-Smorest
* Marshmallow – validación de tipos de datos
* Integración SQLite con SQLAlchemy

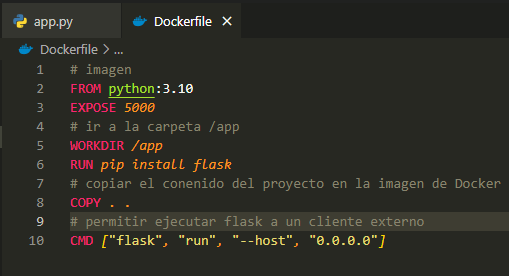
# Docker

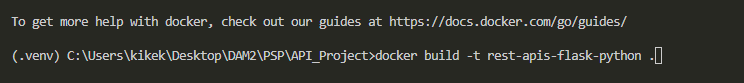
La aplicación flask se ejecutará dentro de un Docker container**. Es necesario tener instalado Docker Desktop y construir una imagen para hacer las requests**

## Construir Imagen Docker

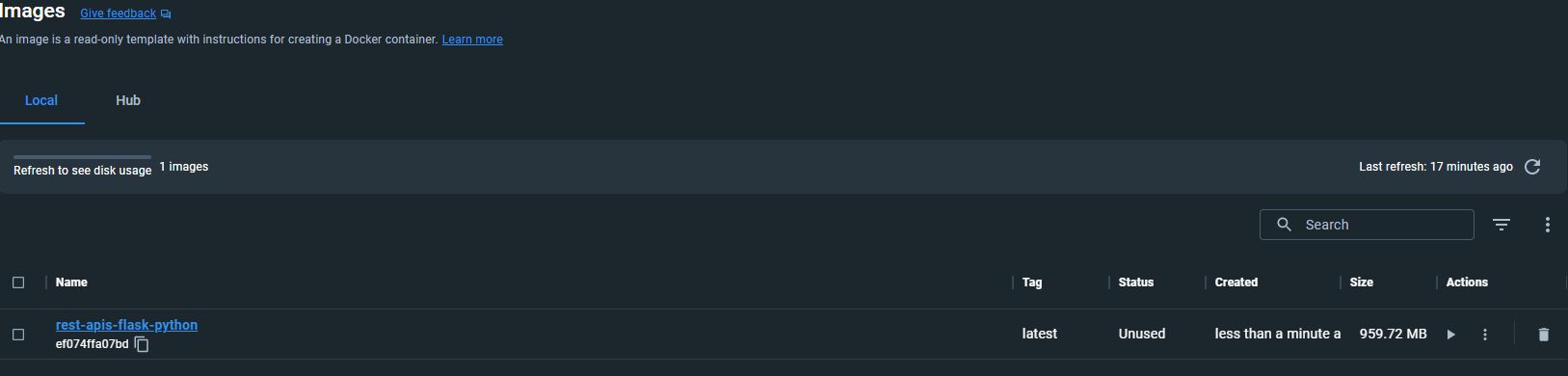
Para ello:

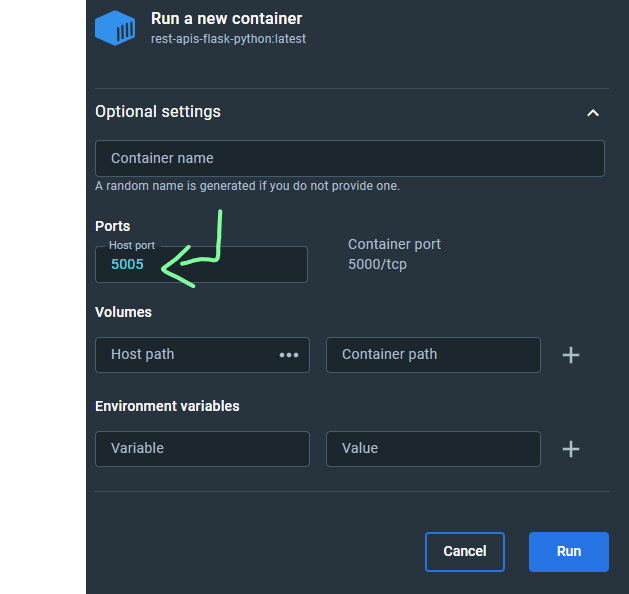
* Tener instalado Docker Desktop
* Tener un dockerfile en el proyecto





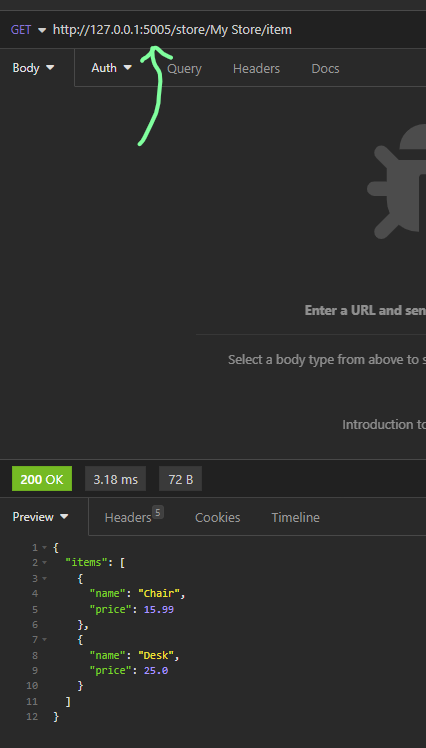
Ahora se puede ejecutar la imagen que contiene la aplicación Flask en el Docker Desktop:



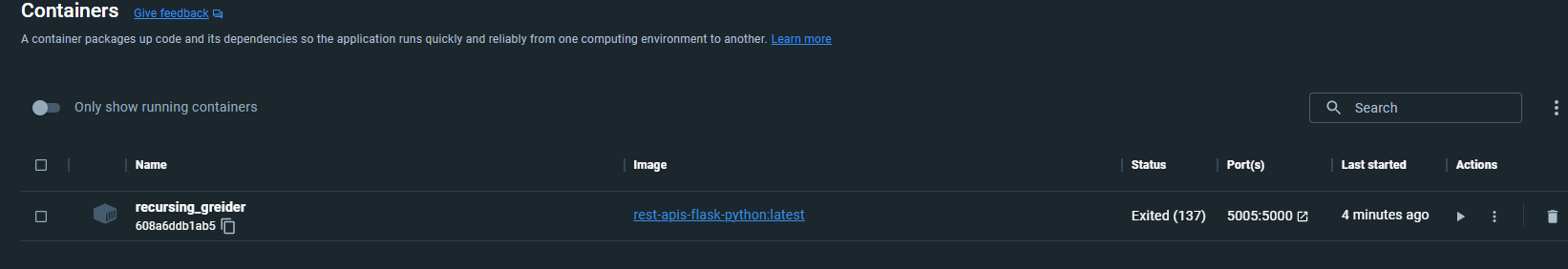




* Si se redireccionan puertos en Docker, se debe usar en el Insomnia para que haga bien las requests



* Ahora se puede ejecutar el Container que a su vez ejecuta la aplicación Flask:



## Ejecutar el Docker con Línea de Comando

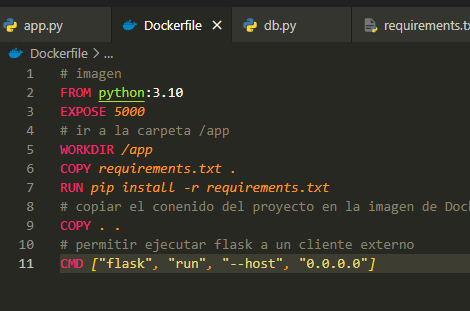
* Sólo se necesita el nombre de la imagen (ya creada):



* Para ejecutar en segundo plano (poder seguir usando la terminal):



## Docker por CLI para que reinicie la app con cambios en el código





* Windows:



* Linux:

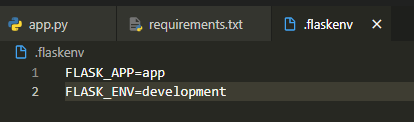


Por ejemplo:

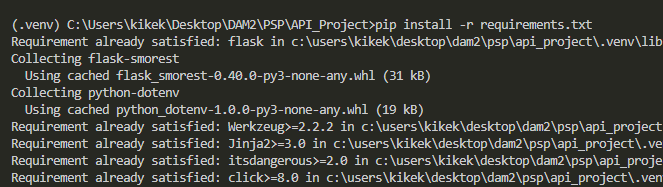
docker run -dp 5000:5000 -w /app -v "/c/Users/kikek/Desktop/DAM2/PSP/API\_Project:/app" flask-smorest-api

# Lanzar la aplicación

* Para lanzar la aplicación, solo necesitamos usar flask-run, ya que el archivo “.flaskenv” hace el resto con estas configuraciones (guardar automáticamente, etc):

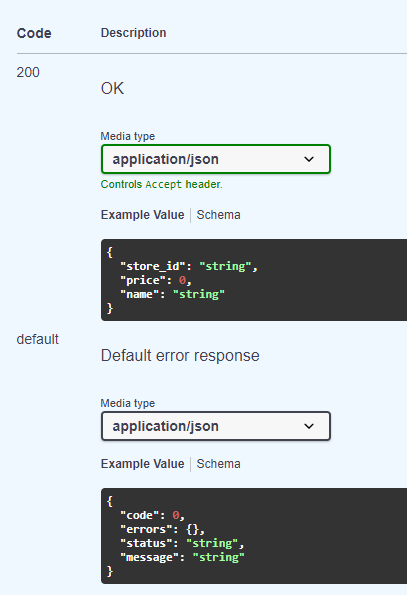


* Antes, hay que instalar todas las librerías:



# Validaciones con Schemas:

* Con Schema, se pueden hacer validaciones más precisas. Para más información de la schema validation de cada request, entra en <http://localhost:5000/swagger-ui> con la aplicación flask corriendo:



# Endpoints de la API

Un asterisco significa que el usuario se ha tenido que autenticar en los últimos días para hacer una petición, dos asteriscos significa que el usuario se habrá tenido que autenticar en los últimos minutos para hacer autenticación

Si no hay asterisco, es que cualquiera puede hacer una petición

## TIENDAS:

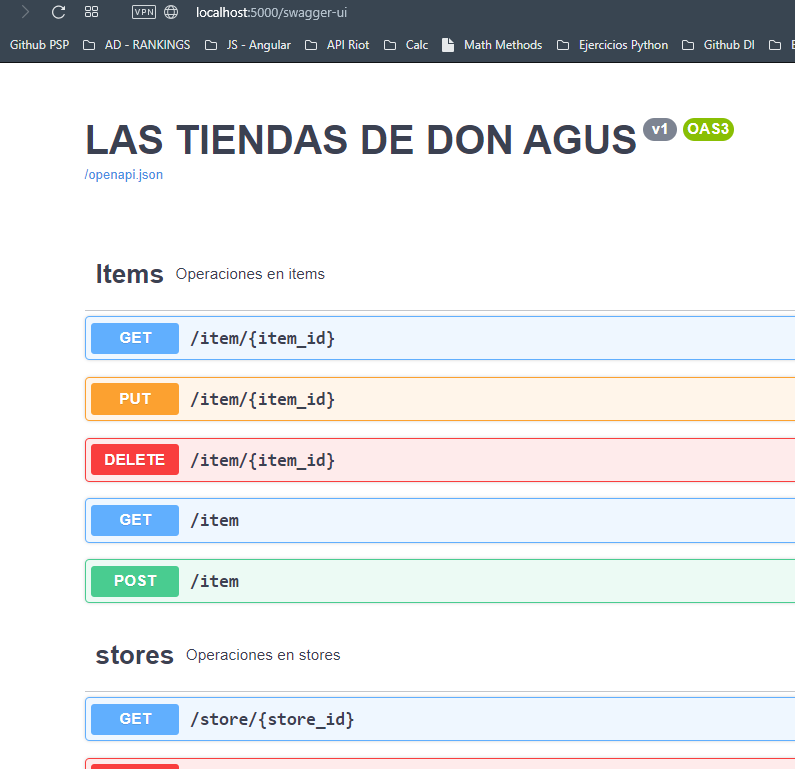
|  |  |  |
| --- | --- | --- |
| METODO | Endpoint | Descripcion |
| GET | /store | Sacar una lista de tiendas |
| POST | /store | Crear una tienda |
| GET | /store{id} | Sacar una tienda, dada su ID |
| DELETE | /store{id} | Borrar una tienda, dada su ID |

## ITEMS:

|  |  |  |
| --- | --- | --- |
| METODO | Endpoint | Descripcion |
| \*GET | /item | Sacar una lista de todos los ítems en todas las tiendas |
| \*\*POST | /item | Crear un nuevo item, dado su nombre y precio en el body de la petición |
| \*GET | /item{id} | Sacar información específica de un item, dada su id |
| PUT | /item{id} | Actualizar un item dada u id única. El nombre del ítem o precio se puede dar en el body de la petición |
| \*DELETE | /item{id} | Eliminar un item, dada su ID |

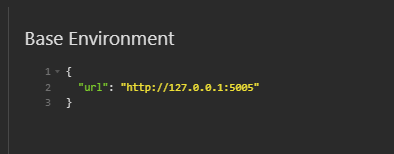
# Hacer Requests con Insomnia

* Se le adjunta el archivo Json para importar todas las requests con Insomnia
* Para más información / documentación sobre los Requests, use : localhost:5000(o 5005)/swagger-ui

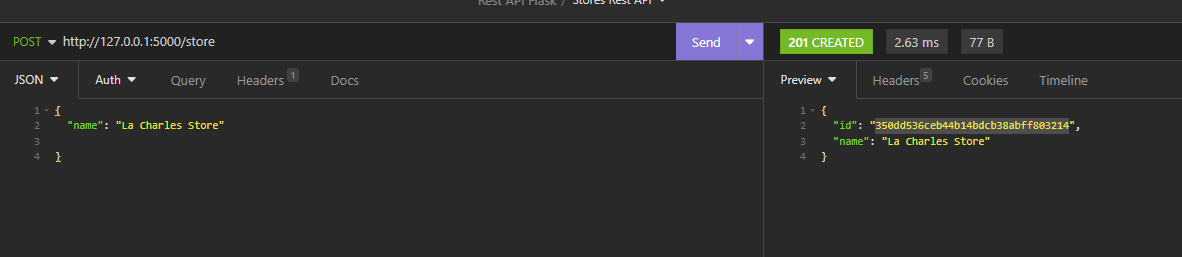


## URL environments:

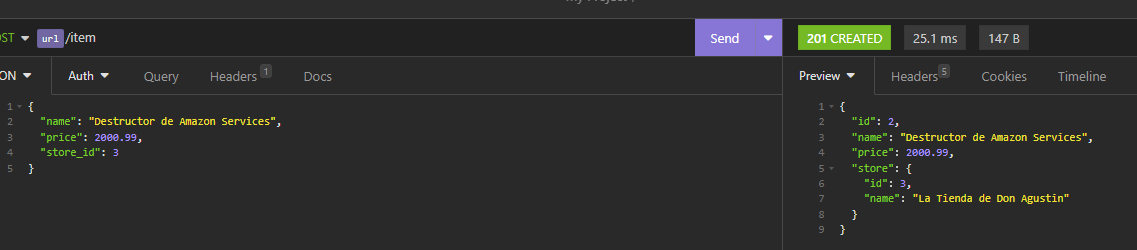
Los endpoints apuntan automáticamente a 127.0.0.1:5005 o al puerto 5000, tal como se haya configurado con el Docker, y se pueden cambiar en el Base Environment en Insomnia



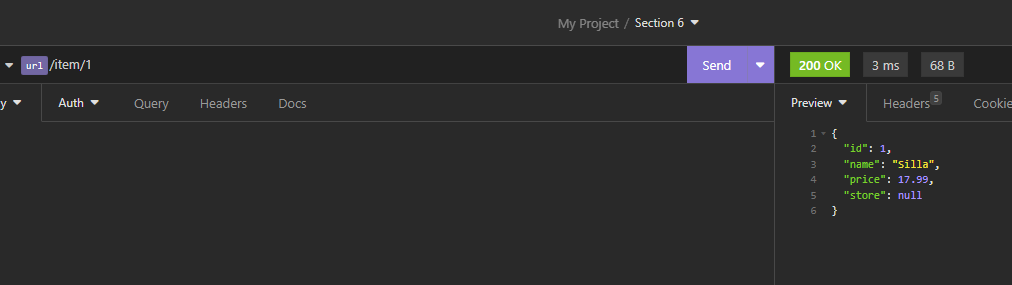
## Crear Tienda:



## Crear Item:

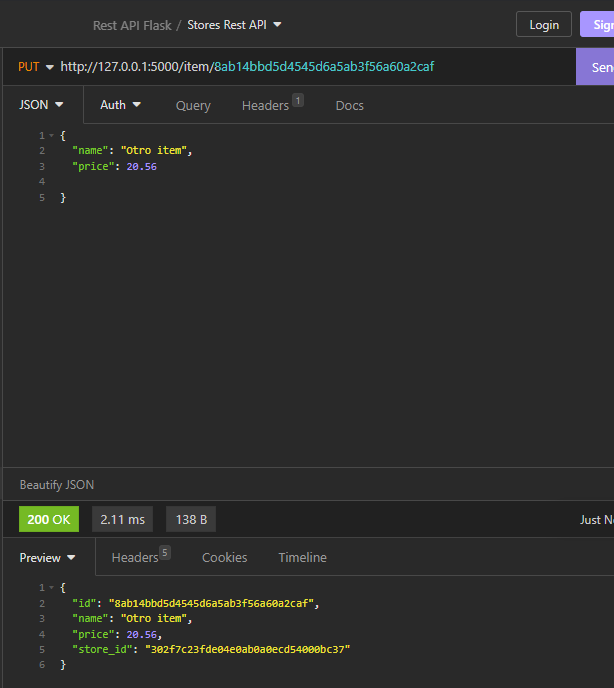


## Get Item por ID:



## Delete Item por ID:

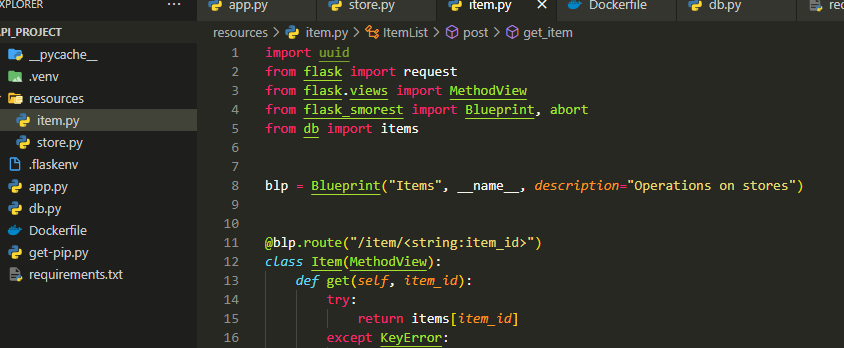
## Actualizar item por ID:



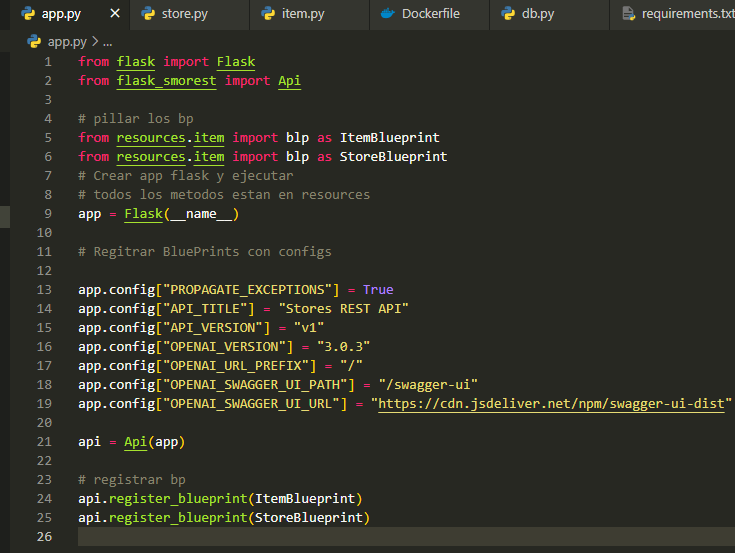
# ¿Dónde están los códigos?

## Métodos con BluePrint y MethodView

* Se encuentran en resources, separados en dos archivos: ítems y stores. Así, se llaman a los métodos de forma más organizada según el endpoint y tipo de dato

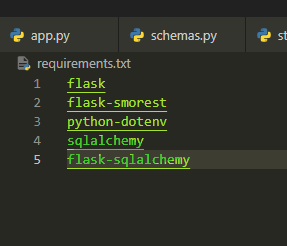


* Y en la app he configurado flask\_smorest, y se importan / registran los BluePrints

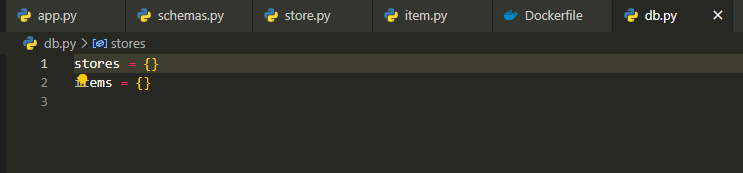


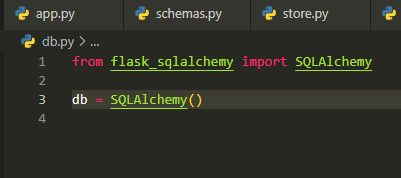
# Integrar SQLAlchemy a nuestra API

* Añadir al requirements

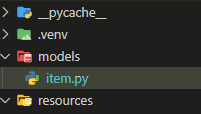


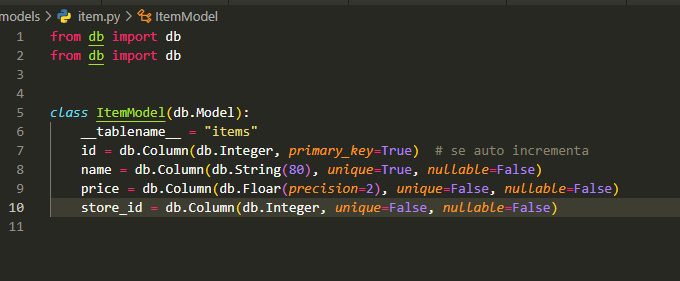
* Pip install -r requirements.txt
* Vamos al archivo que guarda nuestros diccionarios:



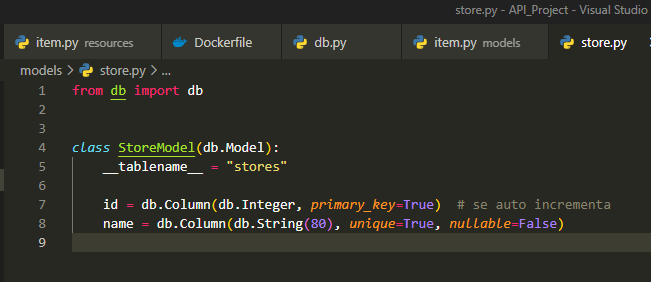


* Crearse carpeta models, y un archivo por cada tabla

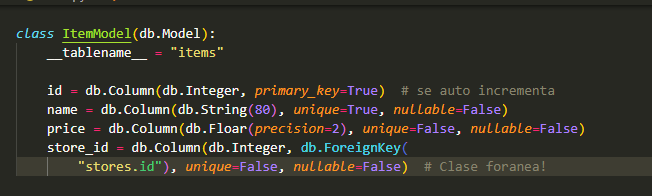




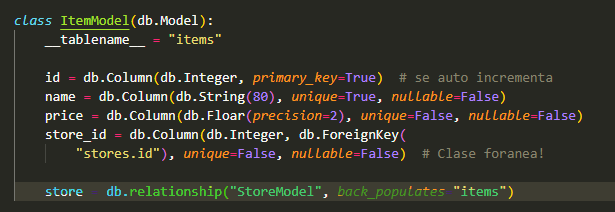
* Tiendas



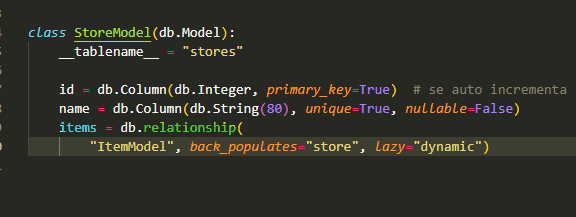
* Como las dos tablas tienen una relación (PK id de tienda es store\_id en item), declaramos clave foránea en store\_id de la tabla item:



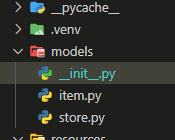
* Añadimos esta línea para poder ver qué tienda está asociada a ese item:
  + Ahora sabe que un item tiene una store\_id, por lo que el item se relaciona con una tienda

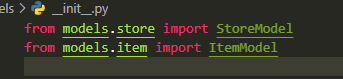


* Enlazamos la relación con la otra tabla:
  + Así sabe que debe buscar todos los ítems que tengan una store\_id igual a la id de esta tabla de tiendas. Lazy dinamic le dice que no haga esa búsqueda a no ser que se lo digamos

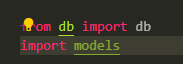


* Me creo un nuevo archivo init en models y me importo las tablas:

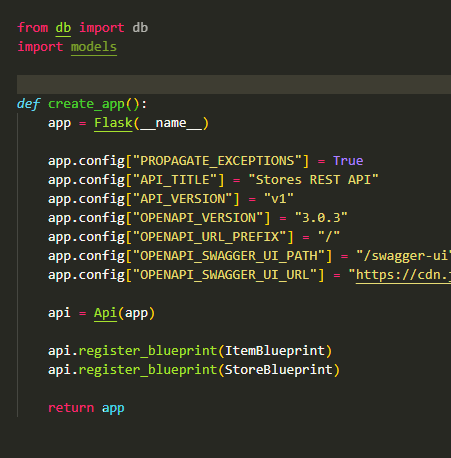




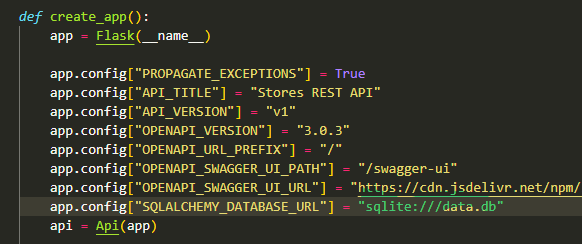
* En app.py, hacer importaciones:



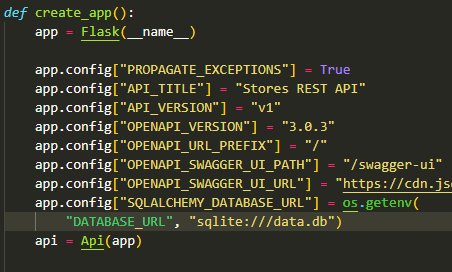
* Se define toda la creación de Flask en un método



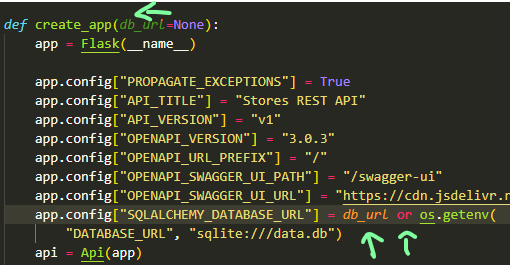
* Añadir esta línea de configuración, conexión al archivo donde se va a guardar los datos:



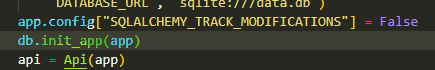
* Importar OS y añadir este parámetro:

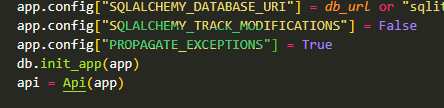


* Añadir esto para futura migración a PostgreSQL:

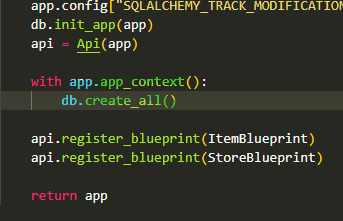


* Colocar estas dos últimas líneas, para conectar la base de datos con la aplicación Flask





* Por último, añadir esto para crear las tablas



* Ahora, al reiniciar la aplicación, se conservarán los datos

App.py

from flask import Flask

from flask\_smorest import Api

from db import db

import models

from resources.item import blp as ItemBlueprint

from resources.store import blp as StoreBlueprint

*def* create\_app(*db\_url*=None):

    app = Flask(\_\_name\_\_)

    app.config["API\_TITLE"] = "Stores REST API"

    app.config["API\_VERSION"] = "v1"

    app.config["OPENAPI\_VERSION"] = "3.0.3"

    app.config["OPENAPI\_URL\_PREFIX"] = "/"

    app.config["OPENAPI\_SWAGGER\_UI\_PATH"] = "/swagger-ui"

    app.config[

        "OPENAPI\_SWAGGER\_UI\_URL"

    ] = "https://cdn.jsdelivr.net/npm/swagger-ui-dist/"

    app.config["SQLALCHEMY\_DATABASE\_URI"] = *db\_url* or "sqlite:///data.db"

    app.config["SQLALCHEMY\_TRACK\_MODIFICATIONS"] = False

    app.config["PROPAGATE\_EXCEPTIONS"] = True

    db.init\_app(app)

    api = Api(app)

    with app.app\_context():

        db.create\_all()

    api.register\_blueprint(ItemBlueprint)

    api.register\_blueprint(StoreBlueprint)

    return app

* Los métodos de requests son actualizados con métodos de query:

