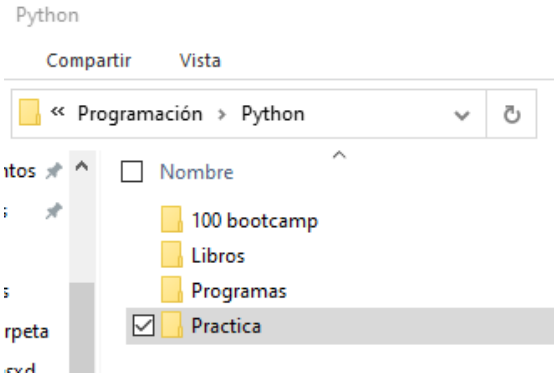
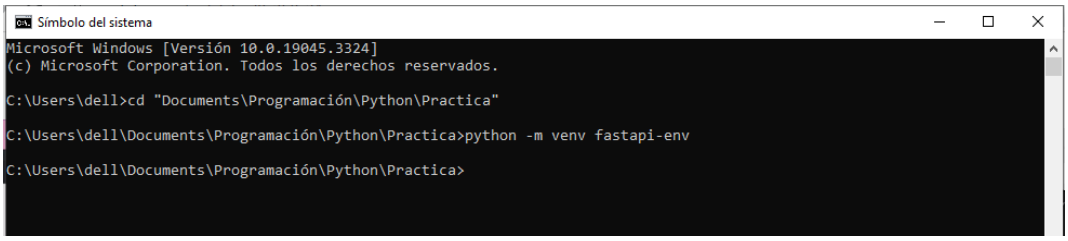
DOCUMENTACIÓN DE LA PRÁCTICA

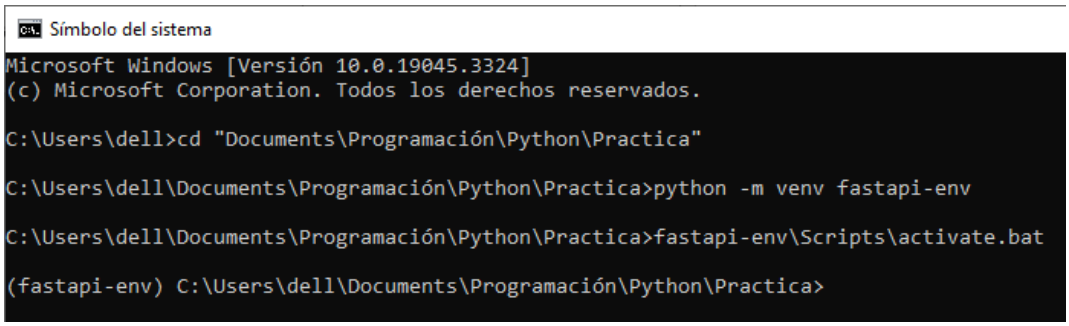
1. Como primer paso se eligió el lenguaje Python, por su versatilidad y una sintaxis sencilla.
2. Se crea una nueva carpeta llamada “Practica”, esto para poder crear un entorno virtual.



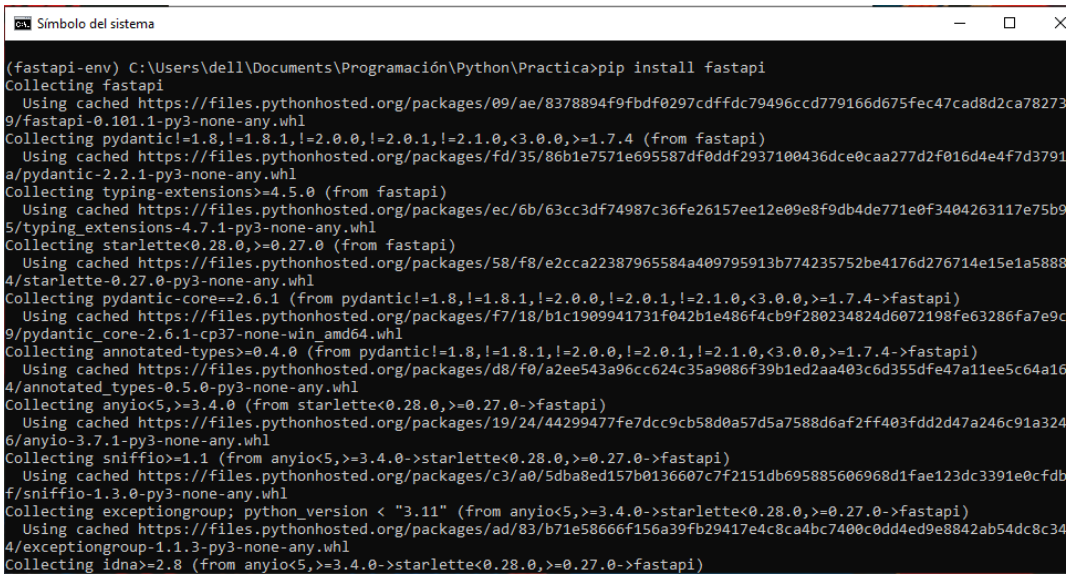
3. Se entra a símbolo del sistema, y se dirige a la carpeta de practicas, y ahí dentro se escribe el código `python -m venv fastapi-env` para crear el entorno virtual.



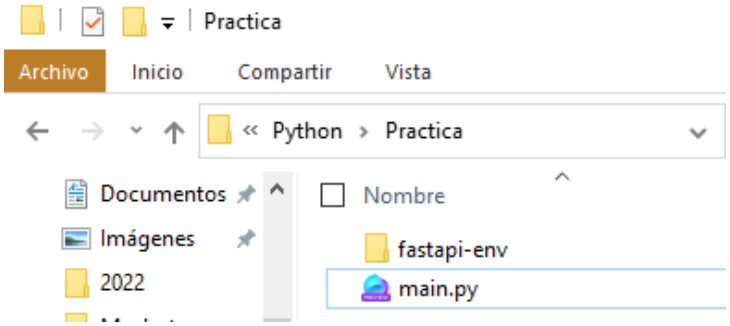
4. Posteriormente se activa el entrono con el código `fastapi-env\Scripts\activate.bat`.



5. Posteriormente se instala FastAPI con el código `pip install fastapi`.



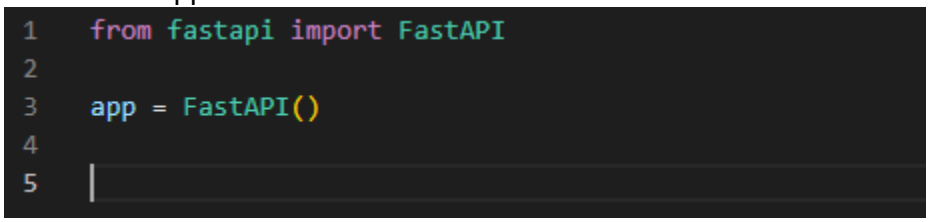
6. Se crea el archivo main.py en el mismo nivel a la carpeta fastapi-env.



7. Dentro del archivo main.py, con el editor Visual Studio Code, se escribirán las líneas de código siguientes

```
from fastapi import FastAPI
app = FastAPI()
```

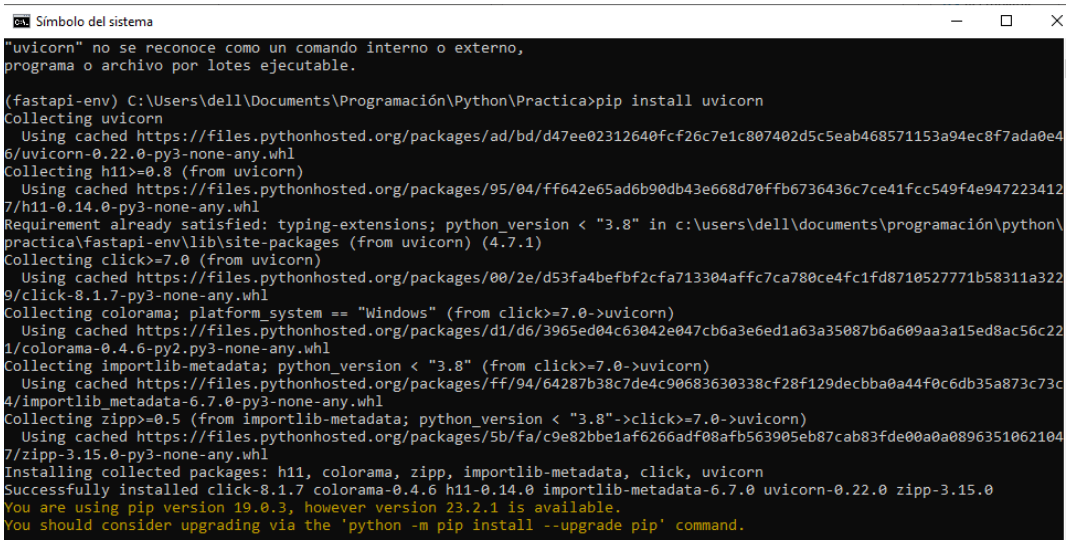
Que significan la importación de FastAPI del módulo fastapi, y luego se instanciará FastAPI en app.



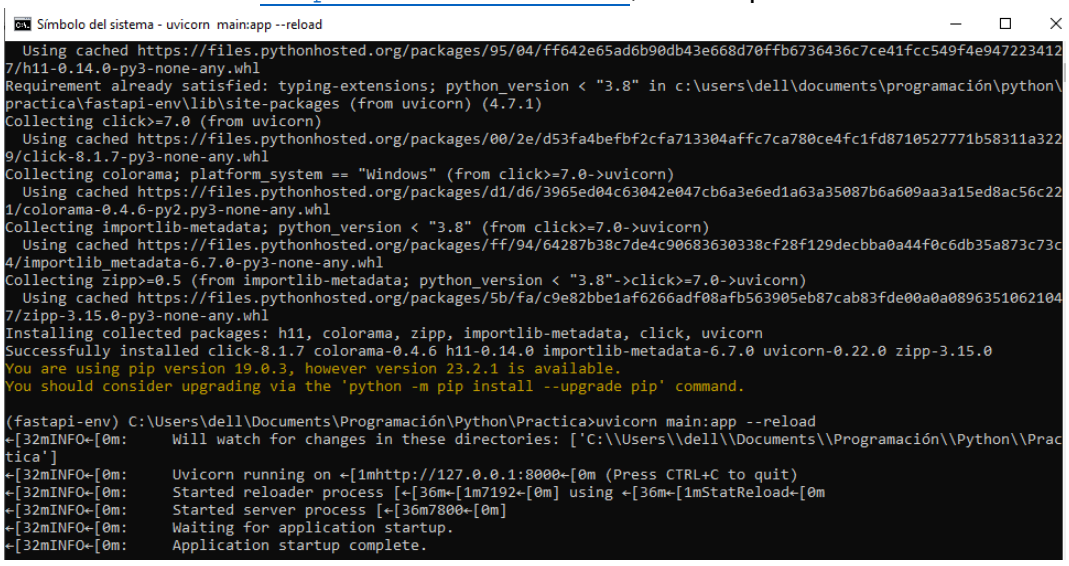
8. Para `@app.get("/")` , ósea la función base con entrada “/”, se programa una salida el diccionario con el mensaje de salida: `“Bienvenidos al reventón musical, elije las canciones que quieras escuchar”.`

```python
@app.get("/")
def index():
    return {"message" : "Bienvenidos al reventón musical, elije las canciones que quieras escuchar."}
```

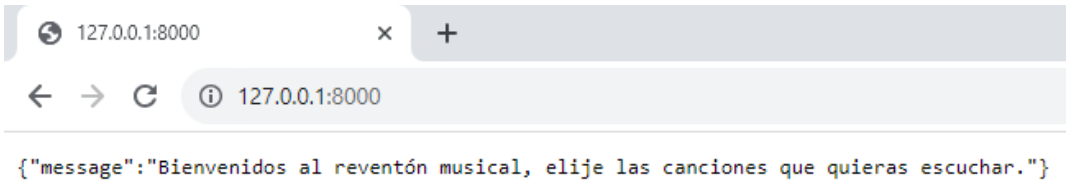
9. En símbolo de sistema, se ejecuta el siguiente comando `pip install uvicorn` .



10. Después se ejecuta el comando `uvicorn main:app --reload` y así se lanza la API en el directorio http://127.0.0.1:8000, ósea el puerto 8000 .



11. Se abre el navegador de preferencia (en este caso Google chrome) y en la barra de búsqueda se coloca el directorio ya desplegado. Como resultado, la API lanza como salida anteriormente descrita en el paso 8.



12. En este caso se realizará una API en base a un setlist de diversas canciones. Para eso primero se definirá un arreglo de diccionarios.

```
canciones = [
    {
        "id": 1,
        "name": "Remote Control",
        "artista": "Sussie 4 y Leon Larregui"
    },
    {
        "id": 2,
        "name": "Bohemian Rhapsody",
        "artista": "Queen"
    }
]
```

13. Definimos algunas funciones, con sus entradas y salidas. Una para la consulta de la lista de canciones, otra para consultar de acuerdo al id y la última para consulta con id y artista al mismo tiempo.

```
@app.get("/canciones")
def get_canciones():
    return canciones

@app.get("/canciones/{id}")
def get_cancion(id: int):
    return list(filter(lambda item: item['id'] == id, canciones))

@app.get("/canciones/")
def get_cancion(id: int, artista: str):
    return list(filter(lambda item: item['id'] == id, canciones))
```

Para desplegar cada una con el cliente REST (Google Chrome), debe tenerse actualizado `uvicorn main:app --reload` en símbolo de sistema, en chrome se busca http://127.0.0.1:8000/docs y de ahí se despliegan las funciones respectivas. Por cada una dar clic en la flecha abajo, luego en “try it out” y al final “execute” (Nota: este procedimiento es válido para las siguientes funciones. Y también, esta es una manera para documentar la API, y ejecutarla por su facilidad visual y lógica).

Curl

```
curl -X 'GET' \
  'http://127.0.0.1:5000/canciones' \
  -H 'accept: application/json'
```

Request URL

```
http://127.0.0.1:5000/canciones
```

Server response

Code	Details
200	Response body

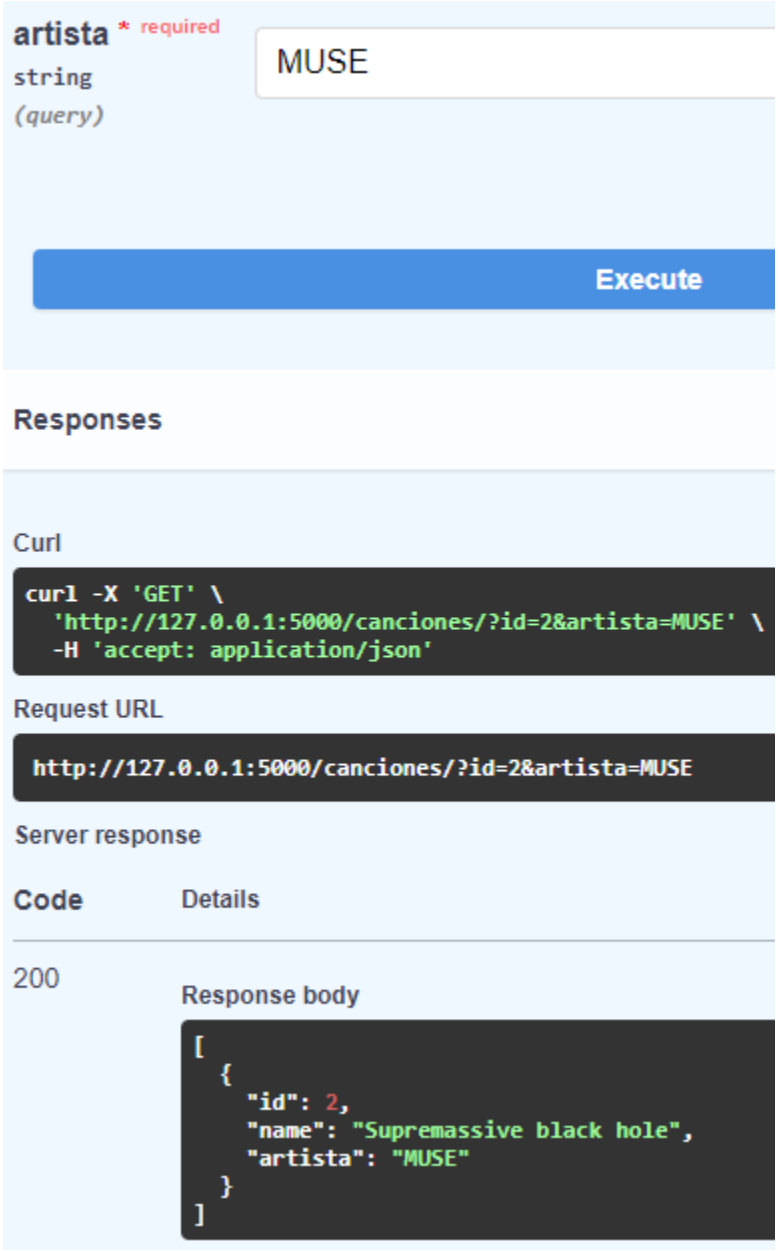
```
[
  {
    "id": 1,
    "name": "Remote Control",
    "artista": "Sussie 4 y Leon Larregui"
  },
  {
    "id": 2,
    "name": "Supremassive black hole",
    "artista": "MUSE"
  }
]
```

Response headers

```
content-length: 130
content-type: application/json
date: Wed,23 Aug 2023 05:37:38 GMT
server: uvicorn
```

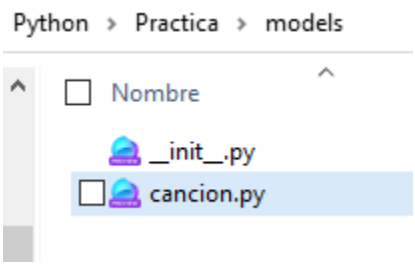
get canciones()

get cancion(id: int)



get_cancion(id: int, artista: str)

14. Ahora se van a crear unos esquemas, para ello se creará una nueva carpeta con el nombre de “models” (al nivel de main.py), y dentro de esa carpeta un archivo llamado __init__.py y también el archivo del modelo llamado cancion.py .



15. Se edita el archivo cancion.py importando BaseModel de Pydantic y se declara la clase Cancion que hereda de BaseModel.

```python
from pydantic import BaseModel

class Cancion(BaseModel):
    id: int
    name: str
    artista: str
```

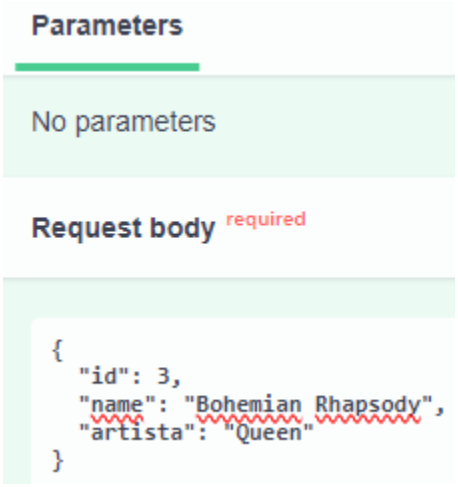
16. Regresando al main.py se importa la clase Cancion de models.cancion.

```python
from fastapi import FastAPI
from models.cancion import Cancion
```

17. En main.py, se crea una función para añadir canciones que es de tipo post.

```python
@app.post("/canciones")
def create_cancion(cancion: Cancion):
    canciones.append(cancion)
    return canciones
```

Función desplegada.



Responses

Curl

```
curl -X 'POST' \
  'http://127.0.0.1:5000/canciones' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "id": 3,
  "name": "Bohemian Rhapsody",
  "artista": "Queen"
}'
```

Request URL

```
http://127.0.0.1:5000/canciones
```

Server response

Code	Details
200	Response body

```
[
  {
    "id": 1,
    "name": "Remote Control",
    "artista": "Sussie 4 y Leon Larregui"
  },
  {
    "id": 2,
    "name": "Supremassive black hole",
    "artista": "MUSE"
  },
  {
    "id": 3,
    "name": "Bohemian Rhapsody",
    "artista": "Queen"
  }
]
```

18. Ahora se define una función para editar canciones, que es de tipo put.

```python
@app.put("/canciones/{id}")
def update_cancion(id: int, cancion: Cancion):
    for index, item in enumerate(canciones):
        if item['id'] == id:
            canciones[index]['name'] = cancion.name
            canciones[index]['artista'] = cancion.artista
    return canciones
```

Función desplegada

Responses

Curl

```
curl -X 'PUT' \
  'http://127.0.0.1:5000/canciones/2' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "name": "Starlight",
  "artista": "MUSE"
}'
```

Request URL

```
http://127.0.0.1:5000/canciones/2
```

Server response

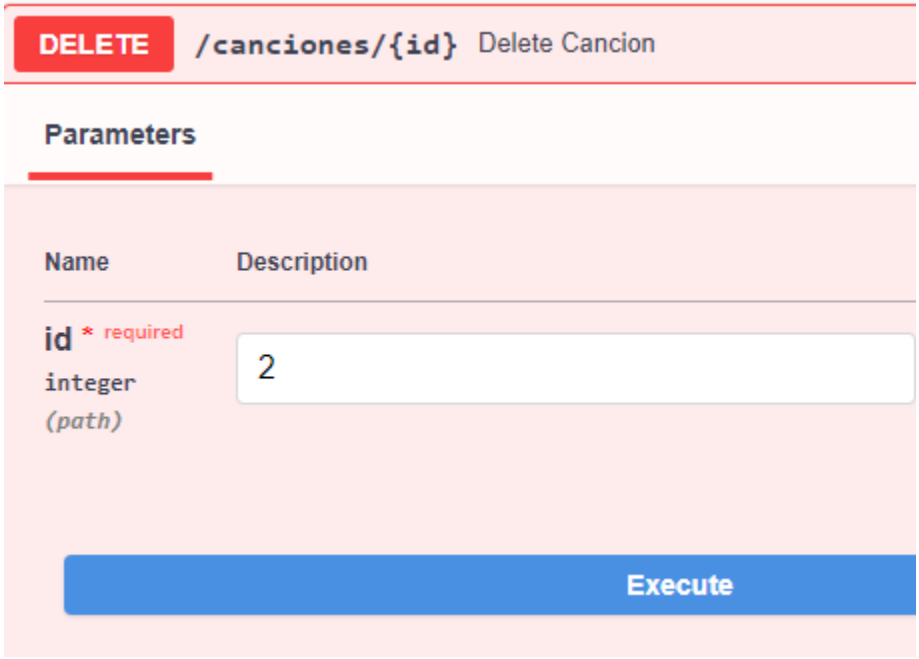
Code	Details
200	Response body

```
[
  {
    "id": 1,
    "name": "Remote Control",
    "artista": "Sussie 4 y Leon Larregui"
  },
  {
    "id": 2,
    "name": "Starlight",
    "artista": "MUSE"
  }
]
```

19. Y también se define la función para eliminar elementos de la lista que es de tipo delete.

```python
@app.delete("/canciones/{id}")
def delete_cancion(id: int):
    for item in canciones:
        if item['id'] == id:
            canciones.remove(item)
    return canciones
```

Función desplegada



Responses

Curl

```
curl -X 'DELETE' \
  'http://127.0.0.1:5000/canciones/2' \
  -H 'accept: application/json'
```

Request URL

```
http://127.0.0.1:5000/canciones/2
```

Server response

Code	Details
200	Response body

```
[
  {
    "id": 1,
    "name": "Remote Control",
    "artista": "Sussie 4 y Leon Larregui"
  }
]
```

Response headers

```
content-length: 71
content-type: application/json
date: Wed,23 Aug 2023 06:12:24 GMT
server: uvicorn
```

20. Para hacerlo más práctico y en caso de que el usuario no sepa el id para poder editar, agregar o eliminar, solo se edita en cancion.py importando Optional de typing y dentro de la clase canción se asignaría a ese atributo como un entero opcional, y en caso de que no se asigne optaría por ser nulo.

```python
from pydantic import BaseModel
from typing import Optional

class Cancion(BaseModel):
    id: Optional[int] = None
    name: str
    artista: str
```

21. Para realizar las validaciones, ya sea poner ciertas restricciones a los atributos de Cancion, se añade en la importación “Field” y después se editan los atributos

```python
from pydantic import BaseModel, Field
from typing import Optional

class Cancion(BaseModel):
    id: Optional[int] = None
    name: str = Field(default="Nueva canción", min_length=2, max_length=20)
    artista: str = Field(default="Nuevo artista", min_length=2, max_length=20)
```

Para comprobarlo, se hará con la función create_cancion(cancion: Cancion).

Curl

```
curl -X 'POST' \
  'http://127.0.0.1:5000/canciones' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "id": 3,
  "name": "Beat It",
  "artista": "Michael Jackson"
}'
```

Request URL

```
http://127.0.0.1:5000/canciones
```

Server response

Code	Details
200	Response body

```
[
  {
    "id": 1,
    "name": "Remote Control",
    "artista": "Sussie 4 y Leon Larregui"
  },
  {
    "id": 2,
    "name": "Supremassive black hole",
    "artista": "MUSE"
  },
  {
    "id": 2,
    "name": "Beat It",
    "artista": "Michael Jackson"
  },
  {
    "id": 3,
    "name": "Beat It",
    "artista": "Michael Jackson"
  }
]
```

22. En main.py se agregan a la importación Query y Path de fastapi.

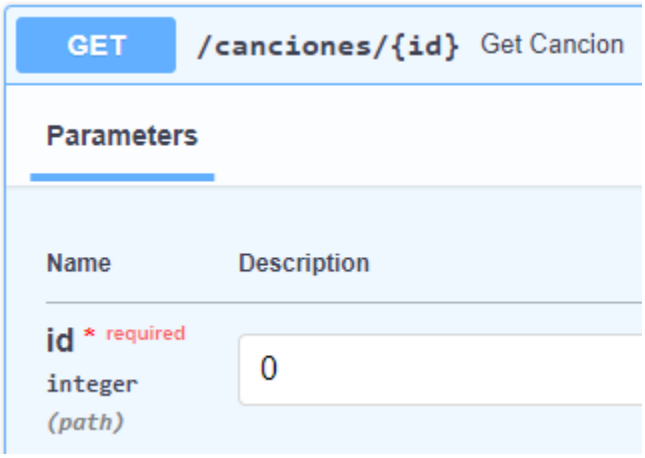
```
from fastapi import FastAPI, Query, Path
from models.cancion import Cancion

app = FastAPI()
```

Para ello modificamos la función get_cancion(id: int = Path(gt=0))

```python
@app.get("/canciones/{id}")
def get_cancion(id: int = Path(gt=0)):
    return list(filter(lambda item: item['id'] == id, canciones))
```

Y después lo comprobamos con la función ya desplegada



Responses

Curl

```
curl -X 'GET' \
  'http://127.0.0.1:5000/canciones/0' \
  -H 'accept: application/json'
```

Request URL

```
http://127.0.0.1:5000/canciones/0
```

Server response

Code	Details
422	Error: Unprocessable Entity

Response body


```
{
  "detail": [
    {
      "type": "greater_than",
      "loc": [
        "path",
        "id"
      ],
      "msg": "Input should be greater than 0",
      "input": "0",
      "ctx": {
        "gt": 0
      },
      "url": "https://errors.pydantic.dev/2.2/v/greater_than"
    }
  ]
}
```


23. Para poder consumir una API externa, solo basta con importar httpx y después escribir la función de la siguiente imagen.

```python
@app.get("/consumir-api-externa")
async def consumir_api_externa():
    url = "https://pokeapi.co/api/v2/pokemon/ditto"
    async with httpx.AsyncClient() as client:
        response = await client.get(url)
        return response.json()
```

Curl

```
curl -X 'GET' \
  'http://127.0.0.1:5002/consumir-api-externa' \
  -H 'accept: application/json'
```

Request URL

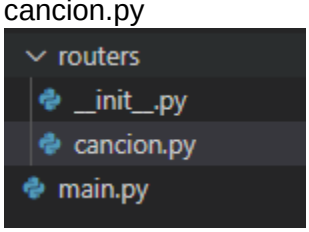
```
http://127.0.0.1:5002/consumir-api-externa
```

Server response

Code	Details
200	Response body


```
{
  "abilities": [
    {
      "ability": {
        "name": "limber",
        "url": "https://pokeapi.co/api/v2/ability/7/"
      },
      "is_hidden": false,
      "slot": 1
    },
    {
      "ability": {
        "name": "imposter",
        "url": "https://pokeapi.co/api/v2/ability/150/"
      },
      "is_hidden": true,
      "slot": 3
    }
  ],
  "base_experience": 101,
  "forms": [
    {
      "name": "ditto",
      "url": "https://pokeapi.co/api/v2/pokemon-form/132/"
    }
  ],
  "game_indices": [
    {
```


24. Con fines de optimización, se crea la carpeta routers y los archivos __init__.py y cancion.py



25. Dentro de cancion.py se importa APIRouter, y se instancia la clase APIRouter en router.

```python
from fastapi import APIRouter

router = APIRouter()
```

26. Del archivo de main.py se cortan las rutas que tengan que ver con canciones, se reemplazan los app de las rutas con router, se corta el arreglo de canciones y se importan las clases necesarias como Path y el modelo de cancion.

```python
from fastapi import Query, Path, APIRouter
from models.cancion import Cancion

router = APIRouter()

canciones = [
    {
        "id": 1,
        "name": "Remote Control",
        "artista": "Sussie 4 y Leon Larregui"
    },
    {
        "id": 2,
        "name": "Supremassive black hole",
        "artista": "MUSE"
    }
]

@router.get("/canciones")
def get_canciones():
    return canciones
#Nota
@router.get("/canciones/{id}")
def get_cancion(id: int = Path(gt=0)):
    return list(filter(lambda item: item['id'] == id, canciones))

@router.get("/canciones/")
def get_cancion(id: int, artista: str):
    return list(filter(lambda item: item['id'] == id and item['artista'] == artista, canciones))
```

27. Y ahora dentro de main.py se importa router renombrándola como song_router y se utiliza dentro de app.include_router(), esto para que se vea reflejado el cambio en la API.

```python
from routers.cancion import router as song_router
import httpx

app = FastAPI()
app.include_router(song_router)
```

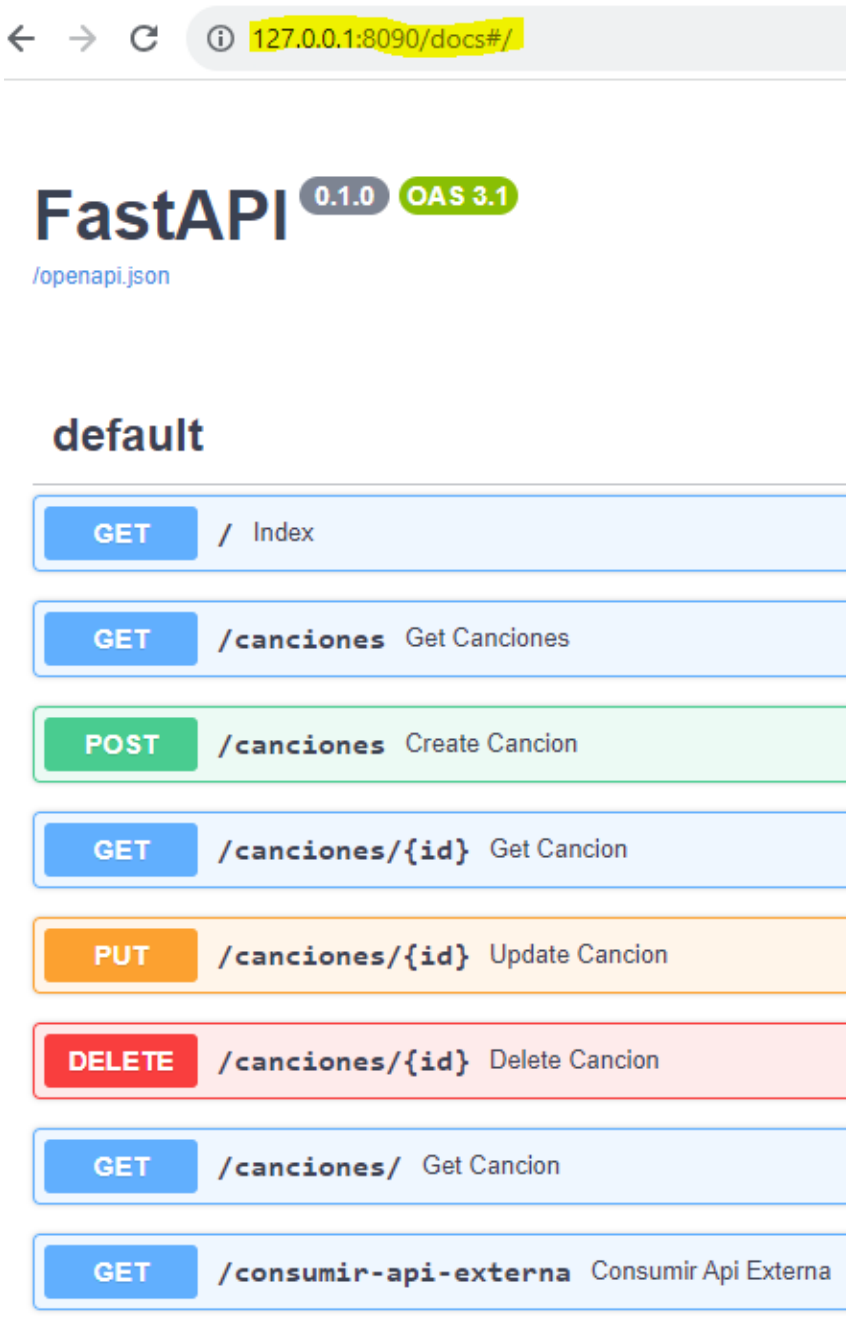
28. Si se desea correr la API desde el puerto 8090, solo basta con cerrar la actual ejecución, volver a realizar los pasos 3, 4 y 10, pero con la diferencia de que no se ejecuta el comando para instalar el entorno virtual y en la ejecución de reload es `uvicorn main:app --reload --port 8090`.

```
Microsoft Windows [Versión 10.0.19045.3324]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\dell>cd "Documents\Programación\Python\Practica"

C:\Users\dell\Documents\Programación\Python\Practica>fastapi-env\Scripts\activate.bat

(fastapi-env) C:\Users\dell\Documents\Programación\Python\Practica>uvicorn main:app --reload --port 8090
←[32mINFO←[0m:     Will watch for changes in these directories: ['C:\\Users\\dell\\Documents\\Programación\\Python\\Prac
tica']
←[32mINFO←[0m:     Uvicorn running on ←[1mhttp://127.0.0.1:8090←[0m (Press CTRL+C to quit)
←[32mINFO←[0m:     Started reloader process [←[36m←[1m14428←[0m] using ←[36m←[1mStatReload←[0m
←[32mINFO←[0m:     Started server process [←[36m4264←[0m]
←[32mINFO←[0m:     Waiting for application startup.
←[32mINFO←[0m:     Application startup complete.
```



Realizado por Joel Gómez - gjoel183@outlook.com