


# **Manual Técnico**

## **Proyecto 2**



# Manual Técnico

Este manual presenta información relacionada con las herramientas usadas para el desarrollo de esta página web con python y API. El API usada en este proyecto fue FastAPI.


- Nombre del programa: Movie Station
  - Lugar y fecha de elaboración: 28 de Junio del 2023
  - Jason Emanuel Arizandieta Guerrero
- 



# OBJETIVOS


- Mostrar el sistema realizado para la posibilidad de crear una página web con varios funcionamientos como plataforma de películas donde hay un cliente y un administrador
- Mostrar las partes del backend utilizadas en este proyecto
- Hacer una guía general sobre el funcionamiento de las clases y módulos dentro de este proyecto

# API's

```
backend >  app.py > ...  
1  ✓ from fastapi import FastAPI  
2    from routes.user import user  
3    from routes.movie import Movie  
4  
5  
6  ✓ app = FastAPI(  
7      title="REST API with FastAPI"  
8  )  
9  
10 app.include_router(user)  
11 app.include_router(Movie)  
12  
13  
14
```

En esta parte en el archivo app.py se importan las librerías y las rutas donde se declara las bases de datos de user y movie.

# Backend

```
backend > config >  db.py > ...  
1  from pymongo import MongoClient  
2  
3  conn = MongoClient()  
4  |  
5  
6
```

En este archivo db.py es donde se le da un nombre con el que conectar la base de datos en pymongo con el nombre de 'conn'



```

backend > routes > user.py > create_user
1  from fastapi import APIRouter, Response, status, HTTPException
2  from config.db import conn
3  from schemas.user import userEntity, usersEntity
4  from models.user import User
5  from passlib.hash import sha256_crypt
6  from bson import ObjectId
7  from starlette.status import HTTP_204_NO_CONTENT
8
9  user = APIRouter()
10
11
12  @user.get('/users', response_model=list[userEntity], tags=["users"])
13  def find_all_users():
14      return usersEntity(conn.local.user.find())
15
16
17  @user.post('/users', response_model=User, tags=["users"])
18  def create_user(user: User):
19      existing_user = conn.local.user.find_one({"name": user.name})
20      if existing_user:
21          raise HTTPException(status_code=400, detail="Username is already taken")
22
23      new_user = dict(user)
24      new_user["password"] = sha256_crypt.encrypt(new_user["password"])
25      del new_user["id"]
26      id = conn.local.user.insert_one(new_user).inserted_id
27
28      user = conn.local.user.find_one({"_id": id})
29      return userEntity(user)
30
31
32  @user.get('/users/{id}', response_model=User, tags=["users"])
33  def find_user(id: str):
34      return userEntity(conn.local.user.find_one({"_id": ObjectId(id)}))
35

```

```

36
37  @user.put('/users/{id}', response_model=User, tags=["users"])
38  def update_user(id: str, user: User):
39      conn.local.user.find_one_and_update(
40          {"_id": ObjectId(id)}, {"$set": dict(user)})
41      return userEntity(conn.local.user.find_one({"_id": ObjectId(id)}))
42
43
44  @user.delete('/users/{id}', status_code=status.HTTP_204_NO_CONTENT, tags=["users"])
45  def delete_user(id: str):
46      userEntity(conn.local.user.find_one_and_delete({"_id": ObjectId(id)}))
47      return Response(status_code=HTTP_204_NO_CONTENT)
48

```

En este archivo user.py dentro de la carpeta routes se encuentran los imports necesarios para que al entrar a FastAPI sea posible probar a enviar datos de usuarios, encontrar usuarios, borrarlos, etc.

```

backend > routes > movie.py > create_movie
1  from fastapi import APIRouter, Response, status, HTTPException
2  from config.db import conn
3  from schemas.movie import movieEntity, moviesEntity
4  from models.movie import movies
5  from bson import ObjectId
6  from starlette.status import HTTP_204_NO_CONTENT
7
8  Movie = APIRouter()
9
10
11  @Movie.get('/movies', response_model=list[movies], tags=["movies"])
12  def find_all_movies():
13      return moviesEntity(conn.local.Movie.find())
14
15
16  @Movie.post('/movies', response_model=movies, tags=["movies"])
17  def create_movie(Movie: movies):
18      new_movie = dict(Movie)
19      del new_movie["id"]
20      id = conn.local.Movie.insert_one(new_movie).inserted_id
21
22      Movie = conn.local.Movie.find_one({"_id": id})
23      return movieEntity(Movie)
24
25
26  @Movie.get('/movies/{id}', response_model=movies, tags=["movies"])
27  def find_movie(id: str):
28      return movieEntity(conn.local.Movie.find_one({"_id": ObjectId(id)}))
29

```

```

30
31  @Movie.put('/movies/{id}', response_model=movies, tags=["movies"])
32  def update_movie(id: str, Movie: movies):
33      conn.local.Movie.find_one_and_update(
34          {"_id": ObjectId(id)}, {"$set": dict(Movie)})
35      return movieEntity(conn.local.Movie.find_one({"_id": ObjectId(id)}))
36
37
38  @Movie.delete('/movies/{id}', status_code=status.HTTP_204_NO_CONTENT, tags=["movies"])
39  def delete_movie(id: str):
40      movieEntity(conn.local.Movie.find_one_and_delete({"_id": ObjectId(id)}))
41      return Response(status_code=HTTP_204_NO_CONTENT)
42

```

En este archivo movie.py dentro de la carpeta routes se encuentran los imports necesarios para que al entrar a FastAPI sea posible probar a enviar datos de las películas, encontrar, borrarlas, etc.

```

backend > schemas > user.py > usersEntity
1  def userEntity(item) -> dict:
2      return{
3          "id": str(item["_id"]),
4          "name": item["name"],
5          "lname": item["lname"],
6          "username": item["username"],
7          "email": item["email"],
8          "password": item["password"]
9      }
10
11 def usersEntity(entity) -> list:
12     return [userEntity(item) for item in entity]

```

```

backend > schemas > movie.py > movieEntity
1  def movieEntity(item) -> dict:
2      return{
3          "idM": str(item["_id"]),
4          "src": item["src"],
5          "genero": item["genero"],
6          "clasif": item["clasif"],
7          "year": item["year"],
8          "duracion": item["duracion"],
9          "comments": item["comments"]
10     }
11
12 def moviesEntity(entity) -> list:
13     return [movieEntity(item) for item in entity]

```

En estos archivos dentro de schemas se realizan las listas y las regresa sobre los datos pedidos de usuarios y películas