API services
Map My World App

TABLA DE CONTENIDO

TABLA DE ILUSTRACIONES

Documentación de servicios API “Map My World”

"Map My World" es una aplicación interactiva diseñada para explorar y revisar diversas ubicaciones y categorías en todo el mundo, tales como restaurantes, parques y museos. La aplicación busca ofrecer a los usuarios un mapa interactivo para descubrir nuevas ubicaciones y recibir recomendaciones personalizadas basadas en categorías específicas. El backend de esta aplicación maneja todas las operaciones relacionadas con el almacenamiento, recuperación y actualización de datos necesarios para asegurar recomendaciones frescas y relevantes.

Endpoints (APIs)

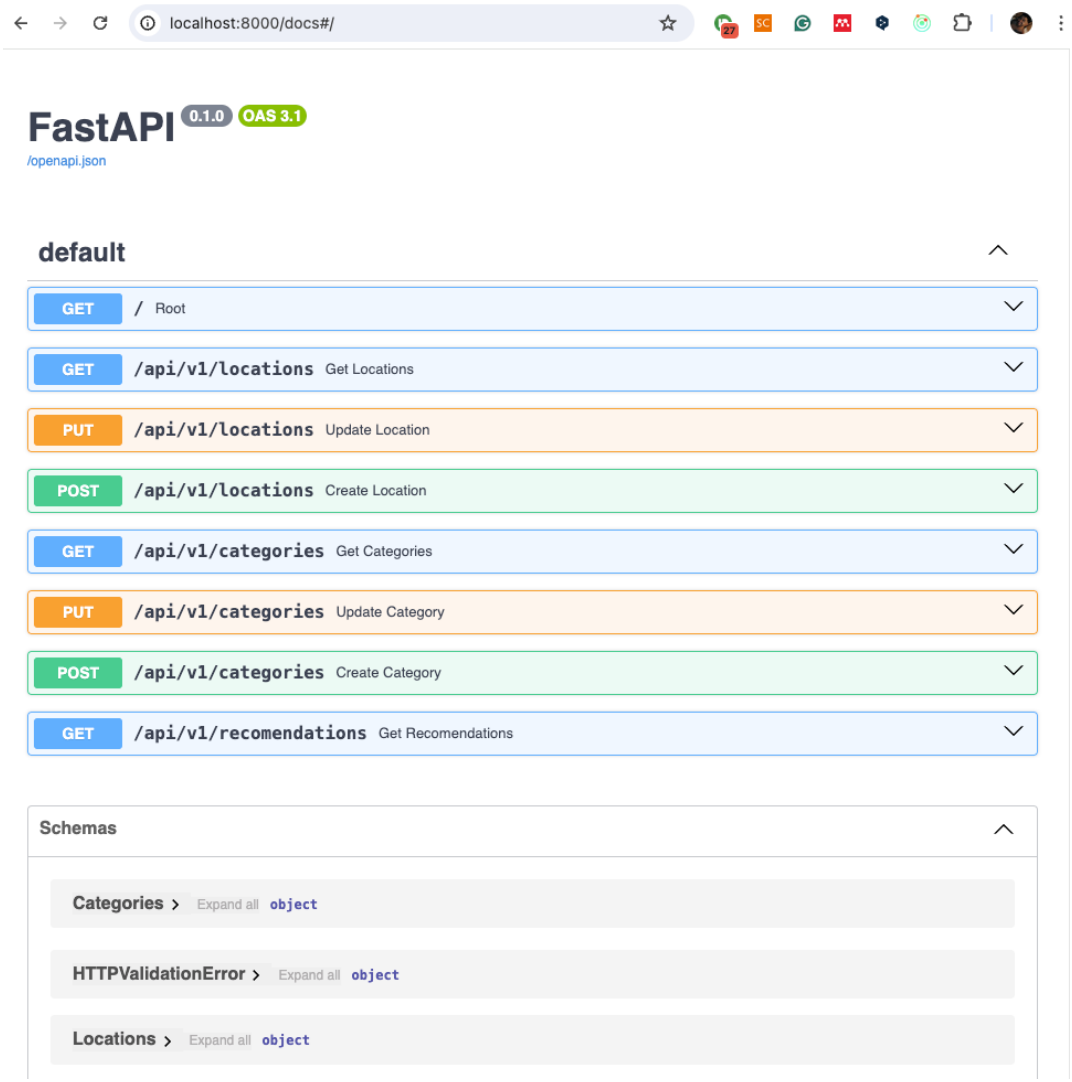


Ilustración 1 - Documentación otorgada por docs

1. Adicionar una ubicación:

URL: 127.0.0.1:8000/api/v1/locations
Método: POST
Descripción: Añade una nueva ubicación al sistema.
Cuerpo de la Solicitud:

```
{
        “id”:30,
        “longitude”: “80.03485”,
        “latitude”: “53.40564”
}
```

Respuesta de la Solicitud:

- 201 Created

Code	Description
201	Successful Response

- 400 Bad Request: Si falta algún dato necesario

Code	Description

422	{
	“detail":[
	{
	"loc": [
	"string",
	0
	],
	"msg": "string",
	"type": "string"
	}
	]
	}

2. Adicionar categorías:

URL: 127.0.0.1:8000/api/v1/categories
Método: POST
Descripción: Añade una nueva categoria al sistema.
Cuerpo de la Solicitud:

```
{
        “id”:100,
        “name”: “Natural Park”
}
```

Respuesta de la Solicitud:

- 201 Created

Code	Description
201	Successful Response

- 422 Validation Error: Si falta algún dato necesario

Code	Description

422	{
	“detail":[
	{
	"loc": [
	"string",
	0
	],
	"msg": "string",
	"type": "string"
	}
	]
	}

3. Obtener recomendaciones:

URL: 127.0.0.1:8000/api/v1/recommendations
Método: GET
Descripción: Devuelve 10 combinaciones de ubicaciones y categorías que no han sido revisadas en los últimos 30 días, priorizando aquellas que nunca han sido revisadas.
Cuerpo de la Solicitud:

```
{
        “id”:100,
        “name”: “Natural Park”
}
```

Respuesta de la Solicitud:

- 200 Ok

Code	Description

200	[
	  {
	  "location_id": 1,
	  "category_id": 5,
	  "last_reviewed_date": null
	  },
	  ...
	]

- 204 No content: Si no hay recomendaciones disponibles

Code	Description
204	No content

Consideraciones generales

- Todos los endpoints esperan y devuelven datos en formato JSON.
- Es importante que todas las solicitudes a estos endpoints incluyan los encabezados adecuados para el contenido en JSON.
- La API está diseñada para ser eficiente y responder rápidamente a todas las solicitudes, pero el rendimiento puede variar según la carga del servidor y el tamaño de los datos manejados.

Descripciones técnicas

1. Diagrama relacional de la aplicación Map My World

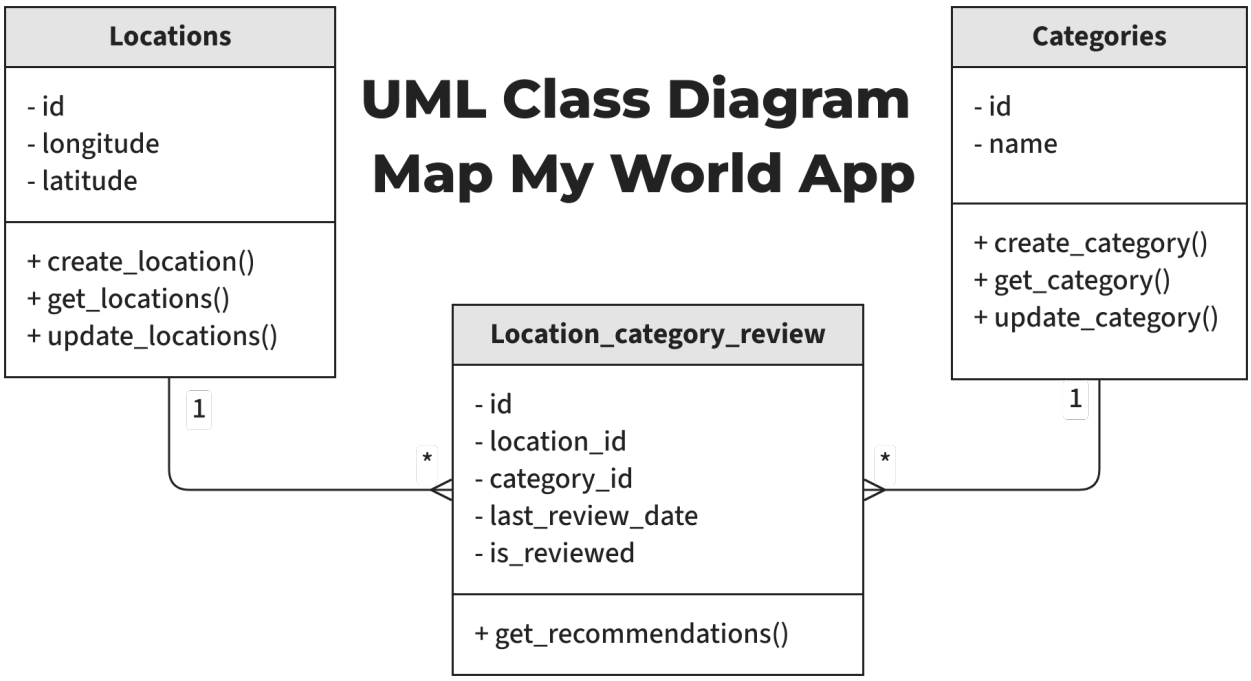


Ilustración 2 - Diagrama relacional "Map My World"

El diagrama presenta las tres clases principales que interactúan entre sí para la funcionalidad de la aplicación:

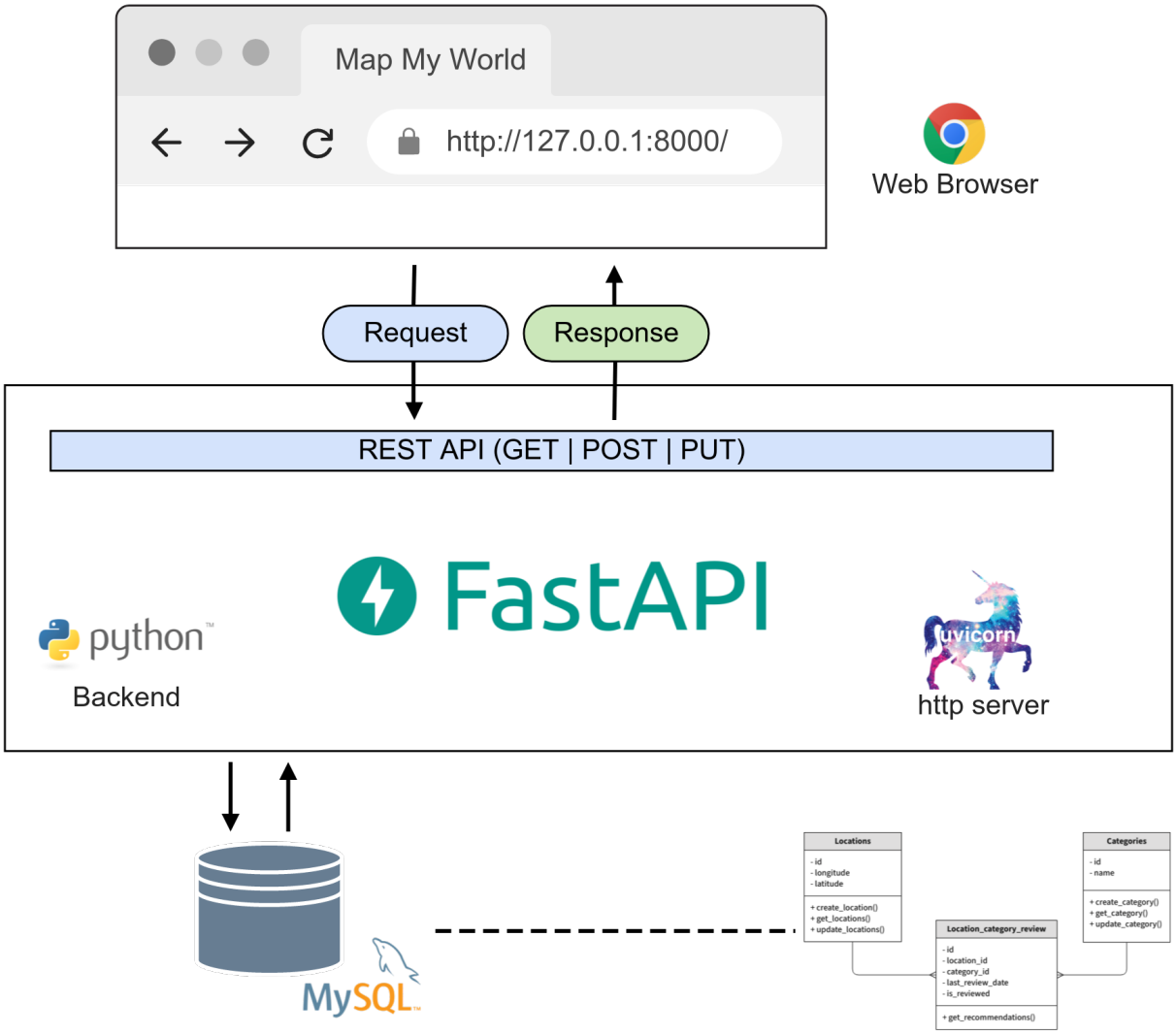
1. Clase Locations:
    - Atributos: Contiene atributos para el identificador (id), la longitud geográfica (longitude) y la latitud geográfica (latitude).
    - Métodos:
        - create_location(): Crea una nueva ubicación.
        - get_locations(): Recupera la lista de ubicaciones.
        - update_locations(): Actualiza la información de una ubicación existente.
2. Clase Categories:
    - Atributos: Incluye un identificador (id) y un nombre (name) para cada categoría.
    - Métodos:
        - create_category(): Crea una nueva categoría.
        - get_category(): Obtiene los detalles de una categoría.
        - update_category(): Modifica una categoría existente.
3. Clase Location_category_review:
    - Atributos: Mantiene registros con un identificador (id), un identificador de ubicación (location_id), un identificador de categoría (category_id), una fecha de última revisión (last_review_date), y un indicador de si ha sido revisado (is_reviewed).
    - Métodos:
        - get_recommendations(): Obtiene recomendaciones de combinaciones de ubicaciones y categorías, posiblemente basado en ciertos criterios como ser no revisados recientemente.

Dinámica del Sistema:

Un usuario puede interactuar con el sistema a través de estas clases para crear, recuperar y actualizar ubicaciones y categorías, proporcionando recomendaciones basadas en la revisión de la combinación de ubicaciones y categorías. Las recomendaciones podrían estar diseñadas para ofrecer a los usuarios opciones frescas y menos exploradas para mejorar su experiencia con la aplicación.

El diagrama modelo entidad relación de la Ilustración 2 se enfoca en las entidades principales y sus interacciones, lo que sugiere un sistema que soporta una funcionalidad de exploración basada en ubicaciones y categorías que pueden ser revisadas y recomendadas dinámicamente.

2. Arquitectura del sistema



1. Interfaz de Usuario (Web Browser)

Los usuarios interactúan con la aplicación a través de un navegador web, que permite enviar y recibir datos desde y hacia el servidor. El navegador hace solicitudes a la dirección local http://127.0.0.1:8000/, que es la dirección predeterminada para esta aplicación local FastAPI.

2. Solicitud y Respuesta (Request / Response)

El navegador web envía solicitudes al backend utilizando el protocolo HTTP. Las solicitudes pueden ser de varios tipos, como GET, POST y PUT, dependiendo de la acción que el usuario quiera realizar (obtener datos, enviar nuevos datos o actualizar datos existentes, respectivamente).

3. *Backend (FastAPI con Uvicorn)*

FastAPI está diseñado para construir APIs con pocas líneas de código. Permite el uso de Uvicorn como servidor HTTP asincrónico, lo cual lo hace adecuado para operaciones de IO-bound y código asincrónico.

El backend recibe las solicitudes HTTP y, basándose en las rutas y métodos definidos, procesa estas solicitudes. Luego, interactúa con la base de datos para recuperar, insertar o modificar datos, y finalmente devuelve una respuesta al navegador.

4. *Base de Datos (MySQL)*

MySQL es el sistema de gestión de bases de datos que almacena y gestiona la información estructurada requerida por la aplicación. La base de datos mantiene las tablas que incluyen datos sobre ubicaciones, categorías y revisiones de la combinación de ubicaciones y categorías.

5. *Estructura de la Base de Datos*

La base de datos diseñada consta de tres tablas principales que corresponden a las entidades del sistema:

- Ubicaciones (locations): Almacena la información de las ubicaciones con atributos como ID, longitud y latitud.

- Categorías (categories): Guarda las diferentes categorías con atributos como ID y nombre.

- Revisiones de Ubicación-Categoría (location_category_review): Registra las revisiones realizadas para cada combinación de ubicación y categoría, incluyendo el ID de la revisión, los IDs de ubicación y categoría, la fecha de la última revisión y un marcador de si ha sido revisada.

Este diseño describe interacciones entre el navegador web que hace solicitudes a un servidor backend a través de una API REST. El backend, construido con FastAPI, maneja las solicitudes y se comunica con una base de datos MySQL para realizar operaciones de CRUD (Crear, Leer, Actualizar, Eliminar) sobre los datos almacenados.

3. Descripción de clases y directorio del proyecto

El proyecto en general se compone de una arquitectura técnica descrita con la siguiente estructura de directorio:

Ilustración 3 - Ficheros del proyecto

Entorno virtual:

venv – Directorio para el entorno virtual donde se instalan y aíslan las dependencias del proyecto. Esto asegura que el proyecto no interfiera con otras instalaciones de Python en el sistema y viceversa.

Configuracion:

config – Contiene archivos de configuración utilizados por el proyecto. Específicamente, aquí se establecen los parámetros para conectarse a una base de datos MySQL, centralizando todas las configuraciones relacionadas con la base de datos y otras posibles configuraciones del proyecto.

```
from sqlalchemy import create_engine, MetaData
# Enable to connect my DB juanvalencia@localhost/mapmyworld
engine=create_engine("mysql+pymysql://juanvalencia:juan10.@localhost:3306/mapmyworld”)
# Metadata
meta_data = MetaData()
```

Modelos:

model – Alberga los archivos que definen los modelos de datos del proyecto. Estos modelos son representaciones de las entidades y su lógica asociada, facilitando la interacción con la base de datos.

```
from sqlalchemy import  Table, Column
from sqlalchemy.sql.sqltypes import Integer, String, DateTime, Boolean
```

```
from config.db  import engine, meta_data

locations = Table("locations",meta_data,
              Column("id",String(255), primary_key=True),
              Column("longitude",String(255), nullable=False),
              Column("latitude",String(255), nullable=False)
            )

categories = Table("categories",meta_data,
              Column("id",Integer, primary_key=True),
              Column("name",String(255), nullable=False)
            )

location_category_reviewed = Table("location_category_reviewed",meta_data,
              Column("id",Integer, primary_key=True),
              Column("location_id",Integer),
              Column("category_id",Integer),
              Column("last_review_date",DateTime),
              Column("is_reviewed",Boolean)
            )

meta_data.create_all(engine)
```

router

router – Incluye los archivos que manejan el enrutamiento de la aplicación. Aquí se definen las rutas de la API para organizarlas de manera modular, evitando sobrecargar el archivo main.py y contribuyendo a un código más limpio y mantenible.

```
from fastapi import APIRouter, Response
from starlette.status import HTTP_201_CREATED
from datetime import datetime, timedelta
from schema.schema import Locations, Categories
from config.db import engine
from model.models import locations, categories, location_category_reviewed

router = APIRouter()
# Root
@router.get("/")
def root():
    return {"message":"Welcome to map my world"}
# Getting all locations
@router.get("/api/v1/locations")
def get_locations():
    with engine.connect() as conn:
        result = conn.execute(locations.select()).fetchall()
        return result
# Create a location
@router.post("/api/v1/locations", status_code=HTTP_201_CREATED)
def create_location(data_location: Locations):
```

```
    with engine.connect() as conn:
        new_location = data_location.dict()
        conn.execute(locations.insert().values(new_location))
        print(new_location)
        return Response(status_code=HTTP_201_CREATED)
# Update a location
@router.put("/api/v1/locations")
def update_location():
    pass

# Getting all categories
@router.get("/api/v1/categories")
def get_categories():
    with engine.connect() as conn:
        result = conn.execute(categories.select()).fetchall()
        return result

# Create a category
@router.post("/api/v1/categories", status_code=HTTP_201_CREATED)
def create_category(data_category: Categories):
    with engine.connect() as conn:
        new_category = data_category.dict()
        conn.execute(categories.insert().values(new_category))
        print(new_category)
        return Response(status_code=HTTP_201_CREATED)

# Update a category
@router.put("/api/v1/categories")
def update_category():
    pass

# Get Recommendations
@router.get("/api/v1/recommendations")
async def get_recomendations():
    # Generating all possible combinations if nos in location_category_reviewed
    all_combinations = [(loc['id'],cat['id']) for loc in locations for cat in categories]
    for loc_id, cat_id in all_combinations:
        if not any(r for r in location_category_reviewed if r.location_id == loc_id and r.category_id
== cat_id):
            location_category_reviewed.append({'location_id':loc_id,              'category_id':cat_id,
'last_reviewed_date':None})

    # Filter unreviewed combinations in the last 30 days or never reviewed
    cutoff_date = datetime.utcnow() - timedelta(days=30)
    potential_reommendations = [r for r in location_category_reviewed if r['last_reviewed_date']
is None or r['last_reviewed_date'] < cutoff_date]

    # Prioritize those never reviewed
    never_reviewed = [r for r in potential_reommendations if r['last_reviewed_date'] is None]
    recently_reviewed = [r for r in potential_reommendations if r['last_reviewed_date'] is not
None]
```

```
    # Sort reviewed recients by date to give prioritize the least recent ones
    recently_reviewed.sort(key=lambda x: x['last_reviewed_date'])

    # Merge and select the first 10 recommendations
    combined_recommendations = never_reviewed + recently_reviewed
    top_recommendations = combined_recommendations[:10]

    return [location_category_reviewed(**r) for r in top_recommendations]
```

Schema:

schema -- Define las clases del esquema que son utilizadas en el proyecto para validación de datos y serialización. Estas clases corresponden a las entidades de Locations, Categories, y LocationCategoryReviewed.

```
from pydantic import BaseModel, Field
from typing import Optional
from uuid import UUID
from datetime import datetime

class Locations(BaseModel):
    id: Optional[int] = Field(..., example=40)
    longitude : str = Field(..., example=80.03485)
    latitude : str = Field(..., example=53.40564)

class Categories(BaseModel):
    id: Optional[int] = Field(..., example=100)
    name: str = Field(..., example="Beach zone")

class Location_category_reviewed(BaseModel):
    id: Optional[int] = Field(..., example=1001)
    location_id: int = Field(..., example=40)
    category_id: int = Field(..., example=100)
    last_review_date : Optional[datetime] = None
    is_reviewed: bool = Field(default=False, example=False)
```

main

main.py → Es el archivo principal de la aplicación donde se inicializa y configura la instancia de FastAPI. Aquí se suelen incluir la configuración de middleware, bases de datos y la inclusión de rutas.

```
from fastapi import FastAPI, HTTPException
from typing import List
from datetime import datetime, timedelta
from uuid import UUID, uuid4
from random import sample
from router.router import router
```

```
app = FastAPI()

app.include_router(router)
```

Requirements

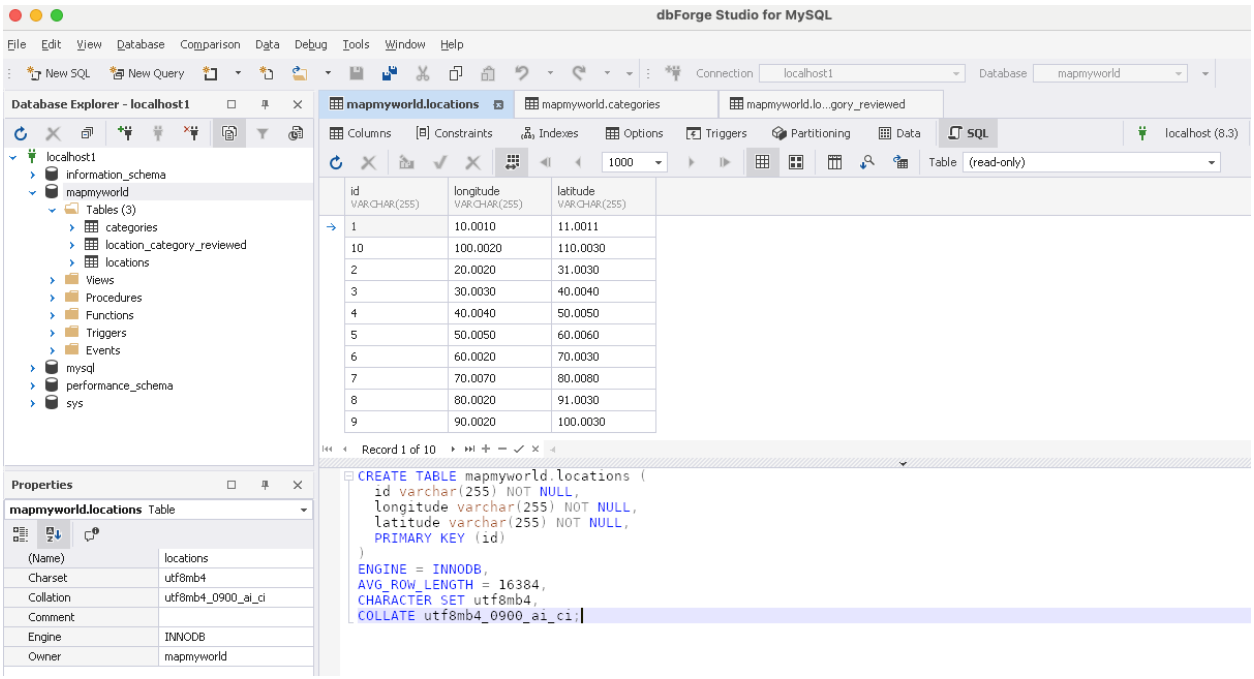
Requirements.txt → Lista todas las dependencias necesarias para el proyecto. Al instalar estas dependencias en un nuevo entorno, se asegura la compatibilidad y el correcto funcionamiento del proyecto, previniendo problemas de versiones entre distintos entornos de desarrollo o producción.

```
annotated-types==0.6.0
anyio==4.3.0
click==8.1.7
fastapi==0.110.2
h11==0.14.0
idna==3.7
pydantic==2.7.0
pydantic_core==2.18.1
PyMySQL==1.1.0
sniffio==1.3.1
SQLAlchemy==2.0.29
starlette==0.37.2
typing_extensions==4.11.0
uvicorn==0.29.0
```

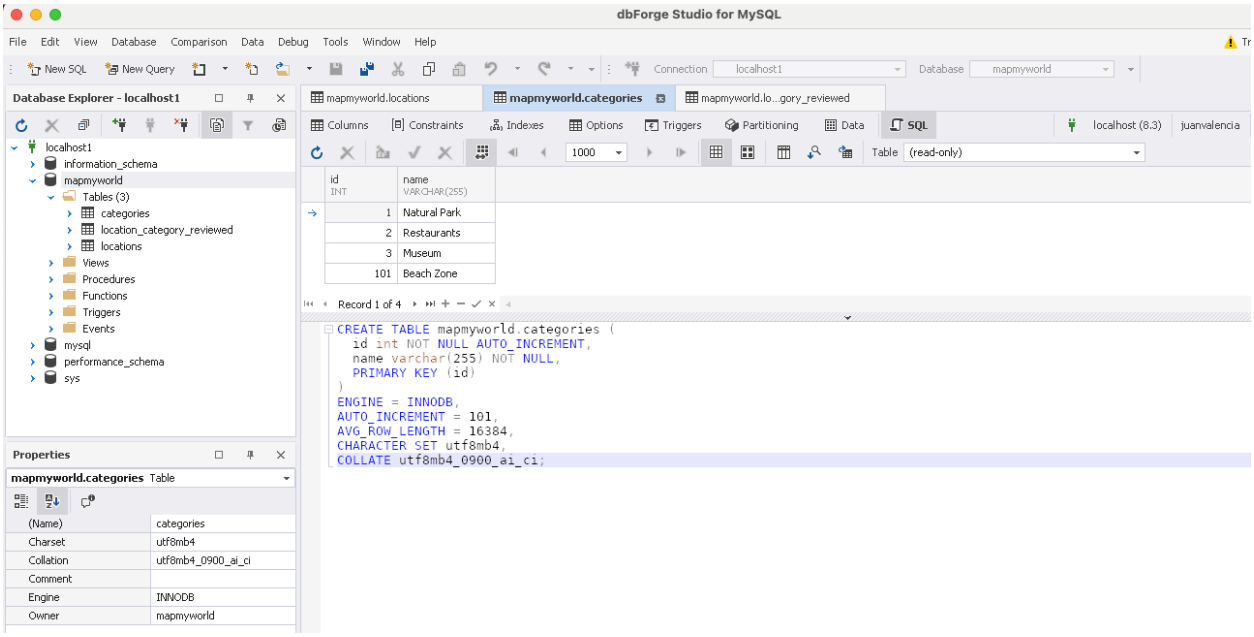
4. Imagen de las entidades en la base de datos implementada (mySql)

A continuación, se detallan las configuraciones y entidades correspondientes a la base de datos implementada para el proyecto map my world:

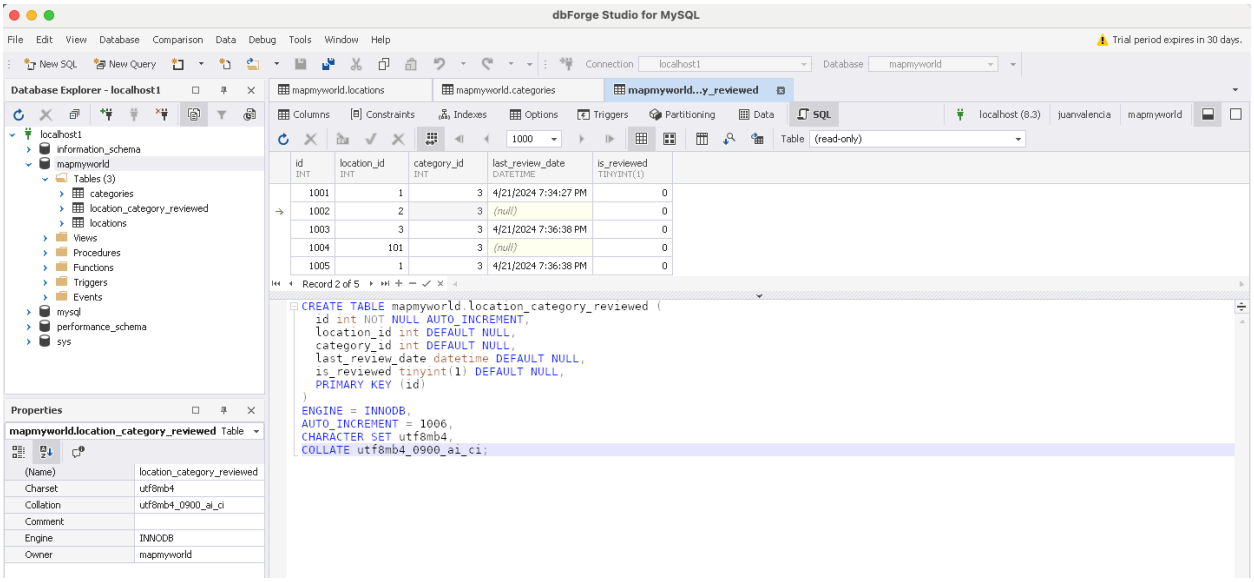
Entidad Locations en mySql ilustrada mediante dbForge:



Entidad Categories en mySql ilustrada mediante dbForge:



Entidad location_category_reviewed en mySql ilustrada mediante dbForge:


dbForge Studio for MySQL
File Edit View Database Comparison Data Debug Tools Window Help
Trial period expires in 30 days.
New SQL
New Query
Connection
localhost1
Database
mapmyworld
Database Explorer - localhost1
localhost1
information_schema
mapmyworld
Tables (3)
categories
location_category_reviewed
locations
Views
Procedures
Functions
Triggers
Events
mysql
performance_schema
sys
mapmyworld.locations
mapmyworld.categories
mapmyworld...y_reviewed
Columns
Constraints
Indexes
Options
Triggers
Partitioning
Data
SQL
localhost (8.3)
juanvalencia
mapmyworld
1000
Table
(read-only)
id INT
location_id INT
category_id INT
last_review_date DATETIME
is_reviewed TINYINT(1)
1001 1 3 4/21/2024 7:34:27 PM 0
1002 2 3 (null) 0
1003 3 3 4/21/2024 7:36:38 PM 0
1004 101 3 (null) 0
1005 1 3 4/21/2024 7:36:38 PM 0
Record 2 of 5
CREATE TABLE mapmyworld.location_category_reviewed (
id int NOT NULL AUTO_INCREMENT,
location_id int DEFAULT NULL,
category_id int DEFAULT NULL,
last_review_date datetime DEFAULT NULL,
is_reviewed tinyint(1) DEFAULT NULL,
PRIMARY KEY (id)
)
ENGINE = INNODB,
AUTO_INCREMENT = 1006,
CHARACTER SET utf8mb4,
COLLATE utf8mb4_0900_ai_ci;
Properties
mapmyworld.location_category_reviewed Table
(Name) location_category_reviewed
Charset utf8mb4
Collation utf8mb4_0900_ai_ci
Comment
Engine INNODB
Owner mapmyworld