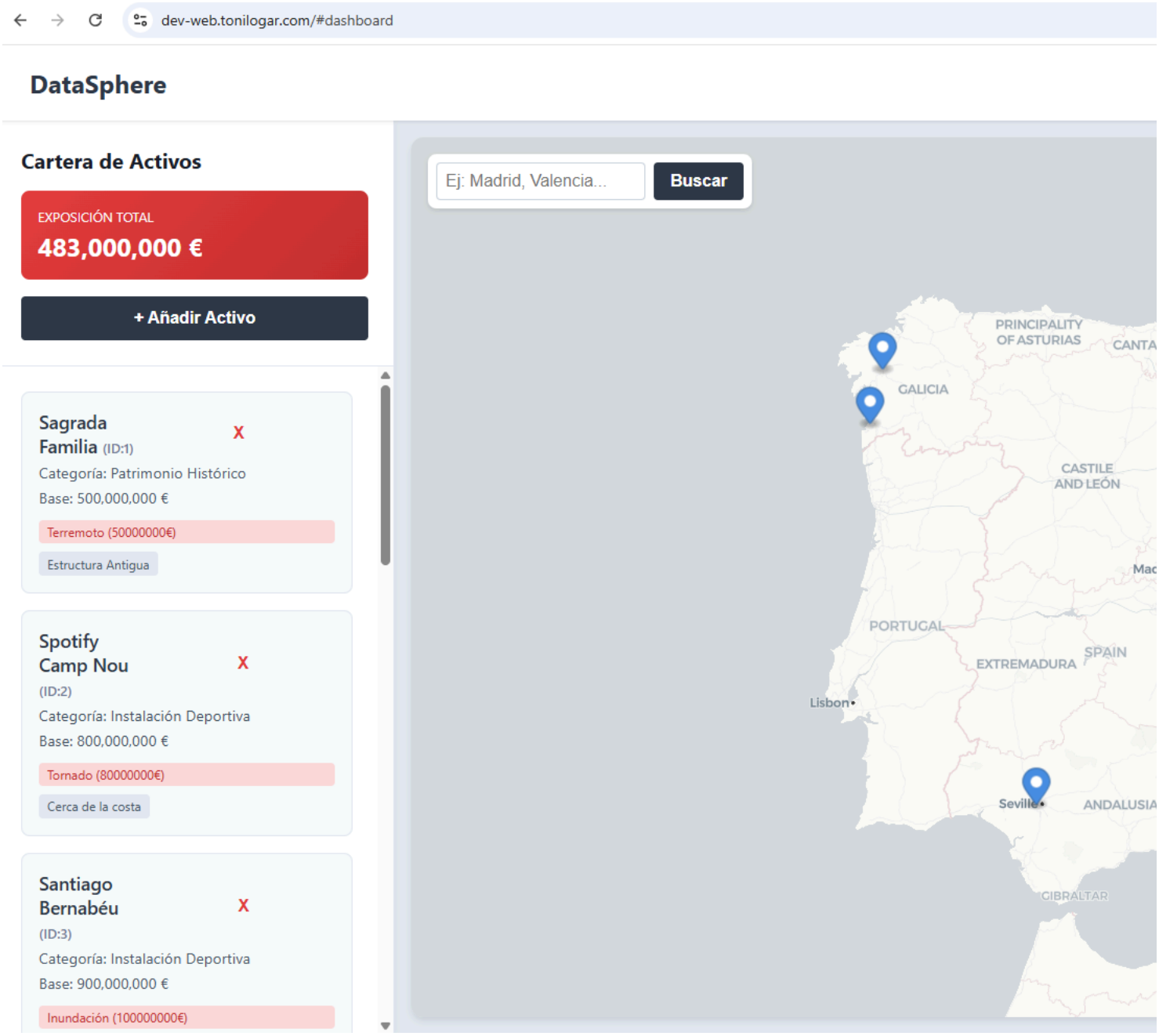
Para la realización de la actividad he construido una herramienta con docker compose 4 contenedores:

1- Base de datos postgresql
2- Backend en nodejs + typescript
3- Frontend en vite + svelte + ts + maplibre
4- Pgadmin4 para visualizar y testearar la base de datos

La herramienta esta alojada en:
https://dev-web.tonilogar.com/#login

La herramienta tiene dos roles de ususario.
Solo tiene permisos de lectura.
usuario: user_read **contraseña:** 321

Tiene permisos para crear y borrar activos.
usuario: user_write **contraseña:** 123



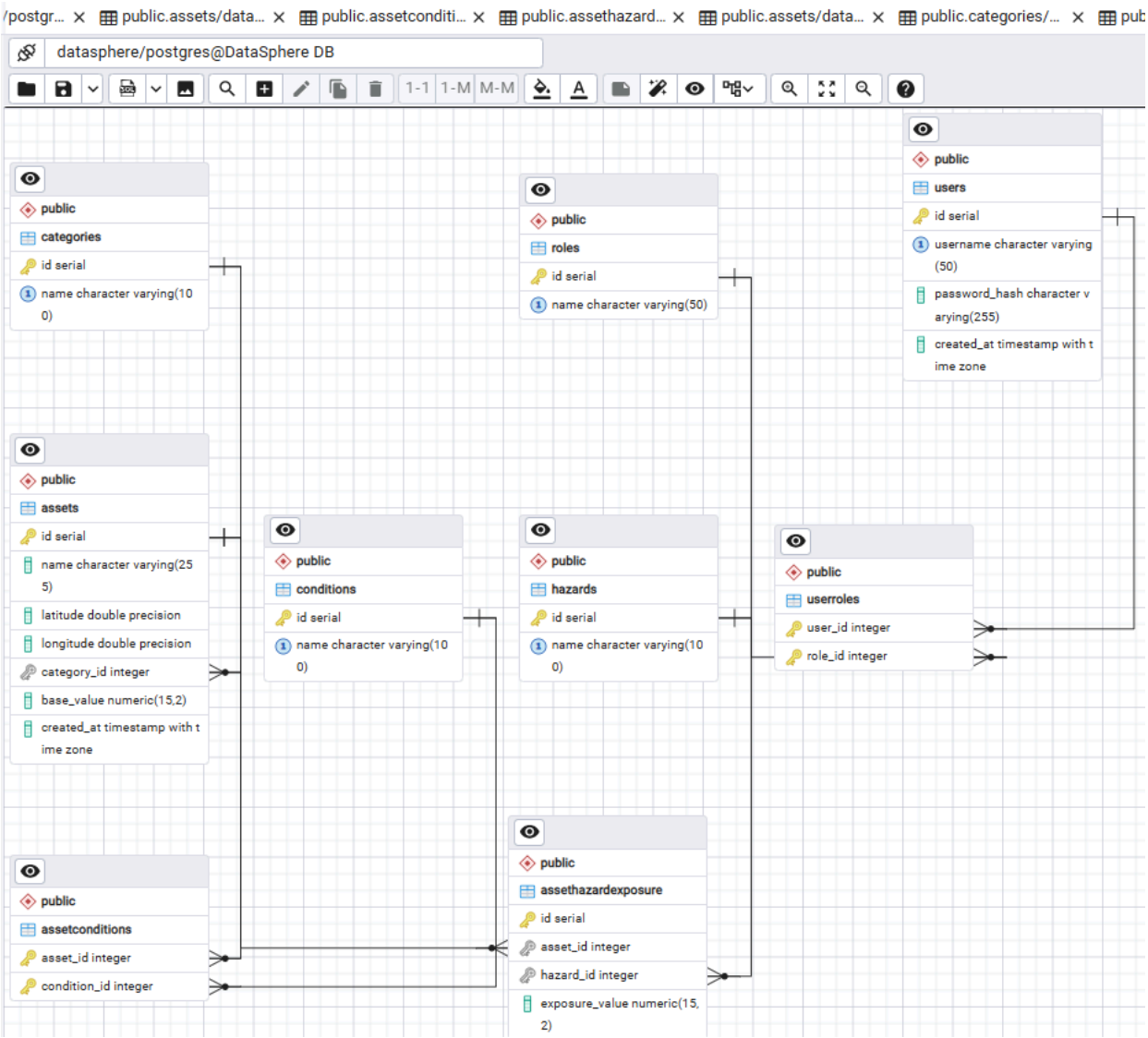
TAREA 1 — Modelado SQL

Enunciado:
Diseña e implementa un modelo relacional en una base de datos local que permita a DataSphere calcular la exposición al riesgo de sus activos frente a distintos peligros naturales. El modelo debe incluir al menos 4–5 tablas (como las que Viktor ha descrito: activos, categorías, peligros y la relación entre activos y peligros, más condiciones

si lo consideras necesario), de forma que sea posible responder preguntas como:
• “¿Cuál es el valor total expuesto ante un tipo de peligro concreto?”
• “¿Qué activos están más expuestos a tornados en una región determinada?”

---

1. Captura del modelo (tablas y relaciones)



2. Captura de `init.sql`

Código SQL para estructurar la base de datos e insertar los datos iniciales:

```sql
-- Activos
CREATE TABLE Categories (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100) UNIQUE NOT NULL
);

-- Peligros naturales
CREATE TABLE Hazards (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100) UNIQUE NOT NULL
);

-- Factores
CREATE TABLE Conditions (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100) UNIQUE NOT NULL
);

-- 4. ENTIDAD PRINCIPAL: ACTIVOS (Assets)
CREATE TABLE Assets (
    id SERIAL PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    latitude DOUBLE PRECISION NOT NULL,
    longitude DOUBLE PRECISION NOT NULL,
    category_id INTEGER REFERENCES Categories(id) ON DELETE RESTRICT,
    base_value DECIMAL(15,2) NOT NULL, -- Valor económico base
    created_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP
);

-- 5. RELACIÓN MUCHOS-A-MUCHOS: EXPOSICIÓN AL RIESGO
-- Esta tabla vincula un Activo con un Peligro, y añade el valor expuesto
CREATE TABLE AssetHazardExposure (
    id SERIAL PRIMARY KEY,
    asset_id INTEGER REFERENCES Assets(id) ON DELETE CASCADE,
    hazard_id INTEGER REFERENCES Hazards(id) ON DELETE CASCADE,
    exposure_value DECIMAL(15,2) NOT NULL, -- Dinero en riesgo por este peligro
    UNIQUE(asset_id, hazard_id)
);

-- 6. RELACIÓN MUCHOS-A-MUCHOS: CONDICIONES DEL ACTIVO
CREATE TABLE AssetConditions (
    asset_id INTEGER REFERENCES Assets(id) ON DELETE CASCADE,
    condition_id INTEGER REFERENCES Conditions(id) ON DELETE CASCADE,
    PRIMARY KEY (asset_id, condition_id)
);

-- 3. Inserción de Datos de Prueba (Dummy Data)
INSERT INTO Categories (name) VALUES ('Patrimonio Histórico'), ('Instalación Deportiva'), ('Infraestructura Crítica');
INSERT INTO Hazards (name) VALUES ('Tornado'), ('Terremoto'), ('Inundación'), ('Huracán'), ('Incendio Forestal');
INSERT INTO Conditions (name) VALUES ('Cerca de la costa'), ('Zona Sísmica Activa'), ('Estructura Antigua'), ('Sismorresistente');

-- Insertar Activos base
INSERT INTO Assets (id, name, latitude, longitude, category_id, base_value) VALUES
-- Barcelona
(1, 'Sagrada Familia', 41.4036, 2.1744, (SELECT id FROM Categories WHERE name = 'Patrimonio Histórico'), 500000000.0),
(2, 'Spotify Camp Nou', 41.3809, 2.1228, (SELECT id FROM Categories WHERE name = 'Instalación Deportiva'), 800000000.0),
-- Madrid
(3, 'Santiago Bernabéu', 40.4530, -3.6883, (SELECT id FROM Categories WHERE name = 'Instalación Deportiva'), 900000000.0),
(4, 'Museo del Prado', 40.4138, -3.6921, (SELECT id FROM Categories WHERE name = 'Patrimonio Histórico'), 450000000.0),
-- Bilbao
(5, 'Museo Guggenheim', 43.2687, -2.9340, (SELECT id FROM Categories WHERE name = 'Patrimonio Histórico'), 150000000.0),
(6, 'Campo San Mamés', 43.2642, -2.9493, (SELECT id FROM Categories WHERE name = 'Instalación Deportiva'), 210000000.0),
-- Galicia
(7, 'Catedral de Santiago de Compostela', 42.8806, -8.5446, (SELECT id FROM Categories WHERE name = 'Patrimonio Histórico'), 300000000.0),
(8, 'Estadio de Balaídos', 42.2123, -8.7401, (SELECT id FROM Categories WHERE name = 'Instalación Deportiva'), 40000000.0),
-- Andalucia
(9, 'La Alhambra', 37.1760, -3.5881, (SELECT id FROM Categories WHERE name = 'Patrimonio Histórico'), 600000000.0),
(10, 'Estadio Benito Villamarín', 37.3565, -5.9818, (SELECT id FROM Categories WHERE name = 'Instalación Deportiva'), 120000000.0);

-- Reset sequence for assets
SELECT setval('assets_id_seq', (SELECT MAX(id) FROM Assets));
```

```sql
-- Vincular Activos con Peligros (Todos los activos tienen un riesgo asignado)
INSERT INTO AssetHazardExposure (asset_id, hazard_id, exposure_value) VALUES
(1, (SELECT id FROM Hazards WHERE name = 'Terremoto'), 50000000.0),
(2, (SELECT id FROM Hazards WHERE name = 'Tornado'), 80000000.0),
(3, (SELECT id FROM Hazards WHERE name = 'Inundación'), 100000000.0),
(4, (SELECT id FROM Hazards WHERE name = 'Incendio Forestal'), 45000000.0),
(5, (SELECT id FROM Hazards WHERE name = 'Inundación'), 15000000.0),
(6, (SELECT id FROM Hazards WHERE name = 'Huracán'), 21000000.0),
(7, (SELECT id FROM Hazards WHERE name = 'Huracán'), 30000000.0),
(8, (SELECT id FROM Hazards WHERE name = 'Inundación'), 10000000.0),
(9, (SELECT id FROM Hazards WHERE name = 'Terremoto'), 120000000.0),
(10, (SELECT id FROM Hazards WHERE name = 'Tornado'), 12000000.0);

-- Vincular Activos con sus Condiciones (Todos los activos tienen una condición)
INSERT INTO AssetConditions (asset_id, condition_id) VALUES
(1, (SELECT id FROM Conditions WHERE name = 'Estructura Antigua')),
(2, (SELECT id FROM Conditions WHERE name = 'Cerca de la costa')),
(3, (SELECT id FROM Conditions WHERE name = 'Sismorresistente')),
(4, (SELECT id FROM Conditions WHERE name = 'Estructura Antigua')),
(5, (SELECT id FROM Conditions WHERE name = 'Cerca de la costa')),
(6, (SELECT id FROM Conditions WHERE name = 'Sismorresistente')),
(7, (SELECT id FROM Conditions WHERE name = 'Estructura Antigua')),
(8, (SELECT id FROM Conditions WHERE name = 'Cerca de la costa')),
(9, (SELECT id FROM Conditions WHERE name = 'Zona Sísmica Activa')),
(10, (SELECT id FROM Conditions WHERE name = 'Sismorresistente'));
```

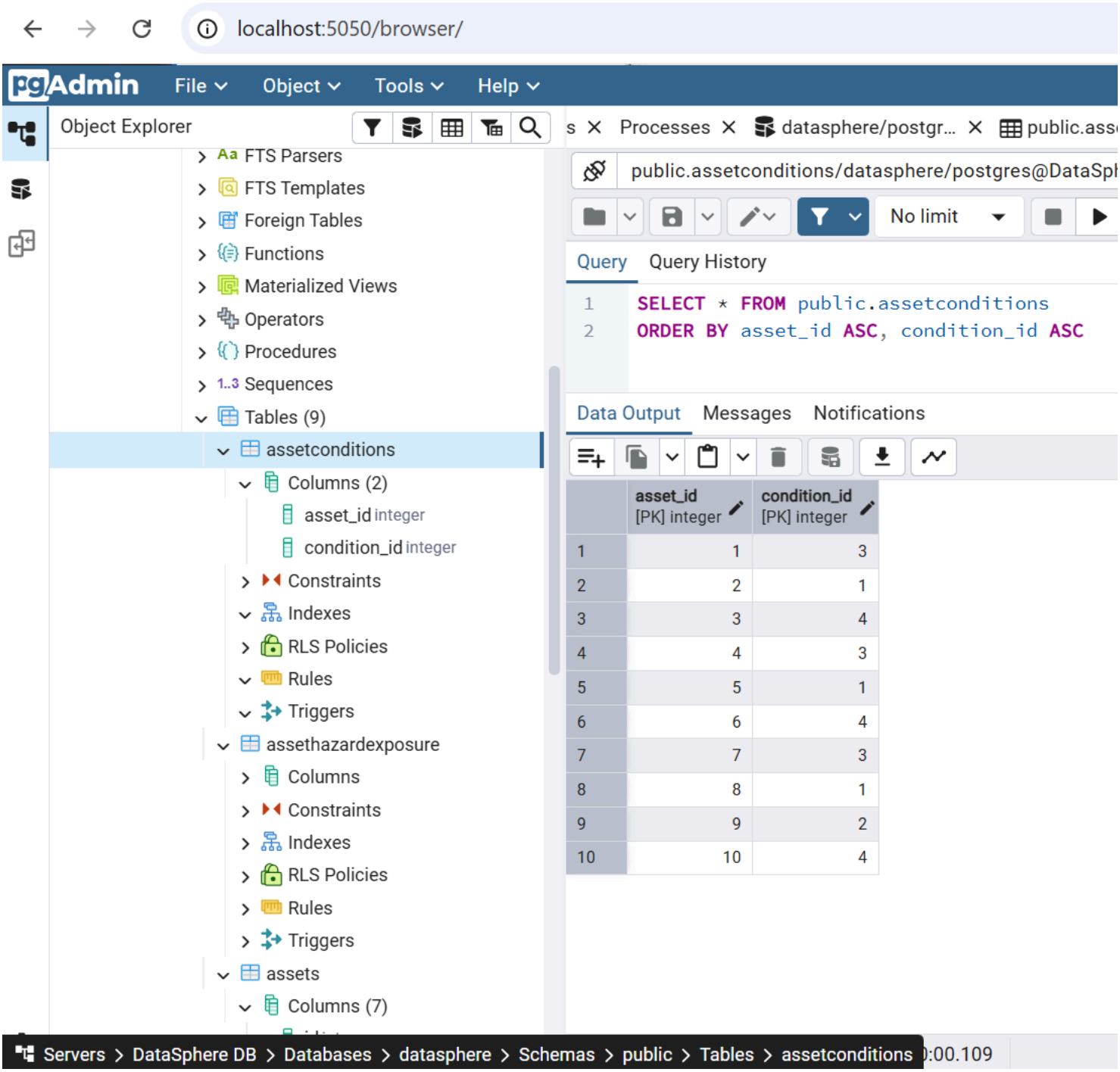
3. Registros en las tablas

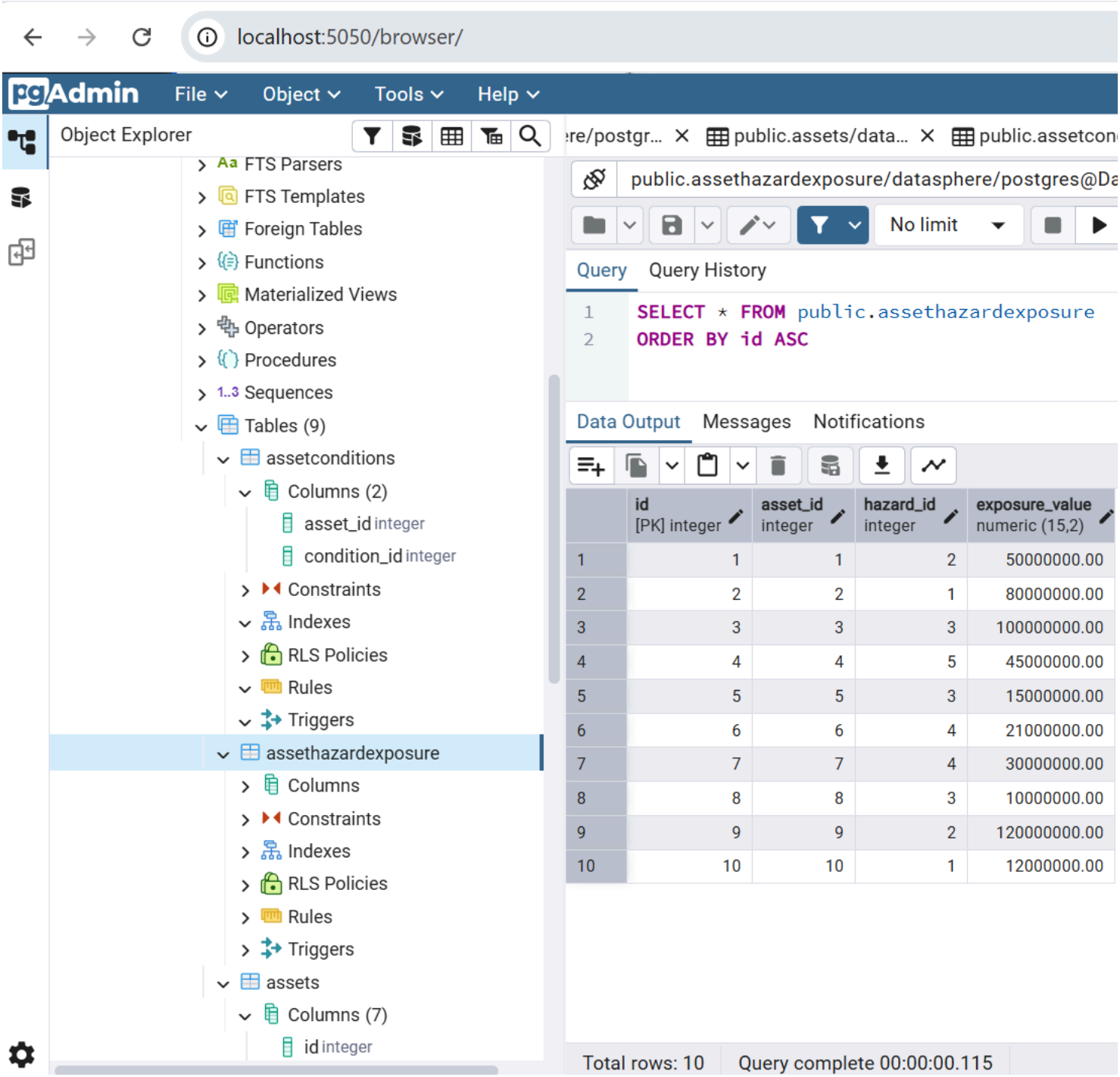
Registros de 10 activos reales de España con valores económicos y vulnerabilidades específicas:

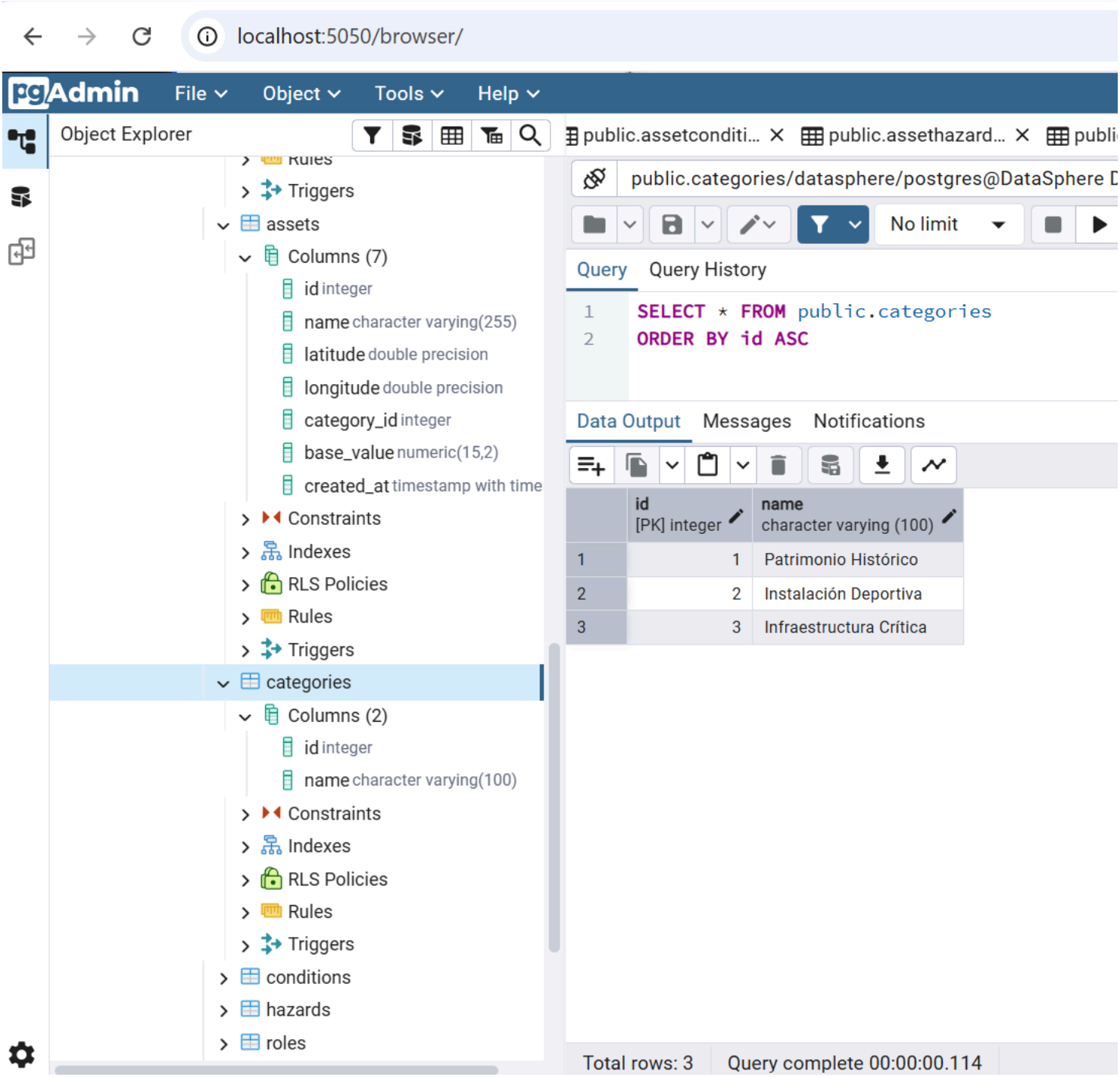
Output | Messages | Notifications

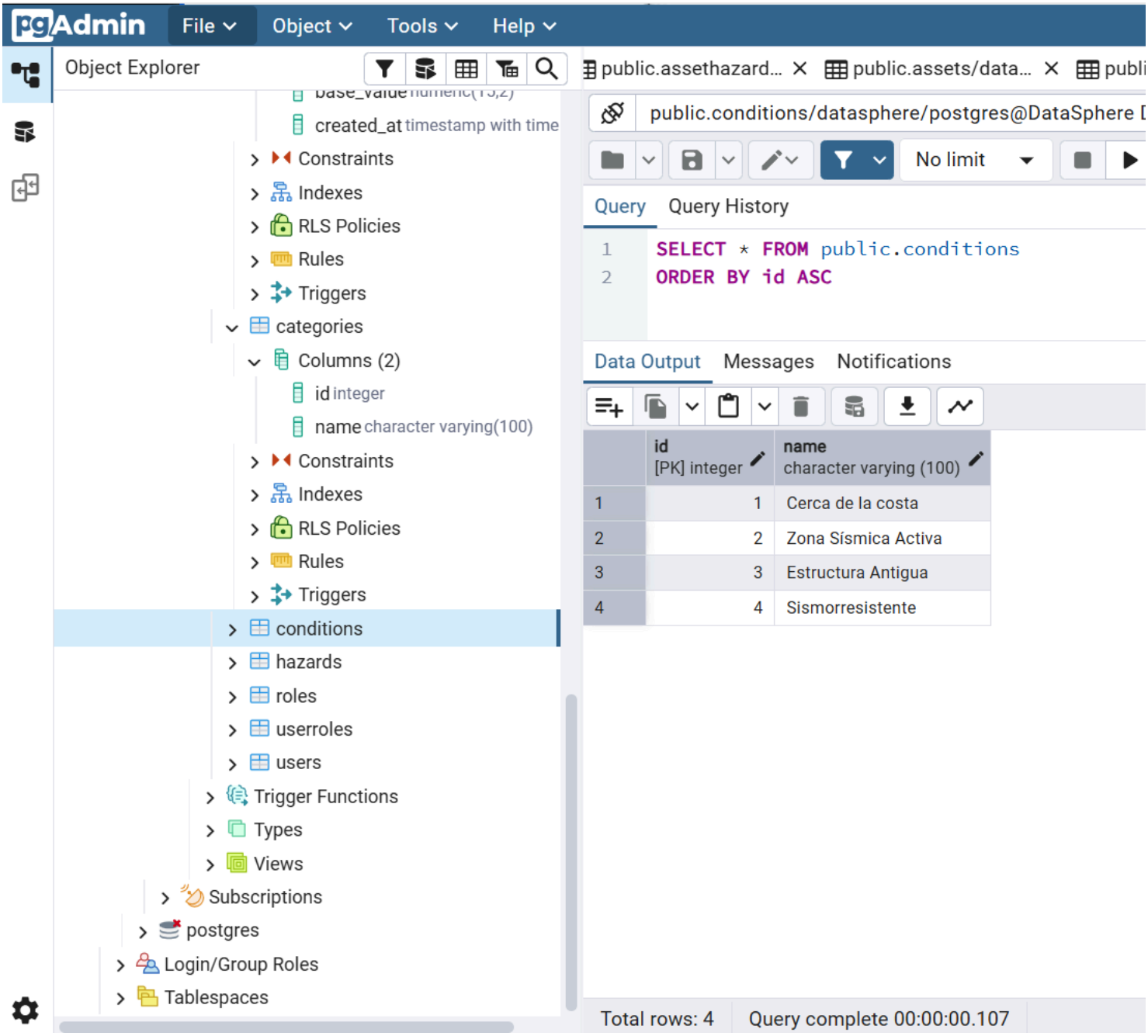
Showing rows: 1 to 10 | Page No: 1 of 1

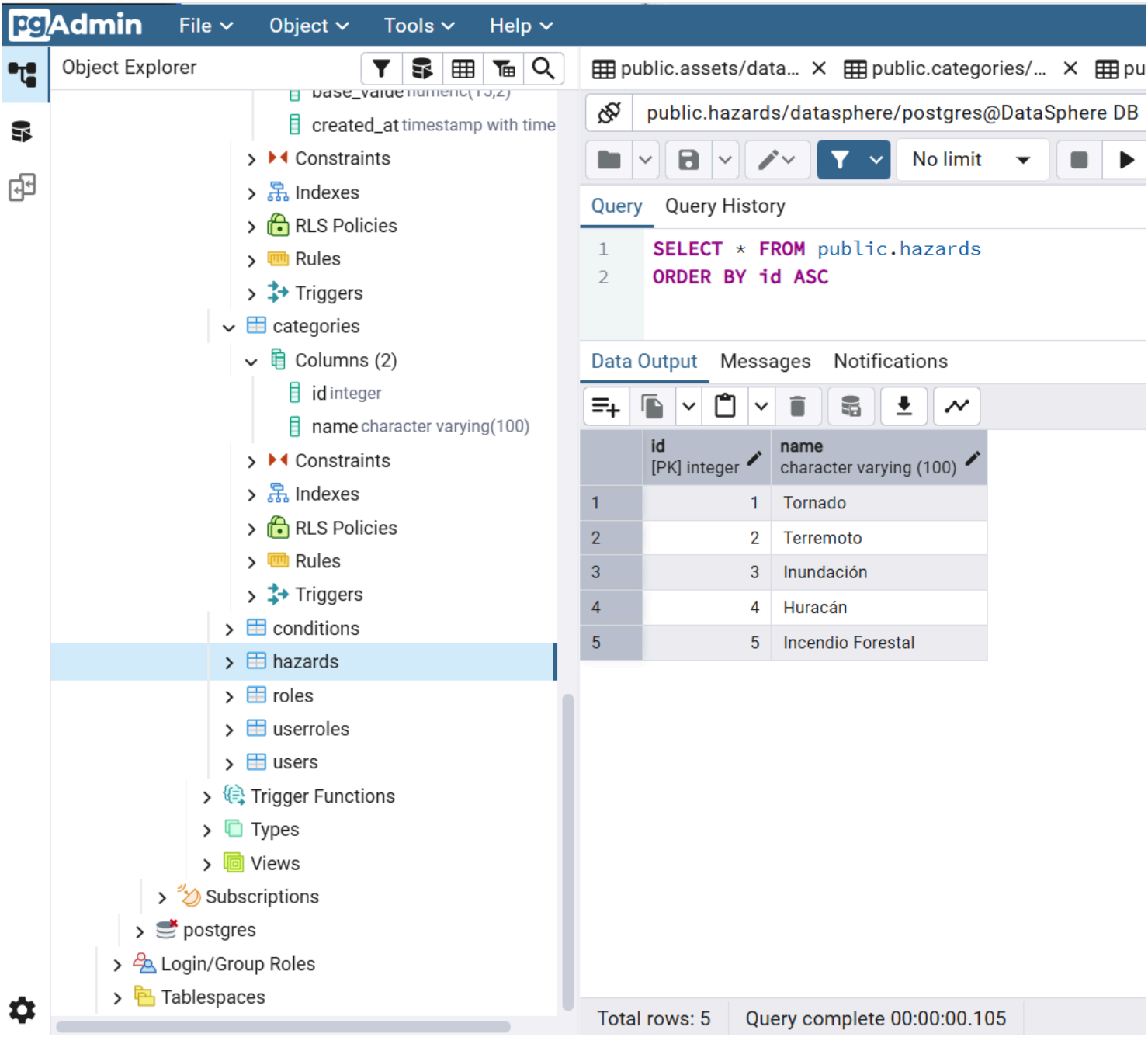
id [PK] integer	name character varying (255)	latitude double precision	longitude double precision	category_id integer	base_value numeric (15,2)	created_at timestamp w
1	Sagrada Familia	41.4036	2.1744	1	500000000.00	2026-06-01 1
2	Spotify Camp Nou	41.3809	2.1228	2	800000000.00	2026-06-01 1
3	Santiago Bernabéu	40.453	-3.6883	2	900000000.00	2026-06-01 1
4	Museo del Prado	40.4138	-3.6921	1	450000000.00	2026-06-01 1
5	Museo Guggenheim	43.2687	-2.934	1	150000000.00	2026-06-01 1
6	Campo San Mamés	43.2642	-2.9493	2	210000000.00	2026-06-01 1
7	Catedral de Santiago de Compost…	42.8806	-8.5446	1	300000000.00	2026-06-01 1
8	Estadio de Balaídos	42.2123	-8.7401	2	40000000.00	2026-06-01 1
9	La Alhambra	37.176	-3.5881	1	600000000.00	2026-06-01 1
10	Estadio Benito Villamarín	37.3565	-5.9818	2	120000000.00	2026-06-01 1


localhost:5050/browser/
pgAdmin
File
Object
Tools
Help
Object Explorer
FTS Parsers
FTS Templates
Foreign Tables
Functions
Materialized Views
Operators
Procedures
Sequences
Tables (9)
assetconditions
Columns (2)
asset_id integer
condition_id integer
Constraints
Indexes
RLS Policies
Rules
Triggers
assethazardexposure
Columns
Constraints
Indexes
RLS Policies
Rules
Triggers
assets
Columns (7)
Processes
datasphere/postgr...
public.assetconditions/datasphere/postgres@DataSph
No limit
Query
Query History
1 SELECT * FROM public.assetconditions
2 ORDER BY asset_id ASC, condition_id ASC
Data Output
Messages
Notifications
asset_id [PK] integer
condition_id [PK] integer
1 1 3
2 2 1
3 3 4
4 4 3
5 5 1
6 6 4
7 7 3
8 8 1
9 9 2
10 10 4
Servers > DataSphere DB > Databases > datasphere > Schemas > public > Tables > assetconditions
:00.109


localhost:5050/browser/
pgAdmin
File
Object
Tools
Help
Object Explorer
FTS Parsers
FTS Templates
Foreign Tables
Functions
Materialized Views
Operators
Procedures
Sequences
Tables (9)
assetconditions
Columns (2)
asset_id integer
condition_id integer
Constraints
Indexes
RLS Policies
Rules
Triggers
assethazardexposure
Columns
Constraints
Indexes
RLS Policies
Rules
Triggers
assets
Columns (7)
id integer
public.assets/data...
public.assetcon
public.assethazardexposure/datasphere/postgres@Da
No limit
Query
Query History
1 SELECT * FROM public.assethazardexposure
2 ORDER BY id ASC
Data Output
Messages
Notifications
id [PK] integer
asset_id integer
hazard_id integer
exposure_value numeric (15,2)
1 1 1 2 50000000.00
2 2 2 1 80000000.00
3 3 3 3 100000000.00
4 4 4 5 45000000.00
5 5 5 3 15000000.00
6 6 6 4 21000000.00
7 7 7 4 30000000.00
8 8 8 3 10000000.00
9 9 9 2 120000000.00
10 10 10 1 12000000.00
Total rows: 10
Query complete 00:00:00.115


localhost:5050/browser/
pgAdmin
File
Object
Tools
Help
Object Explorer
public.assetconditi...
public.assethazard...
public.categories/datasphere/postgres@DataSphere D
No limit
Query
Query History
1 SELECT * FROM public.categories
2 ORDER BY id ASC
Data Output
Messages
Notifications
id [PK] integer
name character varying (100)
1 1 Patrimonio Histórico
2 2 Instalación Deportiva
3 3 Infraestructura Crítica
Total rows: 3
Query complete 00:00:00.114
Triggers
assets
Columns (7)
id integer
name character varying(255)
latitude double precision
longitude double precision
category_id integer
base_value numeric(15,2)
created_at timestamp with time
Constraints
Indexes
RLS Policies
Rules
Triggers
categories
Columns (2)
id integer
name character varying(100)
Constraints
Indexes
RLS Policies
Rules
Triggers
conditions
hazards
roles


pgAdmin
File
Object
Tools
Help
Object Explorer
public.assethazard...
public.assets/data...
public.conditions/datasphere/postgres@DataSphere
No limit
Query
Query History
1 SELECT * FROM public.conditions
2 ORDER BY id ASC
Data Output
Messages
Notifications
id [PK] integer
name character varying (100)
1 1 Cerca de la costa
2 2 Zona Sísmica Activa
3 3 Estructura Antigua
4 4 Sismorresistente
created_at timestamp with time
Constraints
Indexes
RLS Policies
Rules
Triggers
categories
Columns (2)
id integer
name character varying(100)
conditions
hazards
roles
userroles
users
Trigger Functions
Types
Views
Subscriptions
postgres
Login/Group Roles
Tablespaces
Total rows: 4
Query complete 00:00:00.107



4. Justificación Técnica de las Decisiones de Modelado

A. Tipos de Datos: `Decimal` vs `Float`

- **`DOUBLE PRECISION` (`Float`) para Coordenadas:** Las columnas `latitude` y `longitude` utilizan precisión de coma flotante porque es el estándar en GIS para posicionamiento espacial.
- **`DECIMAL(15,2)` para Valores Económicos:** Para las columnas `base_value` y `exposure_value` (el dinero en riesgo), **nunca** se debe usar Float. Los procesadores aproximan los Floats, lo que puede causar errores en cálculos financieros. Se ha usado un numérico exacto de 15 dígitos y 2 decimales para asegurar una contabilidad estricta sin pérdidas de céntimos en agregaciones.

B. Tablas de Relación (Muchos a Muchos)

Se ha implementado la tabla puente **`AssetHazardExposure`** porque un activo puede estar expuesto a múltiples peligros a la vez, y un mismo peligro puede afectar a muchos activos.

C. Claves e Índices (`Constraints`)

1. **Restricción Única (`UNIQUE`)**: En la tabla `AssetHazardExposure` hemos añadido una restricción explícita `UNIQUE(asset_id, hazard_id)`. Esto garantiza que sea imposible asignar dos veces el mismo peligro natural al mismo activo.
2. **Claves Foráneas (Foreign Keys) con Cascadas**: Se ha aplicado `ON DELETE CASCADE` en las relaciones. Si en el futuro un administrador decide borrar un activo, automáticamente se borrarán también todos sus registros de exposición de riesgo huérfanos, manteniendo la base de datos limpia.

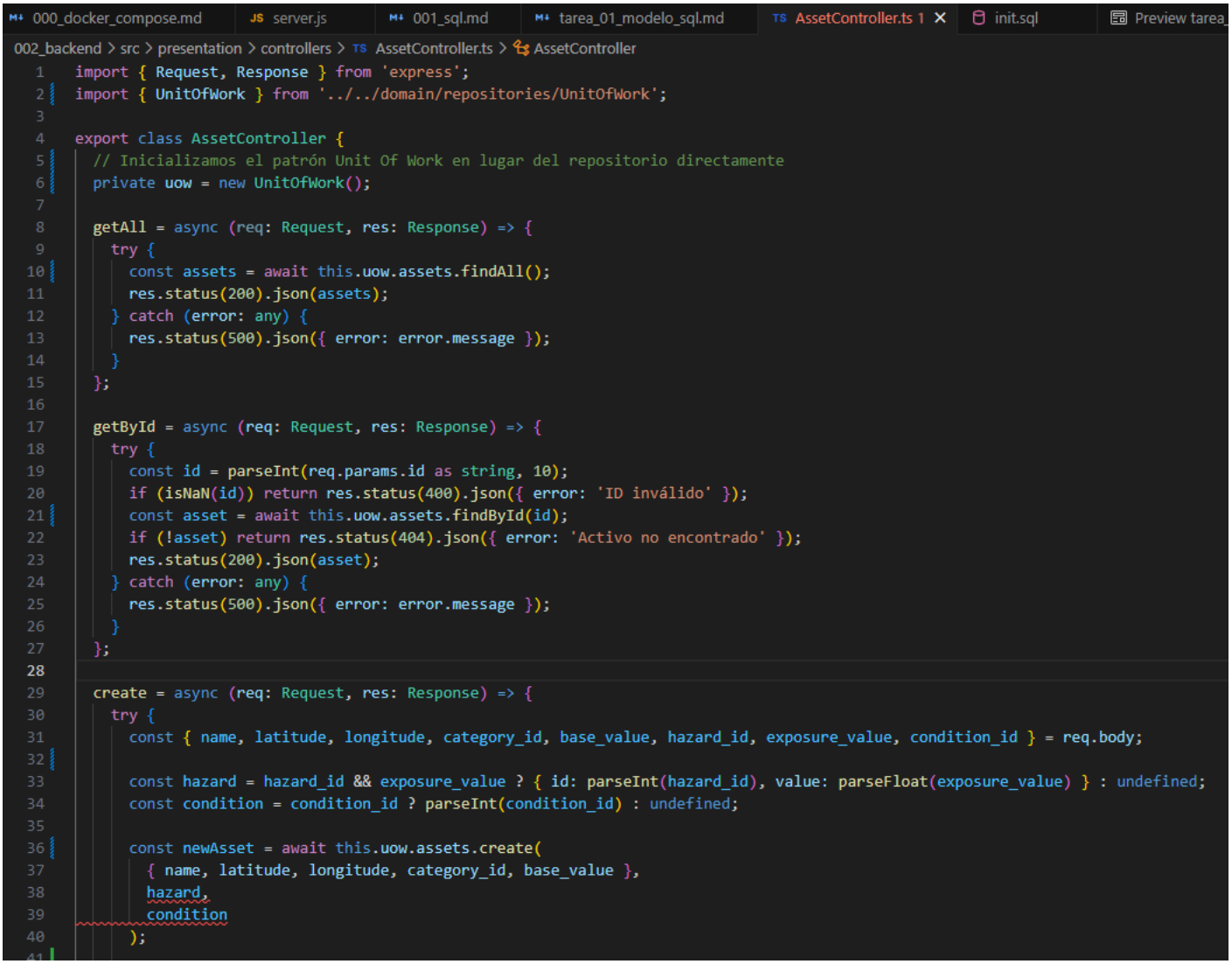
TAREA 2 — Construcción de la API CRUD de activos

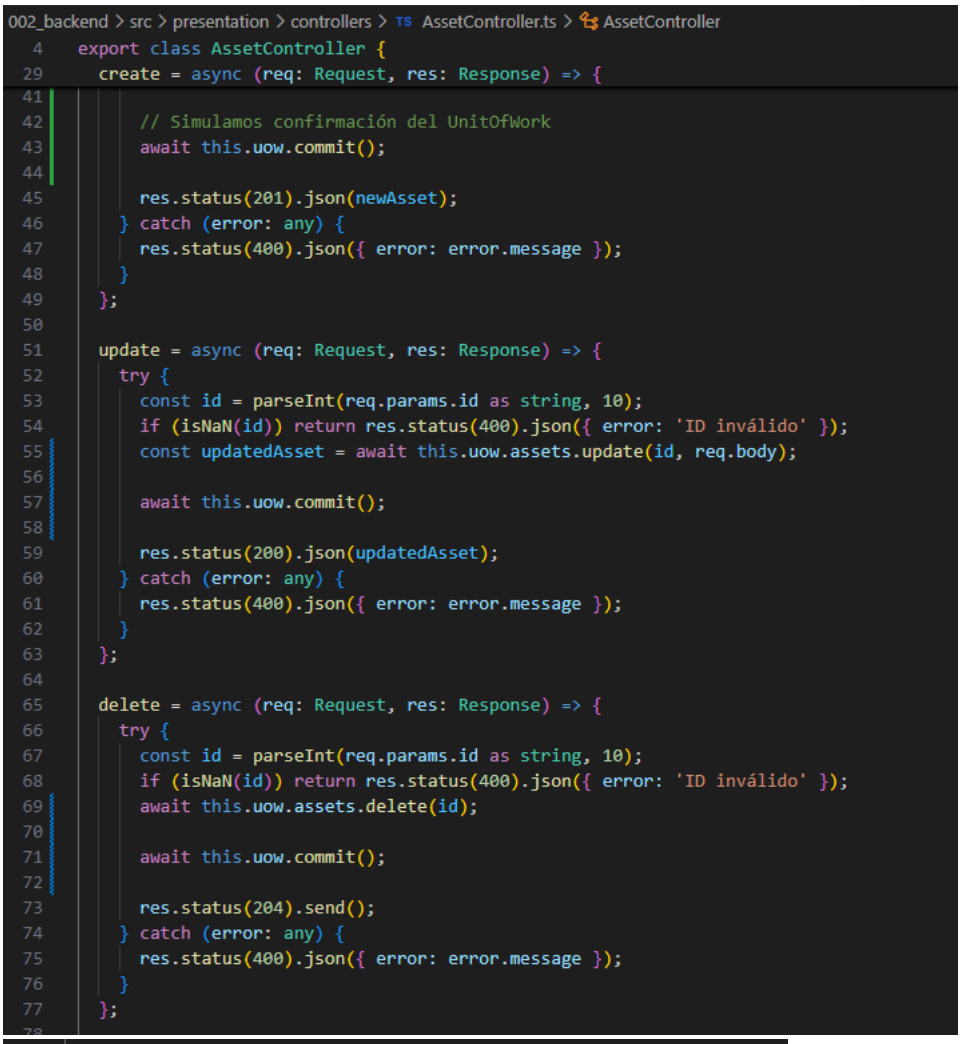
Enunciado:
Construir una API REST mínima que permita gestionar activos del sistema (CRUD) y responder una consulta agregada de exposición. El diseño debe reflejar el uso profesional de una capa de acceso a datos mediante Repository Pattern + Unit of Work, y debe incluir 1–2 tests unitarios sencillos.

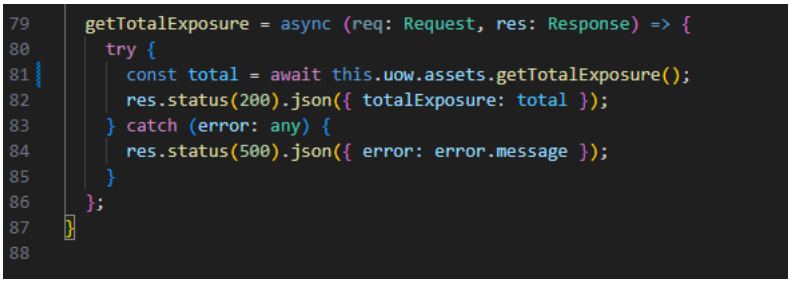
---

1. Código de los endpoints (AssetController)

Se ha implementado un controlador (`AssetController.ts`) que expone los endpoints requeridos para el CRUD básico y la consulta agregada, delegando toda la lógica de persistencia al `UnitOfWork`.


000_docker_compose.md
server.js
001_sql.md
tarea_01_modelo_sql.md
AssetController.ts 1
init.sql
Preview tarea_
002_backend > src > presentation > controllers > AssetController.ts > AssetController
import { Request, Response } from 'express';
import { UnitOfWork } from '../../domain/repositories/UnitOfWork';
export class AssetController {
// Inicializamos el patrón Unit Of Work en lugar del repositorio directamente
private uow = new UnitOfWork();
getAll = async (req: Request, res: Response) => {
try {
const assets = await this.uow.assets.findAll();
res.status(200).json(assets);
} catch (error: any) {
res.status(500).json({ error: error.message });
}
};
getById = async (req: Request, res: Response) => {
try {
const id = parseInt(req.params.id as string, 10);
if (isNaN(id)) return res.status(400).json({ error: 'ID inválido' });
const asset = await this.uow.assets.findById(id);
if (!asset) return res.status(404).json({ error: 'Activo no encontrado' });
res.status(200).json(asset);
} catch (error: any) {
res.status(500).json({ error: error.message });
}
};
create = async (req: Request, res: Response) => {
try {
const { name, latitude, longitude, category_id, base_value, hazard_id, exposure_value, condition_id } = req.body;
const hazard = hazard_id && exposure_value ? { id: parseInt(hazard_id), value: parseFloat(exposure_value) } : undefined;
const condition = condition_id ? parseInt(condition_id) : undefined;
const newAsset = await this.uow.assets.create(
{ name, latitude, longitude, category_id, base_value },
hazard,
condition
);





2. Patrón de Repositorio y Unit of Work

Para abstraer el acceso a datos y asegurar consistencia transaccional, se ha implementado el **Repository Pattern** junto con el **Unit of Work**.

- **Repository Pattern:** Centraliza las consultas a la base de datos (mediante Prisma ORM), aislando la capa de presentación de la capa de datos.
- **Unit of Work:** Actúa como coordinador, permitiendo agrupar múltiples operaciones transaccionales y asegurar que se completen atómicamente antes de hacer `commit()`.

```ts
import { PrismaClient } from '@prisma/client';
import { AssetRepository } from './AssetRepository';

export class UnitOfWork {
  private prisma: PrismaClient;
  public assets: AssetRepository;

  constructor() {
    this.prisma = new PrismaClient();
    this.assets = new AssetRepository(this.prisma);
  }

  // En una base de datos tradicional, aquí iniciaríamos transacción, commit o rollback.
  // Prisma gestiona las transacciones internamente, pero exponer la API mantiene el patrón.
  async commit() {
    // Commit lógico para el patrón UoW
    console.log("UnitOfWork: Transacción completada.");
  }

  async disconnect() {
    await this.prisma.$disconnect();
  }
}

```

000_docker_compose.md | server.js | 001_sql.md | tarea_01

002_backend > src > domain > repositories > AssetRepository.ts > ...

```ts
import { PrismaClient, Asset } from '@prisma/client';

export class AssetRepository {
  private prisma: PrismaClient;

  constructor(prismaClient?: PrismaClient) {
    this.prisma = prismaClient || new PrismaClient();
  }

  async findAll() {
    return this.prisma.asset.findMany({
      include: {
        Category: true,
        AssetHazardExposure: {
          include: { Hazard: true },
        },
        AssetConditions: {
          include: { Condition: true },
        },
      },
    });
  }

  async findById(id: number) {
    return this.prisma.asset.findUnique({
      where: { id },
      include: {
        Category: true,
        AssetHazardExposure: {
          include: { Hazard: true },
        },
        AssetConditions: {
          include: { Condition: true },
        },
      },
    });
  }

```

3. Pruebas Unitarias (Jest)

Se han desarrollado pruebas unitarias automatizadas (utilizando el entorno Jest con mocks) para validar de manera aislada la lógica del repositorio:

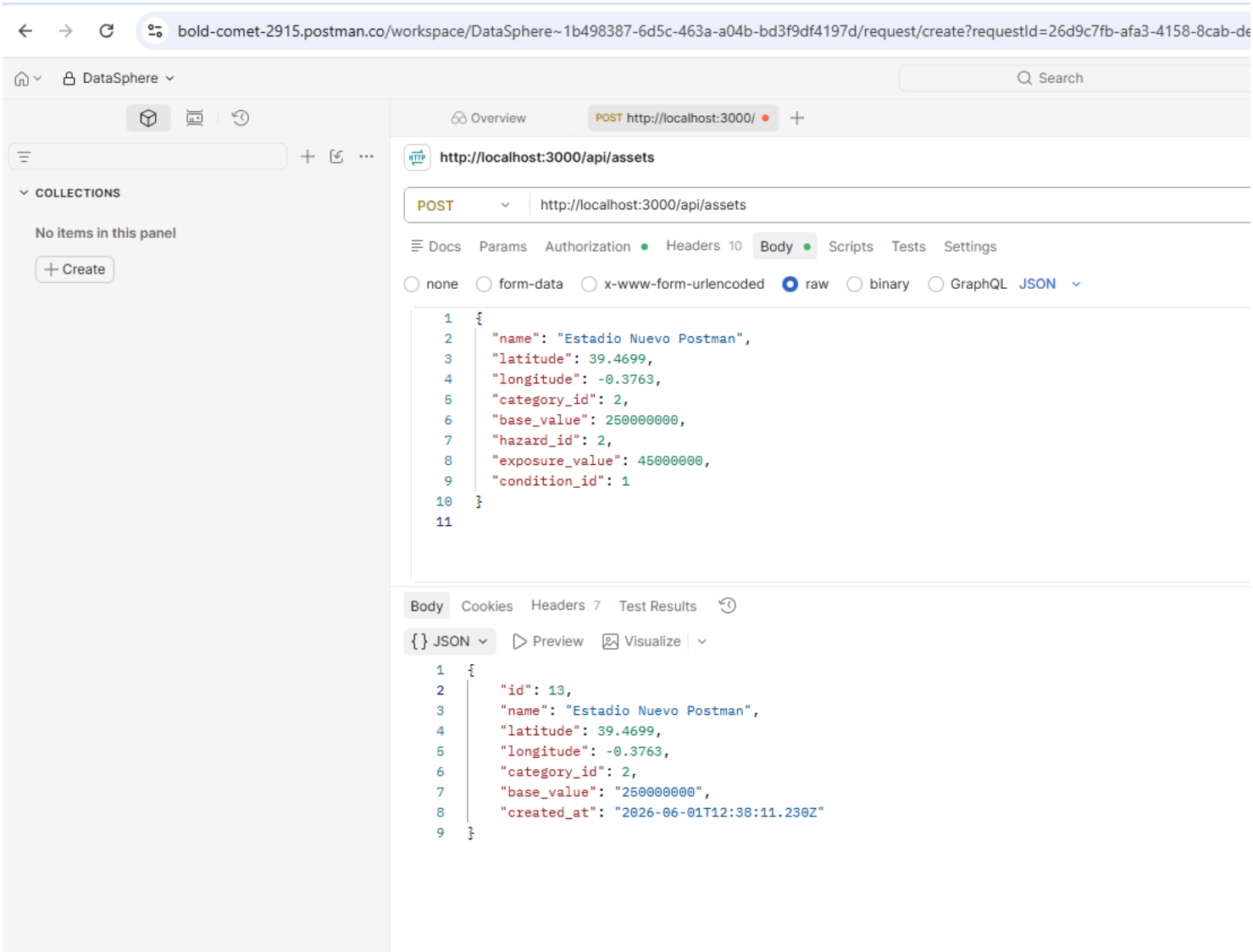
```
tonilogar@SELVA:~/trabajos/programming/DataSphere/002_backend$ npm test

> backend@1.0.0 test
> jest

 PASS  src/tests/asset.test.ts
  AssetRepository - Unit Tests
    ✓ getTotalExposure() debería devolver el valor agregado matemático correcto (3 ms)
    ✓ findById() debería buscar y devolver un único activo por su ID (1 ms)

Test Suites: 1 passed, 1 total
Tests:       2 passed, 2 total
Snapshots:   0 total
Time:        1.083 s, estimated 2 s
Ran all test suites.
tonilogar@SELVA:~/trabajos/programming/DataSphere/002_backend$
```

4. Pruebas de funcionamiento en Vivo (Postman)

Se demuestra su correcto funcionamiento en un entorno vivo mediante la creación de un nuevo activo (POST) y la consulta de activos (GET):

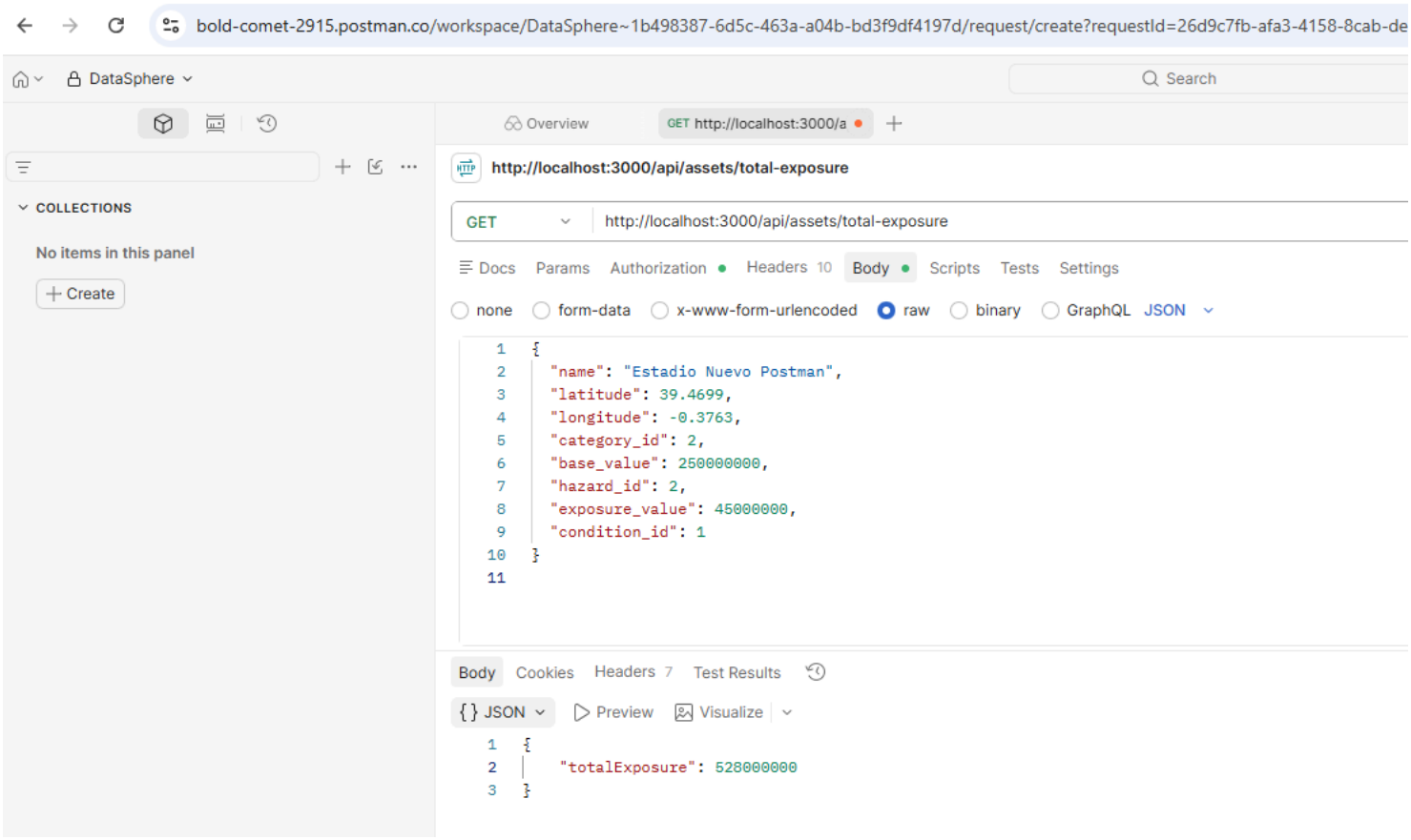
A. Operación CRUD: Creación de un activo (POST)

Petición a `/api/assets` devolviendo una respuesta HTTP `201 Created` tras insertar correctamente un activo (ficticio) superando las validaciones del backend.



B. Consulta Agregada: Exposición Total (GET)

El endpoint `/api/assets/total-exposure` devuelve correctamente la suma de todos los valores de riesgo, delegando la agregación matemática directamente a nivel de base de datos para máxima eficiencia.



TAREA 3 — Seguridad aplicada

Enunciado:
Aplica autenticación y autorización a uno o más endpoints críticos, simulando la protección de información sensible dentro de DataSphere. Protege el endpoint con un rol específico (ej. RiskAnalyst) y demuestra con Postman una petición denegada seguida de una petición permitida.

---

1. Protección del Endpoint con JWT y Roles

Para proteger los valores económicos sensibles de la base de datos (como estipula el ingeniero de seguridad Samir), hemos implementado un middleware de autenticación por JWT (`authenticateJWT`) y un middleware de autorización estricta basada en roles (`requireRole`).

Específicamente, hemos restringido la creación, actualización y borrado de activos al rol especializado **`RiskAnalyst`**. El código en nuestro enrutador (`asset.routes.ts`) ha sido blindado de la siguiente manera:

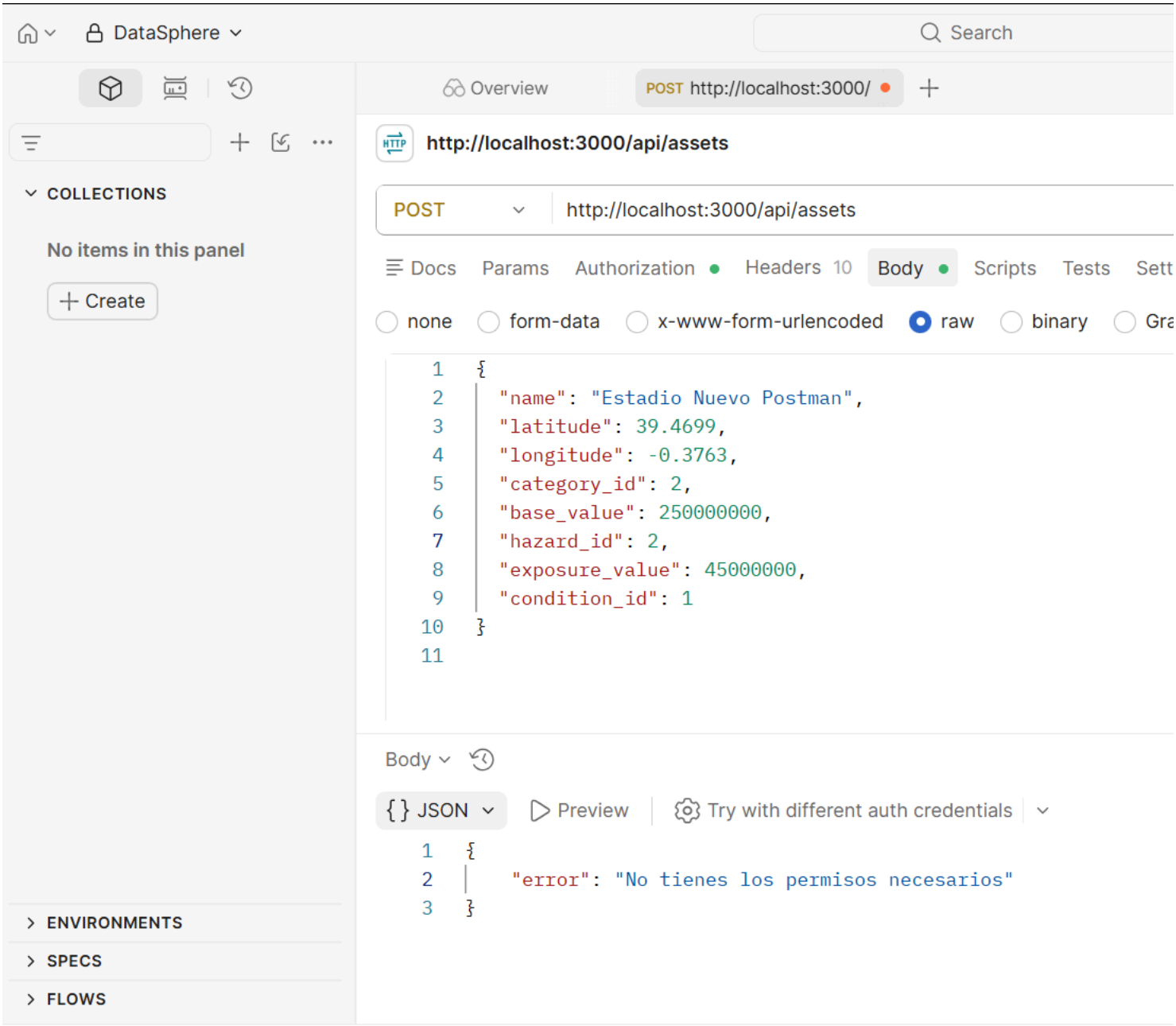
```
// Rutas protegidas con alta seguridad para operaciones de escritura
router.get('/', assetController.getAll);
router.post('/', requireRole('RiskAnalyst'), assetController.create);
router.get('/:id', assetController.getById);
router.put('/:id', requireRole('RiskAnalyst'), assetController.update);
router.delete('/:id', requireRole('RiskAnalyst'), assetController.delete);
```

2. Demostración del flujo de seguridad (Postman)

A continuación se demuestra el funcionamiento real de esta capa de seguridad en la API mediante dos peticiones consecutivas al endpoint protegido `POST /api/assets`.

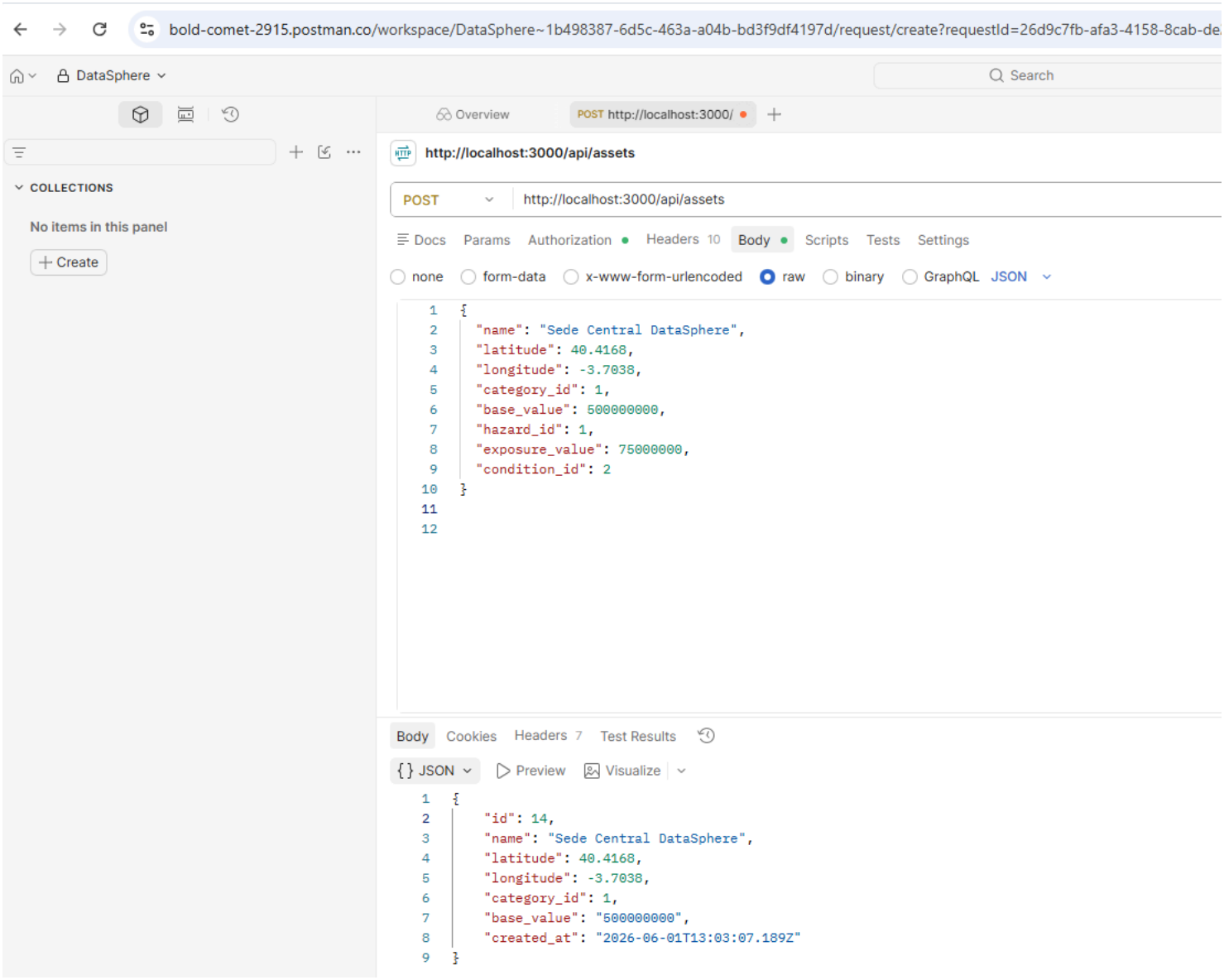
A. Petición Denegada (403 Forbidden)

Se intenta insertar un activo utilizando un token JWT estructuralmente válido, pero que pertenece a un usuario que **no tiene** el rol de `RiskAnalyst` (un usuario estándar). La API bloquea la transacción instantáneamente por seguridad y devuelve el error de permisos insuficientes.



B. Petición Permitida (201 Created)

Tras iniciar sesión con un usuario que **sí posee** el rol de `RiskAnalyst`, el sistema expide un nuevo JWT con los "claims" autorizados. Al inyectar este nuevo token en las cabeceras (*Bearer Token*), el middleware aprueba la validación y permite la ejecución del controlador. El activo se registra exitosamente.

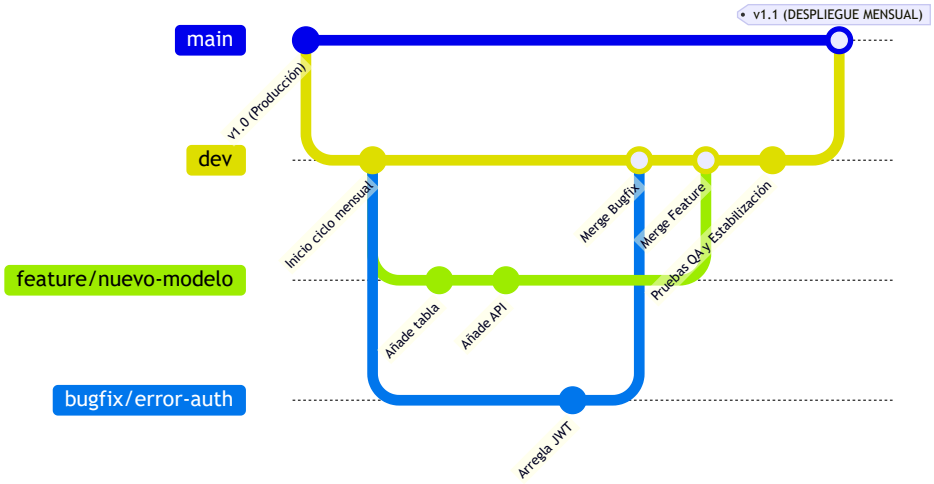


TAREA 4 — Git Strategy corporativa y ciclo de despliegue

Enunciado:
Diseñar una estrategia de ramas en Git que permita a varios desarrolladores trabajar en paralelo, integrar cambios de forma controlada y realizar despliegues mensuales de la aplicación.

---

1. Diagrama de la Estrategia Git



2. Explicación del Flujo de Trabajo

Este flujo de trabajo se basa en un modelo de ramas tipo *GitFlow* simplificado, ideal para equipos que requieren un entorno de pruebas compartido y despliegues programados en fechas concretas.

- **Dónde trabajan los desarrolladores:** Nunca se trabaja directamente sobre `main` ni `dev`. Cuando un programador recibe una tarea, crea una rama efímera basada en `dev` (por ejemplo, `feature/nuevo-modelo` para una nueva funcionalidad, o `bugfix/error-auth` para solucionar un error urgente de código). Aquí es donde ocurre la programación en paralelo sin bloquear a los demás compañeros.
- **Dónde se integran los cambios:** Una vez finalizada la tarea, el código no se empuja directamente. El desarrollador abre un **Pull Request (PR)** hacia la rama `dev`. Esto permite que otros miembros del equipo revisen el código (Code Review). Si todo es correcto, se acepta el PR y el código se fusiona (`merge`) de vuelta hacia la rama `dev`. La rama `dev` actúa como un entorno de **Integración Continua (CI)**, donde se junta el código de todos los miembros del equipo y se realizan las pruebas internas.
- **Cuándo se libera una versión estable y qué ocurre con main:** Al finalizar el ciclo de 30 días, se fusiona el código estabilizado de `dev` hacia la rama **`main`**. En ese instante se genera un "Tag" (etiqueta de versión, ej. `v1.1`), lo que acciona el sistema automático para enviar el código a los servidores en vivo.

3. Relación con el Despliegue Mensual

- **Cuándo se decide cerrar una versión:** Se produce un "Code Freeze" a final de mes. Todo el código que haya logrado ser integrado y probado en `dev` para esa fecha formará parte de la nueva versión. Lo que no esté terminado se quedará en su rama `feature/` esperando al ciclo del mes siguiente.
- **Qué rama se considera "lista para producción":** La rama **`main`** es estrictamente un reflejo del código que están usando los clientes reales. Todo lo que entra a `main` se considera 100% probado y listo.
- **Por qué este modelo facilita los despliegues controlados:** Al separar `main` de `dev`, el equipo puede seguir programando el Día 1 del mes siguiente sin miedo a romper el servidor de producción. Toda la inestabilidad propia del desarrollo ocurre lejos de los clientes, garantizando una liberación mensual segura y sin estrés.
