IES Virgen del Carmen

JUNTA DE ANDALUCIA
CONSEJERÍA DE EDUCACIÓN

Departamento de Informática

Desarrollo de Aplicaciones con Bases de Datos Documentales

Juan Gualberto Gutiérrez Marín

Febrero 2025

UNIÓN EUROPEA
Fondo Social Europeo
El FSE invierte en tu futuro

GOBIERNO DE ESPAÑA
MINISTERIO DE EDUCACIÓN Y FORMACIÓN PROFESIONAL

> AVISO: El texto contiene muchos enlaces embebidos que sólo funcionan si se abre el PDF desde un lector digital. Si se imprime, se perderá mucha información interesante. No obstante, en el apartado “Bibliografía” se han añadido los enlaces para aquellos lectores que deseen una copia impresa y puedan ver la información.

> Para una versión actualizada de este libro visita esta Web: https://gitlab.iesvirgendelcarmen.com/juangu/adt07-flask-mongo-praetorian-reservas.

Índice

1 Desarrollo de Aplicaciones con Bases de Datos Documentales

¿Por qué bases de datos NoSQL (Not Only SQL)?

En los últimos años, están tomando cada vez más importancia las bases de datos NoSQL para persistencia de datos, gracias a su rapidez y eficacia, fácil configuración y cercanía al modelo físico (ejemplo: servicios RESTful con JSON).

Así, por citar algunas de sus ventajas:

- No necesitas un esquema
  - No hay que definir tablas para contener datos
- No es relacional
  - No hay que relacionar tablas (p.ej. Claves foráneas)
- No requiere hardware especial
  - No necesitamos invertir en superservidores
  - Las BBDD NoSQL son fácilmente escalables, añadimos más servidores al clúster (nube) y listo
- Altamente distribuible * Al no estar los datos en tablas, se pueden distribuir libremente en el clúster

Tipos de BBDD NoSQL:

- Basados en columnas
  - Permite más facilidad para hacer medias, varianza, etc.
- Clave-valor
  - como en la base de datos Redis o JSON
- Tripletes de
  - el objeto que describes
  - relaciones con otros objetos
  - valor
- Documentos
  - JSON
  - XML
  - BLOB

    * texto

MongoDB

Mongo es una base de datos documental NoSQL (Not Only SQL) organizada en bases de datos como las bases de datos de MySQL o algo similar a los Schemas de Oracle A su vez las bases de datos se organizan en colecciones lo que en relacional serían las tablas.

Mongo está muy relacionado con el mundo JavaScript y NodeJS porque la información que almacena y gestiona está en formato JSON.

1.1 Repaso JSON

Recordamos que JSON (JavaScript Object Notation) es un objeto JavaScript con el formato:

```
1 { clave: "valor" }
```

Ejemplo: Imagina que quieres almacenar una "persona", pues un ejemplo de este objeto sería así:

```
1 {
2  "Nombre":"John",
3  "Apellidos":"Doe",
4  "DNI":"12345678Z",
5  "FechaNac": "01-01-1980",
6  "Sexo": "V"
7 }
```

1.2 JSON Schema

Si podemos crear una base de datos con documentos en cualquier formato y cualquier esquema, la programación se nos puede complicar.

Por suerte disponemos de herramientas como JSON Schema que nos permite "moldear" los documentos. Para el ejemplo anterior la sintaxis de un JSON Schema define los diferentes campos que una persona tendría sería así:

```
1 {
2     "$schema": "http://json-schema.org/draft-04/schema#",
3     "description": "A representation of a person, company, organization
          , or place",
4     "type": "object",
5     "required": ["familyName", "givenName"],
6     "properties": {
7         "fn": {
8             "description": "Formatted Name",
```

```
            "type": "string"
        },
        "familyName": { "type": "string" },
        "givenName": { "type": "string" },
        "additionalName": { "type": "array", "items": { "type": "string" } },
        "honorificPrefix": { "type": "array", "items": { "type": "string" } },
        "honorificSuffix": { "type": "array", "items": { "type": "string" } },
        "nickname": { "type": "string" },
        "url": { "type": "string", "format": "uri" },
        "email": {
            "type": "object",
            "properties": {
                "type": { "type": "string" },
                "value": { "type": "string", "format": "email" }
            }
        },
        "tel": {
            "type": "object",
            "properties": {
                "type": { "type": "string" },
                "value": { "type": "string", "format": "phone" }
            }
        },
        "adr": { "$ref": "http://json-schema.org/address" },
        "geo": { "$ref": "http://json-schema.org/geo" },
        "tz": { "type": "string" },
        "photo": { "type": "string" },
        "logo": { "type": "string" },
        "sound": { "type": "string" },
        "bday": { "type": "string", "format": "date" },
        "title": { "type": "string" },
        "role": { "type": "string" },
        "org": {
            "type": "object",
            "properties": {
                "organizationName": { "type": "string" },
                "organizationUnit": { "type": "string" }
            }
        }
    }
}
```

Así, cuando en MongoDB veas referencia a un `schema` ya sabes de donde viene el tema. Cuando insertamos objectos, para optimizar la búsqueda e inserción de datos, Mongo va creando y actualizando los esquemas.

2 Instalación de MongoDB

De la imagen oficial de Mongo, adaptamos el Docker Compose:

```yaml
services:

  mongo:
    image: mongo
    restart: "no"
    ports:
      - 27017:27017
    environment:
      MONGO_INITDB_ROOT_USERNAME: root
      MONGO_INITDB_ROOT_PASSWORD: 78agsbjha7834aSDFjhd73

  mongo-express:
    image: mongo-express
    restart: "no"
    ports:
      - 8081:8081
    environment:
      ME_CONFIG_MONGODB_ADMINUSERNAME: root
      ME_CONFIG_MONGODB_ADMINPASSWORD: 78agsbjha7834aSDFjhd73
      ME_CONFIG_MONGODB_URL: mongodb://root:78
          agsbjha7834aSDFjhd73@mongo:27017/
      ME_CONFIG_BASICAUTH: "false"
```

Para comprobar que ha funcionado, abrimos Mongo Express en local: http://localhost:8081/.

Dejamos propuesto como ejercicio crear en la carpeta del stack un fichero **.env** donde almacenar las credenciales y usar este archivo tanto para el contenedor como para los scripts de Python que haremos después. Aunque nosotros para que funcione el proyecto lo estamos incluyendo en el proyecto, **recuerda que es muy mala idea incluir cualquier tipo de credenciales en un repositorio de software**.

Mongo organiza los documentos en bases de datos, las bases de datos en colecciones y las colecciones tienen documentos. Para entender mejor cómo funciona te presentamos esta tabla de equivalencias:

MySQL	MongoDB
Base de datos	Base de Datos
Tablas	Colecciones
Filas o tuplas	Documentos

Sin colecciones no puedo tener bases de datos en Mongo. Siempre, como mínimo estará la colección

“delete_me”.

2.1 Interactuando con MongoShell

Para abrir una terminal interactiva con el contenedor, lo podemos hacer desde el plugin correspondiente del IDE, desde las propias utilidades de Docker Desktop o bien desde terminal así:

```
1 docker exec -ti [nombre_contenedor_mongostack] /bin/bash
```

Ahora entramos en la shell de Mongo con:

```
1 mongosh -u root
```

Cuando nos pida la contraseña recuerda usar la que has puesto en el archivo `docker-compose.yml`.

A continuación vamos a revisar nuestro caso de reservas de pistas deportivas. Verás ejemplos de operaciones CRUD (Crear, Leer, Actualizar y Borrar) y algunas consultas útiles.

Nuestra base de datos sigue este esquema:

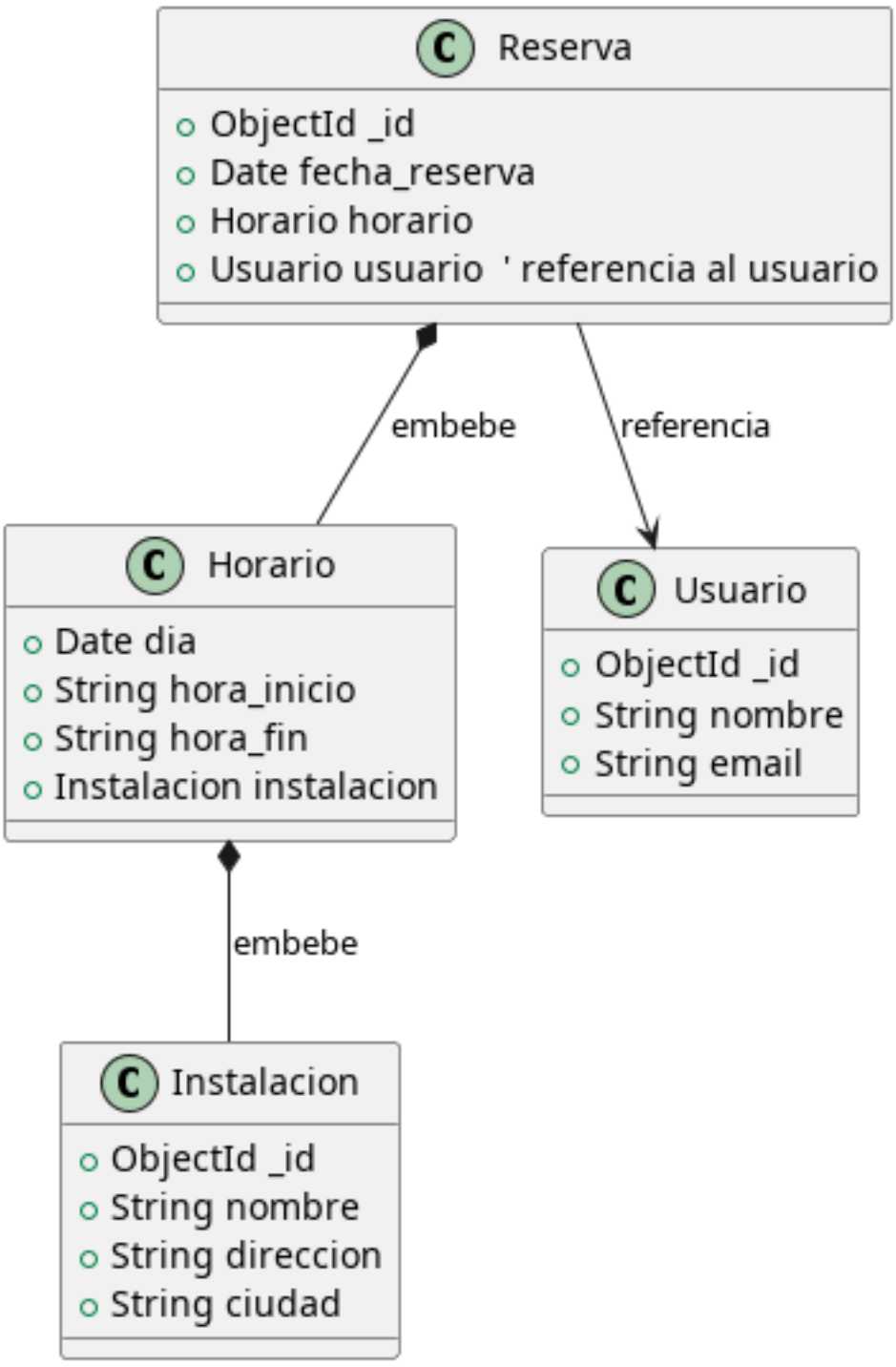

Figura 2: Diagrama PlantUML

2.2 1. Modelo de Datos del Ejemplo

Para nuestro ejemplo, tenemos las siguientes entidades:

- **Instalaciones**: Datos de las pistas o instalaciones deportivas.
- **Horarios**: Disponibilidad de cada instalación. Cada horario lleva *embebida* la información de la instalación (para tener un snapshot de la instalación cuando se crea el horario).
- **Usuarios**: Información de los usuarios que realizan reservas.
- **Reservas**: Cada reserva incluye:
  - La fecha de reserva.
  - Una referencia al usuario (no embebemos el usuario, sino que solo guardamos su `_id`).
  - El horario embebido, con la información del día, hora y la instalación.

2.3 2. Ejemplos de Operaciones CRUD

2.3.1 2.1. Seleccionar la Base de Datos

```
use reservas_db;
```

2.3.2 2.2. Crear (Insertar Documentos)

2.3.2.1 Insertar una Instalación

Insertar una instalación nueva:

```
db.instalaciones.insertOne({
  nombre: "Pista de Tenis",
  direccion: "Calle A 123",
  ciudad: "Madrid"
});
```

2.3.2.2 Insertar un Horario con la Instalación Embebida

Supongamos que ya tenemos la instalación insertada y conocemos su `_id` (por ejemplo, `ObjectId("60f1b2c3d4e5f67890123456")`):

```
db.horarios.insertOne({
  dia: "2025-03-10",
  hora_inicio: "10:00",
  hora_fin: "11:00",
  instalacion: {
    _id: ObjectId("60f1b2c3d4e5f67890123456"),
    nombre: "Pista de Tenis",
    direccion: "Calle A 123"
```

```
  }
});
```

2.3.2.3 Insertar un Usuario

Insertar un Usuario:

```
db.usuarios.insertOne({
  nombre: "Ana García",
  email: "ana.garcia@example.com"
});
```

2.3.2.4 Insertar una Reserva

Imaginemos que:

- El usuario insertado tiene `_id`: `ObjectId("60f1b2c3d4e5f67890123457")`
- El horario (con la instalación embebida) se usó para crear la reserva.

```
db.reservas.insertOne({
  fecha_reserva: "2025-03-01",
  usuario_id: ObjectId("60f1b2c3d4e5f67890123457"),
  horario: {
    dia: "2025-03-10",
    hora_inicio: "10:00",
    hora_fin: "11:00",
    instalacion: {
      _id: ObjectId("60f1b2c3d4e5f67890123456"),
      nombre: "Pista de Tenis",
      direccion: "Calle A 123"
    }
  }
});
```

2.3.3 2.3. Leer (Consultas)

2.3.3.1 Consultar Todas las Instalaciones

Listar todas las Instalaciones

```
db.instalaciones.find().pretty();
```

2.3.3.2 Buscar Horarios para un Día Específico

Buscar Horarios para un día específico:

```
db.horarios.find({ dia: "2025-03-10" }).pretty();
```

2.3.3.3 Buscar Reservas de un Usuario Específico

Reservas de un Usuario específico (por ID):

```
db.reservas.find({ usuario_id: ObjectId("60f1b2c3d4e5f67890123457") }).
    pretty();
```

2.3.3.4 Buscar Reservas para una Instalación

Reservas para una Instalación (usando **dot notation** en el horario):

```
db.reservas.find({ "horario.instalacion.nombre": "Pista de Tenis" }).
    pretty();
```

2.3.4 2.4. Actualizar

2.3.4.1 Actualizar la Dirección de una Instalación

Actualizar la Dirección de una Instalación:

```
db.instalaciones.updateOne(
  { _id: ObjectId("60f1b2c3d4e5f67890123456") },
  { $set: { direccion: "Calle Nueva 456" } }
);
```

> *Nota:* Si la dirección de la instalación cambia y se desea que las modificaciones se reflejen en los horarios o reservas, habría que actualizar esos documentos por separado. Esto es una de las consideraciones al embedir datos.

2.3.4.2 Actualizar el Horario Embebido en una Reserva

Por ejemplo, modificar la hora de inicio y fin de un horario dentro de una reserva:

```
db.reservas.updateOne(
  { _id: ObjectId("60f1b2c3d4e5f67890123458") },
  { $set: { "horario.hora_inicio": "11:00", "horario.hora_fin": "12:00"
      } }
);
```

2.3.5 2.5. Borrar

2.3.5.1 Eliminar un Usuario

Eliminar un Usuario por ID:

```
db.usuarios.deleteOne({ _id: ObjectId("60f1b2c3d4e5f67890123457") });
```

2.3.5.2 Eliminar una Reserva

Eliminar una reserva por ID:

```
db.reservas.deleteOne({ _id: ObjectId("60f1b2c3d4e5f67890123458") });
```

2.4 Objetos Embebidos vs. Objetos Anidados

2.4.1 Objetos Embebidos

- **Definición:** Son documentos completos que se insertan directamente dentro de otro documento.
- **Ventajas:**
  - **Lectura Rápida y Sencilla:** Al estar en un mismo documento, se evitan costosas operaciones de "join".
  - **Atomicidad:** Las operaciones de escritura en el documento completo son atómicas.
- **Desventajas:**
  - **Duplicación de Datos:** Si el objeto embebido es usado en varios documentos, se duplica la información.
  - **Actualizaciones Complejas:** Si la información embebida necesita actualizarse de forma global, se debe actualizar en cada documento donde aparezca.
- **Ejemplo en Nuestro Caso:**
  En la colección `horarios` se embebe la información de la instalación. Esto es útil si el horario necesita conservar un snapshot de la instalación en el momento de creación, sin preocuparse por cambios futuros en la instalación.

3 Mi primera aplicación Flask

Nuestra referencia es el manual de Flask.

Si no tenemos python instalado, lo instalamos:

```
apt install python3.12 python3.12-venv
```

En Windows sería similar pero con `winget`.

Empezamos la instalación.

```
mkdir tuto-flask
cd tuto-flask
python -m venv venv
. ./venv/bin/activate
```

Creamos un archivo `.gitignore`, le añadimos la carpeta `venv`.

Inicializamos el repositorio:

```
git init
```

Añadimos Flask al proyecto:

```
pip install Flask
```

Cada vez que añadimos una dependencia hay que ejecutar esto:

```
pip freeze > requirements.txt
```

Creamos un archivo `hola.py` con este contenido:

```
from flask import Flask

app = Flask(__name__)

@app.route("/")
def hello_world():
    return "<p>Hola Mundo!</p>"
```

Me aseguro que el virtual environment está activo y ejecuto la APP en el puerto 8080:

```
. ./venv/bin/activate
flask --app hola run --port 8080
```