C.4 Laboratorio Semana 4- Angel Gabriel Ortega Corzo

Usando el SHELL de Mongo DB realizar las siguientes operaciones en la base de datos hospital:

Realizar operaciones CRUD en NoSQL (Mongo DB)

- Abrir el shell de mongosh y realice las consultas para poder crear y registrar datos para (historia clínica, receta y medicamento)

en este caso como ya vimos en clases sera bastante sencillo poder realizar estas tareas mediante el shell

Solucion

primero debemos pensar que relacion tendra con las colecciones que tenemos, la mas representativa es cita, donde si fuera sql seria como una relacion de muchos a muchos

ahora pasamos a crear las tablas respectivas, historia medica si o si debe tener una referencia a un usuario espefico, por lo tanto veo necesario añadir el objectid del usuario, a quien le pertenece el registro, usamos la base de datos ejecutando el comando **USE**con el nombre de la base de datos

como tenemos la referencia de la historia clinica entonces podemos de manera facil en receta hacer una descripcion mas a fondo para alguien que tome las riendas del paciente y quiera ver los medicamentos que consumio y las recetas que tiene asociadas, en este caso solo 1, pero podrian ser mas, pero como no tenemos ninguna receta creada seria ilogico crearla, primero debemos ir en orden, entonces primero crearíamos algunos medicamentos.

```
db.medicamentos.insertMany([
  {
    _id: ObjectId("64e9f8e7d9f28610a5a2b85d"),
    nombre: "Ibuprofeno",
    descripcion: "Medicamento para el dolor y fiebre",
    dosis_recomendada: "200mg",
    efectos_secundarios: ["Náuseas", "Dolor de estómago"]
```

```
  },
  {
    _id: ObjectId("64e9f8e7d9f28610a5a2b85e"),
    nombre: "Paracetamol",
    descripcion: "Medicamento para fiebre y dolor",
    dosis_recomendada: "500mg",
    efectos_secundarios: ["acne", "Dolor de cabeza"]
  }
]);
```

```
> use hospital
< switched to db hospital
> db.medicamentos.insertMany([
    {
      _id: ObjectId("64e9f8e7d9f28610a5a2b85d"),
      nombre: "Ibuprofeno",
      descripcion: "Medicamento para el dolor y fiebre",
      dosis_recomendada: "200mg",
      efectos_secundarios: ["Náuseas", "Dolor de estómago"]
    },
    {
      _id: ObjectId("64e9f8e7d9f28610a5a2b85e"),
      nombre: "Paracetamol",
      descripcion: "Medicamento para fiebre y dolor",
      dosis_recomendada: "500mg",
      efectos_secundarios: ["acne", "Dolor de cabeza"]
    }
  ]);
< {
    acknowledged: true,
    insertedIds: {
      '0': ObjectId('64e9f8e7d9f28610a5a2b85d'),
      '1': ObjectId('64e9f8e7d9f28610a5a2b85e')
    }
  }
```

seguidamente pasamos a insertar las recetas con las medicinas

```
db.recetas.insertMany([
  {
    _id: ObjectId("64e9f8e7d9f28610a5a2b85f"),
    paciente_id: ObjectId("66e22039856a8c94e71fbde5"),
    doctor_id: ObjectId("66e2195e18e193ae4c8d4f61"),
    medicamentos: [
      { medicamento_id: ObjectId("64e9f8e7d9f28610a5a2b85d"), dosis:
```

```
"200mg", frecuencia: "2 veces al día" },
      { medicamento_id: ObjectId("64e9f8e7d9f28610a5a2b85e"), dosis:
"500mg", frecuencia: "cada 8 horas" }
    ],
    fecha: new Date("2024-09-12T00:00:00Z"),
    observaciones: "Paciente debe tomar los medicamentos después de las
comidas."
  }
]);
```

```
> db.recetas.insertMany([
    {
      _id: ObjectId("64e9f8e7d9f28610a5a2b85f"),
      paciente_id: ObjectId("66e22039856a8c94e71fbde5"),
      doctor_id: ObjectId("66e2195e18e193ae4c8d4f61"),
      medicamentos: [
        { medicamento_id: ObjectId("64e9f8e7d9f28610a5a2b85d"), dosis: "200mg", frecuencia: "2 veces al día" },
        { medicamento_id: ObjectId("64e9f8e7d9f28610a5a2b85e"), dosis: "500mg", frecuencia: "cada 8 horas" }
      ],
      fecha: new Date("2024-09-12T00:00:00Z"),
      observaciones: "Paciente debe tomar los medicamentos después de las comidas."
    }
  ]);
< {
    acknowledged: true,
    insertedIds: {
      '0': ObjectId('64e9f8e7d9f28610a5a2b85f')
    }
  }
hospital>
```

y finalmente las historias donde asociamos las recetas, coloque la receta como un arreglo para que en dado caso una historia tenga mas de una receta

```
db.historiasClinicas.insertOne({
  _id: ObjectId("64e9f8e7d9f28610a5a2b860"),
  paciente_id: ObjectId("66e22039856a8c94e71fbde5"),
  fecha_inicio: new Date("2023-01-15T00:00:00Z"),
  enfermedades: ["Diabetes", "Hipertension"],
  recetas: [
    ObjectId("64e9f8e7d9f28610a5a2b85f")
  ],
  observaciones: "Paciente muestra signos de cáncer. Requiere seguimiento y
ajuste de tratamiento."
});
```

```
> db.historiasClinicas.insertOne({
    _id: ObjectId("64e9f8e7d9f28610a5a2b860"),
    paciente_id: ObjectId("66e22039856a8c94e71fbde5"),
    fecha_inicio: new Date("2023-01-15T00:00:00Z"),
    enfermedades: ["Diabetes", "Hipertension"],
    recetas: [
      ObjectId("64e9f8e7d9f28610a5a2b85f")
    ],
    observaciones: "Paciente muestra signos de cáncer. Requiere seguimiento y ajuste de tratamiento."
  });
< {
    acknowledged: true,
    insertedId: ObjectId('64e9f8e7d9f28610a5a2b860')
  }
```

- Utilizando mongosh cree y elimine una base de datos llamada "SeguroMedico" (Investigacion)

Entonces con unos cuantos comandos realizamos de manera casi demasiada sencilla, solo con un **USE** la creamos y luego con un **DB.dropdatabase();**la podemos eliminar de forma instantanea

```
> use SeguroMedico
< switched to db SeguroMedico
> db.dropDatabase();
< { ok: 1, dropped: 'SeguroMedico' }
```

- INDEX
  Crear los Indices necesarios para las colecciones (tablas) identificadas en los campos que usted vea necesario crear, y haga una pequeña explicacion de por que se creo.

Para poder crear estos indices como en sql nos ayudan a parametrizar o asignar indice a un campo de la coleccion, lo cual hace que cuando tengamos millones de registros no busque uno por uno.

```
db.recetas.createIndex({ "medicamentos.medicamento_id": 1 });
```

mediante este parametrizamos el id del medicamento, esto puede servir cuando queramos usar el id_medicamento desde la coleccion de recetas

```
db.historiasClinicas.createIndex({ recetas: 1 });
```

```
>_MONGOSH
> db.recetas.createIndex({ "medicamentos.medicamento_id": 1 });
< medicamentos.medicamento_id_1
> db.historiasClinicas.createIndex({ recetas: 1 });
< recetas_1
hospital>
```

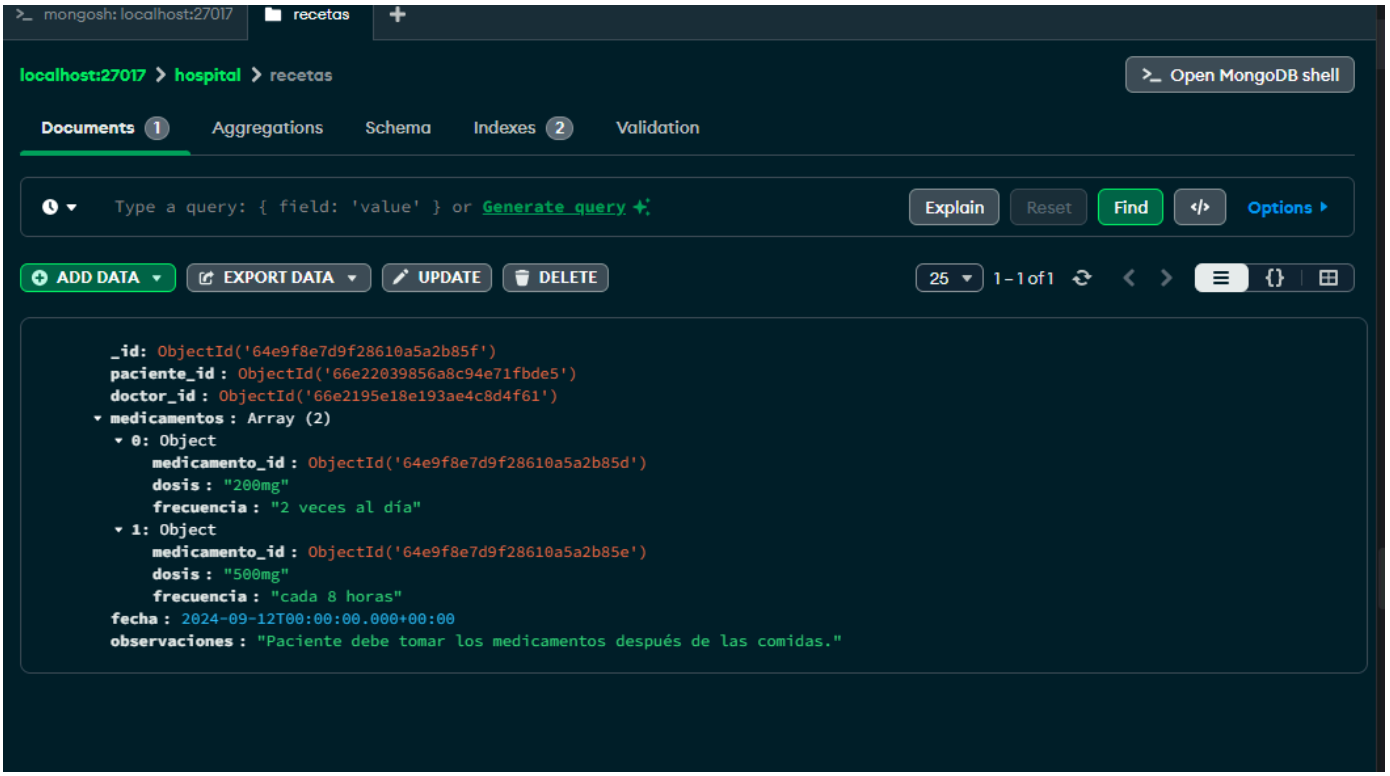
- AGREGACIONES
  Realize 6 consultas usando $(group) con los operadores$(avg, $count, $first, $sortByCount, $min y $sum) en una Collecion(Tabla) y sus campos que usted vea necesario

este tipo de operadores y funciones que tiene por defecto el motor por lo general se usan en los triggers que todos conocemos en sql pero en mongo por el momento lo haremos en consultas lo mas simple posible, pero siempre teniendo presente la funcion aggregate ya que es la funcion que siempre estara en las consultas

#1

```
db.recetas.aggregate([
  {
    $group: {
      _id: null,
      promedioMedicamentos: { $avg: { $size: "$medicamentos" } }
    }
  }
]);
```

como podemos ver el promedio de medicamentos dentro de cada receta es de 2, lo cual es bastante obvio porque solo tenemos un unico registro en las recetas, el cual contiene 2 object id de medicamentos.



#2

```
db.recetas.aggregate([
  {
    $group: {
      _id: null,
      totalRecetas: { $count: {} }
    }
  }
]);
```

```
> db.recetas.aggregate([
    {
      $group: {
        _id: null,
        totalRecetas: { $count: {} }
      }
    }
  ]);
< {
    _id: null,
    totalRecetas: 1
  }
```

esta es bastante simple ya que solo lista la cantidad de recetas, que existen dentro de la coleccion.

#3

```
db.recetas.aggregate([
  {
    $group: {
      _id: null,
      primerMedicamento: { $first: "$medicamentos" }
    }
  }
]);
```

```
> db.recetas.aggregate([
    {
      $group: {
        _id: null,
        primerMedicamento: { $first: "$medicamentos" }
      }
    }
  ]);
< {
    _id: null,
    primerMedicamento: [
      {
        medicamento_id: ObjectId('64e9f8e7d9f28610a5a2b85d'),
        dosis: '200mg',
        frecuencia: '2 veces al día'
      },
      {
        medicamento_id: ObjectId('64e9f8e7d9f28610a5a2b85e'),
        dosis: '500mg',
        frecuencia: 'cada 8 horas'
      }
    ]
  }
```

en este caso esta consulta usamos el **$FIRST**para poder obtener el primer registro del campo esperado, en este caso los medicamentos

#4

```
db.recetas.aggregate([
  { $unwind: "$medicamentos" },
  {
    $group: {
      _id: "$medicamentos.medicamento_id",
      total: { $sum: 1 }
```

```
    }
  },
  { $sort: { total: -1 } }
]);
```

```
> db.recetas.aggregate([
    { $unwind: "$medicamentos" },
    {
      $group: {
        _id: "$medicamentos.medicamento_id",
        total: { $sum: 1 }
      }
    },
    { $sort: { total: -1 } }
  ]);
< {
    _id: ObjectId('64e9f8e7d9f28610a5a2b85e'),
    total: 1
  }
  {
    _id: ObjectId('64e9f8e7d9f28610a5a2b85d'),
    total: 1
  }
```

#5

```
db.medicamentos.aggregate([
  {
    $group: {
      _id: null,
      dosisMinima: { $min: "$dosis_recomendada" }
    }
  }
]);
```

mediante esta consulta usamos el min para encontrar la menor dosis recomendanda entre las medicinas.

#6

```
db.recetas.aggregate([
  {
    $match: { "medicamentos.medicamento_id":
ObjectId("64e9f8e7d9f28610a5a2b85d") }
  },
  {
    $group: {
      _id: "$medicamentos.medicamento_id",
      totalRecetas: { $sum: 1 }
    }
  }
]);
```

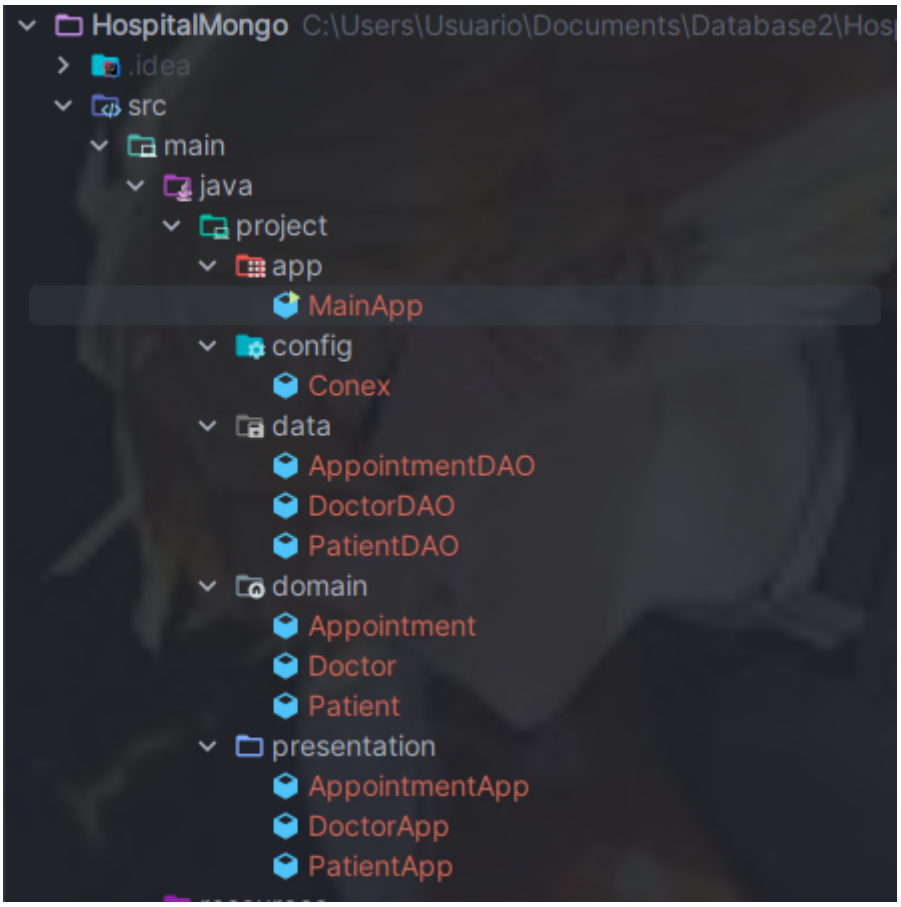
```
> db.recetas.aggregate([
    {
      $match: { "medicamentos.medicamento_id": ObjectId("64e9f8e7d9f28610a5a2b85d") }
    },
    {
      $group: {
        _id: "$medicamentos.medicamento_id",
        totalRecetas: { $sum: 1 }
      }
    }
  ]);
< {
    _id: [
      ObjectId('64e9f8e7d9f28610a5a2b85d'),
      ObjectId('64e9f8e7d9f28610a5a2b85e')
    ],
    totalRecetas: 1
  }
```

mediante esta obtenemos todas las recetas que contengan un medicamento de nuestra preferencia.

Conecciones a Bases de Datos NoSQL utilizando un lenguaje de programacion

- Establecer una conexión

para poder hacer esto podemos hacer uso del mismo patron que usamos en el pasado lab, creando la conexion, la entidad, la capa de mapeo a la db y finalmente donde lo vamos a consumir o la presentacion.



con la anterior arquitectura. de cierta manera separo las responsabilidades, y luego unifico todo en el main app para que se pueda seleccionar con cual entidad de preferencia trabajar:

```
package project.app;

import com.mongodb.client.MongoDatabase;
import project.config.Conex;
import project.presentation.AppointmentApp;
import project.presentation.DoctorApp;
import project.presentation.PatientApp;

import java.util.Scanner;

public class MainApp {

    public static void main(String[] args) {
        Conex conex = new Conex();
        conex.conectar();
        MongoDatabase database = conex.getDatabase();

        Scanner scanner = new Scanner(System.in);
        int option;

        do {
```

```
            System.out.println("\n---- Menú Principal ----");
            System.out.println("1. Gestionar Citas");
            System.out.println("2. Gestionar Pacientes");
            System.out.println("3. Gestionar Doctores");
            System.out.println("0. Salir");
            System.out.print("Elige una opción: ");
            option = scanner.nextInt();
            scanner.nextLine();

            switch (option) {
                case 1:
                    AppointmentApp.run(database);
                    break;
                case 2:
                    PatientApp.run(database);
                    break;
                case 3:
                    DoctorApp.run(database);
                    break;
                case 0:
                    System.out.println("Saliendo...");
                    break;
                default:
                    System.out.println("Opción no válida. Intente
nuevamente.");
            }
        } while (option != 0);

        conex.cerrar();
        scanner.close();
    }
}
```

primero que todo accedemos a la entidad de conexion que es la principal orquestadora y en el case podemos ver que el argumento principal que le paso a cada presentacion de cada entidad, ahora, veremos como estructure la conexion. basandonos en la documentacion oficial [documentacion mongo java]

```
package project.config;

import com.mongodb.client.MongoClient;
import com.mongodb.client.MongoClients;
import com.mongodb.client.MongoDatabase;
```

```
public class Conex {

    private MongoClient mongoClient;
    private MongoDatabase database;

    public void conectar() {
        mongoClient = MongoClients.create("mongodb://localhost:27017");
        database = mongoClient.getDatabase("hospital");
        System.out.println("Conexión establecida con MongoDB");
    }

    public MongoDatabase getDatabase() {
        return database;
    }

    public void cerrar() {
        if (mongoClient != null) {
            mongoClient.close();
            System.out.println("Conexión cerrada");
        }
    }
}
```

de esta manera podemos usar la conexion y sus metodos unicos desde cualquier parte de nuestra app, incluyendo su nucleo el cual es el dao pero solo en main.app llamaremos a la db real.

Dao ejemplo

```
package project.data;

import com.mongodb.client.MongoCollection;
import com.mongodb.client.MongoDatabase;
import org.bson.Document;
import org.bson.types.ObjectId;
import project.domain.Patient;

import java.util.ArrayList;
import java.util.List;

public class PatientDAO {

    private final MongoCollection<Document> collection;

    public PatientDAO(MongoDatabase database) {
```

```
        this.collection = database.getCollection("paciente");
    }

    public void create(Patient patient) {
        Document doc = new Document()
                .append("nombre", patient.getNombre())
                .append("edad", patient.getEdad())
                .append("género", patient.getGenero());

        collection.insertOne(doc);

        ObjectId id = doc.getObjectId("_id");
        patient.setId(id.toString());
    }

    public List<Patient> getAll() {
        List<Patient> patients = new ArrayList<>();
        for (Document doc : collection.find()) {
            patients.add(convertToPatient(doc));
        }
        return patients;
    }

    public Patient getById(String id) {
        Document doc = collection.find(new Document("_id", id)).first();
        return doc != null ? convertToPatient(doc) : null;
    }

    public void update(String id, Patient patient) {
        Document updatedDoc = new Document("nombre", patient.getNombre())
                .append("edad", patient.getEdad())
                .append("género", patient.getGenero());
        collection.updateOne(new Document("_id", id), new Document("$set",
updatedDoc));
    }

    public void delete(String id) {
        collection.deleteOne(new Document("_id", id));
    }

    private Patient convertToPatient(Document doc) {
        return new Patient(
                doc.getObjectId("_id").toString(),
```

```
                doc.getString("nombre"),
                doc.getInteger("edad"),
                doc.getString("género")
        );
    }
}
```

- Ejecutar comandos NoSQL como operaciones CRUD

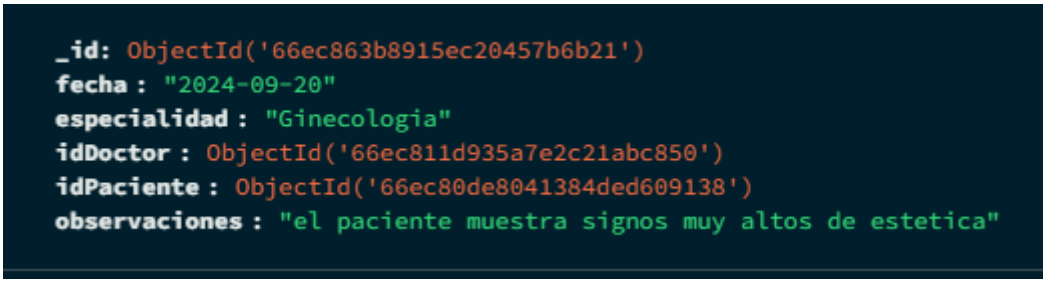
el sistema soporta cada una de las operaciones ya mencionadas y cada presentacion cuenta con una interfaz intuitiva para hacerlo, exepto en cita, porque de alguna manera debemos saber el object id del medico y paciente para poder crearla, sino, es complicado hacerlo.

```
Conexión establecida con MongoDB

---- Menú Principal ----
1. Gestionar Citas
2. Gestionar Pacientes
3. Gestionar Doctores
0. Salir
Elige una opción: 2

---- Menú de Pacientes ----
1. Crear nuevo paciente
2. Mostrar todos los pacientes
3. Buscar paciente por ID
4. Actualizar paciente
5. Eliminar paciente
0. Volver al menú principal
Elige una opción: 2

--- Mostrar Todos los Pacientes ---
Patient{id='66e22039856a8c94e71fbde5', nombre='Juliana UnU', edad=19,
genero='Femenino'}
Patient{id='66e22055856a8c94e71fbde6', nombre='Alana', edad=25,
genero='Femenino'}
Patient{id='66ec80de8041384ded609138', nombre='angel', edad=18,
genero='masculino'}

---- Menú de Pacientes ----
1. Crear nuevo paciente
2. Mostrar todos los pacientes
3. Buscar paciente por ID
```

```
4. Actualizar paciente
5. Eliminar paciente
0. Volver al menú principal
Elige una opción: 0
Volviendo al menú principal...

---- Menú Principal ----
1. Gestionar Citas
2. Gestionar Pacientes
3. Gestionar Doctores
0. Salir
Elige una opción: 3

---- Menú de Doctores ----
1. Crear nuevo doctor
2. Mostrar todos los doctores
3. Buscar doctor por ID
4. Actualizar doctor
5. Eliminar doctor
0. Volver al menú principal
Elige una opción: 2

--- Mostrar Todos los Doctores ---
Doctor{id='66e2195e18e193ae4c8d4f61', nombre='Dra. perolito',
especialidad='Endocrinologia', añosexperiencia=10}
Doctor{id='66ec811d935a7e2c21abc850', nombre='Dra. Alexandra',
especialidad='Ginecologia', añosexperiencia=6}

---- Menú de Doctores ----
1. Crear nuevo doctor
2. Mostrar todos los doctores
3. Buscar doctor por ID
4. Actualizar doctor
5. Eliminar doctor
0. Volver al menú principal
Elige una opción: 0
Volviendo al menú principal...

---- Menú Principal ----
1. Gestionar Citas
2. Gestionar Pacientes
3. Gestionar Doctores
0. Salir
```

```
Elige una opción: 1
*** Sistema de Gestión de Citas ***
1. Crear nueva cita
2. Mostrar todas las citas
3. Buscar cita por ID
4. Actualizar cita
5. Eliminar cita
0. Volver al menú principal
Elige una opción:
1
Registrar Nueva Cita:
Fecha (YYYY-MM-DD): 2024-09-20
Especialidad: Ginecologia
ID del Doctor: 66ec811d935a7e2c21abc850
ID del Paciente: 66ec80de8041384ded609138
Observaciones: el paciente muestra signos muy altos de estetica
Cita registrada exitosamente: Appointment{id='66ec863b8915ec20457b6b21',
fecha=2024-09-20, especialidad='Ginecologia',
idDoctor='66ec811d935a7e2c21abc850', idPaciente='66ec80de8041384ded609138',
observaciones='el paciente muestra signos muy altos de estetica'}

*** Sistema de Gestión de Citas ***
1. Crear nueva cita
2. Mostrar todas las citas
3. Buscar cita por ID
4. Actualizar cita
5. Eliminar cita
0. Volver al menú principal
Elige una opción:
0
Volviendo al menú principal...
```

podemos ver que se creo exitosamente



repositorio fuente para clonar o vizualizar el proyecto:
[repositorio del proyecto]

- Manejo de excepciones (Investigacion)

lo mas cercano que pude encontrar fueron las transacciones que permiten realizar operaciones atomicas. pero al igual que en el proyecto anterior, podemos aplicar en el caso de mongo . en una pequeña transaccion

```
 public static void main(String[] args) {
        try (MongoClient client =
MongoClients.create("mongodb://localhost:27017")) {
            MongoDatabase database = client.getDatabase("test");
            MongoCollection<Document> collection =
database.getCollection("example");

            ClientSession session = client.startSession();

            try {
                session.startTransaction();

                collection.insertOne(session, new Document("name",
"Alice"));
                collection.insertOne(session, new Document("name", "Bob"));

                session.commitTransaction();
                System.out.println("Transacción completada con éxito.");
            } catch (Exception e) {
                session.abortTransaction();
                System.err.println("Error en la transacción: " +
e.getMessage());
                e.printStackTrace();
            } finally {
                session.close();
            }
        }
    }
```

aqui podemos ver donde hacemos uso de **abortTransaction** y devolvermos el error en caso de que se devuelva un resultado fuera del esperado, garantizando así la integridad de los datos. En caso de fallos, se captura la excepción, se aborta la transacción y se reporta el error.

fuentes

Redactado y escrito por:

angelgabrielortega