

Laboratorio 2

Isabella Blanco Chaparro
Jairo Andrés sierra Combariza



Facultad de Ingeniería

Arquitectura de Software

PONTIFICIA UNIVERSIDAD JAVERIANA
BOGOTÁ D.C

Marco Conceptual

El propósito es implementar un servicio web que permita realizar operaciones CRUD (Crear, Leer, Actualizar, Borrar) sobre un modelo de datos relacionado con personas, profesiones, estudios y teléfonos. Se utilizará una arquitectura hexagonal que permita la flexibilidad y modularidad, haciendo que la aplicación sea independiente de las tecnologías externas.

1. **Arquitectura Hexagonal:** También conocida como Ports and Adapters (Puertos y Adaptadores), es un enfoque que promueve la separación de responsabilidades en el diseño de software. En esta arquitectura, el núcleo de la aplicación, que contiene la lógica de negocio, es independiente de los detalles de implementación y las tecnologías externas, como bases de datos o servicios de red. Esto facilita el cambio y reemplazo de dependencias externas sin afectar el núcleo de la aplicación.

- Componentes:

- Dominio (Núcleo de Negocio): Contiene las entidades y la lógica de negocio.
- Aplicación: Contiene los servicios que implementan las operaciones CRUD.
- Infraestructura: Interfaz con la base de datos (MariaDB, MongoDB) y otros servicios externos.
- Interfaz (Adaptadores): Controladores REST que exponen los servicios como endpoints HTTP.

2. **Tecnologías usadas:**

- **JDK 11:** Entorno de ejecución de Java.
- **Spring Boot:** Framework para el desarrollo de aplicaciones Java que facilita la configuración de aplicaciones web.
- **MariaDB:** Base de datos relacional utilizada para almacenar los datos en tablas relacionales.
- **MongoDB:** Base de datos NoSQL utilizada para almacenamiento en caso de que sea necesario expandir el modelo a otro tipo de persistencia.
- **REST:** Estilo arquitectónico utilizado para construir APIs HTTP que permite el intercambio de datos en formato JSON.
- **Swagger 3:** Herramienta para la documentación de APIs, que permite generar y probar los endpoints creados.

3. **CRUD en Arquitectura Hexagonal:**

- Métodos CRUD para cada entidad:
 - Crear

- Leer
- Actualizar
- Eliminar
- Los servicios son independientes de la implementación de la base de datos.
- Repositorios (Infrastructure Layer):
 - Implementación de acceso a datos mediante Spring Data JPA y Spring Data MongoDB.
 - Los repositorios actúan como puentes entre la capa de aplicación y las bases de datos.
 - Patrón Repository: Permite centralizar el acceso a la base de datos para cada entidad.

4. Desarrollo de la API REST:

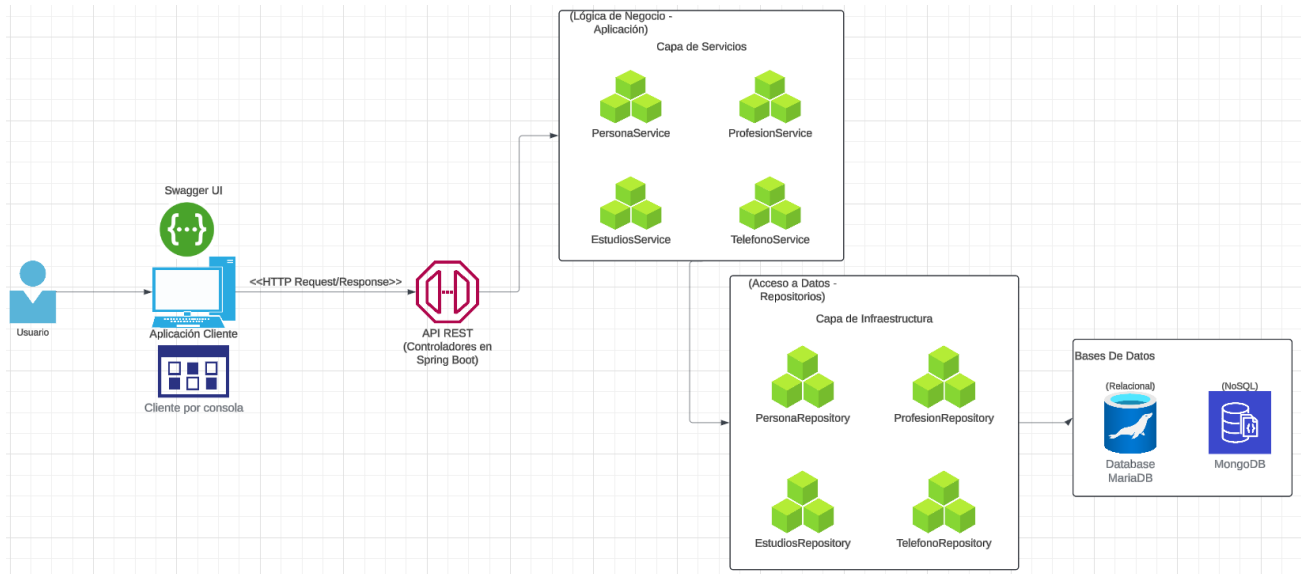
- Endpoints HTTP para cada entidad:
 - Persona: GET /personas, POST /personas, PUT /personas/{id}, DELETE /personas/{id}.
 - Profesión: CRUD similar para manejar profesiones.
 - Estudios: CRUD para relacionar personas y profesiones.
 - Teléfono: CRUD para teléfonos.
- ResponseEntity: Manejo de respuestas HTTP en cada endpoint.
- Validación de Datos: Validar los datos de entrada usando anotaciones como @Valid y @NotNull.

5. Documentación con Swagger: Es necesario configurar Swagger 3 para que genere automáticamente la documentación de los endpoints automáticamente.

- Pruebas Interactivas de Endpoints:
 - Swagger permite probar cada endpoint sin necesidad de herramientas externas como Postman.
 - Incluye parámetros y ejemplos de respuestas para cada operación, validando el funcionamiento correcto de los endpoints.

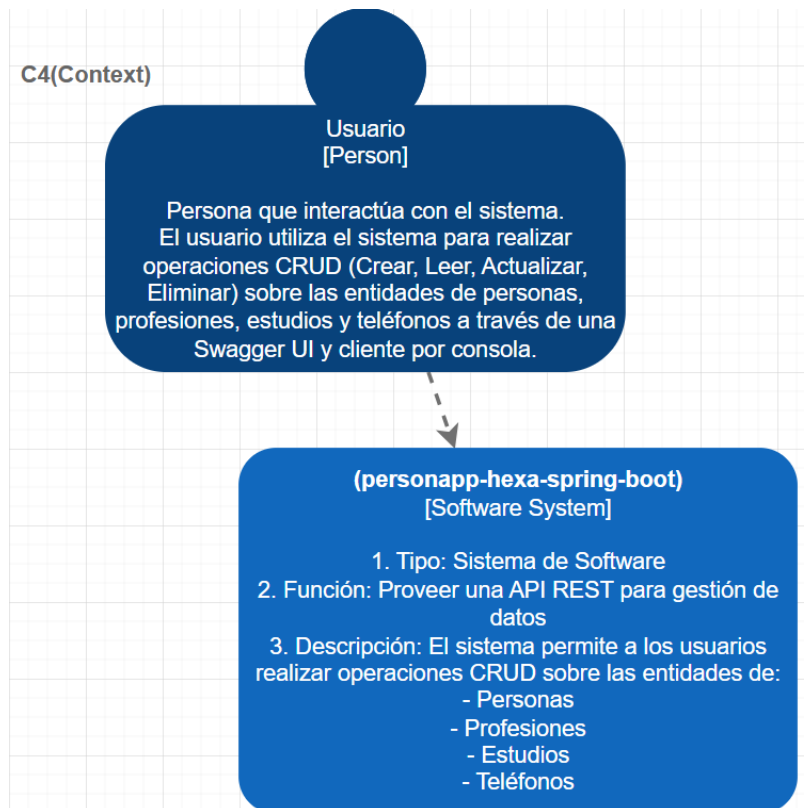
Diseño

1. **Diagrama de Arquitectura de Alto Nivel:** Se puede ver cómo cada componente se relaciona en una implementación de arquitectura hexagonal con Spring Boot y bases de datos relacionales y NoSQL.

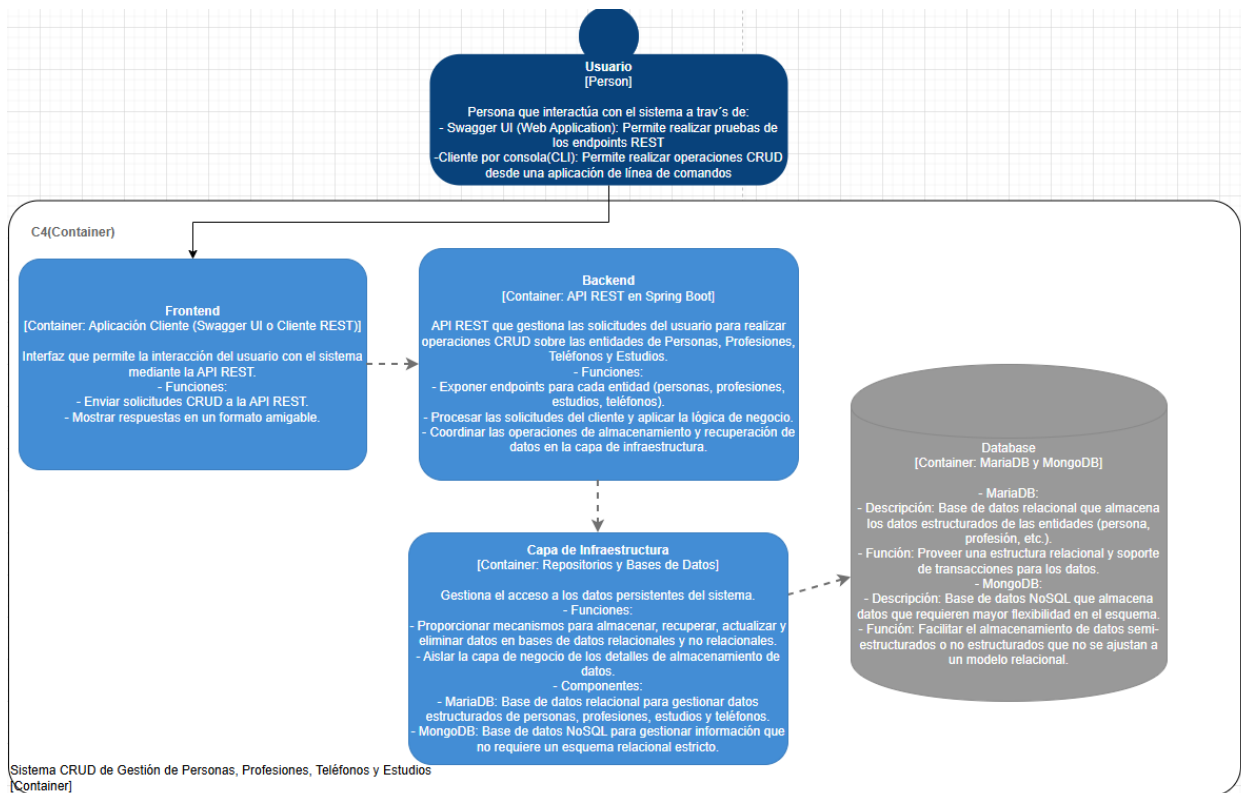


2. **C4 Model:** Representa la interacción usuario-sistema.

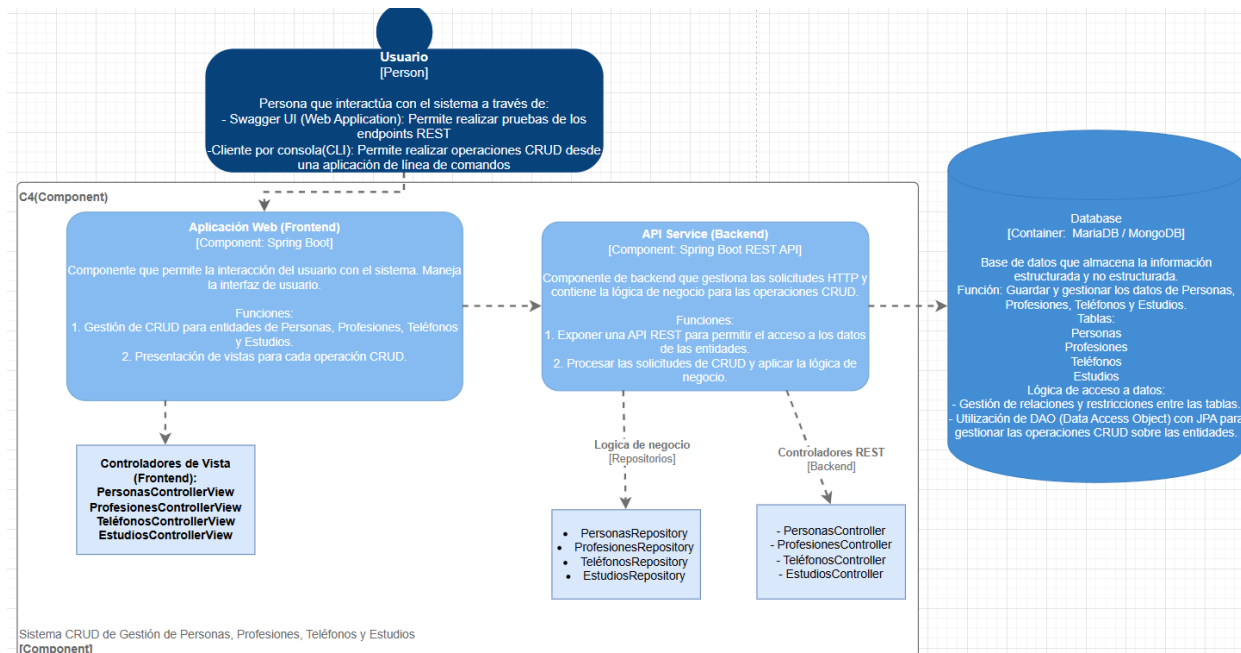
- a. **C4 Context:** El sistema está diseñado en arquitectura hexagonal y expone una API REST para facilitar la interacción con el cliente. Utiliza Spring Boot como framework principal, con MariaDB y MongoDB como sistemas de almacenamiento de datos.



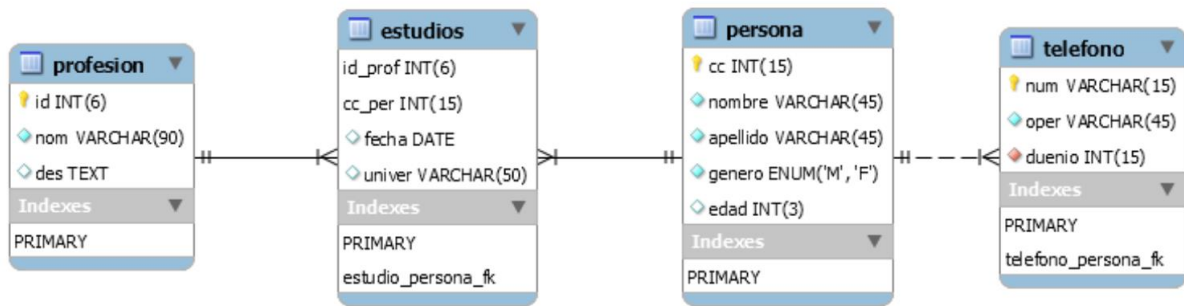
- b. **C4 Container:** Está basado en la arquitectura hexagonal implementada con Spring boot y el uso de MariaDB/MongoDB



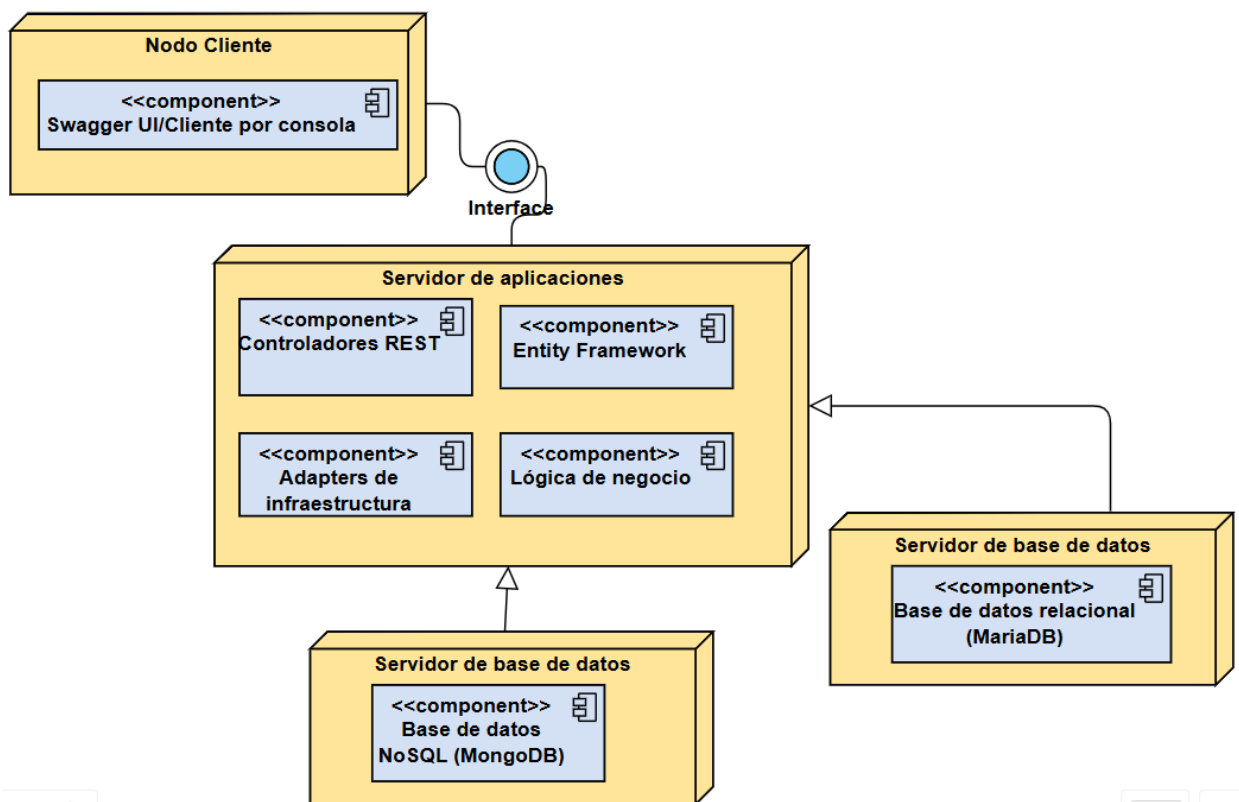
c. C4 Component:



d. C4 Class UML:



e. C4 Despliegue



Procedimiento

- Configuración del Entorno de Desarrollo:
 - Clonar el repositorio de plantilla proporcionada [personapp-hexa-spring-boot](https://github.com/andres-karoll/personapp-hexa-spring-boot).
 - Accede a la carpeta del proyecto con el siguiente comando:
 - `CD personapp-hexa-spring-boot`
 - Asegurarse de tener instalados JDK 11 o 17 ejecutando el siguiente comando:
 - `java -version`

```

● PS C:\Users\Acer\Documents\GitHub\personapp-hexa-spring-boot> java -version
java version "17.0.13" 2024-10-15 LTS
Java(TM) SE Runtime Environment (build 17.0.13+10-LTS-268)
Java HotSpot(TM) 64-Bit Server VM (build 17.0.13+10-LTS-268, mixed mode, sharing)
○ PS C:\Users\Acer\Documents\GitHub\personapp-hexa-spring-boot>

```

- Si no está instalado, descárgalo e instálalo
- Instalar MariaDB y MongoDB
- Configurar las propiedades de conexión a las bases de datos en el archivo 'src/main/resources/application.properties' de Spring Boot.

```

cli-input-adapter > src > main > resources > application.properties
1  # Logging config
2  logging.level.root=INFO
3  logging.file.name=logs/persona.log
4
5  # MariaDB Config
6  spring.datasource.url=jdbc:mariadb://mariadb:3306/persona_db
7  spring.datasource.username=persona_db
8  spring.datasource.password=persona_db
9  spring.datasource.driver-class-name=org.mariadb.jdbc.Driver
10 spring.jpa.hibernate.ddl-auto=update
11 spring.jpa.open-in-view=false
12 spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MariaDBDialect
13
14 # MongoDB Config
15 spring.data.mongodb.authentication-database=admin
16 spring.data.mongodb.username=persona_db
17 spring.data.mongodb.password=persona_db
18 spring.data.mongodb.database=persona_db
19 spring.data.mongodb.port=27017
20 spring.data.mongodb.host=mongodb
21

```

- Configura las propiedades de conexión de MariaDB y MongoDB. Para maria es la siguiente imagen:

Ajustes | Túnel SSH | Avanzado | SSL | Estadísticas

Tipo de red: MariaDB or MySQL (TCP/IP)

Librería: libmariadb.dll

Nombre del host / IP: 127.0.0.1

☐ Pedir credenciales
☐ Usar autenticación de Windows

Usuario: root

Contraseña: ••••

Puerto: 3307

☐ Protocolo cliente/servidor comprimido

Bases de datos: persona_db

Comentario:

Y para Mongo es la siguiente imagen:

MongoDB

Manage your connection settings

While connected, you may only personalize your connection's name, color or favorite status. To fully configure it, you must first disconnect. Beware that disconnecting might cause work in progress to be lost.

Disconnect

URI

mongodb://persona_db:*****@localhost:27017/?authSource=admin

Name

MongoDB

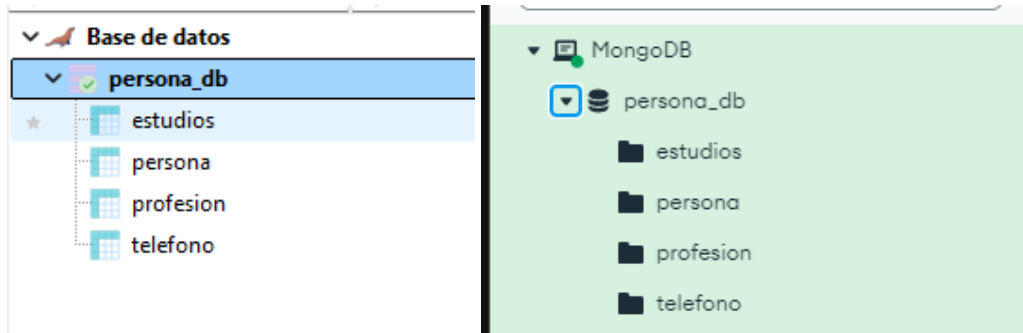
Color

Green

☐ Favorite this connection

- Guarda el archivo y asegúrate de que ambos servicios puedan conectarse.
- 2. Diseño de la Base de Datos:
 - Crear las tablas en MariaDB siguiendo el modelo ER (Entidad-Relación) y conectarse a la instancia MariaDB.

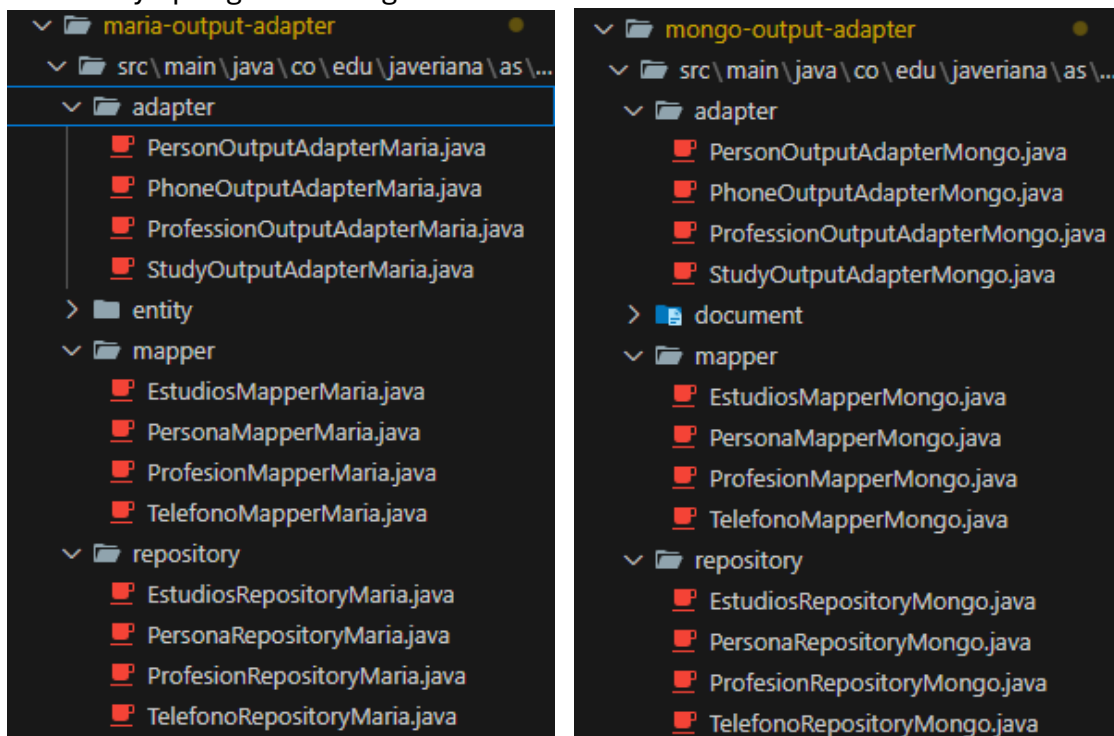
- Usar scripts DDL para crear las tablas necesarias: 'persona', 'profesion', 'estudios' y 'telefono'.



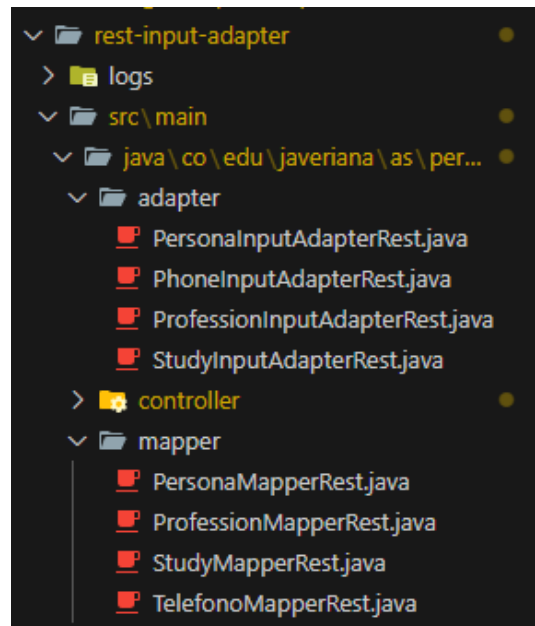
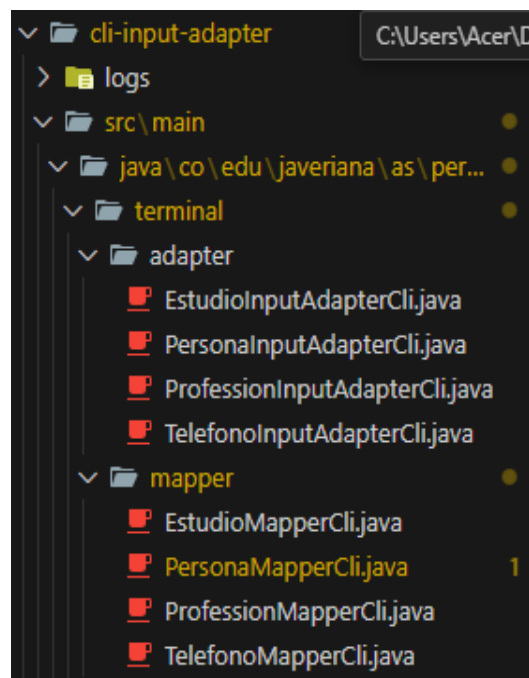
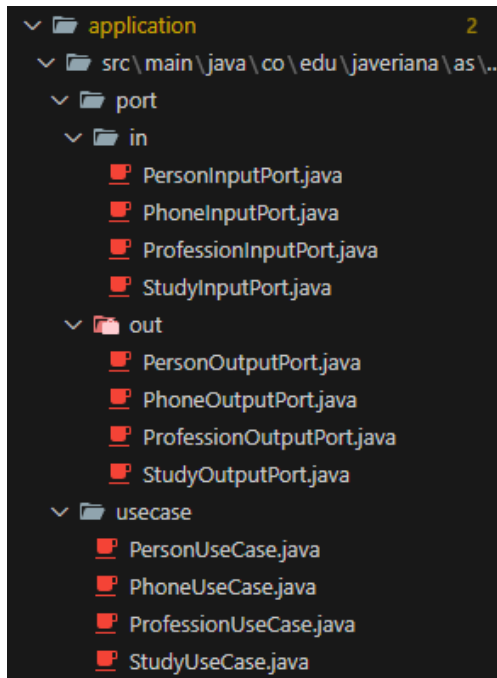
- Insertar datos de prueba utilizando scripts DML:
 - *INSERT INTO persona (nombre, apellido, fecha_nacimiento) VALUES ('Juan', 'Pérez', '1985-06-15');*
 - INSERT INTO profesion (descripcion) VALUES ('Ingeniero');*
 - INSERT INTO estudios (descripcion) VALUES ('Universitario');*
 - INSERT INTO telefono (numero, tipo) VALUES ('555-1234', 'Móvil');*

3. Implementación de Servicios:

- Implementar los repositorios para cada entidad usando Spring Data JPA para MariaDB y Spring Data MongoDB si es necesario.



- Implementar los servicios de aplicación que contienen la lógica de negocio y realizan las operaciones CRUD.



4. Exposición de Endpoints REST:

- Crear controladores REST para cada entidad que exponga los endpoints necesarios.
 - o Ejemplo de controlador para 'PersonaController':

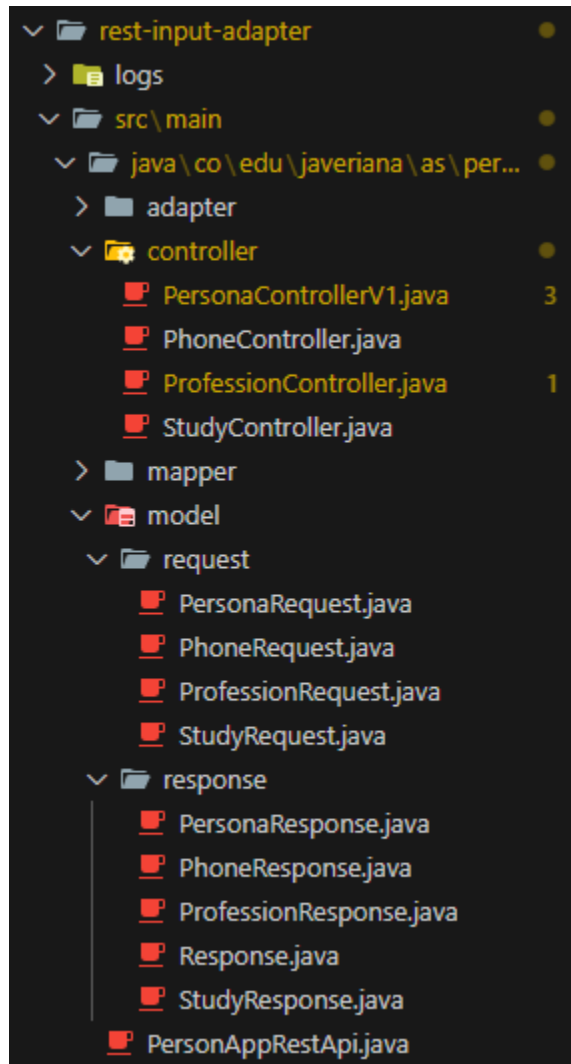

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;
@RestController
@RequestMapping("/api/personas")
public class PersonaController {
```

```
@Autowired
private PersonaService personaService;
```

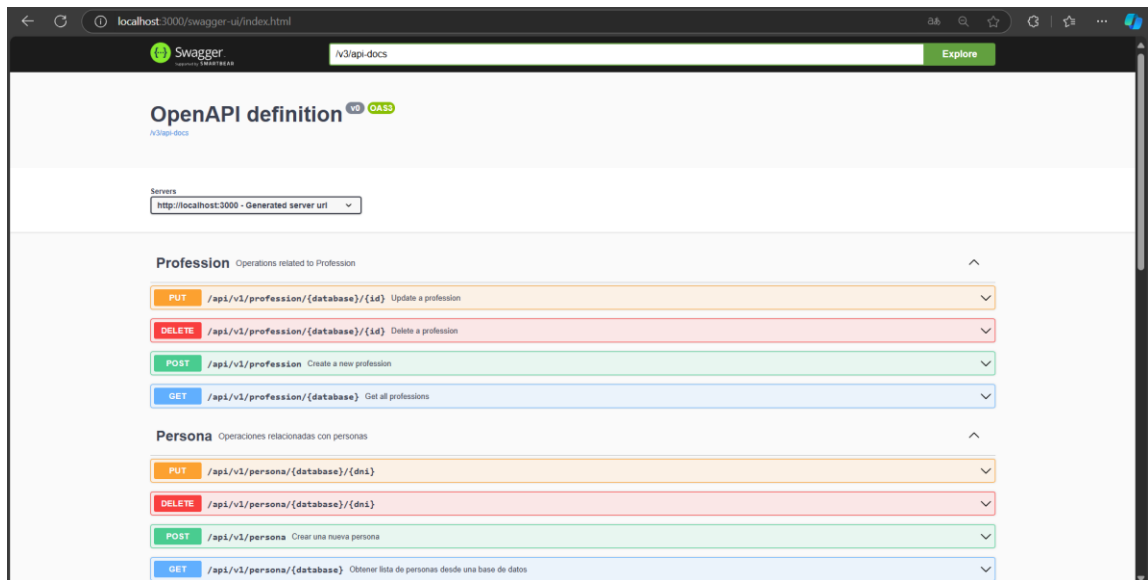
```
@GetMapping
public List<Persona> listarPersonas() {
    return personaService.listarTodas();
}
```

```
@PostMapping
public Persona crearPersona(@RequestBody Persona persona) {
    return personaService.guardarPersona(persona);
}
```

```
}
```

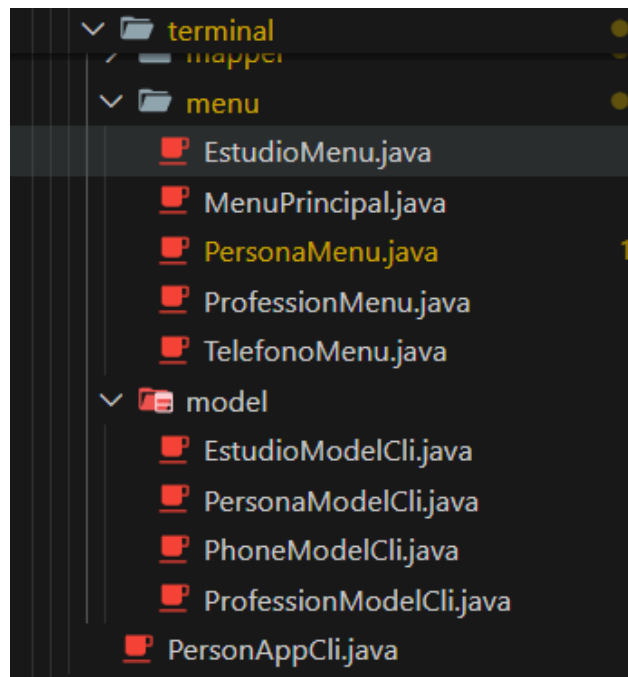


- Probar los endpoint CRUD utilizando el UI de Swagger para verificar su correcto funcionamiento en las respuestas y el estado HTTP de cada petición.



5. Implementación del cliente por consola:

- Crear una clase de cliente por consola: Agrega una clase Java en el proyecto que permita a los usuarios interactuar con el sistema mediante la consola. - Esta clase debe permitir realizar operaciones CRUD básicas en cada entidad (persona, profesion, estudios, y telefono).
- Configura la clase de consola para que se ejecute al iniciar la aplicación. Esto se hace en el método de ejecución por consola de la aplicación Spring Boot, permitiendo que el usuario elija si desea iniciar el cliente por consola o la aplicación web.



- Probar el cliente por consola

6. Implementación de Docker:

- Crear un archivo Dockerfile en los componentes cli y rest para contener la configuración de la imagen Docker.

```
cli-input-adapter > Dockerfile > ...
1  # Usa Maven con OpenJDK 11
2  FROM maven:3.8.1-openjdk-11-slim AS build
3
4  # Establece el directorio de trabajo en /app
5  WORKDIR /app
6
7  # Copia el archivo de configuración de Maven y el archivo pom.xml del proyecto raíz
8  COPY ./pom.xml .
9
10 # Copia la estructura completa del proyecto al contenedor
11 COPY . .
12
13 # Descarga dependencias y construye todo el proyecto
14 RUN mvn clean install -DskipTests
15
16 # Cambia al módulo `cli-input-adapter` y ejecuta la aplicación
17 WORKDIR /app/cli-input-adapter
18
19 CMD ["mvn", "spring-boot:run"]
20
```

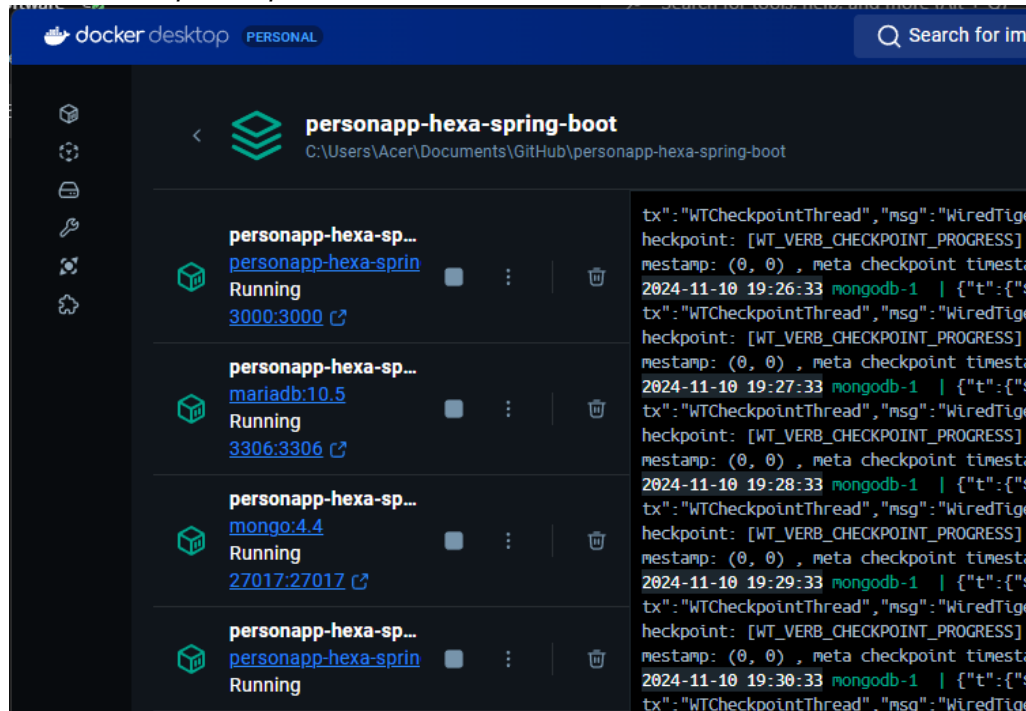
- Construir la imagen Docker en la raíz del proyecto con el Docker-compose.yml, esto generará una imagen llamada personapp-hexa-spring, que incluye todos los componentes necesarios para ejecutar la aplicación:

docker build -t personapp-hexa-spring .

```
docker-compose.yml
3  services:
23    rest:
29      environment:
32        SPRING_DATASOURCE_PASSWORD: persona_db
33      depends_on:
34        - mariadb
35        - mongodb
36      networks:
37        - personapp-network
38
39    cli:
40      build:
41        context: . # Contexto en el directorio raíz del proyecto
42        dockerfile: ./cli-input-adapter/Dockerfile
43      depends_on:
44        - mariadb
45        - mongodb
46      networks:
47        - personapp-network
48
49    networks:
50      personapp-network:
51        driver: bridge
52
```

- Levantamiento de servicios con docker-compose, se iniciará tanto la aplicación Spring Boot como las bases de datos MongoDB y MariaDB en contenedores separados pero conectados.

docker-compose up -d

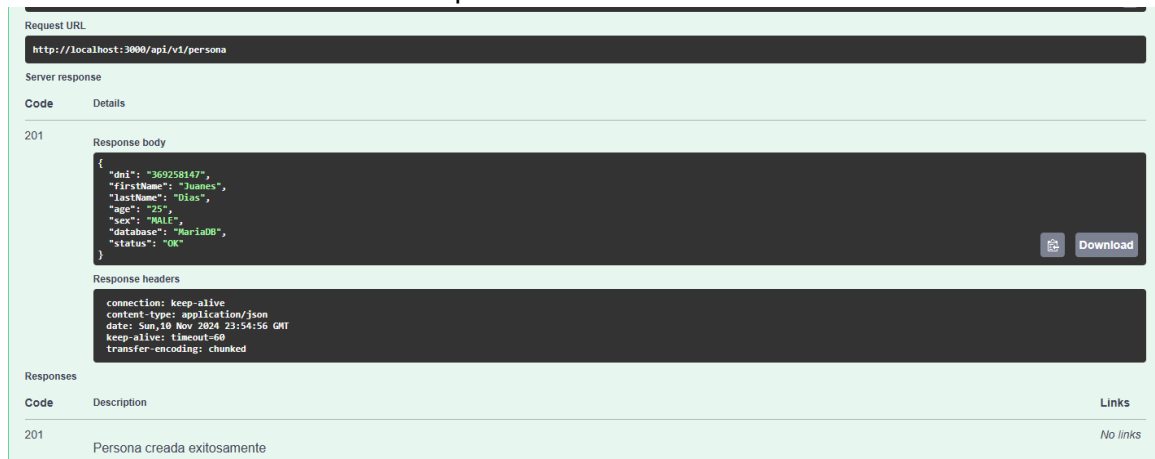


7. Documentación y Pruebas:

- Documentar el proyecto en el README.md con pasos detallados para la configuración y despliegue.
- Validar el funcionamiento correcto mediante pruebas de endpoints en Swagger, asegúrate de que las operaciones CRUD se realicen correctamente en ambas bases de datos.

Rest APIS por swagger

- Creación de persona en rest maria



- Get de persona en rest maria

```
Request URL
http://localhost:3000/api/v1/persona/maria

Server response
Code  Details
200
Response body
{
  "dni": "123456789",
  "firstName": "Juliana",
  "lastName": "Dias",
  "age": "19",
  "sex": "FEMALE",
  "database": "MariaDB",
  "status": "OK"
},
{
  "dni": "369258147",
  "firstName": "Juaimes",
  "lastName": "Dias",
  "age": "25",
  "sex": "MALE",
  "database": "MariaDB",
  "status": "OK"
}
Response headers
connection: keep-alive
content-type: application/json
date: Sun,10 Nov 2024 23:58:28 GMT
keep-alive: timeout=60
transfer-encoding: chunked
```

- Put (actualizar) persona en rest maria

```
Request URL
http://localhost:3000/api/v1/persona/maria/123456789

Server response
Code  Details
200
Response body
{
  "dni": "123456789",
  "firstName": "Juliana",
  "lastName": "Menes",
  "age": "18",
  "sex": "FEMALE",
  "database": "MariaDB",
  "status": "OK"
}
Response headers
connection: keep-alive
content-type: application/json
date: Mon,11 Nov 2024 00:00:38 GMT
keep-alive: timeout=60
transfer-encoding: chunked
```

- Delete persona en rest maria

```
Curl
curl -X 'DELETE' \
  'https://localhost:3000/api/v1/persona/maria/123456789' \
  -H 'accept: */*'

Request URL
http://localhost:3000/api/v1/persona/maria/123456789

Server response
Code  Details
204
Undocumented
Response headers
connection: keep-alive
date: Mon,11 Nov 2024 00:03:02 GMT
keep-alive: timeout=60
```

Estos procedimientos de las operaciones CRUD mostrados para la entidad Persona se repiten de la misma manera para las entidades Estudio, Profesión y Teléfono. Los pasos son idénticos en estructura y funcionalidad.

Cliente **por** **Consola:**
Para esta parte se repiten las mismas entidades y CRUD desplegándose en menus por consola para cada entidad y seleccionando alguna de las bases de datos para hacer la operación.

- Creación de persona en mongo

```
2024-11-10 23:51:02.933 INFO 56 --- [-mongodb:27017] org.mongodb.driver.cluster : Monitor thread successfully connected to server with description ServerDescription{
address=mongodb:27017, type=STANDALONE, state=CONNECTED, ok=true, minWireVersion=0, maxWireVersion=9, maxDocumentSize=16777216, logicalSessionTimeoutMinutes=30, roundTripTimeNanos=
er
-----
1 para trabajar con el Modulo de Personas
2 para trabajar con el Modulo de Profesiones
3 para trabajar con el Modulo de Telefonos
4 para trabajar con el Modulo de Estudios
0 para Salir
Ingrese una opción: 1
-----
1 para MariaDB
2 para MongoDB
0 para regresar
Ingrese una opción: 2
-----
1 para ver todas las personas
2 para crear una nueva persona
3 para actualizar una persona existente
4 para eliminar una persona
0 para regresar
Ingrese una opción: 2
Ingrese la cédula de la persona: 987654321
Ingrese el nombre de la persona: Juana
Ingrese el apellido de la persona: Mendez
Ingrese el género de la persona (M/F): F
Ingrese la edad de la persona: 19
2024-11-11 00:14:58.317 INFO 56 --- [main] c.e.j.a.p.t.a.PersonaInputAdapterCli : Creating person in CLI Adapter
2024-11-11 00:14:58.570 INFO 56 --- [main] org.mongodb.driver.connection : Opened connection [connectionId{localValue:3, serverValue:7}] to mongodb:27017
Persona creada: Person(identification=987654321, firstName=Juana, lastName=Mendez, gender=FEMALE, age=19)
-----
```

- Ver todas las personas en mongo

```
-----
1 para ver todas las personas
2 para crear una nueva persona
3 para actualizar una persona existente
4 para eliminar una persona
0 para regresar
Ingrese una opción: 1
2024-11-11 00:16:58.460 INFO 56 --- [main] c.e.j.a.p.t.a.PersonaInputAdapterCli : Into historial PersonaEntity in Input Adapter
2024-11-11 00:16:58.462 INFO 56 --- [main] c.e.j.a.p.a.usecase.PersonUseCase : Output: class co.edu.javeriana.as.personapp.mongo.adapter.PersonOutputAdapterMongo
PersonaModelCli(cc=987654321, nombre=Juana, apellido=Mendez, genero=FEMALE, edad=19)
PersonaModelCli(cc=123456789, nombre=Andres, apellido=Perez, genero=MALE, edad=20)
-----
```

- Actualizar persona en mongo

```
-----
1 para ver todas las personas
2 para crear una nueva persona
3 para actualizar una persona existente
4 para eliminar una persona
0 para regresar
Ingrese una opción: 3
Ingrese la cédula de la persona a actualizar: 123456789
Ingrese el nuevo nombre de la persona: Juan
Ingrese el nuevo apellido de la persona: Perez
Ingrese el nuevo género de la persona (M/F): M
Ingrese la nueva edad de la persona: 30
2024-11-11 00:19:22.943 INFO 56 --- [main] c.e.j.a.p.t.a.PersonaInputAdapterCli : Updating person in CLI Adapter
Person updated: Person(identification=123456789, firstName=Juan, lastName=Perez, gender=MALE, age=30)
-----
```

- Eliminación de persona en mongo


```

-----
1 para ver todas las personas
2 para crear una nueva persona
3 para actualizar una persona existente
4 para eliminar una persona
0 para regresar
Ingrese una opción: 4
Ingrese la cédula de la persona a eliminar: 123456789
2024-11-11 00:20:11.183 INFO 56 --- [main] c.e.j.a.p.t.a.PersonaInputAdapterCli : Deleting person in CLI Adapter
Person deleted successfully.
-----
1 para ver todas las personas
2 para crear una nueva persona
3 para actualizar una persona existente
4 para eliminar una persona
0 para regresar
Ingrese una opción: 1
2024-11-11 00:20:19.197 INFO 56 --- [main] c.e.j.a.p.t.a.PersonaInputAdapterCli : Into historial PersonaEntity in Input Adapter
2024-11-11 00:20:19.198 INFO 56 --- [main] c.e.j.a.p.a.usecase.PersonUseCase : Output: class co.edu.javeriana.as.personapp.mongo.adapter.PersonOutputAdapterMongo
PersonaModelCli(cc=987654321, nombre=Juana, apellido=Mendez, genero=FEMALE, edad=19)
-----

```

Estos procedimientos de las operaciones CRUD mostrados para la entidad Persona se repiten de la misma manera para las entidades Estudio, Profesión y Teléfono. Los pasos son idénticos en estructura y funcionalidad.

8. Versionamiento y creación de TAG:

- Hacer push de todo el código al repositorio remoto en GitHub.
- Crear un TAG en el repositorio con la versión final del laboratorio.

Conclusiones y Lecciones Aprendidas

- **Modularidad:** La arquitectura hexagonal permitió una clara separación entre la lógica de negocio y los detalles de implementación. Esto facilita el mantenimiento y mejora la extensibilidad del código.
- **Independencia de la base de datos:** La estructura de repositorios en la capa de infraestructura hace que sea fácil cambiar de una base de datos a otra sin modificar el núcleo de la aplicación.
- **Facilidad de documentación con swagger:** Swagger 3 proporcionó una interfaz clara y fácil de usar para probar los endpoints, lo cual facilitó la validación de los servicios REST. Aprendizaje sobre Hexagonal Architecture: Implementar esta arquitectura nos ayudó a entender cómo construir aplicaciones más desacopladas y escalables, facilitando el reemplazo de tecnologías específicas en el futuro.

Referencias:

- <https://spring.io/projects/spring-boot>
- <https://spring.io/projects/spring-data-jpa>
- <https://swagger.io/docs/>
- <https://mariadb.com/kb/en/documentation/>
- <https://docs.mongodb.com/>