PONTIFICIA UNIVERSIDAD JAVERIANA
FACULTAD DE INGENIERÍA

REPORTE DE LABORATORIO JAVA HEXAGONAL ARQUITECTURA DE SOFTWARE

17/11/2025

Juan Diego Romero

Kamilt Bejarano Diaz

Pablo Enrique Quintero

Arquitectura de Software, Universidad Javeriana

Laboratorio de Java Hexagonal Noviembre 2025

Pablo Enrique Quintero, Juan Diego Romero, Kamilt Bejarano Diaz

Tabla de contenido

Arquitectura de Software, Universidad Javeriana

Laboratorio de Java Hexagonal Noviembre 2025

Pablo Enrique Quintero, Juan Diego Romero, Kamilt Bejarano Diaz

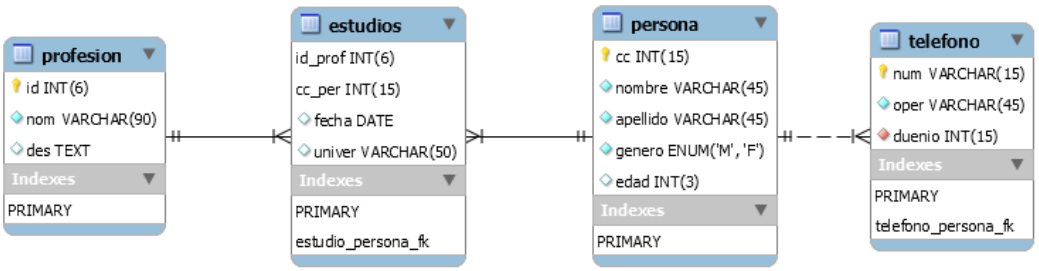
Marco Conceptual

El presente taller busca desarrollar una aplicación de terminal (CLI) en Java utilizando una arquitectura Hexagonal con conexión a dos bases de datos separadas, Mongo y Maria, al igual que autodocumentación por medio de Swagger. El taller toma como base un repositorio existente con implementación para las funcionalidades de una de las entidades.

Diseño

El diseño del sistema se basó en una arquitectura hexagonal. El objetivo fue mantener una estructura altamente modular, permitiendo agregar y eliminar componentes sin afectar otros.

En primer lugar, se diseñó la base de datos con cuatro entidades principales siguiendo el modelo dado en la rúbrica del taller:



Cada una fue implementada con su respectivo repositorio y adaptadores para operar en ambas bases de datos.

La organización general del proyecto incluyó:

Dominio: Entidades, lógica de negocio y puertos.

Aplicación: Servicios que coordinan la interacción entre dominio y adaptadores.

Infraestructura: Adaptadores para CLI, MongoDB y Swagger.

Funcionamiento del sistema

La aplicación es ejecutada desde la terminal, el usuario tiene acceso a un menú donde puede seleccionar qué entidad quiere manipular, posteriormente debe seleccionar la base de datos (Mongo o Maria), y finalmente selecciona qué operación CRUD desea realizar sobre la entidad.

Procedimiento

El desarrollo del laboratorio se llevó a cabo siguiendo las siguientes etapas:

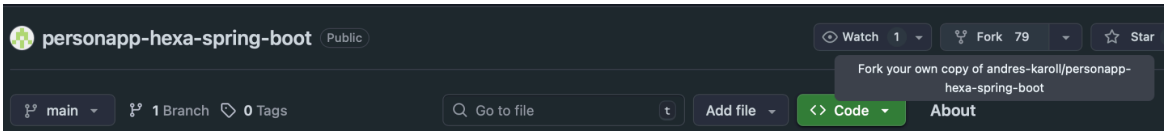
Arquitectura de Software, Universidad Javeriana

Laboratorio de Java Hexagonal Noviembre 2025

Pablo Enrique Quintero, Juan Diego Romero, Kamilt Bejarano Diaz

Fork del repositorio base

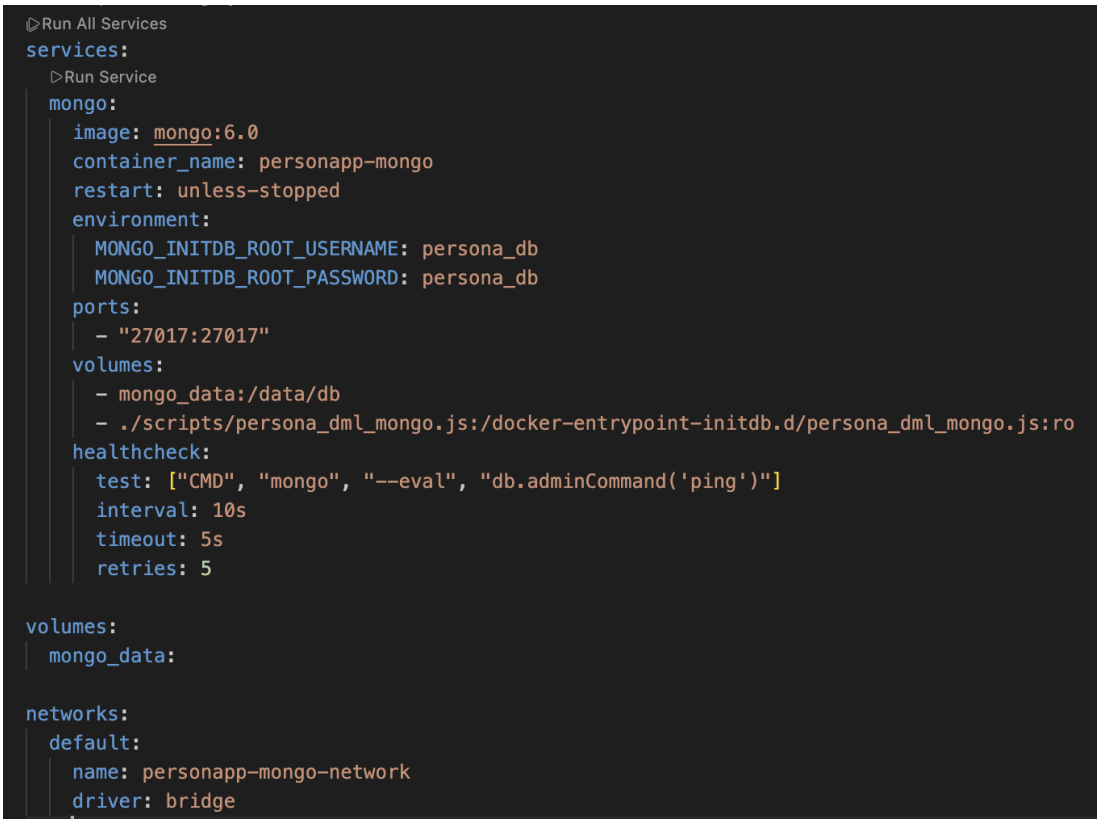
Se realizó un fork del repositorio proporcionado por el profesor para contar con una plantilla estructurada bajo arquitectura Hexagonal.



Configuración del entorno

Se instalaron y configuraron MariaDB en el puerto 3307 y MongoDB en el puerto 27017 mediante Docker, siguiendo las especificaciones indicadas.

```
Run All Services
services:
  Run Service
  mariadb:
    image: mariadb:10.5
    container_name: personapp-mariadb
    restart: unless-stopped
    environment:
      MYSQL_ROOT_PASSWORD: rootpass
      MYSQL_DATABASE: persona_db
      MYSQL_USER: persona_db
      MYSQL_PASSWORD: persona_db
    ports:
      - "3307:3306"
    volumes:
      - mariadb_data:/var/lib/mysql
      - ./scripts/persona_ddl_maria.sql:/docker-entrypoint-initdb.d/persona_ddl_maria.sql:ro
      - ./scripts/persona_dml_maria.sql:/docker-entrypoint-initdb.d/persona_dml_maria.sql:ro
    healthcheck:
      test: ["CMD", "mysqladmin", "ping", "-h", "127.0.0.1"]
      interval: 10s
      timeout: 5s
      retries: 5

volumes:
  mariadb_data:

networks:
  default:
    name: personapp-maria-network
    driver: bridge
```



```
Run All Services
services:
  Run Service
  mongo:
    image: mongo:6.0
    container_name: personapp-mongo
    restart: unless-stopped
    environment:
      MONGO_INITDB_ROOT_USERNAME: persona_db
      MONGO_INITDB_ROOT_PASSWORD: persona_db
    ports:
      - "27017:27017"
    volumes:
      - mongo_data:/data/db
      - ./scripts/persona_dml_mongo.js:/docker-entrypoint-initdb.d/persona_dml_mongo.js:ro
    healthcheck:
      test: ["CMD", "mongo", "--eval", "db.adminCommand('ping')"]
      interval: 10s
      timeout: 5s
      retries: 5

volumes:
  mongo_data:

networks:
  default:
    name: personapp-mongo-network
    driver: bridge
```

Docker Compose para ambas bases de datos

Al ejecutar estos Dockers los scripts dml y ddl crean las bases de datos de forma automática, tanto para Maria como para Mongo.

Implementación del dominio

Se completaron las entidades, mappers, adaptadores, repositorios y puertos faltantes según el modelo de datos al igual que implementar Lombok, utilizando la implementación existente de la entidad Persona como base. Se realizaron en total 3 implementaciones: para la terminal y una para cada base de datos.

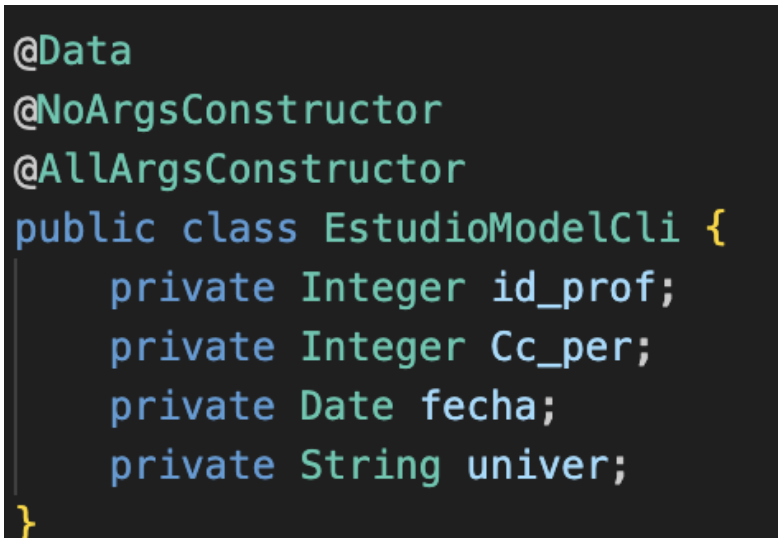
Los mappers son los elementos responsables de “traducir” la información ingresada para que su respectivo componente pueda entenderla

```java
@Data
@NoArgsConstructor
@AllArgsConstructor
@EqualsAndHashCode(onlyExplicitlyIncluded = true)
@ToString(exclude = {"persona", "profesion"})
public class EstudiosEntity implements Serializable {

    private static final long serialVersionUID = 1L;
    @EmbeddedId
    protected EstudiosEntityPK estudiosEntityPK;
    @Temporal(TemporalType.DATE)
    private Date fecha;
    @Column(length = 50)
    private String univer;
    @JoinColumn(name = "cc_per", referencedColumnName = "cc", nullable = false, insertable = false, updatable = false)
    @ManyToOne(optional = false)
    private PersonaEntity persona;
    @JoinColumn(name = "id_prof", referencedColumnName = "id", nullable = false, insertable = false, updatable = false)
    @ManyToOne(optional = false)
    private ProfesionEntity profesion;

    public EstudiosEntityPK getEstudiosPK() {
        return estudiosEntityPK;
    }

    public void setEstudiosPK(EstudiosEntityPK estudiosEntityPK) {
        this.estudiosEntityPK = estudiosEntityPK;
    }
}
```

Ejemplo de entidad para Maria

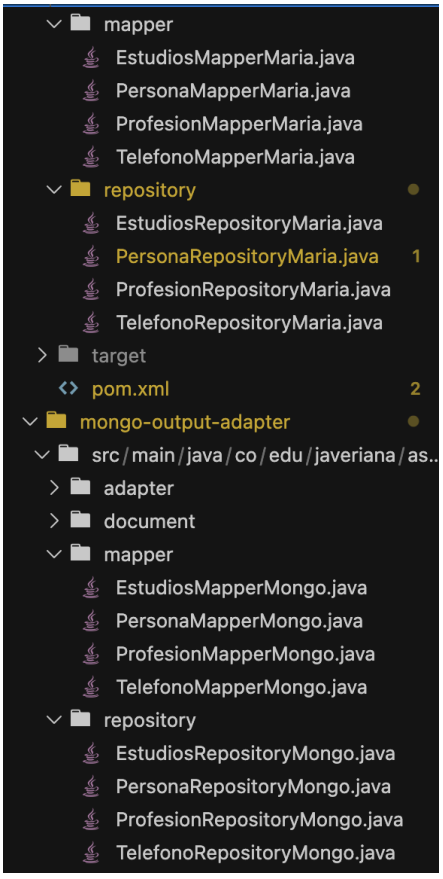
```java
@Data
@AllArgsConstructor
@NoArgsConstructor
@Document("profesion")
public class ProfesionDocument {
    @Id
    private Integer id;
    private String nom;
    private String des;
    @DocumentReference(lazy = true, lookup = "{ 'primaryProfesion' : ?#{#self._id} }")
    @ReadOnlyProperty
    private List<EstudiosDocument> estudios;
}
```

Ejemplo de entidad para Mongo

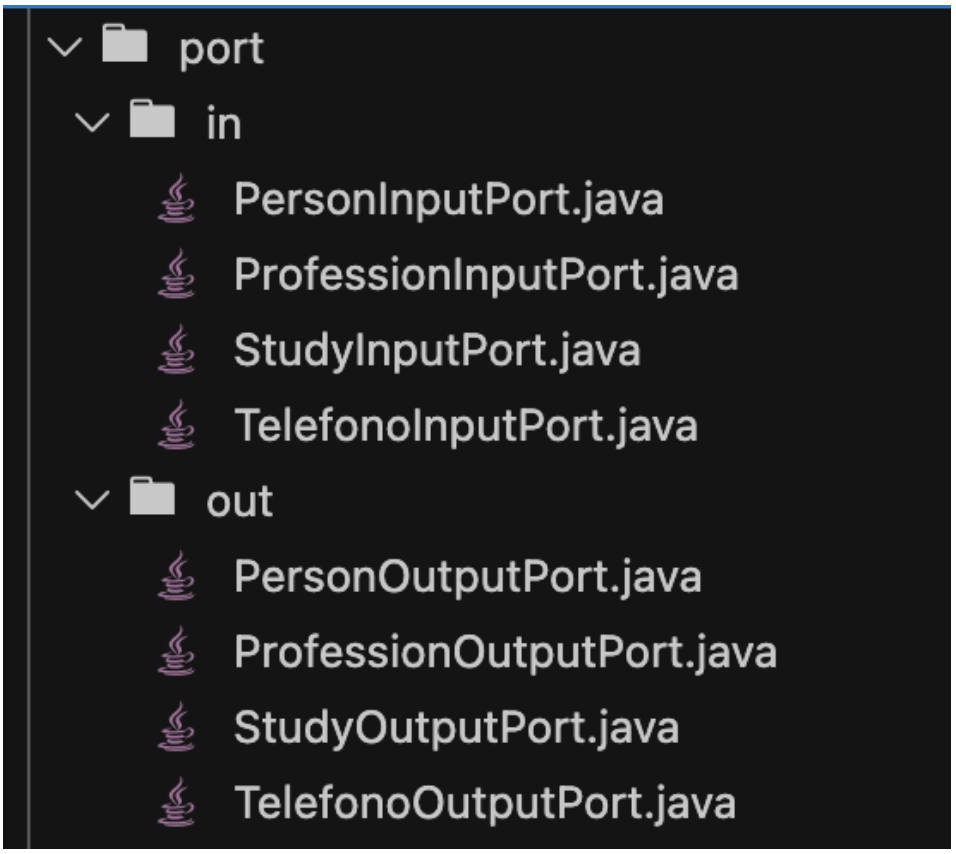


```java
@Data
@NoArgsConstructor
@AllArgsConstructor
public class EstudioModelCli {
    private Integer id_prof;
    private Integer Cc_per;
    private Date fecha;
    private String univer;
}
```

Ejemplo de modelo para la terminal



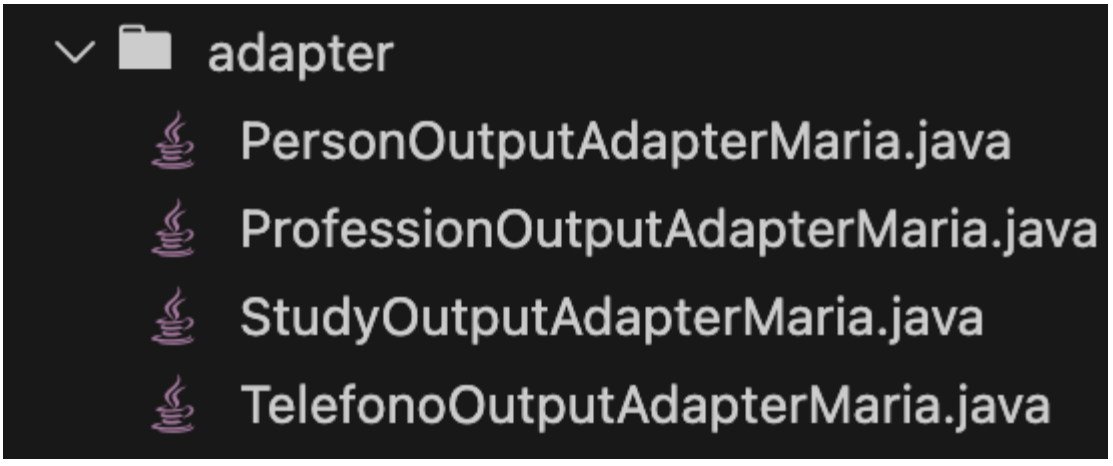
Mappers y Repositorios para ambas bases de datos

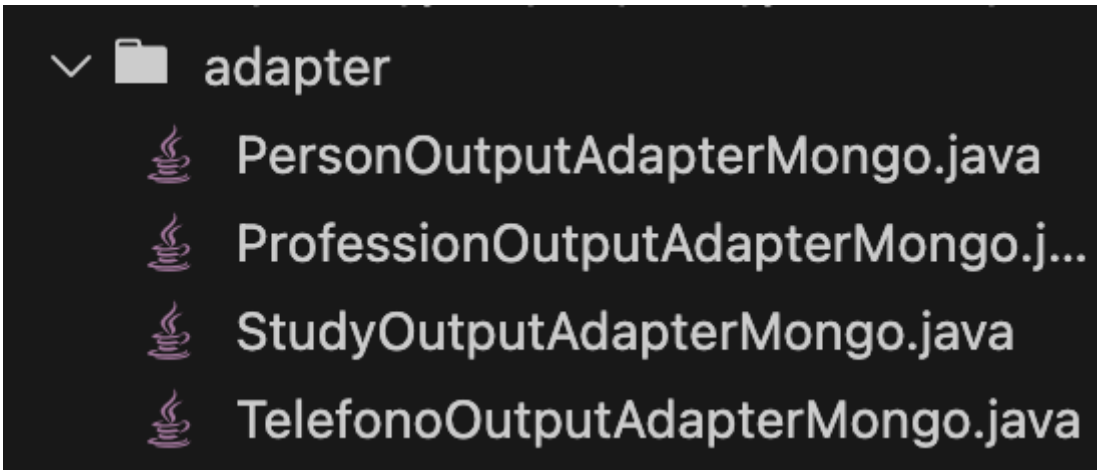


Puertos para las entidades

Desarrollo de los adaptadores

Se implementaron los servicios necesarios para soportar las operaciones CRUD tanto en MariaDB como en MongoDB.

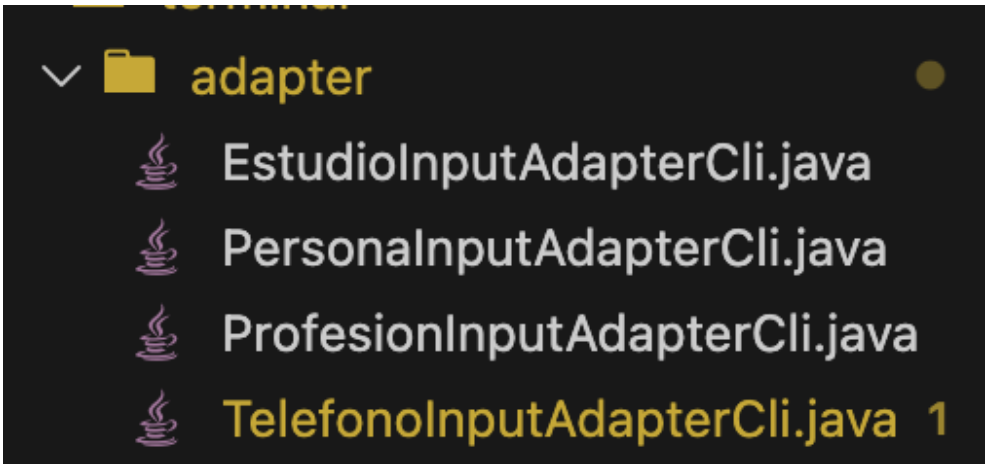




Adapters de ambas bases de datos

Integración del CLI y del API REST

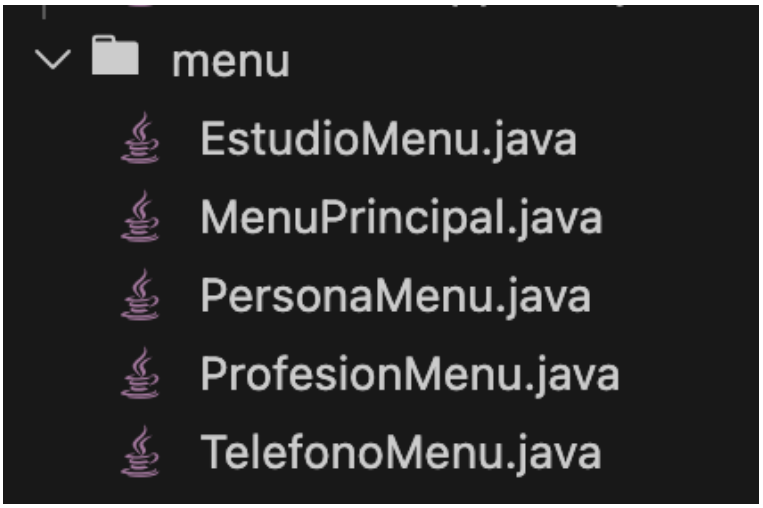
Se configuraron ambos adaptadores de entrada, creando dos aplicaciones Spring Boot independiente, al igual que menús de terminal para cada entidad



Adaptadores CLI

```java
public void buscar(Integer id) {
    log.info(msg: "Into buscar ProfesionEntity in Input Adapter");
    Profession profesion;
    try {
        profesion = professionInputPort.findOne(id);
        System.out.println(profesion);
    } catch (NoExistException e) {
        log.info(msg: "Error: profesion no encontrada");
        System.out.println("Error: La profesion con id " + id + " no existe");
        e.printStackTrace();
    }
}

public Profession editar(Integer id, Profession data) {
    log.info(msg: "Into editar ProfesionEntity in Input Adapter");
    Profession editado;
    try {
        editado = professionInputPort.edit(id, data);
    } catch (NoExistException e) {
        System.out.println(x: "Error: la profesion no existe");
        editado = null;
        e.printStackTrace();
    }
    return editado;
}
```

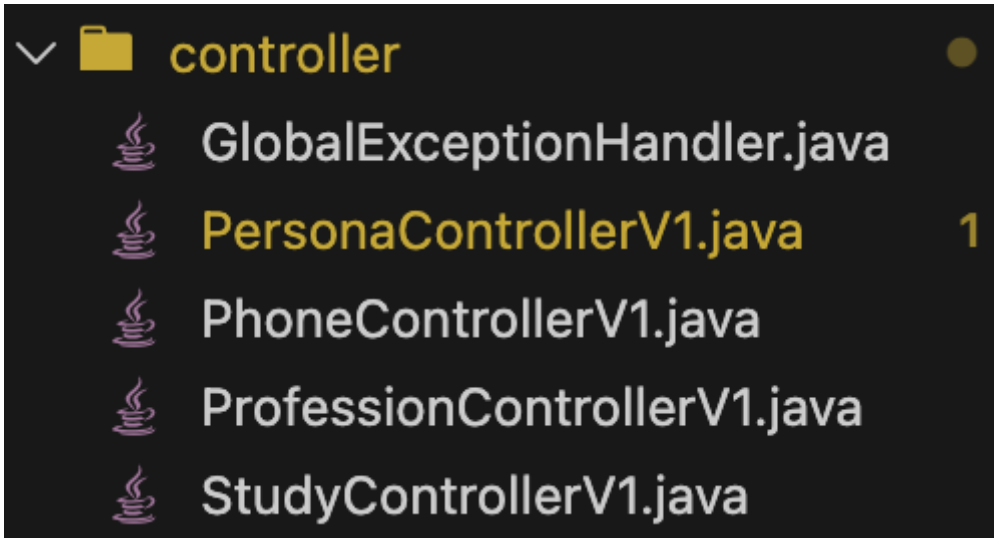
Ejemplo de adaptador



Menús de la aplicación

Configuración de Swagger

Se habilitó la documentación automática en el puerto 3000 disponible en la ruta /swagger-ui.html.



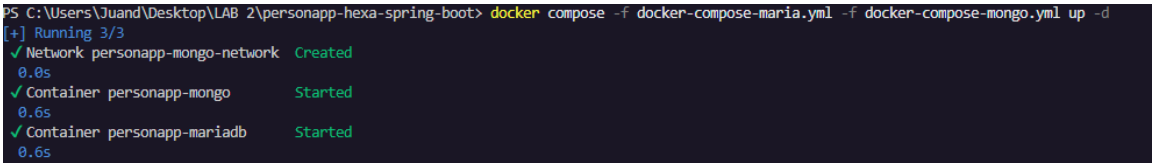
Archivos de configuración de cada entidad para Swagger

Pruebas

Se realizaron pruebas en los dos adaptadores (CLI y REST) para verificar el correcto funcionamiento con ambas bases de datos, asegurando que no hayan problemas en ninguna de las operaciones.

Ejecución del sistema

1. En la terminal, navegar a la carpeta del proyecto con *cd [path]/personapi-hexa-spring-boot*
2. Levantar los docker compose de las bases de datos con *docker compose -f docker-compose-maria.yml up -d* y *docker compose -f docker-compose-mongo.yml up -d*

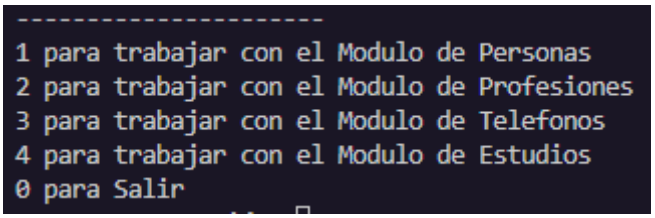


   Importante, tener la aplicación de docker corriendo y asegurarse que los contenedores se levantan, de otro modo el paso 3 no se puede hacer.

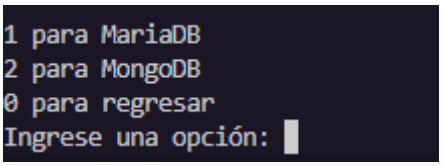
3. Ejecutar la aplicación de **terminal** por medio del siguiente comando, dependiendo del sistema operativo:

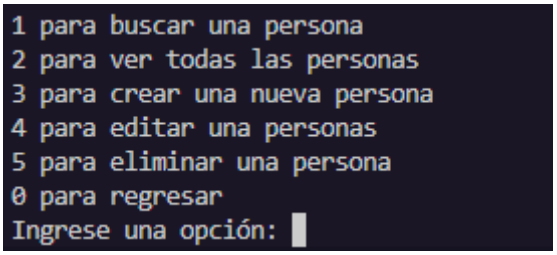
- java @[path].argfile co.edu.javeriana.as.personapp.PersonAppCli (**Windows**)

- */usr/bin/env /Library/Java/JavaVirtualMachines/amazon-corretto-11.jdk/Contents/Home/bin/java @/var/folders/h9/p5vdxgp90y3755thc80tspqh0000gn/T/cp_js8sr5rcwlcpwjkb54awnxmb.argfile co.edu.javeriana.as.personapp.PersonAppCli*(**MacOS**)

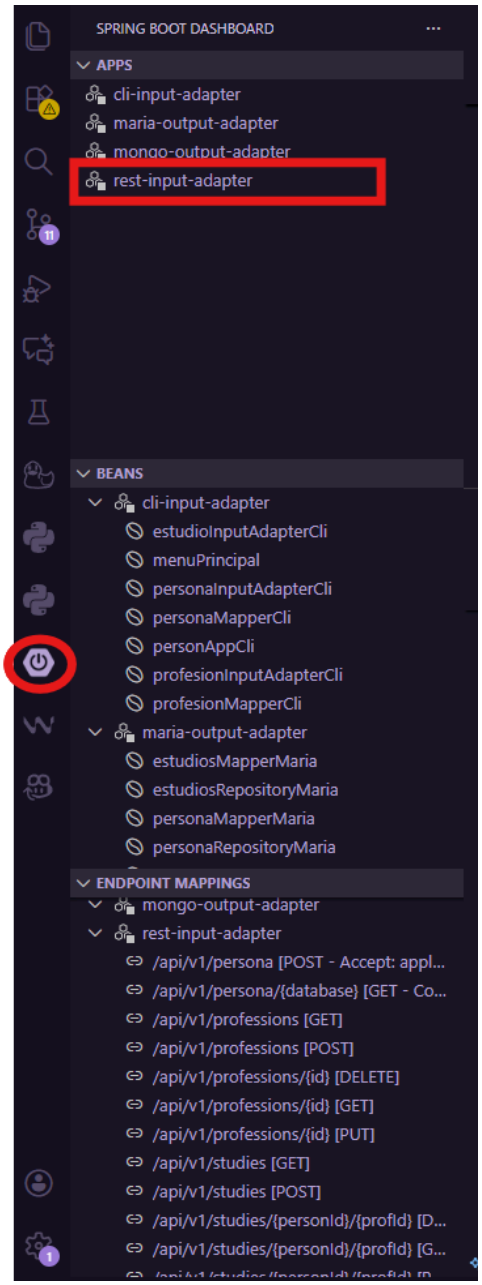


Para alternar entre bases de datos, seleccionar una de las opciones, ya dentro se encontrará un menú para alternar entre las bases de datos y seleccionar qué acciones del CRUD quiere realizar.





Para el caso de swagger UI, lo corremos desde la consola integrada en Visual Studio para proyectos springboot, al igual que el caso anterior es vital tener las bases de datos ya corriendo en docker.

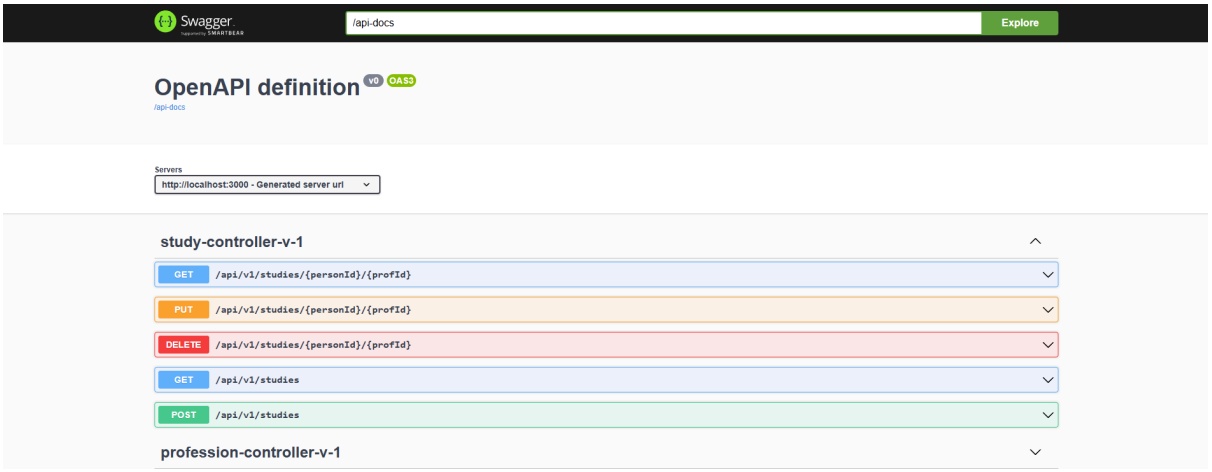


Una vez iniciado el programa dirigirse a la url:
http://localhost:3000/swagger-ui/index.html#

Arquitectura de Software, Universidad Javeriana

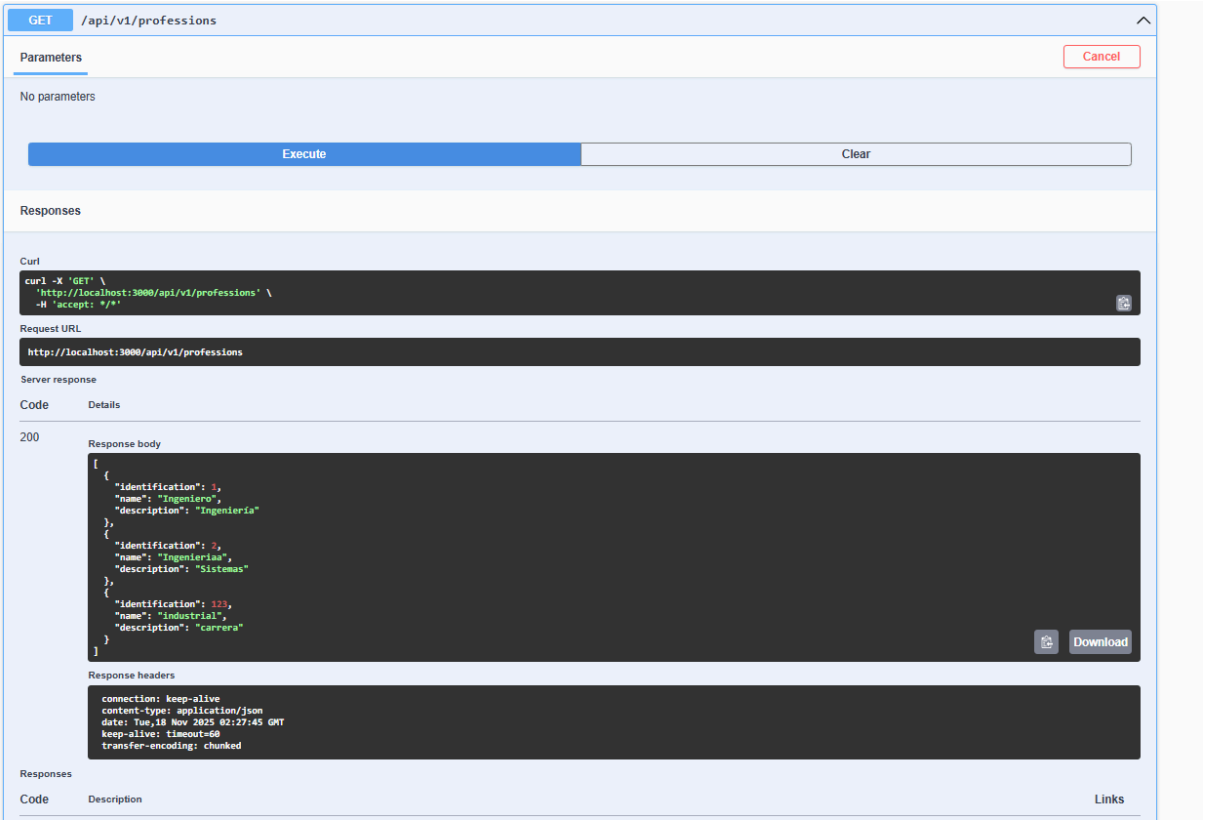
Laboratorio de Java Hexagonal Noviembre 2025

Pablo Enrique Quintero, Juan Diego Romero, Kamilt Bejarano Diaz

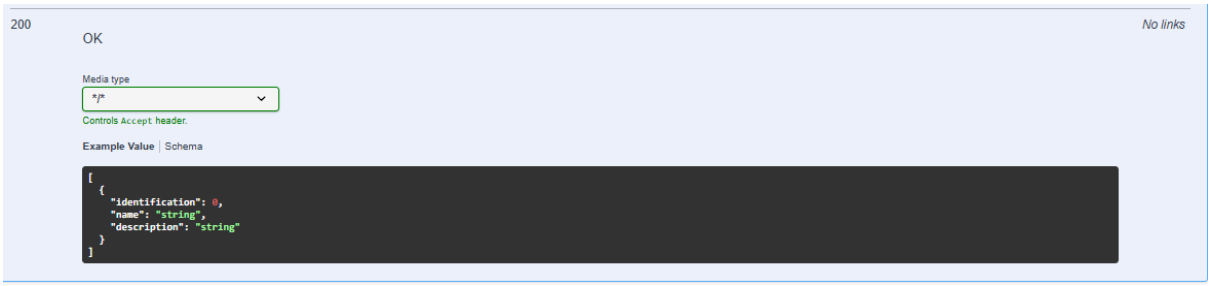


Acá se pueden ver y probar los distintos endpoints que existen en el programa, al igual que la estructura que siguen los datos para ser guardados.

Por ejemplo, al hacer uso del get de profesiones:



En este caso nos muestra todos los datos de la base de datos y justo debajo información de la conexión, al igual que información de la estructura de los datos como se puede ver en la siguiente foto.



Conclusiones

Este taller fue un gran reto en nuestro proceso de aprendizaje, pues la arquitectura hexagonal es bastante diferente a lo que estamos acostumbrados. Pese a esto, encontramos que esta arquitectura nos permite tener un sistema desacoplado, organizado y extensible, pues la separación de dominio, aplicación e infraestructura nos permite conectar diferentes bases de datos sin modificar la lógica de negocio del sistema.

Además, nos parece importante recalcar el uso de adaptadores CLI y REST, pues es una prueba de que una misma lógica de negocio puede exponerse a través de diferentes interfaces sin necesidad de duplicar código.Asimismo, la inclusión de Swagger facilitó la comprensión y prueba del API, demostrando su utilidad en proyectos que manejan múltiples servicios.

En general, este laboratorio contribuyó a reforzar conceptos de modularidad, mantenibilidad, buenas prácticas de diseño de software y la habilidad de los integrantes para leer código ajeno y entender su funcionamiento.