Laboratorio 2 - Arquitectura de Software - Hexagonal

Arquitectura Software

Estudiantes:

Andres Velez
Victor Sanchez
Victor Peñaranda

Profesor:

Ing. Andres Sanchez

Pontificia Universidad Javeriana
Facultad de Ingeniería,
Ingeniería de Sistemas
2025

Índice

1. Marco Conceptual

El presente laboratorio tiene como propósito central la implementación de un Servicio Web basado en Arquitectura Hexagonal, aplicando los patrones Repository y Service, y utilizando un stack tecnológico moderno compuesto por JDK 11, Spring Boot, MongoDB, MariaDB, REST, CLI y Swagger 3. Este conjunto de herramientas y principios arquitectónicos refleja las prácticas más sólidas en el desarrollo de software empresarial actual.

A continuación, se explican y contextualizan en profundidad cada uno de los elementos que componen este laboratorio.

1.1. Arquitectura Hexagonal (Ports and Adapters)

¿Qué es?

La Arquitectura Hexagonal, también conocida como Arquitectura de Puertos y Adaptadores, fue propuesta por Alistair Cockburn en 2005 como una forma de aislar el núcleo del sistema (la lógica del negocio) de sus interfaces externas (bases de datos, APIs, interfaces gráficas, etc.).

¿Cómo funciona?

- El núcleo del sistema (dominio) define interfaces llamadas **puertos**.
- Los **adaptadores** son implementaciones concretas de esos puertos que permiten que el sistema interactúe con el mundo exterior.
- El sistema puede ser ejecutado y probado sin depender de la infraestructura real.

¿Para qué sirve?

- Aumenta la modularidad.
- Permite probar la lógica de negocio sin depender de la infraestructura externa.
- Facilita la evolución tecnológica: cambiar MongoDB por PostgreSQL o REST por GraphQL, sin afectar el núcleo.

1.2. Patrón Repository

¿Qué es?

El patrón Repository actúa como una capa intermedia entre el dominio y la fuente de datos, aislando al sistema de los detalles técnicos del acceso a la base de datos.

¿Para qué sirve?

- Centraliza la lógica de acceso a datos.
- Facilita el mantenimiento y pruebas.
- Permite cambiar la base de datos sin afectar la lógica de negocio.

Ejemplo en Spring Boot:

[language=Java] public interface UsuarioRepository extends JpaRepository¡Usuario, Long¿

1.3. Patrón Service

¿Qué es?

El patrón Service organiza la lógica de negocio en una capa intermedia entre los controladores (adaptadores) y los repositorios.

¿Para qué sirve?

- Encapsula reglas de negocio.
- Reutiliza lógica común.

- Mejora la organización del código.

Ejemplo:
[language=Java] @Service public class UsuarioService public Usuario registrarUsuario(UsuarioDTO dto) // Lógica del negocio

1.4. Java Development Kit 11 (JDK 11)

Historia y contexto
Java fue lanzado en 1995 por Sun Microsystems como un lenguaje portable, orientado a objetos y seguro. Desde entonces, se ha convertido en uno de los lenguajes más utilizados en el mundo. La versión JDK 11 es una versión LTS (Long-Term Support) publicada en 2018.
¿Qué es?
El JDK (Java Development Kit) es el conjunto de herramientas necesarias para desarrollar y ejecutar aplicaciones Java. Incluye:

- Compilador javac
- Máquina virtual java
- Bibliotecas estándar

¿Por qué usar JDK 11?

- Estabilidad garantizada a largo plazo
- Mejoras en rendimiento
- Nuevas APIs como HttpClient y mejoras en String, Optional, Files

1.5. Spring Boot

¿Qué es?
Spring Boot es un framework que permite crear aplicaciones Java rápidamente, eliminando la necesidad de configurar manualmente el entorno de ejecución y los componentes.
¿Por qué es importante?

- Configuración automática (auto-configuration)
- Servidores embebidos (como Tomcat)
- Integración con bases de datos y APIs REST
- Ideal para microservicios y sistemas empresariales

Historia
Spring nació en 2003 como una alternativa a los pesados modelos EJB de Java EE. Spring Boot aparece en 2014 para simplificar el desarrollo con Spring, reduciendo drásticamente el tiempo de configuración.

1.6. MongoDB

¿Qué es?
MongoDB es una base de datos NoSQL orientada a documentos, donde los datos se almacenan en forma de objetos BSON (una versión binaria de JSON).
Características:

- No requiere esquema fijo (schema-less)
- Escalabilidad horizontal
- Buen rendimiento para lecturas y escrituras

¿Cuándo usarla?

- Cuando los datos tienen estructura flexible o cambiante
- Para almacenar logs, eventos o configuraciones dinámicas
- En aplicaciones en tiempo real o big data

1.7. MariaDB

¿Qué es?
MariaDB es una base de datos relacional derivada de MySQL, creada para garantizar un desarrollo libre tras la adquisición de MySQL por Oracle.
Características:

- Lenguaje estándar SQL
- Transacciones ACID
- Integridad referencial
- Soporte de JOINs y relaciones complejas

¿Cuándo usarla?

- Para sistemas con relaciones entre entidades (ej. usuarios, roles, productos)
- Cuando se requiere validación estricta y estructuras normalizadas

1.8. REST (Representational State Transfer)

¿Qué es?
REST es un estilo arquitectónico para servicios web basado en HTTP. Fue introducido en el año 2000 por Roy Fielding.
Principios clave:

- Recursos representados con URLs
- Operaciones estándar: GET, POST, PUT, DELETE
- Stateless (sin estado)
- Comunicación mediante JSON o XML

¿Por qué usar REST?

- Es liviano, escalable y fácil de consumir
- Es el estándar para APIs modernas
- Compatible con cualquier cliente (web, móvil, CLI)

1.9. CLI (Command Line Interface)

¿Qué es?
La Interfaz de Línea de Comandos (CLI) permite a los usuarios interactuar con el sistema usando comandos de texto.
Ventajas:

- Ideal para pruebas técnicas y automatización
- Requiere menos recursos que interfaces gráficas
- Permite ejecutar scripts y tareas rápidamente

En este laboratorio:
Se utiliza la CLI para probar los endpoints del servicio REST, usando herramientas como curl, httpie o scripts personalizados.

1.10. Swagger 3 / OpenAPI 3.0

¿Qué es?

Swagger, ahora conocido como OpenAPI, es un estándar que permite documentar, visualizar y probar APIs REST.

¿Por qué es importante?

- Genera documentación automática
- Facilita la colaboración entre equipos
- Actúa como contrato entre el backend y el frontend

En el laboratorio:

Se utiliza la biblioteca springdoc-openapi para generar una interfaz Swagger accesible desde el navegador.

2. Diseño

2.1. Alto Nivel

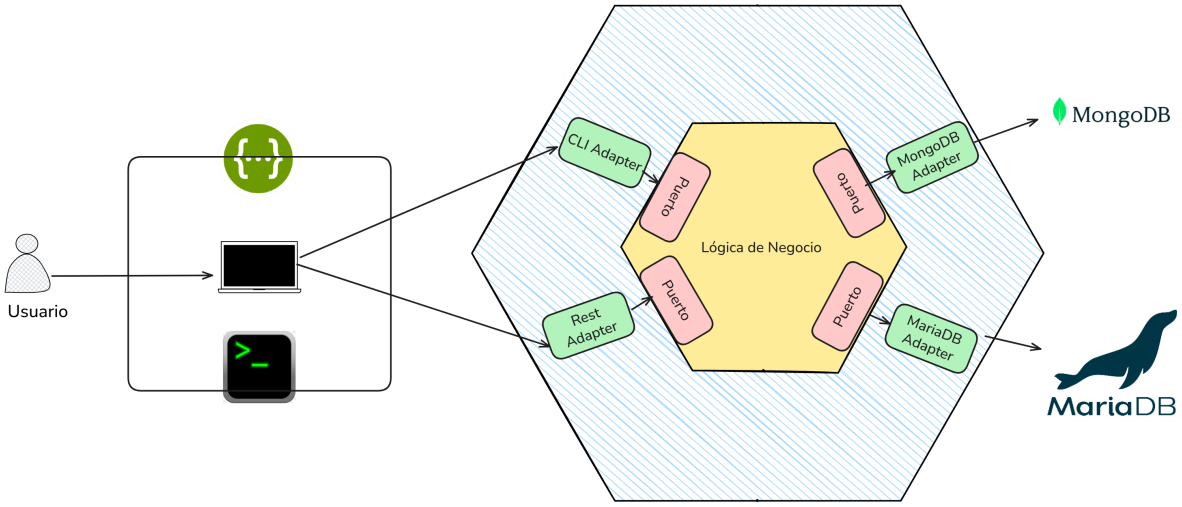


Figura 1: Diagrama Arquitectura Alto Nivel

El diagrama refleja la implementación de una arquitectura hexagonal en el proyecto, resaltando cómo se mantiene separada la lógica central del sistema respecto a sus interfaces externas. En el núcleo se encuentra el Modelo del Negocio, que contiene las reglas y procesos fundamentales, completamente aislado de detalles técnicos y tecnologías particulares. Rodeando este núcleo se ubican los puertos, que actúan como interfaces de comunicación entre la lógica interna y el mundo exterior, promoviendo un diseño modular e independiente. A través de los adaptadores de entrada, como el CLI y el REST, el sistema puede recibir solicitudes tanto desde la línea de comandos como mediante peticiones HTTP. Por otro lado, los adaptadores de salida permiten la interacción con sistemas de persistencia, conectando la aplicación con bases de datos como MongoDB y MariaDB sin que la lógica de negocio dependa directamente de ellas. Este enfoque arquitectónico garantiza que el sistema sea fácilmente extensible, mantenible y adaptable a nuevas tecnologías sin comprometer la integridad del modelo de negocio.

2.2. Diagramas C4

2.2.1. Diagrama de Contexto

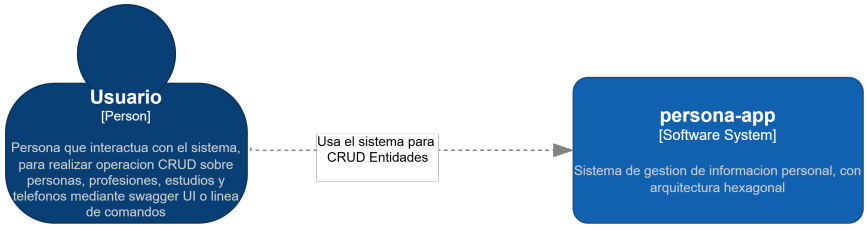


Figura 2: Diagrama Contexto - C4

El sistema se basa en una arquitectura hexagonal, lo que permite una separación clara entre la lógica de negocio y las interfaces externas. Para la exposición de servicios, se implementa una API REST que facilita la comunicación con los clientes. El desarrollo se apoya en el framework Spring Boot, y emplea MariaDB y MongoDB como tecnologías de persistencia de datos.

2.2.2. Diagrama de Contenedores

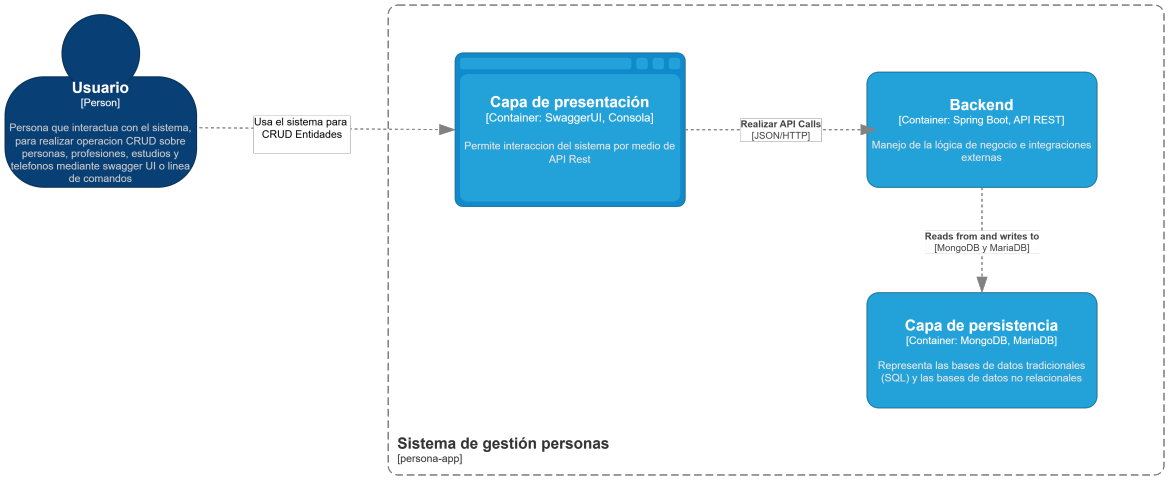


Figura 3: Diagrama Contenedores C4

2.2.3. Diagrama Componentes

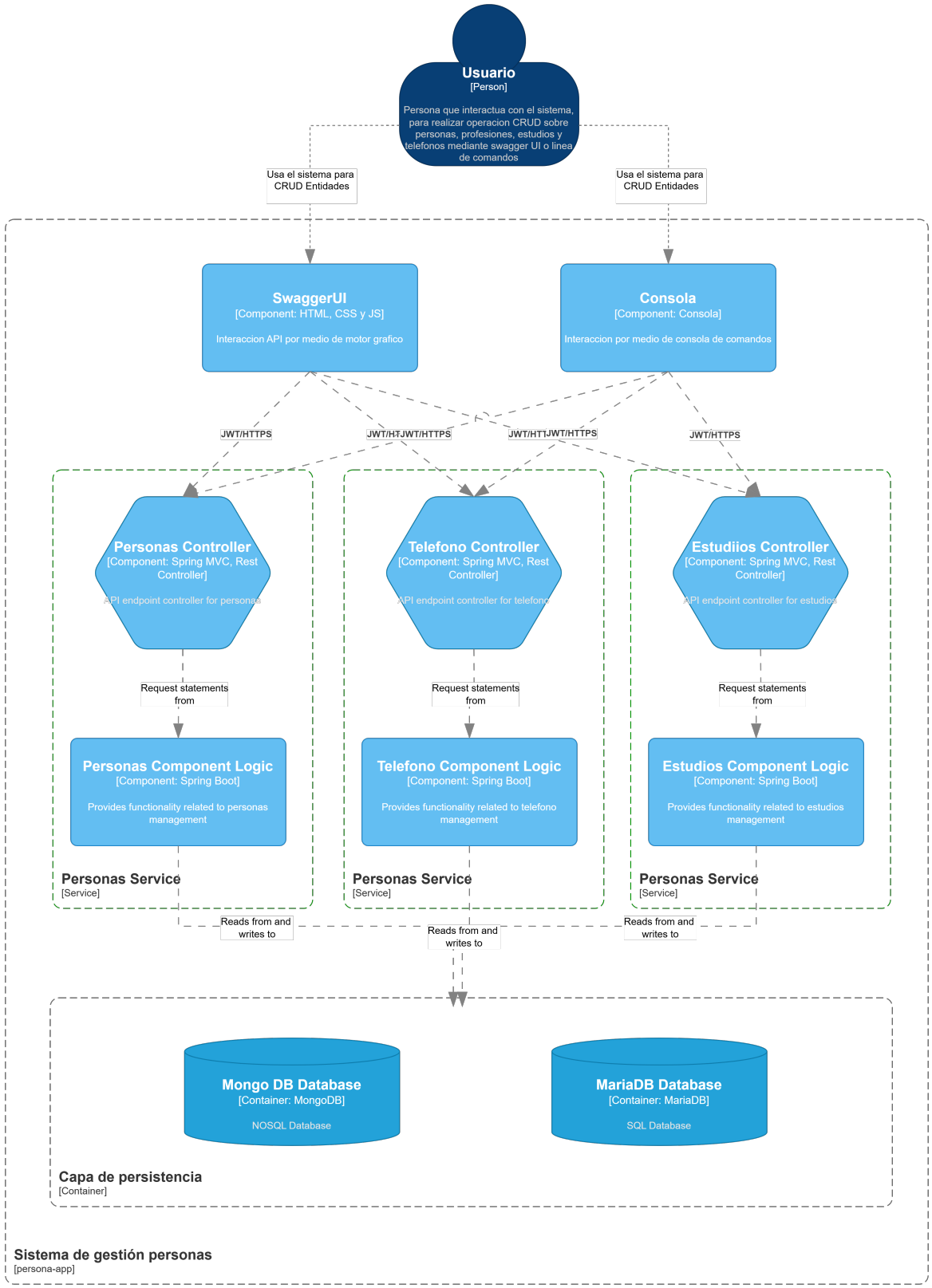


Figura 4: Diagrama Componentes C4

2.2.4. Diagrama de Despliegue

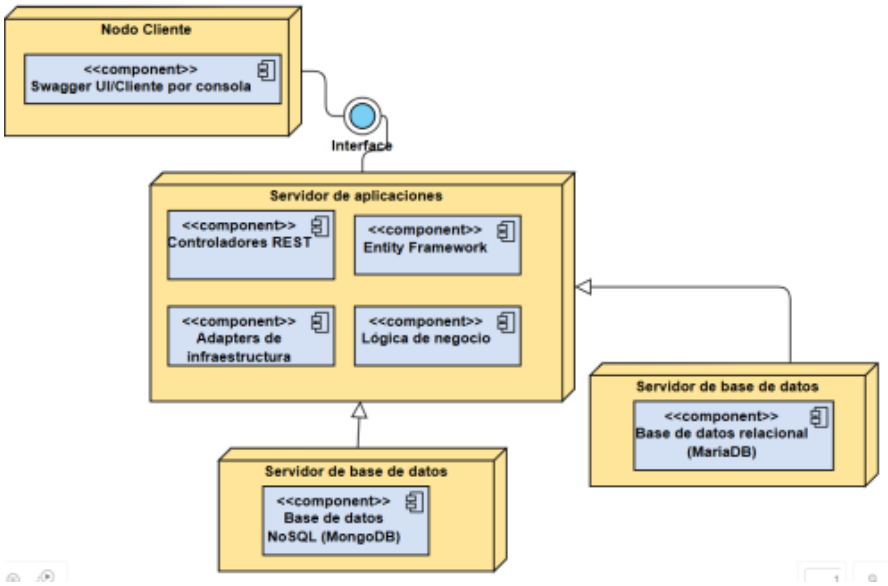


Figura 5: Diagrama de despligue

2.2.5. Diagrama Logico

Este diagrama ilustra un modelo de datos relacional orientado al almacenamiento de información vinculada a personas, incluyendo sus profesiones, estudios y números de teléfono. El objetivo principal del modelo es organizar de manera estructurada los datos sobre los individuos, sus trayectorias académicas y profesionales, así como los medios de contacto asociados

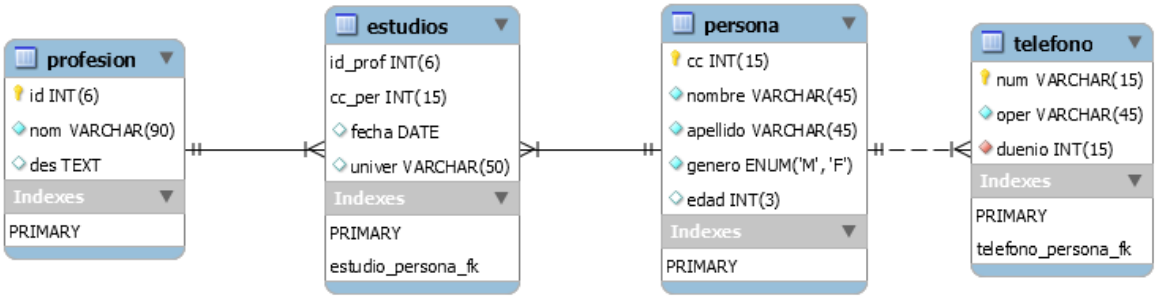


Figura 6: Diseño Base de datos

- **Profesión y estudios**: Existe una relación uno a muchos (1:N), en la que una profesión puede estar vinculada a varios estudios, mientras que cada estudio está asociado únicamente a una profesión.
- **Estudios y persona**: Una persona puede contar con varios estudios registrados, pero cada uno de ellos corresponde a una sola persona, manteniendo también una relación uno a muchos (1:N).
- **Persona y teléfono**: Del mismo modo, una persona puede tener varios números de teléfono asignados, y cada número está relacionado exclusivamente con una persona, lo que constituye otra relación uno a muchos (1:N).

3. Procedimiento

3.1. Creación del proyecto en Spring Boot

Primero se forkea el repositorio: https://github.com/andres-karoll/personapp-hexa-springboot.git

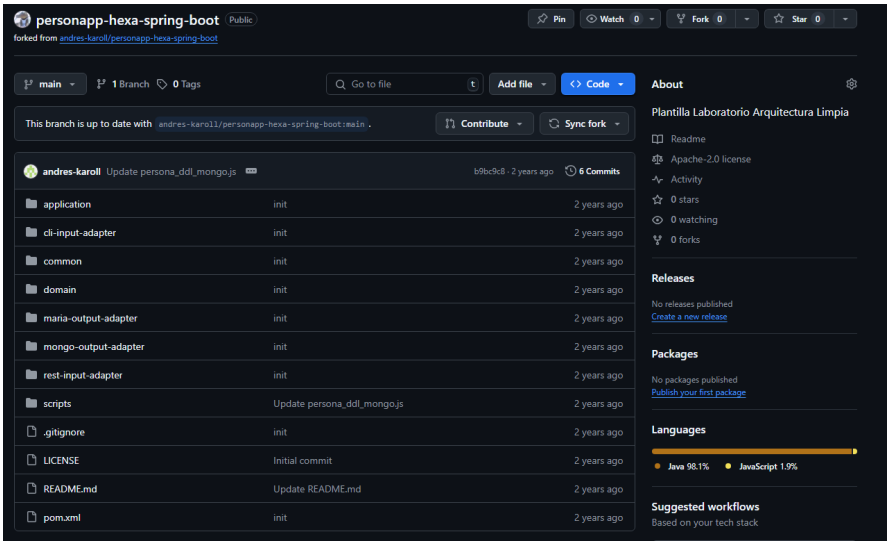


Figura 7: Fork

3.2. Instalación de MongoDB

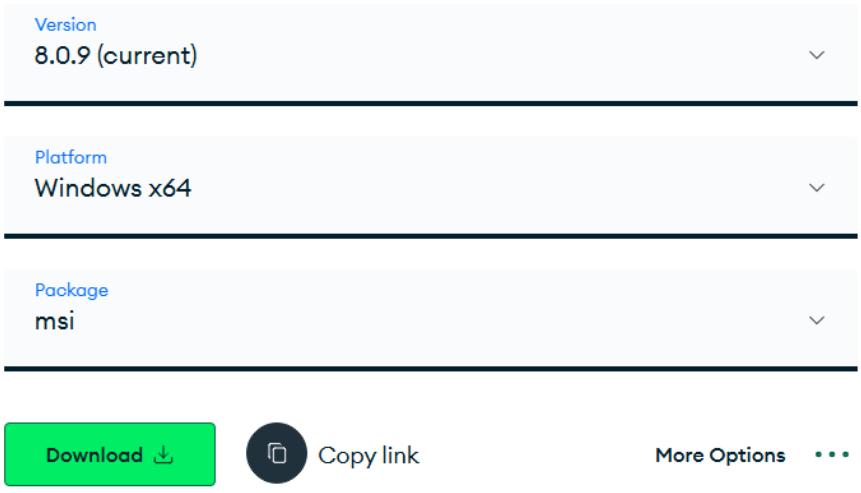


Figura 8: Instalación MongoDB

- Se accede al sitio oficial de MongoDB para descargar la versión 8.0.9.
- Se selecciona el sistema operativo y el instalador MSI.
- Se instala el Shell de MongoDB y la herramienta gráfica MongoDB Compass.
- Se crea una conexión por defecto utilizando el puerto 27017.

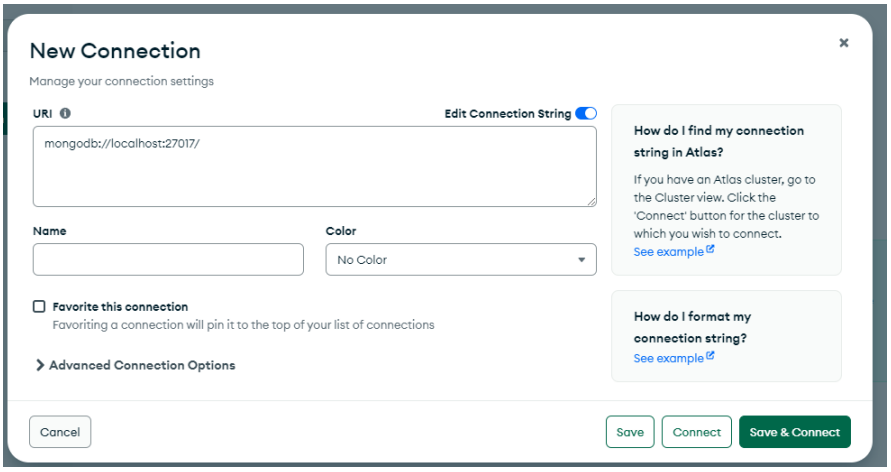


Figura 9: Conexión MongoDB

3.3. Instalación de MariaDB

Figura 10: Instalación Maria DB

- Se utiliza HeidiSQL como herramienta gratuita para conectarse a la base de datos.
- Se descarga el instalador MSI correspondiente.
- Tras instalarlo, se configura una contraseña y un puerto para el servicio.

3.4. Creación de la Base de Datos

En este proyecto se emplea Docker Compose para inicializar automáticamente las bases de datos MongoDB y MariaDB, garantizando que estén listas para operaciones CRUD.

3.4.1. Base de Datos MongoDB

- En el archivo `docker-compose.yml` se definen base de datos, usuario y contraseña.
- Se utilizan los archivos `persona_ddl_mongo.js` y `persona_dml_mongo.js` para crear la estructura y datos iniciales.

- Estos scripts se ubican en `/docker-entrypoint-initdb.d/` para ejecutarse automáticamente en el primer arranque.
- Se garantiza la creación de la base `persona_db` y la colección `persona`.

```
db = db.getSiblingDB("persona_db");

db.persona.insertMany([
    {
        "_id": NumberInt(123456789),
        "nombre": "Pepe",
        "apellido": "Perez",
        "genero": "M",
        "edad": NumberInt(30),
        "_class": "co.edu.javeriana.as.personapp.mongo.document.PersonaDocument"
    },
    {
        "_id": NumberInt(987654321),
        "nombre": "Pepito",
        "apellido": "Perez",
        "genero": "M",
        "_class": "co.edu.javeriana.as.personapp.mongo.document.PersonaDocument"
    },
    {
        "_id": NumberInt(321654987),
        "nombre": "Pepa",
        "apellido": "Juarez",
        "genero": "F",
        "edad": NumberInt(30),
        "_class": "co.edu.javeriana.as.personapp.mongo.document.PersonaDocument"
    },
    {
        "_id": NumberInt(147258369),
        "nombre": "Pepita",
        "apellido": "Juarez",
        "genero": "F",
        "edad": NumberInt(10),
        "_class": "co.edu.javeriana.as.personapp.mongo.document.PersonaDocument"
    },
    {
        "_id": NumberInt(963852741),
        "nombre": "Fede",
        "apellido": "Perez",
        "genero": "M",
        "edad": NumberInt(18),
        "_class": "co.edu.javeriana.as.personapp.mongo.document.PersonaDocument"
    }
], { ordered: false })      You, 20 hours ago • Creación de proyecto …
```

Figura 11: Creación DB MongoDB

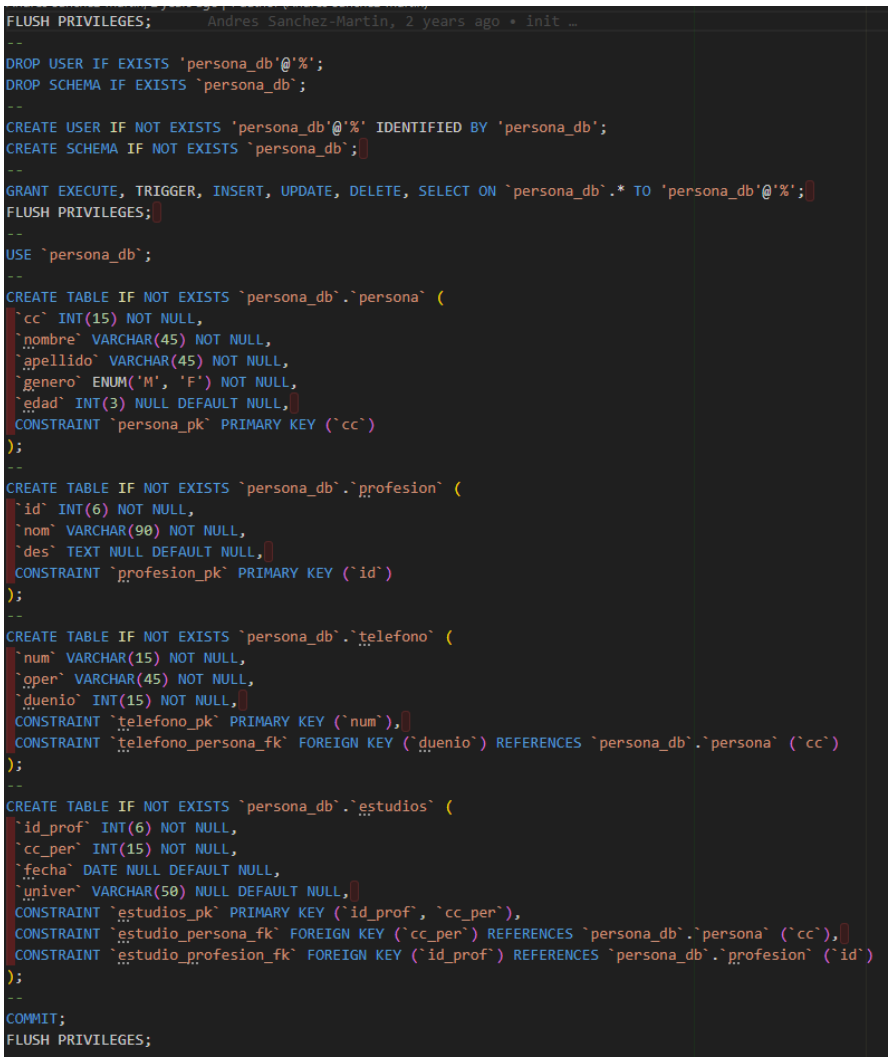
```
db = db.getSiblingDB("persona_db");

db.createUser({
  user: "persona_db",
  pwd: "persona_db",
  roles: [
    { role: "read", db: "persona_db" },
    { role: "readWrite", db: "persona_db" },
    { role: "dbAdmin", db: "persona_db" }
  ],
  mechanisms: ["SCRAM-SHA-1","SCRAM-SHA-256"]
})
```

Figura 12: Creación DB MongoDB 2

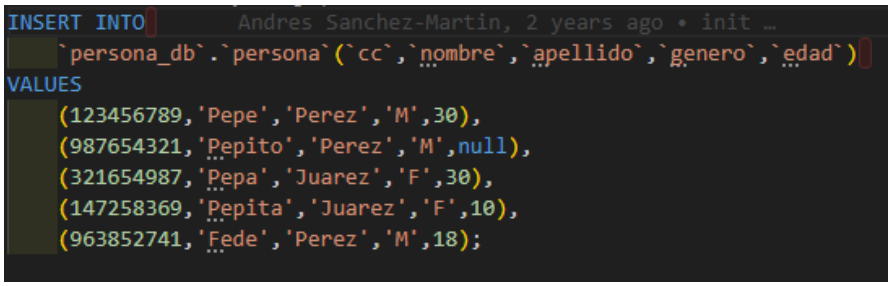
3.4.2. Base de Datos MariaDB

- En `docker-compose.yml` se define la base `persona_db` y las credenciales del usuario raíz.
- Se incluyen los archivos `persona_ddl_maria.sql` y `persona_dml_maria.sql`.
- Estos archivos crean las tablas: `profesion`, `estudios`, `persona`, `telefono`.
- Los scripts se ejecutan automáticamente desde `docker-entrypoint-initdb.d`.



```
FLUSH PRIVILEGES;      Andres Sanchez-Martin, 2 years ago • init …
--
DROP USER IF EXISTS 'persona_db'@'%';
DROP SCHEMA IF EXISTS `persona_db`;
--
CREATE USER IF NOT EXISTS 'persona_db'@'%' IDENTIFIED BY 'persona_db';
CREATE SCHEMA IF NOT EXISTS `persona_db`;
--
GRANT EXECUTE, TRIGGER, INSERT, UPDATE, DELETE, SELECT ON `persona_db`.* TO 'persona_db'@'%';
FLUSH PRIVILEGES;
--
USE `persona_db`;
--
CREATE TABLE IF NOT EXISTS `persona_db`.`persona` (
 `cc` INT(15) NOT NULL,
 `nombre` VARCHAR(45) NOT NULL,
 `apellido` VARCHAR(45) NOT NULL,
 `genero` ENUM('M', 'F') NOT NULL,
 `edad` INT(3) NULL DEFAULT NULL,
 CONSTRAINT `persona_pk` PRIMARY KEY (`cc`)
);
--
CREATE TABLE IF NOT EXISTS `persona_db`.`profesion` (
 `id` INT(6) NOT NULL,
 `nom` VARCHAR(90) NOT NULL,
 `des` TEXT NULL DEFAULT NULL,
 CONSTRAINT `profesion_pk` PRIMARY KEY (`id`)
);
--
CREATE TABLE IF NOT EXISTS `persona_db`.`telefono` (
 `num` VARCHAR(15) NOT NULL,
 `oper` VARCHAR(45) NOT NULL,
 `duenio` INT(15) NOT NULL,
 CONSTRAINT `telefono_pk` PRIMARY KEY (`num`),
 CONSTRAINT `telefono_persona_fk` FOREIGN KEY (`duenio`) REFERENCES `persona_db`.`persona` (`cc`)
);
--
CREATE TABLE IF NOT EXISTS `persona_db`.`estudios` (
 `id_prof` INT(6) NOT NULL,
 `cc_per` INT(15) NOT NULL,
 `fecha` DATE NULL DEFAULT NULL,
 `univer` VARCHAR(50) NULL DEFAULT NULL,
 CONSTRAINT `estudios_pk` PRIMARY KEY (`id_prof`, `cc_per`),
 CONSTRAINT `estudio_persona_fk` FOREIGN KEY (`cc_per`) REFERENCES `persona_db`.`persona` (`cc`),
 CONSTRAINT `estudio_profesion_fk` FOREIGN KEY (`id_prof`) REFERENCES `persona_db`.`profesion` (`id`)
);
--
COMMIT;
FLUSH PRIVILEGES;
```

Figura 13: Maria DB Creacion



```
INSERT INTO      Andres Sanchez-Martin, 2 years ago • init …
    `persona_db`.`persona`(`cc`,`nombre`,`apellido`,`genero`,`edad`)
VALUES
    (123456789,'Pepe','Perez','M',30),
    (987654321,'Pepito','Perez','M',null),
    (321654987,'Pepa','Juarez','F',30),
    (147258369,'Pepita','Juarez','F',10),
    (963852741,'Fede','Perez','M',18);
```

Figura 14: Maria DB Creacion 2

3.5. Estructura de Archivos del Proyecto

El proyecto está dividido en los siguientes módulos:

- **application**: Lógica de negocio y casos de uso.
- **cli-input-adapter**: Interacción por línea de comandos.
- **common**: Clases auxiliares y recursos compartidos.

- **domain**: Entidades del dominio.
- **maria-output-adapter**: Persistencia con MariaDB.
- **mongo-output-adapter**: Persistencia con MongoDB.
- **rest-input-adapter**: Controladores REST.
- **scripts**: Scripts de inicialización de bases de datos.

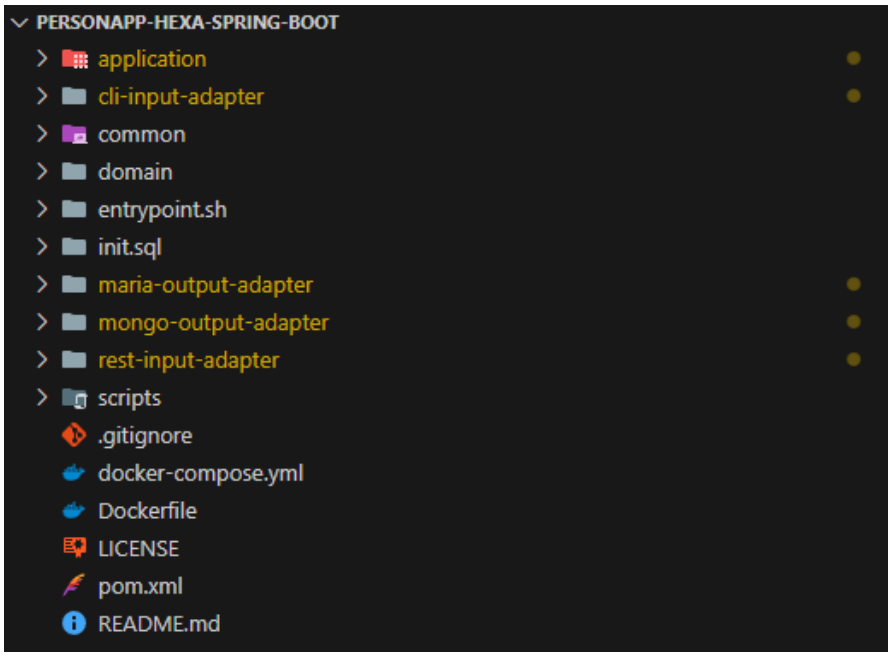


Figura 15: Estructura Archivos

3.6. Lógica de Negocio: InputPorts, OutputPorts y UseCases

- El módulo `application` es el núcleo de la lógica del sistema.
- Contiene los puertos de entrada (InputPorts) y salida (OutputPorts), que desacoplan la lógica del negocio.
- Los casos de uso (UseCases) orquestan las operaciones del sistema.
- Para la entidad `Person`, `PersonInputPort` gestiona las solicitudes de los controladores.
- `PersonUseCase` implementa la lógica del negocio y delega la persistencia a `PersonOutputPort`.

Este diseño permite un sistema testeable, escalable y desacoplado de tecnologías externas.

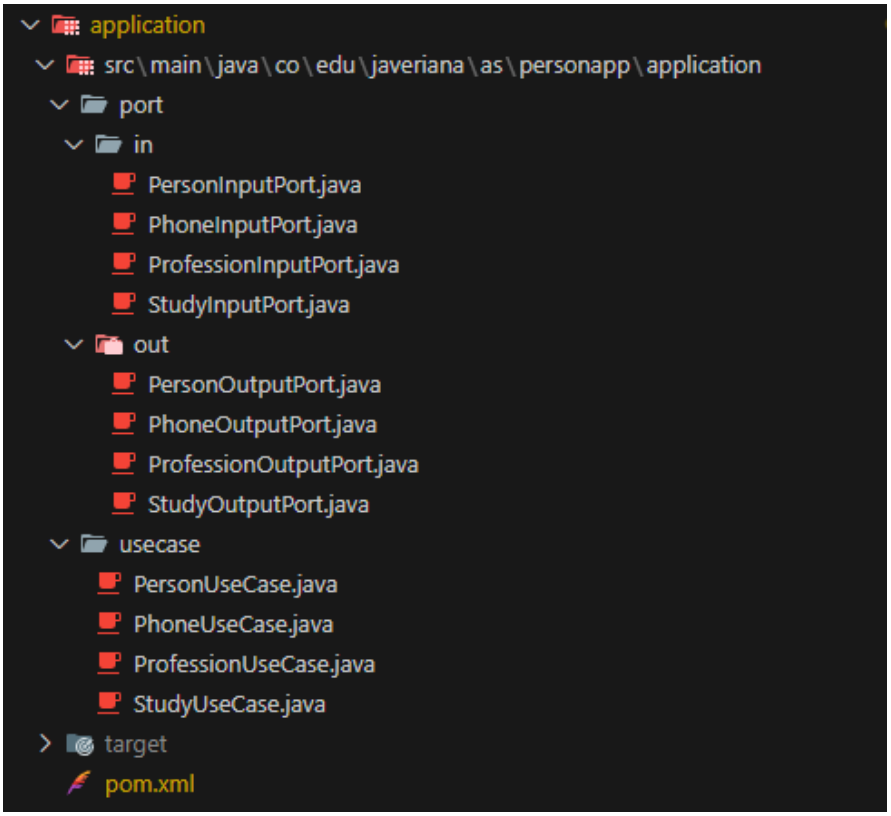


Figura 16: Logica de Negocio

3.7. Adaptadores de Entrada: REST y CLI

3.7.1. REST

- Los controladores REST exponen endpoints HTTP.
- Cada entidad posee su controlador: `PersonaControllerV1`, `EstudioControllerV1`, `ProfesionControllerV1`, `TelefonoControllerV1`.
- Se soportan operaciones CRUD mediante `GET`, `POST`, `PUT`, `DELETE`.

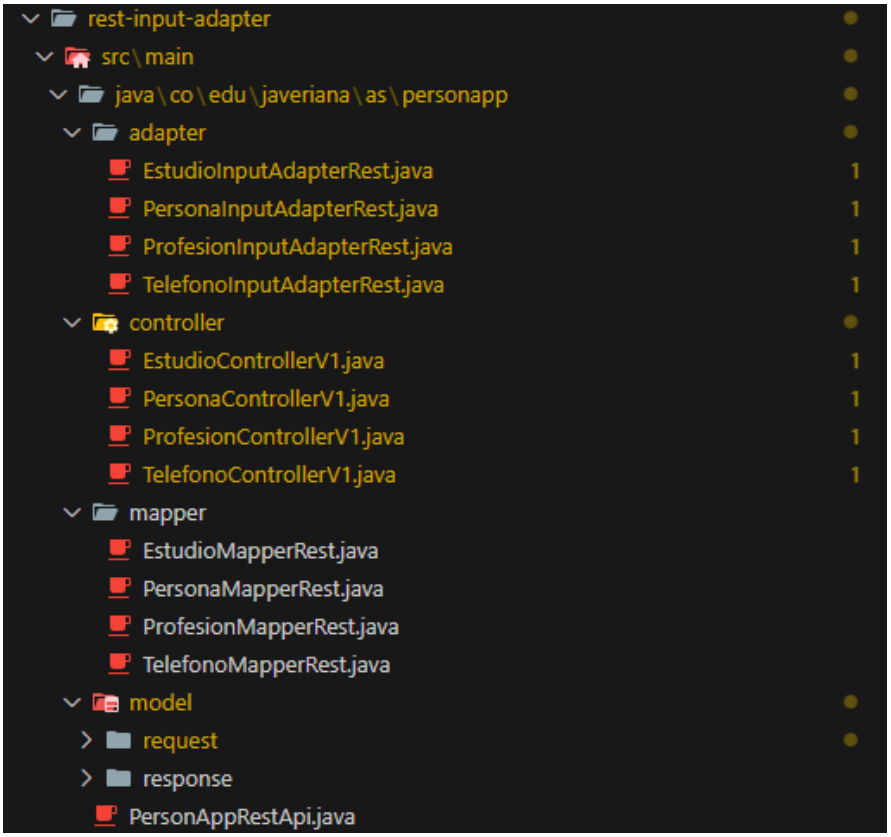


Figura 17: Lógica de Negocio - Adapters

3.7.2. CLI

- Permite interacción por consola mediante `PersonAppCli`.
- Se organizan los subdirectorios: adapter, mapper, menu, model.
- Permite ejecutar operaciones CRUD sin interfaz gráfica.

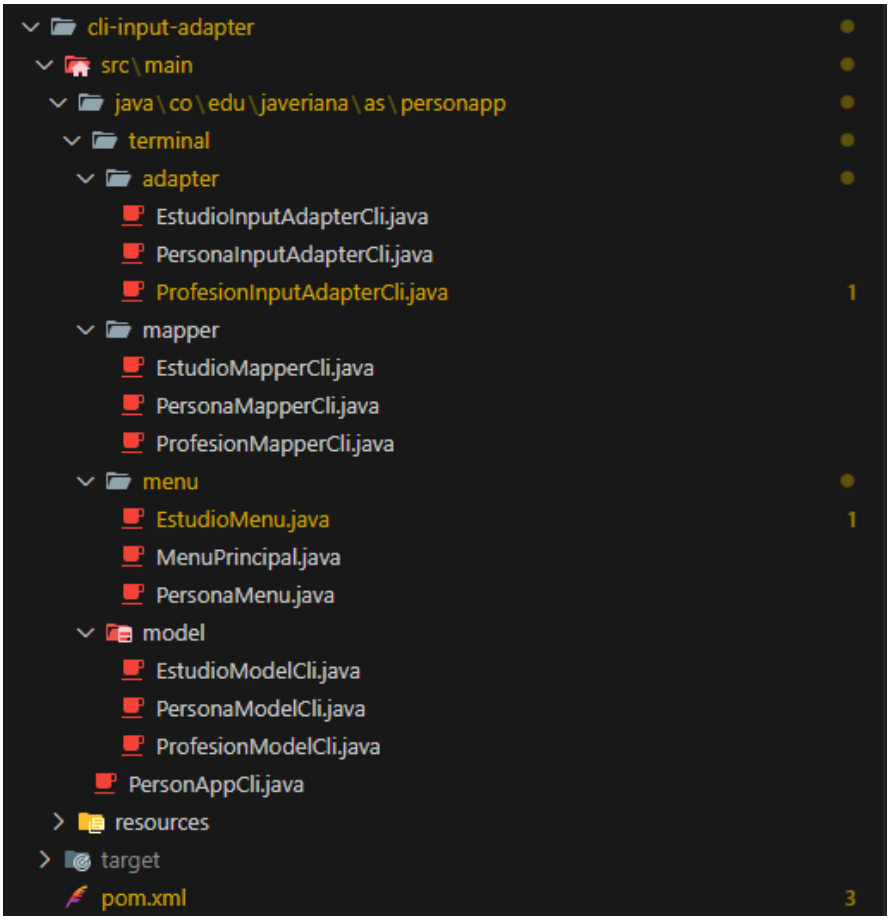


Figura 18: Logica CLI

3.8. Adaptadores de Salida: Persistencia MariaDB y MongoDB

3.8.1. MariaDB

- `PersonOutputAdapterMaria` implementa `PersonOutputPort` usando `JpaRepository`.
- Las entidades relacionales están en el subdirectorio `entity`.
- Los mappers convierten entre modelo de dominio y relacional.
- Los repositorios gestionan operaciones CRUD.

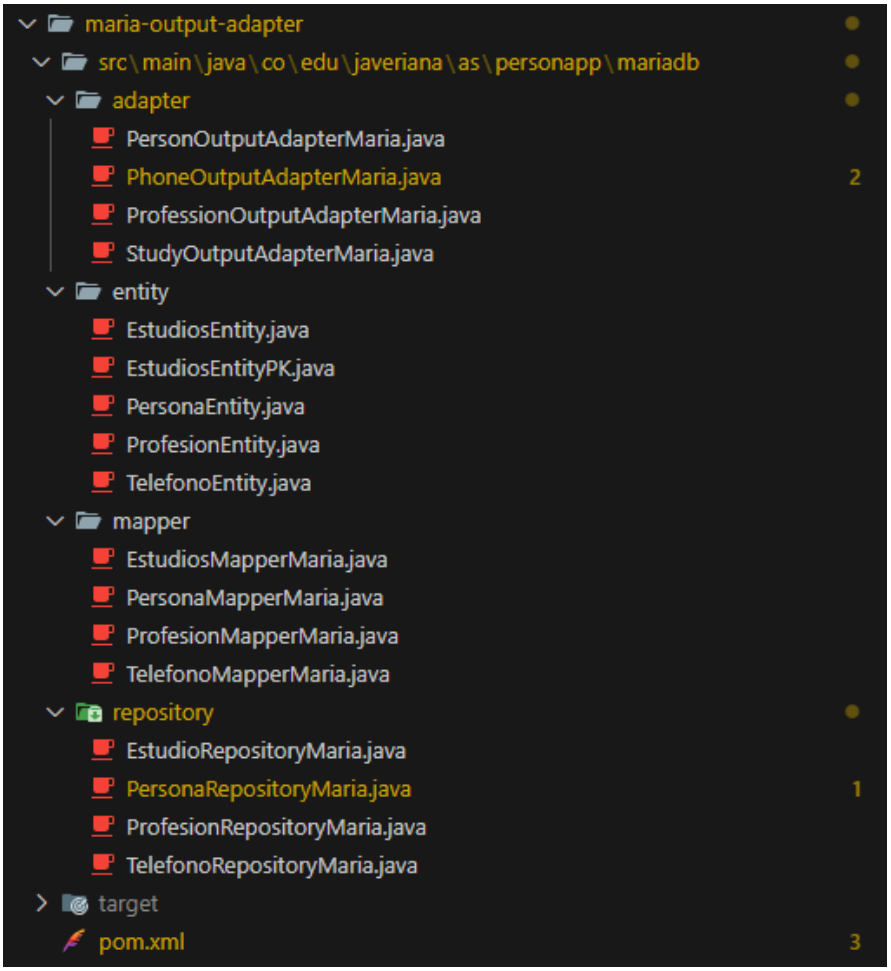


Figura 19: Logica Adapters OUT

3.8.2. MongoDB

- `PersonOutputAdapterMongo` implementa `PersonOutputPort` usando `MongoRepository`.
- Las clases `document` representan colecciones.
- Los mappers convierten documentos a objetos del dominio.
- Los repositorios gestionan la persistencia NoSQL.

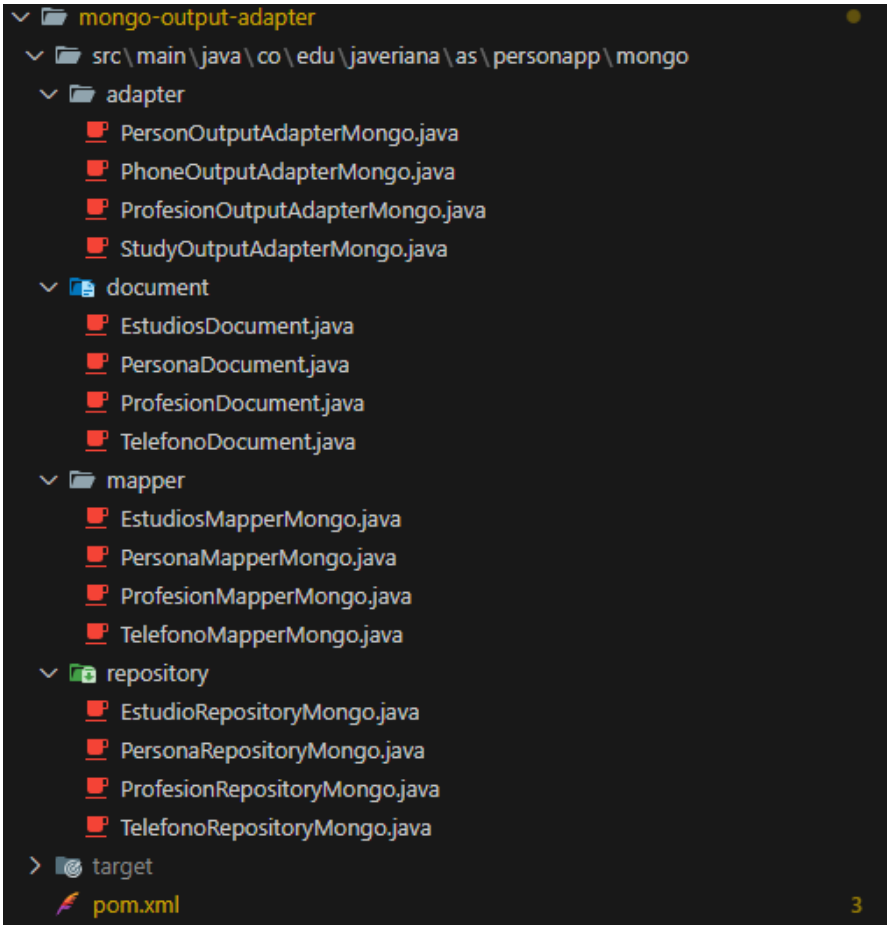


Figura 20: Logica Adapters Out 2

3.9. Configuración de Bases de Datos y Despliegue con Docker

- El módulo `scripts` contiene los archivos necesarios para inicializar las bases de datos.
- Scripts:
  - `persona_ddl_maria.sql`
  - `persona_dml_mongo.js`
- Estos se ejecutan automáticamente desde `/docker-entrypoint-initdb.d`.
- El archivo `docker-compose.yml` orquesta:
  - MariaDB
  - MongoDB
  - Aplicación Spring Boot
- Configura variables de entorno, volúmenes, puertos y dependencias.
- Con `docker-compose up --build` se despliega el entorno completo.

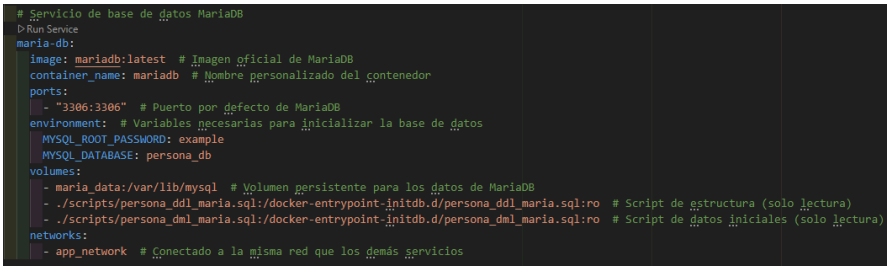


Figura 21: Docker

Figura 22: Docker 2

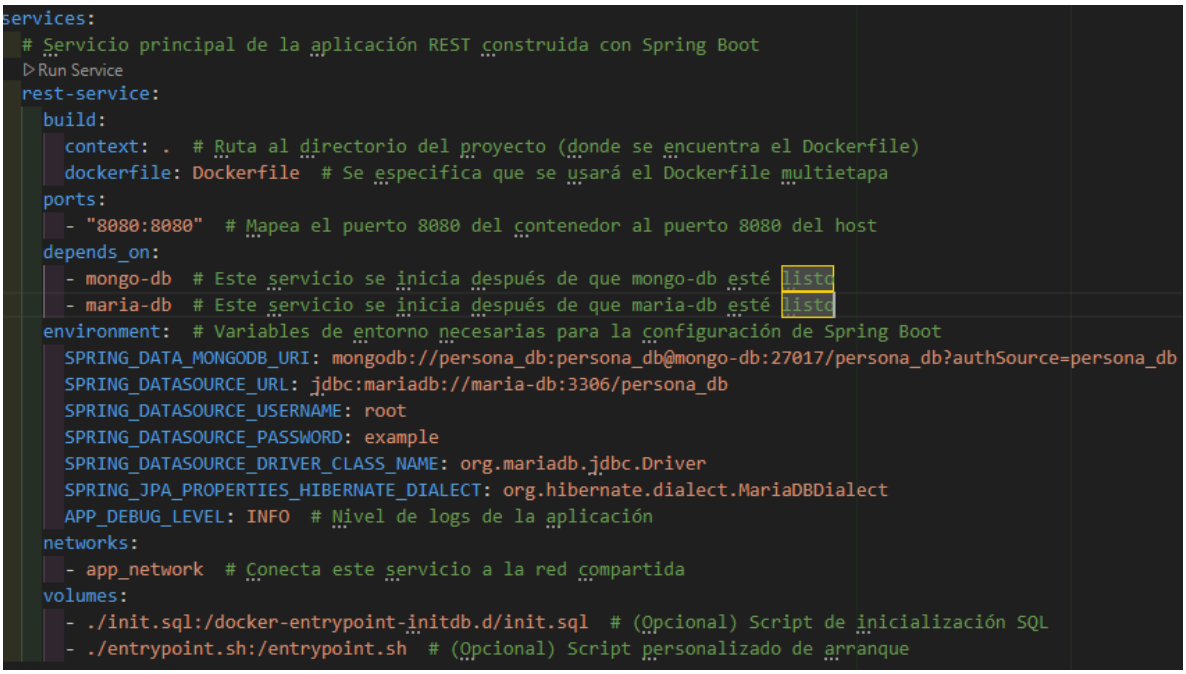


Figura 23: Compose

4. Conclusiones y Lecciones Aprendidas

- La arquitectura hexagonal permitió un sistema modular, mantenible y escalable.
- Separar lógica de negocio e infraestructura facilita reemplazar tecnologías como bases de datos.
- Spring Boot aceleró la creación de APIs REST.
- La planificación previa mediante diagramas C4 ayudó a anticipar desafíos.

- Docker y docker-compose facilitaron entornos reproducibles.
- Documentar con Swagger mejoró las pruebas y el trabajo en equipo.

Esta experiencia reforzó principios clave como diseño limpio, modularidad, desacoplamiento y buenas prácticas de desarrollo profesional.