

Laboratorio #2

Integrantes:

Samuel Ramirez

Juan Diego Pérez

Juan Miguel Zuluaga



GRUPO 8

ARQUITECTURA DE SOFTWARE

PONTIFICIA UNIVERSIDAD JAVERIANA

BOGOTÁ D.C

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. 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.

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:

- `public interface UsuarioRepository extends JpaRepository<Usuario, Long> {}`

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:

- @Service
public class UsuarioService {
 public Usuario registrarUsuario(UsuarioDTO dto) {
 // Lógica del negocio
 }
}

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

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.

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

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

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)

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.

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?

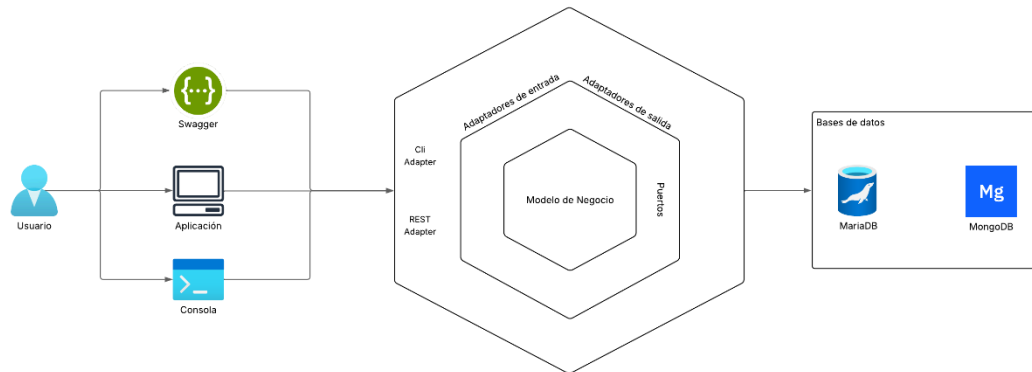
- Genera documentación automática
- Facilita la colaboración entre equipos
- Actúa como contrato entre el backend y el frontend

En el laboratorio:

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

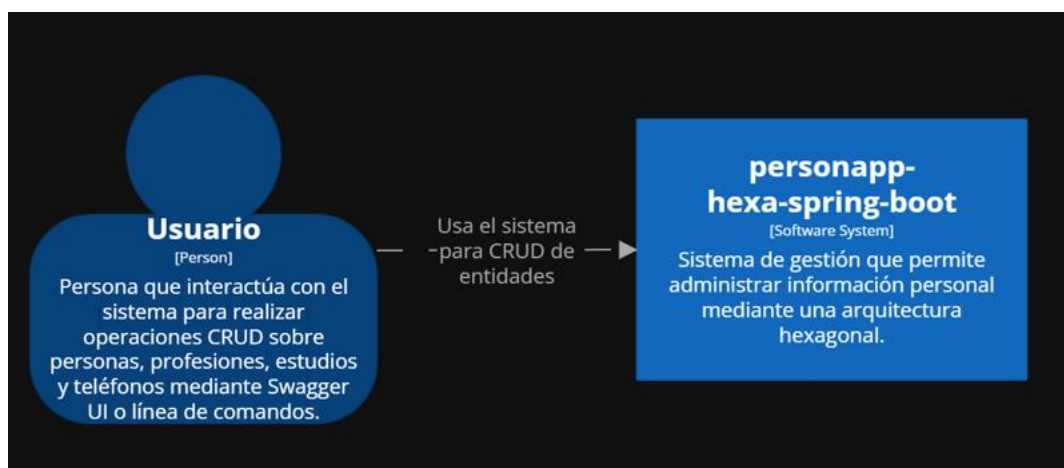
Diseño

- Diagrama de alto nivel



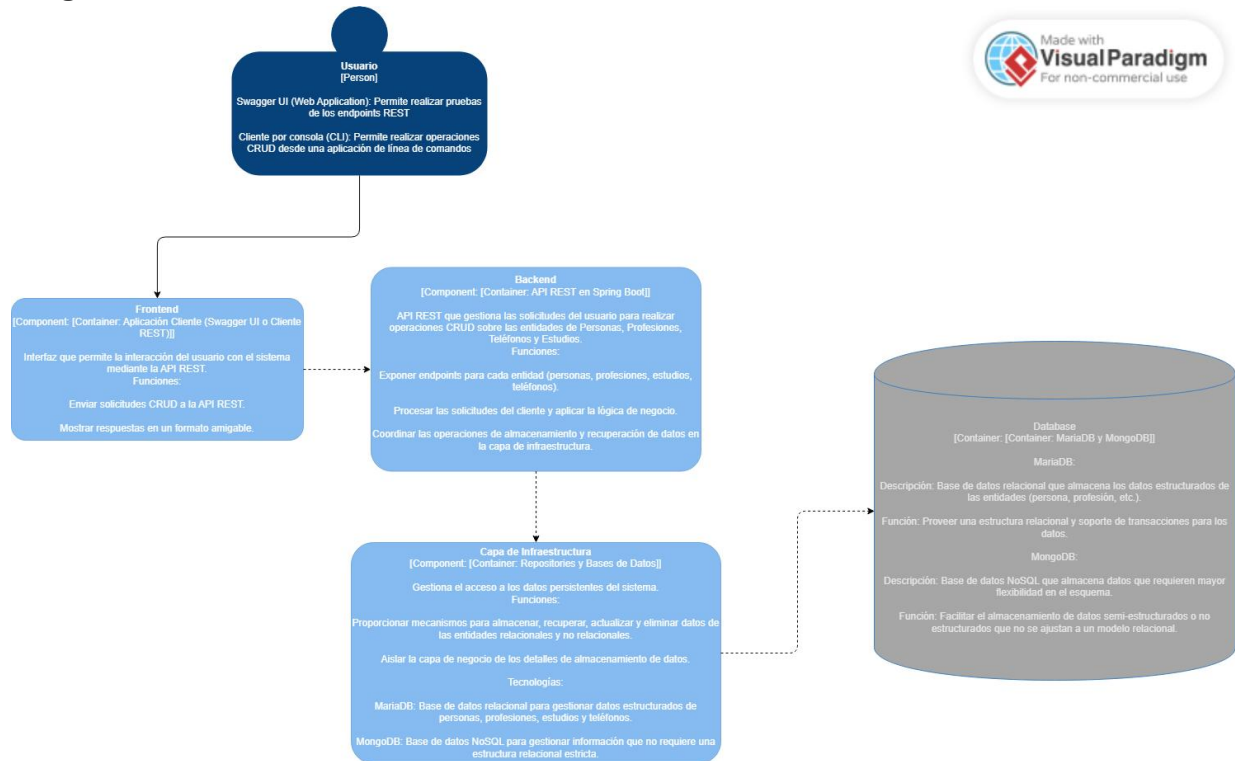
El diagrama refleja la implementación de una arquitectura hexagonal en el proyecto, resaltando cómo se mantiene separada la lógica central del sistema respecto a sus interfaces externas. En el núcleo se encuentra el Modelo del Negocio, que contiene las reglas y procesos fundamentales, completamente aislado 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.

- Diagrama de contexto C4

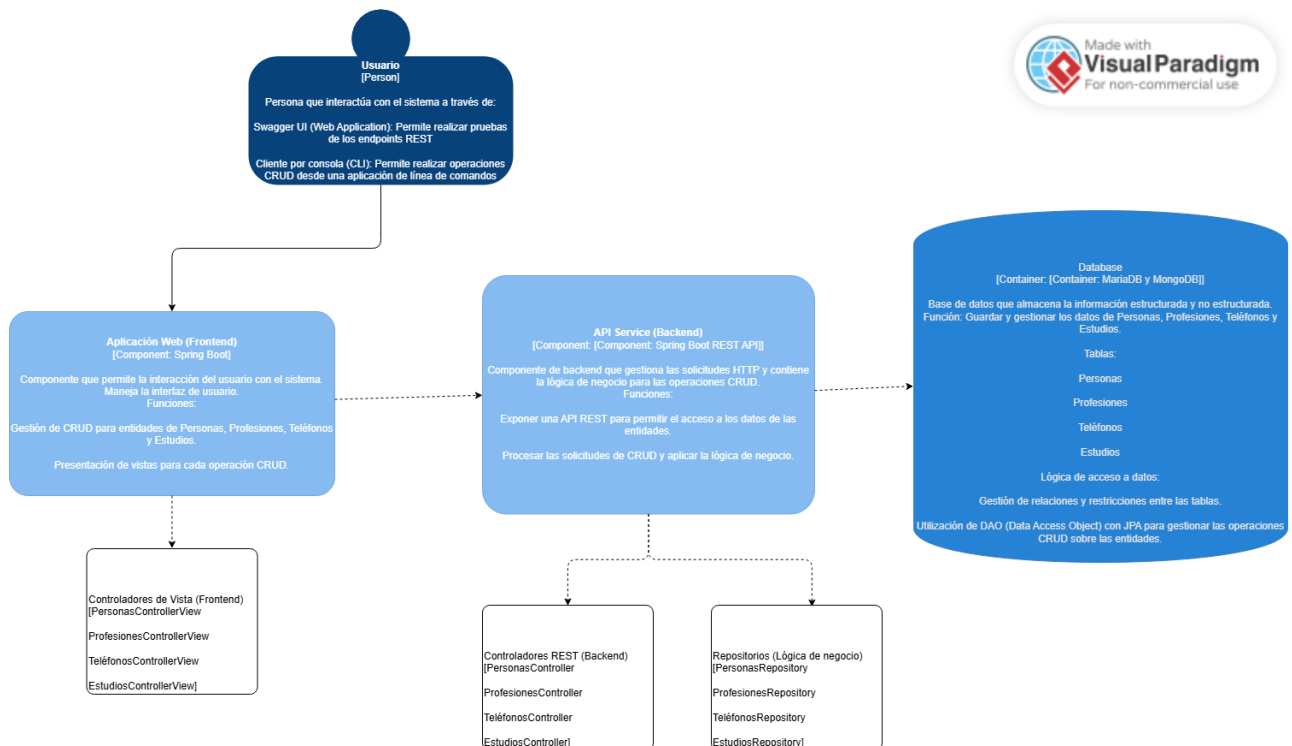


El sistema se basa en una arquitectura hexagonal, lo que permite una separación clara entre la lógica de negocio y las interfaces externas. Para la exposición de servicios, se implementa una API REST que facilita la comunicación con los clientes. El desarrollo se apoya en el framework Spring Boot, y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

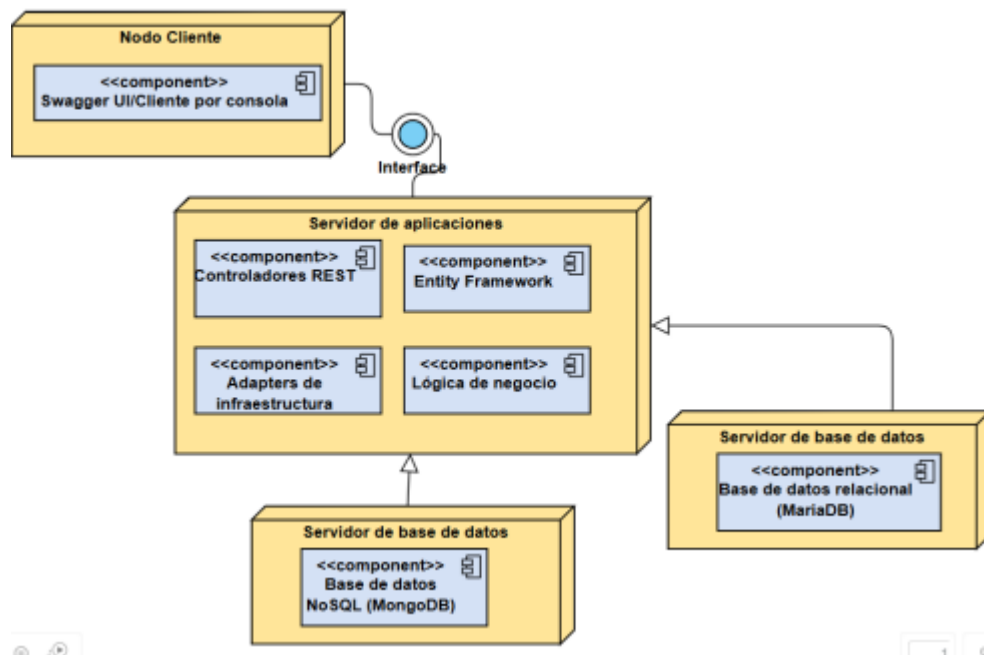
- Diagrama de contenedores C4



- Diagrama de componentes C4

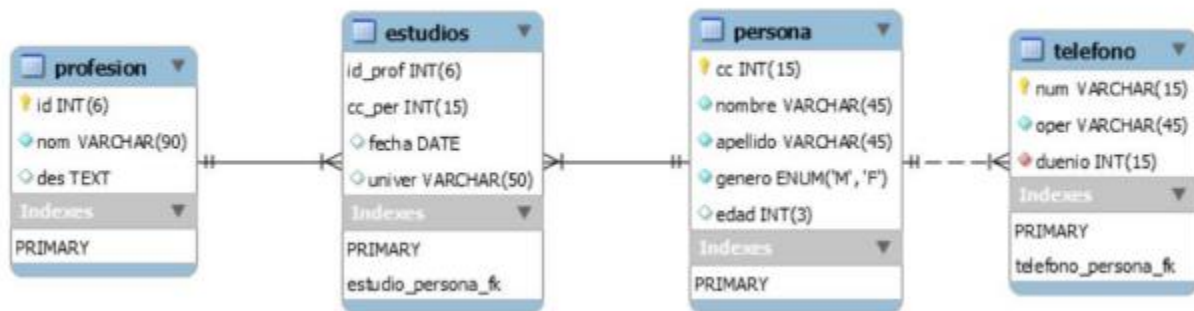


- Diagrama de despliegue C4



- 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.



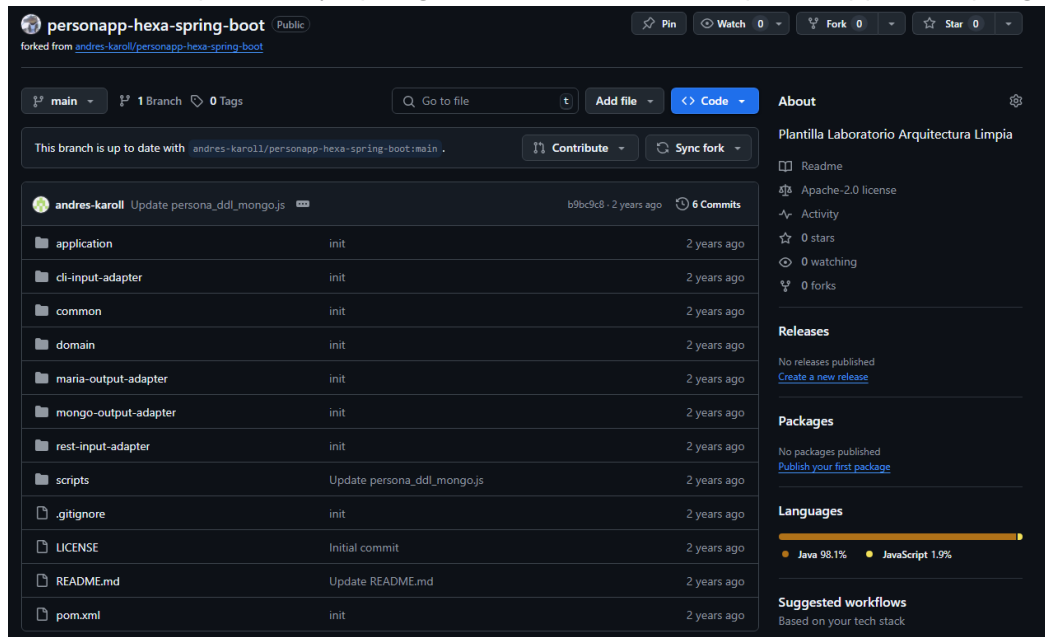
Relaciones entre las Entidades

- 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).

Procedimiento

- **Creación del proyecto en springboot**

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

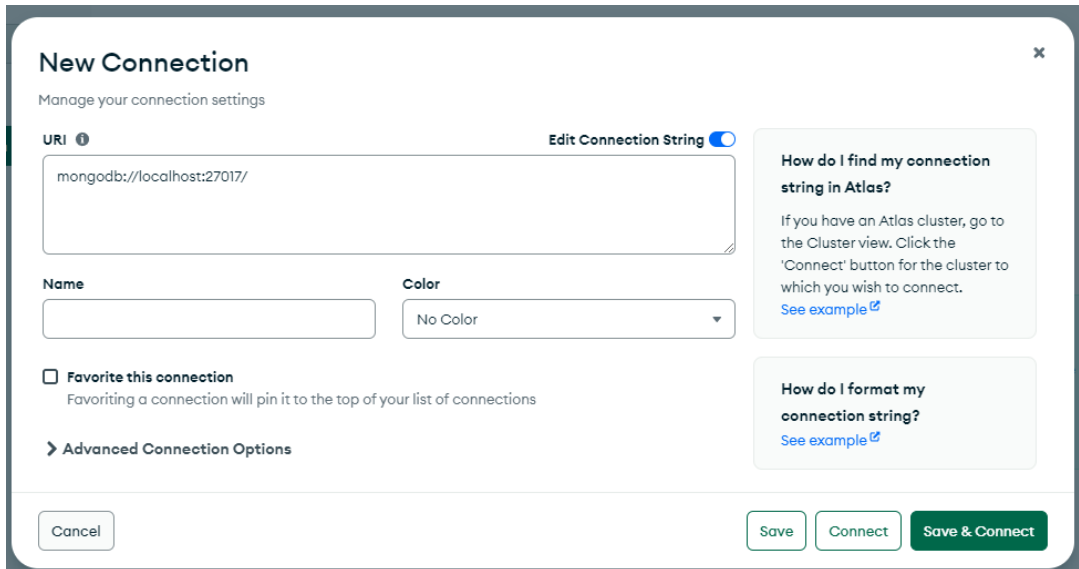


- **Instalar mongo db**



Se accede al sitio web oficial de MongoDB para proceder con la descarga de la versión correspondiente, en este caso la 8.0.9. Allí se selecciona el sistema operativo utilizado y el instalador en formato MSI. Posteriormente, se instala también el Shell de MongoDB y se opta por incluir MongoDB Compass como herramienta gráfica complementaria.



Posteriormente, se crea una nueva conexión por defecto con el puerto 27017



New Connection
Manage your connection settings

URI  Edit Connection String 

mongodb://localhost:27017/

Name

Color

☐ Favorite this connection
Favoriting a connection will pin it to the top of your list of connections

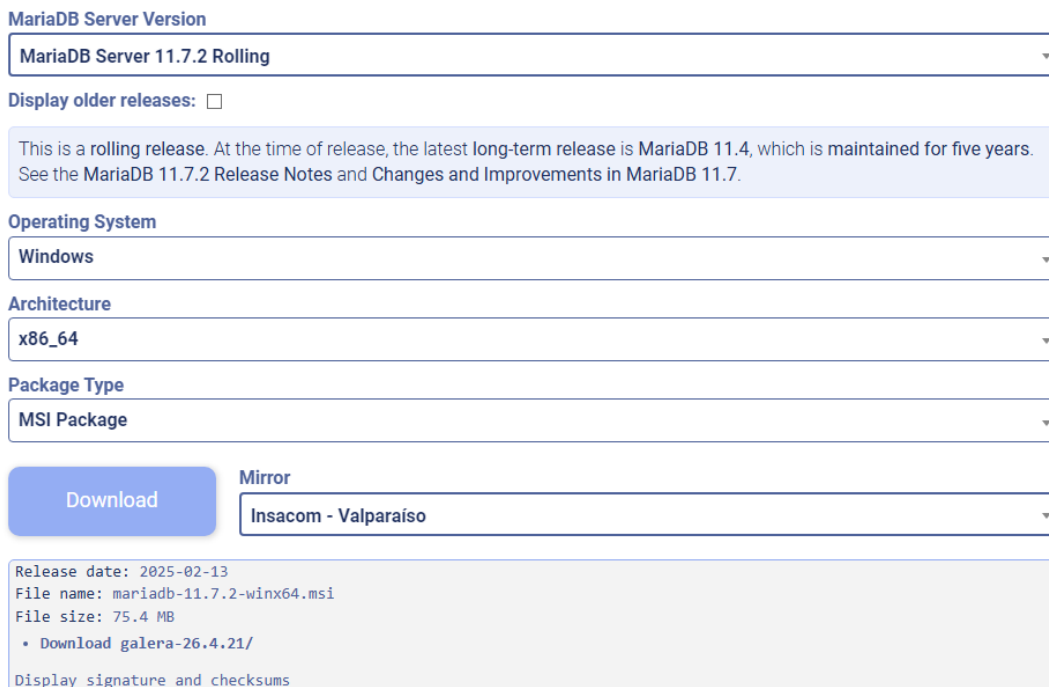
[Advanced Connection Options](#)

[How do I find my connection string in Atlas?](#)
If you have an Atlas cluster, go to the Cluster view. Click the 'Connect' button for the cluster to which you wish to connect.
[See example](#)

[How do I format my connection string?](#)
[See example](#)

- **Instalar maria db**

Para trabajar con MariaDB, se utiliza **HeidiSQL**, una herramienta gratuita que facilita la conexión con diversas bases de datos, incluyendo MariaDB. Se elige la versión adecuada según el sistema operativo y se descarga el instalador en formato MSI.



MariaDB Server Version

MariaDB Server 11.7.2 Rolling

Display older releases: ☐

This is a rolling release. At the time of release, the latest long-term release is MariaDB 11.4, which is maintained for five years. See the MariaDB 11.7.2 Release Notes and Changes and Improvements in MariaDB 11.7.

Operating System

Windows

Architecture

x86_64

Package Type

MSI Package

Mirror

Insacom - Valparaíso

Release date: 2025-02-13
File name: mariadb-11.7.2-winx64.msi
File size: 75.4 MB
• [Download galera-26.4.21/](#)
[Display signature and checksums](#)

Tras completar la descarga del archivo MSI, se procede a su ejecución y se siguen los pasos indicados por el asistente de instalación. Posteriormente, se configura una contraseña y se asigna un puerto específico para el servicio.

- **Creación de la base de datos**

En este proyecto, se emplea Docker Compose para configurar y automatizar la puesta en marcha de las bases de datos MongoDB y MariaDB, garantizando que ambas estén correctamente inicializadas y preparadas para gestionar datos y ejecutar operaciones CRUD.

- **Base de datos MongoDB**

En la configuración del contenedor de MongoDB se especifican el nombre de la base de datos, el usuario y la contraseña directamente en el archivo docker-compose.yml. Además, se emplean archivos JavaScript de inicialización llamados persona_ddl_mongo.js y persona_dml_mongo.js, los cuales contienen los comandos necesarios para crear tanto la estructura de la base de datos como los datos iniciales. Estos archivos se colocan en la ruta especial /docker-entrypoint-initdb.d/ dentro del contenedor, lo que permite que se ejecuten automáticamente en el primer arranque del mismo. Gracias a este proceso, al iniciar el contenedor, se garantiza la creación de la base de datos persona_db junto con la colección persona, y se carga la información necesaria para que MongoDB quede listo para ejecutar las operaciones previstas en los endpoints de la aplicación.

```
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 })

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"]
})
```

- **Base de datos MariaDB**

En la configuración del contenedor de MariaDB, se define la base de datos llamada persona_db, junto con las credenciales del usuario raíz, directamente en el archivo docker-compose.yml. Además, se incorporan dos archivos SQL (persona_ddl_maria.sql y persona_dml_maria.sql) destinados a la creación de la estructura de datos y a la carga de información inicial. Estos archivos se montan en el directorio especial docker-entrypoint-initdb.d, el cual permite a MariaDB ejecutar automáticamente dichos scripts al iniciar el contenedor por primera vez. El archivo persona_ddl_maria.sql contiene instrucciones DDL (Lenguaje de Definición de Datos) para la creación de las tablas que conforman el modelo, como profesion, estudios, persona y telefono. En complemento, persona_dml_maria.sql incluye sentencias DML (Lenguaje de Manipulación de Datos) necesarias para insertar los registros base en esas tablas.

```

FLUSH PRIVILEGES;
--
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;

```

```

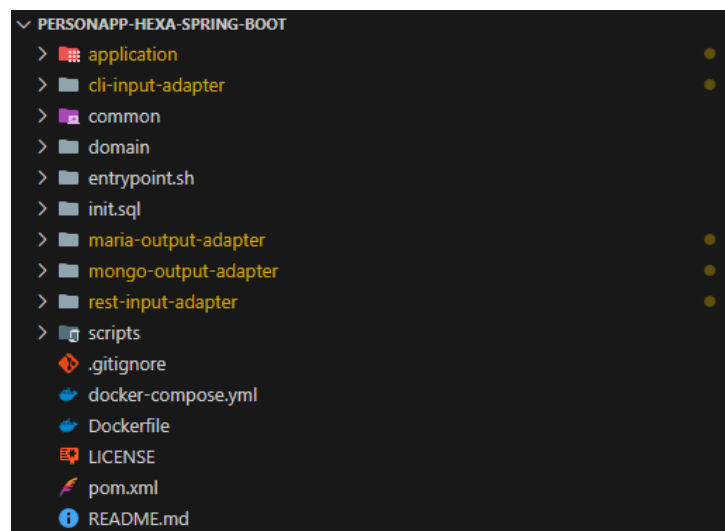
INSERT INTO `persona_db`.`persona` (`cc`, `nombre`, `apellido`, `genero`, `edad`)
VALUES
(123456789, 'Pepe', 'Perez', 'M', 30),
(987654321, 'Pepito', 'Perez', 'M', null),
(321654987, 'Pepa', 'Juarez', 'F', 30),
(147258369, 'Pepita', 'Juarez', 'F', 10),
(963852741, 'Fede', 'Perez', 'M', 18);

```

- **Estructura de archivos del proyecto**

El proyecto se encuentra dividido en los siguientes componentes:

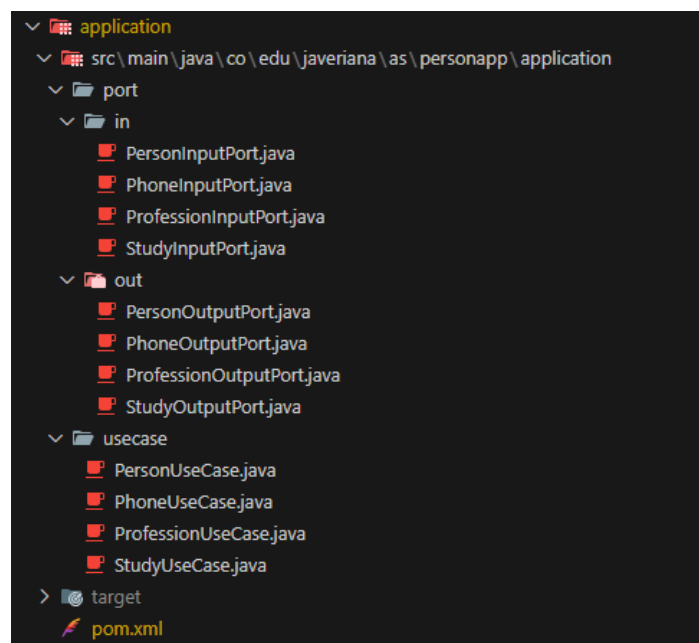
- application: Encargado de la lógica principal del negocio y la implementación de los casos de uso.
- cli-input-adapter: Responsable de la interacción con el usuario mediante la línea de comandos.
- common: Contiene clases auxiliares y recursos reutilizables entre módulos.
- domain: Define las entidades clave del modelo de dominio.
- maria-output-adapter: Se encarga del almacenamiento de datos en una base de datos MariaDB.
- mongo-output-adapter: Administra la persistencia utilizando MongoDB.
- rest-input-adapter: Facilita la comunicación a través de servicios REST.
- scripts: Incluye los scripts necesarios para la configuración e inicialización de las bases de datos.



- **Lógica de Negocio: InputPorts, OutputPorts y UseCases**

El módulo application representa el núcleo de la lógica de negocio de la aplicación. En él se definen las interfaces denominadas puertos de entrada (InputPorts) y puertos de salida (OutputPorts), las cuales permiten aislar la lógica de negocio de los detalles técnicos como la persistencia o los controladores. Además, se implementan los casos de uso (UseCases), los cuales orquestan las acciones del sistema de acuerdo con las reglas del negocio.

Para el caso de la entidad Person, la interfaz PersonInputPort actúa como el canal principal mediante el cual los controladores o adaptadores de entrada solicitan operaciones relacionadas con dicha entidad (como crear, actualizar o eliminar personas). La clase PersonUseCase contiene la lógica específica de estas operaciones y se apoya en la interfaz PersonOutputPort para delegar la persistencia de los datos. Esto garantiza una arquitectura limpia y desacoplada: los casos de uso no dependen de detalles de implementación como bases de datos o frameworks.

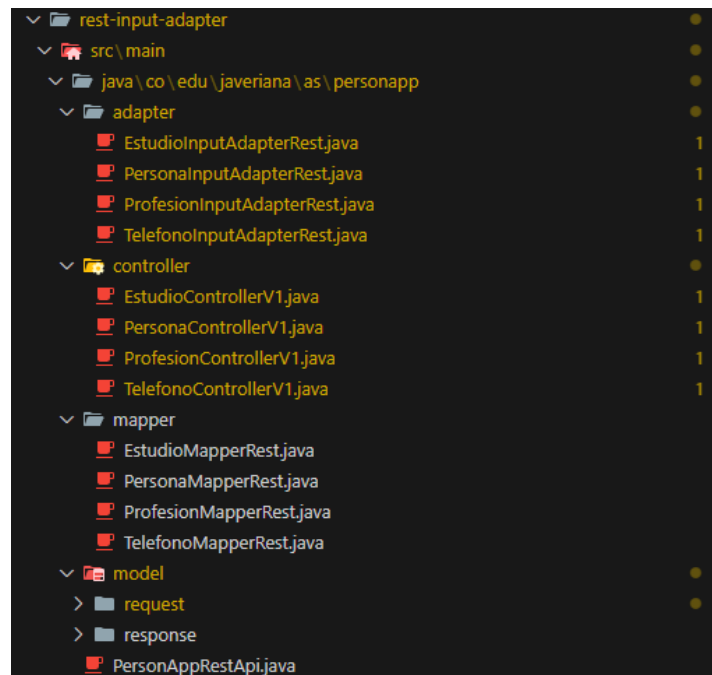


Gracias a este diseño, la lógica del sistema es fácilmente testeable, escalable y puede adaptarse a nuevas tecnologías sin reescribir el núcleo de la aplicación.

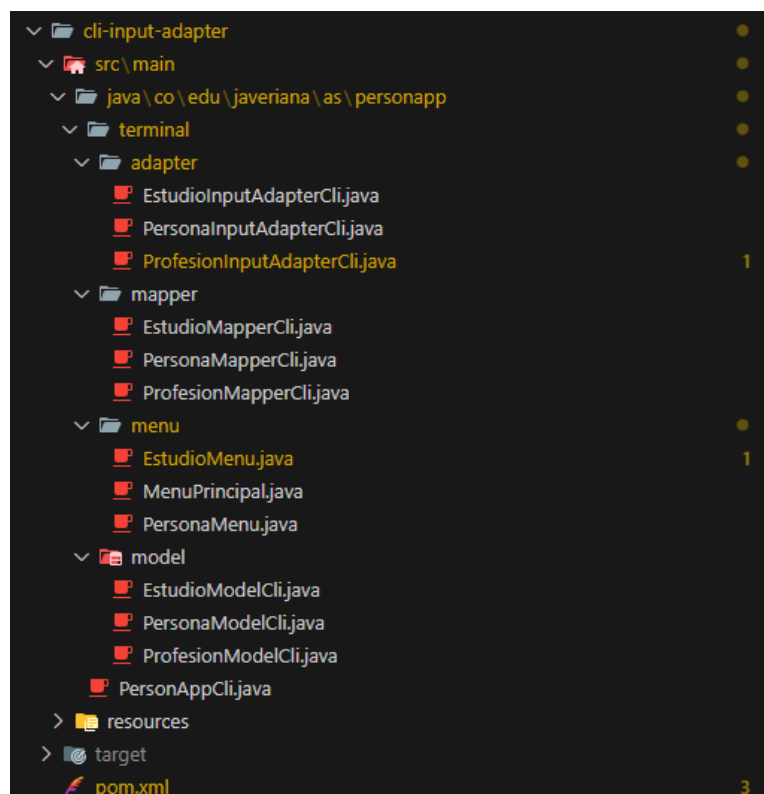
- **Adaptadores de Entrada: Interfaces REST y Línea de Comandos (CLI)**

Los módulos rest-input-adapter y cli-input-adapter funcionan como canales de entrada a la lógica del sistema, permitiendo diferentes formas de interacción con el usuario.

En el módulo REST, se encuentran los controladores REST encargados de exponer los endpoints HTTP. Cada entidad del sistema cuenta con su propio controlador, como `PersonaControllerV1`, `EstudioControllerV1`, `ProfesionControllerV1` y `TelefonoControllerV1`. Estos controladores permiten realizar operaciones CRUD mediante solicitudes HTTP estándar (GET, POST, PUT, DELETE), y actúan como puente entre el mundo exterior y los casos de uso definidos en application.



En el módulo CLI, se implementa una interfaz de texto interactiva, pensada para ejecutar comandos desde la terminal. La clase principal, `PersonAppCli`, permite al usuario realizar operaciones CRUD sin necesidad de una interfaz gráfica. La estructura interna del módulo incluye subdirectorios como `adapter`, `mapper`, `menu` y `model`, los cuales organizan la lógica, navegación y visualización de las opciones en consola, manteniendo una clara separación de responsabilidades y favoreciendo la mantenibilidad del sistema.

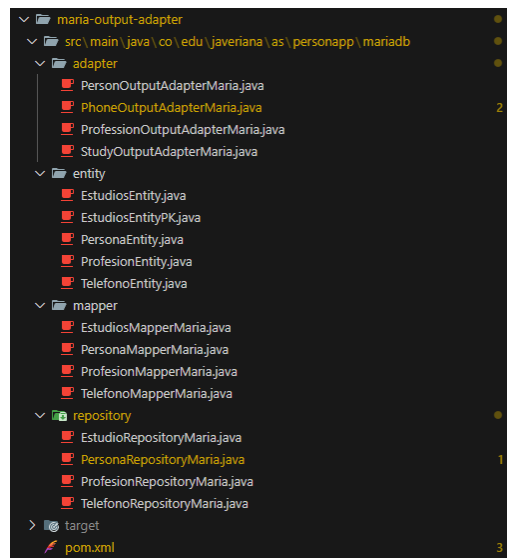


Este enfoque multi-interfaz facilita que la aplicación sea utilizada tanto desde interfaces gráficas (como clientes REST) como desde entornos más técnicos o automatizados (como scripts por consola).

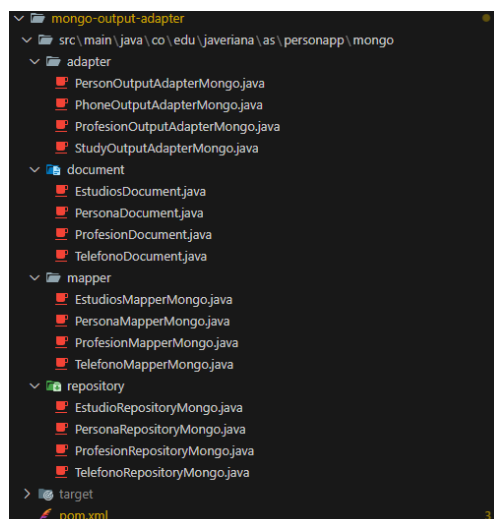
- **Adaptadores de Salida: Persistencia con MariaDB y MongoDB**

Los módulos maria-output-adapter y mongo-output-adapter contienen la lógica para la persistencia de datos, es decir, el almacenamiento y recuperación de información desde las bases de datos MariaDB y MongoDB respectivamente.

En el módulo MariaDB, los adaptadores como PersonOutputAdapterMaria implementan la interfaz PersonOutputPort utilizando el motor de persistencia JpaRepository. Los datos se almacenan en tablas relacionales definidas mediante entidades ubicadas en el subdirectorio entity. A su vez, los archivos dentro de mapper transforman los datos entre el modelo de dominio y el modelo relacional, y los componentes en repository gestionan las operaciones JPA como guardar, actualizar, buscar o eliminar registros.



En el módulo MongoDB, se sigue una estructura análoga, aunque adaptada a un modelo de base de datos NoSQL. Aquí, PersonOutputAdapterMongo implementa los métodos de PersonOutputPort haciendo uso de MongoRepository. Las clases en document representan las colecciones de MongoDB, los mappers convierten documentos a objetos de dominio, y los repositorios permiten acceder a la base de datos de forma sencilla y tipada.



Este enfoque dual permite que el sistema sea flexible y pueda adaptarse a diferentes necesidades tecnológicas, manteniendo la lógica de negocio independiente del tipo de base de datos utilizada.

- **Configuración de Bases de Datos y Despliegue con Docker**

La inicialización y despliegue del entorno completo se encuentra gestionado en el módulo scripts y el archivo docker-compose.yml.

- Dentro de scripts, se ubican los archivos necesarios para preparar las bases de datos en el arranque del sistema:
 - persona_ddl_maria.sql contiene las sentencias SQL para crear las tablas y cargar datos iniciales en MariaDB.

```
# Servicio de base de datos MariaDB
> Run Service
maria-db:
  image: mariadb:latest # Imagen oficial de MariaDB
  container_name: mariadb # Nombre personalizado del contenedor
  ports:
    - "3306:3306" # Puerto por defecto de MariaDB
  environment: # Variables necesarias para inicializar la base de datos
    MYSQL_ROOT_PASSWORD: example
    MYSQL_DATABASE: persona_db
  volumes:
    - maria_data:/var/lib/mysql # Volumen persistente para los datos de MariaDB
    - ./scripts/persona_ddl_maria.sql:/docker-entrypoint-initdb.d/persona_ddl_maria.sql:ro # Script de estructura (solo lectura)
    - ./scripts/persona_dml_maria.sql:/docker-entrypoint-initdb.d/persona_dml_maria.sql:ro # Script de datos iniciales (solo lectura)
  networks:
    - app_network # Conectado a la misma red que los demás servicios
```

- persona_dml_mongo.js define los documentos y configuraciones iniciales para las colecciones de MongoDB.

```
# Servicio de base de datos MongoDB
> Run Service
mongo-db:
  image: mongo:latest # Imagen oficial de MongoDB
  container_name: mongodb # Nombre personalizado del contenedor
  ports:
    - "27017:27017" # Puerto por defecto de MongoDB
  environment: # Configuración inicial del contenedor Mongo
    MONGO_INITDB_DATABASE: persona_db
    MONGO_INITDB_ROOT_USERNAME: persona_db
    MONGO_INITDB_ROOT_PASSWORD: persona_db
  volumes:
    - mongo_data:/data/db # Volumen persistente para los datos de Mongo
    - ./scripts/persona_ddl_mongo.js:/docker-entrypoint-initdb.d/persona_ddl_mongo.js:ro # Script de estructura (solo lectura)
    - ./scripts/persona_dml_mongo.js:/docker-entrypoint-initdb.d/persona_dml_mongo.js:ro # Script de datos iniciales (solo lectura)
  networks:
    - app_network # Red compartida para conexión con otros servicios
```

Estos archivos son ejecutados automáticamente la primera vez que se levantan los contenedores gracias al uso de la carpeta /docker-entrypoint-initdb.d/.

- El archivo **docker-compose.yml** orquesta todos los servicios del sistema: MariaDB, MongoDB y la aplicación Spring Boot. Define variables de entorno, volúmenes persistentes, puertos de exposición y dependencias entre servicios. Al ejecutar docker-compose up --build, se construye y despliega el entorno completo, dejando las bases de datos preparadas y la aplicación lista para recibir solicitudes REST o comandos desde la terminal.


```

services:
  # Servicio principal de la aplicación REST construida con Spring Boot
  > Run Service
  rest-service:
    build:
      context: . # Ruta al directorio del proyecto (donde se encuentra el Dockerfile)
      dockerfile: Dockerfile # Se especifica que se usará el Dockerfile multietapa
    ports:
      - "8080:8080" # Mapea el puerto 8080 del contenedor al puerto 8080 del host
    depends_on:
      - mongo-db # Este servicio se inicia después de que mongo-db esté listo
      - maria-db # Este servicio se inicia después de que maria-db esté listo
    environment: # Variables de entorno necesarias para la configuración de Spring Boot
      SPRING_DATA_MONGODB_URI: mongodb://persona_db:persona_db@mongo-db:27017/persona_db?authSource=persona_db
      SPRING_DATASOURCE_URL: jdbc:mariaDB://maria-db:3306/persona_db
      SPRING_DATASOURCE_USERNAME: root
      SPRING_DATASOURCE_PASSWORD: example
      SPRING_DATASOURCE_DRIVER_CLASS_NAME: org.mariadb.jdbc.Driver
      SPRING_JPA_PROPERTIES_HIBERNATE_DIALECT: org.hibernate.dialect.MariaDBDialect
      APP_DEBUG_LEVEL: INFO # Nivel de logs de la aplicación
    networks:
      - app_network # Conecta este servicio a la red compartida
    volumes:
      - ./init.sql:/docker-entrypoint-initdb.d/init.sql # (Opcional) Script de inicialización SQL
      - ./entrypoint.sh:/entrypoint.sh # (Opcional) Script personalizado de arranque

```

Conclusiones y Lecciones Aprendidas

La implementación de este sistema utilizando una arquitectura hexagonal permitió lograr una separación clara entre la lógica de negocio y las interfaces externas, lo que facilitó el mantenimiento, la extensibilidad y la prueba independiente de los componentes. Se comprobó que estructurar el proyecto en módulos bien definidos — como frontend, backend, infraestructura y persistencia— mejora notablemente la organización del código y la comprensión del sistema en su conjunto.

Además, integrar tecnologías diversas como MariaDB y MongoDB en un mismo entorno evidenció las ventajas de contar con un núcleo de aplicación desacoplado de las decisiones tecnológicas. Esta flexibilidad se traduce en la posibilidad de reemplazar o extender componentes sin afectar el funcionamiento general del sistema. El uso de Spring Boot resultó ser una herramienta eficaz para acelerar el desarrollo de APIs REST, gracias a su capacidad de configuración automática y soporte integrado para controladores y repositorios.

Durante el proceso, se aprendió la importancia de diseñar la arquitectura antes de programar. La elaboración de diagramas C4 fue fundamental para visualizar las relaciones entre componentes, anticipar desafíos técnicos y comunicar claramente la estructura del sistema. También se reafirmó el valor de herramientas como Docker y docker-compose, que facilitaron la creación de entornos consistentes y reproducibles para las bases de datos.

Finalmente, se reconoció el impacto positivo de documentar los servicios con Swagger, tanto para pruebas rápidas como para alinear el trabajo entre desarrolladores. En conjunto, esta experiencia permitió aplicar en un entorno real los principios de diseño limpio, modularidad, desacoplamiento y buenas prácticas de desarrollo de software, consolidando habilidades clave para proyectos profesionales.

Webgrafía

- Andres-Karoll. (s. f.). GitHub - andres-karoll/personapp-hexa-spring-boot: Plantilla Laboratorio Arquitectura Limpia. GitHub. <https://github.com/andres-karoll/personapphexa-spring-boot>
- Qué es la arquitectura hexagonal, ventajas y desventajas. (2023, 10 mayo). HubSpot. <https://blog.hubspot.es/website/que-es-arquitectura-hexagonal>
- Muñoz, P. H. (2024, 29 octubre). Arquitectura Hexagonal: Qué Es Y Por Qué Es Clave Para Software ágil. Sale Systems. <https://salesystems.es/arquitectura-hexagonal/#:~:text=Lo%20que%20hace%20la%20arquitectura,ambos%20deber%C3%A9%20depende%20de%20abstracciones.>
- <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/>