



Pontificia Universidad  
**JAVERIANA**  
Colombia

## **Laboratorio 2**

**Arquitectura de Software**

**Ing. Andrés Sánchez**

**Adrián Eduardo Ruiz Cerquera**

**Miguel Ángel Márquez Posso**

**Juan Felipe Galvis Vargas**

**Pontificia Universidad Javeriana**

**Bogotá, Colombia**

**Noviembre de 2025**

<b>Marco Conceptual.....</b>	<b>4</b>
<b>Arquitectura Hexagonal.....</b>	<b>4</b>
Definición.....	4
Historia y Evolución.....	4
Ventajas.....	4
Desventajas.....	5
Casos de uso y aplicación.....	5
<b>Patrón Modular.....</b>	<b>5</b>
Definición.....	5
Historia y Evolución.....	6
Ventajas.....	6
Desventajas.....	6
Casos de uso y aplicación.....	6
<b>JDK.....</b>	<b>7</b>
Definición.....	7
Historia y Evolución.....	7
Ventajas.....	7
Desventajas.....	7
Casos de uso y aplicación.....	8
<b>Spring Boot.....</b>	<b>8</b>
Definición.....	8
Historia y Evolución.....	8
Ventajas.....	9
Desventajas.....	9
Casos de uso y aplicación.....	9
<b>MongoDB.....</b>	<b>9</b>
Definición.....	9
Historia y Evolución.....	10
Ventajas.....	10
Desventajas.....	10
Casos de uso y aplicación.....	10
<b>MariaDB.....</b>	<b>11</b>
Definición.....	11
Historia y Evolución.....	11
Ventajas.....	11
Desventajas.....	11
Casos de uso y aplicación.....	12
<b>REST.....</b>	<b>12</b>
Definición.....	12
Historia y Evolución.....	12
Ventajas.....	13
Desventajas.....	13
Casos de uso y aplicación.....	13
<b>CLI.....</b>	<b>13</b>

Definición.....	13
Historia y Evolución.....	14
Ventajas.....	14
Desventajas.....	14
Casos de uso y aplicación.....	14
<b>Swagger.....</b>	<b>15</b>
Definición.....	15
Historia y Evolución.....	15
Ventajas.....	15
Desventajas.....	15
Casos de uso y aplicación.....	16
<b>Docker.....</b>	<b>16</b>
Definición.....	16
Historia y Evolución.....	16
Ventajas.....	16
Desventajas.....	17
Casos de uso y aplicación.....	17
<b>Diseño.....</b>	<b>17</b>
Diagrama Alto Nivel.....	17
Diagrama de Contexto.....	18
Diagrama de Contenedores.....	19
Diagrama de Componentes.....	20
Entidades.....	21
<b>Procedimiento.....</b>	<b>22</b>
Fork del repositorio.....	22
Arquitectura Hexagonal.....	23
Estructura Modular con Maven.....	23
Módulo Common.....	24
Módulo Domain.....	24
Módulo Application.....	25
Módulo Maria-Output-Adapter.....	26
Módulo Mongo-Output-Adapter.....	27
Módulo CLI-Input-Adapter.....	28
Módulo REST-Input-Adapter.....	29
Documentación de API con Swagger/OpenAPI.....	30
Configuración de Base de Datos.....	30
Dockerfile y Construcción de Contenedores.....	31
Docker Compose para Despliegue.....	32
Uso de Swagger.....	34
Uso de CLI.....	38
Repositorio.....	43
<b>Conclusiones y Lecciones Aprendidas.....</b>	<b>43</b>
<b>Referencias.....</b>	<b>44</b>

## Marco Conceptual

### Arquitectura Hexagonal

#### Definición

La arquitectura hexagonal, también conocida como Ports and Adapters, es un estilo arquitectónico cuyo objetivo principal es aislar completamente la lógica de negocio del resto de elementos externos, como bases de datos, interfaces de usuario, redes o frameworks. Este aislamiento se logra definiendo puertos (interfaces) que representan los puntos formales de comunicación con el dominio, y adaptadores que implementan dichas interfaces para conectar el núcleo con tecnologías específicas. El dominio no conoce las herramientas que lo rodean; solo define reglas y casos de uso encapsulados en un modelo independiente.

Esta arquitectura establece un diseño altamente desacoplado que facilita la mantenibilidad, la testabilidad y la evolución del software. Además, mejora la capacidad de sustituir tecnologías sin afectar el núcleo funcional, ya que los detalles externos se manejan como "periféricos" intercambiables. Su enfoque parte de la idea de que las reglas de negocio deben tener mayor estabilidad que las decisiones tecnológicas, por lo que estas últimas deben permanecer fuera del corazón del sistema.

#### Historia y Evolución

El concepto fue introducido por Alistair Cockburn a mediados de la década de 2000, como respuesta a los problemas que presentaban las arquitecturas en capas tradicionales, en especial su dependencia en frameworks de presentación o persistencia. Cockburn propuso una arquitectura donde el flujo de dependencia se invierte y se garantiza que el centro no dependa de los bordes. Esta idea se relaciona con la inversión de dependencias de Robert C. Martin, el patrón Dependency Inversion y los principios SOLID.

Con el paso del tiempo, la arquitectura hexagonal influyó en otros modelos estructurales como Onion Architecture (Jeffrey Palermo) y posteriormente Clean Architecture (Robert C. Martin), ambos con el mismo enfoque de dominio independiente. Con la llegada de microservicios alrededor de 2012–2015, la arquitectura hexagonal ganó mayor protagonismo porque facilitaba construir servicios autónomos y adaptables al cambio tecnológico. Actualmente es ampliamente utilizada en sistemas que requieren evolución a largo plazo, pruebas automatizadas y fuerte separación entre negocio e infraestructura.

#### Ventajas

- Bajo acoplamiento al framework y la infraestructura: se puede cambiar de base de datos o de medio de exposición (por ejemplo, de REST a CLI) sin tocar la lógica de negocio.
- Alta testabilidad: como el dominio depende de puertos (interfaces), es fácil usar *mocks* para pruebas unitarias.

- Claridad de responsabilidades: el dominio se concentra en las reglas de negocio; los adaptadores se encargan de detalles técnicos.
- Facilita la evolución tecnológica: permite sustituir adaptadores (por ejemplo, cambiar de MongoDB a MariaDB).

## Desventajas

- Mayor complejidad inicial: para proyectos muy pequeños puede sentirse “sobreingeniería”.
- Más clases y capas: hay más interfaces y adaptadores, lo cual puede aumentar el código “ceremonial”.
- Curva de aprendizaje: se necesita entender bien puertos/adaptadores para usarla correctamente.

## Casos de uso y aplicación

- Sistemas donde se espera longevidad y alta mantenibilidad (banca, salud, educación, etc.).
- Proyectos que requieren cambiar tecnologías de persistencia o protocolos de comunicación.
- Aplicaciones con lógica de negocio compleja que debe estar bien aislada del resto.
- Microservicios que quieren mantener un dominio limpio, independiente de frameworks.

## **Patrón Modular**

### Definición

El patrón modular es una aproximación estructural que organiza una aplicación en unidades lógicas independientes llamadas módulos. Cada módulo encapsula su propia lógica, datos e interfaces públicas, lo que permite construir sistemas más organizados, mantenibles y con límites funcionales claros. La modularidad busca reducir el acoplamiento y aumentar la cohesión mediante una separación explícita y bien definida entre las responsabilidades de cada componente.

En términos arquitectónicos, un sistema modular facilita la división del código en secciones autónomas que pueden evolucionar, probarse y desplegarse de manera independiente, especialmente en arquitecturas modernas como monolitos modulares o microservicios. La modularidad también puede estar soportada por mecanismos del lenguaje, como el Java Platform Module System (JPMS), que permite declarar dependencias entre módulos y restringir qué partes de un módulo son visibles externamente.

## Historia y Evolución

La idea de modularidad tiene sus raíces en los inicios de la ingeniería de software, cuando surgió como respuesta a sistemas monolíticos difíciles de mantener en las décadas de 1960 y 1970. Investigadores como David Parnas promovieron el concepto de ocultamiento de información, proponiendo que los módulos debían definirse por decisiones de diseño susceptibles de cambio. Este pensamiento evolucionó hacia metodologías orientadas a objetos, donde las clases y paquetes representaban unidades básicas de modularidad.

Con el crecimiento del software empresarial durante los años 1990 y 2000, la modularidad ganó importancia para manejar la complejidad creciente. En Java, inicialmente se utilizaban paquetes y convenciones para organizar módulos, pero no existía un sistema formal hasta la introducción de JPMS en Java 9 (2017). A nivel arquitectónico, surgieron enfoques más elaborados como monolitos modulares, Domain-Driven Design y microservicios, que dependen fuertemente de la modularidad para definir límites contextuales y aislar cambios en funcionalidades.

## Ventajas

- Mantenimiento más sencillo: es más fácil localizar y cambiar funcionalidades.
- Reutilización de código: módulos reutilizables entre proyectos.
- Escalabilidad del equipo: distintos equipos pueden trabajar en módulos distintos.
- Menor impacto de cambios: un cambio en un módulo no rompe todo el sistema si las interfaces se respetan.

## Desventajas

- Diseño inicial más complicado: hay que pensar bien los límites de cada módulo.
- Posible sobre-fragmentación: demasiados módulos pueden complicar la gestión del proyecto.
- Configuración más compleja en el build (Maven/Gradle) y en el despliegue si no se maneja bien.

## Casos de uso y aplicación

- Proyectos medianos o grandes donde haya muchos equipos o funcionalidades.
- Arquitecturas orientadas a microservicios o monolitos modulares.
- Sistemas que necesitan versiones o despliegues independientes de partes del sistema.

# JDK

## Definición

El Java Development Kit es el conjunto de herramientas oficial para desarrollar, compilar y ejecutar aplicaciones escritas en Java. Incluye el compilador Java, el Java Virtual Machine, las bibliotecas estándar y diversas herramientas esenciales como depuradores, generadores de documentación, analizadores y utilidades de empaquetamiento. El JDK proporciona el entorno completo necesario para transformar código fuente en bytecode ejecutable sobre la JVM.

Además de las herramientas de compilación, el JDK define la estructura fundamental del ecosistema Java al proveer las APIs centrales, tales como colecciones, concurrencia, entrada/salida, redes, seguridad y herramientas criptográficas. También establece el comportamiento de interoperabilidad entre diferentes versiones del lenguaje, el estándar de ejecución y la compatibilidad hacia atrás que caracteriza a la plataforma Java.

## Historia y Evolución

El JDK fue lanzado por Sun Microsystems en 1996 con Java 1.0, surgiendo como una plataforma multiplataforma orientada al desarrollo de aplicaciones empresariales y de internet. A lo largo de las versiones 1.1 a 1.4 se consolidaron características esenciales como la introspección, la recolección de basura mejorada y el modelo de eventos. En 2004, Java 5 marcó un punto importante con la introducción de genéricos, anotaciones y mejoras en concurrencia.

En 2010, Oracle adquirió Sun Microsystems, lo que llevó a una reestructuración del ciclo de releases y a la apertura de gran parte de Java como software libre a través de OpenJDK. A partir de Java 9, se introdujo el Java Platform Module System, un cambio significativo que permitió construir aplicaciones más modulares. Desde entonces, Java adoptó un ciclo de lanzamiento semestral, lo que agiliza la incorporación de nuevas características y correcciones manteniendo la estabilidad del ecosistema.

## Ventajas

- Multiplataforma: “write once, run anywhere”.
- Ecosistema maduro: muchísimas librerías, frameworks y herramientas.
- Rendimiento aceptable gracias a la JVM y la compilación JIT.
- Compatibilidad hacia atrás: mucho cuidado con no romper código antiguo.

## Desventajas

- Mayor consumo de memoria que lenguajes interpretados más ligeros.
- Curva de aprendizaje algo más larga que lenguajes muy sencillos (por ejemplo, Python para principiantes).
- La diversidad de versiones (Java 8, 11, 17, etc.) puede generar problemas de compatibilidad si no se gestiona bien.

## Casos de uso y aplicación

- Aplicaciones empresariales de gran tamaño.
- Backend de sistemas bancarios, salud, comercio electrónico.
- Servicios web, microservicios, aplicaciones de escritorio y sistemas embebidos específicos que soporten JVM.

## Spring Boot

### Definición

Spring Boot es un framework diseñado para simplificar la creación de aplicaciones Java basadas en Spring, proporcionando un mecanismo de configuración automática que reduce la necesidad de configuraciones manuales o archivos XML extensos. Incluye starters preconfigurados, un servidor embebido y un conjunto de convenciones que permiten desplegar aplicaciones listas para producción de manera rápida. Su objetivo es facilitar el desarrollo de servicios y aplicaciones empresariales altamente configurables sin perder flexibilidad.

El framework abstrae gran parte de la complejidad del ecosistema Spring mediante auto-configuración, detección automática de dependencias y herramientas como Spring Actuator para monitoreo y métricas. Esto permite que los desarrolladores se concentren en la lógica del negocio mientras el framework gestiona aspectos como seguridad, persistencia, manejo de dependencias e integración con servicios externos.

### Historia y Evolución

Spring Framework surgió a inicios de los años 2000 como una alternativa ligera a Java EE. Sin embargo, su configuración manual mediante XML podía ser extensa. Spring Boot fue presentado en 2014 como una extensión que ofrecía auto-configuración y un conjunto de herramientas destinadas a acelerar el desarrollo. Su aparición coincidió con el auge de microservicios, lo que llevó a adoptarlo masivamente en la industria.

Con el tiempo, Spring Boot integró soporte para Spring Cloud, métricas avanzadas, integración con contenedores y un ciclo de releases más rápido. Sus versiones modernas incluyen mejoras en rendimiento, soporte nativo para imágenes de contenedores, configuración reactiva y optimizaciones para entornos en la nube. Actualmente es uno de los frameworks más utilizados en el desarrollo backend empresarial.

## Ventajas

- Arranque rápido de proyectos: crear un proyecto web REST puede tomar minutos.
- Gran ecosistema: soporte para casi cualquier tecnología que necesites.
- Configuración automática: reduce el “boilerplate”.
- Actuator: endpoints para monitoreo, métricas, health checks.
- Excelente integración con arquitecturas como Hexagonal y DDD.

## Desventajas

- Curva de aprendizaje si se explora todo el ecosistema (Spring Data, Security, Cloud, etc.).
- La configuración automática puede esconder detalles que a veces es importante entender.
- Las aplicaciones pueden volverse pesadas si se incluyen demasiados starters sin control.

## Casos de uso y aplicación

- Microservicios REST en Java.
- APIs para aplicaciones móviles y web.
- Backend de sistemas empresariales.
- Proyectos académicos donde se quiere practicar buenas prácticas modernas en Java.

## MongoDB

### Definición

MongoDB es un sistema de gestión de bases de datos NoSQL orientado a documentos. En lugar del modelo relacional clásico, utiliza documentos en formato BSON (una representación binaria de JSON) para almacenar datos de manera flexible y semi-estructurada. Esto permite manejar estructuras jerárquicas, arreglos y objetos embebidos sin requerir un esquema rígido, facilitando la evolución del modelo de datos conforme cambian las necesidades.

La base de datos proporciona un conjunto de características avanzadas como consultas de agregación, índices especializados, replicación, particionamiento y almacenamiento distribuido. Su enfoque en la flexibilidad lo hace ideal para cargas de trabajo que requieren rapidez en escritura, adaptación frecuente del esquema y escalabilidad horizontal.

## Historia y Evolución

MongoDB fue creado por 10gen (hoy MongoDB Inc.) en 2009 como parte de una plataforma de hosting que requería una base de datos más flexible que las relacionales tradicionales. Su diseño se apoyó en tendencias emergentes como bases de datos distribuidas, Big Data y modelos orientados a documentos. A medida que las aplicaciones web comenzaron a manejar datos cada vez más variados, MongoDB ganó popularidad rápidamente.

Con el tiempo, incorporó funcionalidades críticas como índices compuestos, replicación automática, sharding distribuido y posteriormente transacciones ACID multi-documento (2018). Su ecosistema se expandió con herramientas como Atlas (servicio en la nube), drivers oficiales para múltiples lenguajes y características de seguridad ampliadas. Hoy es una de las bases NoSQL más utilizadas en el mundo.

## Ventajas

- Esquema flexible: ideal cuando la estructura de datos cambia con frecuencia.
- Escalabilidad horizontal mediante sharding.
- Buen soporte para altas tasas de lectura y escritura.
- Modelo de documentos muy cercano a los objetos de las aplicaciones.

## Desventajas

- No es la mejor opción para transacciones complejas multi-tabla (aunque ha mejorado con el tiempo).
- Requiere diseñar cuidadosamente los documentos para evitar duplicación excesiva.
- Menos adecuado cuando se necesita un modelo fuertemente relacional.

## Casos de uso y aplicación

- Aplicaciones web con datos semi-estructurados (por ejemplo, perfiles de usuario, contenidos, logs).
- Sistemas de analítica ligera o almacenamiento de eventos.
- Proyectos donde se quiere iterar rápido en el modelo de datos.

# **MariaDB**

## Definición

MariaDB es un sistema de gestión de bases de datos relacional (RDBMS) derivado directamente de MySQL. Funciona con el modelo tradicional basado en tablas, filas y columnas, y utiliza SQL como lenguaje de consulta estándar. MariaDB mantiene compatibilidad con el ecosistema MySQL pero incluye mejoras en rendimiento, seguridad y motores de almacenamiento alternativos que lo diferencian de su predecesor.

El énfasis de MariaDB está en mantener un sistema completamente abierto, estable y adecuado tanto para aplicaciones pequeñas como para sistemas transaccionales complejos. Su arquitectura permite el uso de múltiples motores de almacenamiento como InnoDB, Aria y ColumnStore, adaptándose a diferentes tipos de cargas de trabajo.

## Historia y Evolución

MariaDB nació en 2009 cuando Oracle adquirió Sun Microsystems, empresa propietaria de MySQL. Los creadores originales de MySQL, liderados por Michael Widenius, decidieron bifurcar el proyecto para asegurar una alternativa totalmente abierta y libre de posibles restricciones comerciales. Desde entonces, MariaDB ha evolucionado con una comunidad activa y con una fundación dedicada al mantenimiento del proyecto.

Durante los años siguientes, MariaDB incorporó nuevas características como motores optimizados para analítica, mejoras en replicación, optimizaciones de consultas, encriptación nativa y compatibilidad extendida. Hoy en día es ampliamente utilizada en entornos empresariales, servidores web y plataformas cloud que requieren fiabilidad y alto rendimiento.

## Ventajas

- Compatibilidad alta con MySQL (en muchos casos se puede migrar casi sin cambios).
- Modelo relacional robusto, ideal para datos bien estructurados.
- Soporta transacciones ACID y restricciones de integridad (foreign keys, etc.).
- Ecosistema maduro: herramientas, drivers, soporte en frameworks (Spring Data, etc.).

## Desventajas

- Menos flexible que una base NoSQL cuando el esquema cambia muy rápido.
- Escalado horizontal más complejo que en algunas bases NoSQL (aunque se puede con réplicas y particiones).
- Requiere un diseño cuidadoso del esquema para evitar problemas de rendimiento.

## Casos de uso y aplicación

- Aplicaciones con modelo relacional claro: facturación, inventarios, registros clínicos estructurados, etc.
- Sistemas que necesitan consistencia fuerte y restricciones de integridad.
- Proyectos que ya usan MySQL pero quieren una alternativa open source sostenida por la comunidad.

## REST

### Definición

REST, o Representational State Transfer, es un estilo arquitectónico para diseñar servicios web basados en recursos que se manipulan mediante operaciones estándares del protocolo HTTP. En REST, cada recurso se identifica mediante una URI, y las interacciones se realizan utilizando métodos como GET, POST, PUT y DELETE. El enfoque REST busca simplicidad, escalabilidad y un diseño uniforme que permita a sistemas heterogéneos comunicarse sin acoplamiento fuerte.

Además, REST promueve el uso de representaciones de recursos, típicamente en formatos como JSON o XML, y exige que la comunicación sea sin estado, lo que significa que el servidor no mantiene información de sesiones entre peticiones. Esto facilita la distribución de cargas y mejora la tolerancia a fallos en sistemas distribuidos.

### Historia y Evolución

REST fue definido en el año 2000 por Roy Fielding en su tesis doctoral, como un conjunto de principios para describir la arquitectura de la World Wide Web. En su origen, REST no fue concebido como un estándar de API, sino como una forma de entender cómo la web conseguía resiliencia, escalabilidad y simplicidad a gran escala. Durante los primeros años de los 2000, las empresas utilizaban principalmente SOAP y servicios web basados en XML, pero REST comenzó a ganar adopción por su menor complejidad.

A partir de 2010, con la expansión de aplicaciones móviles y APIs públicas, REST se convirtió en el estilo predominante para diseñar interfaces entre servicios. La comunidad desarrolló mejores prácticas, herramientas de documentación y formatos ligeros como JSON que consolidaron su adopción. Aunque hoy existen alternativas como GraphQL y gRPC, REST sigue siendo la opción dominante para servicios web abiertos y microservicios.

## Ventajas

- Simplicidad: HTTP + JSON es fácil de entender y usar.
- Buena compatibilidad con cualquier lenguaje y plataforma.
- Facilita la escalabilidad al ser stateless.
- Amplia adopción y documentación.

## Desventajas

- No es un estándar formal como tal (es un estilo), entonces hay muchas “variaciones” de cómo se implementa.
- No siempre es la mejor opción para comunicación muy compleja o streaming continuo (donde gRPC o websockets pueden ser mejores).
- Manejo de errores y versionamiento de APIs requiere buenas prácticas.

## Casos de uso y aplicación

- APIs públicas de servicios web (redes sociales, pagos, etc.).
- Microservicios que necesitan comunicarse entre sí.
- Backend para aplicaciones móviles y SPA (Single Page Applications).

## CLI

### Definición

Una CLI, o interfaz de línea de comandos, es un mecanismo de interacción con una aplicación mediante instrucciones textuales ingresadas por el usuario en una consola o terminal. Este tipo de interfaz permite ejecutar comandos, scripts y utilidades de manera precisa, rápida y con un nivel de control detallado que no siempre está disponible en interfaces gráficas. Las CLIs son ampliamente usadas en sistemas operativos, programación, administración de servidores y herramientas de desarrollo.

Las aplicaciones que exponen una CLI suelen incluir un parser de comandos, argumentos y opciones, permitiendo automatizar tareas mediante scripts y pipelines. La CLI también es fundamental para entornos donde no se dispone de interfaces gráficas, como servidores remotos, contenedores o entornos embebidos.

## Historia y Evolución

Las primeras computadoras, en las décadas de 1960 y 1970, operaban exclusivamente mediante comandos escritos en terminales de texto conectados a sistemas centralizados. Sistemas como Unix implementaron shells altamente flexibles que permitían componer comandos y automatizar tareas mediante scripts, estableciendo estándares que aún perduran. Con la llegada de las interfaces gráficas en los años 80 y 90, la CLI perdió protagonismo para usuarios comunes, pero se mantuvo esencial para tareas técnicas.

En la era moderna, herramientas como Git, Docker, Kubernetes, Python, Node.js y muchas plataformas cloud dependen fuertemente de CLIs. La tendencia hacia la automatización, DevOps y la infraestructura como código ha reforzado su relevancia, convirtiéndola en un componente imprescindible para desarrolladores y administradores de sistemas.

## Ventajas

- Ligera y rápida: no requiere recursos gráficos.
- Automatizable: fácil de integrar en scripts y pipelines.
- Precisa: comandos exactos, argumentos claros.
- Muy útil para desarrolladores y administradores de sistemas.

## Desventajas

- Menos amigable para usuarios no técnicos.
- Curva de aprendizaje: hay que memorizar comandos y opciones.
- Difícil de descubrir funcionalidades sin documentación o ayuda

## Casos de uso y aplicación

- Herramientas de desarrollo y DevOps (compilación, despliegue, administración).
- Clientes para consumir APIs (por ejemplo, una CLI que consuma tu servicio hexagonal).
- Scripts de automatización en servidores.

## **Swagger**

### Definición

Swagger, hoy parte del ecosistema OpenAPI, es un conjunto de herramientas y un estándar para describir, documentar y consumir APIs REST mediante archivos estructurados que definen endpoints, parámetros, respuestas, modelos de datos y reglas de validación. La especificación OpenAPI permite representar una API de forma precisa e independiente del lenguaje, lo que facilita la generación automática de documentación, clientes y servidores.

Swagger proporciona herramientas como Swagger UI, que genera una interfaz interactiva para probar endpoints, y Swagger Editor, que permite crear y validar especificaciones. Al formalizar la estructura de una API, facilita la comunicación entre equipos, la integración continua y el versionamiento consistente de servicios.

### Historia y Evolución

Swagger fue creado alrededor de 2011 por Tony Tam con la intención de mejorar la representación y documentación de APIs REST. Su popularidad creció rápidamente, convirtiéndose en una herramienta estándar para equipos de desarrollo. En 2015 se creó la OpenAPI Initiative bajo la Linux Foundation, y Swagger donó su especificación para convertirla en un estándar abierto. A partir de ese momento, la especificación REST pasó a conocerse como OpenAPI Specification, y Swagger se enfocó en las herramientas de implementación.

Desde entonces, OpenAPI ha evolucionado con nuevas versiones que agregan soporte para validaciones avanzadas, documentación más rica y mayor integración con frameworks modernos. Hoy en día, es uno de los pilares fundamentales en la construcción y documentación de APIs empresariales.

### Ventajas

- Documentación clara e interactiva: Swagger UI permite probar los endpoints desde el navegador.
- Facilita la comunicación entre frontend y backend.
- Permite generar código (SDKs, stubs) en varios lenguajes.
- Muy buena integración con Spring Boot a través de librerías como springdoc-openapi.

### Desventajas

- Requiere mantener el archivo de especificación actualizado (puede desincronizarse si no hay disciplina).
- La configuración inicial puede confundir si no se entiende bien OpenAPI.

- Para APIs muy grandes, los archivos de especificación pueden volverse extensos.

## Casos de uso y aplicación

- Cualquier proyecto que exponga una API REST y necesite documentación decente.
- Proyectos académicos donde se quiere mostrar claramente el diseño de la API.
- Organizaciones que generan SDKs para clientes a partir de la especificación.

## Docker

### Definición

Docker es una plataforma de virtualización basada en contenedores que permite empaquetar aplicaciones junto con todas sus dependencias en entornos aislados y reproducibles. Un contenedor ejecuta una instancia ligera y eficiente que comparte el kernel del sistema operativo, pero mantiene archivos, procesos y configuraciones completamente separados del host. Esto garantiza que una aplicación se ejecute de manera consistente independientemente del entorno donde se despliegue.

La plataforma incluye herramientas para construir imágenes a partir de archivos Dockerfile, gestionar contenedores, redes, volúmenes y repositorios de imágenes. Docker se integra fácilmente con pipelines de CI/CD y con herramientas de orquestación como Kubernetes, facilitando la automatización y escalabilidad del ciclo de vida de software.

### Historia y Evolución

Docker fue lanzado en 2013 por Solomon Hykes y su equipo en dotCloud, aprovechando tecnologías existentes de aislamiento como namespaces y cgroups del kernel de Linux. Su principal contribución fue ofrecer una experiencia simple, estandarizada y accesible para crear y compartir contenedores. La comunidad adoptó rápidamente el modelo, lo que transformó el panorama del desarrollo y despliegue de software.

Con el tiempo, Docker impulsó el surgimiento de estándares como OCI (Open Container Initiative), que definió formatos abiertos para imágenes y runtime de contenedores. También permitió la aparición de orquestadores como Kubernetes, que se convirtieron en herramientas centrales para escalar contenedores en clústeres. Actualmente, Docker es un elemento fundamental del desarrollo moderno, especialmente en microservicios, DevOps y computación en la nube.

### Ventajas

- Portabilidad: el mismo contenedor corre en distintas máquinas y nubes.
- Aislamiento: evita conflictos de dependencias entre aplicaciones.

- Ligero comparado con máquinas virtuales completas.
- Facilita la integración continua y despliegue continuo (CI/CD).

## Desventajas

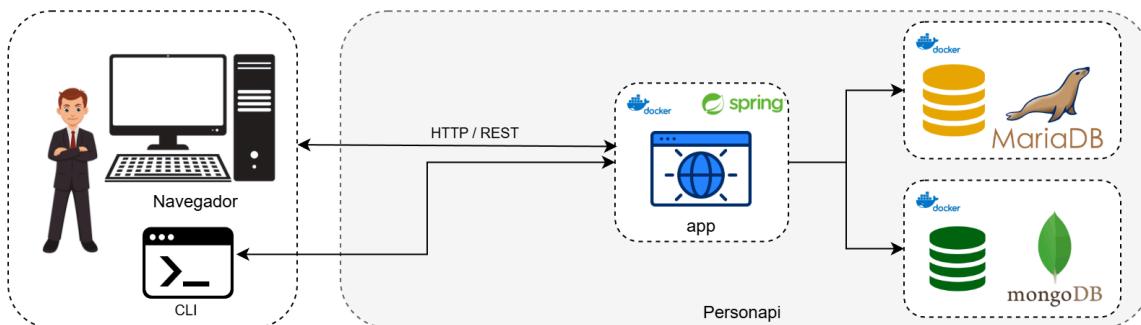
- Requiere entender conceptos de contenedores, imágenes, volúmenes, redes, etc.
- Si se configura mal, puede haber problemas de seguridad o consumo de recursos.
- La gestión de muchos contenedores requiere herramientas adicionales (Kubernetes, Docker Swarm, etc.).

## Casos de uso y aplicación

- Empaquetar aplicaciones Spring Boot con sus dependencias (incluyendo JDK, etc.).
- Entornos de desarrollo homogéneos para todo el equipo.
- Despliegue de microservicios en la nube.
- Laboratorios y prácticas donde se quiere levantar rápidamente bases de datos como MongoDB y MariaDB sin instalarlas en el sistema.

## Diseño

### Diagrama Alto Nivel



El diagrama muestra la arquitectura general del sistema Personapi, donde un usuario puede interactuar con la aplicación mediante dos vías: un navegador web o una interfaz de línea de comandos (CLI). Se envían solicitudes hacia una aplicación desarrollada en Spring Boot, la cual se ejecuta dentro de un contenedor Docker. Esta aplicación actúa como el núcleo del sistema, gestionando la lógica de negocio y exponiendo los servicios necesarios para atender las peticiones externas sin depender directamente de las tecnologías subyacentes.

Una vez recibida una solicitud, la aplicación se comunica con dos bases de datos también desplegadas en contenedores Docker: MariaDB, para el almacenamiento relacional tradicional, y MongoDB, para el manejo de información no estructurada o semiestructurada. Esta separación permite aprovechar las ventajas particulares de cada motor de base de datos y demuestra la flexibilidad del sistema para integrarse con componentes heterogéneos. En conjunto, el diagrama evidencia una arquitectura modular y portable que facilita la escalabilidad, el mantenimiento y la evolución futura del sistema.

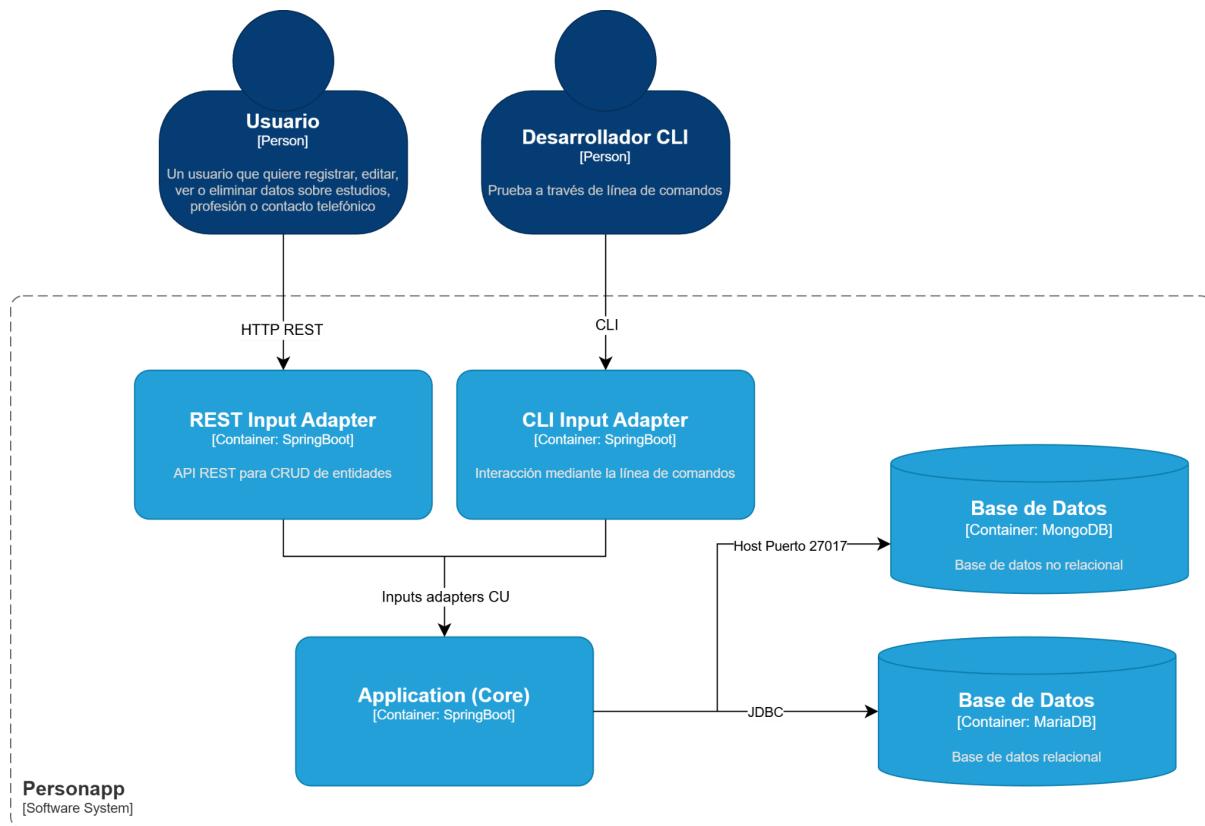
## Diagrama de Contexto



El diagrama de contexto representa, de manera general, cómo diferentes tipos de usuarios interactúan con el Sistema Agenda de Datos. Por un lado, se encuentra el usuario final, quien busca registrar, actualizar, consultar o eliminar información relacionada con estudios, profesión o datos de contacto. Este usuario se comunica con el sistema a través de servicios HTTP REST, lo que permite una interacción sencilla mediante aplicaciones web u otros clientes compatibles con este protocolo.

Por otro lado, el desarrollador CLI utiliza el sistema desde una interfaz de línea de comandos, principalmente con fines de prueba o validación. Ambos actores se conectan al mismo sistema central, el cual ofrece funcionalidades para gestionar datos profesionales, académicos y personales. En conjunto, el diagrama muestra el alcance del sistema, los actores que lo utilizan y los canales de comunicación mediante los cuales se relacionan con la plataforma.

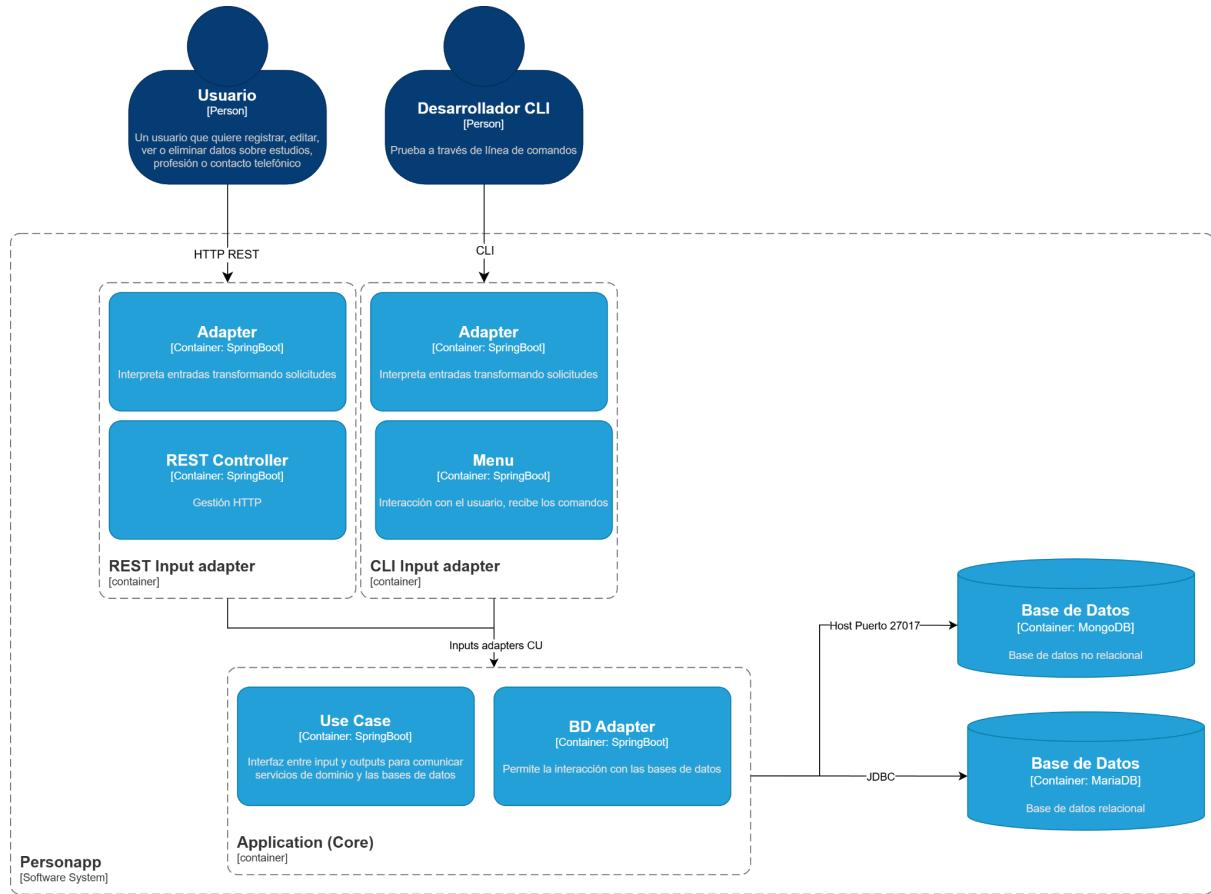
## Diagrama de Contenedores



El diagrama de contenedores describe cómo se estructura internamente el sistema Personapp y cómo se distribuyen sus componentes principales dentro de distintos contenedores. En la parte superior se observan los actores externos: el usuario final, que interactúa mediante peticiones HTTP REST, y el desarrollador CLI, que prueba y ejecuta operaciones desde la línea de comandos. Ambos actores se conectan al sistema a través de dos adaptadores de entrada: uno orientado a REST para gestionar operaciones CRUD mediante una API y otro diseñado para recibir comandos desde una interfaz CLI. Estos adaptadores son los responsables de recibir las solicitudes externas y comunicarlas hacia el núcleo de la aplicación.

Dentro del sistema, la lógica central reside en el contenedor Application (Core), implementado en Spring Boot, que procesa las solicitudes provenientes de los adaptadores y gestiona el flujo de la información. Para persistir y consultar datos, este núcleo se conecta con dos bases de datos ejecutadas también como contenedores independientes: una en MongoDB, destinada al manejo de información no relacional, y otra en MariaDB, orientada a datos relacionales y consultas basadas en JDBC y en Host puerto 27017. En conjunto, el diagrama evidencia una arquitectura modular que separa claramente las responsabilidades y aprovecha contenedores aislados para garantizar mantenibilidad, escalabilidad y facilidad de despliegue.

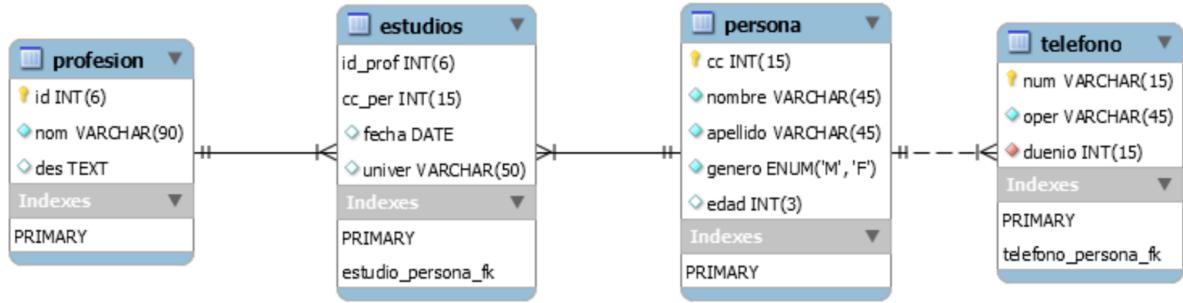
## Diagrama de Componentes



El diagrama de componentes muestra con mayor detalle la estructura interna del sistema Personapp y cómo se organizan las responsabilidades dentro de cada uno de sus módulos. En la capa superior se mantienen los actores externos: el usuario, que interactúa mediante solicitudes HTTP REST, y el desarrollador CLI, que prueba funcionalidades desde la línea de comandos. Para atender estas interacciones, el sistema cuenta con dos conjuntos de componentes de entrada. En el lado REST, un controlador interpreta y gestiona las solicitudes HTTP, mientras que un adaptador transforma estas peticiones para que el núcleo del sistema pueda procesarlas. En paralelo, el adaptador CLI y el componente de menú reciben y traducen los comandos ingresados por el desarrollador, manteniendo una interfaz coherente entre ambos modos de uso.

En la capa central se encuentra el núcleo de la aplicación, compuesto por los casos de uso y el adaptador de base de datos. Los casos de uso actúan como intermediarios entre los adaptadores de entrada y la lógica de dominio, coordinando el flujo de información y asegurando que se cumplan las reglas del sistema. A través del adaptador de base de datos, el núcleo se comunica con los dos motores de persistencia, implementados como contenedores independientes: MongoDB para el manejo de datos no relacionales y MariaDB para datos relacionales mediante operaciones JDBC y de host puerto 27017. La organización modular de estos componentes facilita la separación de responsabilidades, la extensibilidad del sistema y la integración con distintas tecnologías sin afectar el funcionamiento del dominio central.

## Entidades



La **entidad profesión** corresponde a la tabla profesion, donde se registra la información sobre las distintas ocupaciones o áreas laborales que puede ejercer una persona. Cada profesión cuenta con un identificador único (**id**), un nombre (**nom**) y una descripción más detallada (**des**), que permiten clasificarla y diferenciarla dentro del sistema. Esta entidad está relacionada con la de **Estudios** a través de una relación de uno a muchos, ya que una misma profesión puede estar vinculada a varios estudios realizados por diferentes personas.

La **entidad estudios** se representa en la tabla estudios, encargada de almacenar los datos académicos o formativos de las personas. Entre sus principales atributos se encuentran **id\_prof**, que enlaza cada estudio con la profesión correspondiente, y **cc\_per**, que relaciona el registro con la persona que cursó dicho estudio. También incluye la **fecha** en que se realizó y el nombre de la **universidad** donde se llevó a cabo. Su estructura refleja una relación de muchos a uno con las entidades **Persona** y **Profesión**, indicando que cada estudio pertenece a un solo individuo y está asociado a una profesión específica.

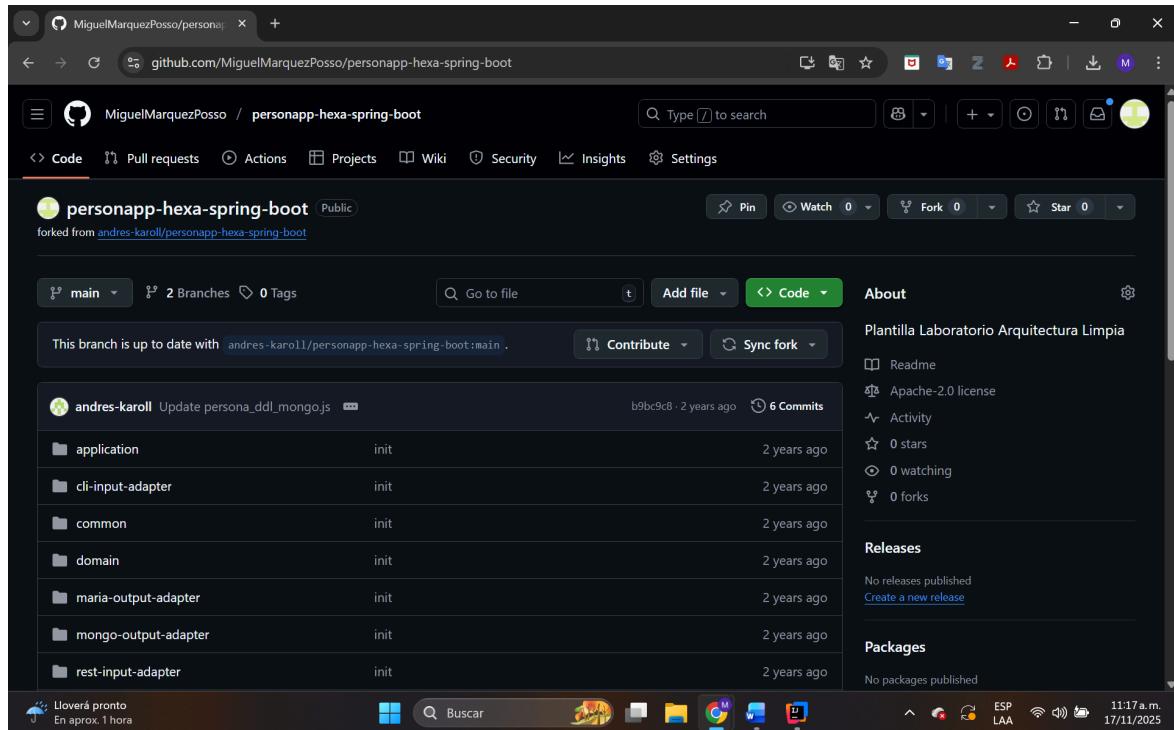
La **entidad persona**, representada por la tabla persona, almacena los datos personales de los usuarios del sistema. Cada registro se identifica mediante la **cédula de ciudadanía** (**cc**), que actúa como clave primaria. Además, contiene atributos como el **nombre**, **apellido**, **género** (masculino o femenino) y **edad**. Esta entidad se relaciona con **Estudios** y **Teléfono** mediante una relación de uno a muchos, lo que permite que una persona pueda tener varios estudios registrados y más de un número telefónico asociado.

Por último, la **entidad teléfono** se define en la tabla telefono, donde se guardan los números de contacto de las personas. Sus principales campos son el **número de teléfono** (**num**), el **operador** (**oper**) y el **dueño** (**duenio**), que es una clave foránea que vincula el número con la persona propietaria mediante su cédula. La relación entre **Teléfono** y **Persona** es de muchos a uno, ya que un mismo individuo puede tener varios teléfonos registrados, pero cada número pertenece únicamente a una persona.

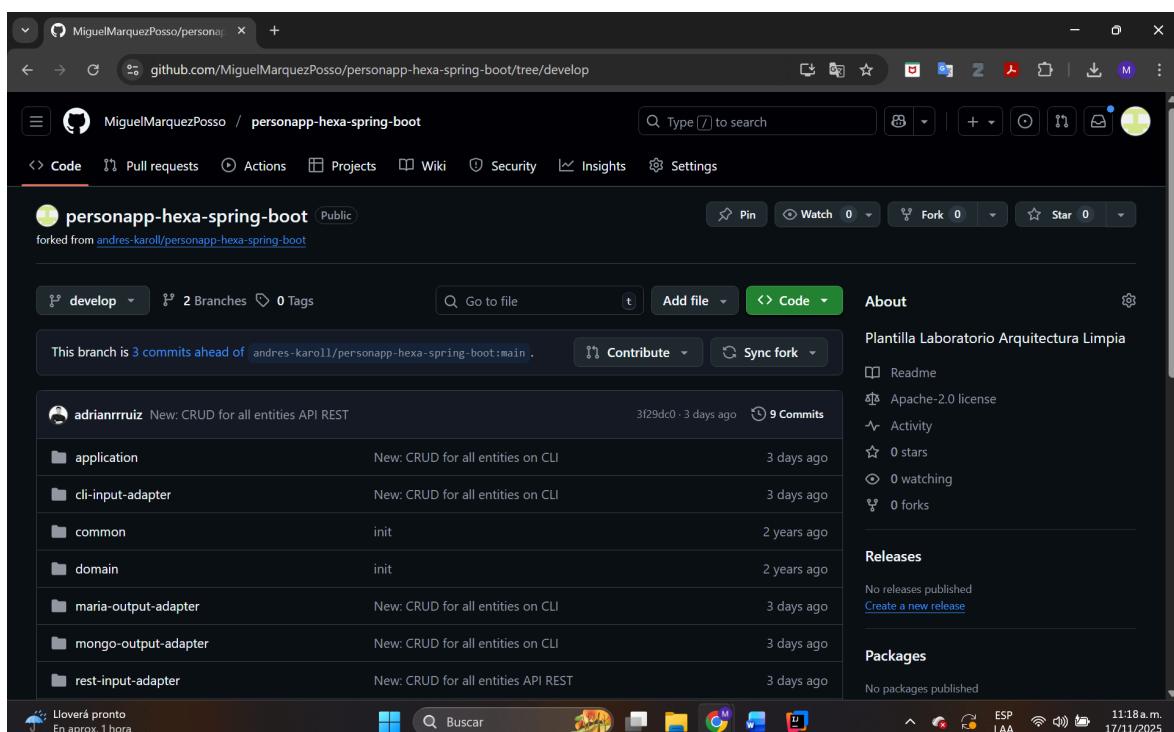
# Procedimiento

## Fork del repositorio

Se realizó un fork del repositorio: <https://github.com/andres-karoll/personapp-hexa-spring-boot>



El desarrollo del laboratorio se encuentra en la rama develop.



## Arquitectura Hexagonal

La aplicación PersonAPP está construida bajo los principios de la Arquitectura Hexagonal. Esta arquitectura se caracteriza por mantener el núcleo de la aplicación (dominio y lógica de negocio) completamente aislado de las tecnologías externas y frameworks específicos.

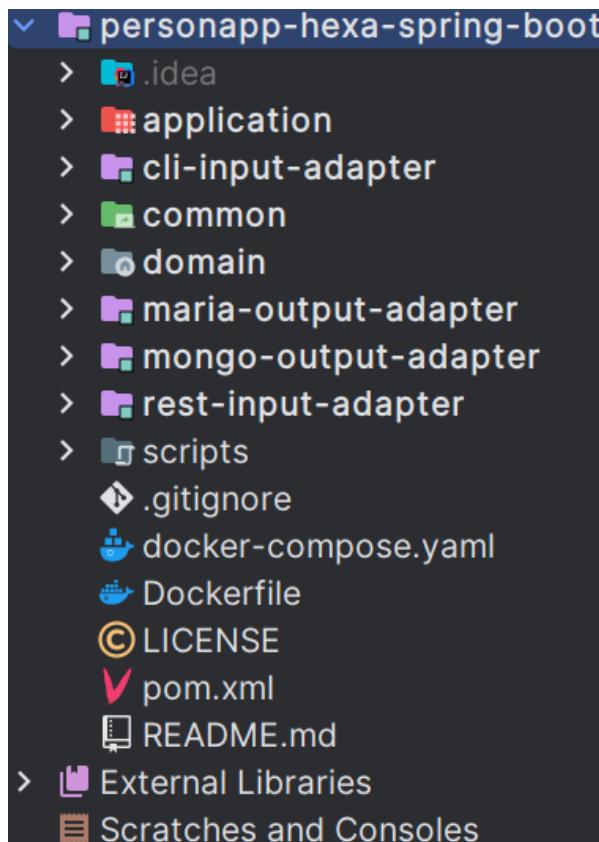
El diseño hexagonal se implementa mediante:

- **Núcleo Central:** Contiene las entidades de dominio y la lógica de negocio pura, sin dependencias externas.
- **Puertos de Entrada:** Interfaces que definen cómo interactuar con la aplicación (casos de uso).
- **Puertos de Salida:** Interfaces que definen cómo la aplicación se comunica con el exterior (persistencia).
- **Adaptadores:** Implementaciones concretas que conectan los puertos con tecnologías específicas

## Estructura Modular con Maven

El proyecto está organizado en 7 módulos Maven independientes pero interconectados:

Cada módulo tiene responsabilidades específicas y dependencias bien definidas, permitiendo un desarrollo, y despliegue independiente. El padre pom.xml centraliza la configuración común y versiones de dependencias.

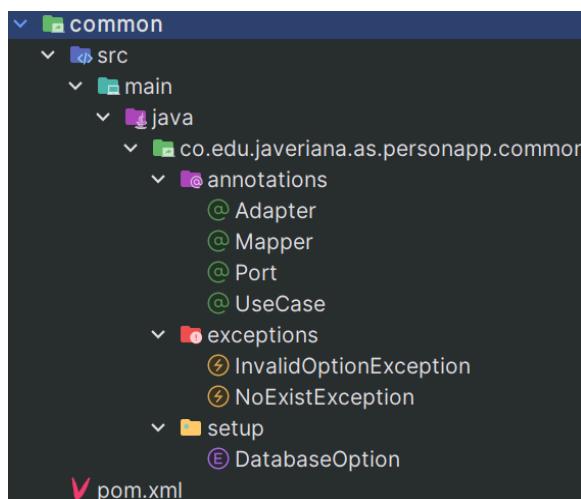


## Módulo Common

El módulo common proporciona utilidades y anotaciones compartidas por todos los demás módulos. Contiene anotaciones personalizadas que mejoran la semántica del código y reducen el acoplamiento con Spring Framework:

- **@Port:** Marca interfaces como puertos de entrada o salida de la arquitectura hexagonal.
- **@UseCase:** Identifica clases que implementan casos de uso de la aplicación.
- **@Adapter:** Señala clases que actúan como adaptadores en la arquitectura.
- **@Mapper:** Designa clases responsables de transformaciones entre diferentes representaciones de datos

Además, incluye excepciones personalizadas como `InvalidOptionException` y `NoExistException` para el manejo consistente de errores, y enumeraciones como `DatabaseOption` que define las opciones de base de datos soportadas (MARIA y MONGO). Este módulo no contiene lógica de negocio, solo elementos transversales necesarios para la cohesión del sistema.

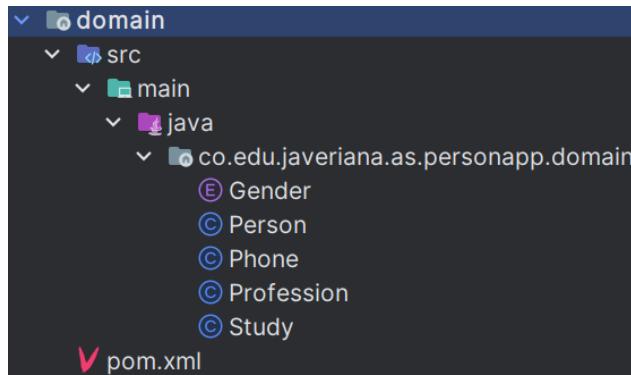


## Módulo Domain

El módulo domain constituye el corazón de la aplicación, conteniendo las entidades de dominio puras y la lógica de negocio fundamental. Las entidades principales son:

- **Person:** Representa una persona con identificación, nombres, apellidos, género, edad, teléfonos y estudios.
- **Phone:** Modela un teléfono con número, compañía y referencia a su dueño.
- **Profession:** Define una profesión con identificación, nombre, descripción y estudios relacionados.
- **Study:** Representa la relación entre una persona y una profesión, incluyendo fecha de graduación y universidad.

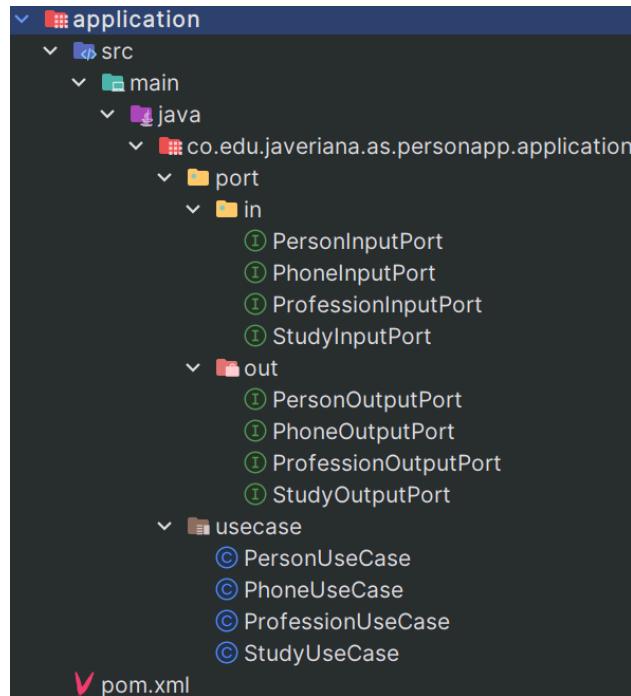
Cada entidad es un POJO simple con anotaciones de Lombok para reducir código boilerplate. La lógica de negocio está encapsulada en métodos como `isValidAge()` en la clase Person, que valida si la edad es un valor positivo. Este módulo no tiene dependencias externas ni anotaciones de frameworks, manteniendo la pureza del dominio según los principios de Domain-Driven Design.



## Módulo Application

El módulo application actúa como intermediario entre el dominio y los adaptadores externos. Define los contratos de la aplicación mediante:

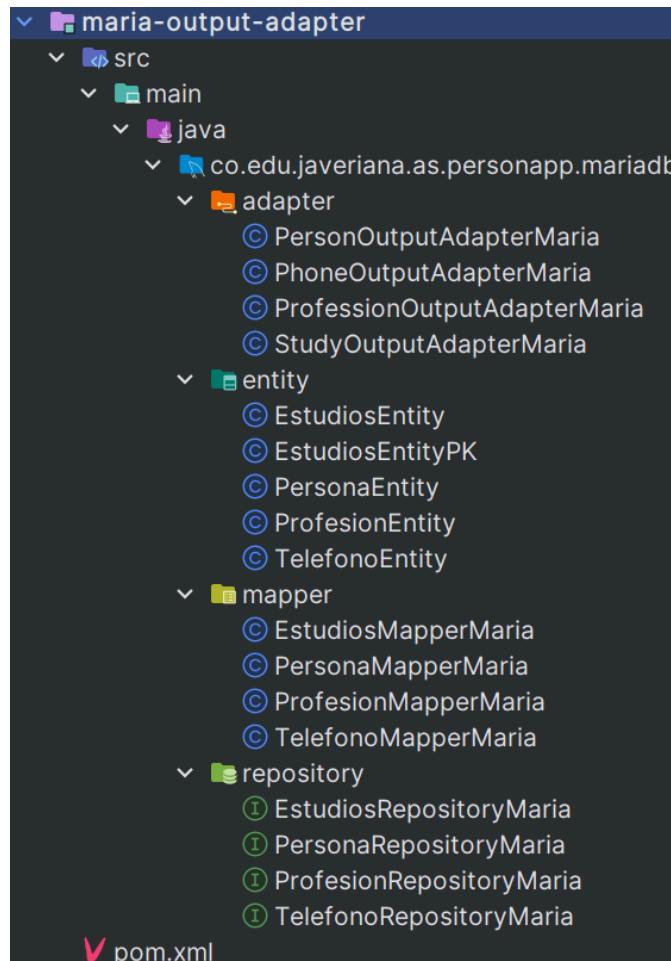
- **Puertos de Entrada (Input Ports):** Interfaces como PersonInputPort, PhoneInputPort, ProfessionInputPort y StudyInputPort definen las operaciones disponibles para interactuar con la aplicación (crear, editar, eliminar, buscar, listar).
- **Puertos de Salida (Output Ports):** Interfaces como PersonOutputPort, PhoneOutputPort, ProfessionOutputPort y StudyOutputPort abstraen las operaciones de persistencia.
- **Casos de Uso (Use Cases):** Clases como PersonUseCase, PhoneUseCase, ProfessionUseCase y StudyUseCase implementan la lógica de aplicación, orquestando las operaciones entre el dominio y los puertos de salida. Estos casos de uso inyectan la implementación específica del puerto de salida mediante qualifiers de Spring (@Qualifier("personOutputAdapterMaria")), permitiendo cambiar la base de datos en tiempo de ejecución.



## Módulo Maria-Output-Adapter

El módulo maria-output-adapter proporciona la implementación concreta para persistir datos en MariaDB. Incluye:

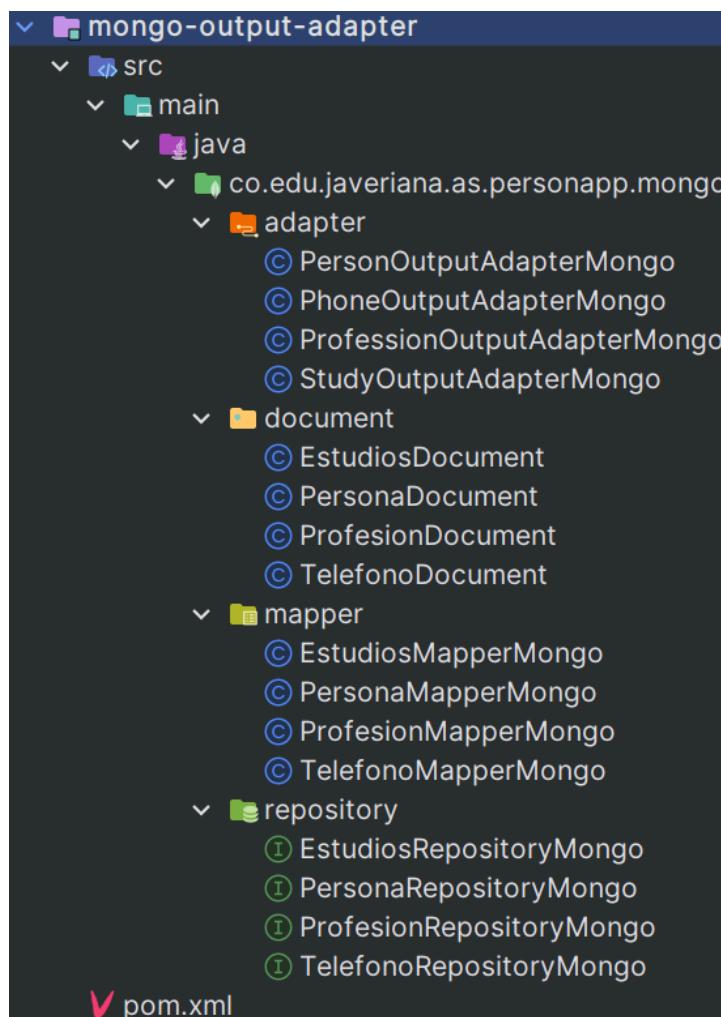
- **Entidades JPA:** PersonaEntity, TelefonoEntity, ProfesionEntity y EstudiosEntity mapean las tablas de la base de datos relacional, usando anotaciones JPA como @Entity, @Table, @Id y @Column.
- **Repositories Spring Data JPA:** Interfaces que extienden de JpaRepository para cada entidad, proporcionando operaciones CRUD básicas automáticamente.
- **Mappers:** Clases como PersonaMapperMaria transforman entre entidades de dominio y entidades JPA, manejando conversiones de tipos como Gender a Character y LocalDate a Date.
- **Adaptadores:** Clases como PersonOutputAdapterMaria implementan los puertos de salida, usando los repositorios y mappers para realizar las operaciones de persistencia. Están anotados con @Adapter("personOutputAdapterMaria") para identificación única y @Transactional para gestión de transacciones.



## Módulo Mongo-Output-Adapter

El módulo mongo-output-adapter ofrece una implementación alternativa para persistir datos en MongoDB, demostrando la flexibilidad de la arquitectura hexagonal. Contiene:

- **Documentos MongoDB:** PersonaDocument, TelefonoDocument, ProfesionDocument y EstudiosDocument representan los documentos en la base de datos NoSQL, usando anotaciones como @Document y @Id.
- **Repositories Spring Data MongoDB:** Interfaces que extienden de MongoRepository para cada documento.
- **Mappers:** Clases como PersonaMapperMongo realizan las transformaciones entre entidades de dominio y documentos MongoDB, manejando referencias entre documentos con @DocumentReference.
- **Adaptadores:** Clases como PersonOutputAdapterMongo implementan los mismos puertos de salida pero para MongoDB, usando los repositorios y mappers correspondientes. La estructura es idéntica al adaptador de MariaDB, pero con la implementación específica para MongoDB.

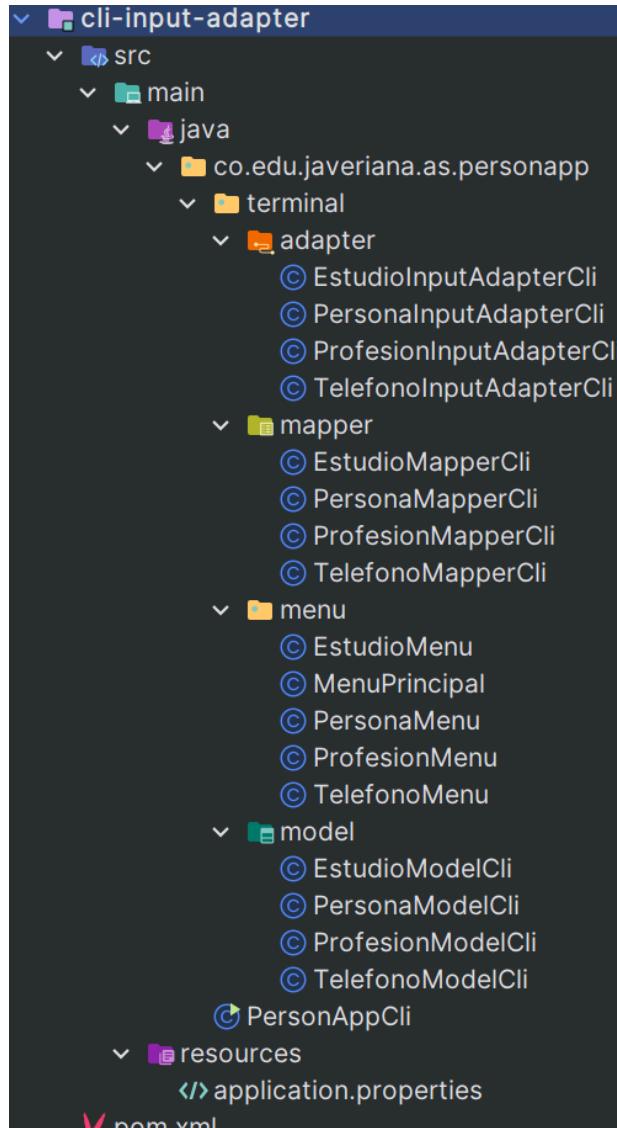


## Módulo CLI-Input-Adapter

El módulo cli-input-adapter proporciona una interfaz de usuario por línea de comandos para interactuar con la aplicación. Su implementación incluye:

- **Adaptadores de Entrada:** Clases como PersonaInputAdapterCli exponen métodos para cada operación (crear, editar, eliminar, listar, buscar) y se comunican con los casos de uso.
- **Menús Interactivos:** Clases como PersonaMenu, ProfesionMenu, TelefonoMenu y EstudioMenu presentan interfaces de usuario jerárquicas en consola, permitiendo navegar entre módulos y seleccionar la base de datos (MariaDB o MongoDB).
- **Mappers CLI:** Clases como PersonaMapperCli transforman entre entidades de dominio y modelos específicos para la CLI (PersonaModelCli), adaptando los datos para su presentación en consola.

La característica principal es que permite seleccionar la base de datos en tiempo de ejecución mediante el método setPersonOutputPortInjection(), que instancia el caso de uso con el adaptador de salida correspondiente.

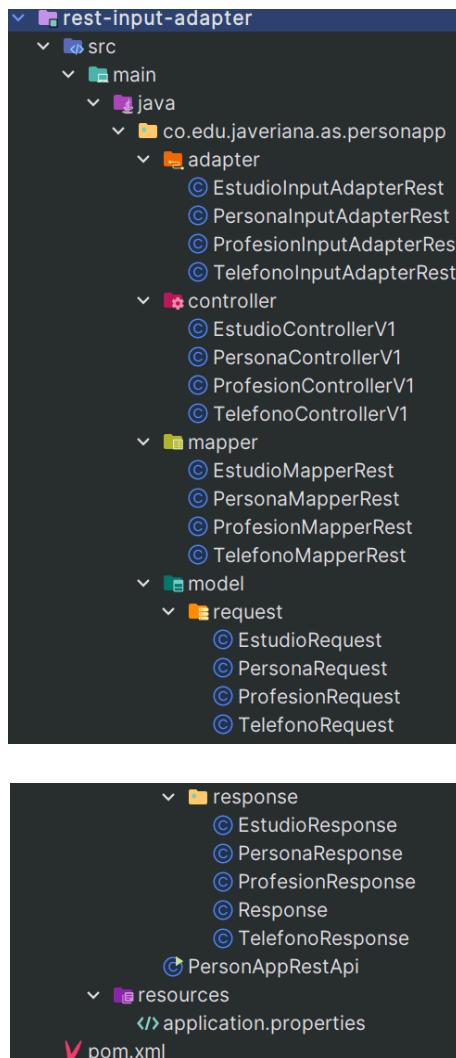


## Módulo REST-Input-Adapter

El módulo rest-input-adapter ofrece una interfaz HTTP REST para consumir los servicios de la aplicación. Componentes principales:

- **Controladores REST:** Clases como PersonaControllerV1 con endpoints para cada operación, usando anotaciones Spring MVC como @RestController, @RequestMapping, @GetMapping, @PostMapping, etc.
- **Adaptadores de Entrada:** Clases como PersonaInputAdapterRest manejan las peticiones HTTP y las transforman en llamadas a los casos de uso.
- **DTOs (Data Transfer Objects):** Clases de request (PersonaRequest, TelefonoRequest, etc.) y response (PersonaResponse, TelefonoResponse, etc.) representan el formato de datos para la API.
- **Mappers REST:** Clases como PersonaMapperRest convierten entre entidades de dominio y DTOs, y generan respuestas con información de la base de datos utilizada y estado de la operación.

La API REST también soporta cambio de base de datos en tiempo de ejecución mediante parámetros en los endpoints, permitiendo que un mismo servicio use diferentes bases de datos según la petición.



## Documentación de API con Swagger/OpenAPI

El módulo rest-input-adapter incorpora Swagger/OpenAPI para la documentación automática e interactiva de la API REST. Esta integración proporciona una interfaz web completa que permite a los desarrolladores y consumidores de la API entender, probar e interactuar con todos los endpoints disponibles sin necesidad de documentación manual.

La configuración de Swagger se realiza mediante la dependencia springdoc-openapi-ui en el archivo POM del módulo REST. Esta dependencia habilita automáticamente la generación de especificaciones OpenAPI 3.0 y proporciona la interfaz Swagger UI. En el archivo de propiedades de la aplicación, se configura la ruta de acceso a la documentación mediante la propiedad springdoc.api-docs.path=/api-docs, lo que establece el endpoint donde estará disponible la especificación OpenAPI en formato JSON.

Una vez que la aplicación REST está ejecutándose, los endpoints de OpenAPI generados incluyen tanto la especificación JSON raw en /api-docs como la interfaz visual en: <http://localhost:3000/swagger-ui.html>

## Configuración de Base de Datos

La aplicación está configurada para trabajar con dos bases de datos simultáneamente:

MariaDB: Configurada mediante propiedades Spring en application.properties:

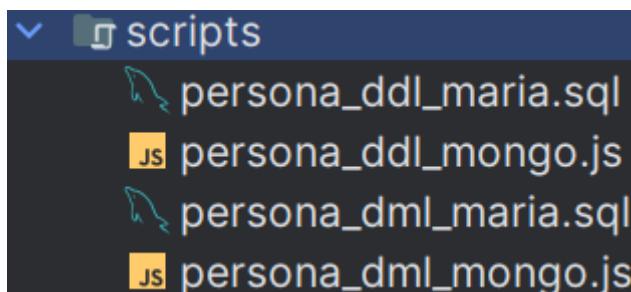
- **URL de conexión:** jdbc:mariadb://mariadb:3306/persona\_db
- **Driver:** org.mariadb.jdbc.Driver
- Configuración JPA para evitar generación automática de DDL

MongoDB: Configurada con Spring Data MongoDB:

- **Host y puerto:** mongo:27017
- **Base de datos:** persona\_db
- Autenticación con usuario y contraseña

**MariaDB:** El script persona\_ddl\_maría.sql crea el esquema con tablas para personas, profesiones, teléfonos y estudios, definiendo constraints y relaciones foreign key.

**MongoDB:** El script persona\_ddl\_mongo.js crea el usuario de la base de datos con los permisos necesarios, y persona\_dml\_mongo.js inserta datos iniciales usando la sintaxis de documentos MongoDB.



## Dockerfile y Construcción de Contenedores

El proyecto utiliza un Dockerfile multi-stage para construir y empaquetar la aplicación de manera eficiente. La configuración incluye:

- **Fase de Construcción (Build Stage):** Utiliza la imagen oficial de Maven con OpenJDK 11 para compilar el proyecto completo. Se copia todo el código fuente y se ejecuta mvn clean package para generar los archivos JAR.
- **Fase de Ejecución (Runtime Stage):** La fase final utiliza una imagen más ligera con solo el JRE (Java Runtime Environment) para reducir el tamaño del contenedor final.

El Dockerfile está diseñado para construir ambos adaptadores de entrada (CLI y REST) en un solo contenedor. Esta estrategia permite tener tanto la interfaz de línea de comandos como la API REST disponibles en el mismo contenedor, optimizando el uso de recursos.

El Dockerfile utiliza variables de entorno para determinar qué aplicación ejecutar. Esta configuración permite flexibilidad en el despliegue, ya que se puede cambiar entre CLI y REST simplemente modificando la variable de entorno APP\_TYPE al ejecutar el contenedor.

```
# Pre-fetch dependencies
FROM maven:3.9.4-eclipse-temurin-11 AS dependencies

# Copy the pom.xml files
WORKDIR /opt/app

COPY application/pom.xml application/pom.xml
COPY cli-input-adapter/pom.xml cli-input-adapter/pom.xml
COPY common/pom.xml common/pom.xml
COPY domain/pom.xml domain/pom.xml
COPY maria-output-adapter/pom.xml maria-output-adapter/pom.xml
COPY mongo-output-adapter/pom.xml mongo-output-adapter/pom.xml
COPY rest-input-adapter/pom.xml rest-input-adapter/pom.xml
COPY pom.xml .

RUN mvn -B -e org.apache.maven.plugins:maven-dependency-plugin:3.1.2:go-offline -DexcludeArtifactIds=domain
```

```

# Build the jar
FROM maven:3.9.4-eclipse-temurin-11 AS build

WORKDIR /opt/app

COPY --from=dependencies /root/.m2 /root/.m2
COPY --from=dependencies /opt/app/ /opt/app/

COPY application/src /opt/app/application/src
COPY cli-input-adapter/src /opt/app/cli-input-adapter/src
COPY common/src /opt/app/common/src
COPY domain/src /opt/app/domain/src
COPY maria-output-adapter/src /opt/app/maria-output-adapter/src
COPY mongo-output-adapter/src /opt/app/mongo-output-adapter/src
COPY rest-input-adapter/src /opt/app/rest-input-adapter/src

RUN mvn -B -e clean install -DskipTests

```

```

# Prepare runtime env
FROM eclipse-temurin:11-jdk-jammy

WORKDIR /opt/app

# COPY --from=build /opt/app/rest-input-adapter/target/*.jar /rest/app.jar
COPY --from=build /opt/app/cli-input-adapter/target/*.jar /cli/app.jar
COPY --from=build /opt/app/rest-input-adapter/target/*.jar /rest/app.jar

ENTRYPOINT ["sh", "-c", "java -jar ${APP_TYPE:-/cli/app.jar}"]

```

## Docker Compose para Despliegue

El archivo docker-compose.yml, referencia la misma imagen para ambos servicios pero con diferentes configuraciones. Esta estrategia de empaquetado permite que una sola imagen Docker pueda servir para múltiples propósitos, reduciendo la complejidad de mantenimiento y asegurando consistencia entre los diferentes componentes de la aplicación.

### Servicios de Base de Datos:

- mariadb: Contenedor con MariaDB 10.3.10, puerto 3307 mapeado a 3306
- mongo: Contenedor con MongoDB, configurado con autenticación
- mongo-express: Interfaz web para administrar MongoDB

### Servicios de Aplicación:

- personapp-cli-service: Contenedor para la interfaz de línea de comandos
- personapp-rest-service: Contenedor para la API REST, con puerto 3000 expuesto

## Configuración:

- Volúmenes persistentes para datos de ambas bases de datos
- Scripts de inicialización montados en directorios de entrada
- Variables de entorno para configuración de conexiones
- Dependencias entre servicios para asegurar el orden de inicio

```
services:  
  mongo:  
    image: mongo  
    restart: always  
    environment:  
      MONGO_INITDB_ROOT_USERNAME: root  
      MONGO_INITDB_ROOT_PASSWORD: example  
      MONGO_INITDB_DATABASE: "persona_db"  
    volumes:  
      - mongo_data:/data/db  
      - ./scripts/persona_ddl_mongo.js:/docker-entrypoint-initdb.d/01_persona_ddl_mongo.js  
      - ./scripts/persona_dml_mongo.js:/docker-entrypoint-initdb.d/02_persona_dml_mongo.js  
    ports:  
      - 27017:27017  
  
  mongo-express:  
    image: mongo-express  
    restart: always  
    ports:  
      - 8081:8081  
    environment:  
      ME_CONFIG_MONGODB_ADMINUSERNAME: root  
      ME_CONFIG_MONGODB_ADMINPASSWORD: example  
      ME_CONFIG_MONGODB_URL: mongodb://root@example@mongo:27017/  
      ME_CONFIG_BASICAUTH: false
```

```
mariadb:  
  image: mariadb:10.3.10  
  container_name: mariadb  
  ports:  
    - "3307:3306"  
  environment:  
    MYSQL_ROOT_PASSWORD: root  
    MYSQL_DATABASE: persona_db  
    MYSQL_USER: user  
    MYSQL_PASSWORD: password  
  volumes:  
    - maria_data:/var/lib/mysql  
    - ./scripts/persona_ddl_maria.sql:/docker-entrypoint-initdb.d/01_persona_ddl_maria.sql  
    - ./scripts/persona_dml_maria.sql:/docker-entrypoint-initdb.d/02_persona_dml_maria.sql  
  restart: always
```

```

personapp-cli-service:
  image: personapp-cli-service
  build:
    context: .
    dockerfile: Dockerfile
  container_name: personapp-cli-service
  environment:
    - MARIADB_HOST=mariadb
    - MARIADB_PORT=3306
    - MARIADB_USER=user
    - MARIADB_PASSWORD=password
    - MONGODB_HOST=mongo
    - MONGODB_PORT=27017
    - MONGODB_USER=root
    - MONGODB_PASSWORD=example
    - APP_TYPE=/cli/app.jar
  stdin_open: true
  tty: true
  depends_on:
    - mariadb
    - mongo

personapp-rest-service:
  image: personapp-rest-service
  build:
    context: .
    dockerfile: Dockerfile
  container_name: personapp-rest-service
  ports:
    - "3000:3000"
  environment:
    - MARIADB_HOST=mariadb
    - MARIADB_PORT=3306
    - MARIADB_USER=user
    - MARIADB_PASSWORD=password
    - MONGODB_HOST=mongo
    - MONGODB_PORT=27017
    - MONGODB_USER=root
    - MONGODB_PASSWORD=example
    - APP_TYPE=/rest/app.jar
  depends_on:
    - mariadb
    - mongo

```

```

volumes:
  maria_data:
  mongo_data:

```

## Uso de Swagger

La aplicación.

PS C:\Users\marqu\IdeaProjects\personapp-hexa-spring-boot> docker-compose up --build personapp-rest-service

[+] Running 24/24

- ✓ mariadb Pulled
  - ✓ a003e0ca99e5 Pull complete
  - ✓ 6f830a8cb936 Pull complete
  - ✓ 181debc3d23d Pull complete
  - ✓ a2434d5c8419 Pull complete
  - ✓ a57998e3e98d Pull complete
  - ✓ 5c9916314a1d Pull complete
  - ✓ da1315cffa03 Pull complete
  - ✓ fa83472a3562 Pull complete
  - ✓ 46fed4bbba52 Pull complete
  - ✓ f0f2a29c99a6 Pull complete
  - ✓ 32802ccfcfa4d Pull complete
  - ✓ f85999a80bef Pull complete
  - ✓ c6becfb25371 Pull complete
  - ✓ 7b5b2b8d4ee Pull complete
- ✓ mongo Pulled
  - ✓ d97355354a5b Pull complete
  - ✓ eef62aa7c052 Pull complete
  - ✓ 9d906841bd13 Pull complete
  - ✓ 2004306d63d5 Pull complete
  - ✓ 46a33c43d8ef Pull complete
  - ✓ fdc0018776b0 Pull complete
  - ✓ a801c4e80002 Pull complete
  - ✓ a0ca463113493 Pull complete
- [+] Building 153.6s (35/35) FINISHED
  - => [internal] load local bake definitions
  - => reading from stdin 645B
  - => [internal] load build definition from Dockerfile
  - => [internal] transfering dockerfile: 1.62KB
  - => [internal] load metadata for docker.io/library/eclipse-temurin:11-jdk-jammy
  - => [internal] load metadata for docker.io/library/maven:3.9.4-eclipse-temurin-11
  - => [internal] library/eclipse-temurin:11 taken for personapp-1 dockerfile

personapp-hexa-spring-boot

PS C:\Users\marqu\IdeaProjects\personapp-hexa-spring-boot> docker-compose up --build personapp-rest-service

[+] Running 7/7

- ✓ personapp-rest-service Built
- ✓ Network personapp-hexa-spring-boot\_default Created
- ✓ Volume personapp-hexa-spring-boot\_mongo\_data Created
- ✓ Volume personapp-hexa-spring-boot\_maría\_data Created
- ✓ Container personapp-hexa-spring-boot-mongo-1 Created
- ✓ Container mariadb Created
- ✓ Container personapp-rest-service Created

Attaching to personapp-rest-service

personapp-rest-service | 20:41:10.311 [main] INFO co.edu.javeriana.as.personapp.PersonAppRestApi - Starting PersonAppRestApi ...

personapp-rest-service |

personapp-rest-service | :: Spring Boot :: (v2.7.11)

personapp-rest-service |

personapp-rest-service | 2025-11-17 20:41:11.301 INFO 7 --- [ main] c.e.j.as.personapp.PersonAppRestApi : Starting PersonAppRestApi

using Java 11.0.29 on d3798f01e041 with PID 7 (/rest/app.jar started by root in /opt/app)

personapp-rest-service | 2025-11-17 20:41:11.303 INFO 7 --- [ main] c.e.j.as.personapp.PersonAppRestApi : No active profile set, falling back to 1 default profile: "default"

personapp-rest-service | 2025-11-17 20:41:12.334 INFO 7 --- [ main] .s.d.r.c.RepositoryConfigurationDelegate : Multiple Spring Data modules found, entering strict repository configuration mode

personapp-rest-service | 2025-11-17 20:41:12.337 INFO 7 --- [ main] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in DEFAULT mode.

personapp-rest-service | 2025-11-17 20:41:12.398 INFO 7 --- [ main] .RepositoryConfigurationExtensionSupport : Spring Data JPA - Could not safely identify store assignment for repository candidate interface co.edu.javeriana.as.personapp.mongo.repository.TelefonoRepositoryMongo; If you want this repository to be a JPA repository, consider annotating your entities with one of these annotations: javax.persistence.Entity, javax.persistence.MappedSuperclass (prefixed), or consider extending one of the following types with your repository: org.springframework.data.jpa.repository

personapp-hexa-spring-boot

The screenshot shows the Swagger UI interface for the **telefono-controller-v-1** API. At the top, there is a navigation bar with tabs for **/api-docs** and **Explore**. Below the navigation bar, the title **OpenAPI definition v0 OAS3** is displayed. A dropdown menu labeled **Servers** contains the URL **http://localhost:3000 - Generated server url**. The main area lists several API endpoints under the heading **telefono-controller-v-1**:

- GET /api/v1/telefono/{database}/{numero}**
- PUT /api/v1/telefono/{database}/{numero}**
- DELETE /api/v1/telefono/{database}/{numero}**
- POST /api/v1/telefono**
- GET /api/v1/telefono/{database}**

The browser status bar at the bottom indicates the date and time as **17/11/2025 3:47 p.m.**

This screenshot shows the same **telefono-controller-v-1** API documentation as the previous one, but with more endpoints listed:

- GET /api/v1/telefono/{database}/{numero}**
- PUT /api/v1/telefono/{database}/{numero}**
- DELETE /api/v1/telefono/{database}/{numero}**
- POST /api/v1/telefono**
- GET /api/v1/telefono/{database}**
- GET /api/v1/telefono/{database}/count**

Below this, there is a section for the **profesion-controller-v-1** API:

- GET /api/v1/profesion/{database}/{id}**
- PUT /api/v1/profesion/{database}/{id}**
- DELETE /api/v1/profesion/{database}/{id}**
- POST /api/v1/profesion**
- GET /api/v1/profesion/{database}**

The browser status bar at the bottom indicates the date and time as **17/11/2025 3:48 p.m.**

The screenshot shows the Swagger UI interface for the **profesion-controller-v-1** API. The URL is `localhost:3000/swagger-ui/index.html`. The interface lists several endpoints:

- GET /api/v1/profesion/{database}/{id}**
- PUT /api/v1/profesion/{database}/{id}**
- DELETE /api/v1/profesion/{database}/{id}**
- POST /api/v1/profesion**
- GET /api/v1/profesion/{database}**
- GET /api/v1/profesion/{database}/count**

Below this section, there is another header for **persona-controller-v-1**, which contains the following endpoints:

- GET /api/v1/persona/{database}/{cc}**
- PUT /api/v1/persona/{database}/{cc}**
- DELETE /api/v1/persona/{database}/{cc}**
- POST /api/v1/persona**
- GET /api/v1/persona/{database}**

The browser status bar at the bottom shows: 16°C Parc. soleado, Buscar, various icons, ESP LAA, 3:48 p.m., 17/11/2025.

The screenshot shows the Swagger UI interface for the **persona-controller-v-1** API. The URL is `localhost:3000/swagger-ui/index.html`. The interface lists the same set of endpoints as the previous screenshot:

- GET /api/v1/persona/{database}/{cc}**
- PUT /api/v1/persona/{database}/{cc}**
- DELETE /api/v1/persona/{database}/{cc}**
- POST /api/v1/persona**
- GET /api/v1/persona/{database}**
- GET /api/v1/persona/{database}/count**

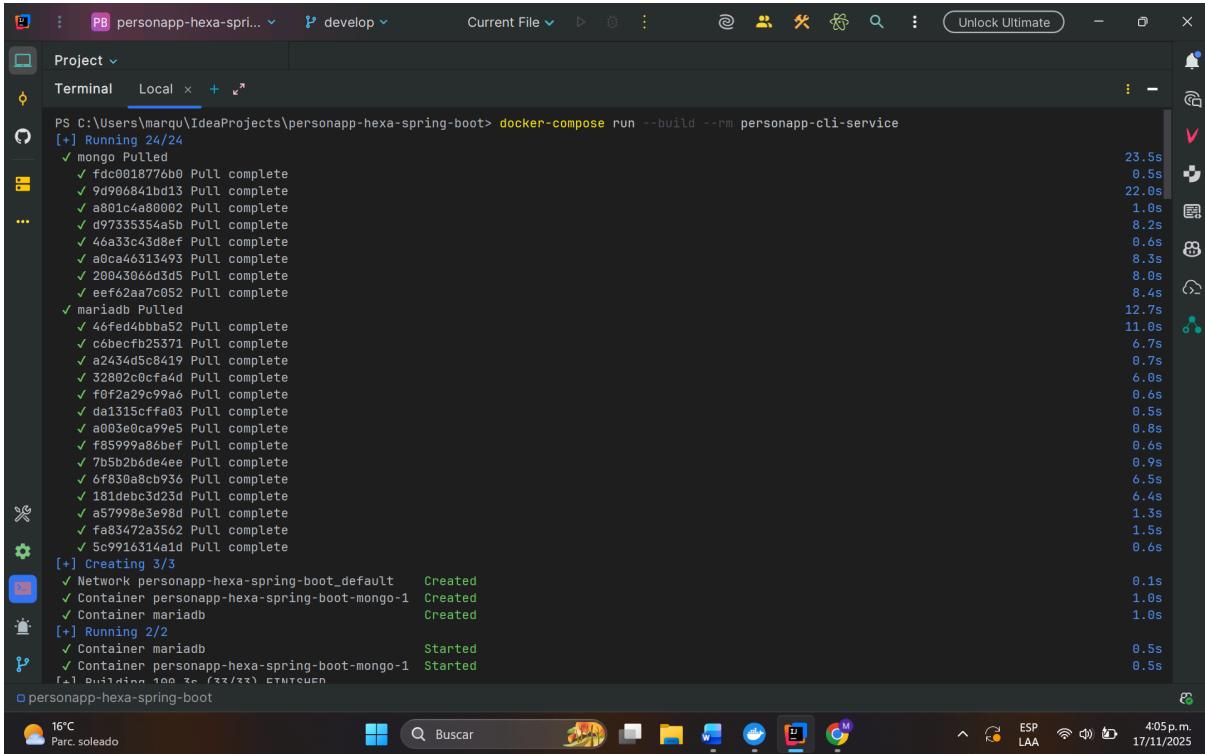
Below this section, there is another header for **estudio-controller-v-1**, which contains the following endpoints:

- GET /api/v1/estudio/{database}/{idProf}/{ccPer}**
- PUT /api/v1/estudio/{database}/{idProf}/{ccPer}**
- DELETE /api/v1/estudio/{database}/{idProf}/{ccPer}**
- POST /api/v1/estudio**
- GET /api/v1/estudio/{database}**

The browser status bar at the bottom shows: 16°C Parc. soleado, Buscar, various icons, ESP LAA, 3:49 p.m., 17/11/2025.

## Uso de CLI

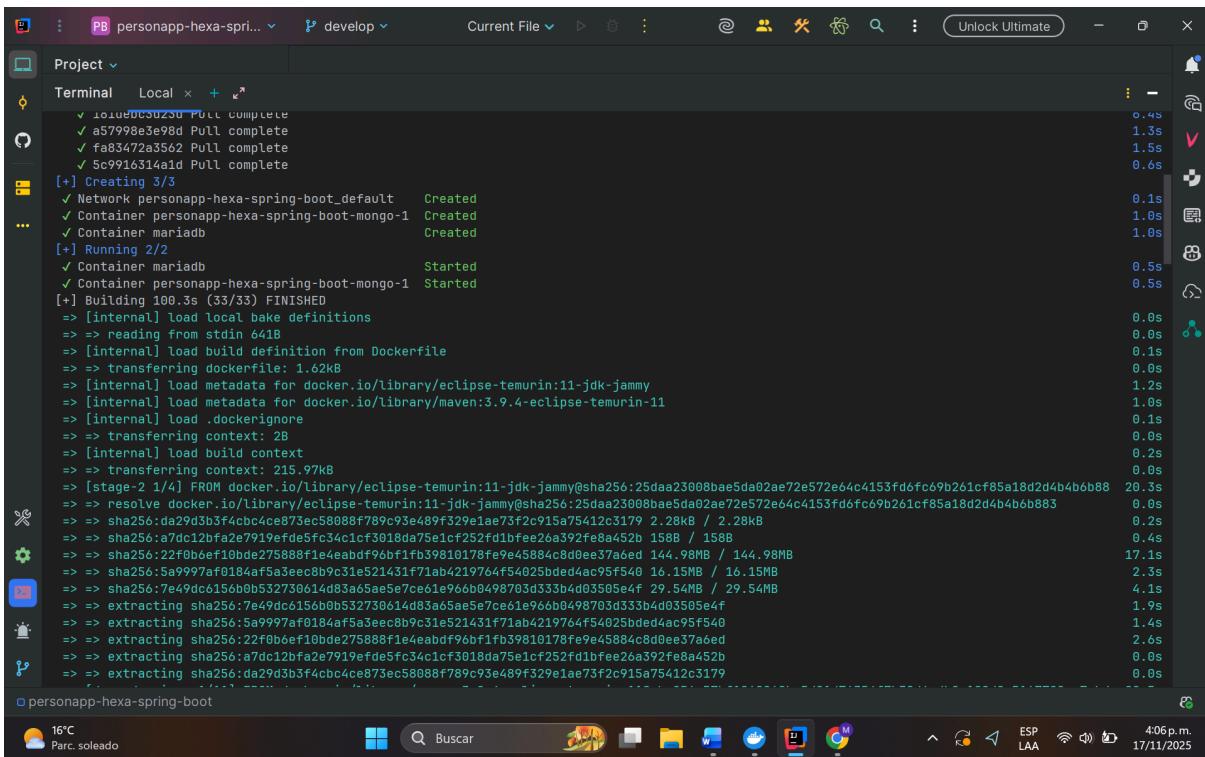
La aplicación.



```

PS C:\Users\marqu\IdeaProjects\personapp-hexa-spring-boot> docker-compose run --build --rm personapp-cli-service
[+] Running 24/24
  ✓ mongo Pulled
  ✓ fdc0018776b0 Pull complete
  ✓ 9d906841bd13 Pull complete
  ✓ a801c4e80002 Pull complete
...
  ✓ d973353545b Pull complete
  ✓ 46a33c43d8ef Pull complete
  ✓ a0ca46313493 Pull complete
  ✓ 2004306d3d5 Pull complete
  ✓ eef62aa7c052 Pull complete
  ✓ mariadb Pulled
  ✓ 46fed4bbfa52 Pull complete
  ✓ c6befcb25371 Pull complete
  ✓ a2434d5c8419 Pull complete
  ✓ 32802cc0cfa4d Pull complete
  ✓ f0f2a29c99a6 Pull complete
  ✓ da1315cffa83 Pull complete
  ✓ a003e0ca99e5 Pull complete
  ✓ f85999a86bef Pull complete
  ✓ 7b5b2b4de4ee Pull complete
  ✓ 6f830a8cb936 Pull complete
  ✓ 181debc3d23d Pull complete
  ✓ a57998e3e98d Pull complete
  ✓ fa83472a5562 Pull complete
  ✓ 5c9916314a1d Pull complete
[+] Creating 3/3
  ✓ Network personapp-hexa-spring-boot_default Created
  ✓ Container personapp-hexa-spring-boot-mongo-1 Created
  ✓ Container mariadb Created
[+] Running 2/2
  ✓ Container mariadb Started
  ✓ Container personapp-hexa-spring-boot-mongo-1 Started
  ⏎ 11 builds, 100% to 100% ⏎ 17/11/2025
  ⌂ personapp-hexa-spring-boot

```



```

PS C:\Users\marqu\IdeaProjects\personapp-hexa-spring-boot> docker-compose run --build --rm personapp-cli-service
[+] Creating 3/3
  ✓ Network personapp-hexa-spring-boot_default Created
  ✓ Container personapp-hexa-spring-boot-mongo-1 Created
  ✓ Container mariadb Created
[+] Running 2/2
  ✓ Container mariadb Started
  ✓ Container personapp-hexa-spring-boot-mongo-1 Started
[+] Building 100.3s (33/33) FINISHED
  => [internal] load local bake definitions
  => >> reading from stdin 641B
  => [internal] load build definition from Dockerfile
  => >> transferring dockerfile: 1.62kB
  => [internal] load metadata for docker.io/library/eclipse-temurin:11-jdk-jammy
  => [internal] load metadata for docker.io/library/maven:3.9.4-eclipse-temurin
  => [internal] load .dockernignore
  => >> transferring context: 2B
  => [internal] load build context
  => >> transferring context: 215.97kB
  => [stage-2 1/4] FROM docker.io/library/eclipse-temurin:11-jdk-jammy@sha256:25daa23008bae5da02ae72e572e64c4153fd6fc69b261cf85a18d2d4b4b6b88
  => >> resolve docker.io/library/eclipse-temurin:11-jdk-jammy@sha256:25daa23008bae5da02ae72e572e64c4153fd6fc69b261cf85a18d2d4b4b6b883
  => >> sha256:da29d3b3f4cb4ce873ec58088f789c93e489f329e1ae73f2c915a75412c3179 2.28kB / 2.28kB
  => >> sha256:a7dc12bfaf2e7919efde5fc34c1cf3018da75e1cf252fd1bfef2e6392fe8a452b 1588 / 1588
  => >> sha256:22f0b6ef10bde275888f1e4abdf96bf1fb39810178fe9e45884c8d00e37a6ed 144.98MB / 144.98MB
  => >> sha256:5a9997af0f184af5a3ec689c31e521431f71ab4219764f54025bdded4ac95f540 16.15MB / 16.15MB
  => >> sha256:7e49dc6156b0b532730614d3a65ae5e7ce61e96b0498703d33354d03505e4f 29.54MB / 29.54MB
  => >> extracting sha256:7e49dc6156b0b532730614d3a65ae5e7ce61e96b0498703d33354d03505e4f
  => >> extracting sha256:5a9997af0f184af5a3ec689c31e521431f71ab4219764f54025bdded4ac95f540
  => >> extracting sha256:22f0b6ef10bde275888f1e4abdf96bf1fb39810178fe9e45884c8d00e37a6ed
  => >> extracting sha256:a7dc12bfaf2e7919efde5fc34c1cf3018da75e1cf252fd1bfef2e6392fe8a452b
  => >> extracting sha256:da29d3b3f4cb4ce873ec58088f789c93e489f329e1ae73f2c915a75412c3179
  ⌂ personapp-hexa-spring-boot

```

```

PB personapp-hexa-spri... develop Current File ... Terminal Local ...
Project Terminal Local ...
:: Spring Boot :: (v2.7.11)

2025-11-17 21:05:32.801 INFO 7 --- [           main] c.e.javeriana.as.personapp.PersonAppCli : Starting PersonAppCli using Java 11.0.29 on fa7e709
47f5e with PID 7 (/cli/app.jar started by root in /opt/app)
2025-11-17 21:05:32.803 INFO 7 --- [           main] c.e.javeriana.as.personapp.PersonAppCli : No active profile set, falling back to 1 default pr
ofile: "default"
2025-11-17 21:05:33.472 INFO 7 --- [           main] s.d.r.c.RepositoryConfigurationDelegate : Multiple Spring Data modules found, entering strict
repository configuration mode
2025-11-17 21:05:33.475 INFO 7 --- [           main] s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in DEFAU
LT mode...
2025-11-17 21:05:33.536 INFO 7 --- [           main] .RepositoryConfigurationExtensionSupport : Spring Data JPA - Could not safely identify store a
ssignment for repository candidate interface co.edu.javeriana.as.personapp.mongo.repository.TelefonoRepositoryMongo; If you want this repository to
be a JPA repository, consider annotating your entities with one of these annotations: javax.persistence.Entity, javax.persistence.MappedSuperclass (p
REFERRED), or consider extending one of the following types with your repository: org.springframework.data.jpa.repository.JpaRepository
2025-11-17 21:05:33.539 INFO 7 --- [           main] .RepositoryConfigurationExtensionSupport : Spring Data JPA - Could not safely identify store a
ssignment for repository candidate interface co.edu.javeriana.as.personapp.mongo.repository.ProyectoRepositoryMongo; If you want this repository to
be a JPA repository, consider annotating your entities with one of these annotations: javax.persistence.Entity, javax.persistence.MappedSuperclass (p
REFERRED), or consider extending one of the following types with your repository: org.springframework.data.jpa.repository.JpaRepository
2025-11-17 21:05:33.542 INFO 7 --- [           main] .RepositoryConfigurationExtensionSupport : Spring Data JPA - Could not safely identify store a
ssignment for repository candidate interface co.edu.javeriana.as.personapp.mongo.repository.EstudiosRepositoryMongo; If you want this repository to
be a JPA repository, consider annotating your entities with one of these annotations: javax.persistence.Entity, javax.persistence.MappedSuperclass (p
REFERRED), or consider extending one of the following types with your repository: org.springframework.data.jpa.repository.JpaRepository
16°C Mayor: soleado Buscar ESP LAA 406 p.m. 17/11/2025

```

```

PB personapp-hexa-spri... develop Current File ... Terminal Local ...
Project Terminal Local ...
e UNIL - default
2025-11-17 21:05:36.731 INFO 7 --- [           main] org.mongodb.driver.client          : MongoClient with metadata {"driver": {"name": "mong
o-java-driver[sync]spring-boot", "version": "4.6.1"}, "os": {"type": "Linux", "name": "Linux", "architecture": "amd64", "version": "6.6.87.2-microso
ft-standard-WSL2"}, "platform": "Java/Eclipse Adoptium/11.0.29+7"} created with settings MongoClientSettings{readPreference=primary, writeConcern=W
riteConcern{w=null, wTimeout=null ms, journal=null}, retryWrites=true, retryReads=true, readConcern=ReadConcern{level=null}, credential=MongoCredenti
al{mechanism=null, userName='root', source='admin', password=<hidden>, mechanismProperties=<hidden>}, streamFactoryFactory=null, commandListeners=[]}
, codecRegistry=ProvidersCodecRegistry{codecProviders=[ValueCodecProvider{}, BsonValueCodecProvider{}, DBRefCodecProvider{}, DBObjectCodecProvider{}]
, DocumentCodecProvider{}, IterableCodecProvider{}, MapCodecProvider{}, GeoJsonCodecProvider{}, GridFSFileCodecProvider{}, JsNumberObjectCodecProvider{}, Js
onObjectCodecProvider{}, BsonCodecProvider{}, EnumCodecProvider{}, com.mongodb.Jep395RecordCodecProvider@7ade27bf], clusterSettings={hosts=[mongo:2
7017], srvServiceName=mongodb, mode=SINGLE, requiredClusterType=UNKNOWN, requiredReplicaSetName='null', serverSelector='null', clusterListeners='[]'}
, serverSelectionTimeout='30000 ms', localThreshold='30000 ms'}, socketSettings=SocketSettings{connectTimeoutMS=10000, readTimeoutMS=0, receiveBuffe
rSize=0, sendBufferSize=0, heartbeatSocketSettings=SocketSettings{connectTimeoutMS=10000, readTimeoutMS=10000, receiveBufferSize=0, sendBufferSize=0
}, connectionPoolSettings=ConnectionPoolSettings{maxSize=100, minSize=0, maxWaitTimerS=120000, maxConnectionLifetimeMS=0, maxConnectionIdleTimeMS=0
, maintenanceInitialDelayMS=0, maintenanceFrequencyMS=60000, connectionPoolListeners=[], maxConnecting=2}, serverSettings=ServerSettings{heartbeatF
requencyMS=10000, minHeartbeatFrequencyMS=500, serverListeners='[]', serverMonitorListeners='[]'}, sslSettings=SslSettings{isEnabled=false, invalidHost
NameAllowed=false, contextProvider=null}, applicationName='null', compressorList=[], uidRepresentation=JAVA_LEGACY, serverApi=null, autoEncryptionSettings=
null, contextProvider=null}
2025-11-17 21:05:36.782 INFO 7 --- [l']-mongo:27017 org.mongodb.driver.connection      : Opened connection [connectionId{localValue:2, serv
erValue:1}] to mongo:27017
2025-11-17 21:05:36.782 INFO 7 --- [l']-mongo:27017 org.mongodb.driver.connection      : Opened connection [connectionId{localValue:1, serv
erValue:2}] to mongo:27017
2025-11-17 21:05:36.787 INFO 7 --- [l']-mongo:27017 org.mongodb.driver.cluster       : Monitor thread successfully connected to server wit
h description ServerDescription{address:mongo:27017, type=STANDALONE, state=CONNECTED, ok=true, minWireVersion=0, maxWireVersion=27, maxDocumentSize
=16777216, logicalSessionTimeoutMinutes=30, roundTripTimeNanos=47455715}
2025-11-17 21:05:37.413 INFO 7 --- [           main] c.e.javeriana.as.personapp.PersonAppCli : Started PersonAppCli in 5.216 seconds (JVM running
for 5.849)
2025-11-17 21:05:37.416 INFO 7 --- [           main] c.e.javeriana.as.personapp.PersonAppCli : EXECUTING : command line runner
-----
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
Ingrrese una opción:
0 personapp-hexa-spring-boot
Chubascos Mañana Buscar ESP LAA 407 p.m. 17/11/2025

```

The screenshot shows a terminal window with the following log output:

```
2025-11-17 21:05:36.782 INFO 7 --- [l']-mongo:27017 org.mongodb.driver.connection : Opened connection [connectionId{localValue:2, serve
rValue:1} to mongo:27017
2025-11-17 21:05:36.782 INFO 7 --- [l']-mongo:27017 org.mongodb.driver.connection : Opened connection [connectionId{localValue:1, serve
rValue:2} to mongo:27017
2025-11-17 21:05:36.787 INFO 7 --- [l']-mongo:27017 org.mongodb.driver.cluster : Monitor thread successfully connected to server wit
h description ServerDescription{address:mongo:27017, type=STANDALONE, state=CONNECTED, ok=true, minWireVersion=0, maxWireVersion=27, maxDocumentSize
=16777216, logicalSessionTimeoutMinutes=30, roundTripTimeNanos=47455715}
2025-11-17 21:05:37.413 INFO 7 --- [main] c.e.javeriana.as.personapp.PersonAppCli : Started PersonAppCli in 5.216 seconds (JVM running
for 5.849)
2025-11-17 21:05:37.416 INFO 7 --- [main] c.e.javeriana.as.personapp.PersonAppCli : EXECUTING : command line runner
```

Below the log, there is a menu with the following options:

- 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: 1

-----

□ personapp-hexa-spring-boot

Chubascos  
Mañana

Buscar

ESP LAA 407 p.m. 17/11/2025

The screenshot shows a terminal window with the same log output as the first one:

```
2025-11-17 21:05:36.782 INFO 7 --- [l']-mongo:27017 org.mongodb.driver.connection : Opened connection [connectionId{localValue:2, serve
rValue:1} to mongo:27017
2025-11-17 21:05:36.782 INFO 7 --- [l']-mongo:27017 org.mongodb.driver.connection : Opened connection [connectionId{localValue:1, serve
rValue:2} to mongo:27017
2025-11-17 21:05:36.787 INFO 7 --- [l']-mongo:27017 org.mongodb.driver.cluster : Monitor thread successfully connected to server wit
h description ServerDescription{address:mongo:27017, type=STANDALONE, state=CONNECTED, ok=true, minWireVersion=0, maxWireVersion=27, maxDocumentSize
=16777216, logicalSessionTimeoutMinutes=30, roundTripTimeNanos=47455715}
2025-11-17 21:05:37.413 INFO 7 --- [main] c.e.javeriana.as.personapp.PersonAppCli : Started PersonAppCli in 5.216 seconds (JVM running
for 5.849)
2025-11-17 21:05:37.416 INFO 7 --- [main] c.e.javeriana.as.personapp.PersonAppCli : EXECUTING : command line runner
```

Below the log, there is a menu with the following options:

- 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: 1

-----

□ personapp-hexa-spring-boot

Chubascos  
Mañana

Buscar

ESP LAA 408 p.m. 17/11/2025

```

Project Terminal Local x + ↵
4 para trabajar con el modulo de estudios
0 para Salir
Ingresar una opción: 1
-----
1 para MariaDB
2 para MongoDB
0 para regresar
Ingresar una opción: 1
-----
1 para ver todas las personas
2 para crear una persona
3 para editar una persona
4 para eliminar una persona
5 para buscar una persona por CC
6 para contar personas
0 para regresar
Ingresar una opción: 1
2025-11-17 21:11:18.835 INFO 7 --- [           main] c.e.j.a.p.t.a.PersonaInputAdapterCli : Into historial PersonaEntity in Input Adapter
2025-11-17 21:11:18.839 INFO 7 --- [           main] c.e.j.a.p.a.usecase.PersonUseCase : Output: class co.edu.javeriana.as.personapp.mariadb
.PersonaModelCli@123456789, nombre=Pepe, apellido=Perez, genero=MALE, edad=30)
PersonaModelCli@147258369, nombre=Pepita, apellido=Juarez, genero=FEMALE, edad=10)
PersonaModelCli@321654987, nombre=Pepa, apellido=Juarez, genero=FEMALE, edad=30)
PersonaModelCli@963852741, nombre=Fede, apellido=Perez, genero=MALE, edad=18)
PersonaModelCli@987654321, nombre=Pepito, apellido=Perez, genero=MALE, edad=null)
-----
1 para ver todas las personas
2 para crear una persona
3 para editar una persona
4 para eliminar una persona
5 para buscar una persona por CC
6 para contar personas
0 para regresar
Ingresar una opción:
0 personapp-hexa-spring-boot

```

Llovizna  
Por la noche

Buscar

ESP LAA 4:11 p.m. 17/11/2025

```

Project Terminal Local x + ↵
2025-11-17 21:05:36.782 INFO / --- [l'---mongo:27017] org.mongodb.driver.connection : opened connection [connectionId{localValue:1, serverName: "localhost:27017"}]
rValue:2} to mongo:27017
2025-11-17 21:05:36.787 INFO 7 --- [l'---mongo:27017] org.mongodb.driver.cluster : Monitor thread successfully connected to server with
h description ServerDescription{address:mongo:27017, type=STANDALONE, state=CONNECTED, ok=true, minWireVersion=0, maxWireVersion=27, maxDocumentSize
=16777216, logicalSessionTimeoutMinutes=30, roundTripTimeNanos=47455715}
2025-11-17 21:05:37.413 INFO 7 --- [           main] c.e.javeriana.as.personapp.PersonAppCli : Started PersonAppCli in 5.216 seconds (JVM running
for 5.849)
2025-11-17 21:05:37.416 INFO 7 --- [           main] c.e.javeriana.as.personapp.PersonAppCli : EXECUTING : command line runner
-----
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
Ingresar una opción: 1
-----
1 para MariaDB
2 para MongoDB
0 para regresar
Ingresar una opción: 1
-----
1 para ver todas las personas
2 para crear una persona
3 para editar una persona
4 para eliminar una persona
5 para buscar una persona por CC
6 para contar personas
0 para regresar
Ingresar una opción: 0
-----
1 para MariaDB
2 para MongoDB
0 para regresar
Ingresar una opción: 0
0 personapp-hexa-spring-boot

```

Chubascos  
Mañana

Buscar

ESP LAA 4:09 p.m. 17/11/2025

```

Project Terminal Local x + ↻
1 para trabajar con el Módulo de Personas
2 para trabajar con el Módulo de Profesiones
3 para trabajar con el Módulo de Teléfonos
4 para trabajar con el Módulo de Estudios
0 para Salir
Ingresá una opción: 1
-----
...
1 para MySQL
2 para MongoDB
0 para regresar
Ingresá una opción: 1
-----
1 para ver todas las personas
2 para crear una persona
3 para editar una persona
4 para eliminar una persona
5 para buscar una persona por CC
6 para contar personas
0 para regresar
Ingresá una opción: 0
-----
1 para MySQL
2 para MongoDB
0 para regresar
Ingresá una opción: 2
-----
⚙️ 1 para ver todas las personas
2 para crear una persona
3 para editar una persona
4 para eliminar una persona
5 para buscar una persona por CC
6 para contar personas
0 para regresar
Ingresá una opción: 1
-----
personapp-hexa-spring-boot
Chubascos
Mañana
Buscar ESP LAA 4:09 p.m. 17/11/2025

```

comando para rest: docker-compose up --build personapp-rest-service

comando para cli: docker-compose run --build --rm personapp-cli-service

## Repositorio

Link: <https://github.com/MiguelMarquezPosso/personapp-hexa-spring-boot/tree/develop>

## Conclusiones y Lecciones Aprendidas

- Trabajar en la construcción del sistema bajo una arquitectura hexagonal y modular fue un ejercicio sumamente valioso. Más allá de permitirnos implementar correctamente los componentes del proyecto, fue una oportunidad para comprender cómo debe estructurarse un software que, en el futuro, pueda evolucionar hacia una arquitectura de microservicios. La modularidad facilitó separar responsabilidades de manera clara, lo que demuestra que un monolito bien organizado puede transformarse de forma gradual y controlada sin tener que rehacer la solución desde cero ni incurrir en altos costos de rediseño.
- El desarrollo de dos adaptadores de entrada, uno mediante CLI y otro mediante API REST, permitió evidenciar cómo un mismo núcleo de negocio puede exponerse a diferentes mecanismos de interacción sin verse afectado. Este ejercicio mostró de forma práctica la importancia de aislar la lógica central del sistema para que pueda adaptarse a nuevos canales de comunicación, tecnologías o interfaces de usuario sin riesgos ni reescrituras innecesarias. Esta flexibilidad es clave para que un software pueda escalar y mantenerse vigente con el paso del tiempo.

- Utilizar dos motores de base de datos distintos, MongoDB y MariaDB, representó un desafío que ayudó a reforzar la importancia de diseñar puertos y adaptadores con claridad y rigor. La experiencia demostró que una correcta gestión de dependencias, acompañada de una inyección de dependencias bien aplicada, permite cambiar tecnologías, optimizar recursos y mantener el dominio completamente independiente de los detalles de persistencia. Al final, esto se traduce en un sistema más robusto, adaptable y preparado para escenarios reales complejos.
- El proceso de validar los métodos de inyección, configurar adaptadores e implementar los mapeos entre entidades de dominio y las interfaces de entrada fue esencial para asegurar consistencia en el funcionamiento del sistema. Estas pruebas permitieron identificar errores tempranos, fortalecer la estabilidad del conjunto y comprobar la importancia de contar con estrategias de verificación exhaustivas. Un sistema que aspira a crecer debe apoyarse en pruebas sólidas que reduzcan riesgos y garanticen que cada nueva pieza encaja correctamente con las demás.
- La definición de capas bien diferenciadas y el uso de interfaces claras y documentadas proporcionaron una estructura ordenada y sostenible. Esta forma de trabajar facilita el mantenimiento, agiliza la incorporación de nuevas funcionalidades y evita que los cambios afecten de manera innecesaria a otros componentes. Para sistemas que buscan ser escalables y perdurar en el tiempo, la modularidad no es solo una ventaja técnica, sino una necesidad que asegura la integridad del software frente a futuras extensiones.
- Finalmente, la construcción de un monolito modular siguiendo los principios de la arquitectura hexagonal permitió adquirir una comprensión profunda y realista de cómo puede llevarse a cabo una migración progresiva hacia microservicios. Al tener un dominio bien separado y adaptadores claramente definidos, el sistema queda preparado para dividirse en servicios independientes sin grandes reestructuraciones. Este aprendizaje es especialmente valioso, ya que muestra una ruta práctica para evolucionar un sistema monolítico hacia un ecosistema distribuido, manteniendo control sobre los cambios y reduciendo riesgos.

## Referencias

- Home (2024) Docker Documentation. <https://docs.docker.com/>
- Cockburn A. Hexagonal Architecture. <https://alistair.cockburn.us/hexagonal-architecture>
- Martin RC. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Prentice Hall. <https://www.pearson.com/en-us/subject-catalog/p/clean-architecture/P200000006263/9780134494166>
- Fowler M. Patterns of Enterprise Application Architecture. <https://martinfowler.com/books/eaa.html>
- Oracle. Java SE Documentation. <https://docs.oracle.com/javase/11/docs/>

- OpenJDK. <https://openjdk.org/>
- Walls C. Spring Boot in Action. Manning Publications.  
<https://www.manning.com/books/spring-boot-in-action>
- Spring Framework Official Documentation. <https://spring.io/projects/spring-framework>
- Spring Boot Official Documentation. <https://spring.io/projects/spring-boot>
- MongoDB Official Documentation. <https://www.mongodb.com/docs/>
- MariaDB Server Documentation. <https://mariadb.com/kb/en/documentation/>
- Fielding RT. Architectural Styles and the Design of Network-based Software Architectures. (REST thesis) [https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding\\_dissertation.pdf](https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf)
- OpenAPI Specification. <https://spec.openapis.org/>
- Swagger Tools. <https://swagger.io/tools/>
- Pahl C. Containerization and the PaaS Cloud. IEEE Cloud Computing. <https://ieeexplore.ieee.org/document/7160245>