

Desarrollar software a partir de la integración de sus módulos componentes

EVIDENCIA DE DESEMPEÑO GA8-220501096-AA1-EV01

John Jairo Oviedo Mejía
ANALISIS Y DESARROLLO DE SOFTWARE | FICHA: 2627004

1. Introducción

Integrar módulos componentes para desarrollar software es una práctica esencial en la construcción de aplicaciones modernas, especialmente en entornos web. Esta perspectiva de desarrollo permite dividir un sistema complejo en partes más manejables, lo que facilita la colaboración entre equipos y la escalabilidad del proyecto. Con este marco de referencia, la codificación de módulos se convierte en un proceso vital, donde cada componente se desarrolla de acuerdo con los requisitos específicos del sistema para garantizar su funcionamiento adecuado en el contexto de una aplicación orientada a la web. Este punto de vista, no solo promueve una arquitectura modular y flexible, sino que también facilita la mantenibilidad, la depuración y la evolución continua del software. En el presente trabajo, se presentan las tareas específicas para el desarrollo del backend de un proyecto de API denominada apirest.

2. Objetivos

Generales

- Presentar el desarrollo de las tareas específicas utilizadas para el desarrollo del backend de una API.

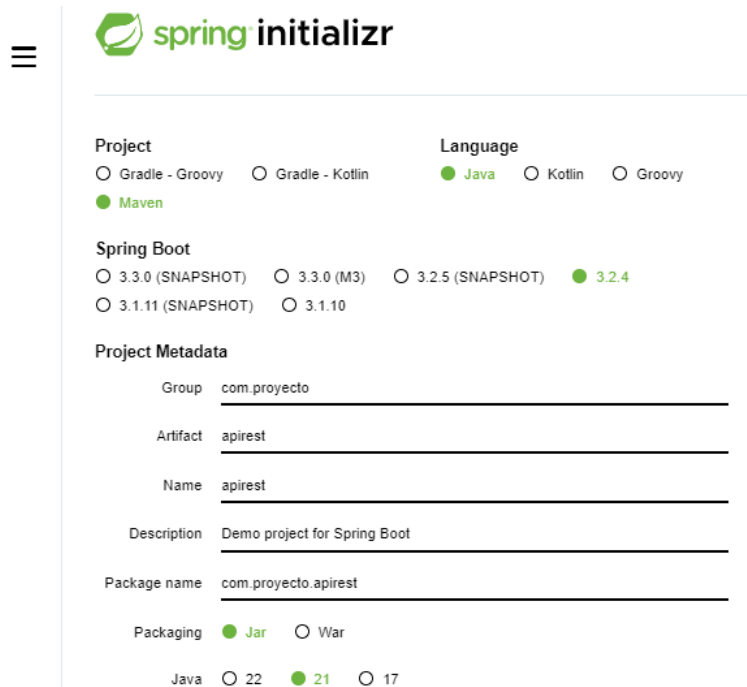
Específicos

- Identificar, analizar y describir cada uno de los pasos utilizados en el desarrollo del backend de una API.

3. Tareas específicas para el desarrollo del Backend

3.1. Codificación en Spring Boot

Se abrió ingresó a la página <https://start.spring.io/>, y se realizó la configuración mostrada en la imagen:



The image shows the Spring Initializr web form. It has a sidebar with a hamburger menu icon. The main form is titled 'spring initializr'. It contains several sections: 'Project' with radio buttons for 'Gradle - Groovy', 'Gradle - Kotlin', 'Maven' (selected), 'Java' (selected), 'Kotlin', and 'Groovy'; 'Spring Boot' with radio buttons for '3.3.0 (SNAPSHOT)', '3.3.0 (M3)', '3.2.5 (SNAPSHOT)', '3.2.4' (selected), '3.1.11 (SNAPSHOT)', and '3.1.10'; 'Project Metadata' with input fields for 'Group' (com.proyecto), 'Artifact' (apiREST), 'Name' (apiREST), 'Description' (Demo project for Spring Boot), and 'Package name' (com.proyecto.apiREST); 'Packaging' with radio buttons for 'Jar' (selected) and 'War'; and 'Java' with radio buttons for '22', '21' (selected), and '17'.

Figura 1. Página de Spring Initializr

Posterior a esto se configuraron las siguientes dependencias necesarias para desarrollar la API:

- ✓ Spring Boot Dev Tool: Esta dependencia proporciona herramientas de desarrollo adicionales que facilitan el ciclo de desarrollo en Spring Boot. Con esta dependencia se pueden realizar cambios en el código fuente y ver los resultados inmediatamente sin necesidad de reiniciar la aplicación.
- ✓ Lombok: Gestiona todas las librerías de JAVA. Esta biblioteca permite simplificar la creación de clases y mejorar la legibilidad del código.
- ✓ Spring Web: Proporciona características para el desarrollo de aplicaciones web en Spring Boot.
- ✓ MySQL Driver: Este controlador es necesario para permitir que una aplicación Spring Boot se conecte y comunique con una base de datos MySQL.
- ✓ Spring Data JPA: es una parte de Spring Data que simplifica el acceso a datos mediante el mapeo de objetos Java a entidades en una base de datos relacional.

Permite el uso de repositorios JPA (Java Persistence API) para realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) en las entidades de la base de datos.

Una vez elegidas estas librerías se procedió a descargar el archivo comprimido al computador, luego se descomprimió este y se cargó en el editor de código Visual Studio Code. Luego de cargar el archivo se crearon las siguientes carpetas:

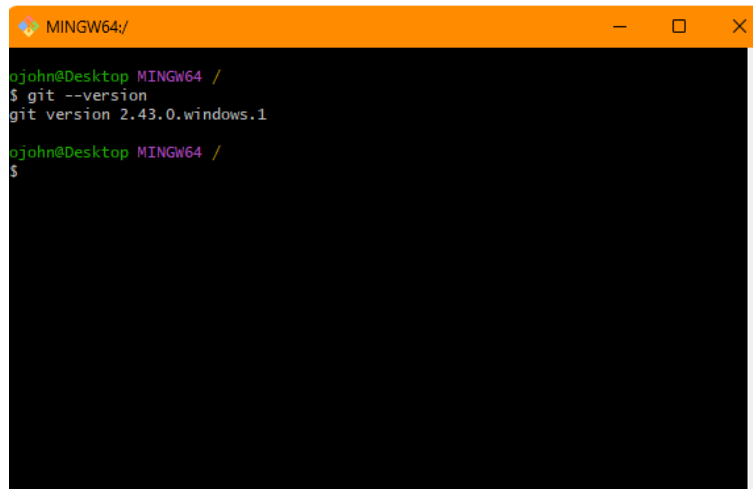
- Person Controller
- Entity (Person)

- Person Repository
- Person Service

Cada una de ellas contiene los módulos .java para el desarrollo del proyecto. En referencia a cada uno de ellos, en el archivo adjunto al presente trabajo se puede ver su respectivo código fuente.

3.2. Control de Versiones

- Se utilizó un repositorio de control de versiones como Git para gestionar y rastrear el código fuente del backend, esto se llevó a cabo de la siguiente manera:
 - ✓ Puesto que tengo instalado Git en mi sistema operativo, entonces lo que hice es verificar que este se encuentra correctamente instalado utilizando el comando `git --version`, en la imagen a continuación se puede apreciar la versión de Git instalada.



```
mingw64:/
john@Desktop MINGW64 /
$ git --version
git version 2.43.0.windows.1
john@Desktop MINGW64 /
$
```

Figura 2. Versión de Git instalada

- ✓ En Github creó un nuevo repositorio como se ve en la imagen,

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Required fields are marked with an asterisk (*).

Owner * JJprogr / Repository name * apirest
apirest is available.

Great repository names are short and memorable. Need inspiration? How about [improved-pancake](#) ?

Description (optional)

API



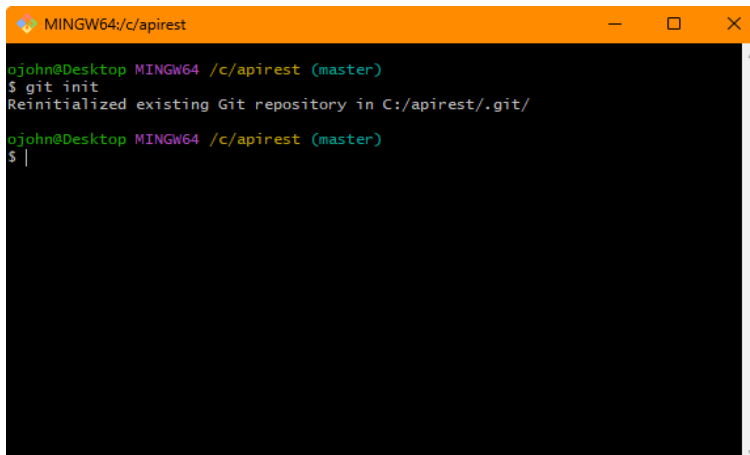
- ☒  **Public**
Anyone on the internet can see this repository. You choose who can commit.
- ☐  **Private**
You choose who can see and commit to this repository.

Figura 3. Creación del repositorio

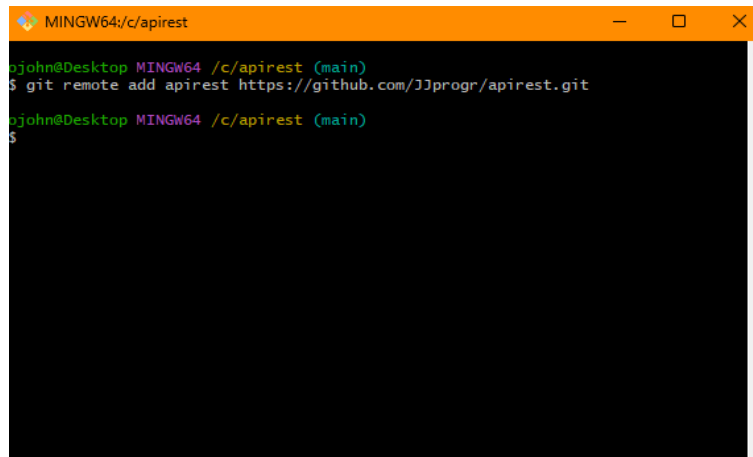
- ✓ Una vez estando en la ruta de la carpeta del proyecto apirest, se inicializa el repositorio, para ello se introduce el comando git init.



```
MINGW64/c/apirest
ajohn@Desktop MINGW64 /c/apirest (master)
$ git init
Reinitialized existing Git repository in C:/apirest/.git/
ajohn@Desktop MINGW64 /c/apirest (master)
$ |
```

Figura 4. Comando de inicialización del repositorio

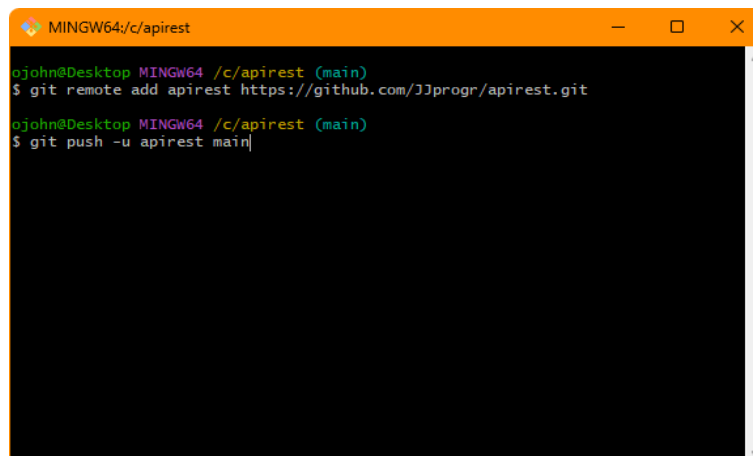
- ✓ Ahora se sube los archivos a la rama principal poniendo la orden, git branch -M main.
- ✓ Ponemos el comando que se ve en la imagen, que contiene la ruta específica en donde está alojado el proyecto.



```
MINGW64:/c/apirest
ojohn@Desktop MINGW64 /c/apirest (main)
$ git remote add apirest https://github.com/JJprogr/apirest.git
ojohn@Desktop MINGW64 /c/apirest (main)
$
```

Figura 5. Comando para añadir el repositorio a Github

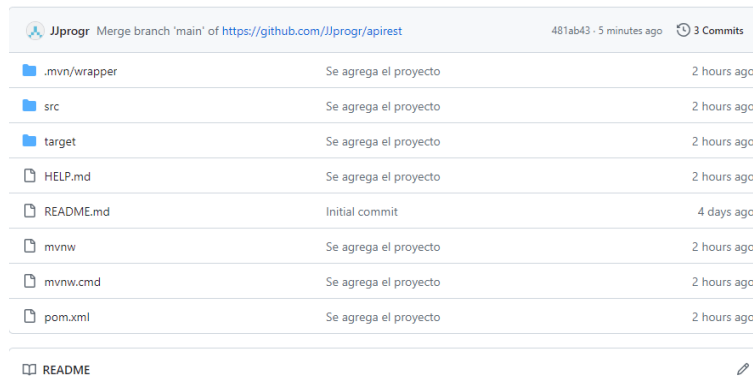
- ✓ El siguiente paso es hacer efectiva la sincronización del repositorio local con el remoto, utilizando el siguiente comando de la imagen:




```
MINGW64:/c/apirest
ojohn@Desktop MINGW64 /c/apirest (main)
$ git remote add apirest https://github.com/JJprogr/apirest.git
ojohn@Desktop MINGW64 /c/apirest (main)
$ git push -u apirest main
```

Figura 6. Sincronización del repositorio local con el remoto

✓ Esta es la imagen del proyecto subido a Github.



JJprogr Merge branch 'main' of https://github.com/JJprogr/apiREST		481ab43 · 5 minutes ago	3 Commits
· .mvn/wrapper	Se agrega el proyecto	2 hours ago	
· src	Se agrega el proyecto	2 hours ago	
· target	Se agrega el proyecto	2 hours ago	
· HELP.md	Se agrega el proyecto	2 hours ago	
· README.md	Initial commit	4 days ago	
· mvnw	Se agrega el proyecto	2 hours ago	
· mvnw.cmd	Se agrega el proyecto	2 hours ago	
· pom.xml	Se agrega el proyecto	2 hours ago	
· README			

apiREST

Figura 7. Proyecto apiREST subido a Github

3.3. Gestión de dependencias

La sección 3.1 explica las dependencias que se utilizaron en el proyecto.

3.4. Frameworks en el Backend

Se utiliza Spring Boot como framework principal para el desarrollo del backend.

Aquí hay algunas razones por las cuales Spring Boot es una excelente elección como framework principal para el desarrollo del backend:

Facilita la Configuración:

Spring Boot simplifica la configuración del proyecto al proporcionar configuraciones predeterminadas y automáticas. Esto significa que se puede comenzar a trabajar en la aplicación al instante sin la necesidad de configurar manualmente cada aspecto.

Desarrollo Rápido:

Con Spring Boot, proporciona configuraciones predeterminadas basadas en las mejores prácticas y estándares de la industria, lo que acelera el proceso de desarrollo.

Arquitectura Modular:

Spring Boot facilita la creación de una arquitectura modular para la aplicación, lo que permite dividir el código en componentes independientes y reutilizables. Esto promueve la escalabilidad y el mantenimiento a largo plazo de la aplicación.

Integración Sencilla:

Spring Boot ofrece integración con una amplia gama de tecnologías y bibliotecas, lo que permite utilizar fácilmente herramientas y servicios externos en la aplicación. Esto incluye integración con bases de datos, sistemas de mensajería, servicios en la nube y más.

Soporte Activo y Comunidad Vibrante:

Spring Boot cuenta con un equipo de desarrollo activo y una comunidad de usuarios muy grande y activa. Esto significa se puede encontrar una amplia documentación, tutoriales y ayuda en línea para resolver cualquier problema o pregunta que puedas tener.

En las imágenes a continuación se encuentra el código del proyecto:

```
package com.proyecto.apirest.Persona;

import jakarta.persistence.Basic;
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.Id;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@Entity
@AllArgsConstructor
@NoArgsConstructor
public class Person {
    @Id
    @GeneratedValue
    private int id;

    @Basic
    private String nombre; // Corregido de "string" a "String"
    private String apellido;
    private String email;
}
```

Figura 8. Código para el archivo denominado Person.java


```

Archivo  Editar  Ver

package com.proyecto.apirest.Persona;

import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import lombok.RequiredArgsConstructor;

@RestController
@RequestMapping("/person")
@RequiredArgsConstructor

public class PersonController {

    private final PersonService personService; // Corregido el nombre del servicio

    @PostMapping
    public void createPersona(@RequestBody Person person) {
        personService.createPersona(person);
    }
}

```

Figura 9. Código del archivo denominado PersonController.java

```

Archivo  Editar  Ver

package com.proyecto.apirest.Persona;

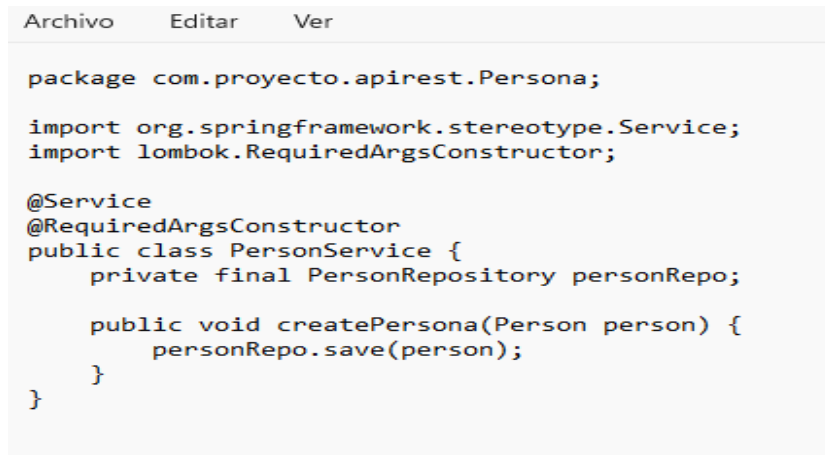
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface PersonRepository extends JpaRepository <Person, Integer> {

}

```

Figura 10. Código del archivo denominado PersonRepository.java



```
Archivo  Editar  Ver

package com.proyecto.apirest.Persona;

import org.springframework.stereotype.Service;
import lombok.RequiredArgsConstructor;

@Service
@RequiredArgsConstructor
public class PersonService {
    private final PersonRepository personRepo;

    public void createPersona(Person person) {
        personRepo.save(person);
    }
}
```

Figura 11. Código del archivo denominado PersonService.java

3.5. Componentes reutilizables

Los componentes reutilizables de las diferentes partes del código son:

Clase Person:

La clase Person representa una entidad de persona en el sistema.

Contiene atributos que representan las propiedades de una persona, como el nombre, apellido y email.

También incluye anotaciones de JPA (@Entity, @Id, @GeneratedValue, @Basic) que especifican cómo se debe mapear esta clase a una tabla en la base de datos y cómo se deben gestionar los atributos.

Utiliza la anotación @Data de Lombok para generar automáticamente los métodos toString(), equals(), hashCode() y getters/setters para todos los campos de la clase.

Además, se utilizan las anotaciones @AllArgsConstructor y @NoArgsConstructor de Lombok para generar automáticamente constructores con y sin argumentos, lo que facilita la creación de instancias de la clase.

Clase PersonController:

La clase PersonController es un controlador de Spring MVC (Modelo-Vista-Controlador) que maneja las solicitudes HTTP relacionadas con las personas en el sistema.

Utiliza las anotaciones @RestController y @RequestMapping para definir el punto de acceso base para todas las solicitudes HTTP relacionadas con personas, que es "/person".

Además, utiliza la anotación `@RequiredArgsConstructor` de Lombok para generar automáticamente un constructor con un argumento para la clase, lo que le permite inyectar el servicio `PersonService` en el controlador.

Método `createPersona`:

El método `createPersona` maneja las solicitudes HTTP POST para crear una nueva persona en el sistema.

Utiliza la anotación `@PostMapping` para mapear las solicitudes HTTP POST al método `createPersona`.

Toma un objeto `Person` como parámetro del cuerpo de la solicitud (`@RequestBody`) que representa la persona a crear.

Llama al método `createPersona` del servicio `PersonService` para realizar la operación de creación de persona.

Interfaz `PersonRepository`:

La interfaz `PersonRepository` extiende la interfaz `JpaRepository`, que es proporcionada por Spring Data JPA.

Utiliza la anotación `@Repository` para indicar que esta interfaz es un componente de repositorio Spring, lo que permite la inyección de dependencias y el manejo de excepciones relacionadas con la base de datos.

Define operaciones CRUD (Create, Read, Update, Delete) estándar para la entidad `Person` utilizando métodos heredados de `JpaRepository`, como `save`, `findById`, `findAll`, `delete`, etc.

Utiliza `Person` como el tipo de entidad y `Integer` como el tipo de su clave primaria en la definición de la interfaz `JpaRepository`.

Clase `PersonService`:

La clase `PersonService` es un componente de servicio de Spring que contiene la lógica de negocio relacionada con las personas en el sistema.

Utiliza la anotación `@Service` para indicar que esta clase es un componente de servicio de Spring, lo que permite la inyección de dependencias y la gestión de transacciones.

Además, utiliza la anotación `@RequiredArgsConstructor` de Lombok para generar automáticamente un constructor con un argumento para la clase, lo que le permite inyectar el repositorio `PersonRepository` en el servicio.

Método createPersona:

El método createPersona encapsula la lógica para crear una nueva persona en el sistema.

Toma un objeto Person como parámetro, que representa la persona a crear.

Llama al método save del repositorio PersonRepository para guardar la persona en la base de datos.

3.6. Buenas prácticas de codificación

Uso de Anotaciones de Spring:

Se utilizan anotaciones de Spring como @RestController, @Service, @Repository, @PostMapping, @RequestMapping, entre otras, de manera consistente y adecuada para definir y configurar los componentes de la aplicación.

Inyección de Dependencias:

Se utiliza la inyección de dependencias de Spring de manera apropiada, permitiendo la fácil configuración y manejo de las dependencias entre los componentes de la aplicación.

Uso de Lombok:

Se aprovecha el proyecto Lombok para reducir la verbosidad del código, generando automáticamente métodos toString(), equals(), hashCode(), y constructores, lo que ayuda a mantener el código más limpio y legible.

Separación de Responsabilidades:

Se emplea una clara separación de responsabilidades entre los diferentes componentes de la aplicación, como los controladores, servicios y repositorios, siguiendo el principio de responsabilidad única.

Documentación y Comentarios:

Se proporciona documentación adecuada mediante comentarios en el código, explicando el propósito y funcionamiento de las clases y métodos.

Uso de Interfaces:

Se utilizan interfaces para definir conexiones entre los componentes, lo que facilita la flexibilidad y la extensibilidad de la aplicación.

3.7. Patrones de diseño

Modelo (Model):

La clase Person (en el archivo Person.java) sirve como el modelo en el patrón MVC. Representa la estructura de datos de una persona en el sistema. Además, el archivo PersonRepository define las operaciones de acceso a datos relacionados con las personas.

Vista (View):

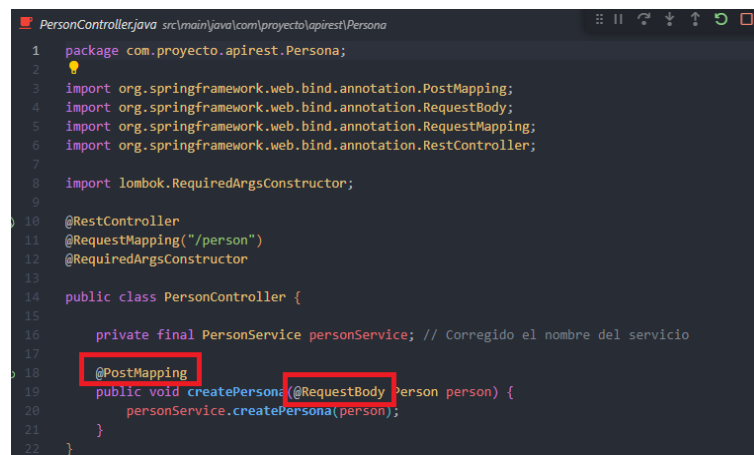
En una arquitectura de backend, la vista generalmente se representa como la respuesta HTTP devuelta al cliente. En el PersonController, los métodos anotados con @PostMapping y @RequestMapping manejan las solicitudes HTTP y definen la respuesta que se envía al cliente. Aunque no hay una representación visual como tal, estos métodos actúan como la "vista" lógica al definir cómo se presenta la información al cliente.

Controlador (Controller):

El PersonController actúa como el controlador en el patrón MVC. Se encarga de manejar las solicitudes HTTP entrantes, interactuar con el modelo correspondiente (en este caso, el servicio PersonService), y devolver una respuesta adecuada al cliente.

3.8. Pruebas unitarias

Para este caso las pruebas de funcionalidad e integridad se llevarán a cabo con postman, por ejemplo para el caso del archivo PersonController.java en el código le estamos diciendo que la solicitud se va a realizar a través de un Post y se tiene que esta solicitud viene en el cuerpo, y la información se va a almacenar en la base de datos persona, a continuación se observa esto en la imagen que muestra el código.



```
1 package com.proyecto.apirest.Persona;
2
3 import org.springframework.web.bind.annotation.PostMapping;
4 import org.springframework.web.bind.annotation.RequestBody;
5 import org.springframework.web.bind.annotation.RequestMapping;
6 import org.springframework.web.bind.annotation.RestController;
7
8 import lombok.RequiredArgsConstructor;
9
10 @RestController
11 @RequestMapping("/person")
12 @RequiredArgsConstructor
13
14 public class PersonController {
15
16     private final PersonService personService; // Corregido el nombre del servicio
17
18     @PostMapping
19     public void createPerson(@RequestBody Person person) {
20         personService.createPersona(person);
21     }
22 }
```

Figura 12. En esta imagen el recuadro rojo indica que la solicitud se

realiza a través de un Post y esta viene dada en el cuerpo

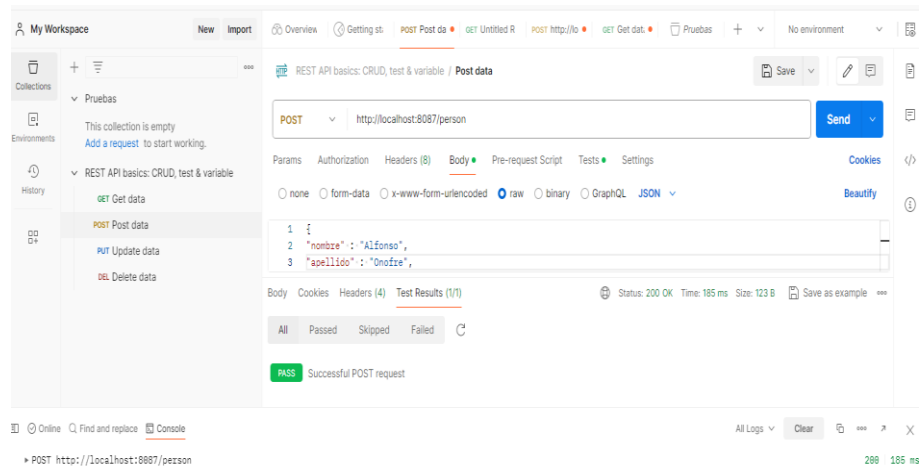


Figura 13. Se muestra que el envío de la petición fue exitoso.

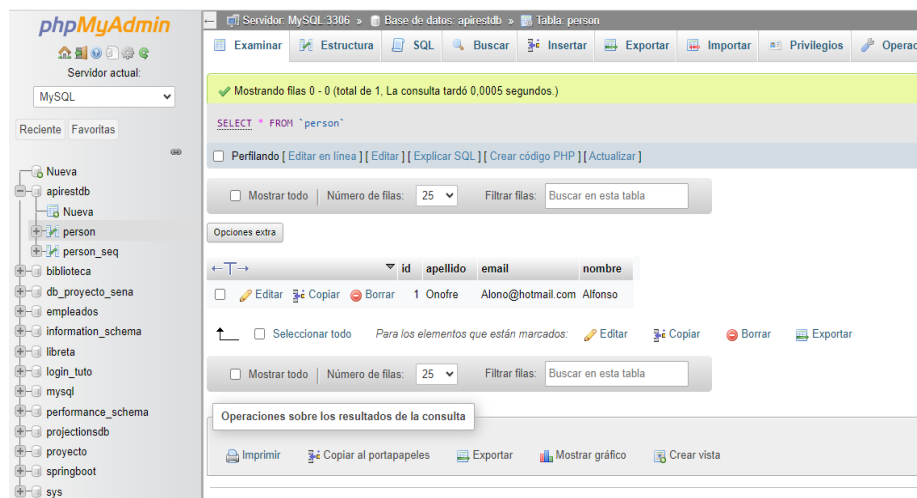
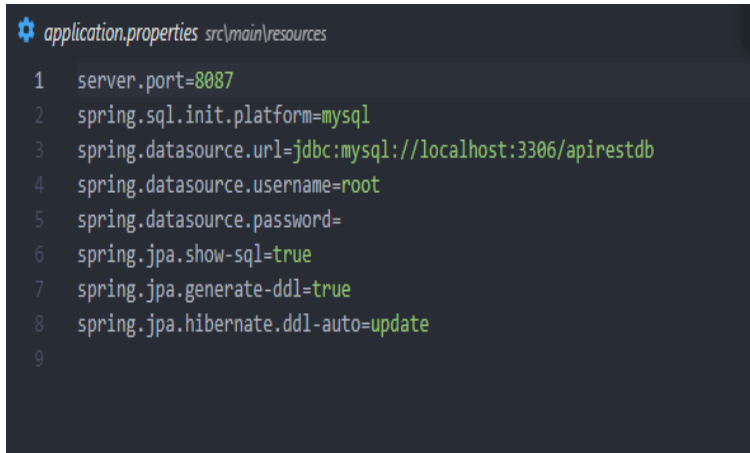


Figura 14. Comprobación de que los datos fueron almacenados en la base de datos apirestdb

De esta manera se realizarán pruebas para cada uno de los archivos de la API.

3.9. Configuración del servidor

En la imagen a continuación se puede observar la configuración del servidor utilizada para el backend de la API desarrollada.



```

1  server.port=8087
2  spring.sql.init.platform=mysql
3  spring.datasource.url=jdbc:mysql://localhost:3306/apirestdb
4  spring.datasource.username=root
5  spring.datasource.password=
6  spring.jpa.show-sql=true
7  spring.jpa.generate-ddl=true
8  spring.jpa.hibernate.ddl-auto=update
9

```

Figura 15. Configuración del servidor

3.10. Documentación del ambiente

Sistema Operativo

Se recomienda utilizar sistemas operativos compatibles con Java y las herramientas de desarrollo utilizadas, como:

- Windows 10
- macOS Catalina (o superior)
- Linux (Ubuntu, CentOS, etc.)

2. Herramientas y Dependencias

Java Development Kit (JDK)

Versión: JDK 21 o superior.

Instalación: Descargar e instalar desde el sitio web oficial de Oracle.

IDE de Desarrollo

Recomendado: IntelliJ IDEA, Eclipse, o cualquier otro IDE compatible con Spring Boot.

Instalación: Descargar e instalar desde los sitios web oficiales respectivos.

Gestión de Dependencias

Maven

Configuración del Proyecto

Clonación del Repositorio

Clonar el repositorio de la aplicación backend desde el repositorio Git remoto.