MANUAL DE USUARIO

Elaborado por: Enrique Condori.

Características del proyecto:

Entorno de desarrollo: IntelliJ IDEA

Librería: JAVA JDK 21.

Sprint Boot Version: 3.3.3

Dependencias utilizadas

- Spring Boot Starter Web
- Spring Boot Starter Validation
- Lombok
- Spring Boot DevTools
- Problem-sprint-web
- Data-jpa

Para ejecutar el proyecto primero debes realizar los sgtes paso:

1. Clonar el repositorio

Primero, necesitas clonar el repositorio en tu máquina local. Abre una terminal o consola y ejecuta el siguiente comando:

git clone https://github.com/kikePrograMer007/crud.git

Esto creará una carpeta llamada "crud" en tu directorio actual.

2. Abrir el proyecto en tu IDE

2.1 Para Intellij IDEA:

2.1.1 Abrir la herramienta: Abre IntellinJ IDEA.

2.1.2 Importar proyecto:

- Ve a `File > Open`.
- Se abrirá una ventana de Open Project, navega hasta el directorio donde clonaste el repositorio `crud`
- Selecciona la carpeta del proyecto y haz clic en `Ok`.

3. Esperar a que Maven Descargue las Dependencias

IntelliJ IDEA detectarán automáticamente el archivo pom.xml y comenzarán a descargar las dependencias necesarias. Esto puede tardar unos minutos dependiendo de la velocidad de tu conexión a Internet**.**

4. Abrir Mysql y Crear la base de datos en MySql con el nombre de "crud_db"

5. Crear usuario en MySql:

- User: admin

- Password: admin

6. Ejecutar el Proyecto

Ejecutar como aplicación Spring Boot:

- Haz clic derecho en la carpeta principal (crud).
- Selecciona Run > Run o Debug.

El servidor Spring Boot se iniciará y empezará a crear las tablas de la base de datos.

7. Verificar la Ejecución

Abre un navegador web y ve a http://localhost:8080/api/categories (o el puerto configurado en el proyecto) para verificar que la aplicación esté corriendo correctamente.

8. (en caso de errores) Resolver Problemas Comunes

- **Dependencias no resueltas:** Si algunas dependencias no se descargan correctamente, ve a `Project > Update Maven Project` para forzar la actualización.
- **Cambiar el puerto**: Si **8080** está ocupado, puedes cambiar el puerto en `application.properties` utilizando `server.port=PUERTO`.

EXPLICACIÓN DE LOS SERVICIOS O ENDPOINT

Se desarrollo un api rest con la funcionalidad de un **CRUD**, estas APIs son Categoría y Producto.

1. API Propia Configurada:

Category: http://localhost:8080/api/categories
Product: http://localhost:8080/api/products

2. Descripción del Desarrollo:

- Nuestra API propia fue configurada como un servicio RESTful utilizando Spring Boot. Esta API permite la gestión completa de las categorías y productos en el sistema a través de operaciones CRUD (Crear, Leer, Actualizar, Eliminar y Buscar).
- Esta API fue estructurada con la arquitectura hexagonal y clean arquitecture.
- Cuenta con dos tablas con relacion de uno a muchos.

3. Ejemplo de Uso:

GET /api/categories:

- **Descripción:** Retorna una lista de categorías.
- **Ejemplo**: http://localhost:8080/api/categories

GET /api/ categories /{id}:

- **Descripción:** Retorna los detalles de una categoría específica.
- **Ejemplo:** http://localhost:8080/api/categories/1

POST /api/categories:

- **Descripción:** Permite la creación de una nueva categoría.
- **Ejemplo:** http://localhost:8080/api/categories

- **Cuerpo de la Solicitud:**

```
{
   "name": "Vinos"
}
```

PUT /api/categories/{id}

- **Descripción:** Actualiza los detalles de una categoría existente.
- **Ejemplo:** http://localhost:8080/api/categories/1
- **Cuerpo de la Solicitud:**

```
{
    "name": "Postres-update"

}
```

DELETE /api/categories/{id}

- **Descripción:** Elimina una categoría existente.
- **Ejemplo:** http://localhost:8080/api/categories/1

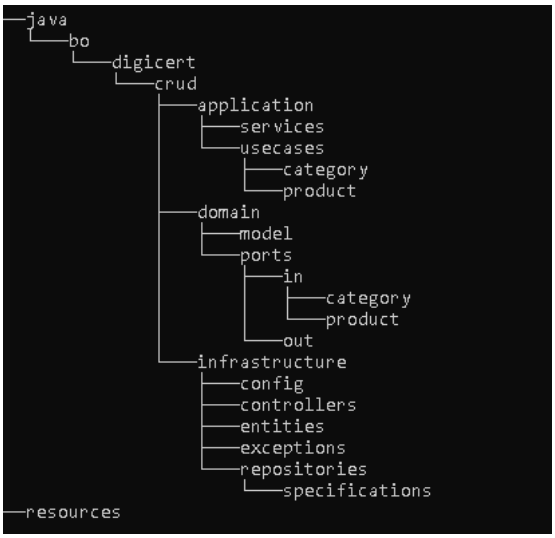
GET /api/categories/search?name=gaseosas
Params
Key : name
Value: tres

- **Descripción:** Busca el nombre de la categoría y devuelve los registros idénticos al criterio.
- **Ejemplo:** http://localhost:8080/api/categories/search

Nota: Los endpoints de Products están en la collection del repositorio. Lo puedes encontrar con el nombre "SPRINT-BOOT-CRUD.postman_collection.json"

Característica del proyecto

Arquitectura Hexagonal y uso del principio SOLID:



Arquitectura Hexagonal

Visión general de cómo los paquetes interactúan entre sí en la arquitectura de la aplicación:

La arquitectura hexagonal, también conocida como Ports and Adapters, es un patrón de diseño que busca mantener una separación clara de las responsabilidades en una aplicación, facilitando la adaptabilidad, escalabilidad y mantenibilidad del software. La arquitectura se organiza en tres capas principales:

Dominio: Esta capa contiene las entidades del dominio, que representan los conceptos clave del negocio y sus relaciones, así como la lógica de negocio asociada. Estas entidades son independientes de la infraestructura y la implementación, lo que permite centrarse en las reglas y restricciones del negocio.

Aplicación: Esta capa contiene los casos de uso, que representan las acciones o funcionalidades que la aplicación puede realizar. Los casos de uso coordinan la comunicación entre los puertos de entrada (interfaces que representan las acciones que se pueden realizar desde el exterior) y los puertos de salida (interfaces que representan las acciones que la aplicación puede realizar hacia el exterior, como interactuar con bases de datos o servicios externos).

Infraestructura: Esta capa contiene los adaptadores y la implementación de los puertos de salida, así como la configuración y la interacción con servicios externos. Los adaptadores son responsables de convertir las solicitudes externas en llamadas a los casos de uso y de convertir las respuestas de los casos de uso en respuestas comprensibles para los sistemas externos.

La arquitectura hexagonal se adhiere a **los principios SOLID:**

Single Responsibility Principle (SRP): Cada capa tiene una responsabilidad única y bien definida, lo que evita la mezcla de responsabilidades y facilita el mantenimiento del código.

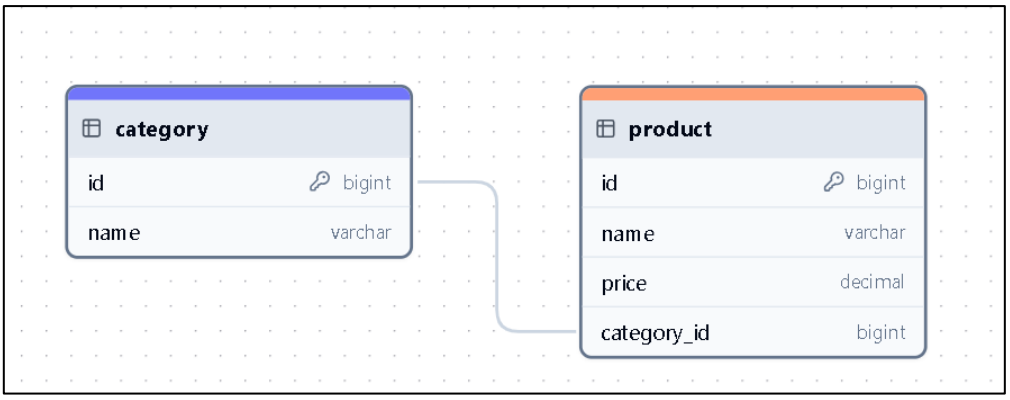
Open/Closed Principle (OCP): Las entidades y los casos de uso están abiertos a la extensión pero cerrados a la modificación. Si se necesita agregar una nueva funcionalidad, se puede hacer extendiendo los casos de uso o creando nuevos adaptadores sin modificar el código existente.

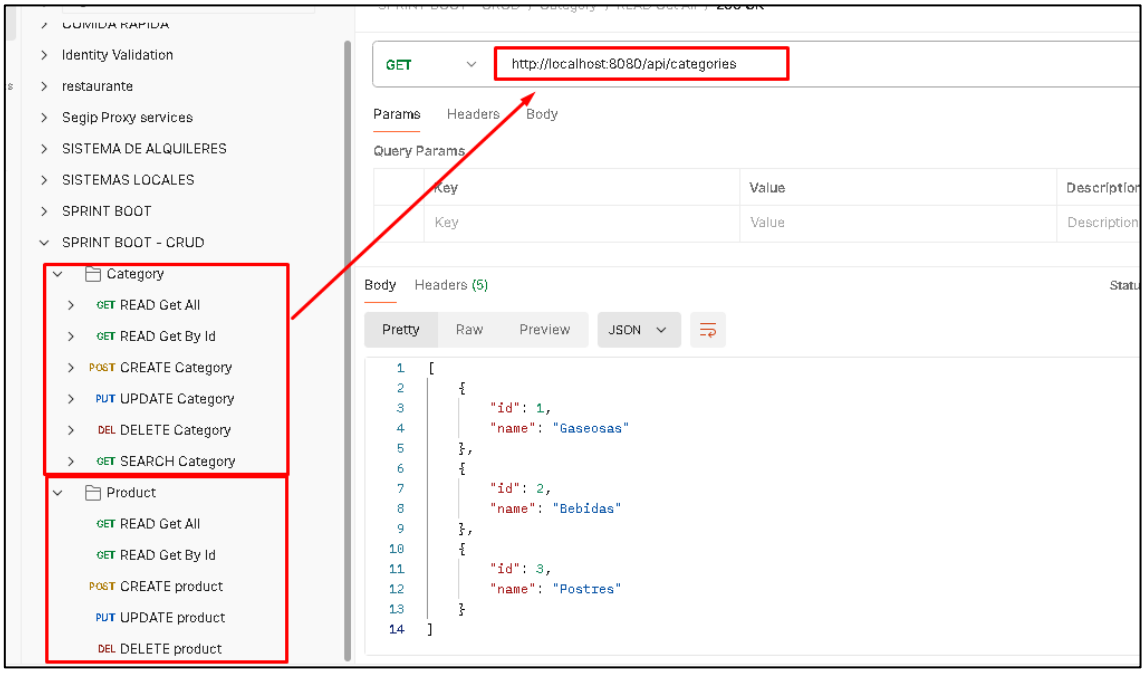
Liskov Substitution Principle (LSP): Los adaptadores y las implementaciones de los puertos deben ser sustituibles sin afectar el comportamiento del sistema, lo que permite cambiar fácilmente entre diferentes implementaciones de infraestructura o servicios externos.

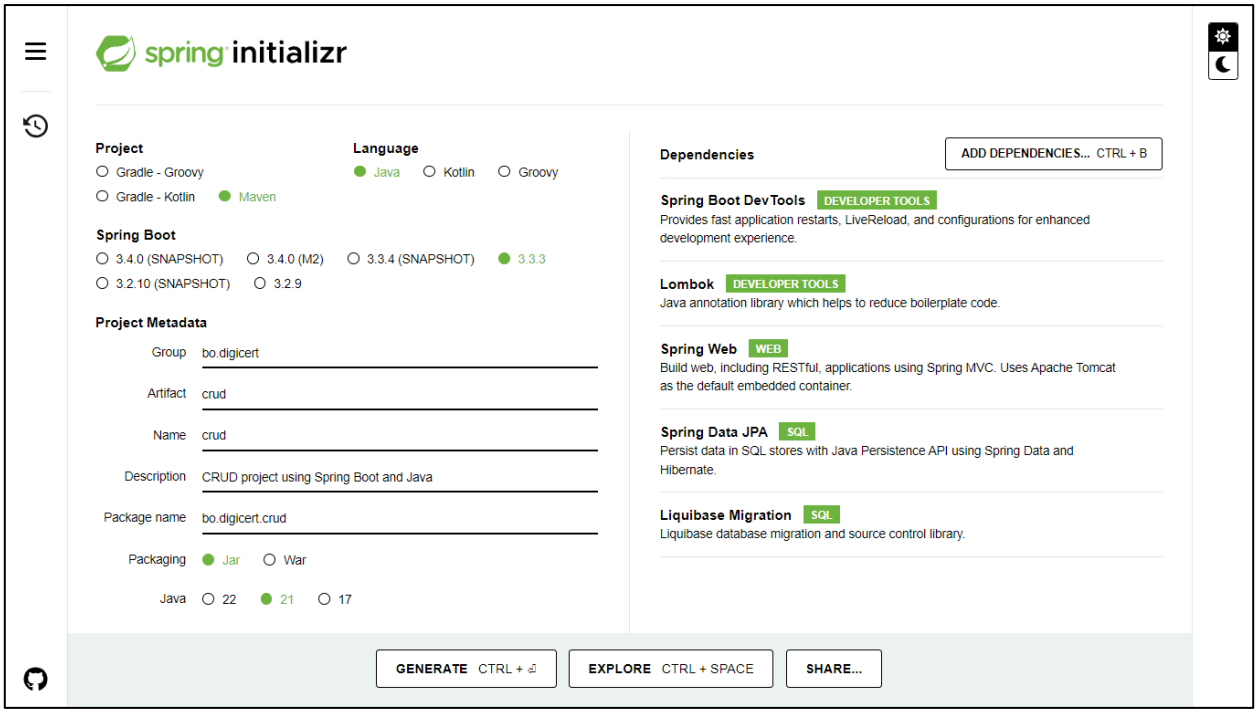
Interface Segregation Principle (ISP): Los puertos de entrada y salida definen interfaces pequeñas y específicas para cada funcionalidad, lo que facilita la implementación de adaptadores y evita depender de interfaces innecesariamente grandes.

Dependency Inversion Principle (DIP): Las dependencias entre las capas se invierten mediante la inyección de dependencias, lo que permite a las capas de dominio y aplicación depender de abstracciones en lugar de implementaciones concretas.

ANEXOS:


category
id
bigint
name
varchar
product
id
bigint
name
varchar
price
decimal
category_id
bigint


Identity Validation
restaurante
Segip Proxy services
SISTEMA DE ALQUILERES
SISTEMAS LOCALES
SPRINT BOOT
SPRINT BOOT - CRUD
Category
GET READ Get All
GET READ Get By Id
POST CREATE Category
PUT UPDATE Category
DEL DELETE Category
GET SEARCH Category
Product
GET READ Get All
GET READ Get By Id
POST CREATE product
PUT UPDATE product
DEL DELETE product
GET
http://localhost:8080/api/categories
Params
Headers
Body
Query Params
Key
Value
Description
Body
Headers (5)
Pretty
Raw
Preview
JSON
[
{
"id": 1,
"name": "Gaseosas"
},
{
"id": 2,
"name": "Bebidas"
},
{
"id": 3,
"name": "Postres"
}
]


spring initializr
Project
Gradle - Groovy
Gradle - Kotlin
Maven
Language
Java
Kotlin
Groovy
Spring Boot
3.4.0 (SNAPSHOT)
3.4.0 (M2)
3.3.4 (SNAPSHOT)
3.3.3
3.2.10 (SNAPSHOT)
3.2.9
Project Metadata
Group
bo.digicert
Artifact
crud
Name
crud
Description
CRUD project using Spring Boot and Java
Package name
bo.digicert.crud
Packaging
Jar
War
Java
22
21
17
Dependencies
ADD DEPENDENCIES... CTRL + B
Spring Boot DevTools
DEVELOPER TOOLS
Provides fast application restarts, LiveReload, and configurations for enhanced development experience.
Lombok
DEVELOPER TOOLS
Java annotation library which helps to reduce boilerplate code.
Spring Web
WEB
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.
Spring Data JPA
SQL
Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.
Liquibase Migration
SQL
Liquibase database migration and source control library.
GENERATE CTRL +
EXPLORE CTRL + SPACE
SHARE...