Centro Educativo Técnico Laboral Kinal

Materia: 5to Perito Informática

Profesor: Eduardo Hor

Sección: IN5CM

Manual de Procedimiento

Nombre: Josué Daniel Jolón Motta

Carné: 2022205

Fecha de Entrega:

06/02/2026

Documentación de Procedimiento

Sistema Repuestos Automotrices – API REST con Spring Boot

1. **Descripción general del proyecto**

En este proyecto desarrollé una API REST utilizando Spring Boot desde IntelliJ IDEA.
El objetivo principal fue crear un sistema que permitiera administrar empleados dentro de un sistema de repuestos automotrices mediante operaciones CRUD.

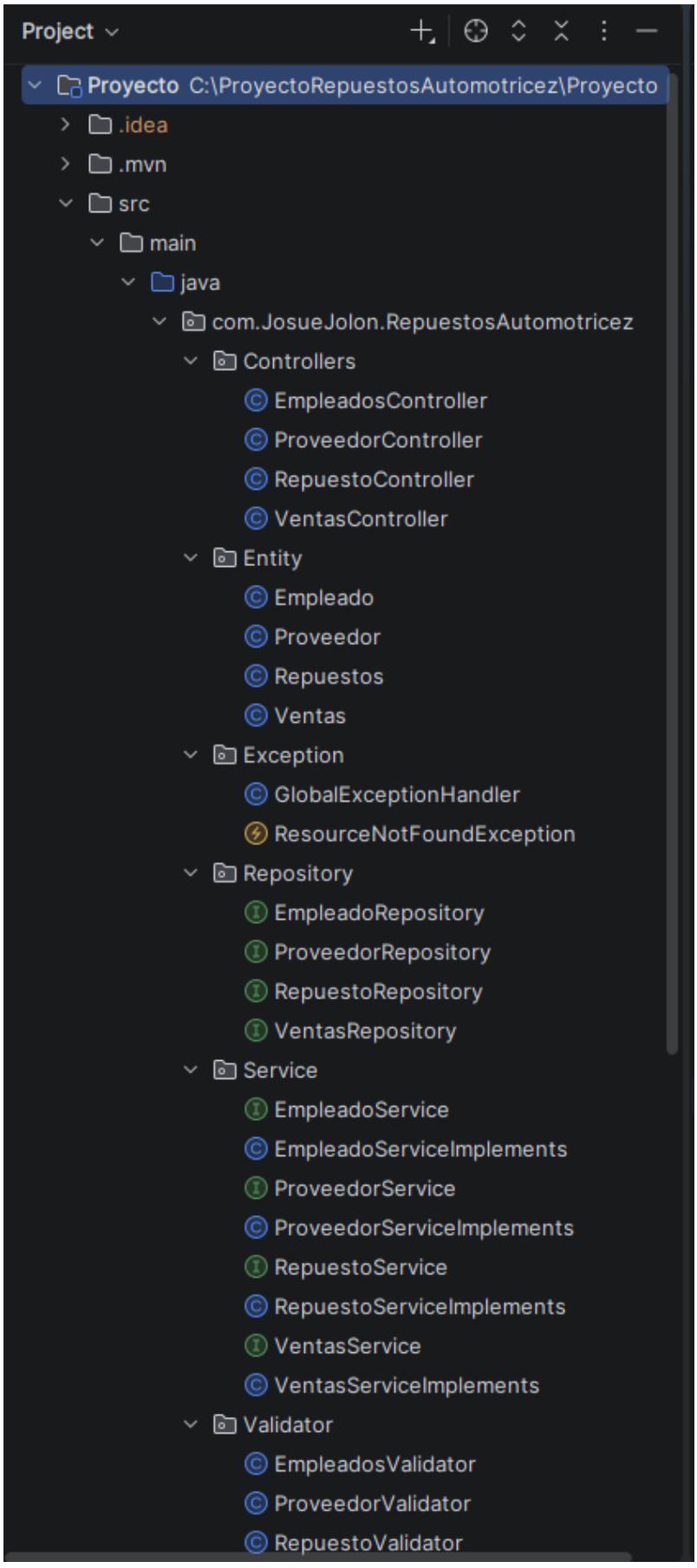
El proyecto permite:

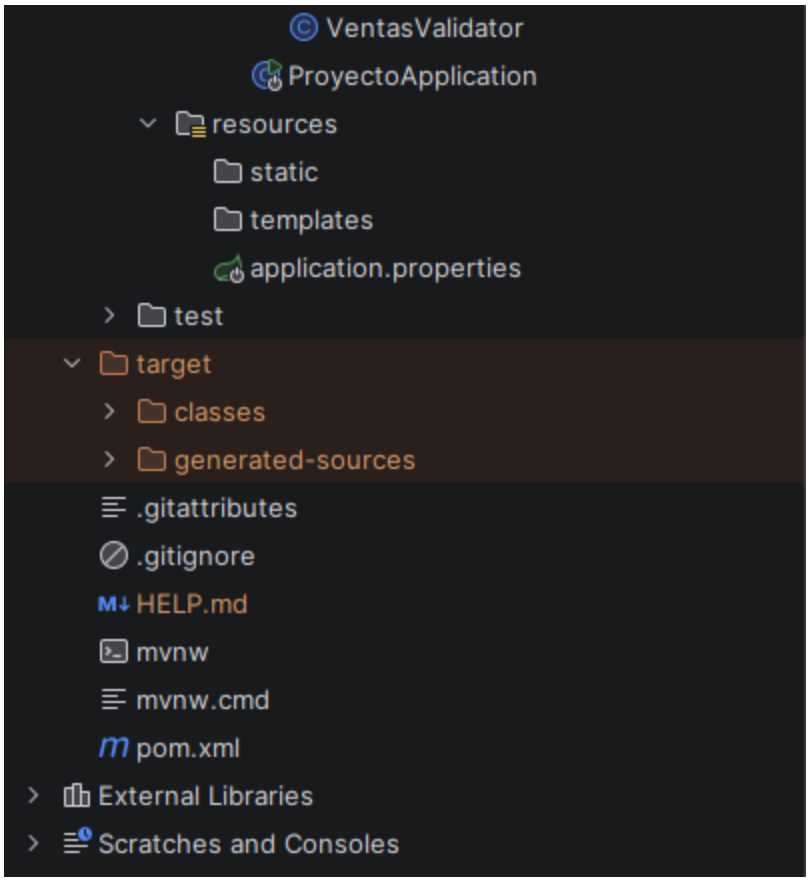
- Registrar empleados
- Mostrar todos los empleados
- Buscar empleados por ID
- Actualizarlos
- Eliminarlos

Para lograr esto utilicé una estructura basada en capas, separando responsabilidades para mantener el código ordenado y fácil de entender.

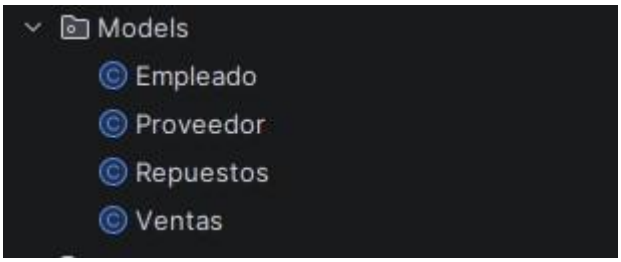
2. Estructura del Proyecto

El proyecto se organizó en paquetes principales para dividir el trabajo:


Project
Proyecto C:\ProyectoRepuestosAutomotricez\Proyecto
.idea
.mvn
src
main
java
com.JosueJolon.RepuestosAutomotricez
Controllers
EmpleadosController
ProveedorController
RepuestoController
VentasController
Entity
Empleado
Proveedor
Repuestos
Ventas
Exception
GlobalExceptionHandler
ResourceNotFoundException
Repository
EmpleadoRepository
ProveedorRepository
RepuestoRepository
VentasRepository
Service
EmpleadoService
EmpleadoServiceImplements
ProveedorService
ProveedorServiceImplements
RepuestoService
RepuestoServiceImplements
VentasService
VentasServiceImplements
Validator
EmpleadosValidator
ProveedorValidator
RepuestoValidator


VentasValidator
ProyectoApplication
resources
static
templates
application.properties
test
target
classes
generated-sources
.gitattributes
.gitignore
HELP.md
mvnw
mvnw.cmd
pom.xml
External Libraries
Scratches and Consoles

Models:



Aquí definí las clases que representan las tablas de la base de datos

por medio del @Entity

En esta parte:

- Creé la entidad **Empleado, Proveedor, Repuestos, Ventas**
- Definí sus atributos (id, nombre, apellido, etc.)
- Usé anotaciones para indicar que es una entidad de base de datos
- Definí los getters y setters para acceder a la información

El objetivo fue representar cómo se guarda la información dentro de la base de datos.

Repository:

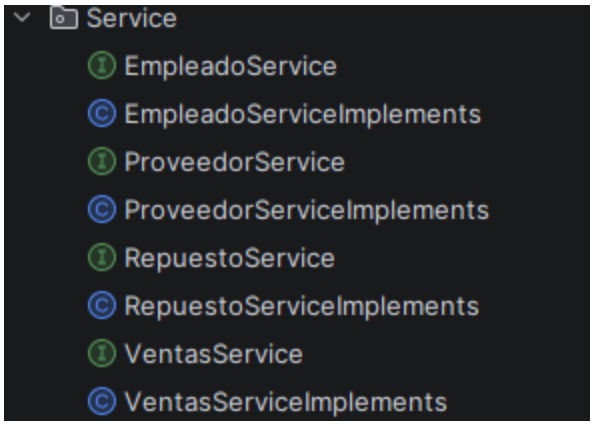
En esta capa creé las interfaces que se encargan de la conexión con la base de datos.

Aquí:

- Extendí JpaRepository
- Permití que Spring Boot genere automáticamente operaciones CRUD
- Evité escribir consultas SQL manualmente

Esta parte funciona como el puente entre el programa y la base de datos.

Service:



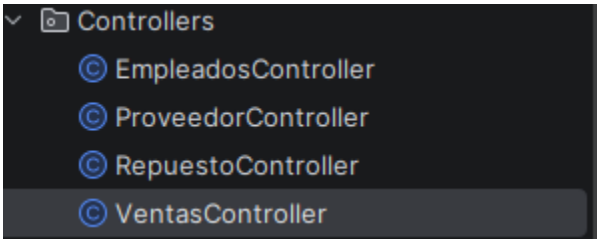
En esta capa desarrollé la lógica del sistema.

Aquí hice:

- Métodos para obtener empleados
- Métodos para buscar por ID
- Métodos para guardar nuevos registros
- Métodos para actualizar datos
- Métodos para eliminar empleados

En el service definimos los métodos que vamos a utilizar, en este caso seria el CRUD y la lógica detrás de los métodos se van en el serviceImplements de cada CRUD y el controller va a hacer el trabajo de mostrar y ser un controlador en general

Controllers:



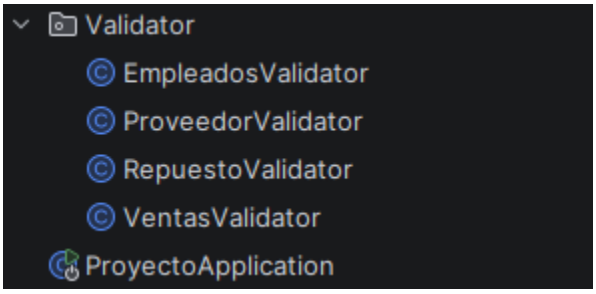
En esta parte creé los endpoints para que el usuario pueda interactuar con la API

Utilicé:

- @RestController
- @RequestMapping
- @GetMapping
- @PostMapping
- @PutMapping
- @DeleteMapping

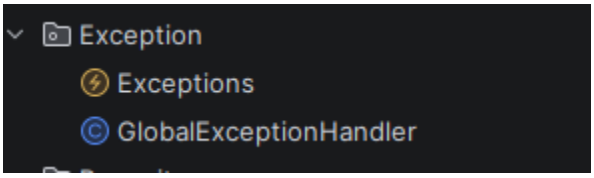
Cada método responde a una petición HTTP diferente.

Validator:



Dentro de esta capa utilizamos la notación de @Component que viene haciendo el mismo trabajo , es decir, la instancia que crean los endpoints para que se pueden inyectar en estas clases. El @Component se utiliza principalmente como ayuda o como auxiliar y en estos casos sirvió para validar ya que el @Controller , @Service, @Repository tienen un rol más especifico

Exception:



Dentro de esta capa manejo los errores de manera global . La clase de Exceptions es una clase la cual tenemos el manejo de los excepciones y en la cual vamos a extender de RuntimeException porque las excepciones en java son clases. El GlobalExceptionHandler es un manejador centralizado en excepciones que captura todas las excepciones en los controladores

Funciones principales:

- Listar empleados
- Buscar por ID
- Crear empleados

- Actualizar empleados
- Eliminar empleados

3. Funcionamiento General del Sistema

El flujo del sistema funciona así:

1. El usuario envía una petición desde Postman o navegador
2. El Controller recibe la solicitud
3. El Controller llama al Service
4. El Service ejecuta la lógica
5. El Repository accede a la base de datos
6. Se devuelve una respuesta al usuario

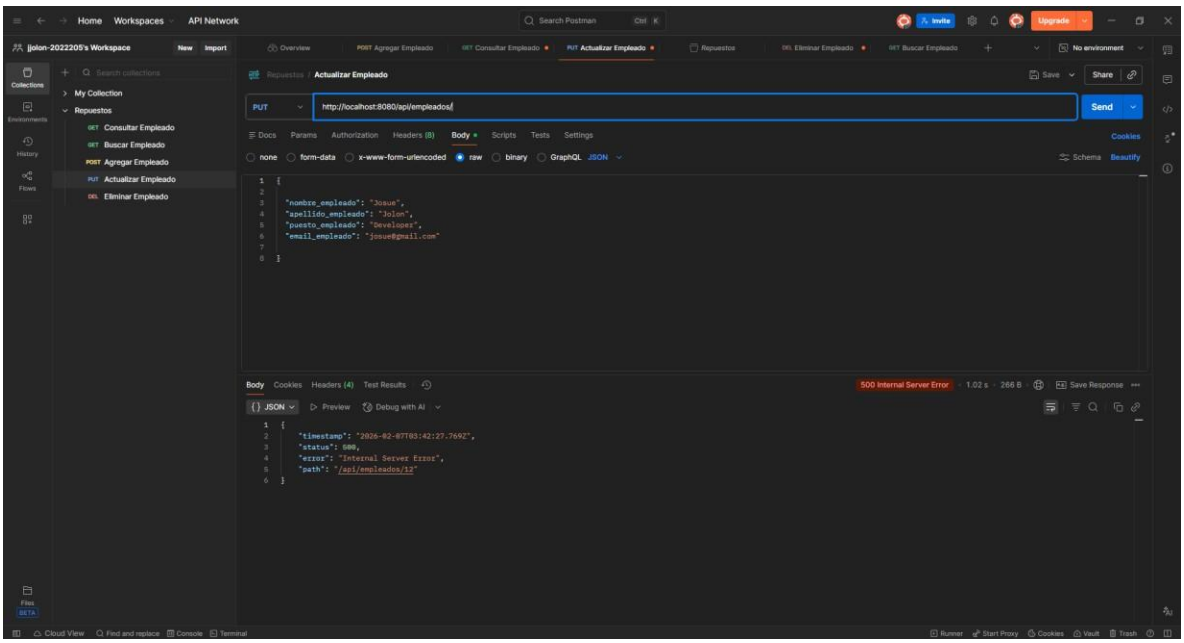
Esto permite tener un sistema organizado y fácil de mantener.

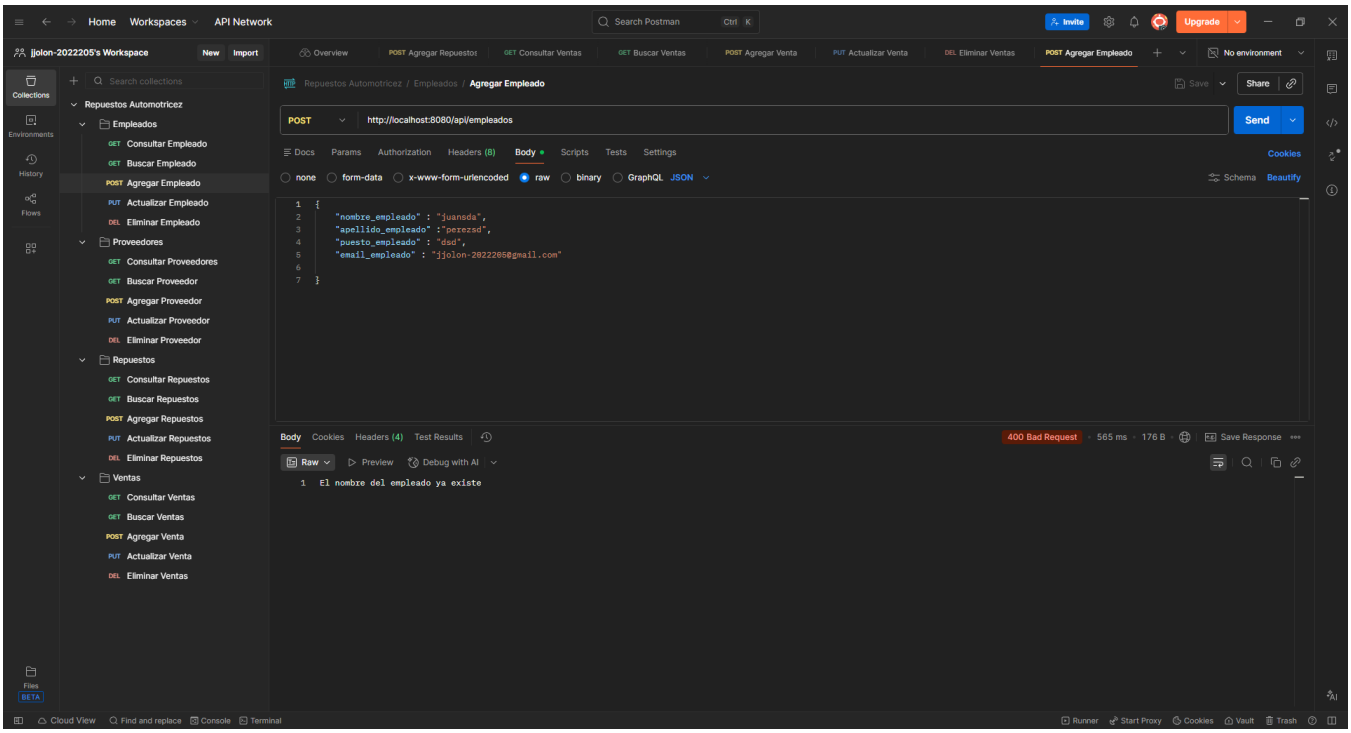
4. Base de Datos

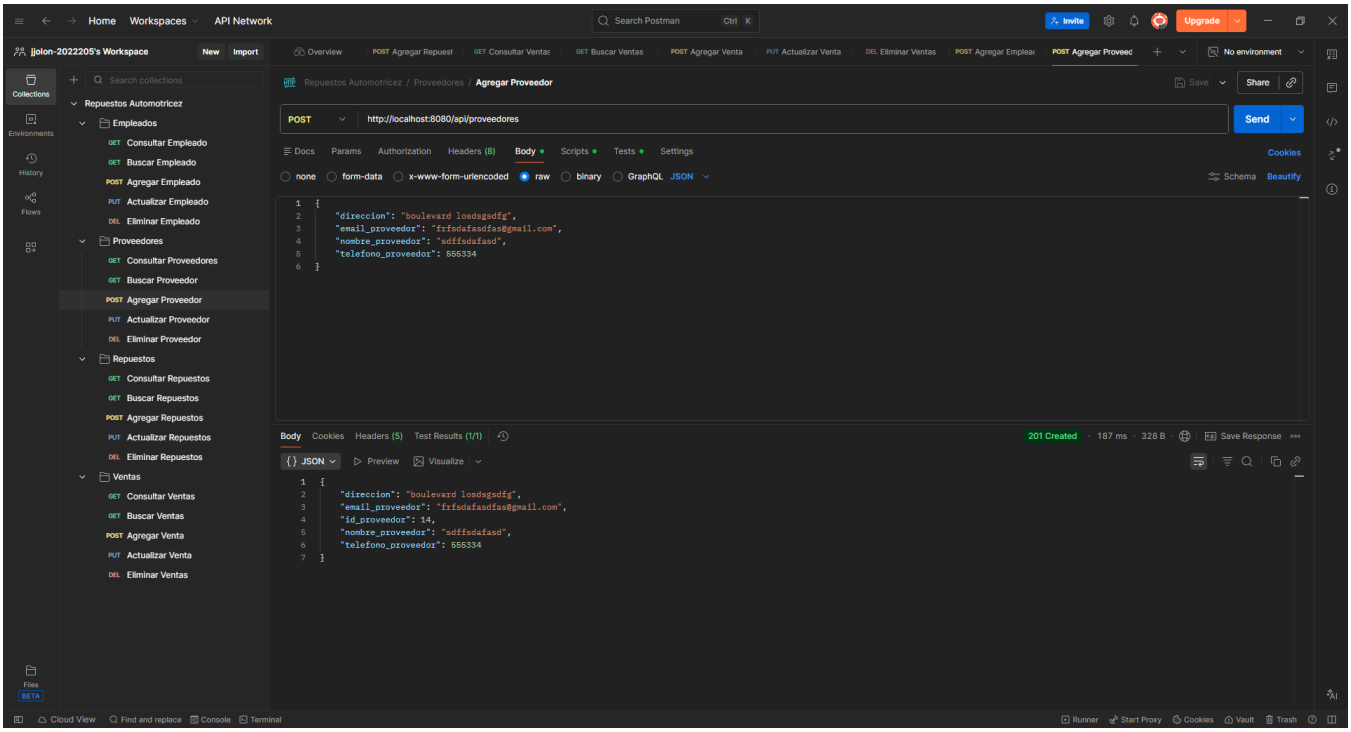
La configuración con la base de datos nos la proporciono el profesor, solo era de cambiar el usuario y contraseña para trabajar en nuestra casa

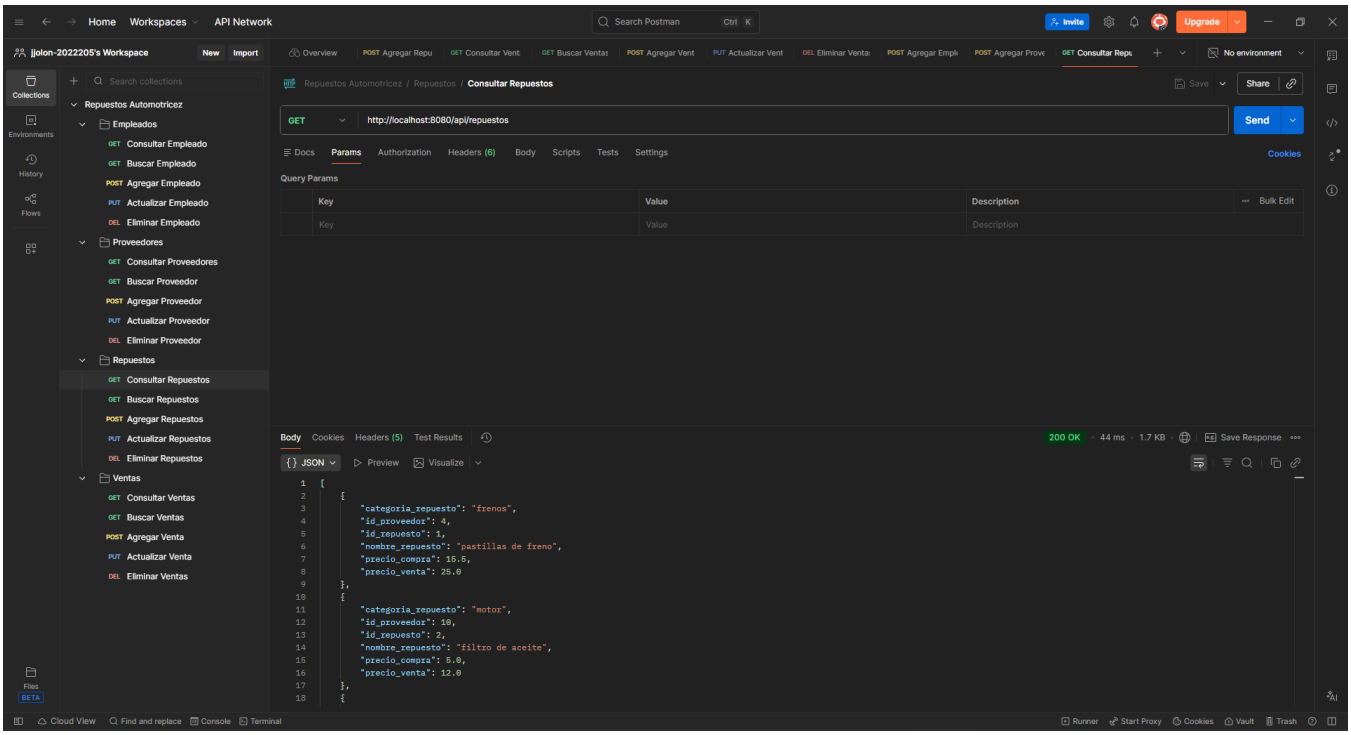
La información de empleados se guarda automáticamente gracias a JPA.

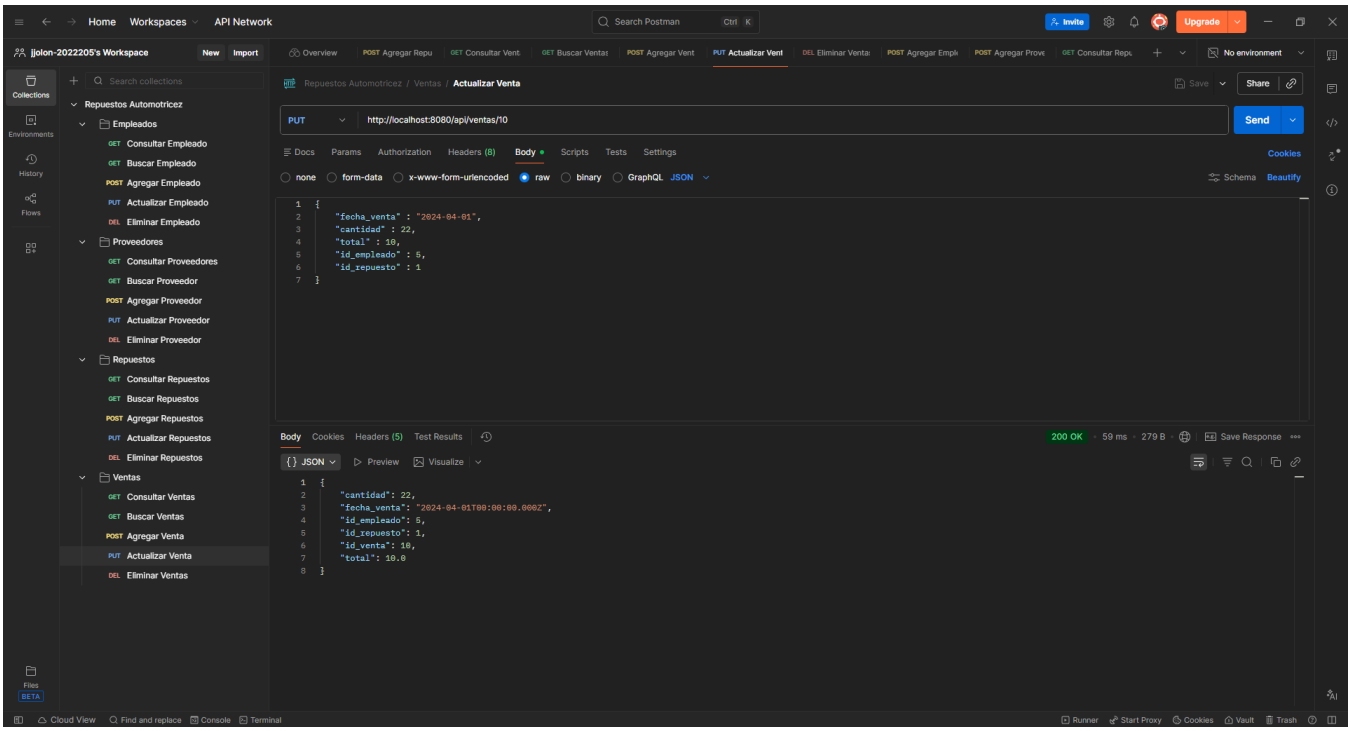
5. Pruebas del Sistema






Home Workspaces API Network
Search Postman
Invite
Upgrade
jjolon-2022205's Workspace
New Import
Collections
Environments
History
Flows
Files BETA
Repuestos Automotricez
Empleados
GET Consultar Empleado
GET Buscar Empleado
POST Agregar Empleado
PUT Actualizar Empleado
DEL Eliminar Empleado
Proveedores
GET Consultar Proveedores
GET Buscar Proveedor
POST Agregar Proveedor
PUT Actualizar Proveedor
DEL Eliminar Proveedor
Repuestos
GET Consultar Repuestos
GET Buscar Repuestos
POST Agregar Repuestos
PUT Actualizar Repuestos
DEL Eliminar Repuestos
Ventas
GET Consultar Ventas
GET Buscar Ventas
POST Agregar Venta
PUT Actualizar Venta
DEL Eliminar Ventas
Repuestos Automotricez / Proveedores / Agregar Proveedor
Save
Share
POST
http://localhost:8080/api/proveedores
Send
Docs Params Authorization Headers (8) Body Scripts Tests Settings
Cookies
none form-data x-www-form-urlencoded raw binary GraphQL JSON
Schema Beautify
{
"direccion": "boulevard losdsgsdfg",
"email_proveedor": "frfsdafasdfas@gmail.com",
"nombre_proveedor": "sdffsdafasd",
"telefono_proveedor": 555334
}
Body Cookies Headers (5) Test Results (1/1)
201 Created 187 ms 328 B
Save Response
JSON Preview Visualize
{
"direccion": "boulevard losdsgsdfg",
"email_proveedor": "frfsdafasdfas@gmail.com",
"id_proveedor": 14,
"nombre_proveedor": "sdffsdafasd",
"telefono_proveedor": 555334
}
Cloud View Find and replace Console Terminal
Runner Start Proxy Cookies Vault Trash


Home Workspaces API Network
Search Postman
Invite
Upgrade
jjolon-2022205's Workspace
New Import
Collections
Environments
History
Flows
Files BETA
Repuestos Automotricez
Empleados
GET Consultar Empleado
GET Buscar Empleado
POST Agregar Empleado
PUT Actualizar Empleado
DEL Eliminar Empleado
Proveedores
GET Consultar Proveedores
GET Buscar Proveedor
POST Agregar Proveedor
PUT Actualizar Proveedor
DEL Eliminar Proveedor
Repuestos
GET Consultar Repuestos
GET Buscar Repuestos
POST Agregar Repuestos
PUT Actualizar Repuestos
DEL Eliminar Repuestos
Ventas
GET Consultar Ventas
GET Buscar Ventas
POST Agregar Venta
PUT Actualizar Venta
DEL Eliminar Ventas
Repuestos Automotricez / Repuestos / Consultar Repuestos
Save
Share
GET
http://localhost:8080/api/repuestos
Send
Docs Params Authorization Headers (6) Body Scripts Tests Settings
Cookies
Query Params
Key Value Description Bulk Edit
Body Cookies Headers (5) Test Results
200 OK 44 ms 1.7 KB
Save Response
JSON Preview Visualize
[
{
"categoria_repuesto": "frenos",
"id_proveedor": 4,
"id_repuesto": 1,
"nombre_repuesto": "pastillas de freno",
"precio_compra": 15.6,
"precio_venta": 25.0
},
{
"categoria_repuesto": "motor",
"id_proveedor": 10,
"id_repuesto": 2,
"nombre_repuesto": "filtro de aceite",
"precio_compra": 5.0,
"precio_venta": 12.0
},
{
Cloud View Find and replace Console Terminal
Runner Start Proxy Cookies Vault Trash



Para probar el funcionamiento utilicé:

- Postman
- Realicé pruebas de:
- GET → listar registros
- GET por ID
- POST → crear
- PUT → actualizar
- DELETE → eliminar

Esto permitió verificar que cada endpoint funcionara correctamente.

6. Aprendizajes durante el desarrollo

Durante la realización del proyecto aprendí:

- Cómo estructurar proyectos Spring Boot
- Cómo crear APIs REST
- Cómo usar anotaciones de Spring
- Cómo conectar con MySQL
- Cómo separar lógica por capas

7. Conclusión del procedimiento

El desarrollo del sistema permitió implementar un CRUD completo utilizando Spring Boot.
Se aplicaron buenas prácticas como la separación por capas, uso de repositorios JPA y controladores REST.
El sistema quedó funcional y listo para ser probado mediante herramientas externas.

Link del repositorio:

https://github.com/josuejolon-2022205/ProyectoRepuestosAutomotricez.git