

SERVICIO DE LOGÍSTICA DE ENTREGA

ESTUDIANTES:

KAROL HERNÁNDEZ GRACIA

JOHN MORALES REVUELTAS

RAFAEL CORREDOR GAMBÍN

ANGIE JIMÉNEZ DORIA

ASIGNATURA:

PROGRAMACIÓN ORIENTADA A OBJETOS

DOCENTE:

DOMINGO GALEANO PUCHE

UNIVERSIDAD DE CARTAGENA

FACULTAD: INGENIERÍA

PROGRAMA:

INGENIERÍA DE SOFTWARE A DISTANCIA

SEDE: CENTRO TUTORIAL CERETÉ

SEMESTRE II

24 DE JUNIO DE 2023

Tabla de contenido

1. Introducción	3
2. Objetivos	4
3. Caso práctico	5
4. Marco teórico	6
5. Desarrollo	7
• Diseño del modelo de entidad-relación utilizando UML	7
• Implementación del crud	8
• Identificar las etiquetas	11
• Conexión a My SQL Workbench	12
• Demostración en postman	12
6. Conclusiones	15

Introducción

En este proyecto se desarrollará una API Backend para el servicio de logística de entrega, para gestionar pedidos, realizar búsquedas y acceder a información relevante sobre los envíos. Mediante la implementación de un CRUD completo para las entidades del sistema. Además, se diseñará y creará un modelo de entidad-relación utilizando el Lenguaje de Modelado Unificado (UML), lo cual será útil para establecer tablas y relaciones entre las entidades. Asimismo, se busca que la API cumpla con todas las necesidades y funcionalidades específicas establecidas. Este proyecto demostrará la realización exitosa de una API Backend para satisfacer las exigencias de gestión logística, ofreciendo una solución eficaz para optimizar el proceso de entrega de pedidos.

Objetivos

- Diseñar y crear el modelo de entidad-relación utilizando el Lenguaje de Modelado Unificado (UML). Este modelo servirá como base para definir la estructura y relaciones entre las entidades en el Backend.
- Implementar un CRUD (crear, leer, actualizar, eliminar, buscar) para todas las entidades del sistema, Usuarios, Empleados y pedidos. Esto permitirá realizar operaciones básicas de gestión de datos en cada entidad.
- Desarrollar la capacidad de búsqueda en la API para los parámetros (Id y Nombre) que contenga cada entidad. Esto permitirá a los usuarios buscar pedidos por diferentes criterios, como código del pedido, id del empleado, nombre de producto, entre las diferentes tablas.
- Garantizar que la API satisfaga todas las necesidades y funcionalidades especificadas en el proyecto de logística de entrega.

Caso practico

Desarrollo de Api Backend – Crearan CRUD de Consulta con todas las entidades -

Se podrá Buscar por: parámetros pedidos como lo son id y Nombre, Agregaran las entidades que crean necesarios, creando el modelo de entidad relación para basarse en este y crear las entidades en el Backend.

La funcionalidad del Servicio de logística de entrega, es que debe recibir una lista de pedidos, en dicha lista el usuario debe escoger uno de los pedidos y verificar donde es la entrega, cual es nombre del paquete, el costo y la modalidad de pago. Y un tiempo estimado de entrega.

Tendrán en cuenta:

- Roles.
- Usuarios.
- Pedidos.
- Tiempo de entrega
- Código del pedido (Único)
- Estado.
- Seguimiento.
- Repartidor
- Numero de guía

Marco teórico

- **API:** (Application Programming Interface) es un conjunto de reglas, protocolos y herramientas que se utilizan para desarrollar software y permitir la comunicación entre diferentes aplicaciones. En términos generales, una API define cómo otras aplicaciones pueden interactuar con la aplicación o el servicio que proporciona la API.
- **MySQL Workbench:** es una herramienta de diseño y administración de bases de datos de código abierto que se utiliza para crear y administrar bases de datos MySQL. Permite a los usuarios diseñar, modelar y visualizar bases de datos, así como crear, modificar y eliminar tablas, vistas, procedimientos almacenados y otros objetos de base de datos.
- **El Lenguaje de Modelado Unificado:** (UML, por sus siglas en inglés) es un lenguaje de modelado visual utilizado en la programación orientada a objetos para representar sistemas de software. UML se utiliza para describir las estructuras y comportamientos de los sistemas de software en términos de clases, objetos, relaciones, estados y eventos.

Desarrollo

En este proyecto vamos a mostrar la realización de un proyecto de API en IntelliJ en colaboración como My SQL Workbench, donde primeramente se revisan las necesidades del proyecto como lo son, que el servicio de logística de entrega, debe recibir una lista de pedidos, en dicha lista el usuario debe escoger uno de los pedidos y verificar donde es la entrega, que contiene el paquete, el costo y la modalidad de pago, y un tiempo estimado de entrega. Con las anteriores necesidades se plantea un Lenguaje de Modelado Unificado (UML), donde se van a presentar tres tablas:

La tabla **Usuario** que tiene los siguientes atributos:

- **ID:** identificador único del usuario.
- **Rol:** rol del usuario en el sistema.
- **Nombre:** nombre del usuario.

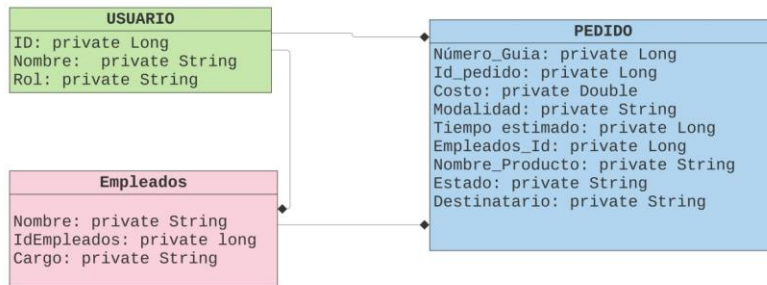
Seguidamente la tabla **Empleado** que tiene los siguientes atributos:

- **Nombre:** nombre del empleado.
- **Cargo:** cargo del empleado en la empresa.
- **IdEmpleados:** identificador único del empleado.

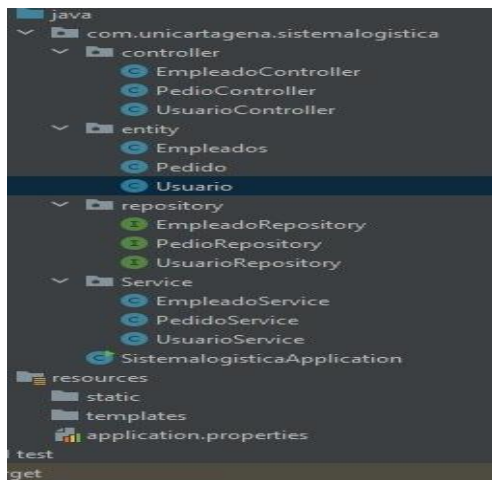
Y, por último, pero no menos importante la tabla **Producto**

- **Número_Guia:** identificador único para cada envío.
- **Id pedido:** identificador único del pedido relacionado con el envío.
- **Costo:** costo del envío.
- **Modalidad:** forma de envío (por ejemplo, terrestre, aéreo, marítimo).
- **Tiempo estimado:** tiempo estimado de entrega.
- **Empleados Id:** identificador único del empleado encargado del envío.
- **Nombre Producto:** nombre del producto a enviar.
- **Estado:** estado actual entregado). del envío (por ejemplo, en tránsito)
- **Destinatario:** información del destinatario del envío.

Lo cual nos da el siguiente resultado



Este CRUD nos muestra como cada una tiene relación entre sí, y se visualiza por medio de una segregación. Después de esto se empezó a ejecutar la creación de entidades, controladores, repositorios y servicios, de la cual tomamos como ejemplo el siguiente video <https://www.youtube.com/watch?v=9XoaU5IMkRY&t=25s> , en el cual se logró crear los métodos `create`, `update`, `delete`, `findAll`, `FindById`.



- **Ejemplo de las entidades (clase Usuario):**


```

1 package com.unicartagena.sistemalogistica.entity;
2
3
4 import ...
5
6
7
8
9 3 usages
10 @Entity
11 @Data
12 @Table(name = "usuarios")
13 public class Usuario {
14
15     @Id
16     @GeneratedValue(strategy = GenerationType.IDENTITY)
17     private Long id;
18
19     private String nombre;
20
21     private String rol;
22
23 }

```

En la anterior clase se debe identificar como `@Entity` donde se comprende que es una entidad, la cual contiene nuestras variables, otra información importante es la de generar un id automáticamente.

- **Ejemplo de la conexión a base de datos en la clase application. properties**

```

Controller.java x EmpleadoController.java x application.properties x PedioRep
1 spring.datasource.url=jdbc:mysql://localhost:3306/baseneu
2 spring.datasource.username=root
3 spring.datasource.password=root
4 spring.datasource.driver-class-name=com.mysql.jdbc.Driver
5 spring.jpa.show-sql:true
6 spring.jpa.hibernate.ddl-auto:update
7

```

- **Ejemplo de los repositorios (Clase Usuario)**

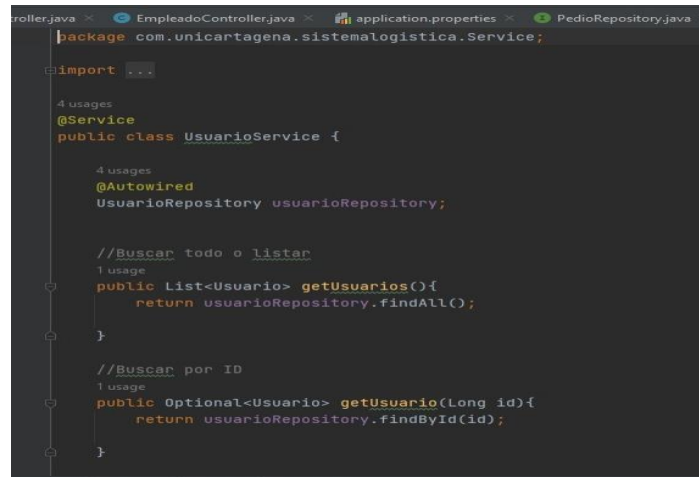
```

1 package com.unicartagena.sistemalogistica.repository;
2
3 import ...
4
5
6
7 3 usages
8 @Repository
9 public interface UsuarioRepository extends JpaRepository<Usuario, Long> {
10
11 }

```

También recibe una etiqueta o identificación que es `@Repository`, aquí se busca extender al objeto y el tipo de dato que es la variable identificadora.

- **Ejemplo de los servicios (Clase Usuario)**



```

package com.unicartagena.sistemalogistica.Service;

import ...

@Service
public class UsuarioService {

    @Autowired
    UsuarioRepository usuarioRepository;

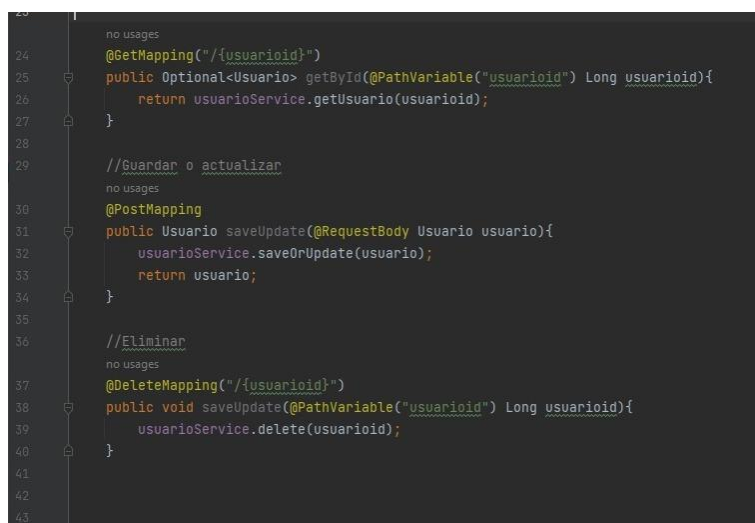
    //Buscar todo o listar
    public List<Usuario> getUsuarios(){
        return usuarioRepository.findAll();
    }

    //Buscar por ID
    public Optional<Usuario> getUsuario(Long id){
        return usuarioRepository.findById(id);
    }
}

```

Es aquí donde se generan los métodos que el proyecto va a ejecutar como lo vemos en el proyecto, nos muestra el método listar, que buscar a todo lo que se encuentre en base de datos.

- **Ejemplo de los controladores (Clase Usuario)**

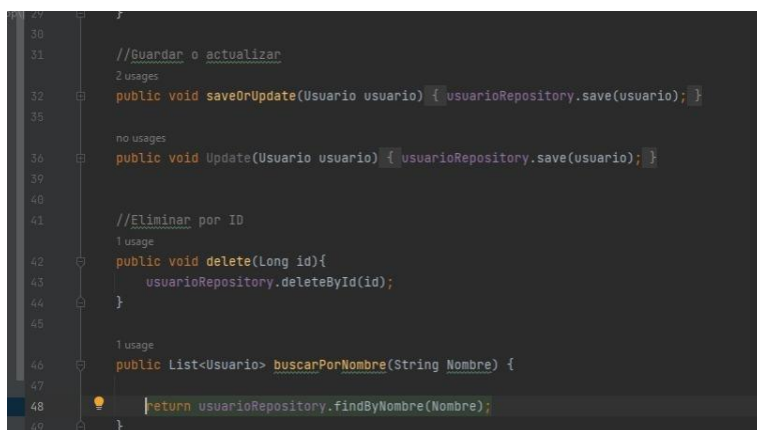


```

no usages
24 @GetMapping("/{usuarioid}")
25 public Optional<Usuario> getById(@PathVariable("usuarioid") Long usuarioid){
26     return usuarioService.getUsuario(usuarioid);
27 }
28
29 //Guardar o actualizar
no usages
30 @PostMapping
31 public Usuario saveUpdate(@RequestBody Usuario usuario){
32     usuarioService.saveOrUpdate(usuario);
33     return usuario;
34 }
35
36 //Eliminar
no usages
37 @DeleteMapping("/{usuarioid}")
38 public void saveUpdate(@PathVariable("usuarioid") Long usuarioid){
39     usuarioService.delete(usuarioid);
40 }
41
42
43

```

Luego de la anterior guía vista (video) se buscó una solución para el método buscar por nombre, A continuación, se muestra su funcionalidad:



```

30
31 //Guardar o actualizar
32 2 usages
33 public void saveOrUpdate(Usuario usuario) { usuarioRepository.save(usuario); }
34
35 no usages
36 public void Update(Usuario usuario) { usuarioRepository.save(usuario); }
37
38
39
40
41 //Eliminar por ID
42 1 usage
43 public void delete(Long id){
44     usuarioRepository.deleteById(id);
45 }
46
47 1 usage
48 public List<Usuario> buscarPorNombre(String Nombre) {
49     return usuarioRepository.findByNombre(Nombre);
50 }

```

A continuación, se contextualiza etiquetas utilizadas en el código. Este tipo de anotaciones hacen que el funcionamiento del proyecto sea el adecuado y cumpla con las funciones deseadas para cada uno de los requerimientos

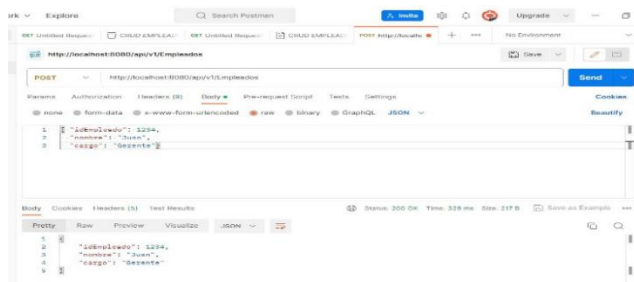
- @RequestMapping es una anotación que se utiliza para asignar una URL o una ruta a un controlador y especificar qué tipo de solicitud HTTP debe manejar.
- ResponseEntity es una clase que representa toda la respuesta HTTP, incluyendo el cuerpo de la respuesta, las cabeceras y el código de estado. Se utiliza para devolver una respuesta personalizada desde un controlador.
- @GetMapping es una anotación que se utiliza para asignar una URL o una ruta a un controlador que maneja solicitudes HTTP GET.
- @RequestParam es una anotación que se utiliza para obtener parámetros de solicitud de una solicitud HTTP.
- @Autowired es una anotación que se utiliza para inyectar dependencias en un componente de Spring.

- @PostMapping es una anotación que se utiliza para asignar una URL o una ruta a un controlador que maneja solicitudes HTTP POST.
- @DeleteMapping es una anotación que se utiliza para asignar una URL o una ruta a un controlador que maneja solicitudes HTTP DELETE.
- @Data es una anotación de Lombok que genera automáticamente los métodos getter, setter, equals, hashCode y toString para las variables de instancia de una clase.
- **Ejemplo de como se muestra en MY SQL cuando se crea dicha conexión**

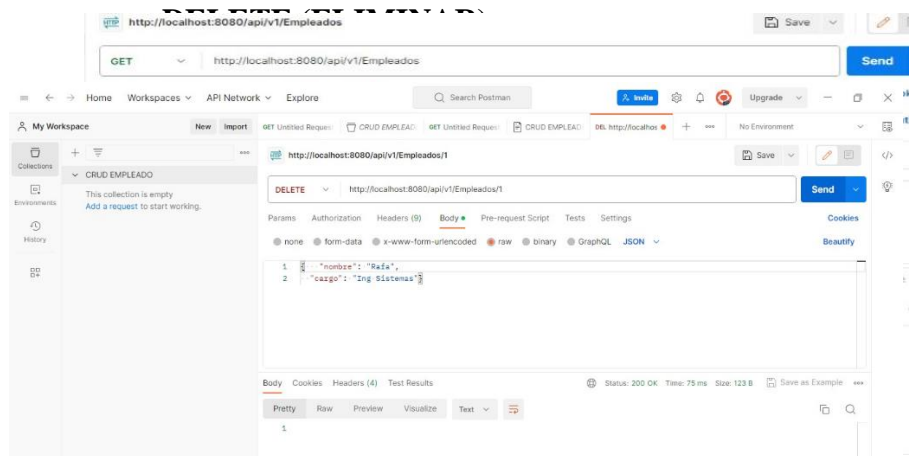


Muestra de cómo se visualiza el proyecto en postman

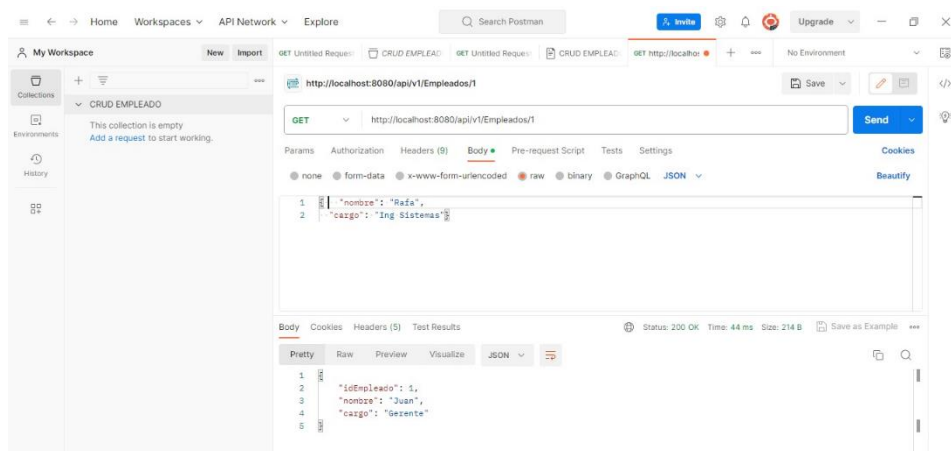
- **POST (CREAR)**



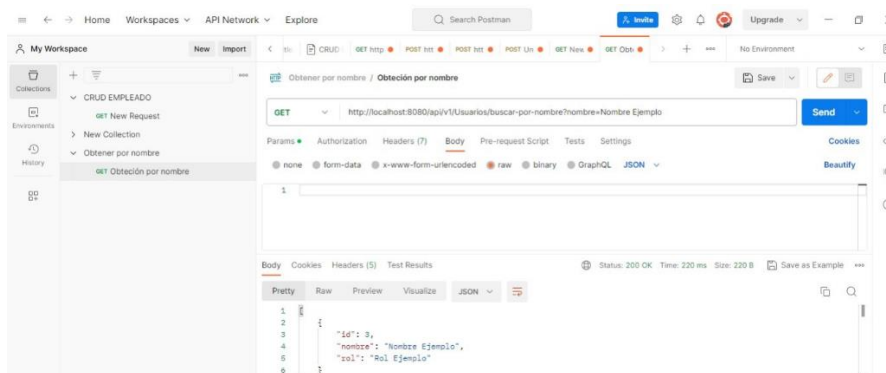
- **GET (OBTENER O LISTAR POR TODO)**



- **GET (BUSCAR POR ID)**



- **GET (BUSCAR POR NOMBRE)**



- **DELETE**

The screenshot shows the Postman interface with a DELETE request configured. The left sidebar lists the API collection 'CRUD USUARIO' with endpoints: 'GET usuario FIND ALL', 'GET usuario FIND NOMBRE', 'GET usuario FIND ID', 'DEL usuario DELETE' (selected), 'POST usuario CREATE', and 'PUT usuario UPDATE'. The main panel shows the 'DEL usuario DELETE' endpoint with the URL 'http://localhost:8080/api/v1/Usuarios/2'. The 'Send' button is visible. Below the URL bar, the 'Params' tab is active, showing a table with columns 'Key', 'Value', and 'Description'. The 'Body' tab is also visible, showing a status of '200 OK' and a response time of '11 ms'.

Key	Value	Description
Key	Value	Description

PUT (UPDATE)

The screenshot shows the Postman interface with a PUT request configured. The left sidebar lists the API collection 'CRUD USUARIO' with endpoints: 'GET usuario FIND ALL', 'GET usuario FIND NOMBRE', 'GET usuario FIND ID', 'DEL usuario DELETE', 'POST usuario CREATE', and 'PUT usuario UPDATE' (selected). The main panel shows the 'PUT usuario UPDATE' endpoint with the URL 'http://localhost:8080/api/v1/Usuarios/modificar'. The 'Send' button is visible. Below the URL bar, the 'Body' tab is active, showing a status of '200 OK' and a response time of '15 ms'. The body is formatted as JSON, showing the updated user data:

```
{  "id": 1,  "nombre": "Jefe",  "rol": "Auditor"}
```

Conclusión

Se logró desarrollar una API Backend funcional para el servicio de logística de entrega, que brinda a los usuarios la capacidad de gestionar pedidos, realizar búsquedas y acceder a información sobre los envíos. Cumple con los requerimientos establecidos y proporciona una solución eficiente para las necesidades de gestión logística. Se llevó cabo este servicio de logística de entrega, que permitió recibir una lista de pedidos y seleccionar uno de ellos para verificar detalles importantes, como la dirección de entrega, los contenidos del paquete, el costo, la modalidad de pago y el tiempo estimado de entrega. Fue posible implementar un CRUD para todas las entidades del sistema, esto permitió realizar operaciones básicas de gestión de datos, como crear nuevos registros, leer información existente, actualizar datos y eliminar registros.