




|   |   |                        |
|---|---|------------------------|
|    | <p>ESCUELA DE INGENIERÍA DE SISTEMAS E<br/>INFORMÁTICA - ENTORNOS DE PROGRAMACION</p> <p>CLASES ENLAZADAS – VENTAS - Usuarios</p> | <p>I Semestre 2024</p> |
|---|---|------------------------|

## API REST (Microservicio) SPRING BOOT – MySQL

### Clases enlazadas.

El proyecto a desarrollar se descomponer en microservicios, cada uno de ellos tiene una responsabilidad y funcionalidad dentro del sistema general. Las funcionalidades se dividen en los siguientes componentes:

#### 1. Microservicio Usuario y de Acceso de Usuario.

En este módulo se debe dar acceso al sistema y a la vez tendremos una fase de frontend y otra de backend que debemos implementar por separado o agrupar según lo que deseemos.

En este ejercicio vamos a ver un ejemplo de microservicio en el backend que nos permite hacer las operaciones básicas de una tabla.

Crearemos una API RESTful utilizando métodos HTTP para operaciones CRUD (Crear, Recuperar, Actualizar y Eliminar) en Spring Boot junto con la base de datos MYSQL. Spring Boot es un marco de código abierto basado en Java para crear aplicaciones empresariales.

### Requerimientos

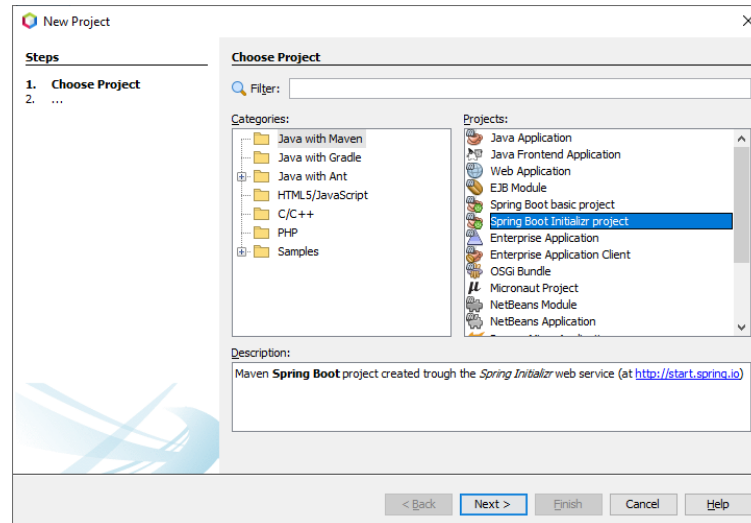
1. Maven 3.0+
2. IDE(Eclipse or IntelliJ)
3. JDK 1.8+
4. MYSQL como servidor de Base de Datos
5. Documentación con Swagger para Probar

### Contenido

1. Cree el proyecto Spring Boot.
2. Revisar y/o crear la base de datos MYSQL y defina sus configuraciones en el proyecto.
3. Crear clase de modelo de las entidades.
4. Crear los repositorios de datos JPA
5. Crear clase de servicio
6. Crear clase de controladores.
7. Documentación del Proyecto (Swagger).
8. Compile y ejecute el proyecto.

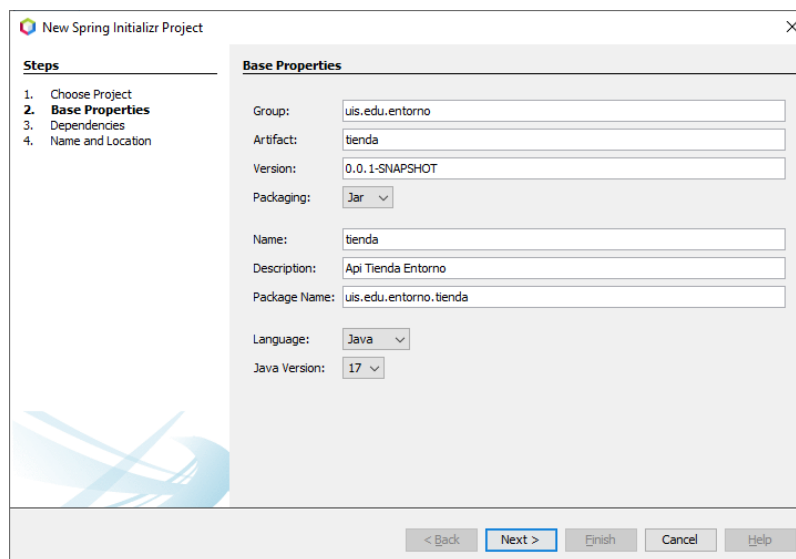
## 1. Crear el proyecto Spring Boot.




Desde Netbeans creamos un nuevo proyecto, seleccionando la opción que se muestra en la figura siguiente:



Ahora definimos los datos deseados o similares a los que se muestran en la siguiente pantalla, vamos a crear un proyecto Maven, jar, con jdk 8 entre las características más importantes.

Tenemos que especificar solo algunas dependencias: Spring Boot Starter Web, Spring Boot Data JPA y el controlador MySQL JDBC.



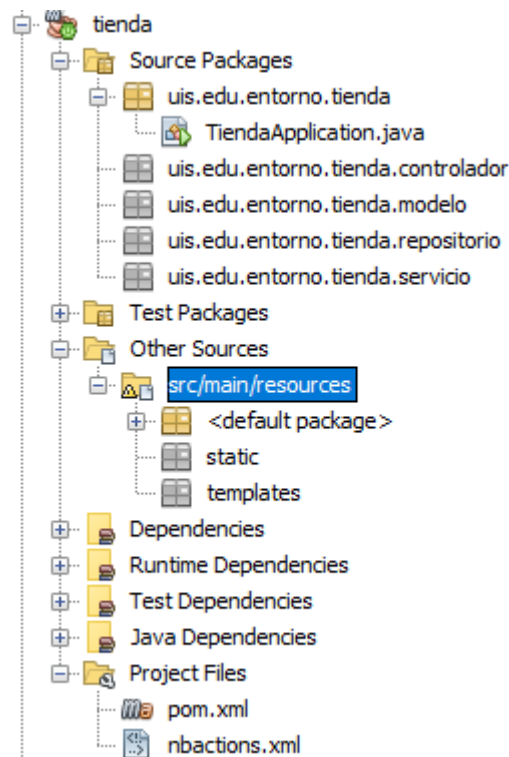
|   |   |                        |
|---|---|------------------------|
|    | <p>ESCUELA DE INGENIERÍA DE SISTEMAS E<br/>INFORMÁTICA - ENTORNOS DE PROGRAMACION</p> <p>CLASES ENLAZADAS – VENTAS - Usuarios</p> | <p>I Semestre 2024</p> |
|---|---|------------------------|

Se agregan las Dependencias web, Dev Tools, Spring web JPA y MySQL Driver.

Se asigna la ubicación y el nombre del proyecto y se da Finish.




## 2. Crear la estructura de packages del proyecto.

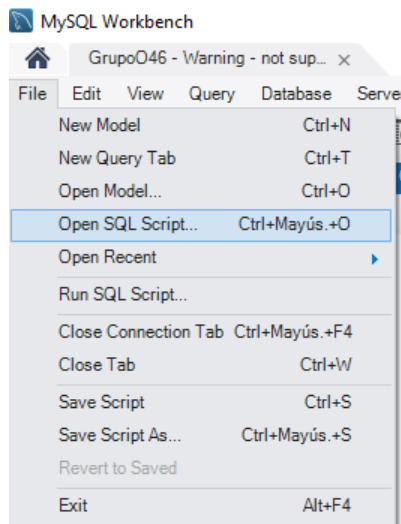
Creamos los packages necesarios para el modelo de datos, el repositorio, el servicio y el controlador, la estructura queda como se aprecia en la figura siguiente.



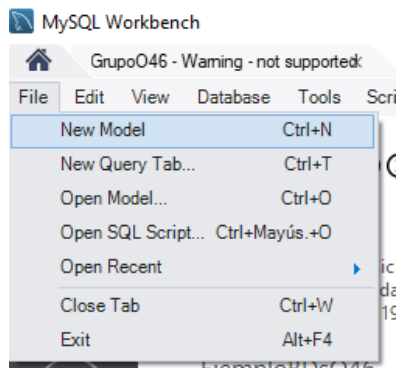
## 3. Revisar y/o crear la base de datos MYSQL y defina sus configuraciones en el proyecto.

Vamos a trabajar sobre la tabla de usuario de la Base de Datos `tiendagenerica` que tiene una estructura que se puede determinar revisando el script `"tiendagenerica.sql"`, para cargar y crear la base de datos se entra a un DBMs, en nuestro caso usamos MySQL Workbench, seleccionamos Open SQL Script, buscamos el archivo y lo ejecutamos.




|   |  |                        |
|---|--|------------------------|
|    | <p>ESCUELA DE INGENIERÍA DE SISTEMAS E<br/> INFORMÁTICA - ENTORNOS DE PROGRAMACION</p> <p>CLASES ENLAZADAS – VENTAS - Usuarios</p> | <p>I Semestre 2024</p> |
|---|--|------------------------|

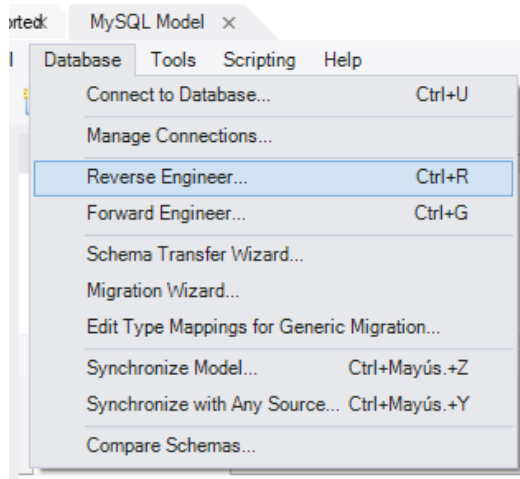


Se debe crear la base de datos tiendagenericaJ1, para crear el diagrama entidad relación, debemos ir al inicio de workbench y seleccionar New Model.



Se ingresa a la pantalla de MySQL Model, allí seleccionamos la opción de Database, Reverse Engineer.




|   |   |                        |
|---|---|------------------------|
|    | <p>ESCUELA DE INGENIERÍA DE SISTEMAS E<br/>INFORMÁTICA - ENTORNOS DE PROGRAMACION</p> <p>CLASES ENLAZADAS – VENTAS - Usuarios</p> | <p>I Semestre 2024</p> |
|---|---|------------------------|




Presionamos Next en la siguiente pantalla donde aparecen las opciones de conexión de MySQL.

Digitamos la clave del usuario root y conectamos a nuestro DBMs

Presionamos el botón de Next y sobre la lista de las Bases de Datos

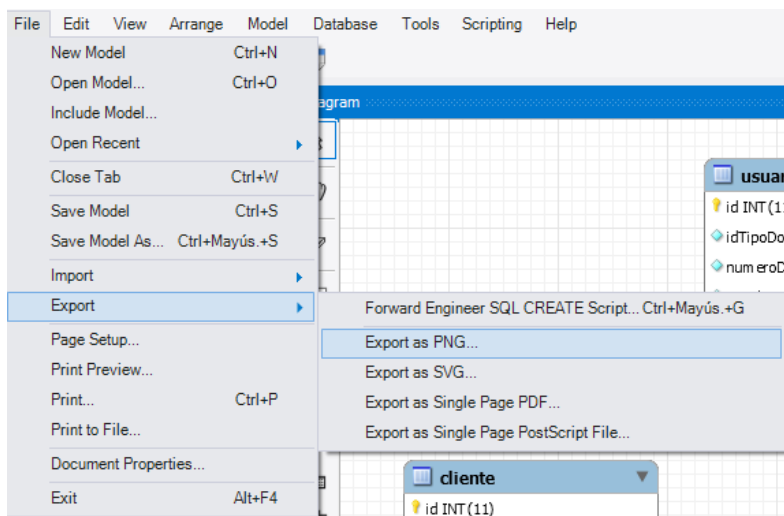
|   |   |                        |
|---|---|------------------------|
|    | <p>ESCUELA DE INGENIERÍA DE SISTEMAS E<br/>INFORMÁTICA - ENTORNOS DE PROGRAMACION</p> <p>CLASES ENLAZADAS – VENTAS - Usuarios</p> | <p>I Semestre 2024</p> |
|---|---|------------------------|

seleccionamos con la cual que vamos a generar el modelo, en nuestro caso tiendagenericaciclo3 y presionamos Next.

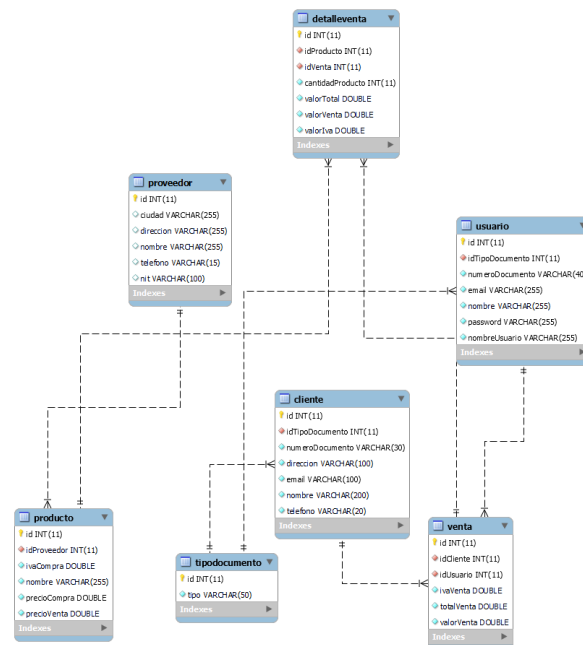
 Select the schemas you want to include:

- ☐ agenda
- ☐ baselogin
- ☐ bd\_clientes
  - ☐ bd\_clienteso46
- ☐ bd\_controlaccess
- ☐ bdprueba
- ☐ boot\_crud
- ☐ db\_ejemplo
  - ☐ db\_tienda046
- ☐ ejemplo046
- ☐ login
- ☐ misiontic
- ☐ mydatabase
- ☐ proyecto
- ☐ proyecto33
- ☐ proyectoiii
- ☐ test
- ☐ tiendagenerica
- ☐ tiendagenerica32
- ☐ tiendagenerica33
- ☒ tiendagenericaciclo3
- ☐ universidad
- ☐ universidad\_a
- ☐ ventas
- ☐ ventaso46
- ☐ ventastempo

Después de seleccionar los objetos a representar y procesar debe aparecer el modelo en la pantalla, después lo grabamos como gráfico con la opción de File/Export/Export as PNG.



Se debe generar un gráfico como el que se aprecia en la siguiente figura:






En el archivo application.properties en el directorio src / main / resources ajustamos el siguiente contenido definiendo la conexión a la base de datos:

```

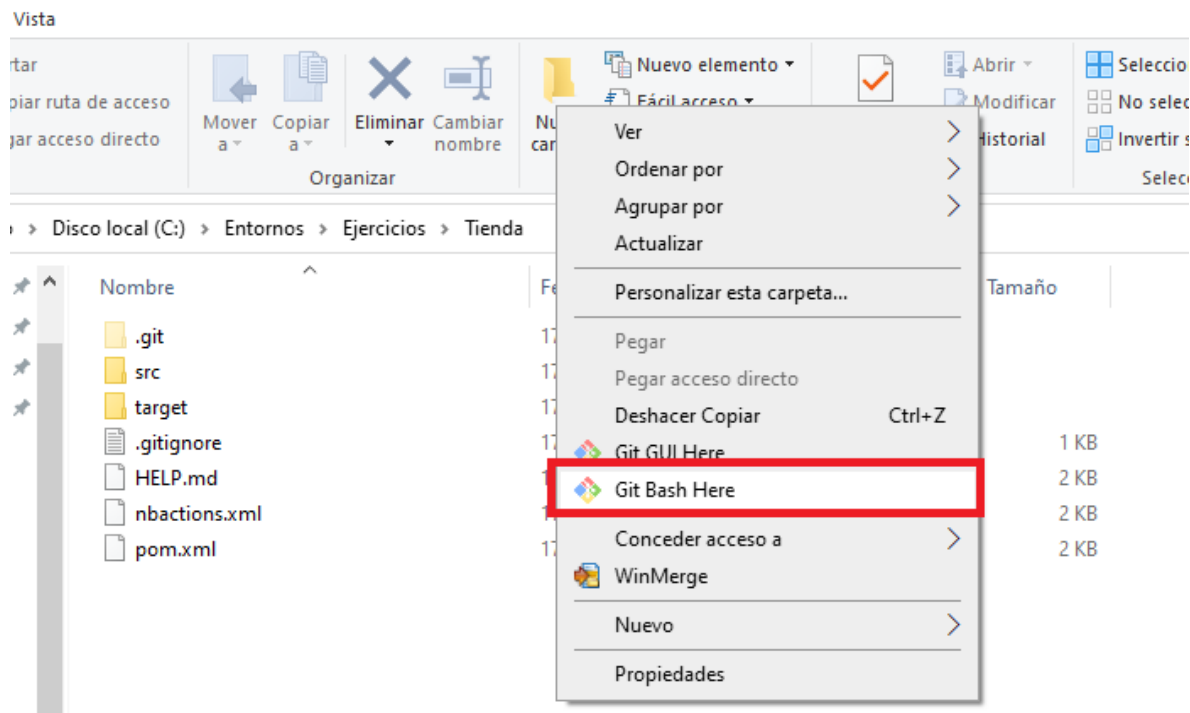
1  spring.datasource.url=jdbc:mysql://localhost:3306/tiendagenericaentorno?serverTimezone=UTC
2  spring.datasource.username=root
3  spring.datasource.password=uis2023
4  spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
5  spring.jpa.database-platform=org.hibernate.dialect.MySQL57Dialect
6  logging.level.org.hibernate.SQL=debug
7  spring.jpa.hibernate.naming.physical-strategy = org.hibernate.boot.model.naming.PhysicalNamingStrategyStandardImpl
8  spring.jackson.time-zone=America/Bogota
9  spring.jackson.locale=es_CO
10 server.port = 8094

```

|   |   |                        |
|---|---|------------------------|
|    | <p>ESCUELA DE INGENIERÍA DE SISTEMAS E<br/>INFORMÁTICA - ENTORNOS DE PROGRAMACION</p> <p>CLASES ENLAZADAS – VENTAS - Usuarios</p> | <p>I Semestre 2024</p> |
|---|---|------------------------|

## Versionar nuestro proyecto con GIT.

Se necesita tener instalado GIT descargarlo desde <https://git-scm.com/download/win> , después desde el directorio del proyecto abrimos Gitbash, como se aprecia en la figura.



Después debemos agregar el usuario y el correo que se gestiona en el proyecto.

```

MINGW64:/c/Entornos/Ejercicios/Tienda
Carlos Beltrán@DESKTOP-1L8L0J7 MINGW64 /c/Entornos/Ejercicios/Tienda (master)
$ git config --global user.name "Carlos A B.C"




Carlos Beltrán@DESKTOP-1L8L0J7 MINGW64 /c/Entornos/Ejercicios/Tienda (master)
$ git config --global user.mail "osocarbel@gmail.com"

Carlos Beltrán@DESKTOP-1L8L0J7 MINGW64 /c/Entornos/Ejercicios/Tienda (master)
$

```

Ahora le indicamos que deseamos que el proyecto sea versionado con **git init**



|   |   |                        |
|---|---|------------------------|
|    | <p>ESCUELA DE INGENIERÍA DE SISTEMAS E<br/>INFORMÁTICA - ENTORNOS DE PROGRAMACION</p> <p>CLASES ENLAZADAS – VENTAS - Usuarios</p> | <p>I Semestre 2024</p> |
|---|---|------------------------|

```
$ git init
Reinitialized existing Git repository in C:/Entornos/Ejercicios/Tienda/.git/

Carlos Beltrán@DESKTOP-1L8L0J7 MINGW64 /c/Entornos/Ejercicios/Tienda (master)
$ |
```

Vamos a agregar los archivos al repositorio con **git add A**

```
$ git log
fatal: your current branch 'master' does not have any commits yet

Carlos Beltrán@DESKTOP-1L8L0J7 MINGW64 /c/Entornos/Ejercicios/Tienda (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   .gitignore
    new file:   nbactions.xml
    new file:   pom.xml
    new file:   src/main/java/uis/edu/entorno/tienda/TiendaApplication.java
    new file:   src/main/java/uis/edu/entorno/tienda/controlador/UsuarioController.java
    new file:   src/main/java/uis/edu/entorno/tienda/modelo/Tipodocumento.java
    new file:   src/main/java/uis/edu/entorno/tienda/modelo/Usuario.java
    new file:   src/main/java/uis/edu/entorno/tienda/repositorio/TipodocumentoRepositorio.java
    new file:   src/main/java/uis/edu/entorno/tienda/repositorio/UsuarioRepositorio.java
    new file:   src/main/java/uis/edu/entorno/tienda/servicio/IUsuarioService.java
    new file:   src/main/java/uis/edu/entorno/tienda/servicio/UsuarioService.java
    new file:   src/main/resources/application.properties
    new file:   src/test/java/uis/edu/entorno/tienda/TiendaApplicationTests.java

Carlos Beltrán@DESKTOP-1L8L0J7 MINGW64 /c/Entornos/Ejercicios/Tienda (master)
$ |
```


Primer commit con git commit -m "Mensaje"

```
$ git commit -m "Inicio del Proyecto"
[master (root-commit) 118dd7b] Inicio del Proyecto
13 files changed, 496 insertions(+)
 create mode 100644 .gitignore
 create mode 100644 nbactions.xml
 create mode 100644 pom.xml
 create mode 100644 src/main/java/uis/edu/entorno/tienda/TiendaApplication.java
 create mode 100644 src/main/java/uis/edu/entorno/tienda/controlador/UsuarioController.java
 create mode 100644 src/main/java/uis/edu/entorno/tienda/modelo/Tipodocumento.java
 create mode 100644 src/main/java/uis/edu/entorno/tienda/modelo/Usuario.java
 create mode 100644 src/main/java/uis/edu/entorno/tienda/repositorio/TipodocumentoRepositorio.java
 create mode 100644 src/main/java/uis/edu/entorno/tienda/repositorio/UsuarioRepositorio.java
 create mode 100644 src/main/java/uis/edu/entorno/tienda/servicio/IUsuarioService.java
 create mode 100644 src/main/java/uis/edu/entorno/tienda/servicio/UsuarioService.java
 create mode 100644 src/main/resources/application.properties
 create mode 100644 src/test/java/uis/edu/entorno/tienda/TiendaApplicationTests.java

Carlos Beltrán@DESKTOP-1L8L0J7 MINGW64 /c/Entornos/Ejercicios/Tienda (master)
$ git log
commit 118dd7b38e6c0c14a0c84f346273562c6d7311c4 (HEAD -> master)
Author: Carlos A B.C <misiontic.formador67@uis.edu.co>
Date: Mon Apr 17 18:18:17 2023 -0500

    Inicio del Proyecto

Carlos Beltrán@DESKTOP-1L8L0J7 MINGW64 /c/Entornos/Ejercicios/Tienda (master)
```

|   |   |                        |
|---|---|------------------------|
|  | <p>ESCUELA DE INGENIERÍA DE SISTEMAS E<br/>INFORMÁTICA - ENTORNOS DE PROGRAMACION</p> <p>CLASES ENLAZADAS – VENTAS - Usuarios</p> | <p>I Semestre 2024</p> |
|---|---|------------------------|

Pueden revisar el document de netbeans con la forma que se puede gestionar git desde el IDE en:

<https://netbeans.apache.org/kb/docs/ide/git.html>.

También puede encontrarse información adicional en:

<https://javiergarciaescobedo.es/programacion-en-java/29-trucos/354-alojar-proyecto-netbeans-en-github>

## 4. Clases de Modelo de Datos.

Cree la clase de modelo de dominio Usuario (debe tener el mismo nombre de la Tabla) y Tipodocumento dado que esta tiene una relación con la tabla Usuario, sobre un nuevo package llamado modelo por debajo del de por defecto, para mapear con la tabla de productos con las anotaciones.

Agregamos el constructor con todos los atributos, los setter, getter y el toString para usar en caso de ser necesario.

Esta es una clase de entidad JPA simple con el nombre de la clase y los nombres de los campos son idénticos a los nombres de las columnas del producto de la tabla en la base de datos, para minimizar las anotaciones utilizadas.

## • Entidad Tipodocumento:

```

1  package uis.edu.entorno.tienda.modelo;
2
3  import javax.persistence.Column;
4  import javax.persistence.Entity;
5  import javax.persistence.GeneratedValue;
6  import javax.persistence.GenerationType;
7  import javax.persistence.Id;
8  import javax.persistence.Table;
9
10 @Entity
11 @Table(name=Tipodocumento.TABLE_NAME)
12 public class Tipodocumento {
13
14     public static final String TABLE_NAME = "tipodocumento";
15     @Id
16     @GeneratedValue(strategy = GenerationType.IDENTITY)
17     private Long id;
18
19     @Column(name = "tipo")
20     private String tipo;
21
22     // Constructor
23
24     public Tipodocumento() {
25     }
26
27     public Tipodocumento(Long id, String tipo) {
28         this.id = id;
29         this.tipo = tipo;
30     }
31
32     public Long getId() {
33         return id;
34     }
35
36     public void setId(Long id) {
37         this.id = id;
38     }
39
40     public String getTipo() {
41         return tipo;
42     }
43
44     public void setTipo(String tipo) {
45         this.tipo = tipo;
46     }
47
48 }

```

## • Entidad Usuario:

```




1  package uis.edu.entorno.tienda.modelo;
2
3  import javax.persistence.Column;
4  import javax.persistence.Entity;
5  import javax.persistence.GeneratedValue;
6  import javax.persistence.GenerationType;
7  import javax.persistence.Id;
8  import javax.persistence.JoinColumn;
9  import javax.persistence.ManyToOne;
10 import javax.persistence.Table;
11
12 @Entity
13 @Table(name = Usuario.TABLE_NAME)
14 public class Usuario {
15     public static final String TABLE_NAME = "usuario";
16
17     /*
18      * @id para identificar la llave primaria
19      * @GeneratedValue(strategy = GenerationType.IDENTITY)
20      * se define el autoincremental
21      */
22     @Id
23     @GeneratedValue(strategy = GenerationType.IDENTITY)
24     private Long id;
25
26     /*@ManyToOne hace referencia la relacion muchos a uno en este caso
27     muchos usuario tienen un tipo de documento
28     * @JoinColumn el campo que hace de referencia a la llave foranea
29     */
30     @ManyToOne
31     @JoinColumn(name = "idTipoDocumento")
32     private Tipodocumento idTipoDocumento;
33
34     /*@Column nombre de la columna , si el nombre en la base de datos del
35     campo es igual a el de la variable no es necesario poner la anotacion
36     */
37     @Column(name = "numeroDocumento")
38     private String numeroDocumento;
39
40     @Column(name = "nombre")
41     private String nombre;
42
43     @Column(name = "password")
44     private String password;
45
46     @Column(name = "nombreUsuario")
47     private String nombreUsuario;
48
49     @Column(name = "email")
50     private String email;

```

```

49
50 public Usuario() {
51
52 }
53
54 public Usuario(Long id, Tipodocumento idTipoDocumento,
55 String numeroDocumento, String nombre, String password,
56 String nombreUsuario, String email) {
57     this.id = id;
58     this.idTipoDocumento = idTipoDocumento;
59     this.numeroDocumento = numeroDocumento;
60     this.nombre = nombre;
61     this.password = password;
62     this.nombreUsuario = nombreUsuario;
63     this.email = email;
64 }
65
66 public Long getId() {
67     return id;
68 }
69
70 public void setId(Long id) {
71     this.id = id;
72 }
73
74 public Tipodocumento getIdTipoDocumento() {
75     return idTipoDocumento;
76 }
77
78 public void setIdTipoDocumento(Tipodocumento idTipoDocumento) {
79     this.idTipoDocumento = idTipoDocumento;
80 }
81
82 public String getNumeroDocumento() {
83     return numeroDocumento;
84 }
85
86 public void setNumeroDocumento(String numeroDocumento) {
87     this.numeroDocumento = numeroDocumento;
88 }
89
90 public String getNombre() {
91     return nombre;
92 }
93
94 public void setNombre(String nombre) {
95     this.nombre = nombre;
96 }
97

```

|   |   |                        |
|---|---|------------------------|
|    | <p>ESCUELA DE INGENIERÍA DE SISTEMAS E<br/>INFORMÁTICA - ENTORNOS DE PROGRAMACION</p> <p>CLASES ENLAZADAS – VENTAS - Usuarios</p> | <p>I Semestre 2024</p> |
|---|---|------------------------|

```

97
98 public String getPassword() {
99     return password;
100 }
101
102 public void setPassword(String password) {
103     this.password = password;
104 }
105
106 public String getNombreUsuario() {
107     return nombreUsuario;
108 }
109
110 public void setNombreUsuario(String nombreUsuario) {
111     this.nombreUsuario = nombreUsuario;
112 }
113
114 public String getEmail() {
115     return email;
116 }
117
118 public void setEmail(String email) {
119     this.email = email;
120 }
121

```

## 5. Interface repositorio(DAO).

Cree la interface por cada entidad del modelo en el package repositorio, es decir para Tipodocumento y Usuario




Hay métodos integrados para operaciones CRUD en JpaRepository, no es necesario escribir ninguna consulta SQL.

Creamos TipodocumentoRepositorio con herencia de JpaRepository agregando como parámetros el nombre de la clase y el tipo de dato de su llave primaria.

```

1 package uis.edu.entorno.tienda.repositorio;
2
3 import org.springframework.data.jpa.repository.JpaRepository;
4 import uis.edu.entorno.tienda.modelo.Tipodocumento;
5
6 public interface TipodocumentoRepositorio extends JpaRepository<Tipodocumento, Long> {
7
8 }

```

|   |   |                        |
|---|---|------------------------|
|    | <p>ESCUELA DE INGENIERÍA DE SISTEMAS E<br/>INFORMÁTICA - ENTORNOS DE PROGRAMACION</p> <p>CLASES ENLAZADAS – VENTAS - Usuarios</p> | <p>I Semestre 2024</p> |
|---|---|------------------------|

A continuación, cree la interfaz UsuarioRepositorio sobre el mismo package llamado repositorio de la siguiente manera:

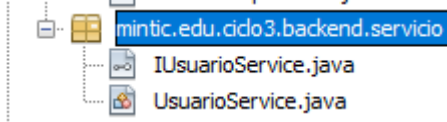
```

1 package uis.edu.entorno.tienda.repositorio;
2
3 import org.springframework.data.jpa.repository.JpaRepository;
4 import uis.edu.entorno.tienda.modelo.Usuario;
5
6 public interface UsuarioRepositorio extends JpaRepository<Usuario, Long>{
7
8 }
9

```

## 6. Clase UsuarioService (Servicio).

Cree una clase de UsuarioService para codificar la lógica empresarial y actúa como una capa intermedia entre el repositorio y la clase de controlador, antes de ello creamos la interface que defina los métodos a implementar en la clase, la estructura de los dos archivos será como se aprecia en la figura:






### • Interface IUsuarioService

Se agregan las cabeceras de los métodos a programar en la clase de servicio, para cada uno de los métodos del CRUD.

```

1 package uis.edu.entorno.tienda.servicio;
2
3 import java.util.List;
4 import uis.edu.entorno.tienda.modelo.Usuario;
5
6 public interface IUsuarioService {
7
8     List<Usuario> getUsuarios();
9
10     Usuario nuevoUsuario(Usuario usuario);
11
12     Usuario buscarUsuario(Long id);
13
14     int borrarUsuario(Long id);
15 }

```

|   |   |                        |
|---|---|------------------------|
|    | <p>ESCUELA DE INGENIERÍA DE SISTEMAS E<br/>INFORMÁTICA - ENTORNOS DE PROGRAMACION</p> <p>CLASES ENLAZADAS – VENTAS - Usuarios</p> | <p>I Semestre 2024</p> |
|---|---|------------------------|

## • Clase UsuarioService

Se utiliza las anotaciones @Service y @Transactional anotar métodos se ejecutan en transacciones.




A continuación, necesitamos codificar la clase UsuarioService en la capa de servicio y / negocio con el siguiente código:

```

1  package uis.edu.entorno.tienda.servicio;
2
3  import java.util.List;
4  import org.springframework.beans.factory.annotation.Autowired;
5  import org.springframework.stereotype.Service;
6  import org.springframework.transaction.annotation.Transactional;
7  import uis.edu.entorno.tienda.modelo.Usuario;
8  import uis.edu.entorno.tienda.repositorio.UsuarioRepositorio;
9
10 @Service
11 @Transactional
12 public class UsuarioService implements IUsuarioService{
13
14     @Autowired
15     UsuarioRepositorio usuarioRepositorio;
16
17     @Override
18     public List<Usuario> getUsuarios() {
19         return usuarioRepositorio.findAll();
20     }
21
22     @Override
23     public Usuario nuevoUsuario(Usuario usuario) {
24         return usuarioRepositorio.save(usuario);
25     }
26
27     @Override
28     public Usuario buscarUsuario(Long id) {
29         Usuario usuario = null;
30         usuario = usuarioRepositorio.findById(id).orElse(null);
31         if (usuario == null) {
32             return null;
33         }
34         return usuario;
35     }
36
37     @Override
38     public int borrarUsuario(Long id) {
39         usuarioRepositorio.deleteById(id);
40         return 1;
41     }
42 }
43

```



|   |   |                        |
|---|---|------------------------|
|    | <p>ESCUELA DE INGENIERÍA DE SISTEMAS E<br/>INFORMÁTICA - ENTORNOS DE PROGRAMACION</p> <p>CLASES ENLAZADAS – VENTAS - Usuarios</p> | <p>I Semestre 2024</p> |
|---|---|------------------------|

## 7. Clase Controlador(Métodos CRUD).

A continuación, cree la clase UsuarioController que actúa como un controlador Spring MVC para manejar las solicitudes de los clientes, además contiene todos los puntos finales(endpoint) de la API REST para las operaciones CRUD con el código inicial de la siguiente manera:

Como se puede ver, inyectamos una instancia de la clase ClienteService a este controlador: Spring creará una automáticamente en tiempo de ejecución. Escribiremos código para los métodos del controlador al implementar cada operación CRUD, el único que implementamos por ahora es el de listar.

```




1  package mintic.edu.ciclo3.backend.controlador;
2
3  import java.util.List;
4  import mintic.edu.ciclo3.backend.modelo.Usuario;
5  import mintic.edu.ciclo3.backend.servicio.UsuarioService;
6  import org.springframework.beans.factory.annotation.Autowired;
7  import org.springframework.http.HttpStatus;
8  import org.springframework.http.ResponseEntity;
9  import org.springframework.web.bind.annotation.DeleteMapping;
10 import org.springframework.web.bind.annotation.GetMapping;
11 import org.springframework.web.bind.annotation.PathVariable;
12 import org.springframework.web.bind.annotation.PostMapping;
13 import org.springframework.web.bind.annotation.PutMapping;
14 import org.springframework.web.bind.annotation.RequestBody;
15 import org.springframework.web.bind.annotation.RestController;
16
17 @RestController
18
19 public class UsuarioController {
20     /*
21      * inyectamos el la interface del servicio para acceder
22      * a los metodos del negocio
23      */
24     @Autowired
25     UsuarioService usuarioService;
26

```

```

27 // Listar los usuarios
28 @GetMapping("/list")
29 public List<Usuario> cargarUsuarios() {
30     return usuarioService.getUsuarios();
31 }
32
33 // Buscar por Id
34 @GetMapping("/list/{id}")
35 public Usuario buscarPorId(@PathVariable Long id) {
36     return usuarioService.buscarUsuario(id);
37 }
38
39 // Agregar un Usuario
40 @PostMapping("/")
41 public ResponseEntity<Usuario> agregar(@RequestBody Usuario usuario) {
42     Usuario obj = usuarioService.nuevoUsuario(usuario);
43     return new ResponseEntity<>(obj, HttpStatus.OK);
44 }
45
46 // Actualizar el Usuario
47 @PutMapping("/")
48 public ResponseEntity<Usuario> editar(@RequestBody Usuario usuario) {
49     Usuario obj = usuarioService.buscarUsuario(usuario.getId());
50     if(obj != null) {
51         obj.setEmail(usuario.getEmail());
52         obj.setIdTipoDocumento(usuario.getIdTipoDocumento());
53         obj.setNombre(usuario.getNombre());
54         obj.setNombreUsuario(usuario.getNombreUsuario());
55         obj.setNumeroDocumento(usuario.getNumeroDocumento());
56         obj.setPassword(usuario.getPassword());
57         usuarioService.nuevoUsuario(obj);
58     } else {
59         return new ResponseEntity<>(obj, HttpStatus.INTERNAL_SERVER_ERROR);
60     }
61     return new ResponseEntity<>(obj, HttpStatus.OK);
62 }
63
64 // Eliminar el Usuario
65 @DeleteMapping("/{id}")
66 public ResponseEntity<Usuario> eliminar(@PathVariable Long id) {
67     Usuario obj = usuarioService.buscarUsuario(id);
68     if(obj != null) {
69         usuarioService.borrarUsuario(id);
70     } else {
71         return new ResponseEntity<>(obj, HttpStatus.INTERNAL_SERVER_ERROR);
72     }
73     return new ResponseEntity<>(obj, HttpStatus.OK);
74 }
75
76 }

```

|   |   |                        |
|---|---|------------------------|
|    | <p>ESCUELA DE INGENIERÍA DE SISTEMAS E<br/>INFORMÁTICA - ENTORNOS DE PROGRAMACION</p> <p>CLASES ENLAZADAS – VENTAS - Usuarios</p> | <p>I Semestre 2024</p> |
|---|---|------------------------|

## 8. Documentación del Proyecto SpringDoc OpenApi(Swagger).

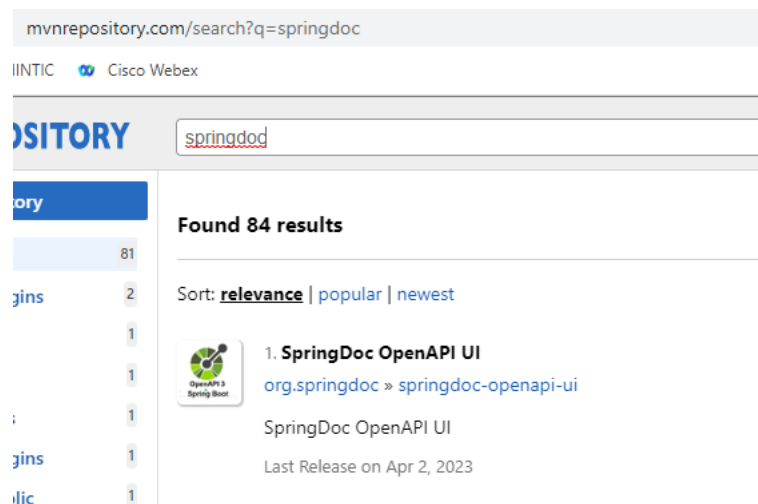
Una API definitivamente pierde su sentido sino es accesible y si no tenemos una documentación que nos ayude a entenderla.

Uno de los mayores problemas de las APIs es que en muchos casos, la documentación que le acompaña es inútil. Swagger nace con la intención de solucionar este problema. Su objetivo es estandarizar el vocabulario que utilizan las APIs. Es el diccionario API.


Cuando hablamos de Swagger nos referimos a una serie de reglas, especificaciones y herramientas que nos ayudan a documentar nuestras APIs. De esta manera, podemos realizar documentación que sea realmente útil para las personas que la necesitan. Swagger nos ayuda a crear documentación que todo el mundo entienda.

### Swagger UI – La interfaz de usuario de Swagger

Swagger UI es una de las herramientas atractivas de la plataforma. Para que una documentación sea útil necesitaremos que sea navegable y que esté perfectamente organizada para un fácil acceso. Por esta razón, realizar una buena documentación puede ser realmente tedioso y consumir mucho tiempo a los desarrolladores. Agregamos la dependencia nueva para gestionar el uso de SpringDoc, buscamos en el navegador en “Maven repository” springdoc como lo muestra la figura:



Copiamos la dependencia correspondiente en nuestro archivo de configuración pom.xml


**SpringDoc OpenAPI UI » 1.7.0**  
 SpringDoc OpenAPI UI

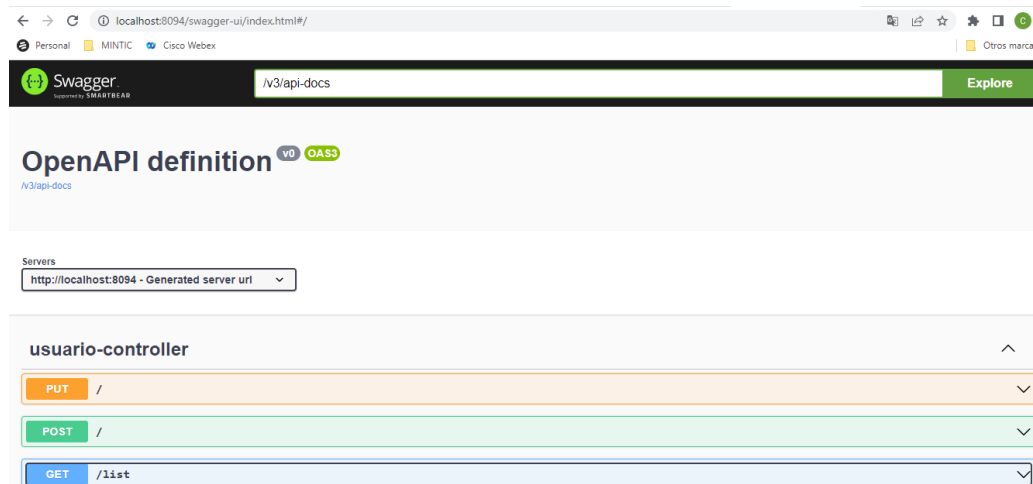
|              |   |
|--------------|---|
| License      | Apache 2.0  |
| Tags         | spring openapi ui api   |
| Date         | Apr 02, 2023  |
| Files        | <a href="#">pom (1 KB)</a> <a href="#">jar (21 KB)</a> <a href="#">View All</a> |
| Repositories | Central   |
| Ranking      | #926 in MvnRepository ( <a href="#">See Top Artifacts</a> )                     |
| Used By      | 483 artifacts   |

[Maven](#)
[Gradle](#)
[Gradle \(Short\)](#)
[Gradle \(Kotlin\)](#)
[SBT](#)
[Ivy](#)
[Grape](#)
[Leiningen](#)
[Buildr](#)

```




<!-- https://mvnrepository.com/artifact/org.springdoc/springdoc-openapi-ui -->
<dependency>
  <groupId>org.springdoc</groupId>
  <artifactId>springdoc-openapi-ui</artifactId>
  <version>1.7.0</version>
</dependency>
  
```

Ahora vamos a probar ejecutando el proyecto, al ejecutar sobre el url y puerto correspondiente marcaría un error de página no encontrada, si se desea visualizar la interfaz de swagger se debe agregar al url "/swagger-ui.html", el resultado se puede apreciar en la siguiente figura:



The screenshot shows the Swagger UI interface. At the top, there's a search bar with "/v3/api-docs" and an "Explore" button. Below that, it says "OpenAPI definition" with "v0" and "OAS3" tags. A "Servers" section shows "http://localhost:8094 - Generated server url". Under the "usuario-controller" section, there are three endpoints listed: "PUT /", "POST /", and "GET /list".

Vale resaltar que las últimas versiones de Spring Boot han presentado dificultades con el uso de Swagger por ello no se recomienda su uso hasta que ese problema sea superado.




|   |   |                        |
|---|---|------------------------|
|    | <p>ESCUELA DE INGENIERÍA DE SISTEMAS E<br/>INFORMÁTICA - ENTORNOS DE PROGRAMACION</p> <p>CLASES ENLAZADAS – VENTAS - Usuarios</p> | <p>I Semestre 2024</p> |
|---|---|------------------------|

## 9. Compile y ejecute el proyecto.

Para probar el proyecto ya no es necesario usar Postman mientras tanto, recordar que el proyecto se ejecuta desde el archivo main del mismo, dando click derecho sobre el mismo y seleccionamos la opción de Run File, recuerden siempre dar clean and build sobre el proyecto para que se genere el archivo ejecutable .jar.

Al ejecutar el proyecto se puede revisar en la url <http://localhost:8094/swagger-ui.html>, la pantalla va a reflejar dos áreas principales, en la primera podemos apreciar los endpoint (UsuarioController) y los esquemas presentes que serían los modelos creados Tipodocumento y Usuario.




|   |   |                        |
|---|---|------------------------|
|    | <p>ESCUELA DE INGENIERÍA DE SISTEMAS E<br/>INFORMÁTICA - ENTORNOS DE PROGRAMACION</p> <p>CLASES ENLAZADAS – VENTAS - Usuarios</p> | <p>I Semestre 2024</p> |
|---|---|------------------------|

Servers

http://localhost:8094 - Generated server url

**usuario-controller**

|        |                         |   |
|--------|-------------------------|---|
| PUT    | /api/usuarios/          | ✓   |
| POST   | /api/usuarios/          | ✓  |
| GET    | /api/usuarios/list      | ✓   |
| GET    | /api/usuarios/list/{id} | ✓   |
| DELETE | /api/usuarios/{id}      | ✓   |

Cada uno de los microservicios se puede ejecutar en la url.