	<p>ESCUELA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA</p> <p>MICROSERVICIO CON MYSQL</p>	<p>I Semestre 2023</p>
---	---	------------------------

API REST (Microservicio) SPRING BOOT - MySQL.

El proyecto a desarrollar en se descompone en microservicios, cada uno de ellos tiene una responsabilidad y funcionalidad dentro del sistema general. En este caso desarrollaremos como ejemplo base de datos sencilla, vamos a crear el backend o microservicio de Clientes.

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 (Create, Read, Update y Delete) 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

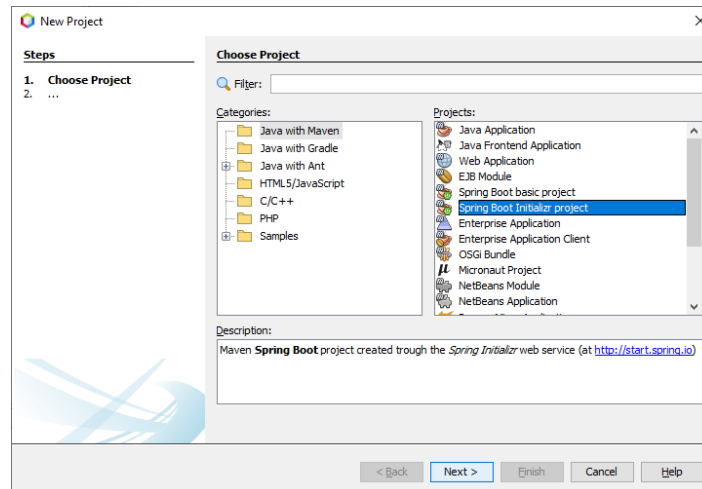
1. Maven 3.0+
2. IDE (Netbeans, STS, Eclipse or IntelliJ)
3. JDK 1.8+
4. MYSQL como servidor de Base de Datos
5. Postman para probar.

Contenido

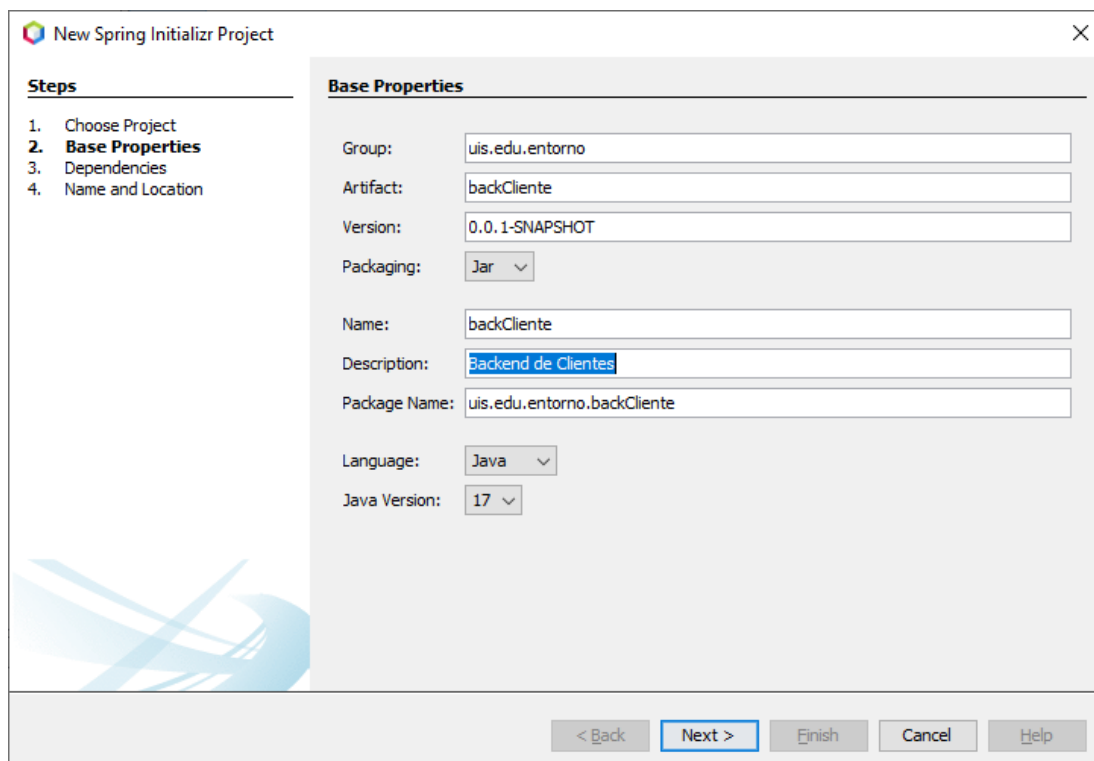
1. Cree el proyecto Spring Boot.
2. Estructura de paquetes del proyecto.
3. Crear y/o revisar la base de datos MYSQL y defina sus configuraciones en el proyecto.
4. Crear clase de modelo de entidad
5. Crear repositorio de datos JPA
6. Crear clase de servicio
7. Crear clase de controladores.
8. Compile y ejecute el proyecto
9. Prueba con Postman

1. Crear el proyecto Spring Boot.

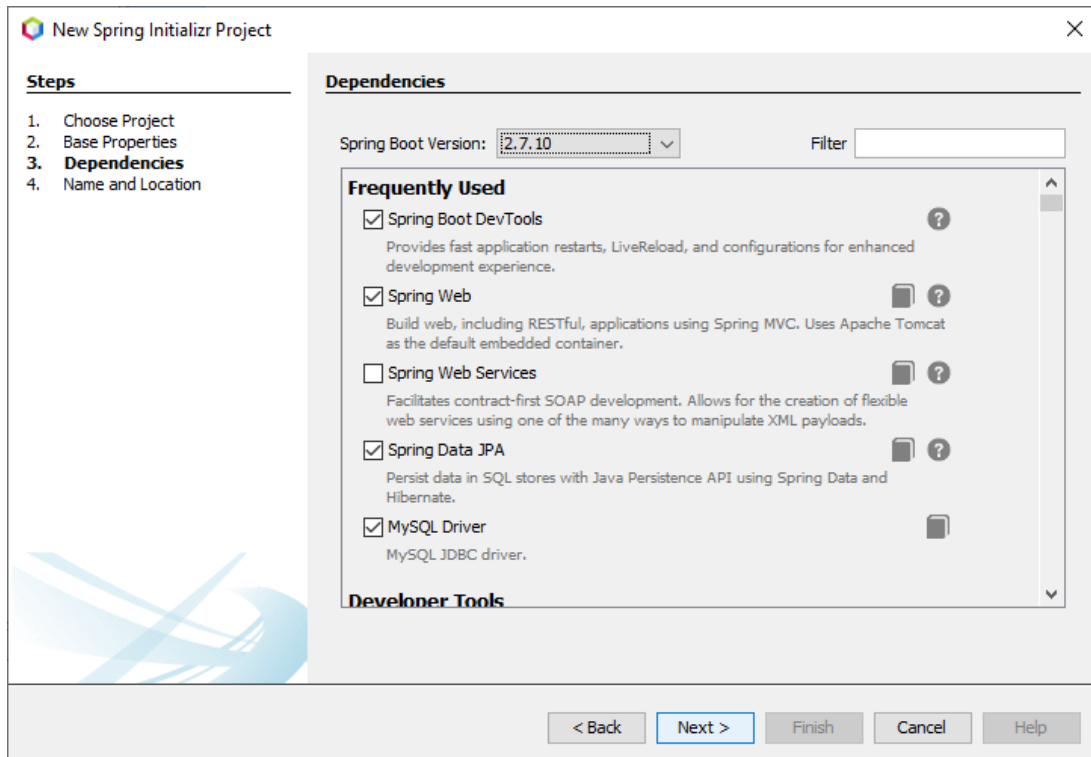
Desde Netbeans creamos un nuevo proyecto, seleccionando la opción que se muestra en la figura siguiente:



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 ó 11 entre las características más importantes.



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



New Spring Initializr Project

Steps

1. Choose Project
2. Base Properties
3. **Dependencies**
4. Name and Location

Dependencies

Spring Boot Version: Filter

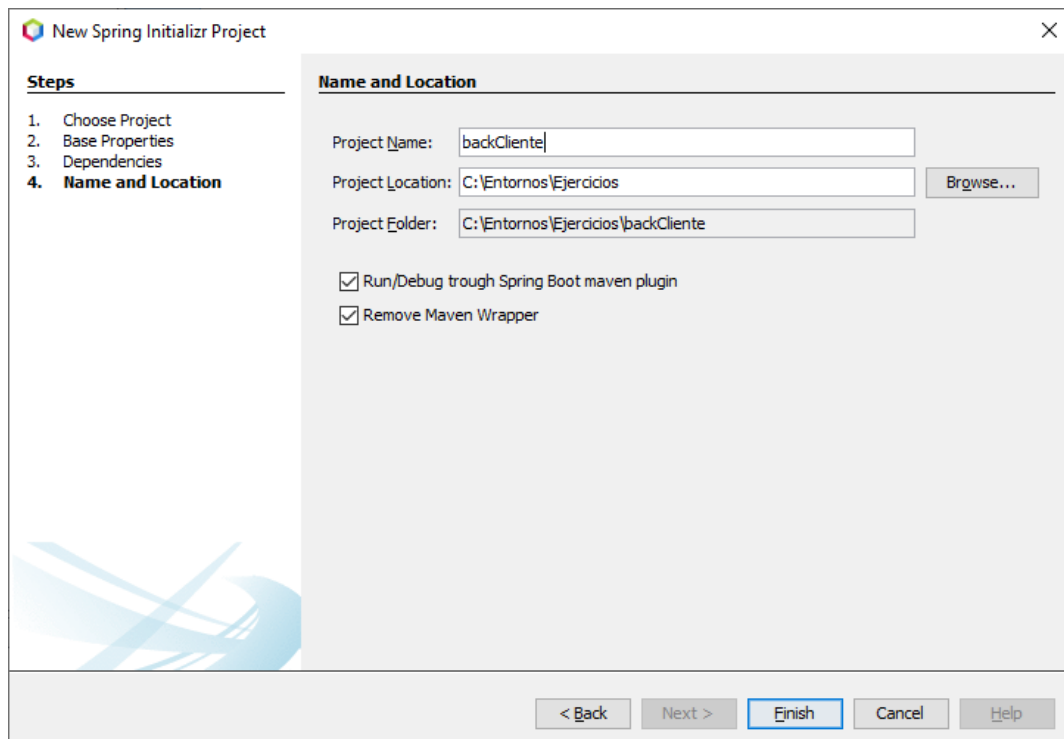
Frequently Used

- ☒ **Spring Boot DevTools**
Provides fast application restarts, LiveReload, and configurations for enhanced development experience.
- ☒ **Spring Web**
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.
- ☐ **Spring Web Services**
Facilitates contract-first SOAP development. Allows for the creation of flexible web services using one of the many ways to manipulate XML payloads.
- ☒ **Spring Data JPA**
Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.
- ☒ **MySQL Driver**
MySQL JDBC driver.

Developer Tools

< Back Next > Finish Cancel Help

Nombre del Proyecto y ruta.



New Spring Initializr Project

Steps

1. Choose Project
2. Base Properties
3. Dependencies
4. **Name and Location**

Name and Location

Project Name:

Project Location: Browse...

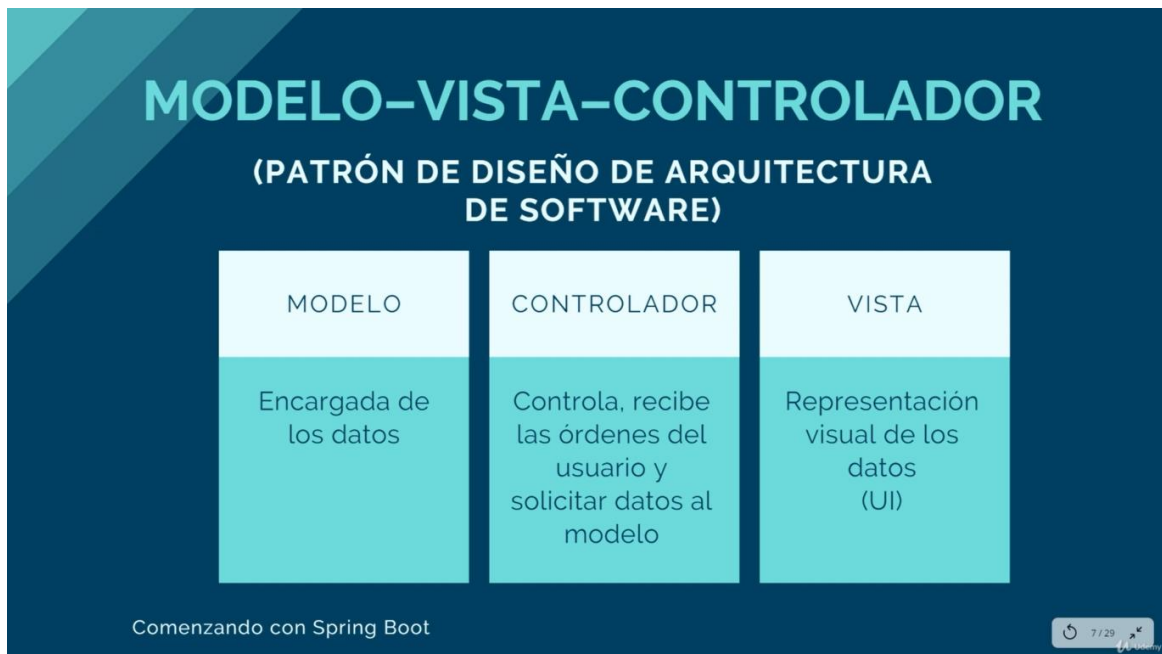
Project Folder:

- ☒ Run/Debug through Spring Boot maven plugin
- ☒ Remove Maven Wrapper

< Back Next > Finish Cancel Help

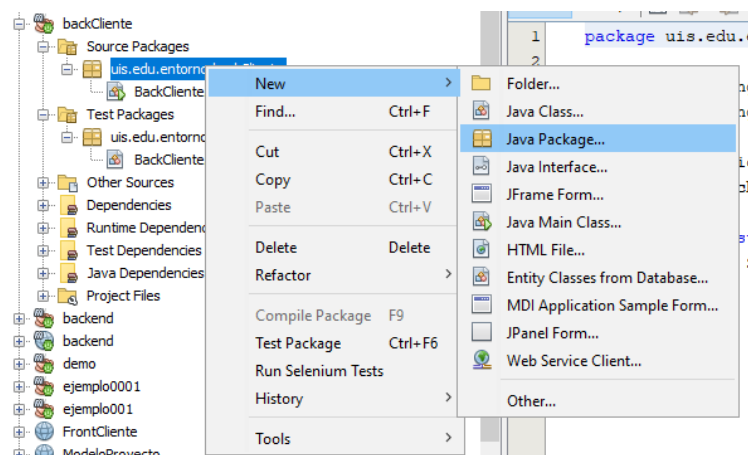
2. Estructura de paquetes del proyecto.

- PATRÓN DE DISEÑO DE ARQUITECTURA MODELO – VISTA . CONTROLADOR.**

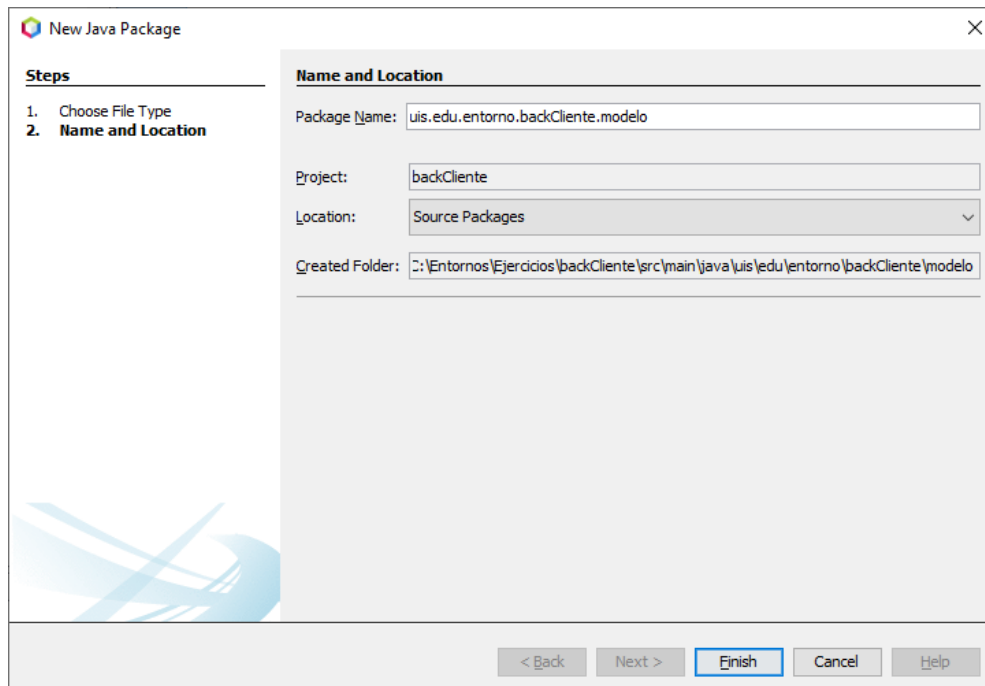


Creamos un paquete por cada una de las capas empleadas dentro del proyecto, todas deben estar anexas al package principal, primero una con el modelo, después otra con el repositorio, otra con el servicio y una con el controlador.

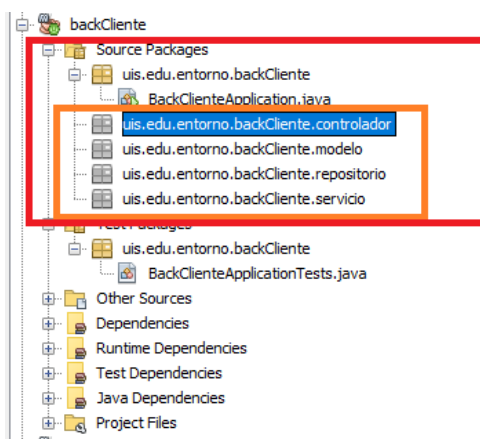
Para construir los packages nos ubicamos sobre el package principal del proyecto y agregamos uno nuevo como se aprecia en la siguiente figura con: click derecho, New, Java Package.

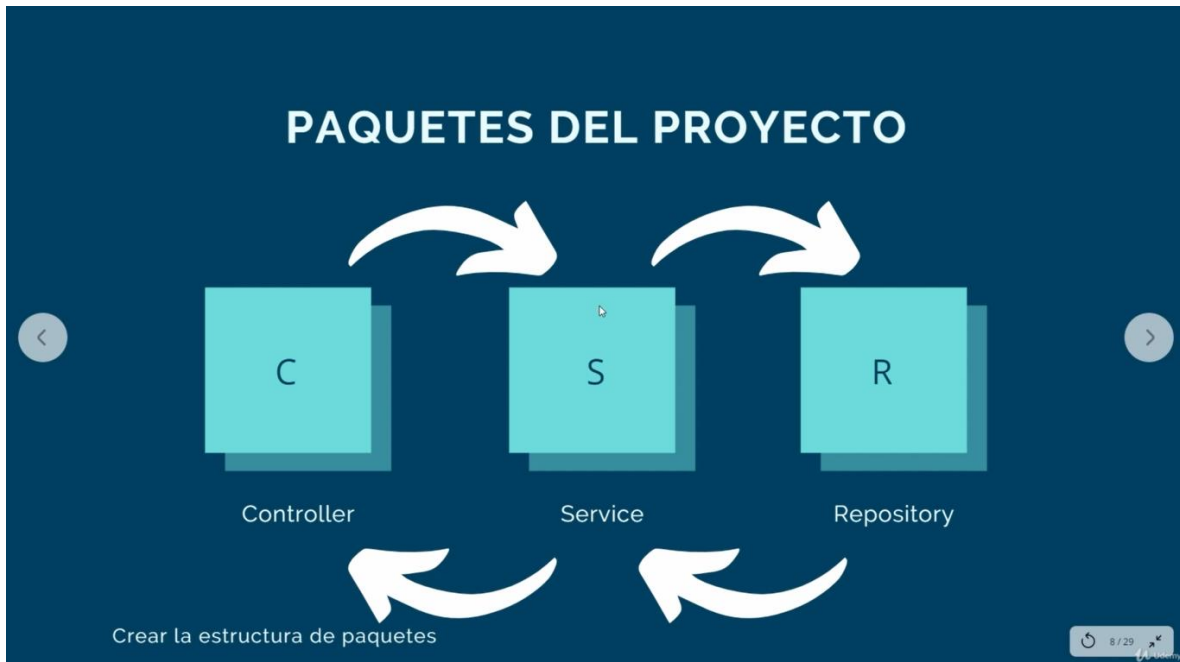


A continuación, damos el nombre al package, en nuestro caso tomamos como ejemplo el modelo y presionamos el botón Finish.



La estructura final de los packages debe quedar como se aprecia en la figura:








Capa de Datos



3. Revisar la base de datos MYSQL y defina sus configuraciones en el proyecto.

Vamos a trabajar sobre la tabla de cliente que estará definida en el script que compartimos anexo al ejercicio llamado clientesJ1.sql que se lista en el siguiente gráfico:

Nombre	Fecha de modificación	Tipo	Tamaño
 clientesJ1.sql	12/04/2023 5:53 p. m.	SQL Text File	6 KB
 MicroservicioSpringBootBDs.pdf	2/09/2022 4:21 p. m.	Foxit PDF Reader ...	875 KB
 properties.txt	2/09/2022 2:57 p. m.	Documento de te...	1 KB

Debemos ejecutar el script sobre alguna de las herramientas instaladas para gestionar la base de datos de MYSQL, para efectos del ejercicio ya está creada.

La base de datos contine una sola tabla llamada cliente que tiene una estructura como la que se aprecia en la siguiente figura:

Field Types							
#	Field	Schema	Table	Type	Character Set	Display Size	Pi
1	idcliente	bd_clientes	cliente	INT	binary	15	
2	documento	bd_clientes	cliente	VARCHAR	utf8mb4	10	
3	tipdoc	bd_clientes	cliente	VARCHAR	utf8mb4	2	
4	nombres	bd_clientes	cliente	VARCHAR	utf8mb4	45	
5	apellidos	bd_clientes	cliente	VARCHAR	utf8mb4	45	
6	direccion	bd_clientes	cliente	TEXT	utf8mb4	65535	
7	email	bd_clientes	cliente	VARCHAR	utf8mb4	150	

- Configuración de conexión de la Base de Datos en el proyecto.**




En el archivo application.properties en el directorio src / main / resources / default package ajustamos el siguiente contenido definiendo la conexión a la base de datos y el puerto de ejecución del proyecto, se puede copiar el contenido del archivo properties.txt compartido:

```

1  spring.datasource.url=jdbc:mysql://localhost:3306/bd_clientes?serverTimezone=UTC
2  spring.datasource.username=root
3  spring.datasource.password=mintic
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

```

Al ejecutar el proyecto podemos revisar si todo va bien incluyendo el JPA, si deseamos comprobar podemos hacer un cambio a un password

  	<p>ESCUELA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA</p> <p>MICROSERVICIO CON MYSQL</p>	<p>I Semestre 2023</p>
---	---	------------------------

errado y podemos apreciar el error de conexión.

```

: Starting BackClienteApplication using Java 16.0.1 on DESKTOP-1L8LQJ7 with PID 19572 (C:\Entornos\Ejercicios\backCliente\
: No active profile set, falling back to 1 default profile: "default"
or : Devtools property defaults active! Set 'spring.devtools.add-properties' to 'false' to disable
or : For additional web related logging consider setting the 'logging.level.web' property to 'DEBUG'
te : Bootstrapping Spring Data JPA repositories in DEFAULT mode.
te : Finished Spring Data repository scanning in 16 ms. Found 0 JPA repository interfaces.
r : Tomcat initialized with port(s): 8094 (http)
: Starting service [Tomcat]
e : Starting Servlet engine: [Apache Tomcat/9.0.73]
: Initializing Spring embedded WebApplicationContext
xt : Root WebApplicationContext: initialization completed in 3884 ms
: HikariPool-1 - Starting...
: HikariPool-1 - Start completed.
r : HHH000204: Processing PersistenceUnitInfo [name: default]
: HHH000412: Hibernate ORM core version 5.6.15.Final
: HCANN000001: Hibernate Commons Annotations {5.1.2.Final}
: HHH000400: Using dialect: org.hibernate.dialect.MySQL57Dialect
: HHH000490: Using JtaPlatform implementation: [org.hibernate.engine.transaction.jta.platform.internal.NoJtaPlatform]
an : Initialized JPA EntityManagerFactory for persistence unit 'default'
on : spring.jpa.open-in-view is enabled by default. Therefore, database queries may be performed during view rendering. Expli
: LiveReload server is running on port 35729
r : Tomcat started on port(s): 8094 (http) with context path ''
: Started BackClienteApplication in 9.6 seconds (JVM running for 10.881)

```

4. Clase Modelo de Datos.

Cree la clase de modelo de dominio Cliente (debe tener el mismo nombre de la Tabla), sobre un nuevo package llamado modelo por debajo del de por defecto, para mapear con la tabla de Cliente en la base de datos de la siguiente manera:


```




1  package uis.edu.entorno.backCliente.modelo;
2
3  import javax.persistence.Entity;
4  import javax.persistence.GeneratedValue;
5  import javax.persistence.GenerationType;
6  import javax.persistence.Id;
7  import javax.persistence.Table;
8
9  @Entity
10 @Table(name= Cliente.TABLE_NAME)
11 public class Cliente {
12     // Atributos
13     public static final String TABLE_NAME = "cliente";
14     @Id
15     @GeneratedValue(strategy = GenerationType.IDENTITY)
16     private int idcliente;
17     private String documento;
18     private String tipdoc;
19     private String nombres;
20     private String apellidos;
21     private String direccion;
22     private String email;
23
24     public Cliente() {
25     }

```

```

28 public Cliente(int idcliente, String documento, String tipdoc, String nombres,
29 String apellidos, String direccion,
30 String email) {
31     this.idcliente = idcliente;
32     this.documento = documento;
33     this.tipdoc = tipdoc;
34     this.nombres = nombres;
35     this.apellidos = apellidos;
36     this.direccion = direccion;
37     this.email = email;
38 }
39
40 public int getIdcliente() {
41     return idcliente;
42 }
43
44 public void setIdcliente(int idcliente) {
45     this.idcliente = idcliente;
46 }
47
48 public String getDocumento() {
49     return documento;
50 }
51
52 public void setDocumento(String documento) {
53     this.documento = documento;
54 }
55
56 public String getTipdoc() {
57     return tipdoc;
58 }
59
60 public void setTipdoc(String tipdoc) {
61     this.tipdoc = tipdoc;
62 }
63
64 public String getNombres() {
65     return nombres;
66 }
67
68 public void setNombres(String nombres) {
69     this.nombres = nombres;
70 }
71
72 public String getApellidos() {
73     return apellidos;
74 }
75
76 public void setApellidos(String apellidos) {
77     this.apellidos = apellidos;
78 }
79
80 public String getDireccion() {
81     return direccion;
82 }
83
84 public void setDireccion(String direccion) {
85     this.direccion = direccion;
86 }
87
88 public String getEmail() {
89     return email;
90 }
91

```

  	<p>ESCUELA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA</p> <p>MICROSERVICIO CON MYSQL</p>	<p>I Semestre 2023</p>
---	---	------------------------

```

78     public void setEmail(String email) {
79         this.email = email;
80     }
81     @Override
82     public String toString() {
83         return "Cliente [idcliente=" + idcliente + ", documento=" + documento
84             + ", tipdoc=" + tipdoc + ", nombres=" + nombres +
85             ", apellidos=" + apellidos + ", direccion=" + direccion
86             + ", email=" + email + "]";
87     }
88 }

```

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.

5. Interface repositorio(DAO).

Cree la **interface** del repositorio de usuarios ampliando el repositorio JPA.

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

A continuación, cree la interfaz ClienteRepositorio sobre un nuevo package llamado repositorio de la siguiente manera:




```

1     package uis.edu.entorno.backCliente.repositorio;
2
3     import org.springframework.data.jpa.repository.JpaRepository;
4     import uis.edu.entorno.backCliente.modelo.Cliente;
5
6     public interface ClienteRepositorio extends JpaRepository<Cliente, Integer>{
7
8     }

```

6. Clase ClienteServicio (DAO).

Debemos definir los métodos a utilizar dentro del microservicio para ello

  	<p>ESCUELA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA</p> <p>MICROSERVICIO CON MYSQL</p>	<p>I Semestre 2023</p>
---	---	------------------------

en el package de servicio agregamos una interface con las con los encabezado de las funcionalidades, necesitamos codificar la interface IClienteServicio en la capa de servicio / negocio con el siguiente código:

```

1  package uis.edu.entorno.backCliente.servicio;
2
3  import java.util.List;
4  import uis.edu.entorno.backCliente.modelo.Cliente;
5
6
7  public interface IClienteServicio {
8
9      // Listar todos los clientes
10     public List<Cliente> getClientes();
11
12     // Buscar un cliente por id
13     public Cliente getCliente(Integer id);
14
15     // Guardar un cliente
16     public Cliente grabarCliente(Cliente cliente);
17
18     // Eliminar un Cliente
19     public void delete(Integer id);
20
21 }
```

Cree una clase de ClienteServicio para codificar la lógica del negocio y actúa como una capa intermedia entre el repositorio y la clase de controlador.

Se utiliza la anotación @Transactional anotar métodos se ejecutan en transacciones.




Ahora creamos la clase basada en la interface creada con anticipación

```

1  package uis.edu.entorno.backCliente.servicio;
2
3  import java.util.List;
4  import javax.transaction.Transactional;
5  import org.springframework.beans.factory.annotation.Autowired;
6  import org.springframework.stereotype.Service;
7
8  import uis.edu.entorno.backCliente.modelo.Cliente;
9  import uis.edu.entorno.backCliente.repositorio.ClienteRepositorio;
10
11  @Service
12  @Transactional
13  public class ClienteServicio implements IClienteServicio{
14
15      @Autowired
16      private ClienteRepositorio clienteRepo;
17
18      @Override
19      public List<Cliente> getClientes() {
20          return clienteRepo.findAll();
21      }
22
23      @Override
24      public Cliente getCliente(Integer id) {
25          return clienteRepo.findById(id).orElse(null);
26      }
27
28      @Override
29      public Cliente grabarCliente(Cliente cliente) {
30          return clienteRepo.save(cliente);
31      }
32
33      @Override
34      public void delete(Integer id) {
35          // TODO Auto-generated method stub
36          clienteRepo.deleteById(id);
37      }
38
39
40  }

```

7. Clase Controlador(Métodos CRUD).

  	<p>ESCUELA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA</p> <p>MICROSERVICIO CON MYSQL</p>	<p>I Semestre 2023</p>
---	--	------------------------

A continuación, cree la clase ClienteController 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 ClienteServicio 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 uis.edu.entorno.backCliente.controlador;
2
3  import java.util.List;
4
5  import org.springframework.beans.factory.annotation.Autowired;
6  import org.springframework.http.HttpStatus;
7  import org.springframework.http.ResponseEntity;
8  import org.springframework.web.bind.annotation.CrossOrigin;
9  import org.springframework.web.bind.annotation.GetMapping;
10 import org.springframework.web.bind.annotation.DeleteMapping;
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.RequestMapping;
16 import org.springframework.web.bind.annotation.RestController;
17
18 import uis.edu.entorno.backCliente.modelo.Cliente;
19 import uis.edu.entorno.backCliente.servicio.ClienteServicio;
20
21 @RestController
22 @CrossOrigin("*")
23 @RequestMapping("/api/clientes")
24 public class ClienteController {

```

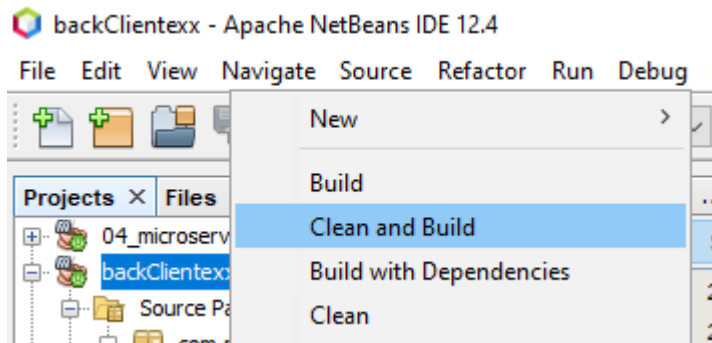
```

25
26     @Autowired
27     private ClienteServicio clienteService;
28
29     @GetMapping("/list")
30     public List<Cliente> consultarTodo() {
31         return (clienteService.getClientes());
32     }
33
34     @GetMapping("/list/{id}")
35     public Cliente buscarPorId(@PathVariable Integer id) {
36         return clienteService.getCliente(id);
37     }
38
39     @PostMapping("/")
40     public ResponseEntity<Cliente> agregar(@RequestBody Cliente cliente) {
41         Cliente obj = clienteService.grabarCliente(cliente);
42         return new ResponseEntity<>(obj, HttpStatus.OK);
43     }
44
45     @PutMapping("/")
46     public ResponseEntity<Cliente> editar(@RequestBody Cliente cliente) {
47         Cliente obj = clienteService.getCliente(cliente.getIdcliente());
48         if(obj != null) {
49             obj.setDireccion(cliente.getDireccion());
50             obj.setApellidos(cliente.getApellidos());
51             obj.setDocumento(cliente.getDocumento());
52             obj.setEmail(cliente.getEmail());
53             obj.setNombres(cliente.getNombres());
54             obj.setTipdoc(cliente.getTipdoc());
55             clienteService.grabarCliente(obj);
56         } else {
57             return new ResponseEntity<>(obj, HttpStatus.INTERNAL_SERVER_ERROR);
58         }
59         return new ResponseEntity<>(obj, HttpStatus.OK);
60     }
61
62     @DeleteMapping("/{id}")
63     public ResponseEntity<Cliente> eliminar(@PathVariable Integer id) {
64         Cliente obj = clienteService.getCliente(id);
65         if(obj != null) {
66             clienteService.delete(id);
67         } else {
68             return new ResponseEntity<>(obj, HttpStatus.INTERNAL_SERVER_ERROR);
69         }
70         return new ResponseEntity<>(obj, HttpStatus.OK);
71     }
72
73 }

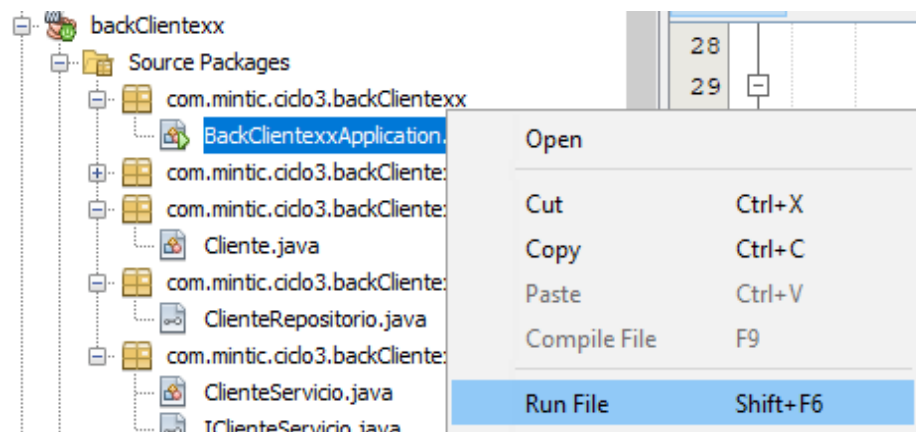
```

10. Compile y ejecute el proyecto.

Se recomienda hacer Clean and Build en el proyecto web, esto genera de nuevo el archivo jar que ejecuta el proyecto, ubicándonos sobre el nombre del proyecto, dando click derecho y seleccionando la opción que muestra la gráfica.






Para ejecutar el proyecto se puede ejecutar dando click derecho sobre el archivo de ejecución y seleccionando sobre click derecho la opción Run File.



Al ejecutar podemos probar en nuestro navegador los métodos get con la siguiente instrucción:

<http://localhost:8094/api/clientes/list>

  	<p>ESCUELA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA</p> <p>MICROSERVICIO CON MYSQL</p>	<p>I Semestre 2023</p>
---	---	------------------------

← → ↻ ⓘ localhost:8094/api/clientes/list

Personal MINTIC Cisco Webex

```
[
  {
    idcliente: 1,
    documento: "736074826",
    tipdoc: "CC",
    nombres: "Sindy Nathalia",
    apellidos: "Chapeta Gamboa",
    direccion: "Carrtera 12 # 10 - 23",
    email: "Chapeta@gmail.com.co"
  },
  {
    idcliente: 2,
    documento: "712152752",
    tipdoc: "CC",
    nombres: "Gaston Eduardo",
    apellidos: "Rojas Segura",
    direccion: "Carrtera 32 # 100 - 124",
    email: "GRojas@hotmail.com.co"
  },
  {
    idcliente: 8,
    documento: "712767626",
    tipdoc: "CC",
    nombres: "Nicklass Rodrigo",
    apellidos: "Garcia Gomez",
    direccion: "Carrtera 121 # 10 - 30",
    email: "GarciaGNicklass@hotmail.com.co"
  }
],
```

<http://localhost:8094/api/clientes/list/8>

← → ↻ ⓘ localhost:8094/api/clientes/list/8

Personal MINTIC Cisco Webex

```
{
  idcliente: 8,
  documento: "712767626",
  tipdoc: "CC",
  nombres: "Nicklass Rodrigo",
  apellidos: "Garcia Gomez",
  direccion: "Carrtera 121 # 10 - 30",
  email: "GarciaGNicklass@hotmail.com.co"
}
```

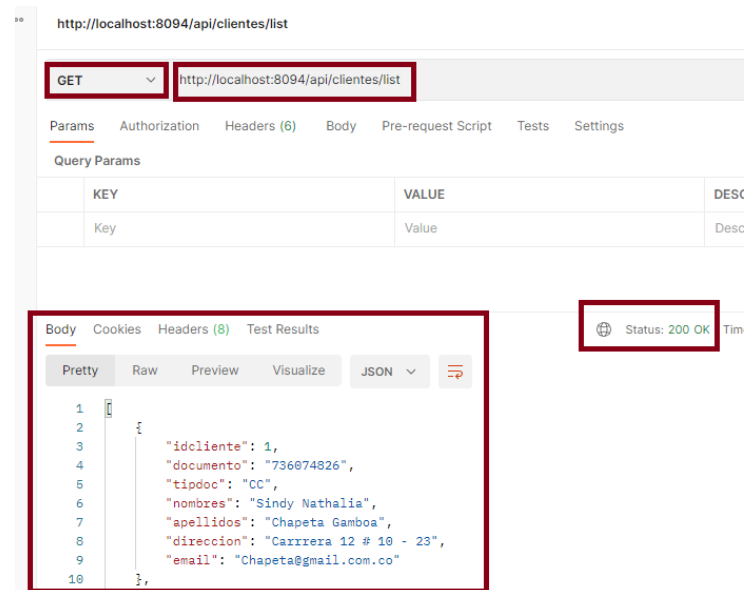
11. Prueba con Postman.

Ingresamos a Postman y probamos cada uno de los endpoint creados en el proyecto en la clase ClienteController.

- **Lista de Clientes.**

Vamos a simular la ejecución del método consultarTodo, el endpoint sería <http://localhost:8094/api/clientes/list> usando el get.

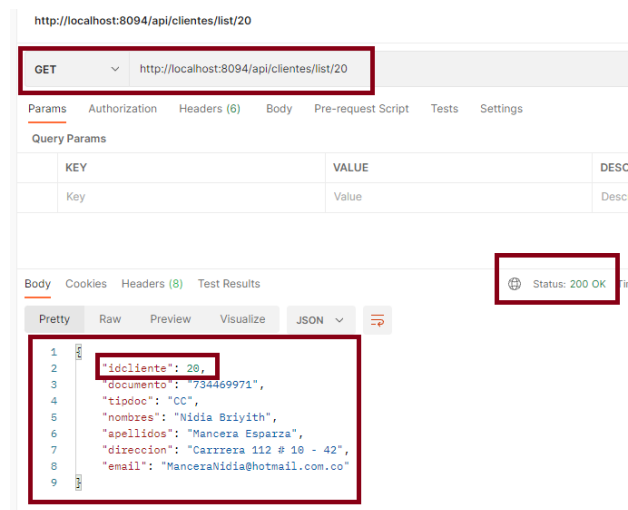
```
@GetMapping("/list")
public List<Cliente> consultarTodo() {
    return (clienteService.getClientes());
}
```



- **Buscar por id el Cliente.**

Vamos a simular la ejecución del método buscarPorId, el endpoint sería `http://localhost:8094/api/clientes/list/20` usando el get.

```
@GetMapping("/list/{id}")
public Cliente buscarPorId(@PathVariable Integer id) {
    return clienteService.getCliente(id);
}
```

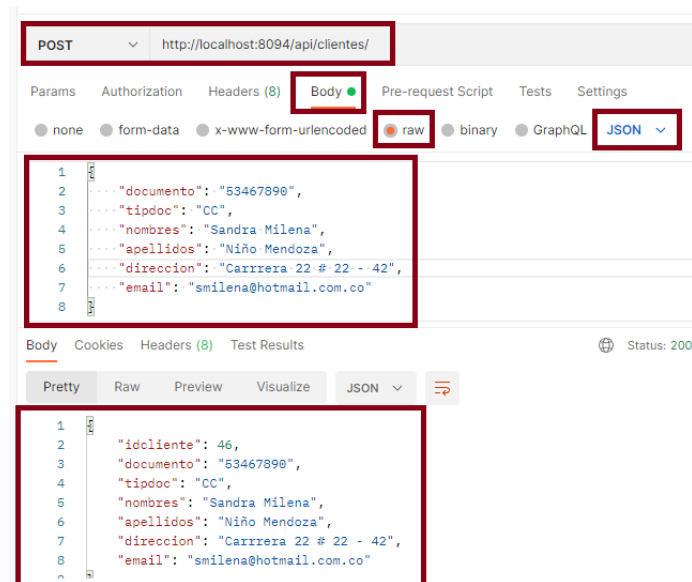


- **Crear un Cliente.**

Vamos a simular la ejecución del método agregar, el endpoint sería `http://localhost:8094/api/clientes/` usando el post.

```
@PostMapping("/")
public ResponseEntity<Cliente> agregar(@RequestBody Cliente cliente) {
    Cliente obj = clienteService.grabarCliente(cliente);
    return new ResponseEntity<>(obj, HttpStatus.OK);
}
```

Ejecución en Postman



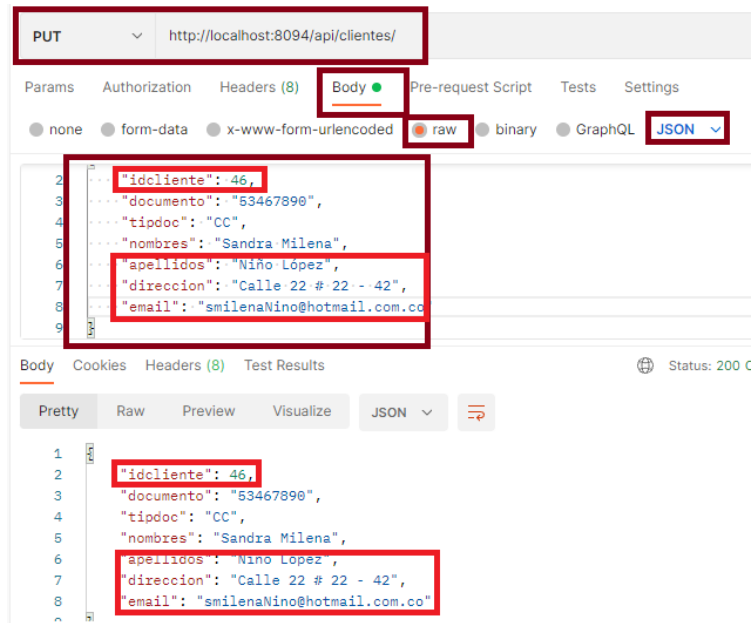
• Actualizar un Cliente.

Vamos a simular la ejecución del método editar, el endpoint sería `http://localhost:8094/api/clientes/` usando el Put.

```
@PostMapping("/")
public ResponseEntity<Cliente> editar(@RequestBody Cliente cliente) {
    Cliente obj = clienteService.getClientes(cliente.getIdcliente());
    if(obj != null) {
        obj.setDireccion(cliente.getDireccion());
        obj.setApellidos(cliente.getApellidos());
        obj.setDocumento(cliente.getDocumento());
        obj.setEmail(cliente.getEmail());
        obj.setNombres(cliente.getNombres());
        obj.setTipdoc(cliente.getTipdoc());
        clienteService.grabarCliente(obj);
    } else {
        return new ResponseEntity<>(obj, HttpStatus.INTERNAL_SERVER_ERROR);
    }

    return new ResponseEntity<>(obj, HttpStatus.OK);
}
```

Ejecución en postman



- **Eliminar un Cliente.**

Vamos a simular la ejecución del método eliminar, el endpoint sería `http://localhost:8094/api/clientes/id` usando el Delete.

```
@DeleteMapping("/{id}")
public ResponseEntity<Cliente> eliminar(@PathVariable Integer id) {
    Cliente obj = clienteService.getCliente(id);
    if(obj != null) {
        clienteService.delete(id);
    }else {
        return new ResponseEntity<>(obj, HttpStatus.INTERNAL_SERVER_ERROR);
    }

    return new ResponseEntity<>(obj,HttpStatus.OK);
}
```

Ejecución postman

http://localhost:8094/api/clientes/46

DELETE

▼

http://localhost:8094/api/clientes/46

Params Authorization Headers (6) Body Pre-request Script Tests Settings

Query Params

KEY	VALUE	DE
Key	Value	Des

Body Cookies Headers (8) Test Results

Pretty

Raw

Preview

Visualize

JSON ▼

```
1  {
2    "idcliente": 46,
3    "documento": "53467890",
4    "tipdoc": "CC",
5    "nombres": "Sandra Milena",
6    "apellidos": "Niño López",
7    "direccion": "Calle 22 # 22 - 42",
8    "email": "smilenaNino@hotmail.com.co"
9  }
```

🌐

Status: 200 OK