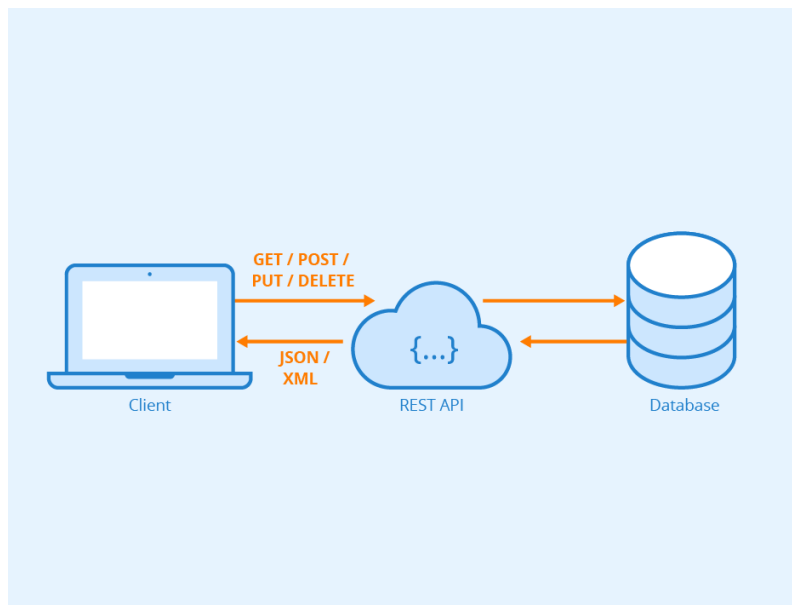


MANUAL SPRING y Api Rest



Indices

Indices	2
1) SPRING ¿Qué es?.....	3
2) Iniciar proyecto	3
3) Usar archivo descomprimido	4
4) Partes del archivo	5
5) Descargar maven.....	8
6) Probando servidor.....	11
7) Hola mundo con Spring boot	12
8) API Rest full Java Spring Boot MySQL:	14
8.1) Conectamos la base de datos.....	15
8.2) Modelo: Crear tabla desde el editor	16
8.3) Revisar conexión de la base de datos	17
8.4) Flujo de trabajo	19
8.5) Repositorio	20
8.6) Servicios.....	20
8.7) Controlador	22
8.8) Revisar funcionamiento de nuestro servidor	23
8.9) Dar de alta usuarios.....	24
8.10) Comprobar registros desde postman.....	26
8.11) Comprobar registros desde el navegador	26
8.12) Actualizar registro desde postman	26
8.13) Consultar o buscar, y borrar por parámetro (lógica)	27
8.13.1) Repositorio	27
8.13.2) Servicios.....	28
8.13.3) Controlador	30
8.14) Consultar o buscar, y borrar por parámetro (pruebas).....	32
8.14.1) Buscar por id.....	32
8.14.2) Buscar por campo prioridad	33
8.14.3) Eliminar registro	33

1) SPRING ¿Qué es?

Spring es un framework del lenguaje de programación java, y un framework en programación es el resultado de la evolución de la ingeniería del software, estos son creados por programadores para programadores, con la finalidad de estandarizar el trabajo, resolver, agilizar y manejar los problemas y complejidades que van apareciendo en el mundo de la programación, a medida las exigencias van creciendo. Creando así, en la comunidad de desarrolladores, un abanico de posibilidades para una creación cada vez más evolucionada de aplicaciones.

Spring nos permite desarrollar aplicaciones de manera más rápida, eficaz y corta, saltándonos tareas repetitivas y ahorrándonos líneas de código.

Spring framework es muy extenso y crece día a día para ayudar al desarrollo de aplicaciones web. A continuación, les vamos a explicar una de sus funciones básicas, la inyección de dependencias de Spring (Spring Di).

2) Iniciar proyecto

Entramos a: <https://start.spring.io/>

Project	Language
<input type="radio"/> Gradle - Groovy	<input checked="" type="radio"/> Java <input type="radio"/> Kotlin
<input type="radio"/> Gradle - Kotlin	<input type="radio"/> Groovy
<input checked="" type="radio"/> Maven	

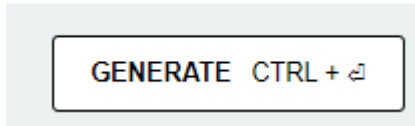
Spring Boot	
<input type="radio"/> 3.0.1 (SNAPSHOT)	<input checked="" type="radio"/> 3.0.0 <input type="radio"/> 2.7.7 (SNAPSHOT)
<input type="radio"/> 2.7.6	

Project Metadata	
Group	<input type="text" value="com.example"/>
Artifact	<input type="text" value="demo"/>
Name	<input type="text" value="demo"/>
Description	<input type="text" value="Demo project for Spring Boot"/>
Package name	<input type="text" value="com.example.demo"/>
Packaging	<input type="radio"/> Jar <input checked="" type="radio"/> War
Java	<input type="radio"/> 19 <input type="radio"/> 17 <input type="radio"/> 11 <input checked="" type="radio"/> 8

Dejamos las opciones como se muestra, podemos cambiar la versión de java

El cuadro rojo indica el nombre que pondremos
Artifact nombre del proyecto a nivel de clases
Name alias del proyecto
Description descripción

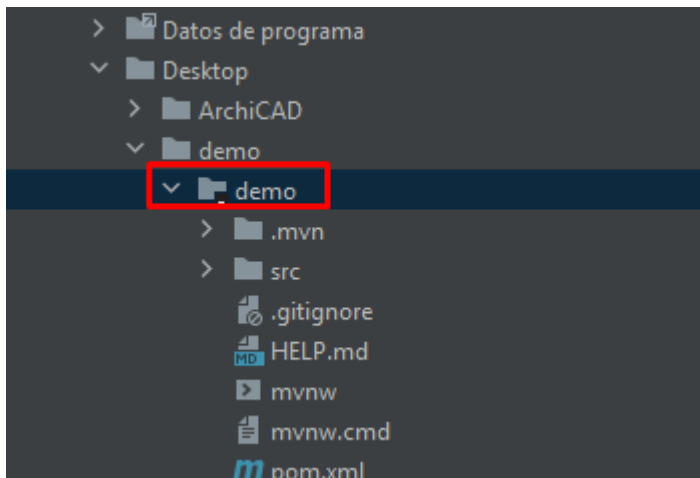
Presionamos



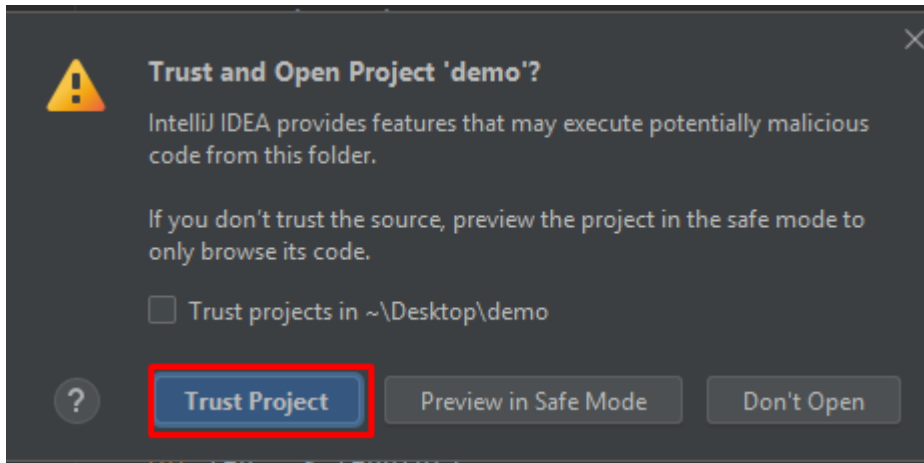
Descargas y descomprimes

3) Usar archivo descomprimido

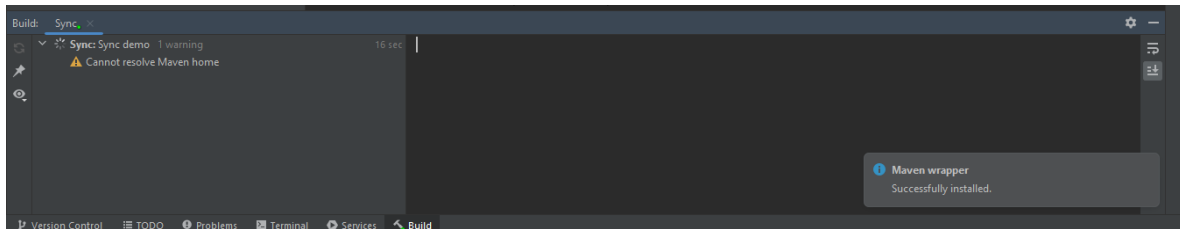
Para usarlo necesitamos un editor de texto, en este caso podemos usar IntelliJ IDEA o VISUAL STUDIO CODE



Damos clic sobre la carpeta con el cuadro negro



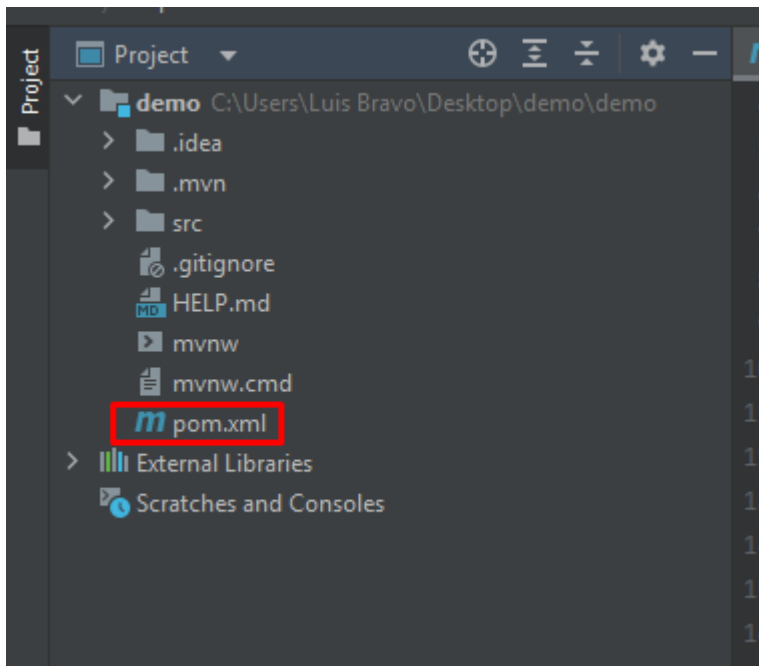
Seleccionamos que si confiamos en el proyecto



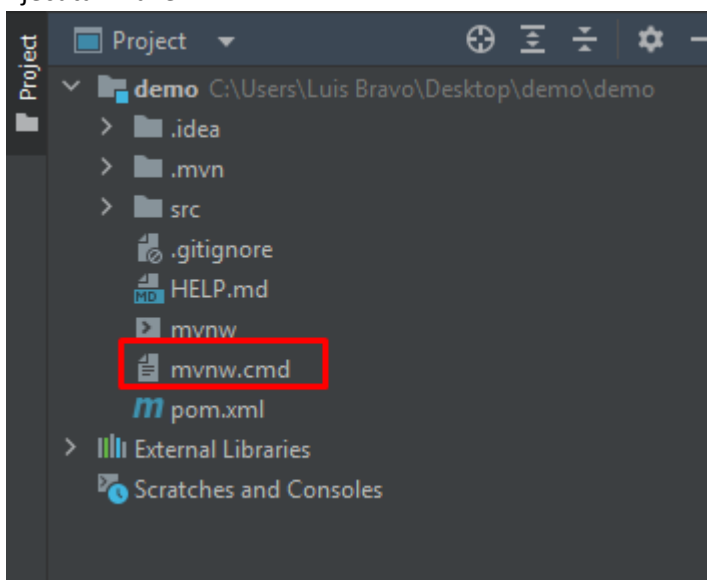
Si es la primera vez que lo usan el editor comenzara a descargar lo que necesite para su correcto funcionamiento

4) Partes del archivo

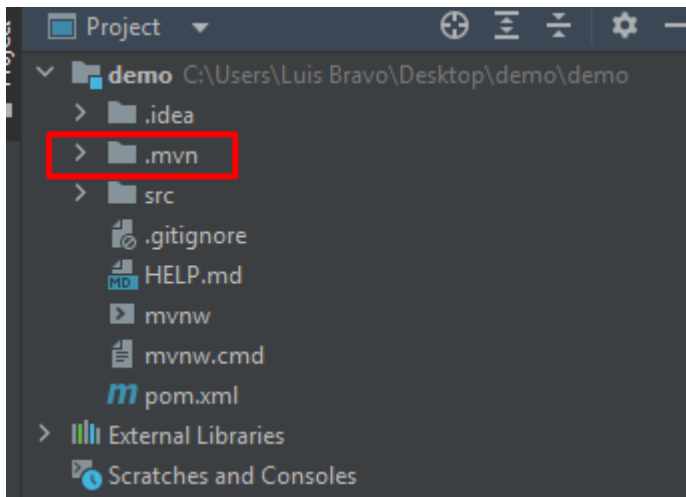
Dependencias que necesita el programa



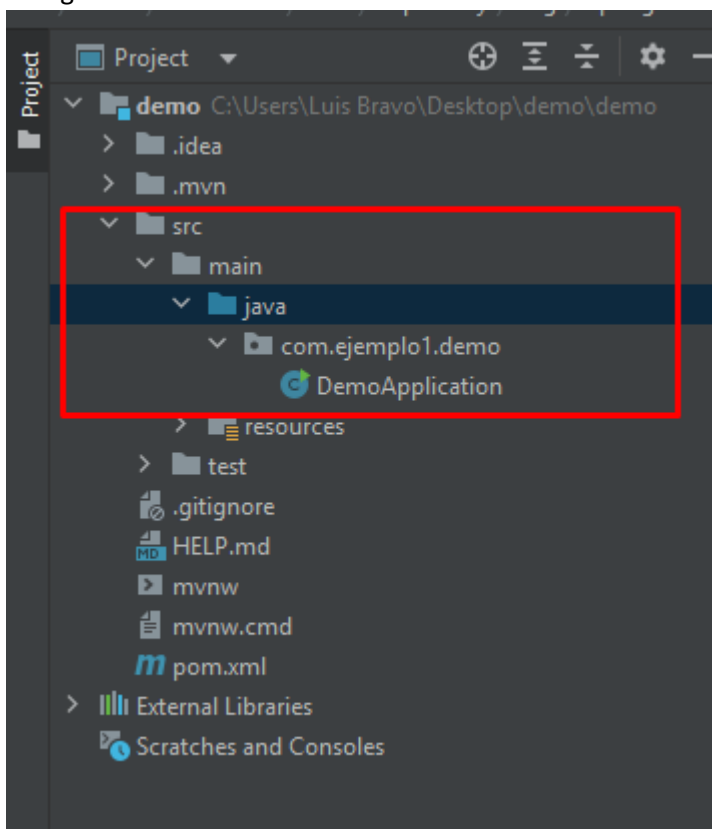
Ejecutar maven



Archivos de maven



Código



5) Descargar maven

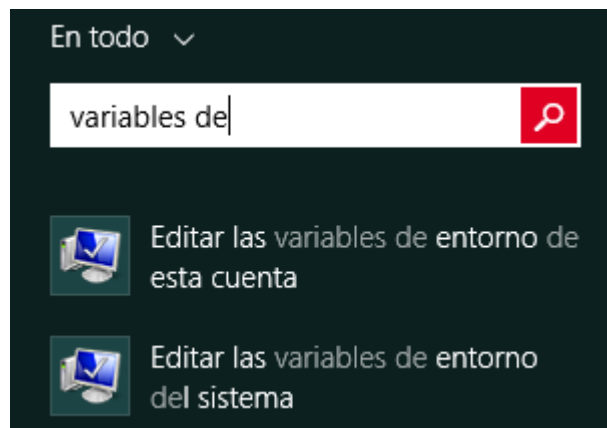
Vamos a <https://maven.apache.org/download.cgi> y descargamos el siguiente archivo

	Link	Checksums	Signature
Binary tar.gz archive	apache-maven-3.8.6-bin.tar.gz	apache-maven-3.8.6-bin.tar.gz.sha512	apache-maven-3.8.6-bin.tar.gz.asc
Binary zip archive	apache-maven-3.8.6-bin.zip	apache-maven-3.8.6-bin.zip.sha512	apache-maven-3.8.6-bin.zip.asc
Source tar.gz archive	apache-maven-3.8.6-src.tar.gz	apache-maven-3.8.6-src.tar.gz.sha512	apache-maven-3.8.6-src.tar.gz.asc
Source zip archive	apache-maven-3.8.6-src.zip	apache-maven-3.8.6-src.zip.sha512	apache-maven-3.8.6-src.zip.asc

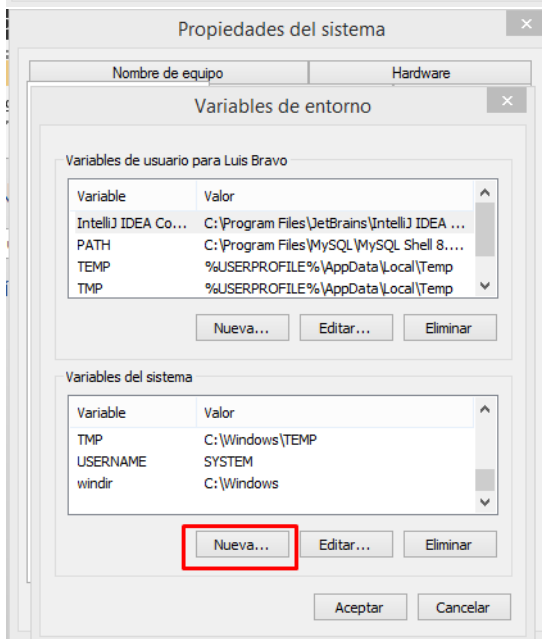
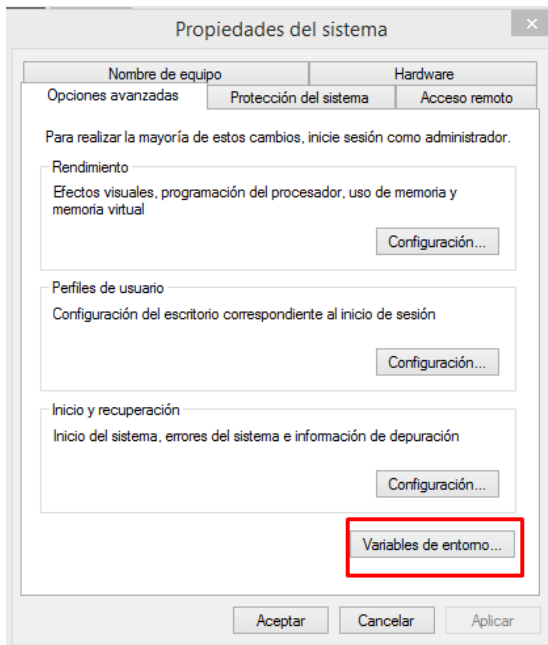
Descomprimos y pegamos en el disco C

Disco local (C:) ▶				
Nombre	Fecha de modifica...	Tipo	Tamaño	
mib.bin	30/08/2022 08:56 ...	Archivo BIN	1 KB	
parte15.prn	10/11/2016 03:04 ...	Archivo PRN	11,266 KB	
apache-maven-3.8.6	21/12/2022 09:46 a...	Carpeta de archivos		

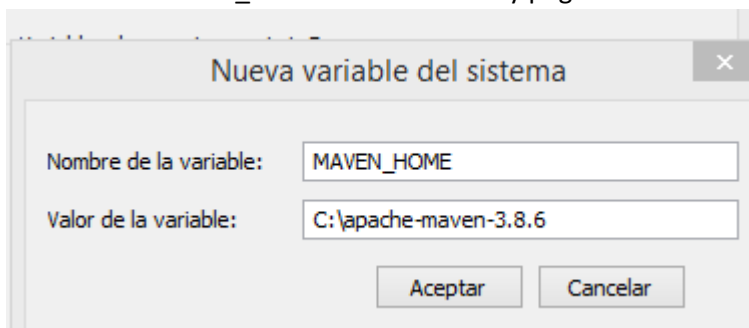
Copia la ruta de la carpeta por ejemplo: C:\apache-maven-3.8.6 , ahora abre el buscador y escribe variables de entorno del sistema



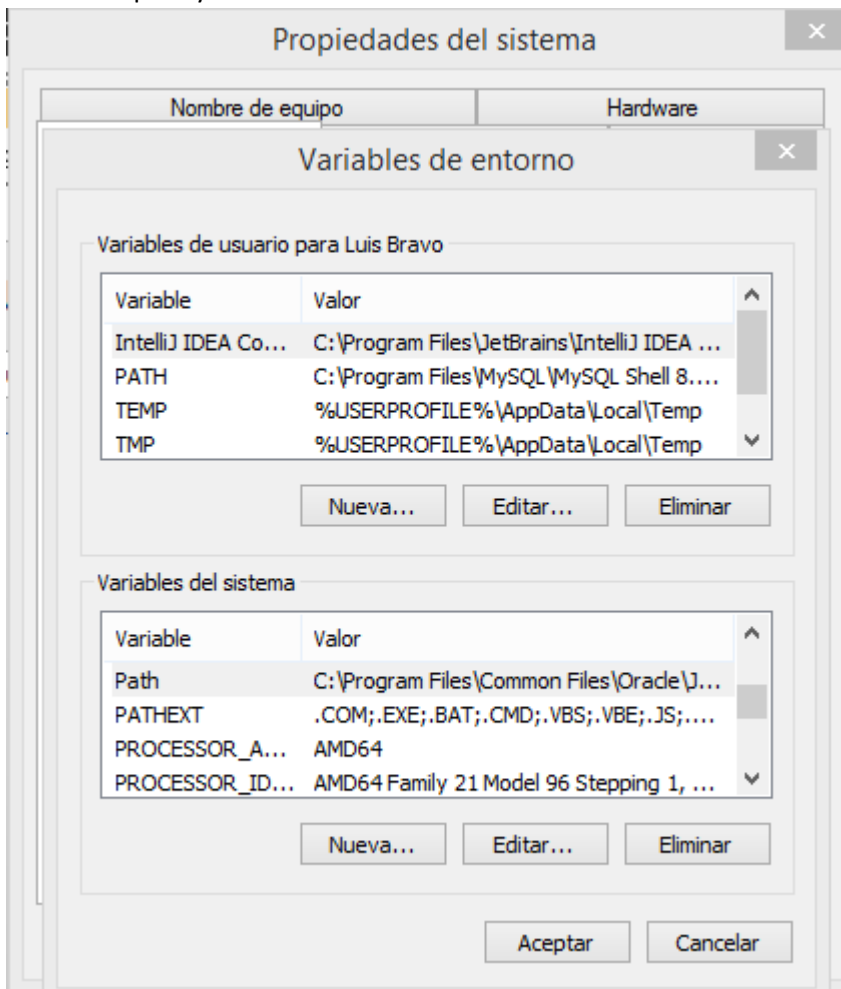
Clic en



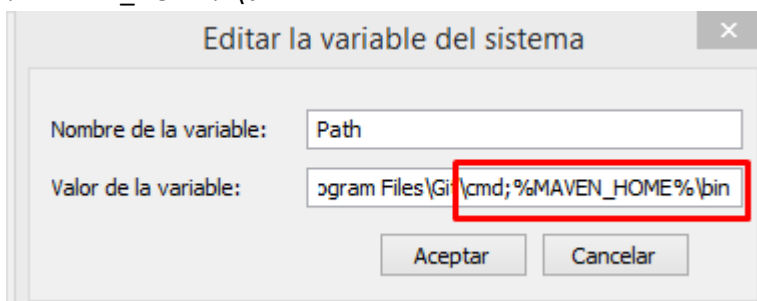
Colocamos MAVEN_HOME como nombre y pegamos la ruta



Buscamos path y clic en editar

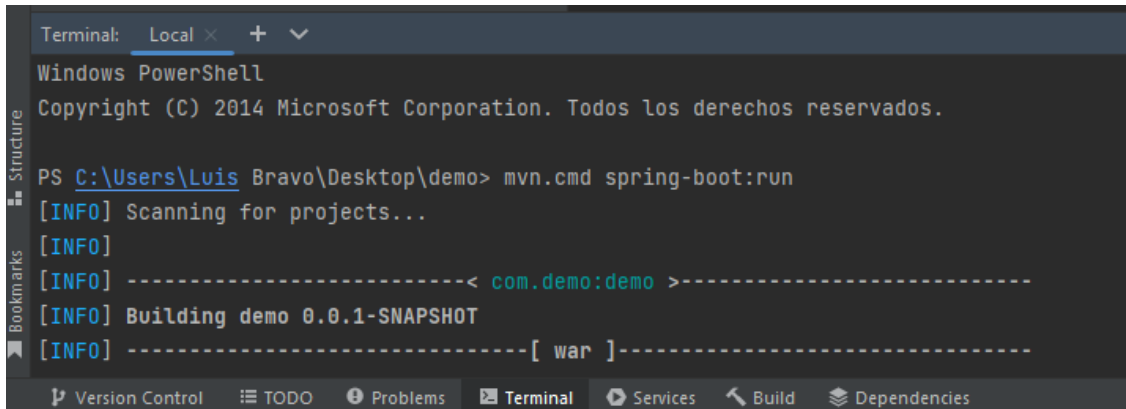


Dependiendo del sistema, por ejemplo window 10 en adelante tenemos la opción de agregar un nuevo valor, para window 8, nos vamos hasta el final ponemos punto y coma y como y escribimos %MAVEN_HOME%\bin



6) Probando servidor

Vamos a nuestro editor y abrimos la terminal, para el caso de IntelliJ IDEA se abre con ALT+F12 y escribimos mvn.cmd spring-boot:run o mvnw.cmd spring-boot:run

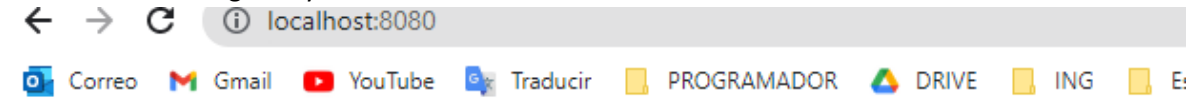


```
Terminal: Local x + v
Windows PowerShell
Copyright (C) 2014 Microsoft Corporation. Todos los derechos reservados.

PS C:\Users\Luis Bravo\Desktop\demo> mvn.cmd spring-boot:run
[INFO] Scanning for projects...
[INFO]
[INFO] -----< com.demo:demo >-----
[INFO] Building demo 0.0.1-SNAPSHOT
[INFO] -----[ war ]-----

Version Control  TODO  Problems  Terminal  Services  Build  Dependencies
```

Nos vamos al navegador y entramos a localhost:8080



Whitelabel Error Page

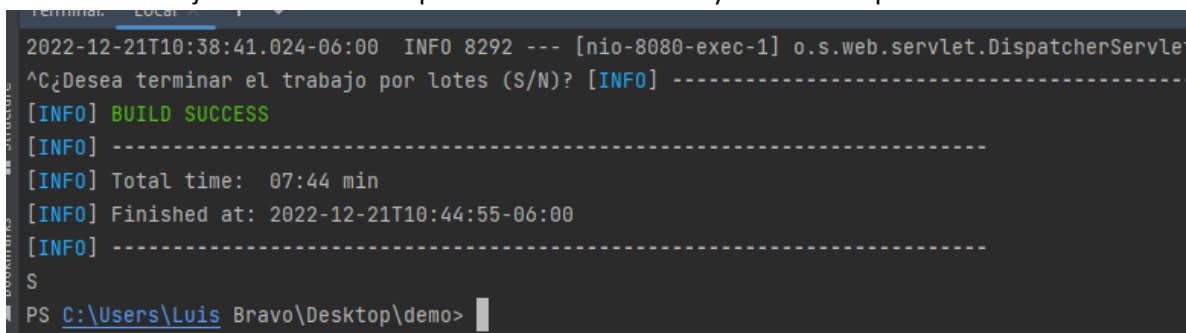
This application has no explicit mapping for /error, so you are seeing this as a fallback.

Wed Dec 21 10:38:41 CST 2022

There was an unexpected error (type=Not Found, status=404).

Nos saldrá un error porque aún no programamos nada, pero podemos ver la hora en que se creó el servidor y con eso verificamos que si funciona

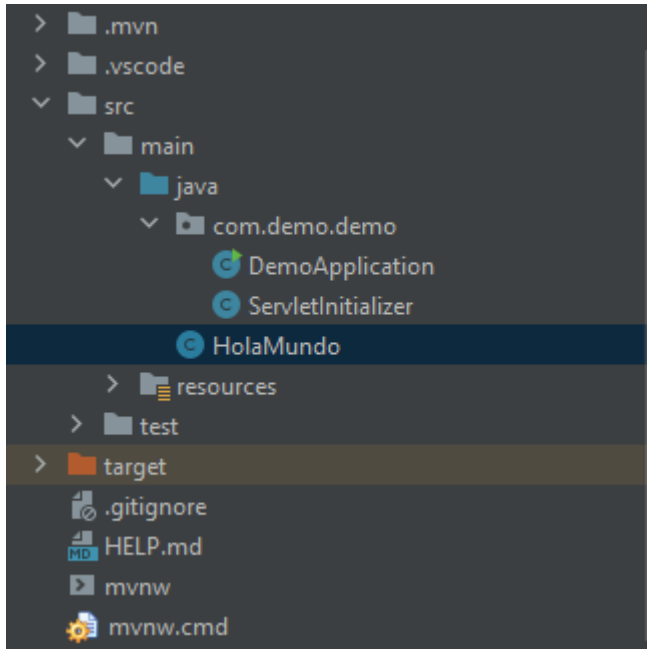
En la terminal ejecutamos CTRL+C para detener el servidor y le decimos que si



```
Terminal: Local x + v
2022-12-21T10:38:41.024-06:00 INFO 8292 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet
^C¿Desea terminar el trabajo por lotes (S/N)? [INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 07:44 min
[INFO] Finished at: 2022-12-21T10:44:55-06:00
[INFO] -----
S
PS C:\Users\Luis Bravo\Desktop\demo>
```

7) Hola mundo con Spring boot

Creamos una nueva clase en la carpeta de java



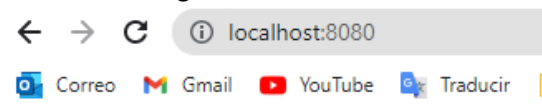
Escribimos el siguiente código

```
import org.springframework.web.bind.annotation.RequestMapping; //
// biblioteca para Reques
import org.springframework.web.bind.annotation.RestController; //
// biblioteca para restcontroller
@RestController // indica que la clase es un controlador
public class HolaMundo {
    @RequestMapping("/") // indica que se ejecute el metodo desde la raiz
    // desde el servidor
    // es decir desde que se ejecuta
    public String hola(){
        return "hola mundo";
    }
}
```

Ejecutan el servidor con mvn.cmd spring-boot:run

```
S
PS C:\Users\Luis Bravo\Desktop\demo> mvn.cmd spring-boot:run
[INFO] Scanning for projects...
[INFO]
[INFO] -----< com.demo:demo >-----
[INFO] Building demo 0.0.1-SNAPSHOT
[INFO] -----[ war ]-----
[INFO]
[INFO] >>> spring-boot-maven-plugin:3.0.0:run (default-cli) > test-compile @ demo >>>
[INFO]
[INFO] --- maven-resources-plugin:3.3.0:resources (default-resources) @ demo ---
[INFO] Copying 1 resource
[INFO] Copying 0 resource
[INFO]
[INFO] --- maven-compiler-plugin:3.10.1:compile (default-compile) @ demo ---
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 3 source files to C:\Users\Luis Bravo\Desktop\demo\target\classes
[INFO]
```

Vamos al navegador al localhost:8080



hola mundo

En caso de tener problemas podemos ejecutar cmd para trabajar

```
C:\Users\Luis Bravo> mvnw.cmd
Microsoft Windows [Versión 6.3.9600]
(c) 2013 Microsoft Corporation. Todos los derechos reservados.

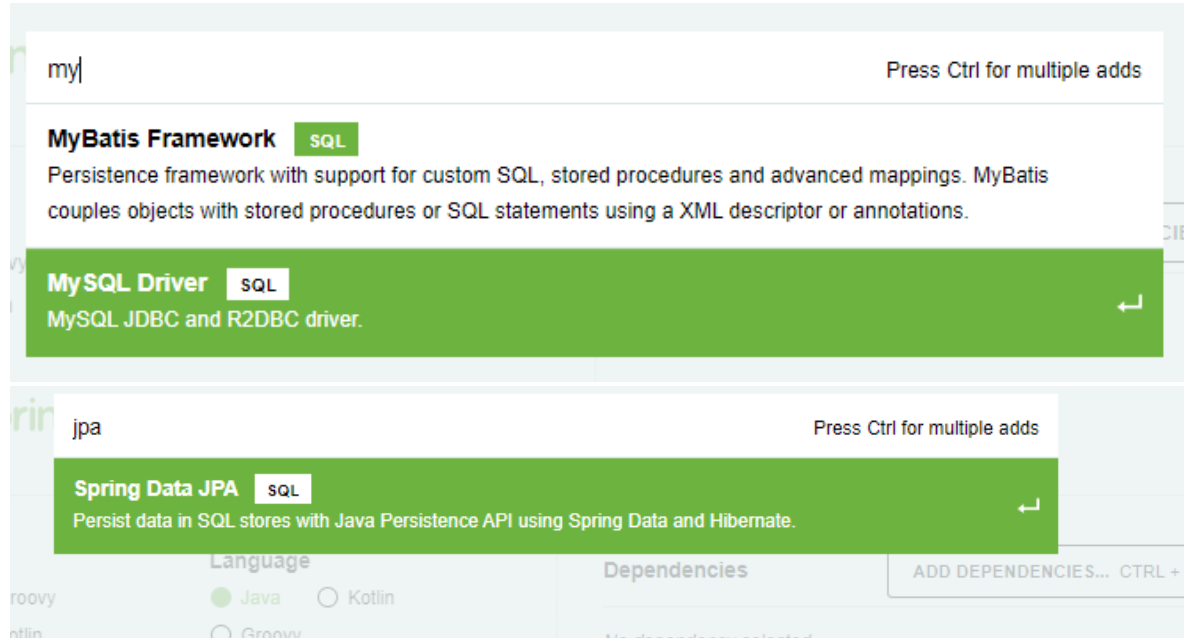
C:\Users\Luis Bravo> cd desktop
C:\Users\Luis Bravo\Desktop> cd pueba2
C:\Users\Luis Bravo\Desktop\pueba2> mvnw.cmd spring-boot:run
[INFO] Scanning for projects...
[INFO]
[INFO] -----< com.pueba2:pueba2 >-----
[INFO] Building pueba2 0.0.1-SNAPSHOT
[INFO] -----[ war ]-----
[INFO]
[INFO] >>> spring-boot-maven-plugin:3.0.0:run (default-cli) > test-compile @ pue
ba2 >>>
[INFO]
[INFO] --- maven-resources-plugin:3.3.0:resources (default-resources) @ pueba2 -
[INFO] Copying 1 resource
[INFO] Copying 0 resource
[INFO]
[INFO] --- maven-compiler-plugin:3.10.1:compile (default-compile) @ pueba2 ---
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 3 source files to C:\Users\Luis Bravo\Desktop\pueba2\target\cla
```

+

8) API Rest full Java Spring Boot MySQL:

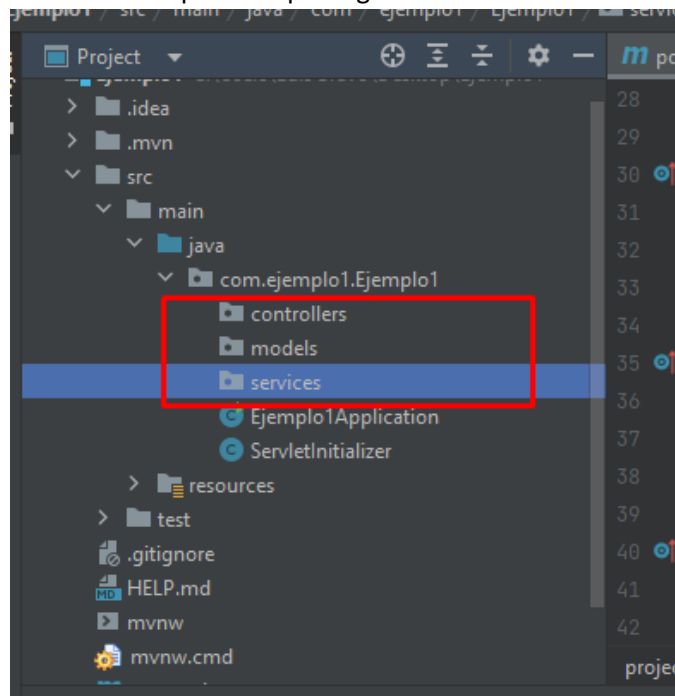
Vamos a <https://start.spring.io/>

Al igual que hicimos en [Iniciar proyecto](#), pero ahora vamos a agregar dependencias que son MySQL y JPA



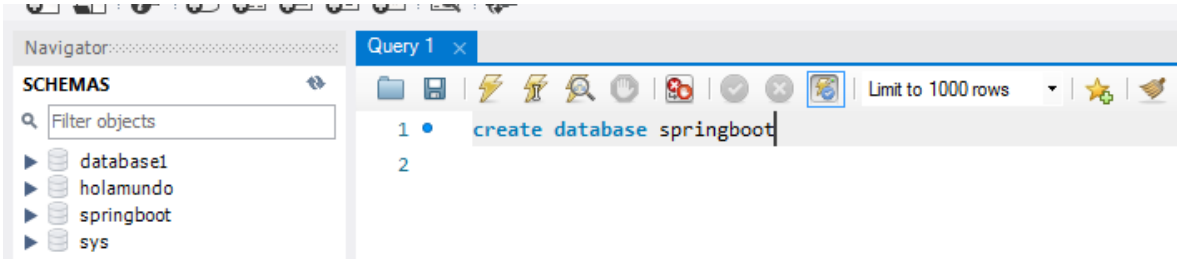
MySQL para administrar los datos y JPA para mapear (El mapeo de datos es el proceso de integración de campos de muchos conjuntos de datos en un diseño, o base de datos centralizada), ponemos un nombre en este caso ejemplo1, descargamos y abrimos con nuestro editor

Creamos 3 carpetas o packages

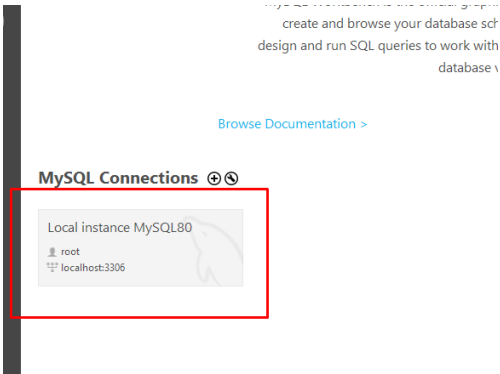


8.1) Conectamos la base de datos

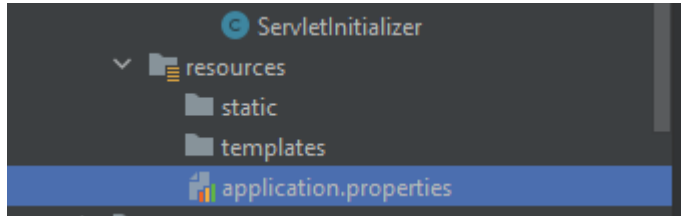
Para ello abrimos MySQL y creamos una base de datos llamada **springboot**



Dejamos nuestra base de datos sin tablas pues crearemos estas desde el editor de texto, tomamos nuestro nombre de usuario y nuestra localización de servidor



Regresamos al editor de texto y vamos a la carpeta resources y abrimos application.properties



Escribimos el siguiente código

```
spring.datasource.url= jdbc:mysql://localhost:3306/springboot
spring.datasource.username =root
spring.datasource.password=123456
spring.jpa.hibernate.ddl-auto=update
```

Nos debe quedar de esta forma

```
spring.datasource.url= jdbc:mysql://localhost:3306/springboot
spring.datasource.username =root
spring.datasource.password=123456
spring.jpa.hibernate.ddl-auto=update
```

A continuación se coloca el funcionamiento de cada línea de código

[//cadena de conexión estandarizada 35.225.57.117 es el ip](#)
[// 3306 es el puerto y springboot es el nombre de la base de datos](#)

spring.datasource.url=jdbc:mysql://localhost:3306/springboot

[//usuario de nuestro servidor que esta en MySQL](#)

spring.datasource.username=root

[//contraseña que definimos para nuestro servidor en MySQL](#)

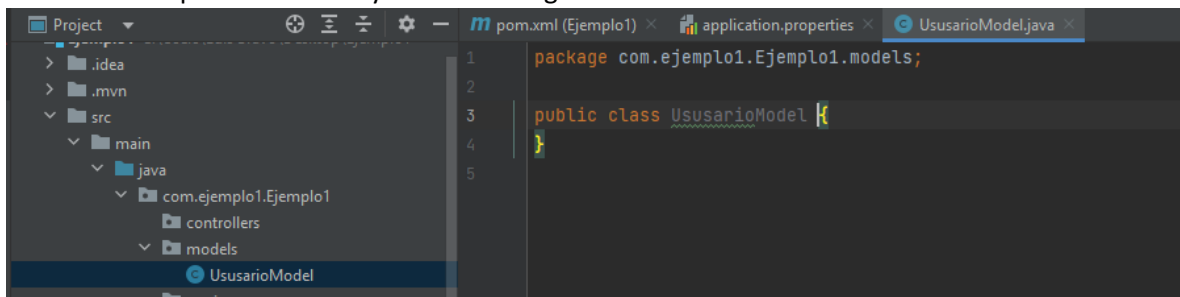
spring.datasource.password=123456

[//definir creacion de base de datos none si ya esta creada o update si queremos crearla](#)

spring.jpa.hibernate.ddl-auto=update

8.2) Modelo: Crear tabla desde el editor

Vamos a la carpeta de models y creamos la siguiente clase: UsuarioModel



[//por error escribi mal Usuario, en su lugar puse Ususario tratar de evitar esos errores](#)

Escribimos el siguiente código

```
package com.ejemplo1.Ejemplo1.models;
import jakarta.persistence.*;
// podemos colocar import jakarta.persistence.*; en lugar de importar
cada libreria que necesitamos

@Entity //para decir que es un modelo real o una entidad
@Table(name="usuario") //redefinimos el nombre de la tabla
public class UsuarioModel { //si no colocamos @Table el nombre seria
UsuarioModel
    //definimos las columnas de nuestra tabla
    @Id // decimos que la variable que sigue es una ID
    // que genero los valores la estrategia sera generar automaticamente
tipo identidad
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    // el valor generado en esta columna sera unico (verdadero) y nulo
(falso) es decir
    // no puede estar vacio
    @Column(unique = true, nullable = false)
    private Long id;
    private String nombre;
    private String email;
    private Integer prioridad;
    //realizamos los getter y setter de cada uno
    //get de id
    public Long getId() {
        return id;
    }
    //set de id
```



```
public void setId(Long id) {
    this.id = id;
}
//get de nombre
public String getNombre() {
    return nombre;
}
//set de nombre
public void setNombre(String nombre) {
    this.nombre = nombre;
}
//get de email
public String getEmail() {
    return email;
}
//set de email
public void setEmail(String email) {
    this.email = email;
}
//get de prioridad
public Integer getPrioridad() {
    return prioridad;
}
//set de prioridad
public void setPrioridad(Integer prioridad) {
    this.prioridad = prioridad;
}
}
```

8.3) Revisar conexión de la base de datos

Guardamos y nos vamos a la terminal, ya sea cmd o del mismo editor, escribimos mvn.cmd spring-boot:run

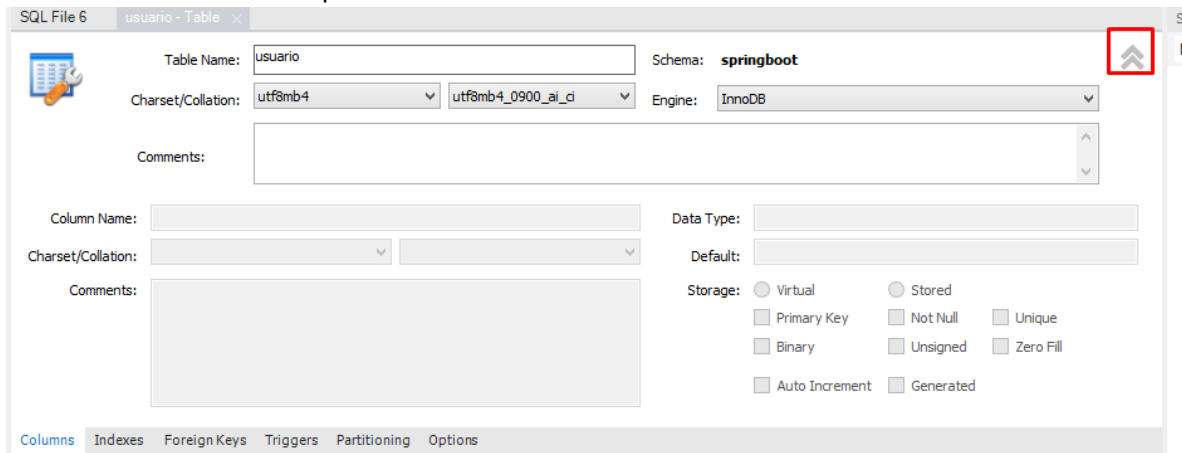
```
Copyright (C) 2014 Microsoft Corporation. Todos los derechos reservados.

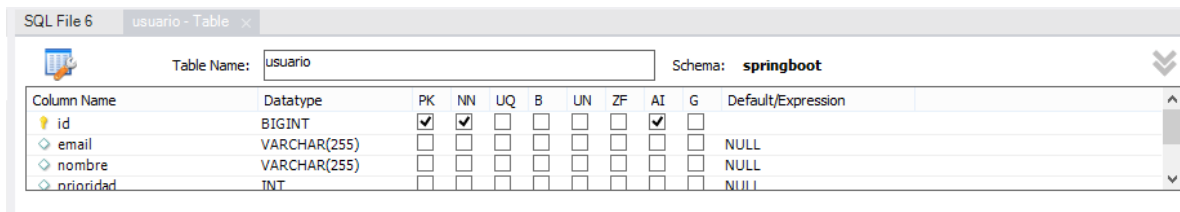
PS C:\Users\Luis Bravo\Desktop\Ejemplo1> mvn.cmd spring-boot:run
[INFO] Scanning for projects...
[INFO]
[INFO] -----< com.ejemplo1:Ejemplo1 >-----
[INFO] Building Ejemplo1 0.0.1-SNAPSHOT
[INFO] -----[ war ]-----
[INFO]
[INFO] >>> spring-boot-maven-plugin:3.0.0:run (default-cli) > test-compile @ Ejemplo1 >>>
[INFO]
[INFO] --- maven-resources-plugin:3.3.0:resources (default-resources) @ Ejemplo1 ---
[INFO] Copying 1 resource
```

Ahora nos vamos MySQL y revisamos nuestra base de datos, podemos observar que se creó la tabla con sus respectivas columnas, damos clic para abrir las propiedades de la tabla



Le damos clic a las flechas para revisar

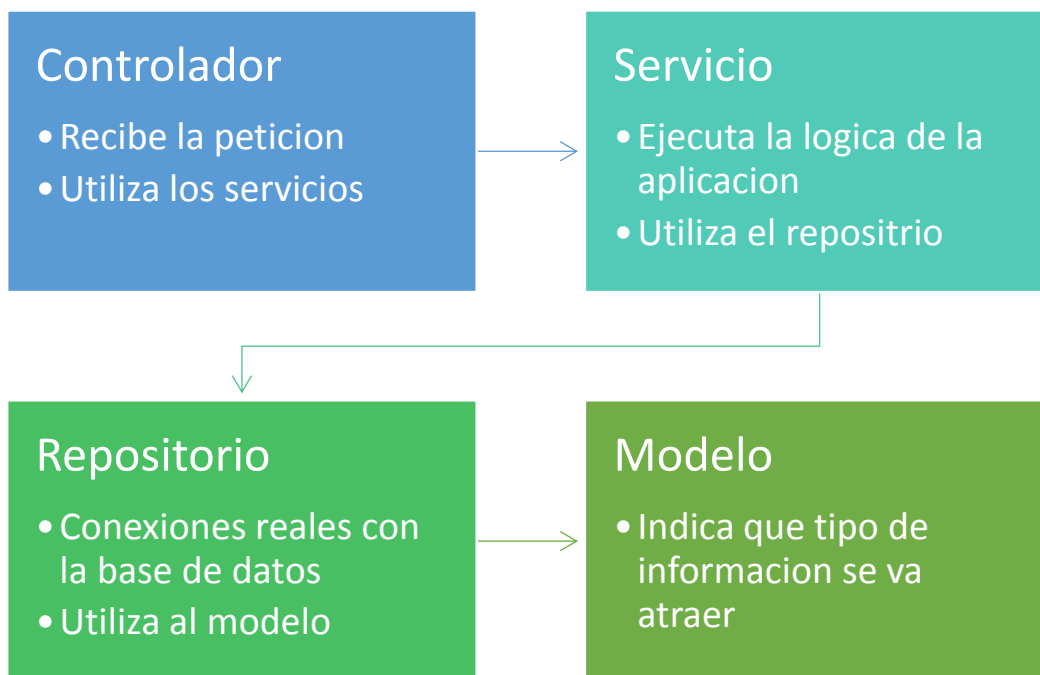




Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
id	BIGINT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
email	VARCHAR(255)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
nombre	VARCHAR(255)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
prioridad	INT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL

Podemos observar que en efecto el tipo de dato de cada columna y que el Id es la llave principal (PK), no puede ser nula (NN) y es auto incrementable (AI) tal como lo definimos en el código en la sección de [Crear tabla desde el editor](#)

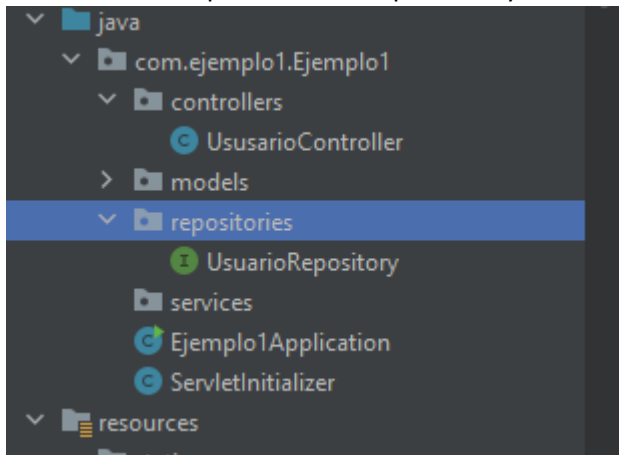
8.4) Flujo de trabajo



Hasta ahora ya tenemos el modelo crearemos las clases restantes

8.5) Repositorio

Creamos una carpeta llamada repositories y una interfaz UsuarioRepository



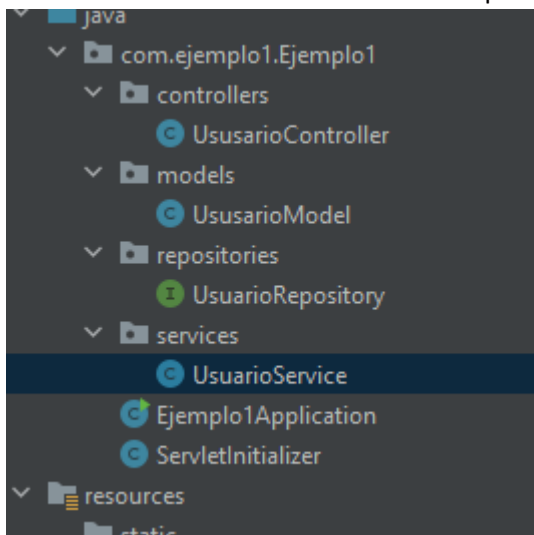
Codigo

```
package com.ejemplo1.Ejemplo1.repositories;
import com.ejemplo1.Ejemplo1.models.UsusarioModel;
import org.springframework.data.repository.CrudRepository;
import org.springframework.stereotype.Repository;

@Repository //colocamos repository para indicar la funcion de la interfaz
public interface UsuarioRepository extends CrudRepository<UsusarioModel,
Long> {
    // extends CrudRepository de donde heredaremos las funciones
    // entre <> <nombreClaseModelo, tipoData>
    //aclaro otra vez escribi mal usuario en su lugar puse Ususario
}
```

8.6) Servicios

Creamos una clase UsuarioService en el package de servicios



Codigo

```
package com.ejemplo1.Ejemplo1.services;
import com.ejemplo1.Ejemplo1.models.UsusarioModel;
import com.ejemplo1.Ejemplo1.repositories.UsuarioRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.ArrayList;

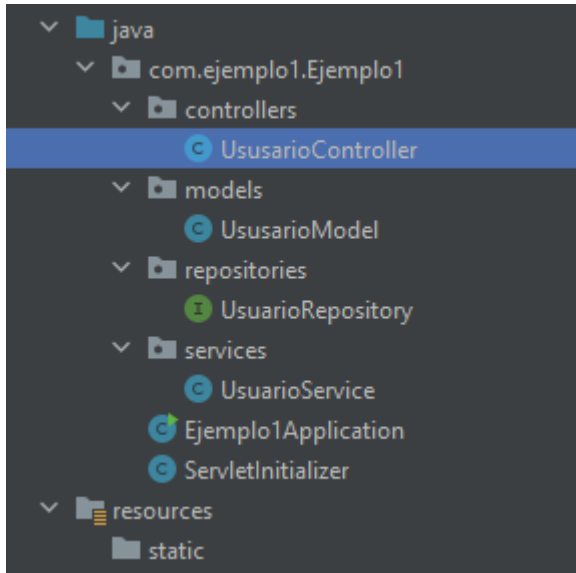
@Service // colocamos service para indicar que la clase es servicio
public class UsuarioService {
    @Autowired //con esto ya no tenemos que instanciar spring lo hace de
    manera automatica
    // marca error hasta que definamos
    UsuarioRepository usuarioRepository; //definimos una variable tipo
    UsuarioRepository
    //para este caso la variable se llama usuarioRepository

    //definimos un metodo tipo ArrayList por lo que tendra que regresar un
    valor igual
    public ArrayList <UsusarioModel> obtenerUsuarios(){ // el metodo se
    llama obtenerUsusario
        return (ArrayList<UsusarioModel>) usuarioRepository.findAll();
        //return tipoDeDato VariableTipoRepositoryio .accionAEjecutar();
        //find all es para encontrar todos los registros podemos elegir la
        opcion que necesitemos
    }

    //metodo para guardar usuarios en este caso sera tipo UsusarioModel
    //nombre del metodo guardar usuauri en parentesis definimos una variable
    tipo UsusarioModel
    public UsusarioModel guardarUsuario(UsusarioModel usuario){
        return usuarioRepository.save(usuario);
    }
    //return variableTipoRepositoryio .save(variableDefinidaParaElMetodo);
}
}
```

8.7) Controlador

Creemos una clase UsuarioController en el package indicado



Codigo

```
package com.ejemplo1.Ejemplo1.controllers;
import com.ejemplo1.Ejemplo1.models.UsuarioModel;
import com.ejemplo1.Ejemplo1.services.UsuarioService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;
import java.util.ArrayList;

@RestController //indicar a aspring que va a controlar
@RequestMapping("/usuario") //en que direccion del servidor se activira
la clase, en este caso
// el nombre de la tabla que renombramos en UsusuarioModel como usuario
public class UsuarioController {
    @Autowired//con esto ya no tenemos que instanciar spring lo hace de
manera automatica
    // marca error hasta que definamos la variable
    UsuarioService usuarioService; //definimos variable tipo
UsuarioService

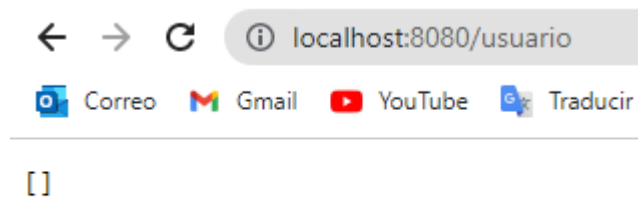
    @GetMapping() //cuando llegue una peticion ejecute este metodo
    //ahora declaramos nuestro metodo tipó ArrayList de la clase
UsusuarioModel nombre del metodo
    public ArrayList<UsuarioModel> obtenerUsuarios(){
        return usuarioService.obtenerUsuarios();
        //return variableUsuarioService .metodoObtenerDeLaClaseService
    }

    @PostMapping() //para mostrar los valores que ya estan guarados
en la base de datos
    // public de la clase UsusuarioModel nombre del metodo
    public UsuarioModel guardarUsuario(@RequestBody UsuarioModel
usuario){
```

```
//RequestBody para indicar que todos los clientes pueden mandar dentro  
del cuerpo solicitud http  
    return this.usuarioService.guardarUsuario(usuario);  
//return this.variableUsuarioService.metodoGuardarDeLaClaseService  
    }  
}
```

8.8) Revisar funcionamiento de nuestro servidor

Guardamos y nos vamos a la terminal, ya sea cmd o del mismo editor, escribimos `mvn.cmd spring-boot:run`, a continuación en nuestro navegador nos dirigimos a <http://localhost:8080/usuario> y nos saldrá lo siguiente



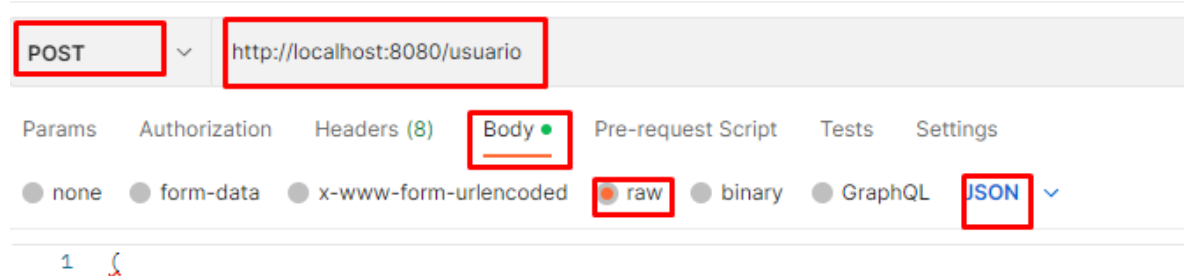
Un arreglo vacío, ya que no tenemos usuario

8.9) Dar de alta usuarios

Ya que necesitamos dar de alta los usuarios para ello necesitamos hacerlo por medio de código o de algún programa que nos ayude como Postman, lo descargamos y lo instalamos:

<https://www.postman.com/downloads/>

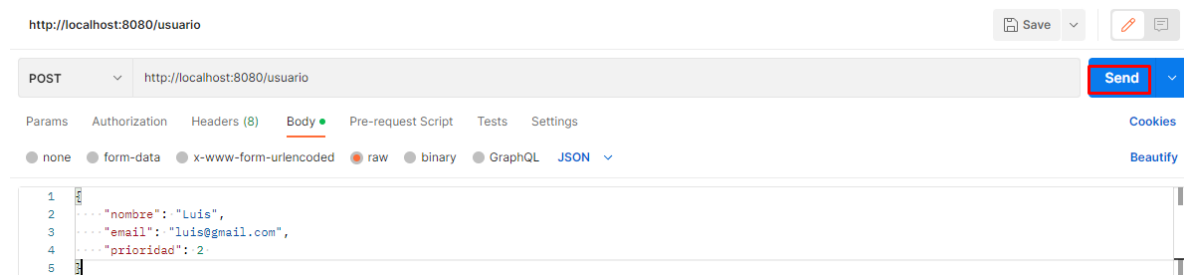
Estando en Postman nos aseguramos de tener la opción POST, la ruta <http://localhost:8080/usuario>, en la pestaña Body, la opción raw y tipo JSON



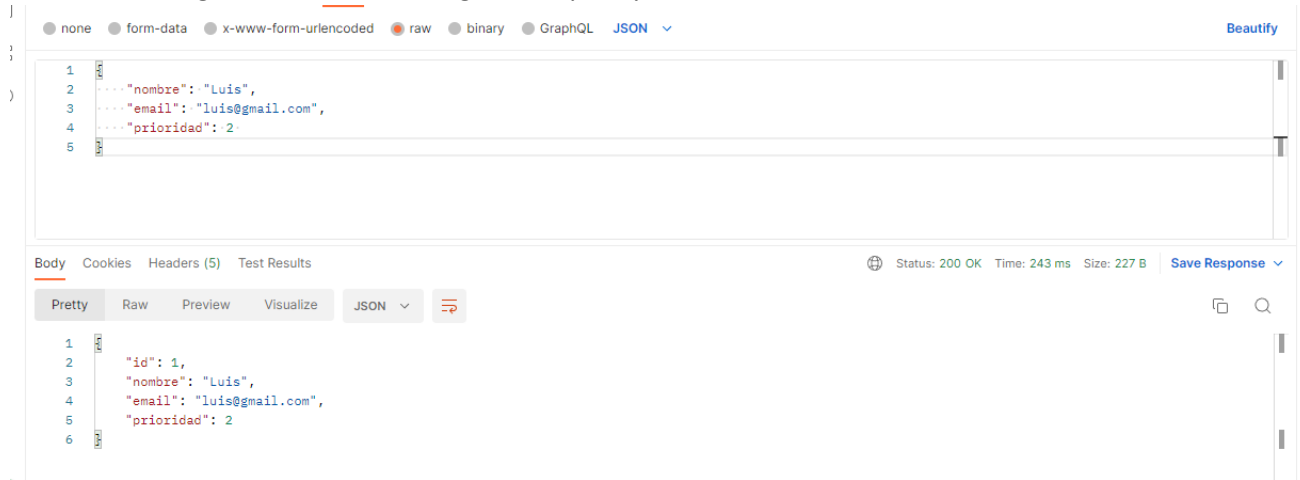
Escribimos las siguientes líneas de código:

```
{  
  "nombre": "Luis",  
  "email": "luis@gmail.com",  
  "prioridad": 2  
}
```

Y damos enviar



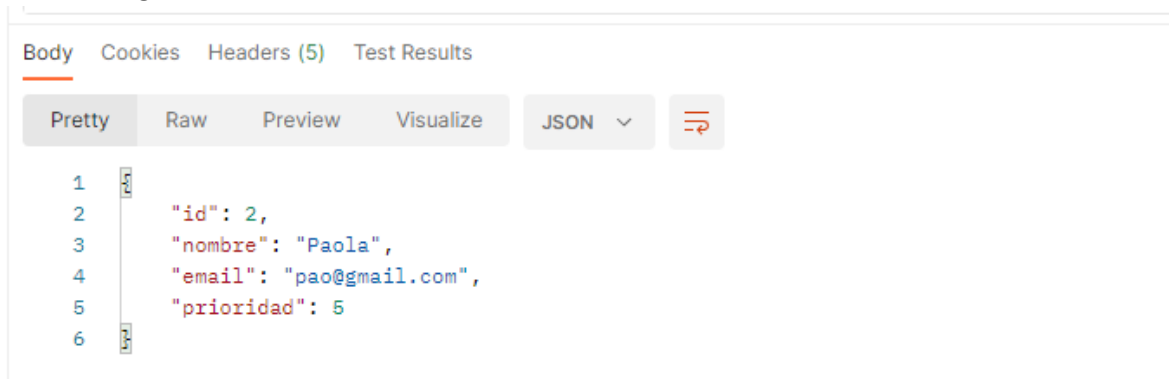
Postman nos regresara como se ha registrado, pero ya con un id en este caso 1



Demos de alta otro usuario

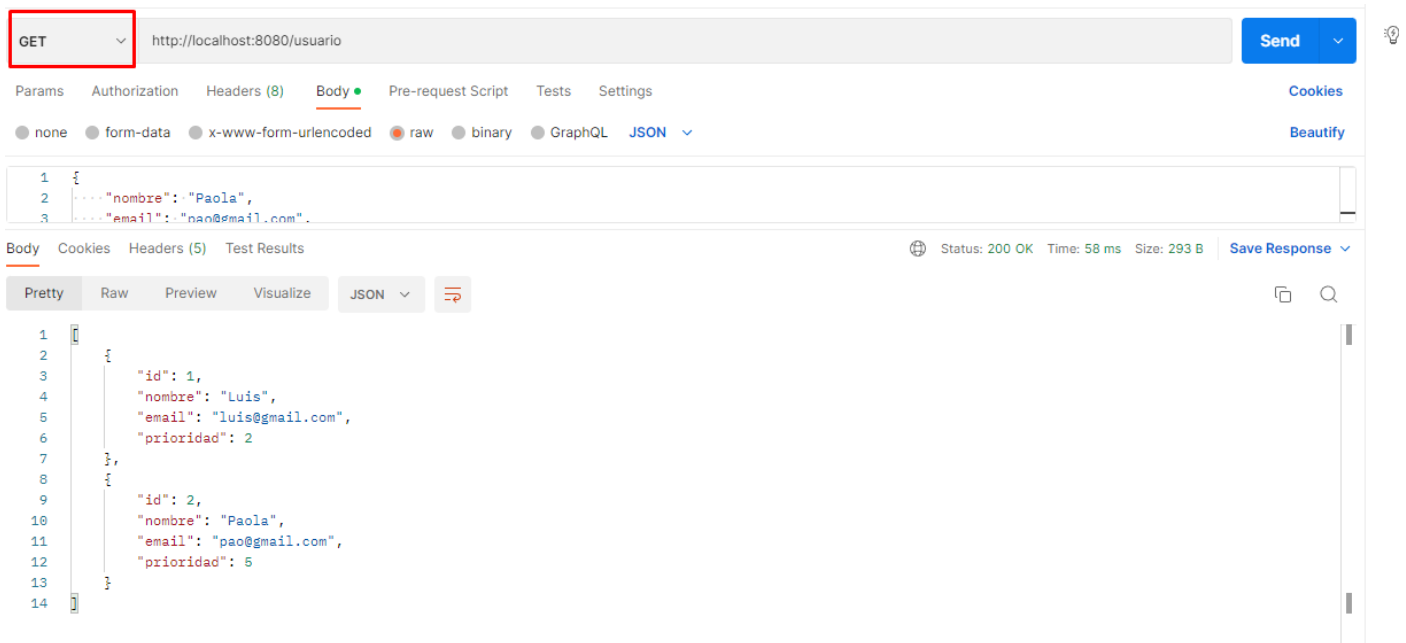
```
{
  "nombre": "Paola",
  "email": "pao@gmail.com",
  "prioridad": 5
}
```

Nos da el registro ahora con Id 2



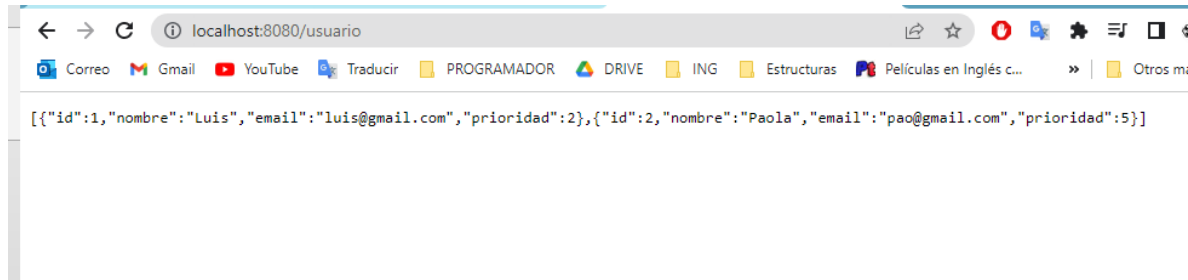
8.10) Comprobar registros desde postman

Cambiamos la acción por GET y damos enviar, en automático nos saldrá los registros ya realizados



8.11) Comprobar registros desde el navegador

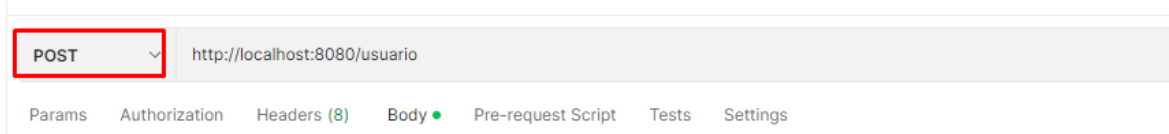
Simplemente basta con dar recargar a la dirección <http://localhost:8080/usuario>



Podemos ver que nos salen los 2 registros ya realizados

8.12) Actualizar registro desde postman

Si recordamos en el método [Servicios](#) con ayuda de la interfaz [Repositorio](#) creamos un método sabe, que en este caso llamamos guardarUsuario, entonces nos basta con ir a postman y vamos a actualizar el segundo registro, nos aseguramos de tener la acción POST



Y escribimos el código

```
{
    "id": 2,
    "nombre": "Paola Iridian",
    "email": "pao@gmail.com",
    "prioridad": 6
}
```

Podemos observar que en esta ocasión colocamos el id, con ello indicamos que estamos editando o actualizando el registro número 2, presionamos SEND

```
1  {
2    "id": 2,
3    "nombre": "Paola Iridian",
4    "email": "pao@gmail.com",
5    "prioridad": 6
6  }
```

Nos regresa el registro actualizado

8.13) Consultar o buscar, y borrar por parámetro (lógica)

8.13.1) Repositorio

Vamos a la interfaz que creamos en [Repositorio](#) y agregamos la siguiente línea de código

```
public abstract ArrayList<UsusarioModel> findByPrioridad(Integer prioridad);
```

Así nos tendría que quedar

```
package com.ejemplo1.Ejemplo1.repositories;
import com.ejemplo1.Ejemplo1.models.UsusarioModel;
import org.springframework.data.repository.CrudRepository;
import org.springframework.stereotype.Repository;

import java.util.ArrayList;

@Repository //colocamos repository para indicar la funcion de la interfaz
public interface UsuarioRepository extends CrudRepository<UsusarioModel, Long> {
    // extends CrudRepository de donde heredaremos las funciones
    // entre <> <nombreClaseModelo, tipoDato>
    //aclaro otra vez escribi mal usuario en su lugar puse Ususario

    //declaramos un public abstract ArratList<nombreClaseModelo>
    nombreMetodo (TipoDeDato nombreColumna)
    public abstract ArrayList<UsusarioModel> findByPrioridad(Integer
```

```
prioridad);  
    //mientras declaremos que es un metodo abstracto spring boot se  
    encarga del resto  
    //tambien podriamos hacer de findByIdNombre o findByIdEmail, etc  
}
```

8.13.2) Servicios

Ahora debemos indicar el método en la clase de [Servicios](#), escribimos el siguiente código

```
public Optional<UsusarioModel> obtenerPorId(Long id){  
    return usuarioRepository.findById(id);  
}  
  
public ArrayList<UsusarioModel> obtenerPorPrioridad(Integer prioridad){  
    return usuarioRepository.findByPrioridad(prioridad);  
}  
  
public boolean eliminarUsuario(Long id){  
    try{  
        usuarioRepository.deleteById(id);  
        return true;  
    }catch(Exception err){  
        return false;  
    }  
}
```

Así nos quedaría al final la clase servicios

```
package com.ejemplo1.Ejemplo1.services;  
import com.ejemplo1.Ejemplo1.models.UsusarioModel;  
import com.ejemplo1.Ejemplo1.repositories.UsuarioRepository;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
  
import java.util.ArrayList;  
import java.util.Optional;  
  
@Service // colocamos service para indicar que la clase es servicio  
public class UsuarioService {  
    @Autowired //con esto ya no tenemos que instanciar spring lo hace de  
    manera automatica  
    // marca error hasta que definamos la variable  
    UsuarioRepository usuarioRepository; //definimos una variable tipo  
    UsuarioRepository  
    //para este caso la variable se llama usuarioRepository  
  
    //definimos un metodo tipo ArrayList por lo que tendra que regresar un  
    valor igual  
    public ArrayList <UsusarioModel> obtenerUsuarios(){ // el metodo se  
    llama obtenerUsusario  
        return (ArrayList<UsusarioModel>) usuarioRepository.findAll();  
        //return tipoDeDato VariableTipoRepositorio .accionAEjecutar();  
        //find all es para encontrar todos los registros podemos elegir la
```

```
opcion que necesitamos
    }

    //metodo para guardar usuarios en este caso sera tipo UsusarioModel
    //nombre del metodo guardar usuauri en parentesis definimos una variable
    tipo UsusarioModel
    public UsusarioModel guardarUsuario(UsusarioModel usuario){
        return usuarioRepository.save(usuario);
    }
    //return variableTipoRepository .save(variableDefinidaParaElMetodo);
    }

    // este metdo es el mas comun , por lo que lo dejamos opcional
    // al tratarse de un metodo comun no tuvimos que definirlos en la
    interfaz de repositorio por que ya
    // lo trabaja spring
    //public Optional<claseUsuarioModel> nombre metodo (tipoDato
    nombreColumna)
    public Optional<UsusarioModel> obtenerPorId(Long id){
        return usuarioRepository.findById(id);
    }
    //return variableTipoRepository . findById(nombreColumna)
    }

    //ahora el metodo que definimos en la interfaz
    //public tipoDeDato<claseUsuarioModel> nombre metodo (tipoDato
    nombreColumna)
    public ArrayList<UsusarioModel> obtenerPorPrioridad(Integer
    prioridad){
        return usuarioRepository.findByPrioridad(prioridad);
    }
    //return variableTipoRepository . findById(prioridad(nombreColumna)
    }

    //declaramos metodo para borrar tipó boolean
    // deletedById al tratarse de un metodo comun no tuvimos que
    definirlos en la interfaz
    // de repositorio por que ya lo trabaja spring
    // public boolean nombremetodo (tipoDato nombreColumna)
    public boolean eliminarUsuario(Long id){
        //colocamos un try y catch ya que puede causar un error si falla
        try{
            // si sucede borra el usuario por la id
            usuarioRepository.deleteById(id);
            //variableTipoRepository .deletById
            return true;
        }catch(Exception err){
            return false;
        }
        //si se borra manda verdadero, si no existe o no ya se borro
        manda false
    }
}
```

8.13.3) Controlador

Vamos a la clase controlador y pegamos las siguientes líneas

```
@GetMapping (path="/{id}")
public Optional<UsusarioModel> obtenerUsusrioPorId (@PathVariable ("id")
Long id){
    return this.usuarioService.obtenerPorId(id);
}
@GetMapping (path="/query")
public ArrayList<UsusarioModel> obtenerUsusrioPorPrioridad (@RequestParam
("prioridad") Integer prioridad){
    return this.usuarioService.obtenerPorPrioridad(prioridad);
}

@DeleteMapping (path="/{id}")
public String eliminarPorId (@PathVariable("id") Long id){
    boolean ok = this.usuarioService.eliminarUsuario(id);
    if(ok){
        return "Se elimino el usuario con id "+ id;
    }else{
        return "No pudo eliminar el usuario con id"+ id;
    }
}
```

Así nos quedaría al final la clase controlador

```
package com.ejemplo1.Ejemplo1.controllers;
import com.ejemplo1.Ejemplo1.models.UsusarioModel;
import com.ejemplo1.Ejemplo1.services.UsuarioService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;
import java.util.ArrayList;
import java.util.Optional;

@RestController //indicar a aspring que va a controlar
@RequestMapping("/usuario") //en que direccion del servidor se activira
la clase, en este caso
// el nombre de la tabla que renombramos en UsusarioModel como usuario
public class UsusarioController {
    @Autowired//con esto ya no tenemos que instanciar spring lo hace de
manera automatica
    // marca error hasta que definamos la variable
    UsuarioService usuarioService; //definimos variable tipo
    UsuarioService

    @GetMapping() //cuando llegue una peticion ejecute este metodo
    //ahora declaramos nuestro metodo tipó ArrayList de la clase
    UsusarioModel nombre del metodo
    public ArrayList<UsusarioModel> obtenerUsuarios(){
        return usuarioService.obtenerUsuarios();
    }
    //return variableUsuarioService .metodoObtenerDeLaClaseService

    @PostMapping() //para mostrar los valores que ya estan guarados
en la base de datos
    // public de la clase UsusarioModel nombre del metodo
```

```
    public UsuarioModel guardarUsuario(@RequestBody UsuarioModel
usuario){
    //RequestBody para indicar que todos los clientes pueden mandar dentro
del cuerpo solicitud http
    return this.usuarioService.guardarUsuario(usuario);
    //return this.variableUsuarioService.metodoGuardarDeLaClaseService
    }

@GetMapping (path="/{id}") // para que busque pero tomara el camino por
la id
// public Optional<clase UsuarioModel> nombrMetodo (@PathVariable
("nombreColumna") tipoDato nombreColumna){
public Optional<UsuarioModel> obtenerUsuarioPorId (@PathVariable ("id")
Long id){
    return this.usuarioService.obtenerPorId(id);
    //return this.ClaseServicio.nombremetodo(nombrecolumna)
}

@GetMapping (path="/query") // para que busque pero tomara el camino otro
servicio
//public ArrayList <clase UsuarioModel> nombrMetodo (@RequestParam
("nombreColumna") tipoDato nombreColumna){
public ArrayList<UsuarioModel> obtenerUsuarioPorPrioridad (@RequestParam
("prioridad") Integer prioridad){
    return this.usuarioService.obtenerPorPrioridad(prioridad);
    //return this.ClaseServicio.nombremetodo(nombrecolumna)
}
//en este caso @RequestParam indica que el camino que buscara sera por
prioridad en este caso

@DeleteMapping (path="/{id}") //para borrar por id
//public String nombreMetodo (@PathVariable ("nombreColumna") tipoDato
nombreColumna){
public String eliminarPorId (@PathVariable("id") Long id){
// declaramos boolean nombreVariable =
this.ClaseServicio.nombremetodo(nombrecolumna)
boolean ok = this.usuarioService.eliminarUsuario(id);
//condiciones si es
if(ok){ //true
    return "Se elimino el usuario con id "+ id;
}else{ //false
    return "No pudo eliminar el usuario con id"+ id;
}
}
}
}
```

8.14) Consultar o buscar, y borrar por parámetro (pruebas)

Damos de alta otros 2 registros para el ejemplo

```
{  
  "nombre": "Omar",  
  "email": "Omar@gmail.com",  
  "prioridad": 2  
}  
  
{  
  "nombre": "Omar",  
  "email": "Omar@gmail.com",  
  "prioridad": 2  
}
```

Así Juan y Omar tendrán 3 y 4 de id respectivamente, revisamos el local host

```
[{"id":1,"nombre":"Luis","email":"luis@gmail.com","prioridad":2}, {"id":2,"nombre":"Paola Iridian","email":"pao@gmail.com","prioridad":6}, {"id":3,"nombre":"Juan","email":"Juan@gmail.com","prioridad":3}, {"id":4,"nombre":"Omar","email":"Omar@gmail.com","prioridad":2}]
```

Y aparecen los 4 registros

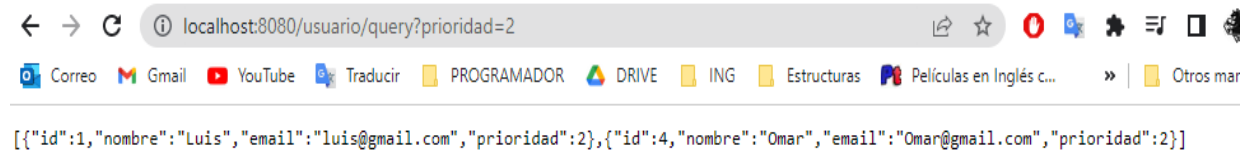
8.14.1) Buscar por id

Nos vamos al local host y escribimos después de usuario /numero, por ejemplo para mostrar el registro con id 3

```
{"id":3,"nombre":"Juan","email":"Juan@gmail.com","prioridad":3}
```

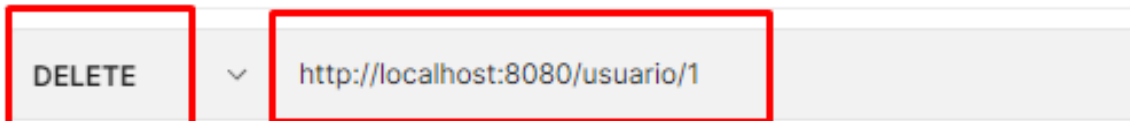

8.14.2) Buscar por campo prioridad

Ahora para buscar por prioridad después de usuario ponemos /query?columna=valor, en este caso buscaremos a los usuarios de prioridad

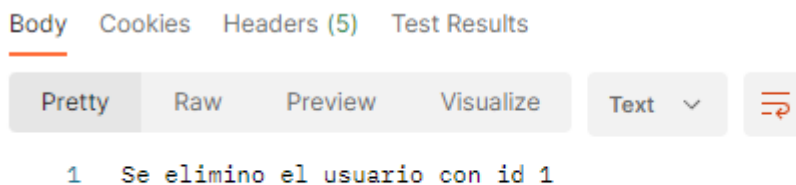


8.14.3) Eliminar registro

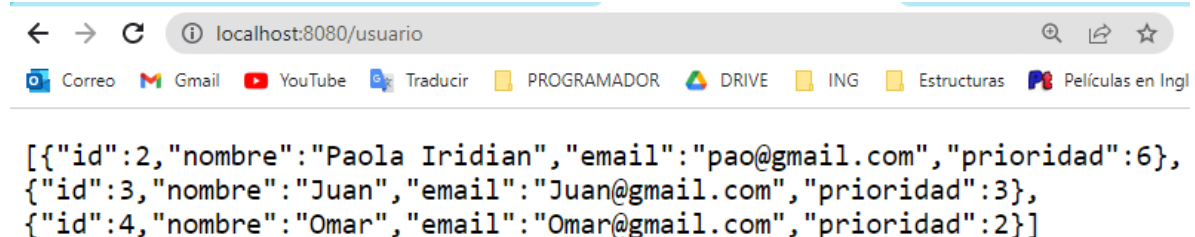
Para eliminar nos vamos a postman, indicamos que se eliminara el usuario id 1, es decir en la abrra colocaremos /1 después de usuario, nos aseguramos que la opción este en DELETE



Nos saldrá el siguiente mensaje



Revisamos en el navegador



Y en efecto el usuario 1 se elimino