

	<p align="center"><b>MINTIC 2022</b>  <b>Desarrollo de Aplicaciones Web</b>  <b>Proyecto Back con SPRING BOOT</b></p>	
---	---	---

## **Proyecto BackEnd con SPRING BOOT.**

En este documento explicaremos la forma de construir el backend resolviendo la parte del login de acceso al aplicativo, se debe crear el proyecto con las dependencias añadir las.

### 1. Acceso de Usuario (FrontEnd).

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 nuestra parte front se debe construir el formulario que captura el ingreso al sistema y el menú de navegación del mismo.

### **Requerimientos**

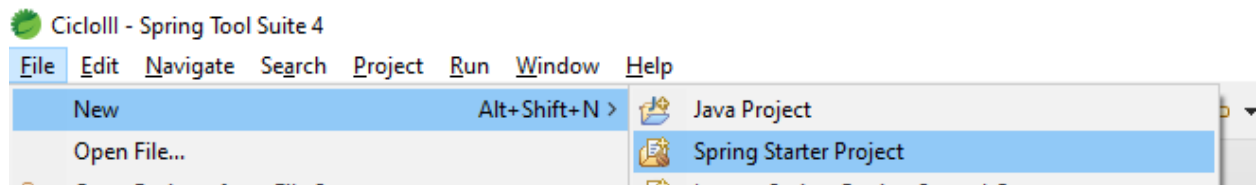
1. Maven 3.0+
2. IDE(STS o Eclipse o Netbeans o Visual Estudio Code)
3. JDK 1.8+

### **Contenido**

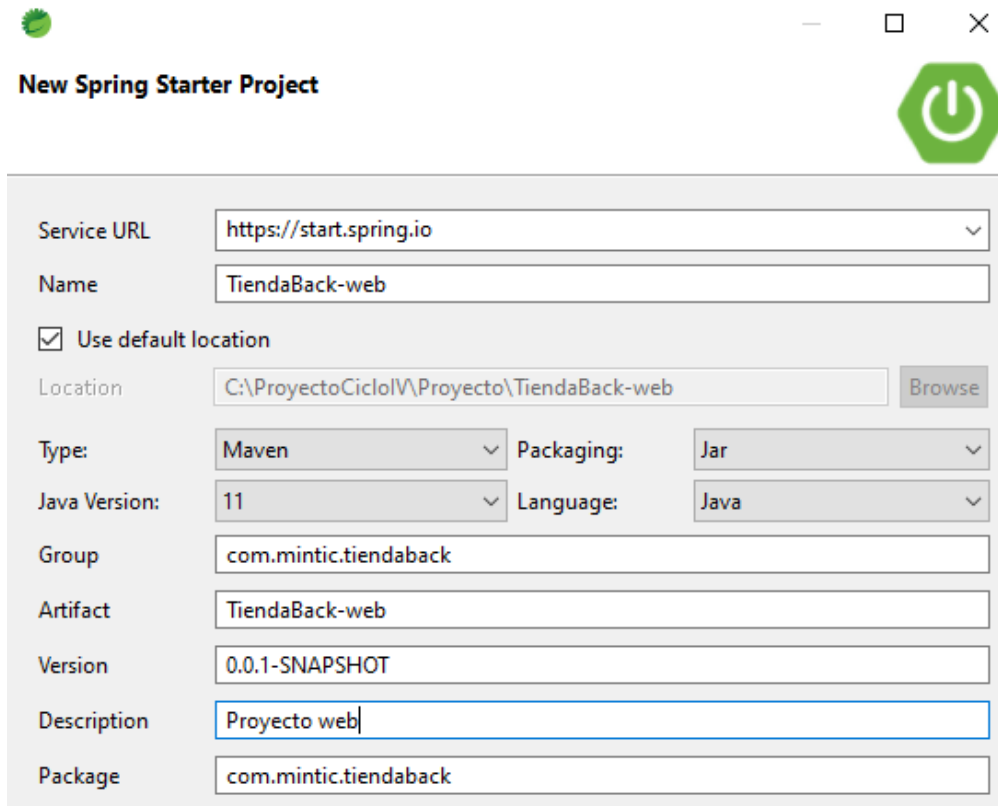
1. Cree el proyecto Spring Boot.
2. Revisar la base de datos MYSQL y defina sus configuraciones en el proyecto.
3. Creación de la estructura del proyecto
4. Crear clase Dto de formulario
5. Crear las clases de modelo de entidad
6. Crear DAO repositorios
7. Crear Servicios.
8. Creación del enlace cliente entre el FronEnd y el Backend
9. Compile y ejecute el proyecto

### **1. Crear el proyecto Spring Boot.**

Desde STS 4.0 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 11 entre las características más importantes.



**New Spring Starter Project**

Service URL:

Name:

☒ Use default location

Location:

Type:  Packaging:

Java Version:  Language:

Group:

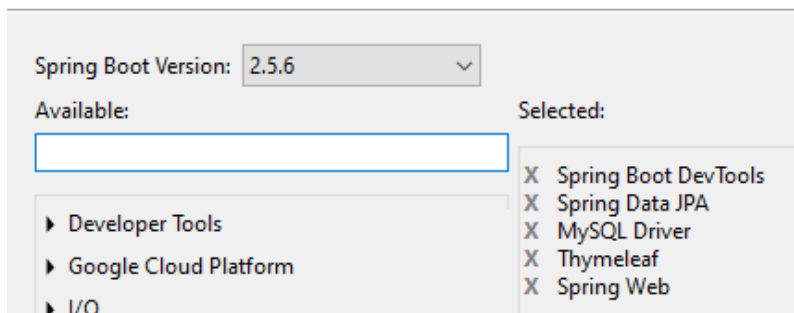
Artifact:

Version:

Description:

Package:

Tenemos que especificar las dependencias que aparecen en la siguiente gráfica:



Spring Boot Version:

Available:

- Developer Tools
- Google Cloud Platform
- I/O

Selected:

- X Spring Boot DevTools
- X Spring Data JPA
- X MySQL Driver
- X Thymeleaf
- X Spring Web

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

Por ser el proyecto Backend debe conectar a la base de datos, entonces agregamos la siguiente información al archivo properties.

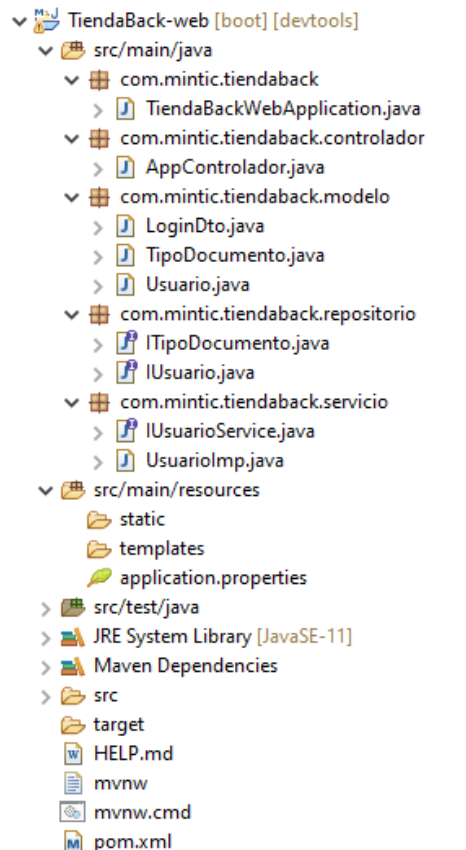
```

1 spring.datasource.url=jdbc:mysql://localhost:3306/tiendagenerica?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 = 8090
11 server.servlet.contextPath=/tiendagenerica/v1

```

## 3. Creación de la estructura del proyecto.

En el proyecto para facilitar su ordenamiento creamos una estructura de packages como el que se muestra en la figura:



## 4. Crear clase Dto de formulario

Para poder acceder a los atributos de los datos enviados desde el front debemos crear una clase con las mismas características, a continuación mostramos su contenido:

```
1 package com.mintic.tiendaback.modelo;
2
3 public class LoginDto {
4
5     private String nombreUsuario;
6
7     private String password;
8
9     public String getNombreUsuario() {
10         return nombreUsuario;
11     }
12
13     public void setNombreUsuario(String nombreUsuario) {
14         this.nombreUsuario = nombreUsuario;
15     }
16
17     public String getPassword() {
18         return password;
19     }
20
21     public void setPassword(String password) {
22         this.password = password;
23     }
24
25
26
27 }
```

## 5. Crear las clases de modelo de entidad.

En el backend vamos a conectar a la Base de Datos y hacer la consulta de nombre de usuario y clave, por ello debemos definir las entidades correspondientes mapeadas a través de JPA.

La Tabla usuario tiene relación con la tabla tipodocumento por ello se deben crear las dos clases como se muestra en las siguientes gráficas, la primera refiere a la tabla tipodocumento:

```
1 package com.mintic.tiendaback.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     public static final String TABLE_NAME = "tipodocumento";
14
15     @Id
16     @GeneratedValue(strategy = GenerationType.IDENTITY)
17     private Long id;
18
19     @Column(name = "tipo")
20     private String tipo;
21
22     public TipoDocumento() {
23     }
24
25     public TipoDocumento(Long id, String tipo) {
26
27         this.id = id;
28         this.tipo = tipo;
29     }
30
31     public Long getId() {
32         return id;
33     }
34
35     public void setId(Long id) {
36         this.id = id;
37     }
38
39     public String getTipo() {
40         return tipo;
41     }
42
43     public void setTipo(String tipo) {
44         this.tipo = tipo;
45     }
46
47 }
48 }
```

Clase Usuario:

```
1 package com.mintic.tiendaback.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
13 /*
14  * Se mapea la bd las tablas de la base de datos son las entidades
15  * la entidad se utiliza en los repositorios
16  */
17 @Entity
18 @Table(name = Usuario.TABLE_NAME)
19 public class Usuario {
20     public static final String TABLE_NAME = "usuario";
21
22
23     /*
24      * @id para identificar la llave primaria
25      * @GeneratedValue(strategy = GenerationType.IDENTITY) se define el autoincremental
26      */
27     @Id
28     @GeneratedValue(strategy = GenerationType.IDENTITY)
29     private Long id;
30
31
32     /*@ManyToOne hace referencia la relacion muchos a uno en este caso muchos usuario
33      * tienen un tipo de documento
34      * @JoinColumn el campo que hace de referencia a la llave foranea
35      */
36     @ManyToOne
37     @JoinColumn(name = "idTipoDocumento")
38     private TipoDocumento idTipoDocumento;
39
40     /*@Column nombre de la columna , si el nombre en la base de datos del campo es igual
41      * a el de la variable no es necesario poner la anotacion
42      */
43     @Column(name = "numeroDocumento")
44     private String numeroDocumento;
45
46     @Column(name = "nombre")
47     private String nombre;
48
49     @Column(name = "password")
50     private String password;
51
52     @Column(name = "nombreUsuario")
53     private String nombreUsuario;
54
55     @Column(name = "email")
56     private String email;
57
58     public Usuario() {
59
60     }
```

Agregar el constructor pendiente, los setter y getter.

## 6. Crear DAO repositorios.

Agregar las interfaces que permitan acceder a los métodos del crud con la implementación se deben crear para las dos entidades.

En la siguiente figura se aprecia el contenido de ITipoDocumento:

```
1 package com.mintic.tiendaback.repositorio;
2
3 import org.springframework.data.repository.CrudRepository;
4
5 import com.mintic.tiendaback.modelo.TipoDocumento;
6
7
8 public interface ITipoDocumento extends CrudRepository<TipoDocumento, Long> {
9
10 }
11
```

En la siguiente figura se aprecia el contenido de IUserario:

```
1 package com.mintic.tiendaback.repositorio;
2
3 import org.springframework.data.jpa.repository.Query;
4 import org.springframework.data.repository.CrudRepository;
5 import org.springframework.data.repository.query.Param;
6
7 import com.mintic.tiendaback.modelo.Usuario;
8
9
10 /*
11  * Aqui se realizan las operaciones crud
12  * los parametros son la entidad y el tipo de datos que se definio como @id en la entidad
13  *
14  * el id es long debe ir igual en el crud repository <Usuario, Long>
15  *
16  *
17  */
18 public interface IUserario extends CrudRepository<Usuario, Long> {
19
20     // JPQL: Se hace la consulta sobre la clase
21     @Query("select count(*) from Usuario as p where p.nombreUsuario= :nombreUsuario and p.password=:password")
22     Integer findByNameAndPassword(@Param("nombreUsuario") String nombreUsuario,
23     @Param("password") String password);
24
25
26     @Query("select p from Usuario as p where p.nombreUsuario= :nombreUsuario and p.password=:password")
27     Usuario findByNameAndPassword(@Param("nombreUsuario") String nombreUsuario,
28     @Param("password") String password);
29 }
```

Se debe crear un método apropiado que valide la coincidencia de nombre de usuario y clave, se utiliza JPQL para ello, es hacer consultas sobre Clases o Entity que representan las tablas, en el primer método se calcula el número de objetos que coinciden con nombre de usuario y clave, en el segundo se retorna un objeto tipo Usuario.

## 7. Crear Servicios.

Vamos a dividirlo en dos partes, en la primera definimos las cabeceras de los métodos a implementar.

```
1 package com.mintic.tiendaback.servicio;
2
3 import org.springframework.http.ResponseEntity;
4
5 import com.mintic.tiendaback.modelo.LoginDto;
6
7
8 /*
9  * Aqui se definen los metodos que se van a utilizar (el contrato)
10  * */
11 public interface IUserService {
12
13     int login(LoginDto usuarioDto);
14
15     ResponseEntity<?> ingresar(LoginDto usuarioDto);
16
17 }
18
```

El primero define la validación de las credenciales del usuario y el segundo retorna un objeto para poder ingresar al programa.

En la segunda parte se implementan los métodos en una clase "UsuarioImp".



```
1 package com.mintic.tiendaback.servicio;
2
3 import java.util.HashMap;
4 import java.util.Map;
5 import org.springframework.beans.factory.annotation.Autowired;
6 import org.springframework.http.HttpStatus;
7 import org.springframework.http.ResponseEntity;
8 import org.springframework.stereotype.Service;
9 import com.mintic.tiendaback.modelo.LoginDto;
10 import com.mintic.tiendaback.modelo.Usuario;
11 import com.mintic.tiendaback.repositorio.IUsuario;
12
13 /*
14  * Implementamos la interface con sus metodos
15  * @Service indica la logica empresarial toda la logica de negocio */
16 @Service
17 public class UsuarioImp implements IUsuarioService {
18
19     /*
20     * @Autowired inyeccion de dependencia en este caso para acceder a los metodos del repositorio IUsuario
21     * (accedemos al login)
22     */
23     @Autowired
24     IUsuario iUsuario;
25
26     /*
27     * Metodo para validar si existe el usuario
28     */
29     @Override
30     public int login(LoginDto usuarioDto) {
31         int u = iUsuario.findByNombreUsuarioAndPassword(usuarioDto.getNombreUsuario(), usuarioDto.getPassword());
32         return u;
33     }
34
35     @Override
36     public ResponseEntity<> ingresar(LoginDto usuarioDto) {
37         Map<String, Object> response = new HashMap<>();
38         Usuario usuario = null;
39         try {
40             usuario = iUsuario.findByNameAndPassword(usuarioDto.getNombreUsuario(), usuarioDto.getPassword());
41             if (usuario == null) {
42                 response.put("Usuario", null);
43                 response.put("Mensaje", "Alerta:Usuario o Password incorrectos");
44                 response.put("statusCode", HttpStatus.NOT_FOUND.value());
45                 return new ResponseEntity<>(response, HttpStatus.NOT_FOUND);
46             } else {
47                 response.put("Usuario", usuario);
48                 response.put("Mensaje", "Datos correctos");
49                 response.put("statusCode", HttpStatus.OK.value());
50                 return new ResponseEntity<>(response, HttpStatus.OK);
51             }
52         } catch (Exception e) {
53             response.put("Usuario", null);
54             response.put("Mensaje", "Ha ocurrido un error");
55             response.put("statusCode", HttpStatus.INTERNAL_SERVER_ERROR.value());
56             return new ResponseEntity<>(response, HttpStatus.INTERNAL_SERVER_ERROR);
57         }
58     }
59 }
```

## 8. Creación del enlace cliente entre el FronEnd y el Backend.

En este momento es en donde se hace el enlace entre los dos proyectos, para ello vamos a utilizar "Spring WebClient" que nos permite hacer solicitudes web reactivas y sin bloqueo a otros servicios.

En palabras simples, Spring WebClient es un componente que se utiliza para realizar llamadas HTTP a otros servicios. Es parte del marco reactivo

web de Spring, ayuda a construir aplicaciones reactivas y sin bloqueo.

Además, al ser reactivo, WebClient admite solicitudes web sin bloqueo utilizando todas las funciones de la biblioteca Webflux de Spring. Lo más importante es que también podemos usar WebClient en modo de bloqueo, donde el código esperará a que finalice la solicitud antes de continuar. Al tener soporte para flujos reactivos sin bloqueo, el cliente web puede decidir si esperar a que finalice la solicitud o continuar con otras tareas.

Podemos encontrar documentación oficial en el siguiente link:

<https://docs.spring.io/spring-boot/docs/2.0.x/reference/html/boot-features-webclient.html>

## 1. Ajustes en el FrontEnd.

- Dependencia de WebClient.

Spring WebClient se envía en la biblioteca de Webflux. Para usar WebClient en un proyecto Spring Boot, necesitamos agregar dependencia en la biblioteca WebFlux. Como cualquier otra dependencia de Spring Boot, tenemos que agregar una dependencia de inicio para WebFlux (spring-boot-starter-webflux).

Para proyectos contruidos por Maven, agregue la dependencia de inicio para WebClient en el archivo pom.xml.

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-webflux</artifactId>
</dependency>
```

- Crear una instancia WebClient.

En el front hacemos la solicitud al servicio del backend para ello debemos construir una interface que nos muestre el contrato del método a utilizar y su implementación en una clase respectiva, para efecto de facilitar el entendimiento creamos un package llamado cliente en donde se agregaran los dos archivos.

Interface IClienteTienda

```

1 package com.mintic.tiendafront.cliente;
2
3 import com.mintic.tiendafront.modelo.LoginDto;
4
5 public interface IClientTienda {
6
7     public int login(LoginDto loginDto);
8
9 }
10

```

### Clase ClienteImpl

```

1 package com.mintic.tiendafront.cliente;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.http.MediaType;
5 import org.springframework.stereotype.Service;
6
7 import com.mintic.tiendafront.modelo.LoginDto;
8
9 import reactor.core.publisher.Mono;
10
11 import org.springframework.web.reactive.function.client.WebClient;
12 import org.springframework.web.reactive.function.client.WebClientResponseException;
13
14 @Service
15 public class ClientImp implements IClientTienda{
16
17     // URL del Backend incluye contextPath definido en el servidor
18     private static final String URL = "http://localhost:8090/tiendagenerica/v1";
19
20     @Autowired
21     private WebClient.Builder webClient;
22
23     @Override
24     public int login(LoginDto loginDto) {
25         try {
26             /*
27              * aqui nos conectamos al back directamente al controlador donde estan las rutas
28              * el back espera recibir un dto por eso enviamos el dto login dto
29              */
30
31             Mono<Integer> response = webClient.build().post().uri(URL + "/loginclient")
32                 .accept(MediaType.APPLICATION_JSON).body(Mono.just(loginDto), LoginDto.class).retrieve()
33                 .bodyToMono(Integer.class);
34
35             //Aqui se captura la respuesta del back
36             return response.block();
37
38         } catch (WebClientResponseException e) {
39             e.getMessage();
40             System.out.println("---->" + e.getMessage());
41             return 0;
42         }
43     }
44 }
45
46 }

```

Para utilizar WebClient nos valdremos del builder que nos provee  
 WebClient WebClient.builder()

De este modo vamos a crear una instancia de WebClient con una url base.

Ver documentación adicional en el siguiente link:

<https://gustavopeiretti.com/spring-boot-como-usar-webclient/>

Ajuste del controlador: Se debe crear un método que haga el llamado a la implementación del ClienteTienda, el resultado del controlador ajustado será el siguiente:

```
1 package com.mintic.tiendafront.controlador;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.stereotype.Controller;
5 import org.springframework.ui.Model;
6 import org.springframework.web.bind.annotation.GetMapping;
7 import org.springframework.web.bind.annotation.PostMapping;
8
9 import com.mintic.tiendafront.cliente.IClientTienda;
10 import com.mintic.tiendafront.modelo.LoginDto;
11
12 @Controller
13 public class AppControlador {
14
15     @Autowired
16     IClientTienda clienteTienda;
17
18     @GetMapping("/")
19     public String index() {
20         return "index";
21     }
22
23     @GetMapping("/menu")
24     public String menu() {
25         return "menu";
26     }
27 }
```

```
28 @PostMapping("/login")
29 public String login(Model model, LoginDto loginDto) {
30     int validLogin = clienteTienda.login(loginDto);
31
32     if (validLogin == 1) {
33         return "menu";
34     } else {
35         model.addAttribute("error", "Usuario o contraseña invalidos.");
36         return "index";
37     }
38 }
39 }
40
41 }
```

Vemos que al formulario inicial index.html debemos agregar la ruta url /login que está presente en este controlador, al ejecutar el método login de clienteTienda hacemos la conexión al backend que va a llamar al url llamado "/logincliente", debemos implementar la lógica en el servidor partiendo desde el url creado en su controlador.

## 2. Programación en el BackEnd.

```
1 package com.mintic.tiendaback.controlador;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.http.ResponseEntity;
5 import org.springframework.web.bind.annotation.CrossOrigin;
6 import org.springframework.web.bind.annotation.PostMapping;
7 import org.springframework.web.bind.annotation.RequestBody;
8 import org.springframework.web.bind.annotation.RestController;
9
10 import com.mintic.tiendaback.modelo.LoginDto;
11 import com.mintic.tiendaback.repositorio.ITipoDocumento;
12 import com.mintic.tiendaback.servicio.IUsuarioService;
13
14 @RestController
15 public class AppControlador {
16     /*
17      * inyectamos el la interface del servicio para acceder a los metodos del negocio
18      */
19     @Autowired
20     IUsuarioService iUsuario;
21
22     @Autowired
23     ITipoDocumento iTipoDocumento;
24 }
```

```

25-  /*
26-   * @CrossOrigin indica desde que sitios se van a realizar peticiones
27-   */
28-  @CrossOrigin(origins = { "http://localhost:8010" })
29-  @PostMapping("/loginclient") // ruta del servicio desde el front deben direccionar a esta ruta
30-  public int login(@RequestBody LoginDto usuario) {
31-      int responseLogin = iUsuario.login(usuario);
32-      return responseLogin;
33-  }
34-
35-  @PostMapping("/login") // ruta del servicio desde el front deben direccionar a esta ruta
36-  public ResponseEntity<?> loginCliente(@RequestBody LoginDto usuario) {
37-      return iUsuario.ingresar(usuario);
38-  }
39-  }
40- }

```

Podemos apreciar que la url /loginclient define en primera instancia con la anotación @CrossOrigin.

El intercambio de recursos de origen cruzado (CORS) es un protocolo estándar que define la interacción entre un navegador y un servidor para manejar de forma segura las solicitudes HTTP de origen cruzado.

En pocas palabras, una solicitud HTTP de origen cruzado es una solicitud a un recurso específico, que se encuentra en un origen diferente, es decir, un dominio, protocolo y puerto, que el del cliente que realiza la solicitud.

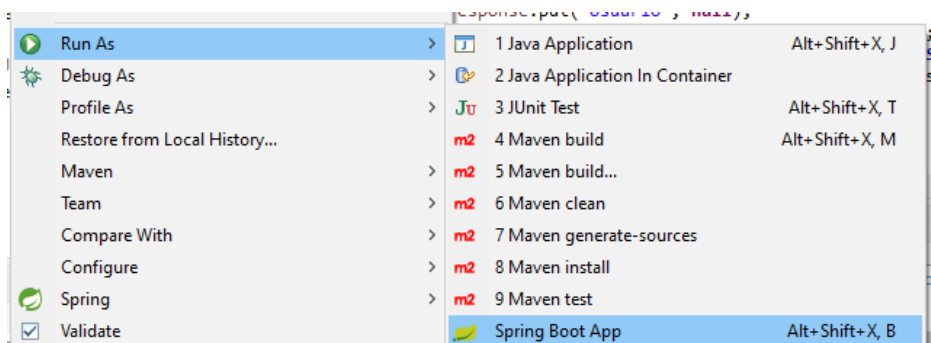
Documentación complementaria en el siguiente link:

<https://ricardogeek.com/spring-boot-y-la-anotacion-crossorigin/>

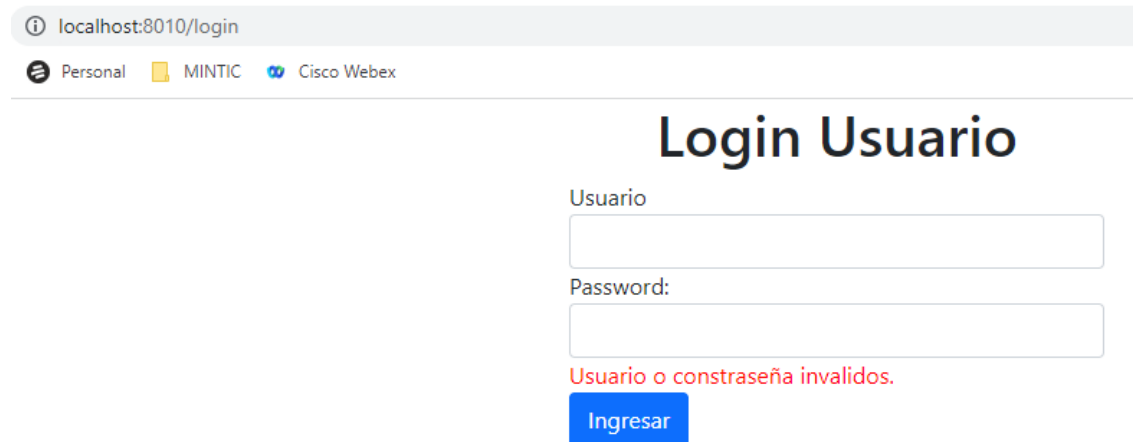
El complemento a esta operación esta implementada en la clase de servicio "UsuarioImp" explicada anteriormente.

## 9. Compile y ejecute el proyecto.

Para probar el proyecto ejecutamos en su orden el backend y el frontend con la opción Run As Sprint Boot App como lo muestra la gráfica.



Al probar el proyecto escribimos una clave errada para el usuario creado con anticipación llamado admin cuya clave es 123, además debemos crear antes datos en el tipodocumento.



Al escribir unas credenciales válidas el sistema debe responder de la siguiente forma.

