



UNIVERSIDAD
DE LA FRONTERA



Desafío 7

El objetivo de este desafío es poner en práctica todo lo aprendido sobre la conexión de backend y frontend que se ha desarrollado durante las clases.

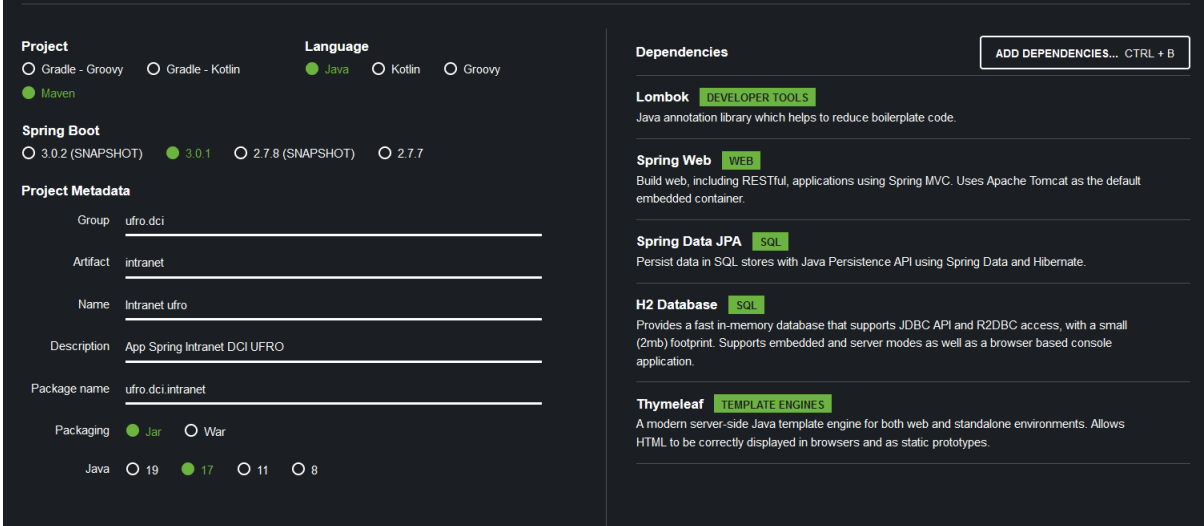
A partir de los desafíos desarrollados 1 y 5, se debe hacer uso de todas las herramientas necesarias para poder realizar una conexión entre la API y la página web de Inicio de Sesión. Los objetivos de este desafío son, primero permitir que un usuario pueda ingresar a su cuenta utilizando su correo electrónico y contraseña que registró previamente.

Lo segundo es crear una nueva pestaña que permita que los nuevos usuarios puedan registrar una cuenta, para ello debe haber un formulario que puedan ingresar todos los datos necesarios que solicita la API para poder registrar una nueva cuenta.

Y como último objetivo, que el usuario al ingresar a su cuenta la página lo redirija a través de su rol .

Resolución Desafío

Primero, generamos el proyecto a través de la herramienta de Spring Initializr, la configuración a utilizar será la siguiente.

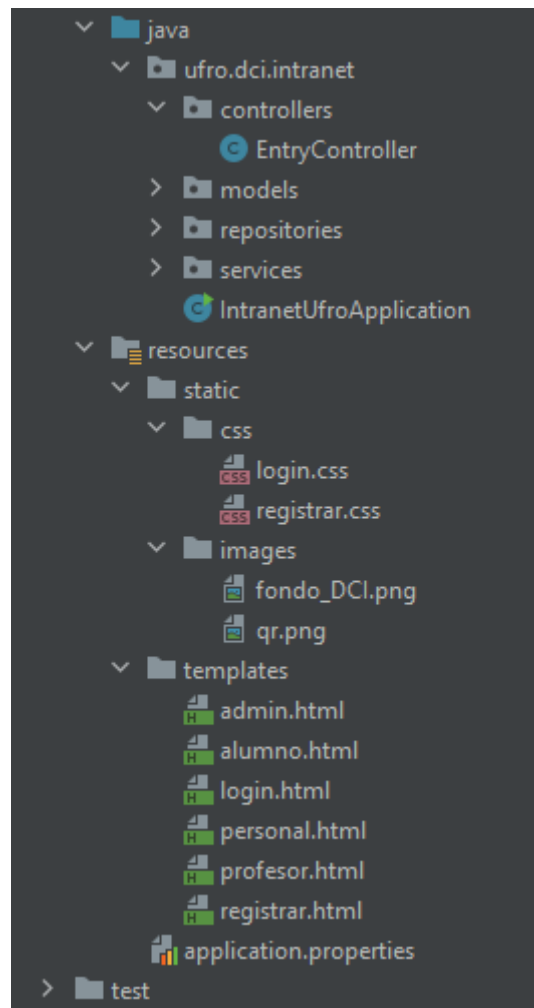


The screenshot shows the Spring Initializr configuration page. On the left, under 'Project', 'Maven' is selected. Under 'Language', 'Java' is selected. Under 'Spring Boot', '3.0.1' is selected. The 'Project Metadata' section includes: Group (ufro.dci), Artifact (intranet), Name (Intranet ufro), Description (App Spring Intranet DCI UFRO), Package name (ufro.dci.intranet), Packaging (Jar), and Java version (17). On the right, the 'Dependencies' section lists: Lombok (Developer Tools), Spring Web (Web), Spring Data JPA (SQL), H2 Database (SQL), and Thymeleaf (Template Engines). A button 'ADD DEPENDENCIES... CTRL + B' is at the top right of the dependencies section.

Este pequeño programa, estará basado en la resolución del desafío 1 y 5 realizado anteriormente, por ello, sólo agregaremos la dependencia de Thymeleaf.

- H2 Database: Como nuestro sistema de base de datos.
- Lombok: Para reducir el código repetido
- Spring Data JPA: Para manejar la persistencia de los objetos en nuestra base de datos
- Spring Web: Ya que la aplicación corresponderá a una API.
- Thymeleaf: Como motor de plantillas web, usaremos Html en el proyecto, pero se pueden utilizar diversos formatos.

Una vez descargado y extraído el archivo zip generado por la página, estructuramos nuestro proyecto de la siguiente manera.



La diferencia con el desafío 5, es que hemos eliminado el controlador “**UsuarioController**” que trabajaba como un controlador Rest Api, y en cambio crearemos una clase llamada **EntryController** que servirá para manejar las vistas y peticiones de nuestro usuario.

Además hemos añadido el desafío 1 en nuestro template de archivos html que servirá de login, junto con sus archivos css y recursos correspondientes.

Lo primero a realizar, es crear nuestro controlador EntryController, el cuál se verá de la siguiente manera.

```

@Controller
public class EntryController {
    @Autowired
    private UsuarioServices usuarioServices;

    @GetMapping("")
    public String initialPage(){
        return "redirect:/login";
    }

    @GetMapping("/login")
    public String loginView(Model model){
        model.addAttribute("loginDTO",new LoginDTO());
        return "login";
    }

    @GetMapping("/registrar")
    public String registrarView(Model model){
        model.addAttribute("registerDTO",new RegisterDTO());
        return "registrar";
    }

    @PostMapping("/api/login")
    public String login(LoginDTO loginDTO, Model model){
        try {
            Rol rol = usuarioServices.login(loginDTO);
            if (rol.equals(Rol.ADMIN)) return "admin";
            if (rol.equals(Rol.ALUMNO)) return "alumno";
            if (rol.equals(Rol.PERSONAL)) return "personal";
            return "profesor";
        } catch (Exception e){
            return "redirect:/login";
        }
    }

    @PostMapping("/api/registrar")
    public String registrarUsuario(RegisterDTO registerDTO,Model model){
        try {
            usuarioServices.registrarUsuario(registerDTO);
        } catch (Exception e) {
            return "registrar";
        }
        return "redirect:/login";
    }
}

```

Primero, implementaremos la clase **UsuarioServices** generada en el desafío 5 y lo integraremos a nuestro controlador con la etiqueta **@Autowired**.

Segundo, crearemos un método Get, el cuál recuperará cualquier dirección incorrecta hacia nuestro login. (Método **initialPage**)

Tercero, crearemos nuestra dirección **/login** para poder realizar el proceso de inicio de sesión de nuestro usuario, para ello le asignamos la dirección en nuestro **@GetMapping**. Además como trabajaremos con datos de inicio de sesión (correo, contraseña) utilizaremos un objeto creado anteriormente (**LoginDTO en desafío 5**) para obtener esos datos y compararlo con nuestra base de datos. Para utilizar estos datos, asignaremos un nombre de referencia, e iniciaremos el objeto. Aprovecharemos de crear el método para acceder a la página de registro, la cuál tiene la misma estructura, sólo cambiaremos el objeto a recibir, el cuál será de tipo **RegistroDTO** (también creado en nuestro desafío 5), lo referenciamos e iniciamos.

Finalmente, crearemos los métodos de inicio de sesión y registro, métodos que recibirán los datos y realizarán la parte lógica de nuestro programa, utilizando la clase **UsuarioServices** consultaremos en nuestra base de datos, obtendremos el rol, y a partir de ahí, realizaremos la comparación para acceder a las diferentes páginas de nuestros usuarios. Siguiendo con el método de registro, realizaremos el proceso de guardado de datos en la base de datos, y si los datos han sido registrados correctamente, enviaremos el usuario a nuestra página login.

Para que Thymeleaf pueda acceder a estos métodos POST, tendremos que acceder a ellos mediante formularios en nuestra página web. Para utilizar la misma plantilla del desafío 1 se deberán realizar pequeñas modificaciones en nuestro archivo html.

login.html

```
<form th:action="@{/api/login}" th:object="${loginDTO}" method="POST">
  <label>Correo Electrónico</label> <br>
  <input type="text" th:field="*{correo}"><br><br>
  <label>Contraseña</label> <br>
  <input type="password" th:field="*{contrasenna}" ><br><br><br>
  <input type="submit" value="Iniciar Sesión" class="submit-btn" />
</form>
<a th:href="@{/registrar}">
  <button class="btn-registrar">Registrar</button>
</a>
```

Para que nuestro formulario contacte a la dirección generada con los métodos **@PostMapping**, tendremos que usar algunas etiquetas extra.

Nuestro formulario ahora tendrá los siguientes elementos.

- **th:action="@{/api/login}"**: Con este elemento, indicaremos el método al cuál queremos acceder con nuestro formulario al momento de hacer clic en *"Iniciar Sesión"* por su URL.
- **th:object="{loginDTO}"**: Con th:object estaremos indicando a Thymeleaf qué elemento estamos referenciando. Este nombre debe ser igual al generado en el método **loginView**.
- **method="POST"**: Indicaremos que estamos utilizando un método POST.
- **th:field="**{correo}"**: Indicaremos a qué atributo pertenece cada uno de nuestros inputs, referentes a nuestro objeto. En este caso de la clase **LoginDTO**.

En el apartado visual, también tendremos que hacer cambios, como nuestro archivo **CSS** e imágenes ahora se encuentran en la carpeta **"static"**, deberemos referenciarlas. Thymeleaf ayuda un poco al momento de referenciar archivos si se utilizan las carpetas específicas, la carpeta **"static"** está orientada a guardar archivos "estáticos", como imágenes, archivos css, javascript, etc.

```
<link rel='stylesheet' type='text/css' media='screen'  
th:href="@{/css/login.css}">
```

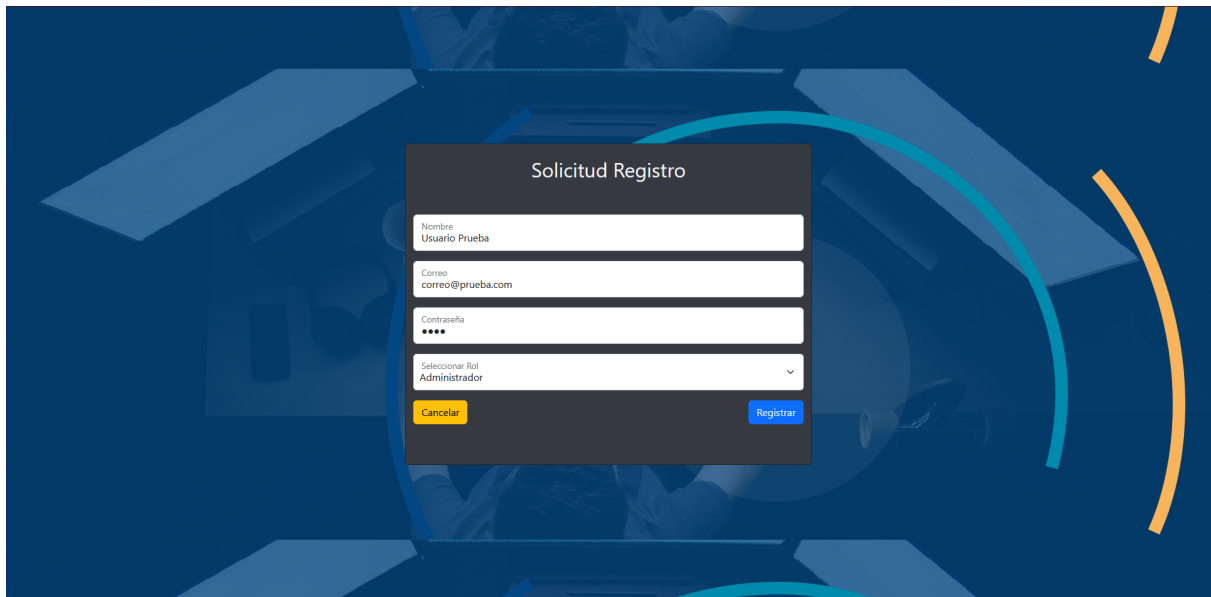
- **th:href="@{/css/login.css}"** : acá realizaremos la conexión con nuestro archivo css, como puedes notar, la referencia se hace directamente a la carpeta **"static"**, por lo que sólo basta agregar la ruta a partir de ahí.

Finalmente, sólo queda probar la aplicación y ver cómo todo funciona acorde a lo programado.

Pantalla login



Pantalla Registro

The image shows a registration form titled "Solicitud Registro" centered on a dark blue background with abstract geometric shapes. The form is a light gray rectangle with rounded corners. It contains four input fields: "Nombre" with the value "Usuario Prueba", "Correo" with the value "correo@prueba.com", "Contraseña" with masked characters "****", and a dropdown menu for "Seleccionar Rol" with "Administrador" selected. At the bottom of the form are two buttons: a yellow "Cancelar" button on the left and a blue "Registrar" button on the right.

Esta pantalla se hizo utilizando bootstrap, como el objetivo del desafío no es enseñar a realizar el front-end, se omitirá el cómo se realizó esta pantalla. Lo importante es hacer un formulario referenciando al método POST de registro, similarmente a como se explicó anteriormente para el apartado del login.

Para probar la aplicación, crearemos los siguientes usuarios.

Administrador:

- Usuario: Usuario Administrador
- Correo: admin@prueba.com
- Contraseña: 1234
- Rol: Administrador

Personal:

- Usuario: Usuario Personal
- Correo: personal@prueba.com
- Contraseña: 1234
- Rol: Personal

Profesor:

- Usuario: Usuario Profesor
- Correo: profesor@prueba.com
- Contraseña: 1234
- Rol: Profesor

Alumno:

- Usuario: Usuario Alumno
- Correo: alumno@prueba.com
- Contraseña: 1234
- Rol: Alumno

Finalmente, en nuestra pantalla de login accederemos con las credenciales. Podremos observar cada una de las vistas para los usuarios.

Pantalla Alumno

ALUMNO

Trabajo Futuro

- Pantalla para inicio de sesión del Administrador
- Pantalla para inicio de sesión del Personal
- Pantalla para inicio de sesión del Profesor
- Pantalla para inicio de sesión del Alumno

Acceso al github con la solución

<https://github.com/Spawnbig/desafios-proyecto-aplicacion>