**Hilla + React + SringBoot**

**JWT Authentication**



Security setup

Habilitaremos la autenticación mediante el uso de Json Web Tokens (JWT). Estos tokens son generados por el servidor para ser utilizados por el cliente en solicitudes posteriores.

## Agregando dependencias

Agregaremos las siguientes dependencias a pom.xml. Estas dependencias nos permiten configurar Spring Boot Security para proteger nuestra aplicación.

<!-- pom.xml -->

        <dependency>

            <groupId>org.springframework.boot</groupId>

            <artifactId>spring-boot-starter-security</artifactId>

        </dependency>

        <dependency>

            <groupId>org.springframework.security</groupId>

            <artifactId>spring-security-oauth2-jose</artifactId>

        </dependency>

        <dependency>

            <groupId>org.springframework.security</groupId>

            <artifactId>spring-security-oauth2-resource-server</artifactId>

        </dependency>

## Generación de una clave

Necesitamos una clave secreta. Puedes generar una según tus preferencias. Aquí tienes un enlace a un generador en línea que puedes usar

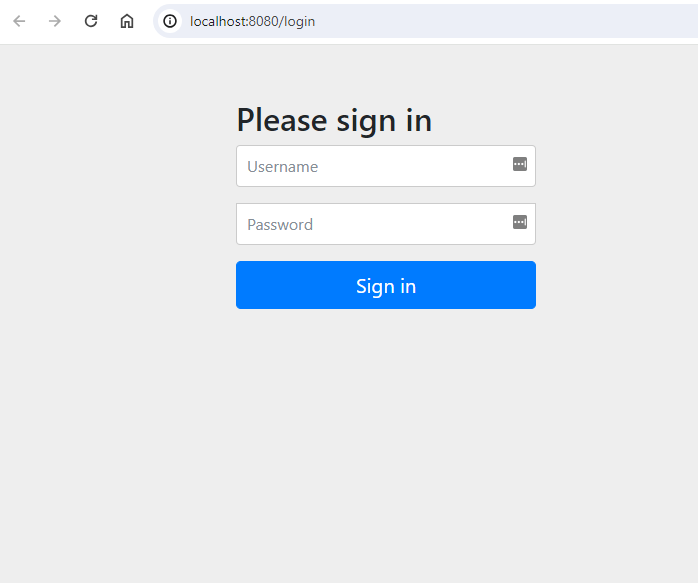
<https://generate.plus/en/base64>

Una vez que tenga una clave, deberá guardarla donde Spring pueda acceder a ella.

**//src/resources/application.properties**

app.secret=tM12lOHxuA4lqLvPyhcq3Q==

Si inicia la aplicación en este punto, será redirigido a http://localhost:8080/login, donde debería ver la siguiente pantalla.



## Backend configuration

En este punto la seguridad está habilitada, pero no configurada y el siguiente paso es poder iniciar sesión en nuestra aplicación. Para hacer esto, primero crearemos otro archivo Java

**//src/main/java/com/example/application/security/SecurityConfiguration.java**

package com.example.application.security;

import java.util.Base64;

import javax.crypto.spec.SecretKeySpec;

import com.vaadin.flow.spring.security.VaadinWebSecurity;

import org.springframework.beans.factory.annotation.Value;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.security.config.annotation.web.builders.HttpSecurity;

import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;

import org.springframework.security.config.http.SessionCreationPolicy;

import org.springframework.security.core.userdetails.User;

import org.springframework.security.oauth2.jose.jws.JwsAlgorithms;

import org.springframework.security.provisioning.InMemoryUserDetailsManager;

import org.springframework.security.provisioning.UserDetailsManager;

/\*\*

 \* <code>VaadinWebSecurity</code> used in favor of the now deprecated

 \* <code>VaadinWebSecurityConfigurerAdapter</code>

 \*

 \* @see com.vaadin.flow.spring.security.VaadinWebSecurityConfigurerAdapter

 \*/

@*EnableWebSecurity*

@*Configuration*

public class SecurityConfiguration extends *VaadinWebSecurity* { // punto 1

    @*Value*("${app.secret}")

    private *String* appSecret;

    @*Override*

    protected *void* configure(*HttpSecurity* *http*) throws *Exception* { // punto 2

        super.configure(http);

        // Disable creating and using sessions in Spring Security

        http.sessionManagement(management *->* management

                .sessionCreationPolicy(SessionCreationPolicy.STATELESS));

        // Register your login view to the view access checker mechanism

        setLoginView(http, "/login");

        // Enable stateless authentication

        setStatelessAuthentication(http,

                new SecretKeySpec(Base64.getDecoder().decode(

                        appSecret),

                        JwsAlgorithms.HS256),

                "localhost");

    }

    @*Bean*

    public *UserDetailsManager* userDetailsService() { // punto 3

        // Configure users and roles in memory

        return new InMemoryUserDetailsManager(

                // the {noop} prefix tells Spring that the password is not encoded

                //User.withUsername("user").password("{noop}user").roles("USER").build(),

                //User.withUsername("admin").password("{noop}admin").roles("ADMIN", "USER").build()

                User.withDefaultPasswordEncoder().username("user").password("password").roles("USER").build()

        );

    }

}

Vamos a comentar estos tres puntos marcados en el código: **// punto 1, 2 y 3**

1. Nuestra clase SecurityConfiguration extiende VaadinWebSecurity. Esta clase proporciona valores predeterminados seguros para el backend de Hilla: de forma predeterminada, restringe el acceso a todas las URL a los usuarios que han iniciado sesión.
2. Aquí definimos nuestro punto final de inicio de sesión como /login y configuramos nuestra aplicación para usar sesiones sin estado con JWT.
3. Finalmente, creamos un usuario en memoria que podemos usar para iniciar sesión en la aplicación mientras la ejecutamos en localhost. Tenga en cuenta que esto no es seguro para implementaciones de producción y el compilador tendrá la amabilidad de advertirle sobre esto.

## Rutas

Podemos usar estas funciones para implementar rápidamente una vista de inicio de sesión agregando lo siguiente a **frontend/routes.tsx**.

import HelloWorldView from 'Frontend/views/helloworld/HelloWorldView.js';

import MainLayout from 'Frontend/views/MainLayout.js';

import { lazy } from 'react';

import { createBrowserRouter, RouteObject } from 'react-router-dom';

*const* AboutView = lazy(async () *=>* import('Frontend/views/about/AboutView.js'));

import TodoView from './views/todo/TodoView';

import { LoginView } from './views/LoginView';

*const* routing = [

  {path:'/login', element: <*LoginView*/>},

  {

    element: <*MainLayout* />,

    handle: { title: 'Main' },

    children: [

      { path: '/', element: <*HelloWorldView* />, handle: { title: 'Hello World' } },

      { path: '/about', element: <*AboutView* />, handle: { title: 'About' } },

      { path: '/todo', element: <*TodoView* />, handle: { title: 'Todo' } },

    ],

  },

] as RouteObject[];

export *const* routes = routing;

export default createBrowserRouter(routes);

y por último la vista del login

**//frontend/views/LoginView.tsx**

import { login } from '@hilla/frontend';

import { LoginForm } from '@hilla/react-components/LoginForm.js';

import { useNavigate } from 'react-router-dom';

import { useState } from 'react';

export *function* LoginView() {

*const* navigateTo = useNavigate();

*const* [hasError, setError] = useState<*boolean*>();

   return(

       <div className="flex items-center justify-center h-full">

           <*LoginForm*

              noForgotPassword

              title='TODO List'

              error={hasError}

              onLogin={

              async ({ detail: { *username*, *password* } }) *=>* {

                setError(false);

*const* result = await login(*username*, *password*);

                //console.dir(JSON.stringify(result));

                if (result.error) {

                  setError(true);

                } else {

                  navigateTo('/');

                }

              }

           }/>

       </div>

   )

}

## Enlaces útiles

<https://vaadin.com/docs/next/hilla/guides/security/spring-login>

<https://medium.com/@robert83sutherland/hilla-react-jwt-authentication-60e0bd7814fe>