Clase grabada

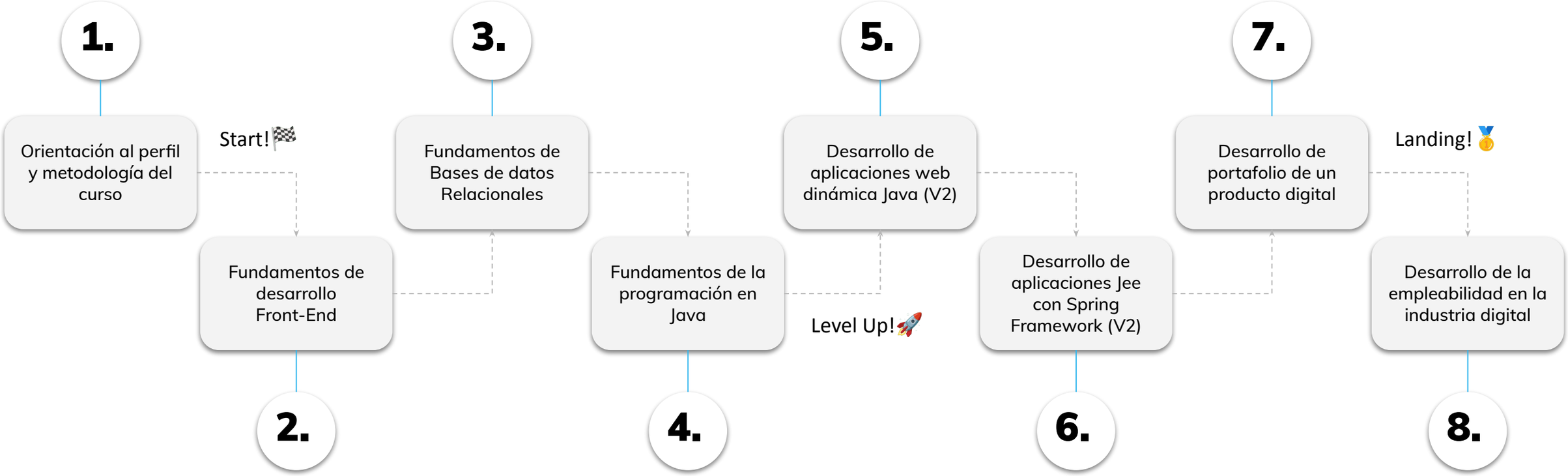
Control de Acceso mediante Spring Security

Plan formativo: Desarrollo de Aplicaciones Full Stack Java Trainee V2.0

English Always | alkemy

HOJA DE RUTA

¿Cuáles **skill** conforman el programa?



English Always | alkemy

REPASO CLASE ANTERIOR

En la clase anterior trabajamos 📚:

- ✔ *Incorporar el módulo Spring Security al proyecto*
- ✔ *Configurar Spring Security en el proyecto*

English Always | alkemy

LEARNING PATHWAY

¿Sobre qué temas trabajaremos?

6.3 Start! 🏁

Control de Acceso mediante Spring Security

Creando un formulario de Login
Realizando Login y Logout de una aplicación

Login

Configurando usuarios

English Always | alkemy

OBJETIVOS DE APRENDIZAJE

¿Qué aprenderemos?

- *Aprender a crear un formulario de login con Spring Security*
- *Comprender la configuración de login y logout de una aplicación*

Administrando los usuarios del programa

Administrando los usuarios del programa

En Spring Security, la interfaz UserDetailsService juega un papel fundamental en la **autenticación**. Su función principal es cargar los detalles del usuario a partir de algún origen de datos, como una base de datos, un servicio web o incluso una configuración en memoria. Al implementar esta interfaz, puedes personalizar la carga de usuarios y roles según las necesidades de tu aplicación.

Administrando los usuarios del programa

Para que Spring pueda obtener la información de un usuario durante el proceso de login, se solicitará al conjunto de clases Java registradas en el contexto de Spring, que alguna implemente la interfaz UserDetailsService. Esta interfaz tiene un único método de obligada implementación que es aquel que devuelve los detalles de un usuario UserDetails dado un nombre de usuario, el que trata de ingresar en la aplicación. Por tanto será obligatorio implementar un bean con esa interfaz dando cuerpo a ese método.

Administrando los usuarios del programa

Características Clave de UserDetailsService:

1- **Carga de Detalles del Usuario**: la interfaz UserDetailsService contiene un solo método, loadUserByUsername, que se encarga de cargar los detalles del usuario según su nombre de usuario.

2- **Retorno de UserDetails**: el método loadUserByUsername devuelve un objeto UserDetails, que representa los detalles del usuario, incluyendo su nombre de usuario, contraseña encriptada y roles asignados.

3- **Integración con Spring Security**: la implementación de UserDetailsService se integra en la configuración de Spring Security para proporcionar detalles de usuario durante el proceso de autenticación.

Administrando los usuarios del programa

Veamos un ejemplo práctico dónde la interfaz UserDetailsService será implementada por una clase que maneja el servicio de Usuario: UserService, y el método a sobreescribir se llama loadUserByUsername().

```
@Service
public class UserService implements UserDetailsService {

    @Autowired
    private UserRepository userRepository;
```

Administrando los usuarios del programa

También debemos tener en cuenta que la interfaz de repositorio (en este ejemplo: UserRepository) extienda de JpaRepository o CrudRepository:

```java
@Repository
public interface UserRepository extends JpaRepository<UserEntity, String> {

    @Query("SELECT u FROM UserEntity u WHERE u.email = :email")
    public UserEntity searchByEmail(@Param("email")String email);
}
```

En este repositorio hemos declarado un pequeño método personalizado de **búsqueda de usuario por email** (searchByEmail(String email)).

Administrando los usuarios del programa

Finalmente, podemos ver como quedaría el método loadUserByUsername en la clase de servicio. Avancemos para poder ver la explicación de cada línea de código.

```
149 @Override
150 public UserDetails loadUserByUsername(String email) throws UsernameNotFoundException {
151
152    UserEntity user = userRepository.searchByEmail(email);
153
154    if (user != null) {
155
156     List<GrantedAuthority> permissions = new ArrayList<>();
157
158     GrantedAuthority p = new SimpleGrantedAuthority("ROLE_" + user.getRole().toString());
159
160     permissions.add(p);
161
162     ServletRequestAttributes attr = (ServletRequestAttributes) RequestContextHolder.currentRequestAttributes();
163
164     HttpSession session = attr.getRequest().getSession(true);
165
166     session.setAttribute("usuariosession", user);
167
168    return new User(user.getEmail(), user.getPassword(), permissions);
169
170    } else {
171
172    return null;
173    }
174 }
```

Vamos a referenciar las líneas del código para explicar el método:

152- Instanciamos un objeto del tipo UserEntity, haciendo uso del método searchByEmail(email) de la clase UserRepository.

154- Verificamos que el objeto Usuario **no esté nulo**.

```
149 @Override
150 public UserDetails loadUserByUsername(String email) throws UsernameNotFoundException {
151
152    UserEntity user = userRepository.searchByEmail(email);
153
154    if (user != null) {
155
156     List<GrantedAuthority> permissions = new ArrayList<>();
157
158     GrantedAuthority p = new SimpleGrantedAuthority("ROLE_" + user.getRole().toString());
159
160     permissions.add(p);
161
162     ServletRequestAttributes attr = (ServletRequestAttributes) RequestContextHolder.currentRequestAttributes();
163
164     HttpSession session = attr.getRequest().getSession(true);
165
166     session.setAttribute("usuariosession", user);
167
168    return new User(user.getEmail(), user.getPassword(), permissions);
169
170    } else {
171
172    return null;
173    }
174 }
```

Vamos a referenciar las líneas del código para explicar el método:

156- Otorgamos permisos según el rol que tenga cada usuario. Si el usuario existe, creamos una lista de permisos llamada permissions.

158- Creamos un objeto GrantedAuthority 'p' y concatenamos la palabra ROLE_ + el rol del usuario.

```
149 @Override
150 public UserDetails loadUserByUsername(String email) throws UsernameNotFoundException {
151
152     UserEntity user = userRepository.searchByEmail(email);
153
154     if (user != null) {
155
156      List<GrantedAuthority> permissions = new ArrayList<>();
157
158      GrantedAuthority p = new SimpleGrantedAuthority("ROLE_" + user.getRole().toString());
159
160      permissions.add(p);
161
162      ServletRequestAttributes attr = (ServletRequestAttributes) RequestContextHolder.currentRequestAttributes();
163
164      HttpSession session = attr.getRequest().getSession(true);
165
166      session.setAttribute("usuariosession", user);
167
168     return new User(user.getEmail(), user.getPassword(), permissions);
169
170     } else {
171
172     return null;
173     }
174 }
```

Vamos a referenciar las líneas del código para explicar el método:

160- Agregamos el objeto 'p' a la lista de permisos.

162/164- Utilizamos los atributos que nos otorga el pedido al servlet, para poder guardar la información de nuestra HttpSession.

```
149 @Override
150 public UserDetails loadUserByUsername(String email) throws UsernameNotFoundException {
151
152    UserEntity user = userRepository.searchByEmail(email);
153
154    if (user != null) {
155
156     List<GrantedAuthority> permissions = new ArrayList<>();
157
158     GrantedAuthority p = new SimpleGrantedAuthority("ROLE_" + user.getRole().toString());
159
160     permissions.add(p);
161
162     ServletRequestAttributes attr = (ServletRequestAttributes) RequestContextHolder.currentRequestAttributes();
163
164     HttpSession session = attr.getRequest().getSession(true);
165
166     session.setAttribute("usuariosession", user);
167
168    return new User(user.getEmail(), user.getPassword(), permissions);
169
170    } else {
171
172    return null;
173    }
174 }
```

Vamos a referenciar las líneas del código para explicar el método:

168- por último, retornamos un User con su email, contraseña y permisos!

```
149 @Override
150 public UserDetails loadUserByUsername(String email) throws UsernameNotFoundException {
151
152    UserEntity user = userRepository.searchByEmail(email);
153
154    if (user != null) {
155
156     List<GrantedAuthority> permissions = new ArrayList<>();
157
158     GrantedAuthority p = new SimpleGrantedAuthority("ROLE_" + user.getRole().toString());
159
160     permissions.add(p);
161
162     ServletRequestAttributes attr = (ServletRequestAttributes) RequestContextHolder.currentRequestAttributes();
163
164     HttpSession session = attr.getRequest().getSession(true);
165
166     session.setAttribute("usuariosession", user);
167
168    return new User(user.getEmail(), user.getPassword(), permissions);
169
170    } else {
171
172    return null;
173    }
174 }
```

Administrando los usuarios del programa

La implementación de este bean **dependerá de dónde tengamos registrados los usuarios**. Normalmente éstos estarán en una tabla de la base de datos, así que ya sea con SQL nativo o usando JPA con EntityManager deberemos consultar la fila asociada según el nombre de usuario. Una vez localizado sólo restará convertir o mapear ese objeto a uno esperado por Spring, del tipo UserDetails.

Creando un formulario de login

Creando un formulario de login

Spring Security facilita la implementación de un formulario de login personalizado. Al configurar Spring Security, puedes definir la página de login, las URL de autenticación, y personalizar el comportamiento del formulario. La autenticación basada en formularios es una de las estrategias más comunes, y Spring Security la hace fácil de implementar.

Creando un formulario de login

Dentro de la clase que extiende de WebSecurityConfigurerAdapter (en nuestro ejemplo, la clase SecurityConfig), vamos a sobrescribir el método configure():

```
@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
            .authorizeRequests()
                .antMatchers("/public/**").permitAll()
                .anyRequest().authenticated()
                .and()
            .formLogin()
                .loginPage("/login") // Ruta a la página de login personalizada
                .permitAll() // Permite a todos acceder a la página de login
                .and()
            .logout()
                .permitAll();
    }
}
```

Creando un formulario de login

.anyRequest().authenticated() hace referencia a que se requiere estar autenticado para todas las peticiones.

```
@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
            .authorizeRequests()
                .antMatchers("/public/**").permitAll()
                .anyRequest().authenticated()
                .and()
            .formLogin()
                .loginPage("/login") // Ruta a la página de login personalizada
                .permitAll() // Permite a todos acceder a la página de login
                .and()
            .logout()
                .permitAll();
    }
}
```

Creando un formulario de login

.formLogin().loginPage("/login").permitAll() refiere a que el formulario será definido en la página login.jsp (o login.html) que crearemos más adelante, las peticiones a esta página no requieren autenticación, cualquier usuario puede acceder a ella.

```
@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
            .authorizeRequests()
                .antMatchers("/public/**").permitAll()
                .anyRequest().authenticated()
                .and()
            .formLogin()
                .loginPage("/login") // Ruta a la página de login personalizada
                .permitAll() // Permite a todos acceder a la página de login
                .and()
            .logout()
                .permitAll();
    }
}
```

Creando un formulario de login

.logout().permitAll() activa el cierre de sesión mediante la URL "/logout". Cualquier usuario puede acceder a esta URL.

```
@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
            .authorizeRequests()
                .antMatchers("/public/**").permitAll()
                .anyRequest().authenticated()
                .and()
            .formLogin()
                .loginPage("/login") // Ruta a la página de login personalizada
                .permitAll() // Permite a todos acceder a la página de login
                .and()
            .logout()
                .permitAll();
    }
}
```

Creando un formulario de login

Lo que prosigue es crear el formulario JSP (o HTML), para que funcione correctamente debe tener dos campos: username y password, estos deben ser enviados usando el método POST a la URL "/login" (como se especificó en la configuración.

```
<!-- login.jsp -->
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncodi
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Login</title>
</head>
<body>
    <h2>Login</h2>
    <form action="<c:url value='/login' />" method="post">
        <label for="username">Username:</label>
        <input type="text" id="username" name="username" required>
        <br>
        <label for="password">Password:</label>
        <input type="password" id="password" name="password" required>
        <br>
        <button type="submit">Login</button>
    </form>
</body>
</html>
```

Creando un formulario de login

Finalmente, crea un controlador simple para manejar la solicitud de la página de login. Este controlador simplemente devuelve el nombre de la vista (el nombre del archivo JSP sin extensión) cuando se accede a la URL /login.

```
@Controller
public class LoginController {

    @GetMapping("/login")
    public String login() {
        return "login";
    }
}
```

Ahora, puedes ejecutar tu aplicación y acceder a la página de login personalizada en http://localhost:8080/login. Ingresa las credenciales de usuario configuradas en la clase SecurityConfig y observa cómo Spring Security maneja la autenticación.

Momento:

Time-out!

⏰5 min.

Evaluación Integradora ✨

¿Listos para un nuevo desafío? En esta clase comenzamos a construir nuestro....

Trabajo Integrador del Módulo 💪🏽

Iremos completándolo progresivamente clase a clase.

English Always | alkemy

LIVE CODING

Ejemplo en vivo

Login en la Wallet

- *Vamos a agregar las configuraciones de autenticación para el proyecto AlkeWallet.*

*1- Implementar la interfaz **UserDetailsService** en el servicio de usuario y sobreescribir el método **loadUserByUsername**.*

*2- Personalizar la URL de **login** y **logout** en la clase de configuración.*

Tiempo: 20 minutos

Ejercicio N° 1

Configurando usuarios

Módulo de seguridad

Contexto: 🙌

Vamos a continuar con el ejercicio de la clase anterior. En este caso te toca a ti configurar la administración de usuarios de tu aplicación.

Consigna: ✍️

1- Implementar la interfaz UserDetailsService en el servicio de usuario y sobreescribir el método loadUserByUsername.

Tiempo🕐: *15 minutos (puedes continuar de manera asíncrona)*

¿Alguna consulta?

RESUMEN

¿Qué logramos en esta clase?

- ✓ *Comprender la administración de usuarios con Spring Security*
- ✓ *Aprender a configurar un formulario de login con Spring Security*

English Always | alkemy

#WorkingTime

Continuemos ejercitando

¡Antes de cerrar la clase! Te invitamos a: 👇👇👇

1. Repasar nuevamente la grabación de esta clase
2. Revisar el material compartido en la plataforma de Moodle (lo que se vio en clase y algún ejercicio adicional)
   a. *Lectura Modulo 6, Lección 4: páginas 5 - 9*
3. Traer al próximo encuentro, todas tus dudas y consultas para verlas antes de iniciar nuevo tema.

Momento:

Time-out!

⏰5 min.