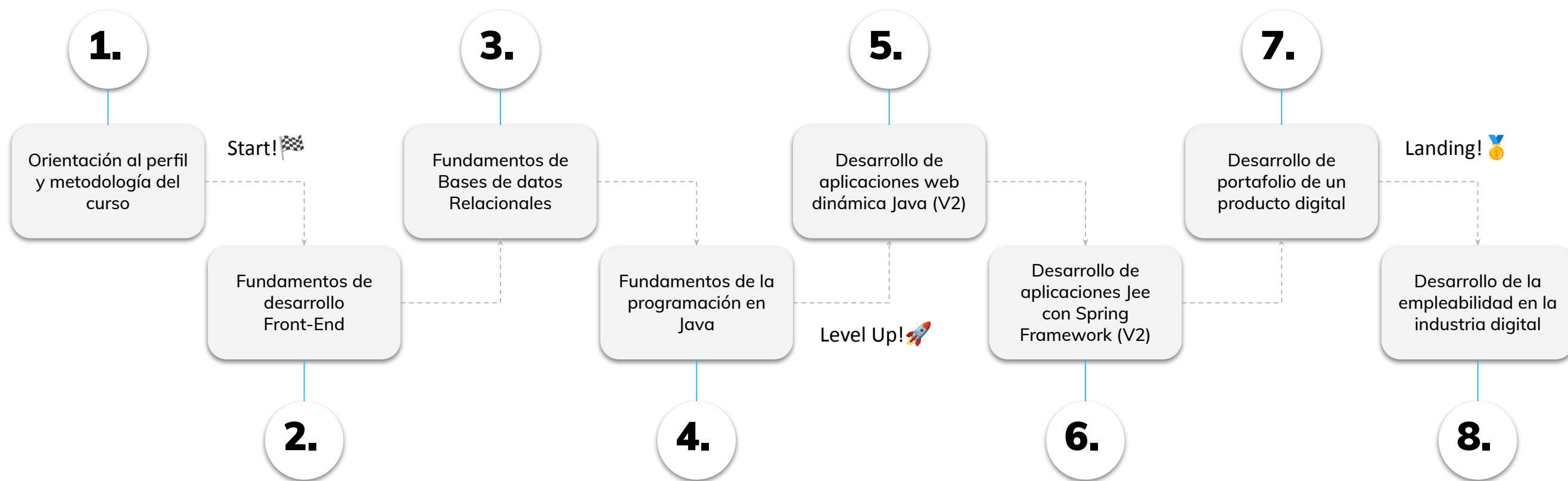


➤ Control de Acceso mediante Spring Security

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

HOJA DE RUTA

¿Cuáles **skill** conforman el programa?



REPASO CLASE ANTERIOR

En la clase anterior trabajamos :

- ✓ Aprender a manejar roles de usuarios autenticados
- ✓ Comprender la configuración de seguridad de la vista con Spring Security

LEARNING PATHWAY

¿Sobre qué temas trabajaremos?

6.3

Start! 🏁

Control de Acceso
mediante Spring
Security

Anotaciones de seguridad

Securizando

Agregando seguridad en los controladores mediante anotaciones

OBJETIVOS DE APRENDIZAJE

¿Qué aprenderemos?



Aprender a manejar roles de usuarios autenticados



Comprender la configuración de seguridad de la vista con Spring Security



➤ Agregando seguridad en los controladores mediante anotaciones

Agregando seguridad

en los controladores mediante anotaciones

Spring Security nos permite hacer uso de anotaciones que restringen los accesos a nivel método. Normalmente estas anotaciones se presentan en los componentes de tipo controlador, para poder manejar el nivel de acceso a cada endpoint.



Agregando seguridad

en los controladores mediante anotaciones

Ventajas de Agregar Seguridad con Anotaciones

1. **Declarativo y Expresivo:** Las anotaciones se integran directamente en el código, facilitando la comprensión de las reglas de seguridad.
2. **Flexibilidad y Granularidad:** Puedes aplicar restricciones específicas a nivel de método, adaptando la seguridad según las necesidades específicas de cada funcionalidad.






Agregando seguridad

en los controladores mediante anotaciones



Ventajas de Agregar Seguridad con Anotaciones

- 
3. **Fácil Mantenimiento:** Las anotaciones proporcionan una forma clara y fácil de mantener las reglas de seguridad. La capacidad de visualizar rápidamente las restricciones en el propio código facilita la tarea de mantenimiento y actualización.
4. **Integración con Expresiones y Parámetros:** El uso de expresiones y parámetros en las anotaciones ofrece una capacidad poderosa para personalizar las reglas de seguridad según la lógica específica de cada método.
5. **Menos Configuración Externa:** Añadir seguridad mediante anotaciones puede reducir la necesidad de configuración externa, ya que las reglas de seguridad se especifican directamente en el código.




Agregando seguridad

en los controladores mediante anotaciones




Desventajas de Agregar Seguridad con Anotaciones

1. Acoplamiento con el Código: El principal inconveniente de agregar seguridad mediante anotaciones es que puede acoplar el código con las reglas de seguridad. Esto podría hacer que el código sea más difícil de entender para alguien que no esté familiarizado con la lógica de seguridad.



2. Complejidad en Anotaciones: A medida que las expresiones de seguridad se vuelven más complejas, la lectura y mantenimiento del código puede volverse desafiante. Las expresiones extremadamente complejas pueden dificultar la comprensión de las reglas de seguridad.



3. Menos Separación de Preocupaciones: Al agregar seguridad directamente en el código, puede haber una menor separación de preocupaciones entre la lógica de negocio y las reglas de seguridad. Algunos desarrolladores prefieren mantener estas responsabilidades separadas para una mayor modularidad.

Agregando seguridad

en los controladores mediante anotaciones

1- La anotación **@Secured** se utiliza para restringir el acceso a métodos o clases a usuarios con roles específicos.

```
@Secured("ROLE_USER")
@GetMapping("/user/profile")
public String userProfile() {
    return "user";
}

@Secured({"ROLE_ADMIN", "ROLE_MANAGER"})
@GetMapping("/admin")
public String adminPage() {
    return "admin";
}
```

Agregando seguridad

en los controladores mediante anotaciones

@Secured

Ventajas:

- Simplicidad: Es fácil de entender y aplicar.
- Roles Estáticos: Funciona bien cuando los roles son estáticos y conocidos de antemano.

Desventajas:

- Rigidez: Puede ser rígido cuando se trata de roles dinámicos o cuando la lógica de autorización es más compleja.



Agregando seguridad

en los controladores mediante anotaciones

2- **@PreAuthorize** y **@PostAuthorize**: Expresiones de Seguridad más Flexibles
Estas anotaciones permiten la utilización de expresiones de seguridad basadas en el lenguaje SpEL (Spring Expression Language).

```
@PreAuthorize("hasRole('USER') and #username == authentication.name")
@GetMapping("/user/{username}")
public String userDetails(@PathVariable String username) {
    return "user-details";
}

@PostAuthorize("hasRole('ADMIN') or hasRole('MANAGER')")
@GetMapping("/admin/details")
public String adminDetails() {
    return "admin-details";
}
```

Agregando seguridad

en los controladores mediante anotaciones

@PreAuthorize y @PostAuthorize

También puede pre-autorizar el acceso al endpoint para cualquier rol declarado:

```
@PreAuthorize("hasAnyRole('ROLE_USER', 'ROLE_ADMIN')")
@GetMapping("/user/profile")
public String userProfile() {
    return "user";
}
```

La anotación **@PreAuthorize** verifica la expresión dada antes de ingresar el método , mientras que la anotación **@PostAuthorize** la verifica después de la ejecución del método y podría alterar el resultado.



Agregando seguridad

en los controladores mediante anotaciones




@PreAuthorize y **@PostAuthorize**

Ventajas:



- Expresiones Dinámicas: Permite expresiones más dinámicas y complejas.
- Más Control: Mayor control sobre cuándo se evalúa la expresión (antes o después de la ejecución del método).

Desventajas:

- Curva de Aprendizaje: Puede tener una curva de aprendizaje mayor debido a la complejidad de las expresiones.
- 

Agregando seguridad

en los controladores mediante anotaciones

3- **@Secured**, **@PreAuthorize** y **@PostAuthorize** Juntos: Mayor Flexibilidad
En muchos casos, combinar estas anotaciones proporciona la máxima flexibilidad.

```
@Secured("ROLE_USER")
@PreAuthorize("hasRole('USER') and #username == authentication.name")
@GetMapping("/user/{username}")
public String userDetails(@PathVariable String username) {
    return "user-details";
}

@Secured({"ROLE_ADMIN", "ROLE_MANAGER"})
@PostAuthorize("hasRole('ADMIN') or hasRole('MANAGER')")
@GetMapping("/admin/details")
public String adminDetails() {
    return "admin-details";
}
```


Agregando seguridad

en los controladores mediante anotaciones

@Secured, **@PreAuthorize** y **@PostAuthorize** Juntos

Ventajas:

- Flexibilidad Máxima: Permite aplicar múltiples niveles de restricciones según la situación.

Desventajas:

- Complejidad: Puede resultar complejo si no se maneja cuidadosamente.



Agregando seguridad

en los controladores mediante anotaciones

4- La anotación **@RolesAllowed** permite especificar roles permitidos utilizando una lista de cadenas.

```
@RolesAllowed("ROLE_USER")
@GetMapping("/user")
public String userPage() {
    return "user";
}
```



Agregando seguridad

en los controladores mediante anotaciones

@RolesAllowed

Ventajas:

- Declaratividad: La declaración explícita de roles hace que sea claro cuáles son los roles permitidos.

Desventajas:

- Limitado: Puede ser menos flexible que otras opciones cuando se necesitan expresiones más complejas.



Agregando seguridad

en los controladores mediante anotaciones

Consideraciones importantes:

- **Habilitar Anotaciones de Seguridad:** Asegúrate de que las anotaciones de seguridad estén habilitadas en la configuración de tu aplicación. Puedes lograr esto utilizando la anotación `@EnableGlobalMethodSecurity` en tu clase de configuración.
- **Uso con Métodos de Servicio:** Las anotaciones de seguridad no están limitadas a los controladores. También puedes aplicarlas en métodos de servicio u otros componentes de tu aplicación donde sea necesario controlar el acceso.



Evaluación Integradora ✨

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

Trabajo Integrador del Módulo 💪

Iremos completándolo progresivamente clase a clase.



LIVE CODING

Ejemplo en vivo

Securizando la Wallet

Vamos a agregar las anotaciones de seguridad al proyecto AlkeWallet.

- *Implementar 5 métodos controladores:*

1- controlador para la página de perfil del usuario: Sólo accede el usuario autenticado.

2- controlador para una página con dashboard de administración: Sólo accede un usuario autenticado con rol Admin

LIVE CODING

Ejemplo en vivo

3- controlador para una página de inicio (index): Acceso permitido a cualquier usuario no autenticado

4- controlador de login: Acceso permitido a cualquier usuario no autenticado

5- controlador página de transacciones: acceso permitido a usuario autenticado en sesión.

  **Tiempo: 30 minutos**





Ejercicio **Securizando**



Securizando



Contexto: 🙌

Vamos a continuar con el ejercicio de la clase anterior. En este caso debemos securizar una página para que solo los usuarios con el rol "**USER**" puedan acceder a ella.

Consigna: 📝

- 1- Crea un controlador que maneje la solicitud para la página protegida.
- 2- Agrega la anotación **@PreAuthorize** sólo para roles **USER**.

Tiempo 🕒: 10 minutos (puedes continuar de manera asíncrona)

○

¿Alguna consulta?

+



RESUMEN

¿Qué logramos en esta clase?

- ✓ **Comprender cómo implementar seguridad en los controladores mediante anotaciones**



#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 13 - 14*
3. Traer al próximo encuentro, todas tus dudas y consultas para verlas antes de iniciar nuevo tema.

Momento: ✚

Time-out!

🕒 5 min.

