Clase grabada

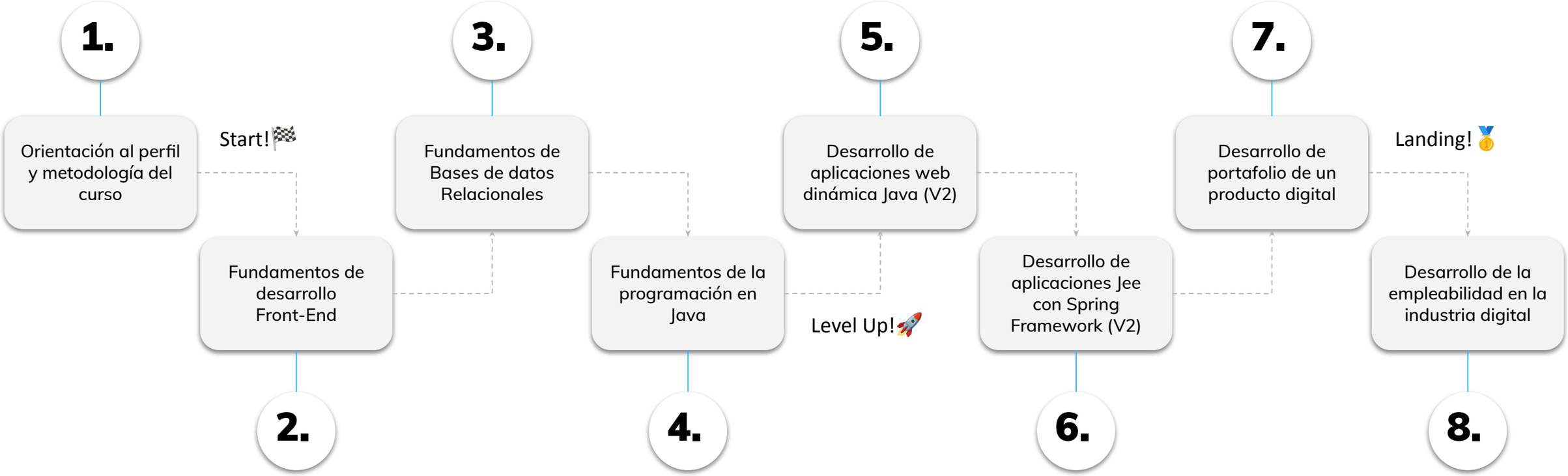
La interoperabilidad entre los sistemas

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 📚:

✓ *Aprender a crear una API REST con Spring MVC*

LEARNING PATHWAY

¿Sobre qué temas trabajaremos?

6.5 Start! 🏁

La interoperabilidad entre los sistemas

JWT

JWT

Securización de una API REST mediante JWT

English Always | alkemy

OBJETIVOS DE APRENDIZAJE

¿Qué aprenderemos?

- ***Securizar una API REST mediante JWT***

Securización de una API REST mediante JWT

Securización de una API REST mediante JWT

Una de las formas más utilizadas actualmente para securizar APIs REST es mediante la implementación de **JSON Web Tokens** (**JWT**). Los JWT permiten la autenticación stateless entre el cliente y el servidor, ya que encapsulan la información de acceso en un token firmado que se envía en cada request.

Securización de una API REST mediante JWT

Spring Framework provee el soporte necesario para incorporar JWT y seguridad basada en tokens dentro de una API REST creada con Spring MVC. Mediante la integración con Spring Security es posible agregar filtros de autenticación JWT, endpoints protegidos, roles y autorizaciones.

Securización de una API REST mediante JWT

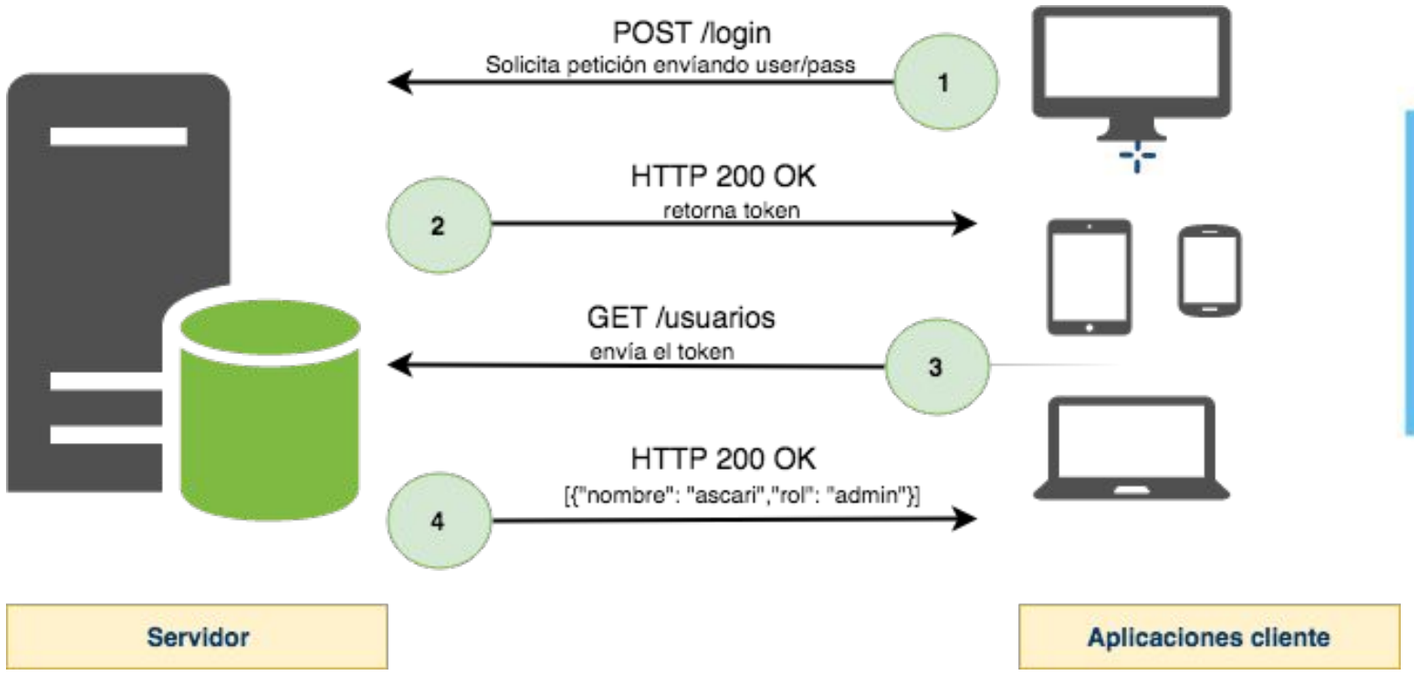
Un JWT contiene tres partes:
1- Header: algoritmo usado (HS256, RS256, etc) y tipo de token.
2- Payload: claims o afirmaciones sobre la entidad. Puede contener datos como usuario, roles, etc.
3- Firma: permite verificar la integridad del mensaje.

El JWT se codifica en Base64 y se envía en el header Authorization de cada request.
El servidor valida la firma para autenticar al cliente. Así se implementa la autenticación stateless.

Securización de una API REST mediante JWT

El siguiente diagrama muestra el flujo general de un proceso de autenticación basada en token.

1- El cliente envía sus credenciales (usuario y password) al servidor.
2- Si las credenciales son válidas, el servidor devuelve al cliente un token de acceso.
3- El cliente solicita un recurso protegido. En la petición, se envía el token de acceso.
4- El servidor valida el token y en caso de ser válido, devuelve el recurso solicitado.



Securización de una API REST mediante JWT

Veamos un ejemplo de un controlador REST para implementar el proceso de autenticación mediante un login usuario/contraseña securizado con JWT:
El método login(...) intercepta las peticiones POST al endpoint /user y recibirá como parámetros el usuario y contraseña.

```
@RestController
public class UserController {

    @PostMapping("user")
    public User login(@RequestParam("user") String username, @RequestParam("password") String pwd) {

        String token = getJWTToken(username);
        User user = new User();
        user.setUser(username);
        user.setToken(token);
        return user;

    }
```

Securización de una API REST mediante JWT

Utilizamos el método **getJWTToken**(...) para construir el token, delegando en la clase de utilidad Jwts que incluye información sobre su expiración y un objeto de GrantedAuthority de Spring que, como veremos más adelante, usaremos para autorizar las peticiones a los recursos protegidos.

```
private String getJWTToken(String username) {
    String secretKey = "mySecretKey";
    List<GrantedAuthority> grantedAuthorities = AuthorityUtils
            .commaSeparatedStringToAuthorityList("ROLE_USER");

    String token = Jwts
            .builder()
            .setId("softtekJWT")
            .setSubject(username)
            .claim("authorities",
                    grantedAuthorities.stream()
                            .map(GrantedAuthority::getAuthority)
                            .collect(Collectors.toList()))
            .setIssuedAt(new Date(System.currentTimeMillis()))
            .setExpiration(new Date(System.currentTimeMillis() + 600000))
            .signWith(SignatureAlgorithm.HS512,
                    secretKey.getBytes()).compact();

    return "Bearer " + token;
}
```

Securización de una API REST mediante JWT

Por último, editaremos nuestra clase de configuración de seguridad para añadir la siguiente configuración:

```
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.csrf().disable()
            .addFilterAfter(new JWTAuthorizationFilter(), UsernamePasswordAuthenticationFilter.class)
            .authorizeRequests()
            .antMatchers(HttpMethod.POST, "/user").permitAll()
            .anyRequest().authenticated();
    }
}
```

En este caso se permiten todas las llamadas al controlador /user, pero el resto de las llamadas requieren autenticación.

En este momento, si reiniciamos la aplicación y hacemos una llamada a http://localhost:8080/hello, nos devolverá un error 403 informando al usuario de que no está autorizado para acceder a ese recurso que se encuentra protegido:

Securización de una API REST mediante JWT

Ahora necesitamos implementar el proceso de autorización, que sea capaz de interceptar las invocaciones a recursos protegidos para recuperar el token y determinar si el cliente tiene permisos o no. Para ello implementaremos el siguiente filtro, JWTAuthorizationFilter:

```
23  public class JWTAuthorizationFilter extends OncePerRequestFilter {
24
25      private final String HEADER = "Authorization";
26      private final String PREFIX = "Bearer ";
27      private final String SECRET = "mySecretKey";
28
```

Securización de una API REST mediante JWT

Luego agregaremos los métodos: doFilterInternal y validateToken:

```
@Override
protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain chain)
    try {
        if (existeJWTToken(request, response)) {
            Claims claims = validateToken(request);
            if (claims.get("authorities") != null) {
                setUpSpringAuthentication(claims);
            } else {
                SecurityContextHolder.clearContext();
            }
        } else {
                SecurityContextHolder.clearContext();
        }
        chain.doFilter(request, response);
    } catch (ExpiredJwtException | UnsupportedJwtException | MalformedJwtException e) {
        response.setStatus(HttpServletResponse.SC_FORBIDDEN);
        ((HttpServletResponse) response).sendError(HttpServletResponse.SC_FORBIDDEN, e.getMessage());
        return;
    }
}

private Claims validateToken(HttpServletRequest request) {
    String jwtToken = request.getHeader(HEADER).replace(PREFIX, "");
    return Jwts.parser().setSigningKey(SECRET.getBytes()).parseClaimsJws(jwtToken).getBody();
}
```

Securización de una API REST mediante JWT

En la misma clase agregaremos los métodos necesarios para otorgar permisos

```
/**
 * Metodo para autenticarnos dentro del flujo de Spring
 *
 * @param claims
 */
private void setUpSpringAuthentication(Claims claims) {
    @SuppressWarnings("unchecked")
    List<String> authorities = (List) claims.get("authorities");

    UsernamePasswordAuthenticationToken auth = new UsernamePasswordAuthenticationToken(claims.getSubject(), null,
            authorities.stream().map(SimpleGrantedAuthority::new).collect(Collectors.toList()));
    SecurityContextHolder.getContext().setAuthentication(auth);

}

private boolean existeJWTToken(HttpServletRequest request, HttpServletResponse res) {
    String authenticationHeader = request.getHeader(HEADER);
    if (authenticationHeader == null || !authenticationHeader.startsWith(PREFIX))
        return false;
    return true;
}
```

Securización de una API REST mediante JWT

Este filtro intercepta todas las invocaciones al servidor (extiende de OncePerRequestFilter) y:

1. Comprueba la existencia del token (existeJWTToken(...)).
2. Si existe, lo desencripta y valida (validateToken(...)).
3. Si está todo OK, añade la configuración necesaria al contexto de Spring para autorizar la petición (setUpSpringAuthentication(...)).

Para este último punto, se hace uso del objeto **GrantedAuthority** que se incluyó en el token durante el proceso de autenticación.

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

Completamos la Wallet: Agregar securización con JWT al proyecto AlkeWallet

1- Agregar la dependencia necesaria para JWT

2- Agregar las declaraciones pertinentes en la clase de configuración

3- Crear la clase **JWTAuthorizationFilter** y hacerla extender de **OncePerRequestFilter**

4- Realizar las pruebas necesarias.

Momento:

Time-out!

⏰10 min.

¿Alguna consulta?

RESUMEN

¿Qué logramos en esta clase?

- ✓ *Aprender a securizar una API REST mediante JWT en Spring*

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 11 - 13*
3. Traer al próximo encuentro, todas tus dudas y consultas para verlas antes de iniciar nuevo tema.

¡Muchas Gracias!

Nos vemos en la próxima clase 👋🏻

English Always | alkemy