Json Web Token

JWT

¿Qué es un token?

Un token es una credencial digital que un servidor entrega para demostrar que un usuario ya fue autenticado y tiene ciertos permisos. Se usa en lugar de enviar usuario y contraseña en cada petición.

¿Qué es un token JWT?

Un token JWT es un tipo de token que contiene información en formato JSON y está firmado digitalmente. Permite verificar la identidad del usuario sin guardar sesiones en el servidor.

Token de autenticación

Características

- **Identifica** a un usuario (o cliente) en un servidor

- Su **autenticidad** es verificable
  - Significa que se debe poder comprobar que el token fue generado por el emisor

- Su **integridad** es verificable
  - Significa que el emisor comprueba que el token no fue adulterado de ninguna forma
- **Compacto**
  - El token se debe poder intercambiar entre el cliente y nuestro servidor sin que esto represente un problema de performance. Ej: Request HTTP
- **Expiración**
  - Opcional. Tiempo de expiración durante el cual es válido

JWT (Json Web Token)

- Es un estándar de la IETF (RFC 7519)
- Basado en JSON
- URL-safe
- Caso común: Manejo de Autenticación para clientes web y móviles

Estructura

Forma definida y estándar basada en tres partes requeridas en base64

```
<header>.<payload>.<signature>
```

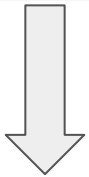
Ejemplo

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJ1c2VyIjoicGVwZSJ9.
G9Mf9kcAnXnrcwIyERksa5tg4js0bwui0TDU6Ozy_OE
```

Header

Contiene información del tipo de token

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9
```



```
{
   "alg": "HS256",
   "typ": "JWT"
}
```

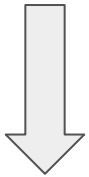
Payload

- Podemos enviar información en formato json dentro del token JWT.
- Dicha información debe ser acotada y por lo general es información referente al usuario (rol, permisos, nombre de usuario).
- Cuanto más información, más pesado será nuestro token
- Esta información estará “protegida” por las características mencionadas anteriormente (autenticidad e integridad)

Payload

Cuidado! El payload es legible

```
eyJ1c2VyIjoicGVwZSJ9
```



```
{
    "user": "pepe"
}
```

Signature (firma)

- El token está firmado criptográficamente.
- Esto le otorga las características anteriormente mencionadas

- La firma se genera de la siguiente forma:
  - Convertir a base64 el header
  - Convertir a base64 el payload
  - Un *secret* (un password que solo es conocido por nuestro API Server)
  - Aplicar el algoritmo del header sobre los 3 datos anteriores

Signature (firma). Ejemplo

- pseudo-código

```
function createJWT(header, payload) {
    var secret = "Top$ecr3t";

    var header64 = base64(header);
    var payload64 = base64(payload);
    var signature = base64(
        HMACSHA256(header64, payload64, secret)
    );

    return header64 + "." + payload64 + "." + signature;
}
```

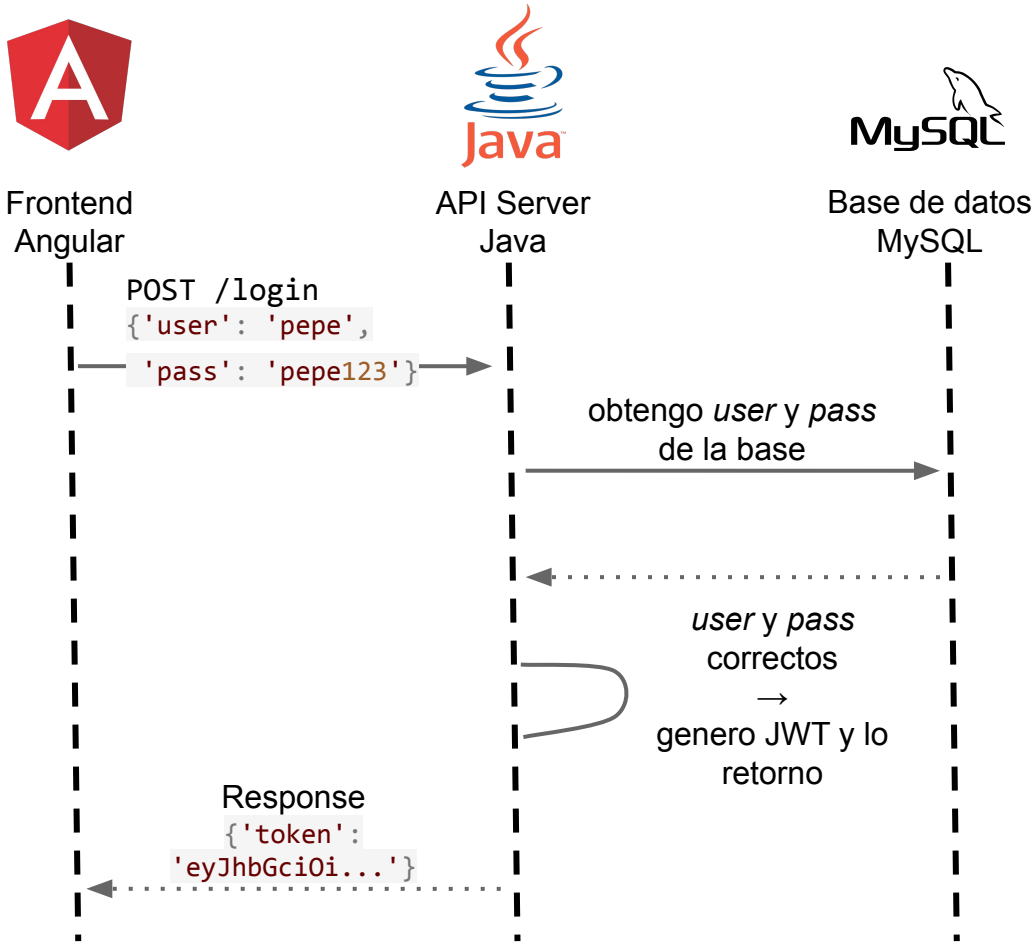
Ejemplo Teórico

Autenticación de un usuario

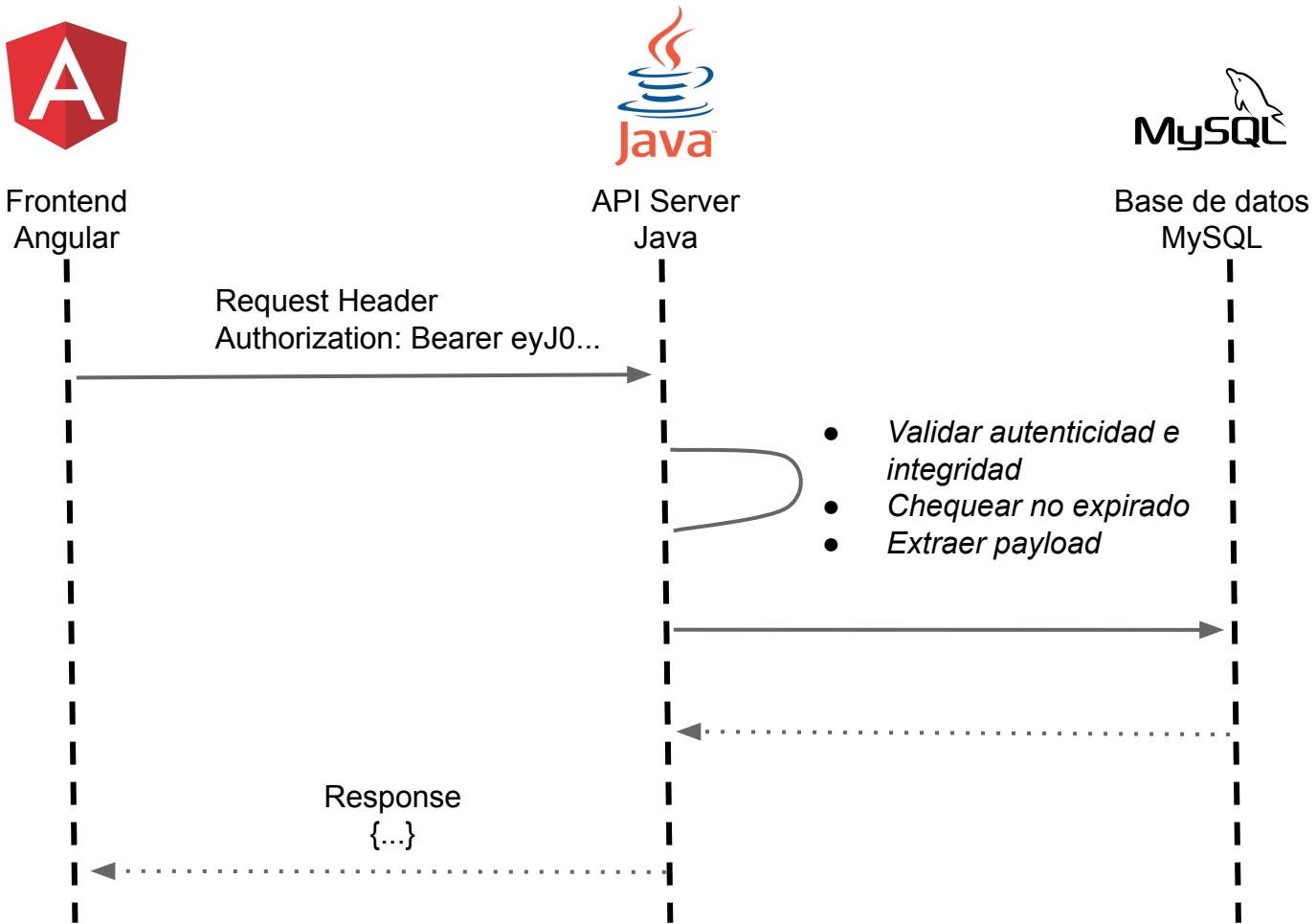
Cómo funciona?

- **Cliente** -> envía usuario y password a un web services
- **Backend** -> comprueba que usuario y password sean correcto
- **Backend** -> Genera y retorna Token JWT con información del usuario
- **Cliente** -> Almacena el token y lo envía en cada request

Diagrama de flujo de autenticación



Subsiguientes invocaciones



API Server

Como generar y validar tokens JWT en Java

API Server | Librerías para trabajar con JWT

Para crear y manipular los token podemos utilizar alguna de las librerías open-sources que nos van a facilitar el trabajo.

- **jjwt** ← ***Esta es la que vamos a usar***
- java-jwt
- nimbus-jose-jwt

Incluimos estas dependencias en el pom.xml

```
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt-api</artifactId>
    <version>0.12.6</version>
</dependency>
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt-impl</artifactId>
    <version>0.12.6</version>
    <scope>runtime</scope>
</dependency>
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt-jackson</artifactId>
    <version>0.12.6</version>
    <scope>runtime</scope>
</dependency>
```

API Server | Login RestController

```java
@RestController
public class LoginController {
    @Autowired
    private UsuarioDAO usuarioDAO;
    @Autowired
    private TokenServices tokenServices;

    private final int EXPIRATION_IN_SEC = 10; // 10 segs!!

    @PostMapping(path = "/auth")
    public ResponseEntity<?> authenticate(@RequestBody UsernameAndPassword userpass) {



        if(isLoginSuccess(userpass.getUsername(), userpass.getPassword())) {

            String token = tokenServices.generateToken(userpass.getUsername(), EXPIRATION_IN_SEC);
            return ResponseEntity.ok(new Credentials(token, EXPIRATION_IN_SEC, userpass.getUsername()));
        } else {
            return ResponseEntity.status(HttpStatus.UNAUTHORIZED).body("Usuario o password incorrecto");
        }
    }
```

Recibo user y pass del cliente

Si las credenciales son correctas, se genero el token

API Server | Token Services

```java
@Service
public class TokenServices {
    final static Key key = Keys.secretKeyFor(SignatureAlgorithm.HS256);
    /**
     * Genera el token de authorizacion para el usuario
     *
     * @param username Username que se guarda dentro del token
     * @param segundos tiempo de validez del token
     * @return token
     */
    public String generateToken(String username, int segundos) {
        Date exp = getExpiration(new Date(), segundos);

        return Jwts.builder().setSubject(username).signWith(key).setExpiration(exp).compact();
    }
```

Clave secreta

Lib JJWT

username en el payload

secret para la firma

API Server | Filter de Verificación

```java
@WebFilter(filterName = "jwt-auth-filter", urlPatterns = "*")
public class JWTAuthenticationFilter implements Filter {

    @Override
    public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws
IOException, ServletException {
        HttpServletRequest req = (HttpServletRequest) request;
        // El login del usuarios es publico
        if ("/jwt/auth".equals(req.getRequestURI()) ||
                HttpMethod.OPTIONS.matches(req.getMethod())) {
            chain.doFilter(request, response);
            return;
        }
        String token = req.getHeader(HttpHeaders.AUTHORIZATION);
        if (token == null || !TokenServices.validateToken(token)) {
            HttpServletResponse res = (HttpServletResponse) response;
            res.setStatus(HttpStatus.FORBIDDEN.value());
            return;
        }
        chain.doFilter(request, response);
    }
```

Intercepta todos los request

Permite acceso al login

Obtiene el token desde un header

Si no tiene token o no es válido
retorno 403 (Forbidden)

API Server | Validación de Token

```java
@Service
public class TokenServices {
...
   public static boolean validateToken(String token) {
       String prefix = "Bearer";
       try {
           if (token.startsWith(prefix)) {
               token = token.substring(prefix.length()).trim();
           }
           Claims claims = Jwts.parser().setSigningKey(key).parseClaimsJws(token).getBody();

           System.out.println("Subject: " + claims.getSubject());
           System.out.println("Expiration: " + claims.getExpiration());
           return true;
       } catch (ExpiredJwtException exp) {
           return false;
       } catch (JwtException e) {
           return false; // Algo salio mal en la verificacion
       }
   }
```

Verifica y obtiene el payload

Token expirado. Acceso denegado

Token corrupto. Acceso Denegado

Configuración en el Frontend

Cómo trabajar con tokens JWT en Angular

Uso de interceptores

Interceptores

- Los interceptores **inspeccionan y transforman peticiones HTTP** que van desde la aplicación al servidor, como también las respuestas que vuelven del servidor a la aplicación.
- Se pueden usar múltiples interceptores *formando una cadena*.
- Los interceptores pueden ejecutar variedad de tareas, como autenticación o logging.
- Sin interceptores los desarrolladores deberían implementar estas tareas explícitamente para cada llamada a HttpClient
- Para implementar un interceptor, se declara una clase que implementa el método intercept() de la interface HttpInterceptor

Interceptores

```
import { HttpInterceptor, HttpSentEvent,
HttpHeaderResponse, HttpHandler, HttpEvent,
HttpRequest, HttpHeaders, HttpClient,
HttpErrorResponse } from "@angular/common/http";
import { Observable } from "rxjs/Observable";
import { Injectable } from "@angular/core";
import { environment
} from "../../../environments/environment";
import { tap, catchError } from "rxjs/operators";
import { ErrorObservable } from
"rxjs/observable/ErrorObservable";

@Injectable()
export class TokenInterceptor implements HttpInterceptor {

 constructor(private http: HttpClient){
 }
```

```
intercept(req: HttpRequest<any>, next: HttpHandler) {
  console.log(`TokenInterceptor - ${req.url}`);

  let authReq: HttpRequest<any> = req.clone({
    setHeaders:{
      Authorization : `Bearer ${localStorage.getItem("token")}`
    }
  });
  return next.handle(authReq);  //para no hacer cambios
                                //podría enviar next.handle(req)
 }
}
```

Interceptores

Los interceptores son dependencias opcionales del servicio HttpClient, deben estar declarados en el módulo donde se importa HttpClientModule

```
import { HttpClientModule, HTTP_INTERCEPTORS } from '@angular/common/http';
import { TokenInterceptor} from './interceptor.service'

@NgModule({
...
  imports: [
    ... ,
    HttpClientModule
  ],

  providers: [
    {provide:HTTP_INTERCEPTORS, useClass: TokenInterceptor, multi:true}
    //  mas interceptores
    // ,{provide:HTTP_INTERCEPTORS,useClass: OtroInterceptor, multi:true} ...
  ],
...
```

Angular aplica los interceptores en el mismo orden que se declaran en el arreglo de providers

Ejemplo Práctico y Demo

- Repositorio en github con los fuentes y las instrucciones
  - https://github.com/ortizman/ttps-jwt

- Demo
  - Backend Java usando la lib JJWT
  - Frontend Angular usando interceptores para enviar el Token