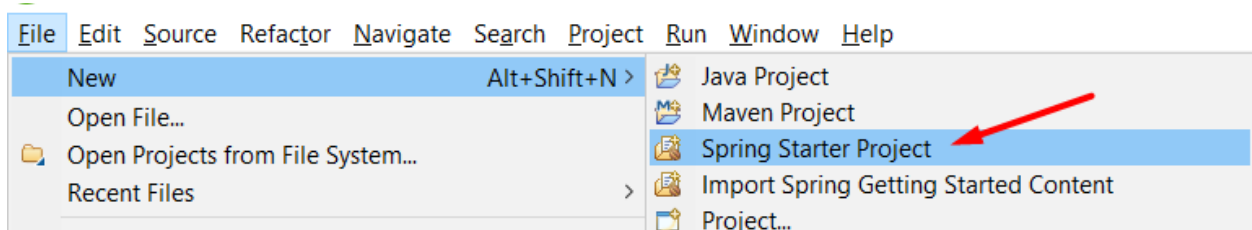


# Proyecto "JWT-TOKEN"

## Paso 1: Crear el Proyecto en STS

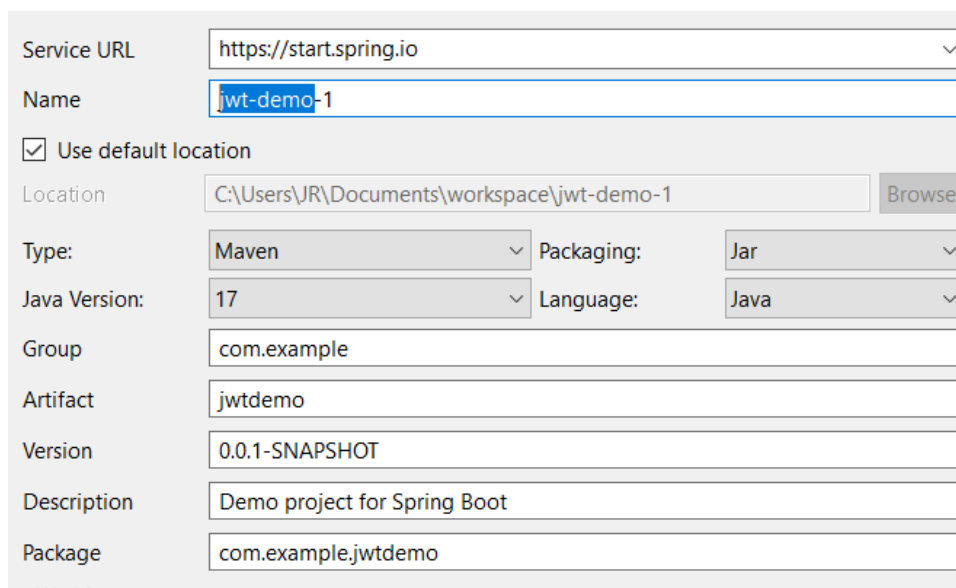
Abrir Spring Tool Suite (STS).

Ir a **File → New → Spring Starter Project**



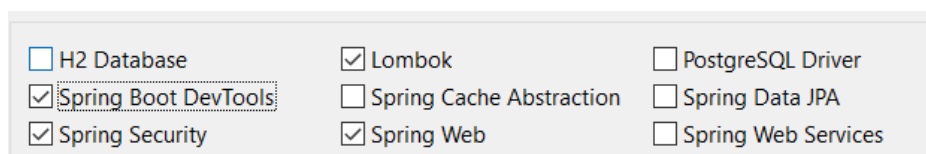
Completar los datos del proyecto:

- **Name:** jwt-demo
- **Type:** Maven
- **Java Version:** 17
- **Group:** com.example
- **Artifact:** jwtdemo
- **Package:** com.example.jwtdemo
- **Packaging:** Jar



Seleccionar las dependencias necesarias:

- **Spring Web**
- **Spring Security**
- **Lombok**
- **Spring Boot DevTools (opcional)**



También agregar las siguientes librerías para el JWT.



```
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt-api</artifactId>
    <version>0.11.5</version>
</dependency>
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt-impl</artifactId>
    <version>0.11.5</version>
    <scope>runtime</scope>
</dependency>
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt-jackson</artifactId>
    <version>0.11.5</version>
    <scope>runtime</scope>
</dependency>
```

Hacer clic en **Finish** para generar el proyecto.

## Paso 2: Clase de Utilidad para JWT

Creamos la clase Util de JwtUtil



```
package com.example.jwtdemo.util;

import io.jsonwebtoken.*;
import io.jsonwebtoken.security.Keys;
import org.springframework.stereotype.Component;

import java.security.Key;
import java.util.Date;

@Component
public class JwtUtil {

    private final Key key = Keys.secretKeyFor(SignatureAlgorithm.HS256);

    public String generateToken(String username) {
        return Jwts.builder()
            .setSubject(username)
            .setIssuer("demo-app")
            .setIssuedAt(new Date())
            .setExpiration(new Date(System.currentTimeMillis() + 10_000)) // 10 segundos
            .signWith(key)
            .compact();
    }

    public boolean validateToken(String token) {
        try {
            Jwts.parserBuilder().setSigningKey(key).build().parseClaimsJws(token);
            return true;
        } catch (JwtException e) {
            return false;
        }
    }

    public String extractUsername(String token) {
        return Jwts.parserBuilder().setSigningKey(key).build()
            .parseClaimsJws(token).getBody().getSubject();
    }
}
```

Explicación:



@Component

Marca esta clase como un componente Spring (para inyección de dependencias).



```
private final Key key = Keys.secretKeyFor(SignatureAlgorithm.HS256);
```

Genera una clave secreta para firmar tokens usando algoritmo HS256.

### Método generateToken:



```
public String generateToken(String username) {  
    return Jwts.builder()  
        .setSubject(username)  
        .setIssuer("demo-app")  
        .setIssuedAt(new Date())  
        .setExpiration(new Date(System.currentTimeMillis() + 10_000)) // 10 segundos  
        .signWith(key)  
        .compact();  
}
```

- **return Jwts.builder():** Crea un constructor de tokens JWT.
- **.setSubject(username):** Establece el sujeto (usuario) del token.
- **.setIssuer("demo-app"):** Establece el emisor del token.
- **.setIssuedAt(new Date()):** Establece la fecha de emisión (ahora).
- **.setExpiration(new Date(System.currentTimeMillis() + 10\_000)):** Establece la expiración (10 segundos después de ahora).
- **.signWith(key):** Firma el token con la clave secreta.
- **.compact():** Genera el token como cadena compacta.

### Método validateToken:



```
try {  
    Jwts.parserBuilder().setSigningKey(key).build().parseClaimsJws(token);  
    return true;  
} catch (JwtException e) {  
    return false;  
}
```

Intenta parsear el token con la clave, retorna true si es válido, false si hay error.

### Método extractUsername:



```
public String extractUsername(String token) {  
    return Jwts.parserBuilder().setSigningKey(key).build()  
        .parseClaimsJws(token).getBody().getSubject();  
}
```

Parsea el token y extrae el sujeto (nombre de usuario).

## Paso 3: Crear en el paquete Dto la clase AuthRequest

Creamos la clase Dto de AuthRequest



```
package com.example.jwtdemo.dto;  
  
import lombok.Getter;  
import lombok.NoArgsConstructor;  
import lombok.Setter;  
  
@Getter  
@Setter  
@NoArgsConstructor  
public class AuthRequest {  
    private String username;  
    private String password;  
}
```

Explicación:



```
private String username;  
private String password;
```

Campos que representan las credenciales del usuario.

## Paso 4: Crear en el paquete controller la clase AuthController

Creamos la clase Controller de AuthController



```
package com.example.jwtdemo.controller;

import com.example.jwtdemo.dto.AuthRequest;
import com.example.jwtdemo.util.JwtUtil;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/auth")
public class AuthController {

    private final JwtUtil jwtUtil;

    public AuthController(JwtUtil jwtUtil) {
        this.jwtUtil = jwtUtil;
    }

    @PostMapping("/login")
    public String login(@RequestBody AuthRequest request) {
        // Autenticación básica simulada (usuario: user, pass: 1234)
        if ("user".equals(request.getUsername()) && "1234".equals(request.getPassword())) {
            return jwtUtil.generateToken(request.getUsername());
        } else {
            return "Credenciales inválidas";
        }
    }

    @GetMapping("/hello")
    public String hello(@RequestHeader("Authorization") String authHeader) {
        String token = authHeader.replace("Bearer ", "");
        if (jwtUtil.validateToken(token)) {
            String user = jwtUtil.extractUsername(token);
            return "Hola, " + user + "! Acceso autorizado.";
        }
        return "Token inválido o expirado.";
    }
}
```

Explicación:



```
private final JwtUtil jwtUtil;
```

Declara la dependencia de JwtUtil.



```
public AuthController(JwtUtil jwtUtil) {  
    this.jwtUtil = jwtUtil;  
}
```

Constructor con inyección de dependencia de JwtUtil.



```
@PostMapping("/login")  
  
public String login(@RequestBody AuthRequest request) {  
    // Autenticación básica simulada (usuario: user, pass: 1234)  
    if ("user".equals(request.getUsername()) && "1234".equals(request.getPassword())) {  
        return jwtUtil.generateToken(request.getUsername());  
    } else {  
        return "Credenciales inválidas";  
    }  
}
```

- Lógica de autenticación básica (usuario: user, contraseña: 1234).
- Si es correcto, genera y retorna un token JWT.



```
@GetMapping("/hello")  
  
public String hello(@RequestHeader("Authorization") String authHeader) {  
    String token = authHeader.replace("Bearer ", "");  
    if (jwtUtil.validateToken(token)) {  
        String user = jwtUtil.extractUsername(token);  
        return "Hola, " + user + "! Acceso autorizado.";  
    }  
    return "Token inválido o expirado.";  
}
```

- Valida el token y si es válido, extrae el usuario y retorna mensaje personalizado.
- Mensaje si el token no es válido.

## Paso 5: Crear en el paquete Config la clase SecurityConfig

Creamos la clase Config de SecurityConfig



```
package com.example.jwtdemo.config;

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.web.SecurityFilterChain;

@Configuration
public class SecurityConfig {
    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        http.csrf(csrf -> csrf.disable())
            .authorizeHttpRequests(auth -> auth.anyRequest().permitAll());
        return http.build();
    }
}
```

Explicación:



```
http.csrf(csrf -> csrf.disable())
```

- Deshabilita la protección CSRF (Cross-Site Request Forgery).



```
.authorizeHttpRequests(auth -> auth.anyRequest().permitAll());
```

- Configura la autorización para permitir todas las solicitudes sin autenticación.



```
return http.build();
```

- Construye y retorna la configuración de seguridad.

## Paso 6: Ejecutar los endpoints en Postman

POST <http://localhost:8080/auth/login>



```
{
  "username": "user",
  "password": "1234"
}
```



POST http://localhost:8080/auth/login

Params Authorization Headers (8) Body Scripts Settings Cookies Beautify

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 {
2   "username": "user",
3   "password": "1234"
4 }
5
```

Body Cookies Headers (11) Test Results 200 OK 6 ms 487 B Save Response

Raw Preview Visualize

```
1 eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJ1c2VyIiwiaXNzIjoie3N0M3NTAzODQsImV4cCI6MTc0Mzc1MDM5NH0.6VEdZcNF6MIE3_755uqPM43NRgfJMMZSouxV77CYhBk
```

GET http://localhost:8080/auth/hello

POST http://localhost:8080/crypto/aes/encrypt

Params Authorization Headers (8) Body Scripts Settings Cookies Beautify

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 {
2   "message": "Hola desde AES"
3 }
4
```

Body Cookies Headers (5) Test Results 200 OK 63 ms 188 B Save Response

Raw Preview Visualize

```
1 L2gdxg0LF089w+WBtwXT0g==
```

GET http://localhost:8080/auth/hello

GET http://localhost:8080/auth/hello

Params Authorization Headers (7) Body Scripts Settings Cookies

Auth Type: Bearer Token

Token: eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJ1c2VyIiwiaXNzIjoie3N0M3NTAzODQsImV4cCI6MTc0Mzc1MDM5NH0.6VEdZcNF6MIE3\_755uqPM43NRgfJMMZSouxV77CYhBk

The authorization header will be automatically generated when you send the request. Learn more about [Bearer Token](#) authorization.

Body Cookies Headers (11) Test Results 200 OK 26 ms 364 B Save Response

Raw Preview Visualize

```
1 Hola, user! Acceso autorizado.
```