Spring Security

NOSERYOUNG

Spring Security

What is it?

- Spring Security is a Spring module that provides a **framework for security on a webpage- or app-backend**
- Functionality for both **Authentication and Authorization**
- **Protection against attacks** like session fixation, cross site request forgery etc.

NOSERYOUNG

Spring Security

Setup

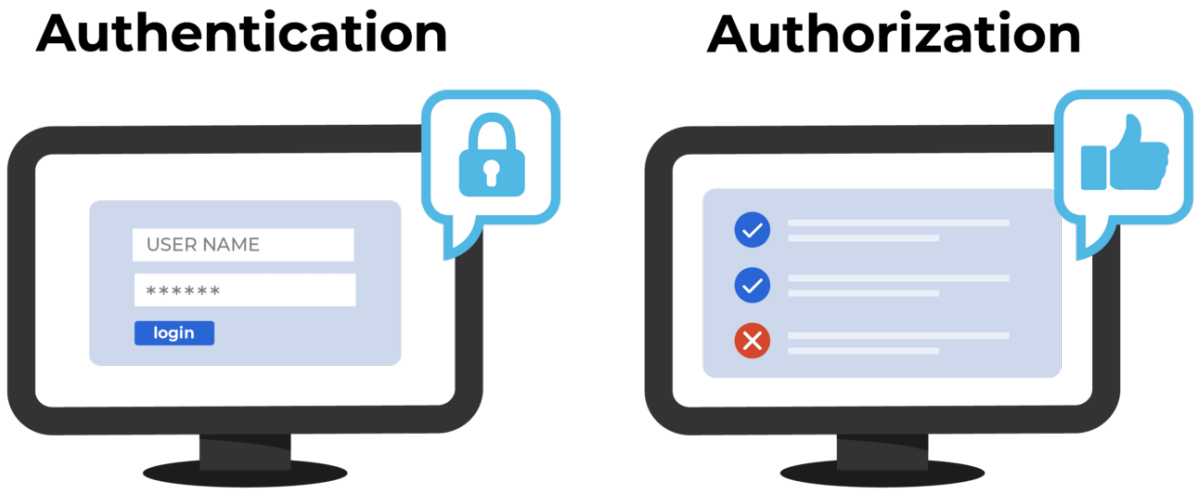
- Add to build.gradle:
  - implementation ‘org.springframework.boot:spring-boot-starter-security’
- Implement a security configuration class
- For advanced security, user and role classes.

NOSERYOUNG

Authorization vs. Authentication

- **Authentication** Confirms users are who tey say they are.

- **Authorization**: Confirms a user has permisson to acces a resource.

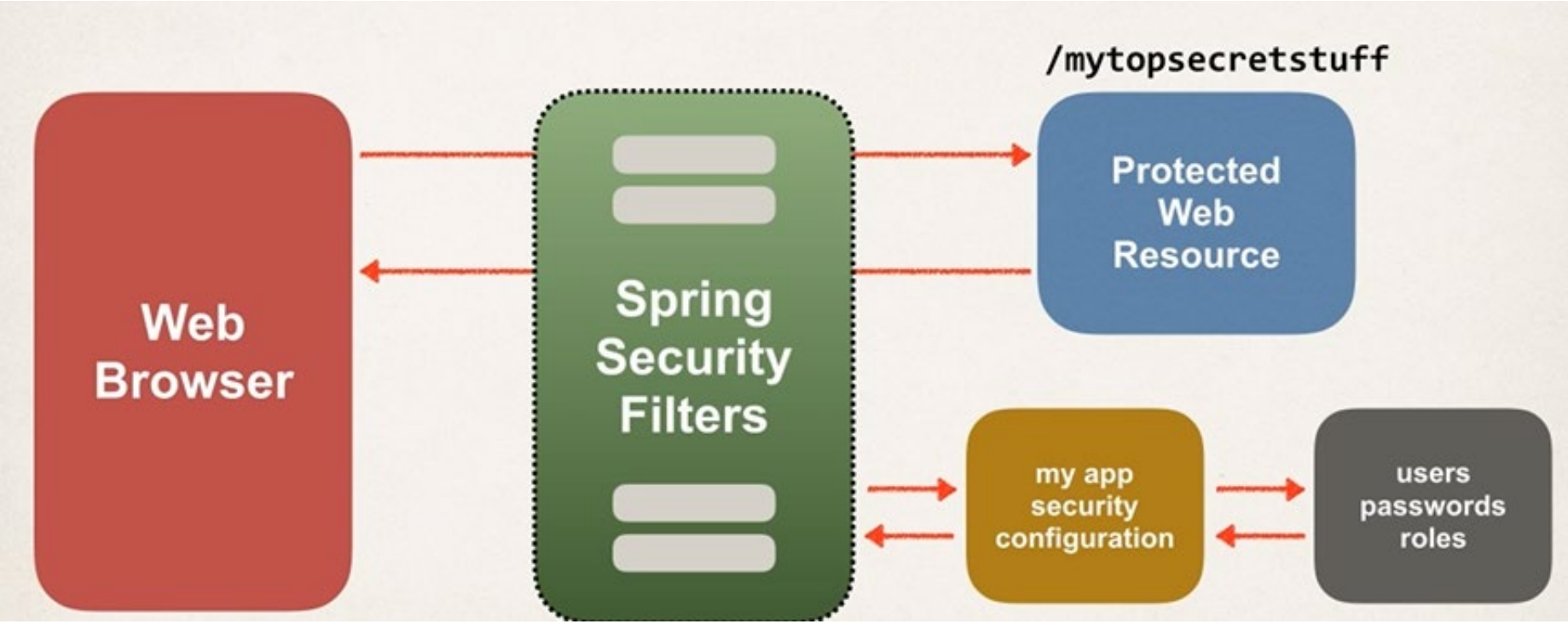


NOSERYOUNG

Spring Security

How does it work?

- Spring securtiy is implemented with **servlet filters:**

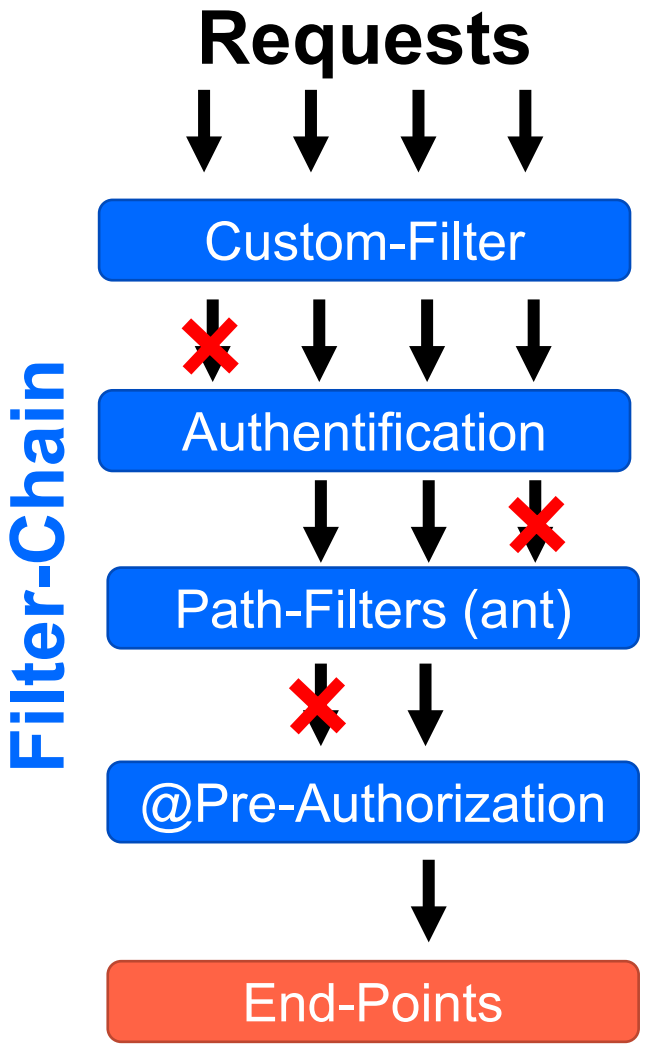


NOSERYOUNG

Filters

- **Idea**: Chain Filters on Requests to achieve desired security strategy
- Filterchain is configured using `WebSecurityConfigurerAdapter.configure(http)`
- Custom Filters can be inserted in Filter chain

```
http.addFilterBefore( new CustomFilter(),
BasicAuthenticationFilter.class);
```



NOSERYOUNG

Roles & Privileges

In Spring Boot

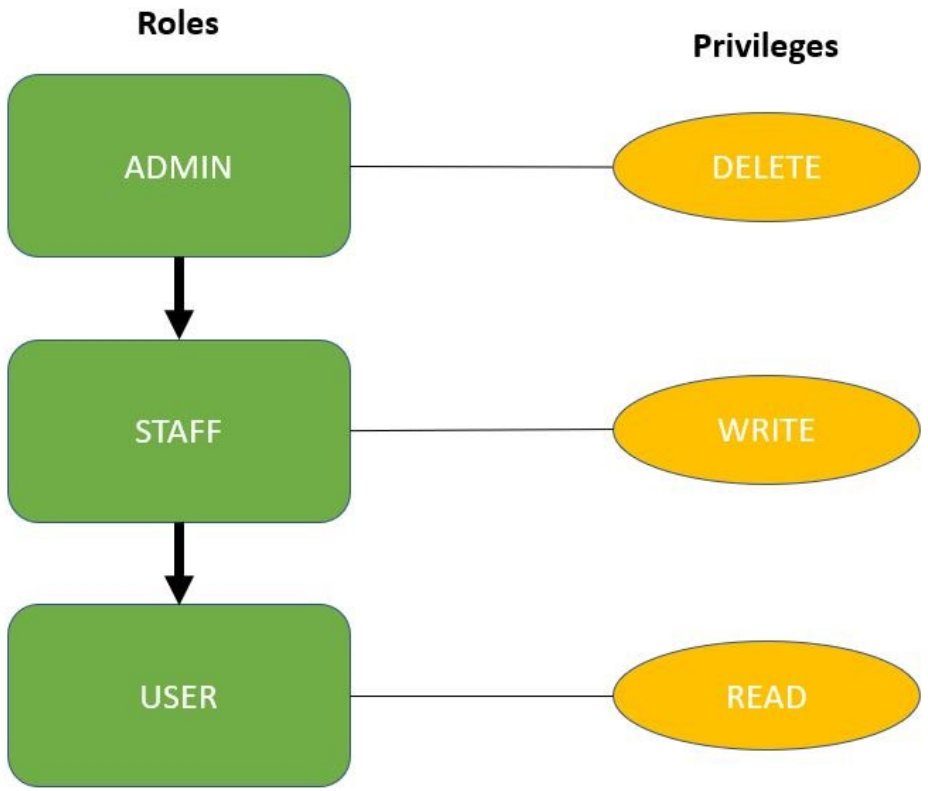
- We have three main entities:
- the ***User*** – might have multiple Roles
- the ***Role*** – this represents the **high-leve**l roles of the user in the system; each role will have a set of low-level privileges
  - *E.g. hasRole("ADMIN")*
- the ***Privilege/Authority*** – represents a **low-level**, granular privilege/authority in the system
  - *e.g. hasAuthority('READ_REPORT')*

Role Hierarchies

optional

- Reduces number of Roles a User needs.
- E.g. The Role *ADMIN* might automatically grant all rights of the role *STAFF* and *USER*

- Simple implementation using Sprig Bean `roleHierarchy`



NOSERYOUNG

Most basic example of Security Config

Without User & Role Entities

- In-Memory authentication for setup and testing, not real applications!

```
 @EnableWebSecurity
public class SecurityConfig extends
WebSecurityConfigurerAdapter {

    @Autowired
    public void configureGlobal
(AuthenticationManagerBuilder auth) throws Exception {

          auth.inMemoryAuthentication().withUser("user")
.password(passwordEncoder().encode("password")).roles(
"USER");
    }
```

NOSERYOUNG

Classes/Components

With User & Role Entities

- **User Entity**: Manage user information in DB (saving, retrieving etc.)
- **Role Entity:** Sets of Permissions / Authorities
- **Authority Entity**
- **UserDetailsService:** Gives Spring Beans access to user information
- **WebSecurityConfigurerAdapter:** Manages acces restrictions based on roles and defines general security strategy.

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

      private final UserDetailsService
userDetailsService;
      private final PasswordEncoder
passwordEncoder;

@Override
      protected void configure(HttpSecurity
http) throws Exception {
         http.httpBasic().and()
               .authorizeRequests()
               .antMatchers("/", "/home")
            .hasAnyRole("ADMIN,,,,USER")
      } }
```

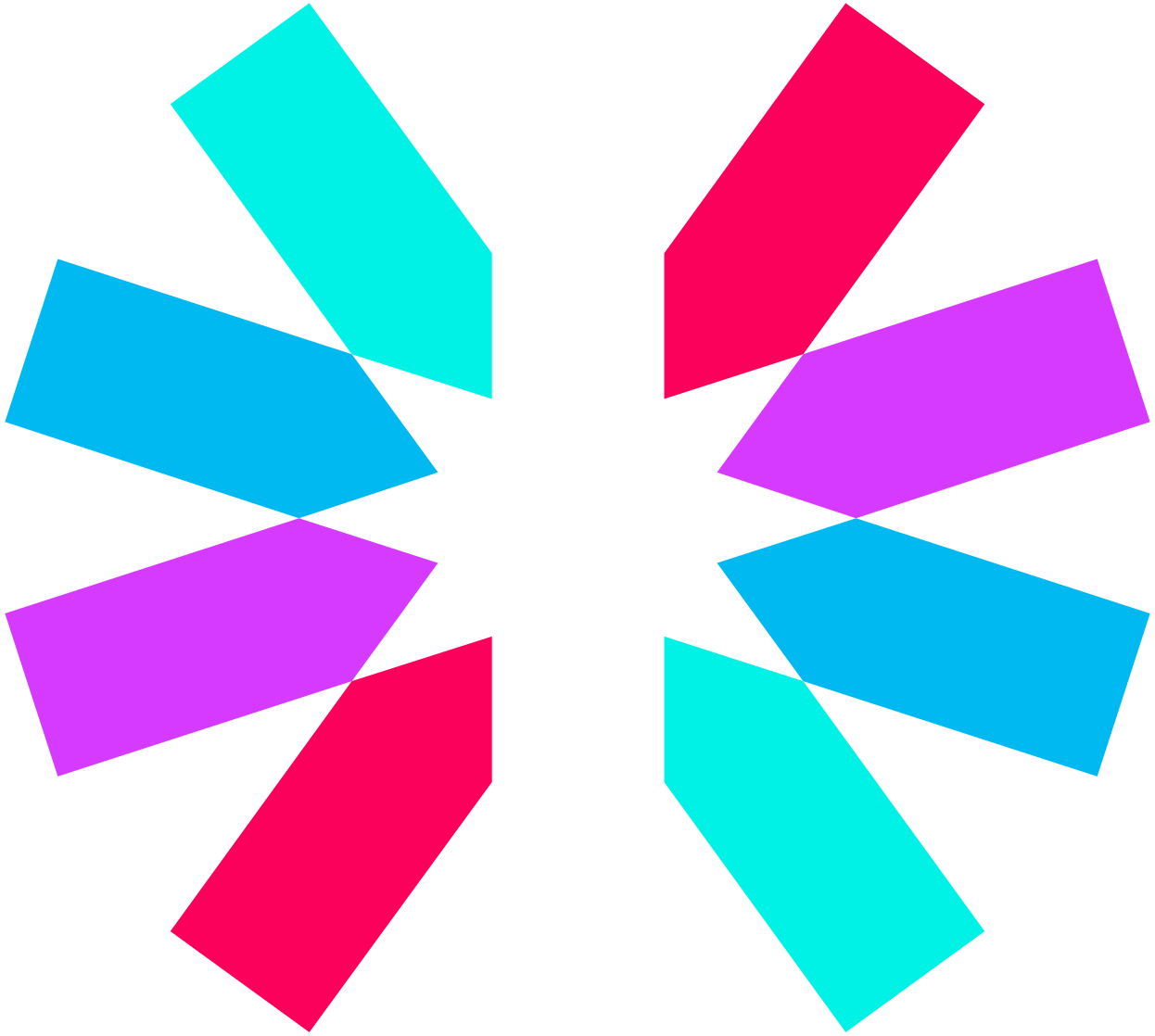
NOSERYOUNG

Securing Site using annotations

- On an Endpoint level
  - @RolesAllowed (*List of Roles*)
  - **@Pre-/PostAuthorize**(*SpEL statement*) using Spring Expression Language
- Globally
  - **antMatchers** in SecurityConfig Class

```
@GetMapping({"/{userId}/educationalReports"})
@PreAuthorize(„
(hasAuthority('READ_EDUCATIONALREPORT ‘)
&&
@userPermissionEvaluator.isUserInSameCompany
(authentication.principal.username, #userId))
||
hasAuthority('READ_EDUCATIONALREPORT‘)“)
public
ResponseEntity<List<EducationalReportEntryDTO>>
getAllEducationalEntries(@PathVariable("userId")
...
```

NOSERYOUNG



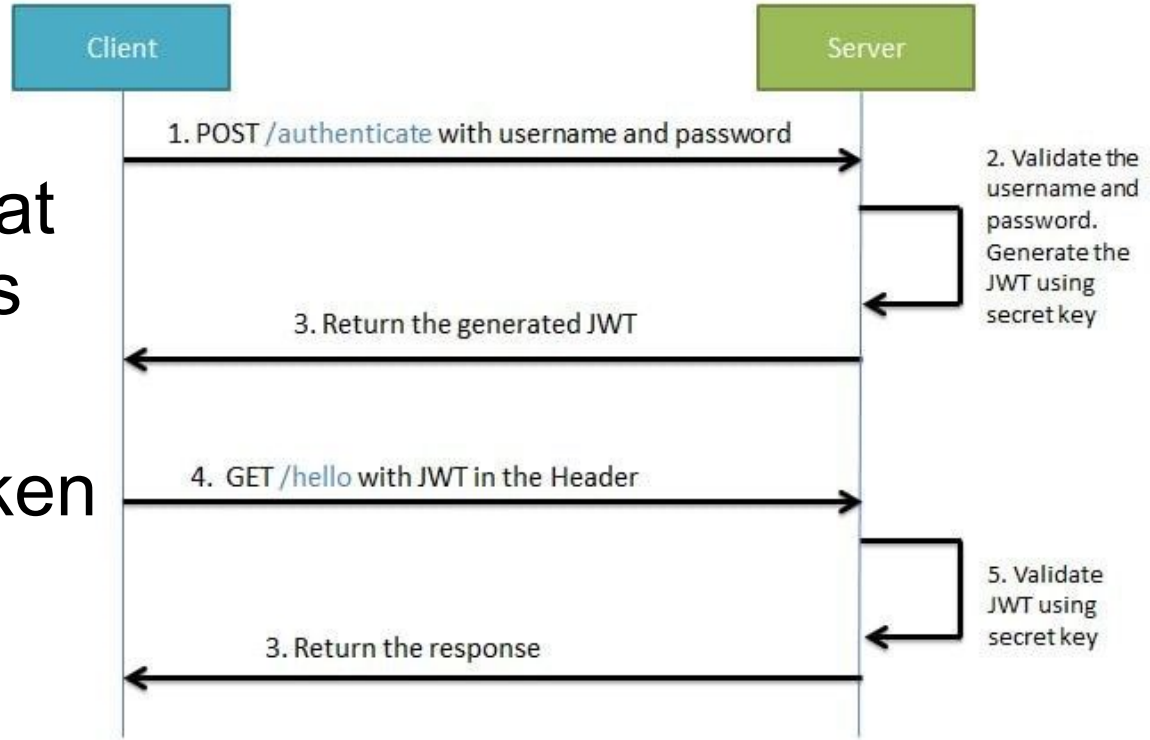
JSON Web Token

NOSERYOUNG

JSON Web Token

What is it?

- **Idea**: Send credentials in a way that can be verified by both participants
- Header + Payload + Signature
  - Header: Meta-info about the token
  - Payload: Data
  - Signature: Hashed version of Header & Payload, generated using a secret key only known to Server & Client



NOSERYOUNG

JSON Web Token

What we need

- **JWT Utility class**: handles reading, authenticating, constructing JWTs using dependency com.auth0:java-jwt
- An Endpoint for users to Authenticate and **get a new JWT**
- **A Filter** that checks the JWT in each request and fetches the corresponding user data for **the principal**

Side Note:

"The principal" refers to the currently logged in user, or the user sending a request.

Access it with :

```
SecurityContextHolder.getContext().getAuthentication()
```

NOSERYOUNG

Further Reading

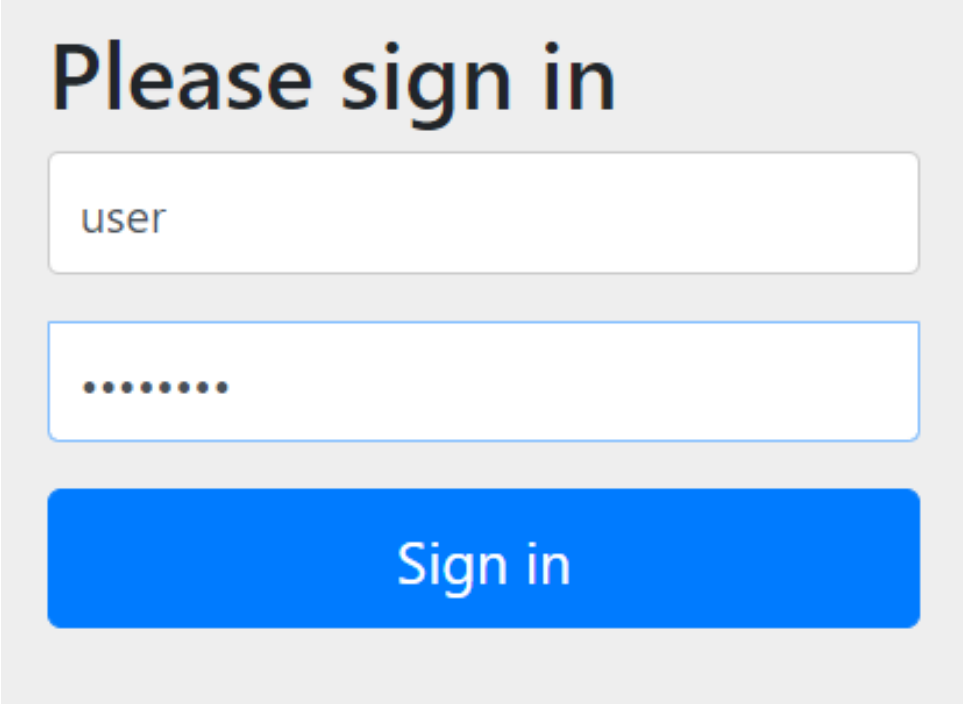
- Token visualization: https://jwt.io/
- Information: https://www.baeldung.com/java-json-web-tokens-jjwt
- Tutorial: https://www.youtube.com/watch?v=VVn9OG9nfH0&t=2301s

To-Do’s

- Include Spring Security dependency:

```
implementation  'org.springframework.boot:
spring-boot-starter-security'
```

- Configure secured and public routes using the *WebSecurityConfigurerAdapter* Class



NOSERYOUNG

Further reading

- Documentation: https://docs.spring.io/spring-security/site/docs/current/reference/html5/
- Example applications:
  - https://github.com/spring-projects/spring-security-samples
  - https://www.baeldung.com/security-spring
  - https://www.baeldung.com/role-and-privilege-for-spring-security-registration