Segurança

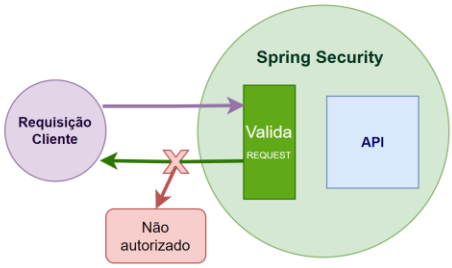
É n**ecessário que API's tenham uma camada de segurança** que restringe o **acesso aos Endpoints para quem vai consumi-la**, afinal de contas não quero que todo mundo possa manipular qualquer recurso sem controle nenhum.

Para esse tipo de situação o **Spring Boot** oferece uma solução chamada **Spring Security**. É ridiculamente fácil **habilitar segurança na API** com esse módulo.

Basta **adicionar na dependência do projeto**, **anotar** uma **classe** **de segurança** com **@EnableWebSecurity** e **@Configuration** e fazer essa **classe** **herdar** de **WebSecurityConfigurerAdapter**. Só de fazer isso eu já **protejo os endpoints de todo o acesso não autorizado:**

```xml
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>
```

```java
@EnableWebSecurity
@Configuration
public class SecurityConfigurations extends WebSecurityConfigurerAdapter {
}
```



"configure" herdados

A classe contém alguns **métodos de configuração de segurança herdados** que precisam ser sobrescritos, esses métodos tem 3 sobrecargas diferentes. Ou seja, recebe parâmetros diferentes.

```java
@EnableWebSecurity
@Configuration
public class SecurityConfigurations extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {

    }

    @Override
    protected void configure(HttpSecurity http) throws Exception {

    }

    @Override
    public void configure(WebSecurity web) throws Exception {

    }
```

O configure que recebe AuthenticationManagerBuilder: É uma configuração responsável por autenticação do usuário no sistema.

O configure que recebe HttpSecurity: É uma configuração responsável por **autorização de request em endpoints**, quem pode acessar qual url e etc..

O configure que recebe WebSecurity: É a configuração para recursos estáticos como JS, imagens e etc... caso o frontend fosse integrado.