

Índice

Spring security	1
Filtros para autorização/ Dependência Spring Web	2
Token	4
HTTP Authentication	5
Criação security	6
Links e afins	6
Dúvidas	7

Assuntos:

Abertura Alunos:

Apresentação pitch
Matheus Libanio.

Spring security

Spring Security é um framework para Java, que provê autenticação, autorização, filtros de servlet e diversas outras funcionalidades para aplicações corporativas, com o objetivo de proteger a aplicação contra acessos indevidos.

A Dependência `spring-boot-starter-security` é responsável por todas as dependências relacionadas à segurança do Spring. Dentro desta dependência, existem outras 3 dependências:

spring-security-core: Implementa os principais recursos do Spring Security;


spring-security-config: Fornece o namespace (contexto) e configurações Spring Security;

spring-security-web: fornece filtros e outros recursos necessários para proteger aplicativos da web.

Essas 3 dependências devem ser adicionadas no POM do código, e são essas 3 dependências que vão configurar e definir os padrões que vamos requerer do usuário. Ex: quantas letras, se vai ser número, se vai ser email, etc...

Depois de inserir a Dependência spring-boot-starter-security, o Spring habilitará alguns padrões de configuração de segurança automática, incluindo a segurança de todos os endpoints, bem como a definição da estratégia de autenticação no formato http Basic (Autenticação básica através de usuário e senha).

```
28<  <dependency>
29    <groupId>org.springframework.boot</groupId>
30    <artifactId>spring-boot-starter-web</artifactId>
31  </dependency>
32  <!-- Dependência Spring Security -->
33<  <dependency>
34    <groupId>org.springframework.boot</groupId>
35    <artifactId>spring-boot-starter-security</artifactId>
36  </dependency>
37<  <dependency>
38    <groupId>org.springframework.boot</groupId>
39    <artifactId>spring-boot-devtools</artifactId>
40    <scope>runtime</scope>
41    <optional>true</optional>
42  </dependency>
```



É uma função que cria usuário e senha para editar funções e habilitar padrões de segurança, endpoints, etc, definindo uma estratégia para autenticar meu usuário e desenvolver um formato para autenticar o usuário que se chama HTTP Basic.

- **Autenticação** é o processo de reconhecimento de usuário, processo de login e senha.
 - Valido o meu usuário.
- **Autorização** é o processo dentro da segurança da informação que vai ser definido direitos e necessidades do usuário que vai ter no sistema, que possuem o nome **ROLES**.
 - O que pode e não pode dentro do sistema.

httpBasic : Autenticação básica através de usuário e senha.

Filtros para autorização/ Dependência Spring Web

Qualquer aplicativo dentro da minha Spring Web é considerado uma Servlet que é uma classe em Java que redireciona as REQUEST HTTP para minha classe controller específica. E retornará uma RESPONSE.



usuário
admin@email.com.br

senha
••••••••

LOGAR

UsernamePasswordAuthenticationFilter



OK - 200 - Acesso Autorizado!

Nos serviços REST é altamente recomendado enviar a resposta **HTTP Status 401 - Unauthorized**, quando a solicitação vier sem autenticação, em vez de redirecionar para a página de login gerada por padrão para obter a autenticação. Esteja ciente de que, nesse caso, o **Status 401 - Unauthorized** significa, não autenticado.

Token

Bearer eyJhbGciOiJIUzI1NiJ9.eyJzdWUiOiJhZG1pbkBlbWVpbC5jb20uYnliLCJpYXQiOiJlE2Nzc1NTY2MTcslmV4cCI6MTY3NzU2MDIxN30.6p1towU5LlmsMZdIJMDCMf8oPwB1ab48jni2FKnZaso

É um dado combinado aleatório combinado com meu security server, torna-se parte vital da segurança de seu aplicativo.

Bearer é o formato do token que será passada na aplicação.

O token é enviado pelo cliente em solicitações subsequentes para permitir o acesso protegido, sendo validado pelo servidor para garantir a autorização adequada. O uso de tokens é comum em sistemas web para manter a segurança e o estado de autenticação.

Esquema Bearer

A estrutura geral de autenticação HTTP é usada por vários esquemas de autenticação. Os esquemas podem divergir na força da segurança e na disponibilidade do software cliente ou servidor. Os esquemas mais comuns de autenticação são o Basic e o Bearer, mas existem outros esquemas oferecidos por serviços de hospedagem, como AWS, Google ou Microsoft.

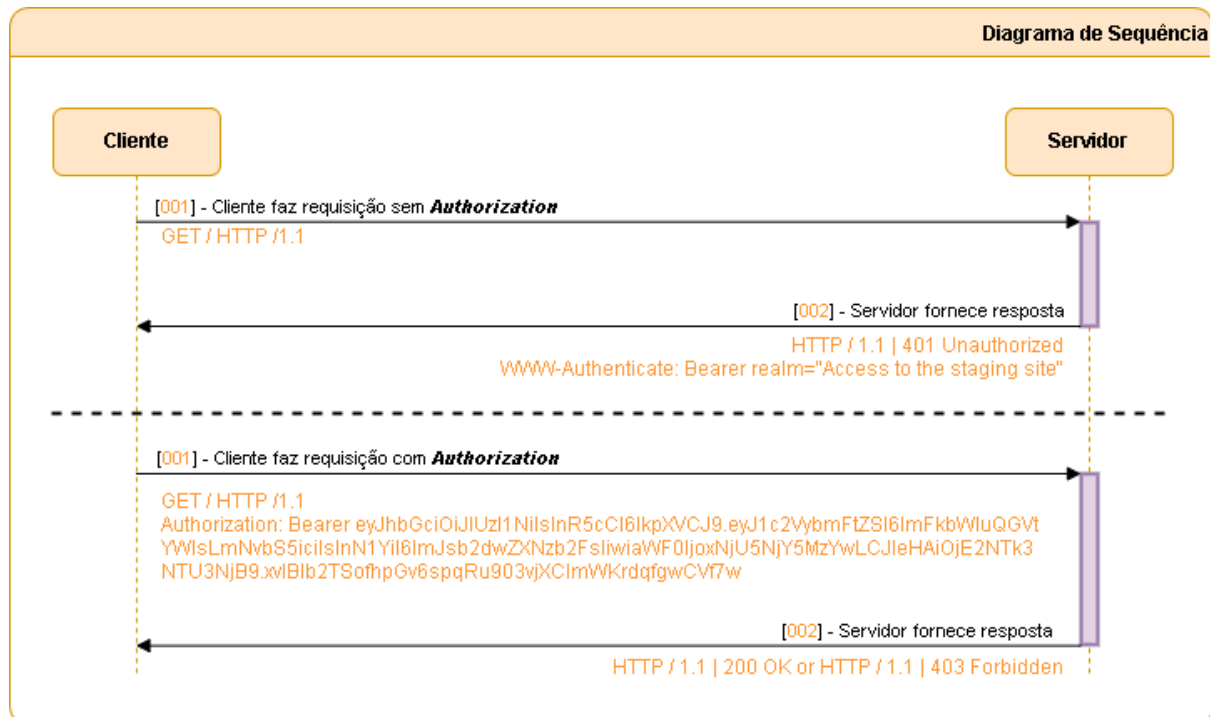
Resumindo tudo:

Para cada usuário autenticado no sistema, será gerado um Token, com prazo de validade, que posteriormente será enviado no Cabeçalho de todas Requisições HTTP. Requisições HTTP sem um Token válido serão rejeitadas pelos endpoints protegidos da aplicação. A validação de um Token funciona como um funil (Filtro de Servlet), onde só passam as Requisições que possuem um Token válido, mesmo com o usuário estando autenticado.

HTTP Authentication

O IETF (Internet Engineering Task Force) tem como missão identificar e propor soluções para as questões/problemas relacionados à utilização da Internet, além de propor a padronização das tecnologias e protocolos envolvidos.

Numa autenticação por token, o servidor responde ao cliente com uma mensagem do tipo HTTP Status 401 (Não autorizado) e fornece informações de como autorizar com um cabeçalho de resposta WWW-Authenticate contendo ao menos uma solicitação. Um cliente que deseja autenticar-se com um servidor pode fazer isso incluindo um campo de cabeçalho de solicitação WWW-Authenticate com as credenciais.



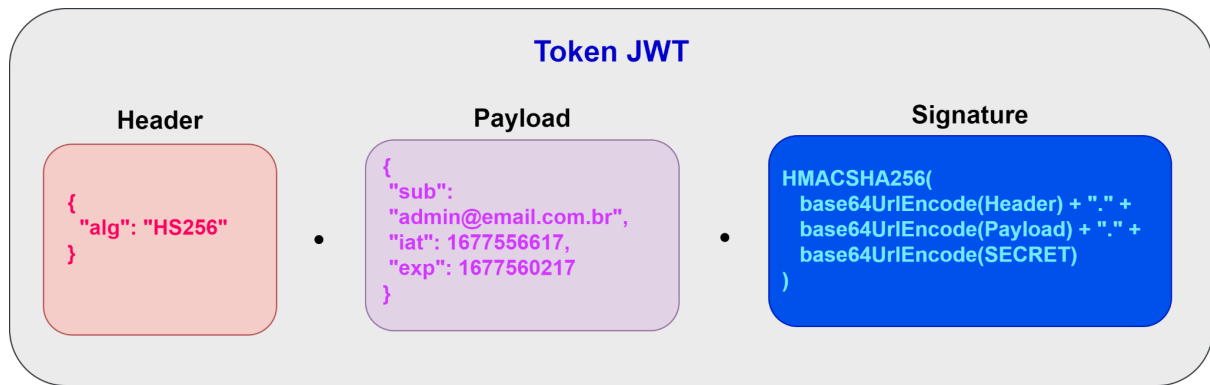
Cabeçalho HTTP: Os cabeçalhos HTTP permitem que o cliente e o servidor passem informações adicionais com a solicitação ou a resposta HTTP. Um cabeçalho de solicitação é composto por seu nome *case-insensitive* (não diferencia letras maiúsculas e minúsculas), chamada de key (chave), seguido por dois pontos ':' e pelo seu valor (sem quebras de linha), chamado value (valor).

WWW-Authenticate: Define o método de autenticação que deve ser utilizado para conseguir acesso ao recurso.

Authorization: Contém as credenciais (token) para autenticar um User-Agent com o servidor.

O Token JWT JsonWebToken

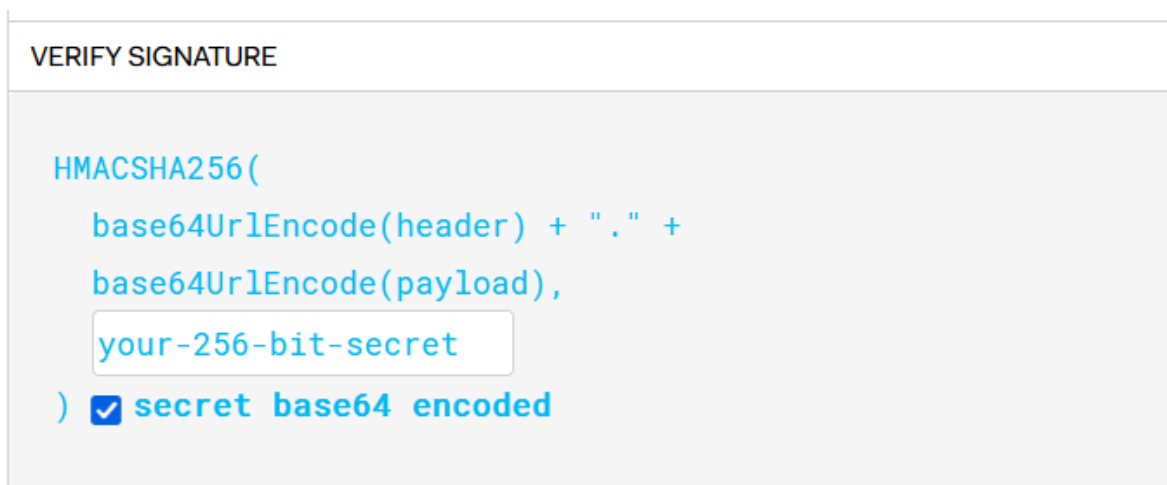
JWT é um **padrão de mercado**, muito popular e amplamente utilizado, que define como transmitir e armazenar objetos JSON de forma compacta e segura entre diferentes aplicações.



Geração do Token JWT

A Signature (assinatura) é a concatenação de String gerada através do algoritmo de codificação `base64UrlEncode` que criptografa o Header, Payload e a chave SECRET (Uma String de 256 bits, aleatória, gerada através de um Algoritmo hash), separados por um ponto final (.).

A String Gerada a partir desta concatenação, será criptografada com o algoritmo HMAC SHA256 ou outro Algoritmo indicado no Header do Token, gerando a assinatura do Token JWT. Veja o modelo de geração da assinatura na imagem abaixo:



Claims - Payload

As claims são declarações sobre uma entidade (normalmente, o usuário) e dados adicionais, passadas no conjunto de chave/valor do payload.

Essas claims podem ser de 3 tipos:

- Reserved

São os atributos não obrigatórios (mas recomendados), que são usados na validação do token pelos protocolos de segurança das APIs.

- Public

São os atributos que usamos em nossas aplicações. Normalmente armazenamos as informações do usuário autenticado na aplicação.

- Private

São os atributos definidos especialmente para compartilhar informações entre aplicações, tais como nome do usuário e os direitos de acesso (Roles: admin, usuario comum, entre outros).

Na tabela abaixo, temos a lista com as principais Claims utilizadas no payload:

- | | |
|--------------------|---|
| • sub (subject) | Entidade à quem o token pertence. |
| • iss (issuer) | Emissor do token. |
| • exp (expiration) | Timestamp de quando o token irá expirar. |
| • iat (issued at) | Timestamp de quando o token foi criado. |
| • aud (audience) | Destinatário do token, representa a aplicação que irá usá-lo. |

Criação security

(Para mais informações, acesse os cookbooks de spring security na aba de links)

Passos base para a criação de um método security com java Spring:

É necessário possuir as dependências a seguir no projeto:

- Spring Security
- Commons Codec
- JWT - Api
- JWT - Impl
- JWT - Jackson

Caso você não as tenha adicionado a princípio, é possível adicioná-las no pom.xml, dentro da tag <dependencies>.

Se tratando de usuários, devem-se criar duas classes model, a **Usuario.java**, e a **UsuarioLogin.java**.

A UsuarioLogin serve apenas para consulta e validação.

Também, deve-se eventualmente criar a **UsuarioRepository**, **UsuarioService** e **UsuarioController**.

Classes security:

1. UserDetailsImpl.java (package security)

A Interface UserDetails contém todas as informações necessárias (como: usuário (e-mail), senha e as autorizações) para construir um objeto Authentication DAO (Data Access Object), que permite utilizar um Banco de dados como meio de validação do usuário. Além disso, é responsável por definir algumas propriedades do usuário como bloqueio do usuário, expiração da senha, entre outras.

- a. Classe que implementa UserDetails. Primeiro se criam as configurações necessárias ao projeto, depois se criam 2 construtores, um vazio e um com as configurações do projeto, depois disso se implementam os métodos de UserDetails.

2. UserDetailsServiceImpl (package security)

A Interface UserDetailsService é responsável por criar o Objeto UserDetails, a partir dos dados do usuário autenticado através da AuthenticationProvider, que define qual tipo de autenticação será utilizada. Em nosso projeto, utilizaremos a autenticação por Banco de dados (DAO - Data Access Object).

UserDetailsService funciona em conjunto com o Banco de dados da aplicação através do Spring Data JPA, utilizando a Interface UsuarioRepository. Ela possui um método para carregar o usuário e retornar um Objeto da Interface UserDetails, que o Spring Security utilizará para a autenticação e validação.

- a. Classe que implementa UserDetailsService. Primeiro se chama a UsuarioRepository, depois se implementam os métodos de UserDetailsService.

3. JwtService.java (package security)

A Classe JwtService valida, valida e gera o Token JWT, a partir do Objeto da Classe UserDetails. Esta Classe será utilizada tanto nas Classes da Camada Security, quanto na Classe UsuarioService para gerar o Token JWT de cada usuário durante a autenticação.

- a. Classe “componente”, onde se cria a chave secreta para criação do token jwt, depois se implementam alguns métodos padrão.

- b. Esses métodos geralmente são: **getSignKey** (para se gerar a assinatura de login, decodificar a chave secreta, e devolver a chave de conexão), **extractAllClaims** (para se gerar o corpo da solicitação em Json), **extractClaim** (método de extração único, para se extrair o token), **extractUsername** (método de extração do username), **extractExpiration** (método de extração da data de expiração do token), **isTokenExpired** (método de verificação se o token está expirado e deve ser verificado novamente), **validateToken** (método de verificação se o token corresponde com as informações recebidas no login), **createToken** (método para se criar um token, vulgo determinar como ele pegará suas características), **generateToken** (método para se gerar um token).

4. JwtAuthFilter.java (package security)

A Classe JwtAuthFilter herda a Classe OncePerRequestFilter. Esta Classe é responsável por pré-processar todas as Requisições HTTP, o Token enviado no Cabeçalho (header) da Requisição, criar e inserir a autenticação na SecurityContext.

A Classe OncePerRequestFilter garante que um Filtro de Servlet específico seja executado apenas uma vez por Requisição HTTP. Ela possui o Método doFilterInternal(), que implementaremos com a responsabilidade de validar o Token JWT recebido no Cabeçalho da Requisição através da Classe JwtService, carregando os detalhes do usuário (através da Interface UserDetailsService) e gerando uma Autenticação através da Classe UsernamePasswordAuthenticationFilter, que posteriormente será adicionada na SecurityContext.

- a. Classe “componente” de filtragem de autenticação, para se retornar ou não o token para o usuário, após o usuário se autenticar. Ela estende o OncePerService.

5. BasicSecurityConfig.java (package security)

- a. Classe para se determinar se o usuário pode ou não acessar determinados endpoints de nosso site, de acordo com as autenticações de segurança anteriores.

A Classe SecurityFilterChain é o ponto crucial da nossa implementação de segurança. Ele configura o CORS (Cross-origin resource sharing), o CSRF (Cross-site request forgery), o gerenciamento da sessão de autenticação e as regras para os Recursos protegidos e não protegidos, que veremos mais adiante. Também podemos estender e personalizar a configuração padrão que contém os elementos a seguir.

Links e afins



CookBook:

- [Spring. MD15](#)

Gravação da aula: <https://www.youtube.com/watch?v=tjAiv7FxJDU>

Código: https://github.com/conteudoGeneration/backend_blogpessoal_v3/tree/14_Classe_usuarioService



[Documentação: HTTP Status Code 401 - Unauthorized](#)



[Documentação: HTTP Status Code 403 - Forbidden](#)



[Documentação: Cabeçalho HTTP WWW-Authenticate](#)



[Documentação: Cabeçalho de Requisição HTTP Authorization](#)



[Documentação: Cabeçalhos HTTP](#)



[Documentação: Esquema de autenticação Bearer](#)

Dúvidas