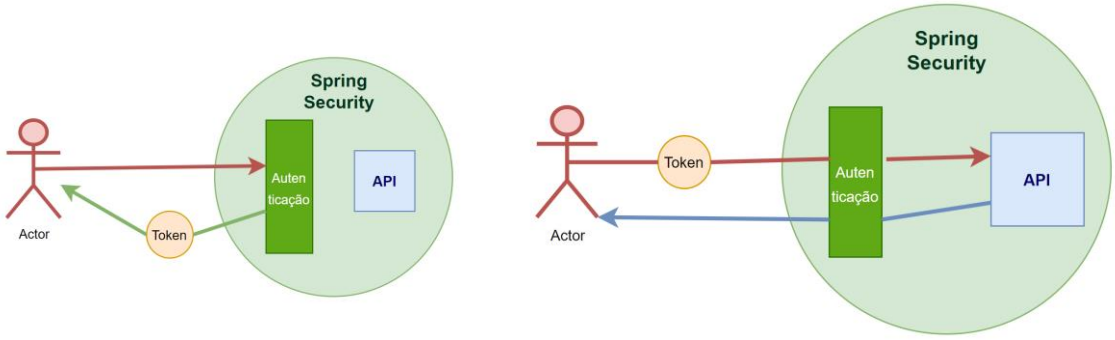
Segurança III

Autenticação em REST

A boa prática *(a única na verdade)* de uma **API REST é manter a sessão do usuário em "Stateless/sem estado"**. Na prática **isso quer dizer que o servidor não deve ter as informações do usuário guardado em memória,** ele tem que **devolver a autenticação** como resposta e já era, cabo a conversa.

Se o servidor não sabe quem está logado já que **não mantém em memória a sessão do usuário**, como ele sabe quem autenticar ou não? Simples, o **servidor** devolve um **Token** pro **usuário** e agora o **usuário** vai enviar o **token** recebido pelo header da **requisição**:



Gerenciando Autenticação

Existem diversas formas de se gerar Token, uma delas é usar **a biblioteca jjwt**. Também é necessário **desabilitar o "crsf" do Spring Security** e também o avisar que a **sessão será Stateless:**

```xml
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt</artifactId>
    <version>0.9.1</version>
</dependency>
```

```java
@Override
protected void configure(HttpSecurity http) throws Exception {
    http.authorizeRequests()
            .antMatchers(HttpMethod.GET, "/topicos").permitAll()
            .antMatchers(HttpMethod.GET, "/topicos/*").permitAll()
            .anyRequest().authenticated()
            .and().csrf().disable()
            .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS);
}
```

***** (crsf é uma proteção contra um tipo de ataque hacker, mas como a aplicação se autentica por token ela já está segura desse tipo de ataque) *****

O **usuário deverá se autenticar solicitando um endpoint especifico** (nesse caso **"/auth" no método de POST**):

```java
@RestController
@RequestMapping("/auth")
public class AutenticacaoController {

    @PostMapping
    public ResponseEntity<?> autenticar(@RequestBody @Valid LoginForm form){
```

Agora eu **preciso gerenciar a autenticação manualmente**, fazemos isso **Injetando o objeto AuthenticationManager** do **Spring Security**, esse objeto **não consegue ser injetado automaticamente**, então temos que configurar isso. A **classe herdada** pela nossa **classe de segurança** contém um **método que sabe criar esse objeto**, então basta **anota-lo** com **@Bean** que o **Spring** vai saber encontra-lo:

```java
@RestController
@RequestMapping("/auth")
public class AutenticacaoController {

    @Autowired
    private AuthenticationManager authManager;
```

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

    @Autowired
    private AutenticacaoService autenticacaoService;

    @Override
    @Bean
    protected AuthenticationManager authenticationManager() throws Exception {
        return super.authenticationManager();
    }
```

O **AuthenticationManager** tem um método de autenticação que recebe um **UserNamePasswordAuthenticationToken** também do **próprio Spring** e devolve um objeto **Authentication**, esse objeto precisa ser instanciado recebendo os **parâmetros de autenticação**, aqui no caso o **email** e **senha** do **form de login**. Com isso a autenticação manual já está feita:

```java
@PostMapping
public ResponseEntity<?> autenticar(@RequestBody @Valid LoginForm form){
    try{
        UsernamePasswordAuthenticationToken dadosLogin = form.converter();
        Authentication auth = authManager.authenticate(dadosLogin);

        return ResponseEntity.ok().build();
    }catch (AuthenticationException ex){
        return ResponseEntity.badRequest().build();
    }
}
```

```java
public class LoginForm {

    @NotNull @NotEmpty
    private String email;

    @NotNull @NotEmpty
    private String senha;

    public UsernamePasswordAuthenticationToken converter (){
        return new UsernamePasswordAuthenticationToken(email, senha);
    }
```

Gerando Token com jwt

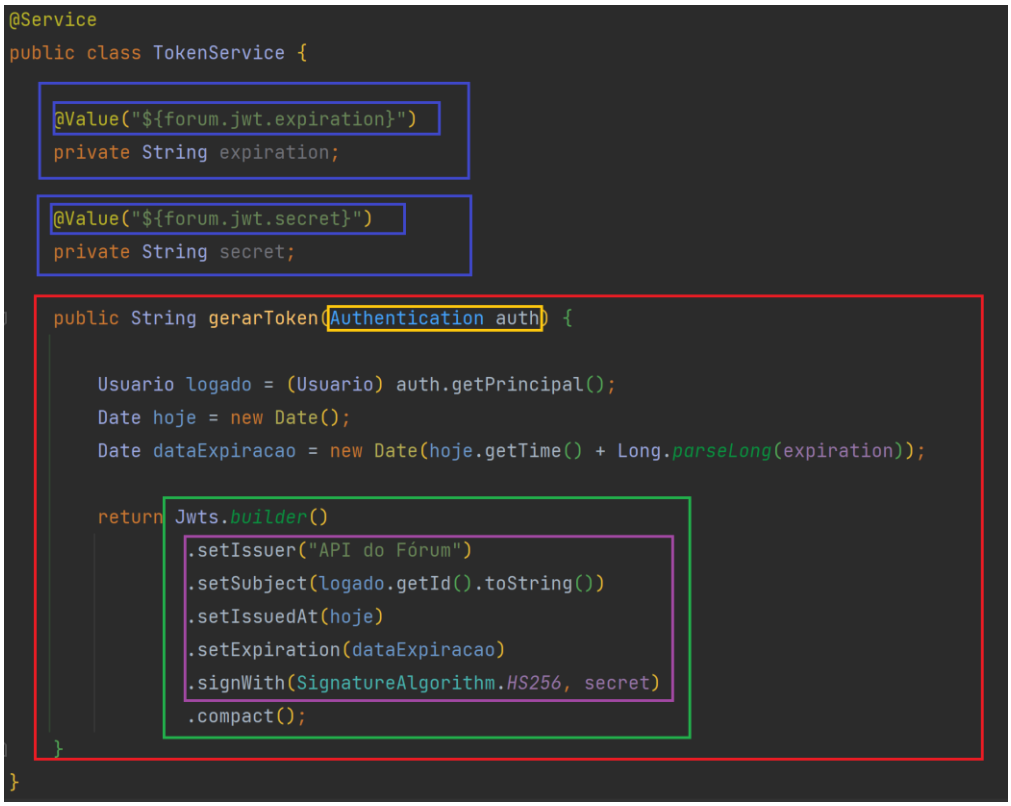
Agora é preciso **gerar o Token**, essa lógica por boa prática fica separado em alguma **classe de service,** a geração do **Token** precisa saber pra **qual usuário** ele será gerado, então passamos o objeto **Authentication** pois ele contém essa informação:

```java
@PostMapping
public ResponseEntity<?> autenticar(@RequestBody @Valid LoginForm form){
    try{
        UsernamePasswordAuthenticationToken dadosLogin = form.converter();
        Authentication auth = authManager.authenticate(dadosLogin);

        String token = tokenService.gerarToken(auth);

        return ResponseEntity.ok().build();
```

Na **criação do Token finalmente usamos o JWT**, precisamos **passar algumas informações para que o Token seja criado**, aqui **alguns parâmetros foram pegos pelo arquivo de properties** para evitar que eles fiquem no código fonte:



Os parâmetros são:

- **Issuer :** quem pediu pra gerar o Token, no caso a aplicação
- **Subject** : quem vai receber esse token, no caso o usuário, e como o ID é único ele vai ser o identificador
- **IssuedAt :** quando foi gerado)
- **Expiration :** quando esse token deve perder a validade
- **SignWith :** o token precisa ser criptografado, fazemos isso com esse parâmetro que diz qual a forma de criptografia e a senha para a criptografar

Agora basta **devolver o Token pro cliente** e dizer qual o tipo dele, a partir daí é responsabilidade do cliente passar o **token** nas requisições: