

# CETEJ35 - Java Web - JAVA\_XXX (2024\_01)

[Meus cursos](#) / [CETEJ35 - Web \(2024\\_01\)](#) / [Semana 09: 28/10 a 03/11](#) / [Segurança](#)

## Segurança

✓ **Feito:** Ver

**A fazer:** Gastar pelo menos 20 minutos na atividade

**A fazer:** Passar pela atividade até o fim

**Fecha:** segunda-feira, 2 dez. 2024, 00:00

A maioria das aplicações Web têm algum tipo de segurança. Segurança é um termo amplo que abrange vários aspectos, como conexão segura, cifragem de dados entre outros. Nesta seção, nós vamos focar em dois dos mecanismos mais comuns de segurança: autenticação e autorização.

## AUTENTICAÇÃO

Autenticação é o processo de verificar se alguém é quem diz ser. Isso pode ser feito de diferentes formas, como usando uma senha ou por biometria. Autenticação é um pré-requisito da autorização. Autorização significa que alguém tem acesso a algum recurso.

A forma mais simples de inserir autenticação em um aplicativo Spring Boot é simplesmente adicionar a dependência **spring-boot-starter-security** no seu **pom.xml**.



```
52      <dependency>
53      |      <groupId>org.springframework.boot</groupId>
54      |      <artifactId>spring-boot-starter-data-jpa</artifactId>
55      |      </dependency>
56
57      <dependency>
58      |      <groupId>org.springframework.boot</groupId>
59      |      <artifactId>spring-boot-starter-security</artifactId>
60      |      </dependency>
61      </dependencies>
```

Ao executar a aplicação você vai perceber que o Spring Boot vai te redirecionar para a tela de login.

Mas como assim? Eu nem criei o usuário. É verdade, mas o Spring Boot cria um usuário de exemplo ( **user** ) pra você e coloca toda a aplicação sob essa segurança. Você pode consultar na console do sistema a senha criada automaticamente pelo Spring Boot.

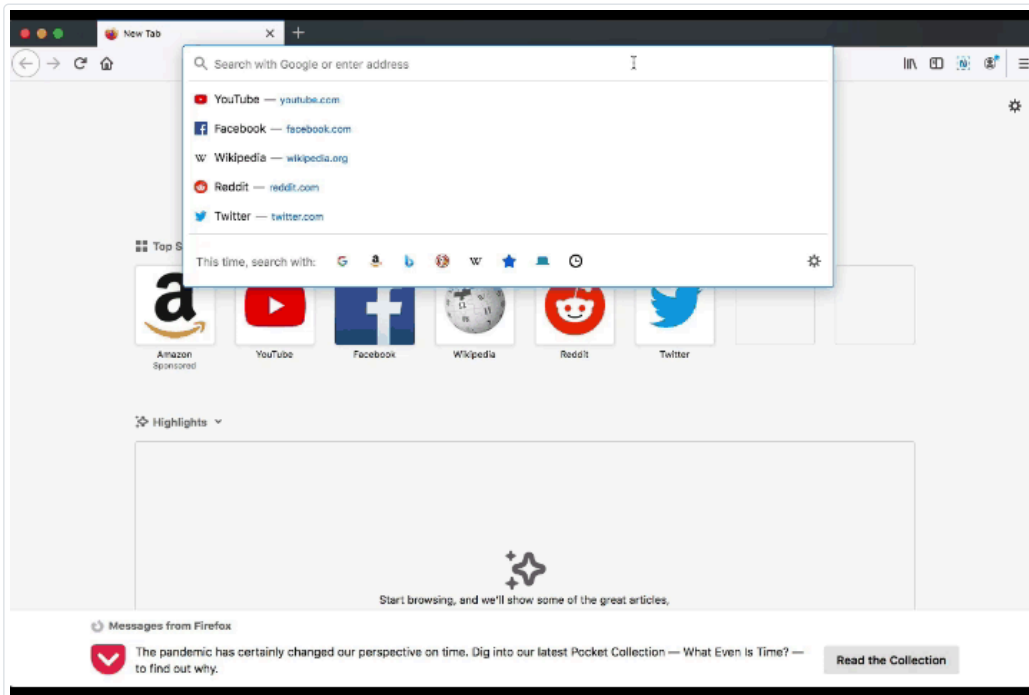
```

TERMINAL  PROBLEMS  2  OUTPUT  DEBUG CONSOLE
2021-04-27 17:34:11.755 INFO 3379 --- [ restartedMain] o.s.s.concurrent.InnredaPoolTaskExecutor : Initializing ExecutorService applica
tionTaskExecutor'
2021-04-27 17:34:12.077 INFO 3379 --- [ restartedMain] .s.s.UserDetailsServiceAutoConfiguration :
Using generated security password: 9eea4982-193e-49d2-987d-132acf47ce5c

2021-04-27 17:34:12.175 INFO 3379 --- [ restartedMain] o.s.s.web.DefaultSecurityFilterChain : Will secure any request with [org.spr
ingframework.security.web.context.request.async.WebAsyncManagerIntegrationFilter@155294d4, org.springframework.security.web.context.Secur
ityContextPersistenceFilter@414d8156, org.springframework.security.web.header.HeaderWriterFilter@1f9fe789, org.springframework.security.w
eb.csrf.CsrfFilter@6d176036, org.springframework.security.web.authentication.logout.LogoutFilter@7da2f140, org.springframework.security.w
eb.authentication.UsernamePasswordAuthenticationFilter@22b23437, org.springframework.security.web.authentication.ui.DefaultLoginPageGener
atingFilter@23337f5e, org.springframework.security.web.authentication.ui.DefaultLogoutPageGeneratingFilter@2b499444, org.springframework
.security.web.authentication.www.BasicAuthenticationFilter@5c47ac11, org.springframework.security.web.savedrequest.RequestCacheAwareFilter
@5440cd01, org.springframework.security.web.servletapi.SecurityContextHolderAwareRequestFilter@3865aed6, org.springframework.security.we

```

Mas, é claro, normalmente, o que queremos é definir usuários e autorizações para esses usuários. Nesta Seção, vamos tratar da autenticação, enquanto na Seção seguinte detalhamos o processo de autorização.



Nosso primeiro passo é indicar para o Spring Boot que vamos usar os recursos de autenticação e autorização nessa aplicação. Para isso, crie uma classe em `br.edu.utfpr.cp.esjava.crudcidades.SecurityConfig` e insira a anotação `org.springframework.security.config.annotation.web.configuration.EnableWebSecurity` imediatamente antes da definição da classe. O nome da anotação descreve muito bem o que ela faz – habilita o uso dos recursos do Spring Security.

Também precisamos adicionar a anotação `org.springframework.context.annotation.Configuration` imediatamente antes da definição da classe. Essa anotação indica que essa classe carrega configurações que devem ser usadas pelo Spring Boot.

Para garantir que as senhas são armazenadas de forma segura, o Spring Security depende de um algoritmo de cifragem. Para definir qual algoritmo iremos usar, vamos criar um método e dizer para o Spring Boot usar o algoritmo definido nesse método.

Vamos começar criando um método que retorna um objeto `org.springframework.security.crypto.password.PasswordEncoder`. Esse objeto define uma interface que todos os algoritmos de cifragem devem seguir. Vamos chamar o método de `cifrador()`.

No corpo do método vamos definir o algoritmo de cifragem que será usado. O Spring Security já possui alguns cifradores, por isso não precisamos criar nada. Para esse exemplo, vamos usar o `org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder`. Tudo que precisamos fazer é retornar uma nova instância dessa classe. Para finalizar, vamos adicionar a anotação `org.springframework.context.annotation.Bean`

imediatamente antes do método. Isso faz com o que o Spring Boot gerencie automaticamente a configuração definida pelo método.



```
12 @EnableWebSecurity
13 @Configuration
14 public class SecurityConfig {
15
16     @Bean
17     public PasswordEncoder cifrador() {
18         return new BCryptPasswordEncoder();
19     }
20 }
```

Snipped

Finalmente, podemos associar nossos usuários e o conjunto de *papéis* que eles têm acesso. Uma *autorização* ou *papel* define um grupo de autorizações/restrições que um ou mais usuários possuem. Os usuários e seus papéis/autorizações podem ser armazenados em diferentes locais, como na memória da aplicação ou no banco de dados. Para esse primeiro exemplo, vamos definir os usuários na memória da aplicação.

Papéis (*roles*) e autorizações (*authorities*) podem ser usados de forma intercambiável em alguns materiais. Contudo, o Spring Security implementa o conceito de papéis e autorizações **como duas coisas diferentes**. Aqui, estamos usando ambos como sinônimos, por simplicidade. Mas em configurações de segurança mais complexas, talvez você precise dos dois.



Para isso, vamos criar um novo método (`configure()`, linha 17, na Figura abaixo). Esse método retorna um objeto `org.springframework.security.provisioning.InMemoryUserDetailsManager`, representando como o gerenciamento de usuários é feito. Nesse primeiro exemplo, os usuários estarão armazenados em memória. Em uma seção posterior, vamos armazenar os usuários em um banco de dados. Veja como fica o código na Figura abaixo.

Cada usuário é definido programaticamente, como uma instância de `org.springframework.security.core.userdetails.UserDetails`. A classe `org.springframework.security.core.userdetails.User` fornece um método utilitário que permite criar um novo usuário (`withUsername(String)`), sua senha (`password(String)`), e papéis associados (`roles(String)`).

Observe o uso do método `cifrador()`, para atribuir a senha do usuário, nas linhas 19 e 24 da Figura abaixo. Veja que o mesmo `cifrador` usado para criar a senha nas linhas 19 e 24 também será usado pelo Spring Security para **verificar** a senha - por isso a necessidade da anotação `@Bean` no método `cifrador()`. Veja como fica o código na Figura abaixo.

Por fim, basta retornar um novo objeto `InMemoryUserDetailsManager` com os dois usuários criados (linha 28). Veja como fica o código completo na Figura abaixo.

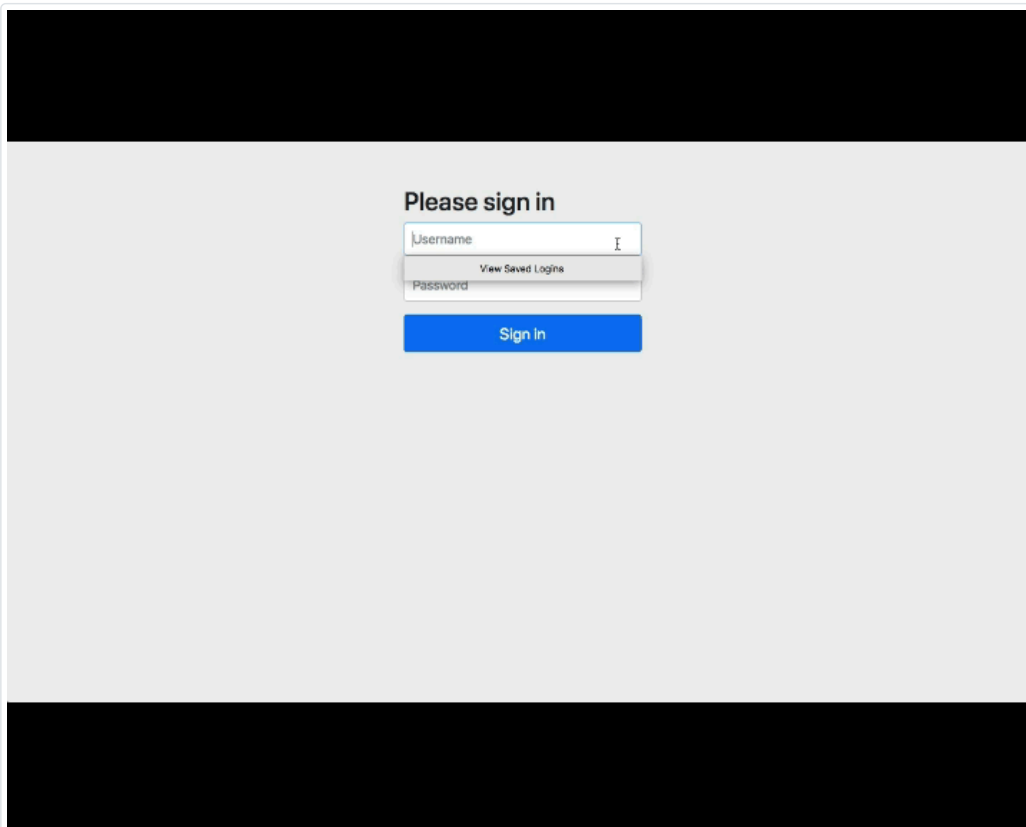
```
SecurityConfig.java

12 @EnableWebSecurity
13 @Configuration
14 public class SecurityConfig {
15
16     @Bean
17     public InMemoryUserDetailsManager configure() throws Exception {
18         UserDetails john = User.withUsername("john")
19             .password(cifrador().encode("test123"))
20             .roles("listar")
21             .build();
22
23         UserDetails anna = User.withUsername("anna")
24             .password(cifrador().encode("test123"))
25             .roles("admin")
26             .build();
27
28         return new InMemoryUserDetailsManager(john, anna);
29     }
30
31     @Bean
32     public PasswordEncoder cifrador() {
33         return new BCryptPasswordEncoder();
34     }
35 }
```

Snipped



Ao executar a aplicação, a tela de login padrão do Spring Security é carregada. Ao inserir um usuário e senha conforme definidos na classe `SecurityConfig`, a tela do aplicativo é carregada.



The image shows a screenshot of the Spring Security login page. At the top, there is a black header bar. Below it, the page has a light gray background. In the center, there is a white box containing the text "Please sign in". Below this text, there are two input fields: "Username" and "Password". The "Username" field has a small "I" icon on the right. Below the "Username" field, there is a link that says "View Saved Logins". Below the "Password" field, there is a blue button that says "Sign in".

O código desenvolvido nesta Seção está disponível no [Github](#), na branch [semana07-10-autenticacao](#).

Retroceder

Avançar

◀ Atividade II WebConf

Seguir para...

[Listeners & Cookies ▶](#)

✉ Contate o suporte do site [↗](#)

Você acessou como RAFAEL ROCHA DA SILVA PROENCA (Sair)  
CETEJ35 - Web (2024\_01)

- Tema
- Adaptable
- Boost
- Clássico
- Campus
- Apucarana
- Campo Mourão
- Cornélio Procópio
- Curitiba
- Dois Vizinhos
- Francisco Beltrão
- Guarapuava
- Londrina
- Medianeira
- Pato Branco
- Ponta Grossa
- Reitoria
- Santa Helena
- Toledo
- UTFPR
- Ajuda
- Chat UTFPR
- Calendário Acadêmico
- Biblioteca
- e-Mail
- Nuvem (OwnCloud )
- Produção Acadêmica
- Secretaria Acadêmica
- Sistemas Corporativos
- Sistema Eletrônico de Informação - SEI
- Suporte ao usuário
- Criação de curso
- Comunidade
- Português - Brasil (pt\_br)
- Deutsch (de)
- English (en)
- Português - Brasil (pt\_br)

Resumo de retenção de dados

Baixar o aplicativo móvel.

🗨 Dê um feedback sobre este software [↗](#)

Universidade Tecnológica Federal do Paraná - UTFPR  
Suporte ao usuário

