CETEJ35 - Java Web - JAVA_XXX (2024_01)

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.

AUTORIZAÇÃO

Com o usuário autenticado, o próximo passo é definir as autorizações. Para definir as autorizações, vamos criar um novo método: `org.springframework.security.web.SecurityFilterChain filter(org.springframework.security.config.annotation.web.builders.HttpSecurity http)`. O nome do método é irrelevante, mas o tipo de retorno e do parâmetro não podem ser alterados. O mecanismo de segurança do Spring Security é baseado na execução de uma cadeia de filtros. Os filtros são objetos que carregam regras de acesso, definindo as autorizações, por exemplo. O objeto `SecurityFilterChain` define esse método como mais um filtro que deve ser executado nessa cadeia. Já o objeto `HttpSecurity` permite acesso à métodos para definir as autorizações dos *papéis* dos usuários.

Quer saber mais sobre como funcionam os filtros de segurança? Que tal dar uma olhada na **documentação oficial**.

O método `filter` precisa ser anotado com `@Bean`, assim o Spring se encarrega de cuidar do ciclo de vida do método e adicioná-lo no momento certo de execução. Com o método criado, podemos criar as regras de autorização. Por simplicidade, neste projeto vamos começar usando o parâmetro `http` para desabilitar o CSRF.

SecurityConfig.java

```
33      @Bean
34      public SecurityFilterChain filter(HttpSecurity http) throws Exception {
35          return http
36                  .csrf().disable()
37
```

Snipped

Agora, vamos começar a mapear as autorizações com as URLs da aplicação. Para isso, vamos usar o método `authorizeHttpRequests()`, seguido das URLs que queremos proteger e dos papéis autorizados para cada URL. Para isso usamos os métodos `requestMatchers(String)`, e `hasAnyRole(String...)` - para um conjunto de papéis, ou `hasRole(String)` - para um único papel, conforme mostra a Figura a seguir.

Na linha 38 definimos que a URL raiz (`requestMatchers("/")`), mapeada no `CidadeController` com o método `listar()`, pode ser acessada por qualquer usuário com os papéis `listar` ou `admin` (`hasAnyRole("listar", "admin")`). No nosso exemplo, isso inclui os dois usuários definidos na seção anterior.

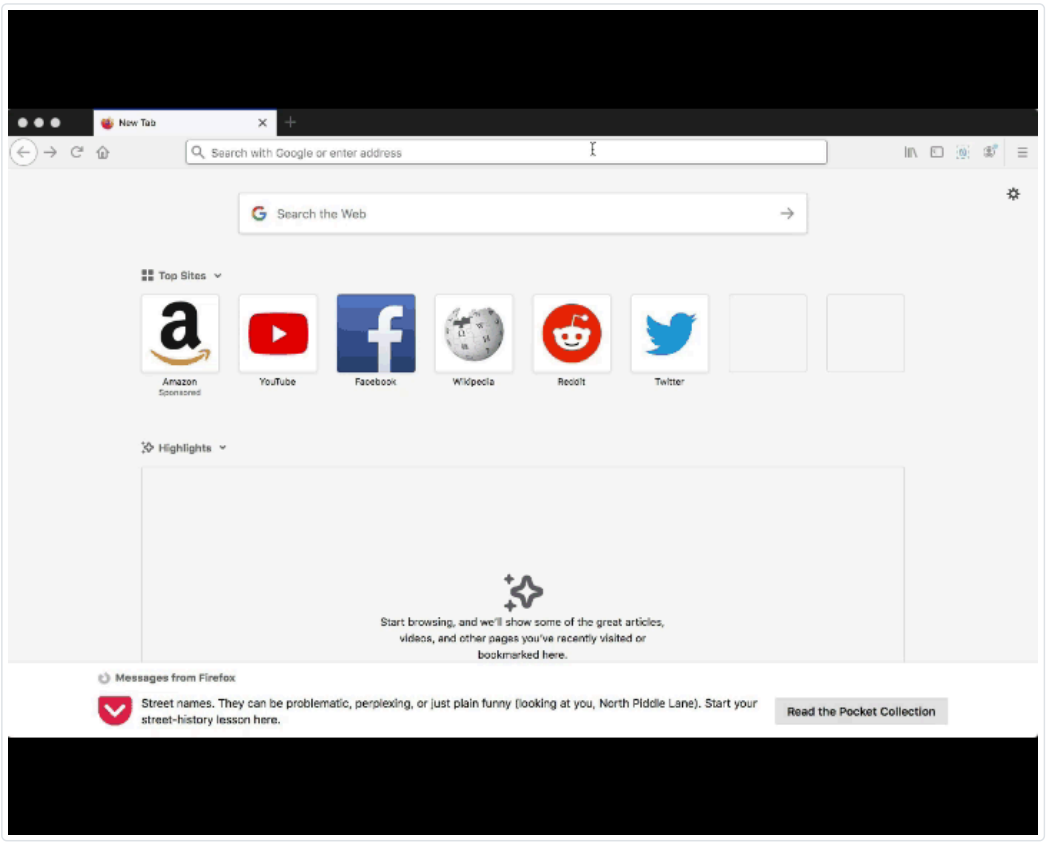
A linhas 39 segue a mesma lógica, porém, definindo apenas um papel (`hasRole(String)`) para todas as URLs definidas. Observe que cada URL está mapeada com uma operação CRUD. Dessa forma, nosso usuário *john*, que tem o papel/autorização *listar*, tem permissão apenas para listar as cidades. Por outro lado, o usuário *anna*, que tem o papel *admin*, tem permissão para criar, alterar ou excluir cidades.

Para evitar que qualquer URL não definida anteriormente seja deixada aberta para acesso, vamos negar o acesso a qualquer outra URL não definida explicitamente usando `anyRequest().denyAll()` (linha 40).

Para finalizar, vamos liberar o acesso à página de login usando `and().formLogin().permitAll()`. O método `and()` é usado como uma conjunção, permitindo adicionar mais um conjunto de regras. O acesso ao formulário de login (`formLogin()`) precisa estar liberado a todos (`permitAll()`), caso contrário os usuários não conseguem inserir suas credenciais para login. Veja como ficou o código completo.

Agora você pode executar a aplicação e ver o resultado.

Observe que ainda não temos um botão de logout. Por isso, para encerrar a sessão você precisa limpar a cache do navegador.



O código desenvolvido nesta Seção está disponível no **Github**, na branch `semana07-20-autorizacao`.

Ficou incomodado com alertas de `deprecated` no seu código enquanto fazia essa lição? Calma, o Spring está sempre em evolução para usar os recursos mais avançados da linguagem. Gradualmente o framework está favorecendo o uso de Lambda DSL. Você pode assistir esse vídeo (https://youtu.be/PWnEZh_t0WI) para saber mais, e também ler esse **artigo**. Nós também temos uma versão do nosso código usando Lambda DSL na branch **`semana07-50-autenticacao-lambda`**.

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