CAPÍTULO 12

Spring MVC: Autenticação e autorização

"Experiência é apenas o nome que damos aos nossos erros."
— Oscar Wilde

Nesse capítulo, você aprenderá:

- O escopo de sessão e como ele funciona;
- O que são interceptadores;
- Implantar sua aplicação em qualquer container.

12.1 - AUTENTICANDO USUÁRIOS: COMO FUNCIONA?

É comum hoje em dia acessarmos algum site que peça para fazermos nosso login para podermos ter acesso a funcionalidades da aplicação. Esse processo também é conhecido como autenticação. Mas, como o site sabe, nas requisições seguintes que fazemos, quem somos nós?

12.2 - COOKIES

O protocolo HTTP utilizado para o acesso à páginas é limitado por não manter detalhes como quem é quem entre uma conexão e outra. Para resolver isso, foi inventado um sistema para facilitar a vida dos programadores.

Um **cookie** é normalmente um par de strings guardado no cliente, assim como um mapa de strings. Esse par de strings possui diversas limitações que variam de acordo com o cliente utilizado, o que torna a técnica de utilizá-los algo do qual não se deva confiar muito. Já que as informações do cookie são armazenadas no cliente, o mesmo pode alterá-la de alguma

maneira... sendo inviável, por exemplo, guardar o nome do usuário logado...

Quando um cookie é salvo no cliente, ele é enviado de volta ao servidor toda vez que o cliente efetuar uma nova requisição. Desta forma, o servidor consegue identificar aquele cliente sempre com os dados que o cookie enviar.

Um exemplo de bom uso de cookies é na tarefa de lembrar o nome de usuário na próxima vez que ele quiser se logar, para que não tenha que redigitar o mesmo.

Cada cookie só é armazenado para um website. Cada website possui seus próprios cookies e estes não são vistos em outra página.

12.3 - Sessão

Usar Cookies parece facilitar muito a vida, mas através de um cookie não é possível marcar um cliente com um objeto, somente com `Strings`. Imagine gravar os dados do usuário logado através de cookies. Seria necessário um cookie para cada atributo: `usuario`, `senha`, `id`, `data de inscrição`, etc. Sem contar a falta de segurança.

O Cookie também pode estar desabilitado no cliente, sendo que não será possível lembrar nada que o usuário fez...

Uma sessão facilita a vida de todos por permitir atrelar objetos de qualquer tipo a um cliente, não sendo limitada somente à strings e é independente de cliente.

A abstração da API facilita o trabalho do programador pois ele não precisa saber como é que a sessão foi implementada no servlet container, ele simplesmente sabe que a funcionalidade existe e está lá para o uso. Se os cookies estiverem desabilitados, a sessão não funcionará e devemos recorrer para uma técnica (trabalhosa) chamada `url-rewriting`.

A sessão nada mais é que um tempo que o usuário permanece ativo no sistema. A cada página visitada, o tempo de sessão é zerado. Quando o tempo ultrapassa um limite demarcado no arquivo `web.xml`, o cliente perde sua sessão.

12.4 - Configurando o tempo limite

Para configurar 3 minutos como o padrão de tempo para o usuário perder a sessão basta incluir o seguinte código no arquivo `web.xml`:

```
<session-config>
  <session-timeout>3</session-timeout>
</session-config>
```

12.5 - Registrando o usuário logado na sessão

Podemos criar uma ação que fará o login da aplicação. Caso o usuário informado na tela de login exista, guardaremos o mesmo na sessão e entraremos no sistema, e caso não exista vamos voltar para a página de login.

O Spring MVC nos possibilita receber a sessão em qualquer método, só é preciso colocar o objeto `HttpSession` como parâmetro. Por exemplo:

```
@Controller
public class LoginController {

  public String efetuaLogin(HttpSession session) {
    //....
```

A sessão é parecida com um objeto do tipo `Map<String, Object>`, podemos guardar nela qualquer objeto que quisermos dando-lhes uma chave que é uma String. Portanto, para logarmos o usuário na aplicação poderíamos criar uma ação que recebe os dados do formulário de login e a sessão HTTP, guardando o usuário logado dentro da mesma:

```
@RequestMapping("efetuaLogin")
public String efetuaLogin(Usuario usuario, HttpSession session) {
  if(new JdbcUsuarioDao().existeUsuario(usuario)) {
    session.setAttribute("usuarioLogado", usuario);
    return "menu";
  } else {
    //....
```

12.6 - Exercício: Fazendo o login na aplicação

1. Vamos criar o formulário de Login, uma ação para chamar este formulário e uma outra que realmente autentica o usuário.

a. Crie a página `formulario-login.jsp` dentro de `WebContent/WEB-INF/views` com o conteúdo:

```
<html>
  <body>
    <h2>Página de Login das Tarefas</h2>
    <form action="efetuaLogin" method="post">
      Login: <input type="text" name="login" /> <br />
      Senha: <input type="password" name="senha" /> <br />
      <input type="submit" value="Entrar nas tarefas" />
    </form>
  </body>
</html>
```

b. Crie uma nova classe chamada `LoginController` no pacote `br.com.caelum.tarefas.controller`. Crie um método para exibir o `formulario-login.jsp`:

```
@Controller
public class LoginController{

  @RequestMapping("loginForm")
  public String loginForm() {
    return "formulario-login";
  }
}
```

c. Na mesma classe `LoginController` coloque o método que verifica o existência do usuario. Acrescente o método `efetuaLogin`:

```
@RequestMapping("efetuaLogin")
public String efetuaLogin(Usuario usuario, HttpSession session) {
  if(new JdbcUsuarioDao().existeUsuario(usuario)) {
    session.setAttribute("usuarioLogado", usuario);
    return "menu";
  }
  return "redirect:loginForm";
```

```
}
```

d. Após o usuário se logar, ele será redirecionado para uma página que conterá links para as outras páginas do sistema e uma mensagem de boas vindas.Crie a página `menu.jsp` em `WebContent/WEB-INF/views` com o código:

```
<html>
  <body>
    <h2>Página inicial da Lista de Tarefas</h2>
    <p>Bem vindo, ${usuarioLogado.login}</p>
    <a href="listaTarefas">Clique aqui</a> para acessar a
    lista de tarefas
  </body>
</html>
```

e. Acesse a página de login em http://localhost:8080/fj21-tarefas/loginForm e se logue na aplicação.

f. Verifique o banco de dados para ter um login e senha válidos. Para isso, no terminal faça:

```
mysql -u root
use fj21;
select * from usuarios;
```

Se a tabela não existir, você pode criá-la executando o comando:

```
create table usuarios (
  login VARCHAR(255),
  senha VARCHAR(255)
);
```

g. Caso não exista usuários cadastrados, cadastre algum utilizando o mesmo terminal aberto antes da seguinte maneira:

```
insert into usuarios(login, senha) values('seu_usuario', 'sua_senha');
```

12.7 - Bloqueando acessos de usuários não logados com Interceptadores

Não podemos permitir que nenhum usuário acesse as tarefas sem que ele esteja logado na aplicação, pois essa não é uma informação pública. Precisamos portanto garantir que antes de executarmos qualquer ação o usuário esteja autenticado, ou seja, armazenado na sessão.

Utilizando o Spring MVC, podemos utilizar o conceito dos Interceptadores, que funcionam como Filtros que aprendemos anteriormente, mas com algumas funcionalidades a mais que estão relacionadas ao framework.

Para criarmos um Interceptador basta criarmos uma classe que implemente a interface `org.springframework.web.servlet.HandlerInterceptor`. Ao implementar essa interface, precisamos implementar 3 métodos: `preHandle`, `postHandle` e `afterCompletion`.

Os métodos `preHandle` e `postHandle` serão executados antes e depois, respectivamente, da ação. Enquanto o método `afterCompletion` é chamado no final da requisição, ou seja após ter renderizado o JSP.

Para nossa aplicação queremos verificar o acesso antes de executar uma ação, por isso vamos usar apenas o método `preHandle` da interface `HandlerInterceptor`. Porém, usando a interface `HandlerInterceptor` somos obrigados implementar todos os métodos definidos na interface. Queremos usar apenas o `preHandle`. Por isso o Spring MVC oferece uma classe auxiliar (`HandlerInterceptorAdapter`) que já vem com uma implementação padrão para cada método da interface. Então para facilitar o trabalho vamos estender essa classe e sobrescrever apenas o método que é do nosso interesse:

```
public class AutorizadorInterceptor extends HandlerInterceptorAdapter {

  @Override
  public boolean preHandle(HttpServletRequest request,
      HttpServletResponse response,
      Object controller) throws Exception {
    //....
    response.sendRedirect("loginForm");
    return false;
  }
}
```

O método `preHandle` recebe a requisição e a resposta, além do controlador que está sendo interceptado. O retorno é um booleano que indica se queremos continuar com a requisição ou não. Portanto, a classe `AutorizadorInterceptor` só deve devolver `true` se o usuário está logado. Caso o usuário não esteja autorizado vamos redirecionar para o formulário de login.

Para pegar o usuário logado é preciso acessar a sessão HTTP. O objeto `request` possui um método que devolve a sessão do usuário atual:

```
HttpSession session = request.getSession();
```

Dessa maneira podemos verificar no Interceptador a existência do atributo `usuarioLogado`:

```
public class AutorizadorInterceptor extends HandlerInterceptorAdapter {

  @Override
  public boolean preHandle(HttpServletRequest request,
```

```
        HttpServletResponse response,
        Object controller) throws Exception {

        if(request.getSession().getAttribute("usuarioLogado") != null) {
          return true;
        }

        response.sendRedirect("loginForm");
        return false;
    }
}
```

Falta só mais uma verificação. Existem duas ações na nossa aplicação que não necessitam de autorização. São as ações do `LoginController`, necessárias para permitir a autenticação do usuário. Além disso, vamos garantir também que a pasta de `resources` pode ser acessada mesmo sem login. Essa pasta possui as imagens, css e arquivos JavaScript. No entanto a classe `AutorizadorInterceptor` fica:

```
public class AutorizadorInterceptor extends HandlerInterceptorAdapter {

  @Override
  public boolean preHandle(HttpServletRequest request,
      HttpServletResponse response,
      Object controller) throws Exception {

      String uri = request.getRequestURI();
      if(uri.endsWith("loginForm") ||
          uri.endsWith("efetuaLogin") ||
              uri.contains("resources")){
        return true;
      }

      if(request.getSession().getAttribute("usuarioLogado") != null) {
        return true;
      }

      response.sendRedirect("loginForm");
      return false;
  }
}
```

Ainda precisamos registrar o nosso novo interceptador. Mas o Spring MVC não permite que façamos isso via anotações, então usaremos a configuração via XML nesse caso. Já vimos o arquivo `spring-context.xml`, nele vamos usar o tag `mvc:interceptors` para cadastrar a classe `AutorizadorInterceptor`:

```
<mvc:interceptors>
  <bean class=
    "br.com.caelum.tarefas.interceptor.AutorizadorInterceptor" />
</mvc:interceptors>
```

Caso seja necessário alguma ordem na execução de diversos interceptors, basta registrá-

los na sequência desejada dentro da tag `mvc:interceptors`.

12.8 - Exercícios: Interceptando as requisições

1. Vamos criar um `Interceptor` que não permitirá que o usuário acesse as ações sem antes ter logado na aplicação.

a. Crie a classe `AutorizadorInterceptor` no pacote `br.com.caelum.tarefas.interceptor`

b. Estenda a classe `HandlerInterceptorAdapter` do package `org.springframework.web.servlet.handler`

c. Sobrescreve o método `preHandle`. O usuário só pode acessar os métodos do `LoginController` SEM ter feito o login. Caso outra lógica seja chamada é preciso verificar se o usuário existe na sessão. Existindo na sessão, seguiremos o fluxo normalmente, caso contrário indicaremos que o usuário não está logado e que deverá ser redirecionado para o formulário de login. O código completo do interceptador fica:

```
public class AutorizadorInterceptor extends HandlerInterceptorAdapter {

  @Override
  public boolean preHandle(HttpServletRequest request,
      HttpServletResponse response,
      Object controller) throws Exception {

      String uri = request.getRequestURI();
      if(uri.endsWith("loginForm") ||
          uri.endsWith("efetuaLogin") ||
                    uri.contains("resources")){
        return true;
      }

      if(request.getSession()
          .getAttribute("usuarioLogado") != null) {
        return true;
      }

      response.sendRedirect("loginForm");
      return false;
  }
}
```

d. Temos que registrar o nosso novo interceptador no XML do spring. Abra o arquivo `spring-context.xml`. Dentro da tag `<beans>` adicione:

```
<mvc:interceptors>
    <bean
    class=
    "br.com.caelum.tarefas.interceptor.AutorizadorInterceptor" />
</mvc:interceptors>
```

2. Reinicie o servidor e tente acessar a lista de tarefas em http://localhost:8080/fj21-tarefas/listaTarefas. Você deverá ser redirecionado para o formulário de login.

12.9 - Exercícios opcionais: Logout

1. Faça o logout da aplicação. Crie um link no `menu.jsp` que invocará um método que removerá o usuário da sessão e redirecione a navegação para a action do formulário de login (`loginForm`).

No `menu.jsp` acrescente:

```
<a href="logout">Sair do sistema</a>
```

Na classe `LoginController` adicione o método para o logout e invalide a sessão do usuário:

```
@RequestMapping("logout")
public String logout(HttpSession session) {
  session.invalidate();
  return "redirect:loginForm";
}
```

CAPÍTULO ANTERIOR:

Spring MVC

PRÓXIMO CAPÍTULO:

Spring IoC e deploy da aplicação

Você encontra a Caelum também em:

Blog Caelum

Cursos Online

Facebook

Newsletter

Casa do Código

Twitter