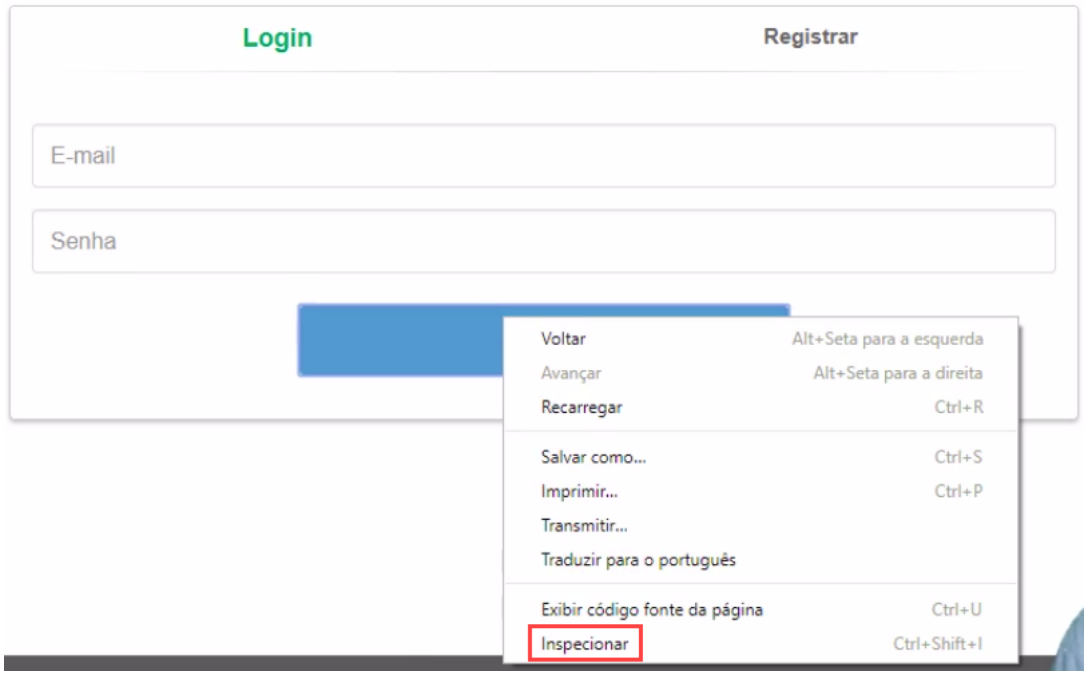
08

Forjando autenticação

Transcrição

Colocamos no campo da senha um caractere especial, que é a aspas simples. Com isso, tivemos um erro de sintaxe no SQL informando qual o tipo de banco que estamos usando. Tentaremos entender o porque essa *Exception* ocorreu.

Começaremos verificando para onde o formulário de *login* está enviando essas informações. Na página de *login*, clicaremos com o botão direito do mouse e selecionaremos "Inspecionar".



O formulário está enviando os dados para o endereço `/alura-shows/login`. Isso significa que temos um *Controller* na aplicação, com um método que está respondendo para o endereço `/login`.

No Eclipse dentro do pacote "controller" temos a classe `UsuarioController`, nela encontramos o método `login()`. O método recebe um `Usuario usuario`, onde é passado como argumento na chamada `dao.procuraUsuario(usuario)` para fazer a validação.

```
@RequestMapping("/login")
public String login(@ModelAttribute("usuario") Usuario usuario,
```

```
            RedirectAttributes redirect, Model model, HttpSession session) {

        Usuario usuarioRetornado = dao.procuraUsuario(usuario);
        model.addAttribute("usuario", usuarioRetornado);
        if (usuarioRetornado == null) {
            redirect.addFlashAttribute("mensagem", "Usuário não encontrado");
            return "redirect:/usuario";
        }

        session.setAttribute("usuario", usuarioRetornado);
        return "usuarioLogado";
    }
```

Acessaremos então a classe `UsuarioDaoImpl` para analisar o método `procuraUsuario()`. No método, temos a *query* SQL, onde concatenamos os valores de e-mail e senha vindas do objeto `usuario`.

```
public Usuario procuraUsuario(Usuario usuario) {

    String query = "SELECT * FROM Usuario WHERE email=" + "'"
            + usuario.getEmail() + "'" + " and senha=" + "'"
            + usuario.getSenha() + "';";

    // ...
}
```

Para analisar o comando SQL, colocaremos em um única linha e removeremos todas as concatenações do Java. A *query* está sendo enviada ao banco de dados da seguinte forma:

```
SELECT * FROM Usuario WHERE email='usuario.getEmail()' and senha='usuario.getSenha()';
```

Quando o usuário envia seus dados para efetuar a autenticação no formulário de *login*, eles são recebidos e colocados na *query* que será enviada para o banco de dados. Os comandos Java serão subsistidos pelos valores e a *query* ficará da seguinte maneira:

```
SELECT * FROM Usuario WHERE email='alex@gmail.com' and senha='';
```

Repare que as aspas simples usadas nos valores possuem abertura e fechamento, da mesma forma que fazemos com Strings em Java. Ao inserir uma aspas simples a mais no campo de senha, foi identificado que está faltando o seu fechamento e provocando o erro de sintaxe. Como as aspas delimitam valores do tipo String, é possível injetar códigos na *query*.

Por exemplo, o usuário **Alex** possui um pouco de conhecimento em programação e conhece o usuário **Fernando**. O Alex tem tem a intenção de efetuar o *login* com a conta do Fernando, por isso ele colocou no campo **E-mail** `fernando@gmail.com` e no campo **Senha** `x`. A *query* será enviada da seguinte forma:

```
SELECT * FROM Usuario WHERE email='fernando@gmail.com' and senha='x';
```

Essa *query* fará uma expressão **booleana**, analisando os valores passados de e-mail e senha com a dos usuários cadastrados. Quando o banco checar a conta do Fernando, será retornado `true` para o e-mail e `false` para a senha.

Dessa forma o acesso será recusado, justamente porque os dois parâmetros precisam ser verdadeiros.

O que o Alex precisaria fazer para acessar a conta do Fernando, era transformar a o segundo parâmetro em verdadeiro. Colocando no campo **Senha** o valor `x' or 'a' = 'a` , a expressão ficará:

```
SELECT * FROM Usuario WHERE email='fernando@gmail.com' and senha='x' or 'a' = 'a';
```

Dessa forma, estamos dizendo que `x` **ou** `'a'='a'` precisam ser verdadeiros. O `x` retornará `false` mas `'a'='a'` sempre retornará `true` . Em expressões booleanas que utilizam operador `OR` (**OU**), apenas um valor precisa ser verdadeiro para que toda a expressão se torne verdadeira.

Com os dois parâmetros retornando verdadeiro, a autenticação será válida permitindo que o Alex entre na aplicação com a conta do Fernando. Em uma situação real, isso poderia trazer prejuízos ao usuário que teve a conta invadida.

Veremos outros detalhes que essa injeção de código SQL pode trazer de problemas para nós.