02

# Administração: listagem e autenticação

Com tudo o que já temos pronto, precisamos definir um papel de administrador no sistema que possa gerenciar os usuários e produtos. Vamos então alterar o modelo de usuário para podermos ter tanto administradores quanto clientes, mas antes a vamos criar algumas telas de gerenciamento para os administradores. Faremos uma listagem dos usuários e dos produtos existentes.

app/views/usuarios.scala.html

```
@(usuarios: List[Usuario])
@main("Lista de usuários") {
    <section class="panel panel-default">
        <header class="panel-heading">
            <h2 class="panel-title">Lista de usuários</h2>
        </header>
        <div class="panel-body">
            <table class="table table-striped">
                <thead>
                    <tr>
                        <td>Id</td>
                        <td>Nome</td>
                        <td>Email</td>
                        <td>Token</td>
                        <td>Acessos</td>
                        <td>Último acesso</td>
                    </tr>
                </thead>
                <tbody>
                @for(usuario <- usuarios) {
                    <tr>
                        <td>@usuario.getId()</td>
                        <td>@usuario.getNome()</td>
                        <td>@usuario.getEmail()</td>
                        <td>@usuario.getToken().getCodigo()</td>
                        <td>@usuario.getAcessos().size()</td>
                        <td>@usuario.getAcessos().get(usuario.getAcessos().size()-1).getData()<
                    </tr>
                }
                </tbody>
            </table>
        </div>
    </section>
}
```

app/views/produtos.scala.html

```
@(produtos: List[Produto])
@main("Lista de produtos") {
    <section class="panel panel-default">
        <header class="panel-heading">
            <h2 class="panel-title">Lista de produtos</h2>
```

```
            </header>
            <div class="panel-body">
                <a class="btn btn-info" href="@routes.ProdutoController.formularioDeNovoProduto">Cr
                <table class="table table-striped">
                    <thead>
                        <tr>
                            <td>Id</td>
                            <td>Título</td>
                            <td>Código</td>
                            <td>Tipo</td>
                            <td>Descrição</td>
                            <td>Preço</td>
                        </tr>
                    </thead>
                    <tbody>
                    @for(produto <- produtos) {
                        <tr>
                            <td>@produto.getId()</td>
                            <td>@produto.getTitulo()</td>
                            <td>@produto.getCodigo()</td>
                            <td>@produto.getTipo()</td>
                            <td>@produto.getDescricao()</td>
                            <td>R$ @produto.getPreco()</td>
                        </tr>
                    }
                    </tbody>
                </table>
            </div>
        </section>
    }
```

Com as views prontas, podemos criar as rotas, métodos do controller e adicionar a listagem de usuários no DAO.

```
GET /admin/usuarios controllers.AdminController.usuarios
GET /admin/produtos controllers.AdminController.produtos
```

```
package controllers;
public class AdminController extends Controller {
    @Inject
    private UsuarioDAO usuarioDAO;
    @Inject
    private ProdutoDAO produtoDAO;
    public Result usuarios() {
        return ok(usuarios.render(usuarioDAO.todos()));
    }
    public Result produtos() {
        return ok(produtos.render(produtoDAO.todos()));
    }
}
```

```
public class UsuarioDAO {
    //...
    public List<Usuario> todos() {
```

```
        return usuarios.all();
    }
}
```

Agora que temos as views e rotas prontas, vamos autenticar os administradores, começando por alterar o modelo com uma marcação de que um usuário pode ser também administrador. Vamos usar uma `booleana`, mas poderia ser feito com um `enumerador`, e já criar a *Evolution* adequada.

```
public class Usuario ... {
  private boolean admin;
  // getter e setter
}
```

```
# --- !Ups
alter table usuario add column admin bit(1);
# --- !Downs
alter table usuario drop column admin;
```

Enfim podemos criar o autenticador de administradores. Repare que a lógica é bem semelhante à de autenticação de usuários comuns, podendo eventualmente ser extraída.

```
package autenticadores;
public class AdminAutenticado extends Authenticator {
    @Inject
    private UsuarioDAO usuarioDAO;
    @Override
    public String getUsername(Context context) {
        String codigo = context.session().get(AUTH);
        Optional<Usuario> possivelUsuario = usuarioDAO.comToken(codigo);
        if (possivelUsuario.isPresent()) {
            Usuario usuario = possivelUsuario.get();
            if (usuario.isAdmin()) {
                return usuario.getNome();
            }
        }
        return null;
    }
    @Override
    public Result onUnauthorized(Context context) {
        return redirect(routes.UsuarioController.painel());
    }
}
```

Como ambos **ProdutoController** e **AdminController** contêm lógicas que somente devem ser acessíveis por administradores, vamos anotar ambos com `@Authenticated(AdminAutenticado.class)` e enfim temos um sistema plenamente funcional!