

Projeto de Interfaces WEB

App Login em Angular
Aula 08

Introdução

- O objetivo desse mini-projeto é:
 - Criar uma aplicações simples de login;
 - Organizar o projeto em pastas e módulos;
 - Trabalhar com rotas;
 - Impedir acesso não-autorizado a páginas específicas via URL;
 - Mensagens de erro;

Preparando o Projeto

- Crie o módulo “core”
 - `ng g m core`
- Dentro de “core” crie os componentes “login” e “home”.
 - `ng g c core/login --spec=false`
 - `ng g c core/home --spec=false`
- Crie as pastas “models” e “services” no mesmo nível que “core” (atenção, essas pastas NÃO são módulos).
- Em “models”, crie um novo arquivo chamado “Usuario.ts”
- Em “services”, crie os seguintes serviços:
 - `ng g s services/autenticacao`
 - `ng g s services/guarda`
 - `ng g s services/usuario`
- Finalmente, instale as bibliotecas:
 - `npm install bootstrap --save`
 - `npm install @angular/animations --save`
 - `npm install ngx-toastr --save`
- Modifique o arquivo `angular.json`:

```
"styles": [  
  "src/styles.css",  
  "../node_modules/bootstrap/dist/css/bootstrap.css",  
  "../node_modules/ngx-toastr/toastr.css"  
],
```

Preparando o Projeto

- “Cara” do projeto:



A screenshot of a file explorer showing the project structure. The 'app' folder is expanded, revealing subfolders 'core' and 'models', and files 'core.module.ts', 'Usuario.ts', 'services', 'autenticacao.service.ts', 'guarda.service.ts', 'usuario.service.ts', 'app.component.css', 'app.component.html', 'app.component.spec.ts', 'app.component.ts', and 'app.module.ts'. The files are color-coded: TypeScript files are blue, CSS is purple, HTML is orange, and spec files are yellow.

```
└─ app
  └─ core
    ├── home
    ├── login
    └─ core.module.ts
  └─ models
    └─ Usuario.ts
  └─ services
    ├── autenticacao.service.ts
    ├── guarda.service.ts
    └─ usuario.service.ts
  ├── app.component.css
  ├── app.component.html
  ├── app.component.spec.ts
  ├── app.component.ts
  └─ app.module.ts
```

Usuario.ts

- Essa classe representa um “modelo” de um Usuário. É uma ótima prática de programação criar esse tipo de classe em nossos sistemas:

```
export class Usuario{  
  id:number;  
  nome:string;  
  login:string;  
  senha:string;  
}
```

login.html

- Um formulário simples, por enquanto.

```
<h1>Login</h1>
<form>
  <div class="form-group">
    <label for="login">Usuário</label>
    <input type="text" class="form-control" name="login" id="login" required>
  </div>

  <div class="form-group">
    <label for="senha">Senha</label>
    <input type="password" class="form-control" name="senha" id="senha" required>
  </div>

  <button type="submit" class="btn btn-primary">Entrar</button>
</form>
```

login.ts

- Inicialmente:

```
export class LoginComponent {  
  usuario: Usuario = new Usuario();
```

O nosso model. Irá futuramente receber os dados do formulário via [(ngModel)].

```
  constructor(private autenticacaoService: AutenticacaoService,  
               private roteador: Router,  
               private toasty: ToastrService) {}
```

Serviços de autenticação (a ser criado por nós), roteamento (fazer redirects) e de mensagens (toastr)

```
  onSubmit(loginForm: NgForm) {  
  }
```

Método que receberá os dados do formulário e fará o login de fato. Recebe também um NgForm, que representa a tag <form> do Angular.

login.ts

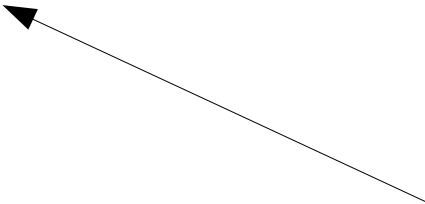
- Imports do login.component.ts:

```
import { Router } from '@angular/router';  
import { ToastrService } from 'ngx-toastr';  
import { NgForm } from '@angular/forms';
```


home.ts

- Componentes simples, apenas para testarmos a segurança do sistema caso o login esteja ok.

```
export class HomeComponent{  
  usuarioLogado:Usuario;  
  
  constructor(private autenticacaoService:AutenticacaoService) {  
  
  }  
  
}
```



Assim como em login.ts, também injetamos o serviço de autenticação em seu construtor.

core.module.ts

```
import { NgModule } from '@angular/core';  
import { FormsModule } from '@angular/forms';  
import { CommonModule } from '@angular/common';  
import { BrowserAnimationsModule } from '@angular/platform-browser/animations';  
import { ToastrModule } from 'ngx-toastr';
```

```
import { LoginComponent } from './login/login.component';  
import { HomeComponent } from './home/home.component';
```

```
@NgModule({  
  declarations: [LoginComponent, HomeComponent],  
  exports: [LoginComponent, HomeComponent],  
  imports: [  
    CommonModule,  
    FormsModule,  
    BrowserAnimationsModule,  
    ToastrModule.forRoot()  
  ]  
})
```

Módulo do Angular para trabalhar com formulários.

Módulo externo para mensagens estilizadas.

usuario.service.ts

- Esse serviço vai ser a interface com o nosso banco de dados. Infelizmente, nessa aula não iremos usar nenhum banco e sim uma variável local. Fica como exercício ao aluno, integrar o login com um banco JSON.

```
export class UsuarioService {  
  
    usuarios: Usuario[] = [  
        {id:0,nome:"Jefferson",login:"jeff",senha:'123'}  
    ];  
  
    getByLogin(login:string):Usuario{  
        for(let u of this.usuarios){  
            if(u.login == login) return u;  
        }  
        return null;  
    }  
}
```

autenticacao.service.ts

- Serviço de autenticação. Ira fazer o login e controlar o objeto usuário logado em “sessão”.
- Vamos analisá-lo por partes:

```
import { BehaviorSubject, Observable } from 'rxjs';
```



Precisamos desses imports para tornar o nosso objeto usuário observável ao resto do sistema.

autenticacao.service.ts

```
export class AutenticacaoService {
```

```
  private usuarioLogadoSubject: BehaviorSubject<Usuario>;
```

```
  public usuarioLogadoObservavel: Observable<Usuario>;
```

```
  constructor(private usuarioService: UsuarioService) {
```

```
    this.usuarioLogadoSubject =
```

```
      new BehaviorSubject<Usuario>(JSON.parse(localStorage.getItem("usuarioLogado")));
```

```
    this.usuarioLogadoObservavel = this.usuarioLogadoSubject.asObservable();
```

```
  }
```

```
//continua no próximo slide...
```

Armazena um valor (value) que pode ser modificado com o tempo. O valor é então emitido a seus observadores.

Objeto observavel para algum inscrito

Recebe um objeto do tipo "Usuario" como valor. Esse objeto foi armazenado localmente, através do objeto "localStorage" (veja próximo slide para mais explicações sobre o "localStorage").

Objeto observável com o valor de BehaviourSubject. Notifica os objetos que se "escreveram" nele caso alguma mudança ocorra.

autenticacao.service.ts

```
//...
autenticar(login:string, senha:string):Usuario{
  let u:Usuario = this.usuarioService.getByLogin(login);
  if(u!=null){
    if(u.senha == senha){
      localStorage.setItem("usuarioLogado",JSON.stringify(u));
      this.usuarioLogadoSubject.next(u);
      return JSON.parse(localStorage.getItem("usuarioLogado"));
    }
  }
  return null;
}
//...
```

Testa login e senha passados como parâmetro com o login e senha do nosso “banco” de UsuarioService.

Armazena um objeto em formato JSON dentro do localStorage, sob a chave “usuarioLogado”

Armazena o usuário logado para que ele possa ser “Observado” (ver próximo slide)

autenticacao.service.ts

```
//...
getUsuarioLogadoValue():Usuario{
  return this.usuarioLogadoSubject.value;
}
```

Retorna o valor dentro do Subject.
Esse método é interessante APENAS se alguma outra classe não precisar monitorar o estado do objeto Usuário.

```
logout(){
  localStorage.removeItem("usuarioLogado");
  this.usuarioLogadoSubject.next(null);
}
} //fim classe
```

Remove o item de localStorage e do Subject. Quem o estiver observando (inscrito), será notificado que ele ficou null.

guarda.service.ts

- O serviço de guarda irá impedir o acesso não autorizado a páginas do sistema.
- Por exemplo, não queremos que um usuário não logado, acesse a página “home.html”


```
//
import { CanActivate, Router, ActivatedRouteSnapshot, RouterStateSnapshot } from '@angular/router';
//...
export class GuardaService implements CanActivate {

  constructor(private autenticacaoService: AutenticacaoService,
               private roteador: Router) {

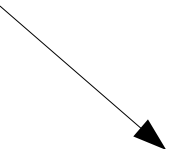
  }

  canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot) {

    if (this.autenticacaoService.getUsuarioLogadoValue()) {
      return true;
    }

    this.roteador.navigate(['/login']);
    return false;
  }

}
```

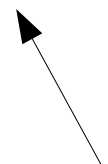


Esse método verifica a lógica de continuar ou não a navegação. Ele retorna true, se está tudo bem de acordo com a lógica da navegação no sistema e false, caso contrário. No nosso caso, será se tiver alguém logado (primeiro if) e, demais casos, redireciona pra página de login.

home.ts

- Vamos voltar agora ao home.component.ts

```
export class HomeComponent{  
  
  usuarioLogado:Usuario;  
  
  constructor(private autenticacaoService:AutenticacaoService) {  
    //this.autenticacaoService.usuarioLogado.subscribe(u => this.usuarioLogado = u);  
    this.usuarioLogado = this.autenticacaoService.getUsuarioLogadoValue();  
  }  
}
```




Aqui podemos escolher entre se inscrever no observável ou apenas verificar o objeto dentro de autenticacaoService. Não iremos fazer nada de especial com o objeto.

login.ts

- Vamos voltar ao login.component.ts

```
//..  
constructor(private autenticacaoService: AutenticacaoService,  
             private roteador: Router,  
             private toasty: ToastrService) {  
  if (this.autenticacaoService.getUsuarioLogadoValue()) {  
    this.roteador.navigate(["/home"]);  
  }  
}  
//...
```



Verifica se já existe algum usuário logado. Se sim, manda diretamente pra home.

login.ts

```
onSubmit(loginForm: NgForm) {
```

```
  if(loginForm.invalid){  
    this.toast.error("Todos os campos devem ser preenchidos.");  
    this.rotador.navigate(["/login"]);  
    return;  
  }
```

Se o formulário for inválido, ou seja, os campos required não foram preenchidos, continua na mesma página.

```
  if (this.autenticacaoService.autenticar(this.usuario.login, this.usuario.senha) != null) {  
    this.toast.success("Login efetuado com sucesso.");  
    this.rotador.navigate(["/home"]);  
  } else {  
    this.toast.error("Login ou Senha inválidos.")  
  }
```

Verifica se o método autenticar retorna algum objeto. Se sim, pela nossa lógica, o login foi um sucesso e redireciona pra home, caso contrário, não.

```
}
```

login.html

(ngSubmit) é um evento chamado quando se clica no botão submit.

```
...  
<form (ngSubmit)="onSubmit(loginForm)" #loginForm="ngForm">
```

Variável que representa o formulário. Em enviada pelo método onSubmit para login.ts

```
  <div class="form-group">
```

```
    <label for="login">Usuário</label>
```

```
    <input type="text" class="form-control" name="login" id="login" required [(ngModel)]="usuario.login" #login="ngModel">
```

```
    <div [hidden]="login.valid || login.pristine" class="alert alert-danger">
```

```
      Login é obrigatório!
```

```
    </div>
```

```
  </div>
```

```
...
```

[(ngModel)] torna possível que os valores do objeto do tipo Usuario, em login.ts, sejam atualizados com os inputs do form.

A variável #login recebe o elemento do form, nesse caso, o input do login. Dessa forma, é possível testar na div abaixo suas propriedades (valid e pristine). Ver slides sobre formulários.

app.component.ts

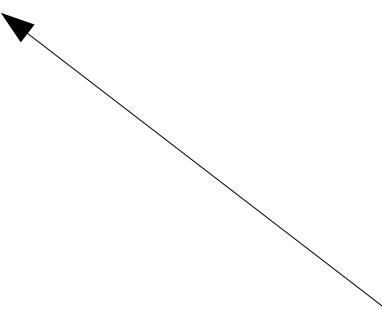
```
export class AppComponent {  
  
  usuarioLogado: Usuario;  
  
  constructor(private autenticacaoService: AutenticacaoService,  
               private roteador: Router){  
    this.autenticacaoService.usuarioLogadoObservavel  
      .subscribe(u => this.usuarioLogado = u);  
  }  
  
  logout(){  
    this.autenticacaoService.logout();  
    this.roteador.navigate(["/login"]);  
  }  
}
```

A variável usuarioLogado se recebe o valor do observavel.
Quaisquer mudanças feitas nesse valor, é avisada.

app.component.html

```
<nav class="navbar navbar-expand navbar-dark bg-dark" *ngIf="usuarioLogado">
  <div class="navbar-nav">
    <a class="nav-item nav-link" routerLink="home">Home</a>
    <a class="nav-item nav-link" (click)="logout()" style="cursor: pointer;">Logout</a>
  </div>
</nav>
```

```
<div class="jumbotron">
  <div class="container">
    <div class="row">
      <div class="col-sm-6 offset-sm-3">
        <router-outlet></router-outlet>
      </div>
    </div>
  </div>
</div>
```



Class do bootstrap para navegação. Apenas é renderizada caso exista um usuário logado.

app.module.ts

```
import { Routes, RouterModule } from '@angular/router';  
import { NgModule } from '@angular/core';  
import { BrowserModule } from '@angular/platform-browser';
```

imports

```
import { CoreModule } from './core/core.module';
```

```
import { AppComponent } from './app.component';  
import { LoginComponent } from './core/login/login.component';  
import { HomeComponent } from './core/home/home.component';  
import { GuardaService } from './services/guarda.service';
```

```
const rotas:Routes =[  
  {path:'login',component:LoginComponent},  
  {path:'home',component:HomeComponent, canActivate: [GuardaService]},  
  {path:'**',redirectTo:'login'}  
]
```

Objeto de rotas. Note que em home, eu recebo um objeto do tipo guarda.

```
@NgModule({  
  declarations: [  
    AppComponent  
  ],  
  imports: [  
    BrowserModule,  
    CoreModule,  
    RouterModule.forRoot(rotas)  
  ],  
  providers: [],  
  bootstrap: [AppComponent]  
})  
export class AppModule { }
```

imports

login.css

```
.ng-valid[required], .ng-valid.required {  
    border-left: 5px solid #42A948; /* green */  
}  
  
.ng-invalid:not(form) {  
    border-left: 5px solid #a94442; /* red */  
}
```

CSS que torna possível a exibição dos erros nos campos requeridos.

Exercício

- Use o JSON-SERVER