Curso: Spring Boot com Angular 7
https://www.udemy.com/user/hugo-silva
Prof. Hugo Silva

Capítulo: Construindo o Front end

Objetivo geral:

- Construir o Front end usando o angular 7
- Autenticação do usuário
- CRUD de usuários
- Refresh Token

Aplicação em camadas

Instalação das ferramentas

O que é Angular?

Site oficial do Angular: https://angular.io/
Site oficial do TypeScript: https://www.typescriptlang.org/

Novidades do Angular 7?

- **CLI Prompts:** https://blog.angular.io/schematics-an-introduction-dc1dfbc2a2b2?gi=72b268e191d3

```
"routing": {
  "type": "boolean",
  "description": "Generates a routing module.",
  "default": false,
```

```
  "x-prompt": "Would you like to add Angular routing?"
},
```

- **Angular Budgets:** https://angular.io/guide/build

```
            "budgets": [
              {
                "type": "initial",
                "maximumWarning": "2mb",
                "maximumError": "5mb"
              }
            ]
```

- **Virtual Scrolling:** https://material.angular.io/cdk/scrolling/overview

```
<cdk-virtual-scroll-viewport itemSize="50" class="example-viewport">
<div *cdkVirtualFor="let item of items" class="example-item">{{item}}</div>
</cdk-virtual-scroll-viewport>
```

- **Drag and Drop:** https://material.angular.io/cdk/drag-drop/overview

```
<div class="example-box" cdkDrag>
  Drag me around
</div>
```

- **Com um simples comando você pode atualizar o seu projeto para o Angular 7**:

```
ng update @angular/cli @angular/core
```

Nota: Informações detalhadas e orientações sobre como atualizar uma aplicação para o Angular 7 podem ser encontradas em: https://update.angular.io

Nota: Para você testar estas novas funcionalidades do Angula7 é preciso adicionar o Angular Material ao projeto através do seguinte comando:

```
ng add @angular/material
```

Checklist:

- Instalar o NodeJS no site oficial (https://nodejs.org)
- Testar o NodeJS:
    - node -v
    - npm -v

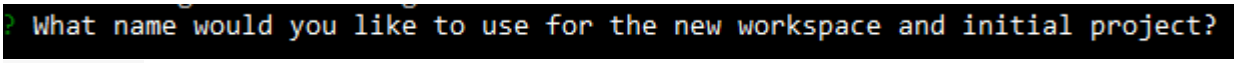
O que é Angular CLI 7?

- Instalar o angular-cli 7 https://cli.angular.io/ (https://www.npmjs.com/package/@angular/cli):
    - `npm install -g @angular/cli`
    - `ng -v`
- Instalar seu editor preferido (sugestão: VS Code: https://code.visualstudio.com)

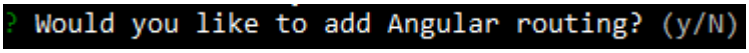
Criando a aplicação

Checklist:

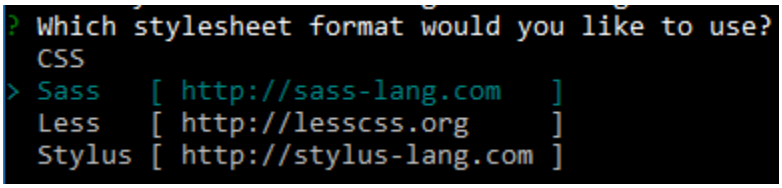
- Abra o terminal no seu computador e execute o comando abaixo para criar o projeto
    - `ng new`



`sba7-app`



Y



    - `cd sba7-app`
- Testar a aplicação
    - `ng serve`
    - `abra o navegado e acesse http://localhost:4200`

Nota: Antes de adicionar qualquer código ao seu projeto, vamos primeiro entender a anatomia do projeto.

`Abra o projeto com seu editor preferido`

- Acessar a pasta do projeto e instalar o MDBootstrap (https://mdbootstrap.com)
    - `npm install angular-bootstrap-md --save`
    - Abra o arquivo **app.module.ts** e adicione as seguintes informações:

```
import { NgModule } from '@angular/core';
```

```
import { MDBBootstrapModule } from 'angular-bootstrap-md';
@NgModule({
imports: [
 MDBBootstrapModule.forRoot()
]});
```

- Adicione as linhas abaixo no **angular.json**

```
"styles": [
  "node_modules/@fortawesome/fontawesome-
free/scss/fontawesome.scss",
  "node_modules/@fortawesome/fontawesome-
free/scss/solid.scss",
 "node_modules/@fortawesome/fontawesome-
free/scss/regular.scss",
 "node_modules/@fortawesome/fontawesome-
free/scss/brands.scss",
 "node_modules/angular-bootstrap-
md/scss/bootstrap/bootstrap.scss",
 "node_modules/angular-bootstrap-md/scss/mdb-free.scss",
 "node_modules/bootstrap/scss/bootstrap.scss",
 "src/styles.scss"],
"scripts": [
"node_modules/chart.js/dist/Chart.js",
"node_modules/hammerjs/hammer.min.js"
],
```

- Instale as seguintes bibliotecas externas
  - `npm install --save chart.js@2.5.0 @types/chart.js @types/chart.js @fortawesome/fontawesome-free hammerjs`

- Para acessar o modo Reactive Extensions For JavaScript
  - `npm install --save rxjs-compat`

Nota: Verificar se as seguintes bibliotecas foram adicionadas no **package.json**

- `"@fortawesome/fontawesome-free"`
- `"@types/chart.js"`
- `"chart.js"`
- `"hammerjs"`
- `"rxjs-compat"`

Criando a estrutura inicial da aplicação

Checklist:

- Atualizar o **angular.json**

```
"styles": [
(...)
"node_modules/bootstrap/scss/bootstrap.scss",
 "src/styles.scss"
],
```

- Instalar o bootstrap
  - `npm install bootstrap --save`
- Criando os componentes da aplicação
  - `ng g c components/login-user --skipTests=true`
  - `ng g c components/register-user --skipTests=true`
  - `ng g c components/resend-registration-token --skipTests=true`
  - `ng g c components/edit-user --skipTests=true`
  - `ng g c components/list-user --skipTests=true`
  - `ng g c components/welcome --skipTests=true`
- Criando os serviços
  - `ng g s core/api --skipTests=true`
  - `ng g s core/interceptor --skipTests=true`
  - `ng g s core/message --skipTests =true`
  - ng g guard core/guards/auth --skipTests=true
- Criando os componentes compartilhados na aplicação
  - `ng g c shared/components/navigation/header --skipTests=true`
  - `ng g c shared/components/modals/delete-user-modal --skipTests=true`
- Criando o arquivo **app.utils.ts** com informações pertinentes a aplicação
  - `shared/comum/ app.utils.ts`

```
import { HttpHeaders } from '@angular/common/http';

export const BASE_URL = 'https://sba7.herokuapp.com';

export const URL_TOKEN = BASE_URL + 'oauth/token';
export const REGISTER_URL = BASE_URL + 'api/public/registration/users';
```

```
export const CONFIRM_REGISTER_URL = BASE_URL + 'api/public/regitrationConfirm/users';
export const RESEND_REGISTER_TOKEN_URL = BASE_URL +
'api/public/resendRegistrationToken/users';

const headersToken = new HttpHeaders({
    Authorization: 'Bearer ' + window.localStorage.getItem('accessToken')
  });
export const OPTIONS_OBJECTO = { headers: headersToken };

export const HEADERS_LOGIN = new HttpHeaders({
    Authorization: 'Basic ' + btoa('cliente' + ':' + '123')
  });

export const HEADERS_REGISTER = new HttpHeaders({
    Authorization: 'Basic ' + btoa('cliente' + ':' + '123')
  });

export const HEADERS_COMMUN = new HttpHeaders({
    Authorization: 'Basic ' + btoa('cliente' + ':' + '123')
  });
```

- Configurando styles.scss para validação dos campos inputs

```
.ng-valid[required], .ng-valid.required  {
    border-left: 5px solid rgba(32, 77, 32, 0.623);
}

.ng-invalid:not(form)  {
    border-left: 5px solid rgb(148, 27, 27);
}


#toast-container > div {
    opacity:1;
}
```

- Criando a pasta de imagens
    - Copiar o arquivo (**man.png**) em anexo nesta aula para a pasta **src/assets/img**

Implementando o modulo de navegação

Checklist:

- Criando as rotas da aplicação no **app-routing.module.ts**

```
const routes: Routes = [
 {path : '', component : LoginUserComponent},
  { path: 'login', component: LoginUserComponent },
  { path: 'register-user', component: RegisterUserComponent },
  { path: 'resend-register-token', component:
ResendRegistrationTokenComponent },
  { path: 'welcome', component: WelcomeComponent },
  { path: 'list-user', component: ListUserComponent },
  { path: 'edit-user/:id', component: EditUserComponent }
];
```

- Editando os componentes **header.component.ts**, **header.component.html** e **header.component.scss**
    - **header.component.scss**

```
.navbar .nav-item.avatar .dropdown-toggle img {
    height: 35px;
}
img.img-fluid.rounded-circle.z-depth-0 {
    height: 35px;
}
.active {
    background:#dc824f;
 }
```

    - **header.component.html**

```
<mdb-navbar SideClass="navbar navbar-expand-lg navbar-dark default-color ie-nav"
[containerInside]="false">
<mdb-navbar-brand>
    <a class="logo navbar-brand" routerLink="/welcome" ><strong>Curso - Spring Boot e
Angular 7</strong></a>
 </mdb-navbar-brand>
  <links>
<ul class="navbar-nav ml-auto nav-flex-icons">
    <li class="nav-item waves-light" mdbWavesEffect>
        <a class="nav-link"    routerLink="login" routerLinkActive="active">Login</a>
    </li>
    <li class="nav-item waves-light" mdbWavesEffect>
        <a class="nav-link"  routerLink="register-user"
routerLinkActive="active">Registre-se</a>
    </li>
</ul>
<ul class="navbar-nav ml-auto nav-flex-icons" >
    <li class="nav-item waves-light py-1 mx-3" mdbWavesEffect>
        <a class="nav-link"   routerLink="list-user"
routerLinkActive="active">Usuários</a>
```

```
    </li>
    <li class="nav-item dropdown" dropdown>
        <a dropdownToggle mdbWavesEffect type="button" class="nav-link dropdown-
toggle waves-light" mdbWavesEffect>
                <img src="../../assets/img/man.png" class="img-fluid rounded-circle
z-depth-0">
                <span class="caret"></span>
        </a>
        <div *dropdownMenu class="dropdown-menu dropdown dropdown-primary"
role="menu">
            <a class="dropdown-item waves-light" mdbWavesEffect
(click)="logout()">Sair</a>
        </div>
    </li>
 </ul>
  </links>
</mdb-navbar>
```

- Editar o componente **app.component.html**

```
<app-header></app-header>
<!-- Main -->
<main>
    <div class="container-fluid ">
        <router-outlet></router-outlet>
    </div>
</main>
<!-- /.Main -->
```

- Testar a aplicação
    - `ng serve`
    - `abra o navegado e acesse http://localhost:4200`

Entendendo os componentes do Angular

```
TS app.component.ts ×
 1   import { Component } from '@angular/core';
 2
 3   @Component({
 4     selector: 'app-root',
 5     templateUrl: './app.component.html',
 6     styleUrls: ['./app.component.scss']
 7   })
 8   export class AppComponent {
 9     title = 'sba7-app';
10   }
11
```

- **@Component():** decorador que pode ser importado de **@angular/core.** O decorador de componentes leva as seguintes informações:
    - ***selector*:** contém o nome da tag que pode ser usada para criar este componente.
    - ***templateUrl:*** contém a URL - caminho relativo para o modelo HTML que será usado na visualização do componente
    - ***styleUrls*:** contém a matriz de estilos SCSS a serem usados para estilizar o componente

- **app.component.scss** : contém todos os estilos SCSS para o componente
- **app.component.html** : contém todo o código HTML usado pelo componente para se exibir
- **app.component.ts** : contém todo o código usado pelo componente para controlar seu comportamento
- Este componente pode ser chamado em seu código HTML como qualquer tag HTML padrão, ou seja:

```
<body>
  <app-root></app-root>
</body>
```

Implementando a tela de login

Checklist:

- Importar **HttpClientModule**, **AppRoutingModule**, **ReactiveFormsModule** e **FormsModule** no módulo principal
- Importar **ApiService** no módulo principal
- Criar a entidade **login.ts** da aplicação na pasta core/model/**login.ts**
- Criar o método **login(user): Observable <any> {} e getMainUser(token: any): Observable<any> {}** no serviço **api.service.ts**
- Importar **HttpClient**, **HttpParams** em **api.service.ts**
- Implementar os métodos **public login() {}, public loginSuccess(data: any) {}** e **public redirectPage(res: any) {}** no componente **login-user.component.ts**

Implementando a tela de boas vindas

Checklist:

- Criar a entidade **userDTO.ts** da aplicação na pasta core/model/**userDTO.ts**
- Editar os componentes **welcome.component.ts** e **welcome.component.html**

Implementando o serviço Interceptor

Checklist:

- Implementar o método **getAccessToken(refreshToken): Observable<any> {}** no serviço **api.service.ts**

- O serviço **interceptor.service.ts** deve implementar **HttpInterceptor**
- Implementar os métodos **intercept(request: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {}, private getAccessToken(req: HttpRequest<any>, next: HttpHandler): Observable<any> {}, handleErrorGeneral(error) {}** e **handle303Error(error) {}** no serviço **interceptor.service.ts**
- Importar **InterceptorService** módulo principal

```
providers: [ApiService,
  {
   provide: HTTP_INTERCEPTORS,
    useClass: InterceptorService,
    multi : true
  }
],
```

Implementando o Route Auth Guards

Checklist:

- Implementar o método **isAuthenticated(): Observable<boolean> {}** no serviço **api.service.ts**
- Criar o método **canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot) {}** no **auth.guard.ts**
- Criar o método **canLoad(route: Route) {}** no **auth.guard.ts**
- Importar **AuthGuard** no módulo principal

Desabilitar os menus de Usuários e Logout

Checklist:

- Implementar o método **isAutenticated(): Observable<boolean> {}** no componente **header.component.ts**
- Desabilitar os menus de Usuários e Logout quando o usuário não estiver autenticado em **header.component.html**
- Atualizar o **app-routing.module.ts** com: **canActivate: [AuthGuard], canLoad: [AuthGuard]** para as rotas **WelcomeComponent, ListUserComponent e EditUserComponent**

Registrando usuários

Checklist:

- Implementar o método **registerUser(user: User): Observable<any> {}** no serviço **api.service.ts**
- Implementar o método **save(): void {}** no componente **register-user.component.ts**
- Implementar o componente **register-user.component.html**

Confirmando registro de usuário

Checklist:

- Implementar o método **confirmationRegisterToken(token: string): Observable<any> {}** no serviço **api.service.ts**
- Criar o componente **register-confirmation.component.ts** em **components/register-user** e implementar o método de verificação de registro em **ngOnInit() {}**

Reenvio de token quando o mesmo expirar

Checklist:

- Implementar o método **resendRegisterToken(user: User): Observable<any> {}** no serviço **api.service.ts**
- Implementar o método **resendToken() {}** no componente **resend-registration-token.component.ts**
- Implementar o componente **resend-registration-token.component.htm**

Relação de usuários

Checklist:

- Implementar os métodos **getUsers(): Observable<any> {} e getRole(roles: Array<any>) {}** no serviço **api.service.ts**
- Implementar o método **ngOnInit() {}** para retornar a lista de usuários e o **getRole(user: User) {}** para retornar o **ROLES** dos usuários no componente **list-user.component.ts**
- Implementar o componente **list-user.component.html**

Deleção de usuários

Checklist:

- Implementar o método **deleteUser(id: string): Observable<any> {}** no serviço **api.service.ts**
- Implementar os métodos **show() {} e delete() {}** no componente **delete-user-modal.component.ts**
- Implementar o componente **delete-user-modal.component.html**
- Atualizar o componente **list-user.component.html** com as seguintes tags html

```
<div class="btn-group btn-group-sm">
<button class="btn btn-danger " (click)="deleteUserModal.show()"> Excluir</button>
```

```
</div>
```

```
<app-delete-user-modal #deleteUserModal [recebeItem]="user" (resposta)="deleteUser($event)"
></app-delete-user-modal>
```

Atualização de usuários

Checklist:

- Implementar os métodos **getUserById(id: string): Observable<any> {} e updateUser(user: User): Observable<any> {}** no serviço **api.service.ts**
- Implementar os métodos **ngOnInit() {} e update() {}** no componente **edit-user.component.ts**
- Implementar o componente **edit-user.component.html**
- Atualizar o componente **list-user.component.html** com as seguintes tags html

```
<button class="btn btn-info " [routerLink]="['/edit-user', user.id]" style="margin-left:
20px;"> Editar</button>
```

Logout de usuários

Checklist:

- Implementar o método **logout(): Observable<any> {}** no serviço **api.service.ts**
- Implementar os métodos **logou() {} e clearLocalStore () {}** no componente **header.component.ts**
- Testar aplicação

Implementando o serviço Mensagem

Checklist:

- Instalar biblioteca para manipular mensagens: https://www.npmjs.com/package/ngx-toastr

  `npm install ngx-toastr --save`

- Antes de instalar esta biblioteca verificar se já está instalada no **package.json**

  `npm install @angular/animations --save`

- Importar no módulo principal o modulo **BrowserAnimationsModule e**

```
ToastrModule.forRoot({
      timeOut: 4000,
```

```
      positionClass: 'toast-top-center'
    })
```

- Implementar os métodos **showSuccess(titulo, message) {}, showWarning(titulo, message) {} e showError(titulo, message) {}** no serviço **message.service.ts**
- Importar **MessageService** módulo principal

Atualizar os Interceptos e componentes com serviços de Mensagens

Checklist:

- Atualizar os métodos do serviço **interceptor.service.ts** com os serviços de mensagens
- Atualizar os métodos do componente **edit-user.component.ts** com os serviços de mensagens
- Atualizar os métodos do componente **list-user.component.ts** com os serviços de mensagens
- Atualizar os métodos do componente **login-user.component.ts** com os serviços de mensagens
- Atualizar os métodos do componente **register-user.component.ts** com os serviços de mensagens
- Atualizar os métodos do componente **register-confirmation.component.component.ts** com os serviços de mensagens
- Atualizar os métodos do componente **resend-registration-token.component.ts** com os serviços de mensagens

Observar e notificar ações na aplicação

Checklist:

- Atualizar os métodos do serviço **message.service.ts** com **Observable** e **Subject** da biblioteca ' rxjs'
- Agora implemente **unsubscribeMessage = new Subject()** em todos os componentes que vocês deseja ser notificado
- Exemplo do botão carregando enquanto aguarda resposta da API REST.

```
<button *ngIf="submitted" class="btn btn-dark btn-block" type="button">
    <span class="spinner-border spinner-border-sm" role="status" aria-hidden="true"></span>
    Carregando...
</button>
```