Routing

Como criar rotas personalizadas no Angular

Routing

- Processo de determinar como um aplicativo responde a um URL ou caminho específico;
- Realiza o mapeamento de URLs para diferentes componentes dentro do aplicativo e a renderização do conteúdo apropriado com base no URL solicitado;
- Em um aplicativo Web de página única (SPA), o roteamento permite que os usuários naveguem entre diferentes visualizações ou páginas sem realmente carregar uma nova página HTML do servidor;
- O aplicativo atualiza dinamicamente o conteúdo do navegador, carregando os componentes e dados necessários com base na rota solicitada.

Vantagens

- Experiência de usuário contínua e interativa, permitindo que os usuários naveguem entre diferentes visualizações ou páginas dentro do aplicativo sem recarregar a página inteira;
- Apenas os componentes e dados necessários para a rota solicitada são carregados, resultando em transições de **página mais rápidas;**
- Incentiva uma estrutura modular para o aplicativo, separando-o em diferentes visualizações ou componentes associados a rotas específicas, promovendo a reutilização do código e a melhor capacidade de manutenção;
- Cada rota pode ter seu próprio componente, facilitando o gerenciamento e a atualização de seções específicas do aplicativo de forma independente;

Vantagens

- Permite a renderização condicional de componentes com base na rota atual permitindo mostrar ou ocultar certas seções do aplicativo com base no caminho de navegação do usuário;
- Suporte à passagem de parâmetros de rota e parâmetros de consulta que permitem que valores dinâmicos passados para dentro do URL, como um ID ou nome de usuário, e recupere-os no componente correspondente, os parâmetros de consulta fornecem uma maneira de transmitir dados adicionais ao URL para filtragem, classificação ou outros fins;
- Protetores de rota (route guards), que são mecanismos para controlar o acesso a rotas específicas com base em determinadas condições. Os protetores de rota podem ser usados para autenticação, autorização e outros fins relacionados à segurança. Eles ajudam a garantir que os usuários só possam acessar rotas ou executar ações se atenderem aos critérios necessários.

Vantagens

- Suporte a rotas aninhadas ou secundárias, permitindo definir uma estrutura de navegação hierárquica dentro do aplicativo. Isto é particularmente útil ao lidar com aplicações complexas com múltiplos níveis de navegação ou seções que precisam ser encapsuladas e gerenciadas de forma independente.

Routes

- Definem o mapeamento entre o caminho da URL e os componentes correspondentes a serem renderizados. Cada rota é definida com um caminho de URL e um componente correspondente a ser exibido quando esse caminho for acessado.

```
const rotasApp: Routes = [
  { path: 'aviao', component: AviaoComponent },
  { path: 'carro', component: CarroComponent },
  { path: 'barco', component: BarcoComponent },
];
```

Router

- O router é responsável por interpretar a URL atual e carregar os componentes apropriados com base nas rotas definidas. Ele escuta alterações de URL e controla a navegação dentro do aplicativo.

```
export class AppComponent {
  title = '1.0_RouterNavigation';
  rota: string = '';
  constructor(private rotas:Router, private rotaAtiva: ActivatedRoute){

  }

  paraAviao(){
    this.rotas.navigate(['aviao']);
  }
}
```

```
imports: [
  BrowserModule,
  //registrar as rotas no Angular
  RouterModule.forRoot(rotasApp)
],
```

```
<router-outlet></router-outlet>
```

Router Outlet

Espaço reservado no modelo da aplicação onde o conteúdo da rota atual é renderizado.

Router Links and Navigation

- Links e elementos de navegação, como tags <a> ou botões, são usados para acionar a navegação para diferentes rotas dentro do aplicativo. Esses elementos podem ser decorados com diretivas como routerLink em Angular para especificar a rota alvo.

```
<a routerLink="/aviao" routerLinkActive="ativo">Avião</a>
<br/>
<a routerLink="/carro" routerLinkActive="ativo">Carro</a>
<br/>
<a routerLink="/barco" routerLinkActive="ativo">Barco</a>
<br/>
<hr>
<button (click)="paraAviao()">Mostrar Aviao</button>
```

```
paraAviao(){
  this.rotas.navigate(['aviao']);
}
```

Passando parâmetros para as rotas

- Existem muitos cenários em que é preciso passar parâmetros para uma rota;
- Exemplo: navegar até a visualização de detalhes de um animal, é preciso passar o id do Animal, para que o componente possa recuperá-lo e exibi-lo ao usuário;
  - É aqui que os parâmetros de rota são utilizados

Passando parâmetros para as rotas

- Considere a rota:

```
{ path: 'animal', component: AnimalComponent },
```

- Para recuperar os detalhes do animal, nosso URL deve ser semelhante:
  - /animal/13
  - /animal/14
- Onde os números 13 e 14 é o id do animal que muda de acordo com o animal selecionado;
- Para lidar com essa situação, é possível incluir parâmetros de rota, onde é possível enviar qualquer valor dinâmico para uma parte da URL.

Passando parâmetros para as rotas

- É possível definir parâmetros na rota, adicionando uma barra seguida de dois pontos e um espaço reservado (id):

```
{ path: 'animal/:id', component: AnimalComponent },
```

- É possível incluir mais de um parâmetro, adicionando mais uma barra seguida de dois pontos e um espaço reservado;

```
{ path: 'animal/:id/:id2/:id3', component: AnimalComponent },
```

Passando parâmetros para as rotas

- Fornecer o caminho e a diretiva routerLink do parâmetro de rota, isso é feito adicionando o id do animal como o segundo elemento à matriz de parâmetros do routerLink:

```
<a [routerLink]="['/animal', `1`]" routerLinkActive="ativo">Para animal</a>
```

```
<input type="text" (input)="onInputChange($event)" name="idTexto">
<a [routerLink]="['/animal', inputText]" routerLinkActive="ativo">Para animal</a>
```

```
paraAnimal() {
  this.rotas.navigate(['animal', this.inputText]);
}
```

Lendo os parâmetros da rota

- Existem duas maneiras pelas quais é possível ler os parâmetros da rota:
  - Usando a propriedade **snapshot** da classe ActivatedRoute;
  - Inscrever-se na propriedade **observable** paramMap ou params do ActivatedRoute

```
ngOnInit() {
  this.animal = {
    id: this.rota.snapshot.params['id'],
  }
```

```
ngOnInit() {
  this.rota.paramMap.subscribe(params => {
    this.animal.id = Number(params.get('id'));
  });
}
```

Snapshot vs Observable

- Quando o usuário navega para o componente novamente, e se o componente já estiver carregado, o Angular não cria o novo componente, mas reutiliza a instância existente. Nessas circunstâncias, o método ngOnInit do componente não é chamado novamente. É preciso de uma maneira de obter o valor do parâmetro;
- Ao assinar o paramMap, uma notificação é recebida quando o valor mudar. Então, é possível recuperar o valor mais recente do parâmetro e atualizar o componente de acordo.

Parâmetros de consulta com router

- Os parâmetros de consulta permitem a passagem de parâmetros opcionais em qualquer rota do aplicativo;
  - Utilizando QueryParams com Roter.navigate:

```
paraAnimalPesados() {
  this.rotas.navigate(
    ['animal'],
    {queryParams: {order: 'pesados'}}
  );
}
```

localhost:4200/animal/?order=pesados

  - Utilizando RouterLink:

```
<a [routerLink]="['/animal', inputText]"
   [queryParams]="{order: 'pesados'}" routerLinkActive="ativo">Para animal</a>
```

localhost:4200/animal/1?order=pesados

Acessando valores de parâmetros de consulta

- Utilizando observable:

```
this.rota.queryParams.subscribe( queryParams => {
    console.log(queryParams);
});
```

- Utilizando snapshot:

```
console.log(this.rota.snapshot.queryParams);
```

Rotas filhas (Child Routes) /rotas aninhadas (Nested Routes)

- À medida que o aplicativo se torna mais complexo, é possível criar rotas relativas a um componente diferente do seu componente raiz;
- Esses tipos de rotas aninhadas são chamadas de rotas secundária;
- Isso significa que é adicionado um segundo <router-outlet> ao aplicativo, porque ele é um acréscimo ao <router-outlet> no AppComponent.

```
{ path: 'animal', component: AnimalComponent, children: [
  { path: ':id', component: AnimalComponent },
  { path: ':id/editar', component: AnimalEditarComponent }
]},
```

Mostrando uma página “404”

- A última rota com o caminho ** é uma rota curinga. O router seleciona essa rota se o URL solicitado não corresponder a nenhum dos caminhos anteriores na lista e envia o usuário para PageNotFoundComponent.

```
const routes: Routes = [
  { path: 'first-component', component: FirstComponent },
  { path: 'second-component', component: SecondComponent },
  { path: '**', component: PageNotFoundComponent },  // Wildcard route for a 404 page
];
```

Exercício 1

- Crie uma aplicação em Angular que tenha um menu com as seguintes páginas: home, sobre, serviços. Crie as um componente para cada página, insira rotas para acessar essas páginas e exiba no componente principal. Se possível, utilize a biblioteca material para dar uma melhor aparência na sua aplicação.

Referências

- https://medium.com/@jaydeepvpatil225/routing-in-angular-924066bde43
- https://angular.io/guide/router
- https://angular.io/guide/routing-overview
- https://angular.io/tutorial/tour-of-heroes/toh-pt5
- https://angular.io/api/router/CanActivate