Angular

Pipes, criação de componentes, módulos e exibição mestre detalhe.

Ely – elydasilvamiranda@gmail.com

Usando Pipes

- Pipes são elementos aplicados aos bindings que permitem realizar modificações no texto exibido;
- Por exemplo, para transformar o nome do herói para caixa alta, utilizamos pipe uppercase:

```
<h2>{{hero.name | uppercase}}</h2>
```

- O pipe se inicia com a barra | e é seguido de um nome.

https://angular.io/guide/pipes

Editando um componente

- Os usuários devem poder editar o nome do herói em uma caixa de texto <input>;
- A caixa de texto deve:
  - Exibir a propriedade name do herói;
  - Atualizar essa propriedade conforme o usuário digita;
- Dessa forma:
  - O fluxo de dados da classe Hero para a tela e da tela de volta para a classe;
  - Chamamos esse fluxo de dados de two-way-binding.

Editando um componente

- Através da diretiva [(ngModel)] no HTML é possível fazer um two-way data binding;
- Criamos uma input em heroes.componet.html e adicionamos a diretiva [(ngModel)] :

```
<h2>{{hero.name | uppercase}} Details</h2>
<div><span>id: </span>{{hero.id}}</div>
<div>
   <label>name:
      <input [(ngModel)]="hero.name"
      placeholder="name"/>
   </label>
</div>
```

Editando um componente

- Dessa forma, o que for digitado na input será associado à propriedade hero.name;
- E também, o valor que estiver definido em hero.name será exibido na caixa de texto.

FormsModule

- Perceba que o app parou de funcionar quando foi adicionado o [(ngModel)];
- Para ver o erro, abra o console das ferramentas de desenvolvimento do navegador e procure por:
  - Template parse errors: Can't bind to 'ngModel' since it isn't a known property of 'input'.
- Embora ngModel seja uma diretiva Angular válida, ela não está disponível por padrão;
- Ele pertence ao FormsModule e devemos importá-lo.

AppModule

- O Angular usa decoradores para organizar as partes de um app em módulos e componentes;
- Dois decoradores importantes são o @Component e @NgModule:
  - Os @Component estão presentes nas classes dos componentes;
  - Já os @NgModule estão presentes nos módulos;

AppModule

- O mais importante decorador @NgModule está presente na classe AppModule;
- Esse é o módulo raiz de toda app Angular;
- O Angular CLI gerou essa classe em src/app/app.module.ts quando criou o projeto;
- É nessa classe aqui que importamos o FormsModule.

Importando o FormsModule

- Em AppModule (app.module.ts), importe o FormsModule da biblioteca @ angular/forms:

```
import { FormsModule } from '@angular/forms';
```

- Em seguida, adicione FormsModule à diretiva @NgModule:

```
imports: [
    BrowserModule, FormsModule
],
```

- Nessa lista estão todos os módulos externos que o aplicativo precisa.

Importando o FormsModule

- Quando o navegador atualizar, o app voltará a funcionar;
- Será possível também editar o nome do herói e ver a alteração imediatamente no <h2>.

AppModule.ts

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';

import { AppComponent } from './app.component';
import { HeroesComponent } from './heroes/heroes.component';

@NgModule({
    declarations: [
        AppComponent,
        HeroesComponent
    ],
    imports: [
        BrowserModule,
        FormsModule
    ],
    providers: [],
    bootstrap: [AppComponent]
})
export class AppModule { }
```

Exibindo uma lista de heróis

- Vamos expandir a aplicação para exibir uma lista de heróis;
- Também será possível selecionar um herói e exibir seus detalhes.

Criando um “mock” de heróis

- Precisaremos de alguns heróis para exibir;
- O ideal seria obter os objetos de um servidor de dados remoto;
- Por enquanto, criaremos alguns heróis falsos e como se viessem do servidor:
  - Crie um arquivo chamado mock-heroes.ts na pasta src / app /;
  - Defina uma constante HEROES como um array de dez heróis e exporte-a conforme o próximo slide.

Criando um “mock” de heróis

- src/app/mock-heroes.ts

```
import { Hero } from ‘./heroes/hero’;
export const HEROES: Hero[] = [
    { id: 11, name: 'Mr. Nice' },
    { id: 12, name: 'Narco' },
    { id: 13, name: 'Bombasto' },
    { id: 14, name: 'Celeritas' },
    { id: 15, name: 'Magneta' },
    { id: 16, name: 'RubberMan' },
    { id: 17, name: 'Dynama' },
    { id: 18, name: 'Dr IQ' },
    { id: 19, name: 'Magma' },
    { id: 20, name: 'Tornado’ }
];
```

Exibindo os heróis

- Precisamos importar o “mock” de heróis no HeroesComponent;

```
import { HEROES } from '../mock-heroes';

//...

export class HeroesComponent implements OnInit {
    heroes = HEROES;

    //...

}
```

Listar heróis com *ngFor

- No arquivo heroes.componente.html faremos as seguintes alterações:
  - Adicione um <h2> no topo;
  - Abaixo, adicione uma lista não ordenada de HTML (<ul>);
  - Insira um <li> dentro do <ul> que exibe as propriedades de um herói;
- O arquivo HTML deve ser semelhante ao do próximo slide.

Listar heróis com *ngFor

```
<h2>My Heroes</h2>
<ul class="heroes">
    <li *ngFor="let hero of heroes">
        <span class="badge">
            {{hero.id}}
        </span> {{hero.name}}
    </li>
</ul>
```

Listar heróis com *ngFor

- O * ngFor é a diretiva de repetição do Angular;
- Ela repete o elemento host para cada elemento em uma lista;
- Neste exemplo, <li> é o elemento host:
  - **heroes** é a lista da classe HeroesComponent;
  - **hero** representa o objeto herói atual para cada iteração na lista.

Aplicando estilos

- Nos slides da aula anterior, foram definidos os estilos básicos para todo o app em styles.css;
- É preferível que estilos específicos de determinados componentes:
  - Os estilo “privados” são definidos no CSS definido no próprio componente;
  - Dessa forma, se mantém tudo o que um componente precisa - o código, o HTML e o CSS - juntos em um só lugar.
  - Essa abordagem facilita a reutilização do componente em outro local.

Aplicando estilos

- Estilos são definidos via de regra no arquivo de folha de estilo no array @ Component.styleUrls;
- Quando o CLI gerou o HeroesComponent, ele criou uma folha de estilo heroes.component.css.

Arquivo heroes.component.css

```css
/* HeroesComponent's private CSS styles */
.selected {
  background-color: #CFD8DC !important;
  color: white;
}
.heroes {
  margin: 0 0 2em 0;
  list-style-type: none;
  padding: 0;
  width: 15em;
}
.heroes li {
  cursor: pointer;
  position: relative;
  left: 0;
  background-color: #EEE;
  margin: .5em;
  padding: .3em 0;
  height: 1.6em;
  border-radius: 4px;
}
.heroes li.selected:hover {
  background-color: #BBD8DC !important;
  color: white;
}
.heroes li:hover {
  color: #607D8B;
  background-color: #DDD;
  left: .1em;
}
.heroes .text {
  position: relative;
  top: -3px;
}
.heroes .badge {
  display: inline-block;
  font-size: small;
  color: white;
  padding: 0.8em 0.7em 0 0.7em;
  background-color: #607D8B;
  line-height: 1em;
  position: relative;
  left: -1px;
  top: -4px;
  height: 1.8em;
  margin-right: .8em;
  border-radius: 4px 0 0 4px;
}
```

Mestre/detalhe

- Ao clicar em um herói na lista principal:
  - O componente deve exibir os detalhes do herói;
  - Essa exibição será na parte inferior da página;
- Iremos configurar o evento de clique em um herói e atualizar os detalhes;
- Para exibir os detalhes, criaremos um novo componente:

  >> ng generate component hero-detail

Adicionando o evento de click

- Podemos fazer um binding de um evento de clique da seguinte forma:

```
<li *ngFor="let hero of heroes" (click)="onSelect(hero)">
```

- Os parênteses em torno de click dizem ao Angular para escutar o evento de clique do elemento <li>;
- Quando o usuário clica no <li>, o Angular executa a expressão onSelect (hero);
- onSelect () será um método HeroesComponent.

Criando o método onSelect

- Criaremos uma propriedade chamada selectedHero;
- Essa propriedade receberá o herói clicado;
- Para passar o herói clicado do template para o componente:
  - Declararemos o método em hero-componet.ts;
  - Receberemos como um dos parâmetros um objeto Hero que foi selecionado;
  - Atualizaremos a propriedade selectedHero.

Criando o método onSelect

- Arquivo heroes.componet.ts:

```
// ...
export class HeroesComponent implements OnInit {

  heroes = HEROES;
  selectedHero: Hero;

  constructor() { }
  ngOnInit() {}

  onSelect(hero: Hero): void {
    this.selectedHero = hero;
  }
}
```

Template com o onSelect

- Arquivo hero.component.html

```
<h2>My Heroes</h2>

<ul class="heroes">
  <li *ngFor="let hero of heroes"
    [class.selected]="hero == selectedHero"
    (click)="onSelect(hero)">
    <span class="badge">{{hero.id}}</span> {{hero.name}}
  </li>
</ul>

<app-hero-detail [hero]="selectedHero">
</app-hero-detail>
```

Atualizando o componente detalhe

- A tag abaixo pega a propriedade selectedHero de HeroComponent após o clique;

```
<app-hero-detail [hero]="selectedHero">
</app-hero-detail>
```

- Essa propriedade hero deve existir no componente HeroDetail;
- Faremos o binding usando o decorador @Input:

```
@Input() hero: Hero;
```

Component HeroDetail

- Arquivo hero-detail-component.ts:

```
import { Component, OnInit, Input } from '@angular/core';
import { Hero } from '../heroes/hero’;
//...
export class HeroDetailComponent implements OnInit {
  @Input()
   hero: Hero;
  constructor() { }

  ngOnInit() {
  }
```

Atualizando o template do detalhe

- Arquivo: hero-detail-component.html:

```
<div *ngIf="hero">

  <h2>{{hero.name | uppercase}} Details</h2>
  <div><span>id: </span>{{hero.id}}</div>
  <div>
    <label>name:
      <input [(ngModel)]="hero.name"
             placeholder="name"/>
    </label>
  </div>

</div>
```

Atualizando o template de detalhe

- Caso nenhum herói tenha sido selecionado, ocorre um erro:

HeroesComponent.html:3 ERROR TypeError: Cannot read property 'name' of undefined

- A fim de evitar isso, a diretiva *ngIf foi adicionada.

Conclusão

- Quando o aplicativo é iniciado, o selectedHero é nulo por padrão;
- Bindings que se referem a propriedades de selectedHero {{selectedHero.name}} falham pois não há nenhum herói selecionado;
- Agora, ao clicar em um dos itens da lista, o herói clicado aparece na parte inferior da página.

Angular

Pipes, criação de componentes, módulos e exibição mestre detalhe.

Ely – elydasilvamiranda@gmail.com