
# Apresentando uma Lista de Tarefas no TODOapp

UNISANTA

# Índice

# Apresentando uma lista de tarefas

Agora vamos preparar nosso App para mostrar uma lista de "Tarefas a Fazer". Cada tarefa deverá possuir duas informações importantes:

- Uma Descrição
  Tipo: string (Ex: "Jogar Futebol com os amigos")

- Um Status
  Tipo: boolean (Ex: "true" = "Realizada" e "false" = "Não Realizada")

Para começar, vamos então criar uma "classe" que determinará que uma "Tarefa" deva possuir as informações definidas acima. Para isso, vamos criar um arquivo chamado "**tarefa.ts**" dentro da pasta "**TODOapp/src/app**" com o seguinte conteúdo:

```
export class Tarefa {
    descricao: string;
    statusRealizada: boolean;

    constructor(_descricao: string, _statusRealizada: boolean) {
        this.descricao = _descricao;
        this.statusRealizada = _statusRealizada;
    }
}
```

Perceba que o código acima define uma classe chamada de "Tarefa". Essa classe possui dois atributos: Um chamado de "descricao" do tipo "string" e outro chamado de "statusRealizada" do tipo boolean.

Adicionalmente a classe define um "método construtor" que recebe uma "descrição" e um "status" como parâmetros. Dentro do corpo desse método, os parâmetros são armazenados nos atributos da classe.

As definições acima permitirão que nosso App manipule facilmente instâncias (objetos) do tipo "Tarefa".

A seguir, vamos inserir as linhas abaixo destacadas em verde, no conteúdo do arquivo **app.component.ts**:

```
import { Component } from '@angular/core';
import { Tarefa } from "./tarefa";

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'TODOapp';

  arrayDeTarefas: Tarefa[] = [];

  constructor() {
    this.READ_tarefas();
  }

  READ_tarefas() {
    this.arrayDeTarefas = [
      new Tarefa("Estudar Frameworks WEB", false),
      new Tarefa("Comer Pizza", false),
      new Tarefa("Ajudar meus pais", false)
    ];
  }

}
```

O código adicionado (em verde), realiza as seguintes atividades:

- A linha do "import" permite que o AppComponent utilize as definições realizadas na classe "Tarefa".
- A variável "arrayDeTarefas" foi definida como um array vazio para armazenar objetos do tipo "Tarefa".
- O método READ_tarefas armazena 3 tarefas dentro do array. O nome deste método pode parecer estranho neste momento, porém ele simula o processo de "Leitura" de tarefas armazenadas em algum tipo de repositório persistente.
- O construtor executa o método READ_tarefas durante a inicialização do programa, garantido que o array seja populado com alguns objetos para a realização de testes.

Agora vamos permitir que as tarefas sejam apresentadas no browser. Para isso vamos definir o seguinte conteúdo para o arquivo **app.component.html:**

```
<div class="main">
    <h1>Mozilla Dev - My To Do List</h1>
    <ul>
        <li *ngFor="let tarefaDoLoop of arrayDeTarefas">
            {{tarefaDoLoop.descricao}}
        </li>
    </ul>
</div>
```

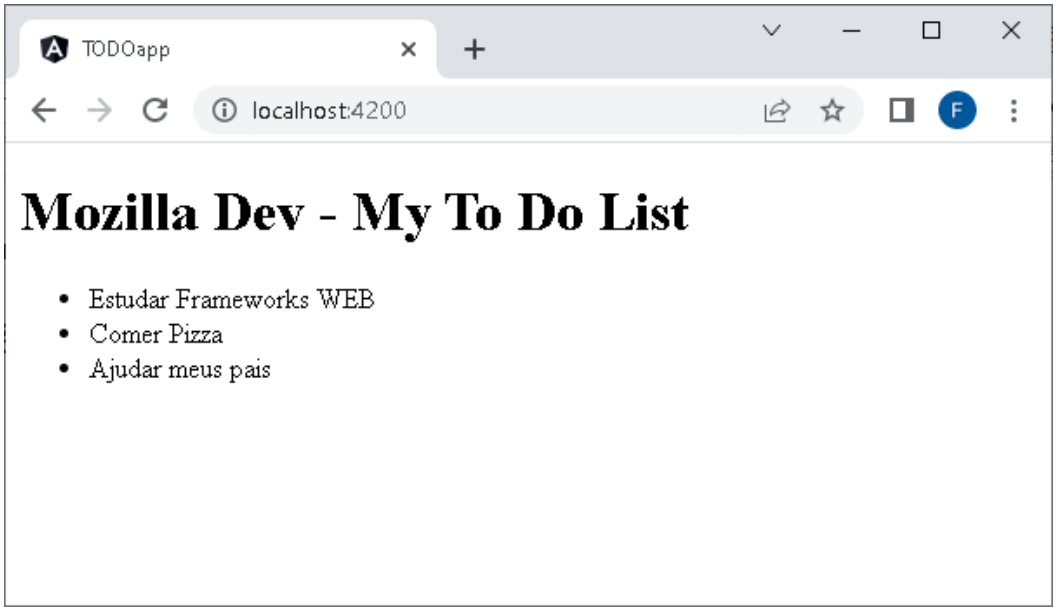
Perceba que o código acima faz uso de uma diretiva Angular de repetição chamada de "ngFor" juntamente com o código HTML. Esta diretiva irá criar um elemento HTML <li> para objeto do tipo "Tarefa" armazenado em "arrayDeTarefas". Adicionalmente, essa linha faz uso também de uma estrutura de "Data Binding" do Angular chamada de "interpolação", que faz uso das chaves duplas para apresentar na view um determinado valor proveniente da lógica do componente (neste caso, o atributo "descricao" do objeto representado pela variável "tarefaDoLoop" da diretiva "ngFor"). Esses detalhes serão explicados de forma mais apropriada na próxima aula.

## Testando no Browser

Abra novamente o browser utilizando o mesmo endereço:

**http://localhost:4200/**

Resultado esperado:



# Adicionando novas tarefas

Para permitir ao usuário inserir novas tarefas, vamos criar um novo método na lógica do AppComponent que permita armazenar uma nova tarefa no array. Para isso, adicione as linhas abaixo destacadas em verde, no conteúdo do arquivo **app.component.ts**:

```
import { Component } from '@angular/core';
import { Tarefa } from "./tarefa";

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'TODOapp';

  arrayDeTarefas: Tarefa[] = [];

  constructor() {
    this.READ_tarefas();
  }

  CREATE_tarefa(descricaoNovaTarefa: string) {
    var novaTarefa = new Tarefa(descricaoNovaTarefa, false);
    this.arrayDeTarefas.unshift(novaTarefa);
  }

  READ_tarefas() {
    this.arrayDeTarefas = [
      new Tarefa("Estudar Frameworks WEB", false),
      new Tarefa("Comer Pizza", false),
      new Tarefa("Ajudar meus pais", false)
    ];
  }

}
```

Perceba que o método adicionado recebe como parâmetro somente a descrição da tarefa. Na sequência ele cria um novo objeto do tipo "Tarefa", utilizando a descrição recebida como parâmetro e define automaticamente o status "false". A última linha do método insere o objeto criado no arrayDeTarefas.

# Adicionando o campo da descrição e o botão na View

Adicionalmente, vamos precisar inserir dois novos elementos na view de AppComponent:

- Um campo para a digitação da descrição da nova tarefa
- Um botão para confirmar a adição da tarefa

Para isso, insira as linhas abaixo destacadas em verde, no conteúdo do arquivo **app.component.html**:

```
<div class="main">
    <h1>Mozilla Dev - My To Do List</h1>
    <input class="lg-text-input" #campoNovoItem />
    <button class="btn-primary" (click)="CREATE_tarefa(campoNovoItem.value)">
        Adicionar Tarefa</button>
    <ul>
        <li *ngFor="let tarefaDoLoop of arrayDeTarefas">
            {{tarefaDoLoop.descricao}}
        </li>
    </ul>
</div>
```

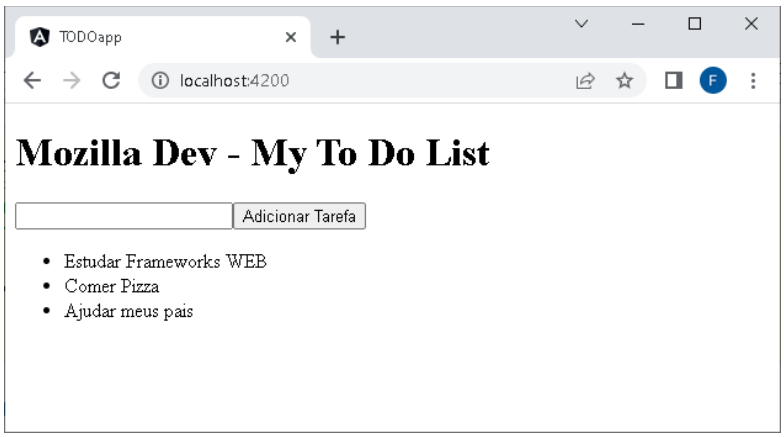
A primeira linha adicionada representa um campo de input. As próximas linhas adicionadas representam o botão. Perceba associado ao evento de "click" do botão, foi inserida a chamada ao método CREATE_tarefa definido anteriormente, passando o texto digitado no campo de input.

# Testando no Browser

Abra novamente o browser utilizando o mesmo endereço:

**http://localhost:4200/**

Resultado esperado:




Experimente inserir novas tarefas na lista a partir do browser.

# Adicionando Estilos ao app

O Framework Angular permite originalmente a definição de dois escopos de estilo:

- Estilo da aplicação
- Estilo do componente

## Estilo da aplicação

O estilos globais do App são definidos a partir do arquivo "**src/styles.css**"

Experimente definir os estilos de escopo geral do App, alterando o conteúdo do arquivo "**src/styles.css**" com as seguintes definições:

```
body {
    font-family: Helvetica, Arial, sans-serif;
}

.btn-wrapper {
    /* flexbox */
    display: flex;
    flex-wrap: nowrap;
    justify-content: space-between;
}

.btn {
    color: #000;
    background-color: #fff;
    border: 2px solid #cecece;
    padding: .35rem 1rem .25rem 1rem;
    font-size: 1rem;
}

.btn:hover {
    background-color: #ecf2fd;
}

.btn:active {
    background-color: #d1e0fe;
}
```

```
.btn:focus {
    outline: none;
    border: black solid 2px;
}

.btn-primary {
    color: #fff;
    background-color: #000;
    width: 100%;
    padding: .75rem;
    font-size: 1.3rem;
    border: black solid 2px;
    margin: 1rem 0;
}

.btn-primary:hover {
    background-color: #444242;
}

.btn-primary:focus {
    color: #000;
    outline: none;
    border: #000 solid 2px;
    background-color: #d7ecff;
}

.btn-primary:active {
    background-color: #212020;
}
```

# Estilo do Componente

É possível também definir estilos específicos para cada componente através do arquivo **XXX.component.css**, onde XXX é o nome do componente específico. Até este momento, nosso projeto possui somente um componente Angular, chamado por default de "app". Dessa forma, as definições de estilo desse componente devem ser declaradas dentro do arquivo "**app.component.css**".

Experimente definir os estilos do AppComponent, alterando o conteúdo do arquivo **app.component.css** com as seguintes definições:

```
body {
    color: #4d4d4d;
    background-color: #f5f5f5;
    color: #4d4d4d;
}

.main {
    max-width: 500px;
    width: 85%;
    margin: 2rem auto;
    padding: 1rem;
    text-align: center;
    box-shadow: 0 2px 4px 0 rgba(0, 0, 0, .2), 0 2.5rem 5rem 0 rgba(0, 0, 0, .1);
}
```

```
@media screen and (min-width: 600px) {
    .main {
        width: 70%;
    }
}

label {
    font-size: 1.5rem;
    font-weight: bold;
    display: block;
    padding-bottom: 1rem;
}

.lg-text-input {
    width: 100%;
    padding: 1rem;
    border: 2px solid #000;
    display: block;
    box-sizing: border-box;
    font-size: 1rem;
}

.btn-wrapper {
    margin-bottom: 2rem;
}

.btn-menu {
    flex-basis: 32%;
}

.active {
    color: green;
}

ul {
    padding-inline-start: 0;
}

ul li {
    list-style: none;
}
```

É importante saber que as definições de estilo acima foram obtidas a partir do tutorial de inspiração deste conteúdo, originalmente publicado em:

https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks/Angular_styling

Dessa forma, é possível que nem todas as definições de estilo presente, estejam efetivamente sendo aproveitadas pelos elementos HTML apresentados pelo nosso App.

# Testando no Browser

Abra novamente o browser utilizando o mesmo endereço:

**http://localhost:4200/**

Resultado esperado:

# Histórico de revisões

Revisão: 00
Data: 13/07/2022
Descrição das alterações:
Documento original