Frameworks WEB

Angular

Sistemas de Informação - UNISANTA

Índice

Client-side Frameworks

Introdução

Link do site de referência para esta introdução:

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

-> Páginas HTML Estáticas

-> JavaScript

-> Pacotes reutilizáveis (Libraries)

-> Frameworks

Exemplos de Frameworks client-side

- Angular (Google)
- React (Facebook)
- Vue (Evan You)
- Ember (Yehuda Katz)

Preparação do Ambiente

Link do site de referência para esta parte do tutorial:

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

Para desenvolver uma aplicação Angular vamos utilizar o Angular CLI que pode ser instalado a partir NPM do Node.js.

Instalação do Node.js

https://nodejs.org/en/

Instalação do git (Recomendado)

https://git-scm.com/downloads

Instalação do Angular CLI

```
npm install -g @angular/cli
```

Criação de uma nova aplicação Angular

Executar o seguinte comando dentro de uma pasta definida para ser nosso "workspace", para criar um app com o nome "todo":

```
ng new todo --routing=false --style=css
```

Execução do servidor para teste da aplicação

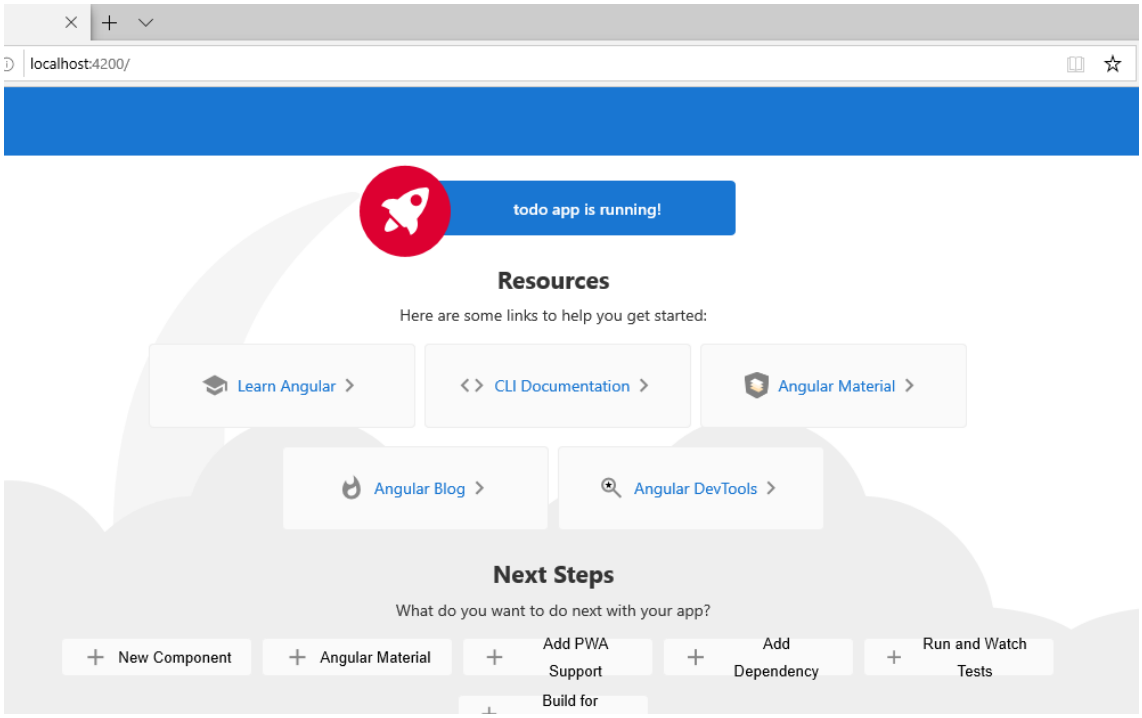
Executar o seguinte comando dentro da pasta do app "todo":

```
ng serve
```

Testar o app a partir do browser

http://localhost:4200/

O app deve aparecer rodando conforme a imagem abaixo:



A estrutura de um app Angular

Link do site de referência para esta parte do tutorial:

https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks/Angular_getting_started#get_familiar_with_your_angular_application

Um app Angular é formado por blocos de construção chamados de "Componentes". Um componente é constituído principalmente de:

- Lógica (arquivo com extensão **".ts"** chamado de class)
- View (arquivo com extensão **".html"** chamado de template)
- Estilos (arquivo com extensão **".css"** chamado de style)

Sempre que você cria um novo componente através do Angular CLI, esses arquivos são automaticamente criados para representá-lo.

Caso um componente chamado de "header" seja criado, então é criada uma pasta chamada "header" dentro da pasta app, com os seguintes arquivos:

- header.component.ts
- header.component.html
- header.component.css

Um arquivo adicional com o nome "header.component.spec.ts" também é criado para o propósito de realização de testes.

Principais arquivos criados em /app

Ao criar uma nova aplicação em Angular, os seguintes arquivos são criados automaticamente na pasta app.

app.module.ts

Especifica todos os componentes que o app usa.

app.component.ts

Contém a lógica do AppComponent. Considerando que ele é o componente principal do app, então este código define a lógica da página principal do app (class).

app.component.html

Conteúdo HTML do AppComponent que será inicialmente aberto no browser (template).

app.component.css

Contém o estilo do AppComponent (style)

Começando nosso app exemplo

Link do site de referência para esta parte do tutorial:

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

Para servir de exemplo de desenvolvimento, vamos criar um app de "Lista de tarefas a fazer" (To-Do List).

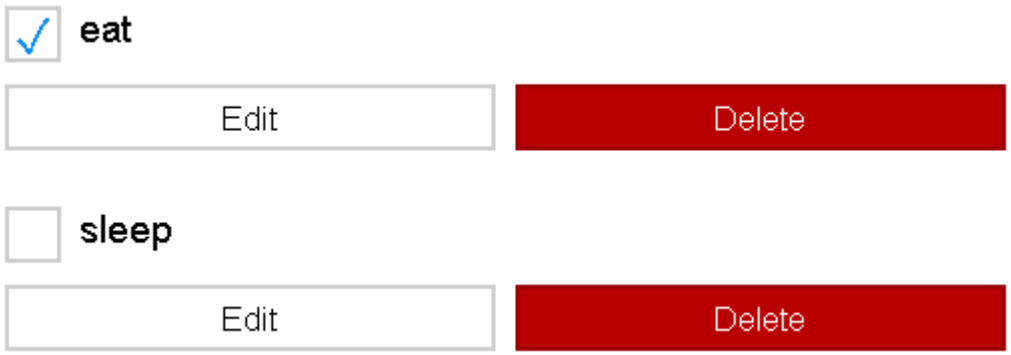
É possível experimentarmos a aplicação final rodando a partir do seguinte link:

http://brangs.com/todo/

Ao acessar o link acima, a aplicação é carregada e executada em seu browser.

Como você pode ver na aplicação rodando, cada tarefa a ser feita é chamada de "item".

É possível ver também, que cada "item" possui uma descrição e um campo de "status", informando se a tarefa já foi ou não realizada. Na imagem abaixo podemos ver duas tarefas chamadas "eat" e "sleep", onde somente a tarefa "eat" está marcada como realizada.



Definindo a lógica do AppComponent

Alterar o arquivo **app.component.ts** com o conteúdo abaixo para definir valores iniciais de itens durante a inicialização do app:

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})

export class AppComponent {
  title = 'todo';

  filter: 'all' | 'active' | 'done' = 'all';

  allItems = [
    { description: 'eat', done: true },
    { description: 'sleep', done: false },
    { description: 'play', done: false },
    { description: 'laugh', done: false },
  ];

  get items() {
    if (this.filter === 'all') {
      return this.allItems;
    }
    return this.allItems.filter(item => this.filter === 'done' ?
item.done : !item.done);
  }
}
```

Definindo a View do AppComponent

Alterar o arquivo **app.component.html** com o conteúdo abaixo para definir o que será mostrado pelo app.

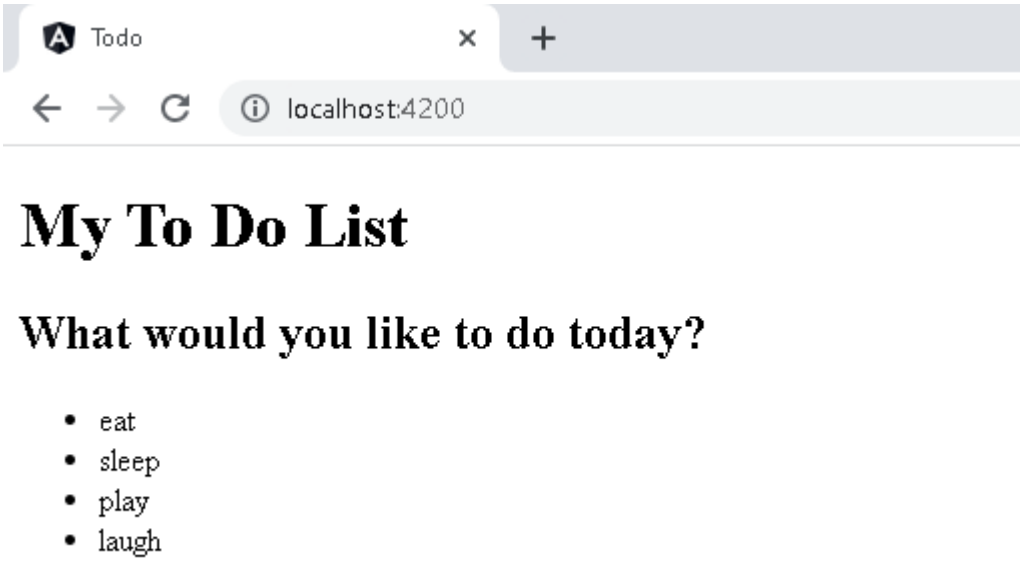
```
<div class="main">
  <h1>My To Do List</h1>
  <h2>What would you like to do today?</h2>

  <ul>
    <li *ngFor="let item of items">{{item.description}}</li>
  </ul>
</div>
```

Visualizar o resultado no browser

Estando com o servidor de teste em execução, visualize o funcionamento do app acessando a seguinte URL:

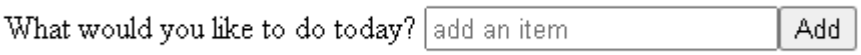
http://localhost:4200/

Resultado esperado:



Permitir a adição de itens pelo usuário

Para permitir que o usuário consiga inserir itens na lista de tarefas, é necessário inserir um campo para a digitação da descrição e um botão "Add".



Para isso, vamos definir o método que acrescenta itens ao array e posteriormente adicionar o campo da descrição e botão no template (view) do app.

Criar método de adição de itens na lista

Adicionar o seguinte método no arquivo **app.component.ts**, imediatamente abaixo do método getItens:

```
addItem(description: string) {
  this.allItems.unshift({
    description,
    done: false
  });
}
```

Adicionar o campo da descrição e botão, na view

Substituir a seguinte linha do template (arquivo **app.component.html**):

```
<h2>What would you like to do today?</h2>
```

por:

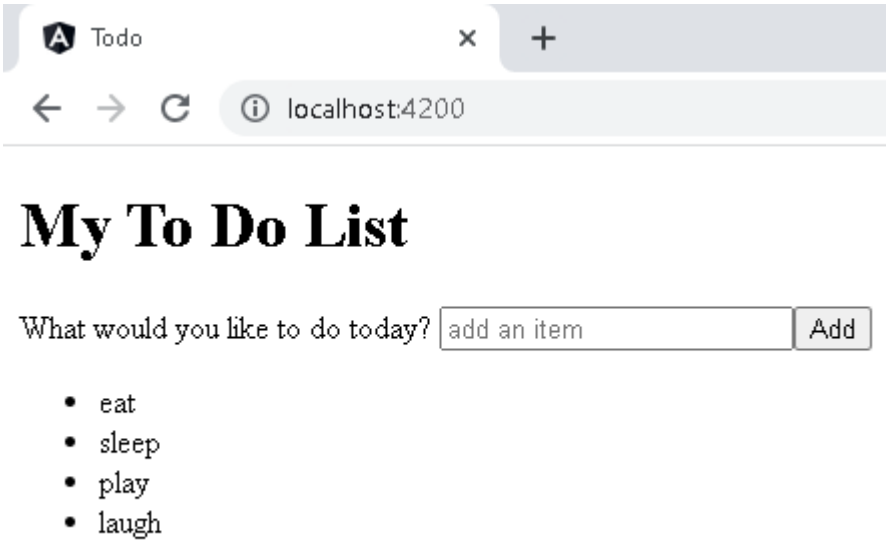
```
<label for="addItemInput">What would you like to do today?</label>

<input
  #newItem
  placeholder="add an item"
  (keyup.enter)="addItem(newItem.value); newItem.value = ''"
  class="lg-text-input"
  id="addItemInput"
/>

<button class="btn-primary"
(click)="addItem(newItem.value)">Add</button>
```

Visualizar o resultado no browser

Estando com o servidor de teste em execução, visualize o funcionamento do app acessando a seguinte URL:

http://localhost:4200/

Resultado esperado:



Experimente adicionar itens para ver se eles aparecem na lista.

Adicionando Estilos ao app

Link do site de referência para esta parte do tutorial:

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

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

- Estilo da aplicação
- Estilo do componente

Estilo da aplicação

O estilos 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 o seguinte conteúdo:

```
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;
```

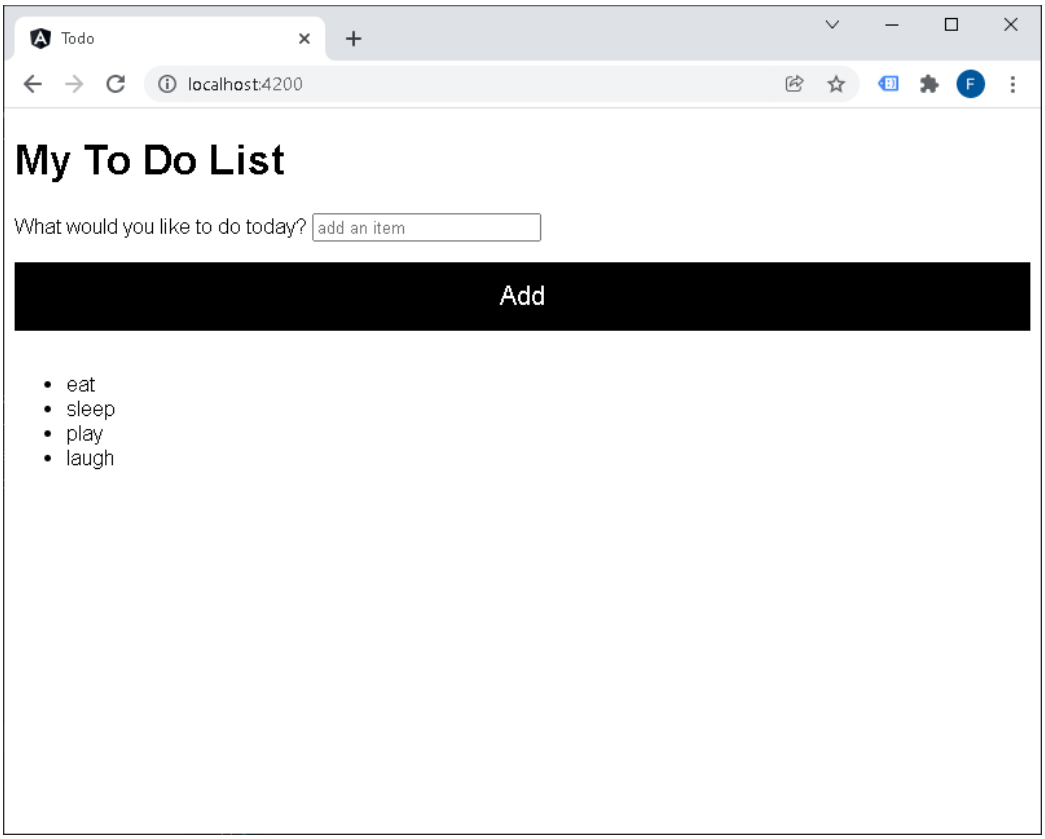
```
  font-size: 1rem;
}

.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;
}
```

Visualizar o resultado no browser

Estando com o servidor de teste em execução, visualize o funcionamento do app acessando a seguinte URL:

http://localhost:4200/

Resultado esperado:



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.

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

```
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 {
```

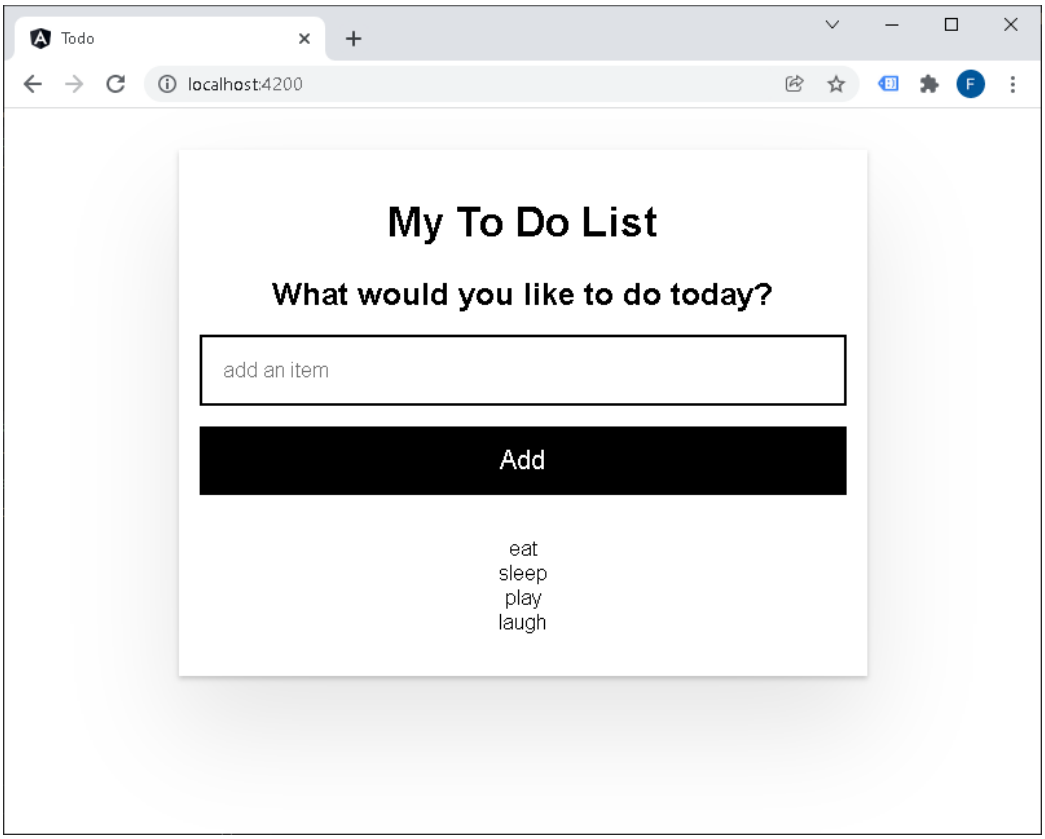
```
  flex-basis: 32%;
}

.active {
  color: green;
}

ul {
  padding-inline-start: 0;
}

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

Visualizar o resultado no browser

Estando com o servidor de teste em execução, visualize o funcionamento do app acessando a seguinte URL:

http://localhost:4200/

Resultado esperado:



Criando um novo componente

Link do site de referência para esta parte do tutorial:

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

Vamos criar um componente para permitir adicionar as seguintes funcionalidades aos itens da lista de tarefas:

- Marcar ou desmarcar a tarefa com o status: "Realizada" (Done)
- Editar a descrição da tarefa
- Deletar a tarefa

Para criar o novo componente, execute o seguinte comando dentro da pasta raiz do app:

```
ng generate component item
```

O comando acima apresentará o seguinte resultado:

```
CREATE src/app/item/item.component.html (19 bytes)
CREATE src/app/item/item.component.spec.ts (612 bytes)
CREATE src/app/item/item.component.ts (267 bytes)
CREATE src/app/item/item.component.css (0 bytes)
UPDATE src/app/app.module.ts (388 bytes)
```

Definindo Item

Antes de adicionar código ao ItemComponent, vamos definir a interface de um Item criando o arquivo /app/item.ts com o seguinte conteúdo:

```
export interface Item {
  description: string;
  done: boolean;
}
```

A definição acima mostra que um "Item" possui um campo string chamado "**description**" e um campo boolean chamado "**done**".

Definindo a lógica de ItemComponent

Alterar a linha de import de **item.component.ts** pela nova definição abaixo:

```
import { Component, Input, Output, EventEmitter } from '@angular/core';
```

Imediatamente abaixo dessa linha, realizar o import do Item definido acima.

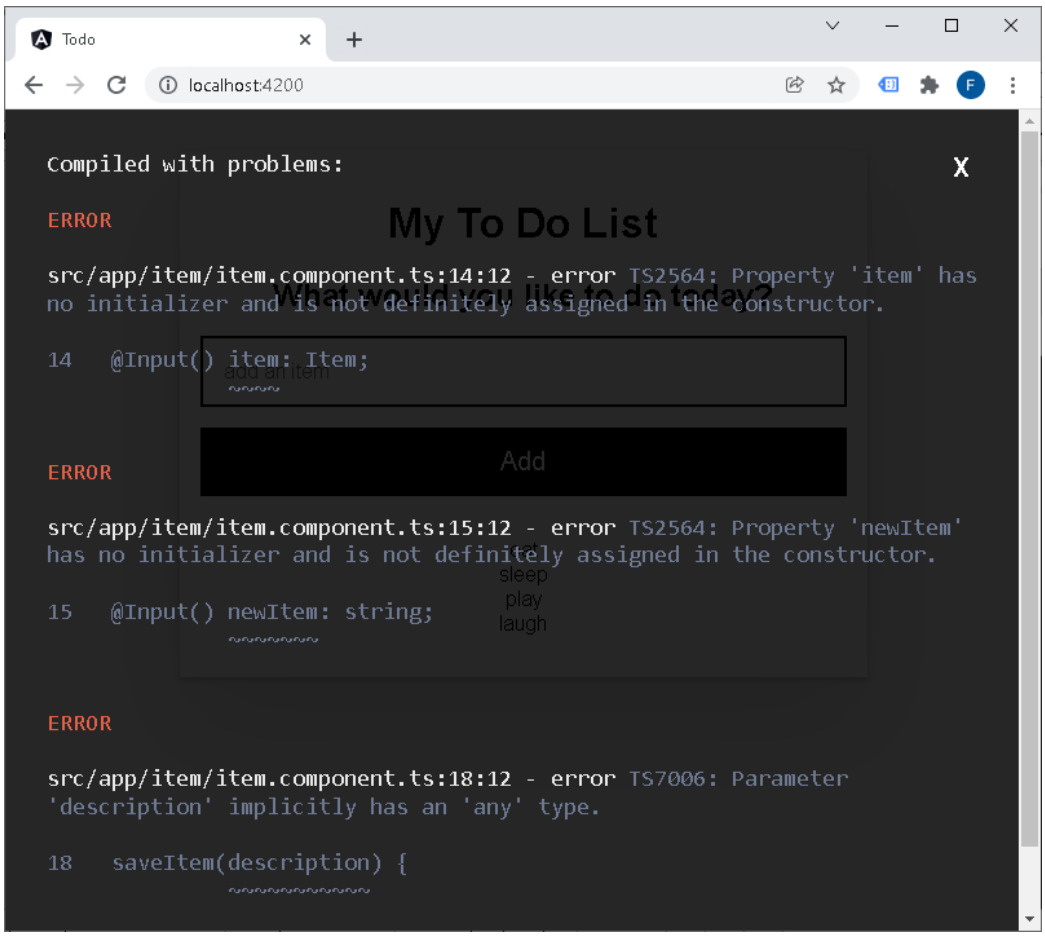
```
import { Item } from "../item";
```

Substituir o código da classe de ItemComponent, pelo conteúdo abaixo:

```
export class ItemComponent {

  editable = false;

  @Input() item: Item;
  @Input() newItem: string;
  @Output() remove = new EventEmitter<Item>();

  saveItem(description) {
    if (!description) return;
    this.editable = false;
    this.item.description = description;
  }
}
```

Contornando o erro de compilação

Ao implementar as alterações acima, poderá aparecer o seguinte erro de compilação:



Para eliminar o erro, basta alterar o valor do atributo "strict" em **/src/tsconfig.json** para "**false**" e salvar novamente o arquivo **item.component.ts** para forçar a recompilação do código sem erros.

Usando o ItemComponent a partir de AppComponent

Agora vamos referenciar o item dentro do template de AppComponent. Antes porém, vamos acrescentar a função em AppComponent que permitirá a remoção de tarefas pelo usuário.

Acrescente a seguinte função dentro de **app.component.ts**:

```
remove(item) {
  this.allItems.splice(this.allItems.indexOf(item), 1);
}
```

Agora, vamos referenciar o item, substituindo o elemento <ul> de app.component.html pelo conteúdo abaixo:

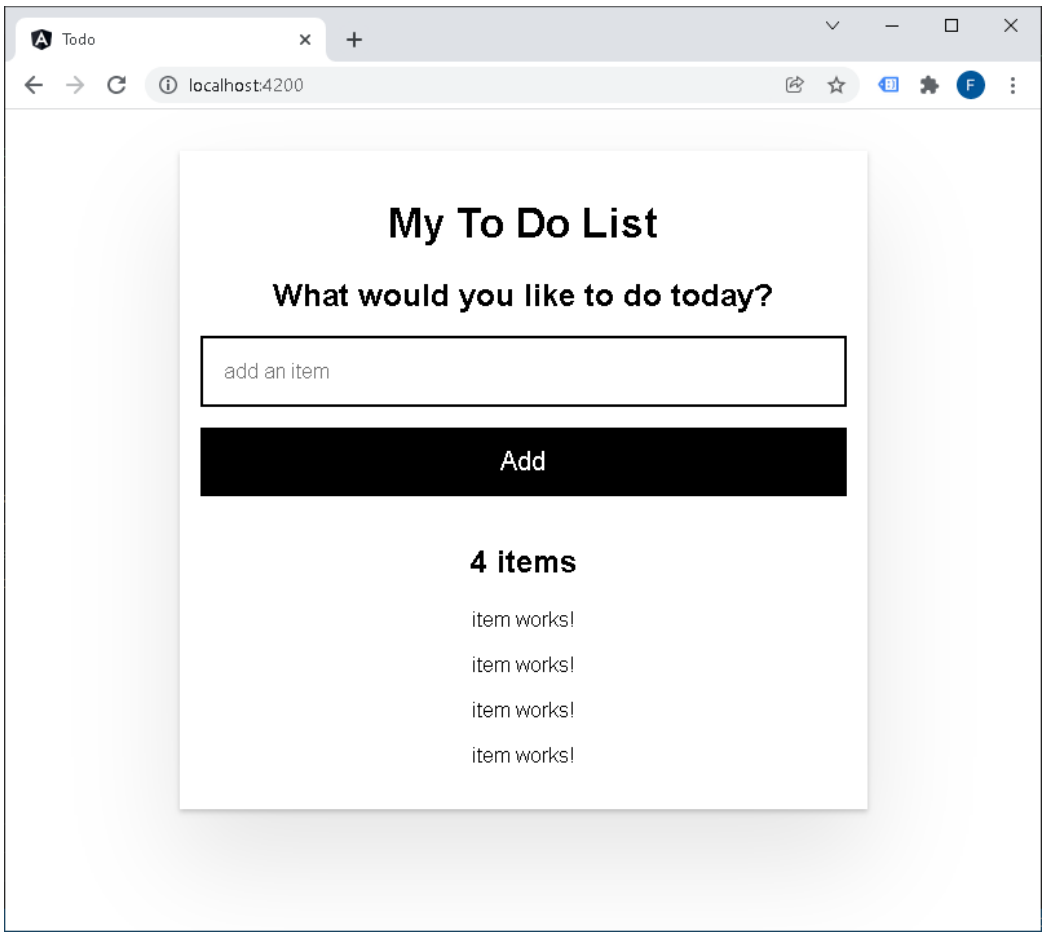
```
<h2>{{items.length}} <span *ngIf="items.length === 1; else
elseBlock">item</span>
<ng-template #elseBlock>items</ng-template></h2>

<ul>
  <li *ngFor="let item of items">
    <app-item (remove)="remove(item)" [item]="item"></app-item>
  </li>
</ul>
```

Visualizar o resultado no browser

Estando com o servidor de teste em execução, visualize o funcionamento do app acessando a seguinte URL:

http://localhost:4200/

Resultado esperado:



Definindo o template de ItemComponent

Definir a visualização de ItemComponent, substituindo o código HTML de **item.component.html** pelo conteúdo abaixo:

```
<div class="item">

  <input [id]="item.description" type="checkbox" (change)="item.done = !
item.done" [checked]="item.done" />
  <label [for]="item.description">{{item.description}}</label>

  <div class="btn-wrapper" *ngIf="!editable">
    <button class="btn" (click)="editable = !editable">Edit</button>
    <button class="btn btn-warn" (click)="remove.emit()">Delete</button>
  </div>

  <!-- This section shows only if user clicks Edit button -->
  <div *ngIf="editable">
    <input class="sm-text-input" placeholder="edit item"
[value]="item.description" #editedItem
(keyup.enter)="saveItem(editedItem.value)">

    <div class="btn-wrapper">
      <button class="btn" (click)="editable = !editable">Cancel</button>
      <button class="btn btn-save"
(click)="saveItem(editedItem.value)">Save</button>
    </div>
  </div>

</div>
```

O código acima, define os seguintes elementos para a DOM do item:

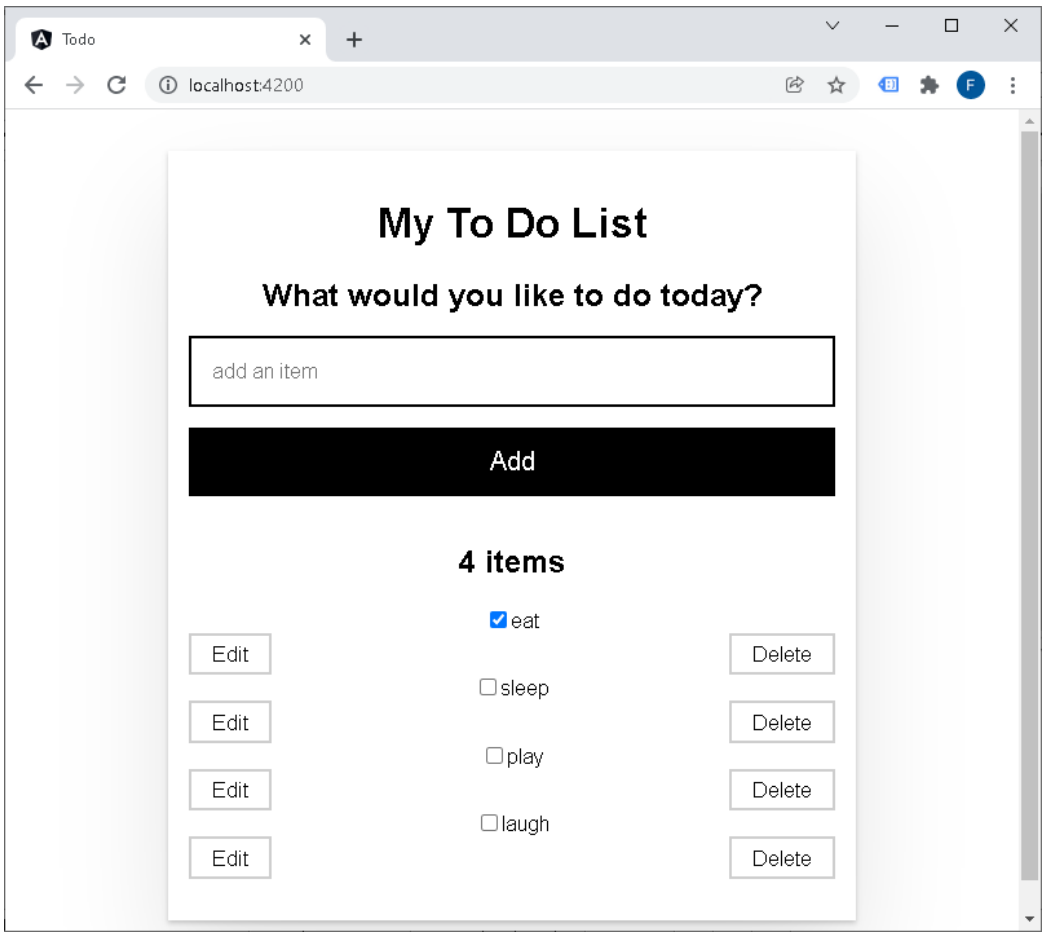
- **CHECKBOX** para informar se a tarefa foi feita ou não
- **LABEL** com a descrição da tarefa
- Quando o item não estiver sendo editado:
  - **BUTTON** EDIT
  - **BUTTON** Delete
- Quando o item estiver sendo editado:
  - **TEXTFIELD** para modificar a descrição
  - **BUTTON** Cancel
  - **BUTTON** Save

Visualizar o resultado no browser

Estando com o servidor de teste em execução, visualize o funcionamento do app acessando a seguinte URL:

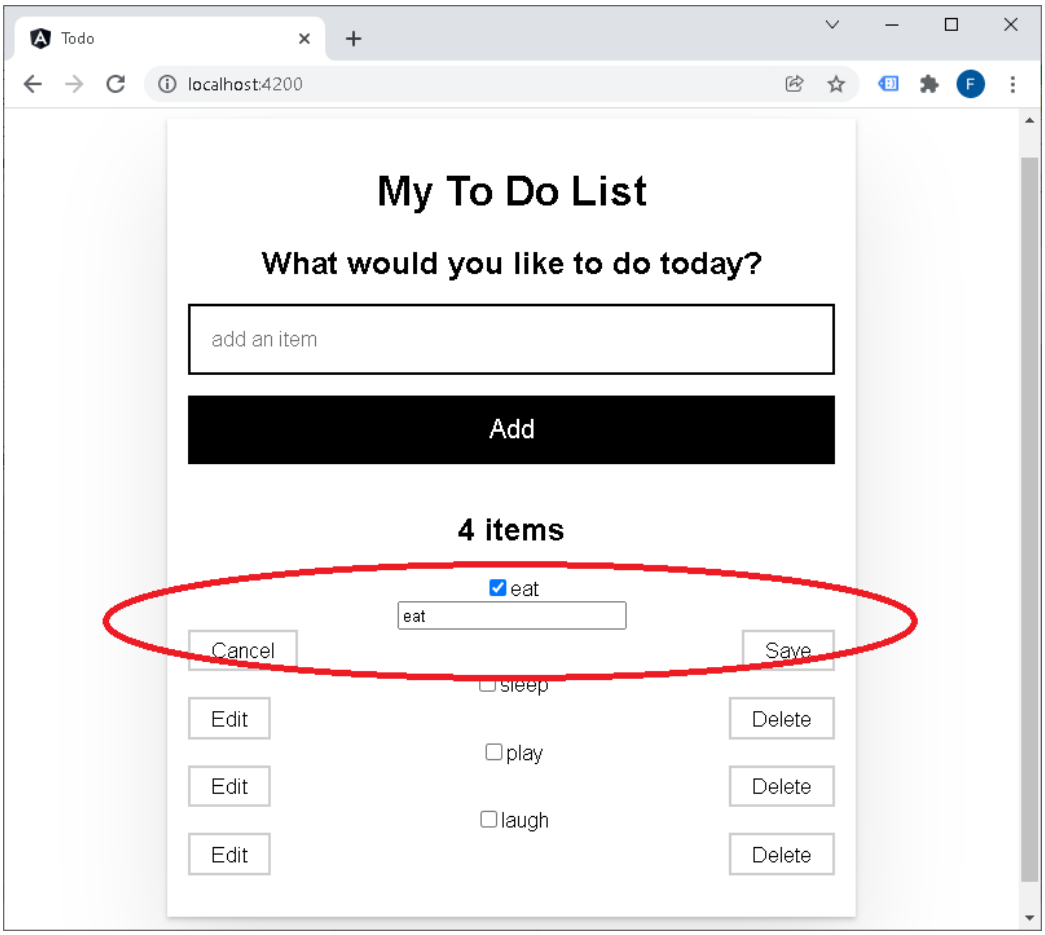
http://localhost:4200/

Resultado esperado:



Efeito ao pressionar EDIT

Ao pressionar o botão "Edit", é possível visualizar o efeito das diretivas *ngIf:



Adicionando estilos ao ItemComponent

Definir o seguinte conteúdo para o arquivo **item.component.css**:

```
.item {
  padding: .5rem 0 .75rem 0;
  text-align: left;
  font-size: 1.2rem;
}

.btn-wrapper {
  margin-top: 1rem;
  margin-bottom: .5rem;
}

.btn {
  /* menu buttons flexbox styles */
  flex-basis: 49%;
}

.btn-save {
  background-color: #000;
  color: #fff;
  border-color: #000;

}

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

.btn-save:focus {
  background-color: #fff;
  color: #000;
}

.checkbox-wrapper {
  margin: .5rem 0;
}

.btn-warn {
  background-color: #b90000;
  color: #fff;
  border-color: #9a0000;
}

.btn-warn:hover {
  background-color: #9a0000;
}

.btn-warn:active {
```

```
  background-color: #e30000;
  border-color: #000;
}

.sm-text-input {
  width: 100%;
  padding: .5rem;
  border: 2px solid #555;
  display: block;
  box-sizing: border-box;
  font-size: 1rem;
  margin: 1rem 0;
}

/* Custom checkboxes
Adapted from https://css-tricks.com/the-checkbox-hack/#custom-designed-
radio-buttons-and-checkboxes */

/* Base for label styling */
[type="checkbox"]:not(:checked),
[type="checkbox"]:checked {
  position: absolute;
  left: -9999px;
}
[type="checkbox"]:not(:checked) + label,
[type="checkbox"]:checked + label {
  position: relative;
  padding-left: 1.95em;
  cursor: pointer;
}

/* checkbox aspect */
[type="checkbox"]:not(:checked) + label:before,
[type="checkbox"]:checked + label:before {
  content: '';
  position: absolute;
  left: 0; top: 0;
  width: 1.25em; height: 1.25em;
  border: 2px solid #ccc;
  background: #fff;
}

/* checked mark aspect */
[type="checkbox"]:not(:checked) + label:after,
[type="checkbox"]:checked + label:after {
  content: '\2713\0020';
  position: absolute;
  top: .15em; left: .22em;
  font-size: 1.3em;
  line-height: 0.8;
  color: #0d8dee;
  transition: all .2s;
```

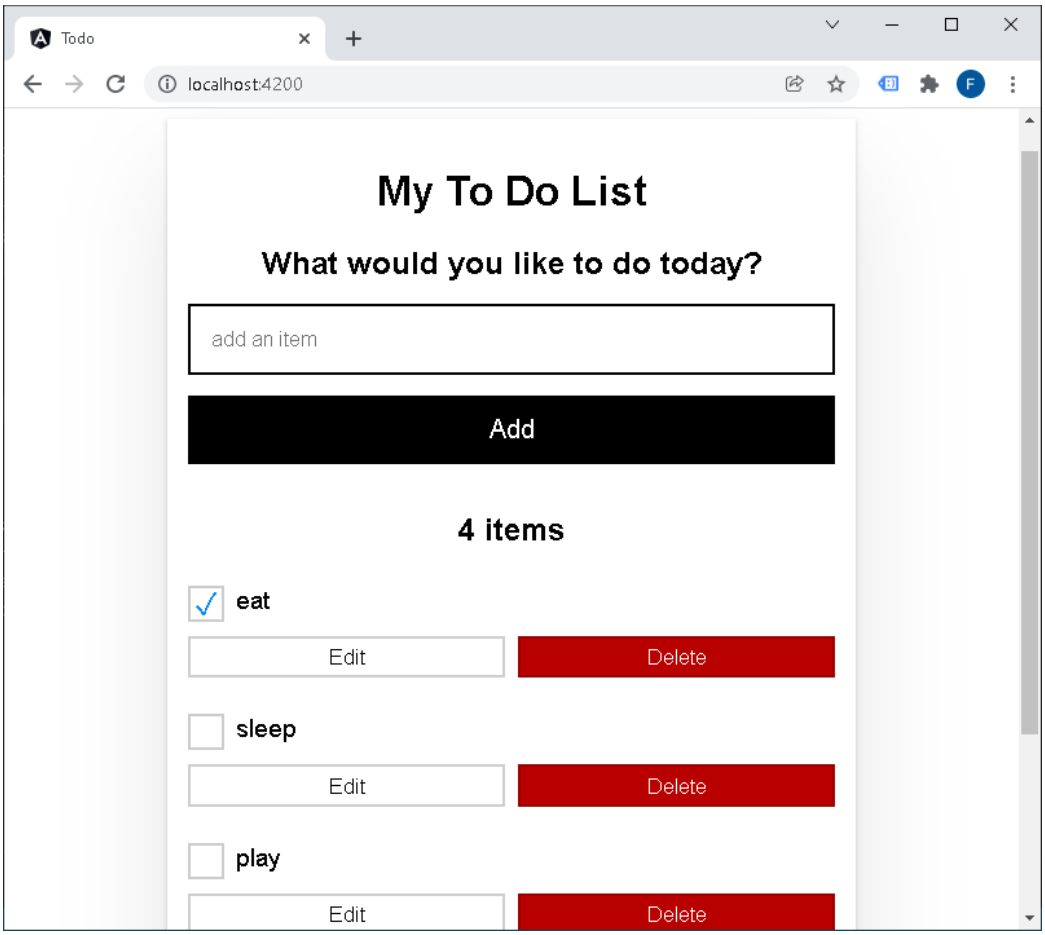
```
  font-family: 'Lucida Sans Unicode', 'Arial Unicode MS', Arial;
}
/* checked mark aspect changes */
[type="checkbox"]:not(:checked) + label:after {
  opacity: 0;
  transform: scale(0);
}
[type="checkbox"]:checked + label:after {
  opacity: 1;
  transform: scale(1);
}

/* accessibility */
[type="checkbox"]:checked:focus + label:before,
[type="checkbox"]:not(:checked):focus + label:before {
  border: 2px dotted blue;
}
```

Visualizar o resultado no browser

Estando com o servidor de teste em execução, visualize o funcionamento do app acessando a seguinte URL:

http://localhost:4200/

Resultado esperado:



Criando filtros de visualização

Link do site de referência para esta parte do tutorial:

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

Considerando que o código previamente criado em **app.component.ts** já prevê a utilização de filtros para visualização dos itens, basta adicionarmos no template, o código html que permite apresentar botões para a atribuição de diferentes valores para a variável **filter**. Insira o seguinte código HTML após a apresentação do botão "Add" e antes da lista de itens em **app.component.html**:

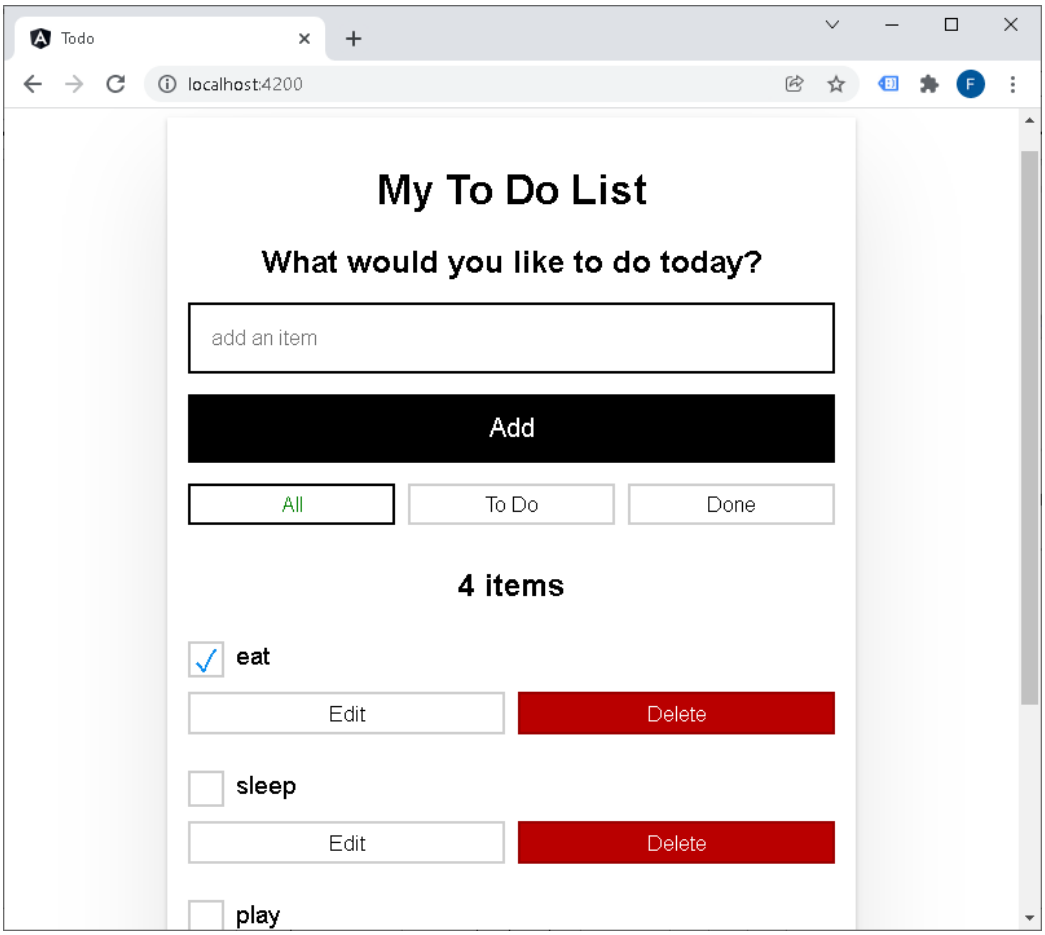
```
<!-- Buttons that show all, still to do, or done items on click -->
<div class="btn-wrapper">
  <button
    class="btn btn-menu"
    [class.active]="filter == 'all'"
    (click)="filter = 'all'">
    All
  </button>

  <button
    class="btn btn-menu"
    [class.active]="filter == 'active'"
    (click)="filter = 'active'">
    To Do
  </button>

  <button
    class="btn btn-menu"
    [class.active]="filter == 'done'"
    (click)="filter = 'done'">
    Done
  </button>
</div>
```

Visualizar o resultado no browser

Estando com o servidor de teste em execução, visualize o funcionamento do app acessando a seguinte URL:

http://localhost:4200/

Resultado esperado:



Realizando o Deploy do app

Link do site de referência para esta parte do tutorial:

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

Agora que nosso app está pronto e testado localmente, vamos preparar sua implantação em um servidor real.

Realizando o Building do app

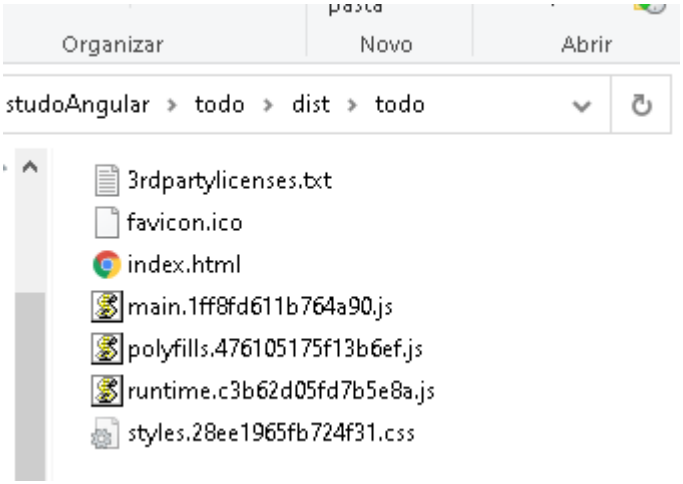
A primeira ação é executar o comando de build dentro da pasta do projeto (/todo):

```
ng  build
```

O comando acima apresenta o seguinte resultado no console:

```
✔ Browser application bundle generation complete.
✔ Copying assets complete.
✔ Index html generation complete.

Initial Chunk Files           | Names         |  Raw Size | Estimated Transfer Size
main.1ff8fd611b764a90.js      | main          | 121.17 kB |                35.13 kB
polyfills.476105175f13b6ef.js | polyfills     |  36.21 kB |                11.48 kB
runtime.c3b62d05fd7b5e8a.js   | runtime       |   1.03 kB |               598 bytes
styles.28ee1965fb724f31.css   | styles        | 648 bytes |               243 bytes

                              | Initial Total | 159.04 kB |                47.43 kB

Build at: 2022-02-07T13:22:38.726Z - Hash: e66f8abe06d38397 - Time: 3629ms
```

Após a execução, uma pasta chamada /dist/todo é criada dentro da pasta do projeto contendo os seguintes arquivos:



Publicando seu projeto em um servidor Web

Transferir os arquivos acima para um servidor WEB configurado em uma VM (igual ao que fizemos na Aula 08 de Sistemas Operacionais). Segue link da gravação dessa aula:

https://unisantabr.sharepoint.com/sites/SISTEMASOPERACIONAISII-1052N2A-30071/Shared%20Documents/Forms/AllItems.aspx?id=%2Fsites%2FSISTEMASOPERACIONAISII%2D1052N2A%2D30071%2FShared%20Documents%2FMaterial%20Did%C3%A1tico%2FRecordings%2FMaterial%20Did%C3%A1tico%2D20211005%5F210012%2DGrava%C3%A7%C3%A3o%20de%20Reuni%C3%A3o%2Emp4&parent=%2Fsites%2FSISTEMASOPERACIONAISII%2D1052N2A%2D30071%2FShared%20Documents%2FMaterial%20Did%C3%A1tico%2FRecordings

Histórico de revisões

Revisão: 00
Data: 04/02/2022
Descrição das alterações:
Documento original