**Frameworks front end**

**Temos varios Frameworkse Bibliotecas dicsponiveis para adoção em projetos de aplicações web. Considerandno os que se baseiam em JS, podemos citar:**

* Vue.js
* Angular
* React
* **Vue.js:** 
  + É um framework progressivo para a construção de interfaces de usuário. Ao contrário 7 de outros frameworks monolíticos, ou seja, frameworks que não podem ser integrados com outros, foi projetado desde sua concepção para ser utilizado de modo incremental, isto é, apenas parte do framework pode ser adotada.
  + Basicamente, o Vue mantém uma função que “observa” um objeto JavaScript e reflete qualquer mudança do seu estado no modelo de documento de objeto (Document Object Model – DOM) da página em linguagem de marcação de hipermídia (HTML).
  + [**https://br.vuejs.org/v2/guide/index.html**](https://br.vuejs.org/v2/guide/index.html)
* Componente de arquivo único contém três trechos de código:
  + Template (HTML); folha de estilo em cascata (CSS); JavaScript.

Primeiros passos:

Você pode [**criar um arquivo index.html**](https://github.com/vuejs/vuejs.org/blob/master/src/v2/examples/vue-20-hello-world/index.html) e incluir Vue com:

<!-- versão de desenvolvimento, inclui avisos úteis no console  -->

<script src="https://cdn.jsdelivr.net/npm/vue@2/dist/vue.js"></script>

ou:

<!-- versão de produção, otimizada para tamanho e velocidade -->

<script src="https://cdn.jsdelivr.net/npm/vue@2"></script>

No núcleo do Vue.js está um sistema que nos permite declarativamente renderizar dados no DOM (Document Object Model) usando uma sintaxe de template simples:

Ex: FrameworksFrontEnd/MeuExercicioVue/index.html

HTML:

<div class="container" id="app">

    <header>

       <h1>{{name}}</h1>

       <div class="avatar"><img src="imgs/avatar.jpg" alt=""></div>

   </header>

</div>

JS:

*var* app = new Vue({

        el: '#app', //elemento (id)

        data:  { //para substituir os dados

            name: 'LaryssaTGS',

        }

});

Acabamos de criar nosso primeiro aplicativo Vue! Isso parece muito similar a renderizar uma *template string*, mas Vue fez bastante trabalho interno. Os dados e o DOM estão agora interligados e tudo se tornou **reativo**. Como podemos ter certeza? Apenas abra o *console* JavaScript de seu navegador (agora mesmo, nesta página) e atribua um valor diferente em app.message. Você verá o exemplo renderizado acima se atualizando de acordo.

Perceba que não temos mais que interagir diretamente com o HTML. Um app Vue acopla-se a um único elemento da DOM (#app no nosso caso) e então o controla completamente. O HTML é o nosso ponto de entrada, mas todo o resto acontece dentro da recém-criada instância do Vue.

Além de simples interpolação de texto, podemos interligar atributos de elementos:

HTML:

<div id="app-2">

  <span v-bind:title="message">

    Pare o mouse sobre mim e veja a dica interligada dinamicamente!

  </span>

</div>

JS:

*var* app2 = new Vue({

  el: '#app-2',

  data: {

    message: 'Você carregou esta página em ' + new *Date*().toLocaleString()

  }

})

Aqui nos deparamos com algo novo. O atributo v-bind que você está vendo é chamado de **diretiva**. Diretivas são prefixadas com v- para indicar que são **atributos** especiais providos pelo Vue, e como você deve ter percebido, aplicam comportamento especial de reatividade ao DOM renderizado. Neste caso, basicamente está sendo dito: “mantenha o atributo title do elemento sempre atualizado em relação à propriedade message da instância Vue”.

Se você abrir seu *console* JavaScript novamente e informar app2.message = 'alguma nova mensagem', novamente poderá ver que o HTML vinculado - neste caso, o atributo title - foi atualizado imediatamente.

## [Condicionais e Laços](https://br.vuejs.org/v2/guide/index.html#Condicionais-e-Lacos)

HTML:

<div id="app-3">

  <p v-if="seen">Agora você me viu</p>

</div>

JS:

*var* app3 = new Vue({

  el: '#app-3',

  data: {

    seen: true

  }

})

Vá em frente e informe app3.seen = false no console. Você verá a mensagem desaparecer.

Este exemplo demonstra que nós podemos interligar dados não apenas ao texto e aos atributos, mas também à **estrutura** do DOM. Mais do que isso, Vue também provê um poderoso sistema de transições que pode automaticamente aplicar [**efeitos de transição**](https://br.vuejs.org/v2/guide/transitions) quando elementos são inseridos/atualizados/removidos pelo Vue.

Existem mais algumas diretivas, cada uma com sua própria funcionalidade. Por exemplo, a diretiva v-for pode ser usada para exibir uma lista de itens usando dados de um Array:

HTML:

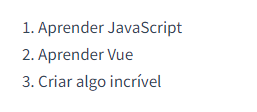
<div id="app-4">

  <ol>

    <li v-for="todo in todos">

      {{ todo.text }}

    </li>

  </ol>

</div>

JS:

*var* app4 = new Vue({

  el: '#app-4',

  data: {

    todos: [

      { text: 'Aprender JavaScript' },

      { text: 'Aprender Vue' },

      { text: 'Criar algo incrível' }

    ]

  }

})

No console, informe app4.todos.push({ text: 'Novo item' }). Você verá um novo item ser acrescentado dinamicamente à lista.

## [Tratando Interação do Usuário](https://br.vuejs.org/v2/guide/index.html#Tratando-Interacao-do-Usuario)

Para permitir aos usuários interagir com o aplicativo, podemos usar a diretiva v-on para anexar escutas a eventos (event listeners) que invocam métodos em nossas instâncias Vue:

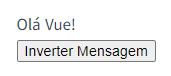
HTML:

<div id="app-5">

  <p>{{ message }}</p>

  <button v-on:click="reverseMessage">Inverter Mensagem</button>

</div>

JS:

*var* app5 = new Vue({

  el: '#app-5',

  data: {

    message: 'Olá Vue!'

  },

  methods: {

    reverseMessage: *function* () {

      this.message = this.message.split('').reverse().join('')

    }

  }

})

Observe que neste método atualizamos o estado da aplicação sem tocar no DOM - todas as manipulações são tratadas pelo Vue, o código que você escreve é focado na lógica de manipulação de dados.

Vue também provê a diretiva v-model, que torna a interligação de mão dupla (*two-way binding*) entre a caixa de texto e o estado da aplicação uma moleza:

HTML:

<div id="app-6">

  <p>{{ message }}</p>

  <input v-model="message">

</div>

JS:

*var* app6 = new Vue({

  el: '#app-6',

  data: {

    message: 'Olá Vue!'

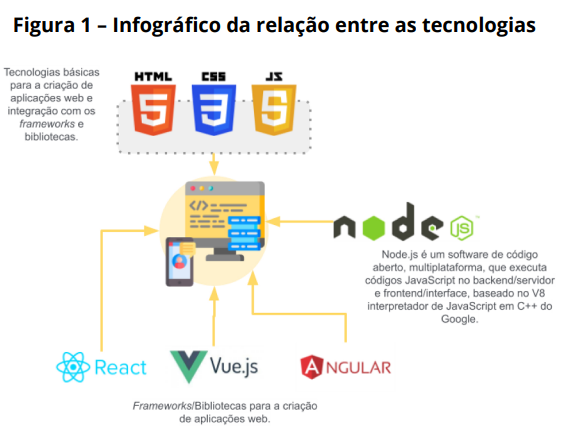
  }

})

* **Angular:**
  + É um framework de código aberto usado para construir aplicativos para web baseados em uma única página dinâmica. Os desenvolvedores também usam essa ferramenta para criar menus animados para páginas de internet baseadas em [**HTML**](https://www.hostinger.com.br/tutoriais/o-que-e-html-conceitos-basicos).
  + O AngularJS usa a arquitetura [**Model-View-Controller**](https://pt.wikipedia.org/wiki/MVC) (MCV), que é usado no desenvolvimento de apps para a internet. Esse tipo de arquitetura consiste em:
    - **Model**(Modelo) – a estrutura de dados que gerencia a informação e recebe comandos do controle.dir
    - **View**(Visão) – a representação da informação.
    - **Controller**(Controle) – responde aos comandos e interage com o modelo.

No contexto do AngularJS, o modelo é o framework, enquanto a visão é o HTML e o controle é o JavaScript. Assim, o AngularJS também pode ser chamado de framework Angular ou Angular Java.

* [**https://www.hostinger.com.br/tutoriais/o-que-e-angular**](https://www.hostinger.com.br/tutoriais/o-que-e-angular)
* **React**
  + O ReactJS é uma das bibliotecas [**de JavaScript**](https://www.hostinger.com.br/tutoriais/o-que-e-javascript) mais populares para desenvolvimento de aplicativos web ou para dispositivos móveis. Criado pelo Facebook, o React contém uma coleção de trechos de código de JavaScript reutilizáveis chamados componentes, usados para a construção da interface do usuário (UI).
  + É importante observar que o ReactJS não é um framework JavaScript. Isso porque ela é responsável apenas por renderizar os componentes da camada de visualização de uma aplicação. O React é uma alternativa a frameworks como [**Angular**](https://www.hostinger.com.br/tutoriais/o-que-e-angular) e Vue, que permitem criar funções complexas.
  + [**https://www.hostinger.com.br/tutoriais/o-que-e-react-javascript**](https://www.hostinger.com.br/tutoriais/o-que-e-react-javascript)

****

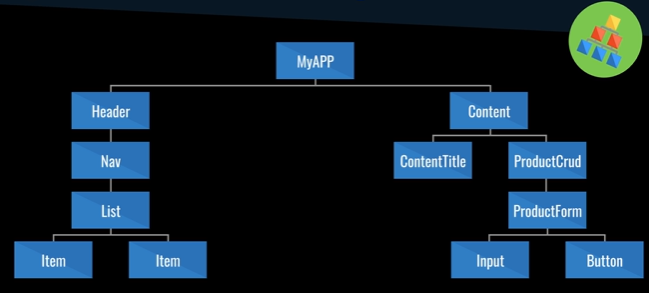
# Vou aprender Angular

A base para criar um projeto Angular:

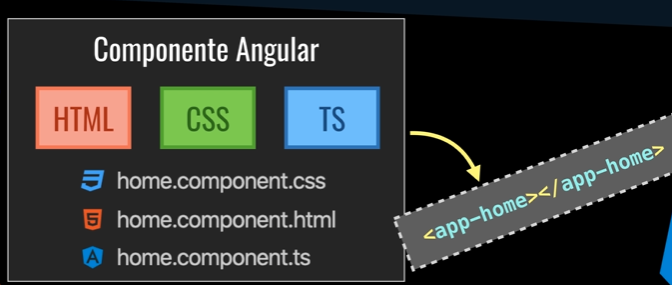
Npm i -g @angular/cli

Ng new minha-app

Arvore de componentes



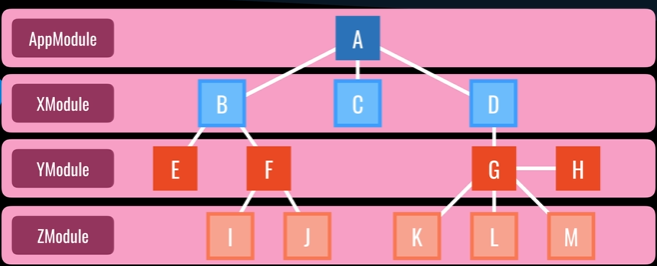
Componente Angular



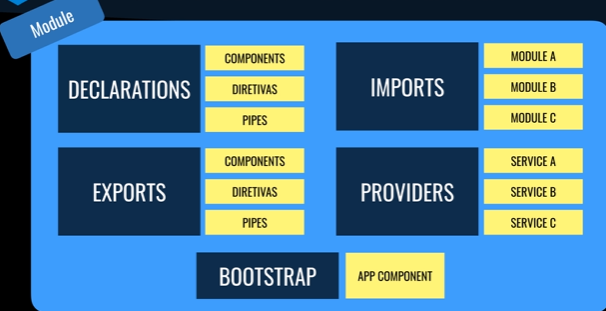
Ex:



Organizando em modulos:



Atributos:



Tem que ter esses 5 atributos no módulo

* Declarations: para declarar todos os componentes, diretivas e pipes que fazem parte do modulo.
* Exports: para declarar todos os componentes, diretivas e pipes, que vão ficar exportado, que vao ficar visíveis para fora do modulo.
* Imports: outros módulos, pode ser do meu próprio aplicação, como pode ser de um modulo externo.
* Providers: vou declarar os services
* Bootstrap: o que define o nosso componente que vai ser carregado no modulo

O appModule não precisa exportar nada.

# Criar um novo projeto do angular

[video iniciando com angular](https://www.youtube.com/watch?v=fOVsQVHey74)

* Abre o Prompt comando
* Vai na pasta desejada (‘cd nome’, ‘dir’ para ver as pastas)
* Para colocar na versão atual digita: npm install -g @angular/cli
* Para criar o novo projeto digita: ng new nomeDoProjeto
  + Ele vai perguntar, se eu quero uma classe de rotas, tu colocas ‘yes’
  + Seleciona o CSS que quer usar, ‘CSS’ é o padrão
  + Ele vai perguntar, se quero ativar a renderização do lado do servidor (SSR) e o site estático, tu colocas ‘yes’
  + Ele vai perguntar, se eu gostaria de usar as APIs Server Routing e App Engine (Developer Preview) para este aplicativo de servidor, coloca ‘yes’
* Integrar com Bootstrap
  + [Bootstrap](https://getbootstrap.com/docs/5.3/getting-started/introduction/)
  + Colocar os links de css e os scripts no html principal *index.html*
* Editar o *app.component.html*

<header>

    <nav class="navbar navbar-dark bg-primary">

        <a class="navbar-brand" [routerLink]="['/']">appAngular</a>

    </nav> <!--routerLink: para clicar nele e voltar para o inicio-->

</header>

<section>

  <main class="container">

      <router-outlet></router-outlet>

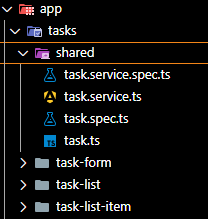
  </main>

</section>

<footer class="mt-2 p-2 text-center">

    <a href="http://www.fabricadecodigo.com">www.fabricadecodigo.com</a>

</footer>

* Fazer rodar o código:
  + No terminal sempre dentro do projeto, coloca ‘ng serve -o’
* Agora vamos criar/gerar a estrutura da aplicação
  + Entrar no <https://v17.angular.io/docs> > Reference > CLI command > ng generate
  + Vou criar uma pasta Tasks.. cd appAngular > cd src > cd app..
* Comando que foi usado para criar as páginas e componentes:
* ng g component tasks/task-list
* ng g component tasks/task-list-item
* ng g component tasks/task-form
* ng g service tasks/shared/task
* ng g class tasks/shared/task
* Agora é organizar as rotas de aplicação:
  + É onde vai estar as urls onde vou navegar e quais componentes que ele deverá abrir. No ‘app-routing.module.ts’ adiciona:

const routes: Routes = [

  {path: '', component: TaskListComponent},

  {path: 'new', component: TaskFormComponent},

  {path: 'edit/:id', component: TaskFormComponent}

];

Path: é a URL

Na url vazia, vai abrir o componente de lista

Depois a url para criar uma nova tarefa, quero que abra o componente formulário

Depois a url para editar uma tarefa, com um parâmetro, que vai abrir o formulário também

* No ‘app.module.ts’ vou importar todos os links

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

import { BrowserModule } from '@angular/platform-browser';

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

import { AppRoutingModule } from './app-routing.module';

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

import { TaskListComponent } from './tasks/task-list/task-list.component';

import { TaskListItemComponent } from './tasks/task-list-item/task-list-item.component';

import { TaskFormComponent } from './tasks/task-form/task-form.component';

@NgModule({

  declarations: [ AppComponent, TaskListComponent, TaskListItemComponent, TaskFormComponent],

  imports: [ BrowserModule, AppRoutingModule, FormsModule ],

  providers: [],

  bootstrap: [AppComponent]

})

export class AppModule { }

* Agora tenho que montar a estrutura de cada componente, entrando nos htmls de cada um:
  + No ‘task-list.component.html’

<h2>Tarefas</h2>

<hr />

<button type="button" class="btn btn-primary mb-2" [routerLink]="['/new']">

Nova tarefa

</button>

<!--class do Bootstrap

    colocando ele para ir na rota new, clicando no botao

-->

Bootstrap:

[Documentação e exemplos de utilitários de texto comuns para controlar alinhamento, ajuste de tamanho, peso e muito mais.](https://getbootstrap.com/docs/5.3/utilities/text/)

* Agora temos que montar as estrutura da aplicação, recuperar os dados, exibir na tela:
  + Tasks > shared > ‘tasks.ts’

export class Task {

  id: number;

  description: string;

  completed: boolean;

  constructor(id: number, description: string, completed: boolean) {

    this.id = id;

    this.description = description;

    this.completed = completed;

  }

}

//Tem que adicionar os valores padrão às propriedades na declaração

//Caso queira que os valores sejam passados ​​ao criar um objeto, use um construtor

* + Tasks > shared > ‘task.service.ts’
  + É onde vai ficar centralizada as regras de negócio da aplicação, as operações de crud

import { Task } from './task';

import { Injectable } from '@angular/core';

@Injectable({

  providedIn: 'root'

})

export class TaskService {

  tasks: Task[] = [

    {id: 1, description: 'Tarefa 1', completed: false},

    {id: 2, description: 'Tarefa 2', completed: false},

    {id: 3, description: 'Tarefa 3', completed: true},

    {id: 4, description: 'Tarefa 4', completed: false},

    {id: 5, description: 'Tarefa 5', completed: false},

    {id: 6, description: 'Tarefa 6', completed: false},

    {id: 7, description: 'Tarefa 7', completed: false},

    {id: 8, description: 'Tarefa 8', completed: false},

    {id: 9, description: 'Tarefa 9', completed: false},

    {id: 10, description: 'Tarefa 10', completed: false}

  ]

  constructor() { }

getAll() { //para pegar todas as tarefas

  return this.tasks;

}

}

* No **task-list e taks-list-item**:
  + No ‘task-list.component.ts’

import { TaskService } from './../shared/task.service';

import { Task } from './../shared/task';

import { Component, OnInit } from '@angular/core';

@Component({

  selector: 'app-task-list',

  templateUrl: './task-list.component.html',

  styleUrls: ['./task-list.component.css']

})

export class TaskListComponent implements OnInit {

  tasks: Task[] = [] //importar o array de tarefas

  constructor(private taskService: TaskService) {}

  //vou receber um construtor da classe taskService

  ngOnInit() { //tem que implementar o método ng ou init

    this.tasks = this.taskService.getAll();

  }//quando o componente iniciar, ele vai la no service que esta cadastrado

}

* + No ‘task-list-item.component.ts’

import { Component, OnInit, Input } from '@angular/core';

import { Task } from '../shared/task';

@Component({

  selector: 'app-task-list-item',

  templateUrl: './task-list-item.component.html',

  styleUrls: ['./task-list-item.component.css']

})

export class TaskListItemComponent implements OnInit {

  @Input() //atritubo do meu componente, onde vou passar por parametro, para esse componente qual é a task, e vou conseguir reenderizar na tela

  task!: Task;

  constructor() {}

  ngOnInit(){

  }

}

* + No ‘task-list-item.component.html’

<div class="form-group form-check">

  <input type="checkbox" class="form-check-input" />

</div>

<div [innerHTML]="task.description" class="task-item-description"

    [ngClass]="task.completed ? 'task-completed' : '' ">

<!-- Interpolation -->

</div>

<div>

  <button type="button" [routerLink]="['/edit', task.id]" class="btn btn-primary mr-1">Alterar</button>

</div>

<div>

  <button type="button" id="button-1"  class="btn btn-danger">Excluir</button>

</div>

CSS:

.task-completed{

  text-decoration: line-through;

}

.task-item-description{

  flex-grow: 1;

}

* + No ‘task-list.component.html’

<h1 class="mt-5 ms-5  p-2 mb-3" style="width: 200px;">Tarefa</h1>

<hr />

<!--class do Bootstrap

    colocando ele para ir na rota new, clicando no botao

-->

<button type="button" class="btn btn-primary bg-primary mb-2" [routerLink]="['/new']">

  Nova Tarefa

</button>

<section>

  <div \*ngIf="tasks.length == 0" class="alert alert-warning" role="alert"> <!--só um alerta-->

    Nenhuma tarefa foi encontrada ainda.

  </div>

  <ul class="list-group">

    <li class="list-group-item" \*ngFor="let task of tasks">

        <app-task-list-item [task]="task" class="task-item"></app-task-list-item>

    </li>

  </ul>

</section>

CSS:

.task-item{

  display: flex;

  flex-direction: row;

  align-items: center;

}

* No **tasks-form**:
  + Volta pa o Tasks > shared > ‘task.service.ts’
  + Para criar os métodos de salvar e recuperar apenas um item para fazer a edição

getAll() {

    return this.tasks;

  }

getByID(id: number) { //ele vai no array de tarefas

    const task = this.tasks.find((value) => value.id == id);

    return task; //procurar(metodo find) a primeria tarefa q ele achar que tem esse id, é

oq ele vai devolver para tela

  }

save(task: Task) {

    if (task.id) { //verifar se tem id, se tiver entao ele entende q é uma edição

      const taskArr = this.getByID(task.id); //recupera a tarefa pelo ID

      if (taskArr) { //verifica se a tarefa foi encontrada

        taskArr.description = task.description; //vai atualizar a descrição

        taskArr.completed = task.completed; //se ele ta completa ou nao

      }

    }

     else { //caso nao tenha id, ele entende q é uma conclusao

      const lastId = this.tasks[this.tasks.length-1].id; //ele vai gerar um id, indo na

ultima tarefa cadastrada

      task.id = lastId + 1; //gera um novo ID

      task.completed = false; //marca como não completada

      this.tasks.push(task); //adiciona a nova tarefa ao array

    }

  }

delete(id: number) {

    const taskIndex = this.tasks.findIndex((value) => value.id == id); //recuperando o index da tarefa

    this.tasks.splice(taskIndex, 1); //chamando o metodo splice, para remover

  }

* No ‘task-form.component.html’

<h2>{{ title }}</h2>

<hr />

<section id="form">

  <form (ngSubmit)="onSubmit()" #formTask="ngForm">

    <div class="form-group">

      <label>Descrição</label>

      <textarea [(ngModel)]="task.description" name="description" rows="5" class="form-control"></textarea>

    </div><!--vai mostrar o que ta na descrição-->

    <button type="submit" class="btn btn-primary mr-2">Salvar</button>

    <button type="button" class="bth btn-secondary" [routerLink]="['']">Cancelar</button>

  </form>

</section>

* No ‘task-form.component.ts

import { Task } from './../shared/task';

import { Component, OnInit } from '@angular/core';

import { ActivatedRoute, Router } from '@angular/router';

import { TaskService } from '../shared/task.service';

@Component({

  selector: 'app-task-form',

  templateUrl: './task-form.component.html',

  styleUrls: ['./task-form.component.css']

})

export class TaskFormComponent implements OnInit {

  task: Task = new Task(); //vou ter uma tarefa, o que vou usar na tela

  title: string = 'Nova tarefa';

  constructor(

    private activatedRoute: ActivatedRoute, //minha URL que ta ativa

    private router: Router, //responsavel por fazer a navegação

    private taskService: TaskService //onde fica as regras de negocio

  ){ }

  ngOnInit() { //ele tem que identificar se é uma edição ou uma inclusao

    const id = this.activatedRoute.snapshot.paramMap.get('id');

    if (id) { //se tem id

      this.task = this.taskService.getByID(parseInt(id)) ?? new Task();

//vai no servise recuperar a tarefa, passando pra int

      // ?? new Task(); se getByID retornar undefined, será atribuído um novo objeto

Task(), evitando o erro.

      this.title = 'Alterando tarefa'

    }

  }

  onSubmit(){

    this.taskService.save(this.task); //para salvar

    this.router.navigate(['']); //redirecionar para listagem

  }

}

* Agora vamos implementar click nos botões
* Excluir uma tarefa
  + Voltar para o ‘task-list-item.component’, pois é la que esta os botões
  + ‘task-list-item.component.html’

Adicionar no botao Excluir: (click)="remove(task)"

<div>

  <button type="button" id="button-1"  class="btn btn-danger" (click)="remove(task)">Excluir</button>

</div>

* + ‘task-list-item.component.ts’

Adicionar o constructor, importando o 'task.service.ts', e o método remove

constructor(private taskService: TaskService) {}

  //importar ele aqui

  ngOnInit(){

  }

  remove(task: Task) {

    this.taskService.delete(task.id); //ja tem o metodo de excluir, no 'task.service.ts'... importar no constructor em cima

  }

* Usar o método change em um checkbox - é um componente que permite ao usuário selecionar ou desmarcar uma opção
  + ‘task-list-item.component.html’

Adicionar no primeiro input [(ngModel)]="task.completed"

<div class="form-group form-check">

  <input type="checkbox" class="form-check-input" [(ngModel)]="task.completed" (change)="onCompletedCheckChange(task)"/>

</div>

Adicionar o método change -  um evento DOM que ocorre quando o usuário confirma alterações em campos de formulário (change)="onCompletedCheckChange(task)"

* + Colocando esse método no ‘task-list-item.component.ts’

onCompletedCheckChange(task: Task){

    this.taskService.save(task); //ja tem o metodo de salvar, no 'task.service.ts'

  }

* Salvar dados no localStorage do navegador. Para poder acessar o site, depois que atualiza o site, ou para acessar novamente... pois ele apaga quando der F5
  + Volta para o ‘task.service.td’

Adiciona dentro do método save

window.localStorage.setItem('lista.tarefas', JSON.stringify(this.tasks));

    //toda vez que salvar uma tarefa ele salva no localStorage

Como o array estava 0, temos que mudar uma parte do código do método save:

save(task: Task) {

    if (task.id) { //verifar se tem id, se tiver entao ele entende q é uma edição

      const taskArr = this.getByID(task.id); //recupera a tarefa pelo ID

      if (taskArr) { //verifica se a tarefa foi encontrada

        taskArr.description = task.description; //vai atualizar a descrição

        taskArr.completed = task.completed; //se ele ta completa ou nao

      }

    }

     else { //caso nao tenha id, ele entende q é uma conclusao

      let lastId = 0; Mudou aqui

      if (this.tasks.length > 0) { //se eu tenho algo nesse array

        lastId = this.tasks[this.tasks.length-1].id; //ele vai gerar um id, indo na ultima tarefa cadastrada

      }

      //se nao ele ..

      task.id = lastId + 1; //gera um novo ID

      task.completed = false; //marca como não completada

      this.tasks.push(task); //adiciona a nova tarefa ao array

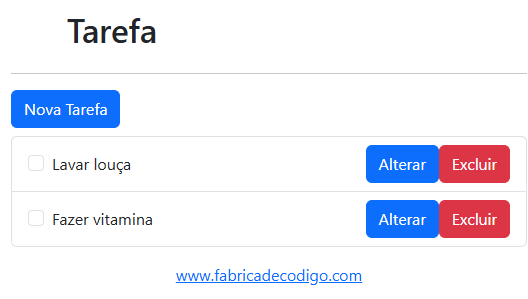
    }

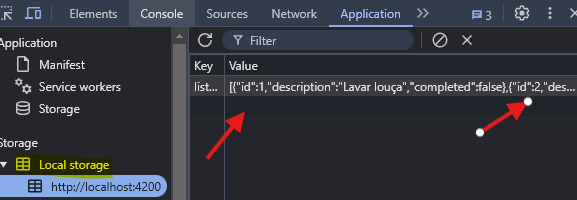
    window.localStorage.setItem('lista.tarefas', JSON.stringify(this.tasks));

    //toda vez que salvar uma tarefa ele salva no localStorage

  }

Para ver onde salvou, inspeciona o site. Adiciona uma nova tarefa  
Vai em application > Local storage





Para deixar ele salvo na tela, toda vez que atualizar a pagina

Vai no getAll:

getAll() {

    return this.tasks;

}

Adiciona:

getAll() {

    const list = window.localStorage.getItem('lista.tarefas');

    if (list) { //se list tem alguma coisa

      this.tasks = JSON.parse(list); //ele te devolve na tela

    }

    return this.tasks;

  }

Tenho que fazer na hora que deletar tbm, adicionando o:

window.localStorage.setItem('lista.tarefas', JSON.stringify(this.tasks));

No delete:

delete(id: number) {

    const taskIndex = this.tasks.findIndex((value) => value.id == id); //recuperando o index da tarefa

    this.tasks.splice(taskIndex, 1); //chamando o metodo splice, para remover

window.localStorage.setItem('lista.tarefas', JSON.stringify(this.tasks)); //sempre que eu tenho que deletar uma tarefa, ele vem aqui e salva no localStorage

  }

## Crud completo com API REST com NestJS e MongoDB - Angular

Notação: Bootstrap: [Controlar a margem responsiva, preenchimento e espaço para modificar a aparência de um elemento.](https://getbootstrap.com/docs/5.3/utilities/spacing/)

Utilitários de espaçamento que se aplicam a todos os breakpoints, de xsa xxl, não têm abreviação de breakpoint neles. Isso ocorre porque essas classes são aplicadas de min-width: 0e para cima e, portanto, não são limitadas por uma media query. Os breakpoints restantes, no entanto, incluem uma abreviação de breakpoint.

As classes são nomeadas usando o formato {property}{sides}-{size}para xse {property}{sides}-{breakpoint}-{size}para sm, md, lg, xl, e xxl.

Onde *a propriedade* é uma das seguintes:

* m- para classes que definemmargin
* p- para classes que definempadding

Onde *lados* é um dos seguintes:

* t- para classes que definem margin-topoupadding-top
* b- para classes que definem margin-bottomoupadding-bottom
* s- (início) para classes que definem margin-leftou padding-leftem LTR, margin-rightou padding-rightem RTL
* e- (fim) para classes que definem margin-rightou padding-rightem LTR, margin-leftou padding-leftem RTL
* x- para classes que definem ambos \*-lefte\*-right
* y- para classes que definem ambos \*-tope\*-bottom
* em branco - para classes que definem um marginou paddingem todos os 4 lados do elemento

Onde *o tamanho* é um dos seguintes:

* 0- para classes que eliminam o marginou paddingconfigurando-o para0
* 1- (por padrão) para classes que definem marginou paddingpara$spacer \* .25
* 2- (por padrão) para classes que definem marginou paddingpara$spacer \* .5
* 3- (por padrão) para classes que definem marginou paddingpara$spacer
* 4- (por padrão) para classes que definem marginou paddingpara$spacer \* 1.5
* 5- (por padrão) para classes que definem marginou paddingpara$spacer \* 3
* auto- para classes que definem o margincomo automático

(Você pode adicionar mais tamanhos adicionando entradas à $spacersvariável do mapa Sass.)