

Angular

O que é?

Angular é uma plataforma de desenvolvimento front-end, nessa plataforma esta um framework baseado em componentes, algumas bibliotecas e uma switch que auxilia no desenvolvimento, no build e nos testes.

Ferramentas:

Visual studio code, Node.js e angular.

Como instalar o Angular de forma geral:

Va para o cmd e utilize o comando:

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

No visual studio adicione a extensão angular language servisse.

Criando projeto angular:

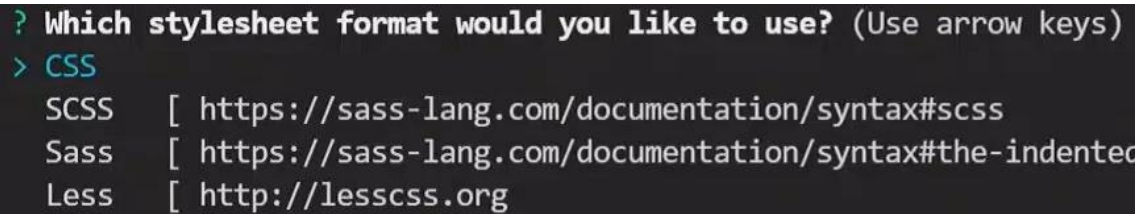
Após criarmos uma pasta acessamos o terminal no local da pasta e usamos o comando:

```
ng new nomedoprojeto
```



```
? Would you like to add Angular routing? (y/N)
```

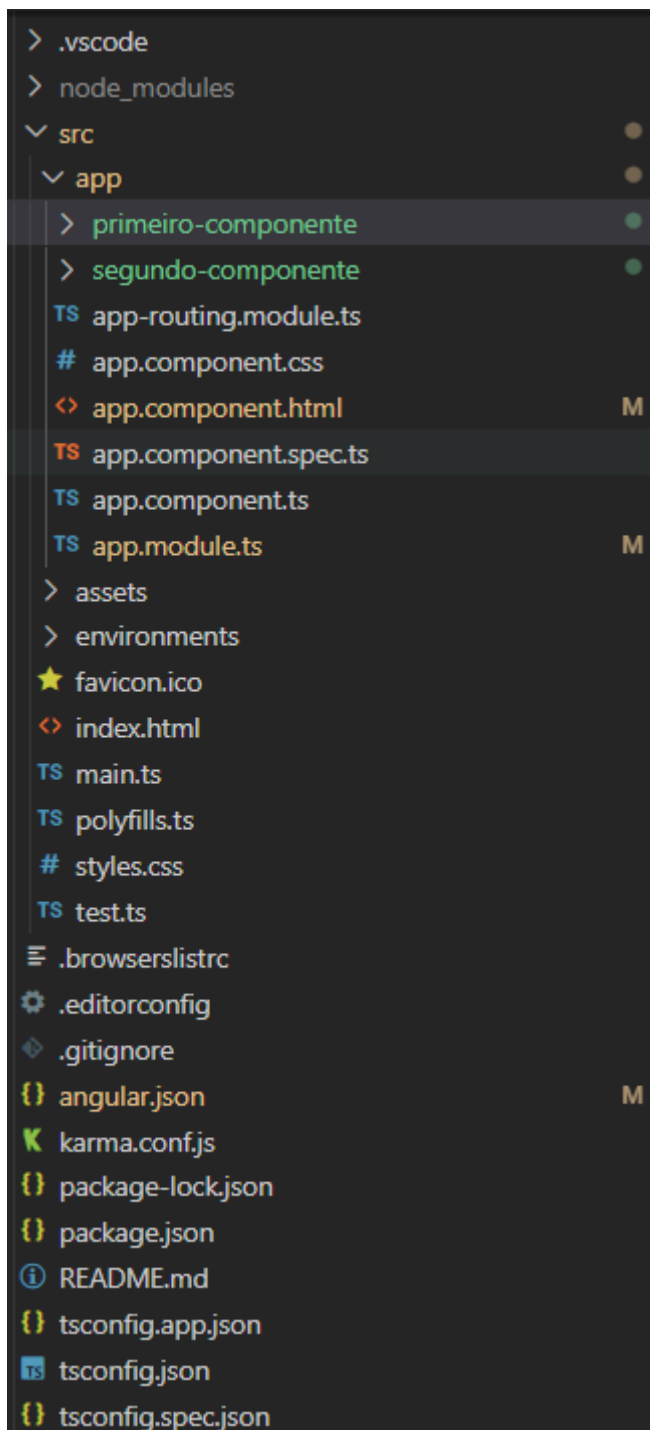
Isso é uma pergunta se queremos adicionar a biblioteca angular routing, que serve para fazermos o roteamento das nossas paginas.



```
? Which stylesheet format would you like to use? (Use arrow keys)  
> CSS  
SCSS [ https://sass-lang.com/documentation/syntax#scss  
Sass [ https://sass-lang.com/documentation/syntax#the-indented  
Less [ http://lesscss.org
```

Pergunta referente ao formato de arquivos de stylesheet utilizado.

Estrutura do projeto:



No arquivo package são colocadas as dependências do projeto.

No arquivo angular são colocadas as configurações do projeto, como sourceRoot que determina a pasta do projeto e prefix que determina o prefixo do projeto.

Como no prefix do angular.json esta app a pasta app ira conter nossos componentes.

```

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

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { PrimeiroComponenteComponent } from './primeiro-componente/Primeiro-Componente.component';
import { SegundoComponenteComponent } from './segundo-componente/segundo-componente.component';

@NgModule({
  declarations: [
    AppComponent,
    PrimeiroComponenteComponent,
    SegundoComponenteComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }

```

No arquivo app.module.ts contém os componentes que importamos.

```

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

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { PrimeiroComponenteComponent } from './primeiro-componente/Primeiro-Componente.component';
import { SegundoComponenteComponent } from './segundo-componente/segundo-componente.component';

@NgModule({
  declarations: [
    AppComponent,
    PrimeiroComponenteComponent,
    SegundoComponenteComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }

```

No arquivo app.component.html possui a aplicação dos componentes na tela.

```

<app-primeiro-componente></app-primeiro-componente>
<app-segundo-componente></app-segundo-componente>

```

Na pasta assets são colocados arquivos de mídia como imagens por exemplo.

Components:

Componentes são “partes de códigos html estilizados e reutilizáveis”.

Estrutura do componente:

Devemos importar a classe componente do angular/core.

Usamos o @Component({}) para especificarmos o componente.

Usamos o método export class nome de importação{} para especificarmos como exportamos o componente.

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

@Component({
  selector: "app-primeiro-componente",
  template: "<h2>Hello-word</h2>",
  styles: ["h2{ color: red;}"]
})
export class PrimeiroComponenteComponent {}
```

No componente temos que especificar o seletor o template e o style o template e o style podem ser usados através de url para outros arquivos:

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

@Component({
  selector: 'app-segundo-componente',
  templateUrl: './segundo-componente.component.html',
  styleUrls: ['./segundo-componente.component.css']
})
export class SegundoComponenteComponent {
```

Podemos criar a estrutura de um componente através do comando:

ng generated component nomedocomponente

Interpolação de texto:

É a forma que inserimos dados da classe no template:

Utilizamos {{nomedavariavel}} para dizermos o nome da variável na classe.

```
<p>{{nome}}</p>
```

```
export class SegundoComponenteComponent {  
  nome = "joão";  
}
```

Podemos usar | para formatarmos o texto o angular chama de **pipes** e existem diversas formatações possíveis

<https://angular.io/guide/pipes>

Pipe personalizado:

Podemos criar uma classe Typescript com um pipe personalizado usando o seguinte comando:

ng generate pipe nomepipe

```
import { Pipe, PipeTransform } from '@angular/core';

@Pipe({
  name: 'multiplicaPor'
})
export class MultiplicaPorPipe implements PipeTransform {

  transform(value: unknown, ...args: unknown[]): unknown {
    return null;
  }
}
```

Essa classe pipe também é adicionado no app.module.ts

Sintaxe do transform:

```
transform(variável: tipoprimitivo, ...args: tipoprimitivo[]): tipoprimitivo {
  return retorno;
}
```

```
transform(value: number, multiplicador: number): number {
  return value * multiplicador;
}
```

Exemplo de aplicação do pipe:

```
<p>segundo-componente works!</p>
<p>{{nome}}</p>
<p>{{2 | multiplicaPor:2}}</p>
```

Property_Binding:

É uma forma de linkarmos propriedades de componentes html com variáveis das classes componente:

Varivéis:

```
export class SegundoComponenteComponent {  
  nome = "joão";  
  urlImagem = "/assets/Capturar3.PNG";  
}
```

Property_binding:

```
<img [src]="urlImagem">
```

Event_Binding:

É uma forma de linkarmos um método nas classes a um componente HTML:

Método:

```
export class SegundoComponenteComponent {  
  nome = "joão";  
  urlImagem = "/assets/Capturar3.PNG";  
  
  mostrarDataNascimento(){  
    alert('18/08/2001');  
  }  
}
```

Event_Binding:

```
<img [src]="urlImagem" />  
<button (click)="mostrarDataNascimento()" type="button"></button>
```

Two_Way_Data_Binding:

Para utilizarmos um input em uma variável utilizamos o data_binding:

Variável:

```
export class TwoWayDataBindingComponent {  
  nome = "";  
}
```

Data_binding:

```
<p>{{ nome }}</p>  
<label for="nome">digite um nome</label>  
<input type="text" [(ngModel)]="nome">
```

Porém devemos importar a biblioteca **FormsModule** no app.module.ts:

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

Renderizando lista:

Para renderizarmos uma lista primeiro criamos uma.

Criamos a interface em um arquivo ts separado com as propriedades que serão criadas na classe do componente:

```
export interface Celular{  
  id: number;  
  nome: string;  
  descricao?: string;  
  esgotado: boolean;  
}
```



```
export class RenderizandoListasComponent {
  celulares: Celular[] = [
    { id: 1, nome: "moto g8", descricao: "muito ruim", esgotado: false },
    { id: 2, nome: "moto g9", descricao: "muito ruim", esgotado: false },
    { id: 3, nome: "moto g10", esgotado: true },
    { id: 4, nome: "moto g11", esgotado: false },
  ]
}
```

```
<p>renderizando-listas works!</p>
<h1>renderizando uma lista</h1>
<ul>
  <li *ngFor="let celular of celulares">
    <strong>{{ celular.nome }}</strong>
    <p>{{ celular.descricao }}</p>
    <span>{{ celular.esgotado ? "esgotado" : "disponivel" }}</span>
  </li>
</ul>
```

Componente Personalizado:

```
export class ComponentePersonalizadoComponent {
  @Input() nome = ""
  @Input() sobrenome = ""
}
```

```
<p>{{ nome }} {{sobrenome}}</p>
```

```
<app-componente-personalizado nome="João" sobrenome="biruleibi"></app-componente-personalizado>
```

Comunicação entre componentes:

```
export class ComponentePaiComponent {  
  sobrenome = "da silva";  
  nomeCompleto(nomeCompleto: any){  
    alert('O nome completo é: ' + nomeCompleto);  
  }  
}
```

```
<h1>interação entre componentes</h1>  
<h2>O sobrenome esta definido no elemento pai: {{sobrenome}}</h2>  
<app-componente-filho [sobrenome]="sobrenome" (mostrarnome)="nomeCompleto($event)" #filho></app-componente-filho>  
<h3>o nome completo é: {{filho.nome}} {{sobrenome}}</h3>
```

```
export class ComponenteFilhoComponent {  
  nome = "";  
  @Input () sobrenome = "";  
  @Output() mostrarnome = new EventEmitter();  
}
```

```
<h3>O sobrenome veio do elemento pai: {{sobrenome}}</h3>  
<label for="nome">Digite o nome</label>  
<input type="text" id="nome" [(ngModel)]="nome">  
<button type="button" (click)="mostrarnome.emit(nome + ' ' + sobrenome)">Mostrar nome</button>
```

Serviços:

Um serviço basicamente é uma funcionalidade que poderá ser utilizada em diversos componentes.

Para gerar um serviço de forma automática se utiliza o comando:

ng generate service nomedosserviço

isso irá gerar um arquivo typescript:

```
TS logger.service.ts
```

U

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

@Injectable({
  providedIn: 'root'
})
export class LoggerService {

  constructor() { }
}
```

```
export class LoggerService {
  mensagens: string[] = [];
  logar(mensagem: string){
    alert(mensagem);
    this.mensagens.push(mensagem);
  }
  exibiTodosOsLogs(){
    console.log(this.mensagens);
  }
}
```

Aplicando o servisse:

```
export class ExemploServicesComponent {
  nome = "";
  adicionarNome(){
    this.logger.logar("o nome " + this.nome + " foi adicionado");
    this.logger.exibiTodosOsLogs;
  }

  constructor(private logger: LoggerService){

  }
}
```

```
export class ExemploService2Component {

  produto = "";

  constructor(public logger: LoggerService){

  }

}
```

No caso de atribuído um public ao serviço podemos utiliza-lo no template:

```
<button type="button" (click)="this.logger.exibiTodosOsLogs()">Adicionar Produto</button>
```

Ciclo de vida dos componentes:

O primeiro evento a ser disparado no componente é o:

```
ngOnChanges(changes: SimpleChanges): void {
  console.log("O evento ngOnChanges disparou com as seguintes mudanças " + changes);
}
```

O segundo evento a ser disparado no componente é o:

```
ngOnInit(): void {
  console.log("OnInit");
  this.timer = setInterval(() => this.horario = new Date(), 1000);
}
```

O ultimo evento a ser disparado acontece quando o componente sai de tela e é o:

```
ngOnDestroy(): void {
  clearInterval(this.timer);
}
```

Ng_Content:

É utilizado quando queremos colocar componentes dentro de outros.

```
export class CardComponent {  
  @Input() titulo = "";  
  @Input() cor = "#ccc";  
}
```

```
<div class="container" [ngStyle]="{'background-color': cor}">  
  <h2>{{ titulo }}</h2>  
  <div class="content">  
    <ng-component></ng-component>  
  </div>  
</div>
```

```
<app-card titulo="baby do baby" cor="red">  
  <app-primeiro-componente></app-primeiro-componente>  
  <app-segundo-componente></app-segundo-componente>  
</app-card>
```

Adicionando_Roteamento:

É importante marcar N na hora de criar o projeto indicando que não quer importar e configurar a biblioteca de roteamento de forma automática.

```
C:\Users\gravacao\curso-angular-proway>ng new roteamento  
? Would you like to add Angular routing? (y/N) █
```

Devemos instalar a biblioteca angular router através do comando:

```
npm i @angular/router
```

Agora devemos criar um modulo responsável pelo roteamento:

```
ng generate module app-routing
```

esse arquivo deve ir na pasta app.

Devemos também acrescentar essa classe no app.module:

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

Após isso vamos acessar essa classe:

```
const routes: Routes = [
  {
    path: "primeira-pagina", component: PrimeiraPaginaComponent
  }
];
@NgModule({
  declarations: [],
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

Com essa configuração acrescentamos a página “primeira-pagina” ao componente PrimeiraPaginaComponent:

▼ primeira-pagina	●
# primeira-pagina.component.css	U
<> primeira-pagina.component.html	U
TS primeira-pagina.component.spec.ts	U
TS primeira-pagina.component.ts	U

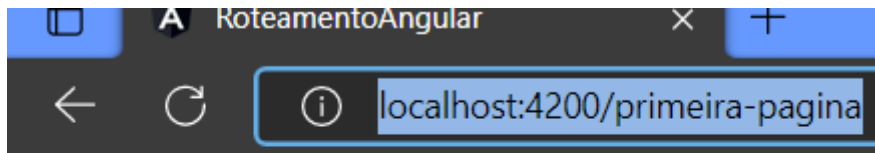
Após isso no arquivo app.component.html adicionamos a tag:

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

Depois disso apenas colocamos o server para rodar através do comando:

ng serve --open

após isso temos a pagina <http://localhost:4200/> como a pagina app.component.html e a pagina <http://localhost:4200/primeira-pagina> como a pagina do componente que criamos:



primeira pagina

Navegando entre páginas:

```
<nav>
  <ul>
    <li>
      <a routerLink="primeira-pagina">Primeira página</a>
    </li>
    <li>
      <a routerLink="segunda-pagina">Segunda página</a>
    </li>
  </ul>
</nav>
<router-outlet></router-outlet>
```

O atributo routerLink navega entre as páginas roteadas.

As tags que estiverem anteriores a tag <router-outlet> seram passadas para as páginas subsequentes.

Routing:


```
const routes: Routes = [
  {
    path: "primeira-pagina", component: PrimeiraPaginaComponent
  },
  {
    path: "segunda-pagina", component: SegundaPaginaComponent
  }
];
@NgModule({
  declarations: [],
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

Movendo através de métodos:

```
<button type="button" (click)="acessarSegundaPagina()">Segunda Pagina</button>
```

```
export class PrimeiraPaginaComponent {

  constructor(
    private router: Router
  ){}

  acessarSegundaPagina(){
    this.router.navigate(["segunda-pagina"]);
  }

}
```

Parâmetros de URL:

http://localhost:4200/produtos/1



Parâmetros da rota

http://localhost:4200/cliente?nome=João&idade=23



Parâmetros de consulta

Para trabalharmos com parâmetros de rota devemos criar o path correspondente:

```
const routes: Routes = [
  { path: "primeira-pagina", component: PrimeiraPaginaComponent },
  { path: "segunda-pagina", component: SegundaPaginaComponent },
  { path: "", redirectTo: "primeira-pagina", pathMatch: "full" },
  { path: "pagina-com-parametros/:id", component: PaginaComParametrosComponent },
  { path: "**", component: PaginaNaoEncontradaComponent }
]
```

Classe do componente gerado pelo id:

```
export class PaginaComParametrosComponent implements OnInit {
  id: number | null = null;

  constructor(private route: ActivatedRoute) { }

  ngOnInit(): void {
    this.route.paramMap.subscribe(params => {
      this.id = params.get("id");
    })
  }
}
```

```

ngOnInit(): void {
  this.route.paramMap.subscribe(params => {
    this.id = Number(params.get("id"));
  });

  this.route.queryParamMap.subscribe(params => {
    this.idade = Number(params.get("idade"));
    this.nome = params.get("nome");
  })
}

```

```

<p>Parâmetro da rota: {{ id }}</p>
<p *ngIf="nome">Query Param de nome: {{ nome }}</p>
<p>Query Param de idade: {{ idade }}</p>

```

Diferença module e componente:

Um modulo irá conter diversos componentes.

Lazy Loading:

Devemos primeiro criar um modulo para o lazy loading:

```
ng generate module lazy-loading --route=lazy-loading --module=app.module
```

interessante observar que nesse commando a rota para acessar o module é lazy-loading e ele está atrelado ao modulo principal.

Interessante demonstrar o path gerado nesse comando:

```
{ path: 'lazy-loading', loadChildren: () => import('./lazy-loading/lazy-loading.module').then(m => m.LazyLoadingModule) }
```

O lazy loading já está funcionando quando acessamos o module app não é carregado o modulo lazy-loading, apenas quando acessamos a url atrelada.

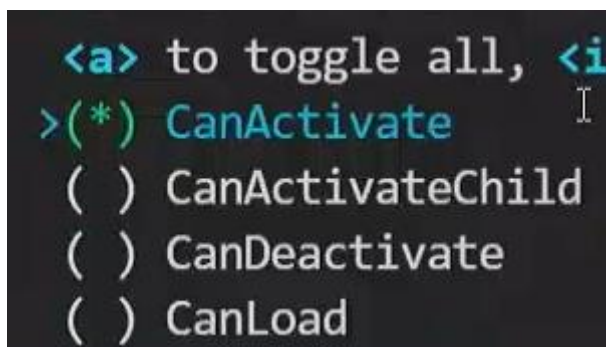
Route_guard:

É uma forma de protegemos nossas páginas.

Devemos criar um arquivo guard:

ng generate guard nomedoarquivo

com isso ira aparecer para selecionarmos o tipo de guard que sera usado:



```
<a> to toggle all, <i>  
>(*) CanActivate  
  ( ) CanActivateChild  
  ( ) CanDeactivate  
  ( ) CanLoad
```

Usaremos o CanActivate.

Para usarmos o guard em uma pagina devemos modificar o path do componente:

Exemplo:

```
{ path: "pagina-protegida", component: PaginaProtegidaComponent, canActivate: [[AuthGuard]] },
```

Ou seja, passamos o tipo de guard e depois a classe do guard que vamos usar.

Configuração padrão da classe:

```
export class AuthGuardGuard implements CanActivate {
  canActivate(
    route: ActivatedRouteSnapshot,
    state: RouterStateSnapshot): Observable<boolean | UrlTree> | Promise<boolean | UrlTree> | boolean | UrlTree {
    return true;
  }
}
```

Devemos então criar uma forma de acessar essa página, para isso vamos criar um componente de login:

Go to component

```
<h2>Login</h2>
<label for="email">E-mail</label>
<input type="text" id="email" [(ngModel)]="email">
<label for="senha">Senha</label>
<input type="text" id="senha" [(ngModel)]="senha">
<button (click)="login()">Entrar</button>
```

```
//
export class LoginComponent implements
  email = "";
  senha = "";

  constructor() { }

  ngOnInit(): void {
  }

  login() {
  }
}
```

Devemos criar um serviço de autenticação para ser usado no componente de login:

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

@Injectable({
  providedIn: 'root'
})
export class AuthService {

  constructor() { }

  estaAutenticado(): boolean {

  }

  login(email: string, senha: string) {
```

```
  }

  logout() {}
}
```

LocalStorage e SessionStorage:

São uma forma de manter informações, o LocalStorage ficara salvo no local, ou seja, mesmo que o usuário feche o navegador ou ate desligue o computador

essas informações serão mantidas. Já o SessionStorage, as informações serão perdidas quando a janela do navegador for fechada.

```
estaAutenticado(): boolean {  
    return !!sessionStorage.getItem("access-token");  
}
```

```
login(email: string, senha: string): boolean {  
    if (email === "admin@admin.com.br" && senha === "123456") {  
        sessionStorage.setItem("access-token", this.accessToken);  
        return true;  
    }  
  
    return false;  
}
```

```
logout() {  
    sessionStorage.clear();  
}
```

Após isso vamos configurar o componente:

```
export class LoginComponent implements OnInit {  
    email = "";  
    senha = "";  
  
    constructor(  
        private auth: AuthService,  
        private router: Router  
    ) { }
```

```
login() {
  if (this.auth.login(this.email, this.senha)) {
    this.router.navigate(["pagina-protegida"]);
    return;
  }

  alert("Login está inválido");
  this.email = "";
  this.senha = "";
}
```

Mesmo com isso nos apenas configuramos a autenticação e o salvamento das informações porem ainda não temos acesso a pagina protegida precisamos configurar o guard:

```
export class AuthGuard implements CanActivate {
  constructor(
    private auth: AuthService,
    private router: Router
  ) {}

  canActivate(
    route: ActivatedRouteSnapshot,
    state: RouterStateSnapshot): Observable<boolean | UrlTree> | Promise<boolean | UrlTree> | boolean | UrlTree {
    if (!this.auth.estaAutenticado()) {
      this.router.navigate(["login"]);
      return false;
    }

    return true;
  }
}
```

by Capgemini

Criando a build do projeto:

Para criar a build do projeto utilizamos o comando:

```
ng build --base-href="url que irá ficar" --output-path nomedapasta
```

exemplo url github pages: <https://usuario.github.io/nomedorepositorio/>

exemplo:

```
ng build --base-href="https://usuario.github.io/nomedorepositorio/" --output-path docs
```

Configurando o git:

Identidade:

```
git config --global user.name "Fulano de tal"
```

```
git config --global user.email fulanodetal@example.com
```

Publicando projetos no github pages:

Após ter criado o build do projeto e configurado o git, vamos seguir os passos:

Comando:

```
git init -irá inicializar um repositório vazio
```

```
git add
```

```
git commit -m "build"
```

```
git branch -M main
```

```
git remote add origin https://github.com/username/repositorio.git
```

```
git push -u origin main
```

