

Universidade Augusto Mota



Luis Felipe Macedo dos Santos
Desenvolvimento de aplicações móveis
Charles, Artur Brandt, Oswaldo Peres.
02/05/2025

Para começar é necessário instalar a interface de linha de comando do [Ionic](#).

Siga as instruções da página citada.

Como exemplo, o seguinte comando pode ser utilizado.

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

O comando anterior instala o cli, programa de linha de comando que será utilizado no ciclo de vida de desenvolvimento da aplicação.

O seguinte comando cria um novo projeto, chamado formadoraII.

```
ionic start formadoraII blank --type=angular --capacitor
```

Uma mensagem será apresentada no terminal, por padrão a opção “Standalone” estará selecionada. Aperte enter.

O comando anterior realiza a criação de um template em branco de um projeto ionic utilizando o capacitor como plataforma de integração (para gerar o aplicativo “nativo”).

O nome do projeto é formadoraII, e será criado um diretório de mesmo nome, onde estarão os arquivos do projeto.

Adicionalmente, uma mensagem do Ionic oferecendo a criação de uma conta gratuita aparecerá no terminal, escolha não, aperte enter e prossiga. Ou, crie uma conta... :)

Tópico 1.

Construindo uma página principal com links para 2 páginas secundárias. O seguinte comando utiliza a interface de linha de comando do Ionic para automatizar a tarefa de criação de componetes e páginas. Os seguintes comandos farão a criação de 2 páginas (sobre e frases) e 1 serviço na ordem que foram citados. A partir daqui vou chamar as páginas pelo nome em inglês para facilitar a identificação das mesmas no código.

```
ionic generate page about
```

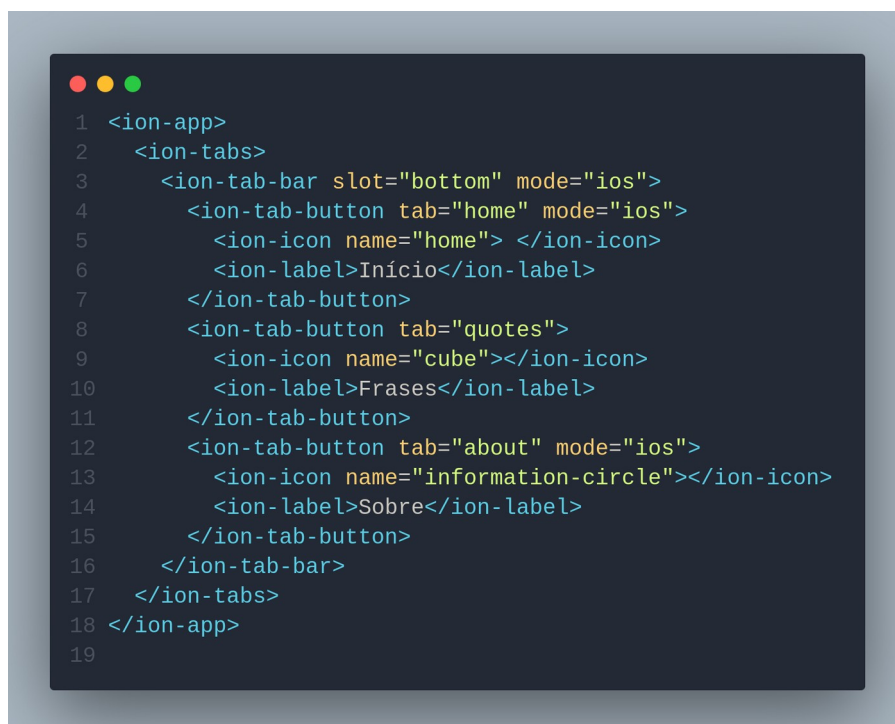
```
ionic generate page quotes
```

```
ionic generate service services/quotes
```

Cada linha citada acima, corresponde a um comando diferente. Assim, será criada uma estrutura no diretório “src” do projeto. Além disso, existe uma página chamada “home”, esta foi criada automaticamente quando iniciamos o projeto ionic.

O arquivo app.component.html é o equivalente ao index.html. Neste arquivo, podemos declarar o mecanismo de navegação entre as páginas. Aqui temos diversas opções, uma delas é o Ionic Tabs (ou guias) “... As guias podem ser usadas com o roteador Ionic para implementar a navegação baseada em guias. A barra de guias e a guia ativa serão resolvidas automaticamente com base na URL. Este é o padrão mais comum para navegação por guias. (Ionic API Docs).

Dessa forma implementamos o código presente na documentação do Ionic no componente app. Em app.component.html...



```
1 <ion-app>
2   <ion-tabs>
3     <ion-tab-bar slot="bottom" mode="ios">
4       <ion-tab-button tab="home" mode="ios">
5         <ion-icon name="home"> </ion-icon>
6         <ion-label>Início</ion-label>
7       </ion-tab-button>
8       <ion-tab-button tab="quotes">
9         <ion-icon name="cube"></ion-icon>
10        <ion-label>Frases</ion-label>
11      </ion-tab-button>
12      <ion-tab-button tab="about" mode="ios">
13        <ion-icon name="information-circle"></ion-icon>
14        <ion-label>Sobre</ion-label>
15      </ion-tab-button>
16    </ion-tab-bar>
17  </ion-tabs>
18 </ion-app>
19
```

Figure 1: Arquivo app.component.html

E em app.component.ts

```
1 import { Component } from '@angular/core';
2 import {
3   IonApp,
4   IonTabs,
5   IonTabBar,
6   IonTabButton,
7   IonIcon,
8   IonLabel,
9 } from '@ionic/angular/standalone';
10 import { addIcons } from 'ionicons';
11 import { informationCircle, home, cube } from 'ionicons/icons';
12 import { MenuController } from '@ionic/angular';
13
14 @Component({
15   selector: 'app-root',
16   templateUrl: 'app.component.html',
17   standalone: true,
18   imports: [IonLabel, IonIcon, IonTabButton, IonTabBar, IonTabs, IonApp],
19 })
20 export class AppComponent {
21   constructor(private menuctl: MenuController) {
22     addIcons({ home, informationCircle, cube });
23   }
24 }
25
```

Figure 2: Arquivo app.component.ts

A documentação do Ionic fornece componentes para serem utilizados nos aplicativos. Refira a [Docs Ionic Components](#) para entender como funcionam, e como utilizar.

No terminal onde executou os comandos de criação do componentes, utilize o seguinte comando para iniciar um servidor local.

ionic serve

Após alguns segundos, o ionic terá criado um servidor local para vermos a aplicação em tempo real enquanto a desenvolvemos.

No terminal haverá um link, siga o link.

O link se parece com <http://localhost:8081>. Leia as instruções no terminal se estiver confuso sobre onde clicar.

No navegador, veja o resultado esperado e compare com as imagens abaixo. Assumindo que tudo esteja funcionando corretamente clique em uma das guias e veja a navegação entre as páginas criadas anteriormente.

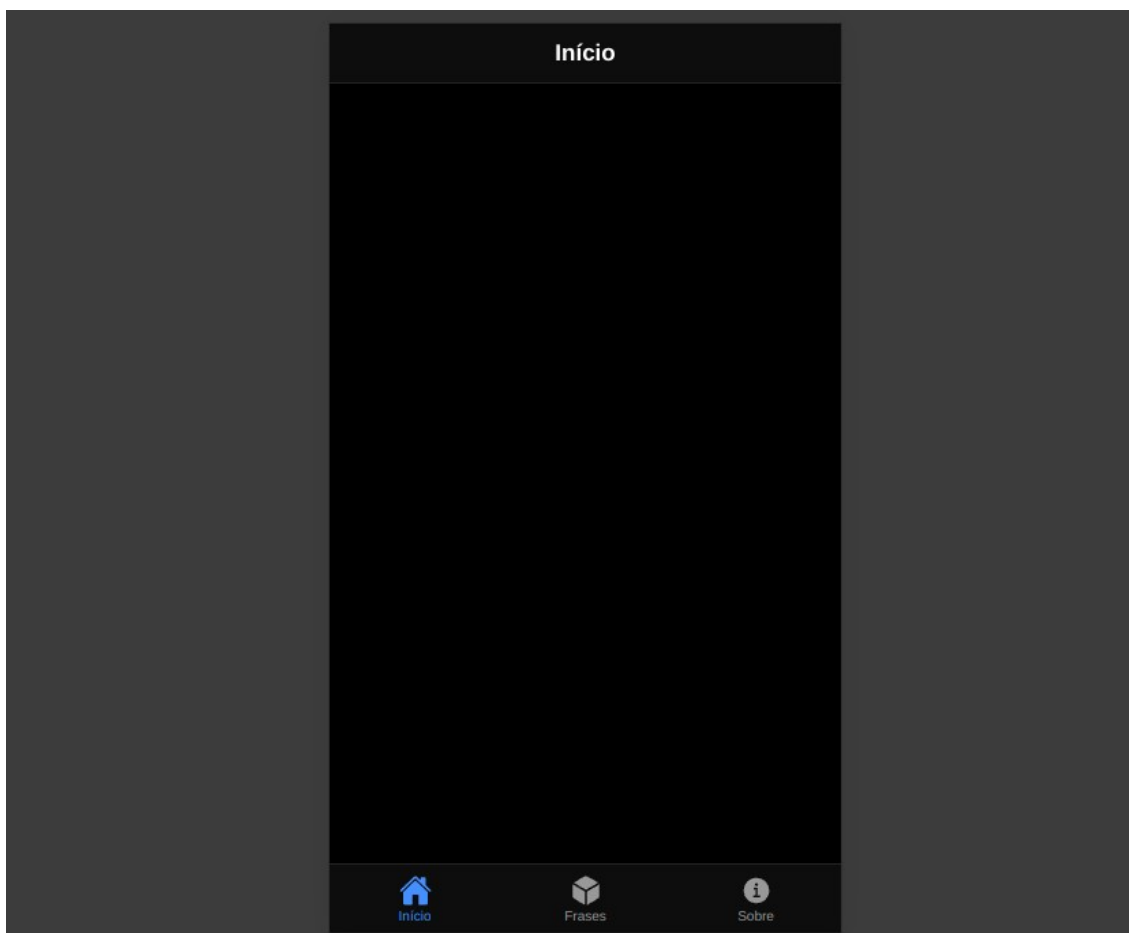


Figure 3: Página inicial Home

As páginas Início e Sobre tem conteúdo estático, apenas textos informativos. Na tela Início haverá instruções sobre como utilizar a aplicação, enquanto na tela Sobre haverá informações sobre as tecnologias que foram utilizadas.

Além dessas duas haverá a página quotes, a página quotes exibirá conteúdo dinâmico, consumindo dados de uma api externa. E para isso faremos a utilização de um serviço. Um serviço no Angular, é uma classe injetável, esse termo injetável se refere ao

conceito de injeção de dependência, que é um padrão de design que visa desacoplar classes, tornando o código modular.

Utilizando-se dessa modularidade, o serviço traz as funcionalidades implementadas na sua classe, podendo ser reutilizado em outros componentes.

Mas antes de falar disso, criaremos as duas primeiras páginas.

Em `home.page.html`



```
1 <ion-header>
2   <ion-toolbar>
3     <ion-title>Início</ion-title>
4   </ion-toolbar>
5 </ion-header>
6 <ion-content class="ion-padding">
7   <ion-card>
8     <ion-card-header>
9       <ion-card-title>Consumo de serviços externos</ion-card-title>
10    </ion-card-header>
11    <ion-card-content>
12      Consuma uma api externa de frases motivacionais. Navegue até a aba
13      "Quotes"
14    </ion-card-content>
15  </ion-card>
16  <ion-card>
17    <ion-card-header>
18      <ion-card-title>Navegação através de abas</ion-card-title>
19    </ion-card-header>
20    <ion-card-content>
21      Selecione uma das abas abaixo para acessar as diferentes seções do
22      aplicativo. Volte quando quiser.
23    </ion-card-content>
24  </ion-card>
25  <ion-card>
26    <ion-card-header>
27      <ion-card-title>Sobre o aplicativo</ion-card-title>
28    </ion-card-header>
29    <ion-card-content>
30      Entenda as tecnologias utilizadas no projeto.
31    </ion-card-content>
32  </ion-card>
33 </ion-content>
34
```

Figure 4: Arquivo `home.page.html`

E em `home.page.ts`

```

1 import { Component } from '@angular/core';
2 import {
3   IonContent,
4   IonHeader,
5   IonTitle,
6   IonToolbar, IonCard, IonCardHeader, IonCardTitle, IonCardContent } from '@ionic/angular/standalone';
7
8 @Component({
9   selector: 'app-home',
10  templateUrl: 'home.page.html',
11  styleUrls: ['home.page.scss'],
12  imports: [IonCardContent, IonCardTitle, IonCardHeader, IonCard, IonContent, IonHeader, IonTitle, IonToolbar],
13 })
14 export class HomePage {
15   constructor() {}
16 }
17

```

Figure 5: Arquivo home.page.ts

Em about.page.html

```

1 <ion-header>
2   <ion-toolbar>
3     <ion-title>Sobre</ion-title>
4   </ion-toolbar>
5 </ion-header>
6 <ion-content>
7   <ion-list mode="md" lines="none" inset="false">
8     <ion-item>
9       <ion-card>
10        <ion-card-header>
11          <ion-card-title>Ionic Framework</ion-card-title>
12          <ion-card-subtitle>7.2.1</ion-card-subtitle>
13        </ion-card-header>
14        <ion-card-content>
15          <ion-text>
16            Ionic é um framework de desenvolvimento de aplicativos móveis que
17            permite criar aplicativos híbridos usando tecnologias web como HTML,
18            CSS e JavaScript. Ele fornece uma biblioteca de componentes prontos
19            para uso, facilitando a criação de interfaces de usuário atraentes e
20            responsivas.
21          </ion-text>
22        </ion-card-content>
23      </ion-card>
24    </ion-item>

```

Figure 6: Arquivo about.page.html

(conteúdo da imagem truncado, o conteúdo real da imagem representa uma série de elementos <ion-item> com informações diferentes. Veja o arquivo completo no código da aplicação.)

E em about.page.ts



```
1 import { Component, OnInit } from '@angular/core';
2 import {
3   IonContent, IonHeader, IonToolbar,
4   IonTitle, IonCard, IonCardHeader,
5   IonCardTitle, IonCardContent, IonList,
6   IonItem, IonCardSubtitle, IonText,
7 } from '@ionic/angular/standalone';
8
9 @Component({
10   selector: 'app-about',
11   templateUrl: './about.page.html',
12   styleUrls: ['./about.page.scss'],
13   imports: [
14     IonText, IonCardSubtitle, IonItem,
15     IonList, IonCardContent, IonCardTitle,
16     IonCardHeader, IonCard, IonContent,
17     IonHeader, IonToolbar, IonTitle,
18   ],
19 })
20 export class AboutPage implements OnInit {
21   constructor() {}
22
23   ngOnInit() {}
24 }
25
```

Figure 7: Arquivo about.page.ts

Resultado das telas.



Figure 8: Página Início




Figure 9: Página Sobre

O Ionic Tabs fornece um componente de alto nível para navegação. Ionic Tabs pode ser usado tanto para navegação baseado em rotas, quando de componentes com modais. As páginas Home e Sobre foram criadas com auxílio dos componentes do ionic. Que são componentes importados no nosso código, eles auxiliam na velocidade do desenvolvimento pois diminuem o tempo escrevendo, e implementando diversas funções complexas em nosso nome.

A página Quotes será igualmente simples, mas com algumas implementações interessantes do Ionic. Ela terá: Um compoente `<ion-refresher>` que permitirá que o usuário atualize a página com o gesto nativo de “puxar a tela para baixo”; Também utiliza o componente `<ion-skeleton>` para carregar uma animação enquanto os dados vindos dos serviços estão

pendentes; A diretiva `*ngFor` será utilizada para gerar dinamicamente o conteúdo da página; E a diretiva `@if` e `@else` serão utilizadas para controlar quando o componente `<ion-skeleton-text>` deverá ser renderizado, enquanto aguarda a chegada dos dados; Além destas particularidades faremos uso de `AsyncPipe`, uma diretiva do Angular que permite implementar assíncronidade nas diretivas `@if` e `*ngFor`. Leia mais sobre [neste artigo](#).

A começar pelo serviço, que contém o seguinte código.



```
1 import { HttpClient } from '@angular/common/http';
2 import { Injectable } from '@angular/core';
3 import { Observable, Subscriber, Subscription } from 'rxjs';
4
5 import { Quote } from '../types/quotes';
6
7 @Injectable({
8   providedIn: 'root',
9 })
10 export class QuotesService {
11   constructor(private http: HttpClient) {}
12
13   RandomQuotes(nQuotes: number = 10): Observable<Quote[]> {
14     return this.http.get<Quote[]>(
15       'https://dummyjson.com/quotes/random/' + nQuotes
16     );
17   }
18 }
19
```

Figure 10: Arquivo `quotes.service.ts`

Note que no construtor estamos recebendo uma dependência, que está sendo injetada na nossa classe. Esta dependência é o `HttpClient`, cliente para requisições http do Angular, através da biblioteca RxJS. Para termos essa dependência injetada tivemos que importá-la no módulo `main.ts`, no objeto “Providers”. Veja a linha 14, que executa a função `provideHttpClient()`.



```

1 import { bootstrapApplication } from '@angular/platform-browser';
2 import { RouteReuseStrategy, provideRouter, withPreloading, PreloadAllModules } from '@angular/router';
3 import { IonicRouteStrategy, provideIonicAngular } from '@ionic/angular/standalone';
4 import { provideHttpClient } from '@angular/common/http';
5
6 import { routes } from './app/app.routes';
7 import { AppComponent } from './app/app.component';
8
9 bootstrapApplication(AppComponent, {
10   providers: [
11     { provide: RouteReuseStrategy, useClass: IonicRouteStrategy },
12     provideIonicAngular(),
13     provideRouter(routes, withPreloading(PreloadAllModules)),
14     provideHttpClient(),
15   ],
16 });
17

```

Figure 11: Arquivo main.ts

De volta ao serviço, o método `RandomQuotes`, recebe como parâmetro opcional um inteiro que representa a quantidade de objetos que devem ser retornados. De acordo com Angular Docs (2025) “O `HttpClient` produz o que o RxJS chama de Observables “frios”, o que significa que nenhuma requisição real acontece até que o Observable seja inscrito. Somente então a requisição é efetivamente despachada para o servidor. Inscrever-se no mesmo Observable várias vezes acionará múltiplas requisições de backend. Cada inscrição é independente.”

Por esse motivo, o método do serviço se chama `RandomQuotes`, e não `getRandomQuotes`. Pois o retorno da função ainda não contém os dados, mas sim um diagrama de como conseguí-los.

Implementamos um serviço com objetos que abstraem as requisições que faremos. Poderíamos implementar o `get` no serviço mas aqui podemos construir algo mais simples, e mais didático.

No arquivo `quotes.page.ts`, injetaremos o serviço `QuotesService`, e utilizaremos seus métodos em duas funções, uma que será executada quando o componente for renderizado. E outra quando o elemento `<ion-refresher>` disparar o evento para recarregarmos a página com novos dados.

Na classe QuotesPage, definimos uma propriedade publica que receberá um Observable do tipo array de objetos quotes.

Segundo Pardeep Jain (2016) “...

Mostly we use ngOnInit for all the initialization/declaration and avoid stuff to work in the constructor. The constructor should only be used to initialize class members but shouldn't do actual "work".

So you should use constructor() to setup Dependency Injection and not much else. ngOnInit() is better place to "start" - it's where/when components' bindings are resolved...”

A declaração das dependências é feita no método construtor, e o início do ciclo de vida é dado pelo método ngOnInit, portanto recebemos o serviço no construtor e realizamos a atribuição da propriedade quotes com o Observable do serviço outrora injetado, no método ngOnInit.

```

1 import { Component, OnInit } from '@angular/core';
2 import { CommonModule } from '@angular/common';
3 import { FormsModule } from '@angular/forms';
4 import {
5   IonContent, IonList, IonItem,
6   IonLabel, IonTitle, IonHeader,
7   IonToolbar, IonListHeader, IonRefresher,
8   IonRefresherContent, IonSkeletonText,
9 } from '@ionic/angular/standalone';
10 import { QuotesService } from '../services/quotes.service';
11 import { Quote } from '../types/quotes';
12 import { Observable } from 'rxjs';
13 @Component({
14   selector: 'app-quotes',
15   templateUrl: './quotes.page.html',
16   styleUrls: ['./quotes.page.scss'],
17   standalone: true,
18   imports: [
19     IonSkeletonText, IonRefresherContent, IonRefresher,
20     IonTitle, IonItem, IonList,
21     IonContent, CommonModule, FormsModule,
22     IonLabel, IonHeader, IonToolbar,
23     IonListHeader,
24   ],
25 })
26 export class QuotesPage implements OnInit {
27   public quotes: Observable<Quote[]> = new Observable<Quote[]>();
28
29   constructor(private quotesService: QuotesService) {}
30
31   ngOnInit() {
32     this.quotes = this.quotesService.RandomQuotes();
33   }
34
35   refreshRandomQuotes(event: CustomEvent) {
36     this.quotes = this.quotesService.RandomQuotes();
37     (event.target as HTMLIonRefresherElement).complete();
38   }
39 }
40

```

Figure 12: Arquivo quotes.page.ts

O método `refreshRandomQuotes` é executado pelo elemento `<ion-refresher>` quando for disparado o evento de atualização de tela.

Note como no método atribuímos um `Observable` a propriedade `quotes` novamente. Isto é feito pois, o observable será consumido no template, e o evento `ionRefresh` não atualiza a página, ou seja, não renderiza o componente inteiro. Caso o componente fosse renderizado por completo não haveria necessidade de apontar para um novo observable.

O fato é que no dispositivo mobile, essa ação de refresh não renderiza toda a tela do aplicativo, pois estamos tratando de um SPA, e sim apenas o conteúdo que depende dele. Importante deixar isso claro, pois como visto anteriormente um observable pode ser consumido mais de uma vez, e não haveria necessidade de instanciar um novo. Porém, os dados na view já estão presentes, e para inscrever novamente nesse observable, precisaríamos ou de uma nova renderização, para que a diretiva *ngFor fosse re-executada, gerando uma nova inscrição e recebendo novos dados, ou podemos fazer isso atribuindo um novo observable a propriedade quotes, que obrigará o AsyncPipe a se inscrever novamente, recebendo novos dados, pois a referência do objeto mudou.

Por fim, vejamos o conteúdo do arquivo quotes.page.html.



```
1 <ion-header>
2   <ion-toolbar>
3     <ion-title>Frases</ion-title>
4   </ion-toolbar>
5 </ion-header>
6 <ion-content>
7   <ion-refresher
8     slot="fixed"
9     (ionRefresh)="refreshRandomQuotes($event)"
10    mode="ios"
11  >
12    <ion-refresher-content></ion-refresher-content>
13  </ion-refresher>
14  @if(quotes | async) {
15
16    <ion-list mode="md" inset="false" lines="none">
17      <ion-list-header>
18        <ion-label>Frases</ion-label>
19      </ion-list-header>
20      <ion-item *ngFor="let quote of (quotes | async) ">
21        <ion-label>
22          <h2>{{ quote.quote }}</h2>
23          <p>-- {{ quote.author }}</p>
24        </ion-label>
25      </ion-item>
26    </ion-list>
```

Figure 13: Arquivo quotes.page.html, parte 1

```
1  } @else {
2  <ion-list>
3    <ion-list-header>
4      <ion-label>
5        <ion-skeleton-text
6          animated="true"
7          style="width: 40%"
8        ></ion-skeleton-text>
9      </ion-label>
10    </ion-list-header>
11    <ion-item *ngFor="let skel of [1,2,3,4,5]">
12      <ion-label>
13        <h2>
14          <ion-skeleton-text
15            animated="true"
16            style="width: 100%"
17          ></ion-skeleton-text>
18        </h2>
19        <p>
20          <ion-skeleton-text
21            animated="true"
22            style="width: 50%"
23          ></ion-skeleton-text>
24        </p>
25      </ion-label>
26    </ion-item>
27  </ion-list>
28  }
29 </ion-content>
30
```

Figure 14: Arquivo quotes.page.html, parte 2

O conteúdo é relativamente extenso, as partes relevantes são os elementos do Ionic `<ion-refresher>`, `<ion-refresher-content>` e `<ion-skeleton-text>` que foram citados anteriormente e que permitem adicionar funcionalidades que auxiliam na experiência do usuário com facilidade. Além desses elementos, é importante citar as diretivas do Angular `@if` e `@else`, e `*ngFor`. Estas diretrizes permitem respectivamente, renderizar conteúdo html, baseado em tomadas de decisão lógicas. E gerar conteúdo html dinamicamente através de um laço de repetição. Note também a utilização do AsyncPipe nas linhas 14 e 20. A propriedade quotes é um objeto sem dados durante a

inicialização do componente, ao renderizar o componente o observable será consumido pelo AsyncPipe gerando uma requisição http. Dessa forma o conteúdo será gerenciado de forma assíncrona, evitando erros de “race condition” onde o dado ainda não existe mas já está sendo renderizado. Parece um pouco complicado, e talvez seja. Mas é um conceito ligeiramente complexo, e difícil de transmitir a ideia através de poucas palavras.

Ao final, poderemos ver este resultado ao acessar a página quotes.

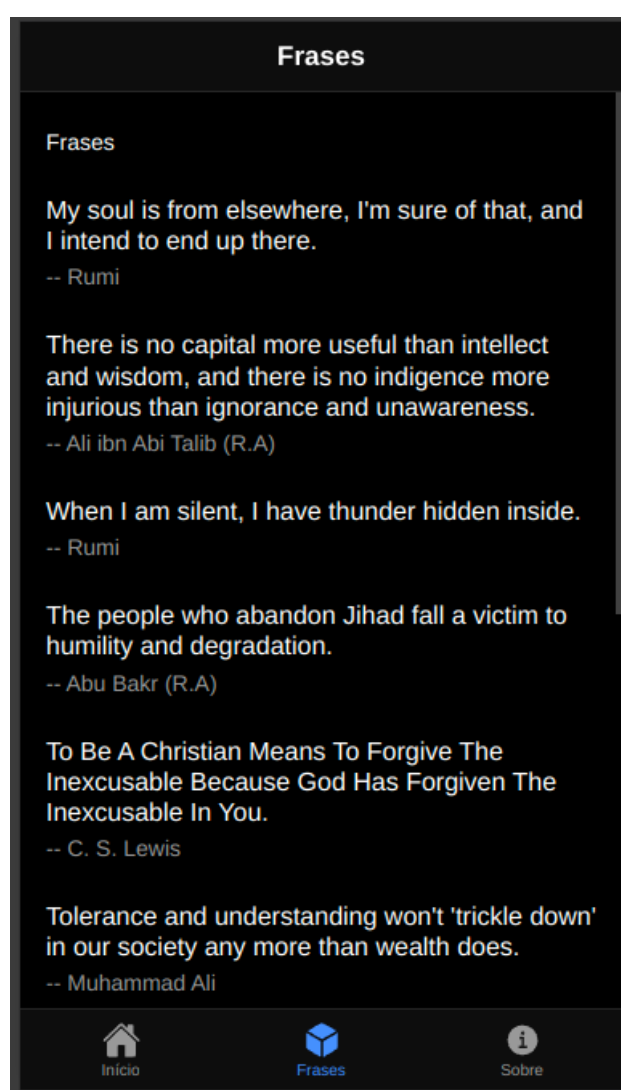


Figure 15: Página quotes

Referências.

IONIC. Ionic Tabs. Disponível em: <<https://ionicframework.com/docs/api/tab-bar>>. Acesso em: 02 maio 2025.

IONIC. Documentação de Componentes do Ionic. Disponível em: <<https://ionicframework.com/docs/api/>>. Acesso em: 02 maio 2025.

ANGULAR. Angular Dev Best Practices. Disponível em: <<https://angular.dev/guide/http/making-requests#best-practices>>. Acesso em: 02 maio 2025.

JAINDEEP. Difference between constructor and ngOnInit. Stack Overflow, 2016. Disponível em: <<https://stackoverflow.com/questions/35763730/difference-between-constructor-and-ngoninit>>. Acesso em: 02 maio 2025.