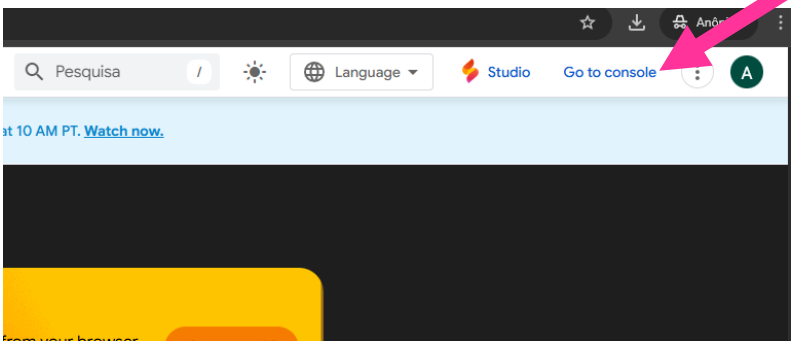
Ionic com Firebase

O **Firebase** é uma plataforma de desenvolvimento de aplicativos móveis e web, mantida pelo Google. Ele funciona como um **Backend as a Service (BaaS),** o que significa que ele oferece um conjunto de serviços e ferramentas de backend prontos para uso, permitindo que desenvolvedores criem aplicativos de alta qualidade de forma mais rápida e eficiente, sem a necessidade de gerenciar sua própria infraestrutura de servidor.

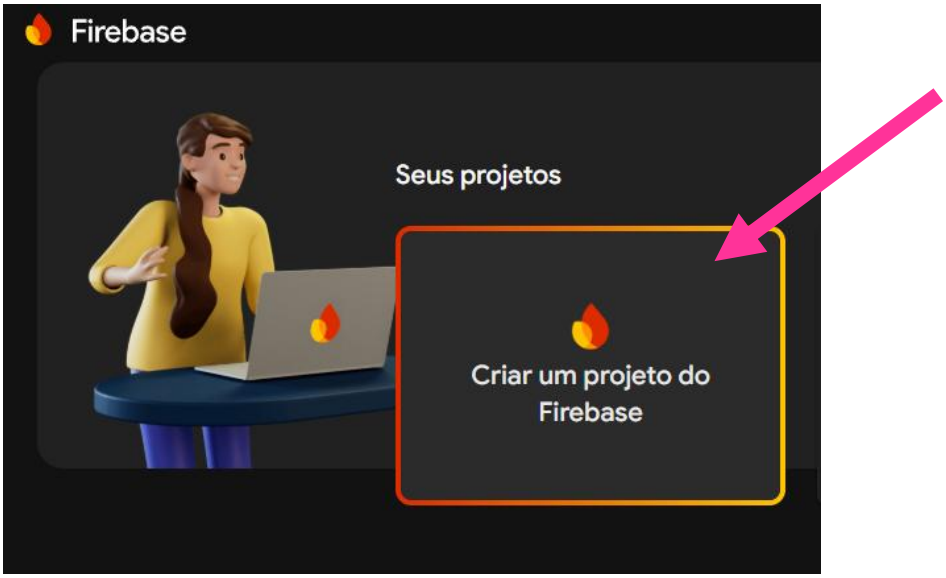
Passo 1: Criando o projeto no Firebase

Acesse o link: https://firebase.google.com/

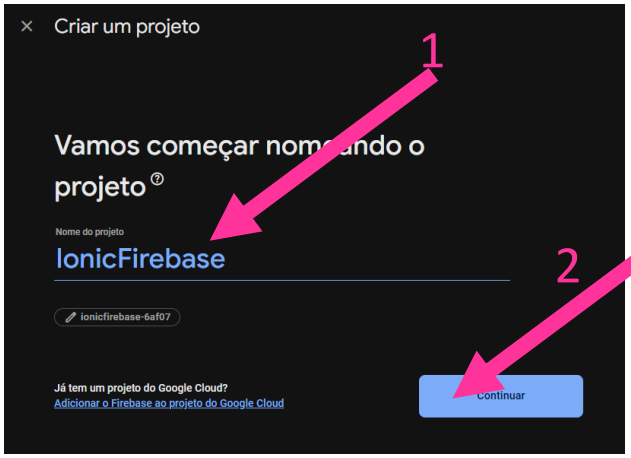
1. Faça login com uma conta Google válida e clique no link **GO TO CONSOLE**



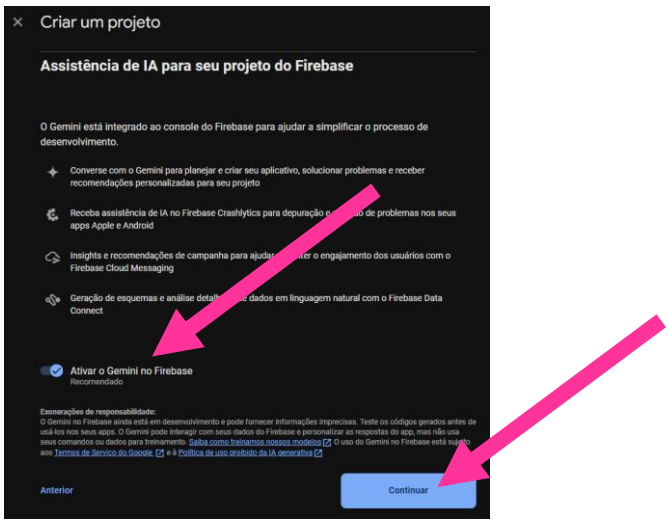
2. Clique em **CRIAR UM PROJETO DO FIREBASE**



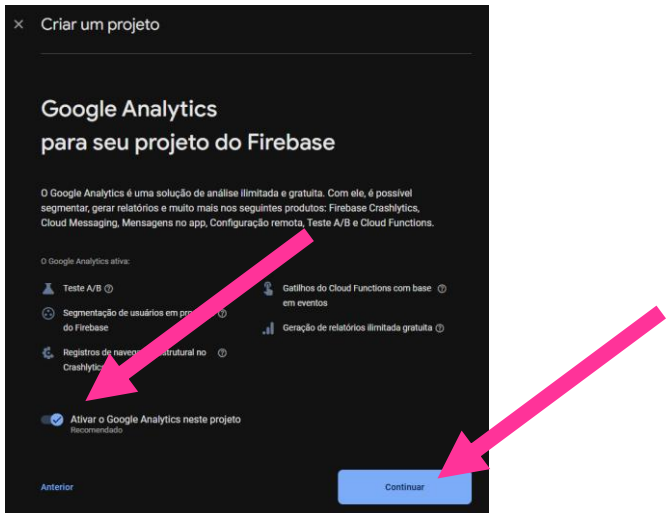
3. Dê um nome ao projeto e clique em **CONTINUAR**



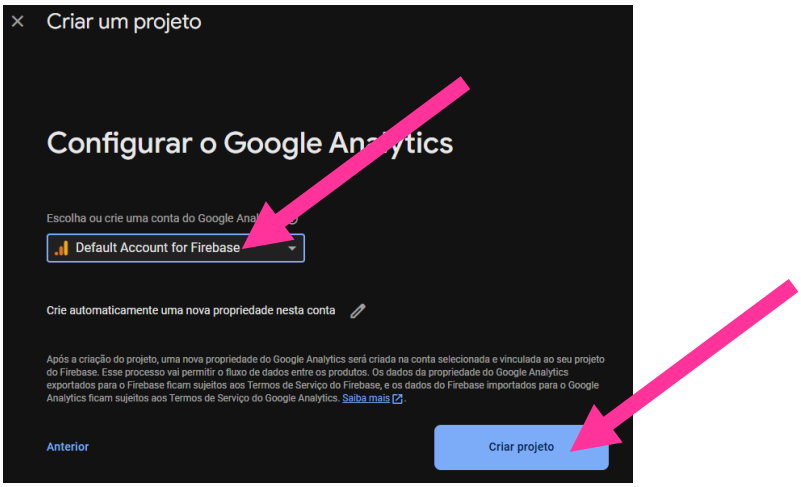
4. Permita o uso de IA no seu projeto. Pode ser útil no futuro, para uma integração com o Gemini



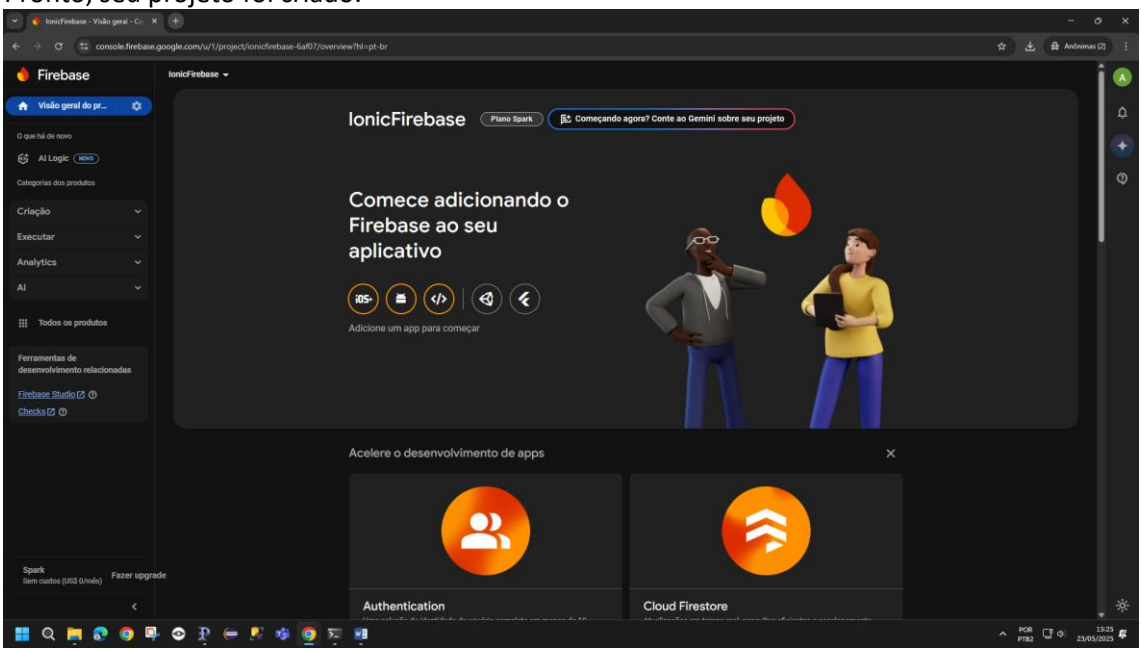
5. Deixe ativado o Google Analitics. O **Google Analytics para Firebase** é uma solução de análise de aplicativos gratuita que fornece insights detalhados sobre o uso de apps e o engajamento do usuário. Ele é a ferramenta de análise central do Firebase, uma plataforma de desenvolvimento de aplicativos móveis e web do Google.



6. Selecione **DEFAULT ACCOUNT FOR FIREBASE**. Isso criará um projeto padrão. As configurações deste projeto podem ser modificadas posteriormente



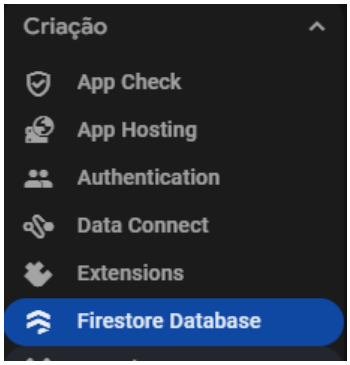
7. Pronto, seu projeto foi criado.



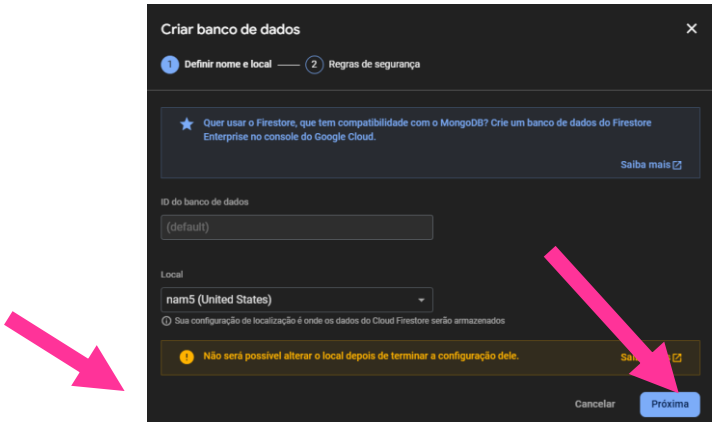
Passo 2: Criando o banco de dados Firestore Database

O **Cloud Firestore** é um banco de dados de documentos NoSQL flexível, escalável e de alto desempenho, desenvolvido pelo Google como parte da plataforma Firebase e Google Cloud. Ele é uma evolução do Realtime Database e foi projetado para armazenar, sincronizar e consultar dados para aplicativos móveis, web e de servidor em escala global.

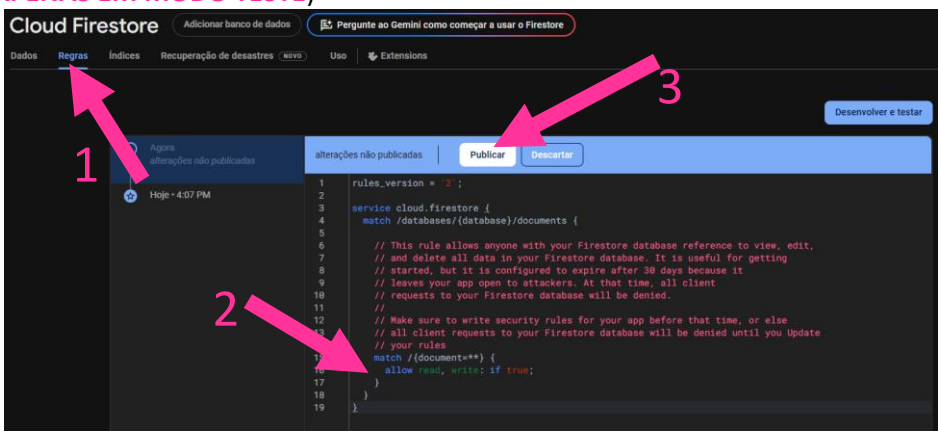
1. No menu lateral (lado esquerdo) selecione **CRIAÇÃO FIRESTORE DATABASE** em seguida clique em CRIAR BANCO DE DADOS



2. Não altere nada na tela seguinte e depois selecione **MODO TESTE**



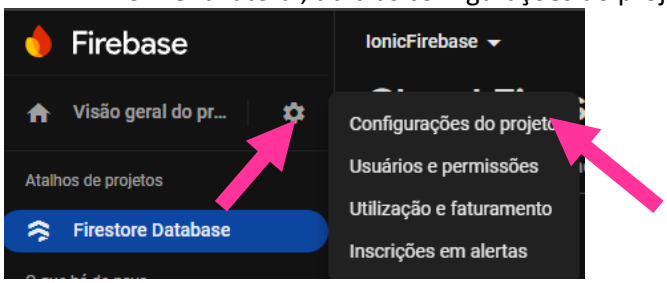
3. Com o banco criado, vamos alterar algumas políticas de segurança (**RECOMENDÁVEL APENAS EM MODO TESTE**)

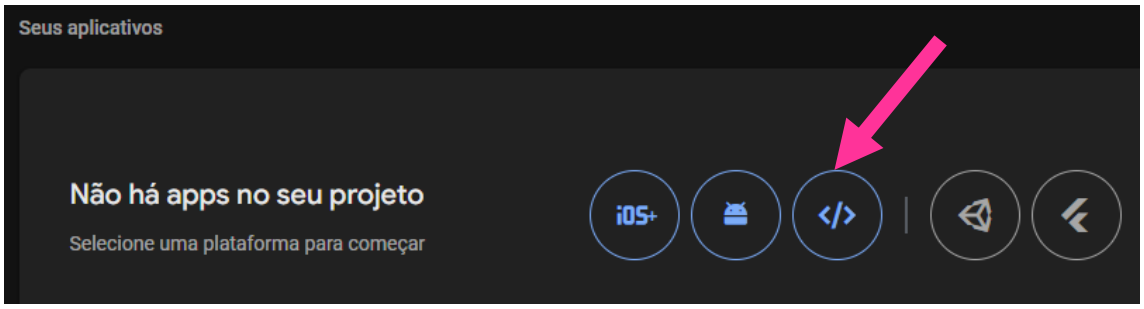


Passo 3: Criar o projeto WEB

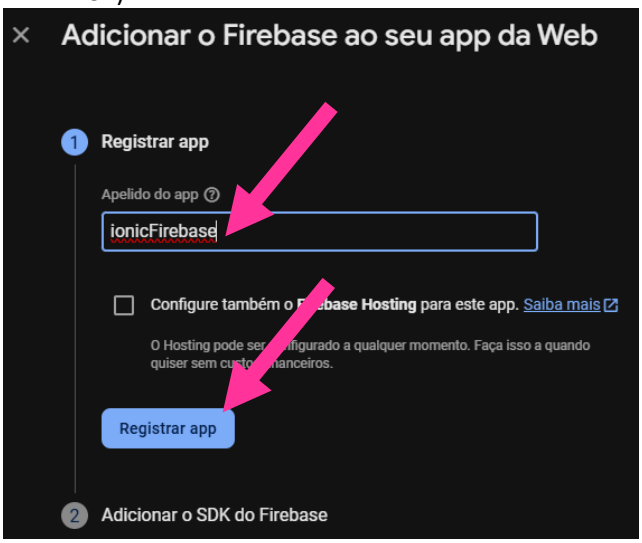
Agora vamos criar o projeto WEB para obter as credenciais que serão utilizadas no projeto IONIC.

1. No menu lateral, abra as configurações do projeto





2. Identifique o projeto (não há necessidade de usar o mesmo nome para o projeto FIREBASE)



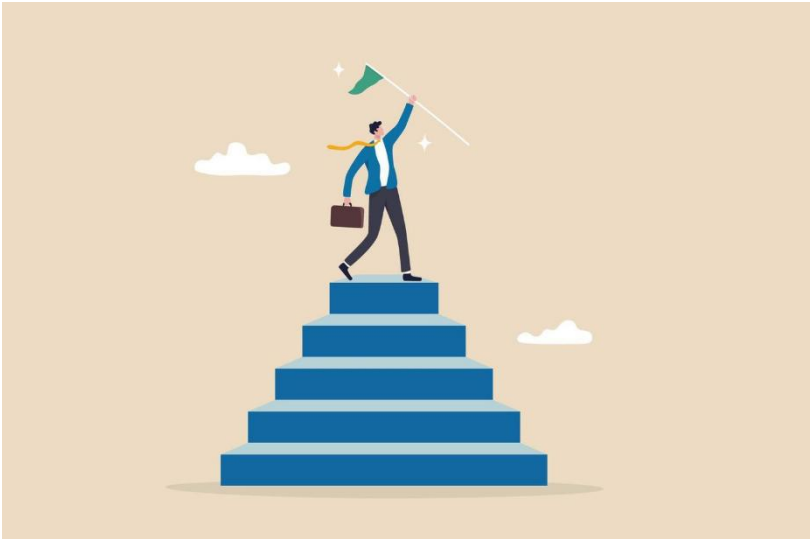
3. Guarde o trecho de código em destaque. Ele será usado no IONIC. Estas são as credenciais de acesso ao projeto FIREBASE. **JAMAIS COMPARTILHE ESTAS INFORMAÇÕES!**

```
// Import the functions you need from the SDKs you need
import { initializeApp } from "firebase/app";
import { getAnalytics } from "firebase/analytics";
// TODO: Add SDKs for Firebase products that you want to use
// https://firebase.google.com/docs/web/setup#available-libraries

// Your web app's Firebase configuration
// For Firebase JS SDK v7.20.0 and later, measurementId is optional
const firebaseConfig = {
  apiKey: "AIz[redacted]PQ",
  authDomain: "ion[redacted].com",
  projectId: "i[redacted]07",
  storageBucket: "i[redacted]ge.app",
  messagingSenderId: "96[redacted]5",
  appId: "1:96793[redacted]bfb56",
  measurementId: "G[redacted]LS"
};

// Initialize Firebase
const app = initializeApp(firebaseConfig);
const analytics = getAnalytics(app);
```

Com isso concluímos a configuração do **Firebase** e **do Firestore Database**. Agora vamos para o projeto IONIC



Passo 4: Criar o App Ionic

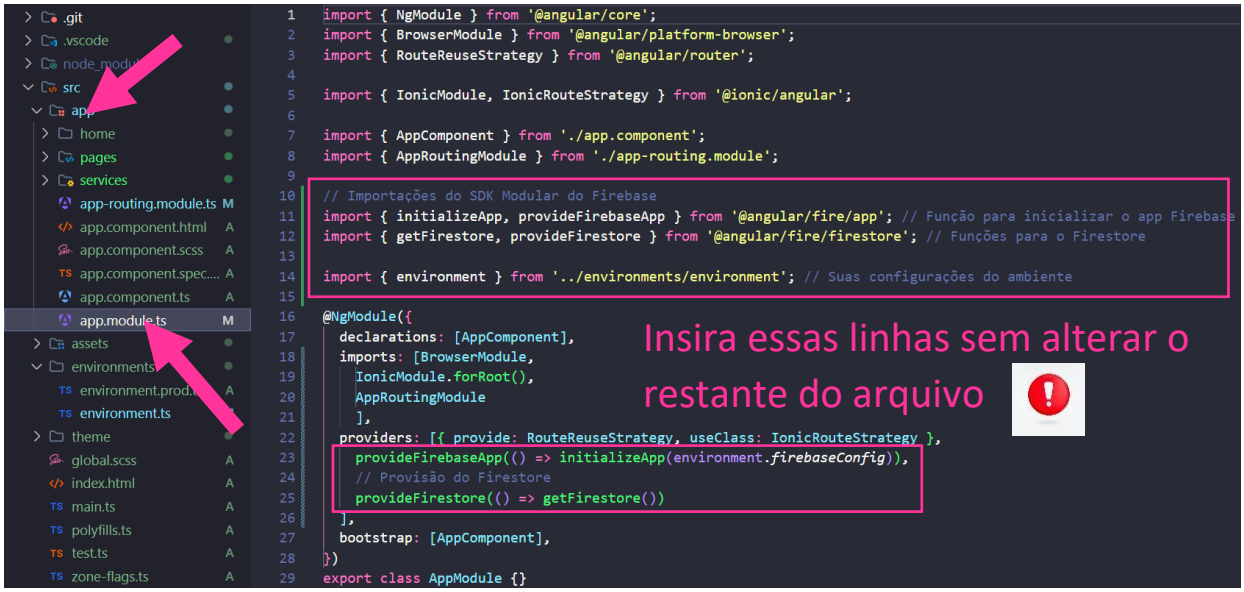
Crie um projeto **IONIC**, usando o comando **ionic start** no console. Lembre-se que a framework é **ANGULAR** que vamos usar o **NgModules**.
Após a criado, **entre na pasta do projeto** e adicione os módulos do firebase, digitando o seguinte comando no console:

```
npm install @angular/fire firebase
```

Abra o Visual Studio Code para iniciar o código.

Inserindo as credenciais do Firebase no projeto:

Agora vamos alterar o arquivo APP.MODULE.TS.
ATENÇÃO: Não altere todo o arquivo, apenas as linhas em destaque



Passo 5 - Criação do Serviço

No console, dentro da pasta do projeto, digite:

```
ionic g service services/data
```

Código do arquivo data.service.ts

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

// Importações do SDK Modular do Firestore
import {
  Firestore, // O serviço principal do Firestore
  collection, // Para obter uma referência a uma coleção
  doc, // Para obter uma referência a um documento
  collectionData, // Para obter dados de uma coleção como Observable
  docData, // Para obter dados de um documento como Observable
  addDoc, // Para adicionar um novo documento
  updateDoc, // Para atualizar um documento
  deleteDoc, // Para deletar um documento
  query, // Para construir consultas (ex: ordenar)
  orderBy // Para ordenar resultados
} from '@angular/fire/firestore';
import { Observable } from 'rxjs';

// Interface para o nosso item
export interface Item {
  id?: string;
  name: string;
  description: string;
  createdAt?: number;
}

@Injectable({
  providedIn: 'root'
})
```

```
export class DataService {

  constructor(private firestore: Firestore){
  }

  // Retorna todos os itens
  getItems(): Observable<Item[]> {
    // Cria uma referência para a coleção 'items'
    const itemsCollectionRef = collection(this.firestore, 'items');
    // Cria uma query para ordenar por 'createdAt' em ordem decrescente
    const q = query(itemsCollectionRef, orderBy('createdAt', 'desc'));
    // Retorna os dados da coleção como um Observable, incluindo o ID do documento
    return collectionData(q, { idField: 'id' }) as Observable<Item[]>;
  }

  // Retorna um item específico pelo ID
  getItem(id: string): Observable<Item | undefined> {
    // Cria uma referência para o documento específico
    const itemDocRef = doc(this.firestore, `items/${id}`);
    // Retorna os dados do documento como um Observable, incluindo o ID do documento
    return docData(itemDocRef, { idField: 'id' }) as Observable<Item | undefined>;
  }

  // Adiciona um novo item
  addItem(item: Item) {
    const itemsCollectionRef = collection(this.firestore, 'items');
    // Adiciona um novo documento à coleção
    return addDoc(itemsCollectionRef, { ...item, createdAt: Date.now() });
  }

  // Atualiza um item existente
  updateItem(item: Item) {
    // Cria uma referência para o documento específico
    const itemDocRef = doc(this.firestore, `items/${item.id}`);
    // Atualiza o documento
    return updateDoc(itemDocRef, { name: item.name, description: item.description });
  }

  // Deleta um item pelo ID
  deleteItem(id: string) {
    // Cria uma referência para o documento específico
    const itemDocRef = doc(this.firestore, `items/${id}`);
    // Deleta o documento
    return deleteDoc(itemDocRef);
  }
}
```

Passo 6 – Home.page.html e home.page.ts

```
<ion-header>
  <ion-toolbar color="primary">
    <ion-title>
      Meus Itens
    </ion-title>
    <ion-buttons slot="end">
      <ion-button (click)="addItem()">
        <ion-icon name="add"></ion-icon>
      </ion-button>
    </ion-buttons>
  </ion-toolbar>
</ion-header>

<ion-content class="ion-padding">
  <ion-list>
    <ion-item *ngIf="items.length === 0">
      <ion-label>Nenhum item cadastrado ainda.</ion-label>
    </ion-item>

    <ion-item-sliding *ngFor="let item of items">
      <ion-item lines="full" button (click)="editItem(item)">
        <ion-label>
          <h2>{{ item.name }}</h2>
          <p>{{ item.description }}</p>
        </ion-label>
      </ion-item>

      <ion-item-options side="end">
        <!--A exclamação aqui indica que o item.id jamais será nulo" -->
        <ion-item-option color="danger" (click)="deleteItem(item.id!)">
          <ion-icon slot="icon-only" name="trash"></ion-icon>
        </ion-item-option>
      </ion-item-options>
    </ion-item-sliding>
  </ion-list>
</ion-content>
```

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

import { DataService, Item } from '../services/data.service';
import { Router } from '@angular/router';
import { AlertController } from '@ionic/angular';

@Component({
  selector: 'app-home',
  templateUrl: 'home.page.html',
  styleUrls: ['home.page.scss'],
  standalone: false,
})
export class HomePage {

  items: Item[] = [];

  constructor(
    private dataService: DataService,
    private router: Router,
    private alertController: AlertController
  ) {}

  ngOnInit() {
    this.dataService.getItems().subscribe(res => {
      this.items = res;
    });
  }

  addItem() {
    this.router.navigateByUrl('/item-detail');
  }
```

Não esquecer os imports

```
32
33    editItem(item: Item) {
34      this.router.navigateByUrl(`/item-detail/${item.id}`);
35    }
36
37    async deleteItem(id: string) {
38      const alert = await this.alertController.create({
39        header: 'Confirmar exclusão',
40        message: 'Tem certeza que deseja excluir este item?',
41        buttons: [
42          {
43            text: 'Cancelar',
44            role: 'cancel',
45            cssClass: 'secondary',
46          },
47          {
48            text: 'Excluir',
49            handler: () => {
50              this.dataService.deleteItem(id);
51            },
52          },
53        ],
54      });
55      await alert.present();
56    }
57  }
```

Passo 7: item-detail.page.html e item.detail.page.ts

No console, dentro da pasta do projeto, digite:

```
ionic g page pages/page-detail
```

```
<ion-header>
  <ion-toolbar color="primary">
    <ion-buttons slot="start">
      <ion-back-button defaultHref="/home"></ion-back-button>
    </ion-buttons>
    <ion-title>{{ isNewItem ? 'Adicionar Novo Item' : 'Editar Item' }}</ion-title>
  </ion-toolbar>
</ion-header>

<ion-content class="ion-padding">
  <form #form="ngForm" (ngSubmit)="saveItem()">
    <ion-item>
      <ion-label position="floating">Nome</ion-label>
      <ion-input type="text" [(ngModel)]="item.name" name="name" required></ion-input>
    </ion-item>

    <ion-item>
      <ion-label position="floating">Descrição</ion-label>
      <ion-textarea rows="4" [(ngModel)]="item.description" name="description"></ion-textarea>
    </ion-item>

    <ion-button expand="full" type="submit" [disabled]="!form.valid" class="ion-margin-top">
      Salvar
    </ion-button>
  </form>
</ion-content>
```

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

import { ActivatedRoute, Router } from '@angular/router';
import { DataService, Item } from '../../services/data.service';
import { LoadingController, ToastController } from '@ionic/angular';


@Component({
  selector: 'app-item-detail',
  templateUrl: './item-detail.page.html',
  styleUrls: ['./item-detail.page.scss'],
  standalone:false,
})
export class ItemDetailPage implements OnInit {

  // Declara uma propriedade 'item' do tipo Item, inicializada com valores vazios.
  // Será usada para vincular os dados do formulário (nome e descrição).
  item: Item = {
    name: '',
    description: ''
  };
  // Declara 'itemId' que pode ser uma string ou null. Armazenará o ID do item se estivermos editando.
  itemId: string | null = null;
  // Uma flag booleana para verificar se estamos criando um novo item (true) ou editando um existente (false).
  isNewItem = true;

    constructor(private route: ActivatedRoute,private dataService: DataService,private router: Router,
                private loadingController: LoadingController, private toastController: ToastController) { }

  ngOnInit() {
    this.itemId = this.route.snapshot.paramMap.get('id');
    // Verifica se um ID de item foi encontrado na URL.
    if (this.itemId) {
      // Se um ID existe, significa que estamos editando um item existente.
      this.isNewItem = false;
      // Chama o método para carregar os dados do item.
      this.loadItem();
    }
  }
  async loadItem() {
    // Cria um controle de carregamento (loading spinner) com uma mensagem.
    const loading = await this.loadingController.create({
      message: 'Carregando item...'
    });
    // Apresenta o loading spinner na tela.
    await loading.present();
    // Chama o método getItem do DataService para obter o item pelo ID.
    // O '!' (non-null assertion operator) informa ao TypeScript que itemId não será null aqui.
    this.dataService.getItem(this.itemId!).subscribe(res => {
      // Dispensa o loading spinner assim que a resposta for recebida.
      loading.dismiss();
      // Verifica se o item foi encontrado.
      if (res) {
        // Se encontrado, atribui os dados retornados à propriedade 'item' do componente.
        this.item = res;
      } else {
        // Se o item não for encontrado, exibe um toast de erro.
        this.presentToast('Item não encontrado!', 'danger');
        // Redireciona o usuário de volta para a página inicial.
        this.router.navigateByUrl('/home');
      }
    }, err => { // Trata erros na requisição.
      // Dispensa o loading spinner em caso de erro.
      loading.dismiss();
      // Exibe um toast de erro genérico.
      this.presentToast('Erro ao carregar item.', 'danger');
      // Redireciona o usuário de volta para a página inicial.
      this.router.navigateByUrl('/home');
    });
  }
```

```
  async saveItem() {
    // Cria um controle de carregamento (loading spinner) com uma mensagem.
    const loading = await this.loadingController.create({
      message: 'Salvando item...'
    });
    // Apresenta o loading spinner na tela.
    await loading.present();

    // Verifica se a flag 'isNewItem' é verdadeira, indicando que é um novo item.
    if (this.isNewItem) {
      // Se for um novo item, chama o método 'addItem' do DataService.
      this.dataService.addItem(this.item).then(() => {
        // Dispensa o loading spinner após o sucesso.
        loading.dismiss();
        // Exibe um toast de sucesso.
        this.presentToast('Item adicionado com sucesso!', 'success');
        // Redireciona o usuário para a página inicial.
        this.router.navigateByUrl('/home');
      }, err => { // Trata erros ao adicionar.
        // Dispensa o loading spinner em caso de erro.
        loading.dismiss();
        // Exibe um toast de erro.
        this.presentToast('Erro ao adicionar item.', 'danger');
      });
    } else { // Se 'isNewItem' for falsa, estamos atualizando um item existente.
      // Chama o método 'updateItem' do DataService.
      this.dataService.updateItem(this.item).then(() => {
        // Dispensa o loading spinner após o sucesso.
        loading.dismiss();
        // Exibe um toast de sucesso.
        this.presentToast('Item atualizado com sucesso!', 'success');
        // Redireciona o usuário para a página inicial.
        this.router.navigateByUrl('/home');
      }, err => { // Trata erros ao atualizar.
        // Dispensa o loading spinner em caso de erro.
        loading.dismiss();
        // Exibe um toast de erro.
        this.presentToast('Erro ao atualizar item.', 'danger');
      });
    }
  }

  async presentToast(message: string, color: string = 'primary') {
    // Cria um controle de toast com a mensagem, duração e cor.
    const toast = await this.toastController.create({
      message: message,
      duration: 2000, // O toast desaparecerá após 2 segundos.
      color: color // A cor do toast (pode ser primary, secondary, success, danger, etc.).
    });
    // Apresenta o toast na tela.
    toast.present();
  }
}
```

Passo 7: Arquivo app-routing.module.ts

Insira no arquivo app-routing.module.ts uma nova rota para abrir a página de detalhes do item com passagem de parâmetro:

```
import { NgModule } from '@angular/core';
import { PreloadAllModules, RouterModule, Routes } from '@angular/router';

const routes: Routes = [
  {
    path: 'home',
    loadChildren: () => import('./home/home.module').then( m => m.HomePageModule)
  },
  {
    path: '',
    redirectTo: 'home',
    pathMatch: 'full'
  },
  {
    path: 'item-detail',
    loadChildren: () => import('./pages/item-detail/item-detail.module').then( m => m.ItemDetailPageModule)
  },
  {
    path: 'item-detail/:id', // Rota para editar item existente
    loadChildren: () => import('./pages/item-detail/item-detail.module').then( m => m.ItemDetailPageModule)
  }
];

@NgModule({
  imports: [
    RouterModule.forRoot(routes, { preloadingStrategy: PreloadAllModules })
  ],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

Apenas acrescente a rota em destaque, sem alterar o restante do arquivo

Finalizamos. Agora é só testar o aplicativo (digitando ionic serve no console, dentro da pasta do projeto)!