Cloud Firestore: O Banco de Dados NoSQL Flexível e Escalável do Google

O Cloud Firestore é um banco de dados de nuvem NoSQL flexível e escalável, construído sobre a robusta infraestrutura do Google Cloud. Ele permite armazenar e sincronizar dados de forma eficiente para o desenvolvimento de aplicativos móveis, web e de servidor, oferecendo recursos poderosos e integração perfeita com o ecossistema Firebase e Google Cloud.

Por Que Escolher o Cloud Firestore?

Focado em Mobile e Web

Desenvolvido para aplicações modernas, garantindo sincronização em tempo real e suporte off-line.

Sincronização em Tempo Real

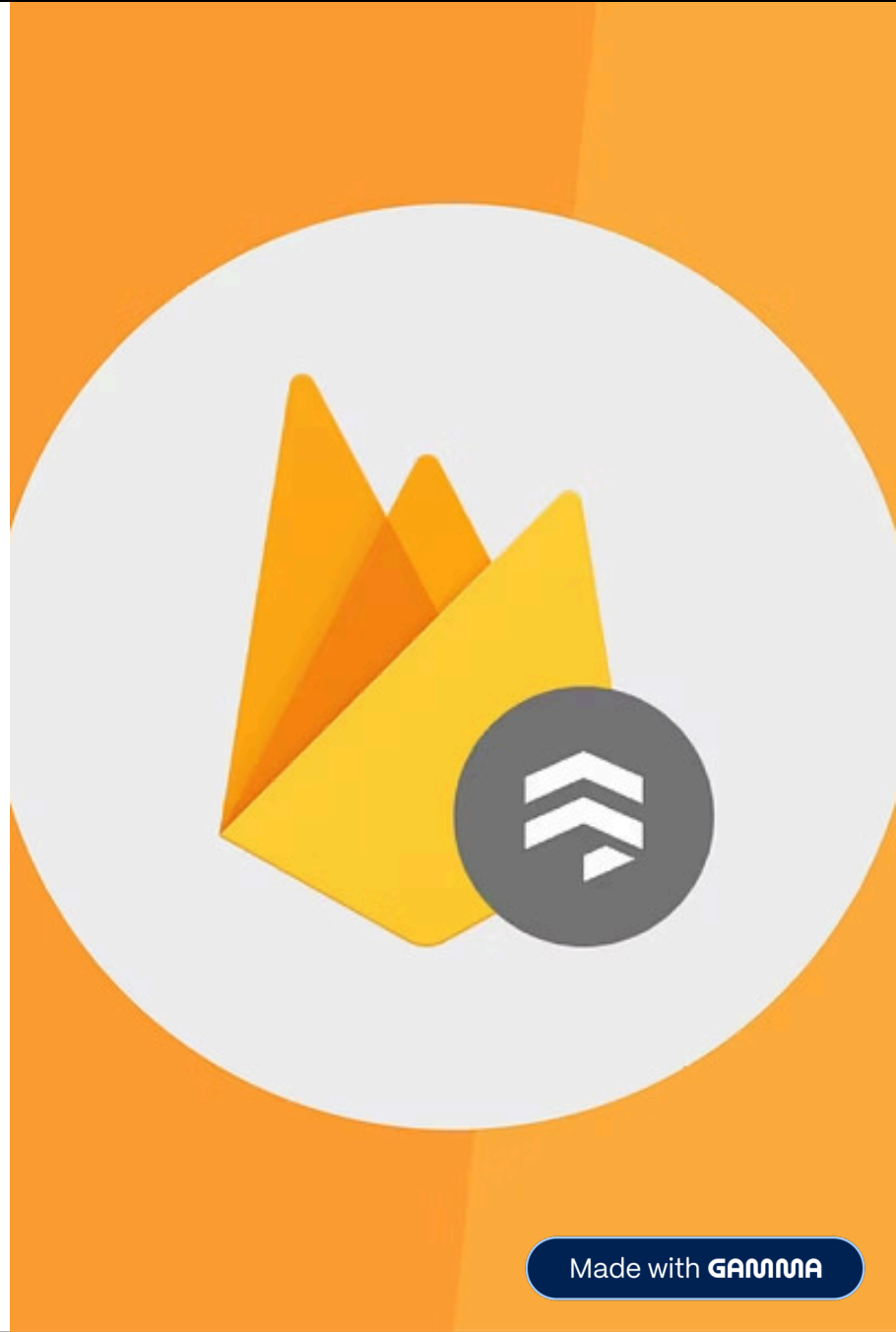
Mantém seus dados atualizados em todos os clientes conectados através de listeners em tempo real.

Suporte Off-line

Permite que aplicativos funcionem perfeitamente mesmo sem conexão, sincronizando as mudanças quando online.

Integração Total

Compatibilidade nativa com Firebase, Google Cloud e Cloud Functions para um ecossistema completo.

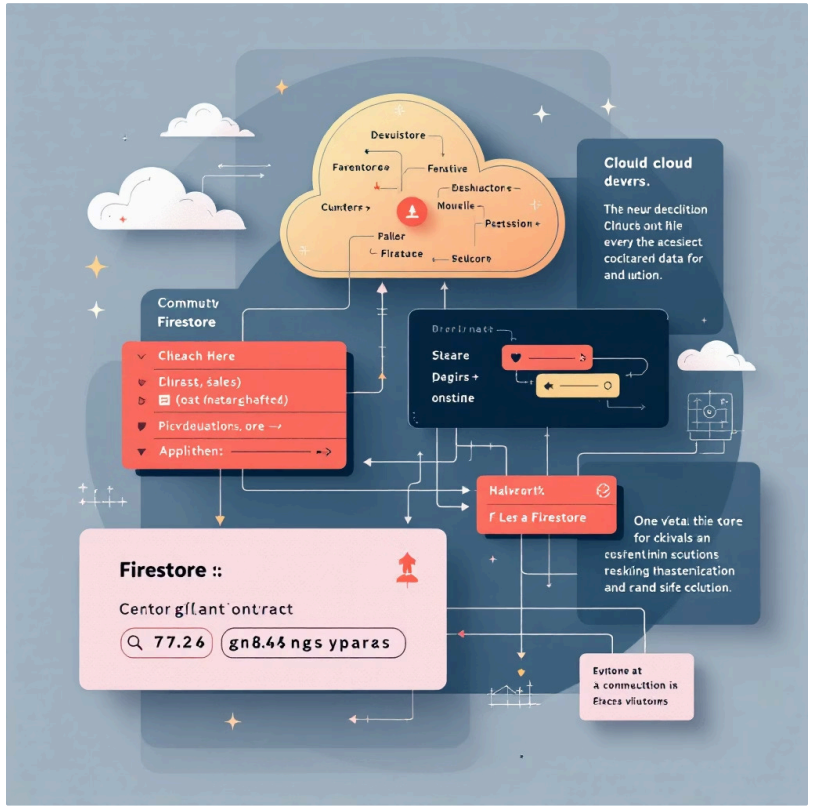


Modelo de Dados Flexível

O Cloud Firestore utiliza um modelo de dados NoSQL baseado em documentos, o que oferece uma flexibilidade incomparável para estruturar suas informações.

- **Documentos:** Unidades de dados que contêm campos mapeados para valores. Podem incluir objetos aninhados complexos e subcoleções.
- **Coleções:** Contêineres de documentos que ajudam a organizar os dados e facilitar consultas eficientes.
- **Subcoleções:** Permitem criar estruturas de dados hierárquicas e complexas que podem escalar à medida que seu banco de dados cresce.

Este modelo suporta diversos **tipos de dados**, desde strings e números simples até objetos JSON.



Consultas Expressivas e Poderosas

No Cloud Firestore, você pode ir além das simples recuperações de dados, utilizando consultas para extrair informações específicas e relevantes de forma eficiente.

Filtros em Cadeia

Combine múltiplos filtros para refinar seus resultados, buscando documentos que correspondam a critérios complexos.

Filtragem e Classificação

Adicione cláusulas de classificação e limites para ordenar e paginar seus dados, garantindo uma experiência de usuário otimizada.

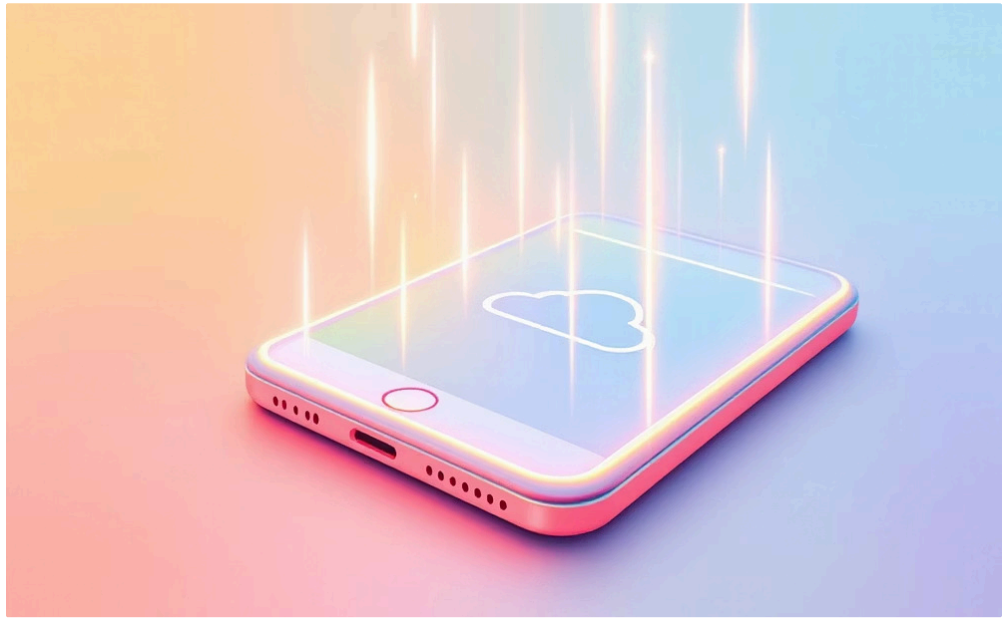
Consultas Superficiais

Recupere apenas os dados necessários no nível do documento, sem a sobrecarga de buscar coleções inteiras ou subcoleções aninhadas.

Atualizações em Tempo Real e Suporte Off-line

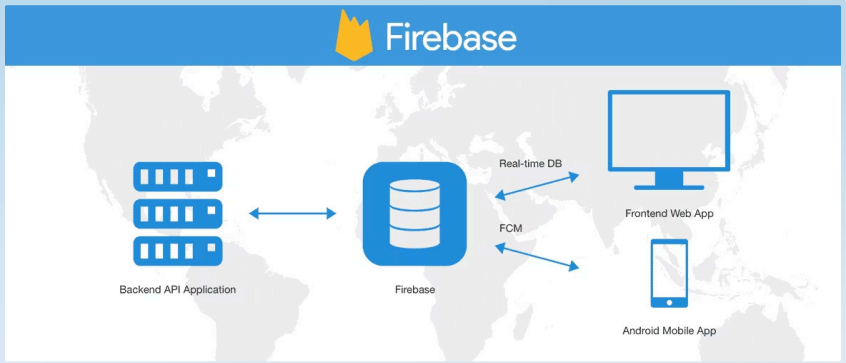
Sincronização Instantânea

Assim como o Realtime Database, o Cloud Firestore usa sincronização para manter os dados atualizados em qualquer dispositivo conectado, enviando apenas as alterações.



Experiência Off-line Robusta

O Cloud Firestore armazena em cache os dados ativos, permitindo que seu aplicativo funcione sem problemas, mesmo em ambientes sem conectividade. As mudanças locais são sincronizadas automaticamente quando a conexão é restabelecida.



Projetado para Escala Global

O Cloud Firestore herda a robustez e a confiabilidade da infraestrutura do Google Cloud, sendo projetado para suportar as maiores e mais exigentes cargas de trabalho do mundo.

01

Replicação Automática

Dados replicados automaticamente em várias regiões para alta disponibilidade e durabilidade.

02

Consistência Forte

Garantias de consistência que asseguram a integridade dos seus dados em todas as operações.

03

Operações Atômicas

Suporte a transações ACID (Atomicidade, Consistência, Isolamento, Durabilidade) e operações em lote atômicas.

04

Infraestrutura Google

Beneficie-se da mesma infraestrutura que impulsiona os serviços do Google, garantindo desempenho e escalabilidade.

Compatibilidade com MongoDB

Para empresas que já utilizam MongoDB, o Cloud Firestore oferece uma transição suave e integração eficaz.

> Na **edição Enterprise** do Cloud Firestore, é possível utilizar uma API compatível com MongoDB. Isso permite que você aproveite seu código, drivers, ferramentas e o vasto ecossistema de código aberto do MongoDB, simplificando a migração e o gerenciamento de dados.

Como o Cloud Firestore Funciona?

O Cloud Firestore é um banco de dados NoSQL hospedado na nuvem, acessível diretamente por SDKs nativos em diversas plataformas.

SDKs Nativos

Acesso direto para aplicativos Apple (iOS/macOS), Android e Web com SDKs otimizados.

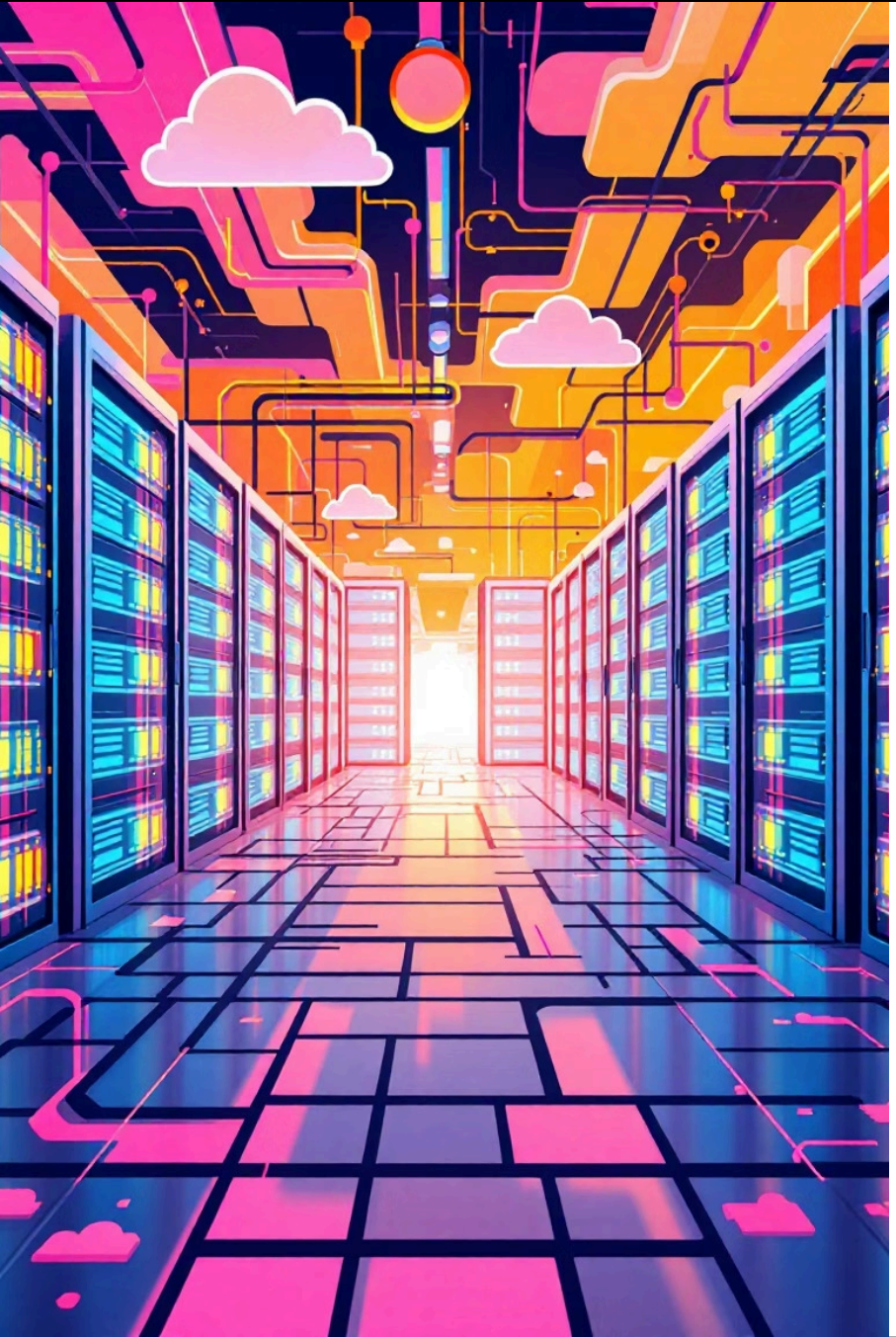
Multi-Linguagem

Disponível para Node.js, Java, Python, Unity, C++, Go e APIs REST/RPC para desenvolvimento de servidor.

Modelo de Documentos

Organiza dados em documentos e coleções, permitindo estruturas hierárquicas flexíveis.

A consulta expressiva e o suporte a listeners em tempo real garantem que seus dados estejam sempre atualizados e facilmente acessíveis, independentemente da complexidade da sua aplicação.



Segurança e Acesso

A segurança dos seus dados é uma prioridade máxima no Cloud Firestore, com opções robustas para controle de acesso.

Firebase Authentication

Integração perfeita com o Firebase Authentication para gerenciar a identidade dos usuários em aplicativos móveis e web.

Cloud Firestore Security Rules

Defina regras de segurança granulares para Android, plataformas da Apple e JavaScript para controlar quem pode ler e escrever seus dados.

Identity and Access Management (IAM)

Para ambientes de servidor, utilize o IAM do Google Cloud para gerenciar permissões de acesso de forma centralizada e segura.

Caminho para Implementação

Começar a usar o Cloud Firestore é um processo direto, projetado para acelerar o desenvolvimento da sua aplicação.

1

Integre os SDKs

Adicione rapidamente os clientes do Cloud Firestore via Gradle, CocoaPods ou script de inclusão.

2

Proteja seus Dados

Use as Regras de Segurança do Cloud Firestore ou o IAM para proteger suas informações de acordo com o ambiente.

3

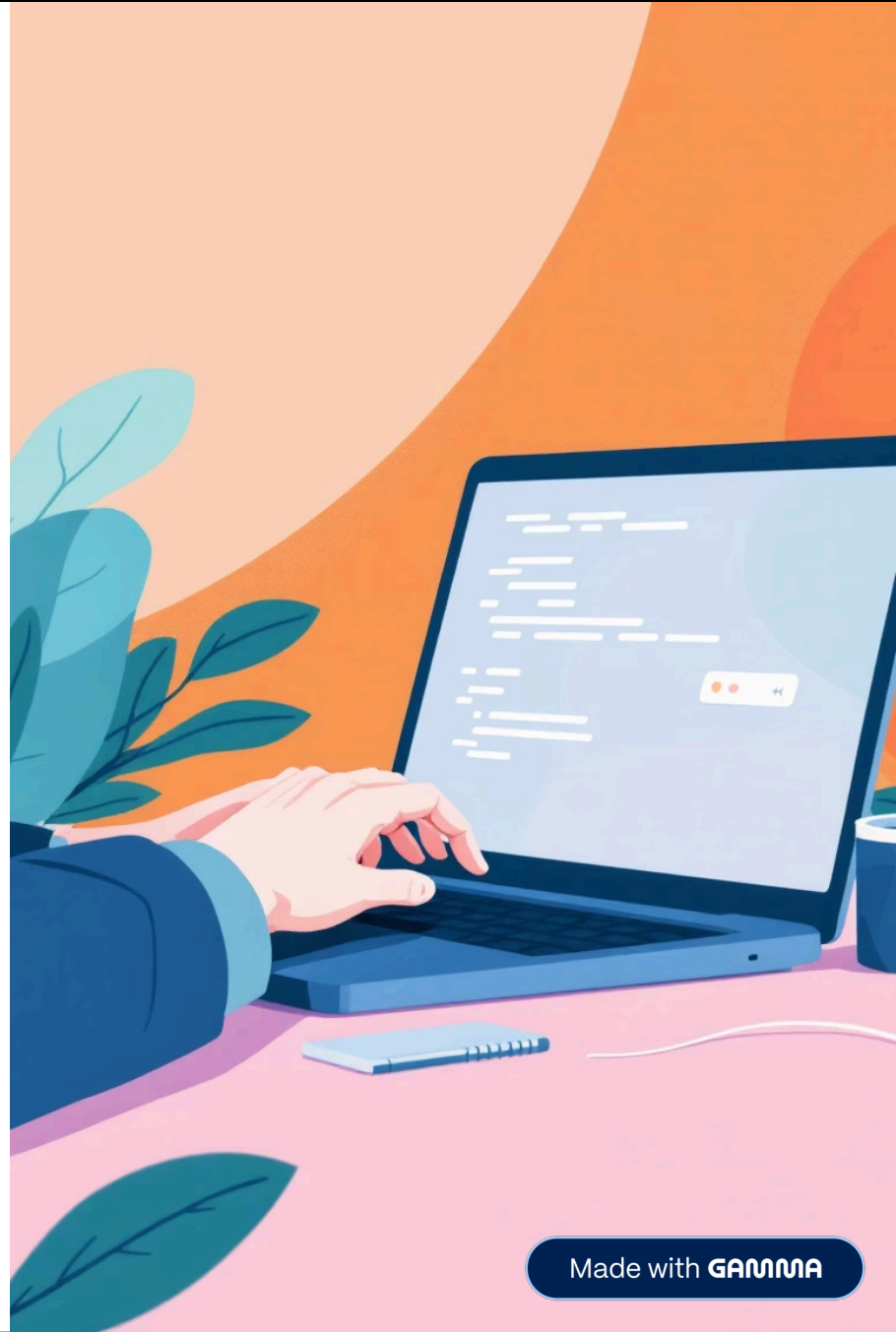
Adicione Dados

Crie documentos e coleções para começar a popular seu banco de dados com as informações da sua aplicação.

4

Receba Dados

Utilize consultas eficientes ou listeners em tempo real para recuperar e manter os dados atualizados em seus aplicativos.



Hora da Pratíca

Instalação das Dependências

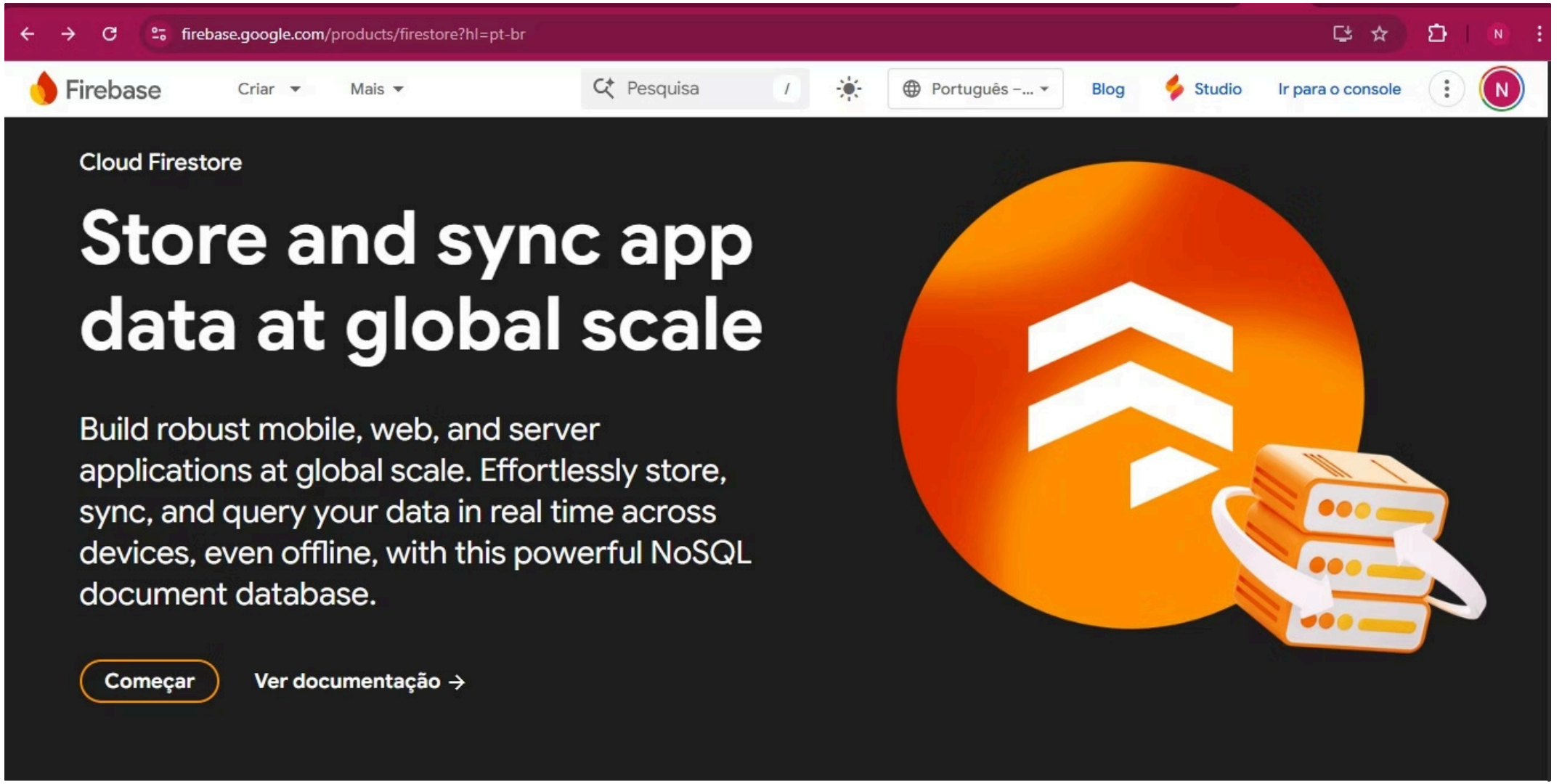
npm install -g firebase-tools

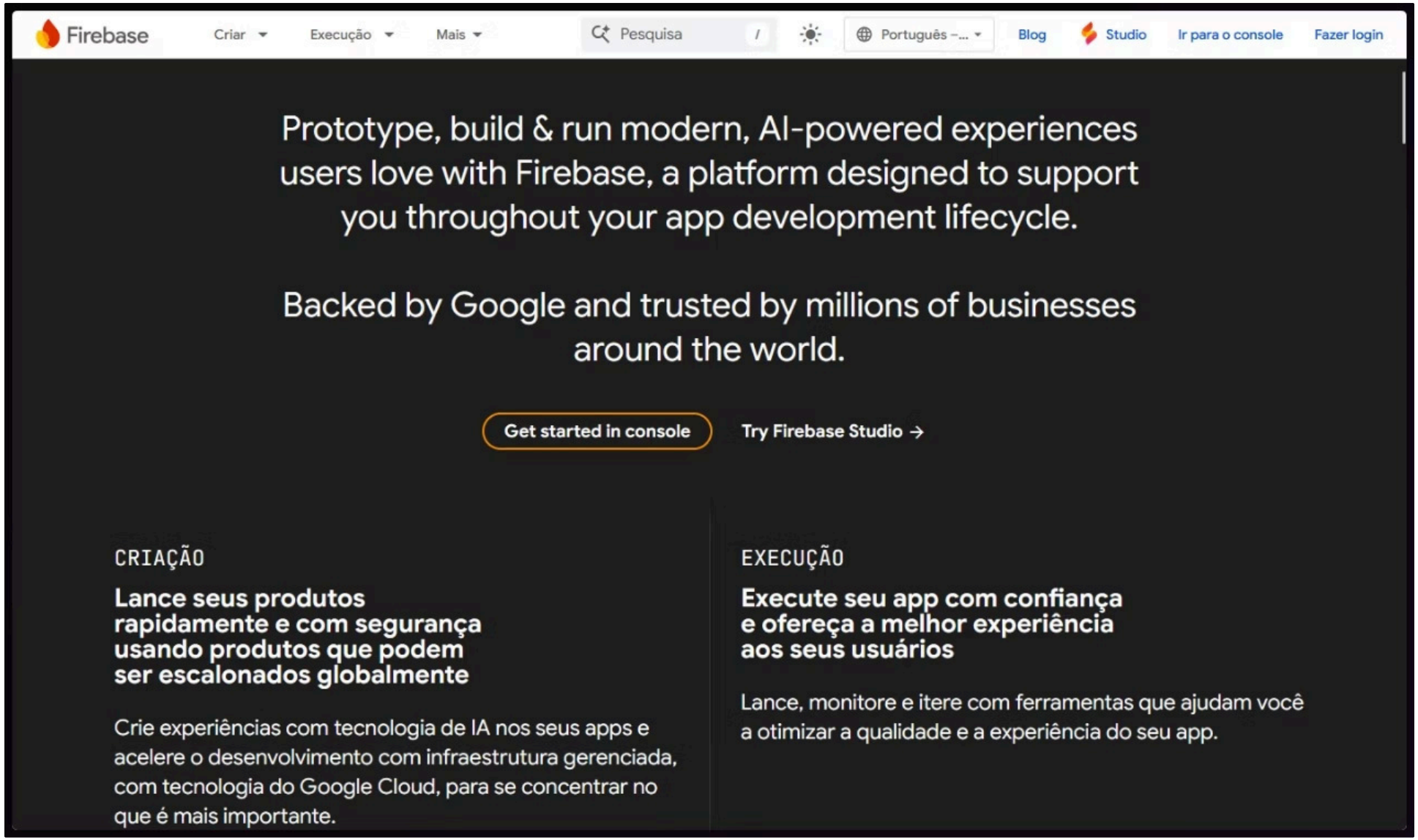
dart pub global activate flutterfire_cli

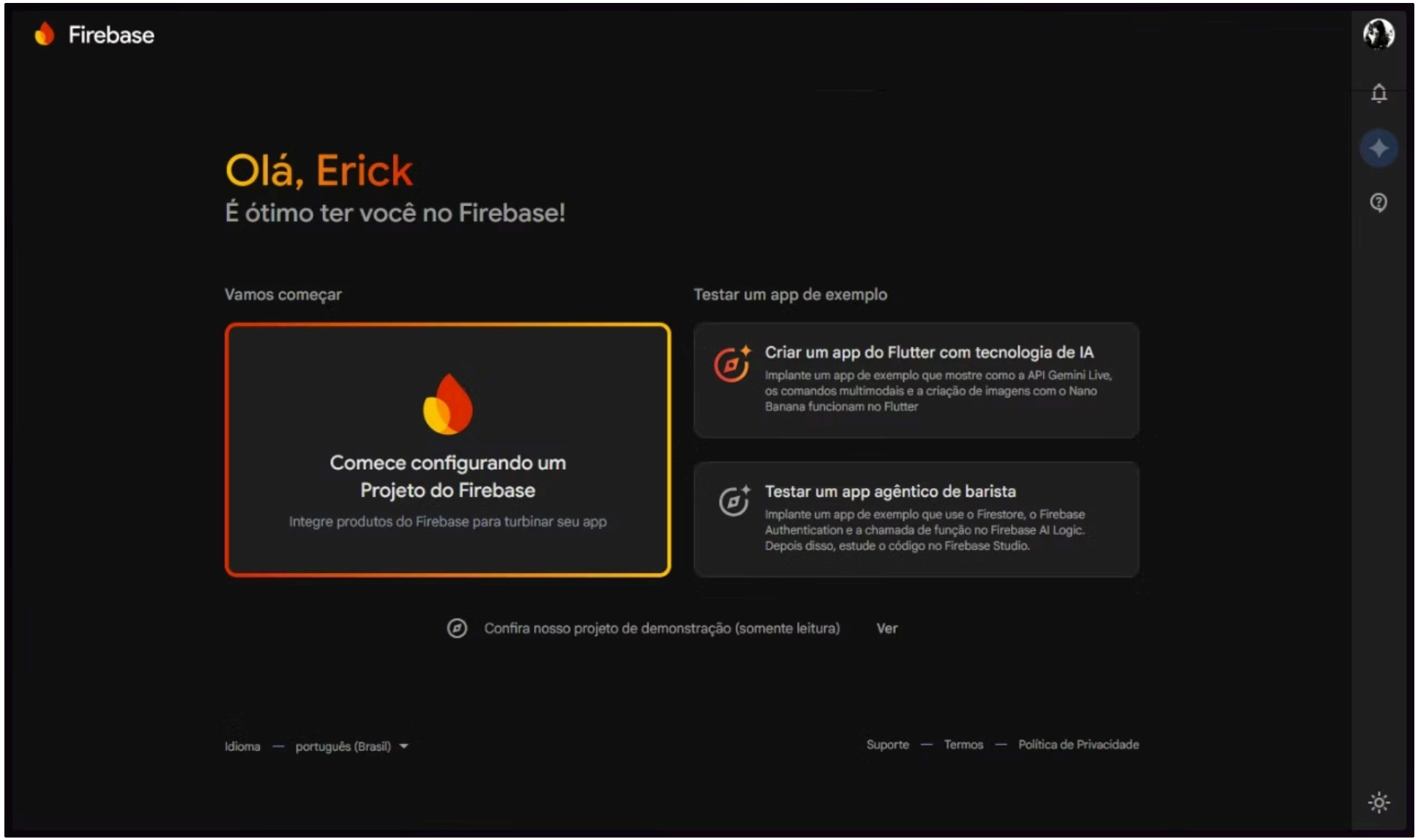
$env:PATH += ";$HOME\AppData\Local\Pub\Cache\bin"

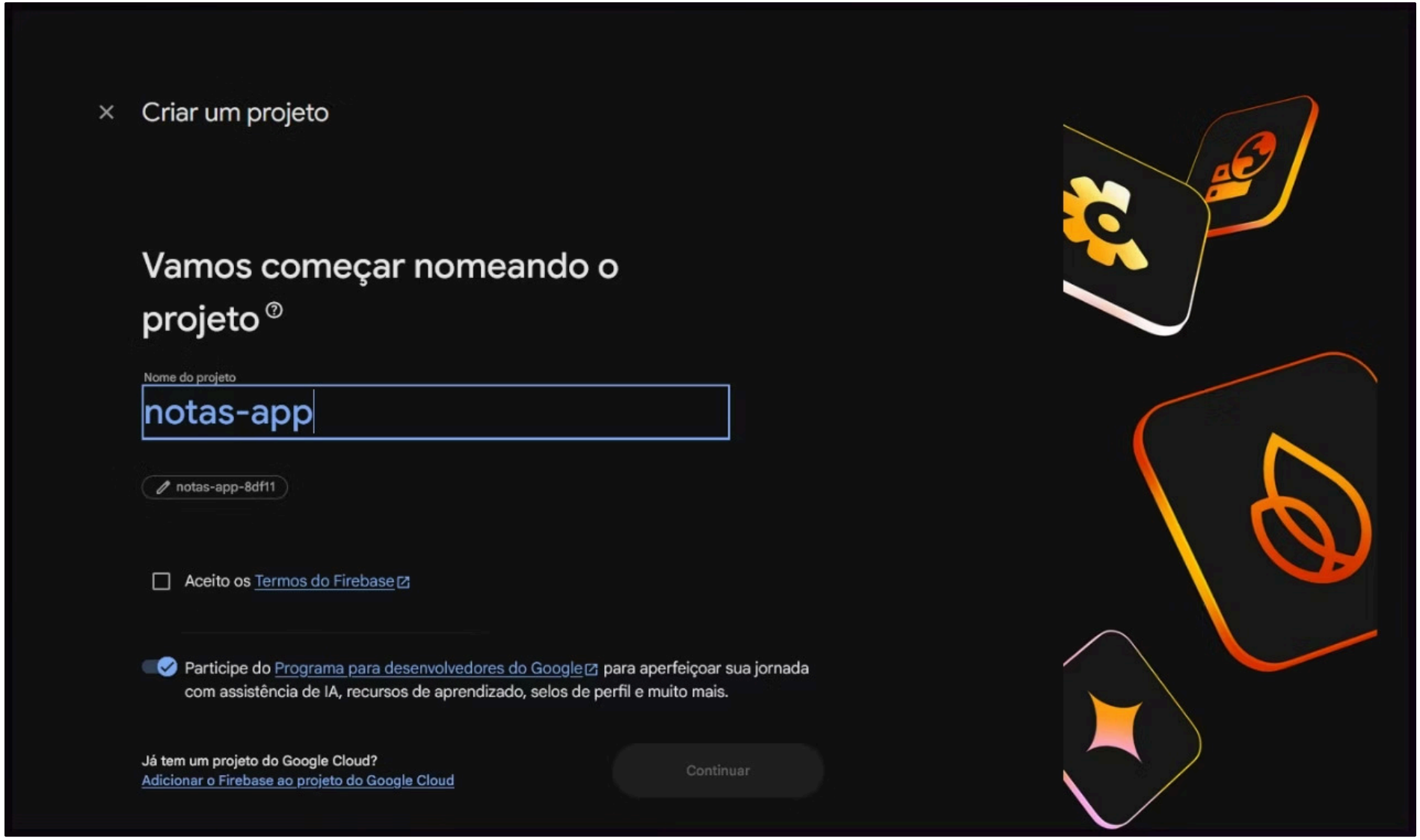
firebase login

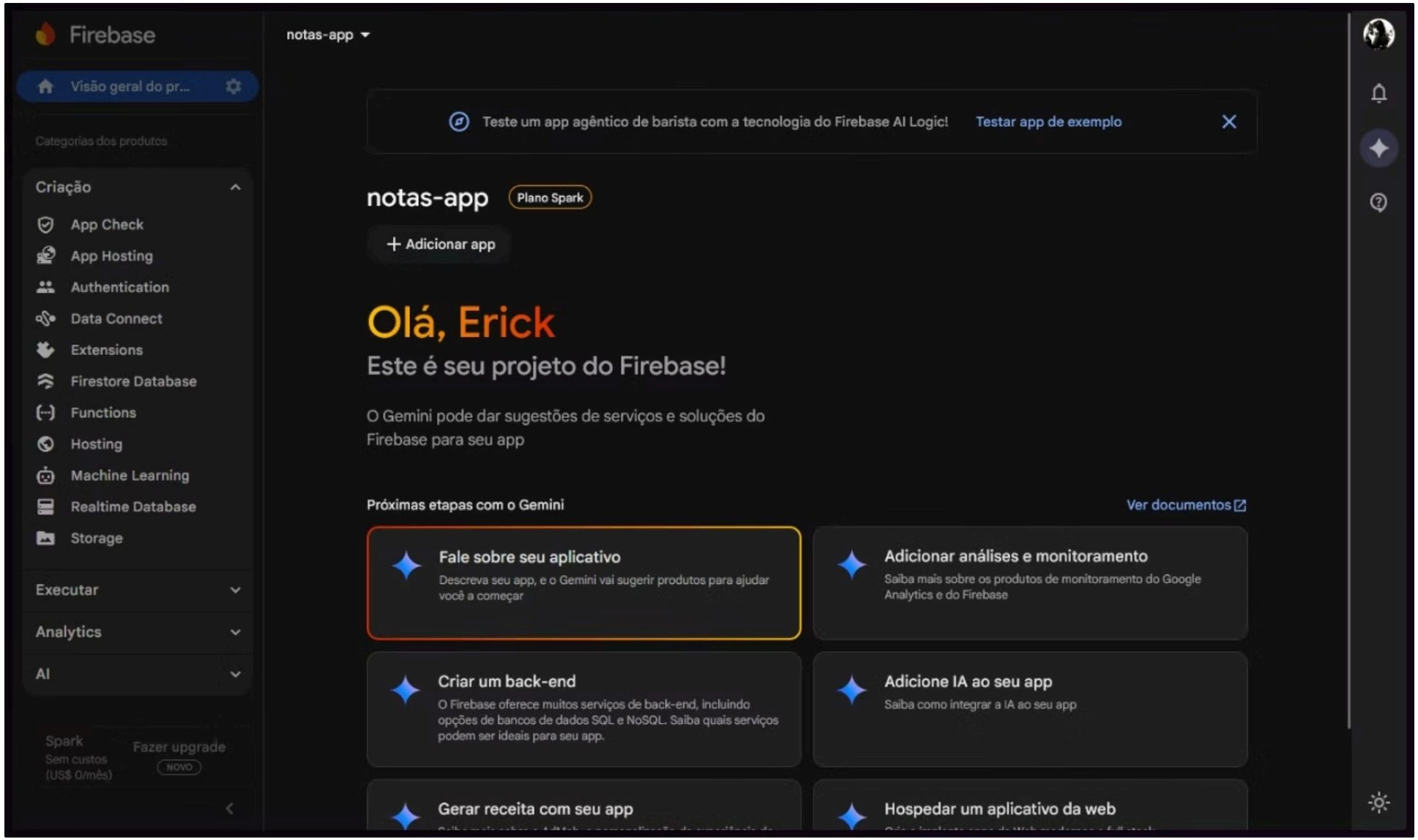
flutterfire configure

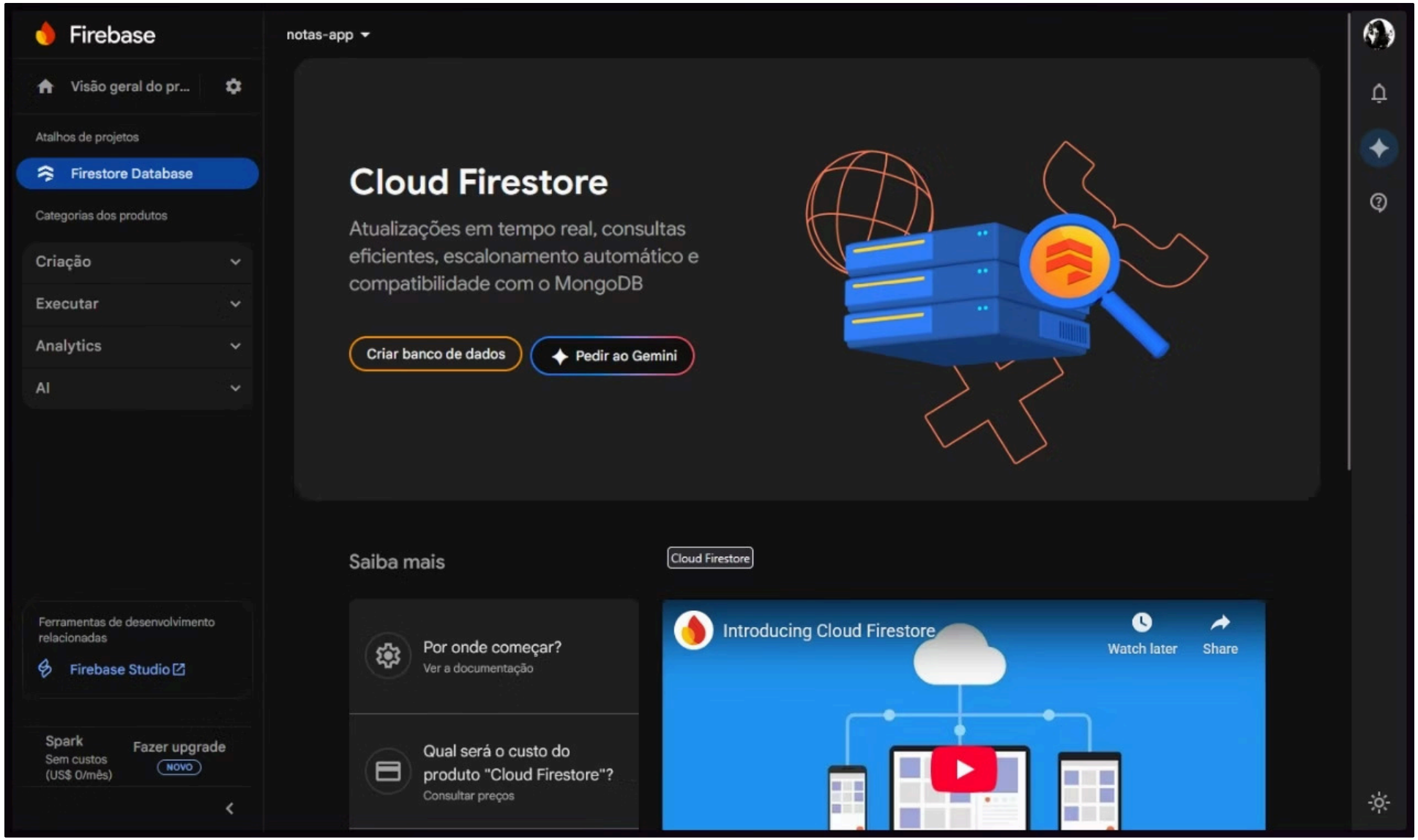


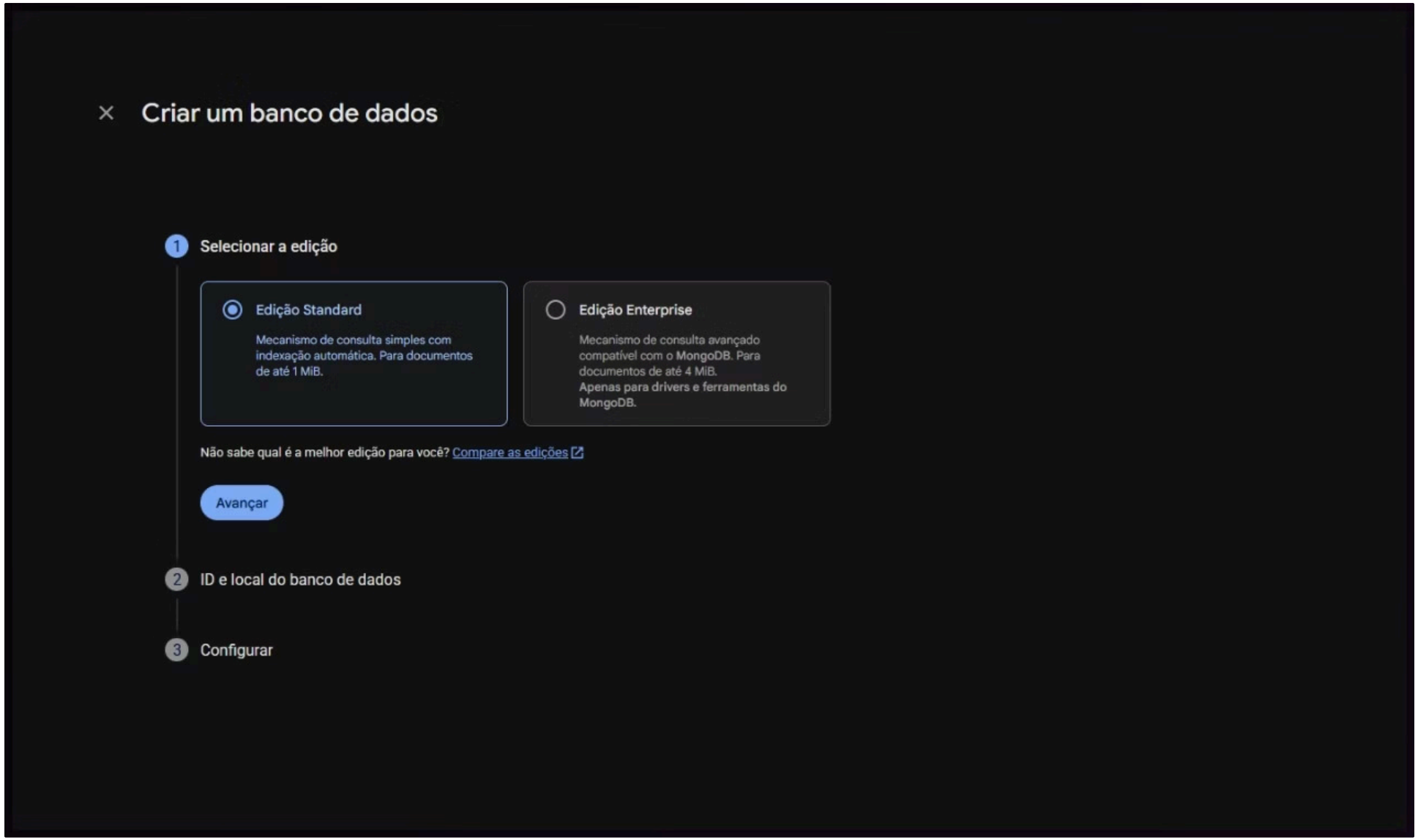












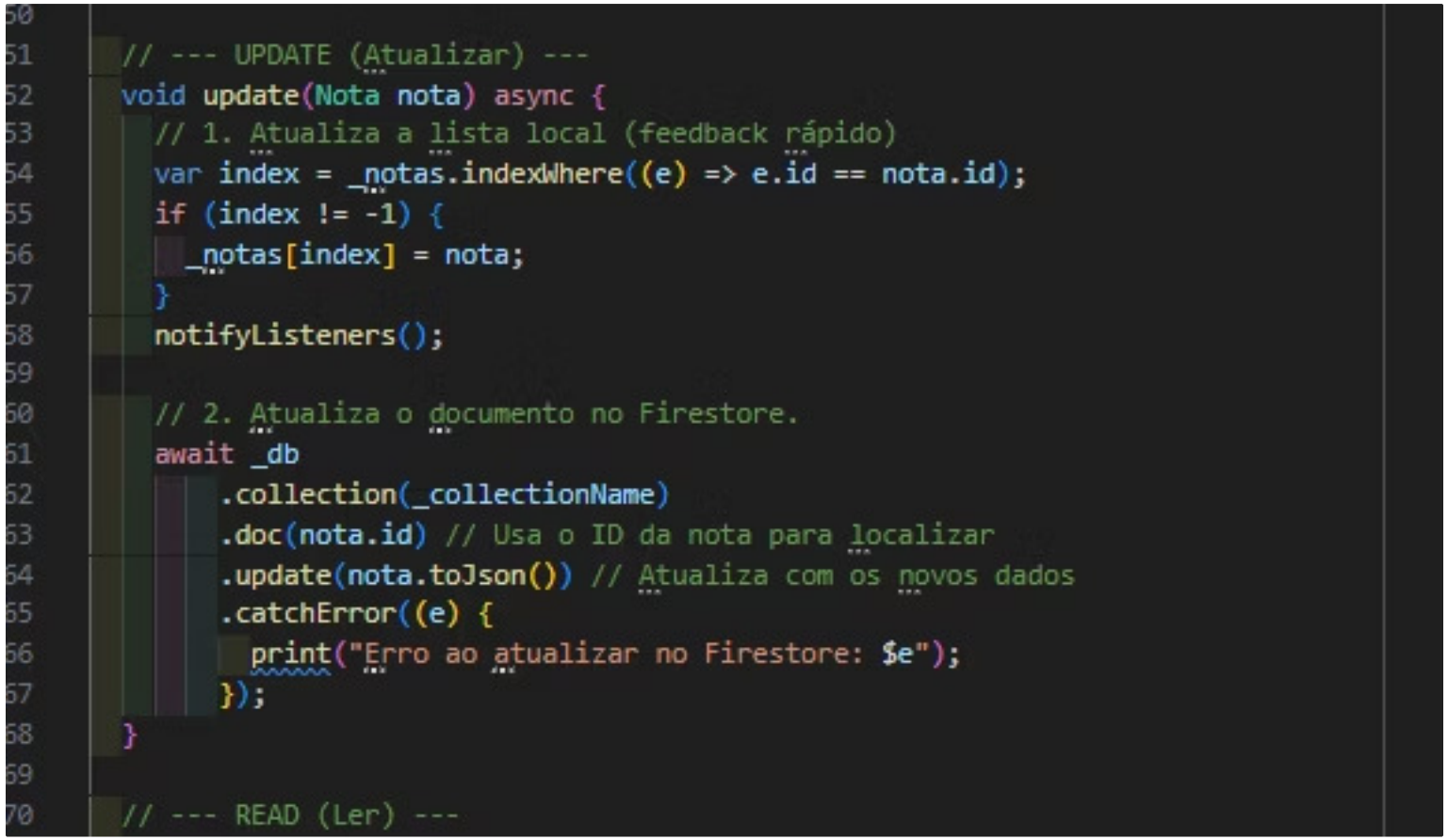
lib > main.dart > ...

```dart
import 'package:flutter/material.dart';
import 'package:firebase_core/firebase_core.dart';
import 'firebase_options.dart'; // NOVO IMPORT!
import 'views/app.dart';

Run | Debug | Profile
void main() async {
  WidgetsFlutterBinding.ensureInitialized();

  // Inicialização correta para todas as plataformas (incluindo Web)
  await Firebase.initializeApp(
    options: DefaultFirebaseOptions.currentPlatform,
  );

  runApp(const NotasApp());
}
```

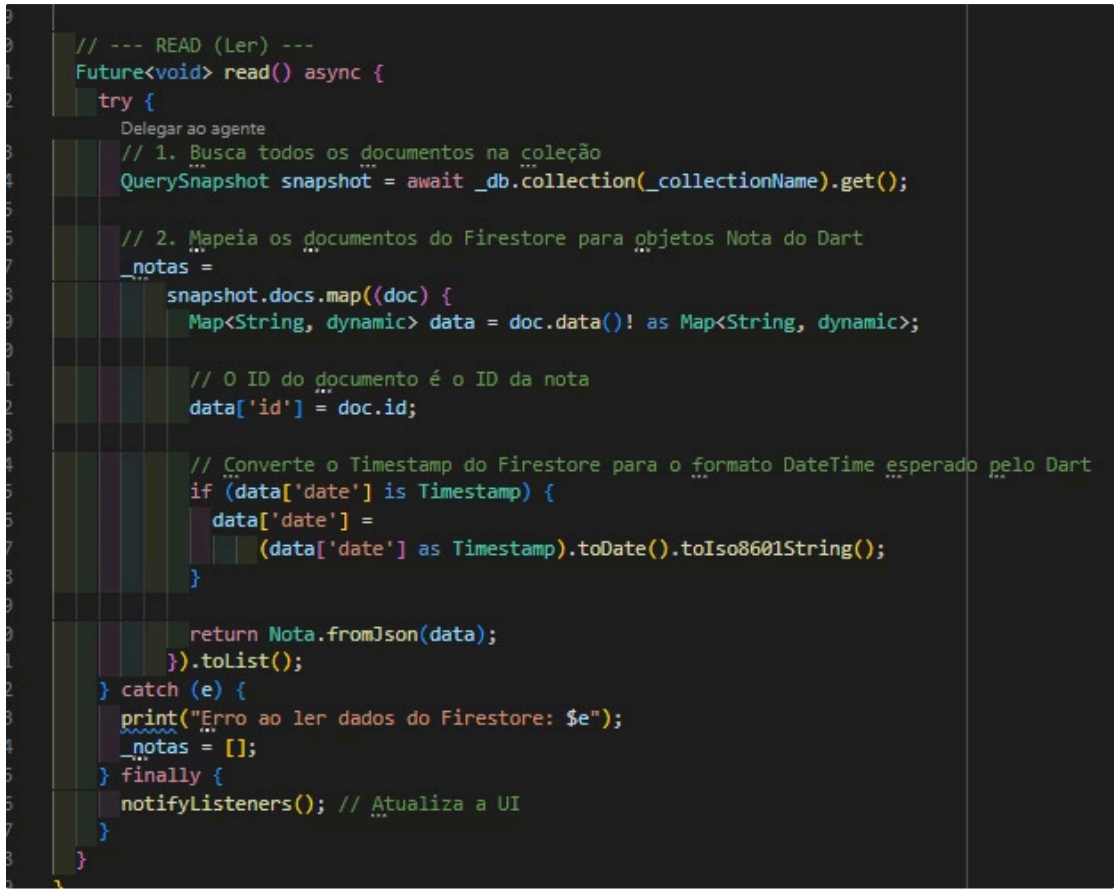
lib > viewmodels > notaviewmodel.dart > NotaViewModel

```dart
// viewmodels/notaviewmodel.dart
import 'package:flutter/material.dart';
import 'package:cloud_firestore/cloud_firestore.dart'; // 💡 NOVO: Import do Firestore
import 'package:provider/provider.dart';
import '../models/nota.dart';
// Remova: os imports de http, dart:convert e uuid (não são mais necessários)

class NotaViewModel extends ChangeNotifier {
  // Referência ao Firestore
  final FirebaseFirestore _db = FirebaseFirestore.instance;

  // Nome da coleção no seu banco de dados
  final String _collectionName = 'notas';
  List<Nota> _notas = [];

  List<Nota> get notas => List.unmodifiable(_notas);

  // O construtor carrega os dados ao iniciar
  NotaViewModel() {
    read();
  }

  // --- CREATE (Criar) ---
  void create(String title) async {
    // 1. Cria um objeto Nota (o ID temporário será substituído)
    Nota novaNota = Nota("nota", title, DateTime.now());

    // 2. Adiciona ao Firestore e obtém a referência do novo documento (docRef)
    await _db.collection(_collectionName).add(novaNota.toJson()).then((docRef) {
      // 3. Atualiza o documento no Firestore com o ID gerado (docRef.id)
      docRef.update({'id': docRef.id});
    });

    // 4. Recarrega a lista para atualizar a UI
    await read();
    notifyListeners();
```

```
38
39  // --- DELETE (Deletar) ---
40  void delete(String id) async {
41    // 1. Atualiza a lista local primeiro (feedback rápido)
42    _notas.removeWhere((e) => e.id == id);
43    notifyListeners();
44
45    // 2. Deleta o documento no Firestore pelo ID
46    await _db.collection(_collectionName).doc(id).delete().catchError((e) {
47      print("Erro ao deletar no Firestore: $e");
48    });
49  }
50
```



```
// --- UPDATE (Atualizar) ---
void update(Nota nota) async {
  // 1. Atualiza a lista local (feedback rápido)
  var index = _notas.indexWhere((e) => e.id == nota.id);
  if (index != -1) {
    _notas[index] = nota;
  }
  notifyListeners();

  // 2. Atualiza o documento no Firestore.
  await _db
      .collection(_collectionName)
      .doc(nota.id) // Usa o ID da nota para localizar
      .update(nota.toJson()) // Atualiza com os novos dados
      .catchError((e) {
        print("Erro ao atualizar no Firestore: $e");
      });
}

// --- READ (Ler) ---
```



```dart
// --- READ (Ler) ---
Future<void> read() async {
  try {
    Delegar ao agente
    // 1. Busca todos os documentos na coleção
    QuerySnapshot snapshot = await _db.collection(_collectionName).get();

    // 2. Mapeia os documentos do Firestore para objetos Nota do Dart
    _notas =
        snapshot.docs.map((doc) {
          Map<String, dynamic> data = doc.data()! as Map<String, dynamic>;

          // O ID do documento é o ID da nota
          data['id'] = doc.id;

          // Converte o Timestamp do Firestore para o formato DateTime esperado pelo Dart
          if (data['date'] is Timestamp) {
            data['date'] =
                (data['date'] as Timestamp).toDate().toIso8601String();
          }

          return Nota.fromJson(data);
        }).toList();
  } catch (e) {
    print("Erro ao ler dados do Firestore: $e");
    _notas = [];
  } finally {
    notifyListeners(); // Atualiza a UI
  }
}
```

CRUD