

Google Cloud Firestore

Objetivo

O Google Cloud Firestore é um banco de dados não relacional baseado em documentos. O Google armazena seus dados nas nuvens, em regiões específicas que você pode escolher, garantindo escala e disponibilidade de seus dados. Neste tutorial, vamos criar um banco de dados no Firestore para armazenar os dados de nossa aplicação.

Código antes destas alterações

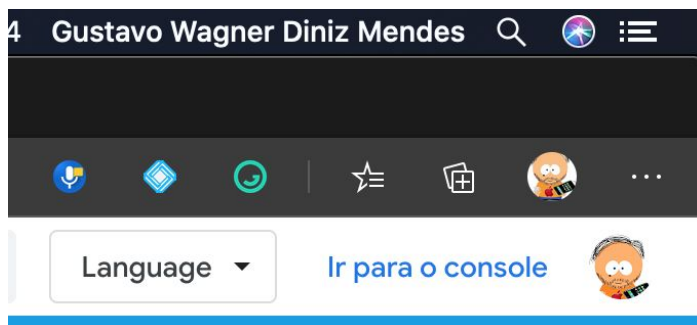
https://github.com/gugawag/social-web/releases/tag/v12-mensagens-tratamento_erros-interceptor_http

Para melhor entendimento, é interessante que se baixe o código acima e aplique as alterações que serão realizadas abaixo.

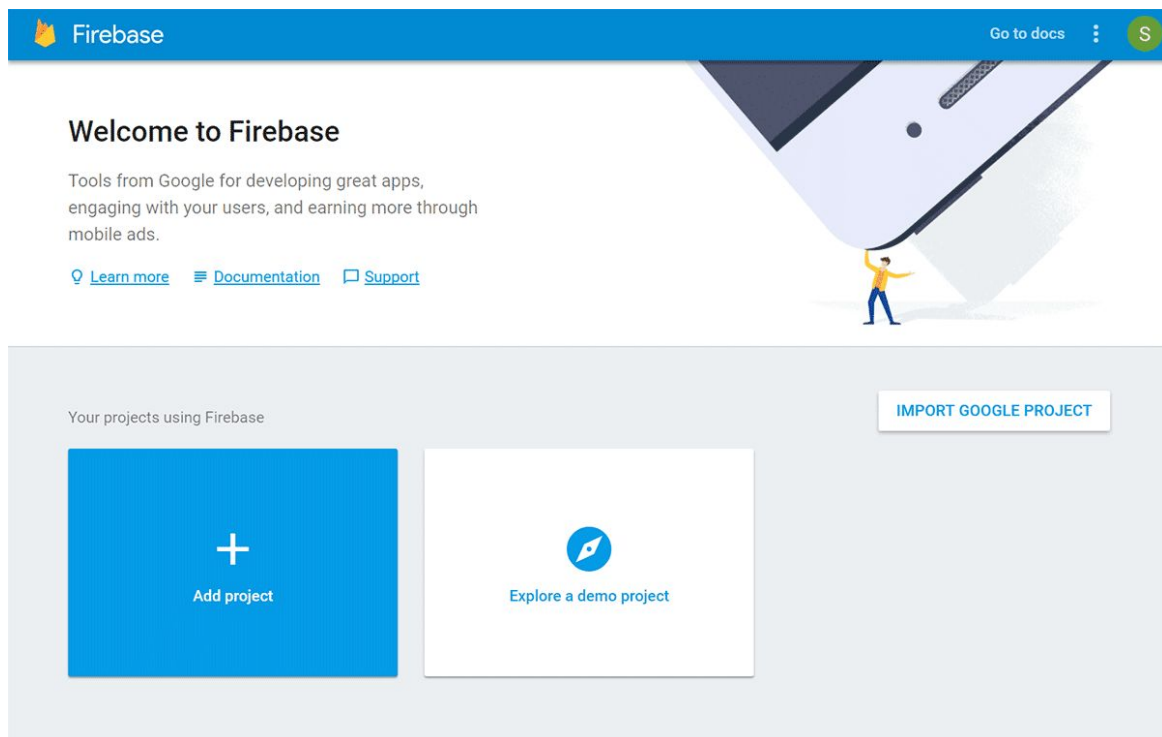
Passos

Para criar um banco de dados no Firestore:

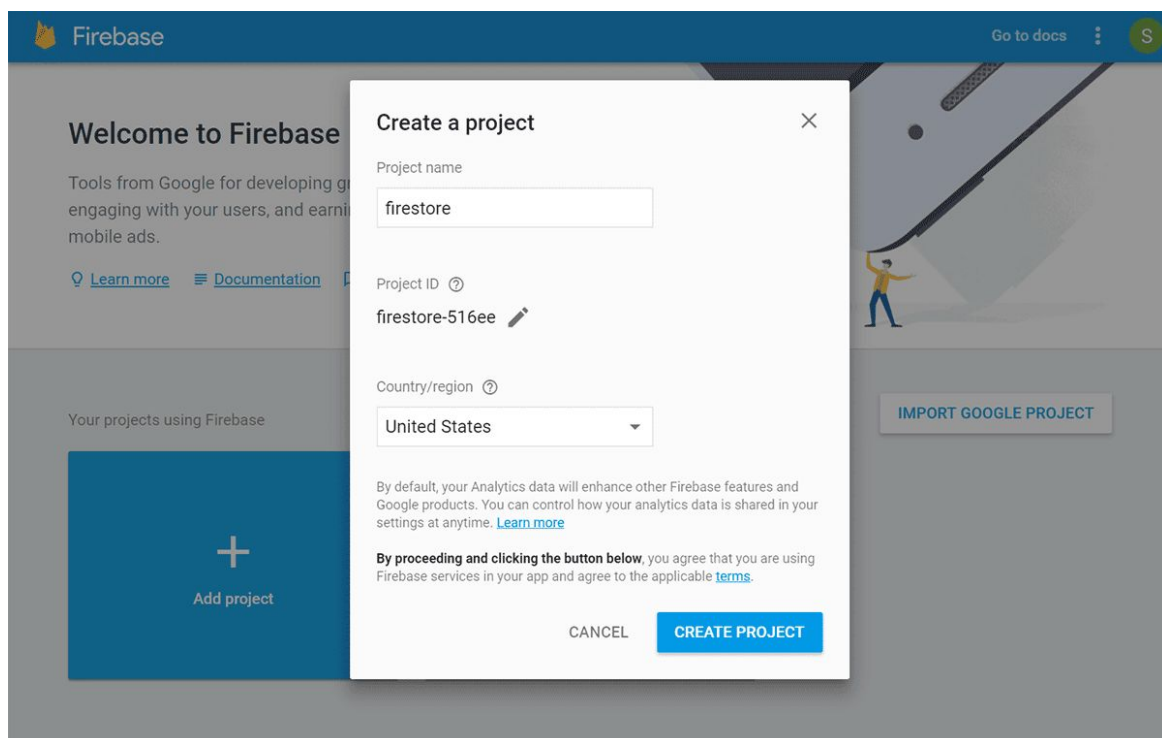
1. Se ainda não tiver, crie uma conta em [Cloud Firestore | Firebase](#)
2. Clique em **"Ir para console"**



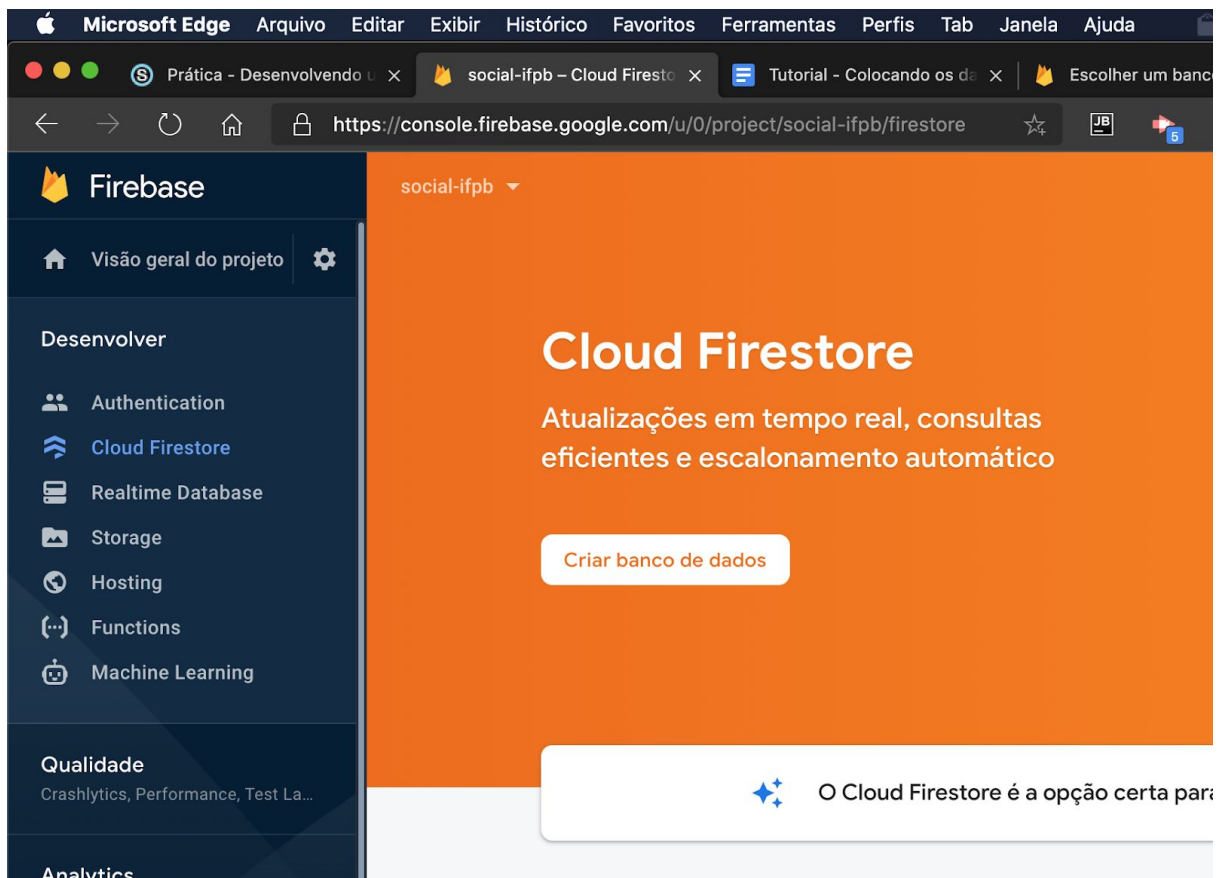
3. Clique em **Adicionar Projeto**



1. Escolha um nome para seu projeto (**social-ifpb**). A tela que você verá pode ser levemente diferente da de baixo)

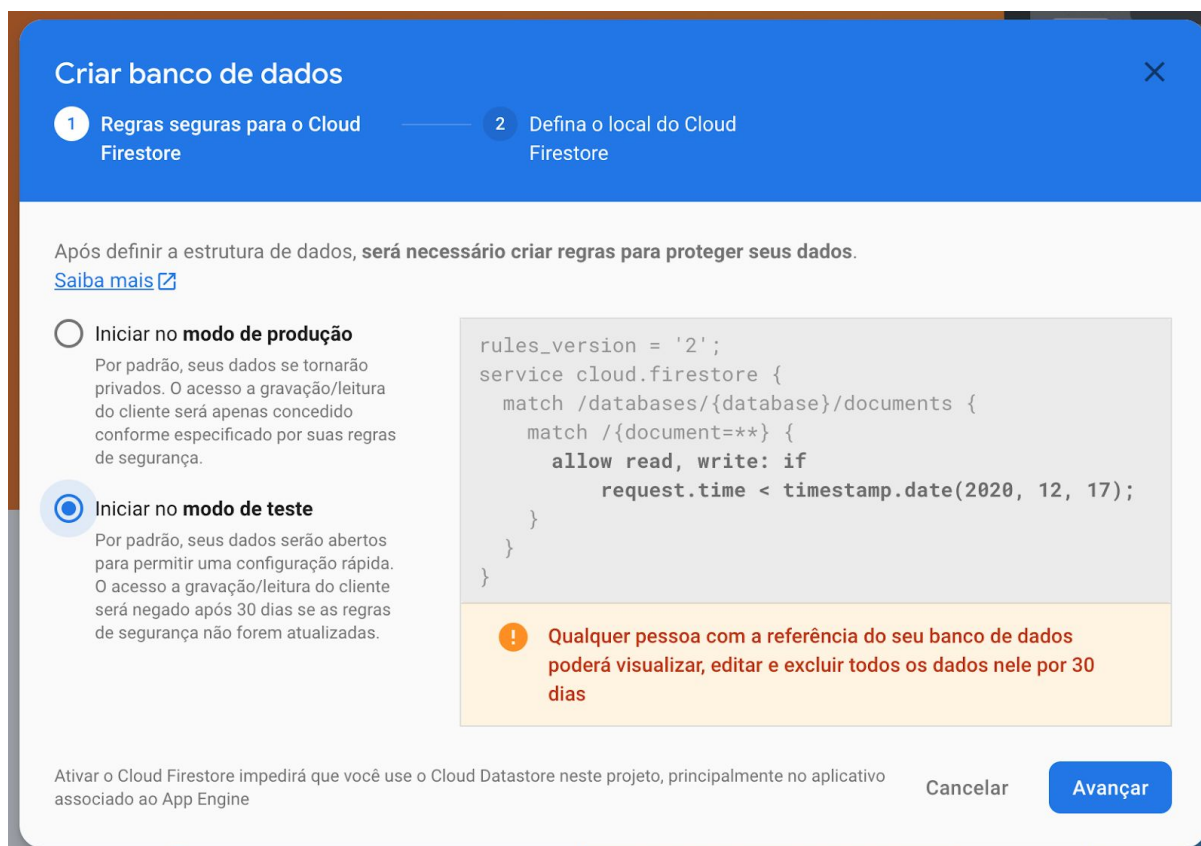


1. Clique em Criar Projeto

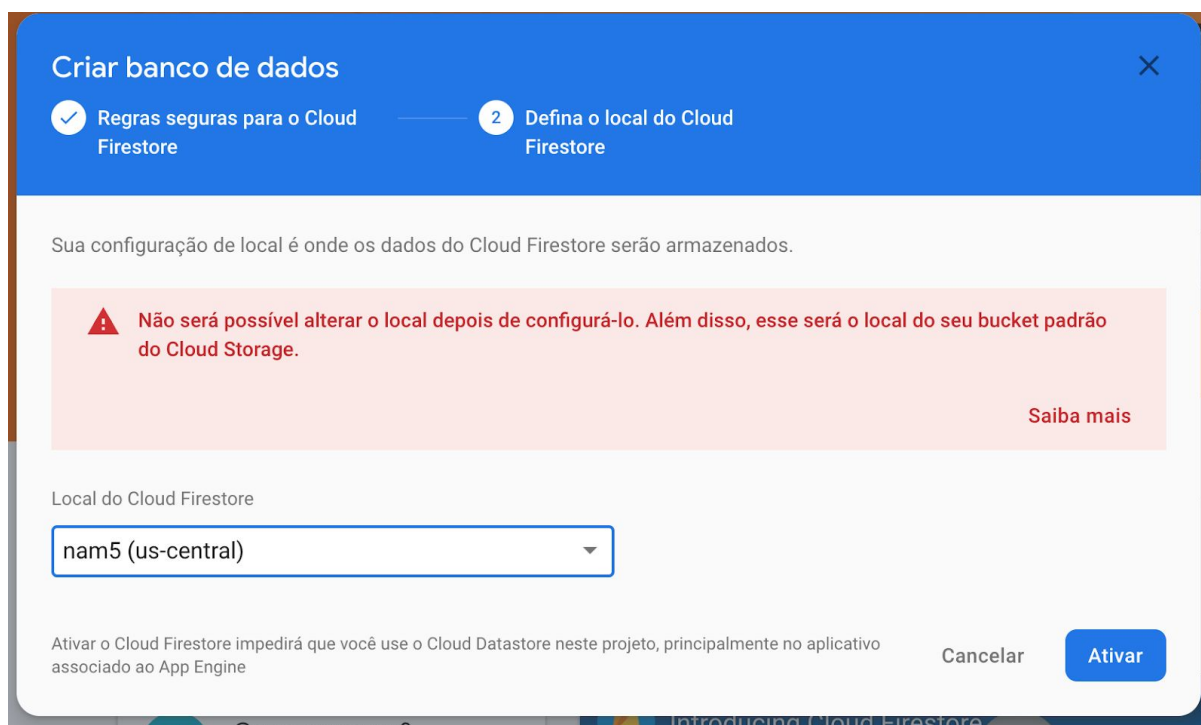


2.

1. Na nova tela (mostrada acima), no menu do lado esquerdo, clique "**Cloud Firestore**". Perceba que há também a opção de "Realtime database". Qual a diferença entre eles? Veja aqui: <https://firebase.google.com/docs/database/rtdb-vs-firestore>.
2. Clique em "**Criar Banco de Dados**"
3. Escolha a opção "Iniciar em modo teste" (qualquer um que tiver a url do seu projeto poderá mexer nos dados)

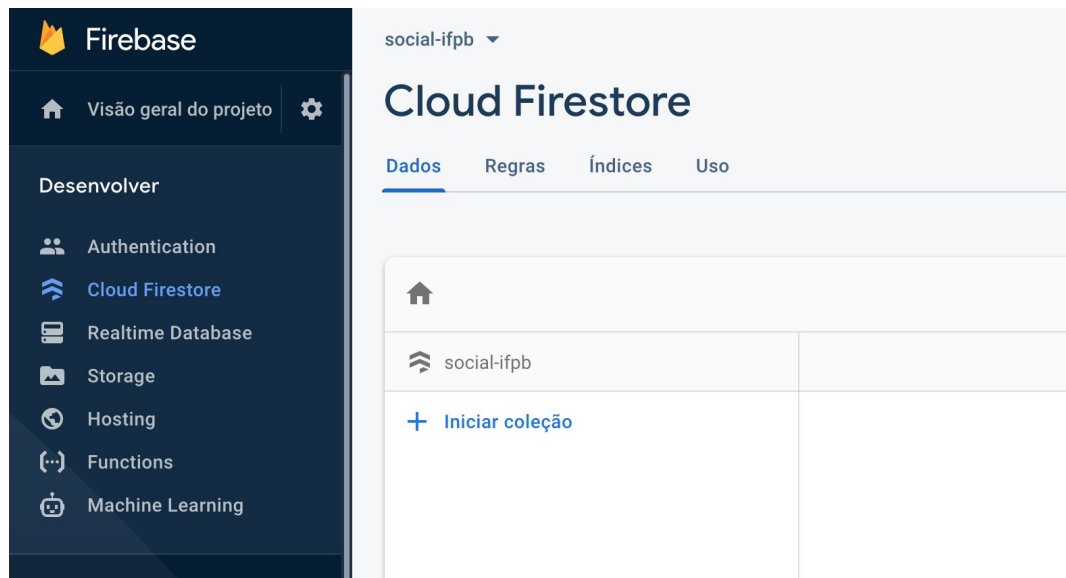


1. Clique em **Avançar**



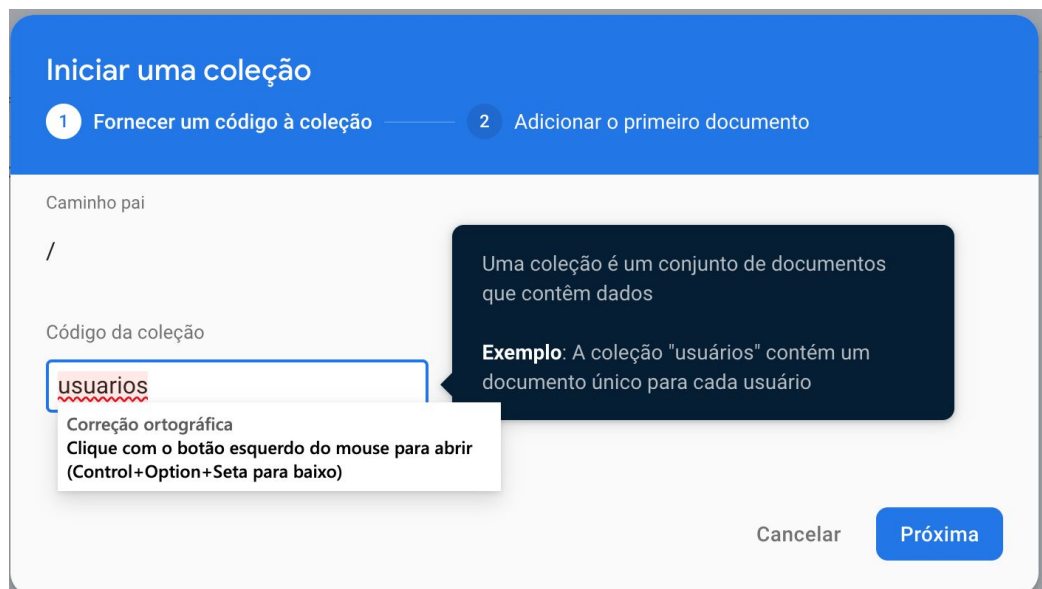
2. Nessa tela acima, é apresentada a opção de escolha de em qual região seus dados serão armazenados. Perceba que essa escolha depende de onde sua aplicação estará, e de onde seus usuários acessaram. Porém, como neste

tutorial estamos fazendo apenas um teste, escolha **us-central**. Clique em **Ativar**. A tela abaixo aparecerá



3.

4. Clique em **+ Iniciar Coleção** para criarmos nossa coleção de usuários. Uma coleção, como o próprio nome já diz, é uma coleção de dados similares, uma coleção de documentos. Cada documento é, na prática, uma estrutura de dados para armazenar seus dados. Por exemplo, ao se armazenar um usuário na coleção **usuários**, você estará armazenando um documento que representa um usuário.



5.

6. Depois de colocar **usuários** no código da coleção, clique em **Próxima**.

Iniciar uma coleção

✓ Fornecer um código à coleção

2 Adicionar o primeiro documento

Caminho pai do documento

/usuarios

Código do documento

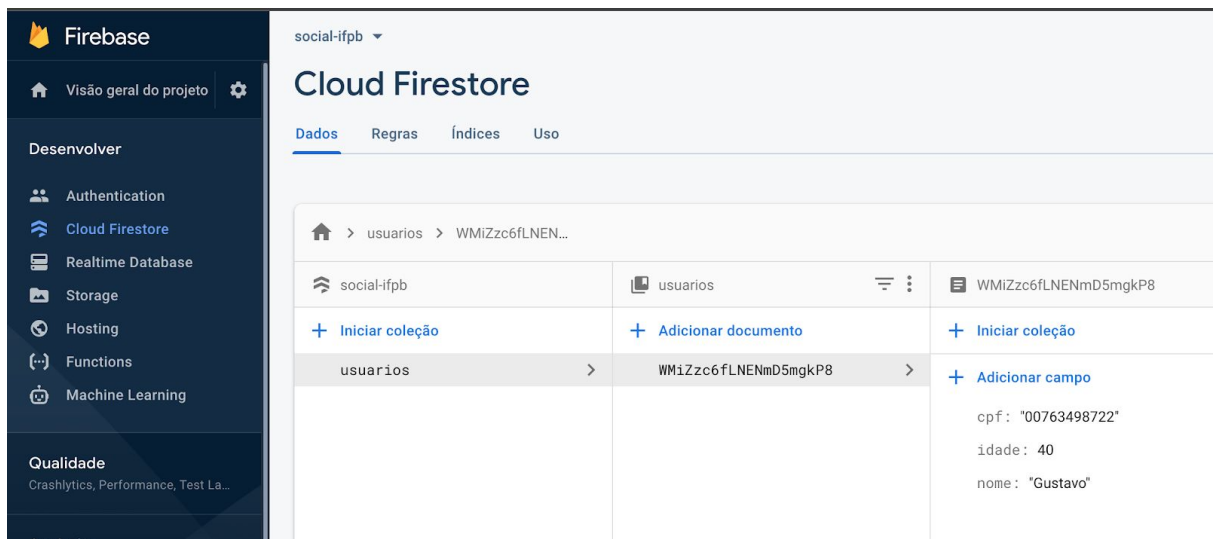
WMiZzc6fLNENmD5mgkP8

| Campo | Tipo | Valor |
|-------|----------|-------------|
| nome | = string | Gustavo |
| cpf | = string | 00763498722 |
| idade | = number | 40 |

+ Adicionar campo

Cancelar Salvar

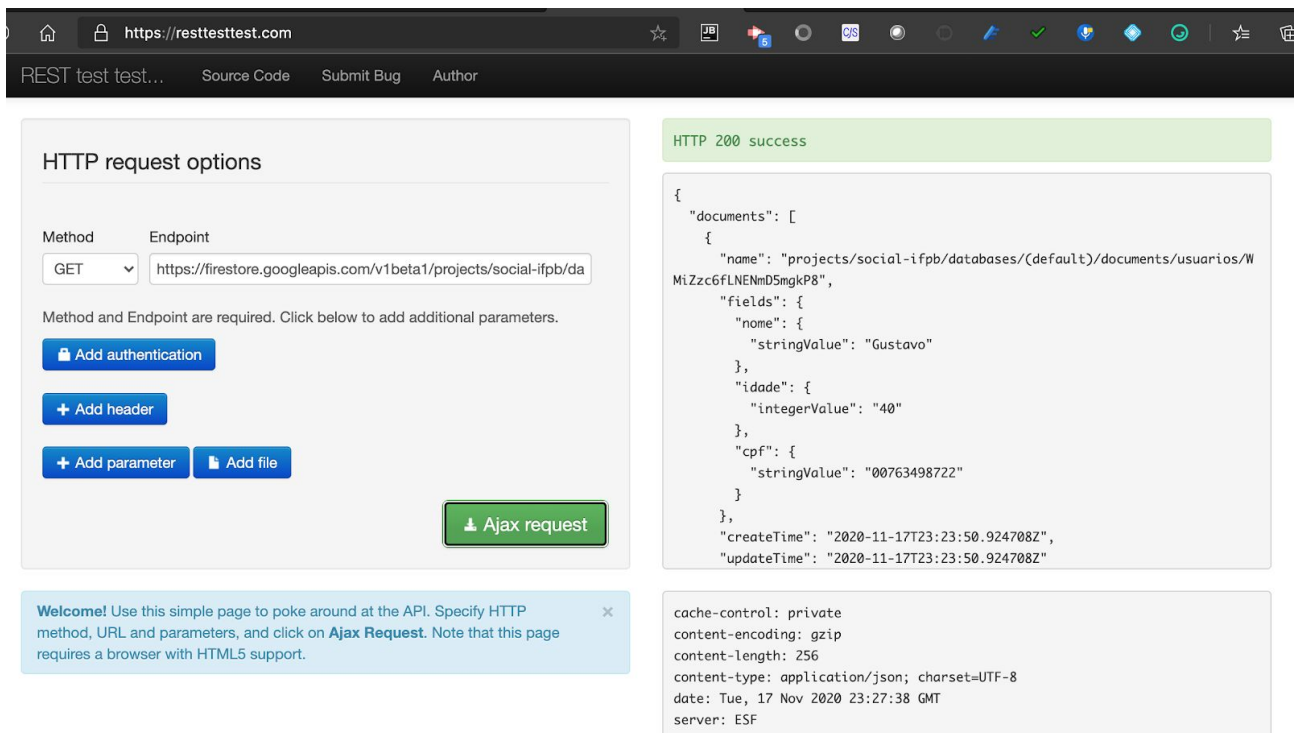
- 7.
8. Na tela acima, você está inserindo o primeiro documento na coleção, ou seja, o primeiro usuário. Perceba que você pode escolher os nomes dos campos, os tipos de dados de cada campo, bem como os valores do usuário específico. Perceba também que como é um banco de dados não relacional, você pode enviar outros campos no futuro, ou mesmo documentos sem todos os campos informados aí, que não terá problema. O código do documento eu cliquei em "**Código Automático**", o que fez o sistema gerar esse código grande. Você pode querer gerenciar o código na mão, ou deixar que ele gerencie (crie automaticamente) os códigos para você. O resultado é o que é mostrado na tela abaixo.



9.

10. Para testar o uso de seu banco de dados, abra a aplicação PostMan (ou um testador rest online):

1. <https://resttesttest.com/>
2. Coloque na url:
`https://firestore.googleapis.com/v1beta1/projects/social-ifpb/databases/(default)/documents/usuarios/`
3. clique em Ajax Request
4. Veja o resultado da consulta. Perceba que os dados foram retornados no formato json, com algumas informações a mais tais como a data de criação e de alteração.



5. Agora mude o método HTTP de GET para POST

6. Coloque no corpo da requisição:

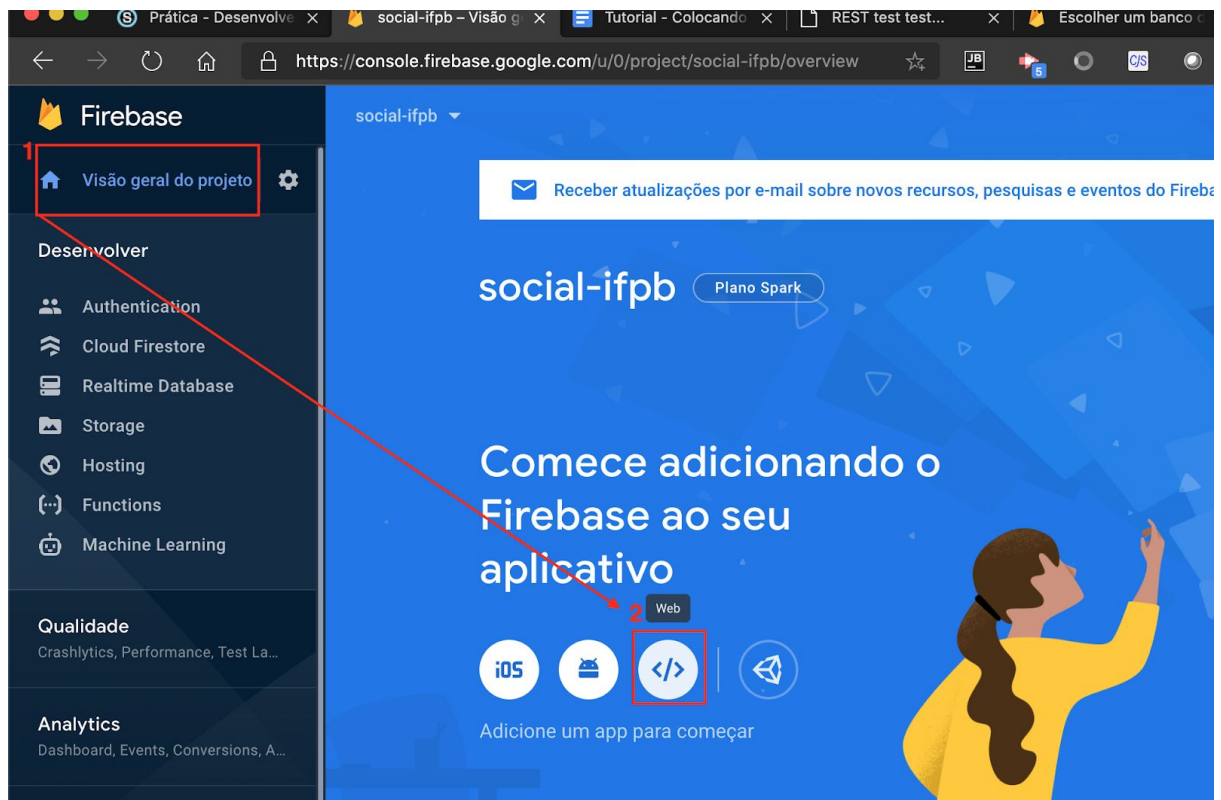
```
{
  "fields": {
    "nome": {
      "stringValue": "SEU NOME AQUI"
    }
  }
}
```

7. Envie e veja a resposta do Firestore

1. Adicionando Firestore no Angular

O firestore tem uma api REST, mas é aconselhável utilizar a api nativa da tecnologia de sua aplicação (web, iOS, Android etc), por simplicidade. Para isso, vamos acrescentar a biblioteca do Firestore em nossa aplicação Angular.

Primeiro, vamos ver como configurar nosso banco de dados para ser usado na nossa app web. Clique, como mostra na imagem abaixo, em **"Visão geral do projeto"** e depois no ícone da **web**.



× Adicionar o Firebase ao seu app da Web

1

Registrar app

Apelido do app ?

social



Também configure o **Firebase Hosting** para este app. [Saiba mais](#) 

A configuração do Hosting também pode ser feita depois. Comece a usar a qualquer momer

Registrar app

2

Adicionar SDK do Firebase

- 1.
2. Digite um apelido para a app (social) e clique em "**Registrar app**". Será mostrada a config abaixo:

✓ Registrar app

2 Adicionar SDK do Firebase

Copie e cole esses scripts na parte inferior da tag <body>, mas antes de usar qualquer serviço do Firebase:

```
<!-- The core Firebase JS SDK is always required and must be listed first -->
<script src="https://www.gstatic.com/firebasejs/8.0.2/firebase-app.js"></script>

<!-- TODO: Add SDKs for Firebase products that you want to use
https://firebase.google.com/docs/web/setup#available-libraries -->

<script>
  // Your web app's Firebase configuration
  var firebaseConfig = {
    apiKey: "AIzaSyCDJa1IHn-556S0nmKcHzfWxw1xR75fyTU",
    authDomain: "social-ifpb.firebaseio.com",
    databaseURL: "https://social-ifpb.firebaseio.com",
    projectId: "social-ifpb",
    storageBucket: "social-ifpb.appspot.com",
    messagingSenderId: "831452993495",
    appId: "1:831452993495:web:e9d7ff442c992653225d92"
  };
  // Initialize Firebase
  firebase.initializeApp(firebaseConfig);
</script>
```



Saiba mais sobre o Firebase para Web: [Primeiros passos](#), [Referência da API Web SDK](#), [Amostras](#)

Continuar no console

- 3.
4. Copie essa configuração pois usaremos parte dela na nossa app Angular. O conteúdo a ser copiado se encontra abaixo (lembre-se que esse código abaixo é do Firebase que eu, Gustavo, criei. Se você está criando seu próprio banco, deve usar o conteúdo gerado no console do seu Firebase):

```
<!-- The core Firebase JS SDK is always required and must be listed first -->

<script
src="https://www.gstatic.com/firebasejs/8.0.2/firebase-app.js"></script>

<!-- TODO: Add SDKs for Firebase products that you want to use
https://firebase.google.com/docs/web/setup#available-libraries -->

<script>
```

```
// Your web app's Firebase configuration

var firebaseConfig = {
  apiKey: "AIzaSyCDJa1IHn-556S0nmKcHzfWxw1xR75fyTU",
  authDomain: "social-ifpb.firebaseio.com",
  databaseURL: "https://social-ifpb.firebaseio.com",
  projectId: "social-ifpb",
  storageBucket: "social-ifpb.appspot.com",
  messagingSenderId: "831452993495",
  appId: "1:831452993495:web:e9d7ff442c992653225d92"
};

// Initialize Firebase
firebase.initializeApp(firebaseConfig);

</script>
```

5. Crie o arquivo `firebase.config.ts` na raiz e coloque conteúdo (altere para os valores corretos de sua app). Perceba que esses valores são similares aos valores que você copiou da tela acima:

```
export const FirebaseConfig = {
  firebase: {
    apiKey: 'AIzaSyCDJa1IHn-556S0nmKcHzfWxw1xR75fyTU',
    authDomain: 'social-ifpb.firebaseio.com',
    databaseURL: 'https://social-ifpb.firebaseio.com',
    projectId: 'social-ifpb',
    storageBucket: 'social-ifpb.appspot.com',
    messagingSenderId: '831452993495',
    appId: '1:831452993495:web:e9d7ff442c992653225d92'
  }
};
```

6. Instale o @angular/fire e o firestore, bibliotecas do Google Cloud Firestore para Angular:

1. **npm install @angular/fire firebase --save**

7. Crie o módulo firestore para ter os módulos necessários para o Firestore e depois o importe no app.module.ts:

1. **ng g module firestore**

2. Altere o módulo gerado para ficar parecido com o abaixo. Perceba que fazemos a inicialização do Angular utilizando as configurações do FirebaseConfig.firebase que fizemos acima:

```
AngularFireModule.initializeApp(FirebaseConfig.firebase),
```

3. Importe esse módulo no **app.module.ts**

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

import { CommonModule } from '@angular/common';

import { AngularFireModule } from '@angular/fire';

import { FirebaseConfig } from '../../firebase.config';

import { AngularFireStoreModule } from '@angular/fire/firestore';

import { AngularFireAuthModule } from '@angular/fire/auth';

import { AngularFireDatabaseModule } from '@angular/fire/database';

@NgModule({

  declarations: [],

  imports: [

    CommonModule,

    AngularFireModule.initializeApp(FirebaseConfig.firebase),

    AngularFireStoreModule,
```

```

    AngularFireAuthModule,

    AngularFireDatabaseModule

]

}))

export class FirestoreModule { }

```

2. Criando um novo service para se comunicar com Firestore

Agora que temos o banco configurado, precisamos alterar o service (ou criar um novo) para se comunicar com o Firestore. Manteremos o outro service apenas por motivo de comparação.

1. Crie o serviço **UsuarioFirestoreService** para conter uma coleção do Firestore de usuários, com os mesmos métodos do UsuarioService já existente.

i. ng g service shared/services/UsuarioFirestore

2. Altere a classe para que fique com código similar ao debaixo:

```

1. import {Injectable} from '@angular/core';
2. import {from, Observable} from 'rxjs';
3. import {Usuario} from '../model/usuario';
4. import {AngularFirestore, AngularFirestoreCollection} from
   '@angular/fire/firestore';
5. import {map} from 'rxjs/operators';
6.
7. @Injectable({
8.   providedIn: 'root'
9. })
10. export class UsuarioFirestoreService {
11.
12.   colecaoUsuarios: AngularFirestoreCollection<Usuario>;
13.   NOME_COLECAO = 'usuarios';
14.
15.   constructor(private afs: AngularFirestore) {
16.     this.colecaoUsuarios = afs.collection(this.NOME_COLECAO);
17.   }
18.
19.   listar(): Observable<Usuario[]> {

```

```

20. // usando options para idField para mapear o id gerado pelo
    firestore para o campo id de usuário
21. return this.colecaoUsuarios.valueChanges({idField: 'id'});
22. }
23.
24. inserir(usuario: Usuario): Observable<object> {
25. // removendo id pois ele está undefined, já que um novo
    usuário
26. delete usuario.id;
27. // Object.assign({}, usuario) é usado para passar um objeto
    json puro. Não se aceita passar um objeto customizado
28. // o from transforma uma promise num Observable, para
    mantermos a assinatura similar ao do outro service
29. return from(this.colecaoUsuarios.add(Object.assign({},
    usuario)));
30. }
31.
32. remover(id: string): Observable<void> {
33. return from(this.colecaoUsuarios.doc(id).delete());
34. }
35.
36. pesquisarPorId(id: string): Observable<Usuario> {
37. // como o objeto retornado pelo get é um DocumentData, e não
    um usuário, transformamos a partir de um pipe e mapeamos de um
    document
38. // para o tipo usuário
39. return this.colecaoUsuarios.doc(id).get().pipe(map(document
    => new Usuario(document.id, document.data())));
40. }
41.
42. atualizar(usuario: Usuario): Observable<void> {
43. // removendo id pois não vamos guardar nos dados do
    documento, mas sim usar apenas como id para recuperar o
    documento
44. delete usuario.id;
45. return
    from(this.colecaoUsuarios.doc(usuario.id).update(Object.assign({
    }, usuario)));
46. }
47.
48. listarMaioresDeIdade(): Observable<Usuario[]> {
49. let usuariosMaioresIdade:
    AngularFireCollection<Usuario>;
50. // fazendo pesquisas usando o where. Um where pode ser
    encadeado com outro

```

```









51. usuariosMaioresIdade = this.afs.collection(this.NOME_COLECAO,
    ref => ref.where('idade', '>', '17'));
52. return usuariosMaioresIdade.valueChanges();
53. }
54.
55. }

```

56.

3. Vamos às explicações do código acima:

1. Primeiro, é necessário que se tenha uma referência à coleção de dados. Isso está sendo feito na linha 12 com a criação de uma coleção do tipo `AngularFirestoreCollection` do tipo `Usuario`: `colecacaoUsuarios: AngularFirestoreCollection<Usuario>;`
2. A comunicação com o Firestore se dá através da classe `AngularFirestore`, e inserimos um atributo desta classe no construtor, ligando a coleção `colecacaoUsuarios` à sua coleção no firestore (linha 16): `this.colecacaoUsuarios = afs.collection(this.NOME_COLECAO);`
3. **Listagem:** Para listar todos os documentos da coleção (no caso específico, todos os usuários), utilizamos o método `valueChanges()`, passando o nome do atributo `id` na classe usuário. Se não for passado esse `{idField: 'id'}`, será retornado um array de usuários, porém com o atributo `id` como `undefined` (linha 21): `return this.colecacaoUsuarios.valueChanges({idField: 'id'});`
 - i. Perceba que o que é retornado é um `Observable`, similar a como retornávamos antes.
4. **Inserir:** O método `inserir`, que começa na linha 24, começa apagando o `id` do objeto usuário (linha 26). Isso porque o documento usuário não tem um atributo `id` em si, internamente, e sim o `id` é gravado fora, como pode ser visto na imagem abaixo:

| | | |
|--|---|--|
|  > usuarios > zAHQ53hMLxvD... | | |
|  social-ifpb |  usuarios |  zAHQ53hMLxvDG8brlySz |
|  Iniciar coleção |  Adicionar documento |  Iniciar coleção |
| usuarios > | zAHQ53hMLxvDG8brlySz > |  Adicionar campo |
| | | cpf: "46573898755" idade: "17" nome: "Wagner" |

- i. Perceba que o `id` que começa por **zAHQ53...** não é armazenado diretamente nos atributos do documento usuário, e sim como `id` para achar o documento. Se se quiser inserir um `id` comercial, que não é auto-gerado pelo Firestore, você pode inserir normalmente um atributo `id` para ser armazenado como atributo do documento. Apagamos com `delete usuario.id;` porque ficaria redundante.

- ii. Na linha 29, temos o seguinte:

```
return  
from(this.colecaoUsuarios.add(Object.assign({},  
usuario)));
```
 - iii. perceba que chamamos o método **add** da coleção para inserir um objeto. Porém, se passássemos o objeto **usuario** em si, o Firestore não aceitaria, já que tem que ser passado um json puro. Por isso, chamamos o método **Object.assign({}, usuario)** para que seja usado o json puro dos atributos do objeto **usuario**.
 - iv. perceba também que usamos **from()** para ser retornado. O método **add** de uma coleção devolve uma **promise**. Para mantermos o padrão do código anterior, que devolvemos um **Observable<Usuario>**, chamamos o **from()**, que recebe uma **promise** e transforma num **Observable**;
5. **Remove:** Na linha 33 fazemos a remoção de um documento. Para isso, recebemos um id do tipo string (explicaremos mais à frente que houve alteração na classe **Usuario** para transformar o id de number para string) e passamos para o método **doc(id)** para buscar o documento e depois apagar (**delete()**). Difere do método **remove** da api REST no retorno, já que, no caso do Firestore, ele retorna um **Promise<void>**, e chamamos o **from** para transformar a **promise** num **Observable**:

```
return  
from(this.colecaoUsuarios.doc(id).delete());
```
6. **Pesquisar por id:** para pesquisar (linha 39), nós pedimos à coleção um doc com o id especificado:

```
this.colecaoUsuarios.doc(id).get()
```

 Porém, o retorno do **get** é um **Observable** de **DocumentSnapshot<DocumentData>**, ou seja, não é um **Observable** de **Usuario**, por ser genérico. Para que devolvamos um **Observable<Usuario>**, transformamos o dado através do **pipe**, e mapeamos o documento (variável **document**) para um objeto do tipo usuário (**new Usuario()**):

```
pipe(map(document => new  
Usuario(document.id, document.data())))
```
- i. Perceba que, para fazer isso, criamos alteramos o tipo **id** da classe **Usuario** para ser **string** (ao invés de **number**), deixamos os atributos como sendo opcionais (a interrogação ao final do nome do atributo) e criamos um construtor com valores default:

```
1. export class Usuario {  
2.   id?: string;  
3.   nome?: string;  
4.   cpf?: string;  
5.   idade?: number;  
6.  
7.   constructor(id?: string, usuario: Usuario = {}) {  
8.     this.id = id;  
9.     this.cpf = usuario.cpf;  
10.    this.nome = usuario.nome;  
11.    this.idade = usuario.idade;  
12.  }
```

7. **Atualizar:** Para atualizar um documento, apagamos o atributo **id** de usuário na linha 44 (como já discutido acima) e chamamos o método **update**, passando o json puro:


```

return
from(this.colecaoUsuarios.doc(usuario.id).update(Object.assign({},
usuario)));

```

8. **Filtrar coleções:** por fim, a partir da linha 48, criamos uma nova coleção e definimos como é seu filtro. Ao criar uma referência para a coleção de usuários, passamos uma função de consulta, onde podemos especificar as restrições: `ref => ref.where('idade', '>', '17'))`

- i. no caso caso específico, estamos querendo pegar a coleção de usuários cuja idade seja maior que 17. Podemos filtrar mais ainda, encadeando **wheres**. Por exemplo:

1. `ref => ref.where('idade', '>', '17').where('nome', '>', 'Guilherme'))`
2. Neste caso, está se filtrando usuários pela idade > 17 e cujo nome seja > Guilherme (alfabeticamente falando).
3. Para conhecer mais possibilidades de filtros pesquise em: <https://firebase.google.com/docs/firestore/query-data/queries>

2.1 Adaptando o código para o novo serviço

Perceba que a forma que desenvolvemos o novo serviço que se comunica com o Firestore as alterações no código que tínhamos é mínima. Precisamos:

1. Alterar o **cadastro-usuario-component.ts** para usar o **UsuarioFirestoreService** ao invés do **UsuarioService** (lembrar de mudar a linha 24 para não mais transformar o valor do id da rota para Number)
2. Alterar o **listagem-usuario-tabela.component.ts** e o **listagem-usuario.component.ts** para usarem o **UsuarioFirestoreService** ao invés do **UsuarioService** (lembrar de alterar o tipo do id no método apagar de number para string)
3. Usar um **<mat-slide-toggle>** no **listagem-usuario.component.html** para escolher entre a pesquisa da coleção completa ou a coleção filtrada pelos usuários com idade maior que 17, e implementar um método no componente tipo o abaixo, chamando no slide-toggle:
 - a. `atualizarListagem(): void {`
 - b. `if (this.maioresIdade) {`
 - c. `this.usuarioService.listarMaioresDeldade().subscribe(`
 - d. `usuarios => this.usuarios = usuarios`
 - e. `);`
 - f. `} else {`
 - g. `this.usuarioService.listar().subscribe(`
 - h. `usuarios => this.usuarios = usuarios`
 - i. `);`
 - j. `}`
 - k. `}`
 - l.

Código após estas alterações

<https://github.com/gugawag/social-web/releases/tag/v13-firestore>