

Os plugins utilizados foram essas versões:

**cloud\_firestore: ^0.13.0+1**

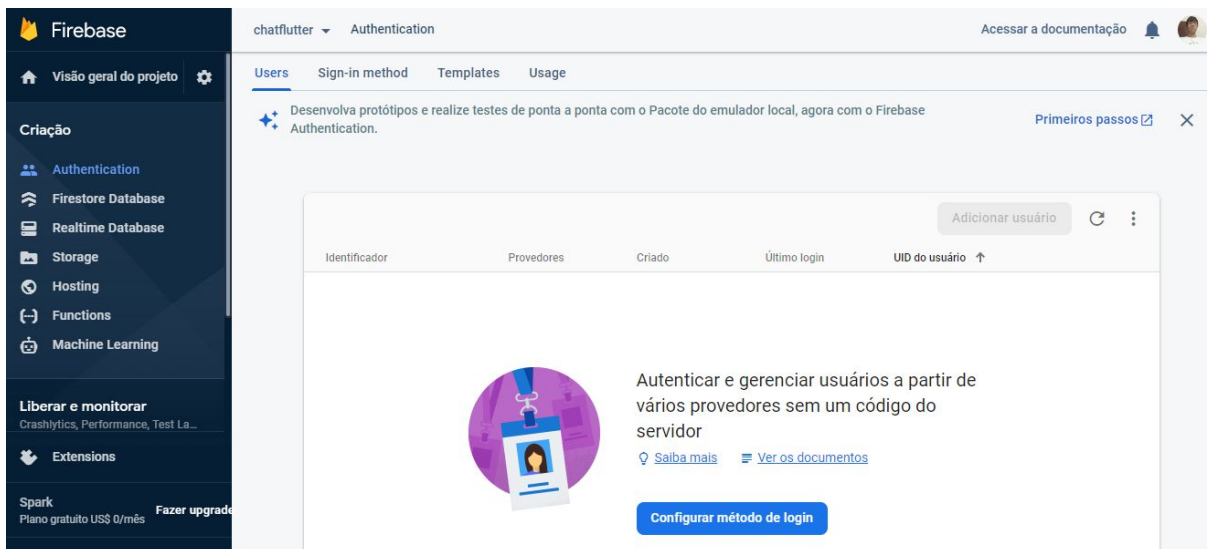
**image\_picker: ^0.6.2+3**

**google\_sign\_in: ^4.1.1**

**firebase\_storage: ^3.1.1**

**firebase\_auth: ^0.15.3**

Autenticação - Lista de todos os usuários e os métodos de login;



Método de Login Google - Escolher seu email para suporte de projeto;

☒ Ativar

O Login do Google é automaticamente configurado nos seus apps iOS e da Web conectados. Para configurar o Login nos seus apps para Android, é necessário adicionar a [impressão digital SHA1](#) em cada aplicativo nas [Configurações do projeto](#).

Atualize abaixo a [configuração no nível do projeto](#) para continuar

Nome público do projeto ⓘ

project-954793226167

E-mail de suporte do projeto ⓘ

chines1998@hotmail.com

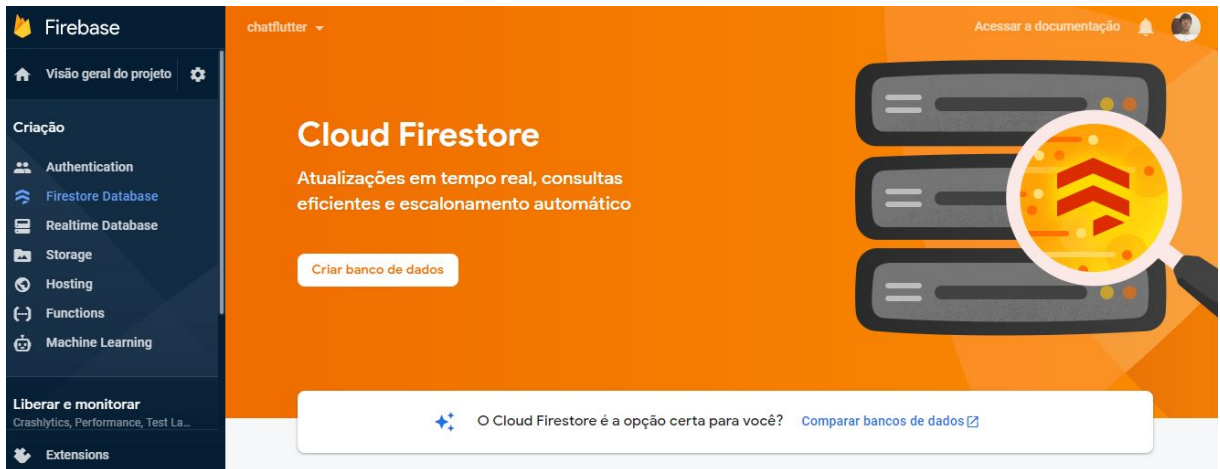
Adicionar IDs de cliente à lista de permissões usando projetos externos (opcional) ⓘ

Configuração do SDK da Web ⓘ

Cancelar Salvar

Database - Armazena dados em formato de texto;

Storage : armazena imagens;



Modo de Produção - Leitura e escrita são bloqueados;  
Modo de Teste - Leitura e escrita por um tempo limitado;



Escolha o modo de teste, em seguida us-central e ativar

### Criar banco de dados

✓ Regras seguras para o Cloud Firestore

2 Defina o local do Cloud Firestore

Sua configuração de local é onde os dados do Cloud Firestore serão armazenados.

⚠ Não será possível alterar o local depois de configurá-lo. Além disso, esse será o local do seu bucket padrão do Cloud Storage.

[Saiba mais](#)

Local do Cloud Firestore

nam5 (us-central)

Ativar o Cloud Firestore impedirá que você use o Cloud Datastore neste projeto, principalmente no aplicativo associado ao App Engine

CancelarAtivar

Dados criados, regras, índices e uso (estatísticas do uso do banco de dados - verifica se não está ultrapassando o limite da cota do plano);

DadosRegrasÍndicesUso


Desenvolva protótipos e realize testes de ponta a ponta com o Pacote do emulador local, agora com o Firebase Authentication.

Primeiros passos

home

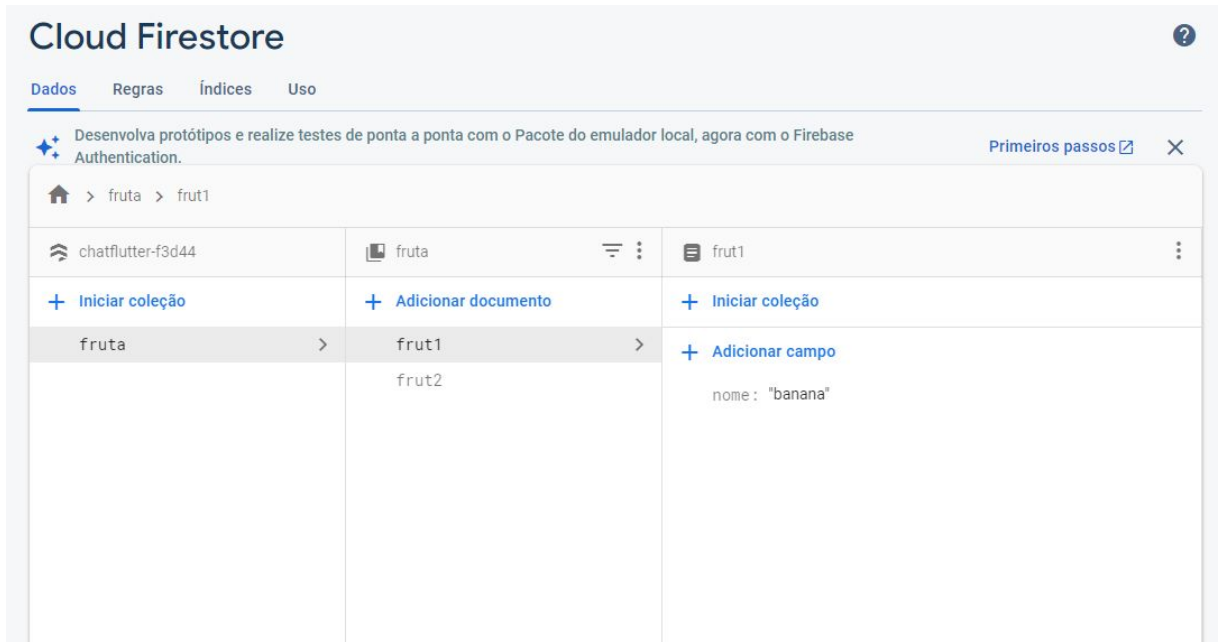
chatflutter-f3d44

+ Iniciar coleção

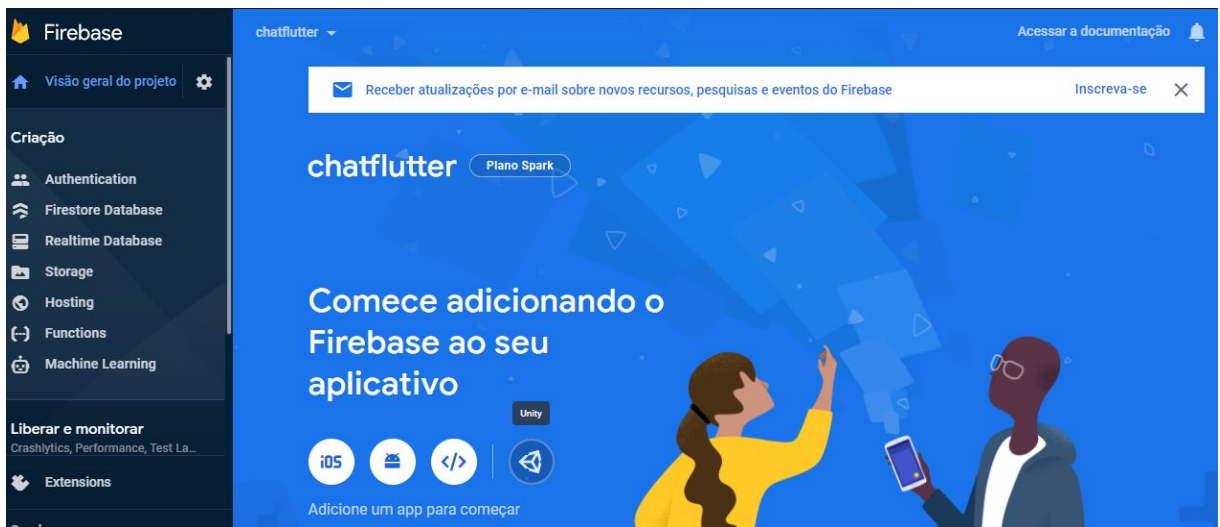


Seu banco de dados está pronto, basta adicionar dados.

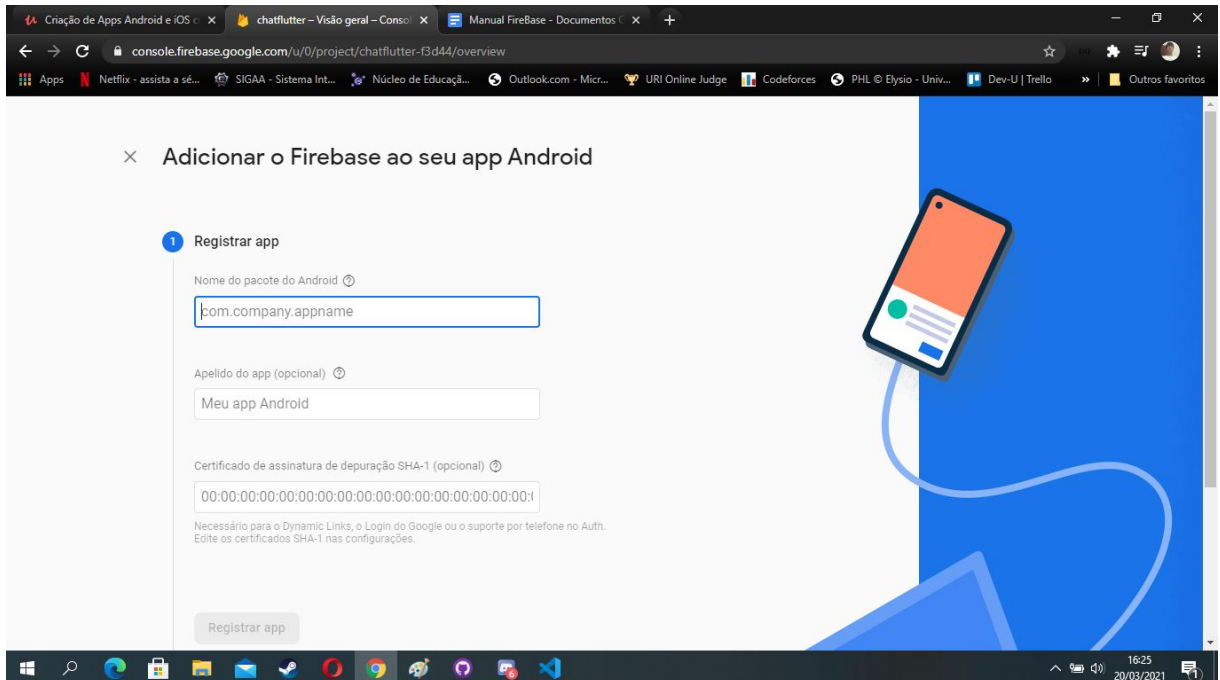
Dados organizados em coleções, dos quais possuem documentos e neste documentos podem possuir outras coleções ou campo (valores);



Adicionando o Firebase ao seu aplicativo - clicar no ícone do android ;



Obtenha o nome do pacote através do projeto android no seguinte endereço, *android/app/src/main/AndroidManifest.xml* em package.



Para obter o Certificado de assinatura de depuração SHA-1 (opcional) que serve para autenticar utilizando conta do google.

Ir na pasta **C:\Program Files\Android\Android Studio\jre\bin** na navegação digitar cmd, no prompt de comando escrever o comando:

**keytool -list -v -keystore "%USERPROFILE%\android\debug.keystore" -alias androiddebugkey -storepass android -keypass android**

Copie e cole SHA1 e por fim clique em Registrar;

```
Selecionar C:\Windows\System32\cmd.exe
Microsoft Windows [versão 10.0.19042.867]
(c) 2020 Microsoft Corporation. Todos os direitos reservados.

C:\Program Files\Android\Android Studio\jre\bin>keytool -list -v -keystore "%USERPROFILE%\android\debug.keystore" -alias androiddebugkey -storepass android -keypass android
Nome do alias: androiddebugkey
Data de criação: 09/03/2021
Tipo de entrada: PrivateKeyEntry
Comprimento da cadeia de certificados: 1
Certificado[1]:
Proprietário: C=US, O=Android, CN=Android Debug
Emissor: C=US, O=Android, CN=Android Debug
Número de série: 1
Válido de Tue Mar 09 16:07:47 BRT 2021 até Thu Mar 02 16:07:47 BRT 2051
Fingerprints do certificado:
MD5: AF:70:30:A8:C4:80:77:7E:11:1A:3F:5F:96:93:50:D6
SHA1: D8:00:F3:F8:4D:F8:68:45:B6:E1:91:0B:E0:12:24:39:3F:54:A2:F3
SHA256: F7:B5:4C:9C:36:F7:D5:85:6B:45:6F:DC:E9:34:E0:3F:E2:93:88:36:E6:AB:83:B4:1B:5D:D8:8A:99:5E:D7:9B
Nome do algoritmo de assinatura: SHA1withRSA
Algoritmo de Chave Pública do Assunto: Chave RSA de 2048 bits
Versão: 1

Warning:
O armazenamento de chaves JKS usa um formato proprietário. É recomendada a migração para PKCS12, que é um formato de padrão industrial que usa "keytool -importkeystore -srckeystore C:\Users\Particular\.android\debug.keystore -destkeystore C:\Users\Particular\.android\debug.keystore -deststoretype pkcs12".

C:\Program Files\Android\Android Studio\jre\bin>
```



Baixe o arquivo e coloque na pasta android/app do Aplicativo Android;

## Adicionar o Firebase ao seu app Android



### Registrar app

Nome do pacote Android: br.com.tanli.chat, apelido do app: Chat-Android



### Fazer o download do arquivo de configuração

Instruções para o Android Studio abaixo | [Unity](#) [C++](#)

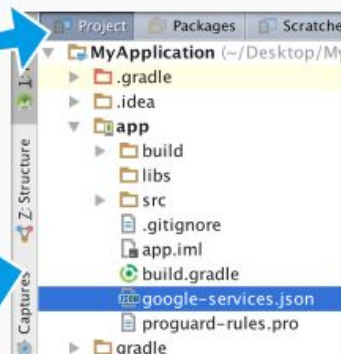
↓ Fazer o download de google-services.json

Mude para a visualização do Projeto no Android Studio para ver o diretório raiz.

Mova o arquivo google-services.json que você acabou de salvar para o diretório raiz do módulo do app Android.



google-services.json



Em seguida clique em próxima;

No projeto Android, vá em **android/app/build.gradle** e cole o **apply plugin: 'com.google.gms.google-services'** no final. E em **defaultConfig** coloque abaixo de **targetSdkVersion** isso **multiDexEnabled true**.

build.gradle no nível do app (<project>/<app-module>/build.gradle):

```
apply plugin: 'com.android.application'
// Add this line
apply plugin: 'com.google.gms.google-services'

dependencies {
    // Import the Firebase BoM
    implementation platform('com.google.firebase:firebase-bom:26.7.0')

    // Add the dependencies for the desired Firebase products
    // https://firebase.google.com/docs/android/setup#available-libraries
}
```

Vá em **build.gradle** fora da pasta app e copie **classpath 'com.google.gms:google-services:4.3.5'** dentro dos parênteses de dependências.

build.gradle no nível do projeto (<project>/build.gradle):

```
buildscript {
    repositories {
        // Check that you have the following line (if not, add it):
        google() // Google's Maven repository
    }
    dependencies {
        ...
        // Add this line
        classpath 'com.google.gms:google-services:4.3.5'
    }
}

allprojects {
    ...
    repositories {
        // Check that you have the following line (if not, add it):
        google() // Google's Maven repository
        ...
    }
}
```

No site já clique em próxima e continuar no console.

Em <https://console.cloud.google.com/>, clique em **tela de consentimento OAuth** e **Editar aplicativo**.

Google Cloud Platform

chatflutter

Pesquisar produtos e recursos

API APIs e serviços

Tela de consentimento OAuth

Painel

Biblioteca

Credenciais

**Tela de consentimento OAuth**

Confirmação de domínio

Contratos de uso de página

project-954793226167

EDITAR APLICATIVO

**Status da verificação**

A verificação não é necessária

A tela de consentimento está sendo exibida, mas o aplicativo não foi revisado. Por isso, os usuários veem todas as suas informações, e não será possível solicitar certos escopos OAuth. [Saiba mais](#)

**Status de publicação**

Em produção

VOLTAR PARA TESTE

Copie o link em Domínios autorizados.

### Domínios autorizados

Quando um domínio é usado na tela de consentimento ou na configuração do cliente OAuth, ele precisa ser pré-registrado. Se o app precisa passar pela verificação, acesse o [Google Search Console](#) para verificar se os domínios estão autorizados. [Saiba mais](#) sobre o limite de domínio autorizado.

chatflutter-f3d44.firebaseio.com

[+ ADICIONAR DOMÍNIO](#)

Acrescente **https://** no link e cole nas três abas em Domínio do app.

### Domínio do app

Para proteger você e seus usuários, o Google permite apenas os apps que utilizam o OAuth a usar os domínios autorizados. As informações a seguir serão exibidas aos seus usuários na tela de consentimento.

Página inicial do aplicativo

https://chatflutter-f3d44.firebaseio.com

Forneça aos usuários um link para sua página inicial

Link da Política de Privacidade do aplicativo

https://chatflutter-f3d44.firebaseio.com

Forneça aos usuários um link para sua Política de Privacidade pública

Link dos Termos de Serviço do aplicativo

https://chatflutter-f3d44.firebaseio.com

Forneça aos usuários um link para seus Termos de Serviço públicos

## Comandos:

Serve tanto para leitura como para a escrita:

```
Firestore.instance.collection("col").document("doc").setData({"name": "jean"});
```

- **collection("Nome da coleção")** que será o nome da coleção salva no firebase;
- **document("Nome do documento que será salvo na coleção")** será o documento salvo, se não colocar nada o firebase gerará um id único;
- **setData({"Nome do campo": "valor"})** recebe um mapa e salva/modifica os valores no banco de dados;



- **`updateData({"Nome da variável que quer modificar": "Valor"})`** atualiza o valor da variável do mapa do documento já criado.

Pode-se fazer também uma coleção dentro de um documento como no exemplo abaixo:

```
Firestore.instance.collection("col").document("doc").collection("col2")
.document().setData({"texto": "jean"});
```

```
QuerySnapshot snapshot = await
Firestore.instance.collection("mensagens").getDocuments();
```

- **`getDocuments()`** devolve uma **`QuerySnapshot`** que é um conjunto de snapshots("fotos" dos documentos armazenados no banco de dados);

Para acessar a lista de documentos utilizar o **`documents`**, para acessar os dados de um documento basta utilizar o **`data`** e para saber o ID/nome do documento utilizar o **`documentID`**.

```
snapshot.documents
snapshot.documents[0].data
snapshot.documents[0].documentID;
```

Para obter um só documento utilizar o **`document`** para acessar um documento específico do banco de dados e a função **`get()`** com isso, devolverá uma **`DocumentSnapshot`** e para acessar o dados basta utilizar o **`data`** e o ID/nome utilizar o **`documentID`**.

```
DocumentSnapshot snapshot = await
Firestore.instance.collection("mensagens").document("nomeDoDocumento").
get();
snapshot.data;
snapshot.documentID;
```

Para acessar a referência do documento utilizar o **`reference`** e para atualizar os dados a função **`updateData`** citada acima.

```
Caso Query
snapshot.documents[0].reference.updateData({"Campo": "Valor"});
Caso Document
snapshot.reference.updateData({"Campo": "Valor"});
```

Para descobrir se houve alguma mudança nos documentos da coleção utiliza-se de **`snapshots`** que nota os documentos encontrados na coleção e a função **`listen`** que percebe as mudanças nos documentos e recebe como parâmetro um **`event`** que é **`QuerySnapshot`**, ou seja todos os documentos da coleção.

```
Firestore.instance.collection("mensagens").snapshots().listen((event) {
});
```

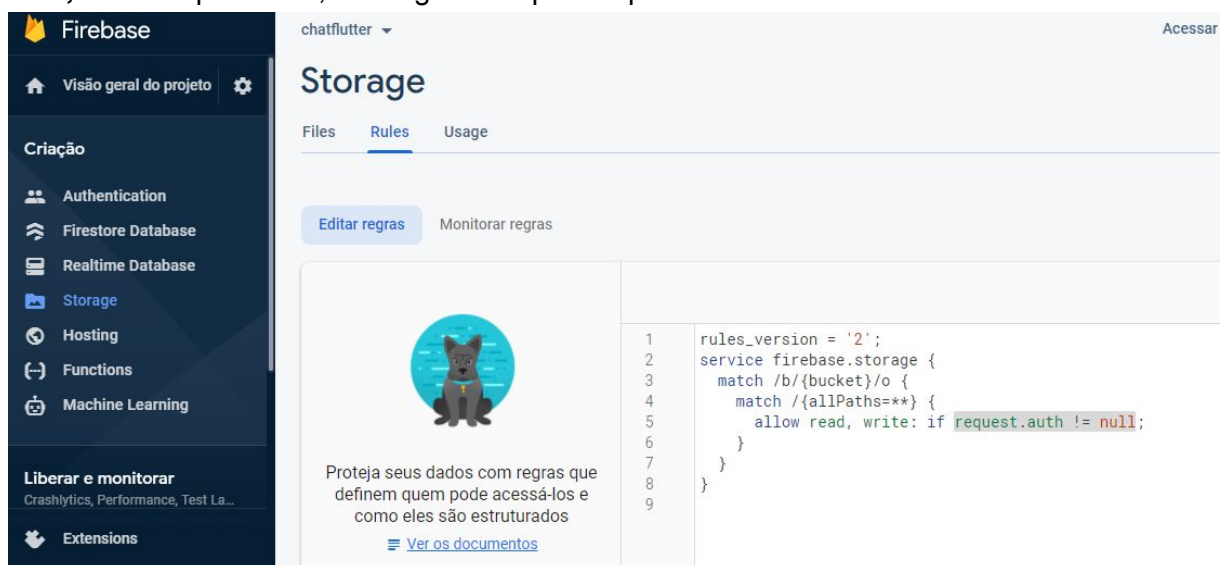
Já para o caso de verificar um único documento o **snapshots** irá notar o documento pesquisado e o **listen** irá fazer a mesma coisa que no caso anterior, porém com apenas o documento escolhido e o **event**, no caso, será uma **DocumentSnapshot**.

```
Firestore.instance.collection("mensagens").document("text").snapshots()
.listen((event) { });
```

Uma outra forma de criar um documento e adicionar um campo/valor é a através da função **add**. No caso abaixo foi criado uma coleção **messages** e uma ID específica para o novo documento e um campo **"text"** com um **valor** do campo.

```
Firestore.instance.collection("messages").add({"text": valor});
```

Para salvar imagens no firebase e testar envios a base de dados mude nas regras o **allow read, write: if** para **true**, em seguida clique em publicar.



Para obter um arquivo de imagem utiliza-se da função **pickImage** da **ImagePicker** que no caso será obtida após tirar uma foto da câmera pois utiliza-se **ImageSource.camera** caso queira da galeria utilizar a **ImageSource.gallery**.

```
final File imgFile = await ImagePicker.pickImage(source:
ImageSource.camera);
```

Para enviar um documento no FireBase Storage basta fazer uma tarefa de envio para ele. Com **FirestoreStorage.instance.ref()** será obtida a referência ao Firebase Storage e com a função **child** você poderá criar pasta no Firebase Storage e a última chamada desta função será o nome do arquivo colocado nele e a função **putFile** dirá qual arquivo deve ser colocado no Firebase Storage, que no exemplo é a imagem obtida da câmera citada acima.

```
StorageUploadTask task = FirestoreStorage.instance.ref().child("Nome da
pasta").child("Nome do arquivo").putFile(imgFile);
```

Com a função **onComplete** espera-se a tarefa de upar o arquivo e devolve uma **StorageTaskSnapshot** que seria uma “foto” tirada do arquivo enviado. E para obter a url para download do arquivo upado no Firebase Storage utiliza-se da função **getDownloadURL()** da **StorageTaskSnapshot**.

```
StorageTaskSnapshot taskSnapshot = await task.onComplete;
String url = await taskSnapshot.ref.getDownloadURL();
```

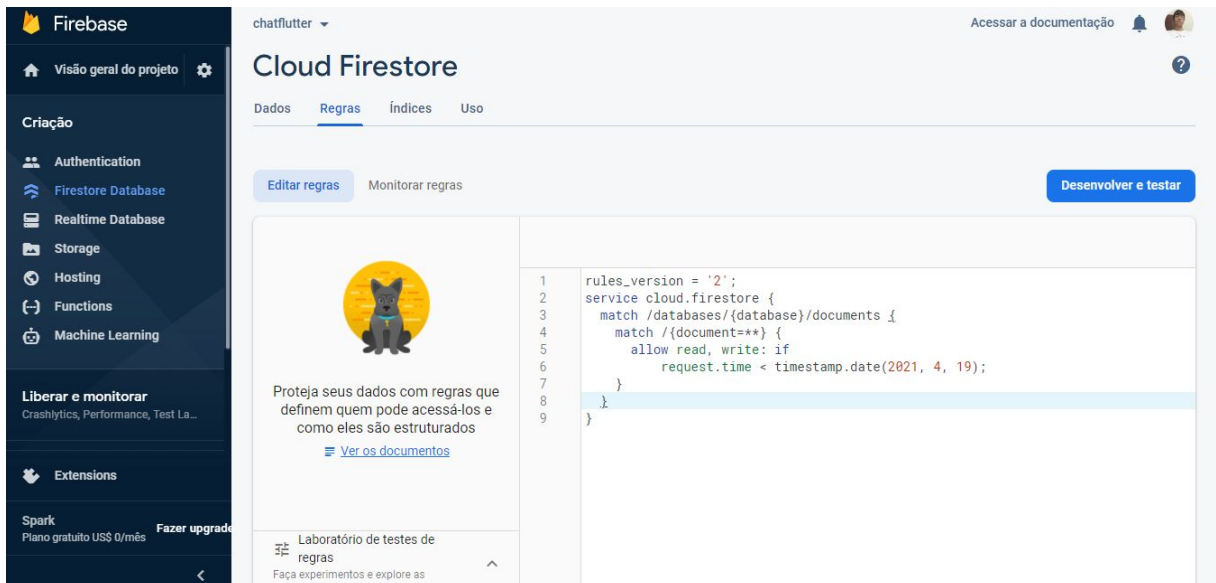
Para logar numa conta do Google e obtê-la utiliza-se do objeto **GoogleSignIn** que através da função **signIn** obtém-se a conta logada. Para obter os dados de autenticação basta utilizar a função **authentication** da **GoogleSignInAccount**. É na autenticação que possui os token de ID e acesso para acessar o login do Firebase. Para obter as credenciais para logar no firebase utiliza-se da função **getCredential**, que no exemplo é de uma conta google por tanto é da **GoogleAuthProvider** e basta passar a esse função o token ID e de acesso que podem ser obtidos pela chamada **idToken** e **accessToken** da **GoogleSignInAuthentication**. Para logar então no FireBase basta utilizar a função **signInWithCredential** da **FirebaseAuth.instance** e colocar a credencial obtida. E para obter a conta basta usar a chamada **user** da **AuthResult**. E através do **initState** saberá se o usuário já está logado através da função **listen** da **FirebaseAuth.instance.onAuthStateChanged** (esta última nota se está logado ou deslogado) que verificará todas as vezes que logar ou deslogar, devolvendo nulo ou o usuário logado.

```
FirebaseUser _currentUser;

@override
void initState() {
  // TODO: implement initState
  super.initState();
  FirebaseAuth.instance.onAuthStateChanged.listen((user) {
    _currentUser = user;
  });
}

Future<FirebaseUser> _getUser() async {
  if (_currentUser != null) return _currentUser;
  try {
    final GoogleSignInAccount googleSignInAccount =
      await googleSignIn.signIn();
    final GoogleSignInAuthentication googleSignInAuthentication =
      await googleSignInAccount.authentication;
    final AuthCredential credential =
      GoogleAuthProvider.getCredential(
        idToken: googleSignInAuthentication.idToken,
        accessToken: googleSignInAuthentication.accessToken);
    final AuthResult authResult =
      await FirebaseAuth.instance.signInWithCredential(credential);
    final FirebaseUser user = authResult.user;
    return user;
  } catch (error) {
    return null;
  }
}
```

# Regras Firebase



A palavra **match** determina uma regra para uma parte do seu banco de dados. Exemplo: **match /{document=\*\*}** diz que essa regra vale para todos os documentos do banco de dados. Já a palavra **request** é a requisição que o usuário faz.

- **allow read: if** condição - determina a regra de leitura;
- **allow create: if** condição - determina a regra para criar documentos
- **allow update, delete: if** condição - determina a regra para atualizar os documentos e deleta-los.

Exemplo:

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    match /{document=**} {
      allow read: if true;
      allow create: if request.auth!=null;
      allow update, delete: if request.auth!=null && request.auth.uid == resource.data.uid;
    }
  }
}
```

No exemplo, a regra é dada a todos os documentos. A leitura é permitida a todas as pessoas, a criação de documentos é permitida a apenas as pessoas autenticadas e a deleção e atualização de dados apenas se a pessoa que solicitou a mudança tem o mesmo ID da pessoa que enviou o recurso (**resource**).