



Firestore Notification



Objetivos

exemploautenticacao-c2c24

Permite enviar notificações de novas mensagens entre os usuários do Chat.













Configurando o Ambiente



Assistant  



Firebase gives you the tools and infrastructure from Google to help you develop, grow and earn money from your app. [Learn more](#)



- ▶  **Analytics**
Measure user activity and engagement with free, easy, and unlimited analytics. [More info](#)
- ▶  **Cloud Messaging**
Deliver and receive messages and notifications reliably across cloud and device. [More info](#)
- ▶  **Authentication**
Sign in and manage users with ease, accepting emails, Google Sign-In, Facebook and other login providers. [More info](#)
- ▶  **Realtime Database**
Store and sync data in realtime across all connected clients. [More info](#)
- ▶  **Storage**
Store and retrieve large files like images, audio, and video without writing server-side code. [More info](#)
- ▶  **Remote Config**
Customize and experiment with app behavior using cloud-based configuration parameters. [More info](#)
- ▶  **Test Lab**
Test your apps against a wide range of physical devices hosted in Google's cloud. [More info](#)
- ▶  **Crash Reporting**
Get actionable insights and reports on app crashes, ANRs or other errors. [More info](#)
- ▶  **App Indexing**
Get your app content into Google Search. [More info](#)
- ▶  **Dynamic Links**
Create web URLs that can be shared to drive app installs and deep-linked into relevant content of your app. [More info](#)

No painel lateral do *Assistente do Firebase*, selecione **Cloud Messaging**

Assistant  

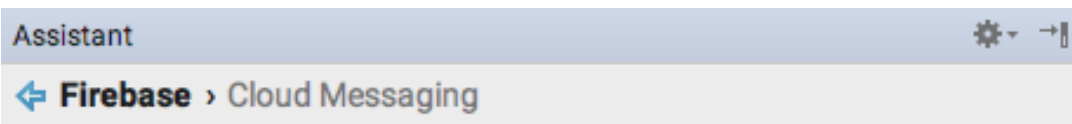


Firebase gives you the tools and infrastructure from Google to help you develop, grow and earn money from your app. [Learn more](#)

- ▶  **Analytics**
Measure user activity and engagement with free, easy, and unlimited analytics. [More info](#)
- ▼  **Cloud Messaging**
Deliver and receive messages and notifications reliably across cloud and device. [More info](#)
[Set up Firebase Cloud Messaging](#)

E clique no link **Set up Firebase Cloud Messaging**

Configurando o Ambiente



Set up Firebase Cloud Messaging

Firebase Cloud Messaging lets you receive and send messages from your app to the server and other clients. This tutorial explains how to set up FCM and enable your app to receive notifications.

[Launch in browser](#)

1 Connect your app to Firebase

OK Connected

2 Add FCM to your app

Add FCM to your app

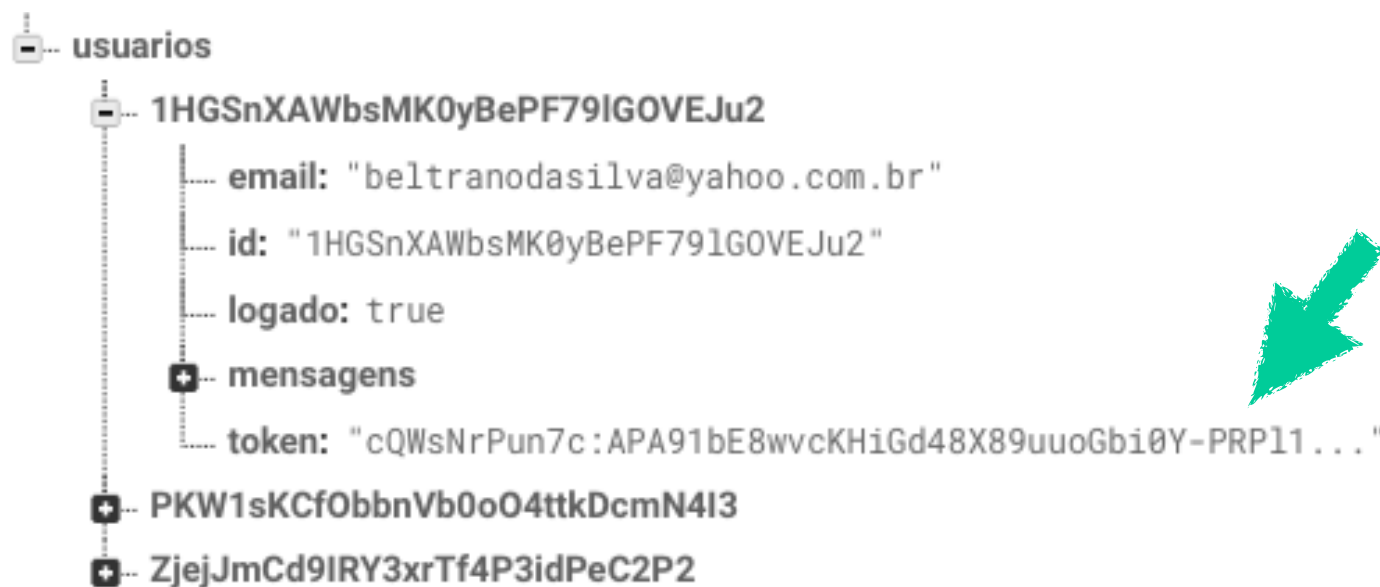
Agora adicione a configuração do Firebase na aplicação clicando no botão **Add FCM to your app**

E em seguida clique no botão **Accept Changes** deste diálogo para configurar a aplicação



Configurando o Ambiente

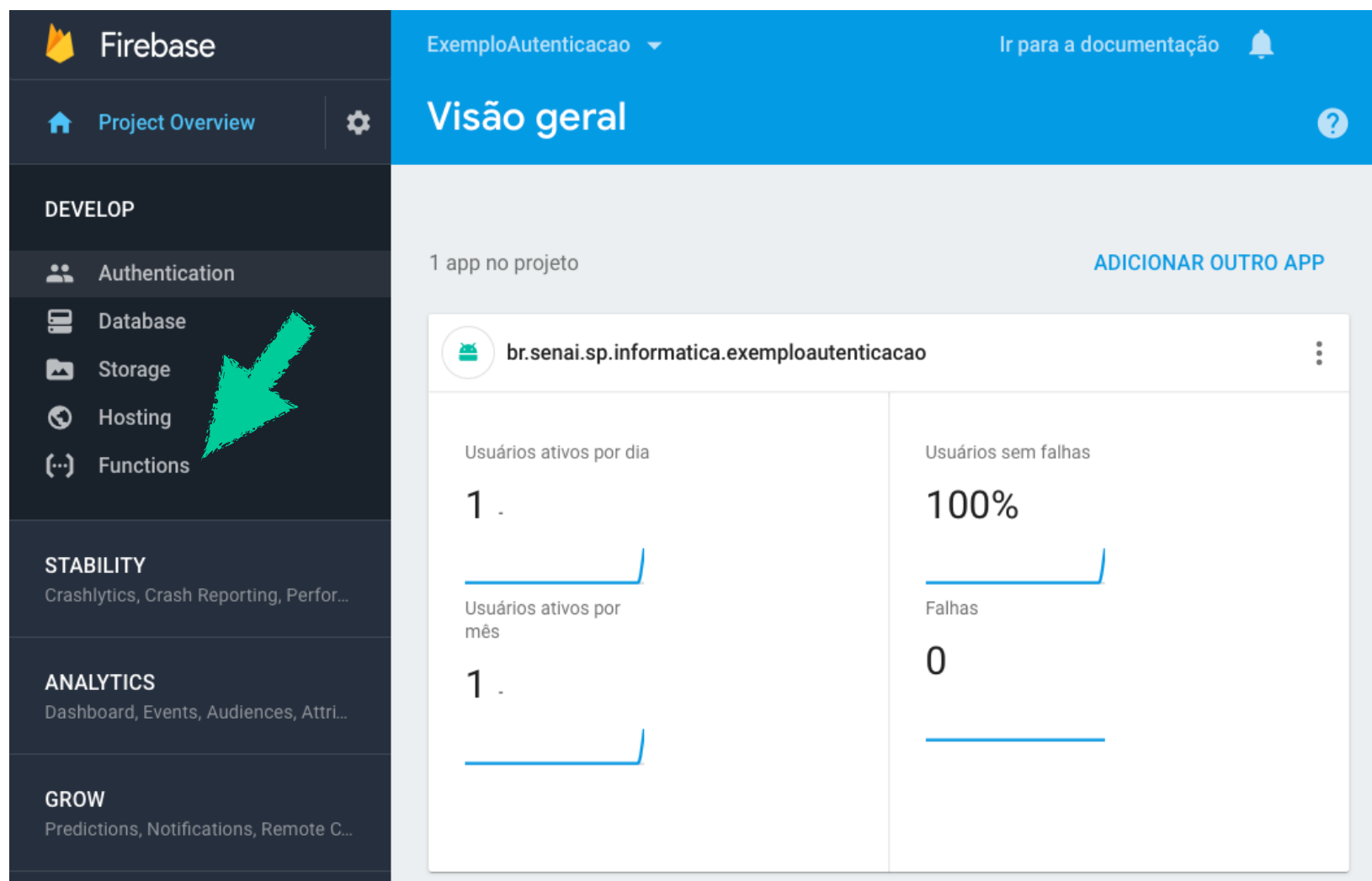
Com o Firebase *Cloud Messaging* uma aplicação Android passa a ser capaz de obter o **Token**, identificação do celular para viabilizar o recebimento de notificações.



Para o envio de notificações aos celulares, identificados pelos *Tokens*, é necessário a implementação de lógica (*function*) que será acionada a partir de evento de alteração na estrutura de mensagens.

Configurando o Ambiente

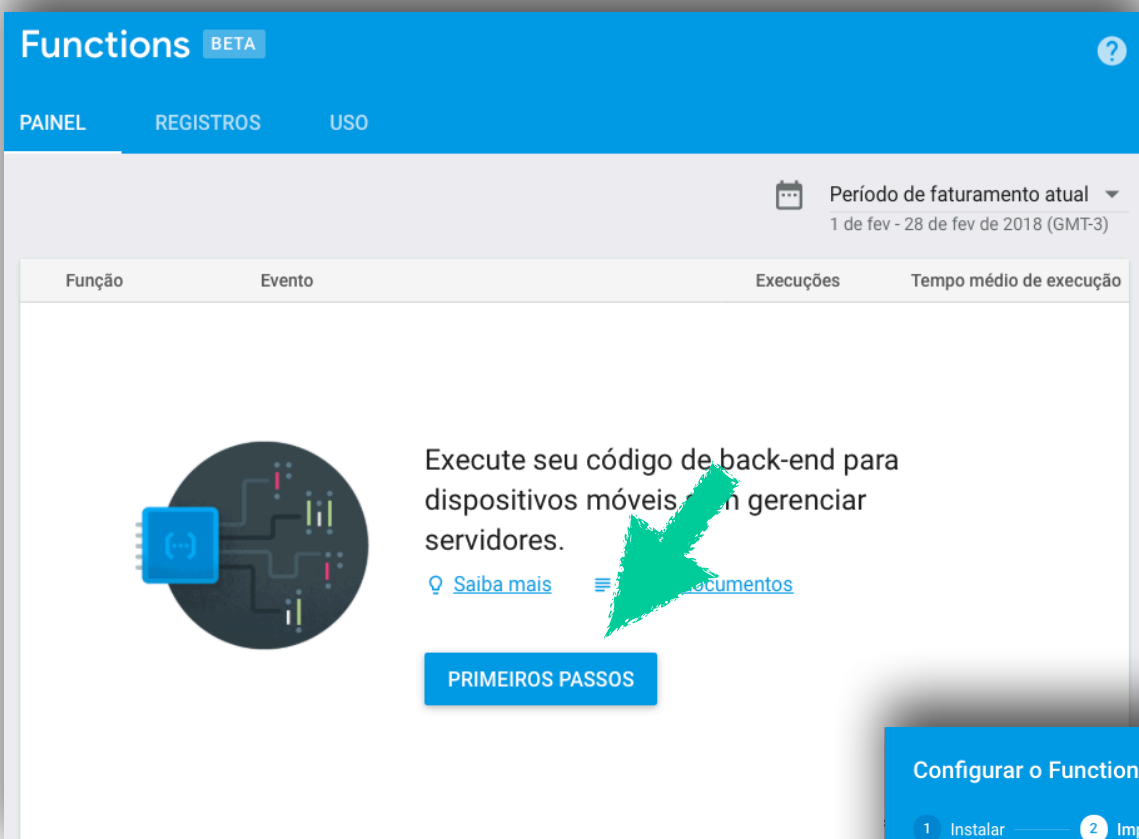
Na console do Firebase, selecione **Functions** na coluna a esquerda da tela.



The screenshot shows the Firebase console interface. On the left sidebar, under the 'DEVELOP' section, the 'Functions' option is highlighted with a red arrow. The main content area displays the 'Visão geral' (Overview) for the project 'ExemploAutenticacao'. It shows '1 app no projeto' and provides a link to 'ADICIONAR OUTRO APP'. The dashboard includes four metrics:

- Usuários ativos por dia: 1
- Usuários ativos por mês: 1
- Usuários sem falhas: 100%
- Falhas: 0

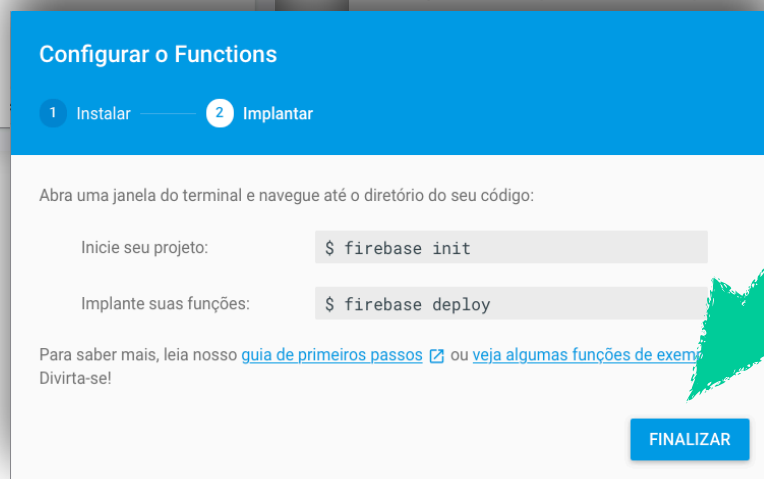
Configurando o Ambiente



Após a configuração da aplicação acesse o **Firebase console** e em Functions clique no botão **Primeiros Passos**



Nos diálogos que se seguem é apresentado os passos necessários à configuração do ambiente de desenvolvimento do **Cloud Functions**



Configurando o Ambiente



You're about to initialize a Firebase project in this directory:

`/Area/Cursos/Contas/web/Node/ExemploFuncao`

Before we get started, keep in mind:

- You are currently outside your home directory

? Which Firebase CLI features do you want to setup for this folder? Press Space to select features, then Enter to confirm your choices.

- ☐ Database: Deploy Firebase Realtime Database Rules
- ☐ Firestore: Deploy rules and create indexes for Firestore
- ☒ Functions: Configure and deploy Cloud Functions
- ☐ Hosting: Configure and deploy Firebase Hosting sites
- ☐ Storage: Deploy Cloud Storage security rules

Console de configuração do
Cloud Function

Configurando o Ambiente



You're about to initialize a Firebase project in this directory:

`/Area/Cursos/Contas/web/Node/ExemploFuncao`

Before we get started, keep in mind:

- You are currently outside your home directory

? Which Firebase CLI features do you want to setup for this folder? Press Space to select features, then Enter to confirm your choices. Functions: Configure and deploy Cloud Functions

=== Project Setup

First, let's associate this project directory with a Firebase project. You can create multiple project aliases by running `firebase use --add`, but for now we'll just set up a default project.

? Select a default Firebase project for this directory: `ExemploAutenticacao (exemploautenticacao-c2c24)`

=== Functions Setup

A `functions` directory will be created in your project with a Node.js package pre-configured. Functions can be deployed with `firebase deploy`.

? What language would you like to use to write Cloud Functions? `JavaScript`

? Do you want to use ESLint to catch probable bugs and enforce style? `Yes`

✓ Wrote `functions/package.json`

✓ Wrote `functions/.eslintrc.json`

✓ Wrote `functions/index.js`

? Do you want to install dependencies with npm now? `Yes`

Console de configuração do
Cloud Function

Configurando o Ambiente

```
const functions = require('firebase-functions');
const admin = require('firebase-admin');
admin.initializeApp(functions.config().firebase);

exports.messageNotification = functions.database.instance('exemploautenticacao-c2c24')
.ref('/mensagens/{destinatarioId}/{mensagemId}/{mensagem}').onCreate((event) => {
  const usuarioId = event.data.child('origem').val();
  const destinatarioId = event.params.destinatarioId;
  if(usuarioId === destinatarioId) { return 'no message'; }
  const getTokenPromise = admin.database().ref(`/usuarios/${destinatarioId}/token`).once('value');
  const getEmailPromise = admin.database().ref(`/usuarios/${usuarioId}/email`).once('value');
  return Promise.all([getTokenPromise, getEmailPromise]).then((results) => {
    const token = results[0];
    const email = results[1];
    if (!token.exists()) { return 'error no token'; }
    const payload = {
      notification: {
        title: 'Tem mensagem para Você!',
        body: `${email.val()} enviou mensagens à Você.`
      },
      data: {
        usuarioId: `${usuarioId}`,
        destinatarioId: `${destinatarioId}`,
        remetente: `${email.val()}`
      }
    };
    return admin.messaging().sendToDevice(token.val(), payload);
  }).then(() => { return 'ok'; })
  .catch((error) => { console.error('Falha ao enviar notificação a', error); });
});
```

O *Cloud Function* utiliza *JavaScript* como linguagem padrão.

A função deve ser enviada ao *Firebase* através do comando `firebase deploy`

Configurando o Ambiente

```
=== Deploying to 'exemploautenticacao-c2c24'...
```

```
i  deploying functions
```

```
Running command: npm --prefix $RESOURCE_DIR run lint
```

```
> functions@ lint /Area/Cursos/Contas/web/Node/ExemploFuncao/functions
```

```
> eslint .
```

```
✓ functions: Finished running predeploy script.
```

```
i  functions: ensuring necessary APIs are enabled...
```

```
✓ functions: all necessary APIs are enabled
```

```
i  functions: preparing functions directory for uploading...
```

```
i  functions: packaged functions (39.93 KB) for uploading
```

```
✓ functions: functions folder uploaded successfully
```

```
i  functions: creating function messageNotification...
```

```
✓ functions[messageNotification]: Successful create operation.
```

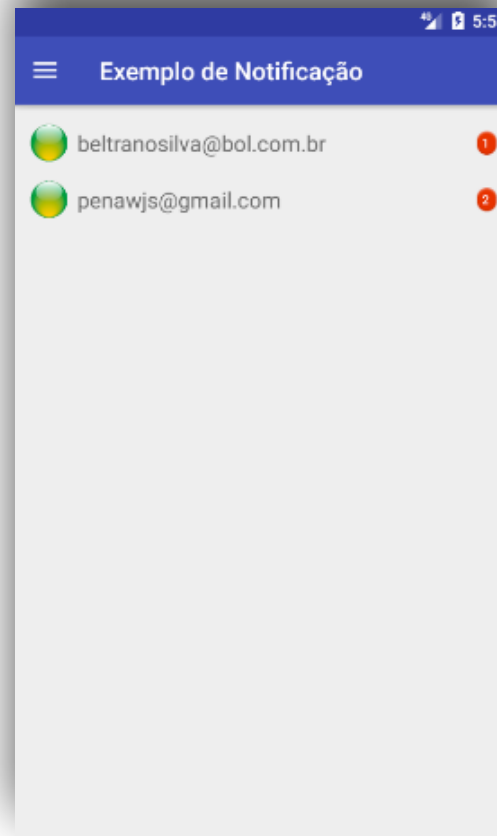
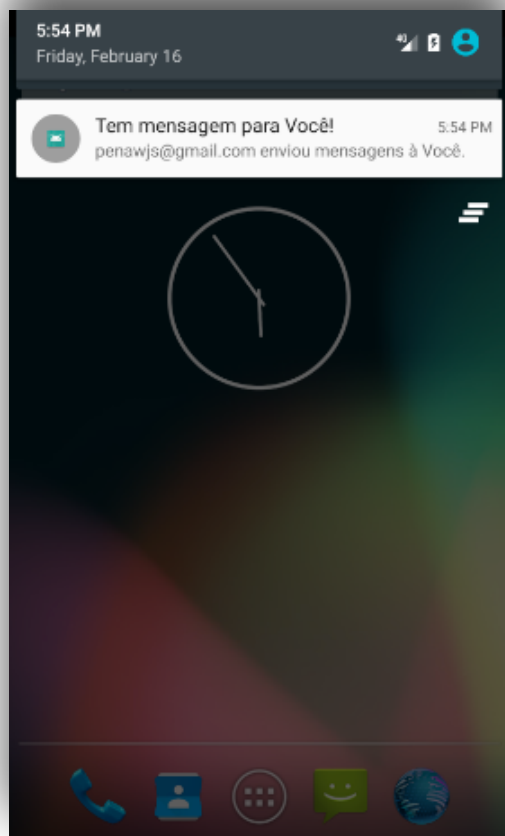
```
✓ Deploy complete!
```

```
Project Console: https://console.firebase.google.com/project/exemploautenticacao-c2c24/overview
```

Publicação de uma função
no *Firebase*

A Aplicação

O Celular apresentará uma notificação quando a aplicação não estiver ativa e terá indicações de novas mensagens quando a mensagem for direcionada para um usuário que não estiver em Chat no momento em que a esta chegar.



Estrutura de Dados

exemploautenticacao-c2c24

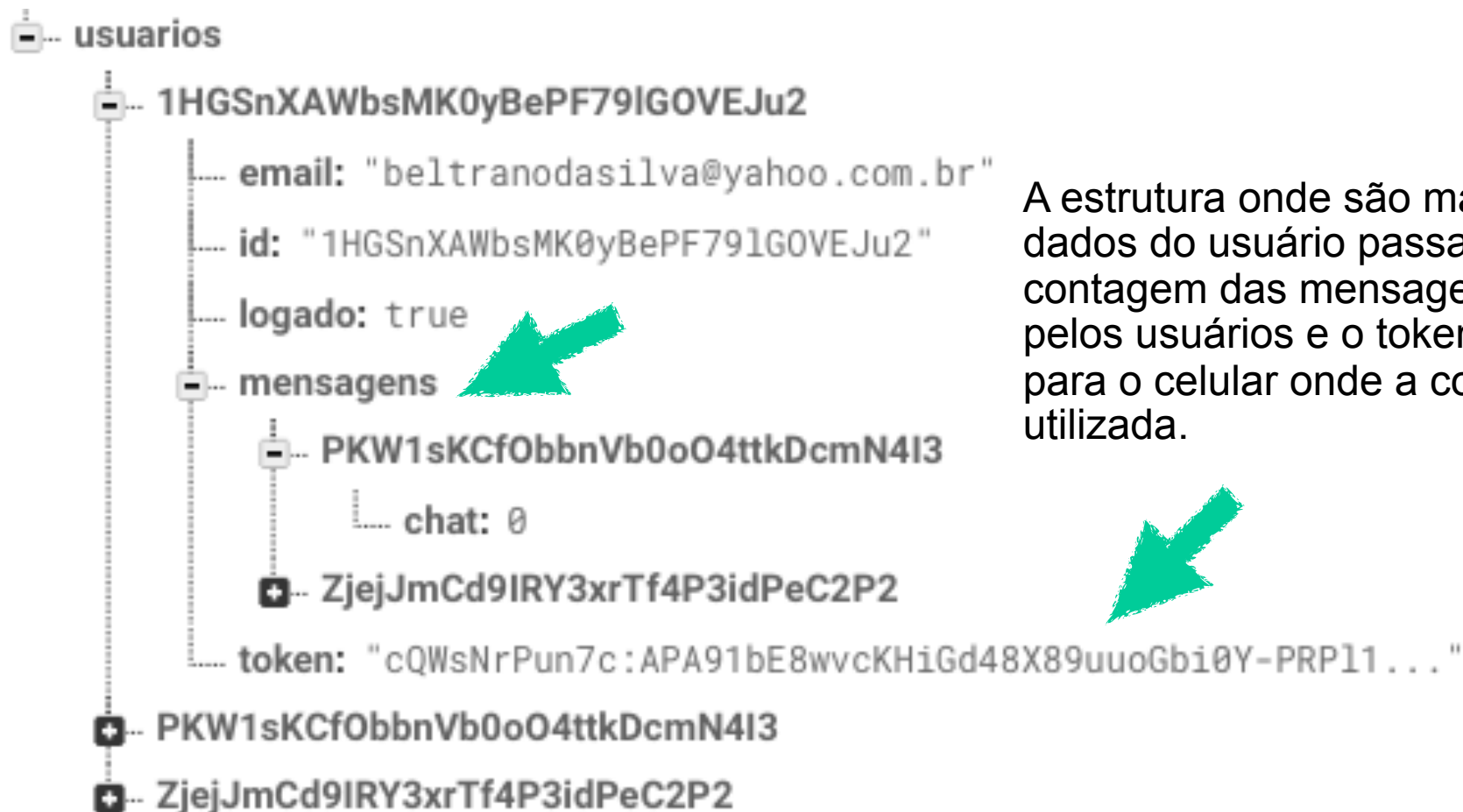
```

- mensagens
  - 1HGSnXAWbsMK0yBePF79lGOVEJu2
    + PKW1sKCfObbnVb0oO4ttkDcmN4I3
    - ZjejJmCd9lRY3xrTf4P3idPeC2P2
      - -L5W0VLHwpwvQXUKTLn_
        data: 1518824981678
        id: "-L5W0VLHwpwvQXUKTLn_"
        mensagem: "teste"
        origem: "1HGSnXAWbsMK0yBePF79lGOVEJu2"
```

A estrutura das mensagens foi readequada para possibilitar que cada mensagem tenha a identificação do usuário que a originou.



Estrutura de Dados





A estrutura onde são mantidos os dados do usuário passa a manter a contagem das mensagens recebidas pelos usuários e o token gerado para o celular onde a conta foi utilizada.


```
public void salvar(Usuario obj, DatabaseReference.CompletionListener callback) {  
    DatabaseReference ref = getReference().child(getUserId());  
    ref.child("id").setValue(obj.getId());  
    ref.child("email").setValue(obj.getEmail());  
    ref.child("token").setValue(obj.getToken());  
    ref.child("logado").setValue(obj.isLogado());  
}
```

Os métodos **salvar()** e **atualizaLogon()** foram modificados para manter o **Token**.

```
public void atualizaLogon(boolean logado) {  
    DatabaseReference ref = getReference().child(getUserId());  
    ref.child("logado").setValue(logado);  
  
    if (!logado) {  
        ref.child("token").setValue(null);  
    }  
}
```

```
public class Main extends Application {  
    private void offline() {  
        SharedPreferences preferences = PreferenceManager.getDefaultSharedPreferences( context: this);  
        SharedPreferences.Editor editor = preferences.edit();  
        editor.putString("status", "offline");  
        editor.apply();  
    }  
  
    @Override  
    public void onTerminate() {  
        offline();  
        super.onTerminate();  
    }  
  
    @Override  
    public void onLowMemory() {  
        offline();  
        super.onLowMemory();  
    }  
}
```




A classe **Main** foi construída para capturar os eventos de finalização da aplicação e forçar o **status** de **offline** para a aplicação.

O **status** é utilizado para determinar se será enviada notificação do recebimento de mensagem.

A Mensagem

```
public void salvar(Mensagem obj, DatabaseReference.CompletionListener callback) {  
    if(obj.getId() == null) {  
        obj.setId(reference.push().getKey());  
    }  
  
    Map<String, Object> map = new HashMap<>();  
    map.put("id", obj.getId());  
    map.put("mensagem", obj.getMensagem());  
    map.put("data", obj.getData());  
    map.put("origem", user.getId());  
  
    Map<String, Object> updates = new HashMap<>();  
    updates.put(makeReference1(obj.getId()), map);  
    updates.put(makeReference2(obj.getId()), map);  
  
    base.updateChildren(updates, callback);  
}
```

O método **salvar()** da classe **MensagemDao** foi alterada para atribuir o ID de quem originou a mensagem.



O InstanceIDService

A classe **FirebaseInstanceIdService** deve ser construída para manter o **Token** caso este seja alterado por algum evento, tais como:

- o aplicativo exclui o código da instância;
- o aplicativo é restaurado em um novo dispositivo;
- o usuário desinstala/reinstala o app;
- o usuário limpa os dados do app.

```
public class InstanceService extends FirebaseInstanceIdService {  
    @Override  
    public void onTokenRefresh() {  
        String refreshedToken = FirebaseInstanceId.getInstance().getToken();  
  
        SharedPreferences preferences = PreferenceManager.getDefaultSharedPreferences(context, this);  
        SharedPreferences.Editor editor = preferences.edit();  
        editor.putString("TokenId", refreshedToken);  
        editor.apply();  
    }  
}
```

O MessagingService

```
public class MessageService extends FirebaseMessagingService {
    @Override
    public void onMessageReceived(RemoteMessage remoteMessage) {
        final String TAG = "MessageService";

        // Aciona o Receiver de Notificação
        Intent intent;

        int currentapiVersion = android.os.Build.VERSION.SDK_INT;
        if (currentapiVersion >= Build.VERSION_CODES.O) {
            intent = new Intent(getBaseContext(), NotificationReceiver.class);
            intent.setAction("br.senai.sp.informatica.exemploautenticacao.NOTIFICA_MENSAGEM");
        } else {
            intent = new Intent( action: "br.senai.sp.informatica.exemploautenticacao.NOTIFICA_MENSAGEM");
        }

        intent.putExtra( name: "chatUser", remoteMessage.getData().get("usuarioId"));
        intent.putExtra( name: "msg", remoteMessage.getNotification().getBody());
        sendBroadcast(intent);

        Map<String, String> data = remoteMessage.getData();
        if (data.size() > 0) {
            for (String key : data.keySet()) {
                Log.e(TAG, msg: key + " : " + data.get(key));
            }
        }

        if (remoteMessage.getNotification() != null) {
            Log.e(TAG, msg: "Message Notification Body: " + remoteMessage.getNotification().getBody());
        }
    }
}
```

A classe **FirebaseMessagingService** deve ser construída para o recebimento das notificações enviadas pelo **Cloud Function**.

O BroadcastReceiver

A classe **NotificationReceiver** é um **BroadcastReceiver** que tem a capacidade de ser acionado através de **Intents** e como não tem *Layout*, executa sua função em background.

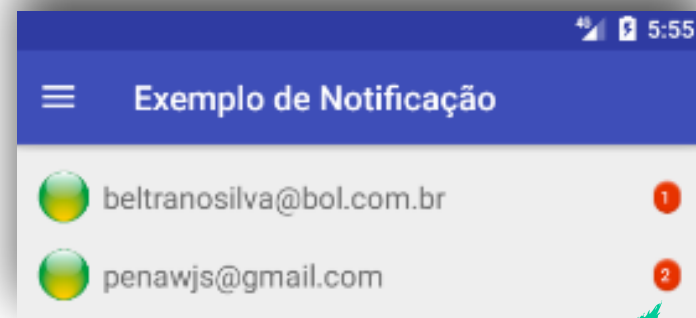
Esta classe tem a responsabilidade de manter os contadores de mensagens que o usuário recebe e que é apresentada na lista de usuários.

```
public class NotificationReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        Bundle extras = intent.getExtras();
        if(extras != null) {
            String chatUser = extras.getString( key: "chatUser");
            String msg = extras.getString( key: "msg");

            SharedPreferences preferences = PreferenceManager.getDefaultSharedPreferences(context);
            String status = preferences.getString( key: "status", defValue: "offline");
            String chatCom = preferences.getString( key: "chatCom", defValue: "");

            Log.e( tag: "NotificationReceiver", msg: "status: " + status);
            Log.e( tag: "NotificationReceiver", msg: "chat com: " + chatCom);

            if (status.equals("online")) {
                if (!chatCom.equals(chatUser)) {
                    UsuarioDao.dao.incrementaMensagens(chatUser);
                }
            }
        }
    }
}
```



O AndroidManifest

Todos os *Services* e *Receivers* devem ser registrados no **AndroidManifest.xml**

```
<service
    android:name=".service.InstanceService"
    tools:ignore="ExportedService">
    <intent-filter>
        <action android:name="com.google.firebase.INSTANCE_ID_EVENT"/>
    </intent-filter>
</service>

<service
    android:name=".service.MessageService"
    tools:ignore="ExportedService">
    <intent-filter>
        <action android:name="com.google.firebase.MESSAGING_EVENT"/>
    </intent-filter>
</service>

<receiver android:name=".receiver.NotificationReceiver"
    android:exported="false">
    <intent-filter>
        <action android:name="br.senai.sp.informatica.exemploautenticacao.NOTIFICA_MENSAGEM"/>
    </intent-filter>
</receiver>
```

O FirebaseArray

Devido a limitação das Queries que podemos utilizar no Firebase, foi necessário a construção de uma classe para suprir a necessidade de manter a lista dos usuários com todas as modificações que possam ocorrer ao longo da execução da aplicação, mas garantindo que as informações do usuário logado nunca aparecerá.

Esta classe é a **UsuarioCharArray** descendente de **FirestoreArray**.

```
public class UsuarioCharArray extends FirestoreArray<Usuario> {  
    private UsuarioDao dao = UsuarioDao.dao;  
  
    public UsuarioCharArray() {  
        super(UsuarioDao.dao.getReference(),  
            new SnapshotParser<Usuario>() {  
                public Usuario parseSnapshot(@NonNull DataSnapshot snapshot) {  
                    return snapshot.getValue(Usuario.class);  
                }  
            }  
        );  
    }  
}
```


O UsuarioCharArray

```
public void onChildAdded(DataSnapshot snapshot, String previousChildKey) {
    new EventManager() {
        @Override
        public void superEvent(DataSnapshot snapshot, String previousChildKey) {
            UsuarioCharArray.super.onChildAdded(snapshot, previousChildKey);
        }
    }.processEvent(snapshot, previousChildKey);
}

@Override
public void onChildChanged(DataSnapshot snapshot, String previousChildKey) {
    new EventManager() {
        @Override
        public void superEvent(DataSnapshot snapshot, String previousChildKey) {
            UsuarioCharArray.super.onChildChanged(snapshot, previousChildKey);
        }
    }.processEvent(snapshot, previousChildKey);
}

@Override
public void onChildMoved(DataSnapshot snapshot, String previousChildKey) {
    new EventManager() {
        @Override
        public void superEvent(DataSnapshot snapshot, String previousChildKey) {
            UsuarioCharArray.super.onChildMoved(snapshot, previousChildKey);
        }
    }.processEvent(snapshot, previousChildKey);
}
```

A classe **UsuarioCharArray** tem sua estrutura adaptada para tratar os vários eventos que ocorrem em um **ListView**.

O EventManager

Todos os métodos da classe **UsuarioCharArray** fazem uso da implementação da classe abstrata **EventManager**, criada para propiciar a adequada implementação de todos os eventos associados ao **FirestoreArray**.

```
private abstract class EventManager {  
    private String chavePreviaAdicionada;  
  
    protected void processEvent(DataSnapshot snapshot, String previousChildKey) {  
        if (previousChildKey == null) chavePreviaAdicionada = null;  
  
        if (!snapshot.getKey().equals(dao.getUserId())) {  
            superEvent(snapshot, chavePreviaAdicionada);  
            chavePreviaAdicionada = snapshot.getKey();  
        }  
    }  
  
    public abstract void superEvent(DataSnapshot snapshot, String previousChildKey);  
}
```


FIM