Payload de notificação para nova solicitação.

1 **static** createSolicitation(solicitationUid: string,

2 contractorName: any): any {

3 **const** payload = {

4 data: {

5 title: "new\_solicitation",

6 body: solicitationUid,

7 },

8 notification: {

9 title: "Solicitação de serviço",

10 body: contractorName.firstName + " " +

11 contractorName.lastName +

12 " te enviou uma solicitação de serviço!"

13 }

14 };

15

16 **return** payload;

17 }

Payload de notificação para avaliações.

1 **static** createAvaliation(avaliationUid: string,

2 profile: Profile) {

3 var userFullName = profile.name.firstName +

4 " " + profile.name.lastName;

5

6 **const** payload = {

7 data: {

8 title: "new\_avaliaiton",

9 body: avaliationUid,

10 },

11 notification: {

12 title: "Avaliação recebida",

13 body: "Você recebeu uma avaliação de " +

14 userFullName

15 }

16 };

17

18 **return** payload;

19 }

Cloud Functions.

Para informar os usuários do aplicativo de que receberam solicitações ou avaliações, foi necessário criar um sistema de notificações, no qual foi usado duas ferramentas do *Firebase*, o *cloud functions* e o *cloud messaging*.

O *cloud functions* é um ambiente de execução sem servidores para conectar com serviços em nuvem. Onde é se escreve um código simples e de uso único e são acionados assim que algum evento da infraestrutura do *firebase* é acionado. Tudo isso em um ambiente gerenciado, sem a necessidade de gerenciar alguma infraestrutura ou servidores.

O *cloud messaging* é um serviço do firebase responsável por enviar mensagens a para um grupo de usuários ou para alguém em especifico, desde que estejam autenticados na infraestrutura do firebase. No caso deste projeto, usuários cadastrados no aplicativo receberão as mensagens, onde a cada usuário tem um *token* de acesso único, que é alterado quando o mesmo realiza *login* e *logout* no aplicativo. Este *token* é usado para identificar cada usuário do aplicativo e lhe enviar mensagens, as quais podem ser as notificações de solicitações de serviços ou avaliações.

Para configurar o ambiente de desenvolvimento com o *cloud functions* foi simples, para isso executamos o seguinte comando no terminal: **npm install -g firebase-tools**. Este comando instala todos os arquivos necessários para criação de um projeto para o *cloud functions*.

O passo seguinte foi realizar o login no firebase pelo terminal, usando o comando: **firebase login**. Após realizar o login no firebase dentro do terminal, executamos mais um comando: **firebase init functions**. Este comando cria uma estrutura para trabalharmos com *cloud functions*, durante a sua execução o firebase nos deu as opções de criar um novo projeto ou usar um já existente, neste caso, foi listado o projeto Tips, que já estava configurado no firebase.

Assim que selecionamos o projeto que irá utilizar o *cloud functions,* o firebase ainda nos possibilita de escolher a linguagem a de programação a ser usada e a instalação de pacotes para verificar a qualidade do código desenvolvido.

Enviando notificações

Para enviar notificações aos usuários com o *cloud functions* foi necessário a criação de determinados métodos que são acionados assim que algum evento do *firebase* é acionado. Como as notificações deste projeto são enviadas quando uma nova solicitação de serviço é feita ou atualizada e em avaliações, utilizamos apenas os eventos que são chamados quando alguma criação ou atualização de dados no banco de dados é feita.

Quando uma nova solicitação de serviço é criada, um evento de criação de dado é disparado e o Código XX é executado.

**Código XX:** Código executado após a criação de uma solicitação de serviço.

1 **export** **function** onSolicitationCreate() {

2 **return** functions.firestore.document(Constants.SOLICITATION\_COLLECTION

3 + SOLICITATION\_PARAMS\_UID)

4 .onCreate((snap: any) => {

5 **const** data = snap.data();

6 **if** (data !== **undefined**) {

7 **const** solicitation = solicitationParse(data);

8 **return** requestProfilesForSolicitations(solicitation,

9 solicitation.hiredUid);

10 } **else** {

11 **return** **null**;

12 }

13 });

14 }

**Fonte:** Elaborado pelos autores (2019)

Na linha 8 é chamado o método *requestProfilesForSolicitations*, no qual irá requisitar os perfis dos usuários envolvidos na solicitação e enviar a notificação para o profissional. Este método pode ser visto no Código XX.

**Código XX:** Requisitando os perfis e notificando o usuário.

1 export function requestProfilesForSolicitations(

2 solicitation: Solicitation, pfUid: string,

3 isAUpdate: boolean = **false**): any {

4 **return** getProfilesToBuildNotification(solicitation, pfUid)

5 .then((profiles: Array<Profile>) => {

6 **return** notifyUserOnSolicitation(solicitation,

7 profiles, isAUpdate);

8 })

9 .catch(() => {

10 **return** null;

11 });

12 }

**Fonte:** Elaborado pelos autores (2019)

A requisição dos perfis é executada no método *getProfilesToBuildNotification*, na linha 4, no qual busca os perfis baseados nos campos *contractorUid* e *hiredUid* do objeto *solicitation*, enviado como parâmetro, que é a solicitação recém-criada no banco de dados. Concluído com sucesso, o método na linha 6, *notifyUserOnSolicitation* é chamado, correspondente ao Código XX a seguir.

**Código XX:** Criando a notificação de solicitação para o usuário.

1 **export** **function** notifyUserOnSolicitation(

2 solicitation: Solicitation, profiles: Profile[],

3 isAUpdate: **boolean** = **false**) {

4 **var** payload: any;

5

6 **if** (!isAUpdate) {

7 payload =

8 NotificationBuilder.createSolicitation(

9 solicitation.solicitationId,

10 profiles[PROFILES.PROFILE\_TO\_SHOW\_ON\_NOTIFICATION].name);

11 } **else** {

12 payload =

13 NotificationBuilder.updateSolicitation(

14 solicitation,

15 profiles[PROFILES.PROFILE\_TO\_SHOW\_ON\_NOTIFICATION]);

16 }

17

18 **return** NotificationSender.sendNotification(

19 profiles[PROFILES.PROFILE\_TO\_NOTIFY].deviceToken,

20 payload);

21 }

**Fonte:** Elaborado pelos autores (2019)

Este método é utilizado para enviar a notificação aos usuários, ele também é utilizado para envio de notificações quando uma solicitação de serviço é alterada, por exemplo na aprovação ou termino de uma solicitação, e envia uma notificação para o usuário especifico.

A classe *NotifiationBuilder* é responsável somente por criar os *payloads* de notificações, na linha 8 é realizado a criação do *payload* de notificação para novas solicitações, e na linha 13 para atualizações de solicitações. A notificação é enviada no método *sendNotification*, na linha 18, ele usa o campo *deviceToken* salvo no perfil do usuário e o payload da notificação, conforme o Código XX, usando o *cloud messaging* através do método *messaging.*

**Código XX:** Enviando notificação.

1 static async sendNotification(deviceToken: string,

2 payload: any): Promise<any> {

3 **return** admin.messaging()

4 .sendToDevice(deviceToken, payload)

5 }

**Fonte:** Elaborado pelos autores (2019)

No caso das avaliações, seguimos a mesma lógica, o Código XX e YY são executados quando uma avaliação é criada ou atualizada, respectivamente. Ambos chamam o método *avaliationHandler* que realiza o fluxo para notificar os usuários envolvidos.

**Código XX:** Código executado após uma avaliação é criada no banco de dados.

1 **export** **function** onAvaliatedCreated() {

2 **return** functions.firestore.document(

3 Constants.AVALIATIONS\_COLLECTION + avaliationUidParams)

4 .onCreate(async (snap: any) => {

5 avaliationHandler(snap.data());

6 });

7 }

**Fonte:** Elaborado pelos autores (2019)

**Código YY:** Código executado após uma avaliação é atualizada no banco de dados.

1 **export** **function** onAvaliatedUpdated() {

2 **return** functions.firestore.document(

3 Constants.AVALIATIONS\_COLLECTION + avaliationUidParams)

4 .onUpdate(async (snap: any) => {

5 avaliationHandler(snap.after.data());

6 });

7 }

**Fonte:** Elaborado pelos autores (2019)

Assim como no Código XX (Requisição de perfis) o método *avaliationHandler* requisita os perfis envolvidos a partir dos Ids de cada usuário salvos na avaliação. Em seguida ele cria o payload para notificação de avaliações e envia ao usuário que recebeu a avaliação.

Em ambos os casos foi necessário a requisição dos perfis de cliente e profissional para enviar e criar os *payloads* de avaliação e solicitação de serviços.

Com a criação ou atualização de uma avaliação, o perfil avaliado deve ser atualizado para receber as novas notas. Para isso usamos os mesmos eventos disparados para as avaliações, no qual é disparado junto com a notificação de avaliação, a alteração do perfil avaliado, conforme os códigos XX e YY abaixo.

**Código XX:** Alterando os dados do perfil em nova avaliação.

1 **export** **function** onProfileRatedByNewAvaliation() {

2 **return** functions.firestore.document(

3 Constants.AVALIATIONS\_COLLECTION + avaliationUidParams)

4 .onCreate(async (snap: any) => {

5 **return** profileHandler(snap.after.data());

6 });

7 }

**Fonte:** Elaborado pelos autores (2019)

**Código YY:** Alterando os dados do perfil em uma atualização de avaliação.

1 **export** **function** onProfileRatedByOldAvaliation() {

2 **return** functions.firestore.document(

3 Constants.AVALIATIONS\_COLLECTION + avaliationUidParams)

4 .onUpdate(async (snap: any) => {

5 **return** profileHandler(snap.data());

6 });

7 }

**Fonte:** Elaborado pelos autores (2019)

O método *profileHandler* na linha 5 dos Códigos XX e YY acima, realiza a atualização das notas mínimas, médias e máximas do perfil avaliado e atualiza seus dados no banco de dados. Após isso é enviado uma notificação ao usuário sem conteúdo, que possui a finalidade de informar o aplicativo que ele deve atualizar os dados perfil do usuário.

Após o desenvolvimento foi feita a publicação dos códigos no *cloud functions*. Para isso usamos o comando **firebase deploy** no terminal. Com esse comando o firebase faz o upload de todo o código para sua infraestrutura pronto para execução. Assim que algum evento é disparado, uma das funções criadas será executada, no caso deste projeto um evento de criação, atualização de solicitação ou avaliação.

Recebendo dados do Cloud Messaging

Para configurar o *cloud messaging*foi necessário acessar o painel do *firebase* e selecionar a opção “Adicionar aplicativo”, no topo da página inicial e seguir com o tutorial passo a passo. Como este projeto visa ser executado em apenas smartphones Android, somente a opção Android foi necessária a ser feita neste processo.

No aplicativo foi necessário a instalação de um plugin no qual é responsável por receber as mensagens vindas do firebase, o FCM, ou Firebase Cloud Messaging. Ele é um plugin no qual provê o *token* de acesso e o sistema que recebe e exibe as notificações para o usuário. Com ele recebemos o *token* de cada usuário no ato do *login* e salvamos este dado no banco de dados, na coleção *profiles,* no perfil do usuário. Este plugin funciona em conjunto com o *cloud messaging*, dando total suporte a ferramenta do *firebase.*

Para instalar o plugin foi executado os seguintes comandos no terminal: **ionic cordova plugin add cordova-plugin-fcm-with-dependecy-updated** e em seguida, **npm install --save @ionic-native/fcm@4**. Com isso foi necessário adicionar alguns códigos no projeto, nos quais são utilizados para recuperar as mensagens recebidas do *cloud messaging.*

O Código XX é o método construtor da classe *MyApp,* classe que é executada sempre que o aplicativo é aberto. Na linha 14 é realizado a chamada do método *initService* da classe *Notifications,* na qual é responsável criar o serviço de notificação e alterar o comportamento do aplicativo para cada tipo de notificação, além de prover o *token* de acesso para envio de mensagens no *cloud functions.*

**Código XX:** Iniciando o serviço de notificações.

1 **constructor**(

2 private platform: Platform,

3 private statusBar: StatusBar,

4 private toast: Toast,

5 private notifications: Notifications,

6 private profileProvider: ProfileProvider,

7 private appConfigProvider: AppConfigProvider) {

8 this.platform.ready()

9 .then(async () => {

10 **this**.verifyUser();

11 **this**.statusBar.backgroundColorByHexString("#273A56");

12 **this**.statusBar.styleLightContent();

13

14 **this**.notifications.initService();

15 });

16 }

**Fonte:** Elaborado pelos autores (2019)

As notificações são recebidas no método *serviceObservable* e tratadas no método *parseNotification*, nas linhas 3 e 6 respectivamente, conforme segue o Código XX, da classe *Notifications*.

**Código XX:** Métodos da classe *Notifications*.

1 initService() { this.serviceObservable(); }

2

3 serviceObservable() {

4 this.fcm.onNotification()

5 .subscribe(**data** => {

6 this.parseNotification(data);

7 });

8

9 this.fcm.onTokenRefresh()

10 .subscribe(token => {

11 this.deviceToken = token;

12 });

13 }

14

15 getToken(): Promise<string> { **return** this.fcm.getToken(); }

**Fonte:** Elaborado pelos autores (2019)

O método *parseNotification* dispara um evento para cada tipo de notificação, seja ela uma solicitação nova, uma atualização de solicitação existente, uma nova avaliação ou atualização de perfil. Estes eventos retornam para o usuário uma mensagem informando que ele recebeu alguma mensagem

Na linha 15, é realizado a requisição do *token* do dispositivo do usuário, para recebimento das notificações. Esta requisição é feita sempre que o usuário realiza *login* no aplicativo, uma vez que este *token* se torna invalido quando o usuário realiza o *logout* na aplicação.