



CodeTalk

APLICAÇÃO DE DISCUSSÃO DE CÓDIGO, DESENVOLVIDA
PARA A WEB E ANDROID

TRABALHO FINAL DE CURSO
LICENCIATURA EM INFORMÁTICA DE GESTÃO

JOAO SANTOS ROCHA, Nº 21000982

ORIENTADORES: BRUNO CIPRIANO E SÉRGIO GUERREIRO

Índice

Índice de Figuras	3
Resumo.....	4
Abstract	5
1. Introdução	6
2. Enquadramento Teórico.....	7
2.1. AngularJS – Componente Web.....	7
2.2. Android – Componente Mobile.....	8
2.3. Armazenamento e Sincronização de Dados – Parse	9
2.4. Armazenamento e Sincronização de Dados - Firebase	10
3. Método.....	11
3.1. Planeamento	11
3.1.1. Componente Web	11
3.1.2. Componente Mobile	12
3.1.3. Estrutura de Dados.....	12
3.1.4. Desnormalização dos Dados	18
3.2. Desenvolvimento	20
3.2.1. Escolhas de Desenvolvimento.....	20
3.2.2. Componente Web	28
3.2.3. Componente Mobile	36
4. Resultados	42
4.1. Funcionalidades Componente Web	42
4.2. Funcionalidades Componente Mobile	42
5. Conclusões e trabalho futuro	43
Bibliografia	44
Anexos	45
A1. Anexo de Manual Técnico da aplicação	46
A1.1. Componente Web.....	46
A1.1.1. Vistas (pasta codetalk-views).....	46
A1.1.2. Scripts (pasta codetalk-scripts)	46
A1.2. Componente Mobile	64
A1.2.1. Application	64
A1.2.2. Model	64
A1.2.3. Adapters.....	66

A1.2.4. Services	67
A1.2.5. Activities.....	69
A.1.2.6. Layouts	78
A2. Anexo de Manual de Utilizador da aplicação.....	79
A2.1. Componente Web	79
A2.1.1. Página Principal.....	79
A2.1.2. Página de Lista de Grupos.....	81
A2.1.3. Página do Grupo	82
A.2.2. Componente Mobile	84
A.2.2.1. Splash Screen	84
A.2.2.2. Página de Lista de Grupos.....	85
A2.2.3. Página do Grupo	85
A3. Glossário	89

Índice de Figuras

Figura 1 - One-way Data Binding.....	7
Figura 2 - Two-Way Data Binding.....	7
Figura 3 - Distribuição das Versões Android no Mercado.....	8
Figura 4 - AngularJS Three-way Binding.....	10
Figura 5 - Planeamento Componente Web.....	11
Figura 6 - Planeamento Componente Mobile.....	12
Figura 7 - Estrutura de Dados.....	12
Figura 8 - Estrutura Users.....	13
Figura 9 - Exemplo Estrutura Users.....	13
Figura 10 - Estrutura Grupos.....	15
Figura 11 - Exemplo Estrutura Grupos.....	16
Figura 12 - Estrutura Notas.....	17
Figura 13 - Exemplo Estrutura Notas.....	17
Figura 14 - Experiência Android Parse - Lista de Notas.....	21
Figura 15 - Experiência Android Parse - Detalhe de Nota.....	21
Figura 16 - Exemplo de Estrutura de Dados Firebase.....	25
Figura 17 - Página Inicial CodeTalk.....	28
Figura 18 - Página de Lista de Groups CodeTalk.....	30
Figura 19 - Página de Grupo CodeTalk (Administrador).....	31
Figura 20 - Página de Grupo CodeTalk - Convidados.....	31
Figura 21 - Editor de Código CodeTalk.....	33
Figura 22 - Adicionar Nota.....	34
Figura 23 - Lista de Notas.....	35
Figura 24 - Ver Nota.....	35
Figura 25 - Android - Lista de Grupos.....	38
Figura 26 - Android - Página de Grupo.....	39
Figura 27 - Android - Grupos Subscritos.....	40
Figura 28 - Android - Notificações.....	41
Figura 29 - Página Principal.....	79
Figura 30 - Diálogo de Registo.....	80
Figura 31 - Página de Grupos.....	81
Figura 32 - Criar novo Grupo.....	81
Figura 33 - Página do Grupo.....	82
Figura 34 - Adicionar nova Nota.....	83
Figura 35 - Nota Adicionada.....	83
Figura 36 - Splash Screen.....	84
Figura 37 - Lista de Grupos Android.....	85
Figura 38 - Página de Grupo Android.....	85
Figura 39 - Detalhe de Nota Android.....	86
Figura 40 - Adicionar nova Nota Android.....	86
Figura 41 - Notificações Android.....	87
Figura 42 - Subscrição do Grupo Android.....	87
Figura 43 - Menu de Opções Android.....	88
Figura 44 - Grupos subscritos Android.....	88

Resumo

Com o advento da Web 2.0 começam a emergir diversas aplicações Web, cada vez mais avançadas ao nível tecnológico, que oferecem serviços direcionados quer para o trabalho ou para o lazer. As empresas necessitam de se adaptar a esta realidade, existe assim uma enorme procura por mão-de-obra especializada para responder a estas necessidades por parte do mundo empresarial.

A área da programação, seja ela direcionada para a Web ou não, tem vindo a despertar curiosidade entre as várias comunidades e até têm surgido vários serviços na Web que providenciam cursos em determinadas linguagens de programação, como o [Udemy](https://www.udemy.com/)¹ ou o [Coursera](https://www.coursera.org/)².

É neste sentido que me proponho a desenvolver um projeto que, explorando as novas tecnologias disponíveis para o desenvolvimento Web, consiga auxiliar tanto os programadores como as empresas a gerirem e melhorarem os seus métodos de trabalho, fomentando a comunicação entre programadores com o objetivo de se tornarem profissionais mais competentes.

A plataforma CodeTalk é uma aplicação Web, desenvolvida utilizando as frameworks Bootstrap, AngularJS e Firebase, que é desenhada para possibilitar a comunicação em tempo-real de programadores que necessitem de partilhar o código que desenvolveram com vista a obter feedback, sugestões ou ideias de outros programadores que estejam inscritos na plataforma.

Atualmente existem outras plataformas onde a discussão de código é possível, seja entre uma comunidade de programadores ([StackOverflow](https://stackoverflow.com/)³) ou direcionadas principalmente ao mundo empresarial ([Crucible](https://www.atlassian.com/software/crucible/overview)⁴, da Atlassian). Ambas as plataformas concretizam os seus objetivos na medida em que fornecem um serviço que ajuda os programadores a alcançarem o seu potencial máximo. Contudo, nenhuma destas plataformas disponibilizam meios através dos quais os utilizadores consigam criar grupos de discussão apenas com os colegas que desejam para que consigam manter o seu código seguro e longe dos olhos de outras pessoas.

O utilizador CodeTalk tem a possibilidade de criar grupos de discussão e convidar outros utilizadores para neles participar. Quando o utilizador cria um Grupo, tem ao seu dispor um editor de código, onde pode introduzir o que quer partilhar. A partir do momento em que o código é introduzido no editor e salvo, torna-se visível para todos os outros utilizadores que estão no grupo, que têm a possibilidade de acrescentar Notas, selecionando ou não secções do código sobre as quais querem comentar, também essas tornando-se visíveis para todos os outros utilizadores em tempo-real.

Acompanhando estas funcionalidades, também está disponível uma aplicação Android. Nela, os utilizadores CodeTalk também conseguem adicionar comentários no código dos grupos em que estão inseridos enquanto não estão junto do seu computador. No entanto, a funcionalidade mais diferenciadora desta aplicação é a possibilidade dos utilizadores subscreverem à atividade dentro de um grupo, recebendo notificações em tempo real sobre os comentários que nele são inseridos, possibilitando a resposta aos mesmos de uma forma atempada.

¹ Udemy - <https://www.udemy.com/>

² Coursera - <https://www.coursera.org/>

³ StackOverflow – <https://www.stackoverflow.com>

⁴ Crucible - <https://www.atlassian.com/software/crucible/overview>

Abstract

With the advent of the Web 2.0, several Web applications start to emerge, each of them more technologically advanced than the other, offering services directed to either work or leisure. Companies also need to adapt to this new reality and therefore they begin to seek specialized labor in order to meet the corporate world demands.

Given these needs, also in education are increasingly emerging initiatives to foster a taste for these new technologies in both early education and college by implementing specialized courses or degrees that, through education, form young people in this area with the goal that in the future they can become helpful to the world's needs.

Also, programming skills have been arousing curiosity among the various communities around the world. Many web-based services (such as [Udemy](https://www.udemy.com/)⁵ or [Coursera](https://www.coursera.org/)⁶) have been created to provide courses in certain programming languages, enabling people to learn more about these new concepts.

This is why I propose to develop a project, exploring new technologies available for web development, which can assist both programmers and companies to manage and improve their working methods fostering communication between programmers with the objective of becoming more competent professionals.

The CodeTalk platform is a Web application developed using the Bootstrap, AngularJS and Firebase frameworks and it is designed to allow real-time communication between programmers who need to share the code they developed in order to obtain feedback, suggestions or ideas by other developers who are registered on the platform.

There are currently other platforms where code discussion is possible either among the developers community ([StackOverflow](https://stackoverflow.com/)⁷) or primarily aimed at the enterprises ([Crucible](https://www.atlassian.com/software/crucible/overview)⁸, from Atlassian). Both of these platforms achieve their goals in having a service which aids developers in reviewing their code to maximize potential. However, none of these tools provide means on which the users can create discussion groups between only those they want to in order to keep their code secure without exposing it to people they don't want to.

The CodeTalk user has the possibility to create discussion Groups and invite others to join them. When the user creates a Group, they have a code editor at their disposal where they can enter what they want to share. From the moment the code is entered and saved in the editor, it becomes visible to all other users who are in the Group. The users can then (either selecting a code section or not) comment on the code by adding Notes on the Group, then becoming visible to every other user in the group in real-time.

To go along with these features, a companion Android application is also available. On it, the CodeTalk users can also add Notes and comment on their peers' code on-the-go, but mainly the user has the possibility to subscribe to a group's activity, receiving up-to-date notifications on the newly added comments so that they can act upon them in a timely fashion.

⁵ Udemy - <https://www.udemy.com/>

⁶ Coursera – <https://www.coursera.org/>

⁷ StackOverflow – <https://www.stackoverflow.com>

⁸ Crucible - <https://www.atlassian.com/software/crucible/overview>

1. Introdução

Nos dias que correm, o mundo da programação torna-se cada vez mais popular. Existe uma vasta comunidade mundial de programadores que necessita diariamente de encontrar novas soluções para problemas que encontram durante o desenvolvimento de novas aplicações.

Atualmente, a principal plataforma escolhida para a resolução de problemas relacionados com programação é o *StackOverflow*, uma plataforma que permite aos utilizadores publicar perguntas e obter respostas de outros utilizadores.

A plataforma CodeTalk assenta sobre o mesmo princípio de *entreeajuda*. No entanto, não se trata de um sistema de pergunta-resposta, mas sim de uma plataforma onde é possível discutir o código inserido com outros utilizadores através de notas. Quando um utilizador se regista na plataforma CodeTalk, é-lhe dada a opção de criar um grupo e adicionar código ao mesmo. Seguidamente, pode convidar outros utilizadores CodeTalk a participar na discussão, convidando-os para o grupo criado.

Um exemplo de uma aplicação possível do CodeTalk são as chamadas *code reviews*, onde certos pedaços de código são disponibilizados para que os colegas de trabalho possam rever o código, sugerindo modificações para o melhorar. Determinada organização poderia criar um grupo na plataforma CodeTalk e convidar os seus colaboradores a partilhar lá o seu código para que os restantes possam comentá-lo e sugerir modificações.

A discussão na plataforma é efetuada em tempo real, sendo que assim que um utilizador acrescenta uma nota a um grupo, todos os utilizadores com acesso a esse grupo recebem instantaneamente a nota no seu painel. O mesmo acontece com o convite para determinado grupo – assim que o utilizador é convidado para participar em determinado grupo, esse grupo torna-se imediatamente disponível na lista de grupos do utilizador. Através da aplicação Android, é possível subscrever um determinado grupo, para que assim que uma nota é adicionada a um grupo subscrito, o utilizador receba uma notificação a alertar esse facto.

2. Enquadramento Teórico

2.1. AngularJS – Componente Web

A framework AngularJS apresenta-se como um método de desenvolver websites dinâmicos, ou aplicações web através da utilização de um modelo MVC, ou MVW ([Model-View-Whatever](https://plus.google.com/+AngularJS/posts/aZNVhj355G2)⁹).

Esta framework gaba-se de uma característica que a distingue de todas as outras, *two-way data-binding*. Esta característica faz com que a modificação de dados contidos num objecto JavaScript quer pelo cliente ou pelo servidor se reflita no lado oposto em tempo-real.

Em grande parte dos sistemas MVC, o *binding* de dados é feito de forma uni-direccional, e isto obriga a que o programador seja forçado a desenvolver mecanismos de sincronização entre a Vista e o Modelo já que as modificações feitas na Vista não se refletem automaticamente no Modelo e vice-versa.

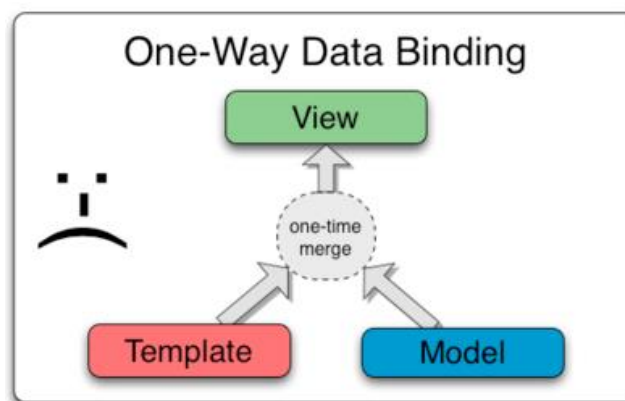


Figura 1 - One-way Data Binding

Isto não acontece num sistema desenvolvido sobre a framework AngularJS, que atualiza continuamente na View os dados modificados no Modelo e vice-versa.

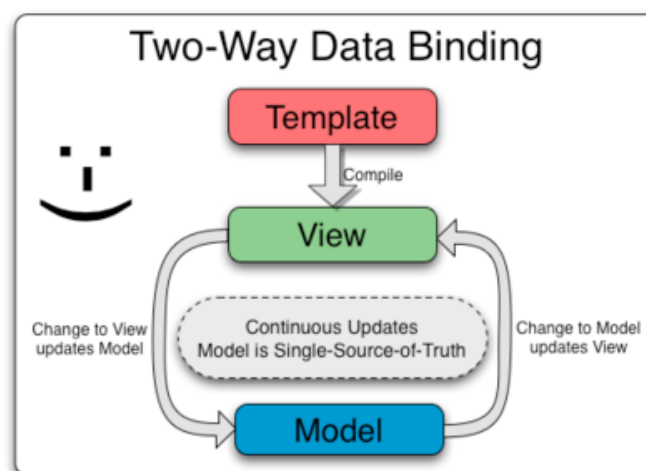


Figura 2 - Two-Way Data Binding

⁹ Model-View-Whatever, AngularJS - <https://plus.google.com/+AngularJS/posts/aZNVhj355G2>

2.2. Android – Componente Mobile

A componente mobile da plataforma CodeTalk assenta sobre o sistema operativo Android. Este sistema operativo é dos mais versáteis pelo que permite o desenvolvimento gratuito sobre o mesmo, sem necessidade de recorrer a ferramentas específicas ou licenças de desenvolvimento.

Esta componente apenas suportará versões iguais ou superiores ao Android 4.0, o que equivale a cerca de 85% do mercado Android, representado abaixo:

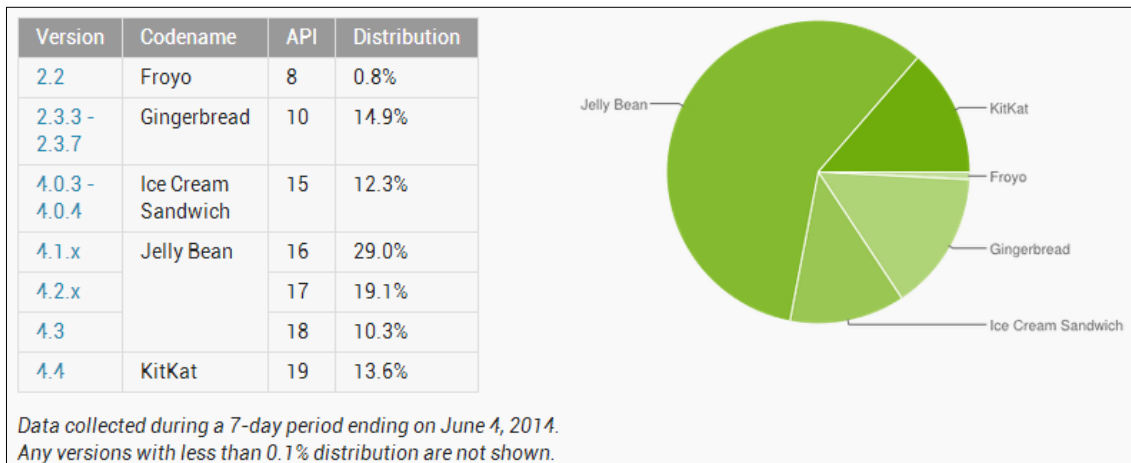


Figura 3 - Distribuição das Versões Android no Mercado¹⁰

Apesar de não abranger todo o mercado Android, a utilização de uma versão mais recente deste sistema operativo faz com que a integração com as restantes componentes seja mais harmoniosa, tanto a nível de utilização de métodos programáticos mais recentes como ao nível do *design* da mesma, possibilitando a utilização de componentes de apresentação de dados mais recentes, enquadrando-se com os padrões de design Android mais atuais.

Num mundo em que cada vez mais os *smartphones* fazem parte das nossas vidas, e a simples notificação de que haverá mais trânsito no caminho para casa já nos parece perfeitamente normal ou se aquela que nos avisa que no dia seguinte estará a chover torrencialmente não aparece é porque qualquer coisa está mal com o nosso dispositivo móvel, torna-se praticamente obrigatório que qualquer aplicação que envolva interação humana tenha uma ou mais componentes Mobile que permitam enviar notificações ao utilizador no caso de ser necessário tomar alguma ação, ou avisá-lo de que tem algo pendente para terminar.

É neste sentido que a integração desta componente na plataforma CodeTalk é fulcral. Tratando-se de uma plataforma que dispõe de comunicação em tempo-real entre os utilizadores, estes sentirão a falta de receber notificações se um pedaço de código sobre o qual manifestaram interesse for modificado, ou algum novo comentário for adicionado.

¹⁰ Retirado de: <https://developer.android.com/about/dashboards/index.html>

2.3. Armazenamento e Sincronização de Dados – Parse

A plataforma Parse, à primeira vista, enquadrava-se perfeitamente na necessidade de armazenamento e sincronização de dados gerados no CodeTalk. Permite guardar e aceder a informação através do armazenamento da mesma numa espécie de base de dados relacional, situada na *Cloud*. Aqui, é possível guardar essa informação em Objetos com um formato semelhante a JSON, os quais estão munidos de métodos *Getter* e *Setter* para possibilitar o armazenamento e acesso à informação de uma forma mais eficiente.

Tanto os métodos de acesso como os de armazenamento da informação são executados de forma assíncrona, possibilitando o registo de funções de *callback* que notificam a aplicação do sucesso ou falha do pedido ao servidor.

Esta plataforma, sendo maioritariamente dirigida ao mercado Mobile, também prevê a possibilidade de os utilizadores das aplicações finais não terem conectividade à internet. Assim, implementa nos métodos utilizados uma funcionalidade que permite o armazenamento da informação localmente, sendo depois enviada para o servidor assim que a conexão seja reestabelecida.

O Parse também disponibiliza funcionalidades de Log-in através de contas de Facebook, Twitter, ou o simples método de introdução de E-mail e Password. Através desta última, é possível enviar um e-mail de verificação ao utilizador no qual ele pode confirmar que a conta de e-mail é sua. Também é possível implementar métodos para prever a situação em que o utilizador se esquece da sua *password* e necessita de efetuar um *reset* à mesma.

Quando um utilizador se regista na aplicação que tem como *back-end* o Parse, é possível associar-lhe permissões para ler ou escrever em determinadas tabelas no Parse. Aqui, utilizando as ACL (Access Control List) é possível ter maior controlo na granularidade da permissão de determinado utilizador a determinados objetos disponíveis no Parse. Além de facilitar o controlo da visualização de determinados conteúdos por parte do utilizador, também é possível utilizar estas ACLs para efetuar validações sobre o conteúdo inserido, sendo que é possível verificar o que está a ser introduzido na tabela antes de essa inserção ser efetivada, tornando possível a rejeição da mesma se os dados a ser inseridos não forem coerentes.

2.4. Armazenamento e Sincronização de Dados - Firebase

Todos os dados gerados pela plataforma CodeTalk são armazenados através do serviço Firebase. Este serviço permite o armazenamento de dados através de uma estrutura hierárquica, semelhante a um objeto JavaScript (JSON). Este método de armazenamento de dados difere do paradigma relacional já que os dados são guardados seguindo uma estrutura hierárquica, apenas com relações de parentesco entre si.

Este paradigma de armazenamento de dados torna-se cada vez mais popular com o advento das aplicações web que necessitam de sincronização de dados em tempo real. A utilização de sistemas de armazenamento deste género tem diversas vantagens neste contexto. Verifica-se um aumento de performance bastante significativo em relação a *queries* relacionais, o que significa que os dados podem ser carregados e mostrados ao utilizador praticamente em tempo real – isto é essencial para uma aplicação de colaboração, onde é necessário que os utilizadores obtenham *feedback* sobre as suas ações instantaneamente.

Outra grande vantagem da utilização do serviço Firebase é a sua integração transparente com a *framework* AngularJS, através do módulo [AngularFire¹¹](https://www.firebase.com/quickstart/angularjs.html). Este módulo foi criado pelo Firebase para tirar proveito das capacidades de *two-way binding* do AngularJS, tornando possível um paradigma de *three-way binding*, em que é possível associar uma referência Firebase a uma variável AngularJS – sincronizando automaticamente alterações locais com o ambiente remoto e vice-versa. Esta interação permite a construção de aplicações web distribuídas onde cada utilizador da plataforma obtém dados inseridos por outros utilizadores em tempo real.

3-way Data Binding

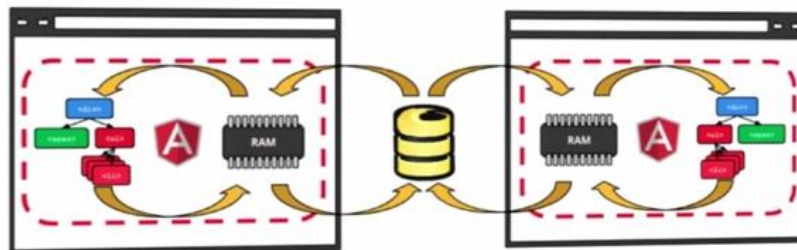


Figura 4 - AngularJS Three-way Binding

¹¹ AngularFire - <https://www.firebase.com/quickstart/angularjs.html>

3. Método

3.1. Planeamento

A plataforma CodeTalk, sendo constituída por dois módulos distintos, uma componente Web – onde estão presentes as principais funcionalidades e uma componente Mobile.

Assim sendo, e tendo em conta que o serviço Firebase integra-se perfeitamente com a framework AngularJS, resolvi começar pelo planeamento da estrutura de dados que serviria de *back-end*, seguido do desenvolvimento da componente Web, e só então implementar nessa estrutura a componente Mobile, em Android.

Tendo em conta que teria cerca de 8 horas por dia durante cerca de mês e meio, aloquei o seguinte a cada componente:

- Componente Web: 240 Horas;
- Componente Mobile: 120 Horas;

3.1.1. Componente Web

Planeamento CodeTalk - Componente Web			
ID da Tarefa	Descrição da Tarefa	Tempo Estimado (horas)	Tempo Total (horas)
0	Início da Implementação Web	-	240
1	Página <i>esqueleto</i> utilizando Bootstrap + Firebase + AngularJS + AngularFire	2	238
2	Página inicial CodeTalk em HTML5 + CSS	4	234
3	Registo de Utilizador (Sign-up)	8	226
4	Log-in / Log-out de Utilizador	8	218
5	Session Tokens	16	202
6	Segregação de Páginas por Utilizador	12	190
7	Página de Listagem de Grupos	4	186
8	Método - Criação de Grupos	8	178
9	Gestão de criação de Grupos duplicados	4	174
10	Segregação de Grupos criados por Utilizador	12	162
11	Método - Listagem de Grupos	5	157
12	Página de Grupo Individual	4	153
13	Método - Ver Detalhe de Grupo	8	145
14	Método - Abandonar Grupo	6	139
15	Método - Convidar Utilizador para o Grupo	6	133
16	Ace-Editor Setup	10	123
17	Método - Escolher Modo (Linguagem)	5	118
18	Método - Adicionar Código no Ace-Editor (Copy & Paste)	3	115
19	Método - Gravar Código	8	107
20	Método - Bloquear Código	5	102
21	Método - Adicionar Código no Ace-Editor (Drag & Drop)	16	86
22	Método - Adicionar Nota	8	78
23	Método - Adicionar Nota com selecção de código Ace-Editor	12	66
24	Diálogo de Detalhe da Nota	4	62
25	Método - Ver Detalhe de Nota	8	54
26	Refinamento de Validações	8	46
27	Testes de Usabilidade	20	26
28	Correção de Bugs	26	0

Figura 5 - Planeamento Componente Web

3.1.2. Componente Mobile

Planeamento CodeTalk - Componente Mobile			
ID da Tarefa	Descrição da Tarefa	Tempo Estimado (horas)	Tempo Total (horas)
0	Início da Implementação Mobile	-	120
1	Aplicação <i>esqueleto</i> utilizando Android SDK + Firebase	2	118
2	Splash Screen CodeTalk	3	115
3	Ecrã de Log-in	4	111
4	Log-in / Log-out de Utilizador	8	103
5	Ecrã de Listagem de Grupos	4	99
6	Método - Listagem de Grupos	5	94
7	Ecrã de Grupo Individual	4	90
8	Método - Ver Detalhe de Grupo	6	84
9	Método - Listagem de Notas	5	79
10	Ecrã de Adicionar Nota	4	75
11	Método - Adicionar Nota	6	69
12	Ecrã de Detalhe de Nota	4	65
13	Método - Subscrição de Notificações de um Grupo	20	45
14	Método - Remoção de Subscrição de um Grupo	5	40
15	Ecrã de Subscrições	4	36
16	Método - Listagem de Subscrições	6	30
17	Testes de Usabilidade	15	15
18	Correcção de Bugs	15	0

Figura 6 - Planeamento Componente Mobile

3.1.3. Estrutura de Dados

A estrutura de dados em que assenta a plataforma CodeTalk é uma estrutura hierárquica, englobando relações de parentesco entre Utilizadores, Grupos e Notas. Um Utilizador pode participar em vários Grupos, e um Grupo pode conter várias Notas.

Assim, a estrutura de dados adotada é a seguinte:

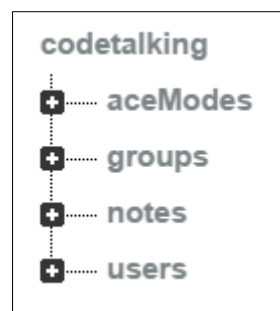


Figura 7 - Estrutura de Dados

É composta por 4 nós principais – *aceModes*, *groups*, *notes* e *users*. Além destes nós principais, também se procedeu a uma desnormalização dos dados neles inseridos, já que em cada um destes nós se pode encontrar informação referente a outros, conforme necessário com vista à otimização das *queries* efetuadas pelo Firebase.

O único método de efetuar *queries* pelo Firebase é a procura por localização, sendo que quando o utilizador está numa vista que está a ser sincronizada pelo nó *Groups*, é muito mais eficiente fazer a procura da informação dentro desse mesmo nó, do que voltar à raiz e procurar noutro nó distinto. Esta desnormalização está patente na estrutura de cada nó, representada abaixo.

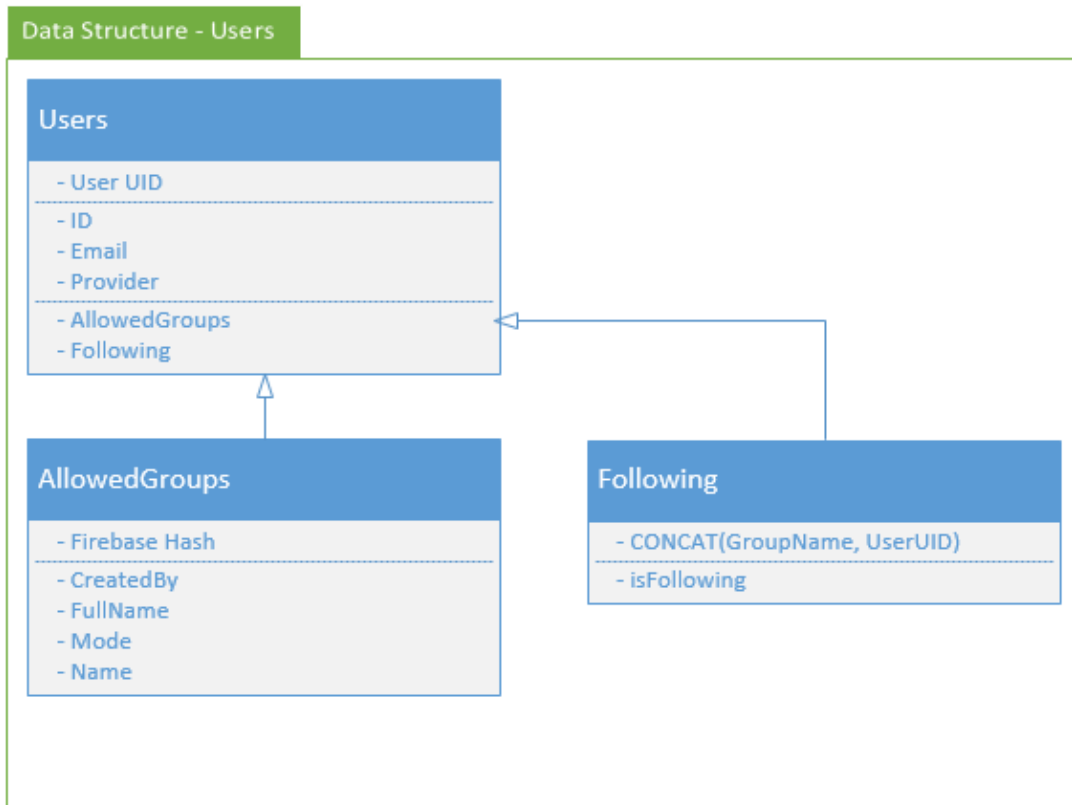


Figura 8 - Estrutura Users

A estrutura de dados da secção *users*, representada acima, contempla informação sobre os utilizadores existentes (ID, Email e Provider) assim como as escolhas de cada utilizador em relação aos grupos aos quais pertencem e têm visibilidade e quais deles estão a seguir.

Abaixo, um exemplo desta estrutura de dados contendo dados de teste introduzidos pela aplicação:



Figura 9 - Exemplo Estrutura Users

Esta estrutura é composta do UID de utilizador (*simplelogin:43*), e como seus filhos diretos *e-mail*, *id* e *provider*. Contem também duas listas: *allowedGroups* e *following*. A lista *allowedGroups* representa grupos a que esse utilizador tem acesso, enquanto a lista *following* representa Grupos que o utilizador subscreveu a partir da aplicação Android.

Cada utilizador contém as seguintes propriedades:

- *e-mail*: endereço de e-mail do utilizador
- *id*: identificador único referente ao utilizador
- *provider*: método pelo qual o utilizador se registou na plataforma

Cada lista contém as seguintes propriedades:

AllowedGroups

- *createdBy*: utilizador que criou o grupo
- *fullName*: nome completo do grupo (o nome guardado no nó *groups* – concatenação entre o nome do grupo e o criador do mesmo)
- *mode*: linguagem de programação definida no grupo
- *name*: nome do grupo

Following

- *isFollowing*: *flag* representativa do estado de subscrição do utilizador a um determinado grupo

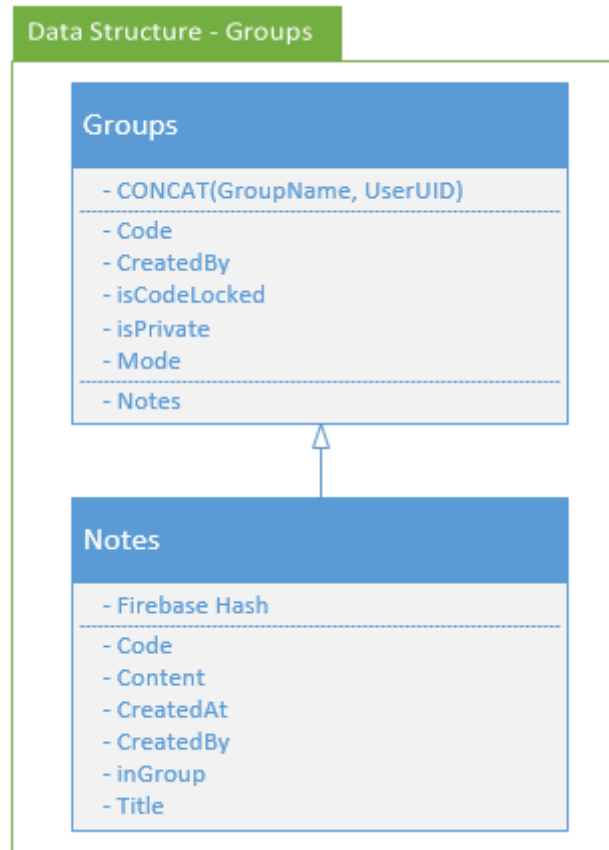


Figura 10 - Estrutura Grupos

Na estrutura apresentada na figura acima, é patente a desnormalização necessária dos dados na plataforma CodeTalk. Como já foi referido acima, isto torna mais eficiente o acesso aos dados necessários para disponibilizar uma experiência de utilização em tempo-real. Dentro da estrutura de cada grupo está disponível a lista de notas existentes no mesmo – se as notas estivessem apenas armazenadas na secção correspondente, a aplicação demoraria mais a encontrá-las já que teria de percorrer toda a árvore, fazendo *match* com determinados parâmetros por forma a determinar se cada nota individual pertenceria ao grupo correto.

Abaixo, um exemplo de como esta estrutura de dados está populada num ambiente de teste:



Figura 11 - Exemplo Estrutura Grupos

Esta secção contem os Grupos criados na plataforma. O nome do grupo é uma concatenação entre o nome dado ao Grupo com o UID do Utilizador que criou o grupo. Esta nomenclatura é necessária para garantir que não são criados grupos com o mesmo nome pelo mesmo utilizador, mas que é possível serem criados grupos com o mesmo nome por utilizadores diferentes.

Foram também definidas as seguintes propriedades em cada grupo:

- *code* – utilizada para guardar o código inserido pelo utilizador
- *createdBy* – guarda o UID do utilizador que criou o grupo
- *isCodeLocked* – determina se o código está bloqueado ou não
- *isPrivate* – determina se o grupo é privado ou não
- *mode* – armazena a linguagem de programação em que o grupo é baseado

Além das propriedades acima, também aqui é armazenada uma lista de Notas, cada Nota com as seguintes propriedades:

- *code* – armazena o código introduzido na nota
- *content* – armazena o conteúdo escrito na nota
- *createdAt* – guarda a data da criação da nota
- *createdBy* – guarda o e-mail do utilizador que criou a nota
- *inGroup* – guarda o nome do grupo onde a nota se situa
- *title* – guarda o título da nota

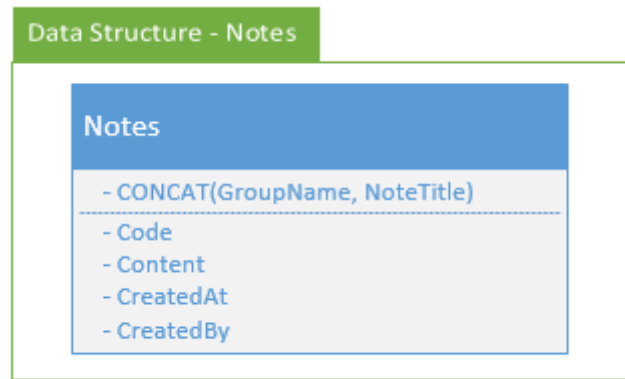


Figura 12 - Estrutura Notas

Acima está representada a estrutura definida para armazenar as notas. As notas individuais são armazenadas também fora da estrutura de grupos para que possam ser acedidas individualmente com vista a mostrar o conteúdo detalhado de cada uma. Na apresentação de uma nota individual, torna-se mais eficiente aceder diretamente a ela através de um identificador único. A razão para esta divisão e consequente desnormalização dos dados armazenados é clarificada na secção 3.1.4, abaixo.

Seguidamente, é apresentado um exemplo de como esta estrutura é utilizada:

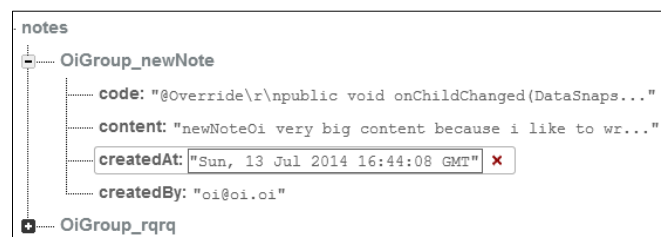


Figura 13 - Exemplo Estrutura Notas

A secção acima representa as notas adicionadas, sendo que o título de cada nota é uma concatenação entre o seu título e o grupo onde ela está contida. Isto é necessário para evitar notas com títulos duplicados no mesmo grupo.

Cada Nota tem as mesmas propriedades representadas na lista que está contida dentro da secção de Grupos.

3.1.4. Desnormalização dos Dados

De acordo com o [Teorema de Brewer¹²](#) (ou teorema CAP) é necessário que um sistema computacional distribuído faça cedências sobre a Estrutura vs Eficiência de acesso aos dados armazenados, sendo impossível que ambos os conceitos estejam totalmente aplicados num dado sistema.

No caso do Firebase, é tida como preferencial a eficiência de acesso aos dados pelo que é utilizado um sistema baseado em NoSQL, uma *key-value store* que essencialmente é composta por um conjunto de registos os que são compostos por uma chave primária (*key*) e o valor correspondente (*value*).

A lógica por detrás do NoSQL implica que se façam cedências na forma de estruturar os dados armazenados, recorrendo muitas vezes à duplicação dos mesmos ao longo da estrutura de forma a serem mais facilmente acedíveis por diversas componentes da aplicação.

Vários exemplos desta cedência podem ser encontrados na estrutura de dados escolhida para suportar o armazenamento da informação CodeTalk:

Começando pelos 3 nós raiz da árvore principal: Groups, Notes e Users.

Cada um destes nós principais são utilizados para pesquisas rápidas por um determinado elemento nessa lista, quando é necessário aceder apenas a ele e não a uma lista de elementos. No entanto, dentro de cada um destes nós é possível encontrar duplicações das listas encontradas na raiz.

No contexto da árvore Users, é possível encontrar uma lista de Grupos aos quais o utilizador tem acesso. Sem esta duplicação de dados seria necessário percorrer toda a lista raiz de grupos, procurando quais deles se associariam a um determinado utilizador, e apresentando os resultados. Existindo uma lista diretamente na árvore Users, sob o nó do respetivo utilizador, torna-se muito mais eficiente a procura por essa lista, sendo apenas necessário especificar o caminho onde ela se encontra.

O padrão descrito acima pode também encontrar-se na árvore Groups, onde existe uma lista de Notes que pertencem a cada grupo.

Esta duplicação de dados pode muitas vezes parecer contraintuitiva para grande parte dos programadores. No entanto, para ser possível o desenvolvimento e implementação de uma aplicação web realmente sincronizável em tempo-real entre utilizadores e escalável ao ponto de não ser necessária uma mudança de estrutura de dados, apenas mais espaço em disco para armazenar novos dados, e necessária esta desnormalização dos dados.

Essencialmente, a ideia é otimizar as leituras de dados escrevendo dados extra em *runtime*, assincronamente e longe do foco do utilizador – espaço em disco é relativamente barato quando se compara com o tempo que o utilizador pode eventualmente gastar a realizar determinada operação na aplicação.

A framework Firebase disponibiliza dois métodos para obter resultados da estrutura de dados – por caminho ou por prioridade.

Os pedidos por caminho são facilmente equiparados a uma *query* SELECT utilizando uma clausula WHERE que indica o ID da linha exata que se pretende obter enquanto que os caminhos

¹² Teorema de Brewer: http://en.wikipedia.org/wiki/CAP_theorem

por prioridade são equiparados a uma *query* SELECT com a mesma clausula WHERE mas que retorne um conjunto de linhas.

A toda a informação armazenada no Firebase é atribuído um valor de prioridade arbitrário que pode ser modificado para tornar o acesso aos dados mais rápido – algo que tem algumas semelhanças com a criação de um índice em SQL.

Na estrutura de dados escolhida para suportar esta plataforma é possível encontrar listas construídas de duas formas distintas – por objetos cuja chave é o nome do mesmo, ou por objetos cuja chave é um *hash* único construído pelo Firebase através de uma *timestamp* da data em que o mesmo foi criado.

A razão para este facto é que as duas formas têm vantagens e desvantagens na implementação desejada. Nas listas compostas por *hashes*, o nível de eficiência pelo qual a leitura dos seus elementos é efetuada é mais alto, no entanto, torna impossível a procura de um determinado elemento individual dentro da lista. Enquanto isso, as listas cujos elementos têm a chave como sendo um nome, têm uma leitura menos eficiente mas torna possível a procura de um elemento individual dentro da mesma.

Assim sendo, em casos que é necessário listar os elementos do grupo numa vista para o utilizador, são usadas listas compostas por *hashes*, enquanto listas apenas utilizadas para verificação de dados introduzidos pelo utilizador são compostas pelo nome dos elementos.

3.2. Desenvolvimento

3.2.1. Escolhas de Desenvolvimento

Framework Parse - Mobile

Começando por fazer diversas experiências ao nível das tecnologias a utilizar no projeto, desenvolvi uma aplicação simples em Android utilizando a framework Parse. Com esta aplicação era possível adicionar, remover e visualizar notas através de um interface simples através do SDK Android disponibilizado pelo Parse.

Optei por mostrar as notas numa *GridView*, modelada conforme as necessidades de forma a receber o título, data de criação, e ações a efetuar sobre as notas (definir alarme e apagar nota).

Cada vez que a aplicação é iniciada, é necessário fazer uma chamada HTTP ao Parse que, ao ser bem-sucedida, carrega todas as notas existentes em memória, possibilitando a sua apresentação na *GridView*, através de um *Adapter*. Para explorar as capacidades de armazenamento em cache dos dados recebidos disponibilizada pelo Parse para trabalho *offline*, também utilizei a *CachePolicy CACHE_THEN_NETWORK* para garantir que primeiro os dados são procurados na Cache, e só então (se não forem encontrados) são carregados da *Cloud*:

```
public void update() {
    noteList = new ArrayList<Note>();
    ParseQuery<ParseObject> query = new ParseQuery<ParseObject>("Notes");
    query.orderByDescending("updatedAt");
    query.setCachePolicy(ParseQuery.CachePolicy.CACHE_THEN_NETWORK);
    query.findInBackground(new FindCallback<ParseObject>() {
        @Override
        public void done(List<ParseObject> objects, ParseException e) {
            if (e == null)
            {
                for (ParseObject note : objects)
                {
                    Note map = new Note();
                    map.setTitle(note.getString("title"));
                    map.setContent(note.getString("content"));
                    map.setId(note.getObjectId());
                    map.setDate(note.getUpdatedAt());
                    noteList.add(map);
                }
                gridView = (GridView) v.findViewById(R.id.gridView1);
                adapter = new NoteViewAdapter(mContext, noteList);
                gridView.setAdapter(adapter);
            }
        }
    });
}
```

A função referida acima (*update()*) é chamada sempre que a aplicação inicia, com vista a popular a *GridView* com os dados necessários. A chamada *findInBackground* disponibilizada pelo Parse é assíncrona, e não bloqueia o interface.

Apesar de a framework Parse dispor de um sistema de Push Notifications, este é pago, pelo que para atualizar os dados seja no início da aplicação ou quando uma nota é adicionada, é necessário fazer o pedido explícito ao servidor. Com isto, sempre que o utilizador adiciona ou remove uma nota, a chamada *update()* tem de ser realizada novamente, para garantir a consistência dos dados entre as plataformas.

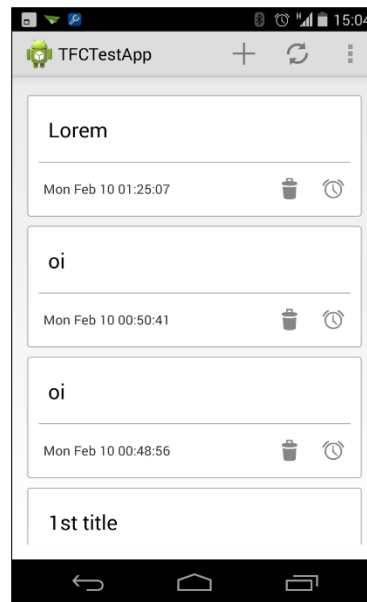


Figura 14 - Experiência Android Parse - Lista de Notas

Para a visualização de notas individuais, é registado um *OnClickListener* na *GridView*, de onde é possível retirar a posição onde o utilizador pressionou e enviar esse ID junto com os dados da nota para uma *Activity* separada, que mostra o detalhe da nota:

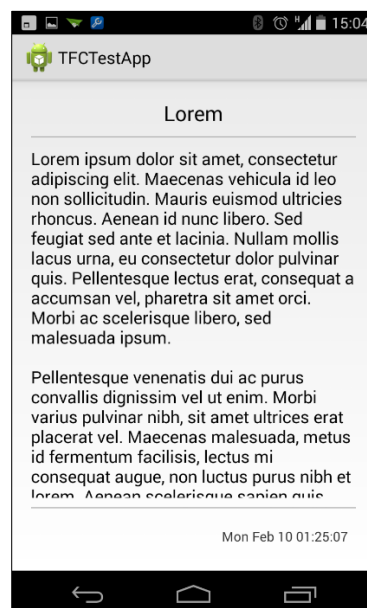


Figura 15 - Experiência Android Parse - Detalhe de Nota

Desta experiência resultou uma aplicação Android funcional que demonstra as capacidades básicas de CRUD (Create, Read, Update, Delete) do Parse.

Framework Parse - Web

Após a experiência com o Android SDK do Parse ter sido bem-sucedida, resolvi começar a experimentar a implementação de uma aplicação semelhante utilizando o AngularJS em conjunção com o serviço Parse através da sua REST API, utilizando chamadas HTTP para recolher e armazenar os dados necessários.

Comecei por explorar a framework AngularJS, que apesar de em teoria parecer ideal para este projeto, nunca tinha trabalhado com ela anteriormente, e é sempre um desafio aprender uma nova forma de trabalhar. Esta framework assenta sobre a linguagem JavaScript – os princípios fundamentais da mesma estão presentes, mas são adicionados diversos conceitos que tornam possível a utilização desta linguagem como back-end de uma aplicação web, *server-side*.

Também utilizo o jQuery, ferramenta de manipulação de DOM com a qual estou já habituado a trabalhar para manipular controlos HTML. No entanto, mais tarde apercebo-me que não é comum este tipo de utilização no AngularJS tanto que até são consideradas más práticas dentro da comunidade.

Ao realizar alguns tutoriais que encontro sobre AngularJS, isto leva-me a compreender melhor a filosofia por trás desta framework. Como em grande parte das web frameworks, o programa de *Hello World* é uma aplicação de TO-DOs, tanto que são disponibilizadas *samples* da mesma aplicação de TO-DOs desenvolvida tendo por base diversas frameworks semelhantes - [TodoMVC¹³](http://todomvc.com/). O processo de aprendizagem no desenvolvimento de uma aplicação deste género engloba operações simples de CRUD (Create, Read, Update and Delete) utilizando as frameworks. Estes são conceitos importantes de absorver já que o cerne da plataforma CodeTalk é a adição, edição e remoção de notas e conteúdos associados.

Após dominar as operações básicas de CRUD no AngularJS, começo a implementar a REST API modificando os métodos originais para suportar chamadas HTTP ao Parse. Estas operações básicas funcionam perfeitamente, mas rápido começo a notar que existe um excesso de operações a serem efetuadas para simples pedidos de dados ao servidor. Com isto, surgem dois problemas:

- Cada vez que a aplicação é iniciada, tem de ser feito um pedido HTTP que retorna todas as notas do utilizador.
- Cada vez que o utilizador adiciona, edita ou remove uma nota, tem de ser feito um pedido HTTP para atualizar a lista com a nova nota, e de seguida outro para mostrar a lista completa ao utilizador.

¹³ TodoMVC: <http://todomvc.com/>

```
//Save Note
$scope.saveLoader = true;
var newNote = {
    'noteTitle': $scope.newNote.title,
    'noteDescription': $scope.newNote.description
};

if ($('#inputTitle').val() == '') {
    $('#titleAlert').show();
    $scope.saveLoader = false;
}
else {
    parseObjFactory.createNote('CodeNotes', newNote)
        .success(function(data) {
            $scope.saveLoader = false;
            $('#inputTitle').val('');
            $('#inputDescription').val('');
            $('#titleAlert').hide();

            //Reload Notes
            parseObjFactory.getParseObj('CodeNotes')
                .success(function(data) {
                    $scope.notes = data.results;
                })
                .error(function(data) {
                    $('#inputTitle').val('Error');
                    $('#inputDescription').val(angular.toJson(data));
                });
        })
        .error(function(data) {
            $scope.saveLoader = false;
            console.log('Error data: '+data);
        });
}
```

No exemplo acima, é demonstrado um pedaço de código utilizado para guardar uma nota no Parse, a partir da aplicação web.

Para acesso ao Parse, utilizo uma *factory* de AngularJS, construída por mim, que contém métodos para chamadas HTTP assíncronas. Neste caso, utilizo a ação *createNote* para guardar a nota no objeto Parse *CodeNotes*. De seguida, tenho de atualizar a vista com a lista de notas (representada por *\$scope.notes*) atualizada, proveniente do Parse através do método *getParseObj*.

Neste caso em particular, é necessário que a primeira chamada HTTP seja bem-sucedida, para que se consiga efetuar a segunda. É algo que se conseguiria resolver através de uma nova tentativa dentro da função de erro, no entanto, é algo que envolve tratamento de exceções em cadeia, o que não é aconselhável em JavaScript. Este é o problema mais grave que encontro enquanto começo a descobrir a API REST do Parse.

Tentando trabalhar à volta deste problema, rapidamente encontro uma maior sucessão de chamadas HTTP, enquanto tento implementar a associação de ficheiros a uma nota. Para associar ficheiros a um determinado objeto Parse, é necessário primeiro fazer *upload* do mesmo para o servidor Parse. Dessa chamada HTTP é retornado um URL onde está disponível o ficheiro, assim como um ID para aceder ao mesmo. Tendo esse ID, é necessário efetuar outra chamada HTTP, enviando esse ID para criar uma relação abstrata entre o objeto Parse e a localização do ficheiro – este ficheiro é guardado de uma forma independente ao Parse, pelo que apenas fica

acessível por URL. Finalmente, é necessário criar um objeto do tipo *Pointer* que é guardado no objeto Parse destino e que aponta para o ficheiro adicionado.

```
//Upload Code to Parse
parseObjFactory.uploadCode(code.content, code.title)
.success(function(data) {
    console.log("SUCCESS - Upload Code");
    parseObjFactory.associateCode(data.name, code.mode, 'CodeChanges')
    .success(function(data) {
        console.log("SUCCESS - Associate Code");
        parseObjFactory.setRelation(data.objClass, data.objId, data.relColumn,
data.relChild, data.childID)
        .success(function(data) {
            console.log("SUCCESS - Relation defined!");
        })
        .error(function(data) {
            console.log("ERROR - Relation Error");
        });
    })
    .error(function(data) {
        console.log("ERROR - Associate Code");
    });
})
.error(function(data) {
    console.log("ERROR - Upload Code");
});
});
```

Uma vez mais, todos os métodos utilizados estão encapsulados na *factory parseObjFactory*. Mesmo existindo um encapsulamento de informação necessária às chamadas HTTP, é fácil notar que existe uma maior sucessão e consequentemente dependência entre as várias chamadas HTTP necessárias para efetuar uma operação tão simples como a associação de um ficheiro a uma nota.

Tendo em conta que a plataforma CodeTalk é desenvolvida para suportar um ambiente multiutilizador, tanto a partir da componente mobile como a partir da componente web, esta sucessão e dependência de chamadas HTTP torna-se incomportável, já que seria necessário demasiado esforço relativo à execução de código *client-side*.

Além disso, a sincronização entre informação mostrada em clientes distintos também se torna instável, já que teria de ser implementado algum tipo de *long-polling* para possibilitar a apresentação de informação atualizada a todos os clientes em tempo real. Sendo que a filosofia da plataforma CodeTalk é a colaboração entre utilizadores, decido procurar uma melhor solução para este problema através da exploração da framework Firebase.

Framework Firebase - Web

Enquanto vejo alguns tutoriais sobre AngularJS na expectativa de encontrar uma solução ao nível dessa framework para o problema que tenho com o Parse, encontro um [vídeo¹⁴](#) gravado numa das conferências de AngularJS ([ng-conf¹⁵](#)) sobre a plataforma Firebase e a sua integração com o AngularJS.

Isto entusiasma-me bastante, já que aparentemente resolve de forma simples os grandes problemas que tenho com a plataforma Parse. Continuando a ver o vídeo, reparo que fazem referência à funcionalidade de *two-way data binding*, referida antes em informação sobre o

¹⁴ Building Realtime Apps With Firebase and Angular:

https://www.youtube.com/watch?v=e4yUTkva_FM

¹⁵ NG-Conf: <http://www.ng-conf.org/>

AngularJS. Com isto, também referem que a plataforma Firebase implementou um sistema de *three-way data binding*, que se integra com o AngularJS.

Esta funcionalidade faz com que seja possível a inserção e atualização em tempo real dos dados inseridos em múltiplos clientes simultaneamente sem a necessidade de implementação de código personalizado para isso ser possível. Tudo acontece dentro da plataforma Firebase, sendo apenas necessário fazer *bind* de uma variável ao serviço Firebase. Essa variável é representada como um objeto JSON e é possível adicionar uma estrutura JSON à mesma, sendo essa estrutura representada no servidor, através do *Data Browser* no website Firebase:



Figura 16 - Exemplo de Estrutura de Dados Firebase

É possível aceder a qualquer um desses nós através de relações de parentesco, assim como através de um URL que representa a árvore definida.

Rapidamente me apercebo que uma estrutura de dados deste género se aplica na perfeição à estrutura de dados que estou a implementar na plataforma CodeTalk. A estrutura *Grupos – Utilizadores – Notas – Ficheiros* é uma estrutura hierárquica, facilmente representável numa estrutura como a representada acima.

Após alguma pesquisa, encontro uma aplicação simples de CRUD utilizando a lógica de TO-DOs construída sobre AngularJS e Firebase. Nessa aplicação, procuro pelos métodos de inserção de dados na plataforma, e reparo que existe uma simplicidade inerente aos mesmos, já que existe um encapsulamento maior do código necessário para tornar esta sincronização de informação possível.

Nesta aplicação exemplo, é utilizado apenas um controlador que comanda todas as operações CRUD da mesma. Neste controlador, é feito inicialmente um *bind* a uma variável que contém a lista de TO-DOs:

```
todomvc.controller('TodoCtrl', function TodoCtrl($scope, $location, $firebase) {
  var url = 'https://todomvc-angular.firebaseio.com/';
  var fireRef = new Firebase(url);

  $scope.$watch('todos', function() {
    var total = 0;
    var remaining = 0;
    $scope.todos.$getIndex().forEach(function(index) {
      var todo = $scope.todos[index];
      //Skip invalid entries so they don't break the entire app
      if (!todo || !todo.title) {
        return;
      }

      total++;

      if (todo.completed === false) {
        remaining++;
      }
    });

    $scope.totalCount = total;
    $scope.remainingCount = remaining;
    $scope.completedCount = total - remaining;
    $scope.allChecked = remaining === 0;
  }, true);
});
```

Assim sendo, a variável *\$scope.todos* fica ligada ao Firebase, sendo que todas as atualizações à mesma utilizando *wrappers* da API do Firebase são instantaneamente replicadas no servidor, assim como noutros clientes que estejam a correr a aplicação.

```
$scope.addTodo = function () {
  var newTodo = $scope.newTodo.trim();
  if (!newTodo.length) {
    return;
  }
  $scope.todos.$add({
    title: newTodo,
    completed: false
  });
  $scope.newTodo = '';
};
```

No código utilizado para adicionar uma nota, é apenas utilizado o *wrapper* *\$add*, que adiciona uma nota em formato JSON à lista (*\$scope.todos*), sendo assim enviado para o servidor e atualizado na própria lista, o que se reflete imediatamente na vista.

```
$scope.doneEditing = function(id) {
  $scope.editedTodo = null;
  var title = $scope.todos[id].title.trim();
  if (title) {
    $scope.todos.$save(id);
  }
  else {
    $scope.removeTodo(id);
  }
}
```

O excerto utilizado para edição de notas também se mostra bastante simples, apenas recebendo o ID da nota a ser editada, e caso o seu título tenha sido editado, utiliza o *wrapper* `$save` para, dentro da lista `$scope.todos` gravar a nova nota, o que também é replicado no servidor e na própria vista onde está representada a lista.

```
$scope.removeTodo = function(id) {  
    $scope.todos.$remove(id);  
}
```

Igualmente simples é o método de remoção de notas, que recebe o ID da nota a remover e a remove da lista utilizando o *wrapper* `$remove`.

Conclusões

Tendo considerado as opções apresentadas acima, decido avançar com o desenvolvimento da componente Web utilizando o AngularJS em conjugação direta com o Firebase. Quanto à componente Android, já que decidi abandonar a framework Parse na componente Web, decido também explorar a componente Firebase em Android já que também disponibiliza um SDK para este sistema operativo, ficando com ambos os componentes facilmente integráveis entre si.

As aplicações web desenvolvidas em AngularJS são maioritariamente aplicações em que existe uma única página, e o conteúdo da aplicação é apresentado nessa página através de *routes*, aplicando *templates* à mesma. Assim sendo, decido também seguir pelo mesmo caminho, utilizando o Bootstrap para facilitar o desenvolvimento de *front-end*, construindo *templates* HTML para popular a vista principal.

3.2.2. Componente Web

1ª Fase - Preparação

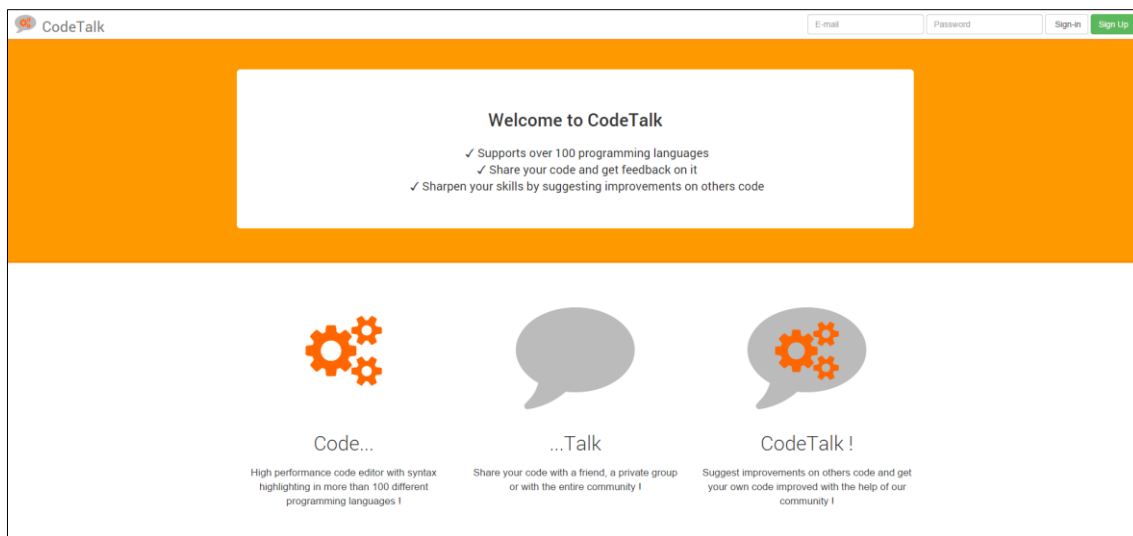


Figura 17 - Página Inicial CodeTalk

Sendo que já tinha alguma experiência na utilização do Bootstrap para construir páginas web, a construção desta não foi um problema. A barra de navegação é separada do conteúdo da página, já que esta é reutilizável entre as várias vistas, apenas modificando a parte superior esquerda, aquando do log-in do utilizador.

Ao longo do processo de desenvolvimento e implementação é utilizada a API *AngularFire* que permite integrar funcionalidades do Firebase com o AngularJS. Esta API contém diversos métodos CRUD que tornam a interação entre eles mais intuitiva:

- **\$add()** – Permite adicionar um objeto ao *array*, atribuindo-lhe um Firebase *Hash* que o identifica univocamente como nó na árvore que representa a estrutura de dados. A adição de objetos utilizando este método é utilizada para popular listas que necessitam de ser apresentadas utilizando o método *ng-repeat* disponibilizado pelo AngularJS.
- **\$set()** – Permite adicionar um objeto à estrutura de dados, na posição escolhida da árvore através de uma referência Firebase para essa posição. Este método é recomendado para a adição de objetos que necessitam de ser acedidos individualmente, sendo possível aceder a eles mais facilmente.
- **\$update()** – Permite atualizar um objeto na estrutura de dados. É necessário passar como parâmetro a posição do mesmo na árvore, através de uma referência Firebase.
- **\$remove()** – Permite remover um objeto da estrutura de dados. É necessário passar como parâmetro a posição do mesmo na árvore, através de uma referência Firebase.

No entanto, esta API é relativamente nova, e alguns métodos ainda não estão implementados, pelo que por vezes para manipulação dos dados é necessária a utilização da API JavaScript disponibilizada pelo Firebase.

Para utilização dos métodos disponibilizados pela API *AngularFire*, é necessário criar referências Firebase que contemplem a utilização do objeto *\$firebase*. Quando se utiliza a API JavaScript nativa do Firebase, isso não é necessário.

Já que é necessário criar referências para várias secções da estrutura de dados ao longo do ciclo de vida da aplicação, foram adicionadas funções ao singleton `$rootScope` que pode ser acedido em qualquer parte da aplicação.

2ª Fase - Execução

Para desenvolver esta componente começo por focar-me na melhor forma de implementar a estrutura de dados definida para aplicação na fase de planeamento, assim como a ordem pela qual teria de implementar os diversos módulos para ser possível a integração encadeada dos mesmos.

Assim, opto por seguir um método hierárquico na construção dos componentes, pela seguinte ordem:

- 1) Preparação do Template Base
- 2) Gestão de Utilizadores
- 3) Criação e Gestão de Grupos
- 4) Gestão da Página de Grupo Individual
- 5) Métodos de Tratamento do Código Inserido
- 6) Criação e Gestão de Notas

1) Preparação do *Template Base*

Começo por implementar a definição de roteamento de páginas através de caminhos introduzidos no URL do *browser*. Teria de ter uma secção pública e uma privada, apenas disponível a utilizadores registados. Tratando-se de uma aplicação de página única, implementei um *template* base que contem apenas o carregamento do CSS e scripts JavaScript necessários à utilização da aplicação. Todas as outras páginas (ou vistas – seguindo a lógica MVC) seriam carregadas neste *template* inicial utilizando a diretiva *ng-view* disponibilizada pelo AngularJS.

Para definir o roteamento entre as vistas é necessária a utilização do módulo *\$routeProvider*, que possibilita a configuração de cada vista – a sua localização no projeto e o controlador que será atribuído à mesma. Utilizando o módulo *\$location* e atributos *href* nos controlos HTML é então possível trocar os *templates* HTML que representam as vistas, e apresentá-los ao utilizador.

2) Gestão de Utilizadores

Para implementar o registo e log-in dos utilizadores, utilizo métodos da API *AngularFire* para que controlem a gestão de *Session Tokens* em *background*.

Foi necessário também implementar métodos que verificassem o estado da sessão do utilizador ao longo do ciclo de vida da aplicação, para que seja possível verificar se o roteamento escolhido é válido (entre a secção pública e a privada). Na implementação destes métodos foi também decidido que seria necessário guardar em cache o estado dessa sessão para evitar que o utilizador tivesse de fazer log-in novamente caso fechasse a janela da aplicação acidentalmente.

Utilizando estes métodos não é necessária a segregação implícita de páginas entre utilizadores distintos – as páginas apresentadas são as mesmas, o que difere é a informação nelas contida. Já que a informação sobre os utilizadores é guardada no Firebase, é possível mostrar apenas informação à qual o utilizador atual tem acesso.

Como medida adicional de gestão da segregação entre utilizadores e da segurança dos dados, é utilizada a Security API do Firebase, que permite a construção de regras para o acesso de leitura/escrita à informação armazenada.

3) Criação e Gestão de Grupos

O acesso à página de Criação e Gestão de Grupos é efetuado através do redireccionamento automático após o log-in de utilizador, ou automaticamente ao aceder à aplicação se já existir uma sessão ativa em cache.

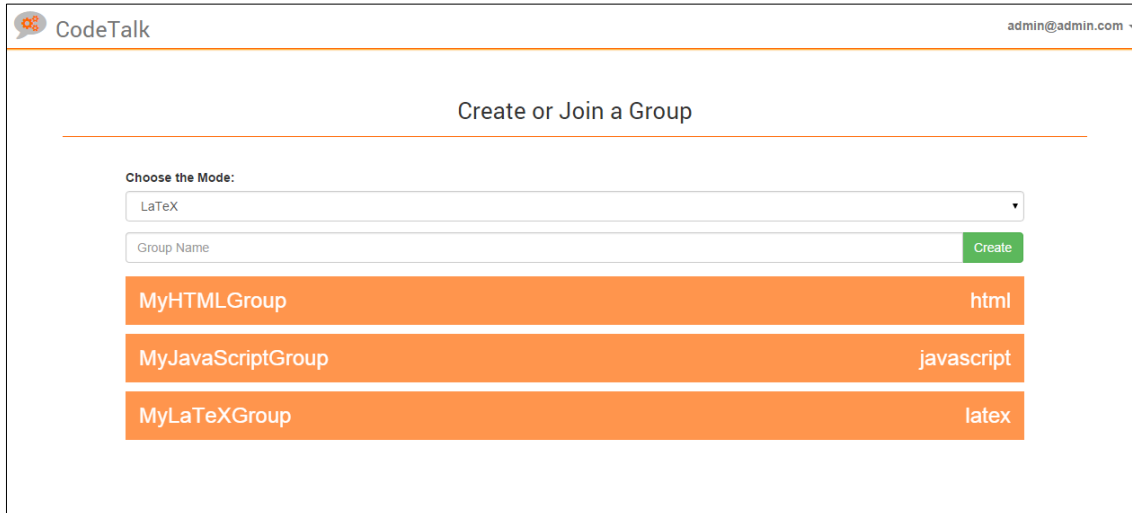


Figura 18 - Página de Lista de Groups CodeTalk

Esta vista foi desenvolvida tendo em conta que ao criar um grupo é necessário definir o Modo do mesmo – a linguagem de programação na qual serão apresentados os trechos de código adicionados. É necessário definir isto para ser possível apresentar na página do grupo um editor de código que faça *highlight* da sintaxe utilizada nessa linguagem.

A criação dos grupos é feita utilizando uma ligação a uma referência Firebase localizada num dos nós filhos da raiz da estrutura de dados (/groups). Sempre que é adicionado um grupo, é adicionado um nó filho a esta estrutura. Além disto, é também adicionada uma entrada no caminho /users/currentUser/allowedGroups.

É a este ultimo caminho que está ligada uma outra lista que é utilizada para apresentar a lista dos grupos ao utilizador através do componente *ng-repeat*, disponibilizado pelo AngularJS.

Este módulo foi particularmente fácil de desenvolver já que é constituído primariamente de métodos que criam ou obtêm listas a partir do Firebase utilizando a API *AngularFire*. A utilização desta API também torna simples a apresentação das listas, uma vez que se adapta na perfeição ao componente *ng-repeat*.

4) Gestão de Página de Grupo Individual

A Página Individual do Grupo foi, de longe, o módulo que envolveu mais trabalho no desenvolvimento desta componente:

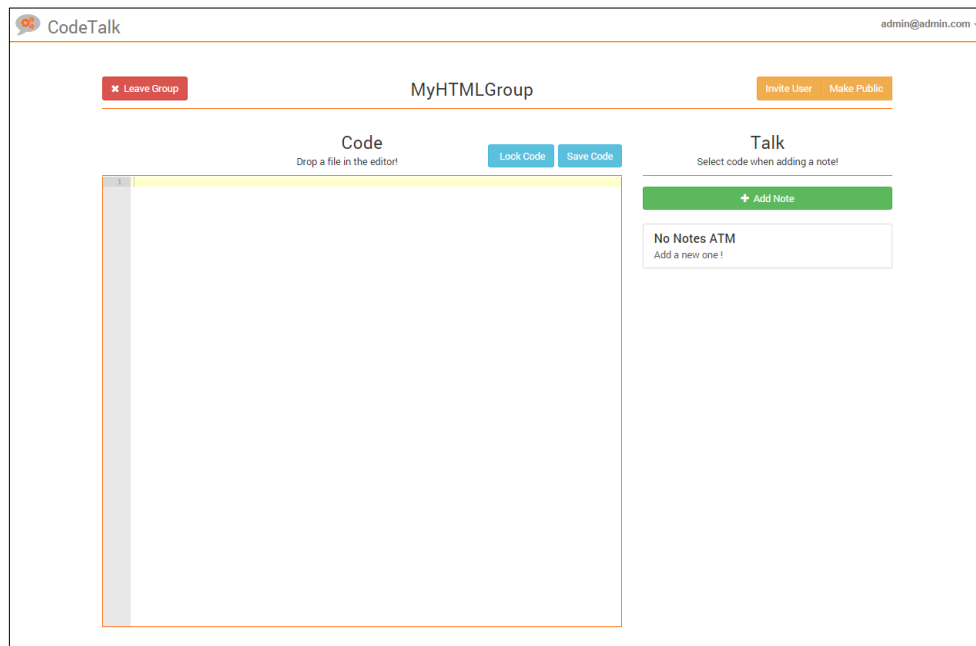


Figura 19 - Página de Grupo CodeTalk (Administrador)

Começando pela implementação do interface deste módulo, é apresentado o Nome do grupo em destaque, assim como ações passíveis de ser tomadas pelo utilizador sobre o grupo.

Algumas ações apenas são mostradas se o utilizador for o criador do grupo. Quando isto não acontece, o interface toma a seguinte forma:

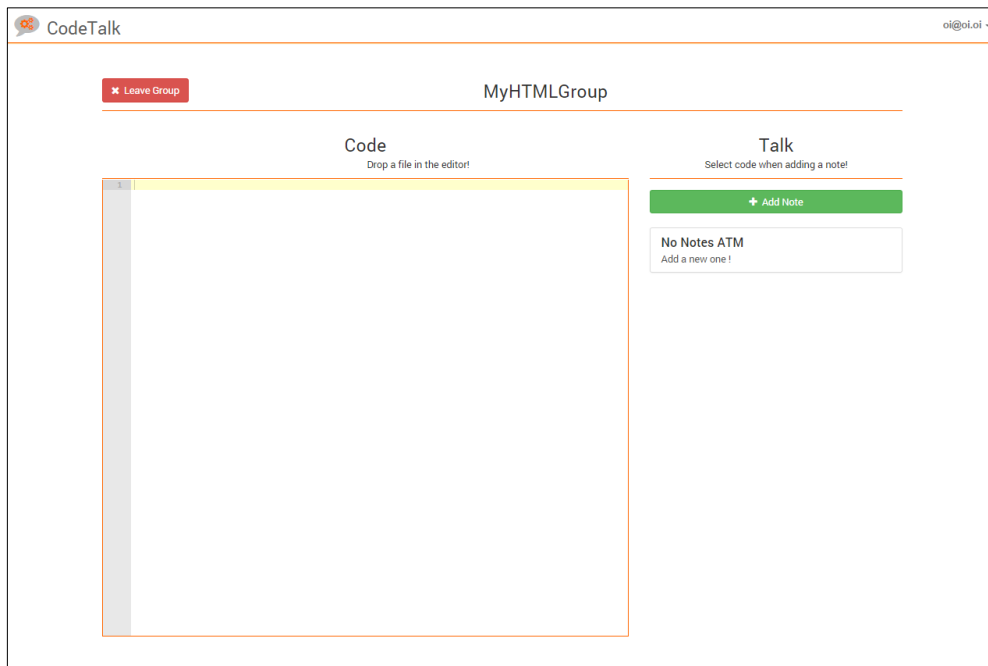


Figura 20 - Página de Grupo CodeTalk - Convidados

A verificação de quais ações a apresentar é feita utilizando o componente *ng-hide* do AngularJS que permite esconder determinados elementos com base numa condição. A condição definida para isto acontecer é verificada com base em dados inseridos no Firebase aquando da criação do grupo (CreatedBy).

Esta página suporta o *Drag & Drop* de ficheiros de código diretamente para o editor, transferindo o código contido no ficheiro para o editor. Esta ação apenas copia o código para o editor – posteriores modificações no editor não são refletidas no ficheiro original.

Para implementar esta funcionalidade foram registados *EventListeners* na DIV que contém o editor de código:

```
editorDiv.addEventListener('dragover', function(e) {
    stopEvent(e);
});

editorDiv.addEventListener('drop', function(e) {
    putFileContentsInAceEditor(e.dataTransfer.files[0], aceObject);
    stopEvent(e);
});
```

O *Listener* para o evento de *dragover* serve para fazer *override* à ação normal deste evento (copy) e substituí-la por uma ação de (move).

Ao registar o evento *drop*, a aplicação chama o método *putFileContentsInAceEditor*:

```
function putFileContentsInAceEditor(file, aceEditor) {
    console.log("PutFileContents");
    var reader, text;
    reader = new FileReader();
    reader.onload = (function (file) {
        text = file.target.result;
        aceEditor.getSession().setValue(text);
    });
    reader.readAsText(file);
}
```

Aqui, através da classe *FileReader* disponibilizada pelo JavaScript nativo, é carregado o texto que o ficheiro contém na sessão atual do editor.

Também nesta página é possível convidar utilizadores para o grupo. Ao pressionar o botão correspondente, é apresentada uma janela que mostra a lista de utilizadores registados na aplicação através da referência Firebase */users*. Isto faz com que o administrador do grupo consiga convidar qualquer pessoa registada na plataforma para fazer parte do seu grupo.

Quando o utilizador escolhe uma pessoa e pressiona OK, uma nova entrada é adicionada à informação do utilizador escolhido em */users/chosenUser/allowedGroups* indicando que esse utilizador agora tem acesso a esse grupo. Se o utilizador que foi escolhido estiver *on-line* na altura, o grupo correspondente aparece instantaneamente na sua lista de grupos disponíveis.

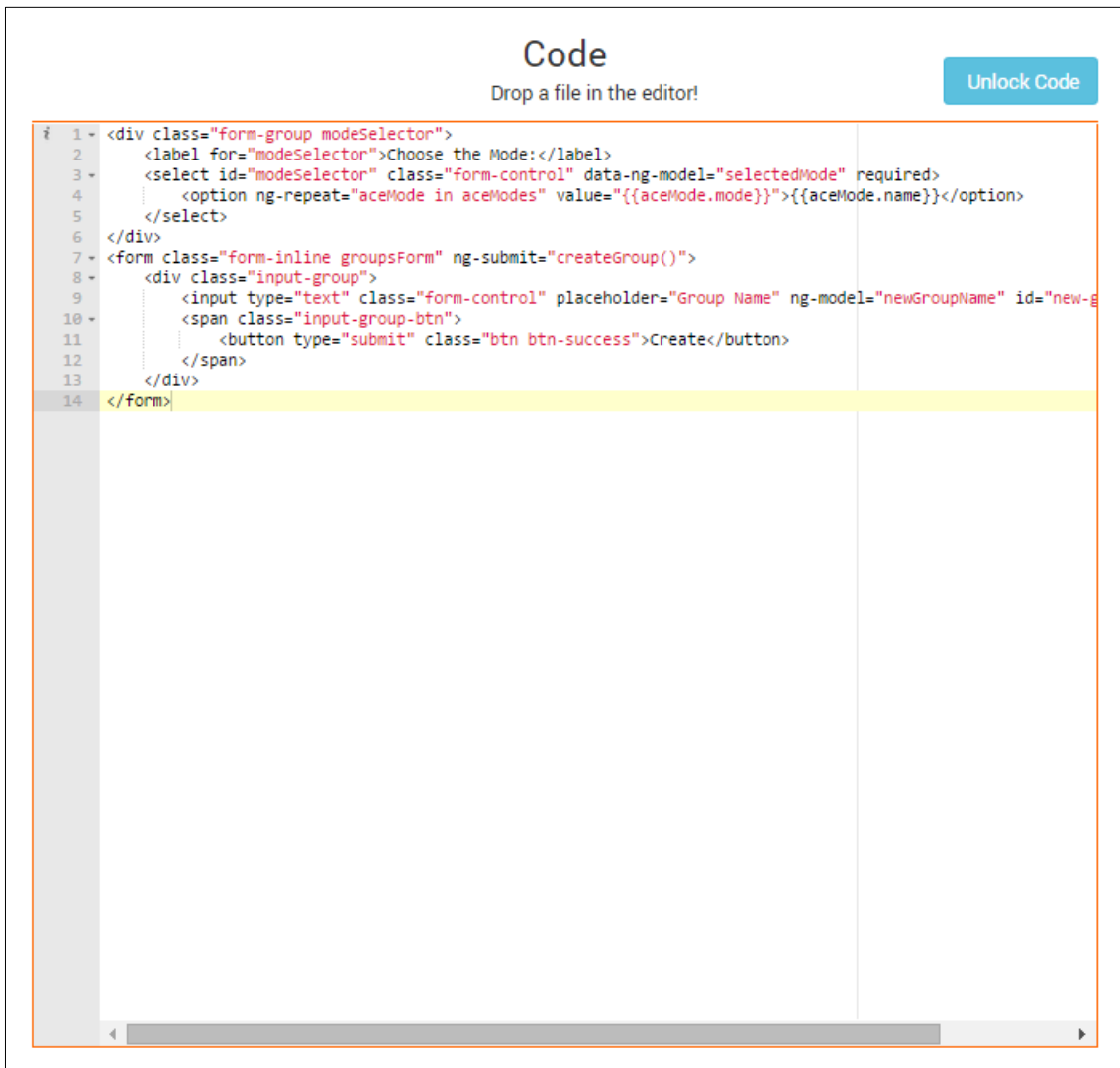
O utilizador tem também a possibilidade de sair do grupo, pressionando o botão *Leave Group*. Isto faz com que a entrada em *./allowedGroups* desse utilizador seja removida. Quando todos os utilizadores saem de um determinado grupo, o mesmo continua a existir, tornando-se inutilizável. Aqui poderia ter sido implementado um qualquer mecanismo de *garbage collection*, mas não foi encontrada uma solução viável para essa funcionalidade.

5) Métodos de Tratamento do Código Inserido

Como editor de código foi escolhido o Ace-Editor (disponível em: <http://ace.c9.io>). Este editor de código é desenvolvido em JavaScript e é facilmente incorporável em aplicações web. É utilizado em aplicações como o GitHub e o IDE Cloud9 tornando-o um editor de referência neste tipo de aplicações. Fornece também uma API intuitiva de tratamento do código lá inserido, facilitando a interação com o mesmo.

Este editor de código é inicializado assim que a Página de Grupo Individual é mostrada e é nele que a plataforma CodeTalk assenta. Após introdução do código no editor, seja por *Drag & Drop* ou via *Copy & Paste* é mostrada a *syntax highlighting* da linguagem de programação escolhida para o grupo. São detetados erros de sintaxe e é feita indentação automática do código inserido.

Ao utilizar a funcionalidade Lock Code, o editor é transformado para *read-mode* para todas as sessões onde ele é inicializado fazendo com que nenhum utilizador que se junte ao grupo consiga modificar o código nele inserido.



```
1 <div class="form-group modeSelector">
2   <label for="modeSelector">Choose the Mode:</label>
3   <select id="modeSelector" class="form-control" data-ng-model="selectedMode" required>
4     <option ng-repeat="aceMode in aceModes" value="{{aceMode.mode}}">{{aceMode.name}}</option>
5   </select>
6 </div>
7 <form class="form-inline groupsForm" ng-submit="createGroup()">
8   <div class="input-group">
9     <input type="text" class="form-control" placeholder="Group Name" ng-model="newGroupName" id="new-g
10    <span class="input-group-btn">
11      <button type="submit" class="btn btn-success">Create</button>
12    </span>
13  </div>
14 </form>
```

Figura 21 - Editor de Código CodeTalk

6) Criação e Gestão de Notas

Quando o utilizador deseja adicionar uma nota sobre um pedaço do código inserido, pode seleccionar o pedaço desejado e pressionar o botão Add Note. Dentro do diálogo que é mostrado quando o utilizador pressiona Add Note, é mostrada uma outra instância deste editor, que contém o código seleccionado em modo *read-only*.

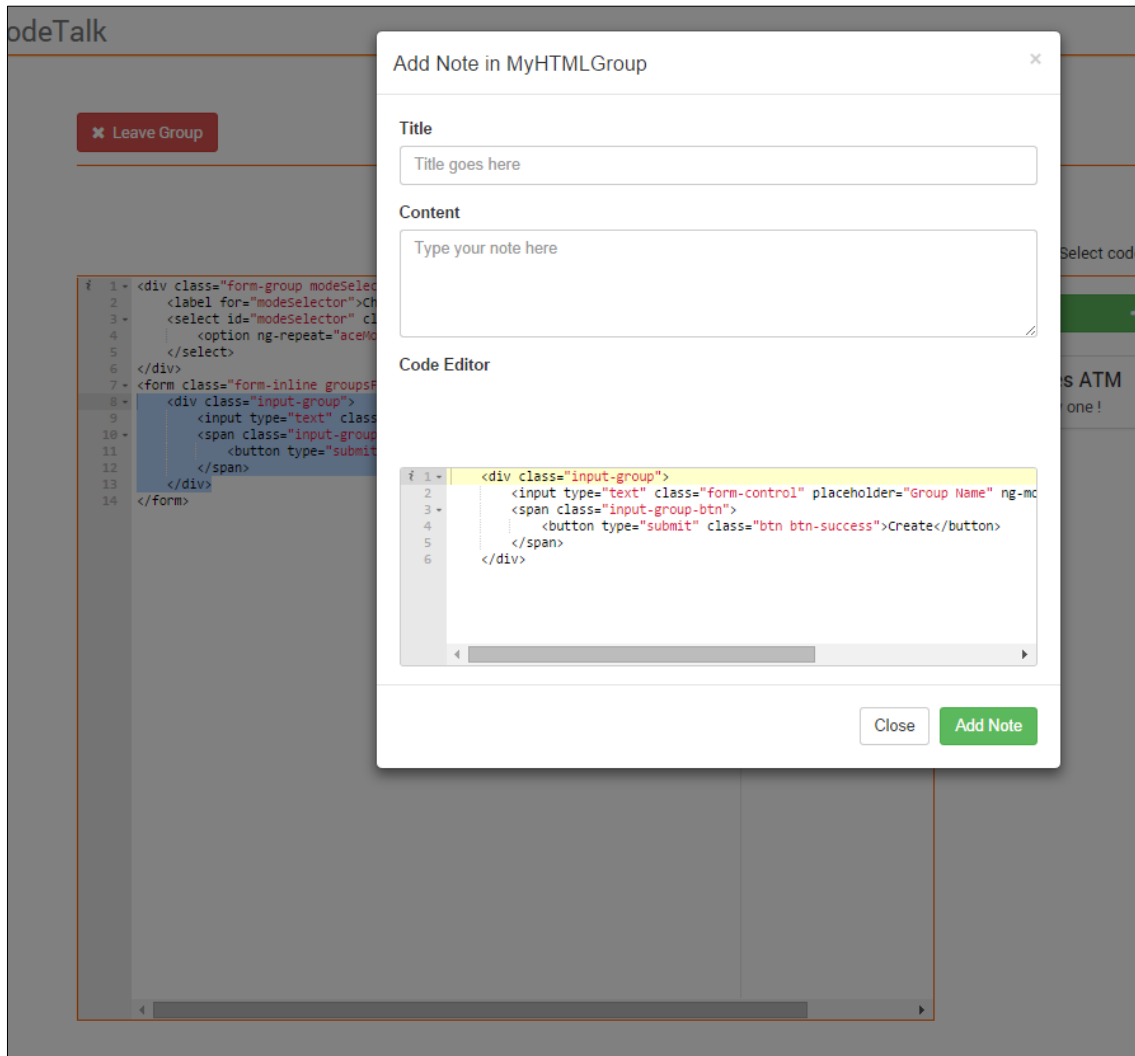
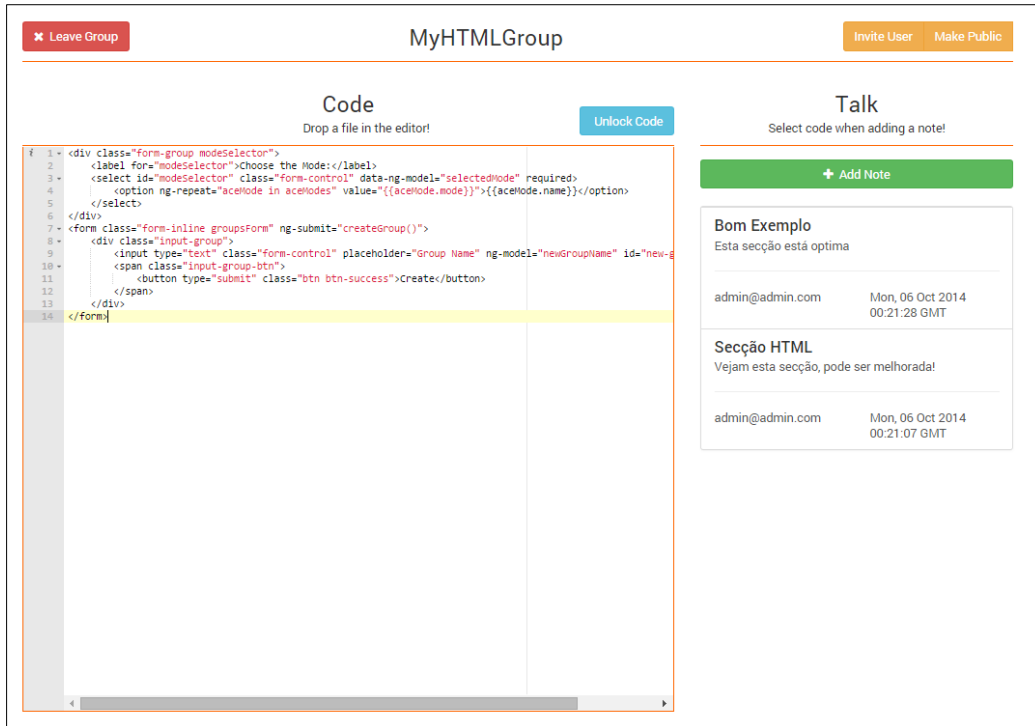


Figura 22 - Adicionar Nota

O utilizador pode então dar um título e adicionar uma descrição à nota e guardá-la. A nota e o respetivo código são guardados imediatamente no Firebase através de referências para duas localizações distintas:

- `/notes` - Para acesso individual, quando é necessário visualizar o detalhe da nota.
- `/groups/currentGroup/notes` - Para acesso via Firebase *hash*, com vista a apresentar na lista lateral na Página do Grupo.

A referência que mostra a lista de notas está constantemente sincronizada por todas as instâncias da aplicação, pelo que mostra a adição de notas em tempo-real entre utilizadores:



The screenshot shows the 'MyHTMLGroup' interface. At the top, there's a 'Leave Group' button on the left and 'Invite User' and 'Make Public' buttons on the right. The main area is split into two columns: 'Code' and 'Talk'.

Code Column: Contains a code editor with the instruction 'Drop a file in the editor!'. A blue 'Unlock Code' button is located at the top right of the editor. The code being edited is an AngularJS form:

```

1 <div class="form-group modeSelector">
2   <label for="modeSelector">Choose the Mode:</label>
3   <select id="modeSelector" class="form-control" data-ng-model="selectedMode" required>
4     <option ng-repeat="aceNode in aceNodes" value="{{aceNode.mode}}">{{aceNode.name}}</option>
5   </select>
6 </div>
7 <form class="form-inline groupsForm" ng-submit="createGroup()">
8   <div class="input-group">
9     <input type="text" class="form-control" placeholder="Group Name" ng-model="newGroupName" id="newGroup">
10    <span class="input-group-btn">
11      <button type="submit" class="btn btn-success">Create</button>
12    </span>
13  </div>
14 </form>

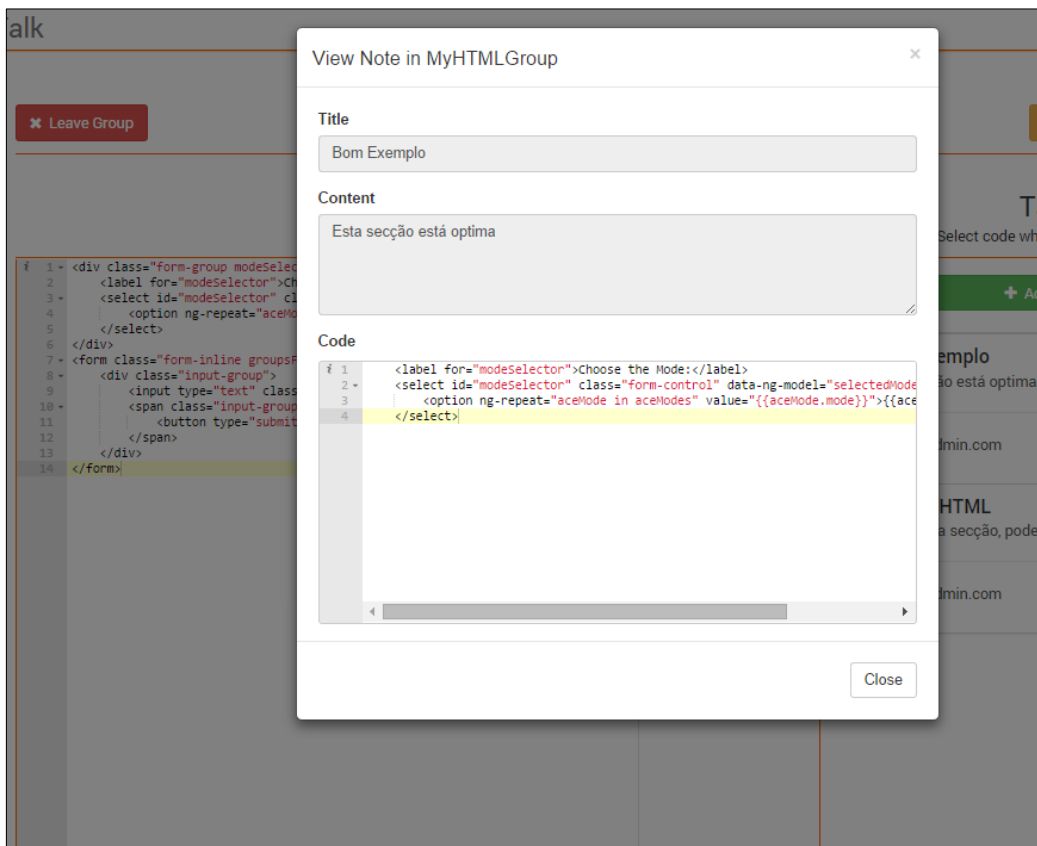
```

Talk Column: Contains a section titled 'Talk' with the instruction 'Select code when adding a note!'. Below this is a green '+ Add Note' button. There are two notes displayed:

- Bom Exemplo**
Esta secção está optima
admin@admin.com Mon, 06 Oct 2014 00:21:28 GMT
- Secção HTML**
Vejam esta secção, pode ser melhorada!
admin@admin.com Mon, 06 Oct 2014 00:21:07 GMT

Figura 23 - Lista de Notas

Clicando num item referente a uma nota, é utilizada a referência /notes para obter a nota clicada e é mostrada uma caixa de diálogo com o conteúdo da mesma:



The screenshot shows a modal dialog box titled 'View Note in MyHTMLGroup'. It displays the details of a specific note:

- Title:** Bom Exemplo
- Content:** Esta secção está optima
- Code:** A code editor showing the relevant part of the HTML code from the previous figure:


```

1 <label for="modeSelector">Choose the Mode:</label>
2 <select id="modeSelector" class="form-control" data-ng-model="selectedMode" required>
3   <option ng-repeat="aceNode in aceNodes" value="{{aceNode.mode}}">{{aceNode.name}}</option>
4 </select>

```

A 'Close' button is located at the bottom right of the dialog box.

Figura 24 - Ver Nota

3.2.3. Componente Mobile

1ª Fase – Preparação

Para o desenvolvimento da componente Mobile da plataforma CodeTalk foi utilizado o Android Studio 0.8.11 e a preparação da aplicação esqueleto foi relativamente fácil uma vez compreendido o conceito de utilização do *Gradle*, que à semelhança do *Maven*, constrói todo o projeto com base em dependências que lhe são passadas. Uma vez que o Firebase suporta este método de *build*, foram importadas as dependências necessárias no *Gradle*, e a aplicação esqueleto ficou então devidamente preparada.

Começo então por explorar o Android SDK disponibilizado pelo Firebase, com o qual ainda não tinha tido contacto prático – apenas leitura da API de uma forma geral. Este rapidamente se mostra em tudo semelhante aos métodos aplicados na componente Web, existindo apenas um conceito importante a ter em conta – a gestão de memória do dispositivo móvel.

Apesar do aumento considerável de características disponíveis nos dispositivos móveis hoje em dia, o próprio sistema operativo Android usa métodos de gestão de memória com vista à conservação de energia e melhoramento do funcionamento multi-task e muitas vezes esses métodos fazem com que seja dada prioridade a determinadas aplicações deixando para trás outras e muitas vezes terminando o processo que as faz trabalhar.

Assim, ao longo do desenvolvimento terei de ter atenção a referências Firebase que deixo ativas sem estarem a ser utilizadas assim como a *listeners* que possam estar registados sem necessidade.

À semelhança do desenvolvimento da componente Web, decidi dividir o desenvolvimento e implementação em três módulos distintos:

- 1) Módulo de Gestão de Utilizadores
- 2) Módulo de Gestão de Grupos e Notas
- 3) Módulo de Gestão de Notificações e Subscrições

A aplicação Android a desenvolver tem um foco maioritariamente de *companion-app*, sendo com ela possível efetuar algumas das ações disponíveis na aplicação web mas não todas, deixando de fora as seguintes funcionalidades principais:

- Registo de Utilizador
- Criação de Grupo
- Edição de Código

No entanto, através da plataforma Mobile também será implementada uma funcionalidade que não está disponível na aplicação Web – a possibilidade de subscrever determinados grupos, podendo obter notificações no seu dispositivo móvel cada vez que uma nota é adicionada ao grupo.

Estas escolhas são meramente estratégicas – uma vez que o objetivo da aplicação Mobile é suportar o funcionamento da aplicação Web através da possibilidade de subscrição à atividade nos grupos, não são disponibilizadas todas as funcionalidades já que a aplicação Web é muito mais *performant* a efetuar as ações que a plataforma CodeTalk disponibiliza.

2ª Fase – Execução

1) Módulo de Gestão de Utilizadores

Seguindo o mesmo princípio aplicado na aplicação Web, começo por explorar o Firebase Simple Login que permite fazer a o log-in dos utilizadores assim como a gestão das suas sessões ao longo do ciclo de vida da aplicação.

Ao fim de alguma exploração, começo por perceber que será necessário criar uma classe que estende a `android.Application` para gerir todos os métodos utilizados pelo Firebase Simple Login, sendo assim possível verificar a existência de sessão de utilizador ao longo da aplicação quando ações de escrita ou leitura são solicitadas.

Esta classe é normalmente utilizada nas aplicações Android como um singleton global que é acessível a partir de qualquer contexto na aplicação – é também onde crio os métodos de criação de referências Firebase, à semelhança do que acontecia na aplicação Web.

Uma vez que a gestão do utilizador e consequente sincronização com o Firebase só é possível com uma ligação à internet, foi adicionada uma validação para evitar que o utilizador tente efetuar log-in na aplicação, avisando-o de que não tem acesso à internet.

Também é efetuada uma verificação que procura na cache se existe uma sessão autenticada para o utilizador em questão – sendo essa verificação validada corretamente, é possível ultrapassar diretamente o ecrã de log-in, entrando diretamente na aplicação.

Todas estas verificações são passíveis de demorar algum tempo, pelo que foi também implementado um Splash Screen que é mostrado enquanto estas ações estão a ser realizadas em background.

2) Módulo de Gestão de Grupos e Notas

A montagem da estrutura de dados na componente Web reduziu drasticamente o tempo de desenvolvimento deste módulo em Android, já que todos os métodos já tinham sido desenvolvidos e as melhores formas de acesso ao Firebase tinham sido estudadas e implementadas.

O desenvolvimento sobre o Firebase utilizando a sua Android SDK é muito semelhante à forma de trabalhar em JavaScript, apenas mudam as bases, que se focam maioritariamente na linguagem Java.

Entre a comunidade Firebase, a forma mais comum de desenvolver aplicações Android é a criação de POJOs (Plain Old Java Objects) – objetos que podem ser instanciados e contêm métodos de *get* e *set* sobre as propriedades que lhes forem atribuídas. Estes POJOs representam os nós raiz da árvore da estrutura de dados Firebase e no caso da plataforma CodeTalk-Android, estes são *Note* e *AllowedGroup*.

Independentemente da implementação Firebase, a prática mais conhecida para apresentar dados na plataforma Android é a utilização de *Views* (sendo as mais utilizadas *ListView* e *GridView*). Este conceito também se aplica ao Firebase, que disponibiliza uma classe denominada *FirebaseListAdapter* no seu SDK Android. Utilizando esta classe como base, foi possível criar dois *Adapters* (que herdam os métodos dessa classe utilizando *extends*) customizados (*GroupsAdapter* e *NotesAdapter*) e passar-lhes os objetos POJO criados anteriormente, assim como os *layouts* correspondentes a cada uma das *ListView*s criadas para apresentar os itens provenientes do Firebase.

A classe *FirestoreListAdapter* trata de toda a sincronização necessária com o Firebase, assim como a gestão de memória disponível na aplicação, fazendo *attach* e *detach* das várias referências Firebase consoante o contexto em que o ciclo de vida da aplicação se encontra – quando o utilizador está a visualizar a lista de grupos, é feito *attach* da referência que está a ser utilizada pelo *GroupsAdapter* e quando o utilizador entra dentro de um grupo para visualizar a lista das suas notas, é feito *detach* da referência *Groups* e *attach* da referência *Notes*.

Para mostrar os dados correspondentes a cada *Activity*, apenas é necessário inicializar tanto a *ListView* como o seu *Adapter* respetivo, fazer *bind* desse *Adapter* à *ListView* e os dados começam a ser sincronizados automaticamente em tempo-real:

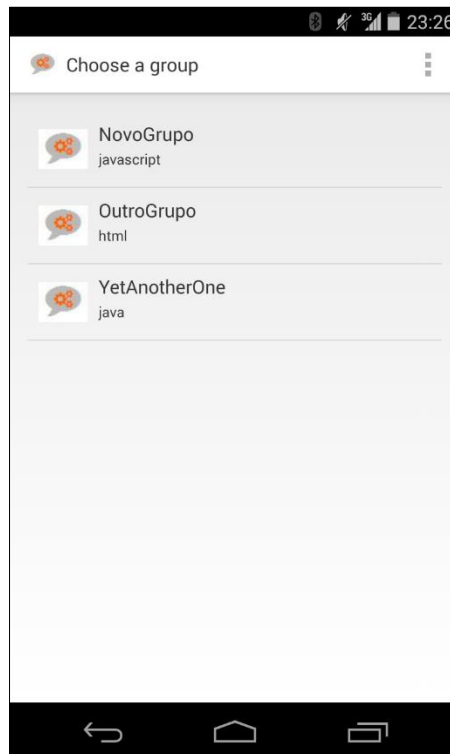


Figura 25 - Android - Lista de Grupos

3) Módulo de Notificações e Subscrições

Este módulo, sendo o mais inovador no contexto da plataforma CodeTalk, demorou bastante tempo a ficar concluído, sendo que mesmo assim, não atingi todos os objetivos aos quais me propus na fase de planeamento para este módulo.

O objetivo principal deste módulo é o desenvolvimento de um sistema de notificações que, através de um serviço que corra em *background* no dispositivo, receba e mostre notificações sobre notas que são adicionadas em grupos aos quais o utilizador subscreveu.

A primeira grande barreira neste desenvolvimento foi a procura de funcionalidades nos métodos e classes nativas Android que fossem capazes de suportar estas funcionalidades. Existem muitos métodos e classes representativas de serviços capazes de correr em *background*, e cada um tem funcionalidades e aplicações distintas, pelo que foram muitas as experiências até encontrar um método que satisfizesse as necessidades.

Depois de alguma pesquisa sobre este assunto, implementei um serviço que, herdando da classe *Service*, possibilita a criação de um processo que corre em paralelo à aplicação, executando os métodos necessários independentemente do estado da aplicação (ligada/desligada).

Neste caso particular, o serviço não necessita de suportar comunicações bidirecionais já que não precisa de comunicar resultados de volta para a aplicação – se tal fosse necessário teria de enveredar pelo caminho dos *BroadcastReceivers* no módulo aplicacional para conseguir estabelecer comunicação com o processo paralelo.

Assim, sempre que o utilizador deseja subscrever um grupo, uma instância do serviço é criada, sendo-lhe passada a referência *Firebase* para o grupo ao qual o utilizador deseja subscrever. Estas referências são guardadas num objeto *Set<String>* à medida que o utilizador vai subscrevendo mais grupos sendo este objeto guardado nas *SharedPreferences* da aplicação. As *SharedPreferences* são o método utilizado na maior parte das aplicações para guardar as preferências (ou definições) do utilizador para uma determinada aplicação – são persistentes e acedíveis através de um sistema *chave-valor*.

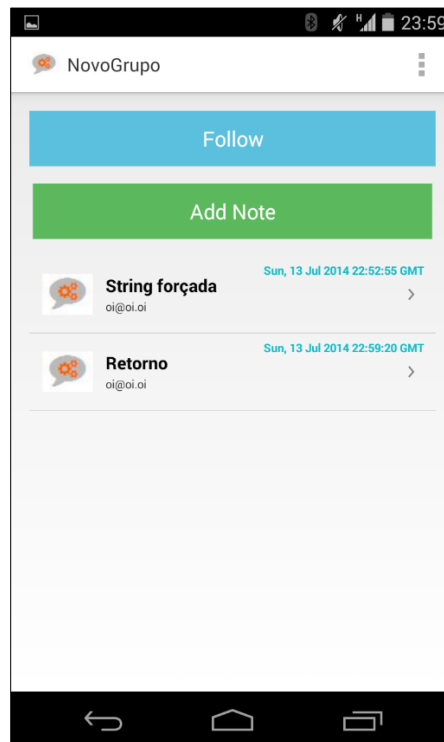


Figura 26 - Android - Página de Grupo

Sempre que é adicionada uma referência *Firebase* ao *Set*, o serviço é reiniciado e no seu início são verificados os respetivos valores existentes nas *SharedPreferences* para criar ligações *Firebase* aos mesmos. Assim que uma nota é adicionada a qualquer uma das referências existentes, o próprio serviço lança uma notificação de sistema para o dispositivo, que pode ser acedida através do menu de notificações, e ao ser pressionada, abre a aplicação no grupo ao qual foi adicionada a informação.

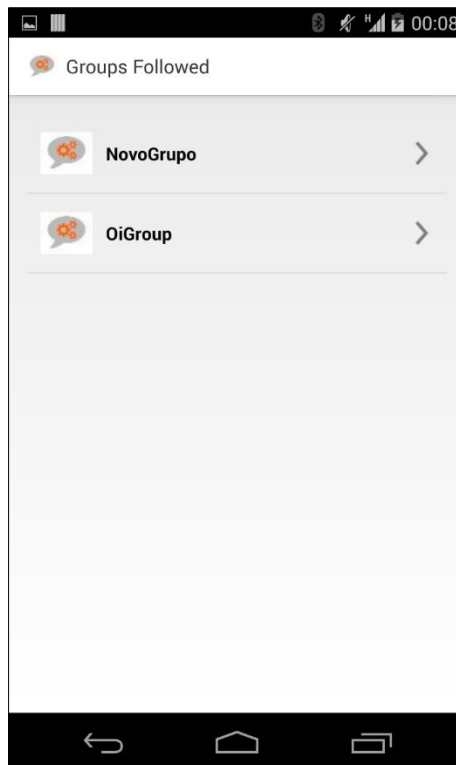


Figura 27 - Android - Grupos Subscritos

Este método de reiniciar o serviço sempre que um grupo é subscrito deve-se ao facto de tentar evitar criar métodos de *long-polling* que verificassem mudanças nas *SharedPreferences*. Nos dispositivos móveis qualquer tipo de *long-polling* é altamente desaconselhável já que ocupa bastante memória e pode levar à terminação do processo que o está a efetuar pelo sistema.

Como já referi, a gestão da memória é fulcral em aplicações Android, pelo que essa gestão tem de ser endereçada devidamente. No entanto, com isto surge o problema que fez com que este módulo não cumprisse totalmente os objetivos definidos inicialmente.

Ao reiniciar o serviço, as referências Firebase também são reinicializadas, o que faz com que o SDK disponibilizado pelo Firebase para Android detete informação já existente na estrutura de dados como nova informação – o serviço não tem qualquer forma de saber se a informação é nova ou não, e despoleta as notificações de novas notas existentes quando realmente são notas já existentes.

Isto fazia com que sempre que o utilizador subscrevia um grupo, fossem lançadas N notificações, tantas quanto as notas já existentes nesse grupo. Para evitar esta situação introduzi um limite no pedido para retornar apenas uma nota nova de cada vez – isto funciona como um *minor workaround* já que continua a trazer apenas uma nota que não é realmente nova, mas pelo menos não notifica tantas quanto as que existem.

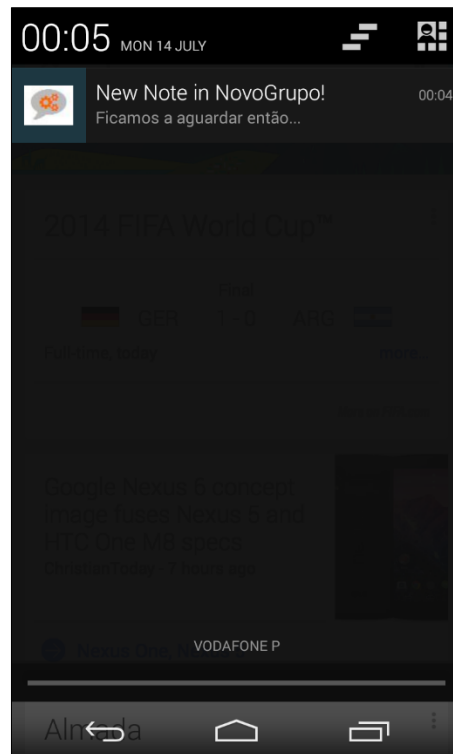


Figura 28 - Android - Notificações

O único *workaround* real para esta limitação do Android SDK do Firebase que encontrei seria a criação de uma base de dados local em SQLite que armazenasse informação existente como uma espécie de *cache*. No entanto, todo este projeto assenta no armazenamento da informação na *cloud* e a implementação de armazenamento de dados local seria ir contra esse princípio fundamental que é a disponibilização da informação remota em tempo-real ao utilizador.

4. Resultados

4.1. Funcionalidades Componente Web

Acesso:

- Sign-up;
- Log-in;
- Log-out;

Grupos:

- Escolher de Linguagem de Programação;
- Adicionar Grupo;
- Ver detalhe de Grupo;
- Abandonar Grupo;
- Adicionar Código (Copy & Paste) (disponível apenas se não bloqueado);
- Adicionar Código (Drag & Drop) (disponível apenas se não bloqueado);
- Gravar Código (disponível para Administração apenas);
- Bloquear Código (disponível para Administração apenas);
- Convidar Utilizador (disponível para Administração apenas);
- Tornar Público / Privado

Notas:

- Adicionar Nota (Título, Conteúdo, Código);
- Adicionar Nota com Seleção de Código;
- Ver detalhe da Nota.

4.2. Funcionalidades Componente Mobile

Acesso:

- Log-in;
- Log-out;

Grupos:

- Ver detalhe de Grupo;
- Subscrever Notificações;

Notas:

- Adicionar Nota (Título, Conteúdo);
- Ver detalhe de Nota;

5. Conclusões e trabalho futuro

Já antes de iniciar o projeto CodeTalk tinha adquirido alguns conhecimentos sobre aplicações Web e bastante curiosidade sobre a forma como as novas tecnologias e frameworks se integravam no desenvolvimento destas aplicações nos dias que correm.

Infelizmente atualmente não disponho de muito tempo para me dedicar à exploração de novas tecnologias. No entanto, este projeto deu-me uma razão para aplicar algum tempo em busca desse objetivo.

Ao longo do curso do projeto tive oportunidade de estudar e aprimorar o meu conhecimento em diversas frameworks utilizadas atualmente – a framework de desenvolvimento front-end Bootstrap, a framework MVC AngularJS, e ferramentas de armazenamento de dados na Cloud, como o Parse e o Firebase. Paralelamente, também tive a oportunidade de melhorar o meu conhecimento sobre o desenvolvimento para a plataforma Android.

Fiquei impressionado pela positiva quando comecei a desenvolver utilizando a framework AngularJS, a qual utiliza a linguagem JavaScript como base para a construção de métodos intuitivos que se integram de forma homogênea com a linguagem HTML facilitando e estendendo as funcionalidades disponíveis no desenvolvimento de aplicações para a Web.

O estudo das plataformas de armazenamento de dados (como o Parse e o Firebase) também foi deveras enriquecedor já que tive a oportunidade de conhecer as vantagens e as desvantagens existentes em cada uma delas. Acabei por me decidir pela utilização do Firebase, que apesar de não emular uma base de dados relacional (como o Parse), conseguia através de uma estrutura de dados hierárquica, disponibilizar uma API tanto para o AngularJS como para a plataforma Android, que se integrava na perfeição com os meus objetivos, possibilitando o desenvolvimento de uma aplicação Web com interação entre utilizadores em tempo real.

Com o projeto CodeTalk concluído, considero que foi uma experiência enriquecedora na qual tive a possibilidade de explorar tecnologias recentes de desenvolvimento Web, descobrir as suas vantagens e desvantagens, e implementá-las de forma a obter um resultado final bastante melhor do que o esperado.

Bibliografia

- i. Ace-Editor, disponível em: <http://ace.c9.io/>
- ii. Android Developer Resources, disponível em:
<http://developer.android.com/develop/index.html>
- iii. AngularFire, API Firebase para AngularJS, disponível em: <http://angularfire.com>
- iv. AngularJS, web framework, disponível em: <https://angularjs.org/>
- v. Bootstrap, web framework, disponível em: <http://getbootstrap.com/>
- vi. Firebase, armazenamento cloud, disponível em: <https://www.firebase.com/>
- vii. NG-Conf, recursos AngularJS, disponível em: <http://ng-conf.org/>
- viii. Parse, armazenamento cloud, disponível em: <https://parse.com/>
- ix. StackOverflow, aconselhamento ao nível do desenvolvimento, disponível em:
<https://www.stackoverflow.com>
- x. Teorema de Brewer, relação Eficiência/Eficácia, disponível em:
http://en.wikipedia.org/wiki/CAP_theorem

ANEXOS

A1. Anexo de Manual Técnico da aplicação

A1.1. Componente Web

Para o desenvolvimento da componente Web foi utilizado o IDE IntelliJ WebStorm 8.0.4.

O código-fonte completo está disponível em:

<https://github.com/jsfrocha/CodeTalk/tree/master/firebaseApp>

O projeto final é constituído pelos seguintes ficheiros:

A1.1.1. Vistas (pasta codetalk-views)

Todas as vistas são carregadas numa DIV existente no ficheiro index.html, na raiz da aplicação:

```
<div class="codetalk-views" ng-view></div>
```

Este ficheiro index.html engloba o carregamento de todos os ficheiros de CSS e JavaScript necessários ao funcionamento da aplicação. Também existem duas DIVs onde são carregadas as barras de navegação (*nav-public* e *nav-private*).

Private

- groupitem.html: Vista de Detalhe do Grupo
- groups.html: Vista de Listagem de Grupos
- nav.html: Vista de Barra de Navegação Privada

Public

- landing.html: Vista de Página Inicial
- nav.html: Vista de Barra de Navegação Pública

Todas as vistas construídas em HTML5 juntamente com CSS. Para integração com os diversos métodos disponíveis na framework AngularJS, também são introduzidos atributos com o prefixo *ng-* nas diversas DIVs, com o objectivo de controlar o seu comportamento de forma dinâmica através do paradigma de *three-way binding* pelo AngularJS e Firebase.

A1.1.2. Scripts (pasta codetalk-scripts)

Aqui encontra-se o centro da aplicação web. Esta pasta contém 4 ficheiros principais e um auxiliar (utils.js) onde está armazenado todo o código AngularJS e AngularFire:

- index-routes.js
- index-controllers.js
- index-directives.js
- index-filters.js
- utils.js

index-routes.js

Este ficheiro necessita de ser inicializado primeiro, já que contem toda a informação necessária ao funcionamento da aplicação.

Aqui é possível encontrar a inicialização da aplicação AngularJS, assim como a injeção dos vários módulos necessários:

```
var app = angular.module('indexApp', ['ngRoute', 'ngCookies', 'firebase']);
```

Seguidamente, utilizarei esta variável (que representa a aplicação AngularJS) para definir configurações, roteamento, controladores (*index-controllers.js*), diretivas (*index-directives.js*) e filtros (*index-filters.js*) na mesma.

A definição do roteamento entre vistas é definida em seguida:

```
app.config(function($routeProvider) {  
    $routeProvider  
        .when('/', {  
            templateUrl: 'codetalk-views/public/landing.html',  
            controller: 'LandingCtrl'  
        })  
        .when('/home', {  
            templateUrl: 'codetalk-views/private/groupitem.html',  
            controller: 'HomeCtrl'  
        })  
        .when('/start', {  
            templateUrl: 'codetalk-views/private/groups.html',  
            controller: 'GroupsCtrl'  
        })  
        .when('/start/:groupName', {  
            templateUrl: 'codetalk-views/private/groupitem.html',  
            controller: 'SingleGroupCtrl'  
        })  
        .otherwise({  
            redirectTo: '/notfound',  
            templateUrl: 'codetalk-views/notfound.html'  
        });  
});
```

Aqui é possível aceder ao URL do browser, e tomar ações consoante o que lá foi introduzido. Por exemplo:

```
.when('/home', {  
    templateUrl: 'codetalk-views/private/groupitem.html',  
    controller: 'HomeCtrl'  
});
```

Significa que quando o browser deteta o sufixo */home*, carrega o controlador *HomeCtrl* na vista *groupitem.html*.

Aqui também é possível definir roteamento para páginas de detalhe de Grupo, já que ao passar o ID do Grupo no URL, este roteador consegue guardá-lo no respetivo controlador, que o usa para mostrar a informação correta:

```
.when('/start/:groupName', {  
  templateUrl: 'codetalk-views/private/groupitem.html',  
  controller: 'SingleGroupCtrl'  
})
```

No método `.run`, abaixo, são definidas funções que correm quando a aplicação inicia. Estas funções são definidas na variável `$rootScope`, que pode ser acedida em todos os controladores que constituem a aplicação, uma espécie de *singleton* global.

A primeira ação a tomar é verificar se existe algum utilizador com sessão ativa, para o redirecionar de acordo com o resultado – se existe, redireciona para a página de grupos, senão, redireciona para a página inicial.

```
$rootScope.auth = $firebaseSimpleLogin(dataRef);  
  
//Get current user  
$rootScope.auth.$getCurrentUser().then(function(user) {  
  if (user) {  
    $location.path('/start');  
  }  
  else {  
    $location.path('/');  
  }  
}, function(error) {  
  console.log(error);  
});
```

Esta função corre uma vez no início da aplicação, porém é também necessário registar um *listener* que detete mudanças no roteamento (como por exemplo se o utilizador carregar no botão *Back* do browser ou introduzir um URL na barra de endereço).

```
//Register Listener to watch route changes  
$rootScope.$on("$routeChangeStart", function (event, next, current) {  
  if ($rootScope.auth.user == null || $rootScope.auth.user == undefined) {  
    //No Logged in user, we should go to Login page  
    if (next.templateUrl == "codetalk-views/public/landing.html"  
        /*next.templateUrl == "codetalk-views/private/groups.html"*/) {  
      //Already going to landing, no redirect needed  
    } else {  
      //Not going to landing, needs to redirect  
      $location.path("/");  
    }  
  }  
  else {  
    if (current.templateUrl == "codetalk-views/private/groups.html") {  
    } else {  
      $location.path('/start');  
    }  
  }  
});
```

Com a função acima é possível detetar essas mudanças no URL e tomar ações consoante o estado da sessão do utilizador.

São também aqui definidas duas funções de simplificação de sintaxe, já que durante a implementação da aplicação é necessário criar várias referências ao Firebase, por via de URL, o que é simplificado chamando as seguintes funções agora associadas à variável `$rootScope`:

```
$rootScope.getFBRef = function (urlAdded) {  
    return $firebase(new Firebase('https://codetalking.firebaseio.com/'+urlAdded));  
}  
  
$rootScope.getNormalFBRef = function (urlAdded) {  
    return new Firebase('https://codetalking.firebaseio.com/'+urlAdded);  
}
```

A função `getFBRef` é utilizada para obter uma referência `AngularFire`, possibilitando a utilização de métodos que são facilmente integráveis com o `AngularJS`. No entanto, esta API é bastante recente, pelo que ainda não estão integrados todos os métodos necessários à implementação desta plataforma, pelo que também é utilizada a criação de referências `Firebase JavaScript` através do método `getNormalFBRef`.

É também atribuída à variável `$rootScope` um método que permite inicializar a funcionalidade de *Drag & Drop* de ficheiros de código no editor. Este método apenas é inicializado aqui para desocupar espaço necessário no ficheiro de controladores – apenas é utilizado uma vez nesse ficheiro, mas como é possível ver na respetiva secção, esse ficheiro cresce de forma exponencial.

```
$rootScope.setupFileDragAndDrop = function (editorDiv, editor) {  
    addFileDragAndDropEventListeners(editorDiv, editor);  
  
    function addFileDragAndDropEventListeners (editorDiv, aceObject) {  
  
        editorDiv.addEventListener('dragover', function(e) {  
            stopEvent(e);  
        });  
  
        editorDiv.addEventListener('drop', function(e) {  
            putFileContentsInAceEditor(e.dataTransfer.files[0], aceObject);  
            stopEvent(e);  
        });  
  
        function putFileContentsInAceEditor(file, aceEditor) {  
            console.log("PutFileContents");  
            var reader, text;  
            reader = new FileReader();  
            reader.onload = (function (file) {  
                text = file.target.result;  
                aceEditor.getSession().setValue(text);  
            });  
            reader.readAsText(file);  
        }  
  
        function stopEvent(e) {  
            e.stopPropagation();  
            e.preventDefault();  
        }  
    }  
}
```

Apesar da necessidade de reutilização de métodos, existe um método utilizado em diversos controladores que não foi possível adicionar à *\$rootScope*:

```
$scope.groupsAlert = {  
  alertType: "",  
  message: "",  
  isShown: false  
};  
  
function showAlert(alertType, message) {  
  $scope.groupsAlert.message = message;  
  $scope.groupsAlert.isShown = true;  
  $scope.groupsAlert.alertType = alertType;  
}  
  
$scope.closeAlert = function () {  
  $scope.groupsAlert.isShown = false;  
  if (!$scope.$$phase) { //SAFE APPLY TO ANGULAR  
    $scope.$apply();  
  }  
}
```

Este conjunto de métodos é utilizado para lançar uma notificação de erro na validação, utilizando as mensagens de alerta disponibilizadas pelo Bootstrap. Apesar de ser utilizado em diversos controladores, não foi possível adicioná-lo à *\$rootScope* no início desta aplicação porque depende de uma noção de *scope* do controlador onde está inserido, o que torna a sua utilização através da *\$rootScope* impossível.

index-controllers.js

Dentro deste ficheiro é possível encontrar todos os controladores utilizados para gerir o comportamento da aplicação:

NavbarCtrl – Controlador que gere o comportamento da Barra de Navegação

Aqui estão localizados os métodos que gerem o estado da sessão do utilizador:

- Sign-up

```
$rootScope.auth.$createUser($scope.signUpData.email, $scope.signUpData.password,
true)//False -> Log-in after sign up ; True -> Don't Login after signup
.then(function(user) {
    var newUserRef = $rootScope.getFBRef('users/'+user.uid);
    //Add user to firebase and angular

    if (!$scope.$$phase) { //SAFE APPLY TO ANGULAR
        $scope.$apply();
    }
    newUserRef.$set({
        id: user.id,
        email: user.email,
        provider: user.provider
    });
    if (!$scope.$$phase) { //SAFE APPLY TO ANGULAR
        $scope.$apply();
    }
    $scope.authLoader = false;
    $scope.signUpData.email = '';
    $scope.signUpData.password = '';
    $scope.authLoader = false;

    angular.element('#signUpModal').modal('hide');
    $location.path('/start');

}, function(error) {
    $scope.authLoader = false;
    showAlert('alert-danger', error.code);
    console.log("Error creating user: "+angular.toJson(error));
})
};
```

- Log-in

```
$scope.submitAuth = function () {  
  if (!$scope.emailAuth.match(emailValidation)) {  
    showAlert('alert-danger', 'Email is not valid');  
  } else {  
    $scope.authLoader = true;  
    $rootScope.auth.$login('password', {  
      email: $scope.emailAuth,  
      password: $scope.passwordAuth  
    }).then(function (user) {  
      if (!$scope.$$phase) { //SAFE APPLY TO ANGULAR  
        $scope.$apply();  
      }  
      $scope.emailAuth = '';  
      $scope.passwordAuth = '';  
      $scope.authLoader = false;  
      $location.path('/start');  
    }, function (error) {  
      if (error.code == 'INVALID_USER') {  
        showAlert('alert-danger', 'User does not exist');  
        $scope.emailAuth = '';  
        $scope.passwordAuth = '';  
        angular.element('#signInEmailId').val('');  
        angular.element('#signInPasswordId').val('');  
        $scope.authLoader = false;  
      }  
    })  
  }  
}
```

- Log-out

```
$scope.logout = function () {  
  $rootScope.auth.$logout();  
  $rootScope.auth.user = null;  
  $location.path('/');  
}
```

GroupsCtrl – Controlador que gere o comportamento da página de Listagem de Grupos

- Criar Grupo

```
$scope.createGroup = function () {

    var suggestName = function (name) {
        return name.replace(/[^a-z0-9]/gi, '');
    };
    var isNameValid = function (name) {
        var regex = /^[^a-z0-9]/ig;
        if (name.match(regex)) return false;
        else return true;
    }

    $scope.groupsAlert = {
        alertType: "",
        message: "",
        isShown: false
    };

    var currentUserGroupRef = $rootScope.getFBRef('users/' +
$rootScope.auth.user.uid + '/allowedGroups');

    var newGroupName = $scope.newGroupName;
    var selectedMode = $scope.selectedMode;

    var fullGroupName = newGroupName + '_' + currentUserId;

    if (!!newGroupName) {
        if ($scope.selectedMode != "NONE") {
            if (isNameValid(newGroupName)) { //Happy Path
                currentUserGroupRef.$add({
                    name: newGroupName,
                    mode: selectedMode,
                    createdBy: currentUserId,
                    fullName: fullGroupName
                })
                .then(function (ref) {
                    var groupRef = $rootScope.getFBRef('groups/' + newGroupName
+ '_' + currentUserId);
                    groupRef.$set({
                        mode: selectedMode,
                        isPrivate: true,
                        isCodeLocked: false,
                        code: '',
                        createdBy: currentUserId
                    });
                }, function(err) {
                    if (err.code == "PERMISSION_DENIED") {
                        showAlert('alert-danger', 'You already created a group
with that name.');
```

```
                        angular.element('#new-groupname-input').val('');
                    }
                });
                angular.element('#new-groupname-input').val('');
            }
            else { //Invalid Characters
                showAlert('alert-danger', "Invalid characters, try: '" +
suggestName(newGroupName) + "'");
                angular.element('#new-groupname-input').val('');
            }
        }
        else { //No selected Mode
            showAlert('alert-danger', "A mode needs to be selected");
            angular.element('#new-groupname-input').val('');
        }
    }
    else { //Group name empty
        showAlert('alert-danger', "The group needs a name");
        angular.element('#new-groupname-input').val('');
    }
}
```

Esta é a função utilizada para criar um Grupo. Apesar de longa, é constituída maioritariamente de validações, onde se o nome não corresponde ao válido, são feitas sugestões de nomes válidos. O excerto estritamente necessário para criar um grupo na plataforma é apenas o seguinte:

```
var currentUserGroupsRef =  
$rootScope.getFBRef('users/'+$rootScope.auth.user.uid+'/allowedGroups');
```

```
currentUserGroupRef.$add({  
  name: newGroupName,  
  mode: selectedMode,  
  createdBy: currentUserId,  
  fullName: fullGroupName  
})  
  .then(function (ref) {  
    var groupRef = $rootScope.getFBRef('groups/' + newGroupName + '_' +  
currentUserId);  
    groupRef.$set({  
      mode: selectedMode,  
      isPrivate: true,  
      isCodeLocked: false,  
      code: '',  
      createdBy: currentUserId  
    });  
  });
```

Começando por definir a referência para o caminho Firebase que contém os *allowedGroups*, é então adicionada uma entrada a essa referência utilizando o método *\$add* e introduzindo os respetivos campos referidos na Estrutura de Dados. Ao utilizar este método é criado um *hash* como nome do grupo.

Seguidamente, quando a *promise* resolve (*.then*) é adicionado esse mesmo grupo à lista de grupos utilizando a função *\$set* que cria um grupo com o nome *newGroupName_currentUserId*, conforme especificado na referência *groupRef*.

SingleGroupCtrl – Controlador que gere a página de Detalhe de Grupo

De todos, este é o controlador mais extenso, já que necessita de gerir diversos componentes nesta vista. Aqui, existem 3 instâncias do Ace-Editor – uma para a página inicial, uma para o diálogo de inserção de nota, e outra para o diálogo de visualização de nota.

Ao ser carregado, o controlador começa por obter o ID do Grupo que foi passado no URL:

```
$scope.currentGroupFull = $routeParams.groupName;  
$scope.currentGroup = $routeParams.groupName.split("_")[0];
```

Inicializa as 3 instâncias do Ace-Editor:

```
$scope.editor = ace.edit("code-editor");
$scope.editor.setTheme("ace/theme/github");

$scope.modalEditor = ace.edit("modal-code-editor");
$scope.modalEditor.setTheme("ace/theme/github");

$scope.viewModalEditor = ace.edit("view-modal-code-editor");
$scope.viewModalEditor.setTheme("ace/theme/github");
$scope.viewModalEditor.setReadOnly(true);
```

E inicializa os *listeners* para *Drag & Drop* no Editor utilizando os métodos definidos anteriormente na *\$rootScope*:

```
$rootScope.setupFileDragAndDrop(editorDiv, $scope.editor);
$rootScope.setupFileDragAndDrop(modalEditorDiv, $scope.modalEditor);
```

Após concluir as inicializações, é necessário fazer uma *query* ao Firebase (1) que retorna uma lista dos grupos existentes (2) onde depois será possível encontrar o grupo atual (3) e inicializar o resto dos controlos:

```
var groupsRef = $rootScope.getNormalFBRef('groups');
1. groupsRef.once('value', function(snap) {
  2. snap.forEach(function(child) {
    3. if (child.name() == $scope.currentGroupFull) {
      //Found current group
      if (child.val().createdBy == currentUserId) $scope.isUserAdmin = true;

      //Set editor modes
      $scope.groupMode = child.val().mode;
      $scope.editor.getSession().setMode("ace/mode/"+$scope.groupMode);
      $scope.modalEditor.getSession().setMode("ace/mode/"+$scope.groupMode);

$scope.viewModalEditor.getSession().setMode("ace/mode/"+$scope.groupMode);

      if (child.val().isCodeLocked == true) {
        //Lock editor if needed
        $scope.codeLocked = true;
        $scope.editor.setReadOnly(true);
      }
      else {
        //Unlock editor if needed
        $scope.codeLocked = false;
        $scope.editor.setReadOnly(false);
      }

      if (child.val().isPrivate) {
        //Set private if needed
        $scope.isGroupPrivate = true;
      }
      else {
        //Set public if needed
        $scope.isGroupPrivate = false;
      }

      if (child.val().code != '') {
        //Set code in Editor - Code from Firebase
        $scope.editor.getSession().setValue(child.val().code);
      }
    }
  });
  if (!$scope.$$phase) { //SAFE APPLY TO ANGULAR
    $scope.$apply();
  }
});
```


Com os controlos inicializados, é então possível registar as funções na *\$scope* para poderem ser acedidas pelo utilizador:

- Save Code

```
$scope.saveCode = function() {  
    var code = $scope.editor.getSession().getValue();  
    var currentGroup = $routeParams.groupName;  
    var currentGroupRef = $rootScope.getFBRef('groups/'+currentGroup);  
  
    if (!!code) {  
        currentGroupRef.$update({  
            code: code  
        });  
        alert('Code Saved!');  
    } else {  
        alert('There is no Code to Save');  
    }  
};
```

Ao ser chamada, esta função guarda o código actualmente introduzido no Editor, através da função *\$update*, no Firebase.

- Lock Code

```
$scope.lockCode = function() {  
    var currentGroup = $routeParams.groupName;  
    var currentGroupRef = $rootScope.getFBRef('groups/'+currentGroup);  
  
    currentGroupRef.$update({  
        isCodeLocked: true  
    });  
  
    $scope.editor.setReadOnly(true);  
    $scope.codeLocked = true;  
  
    if (!$scope.$$phase) { //SAFE APPLY TO ANGULAR  
        $scope.$apply();  
    }  
};
```

Esta função faz update da propriedade *isCodeLocked* no objeto Firebase e faz *setReadOnly* ao Editor, bloqueando-o.

- Unlock Code

```
$scope.unlockCode = function () {  
    var currentGroup = $routeParams.groupName;  
    var currentGroupRef = $rootScope.getFBRef('groups/'+currentGroup);  
  
    $scope.saveLoader = true;  
  
    $scope.editor.setReadOnly(false);  
  
    currentGroupRef.$update({  
        isCodeLocked: false  
    });  
  
    $scope.codeLocked = false;  
  
    if (!$scope.$$phase) { //SAFE APPLY TO ANGULAR  
        $scope.$apply();  
    }  
  
    $scope.saveLoader = false;  
  
};
```

A função *Unlock Code* executa o contrário da anterior, desbloqueando o Editor e guardando o novo valor da propriedade *isCodeLocked* no Firebase.

- View Note

```
$scope.viewNote = function (noteKey) {  
    $scope.noteToView =  
    $rootScope.getFBRef('groups/'+$scope.currentGroupFull+'/notes/'+noteKey);  
  
    if (!$scope.$$phase) { //SAFE APPLY TO ANGULAR  
        $scope.$apply();  
    }  
  
    $scope.viewModalEditor.getSession().setValue($scope.noteToView.code);  
  
    $scope.viewTitle = $scope.noteToView.title;  
    $scope.viewContent = $scope.noteToView.content;  
  
    var date = new Date($scope.noteToView.createdAt);  
  
};
```

Esta função é chamada quando o utilizador pretende visualizar uma nota individual. Faz uma chamada ao Firebase com o objetivo de obter a Nota em questão e introduz o código que nela tinha sido inserido (se existir) no Editor do diálogo respetivo, assim como os valores referentes às restantes propriedades.

Por último, também existe aqui um *listener* que monitoriza o Editor principal:

```
$scope.editor.on('blur', function() {  
  
    $scope.modalEditor.getSession().setValue($scope.editor.session.getTextRange($scope.editor.getSelectionRange()));  
  
});
```

Este *listener* faz com que, sempre que o utilizador tirar o foco do Editor, o código selecionado seja transportado para o Editor presente no diálogo de criação de Nota. Apesar deste Editor estar escondido, isto é feito para assim que o utilizador carregue no botão de Criar Nota, o código selecionado esteja disponível no diálogo que seguidamente aparecerá.

AddNoteCtrl – Controlador que gere a adição de notas

Apesar de este controlador também utilizar a vista de Detalhe de Grupo, a adição de notas é feita a partir de um diálogo. Assim sendo, é necessário existir um controlador separado nessa secção para gerir as ações lá tomadas. Apenas contém uma função, Adicionar Nota:

```
$scope.addNote = function () {  
    if (typeof $scope.newNote != 'undefined' && $scope.newNote.noteTitle &&  
        $scope.newNote.noteContent) {  
        var title = $scope.newNote.noteTitle;  
        var content = $scope.newNote.noteContent;  
        var noteCode = $scope.modalEditor.getSession().getValue();  
        console.log("NoteCode. "+noteCode);  
        var currentNoteRef =  
$rootScope.getFBRef('notes/'+$scope.currentGroup+'_'+title);  
        var notesRef = $rootScope.getNormalFBRef('notes');  
        var existingNotesArray = new Array();  
  
        notesRef.once('value', function(snap) {  
            snap.forEach(function(child) {  
                existingNotesArray.push(child.name());  
            });  
  
            var existingNotes = new Array();  
            for (var i = 0; i < existingNotesArray.length; i++) {  
                if (existingNotesArray[i].split('_')[0] == $scope.currentGroup)  
                    existingNotes.push(existingNotesArray[i].split('_')[1]);  
            }  
            if (existingNotes.indexOf(title) == -1) {  
                var currentDate = new Date();  
                var dateToAdd = currentDate.toUTCString();  
                var currentUserEmail = $rootScope.auth.user.email;  
  
                //Set Note in /Notes -> [GROUPNAME]_[NOTETITLE]  
                currentNoteRef.$set({  
                    content: content,  
                    code: noteCode,  
                    createdAt: dateToAdd,  
                    createdBy: currentUserEmail  
                });  
                var groupNotesRef =  
$rootScope.getFBRef('groups/'+$scope.currentGroupFull+'/notes');  
  
                //Add Note to Groups/Notes  
                groupNotesRef.$add({  
                    title: title,  
                    content: content,  
                    code: noteCode,  
                    createdAt: dateToAdd,  
                    createdBy: currentUserEmail,  
                    inGroup: $scope.currentGroupFull  
                });  
                $scope.newNote.noteTitle = "";  
                $scope.newNote.noteContent = "";  
                $scope.modalEditor.getSession().setValue('');  
                angular.element('#addNoteModal').modal('hide');  
            }  
            else {  
                $scope.showAlert('alert-danger', 'Note "'+title+'" already exists in  
this group');  
            }  
        });  
    }  
    else {  
        $scope.showAlert('alert-danger', 'Both fields are mandatory');  
    }  
};
```

À semelhança do código utilizado para criar um grupo, também este código é construído maioritariamente de validações, verificando as notas existentes e impedindo o utilizador de criar uma nota com o mesmo título de uma já existente.

Para isto, é criado um *Array* onde são introduzidos os títulos das Notas existentes, sendo a nova Nota apenas criada se:

```
existingNotes.indexOf(title) == -1
```

Para criar a nota, é utilizada a mesma lógica da criação de novo Grupo:

```
var currentDate = new Date();
var dateToAdd = currentDate.toUTCString();
var currentUserEmail = $rootScope.auth.user.email;

//Set Note in /Notes -> [GROUPNAME]_[NOTETITLE]
currentNoteRef.$set({
  content: content,
  code: noteCode,
  createdAt: dateToAdd,
  createdBy: currentUserEmail
});
var groupNotesRef =
$rootScope.getFBRef('groups/'+$scope.currentGroupFull+'/notes');

//Add Note to Groups/Notes
groupNotesRef.$add({
  title: title,
  content: content,
  code: noteCode,
  createdAt: dateToAdd,
  createdBy: currentUserEmail,
  inGroup: $scope.currentGroupFull
});

$scope.newNote.noteTitle = "";
$scope.newNote.noteContent = "";
$scope.modalEditor.getSession().setValue('');
angular.element('#addNoteModal').modal('hide');
```

DeleteGroupCtrl – Controlador que gere o abandono do grupo por parte do Utilizador

Também este controlador é acedido diretamente por um diálogo, sendo necessário estar separado dos restantes. Aqui é verificado se o utilizador introduziu o nome certo nesse diálogo, e se sim, remove esse grupo da lista de grupos permitidos do utilizador.

```
$scope.deleteGroup = function () {  
    var groupNameConfirmation = $scope.deleteGroupConfirm;  
    var currentGroup = $scope.currentGroup;  
    var currentUser = $rootScope.auth.user.uid;  
  
    if (groupNameConfirmation == currentGroup) {  
        var groupsRef = $rootScope.getFBRef('users/'+currentUser+'/allowedGroups');  
        //Delete from /AllowedGroups  
        var keys = groupsRef.$getIndex();  
        keys.forEach(function(key, i) {  
            if (groupsRef[key].name == currentGroup) {  
                groupsRef.$remove(key);  
            }  
        });  
        angular.element('#deleteGroupModal').modal('hide');  
        $location.path('/start');  
    }  
    else {  
        $scope.showAlert('alert-danger', 'The group name inserted does not match the  
current group!');  
    }  
};
```

InviteFriendsCtrl – Controlador que gere o convite de Utilizadores para o Grupo

Acedido a partir do diálogo *Invite Friends*, este controlador mostra uma lista dos utilizadores da plataforma CodeTalk e permite ao utilizador convidar outros utilizadores para o seu Grupo.

Ao escolher um utilizador, é-lhe adicionado o grupo atual à lista de *allowedGroups*, permitindo a sua visualização e acesso:

```
$scope.addUserToGroup = function (userId, userEmail) {

    var userRef = $rootScope.getFBRef('users/'+userId+'/allowedGroups');
    var userNormalRef = $rootScope.getNormalFBRef('users/'+userId+'/allowedGroups');
    var currentGroupFull = $routeParams.groupName;
    var currentGroup = currentGroupFull.split("_")[0];
    var adminUser = $rootScope.auth.user.uid;
    var groupToAddTo = {};
    var groupsRef = $rootScope.getNormalFBRef('groups');
    var fullName = currentGroup + '_' + adminUser;
    var alreadyHasGroup = false;

    groupsRef.once('value', function(snap) {
        snap.forEach(function(child) {
            if (child.name() == fullName) {
                groupToAddTo = child;
            }
        });
        userNormalRef.once('value', function(snap) {
            snap.forEach(function(child) {
                if (child.val().name == currentGroup) {
                    alreadyHasGroup = true;
                }
            });
            if (!alreadyHasGroup) {
                userRef.$add({
                    createdBy: groupToAddTo.val().createdBy,
                    mode: groupToAddTo.val().mode,
                    name: currentGroup,
                    fullName: currentGroupFull
                });
                $scope.showAlert('alert-success', 'User '+userEmail+' was added to
group '+currentGroup+'.');
            }
            else {
                $scope.showAlert('alert-danger', 'User '+userEmail+' is already in
this group');
            }
        });
    });
};
```

index-directives.js

Através do uso de *directives*, o AngularJS permite que sejam atribuídos *templates* HTML a certas DIVs através de uma referência que cria a um determinado atributo definido pelo utilizador.

Neste ficheiro estão definidas as *directives* que permitem a substituição das DIVs *nav-private* e *nav-public* pelos *templates* HTML corretos, no *index.html*.

Barra de Navegação Pública

```
app.directive('navpublic', function(){
  // Runs during compile
  return {
    scope: true,
    restrict: 'A',
    templateUrl: 'codetalk-views/public/nav.html',
    link: function(scope, element, attrs) {

    }
  };
});
```

É acedida no ficheiro *index.html* através de:

```
<div id="nav-public" navpublic ng-show="!auth.user"></div>
```

Barra de Navegação Privada

```
app.directive('navprivate', function(){
  // Runs during compile
  return {
    scope: true,
    restrict: 'A',
    templateUrl: 'codetalk-views/private/nav.html',
    link: function(scope, element, attrs) {

    }
  };
});
```

É acedida no ficheiro *index.html* através de:

```
<div id="nav-private" navprivate ng-show="auth.user"></div>
```

O estado de visualização destas DIVs é controlado pelo atributo *ng-show* que mostra a DIV respetiva consoante a avaliação da expressão *auth.user* que determina se o utilizador que está a ver a página tem ou não sessão ativa.

index-filters.js

Neste ficheiro, são tipicamente definidos os *filters* de AngularJS. Estes *filters* não são necessariamente filtros, mas opções que se podem adicionar a um *ng-repeat*. O *ng-repeat* é uma forma de instanciar um *template* HTML uma vez por cada item existente numa *collection*.

Isto é útil para mostrar as listas tanto de Grupos (página Listagem de Grupos), Utilizadores (diálogo Convidar Utilizadores) ou Notas (página Detalhe de Grupo). No entanto, o *ng-repeat* devolve os itens pela ordem em que foram inseridos no Firebase – isto faz com que a vista Detalhe de Grupo não seja *user-friendly*, já que as novas notas estão nas últimas posições da lista.

Assim sendo, foi criado um *custom filter* que permite a reordenação dos itens da lista à medida que são mostrados com o *ng-repeat*, fazendo com que o primeiro item da lista seja o mais recente:

```
app.filter('reverse', function(){
  function toArray(list) {
    var k, out = [];
    if (list) {
      if (angular.isArray(list)) {
        out = list;
      }
      else if (typeof (list) === 'object' ) {
        for (k in list) {
          if (list.hasOwnProperty(k)) { out.push(list[k]); }
        }
      }
    }
    return out;
  }
  return function(items) {
    return toArray(items).slice().reverse();
  }
});
```

Este filtro é utilizado da seguinte forma:

```
ng-repeat="note in notes | orderByPriority | reverse"
```

utils.js

Neste ficheiro são inicializados os modos que o Ace-Editor suporta, assim como uma função que fica *bound* ao *document* de todas as páginas mostradas.

Este método faz com que o browser não detete a função normal do *Backspace*, de voltar atrás nas páginas. Isto necessita de ser desabilitado já que, nos testes de usabilidade notou-se que grande parte dos utilizadores tentavam utilizar o *Backspace* quando o Editor está em Read Mode. Isto fazia com que o browser detetasse o *Backspace* como forma de voltar atrás uma página.

A1.2. Componente Mobile

Para o desenvolvimento da componente Mobile, foi utilizado o IDE Android Studio 0.8.2 (Beta).

O código-fonte completo está disponível em: <https://github.com/jsfrocha/CodeTalk-Android>

O projeto final é constituído dos seguintes ficheiros:

A1.2.1. Application

CodeTalk

A classe *CodeTalk* serve de *singleton* global da aplicação, estendendo a classe *Application* do Android SDK. Sempre que necessário, é acedida para guardar ou obter valores necessários pelas *activities*. Um exemplo disto são os dados da sessão atual do utilizador.

A1.2.2. Model

Nas classes que contêm modelos, são referenciadas as assinaturas das mesmas, cada uma com o propósito de representar um objeto necessário de ser listado na aplicação através de *ListView*s. Em todas elas, é necessário existir um construtor privado para que o *adapter* utilizado pelo Firebase (*FirestoreListAdapter*) funcione como previsto:

AllowedGroup

```
public class AllowedGroup {

    private String createdBy;
    private String fullName;
    private String mode;
    private String name;

    @SuppressWarnings("unused")
    private AllowedGroup() { }

    @SuppressWarnings("unused")
    AllowedGroup(String createdBy, String fullName, String mode, String name) {
        this.createdBy = createdBy;
        this.fullName = fullName;
        this.mode = mode;
        this.name = name;
    }

    public String getCreatedBy() {
        return createdBy;
    }

    public String getFullName() {
        return fullName;
    }

    public String getMode() {
        return mode;
    }

    public String getName() {
        return name;
    }
}
```

FollowedGroup

```
public class FollowedGroup {  
  
    private String fullName;  
    private String mode;  
    private String name;  
  
    @SuppressWarnings("unused")  
    private FollowedGroup() { }  
  
    @SuppressWarnings("unused")  
    FollowedGroup(String fullName, String mode, String name) {  
        this.fullName = fullName;  
        this.mode = mode;  
        this.name = name;  
    }  
  
    public String getFullName() {  
        return fullName;  
    }  
  
    public String getMode() {  
        return mode;  
    }  
  
    public String getName() {  
        return name;  
    }  
}
```

Note

```
public class Note {  
    private String code;  
    private String content;  
    private String createdAt;  
    private String createdBy;  
    private String title;  
    private String inGroup;  
  
    @SuppressWarnings("unused")  
    private Note() { }  
  
    @SuppressWarnings("unused")  
    Note(String code, String content, String createdAt, String createdBy, String title)  
    {  
        this.code = code;  
        this.content = content;  
        this.createdAt = createdAt;  
        this.createdBy = createdBy;  
        this.title = title;  
        this.inGroup = inGroup;  
    }  
  
    public String getCode() { return code; }  
    public String getContent() { return content; }  
    public String getCreatedAt() { return createdAt; }  
    public String getCreatedBy() { return createdBy; }  
    public String getTitle() { return title; }  
    public String getInGroup() { return inGroup; }  
}
```

A1.2.3. Adapters

Os *adapters* funcionam como uma ponte entre os dados provenientes do Firebase e as Vistas existentes na aplicação Android. Os *adapters* utilizados nesta aplicação foram construídos de acordo com as especificações indicadas na API do Firebase.

Ambos os *adapters* estendem o *adapter* disponibilizado pelo Firebase (*FirebaseListAdapter*) sendo que cada um deles estende este *adapter* indicando qual o tipo de objeto sobre o qual querem trabalhar:

```
public class GroupsAdapter extends FirebaseListAdapter<AllowedGroup>
```

```
public class NotesAdapter extends FirebaseListAdapter<Note>
```

Seguidamente, apenas necessitam de popular a Vista com os dados recebidos a partir do *FirebaseListAdapter*. Estes dados são recebidos e atualizados nas respetivas vistas em tempo real em múltiplos clientes, utilizando todas as potencialidades do Firebase.

GroupsAdapter

```
public GroupsAdapter(Query ref, Activity activity, int layout) {
    super(ref, AllowedGroup.class, layout, activity);
}

@Override
protected void populateView(View view, AllowedGroup group) {
    //Map a Group object to an entry in our listview
    String name = group.getName();
    String mode = group.getMode();
    String fullName = group.getFullName();

    TextView tvName = (TextView) view.findViewById(R.id.lvItemGroupName);
    TextView tvMode = (TextView) view.findViewById(R.id.lvItemGroupMode);
    TextView tvFull = (TextView) view.findViewById(R.id.lvItemGroupFullName);

    tvName.setText(name);
    tvMode.setText(mode);
    tvFull.setText(fullName);
}
```

NotesAdapter

```
public NotesAdapter(Query ref, Activity activity, int layout) {
    super(ref, Note.class, layout, activity);
}

@Override
protected void populateView(View view, Note note) {

    String content = note.getContent();
    String createdAt = note.getCreatedAt();
    String createdBy = note.getCreatedBy();
    String title = note.getTitle();

    TextView tvContent = (TextView) view.findViewById(R.id.lvItemNoteContent);
    TextView tvCreatedAt = (TextView) view.findViewById(R.id.lvItemNoteDate);
    TextView tvCreatedBy = (TextView) view.findViewById(R.id.lvItemNoteUser);
    TextView tvTitle = (TextView) view.findViewById(R.id.lvItemNoteTitle);

    tvContent.setText(content);
    tvCreatedAt.setText(createdAt);
    tvCreatedBy.setText(createdBy);
    tvTitle.setText(title);

}
```

A1.2.4. Services

Com a necessidade de correr métodos de monitorização e lançamento de notificações à margem da aplicação, recorreu-se à utilização de um serviço Android, que possibilita a criação de um módulo da aplicação que está sempre ativo, independentemente do facto de o utilizador ter a aplicação a correr.

FirebaseBackgroundService

Este serviço é um componente fundamental da aplicação Android, já que é aqui que é monitorizado o estado de subscrição dos Grupos, assim como a existência ou não de novas notas no mesmo e o lançamento de notificações se existirem.

Ao longo do serviço é utilizado por diversas vezes a classe *SharedPreferences* do Android SDK. É uma classe que permite guardar pares *chave-valor* e é persistente e independente do ciclo de vida da aplicação – isto permite que sejam transportados valores entre os métodos do serviço e até para fora do mesmo.

O serviço é dividido em 3 componentes, inerentemente ligadas ao seu ciclo de vida:

onCreate() – Este método é o primeiro a ser executado, assim que a aplicação lança o serviço através do método *startService()*

```
mShared = getSharedPreferences("pt.ulht.codetalk", Context.MODE_PRIVATE);
this.childListener = new ChildEventListener() {
    @Override
    public void onChildAdded(DataSnapshot dataSnapshot, String s) {
        Note n = dataSnapshot.getValue(Note.class);
        String title = "New Note in "+n.getInGroup().split("_")[0]+"!";
        String description = n.getContent();
        postNotif(title, description, n.getInGroup());
    }

    @Override
    public void onChildChanged(DataSnapshot dataSnapshot, String s) {

    }

    @Override
    public void onChildRemoved(DataSnapshot dataSnapshot) {

    }

    @Override
    public void onChildMoved(DataSnapshot dataSnapshot, String s) {

    }

    @Override
    public void onCancelled(FirebaseError firebaseError) {
        Log.d("FBERROR", firebaseError.toString());
    }
};
```

Aqui, é inicializada a variável *mShared* para conter os valores existentes nas *SharedPreferences*, e é utilizada ao longo do ciclo de vida do serviço. Além disso, é também inicializado um *ChildEventListener()* que será utilizado para trazer novas Notas detetadas no Firebase.

Quando é detectada uma nova nota, chama o método *postNotif()* que lança a notificação para o utilizador com o conteúdo da nova nota.

onStartCommand() – Lançado logo após o fim do método *onCreate()*

```
followedGroups = mShared.getStringSet("followedGroups", null);

if (followedGroups != null) {
    for (String group : followedGroups) {
        Log.d(SERVICE_TAG, "Service Listen Group: " + group);
        groupRef = new Firebase("https://codetalking.firebaseio.com/groups/" + group +
"/notes");
        groupRef.endAt().limit(1).addChildEventListener(childListener);
    }
}
```

Assim que este método é lançado, é carregado um *Set* (*followedGroups*) com informação das *SharedPreferences*, proveniente da *Activity* que lançou o serviço. Este *Set* contém pares chave-valor que referencia os Grupos que o utilizador subscreveu.

De seguida, para cada um dos grupos, cria uma referência Firebase e adiciona-lhe uma instância do *Listener* criado no método *onCreate()* para que seja possível lançar notificações referentes a todos os grupos que o utilizador está a seguir.

postNotif() – Método independente do ciclo de vida do serviço, apenas utilizado para lançar notificações

```
Log.d("POSTNOTIF", intentDestination);
Context mContext = getApplicationContext();
Intent notificationIntent = new Intent(mContext, GroupActivity.class);
notificationIntent.putExtra("groupName", intentDestination);
PendingIntent contentIntent = PendingIntent.getActivity(mContext, 0, notificationIntent,
PendingIntent.FLAG_UPDATE_CURRENT);
NotificationManager mNotificationManager = (NotificationManager)
getSystemService(Context.NOTIFICATION_SERVICE);
Notification.Builder builder = new Notification.Builder(mContext);

builder.setContentIntent(contentIntent)
    .setSmallIcon(R.drawable.ic_launcher)
    .setWhen(System.currentTimeMillis())
    .setContentTitle(title)
    .setContentText(description)
    .setAutoCancel(true);

Notification n = builder.build();

mNotificationManager.notify(notificationsSent++, n);
```

Neste método é construída a notificação a ser lançada. Propriedades como o seu título, descrição e ícone são adicionadas. Também é adicionado um *PendingIntent* que referencia o nome do Grupo para que seja possível navegar para a página do mesmo quando se clica na notificação.

A1.2.5. Activities

As *activities* também são parte integral da aplicação, já que à semelhança dos controladores no AngularJS, são elas que controlam as Vistas da aplicação Android. São declaradas no ficheiro *AndroidManifest.xml*, juntamente com as permissões que a aplicação requer, e os themes utilizados na mesma.

SplashActivity

Esta *activity* é declarada como *MAIN* no *AndroidManifest*:

```
<activity
    android:name=".activities.SplashActivity"
    android:excludeFromRecents="true"
    android:label="@string/app_name"
    android:noHistory="true"
    android:screenOrientation="portrait"
    android:theme="@android:style/Theme.Holo.Light.NoActionBar.Fullscreen" >
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />

        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

Assim sendo, inicia quando a aplicação é lançada, mostrando um ecrã com o logotipo da aplicação, enquanto os dados referentes à sessão do utilizador estão a ser carregados.

Aqui, consoante o estado da sessão do utilizador, é decidido qual será a próxima *activity* para a qual o utilizador será redirecionado:

```
app.getAuthClient().checkAuthStatus(new SimpleLoginAuthenticatedHandler() {  
    @Override  
    public void authenticated(com.firebase.simplelogin.enums.Error error, User user) {  
        if (error != null) {  
            Log.d(ASYNC_TAG, "Error: " + error);  
        }  
        else if (user == null) {  
            Log.d(ASYNC_TAG, "No user logged in");  
            Intent i = new Intent(app, MainActivity.class);  
            i.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);  
            startActivity(i);  
        }  
        else {  
            Log.d(ASYNC_TAG, "User logged in");  
            app.setCurrentUser(user);  
            app.setCurrentUserId(user.getUserId());  
            app.setCurrentUserEmail(user.getEmail());  
            Intent i = new Intent(app, StartActivity.class);  
            i.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);  
            startActivity(i);  
        }  
    }  
});
```

Se o utilizador não tiver uma sessão ativa, então é redirecionado para a *MainActivity* para efetuar o log-in, senão é redirecionado para a *StartActivity*.

MainActivity

Nesta *activity*, o utilizador deve fazer log-in utilizando os campos para o efeito.

No método *onCreate()*, lançado quando a *activity* inicia, são feitas as inicializações necessárias:

```
setContentView(R.layout.activity_main);  
  
CodeTalk app = (CodeTalk) getApplication();  
  
final EditText etEmail = (EditText) findViewById(R.id.etEmail);  
final EditText etPassword = (EditText) findViewById(R.id.etPassword);  
Button btnLogin = (Button) findViewById(R.id.btnLogin);  
  
btnLogin.setOnClickListener(new View.OnClickListener() {  
    @Override  
    public void onClick(View v) {  
        CodeTalk codeTalkApp = (CodeTalk) getApplication();  
        String email;  
        String password;  
  
        email = etEmail.getText().toString();  
        password = etPassword.getText().toString();  
  
        if (!email.equalsIgnoreCase("") && !password.equalsIgnoreCase(""))  
            loginUser(codeTalkApp, etEmail.getText().toString(),  
etPassword.getText().toString());  
    }  
});
```

É também implementado um *listener* no botão de log-in que despoleta a seguinte função quando pressionado:

```
final SimpleLogin authClient = application.getAuthClient();

authClient.checkAuthStatus(new SimpleLoginAuthenticatedHandler() {
@Override
public void authenticated(com.firebase.simplelogin.enums.Error error, User user) {
    if (error != null) {
        Log.d(LOGIN_TAG, "Error: " + error);
    } else if (user == null) {
        Log.d(LOGIN_TAG, "No logged in user, logging-in.");
        authClient.loginWithEmail(email, password, new SimpleLoginAuthenticatedHandler()
        {
            @Override
            public void authenticated(Error error, User user) {
                if (error != null) {
                    if (Error.UserDoesNotExist == error) {
                        Log.d(LOGIN_TAG, "User does not exist");
                    } else {
                        Log.d(LOGIN_TAG, "Error: " + error);
                    }
                }
                else {
                    Log.d(LOGIN_TAG, "User logged into firebase!");
                    application.setCurrentUser(user);
                    application.setCurrentUserId(user.getUserId());
                    application.setCurrentUserEmail(user.getEmail());
                    Intent i = new Intent(MainActivity.this, StartActivity.class);
                    startActivity(i);
                }
            }
        });
    } else {
        Log.d(LOGIN_TAG, "User is already logged-in");
        application.setCurrentUser(user);
        application.setCurrentUserId(user.getUserId());
        application.setCurrentUserEmail(user.getEmail());
        Intent i = new Intent(MainActivity.this, StartActivity.class);
        startActivity(i);
    }
});
```

Esta função faz uma chamada ao Firebase para fazer log-in ao utilizador, devolvendo um objeto *User*, e em caso de erro, um objeto *Error*. Após verificar que não existem erros, os dados de sessão são guardados no *singleton* CodeTalk e o utilizador é redireccionado para a *StartActivity*.

StartActivity

Esta *activity* contém a lista de Grupos aos quais o utilizador tem acesso (guardados em *allowedGroups*).

As referências para o Firebase são inicializadas no método *onCreate()*. Também aqui é obtida a lista dos grupos que estão a ser subscritos pelo utilizador, introduzidos nas *SharedPreferences*, e inicializado o serviço de notificações:

```
setContentView(R.layout.activity_start);

app = (CodeTalk) getApplication();
User user = app.getCurrentUser();
mShared = getSharedPreferences("pt.ulht.codetalk", Context.MODE_PRIVATE);
ref = new
Firebase("https://codetalking.firebaseio.com/users/"+app.getCurrentUserUid()+"/allowedGr
oups");

followRef = new
Firebase("https://codetalking.firebaseio.com/users/"+app.getCurrentUserUid()+"/following
");
followRef.addListenerForSingleValueEvent(new ValueEventListener() {
    @Override
    public void onDataChange(DataSnapshot dataSnapshot) {
        SharedPreferences.Editor editor = mShared.edit();
        followedGroups = new HashSet<String>();
        for (DataSnapshot child : dataSnapshot.getChildren()) {
            followedGroups.add(child.getName());
            Log.d(START_TAG, "Following Group: "+child.getName());
        }
        editor.putStringSet("followedGroups", followedGroups);
        editor.commit();

        Log.d(START_TAG, "Starting Service");
        //Start notifications service
        Intent i = new Intent(FirebaseBackgroundService.class.getName());
        startService(i);
    }

    @Override
    public void onCancelled(FirebaseError firebaseError) {

    }
});
```

No método *onStart()*, despoletado depois de terminado o *onCreate()*, é activado o *GroupsAdapter*, que preenche a *ListView* com os grupos a que o utilizador tem acesso:

```
//Setup list adapter
final ListView listView = getListView();
groupsAdapter = new GroupsAdapter(ref, this, R.layout.group_item);
listView.setAdapter(groupsAdapter);

groupsAdapter.registerDataSetObserver(new DataSetObserver() {
    @Override
    public void onChanged() {
        super.onChanged();
        listView.setSelection(groupsAdapter.getCount() - 1);
    }
});
```

Para que seja possível navegar para o ecrã correto quando se pressiona um dos grupos, é utilizado o método *onItemClickListener*, que despoleta a *GroupActivity* através de um *Intent* ao qual se adiciona o nome do grupo em que o utilizador carregou.

Isto faz com que a *GroupActivity* (ecrã de Detalhe do Grupo) consiga obter os valores corretos do Firebase a partir do nome do grupo.

GroupActivity

A *GroupActivity* gere a Vista de Detalhe do Grupo, e inicialmente tem de obter o nome do grupo que lhe foi passado pela *StartActivity* para poder inicializar todas as referências Firebase corretas:

```
Intent i = getIntent();
groupName = i.getStringExtra("groupName");
Log.d("GroupAct", groupName);
setTitle(groupName.split("_")[0]);

currentUserUid = app.getCurrentUserUid();

notesRef = new
Firebase("https://codetalking.firebaseio.com/groups/"+groupName+"/notes");
followingRef = new
Firebase("https://codetalking.firebaseio.com/users/"+currentUserUid+"/following");
followRef = new
Firebase("https://codetalking.firebaseio.com/users/"+currentUserUid+"/following/"+groupN
ame);
```

Também inicializa os botões de Follow e Unfollow (que são mostrados consoante o estado de subscrição do grupo – através de dados recebidos do Firebase) e os respetivos *listeners*:

Follow

```
btnFollow.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        btnFollow.setVisibility(View.GONE);
        btnUnfollow.setVisibility(View.VISIBLE);

        Map<String, Boolean> toSet = new HashMap<String, Boolean>();
        toSet.put("isFollowing", true);
        followRef.setValue(toSet);

        addGroupToFollowed(groupName);
        Intent i = new Intent(FirebaseBackgroundService.class.getName());
        stopService(i);
        startService(i);
    }
});
```

Unfollow

```
btnUnfollow.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        btnUnfollow.setVisibility(View.GONE);
        btnFollow.setVisibility(View.VISIBLE);

        Firebase ref = new
        Firebase("http://codetalking.firebaseio.com/users/"+currentUserUid+"/following/"+groupName);
        ref.removeValue(new Firebase.CompletionListener() {
            @Override
            public void onComplete(FirebaseError firebaseError, Firebase firebase) {
                if (firebaseError != null) {
                    Log.d("Removing", "Removal Complete");

                    removeGroupFromFollowed(groupName);
                    Intent i = new Intent(FirebaseBackgroundService.class.getName());
                    stopService(i);
                    startService(i);
                } else {
                    Log.d("Removing", "Removal Error: "+firebaseError);

                    removeGroupFromFollowed(groupName);
                    Intent i = new Intent(FirebaseBackgroundService.class.getName());
                    stopService(i);
                    startService(i);
                }
            }
        });
    }
});
```

Tanto um *listener* como o outro utilizam as funções abaixo para remover/adicionar o grupo à lista *followed* no Firebase, e reiniciam o serviço *FirebaseBackgroundService* que refresca a informação que obtém para lançar notificações.

addGroupToFollowed

```
mShared = getSharedPreferences("pt.ulht.codetalk", Context.MODE_PRIVATE);
followedGroups = mShared.getStringSet("followedGroups", null);

followedGroups.add(groupName);

SharedPreferences.Editor editor = mShared.edit();

editor.putStringSet("followedGroups", followedGroups);
editor.commit();
```

removeGroupFromFollowed

```
mShared = getSharedPreferences("pt.ulht.codetalk", Context.MODE_PRIVATE);
followedGroups = mShared.getStringSet("followedGroups", null);

followedGroups.remove(groupName);

SharedPreferences.Editor editor = mShared.edit();

editor.putStringSet("followedGroups", followedGroups);
editor.commit();
```

Quando o método *onCreate()* termina, é lançado o método *onStart()*, onde é populada a *ListView* que contem a lista de notas:

```
final ListView listView = getListView();
notesAdapter = new NotesAdapter(notesRef, this, R.layout.note_item);
listView.setEmptyView(findViewById(R.id.emptyNotes));
listView.setAdapter(notesAdapter);

notesAdapter.registerDataSetObserver(new DataSetObserver() {
    @Override
    public void onChanged() {
        super.onChanged();
        listView.setSelection(notesAdapter.getCount() - 1);
    }
});
```

E, à semelhança da *StartActivity*, é utilizado um método para monitorizar os items da lista e gerir o click nos mesmos.

```
super.onListItemClick(lv, v, position, id);

String noteTitle;
String fullNoteTitle;

TextView tvTitle = (TextView) v.findViewById(R.id.lvItemNoteTitle);

noteTitle = tvTitle.getText().toString();

fullNoteTitle = groupName.split("_")[0] + "_" + noteTitle;
Log.d("GROUP.ACT", "FullNoteTitle: "+fullNoteTitle);
Intent i = new Intent(GroupActivity.this, NoteActivity.class);
i.putExtra("fullNoteTitle", fullNoteTitle);
startActivity(i);
```

Ao pressionar um item Nota, o utilizador é redirecionado para o ecrã Detalhe da Nota, gerido pela *activity NoteActivity*. Também é enviado com o *Intent* o título da nota para preenchimento dos dados corretos.

NewNoteActivity

Ao pressionar o botão *Add Note*, no ecrã de Detalhe de Grupo, esta *activity* é lançada, inicializando as referências Firebase necessárias no método *onCreate()* e seguidamente, no método *onStart()*, inicializa o *listener* do botão que permite adicionar a nova Nota.

Pressionando esse botão, é despoletado o *onClickListener* que executa o seguinte código:

```
String title;
String content;

title = etNoteTitle.getText().toString();
content = etNoteContent.getText().toString();

if (!title.equalsIgnoreCase("") && !content.equalsIgnoreCase("")) {
    //Add to Groups/Notes
    Firebase listRef = groupNotesListRef.push();

    SimpleDateFormat sdf = new SimpleDateFormat("EEE, dd LLL yyyy kk:mm:ss zzz"); /*Sun, 13
    Jul 2014 02:13:05 GMT*/
    String currentDate = sdf.format(Calendar.getInstance().getTime());

    Map<String, String> newNote = new HashMap<String, String>();

    newNote.put("code", "");
    newNote.put("content", content);
    newNote.put("createdAt", currentDate);
    newNote.put("createdBy", currentUserEmail);
    newNote.put("title", title);
    newNote.put("inGroup", groupName);

    listRef.setValue(newNote);

    //Add to /Notes
    Firebase newNoteRef = notesRef.child(groupName.split("_")[0] + "_" + title);

    Map<String, String> newNote2 = new HashMap<String, String>();

    newNote2.put("code", "");
    newNote2.put("content", content);
    newNote2.put("createdAt", currentDate);
    newNote2.put("createdBy", currentUserEmail);

    newNoteRef.setValue(newNote2);
}
```

Aqui, é adicionada uma nova Nota às respetivas listas no Firebase através da utilização de *HashMaps* – o método utilizado pela Firebase API do Android SDK.

NoteActivity

Esta *activity* gere a página de Detalhe de Nota, e é lançada quando o utilizador pressiona um dos itens da lista de notas. É-lhe passado o título da Nota, e esse título é guardado assim que o método *onCreate()* inicia, juntamente com a referência Firebase para essa nota e as *TextViews* necessárias para mostrar o conteúdo da mesma:

```
setContentView(R.layout.activity_note);

Intent i = getIntent();
fullNoteTitle = i.getStringExtra("fullNoteTitle");
Log.d("NOTE.ACT", "Arriving FullTitle: "+fullNoteTitle);

setTitle(fullNoteTitle.split("_")[1]);

ref = new Firebase("https://codetalking.firebaseio.com/notes/"+fullNoteTitle);

tvTitle = (TextView) findViewById(R.id.tvNoteTitle);
tvContent = (TextView) findViewById(R.id.tvNoteContent);
tvCode = (TextView) findViewById(R.id.tvNoteCode);
tvCreatedAt = (TextView) findViewById(R.id.tvNoteDate);
tvCreatedBy = (TextView) findViewById(R.id.tvNoteUser);
```

Seguidamente, no método *onStart()*, é utilizada a referência Firebase para obter a informação relativa à Nota escolhida, e as respetivas *TextViews* são populadas com esses valores:

```
ref.addListenerForSingleValueEvent(new ValueEventListener() {
    @Override
    public void onDataChange(DataSnapshot dataSnapshot) {

        String title = dataSnapshot.getName().split("_")[1];
        String content = (String) dataSnapshot.child("content").getValue();
        String code = (String) dataSnapshot.child("code").getValue();
        String createdAt = (String) dataSnapshot.child("createdAt").getValue();
        String createdBy = (String) dataSnapshot.child("createdBy").getValue();

        tvTitle.setText(title);
        tvContent.setText(content);
        tvCode.setText(code);
        tvCreatedAt.setText(createdAt);
        tvCreatedBy.setText(createdBy);
    }

    @Override
    public void onCancelled(FirebaseError firebaseError) {

    }
});
```

FollowingActivity

Esta *Activity* é acedida através do menu de opções, representado como padrão no sistema Android por 3 quadrados alinhados na vertical, no canto superior direito da aplicação. Escolhendo a opção *Following*, esta *activity* é despoletada.

Aqui, utilizando a referência Firebase que aponta para a lista *following* do utilizador com sessão ativa, é mostrada uma lista de grupos utilizando um *ArrayAdapter* que mostra os nomes dos mesmos numa *ListView*.

É também adicionado um método *onItemClickListener* à *ListView* para, ao pressionar o item referente ao Grupo, o utilizador ser redirecionado para o ecrã relativo a esse grupo:

```
ref.addListenerForSingleValueEvent(new ValueEventListener() {
    @Override
    public void onDataChange(DataSnapshot dataSnapshot) {
        for (DataSnapshot child : dataSnapshot.getChildren()) {
            Log.d("Follow", "Group: "+child.getName());
            followingGroups.add(child.getName().split("_")[0]);
        }

        final ListView listView = getListView();
        ArrayAdapter<String> simpleAdapter = new
        ArrayAdapter<String>(getApplicationContext(), R.layout.following_item,
        R.id.lvItemFollowedGroup, followingGroups);
        listView.setAdapter(simpleAdapter);

        listView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView<?> parent, View view, int position, long
id) {
                String title;
                String currentUserId;

                TextView tvTitle = (TextView)
view.findViewById(R.id.lvItemFollowedGroup);
                CodeTalk app = (CodeTalk) getApplication();

                title = tvTitle.getText().toString();
                currentUserId = app.getCurrentUserId();

                Intent i = new Intent(FollowingActivity.this, GroupActivity.class);
                i.putExtra("groupName", title + "_" + currentUserId);
                startActivity(i);
            }
        });
    }
});
```

A.1.2.6. Layouts

Os *layouts* utilizados nesta aplicação para representar os vários ecrãs são:

- *activity_splash.xml*: Ecrã Inicial (Logotipo);
- *activity_main.xml*: Ecrã de Log-in;
- *activity_start.xml*: Ecrã de Listagem de Grupos;
- *group_item.xml*: Item representativo de Grupo no Ecrã de Listagem;
- *activity_group.xml*: Ecrã de Detalhe de Grupo;
- *note_item.xml*: Item representativo de Nota no Ecrã de Detalhe de Grupo;
- *activity_new_note.xml*: Ecrã de Adicionar Nota;
- *activity_note.xml*: Ecrã de Detalhe de Nota;
- *activity_following.xml*: Ecrã de Grupos Subscritos;
- *following_item.xml*: Item representativo de Grupo no Ecrã de Grupos Subscritos.

A2. Anexo de Manual de Utilizador da aplicação

A2.1. Componente Web

A componente Web da plataforma CodeTalk está disponível em:

<https://codetalking.firebaseio.com>

A2.1.1. Página Principal

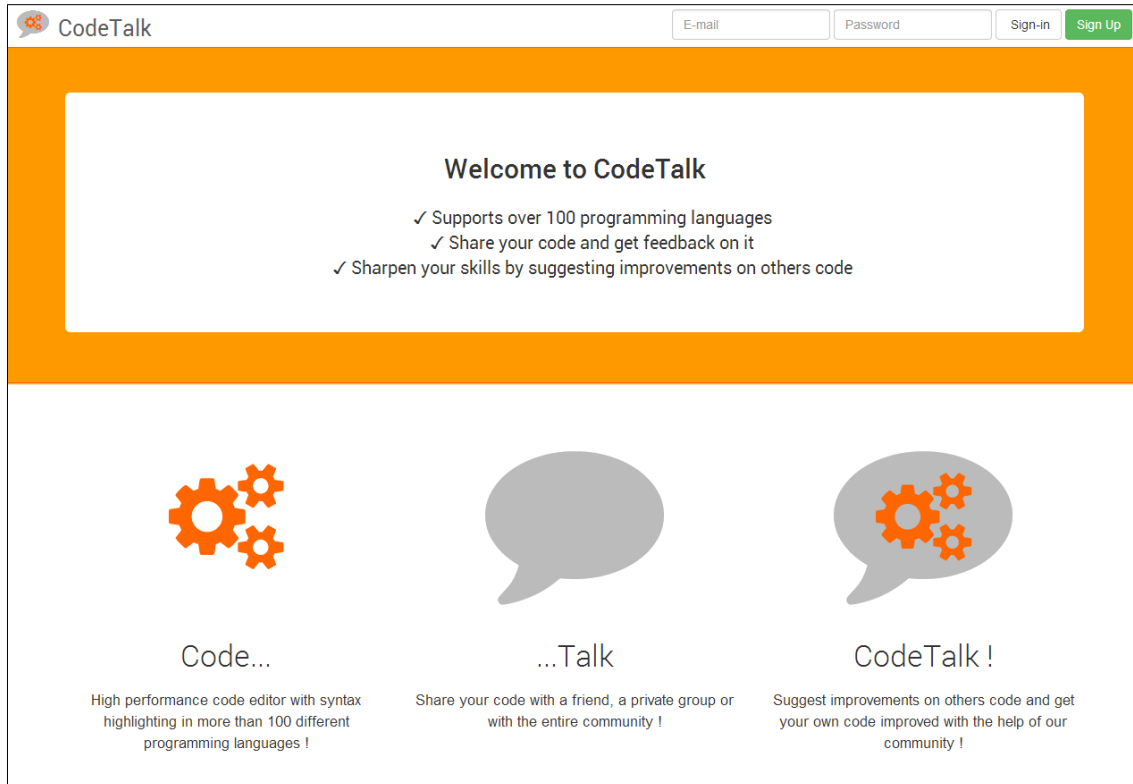


Figura 29 - Página Principal

A partir desta página, o utilizador pode fazer o registo na plataforma (através do botão **Sign-up**) ou entrar diretamente na mesma, se já tiver um registo efetuado (através do botão **Sign-in**).

Ao pressionar o botão **Sign-up**, é mostrado o seguinte diálogo:

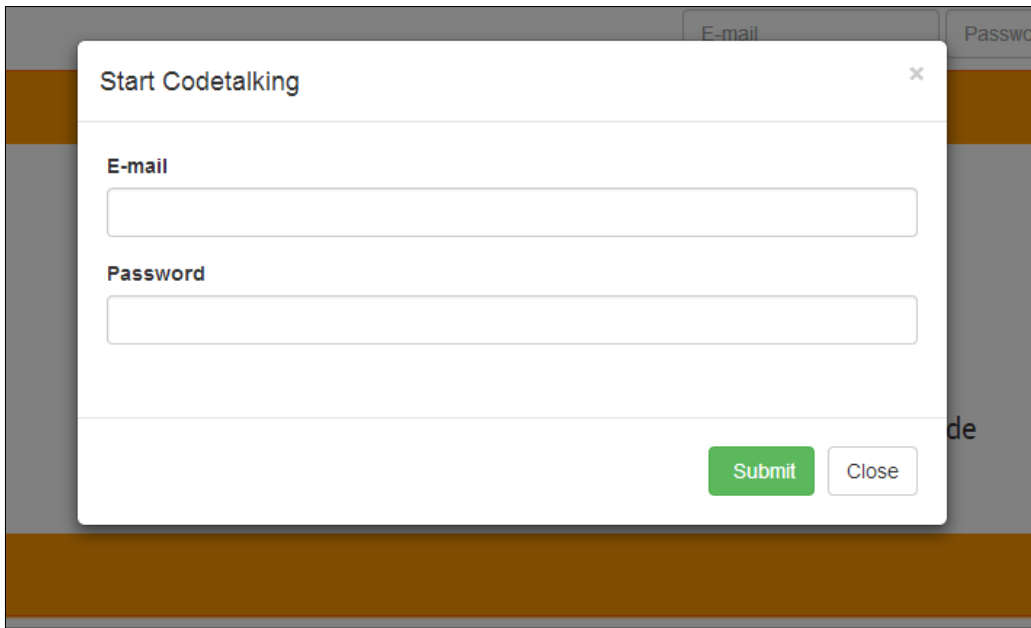


Figura 30 - Diálogo de Registo

O utilizador pode então introduzir o e-mail e password pretendidos para o acesso à plataforma. Não são permitidos e-mails duplicados, e é lançada uma mensagem de erro quando isso acontece.

Após efetuar o registo, o diálogo desaparece e o utilizador pode então entrar na plataforma introduzindo o e-mail e password escolhidos nos campos para o efeito, no canto superior direito, pressionando seguidamente o botão **Sign-In**.

A2.1.2. Página de Lista de Grupos

Após a entrada na plataforma, é apresentada a Página de Grupos:

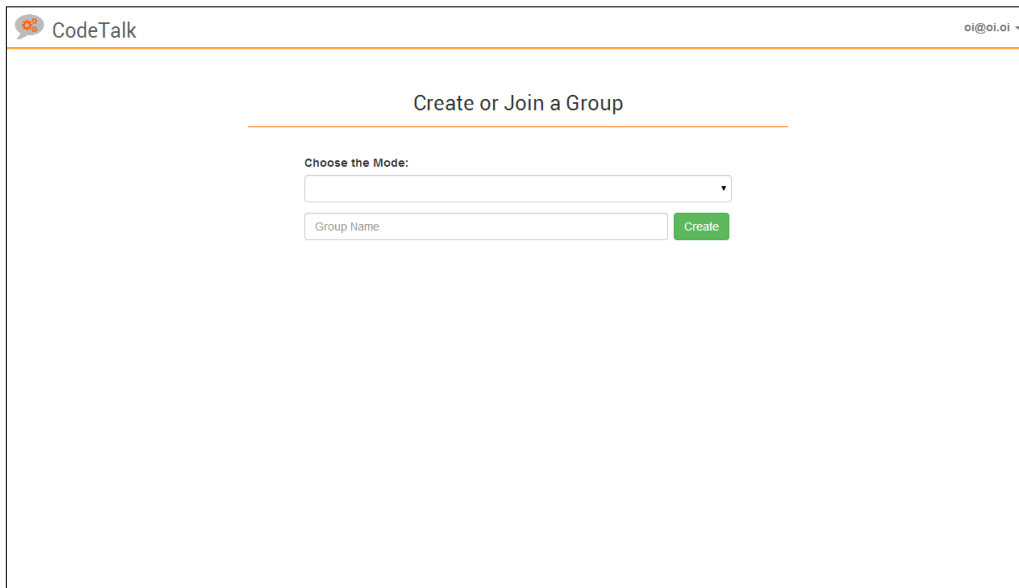


Figura 31 - Página de Grupos

Aqui, o utilizador pode criar o seu primeiro Grupo, escolhendo um modo (linguagem de programação pretendida) e introduzindo um nome para o mesmo na caixa respetiva. Após pressionar o botão **Create**, é apresentado o grupo abaixo:

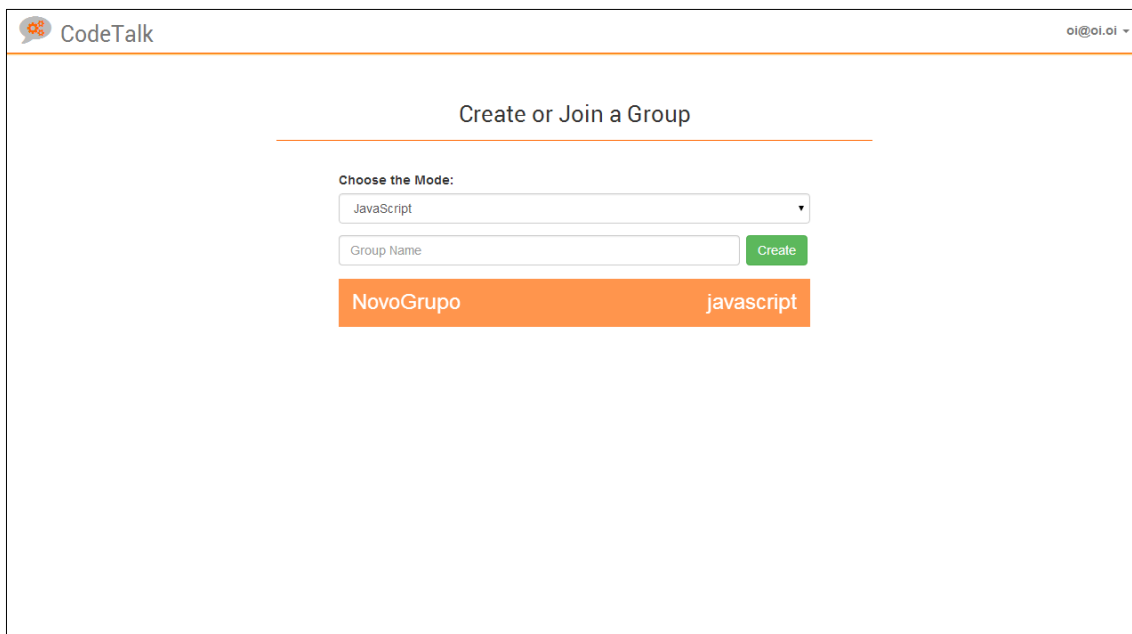


Figura 32 - Criar novo Grupo

Ao carregar na caixa representativa do grupo criado, é apresentada a página do Grupo:

A2.1.3. Página do Grupo

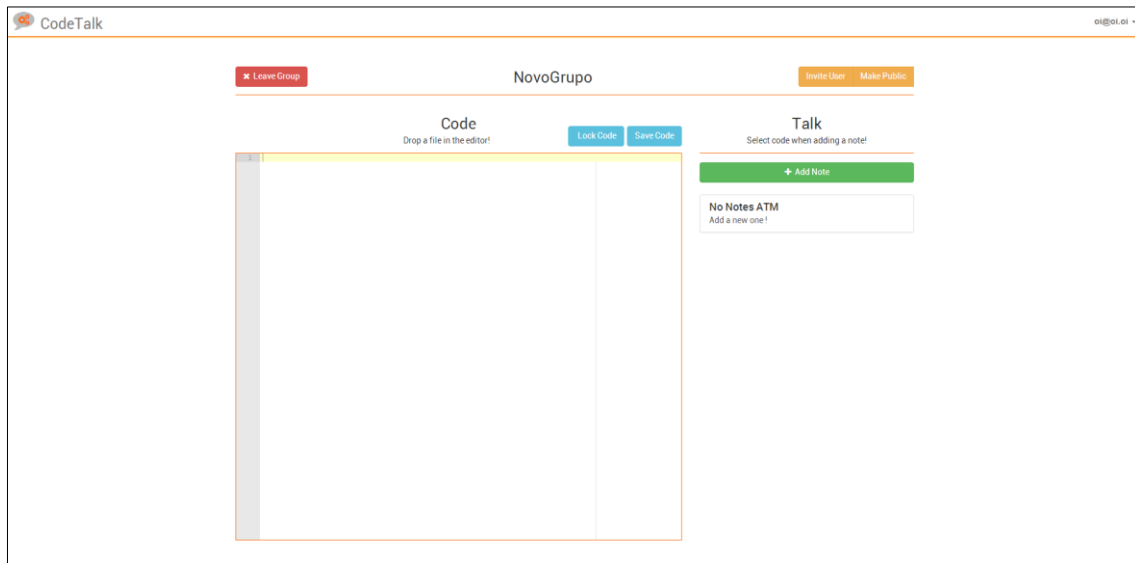


Figura 33 - Página do Grupo

Aqui, o utilizador tem ao seu dispor diversas ações:

- *Leave Group* – permite ao utilizador abandonar o grupo
- *Lock Code / Unlock Code* – permite bloquear o editor (e desbloquear quando bloqueado)
- *Save Code* – permite gravar o código inserido
- *Invite User* – permite convidar um outro utilizador para participar no grupo
- *Make Private* – permite tornar o grupo público inicialmente ou privado quando público
- *Add Note* – permite adicionar uma nova nota ao grupo

Para regressar à página de lista de grupos, o utilizador necessita de pressionar o logotipo CodeTalk, representado na parte superior esquerda da página.

Para introduzir código no editor, o utilizador pode proceder de duas formas – copiar o código e colar no editor ou arrastar um ficheiro de código diretamente para o editor.

Uma vez que o código é inserido no editor, é possível selecionar um excerto do código inserido, e ao pressionar o botão **Add Note**, o excerto selecionado é transferido para o diálogo de criação de nota:

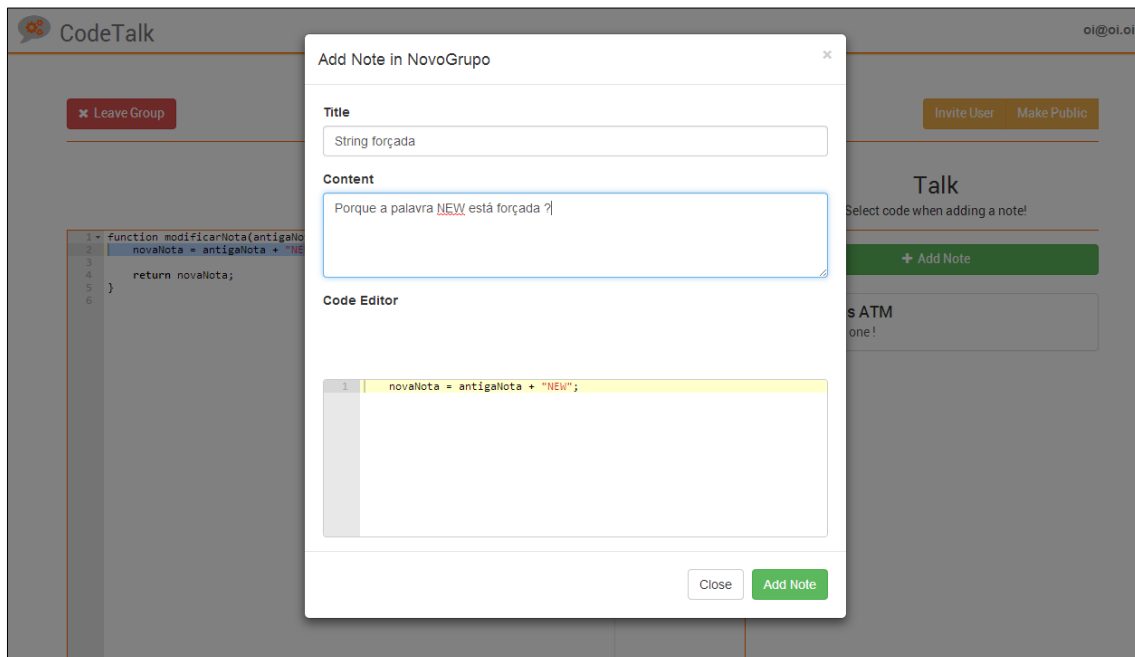


Figura 34 - Adicionar nova Nota

Adicionando a nota acima, a mesma aparece então na Página de Grupo, do lado direito, da seguinte forma:

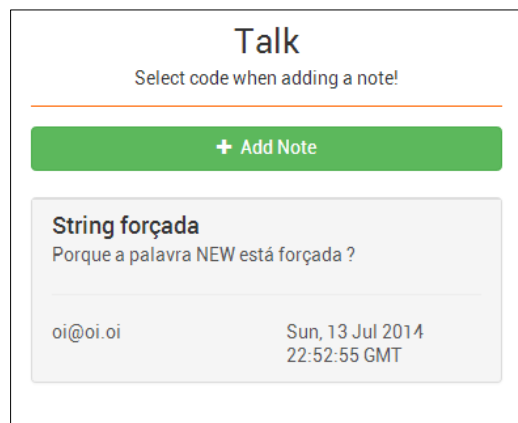


Figura 35 - Nota Adicionada

Além do título e conteúdo da nota, também é representado o e-mail do criador e a data de criação. Ao pressionar a caixa representativa da nota, é possível ver o código contido dentro da mesma, através de um diálogo semelhante ao apresentado na criação da nota, onde não é possível editar os campos, apenas visualizá-los.

A.2.2. Componente Mobile

A aplicação Mobile, desenvolvida em Android, é apresentada como um complemento à aplicação Web, sendo que grande parte das funcionalidades presentes na aplicação Web não estão disponíveis na aplicação Android.

A principal funcionalidade desta aplicação é a possibilidade de subscrever grupos. Isto faz com que o utilizador consiga receber notificações no seu dispositivo móvel a cada vez que alguém adiciona uma nota ao grupo subscrito.

A.2.2.1. Splash Screen



Figura 36 - Splash Screen

Quando o utilizador lança a aplicação, é mostrado um *Splash Screen* enquanto os dados iniciais estão a ser carregados.

A.2.2.2. Página de Lista de Grupos

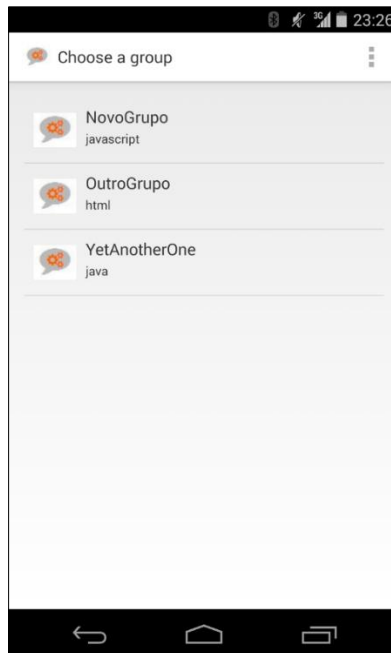


Figura 37 - Lista de Grupos Android

Aqui, é possível entrar no detalhe de um grupo, pressionando a linha ao grupo correspondente, sendo então mostrada a página do grupo

A2.2.3. Página do Grupo

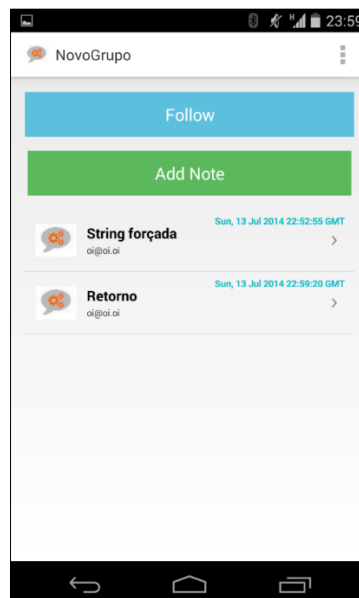


Figura 38 - Página de Grupo Android

A partir da página de detalhe do grupo, o utilizador pode visualizar as notas inseridas no grupo, através de uma lista, adicionar uma nova nota (através do botão **Add Note**) ou subscrever notificações de novas notas adicionadas ao grupo (através do botão **Follow**).

Pressionando uma nota da lista, é apresentado o ecrã de detalhe dessa nota:

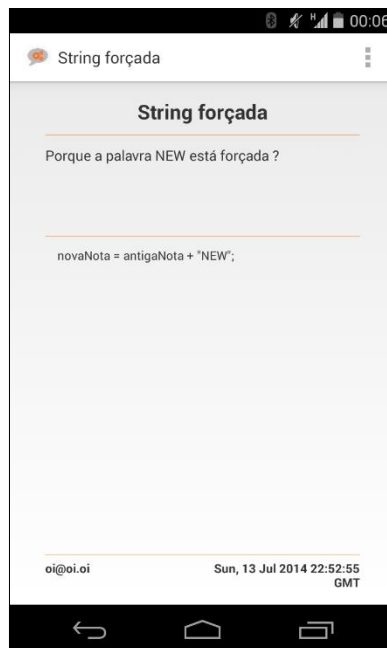


Figura 39 - Detalhe de Nota Android

É também possível adicionar uma nova nota, pressionando o botão **Add Note** que redireciona o utilizador para o ecrã onde é possível adicionar a nota:

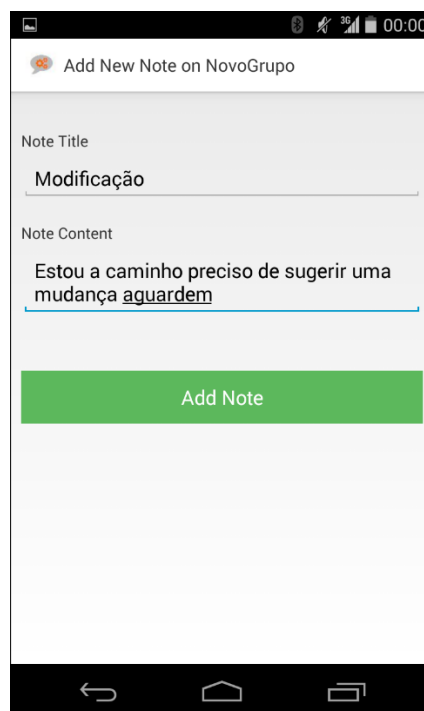


Figura 40 - Adicionar nova Nota Android

Através do botão **Follow**, o utilizador pode subscrever esse grupo. Ao subscrever um grupo, novas notas adicionadas ao mesmo lançarão notificações do sistema, alertando o utilizador de que uma nova nota foi adicionada ao grupo:

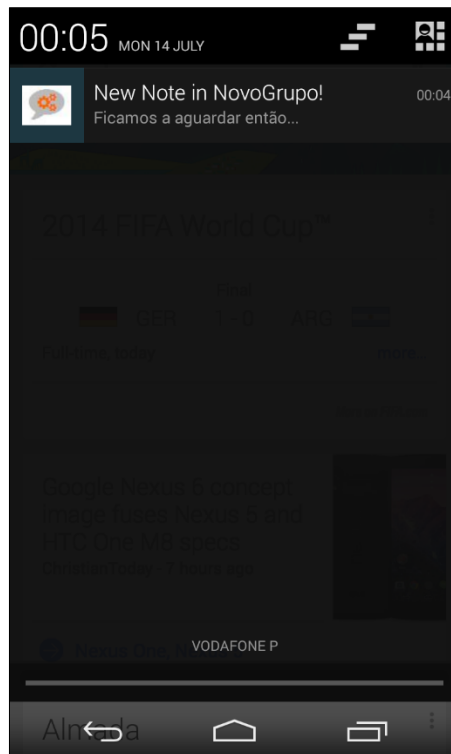


Figura 41 - Notificações Android

Ao pressionar a notificação, o utilizador é redirecionado para o Grupo que contem a respetiva nota.

É também possível revogar a subscrição de um determinado grupo, pressionando o botão **Stop Following**, que apenas é mostrado caso o utilizador seja um subscritor do grupo:

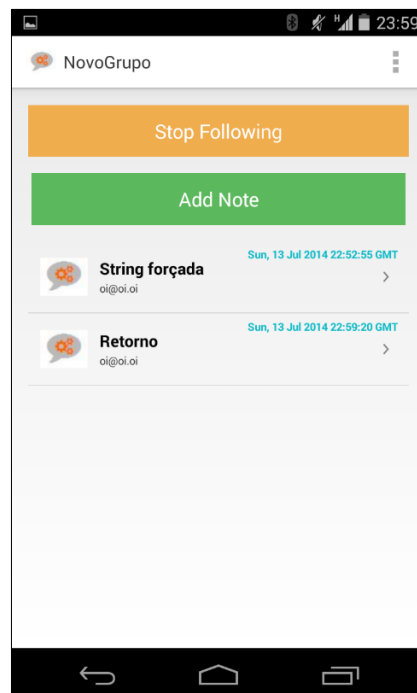


Figura 42 - Subscrição do Grupo Android

Para visualizar todos os grupos que o utilizador está a seguir, existe uma opção dentro da dropdown de definições chamada **Following**.

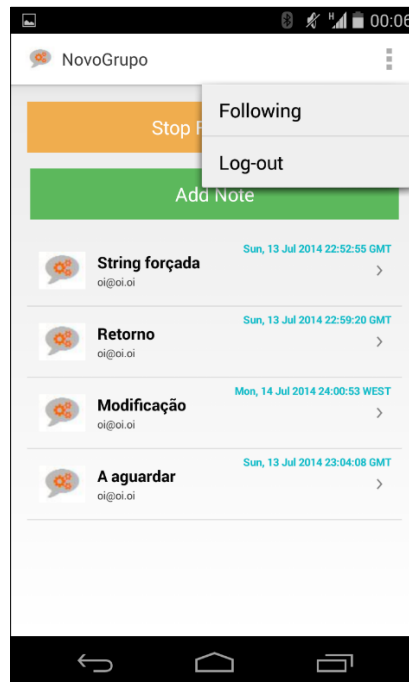


Figura 43 - Menu de Opções Android

Ao pressionar esta opção, o utilizador é redireccionado para a seguinte Página:

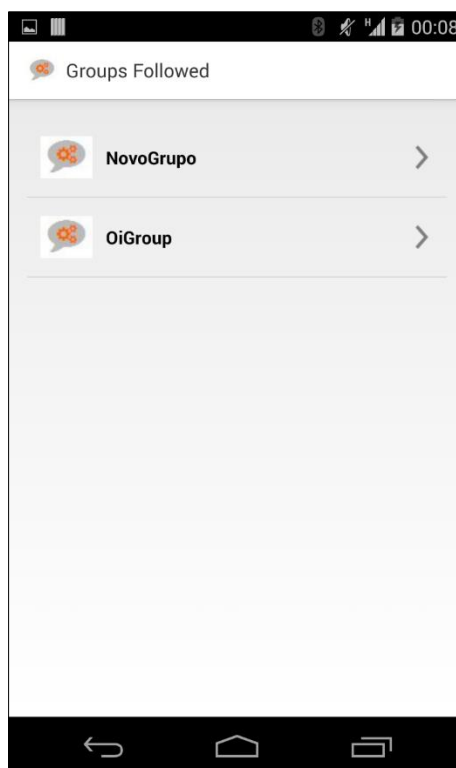


Figura 44 - Grupos subscritos Android

A partir daqui, o utilizador consegue visualizar os grupos que subscreveu, e ao pressionar um deles, é direccionado para a página do mesmo.

A3. Glossário

Acrónimo	Definição
ACL	Access Control List
API	Application Programming Interface
CAP	Consistency, Availability and Partition tolerance
CRUD	Create, Read, Update and Delete
DOM	Document Object Model
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IDE	Integrated Development Environment
JSON	JavaScript Object Notation
MVC	Model-View-Controller
MVW	Model-View-Whatever
NoSQL	Not Only SQL
POJO	Plain Old Java Object
SDK	Software Development Kit
SQL	Structured Query Language
UID	Unique Identifier
URL	Uniform Resource Locator