Arquitetura CodeTalk

# Introdução

A plataforma CodeTalk foi pensada para ser composta de uma estrutura de dados robusta, que é engloba Comunidades, Grupos Privados, Utilizadores e Notas. Para suportar esta estrutura, é necessário um *back-end* que consiga suportar todos os dados provenientes de cada um dos componentes, assim como as relações entre si.

Esta plataforma é composta por duas componentes – Web e Mobile. A componente Web foi desenvolvida utilizando a [Framework AngularJS](https://angularjs.org/) para desenvolvimento de back-end em *client-side* e a [Framework Bootstrap](http://getbootstrap.com/) para front-end. Ambas as frameworks complementam-se entre si, sendo que o AngularJS é facilmente adaptável a qualquer framework de front-end, ou até mesmo HTML5 puro.

# 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)).

Esta framework gaba-se de uma característica que a distingue de todas as outras, *two-way data-binding*. Esta caracerística 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.

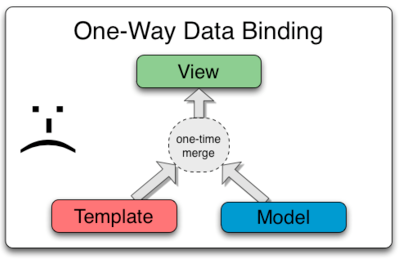


Figura - One-way Data Binding

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

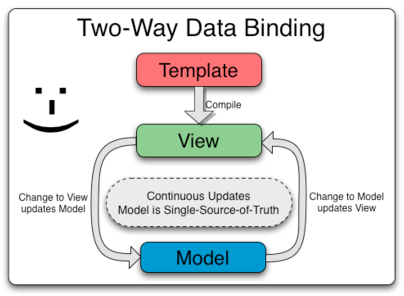


Figura - Two-Way Data Binding

# 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.

A componente mobile da plataforma Codetalk apenas suportará versões iguais ou superiores ao Android 4.0, o que equivale a cerca de 85% do mercado Android:

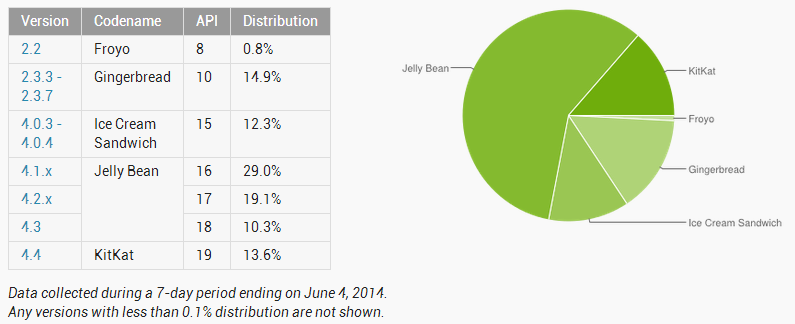


Figura - Distribuição das Versões Android no Mercado

# Parse

Sendo que o desenvolvimento desta plataforma culmina na apresentação da mesma online, foi realizada uma pesquisa pelos *providers* mais utilizados a nível de armazenamento de estruturas de dados na *Cloud*. Uma das opções que se destacou numa primeira fase foi a plataforma [*Parse*](https://parse.com) que engloba um conjunto de serviços de *back-end* na *Cloud* e diversas API’s para acesso aos mesmos.

Esta framework disponibiliza uma API Javascript, muito direccionada para a utilização em sintonia com o [*Backbone.js*](http://backbonejs.org/). No entanto, como a componente mobile da plataforma Codetalk é desenvolvida com base em *AngularJS*, a utilização desta API não parece a mais adequada à situação. Assim sendo, e depois de alguma pesquisa sobre a REST API do Parse e a sua integração com AngularJS, foi essa a escolha tomada.

No que diz respeito à componente mobile da plataforma Codetalk, é utilizado o Android SDK para interagir com a framework Parse. Este SDK disponibiliza métodos intuitivos e alinhados com toda a estrutura de ciclo de vida de uma aplicação Android, sendo que sem dúvida será a escolha mais acertada.

# Estrutura de Dados

Tendo em conta que a plataforma escolhida para armazenamento de dados (Parse) suporta um modelo de dados relacional, a estrutura escolhida para representar a informação recolhida na plataforma Codetalk foi a seguinte:

DIAGRAMA RELACIONAL

A base da plataforma Codetalk são os Grupos, sejam eles públicos (Comunidades) ou privados. Em cada grupo, podem existir vários utilizadores. Esses utilizadores têm a possibilidade de inserir notas (ou comentários) sobre determinadas seleções de código. Quando o utilizador introduz uma nota sobre determinado pedaço de código, esse pedaço de código fica associado à nota, que por sua vez fica associada a esse utilizador.

# Desenvolvimento da Componente Web

Apenas com alguns conhecimentos de JavaScript, HTML5 e CSS, começa o processo de aprendizagem necessário para compreender os métodos, padrões de desenho e código utilizados nas várias tecnologias que proponho para o desenvolvimento desta componente.

Começo por realizar alguns tutoriais que encontro sobre AngularJS, o que me leva 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 TODOs, tanto que são disponibilizadas samples da mesma aplicação de TODOs desenvolvida tendo por base diversas frameworks semelhantes - [TodoMVC](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 efectuadas 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 actualizar a lista com a nova nota, e de seguida outro para mostrar a lista completa ao utilizador.

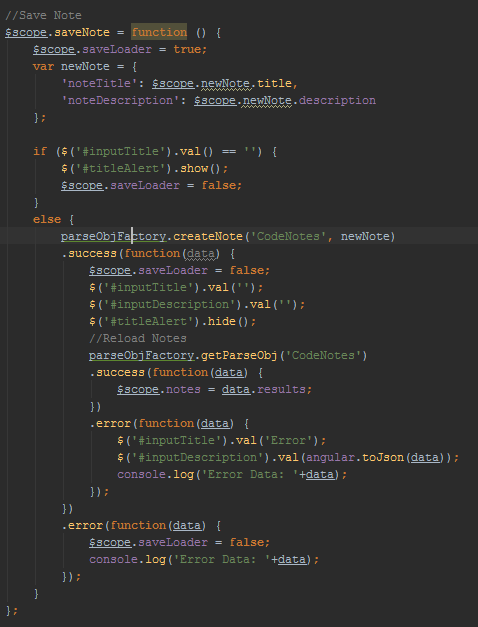


Figura - Exemplo de Código utilizado para guardar uma nota

No exemplo acima, é demonstrado um pedaço de código utilizado para guardar uma nota no Parse, a partir da aplicação web. Numa fase inicial, este código não previa a associação de pedaços de código ou outra informação á nota, apenas o seu título e descrição, representados como *newNote*.

Para acesso ao Parse, utilizo uma *factory* de AngularJS, construída por mim, que contém métodos para chamadas HTTP assíncronas. Neste caso, utilizo a acção *createNote* para guardar a nota no objecto Parse *CodeNotes*. De seguida, tenho de actualizar a vista com a lista de notas (representada por *$scope.notes*) actualizada, 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 excepções 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 objecto 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 objecto do tipo *Pointer* que é guardado no objeto Parse destino e que aponta para o ficheiro adicionado.

O código que torna tudo isto possível é o seguinte:

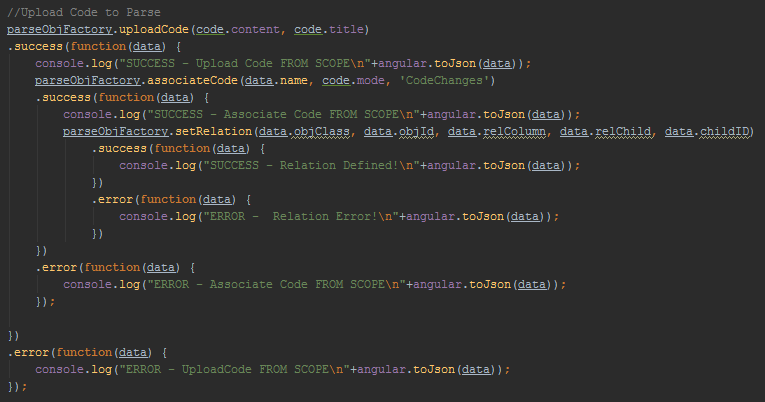


Figura - Exemplo de Código utilizado para associar um ficheiro a uma nota

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 efectuar 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 multi-utilizador, 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 actualizada 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.

# Firebase

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 [vídeo](https://www.youtube.com/watch?v=e4yUTkva_FM) gravado numa das conferências de AngularJS ([ng-conf](http://www.ng-conf.org)) sobre a plataforma Firebase e a sua integração com o AngularJS.

O mote principal deste vídeo é *Building Realtime Apps with Firebase and AngularJS* e começa por mostrar a evolução por detrás das web apps, desde o DHTML utilizando VBScripts, passando por jQuery e DOJO – frameworks de manipulação de DOM (Document Object Model) com a ajuda de browsers modernos, e terminando em AngularJS, uma framework que permite construir aplicações web.

No entanto, a melhor parte do vídeo é onde a plataforma Firebase é sucintamente descrita:

* API que permite guardar e sincronizar informação em tempo real
* Serviços de Login para autenticação de utilizadores
* Regras de segurança declarativas para proteger os dados inseridos

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

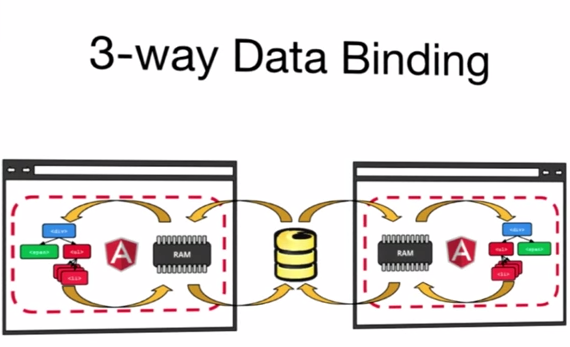


Figura - Three-way Data Binding

Esta funcionalidade faz com que seja possível a inserção e actualização 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 objecto JSON e é possível adicionar uma estrutura JSON à mesma, sendo essa estrutura representada no servidor, através do *Data Browser* no website Firebase:

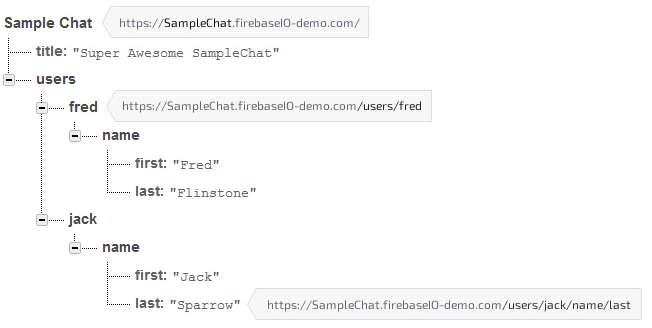


Figura - Exemplo de Estrutura de Dados no Firebase

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 TODOs 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 TODOs:

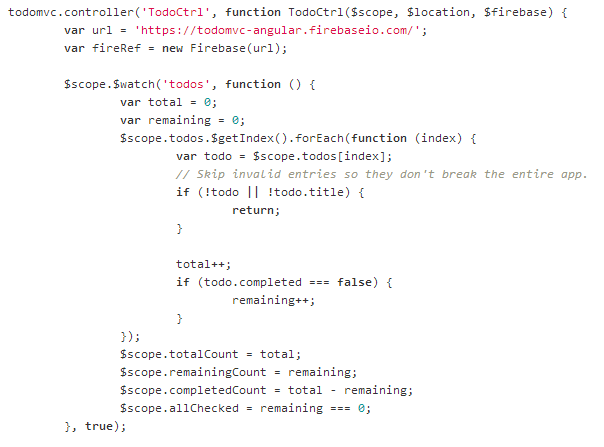


Figura - AngularJS + Firebase: Binding da Lista

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

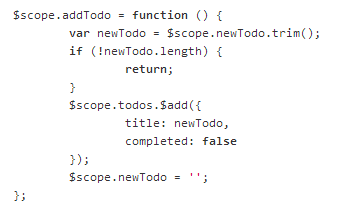


Figura - AngularJS + Firebase: Adicionar Nota

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 actualizado na própria lista, o que se reflete imediatamente na vista.

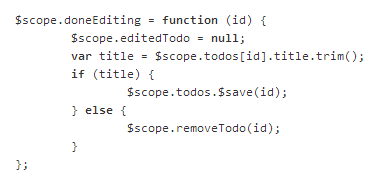


Figura - Firebase + AngularJS: Edição de Nota

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.



Figura - Firebase + AngularJS: Remoção de Nota

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*.