ANA CLARA SOUZA SILVA, CAMILA SANTOS CORDEIRO, GABRIEL LUIZ RAMOS, GABRIEL SANTOS IZAGUIRRE, NATHÁLIA FERREIRA PIRES, NÍCOLAS MOURA DA SILVA

DATA CLOUD

SAAS com performance de dados em nuvem

Cubatão

2015

ANA CLARA SOUZA SILVA, CAMILA SANTOS CORDEIRO, GABRIEL LUIZ RAMOS, GABRIEL SANTOS IZAGUIRRE, NATHÁLIA FERREIRA PIRES, NÍCOLAS MOURA DA SILVA

DATA CLOUD

SAAS com performance de dados em nuvem

Projeto de Iniciação Científica, apresentado ao Instituto Federal de Educação, Ciência e Tecnologia de São Paulo e ao VII COBRIC (Congresso Brasileiro de Iniciação Científica), como parte das exigências para a obtenção do título de Técnico de Informática.

Orientador: Prof. Maurício Neves Asenjo

Cubatão

2015

“O futuro é na nuvem”

(Eduardo Henrique Gomes)

RESUMO

O projeto visa exemplificar uma nova plataforma de Banco de Dados na nuvem orientado a objetos, o Parse, por meio da criação de um aplicativo que possa ser inteiramente executado no navegador do usuário, utilizando linguagens já bem conhecidas no ambiente "web" como: "HTML", "CSS" e “JavaScript”, sendo a segunda desenvolvida a partir da ferramenta “Sass” (facilitando a escrita e o desenvolvimento responsivo) e a última a principal linguagem, uma vez que o sistema faz uso do modelo MVC ("Model-View-Controller") para executar sua aplicação, através do "framework" AngularJS desenvolvido pela "Google". Para melhor análise dos resultados, o "webapp" criado recebeu a alcunha de "ContactApp", levando os usuários a cadastrarem dados referentes a uma lista de contatos.

Palavras-chave: Parse, Banco de dado, nuvem, orientação a objetos.

ABSTRACT

The project aims to exemplify a new object-oriented cloud database platform, the Parse , by creating an application that can be fully implemented in the user's browser, using well-known programming language in the web environment such as HTML, CSS and JavaScript , the second one being developed with the tool "Sass" (which facilitates the writing and the responsive development) and the last one being the main language, once the system makes use of the Model-View-Controller to run its application, using the framework AngularJs developed by Google. To reach a better analysis of the results, the webapp created received the name of ContactApp, leading users to register data from a contact list.

Keywords: Parse, Data-base, Cloud, object-orientation.

SUMÁRIO

INTRODUÇÃO..... 7
1 PARSE E ANGULARJS 9
1.1 Parse 9
1.1.1 HTML 10
1.1.2 Funções JavaScript 11
1.1.3 No painel do Parse 14
1.2 ANGULARJS 15
1.2.1 Iniciando uma aplicação com as referidas diretivas do *framework* 16
2 CONTACTAPP: A APLICAÇÃO 20
2.1 Tratamento interno da aplicação..... 21
CONCLUSÃO 23
REFERÊNCIAS 24

INTRODUÇÃO

Dados, armazenamento e computação em nuvem são temas discutidos há muito tempo.

Segundo a IBM (International Business Machines): “Computação em nuvem, às vezes referida simplesmente como ‘a nuvem’, é a entrega de recursos computacionais por demanda - tudo em aplicações para bases de dados.” (http://www.ibm.com/cloud-computing/us/en/what-is-cloud-computing.html).

Algumas empresas, como a referida IBM, além da Google e Microsoft foram as primeiras a tomar isso como nova tendência e a investir nesses processos que visam ao gerenciamento de documentos, arquivos e informações por meio da *internet*.

Atualmente, esse gerenciamento é algo comum. Qualquer usuário é capaz de criar uma conta e utilizar serviços que façam *uploads*, *downloads* e alterações de arquivos na *internet.* As possibilidades são enormes.

Assim como os armazenamentos, os conceitos de programação também foram evoluindo, fazendo com que o relacionamento e a interação entre usuário e o sistema ficassem cada vez mais trivial.

Uma das plataformas recentemente desenvolvidas chama-se Parse, que procura integrar sistemas de diversos tipos (*mobile, desktop* e via *browser*), valendo-se da tecnologia de armazenamento de dados em nuvem. A central de desenvolvimento dessa plataforma encontra-se na Califórnia, no Quartel General do Facebook, na cidade de Menlo Park.

Tendo isso em mente, tratar-se-ão das linguagens essenciais para um relacionamento entre homem e *website*, em seu atual estado evolutivo, utilizando de marcações e APIs (provém do Inglês *Application Programming Interface*) feitas com HTML5 (abreviação para a expressão inglesa “*HyperText Markup Language”*, que significa “Linguagem de Marcação de Hipertexto”), estilos aplicados via CSS (uma das linguagens núcleo da *open web*), também utilizando o pré-processador SASS (*Syntactically Awesome Style Sheets*), permitindo maior compressão dos arquivos em folha, intercâmbio de informações com JavaScript, sendo esse o principal componente do projeto, através do qual o foco será a plataforma de armazenamento

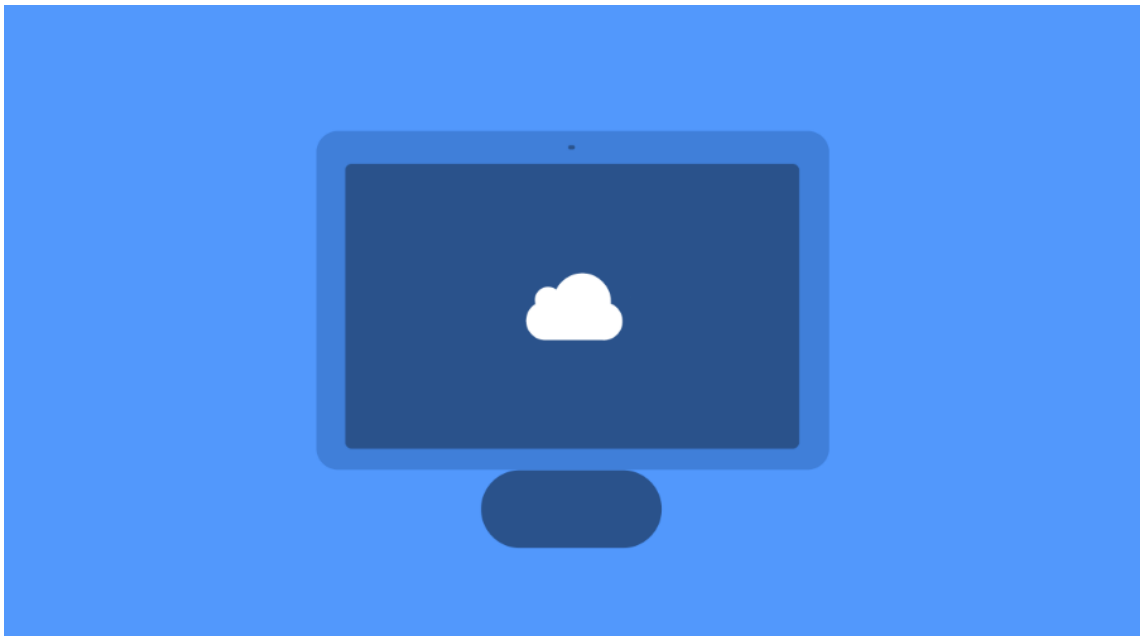
de dados em nuvem aqui citada (Parse) que será gerenciada com o *framework* desenvolvido pela Google, o AngularJS.

Com esses conceitos, criar-se-á uma aplicação *web* (*webapp*) que fará conexões, armazenamentos, leituras, atualizações e exclusões de objetos (dados) na nuvem.

1 PARSE E ANGULARJS

O capítulo sobre o Parse abordará o primeiro contato com a plataforma, algumas especificações e a criação de uma breve interação utilizando Javascript; sobre o AngularJS, abordará os conceitos do *framework*, apresentará as diretivas utilizadas na aplicação e sua interação com as linguagens interpretadas pelo navegador.

1.1 Parse



O Parse é uma plataforma recentemente desenvolvida para gerenciamento, armazenamento e documentação de dados em nuvem, utilizando o conceito de Programação Orientada a Objetos. Possui foco e usabilidade em diversas plataformas e linguagens, podendo ser utilizada por linguagens *server-side*, *client--side* e por aquelas que se valem de aplicativos *mobile* para suas execuções. Tratar-se-á da a conexão e dos componentes via JavaScript.

Antes de se iniciar o contato com a plataforma, é necessário possuir alguns conceitos:

- Classe: para essa plataforma, as Classes são as bases de dados, ou seja, as tabelas com as informações de cada usuário;

- Objetos: o Parse trata os dados como objetos, ou seja, cada registro criado na Classe será tratado como um objeto independente, possuindo seus campos e sua devida ID (gerar-se-á, aleatoriamente, uma chave com 10 caracteres);
- Campos: campos, em suas Classes, são as colunas dos registros de cada objeto.

Ir-se-á à utilização da plataforma. Antes de tudo, é necessário criar uma conta no Parse (https://parse.com/). Para esclarecer qualquer dúvida, apresentar-se-á o passo a passo dessa criação.

Ao clicar para criar a conta, o usuário preencherá os dados necessários e será informado sobre o nome do primeiro aplicativo. O usuário criará um nome, podendo criar outros posteriormente. Após criar a conta, aparecerá uma janela para o usuário dar início às suas aplicações com o *Parse*. Como se trata de um exemplo via JavaScript, o usuário deve clicar em "*Data*", depois em "*Web*" e, após isso, em "*New Project*".

O usuário será levado a uma página contendo uma explicação sobre o SDK do Parse, e também a ID e *Key* de usuário, para que este possa fazer as conexões com os aplicativos do Parse. Um exemplo será disponibilizado para o usuário baixar em um botão, como "*Download the blank Javascript/HTML5 Project*", onde ele poderá baixar e dar uma olhada em um exemplo "em branco" de uma página, acessando seu banco de dados. Baixar-se-á essa página, pois ela será necessária.
A ID e *Key* aparecerão, desta forma, na página em um trecho JavaScript:

```
1. <script>
2.   Parse.initialize("valor-da-sua-id", "valor-da-sua-key");
3. </script>
```

1.1.1 HTML

Construiu-se uma página HTML básica com um parágrafo para um campo de texto, dois botões que chamam duas funções (uma para gravar objetos na base de dados e outra para retornar) e um parágrafo vazio, com uma ID para receber os objetos que serão gravados somente para fazer o envio e o recebimento dos objetos à Classe do Parse, desta forma:

```
1. <p>
2.   Digite seu nome: <input type="text" id="enviaNome">
3. <br>
4. <button onclick="armazena()">Armazenar nome no Parse</button>
5. <button onclick="retorna()">Retornar ID dos objetos do Parse</button>
6. </p>
7. <p id="tempNames">
8. </p>
```

1.1.2 Funções Javascript

Feito isso, começar-se-á a utilização das informações que o Parse fornecerá. Separar-se-ão os scripts desta forma, para facilitar a visualização (nada impede que o usuário os coloque da forma que quiser, onde quiser, contanto que atinja o objetivo e utilize o padrão ao qual ele está acostumado). A linha com o "*Parse.initialize*" foi colocada da seguinte forma, dentro da <*head*> do documento HTML:

```
1. <script>
2.   Parse.initialize("minha-id", "minha-key");
3. </script>
```

Como é um trecho de código que "inicializa" a aplicação com o Parse, decidiu-se por colocá-lo dentro da *tag* <*head*>. O *script* do Parse é disponibilizado naquela página "em branco". O usuário pode baixá-lo ou utilizar pelo trecho que é disponibilizado:

```
<script src="http://www.parsecdn.com/js/parse-1.4.2.min.js"></script>
```

A aplicação com o ID e com o *script* já está funcionando corretamente. Agora ir-se-á às funções.

Na página “em branco”, também existirá um exemplo de gravação de objetos, localizado dentro do Parse. Separou-se um *script* com duas funções para realizar isso:

```
<script>
 function armazena(){
   var nome = document.getElementById("enviaNome").value;
   var conectaParse = Parse.Object.extend("ExemploClasse")
   obj = new conectaParse();
   obj.set("nome", nome);
   obj.save(null, {
     success: function(valor) {
       alert("Objeto criado com ID: " + obj.id);
     },
     error: function(valor, error) {
       alert("Falha ao criar objeto" + error.message);
     }
   });
 }

 function retorna(){
   document.getElementById("tempNames").innerHTML ="";
   var query = new Parse.Query("ExemploClasse");
   query.greaterThan(
     "createdAt","2014-06-23T18:20:17.149Z"
   );
   query.find({
     success: function(results) {
       for (var i = 0; i < results.length; i++) {
document.getElementById("tempNames").innerHTML += (i+1)+"º Objeto da Classe: "+ results[i].id +"";
       }
     },
     error: function(error) {
       alert('não foi');
     }
   });
 }
</script>
```

Nota-se que as duas funções são ativadas pelos botões que foram criados: a função armazena e a função retorna, que fazem o seguinte:

Na função armazena, uma variável “nome” é criada, a qual pega o valor do campo “enviaNome” e, assim, uma classe “conectaParse” é criada para realizar a conexão com a classe de nome “NomeParse” (vale lembrar também que o *Parse* não aceita Classes com vírgulas, pontos, acentos ou caracteres especiais). Um objeto “obj” foi criado, seguindo as especificações da classe “conectaParse”, para realizar o armazenamento dos objetos num campo “nome”. Se o dado for armazenado como sucesso, um *alert* aparecerá na tela com uma mensagem mostrando a ID do objeto dentro do banco de dados; caso contrário, uma mensagem de erro aparecerá. Vale lembrar que o tratamento das gravações dentro do Parse é um tanto quanto diferente. Se no seu código o usuário indicar uma Classe (lembrando que: Classe = Tabela) ou um campo não existente, a aplicação criará automaticamente essa Classe ou esse campo ao ser executada.

Na função “retorna”, é tratado primeiro aquele parágrafo que foi criado, para que, toda vez que o botão for clicado, todo o conteúdo do parágrafo seja apagado antes de se pegarem os objetos do Parse. Uma variável com nome de *query* é criada e igualada à *query* (consulta) do Parse, a qual recebe o parâmetro “ExemploClasse*”*, que é o nome da Classe na qual se irá fazer a consulta. Feito isso, o trecho seguinte utiliza o método *greaterThan* para passar dois parâmetros dentro da pesquisa, (ou *query*), que será executada. Um desses parâmetros é o *createdAt* e o outro é o de data, ou seja, dentro dessa pesquisa todos os objetos que estiverem na Classe do Parse que tenham sido criados na data que foi especificada, ou depois, devem ser retornados (notem que se passou o ano de 2014, ou seja, como essa consulta está sendo criada agora, todos os objetos serão retornados).

É possível criar diversas pesquisas, com diversos métodos e diversos parâmetros. Esses mesmos utilizados foram fornecidos pelos próprios campos gerados na Classe do Parse, com a data de um dos objetos. Somente trocou-se o

ano, para que a função retornasse todos os objetos. Feito isso, o método *find* é chamado, podendo resultar em sucesso ou erro. Caso retorne erro, um *alert* aparecerá com uma mensagem de “não foi”. Caso a pesquisa resulte em sucesso, um laço de repetição será executado, inserindo no conteúdo HTML do parágrafo “tempNames*”* as IDs de cada objeto que foi retornado como resultado da *query*.

O tratamento das pesquisas dentro do Parse é muito interessante. Tentou-se criar *arrays* fora do método *find*, somente para armazenar os ID*s* dos objetos, mas isso não foi concluído com sucesso. Todo o tratamento com os objetos que serão retornados como *results* da pesquisa devem ser trabalhados dentro da própria pesquisa.

Com isso, ter-se-á uma aplicação simples, página HTML que, por JavaScript, salvará objetos em um banco de dados na nuvem e retornará todos os seus ID*s*.

O usuário pode encontrar no *site* do Parse toda a documentação com os códigos que eles possuem para todas as plataformas que eles atendem. Dentro da página “em branco”, que é disponibilizada para *download* quando se cria um novo projeto, o usuário terá um *link* para as informações que estarão de acordo com a linguagem que este estiver utilizando.

1.1.3 No painel do Parse

Feito isso, o usuário pode gerenciar os aplicativos, as classes e os objetos dentro do seu painel no *site* do Parse. Acessando a aba “Core”, dentro do campo “Data”, estarão as classes, e dentro das classes estarão os objetos com as informações.

É possível acessar um exemplo com esse conteúdo através do link: <http://gabrieluizramos.github.io/contactapp/parse>

1.2 ANGULARJS

AngularJS é um *framework* JavaScript desenvolvido pela Google para a criação de aplicações *web*. Funciona através de diretivas, ou seja, atributos (extensões das *tags*) HTML que, ao serem utilizados, fazem com que a página trabalhe como uma aplicação, seguindo o padrão MVC (*Model-View-Controller*, onde o *Model* está nas *tags* da página, o *Controller* em um arquivo com extensão .js separado e a *View* é o resultado gerado para o usuário).

As diretivas principais são as seguintes:

- *ng-app*: especifica um nome para a aplicação em utilização;
- *ng- controller*: especifica um *controller* para a aplicação em utilização;
- *ng-model*: aplica um modelo a alguma *tag*, para tratamento de dados em tempo de execução;
- *ng-bind*: aplica uma entrada de dados via *controller*;
- *ng-init*: especifica um valor inicial para algum elemento;
- *ng-click*: ativa funções ao clicar no elemento;
- *ng-show*: exibe um conteúdo a partir de algum item clicado (semelhante às abas dos navegadores, onde todo o conteúdo é carregado, podendo gerar uma aplicação com diversas abas, mas contendo apenas uma página);
- *ng-repeat*: cria elementos com a quantidade equivalente de algum *array* de dados do *controller*;
- *ng-class*: aplica alguma classe ao elemento.

Toda a documentação e a biblioteca do AngularJS estão disponíveis no próprio *site* (https://angularjs.org/).

1.2.1 Iniciando uma aplicação com as referidas diretivas do *framework*

Antes de tudo, é necessário entender o conceito MVC, em que se tem três estruturas que ficam em torno dessa aplicação. Essas três estruturas são:

- *Model*: o código da página, que deve executar ações e comandos;
- *View*: a visualização do conteúdo, retorno final ao usuário, em que se recebem os dados do controlador e do *Model*;
- *Controller*: o controlador, que gerencia a utilização dos códigos do *Model* e realiza a estruturação da *View*.

Por tratar-se de um *framework* MVC, o *Controller* da nossa aplicação deve ser construído em Javascript, num arquivo separado, como o seguinte:

```
1. var app = angular.module('angularAPP', []);
2. app.controller('angularCtrl', function($scope) {
3. $scope.nome= "Gabriel";
4. $scope.sobrenome= "Ramos";
5. $scope.quantidade= 1;
6. $scope.preco= 2;
7. $scope.nomes=[
8. {nome:'ContactAPP',cidade:'Santos'},
9. {nome:'Gabriel Ramos',cidade:'São Vicente'},
10. {nome:'IFSP',cidade:'Cubatão'}]
11.
12.});
```

Nesse trecho, é criado um *controller* para gerenciar a aplicação *angularAPP*.

Os códigos HTML são simples, basta criar um *template* básico e, para iniciar a utilização do *framework*, baixar as bibliotecas (do próprio AngularJS e do jQuery) e importá-las no projeto, utilizando a *tag script*.

Salva-se o *controller* citado como “*controller.js*”, na pasta “*js*” (mesma pasta em que, no exemplo, ficam salvos todos os arquivos javascript). Assim, as chamadas dos documentos ficariam:

```
1. <script src=”js/angular.min.js"></script>
2. <script src=”js/jquery-2.1.1.min.js"></script>
3. <script src=”js/controller.js"></script>
```

Dentro do html, é preciso especificar a aplicação que se está em utilização e o *controller* (de acordo com o nome dado no arquivo feito), da seguinte forma:

```
1. <body ng-app="angularAPP" ng-controller="angularCtrl">
```

No exemplo acima, especificou-se que toda a página será somente uma aplicação (*angularAPP*) e controlada somente pelo *controller angularCtrl* (diretivas *ng-app* e *ng-controller*).

Utilizando as diretivas *ng-click* e *ng-class*, é possível tratar a aplicação como abas, semelhante a um navegador, que carrega o conteúdo, mas exibe somente a página em que o usuário estiver navegando, da seguinte forma:

```
1. <ul>
2.<li class="tabs-li"><a ng-click="tab=1" ng-class="{'active' : tab==1}">INÍCIO</a></li>
3.<li class="tabs-li"><a ng-click="tab=2" ng-class="{'active' : tab==2}">SEGUNDA ABA</a></li>4.</ul>
```

Nesse trecho, indicou-se que cada *link*, dentro de cada elemento da lista, é correspondente a uma aba da aplicação, recebendo a classe “*active*” ao ser clicado (diretivas *ng-click* e *ng-class*). O conteúdo de abas pode ser exibido da seguinte forma:

```
1. <section ng-show="tab==1">
2. </section>
3. <section ng-show="tab==2">
4. </section>
```

Assim, quando o primeiro *link* for clicado, somente o conteúdo da primeira *section* será exibido. O mesmo ocorrerá com o segundo *link* (diretiva *ng-show*).

Do tratamento de dados, possui-se as diretivas *ng-model, ng-bind, ng-init e ng-repeat*. Estas podem ser utilizadas das seguintes formas:

```
Nome: <span ng-bind="nome"></span><br>
        Sobrenome: <span ng-bind="sobrenome"></span><br><br>
Concatenação <br>
        Nome completo: {{nome+ " "+sobrenome}}
```

Dessa forma, ao executar a aplicação, os campos com "*ng-bind*" retornarão aos valores correspondentes (no caso, nome e sobrenome) que estão sendo inseridos pelo *controller*, que possui as variáveis indicadas, sendo concatenados através das chaves duplas.

```
1. Digite seu nome <input type="text" ng-model="name" ng-init="name='aqui'"><br>
2. Seu nome vai aparecer <span ng-bind="name"></span>
```

No exemplo acima, um campo de texto possui o modelo (diretiva *ng-model*) aplicado à *name* e um valor inicial (*ng-init*) atribuído como "aqui". Utilizando a diretiva *ng-bind*, é possível pegar, em tempo de execução, o que o usuário digitar, e mostrar, dentro do *span*, que possui o mesmo valor correspondente, porém, em seu *bind*. Ou seja, o nome do modelo aplicado ao "*model*" é direcionado ao *bind* que possuir o mesmo nome.

A diretiva *ng-init* serve para atribuir valores iniciais à aplicação. No último exemplo, é utilizado um valor "aqui" como inicial ao campo de texto que possui *ng-model*="*name*". O mesmo é possível fazer até com as abas. Caso seja especificado *ng-init*="*tab*=1" em que é iniciada a aplicação (ao abrir a *tag body*), a primeira aba já estará pronta para visualização assim que a página for executada.

A diretiva *ng-repeat* é utilizada para criar um laço de repetição (semelhante ao *for each* em outras linguagens de programação), mediante cada item dentro de um *array* no *controller*, como o exemplo a seguir:

```
1. <ul class="list">
2.   <li ng-repeat="x in nomes">
3.        {{"Nome: "+ x.nome + " - Cidade: " + x.cidade }}
4.</li>
5. </ul>
```

Ao executar a aplicação, uma quantidade de itens da lista será gerada dinamicamente, dependendo da quantidade de registros que existirem no *array* “nomes” dentro do *controller*.

Outros exemplos estão disponíveis no link a seguir: <http://gabrieluizramos.github.io/contactapp/angularjs/>.

2 CONTACTAPP: A APLICAÇÃO

Figura 1 – Aplicação ContactApp

Partindo do apresentado, a aplicação final foi desenvolvida utilizando as duas tecnologias expostas: AngularJS, para gerenciar as páginas do aplicativo, e o banco de dados Parse, para regularizar a performance dos dados a serem trabalhados.

A aplicação conta com um *controller*, o qual insere alguns valores no conteúdo HTML da página, um *script* para as funções de cadastro, leitura, armazenamento e atualização de dados e outros dois *scripts* que realizam uma integração entre o sistema desenvolvido e o meio de acesso pelo qual o usuário está conectado, sendo essa integração com *hardware* (pegando a localização do usuário e a respectiva latitude e longitude) e com *software* (capaz de compartilhar a aplicação no aplicativo *Whatsapp*, caso o usuário esteja acessando por um dispositivo móvel).

Para exemplificar e complementar o objetivo do projeto, o mesmo recebeu, ludicamente, a alcunha “*ContactApp*”, o qual possui como foco armazenar contatos do usuário e ser uma “*WebAgenda*”.

Figura 2 – Tela inicial

A utilização inicia-se a partir do item “Nova Agenda”, no qual o usuário realiza cadastro, informando seu próprio nome e um nome para sua agenda. A seguir, uma ID é retornada ao usuário. É através dessa informação que o usuário utilizará o sistema.

No item “Agenda já Existente”, o usuário deve informar a agenda e sua referida ID.

Através desses dados, o usuário possui a opção de cadastro, consulta, atualização e remoção dos seus contatos da agenda, também se valendo de uma ID para cada contato.

No item “Localização”, ocorre a integração com a geolocalização do usuário, retornando, ao mesmo, as informações sobre sua posição, as quais também já foram consultadas ao iniciar a aplicação e armazenadas ao criar a agenda.

No item “Sobre”, o usuário pode conferir algumas informações sobre o projeto, (dentre elas, estão os nomes dos desenvolvedores e o nome científico dado ao projeto) e realizar o compartilhamento da aplicação via WhatsApp, caso possua e esteja acessando por um smartphone ou semelhante (função encontrada no destino <http://whatsapp-sharing.com/> para uso livre).

2.1 Tratamento interno da aplicação

O formulário para cadastro dos usuários e suas agendas possui tratamento de erro, pois o Parse, o banco de dados, só permite caracteres alfanuméricos. Tal validação é feita através de Javascript, utilizando uma expressão regular.

Por tratar-se de uma aplicação de página única (*single page application*, sigla *SPA*) e, portanto, realizar somente uma única requisição ao servidor ao acessar a página, foi utilizado, também, SASS, o pré-processador de CSS, capaz de comprimir o código CSS e criar variáveis e funções, melhorando o desenvolvimento e carregamento da página pelo *browser*. A aplicação apresenta-se ao usuário de forma responsiva, ou seja, adaptando seu conteúdo para melhor visualização através do meio de acesso em que o usuário se encontra. Aplicações de conteúdo, atributos e extensões *WAI-ARIA* também foram utilizados, melhorando a aplicação quanto à sua disponibilidade para leitores de tela e dispositivos que necessitem da referida tecnologia.

Ao cadastrar a agenda, o usuário é cadastrado no banco de dados *Parse*, na classe “usuários”, e uma nova classe é criada, na qual ficarão armazenados os contatos. Essa nova classe foi desenvolvida para ter como nome a ID do usuário e o nome que lhe foi atribuída na hora do cadastro.

A aplicação completa está disponível em:

<http://gabrieluizramos.github.io/contactapp>

CONCLUSÃO

Ao desenvolver a aplicação, foram possíveis a descoberta e a integração dos desenvolvedores com o ambiente em nuvem (o banco de dados), criando uma nova perspectiva de programação e desenvolvimento.

Com os conhecimentos obtidos, novos ideais foram afirmados, como a importância de uma plataforma estável e com seu conteúdo e documentação consolidada, como o *Parse* e o *AngularJS*.

Foi de suma importância o desempenho dos participantes, principalmente no que diz respeito a pesquisas e as tentativas de soluções de possíveis erros operacionais. É também fundamental informar a necessidade e os agradecimentos à *internet* e a todos os colaboradores presentes em fóruns de discussão e qualquer tipo de debate realizado *on-line*, pois a participação externa foi imprescindível para que o resultado final do sistema fosse alcançado.

Com base nesses pensamentos, todos os códigos aqui expostos e operacionais podem ser encontrados e estão disponíveis para eventuais consultas e *download*s através do link abaixo (repositório da aplicação no *github*).

<https://github.com/gabrieluizramos/contactapp/>

REFERÊNCIAS

PARSE. Disponível em: <https://parse.com/platforms>. Acesso em 2 de agosto de 2015.

DIRECTIVE componentes in ng. Disponível em: <https://angularjs.org>. Acesso em 2 de agosto de 2015.

WEB Design & Aplicações. Disponível em: <http://www.w3c.br/Padroes/WebDesignAplicacoes>. Acesso em 2 de agosto de 2015.

RAMOS, Gabriel. Parse – Objetos na nuvem: Parse, um banco de dados, na nuvem, orientado a objetos. Disponível em: <http://tableless.com.br/parse-objetos-na-nuvem>. Acesso em 2 de agosto de 2015.