Projeto Final – Programação Avançada Para Dispositivos Móveis

Felipe Augusto Massari
Universidade Federal do ABC
rafael.pissardo@aluno.ufabc.edu.br

Rafael Augusto Pissardo
Universidade Federal do ABC
felipe.massari@aluno.ufabc.edu.br

OBJETIVO

O objetivo principal deste artigo é descrever e detalhar o Projeto Final realizado para a disciplina Programação Avançada para Dispositivos Móveis. A disciplina foi ministrada pelo Professor Doutor Diogo Martins, vinculado ao curso de Bacharelado em Ciência da Computação pela Universidade Federal do ABC.

Para a realização do Projeto, três linguagens de programação e duas arquiteturas foram fundamentais:

- Java
- Python
- Cake
- Android Studio
- CakePhP, Apache, PhPMyAdmin, Mysql

Palavras-chave

Android. Raspberry-pi. Internet Of Things. CakePhP. Web. Desenvolvimento Ágil.

1. INTRODUÇÃO

As tecnologias sem fio impactou as interações e formas de mobilidade, relações interpessoais e os espaços urbanos[1]. A popularização dos dispositivos móveis foi fundamental para o dinamismos das cidades disseminação de informações. Diante desta conjuntura a região do Metropolitana da cidade de São Paulo, que é a maior Metrópole brasileira com 39 municípios, 21 milhões de habitantes. com efeito a tecnologia se torna essencial para o dinamismo e agilidade no fluxo de informações e das pessoas em seu território.

Figura 1: Representação ilustrativa da região metropolitana

Só na cidade de São Paulo são 11 milhões[2] de habitantes residentes fora aqueles que se deslocam para usufruir dos recursos, gerando um grande fluxo e demanda por mobilidade. Em 2013, a cidade de São Paulo, contava com 700 empresas de fretamento[3], além do grande número de empresas, fazer o controle dos veículos e das linhas é um grande desafio para as empresas.

Como no Brasil o transporte coletivo é de baixa qualidade e eficiência[4], a população faz muito uso do transporte particular, como carros e motos por exemplo. Nesse contexto a tecnologia da informação vem ajudando a desenvolver ferramentas que auxiliam nos sistemas de transportes e eficiência dos serviços prestados[5].

A popularização dos smartphones fez deste uma ferramenta importante no cotidiano do cidadão e assumiu diversas funções[1] que facilitam as mais diversas atividades. Deste modo esta importante ferramenta é estratégica para o desenvolvimento de um aplicativo para dispositivos móveis, sendo esta capaz de situar com precisão para o usuário a localização do fretado, evitando longas esperas e aumentariam a eficiência do transporte fretado, deixando a opção mais atraente.

2. DEFINIÇÃO DO PROBLEMA

Para contextualizar o problema, considere-se um usuário que reside na cidade de São Caetano do Sul, e trabalha na Vila Olímpia. Ele deve percorrer de carro, cerca de 16 km de casa até o trabalho. Em horário de pico, este usuário leva cerca de duas horas para chegar ao trabalho e percorre cerca de 640 km por mês. No trânsito de São Paulo o consumo médio do veículo é de 7 km/Litro. O total gasto pelo usuário fica em torno de R$700,00. Já utilizando fretado, o custo sai no total de R$360,00. Existem empresa que fazem o custeio deste tipo de transporte para seus funcionários, somando mais uma característica atrativa, além do tempo gasto sendo de em média 50 minutos.

Com esses dados obtido através de um cidadão que utilizava transporte individual e migrou para o serviço de fretado, fica claro as vantagens de utilizar tal serviço. Para trazer mais conforto e dinamismo, foi proposto um aplicativo que vai além dos já disponíveis no mercado, pois o app proposto não precisa que o motorista ligue seu celular ou dados móveis, uma vez que o próprio ônibus tem um sistema que envia as coordenadas via GPS. Além disso, a interface é simples e prática com atualização a cada dois segundos.

O problema encontrado foi em desenvolver um aplicativo, inicialmente em Android, que pudesse suportar duas funcionalidade básicas, apoiar o usuário e ao mesmo ser dinâmico o suficiente para apoiar os gerenciadores da linha sem burocracias.

Fez-se uma pesquisa no Google Play para saber os aplicativos disponíveis para o usuário utilizar para fretado. Alguns apps tinham nota baixa com reclamações dos usuários como login, notificação, baixa acurácia entre outros. Os aplicativos consultados foram OrbBus, Vá de Fretado, Cade meu Fretado e Fretadão. O maior problema encontrado com em todos foi o fato de o motorista também precisa de um celular conectado para o aplicativo funcionar.

Para saber a real necessidade dos aplicativos tanto para funcionários do fretado quanto para os usuários, foi feito uma entrevista com o dono de uma linha de fretado, o Sr. Ismar da Loredo Turismo. Dentre as sugestões dada pelo proprietário do fretado, o principal foi a conectividade automática do ônibus no sistema e fácil uso por ambas as partes. Sendo assim, o foco do problema foi resolver essas exigências. O problema de praticidade na utilização foi resolvido com uma interface simples e auto explicativa com conexão através de login e senha. O problema de conexão automática foi resolvido usando Raspberry PI com GPS que são ligados assim que o veículo é ligado.

3. ARQUITETURA E IMPLEMENTAÇÃO

3.1 Serviço WEB

O sistema intermediário de toda aplicação é um serviço em nuvem, utilizando um sistema distribuído em chamado Cloud9 (C9). O C9 permite criar um máquina virtual em Linux que comporta toda inteligência de Banco de Dados, Autorização de Serviços e envio e recebimento de dados externos.

Na máquina utilizada, instalou-se um sistema com o Framework do PHP, o CakePhP, vinculado a ele tem-se um sistema de Banco de Dados (MySql), um gerenciador de Banco de Dados (PhPMyAdmin) e um serviço Apache que permite acesso a todo serviço.

O Bando de Dados foi estruturado da seguinte forma:

- Tabela buses
- Tabela data
- Tabela log

A tabela buses, tem como objetivo único, o armazenamento de informações genéricas e de controle de cada ônibus, referenciando sua Companhia:

#	Nome	Tipo	Agrupamento (Collation)	Atributos	Nulo	Predefinido	Extra
1	id	int(11)			Não	None	AUTO_INCREMENT
2	company	varchar(11)	latin1_swedish_ci		Não	None	
3	photo	int(11)			Sim	NULL	
4	dir	int(11)			Sim	NULL	
5	created	datetime			Não	None	

A tabela data, tem como o principal objetivo guardar a última informação recebida pelo Sistema Web de geolocalização e comportamento do ônibus, bem como algumas informações de acesso do aplicativo. Ela é dinamicamente ativa, e pode ser vista como um quadro ao vivo do veículo rastreado.

#	Nome	Tipo	Agrupamento (Collation)	Atributos	Nulo	Predefinido	Extra
1	id	int(11)			Não	None	AUTO_INCREMENT
2	lat	varchar(13)	latin1_swedish_ci		Sim	NULL	
3	lon	varchar(13)	latin1_swedish_ci		Sim	NULL	
4	bus_id	int(11)			Não	None	
5	modified	datetime			Não	None	
6	vel	int(11)			Não	None	
7	login	varchar(258)	latin1_swedish_ci		Não	None	
8	password	varchar(258)	latin1_swedish_ci		Não	None	

Os dados são recebidos por um Controller do CakePhP, e após alguns tratamentos, é salvo nesta tabela. Abaixo, um exemplo de como é armazenado:

	id	lat	lon	bus_id	modified	vel	login	password
Edita Copiar Apagar	1	-23.616435	-46.5558363	1	2016-11-29 21:47:00	30	fretado	fretado

A tabela log, armazena todo o histórico do veículo. Ou seja, cada dado recebido e armazenado (por algum instante) no Data, é armazenado definitivamente no Log. Esta tabela é de fundamental importância para as recuperações do percurso, e para a definição de uma das funções do aplicativo, abordada mais adiante. Ela foi estruturada da seguinte forma:

#	Nome	Tipo	Agrupamento (Collation)	Atributos	Nulo	Predefinido
1	id	int(11)			Não	None
2	bus_id	int(11)			Não	None
3	lat	varchar(11)	latin1_swedish_ci		Não	None
4	lon	varchar(11)	latin1_swedish_ci		Não	None
5	vel	int(11)			Não	None
6	modified	datetime			Não	None

Os dados armazenados têm a seguinte estrutura:

bus_id	lat	lon	vel	modified
1	-23.5644841	-46.6525808	0	2016-09-15 13:55:18
1	-23.5644841	-46.6525808	0	2016-09-15 13:57:35
1	-23.5644841	-46.6525808	0	2016-09-15 14:09:42
1	-23.5644841	-46.6525808	0	2016-09-15 14:09:55
1	-23.5644841	-46.6525818	0	2016-09-15 14:10:26
1	-23.5644841	-46.6525808	0	2016-09-15 14:25:45
1	-23.5679214	-46.6488198	0	2016-09-15 14:28:44
1	-23.5679214	-46.6438198	0	2016-09-15 14:28:53
1	-23.5679773	-46.6488438	0	2016-09-15 14:32:46

Os testes realizados neste relatório, foram simulados com 1800 dados coletados de forma real. Considerando que os dados são enviados ao Serviço Web a cada 3 segundos, podemos considerar que os dados foram coletados em um período total de uma hora e meia. Tal quantidade mostrou-se satisfatória para testes e simulações futuras.

3.2 Sistema Físico de Rastreamento

O envio de dados para o Serviço Web e a coleta das variáveis dinâmicas desejáveis foram realizadas por um dispositivo construído com a placa de Desenvolvimento Ágil, Raspberry-Pi 2.

Estruturado em três bibliotecas básicas, o Raspberry utiliza um Modem 3g pré-configurado de qualquer operadora do Brasil, um GPS Neo-6M para comunicação com satélites de geolocalização, e para vincular a transferência de dados, utiliza uma biblioteca de requisição por URL.

Portanto, as três bibliotecas são:

- Sakis 3G - Conexão com o serviço de Internet da Operadoras
- GPSD - Processamento e tradução dos dados recebidos por satélite
- URLLIB - Requisições HTTP para o Serviço Web

Esta é a única parte do projeto que há um custo de implementação propriamente dito. Ao todo, foi gasto de material, 600 reais. Tal custo é básico para o projeto.

- Raspberry Pi 2, em torno de 300 Reais.
- GPS Neo 6M, 90 Reais.
- Sistema 3G, 100 Reais

O Sistema 3G é calculado como um valor fixo, pois, caso a linha de fretado tenha WiFi, o Sistema Físico pode utilizá-lo.

Portanto, o Sistema Físico apresentado é de certa simplicidade, porém, representa o ponto chave do Sistema como um todo. A quantidade de amostra, a precisão dos dados gerados, a confiabilidade do Sistema, depende diretamente deste dispositivo criado.

3.3 Aplicação Android

Por motivos óbvios, esta foi a parte do Sistema que mais demorou para ser gerada e a que mais teve ajustes.

Ao final, a Aplicação conta com três atividades:

- Mapa
- Login
- Splash (Tela de Apresentação)

Na Splash, é criada uma atividade que dura cinco segundos e tem como o objetivo apresentar o logo do Aplicativo criado e, assim, gerar uma identidade visual para o usuário, seja ele Administrador ou Usuário Final.

```
public class Splash extends Activity implements Runnable {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_splash);

        Handler handler = new Handler();
        handler.postDelayed(this, 5000);
    }

    public void run(){
        startActivity(new Intent(this, Login.class));
        finish();
    }
}
```

Nesta Atividade, implementou-se alguns aspectos básicos da programação em Android, como a invocação de outra Atividade externa e o uso de métodos especificos do Android.

A Atividade Splash invoca outra Atividade, chamada Login. Esta atividade tem como objetivo limitar o acesso dos usuários à localização do fretado, porém sem adicionar algum processo burocrático.

Como mencionado anteriormente, todos os aplicativos existentes no mercado, como o carro-chefe, 'CadeMeuFretado', expõe de forma aberta e sem verificações outros sistemas rastreados de fretado. Ou seja, um usuário de fretado em São Paulo, consegue visualizar um fretado que utilize o mesmo Sistema em Minas Gerais, por exemplo.

Como apontado por um dos donos de uma companhia de fretados, o interesse de um controle mínimo dos usuários, o Aplicativo solicita uma senha e um usuário, fixos, que devem ser repassados por colaboradores da companhia aos novos usuários.

Uma vez colocada a senha e o usuário, corretamente, o Aplicativo utiliza uma função do Android que salva as preferências pré-determinadas pelo desenvolvedor. No caso, criou-se uma flag, que avisa ao Aplicativo se o login já teria sido realizado anteriormente com sucesso. Caso tenho, a tela de login é ignorada, e a aplicação contínua imediatamente.

Portanto, temos alguns métodos fundamentais nesta Atividade:

```
public void verifyingAccess(final String username, final String password){
    button_login = (Button)findViewById(R.id.button);
    edit_bus_code = (EditText)findViewById(R.id.editText);
    edit_password = (EditText)findViewById(R.id.editText2);
    button_close = (Button)findViewById(R.id.button2);
    sharedpreferencesPass = getSharedPreferences(MyPREFERENCES, Context.MODE_PRIVATE);
    final String[] j = {sharedpreferencesPass.getString(prefPassword, " ")};
    Log.i(TAG, "**************************************************");
    Log.i(TAG, j[0]);
    Log.i(TAG, password);
    if(j[0].equals(password)){
        Toast.makeText(getApplicationContext(),
                "Localizando...", Toast.LENGTH_LONG).show();
     callmap();
    }else {
        button_login.setOnClickListener((v) -> {
                if (edit_bus_code.getText().toString().equals(username) &&
                        edit_password.getText().toString().equals(password)) {
                    Toast.makeText(getApplicationContext(),
                            "Localizando...", Toast.LENGTH_LONG).show();
                    SharedPreferences.Editor editorPass = sharedpreferencesPass.edit();
                    editorPass.putString(prefPassword, edit_password.getText().toString());
                    editorPass.commit();
                    j[0] = sharedpreferencesPass.getString(prefPassword, " ");
                    Log.i(TAG, "**************************************************");
                    Log.i(TAG, j[0]);
                    callmap();
                } else {
                    Toast.makeText(getApplicationContext(), "Dados Inválidos", Toast.LENGTH_LONG).show();
                }
        });
    }
    button_close.setOnClickListener((v) -> {

            finish();
    });
}
```

No código acima, fica claro o uso da flag para pular a verificação de usuário e senha da atividade, bem como a utilização do método getSharedPreferences, que permite o armazenamento da flag.

Para saber, dinamicamente, qual os dados de acesso daquela linha de fretado, ou seja, daquele Aplicação, utilizou-se o seguinte bloco de código:

```
private class MarkerTask extends AsyncTask<Void, Void, String> {
    private static final String LOG_TAG = "ExampleApp";
    private static final String SERVICE_URL = "https://fretbus-rpissardo.c9users.io/fretbus/data/map2/1";
    @Override
    protected String doInBackground(Void... args) {
        HttpURLConnection conn = null;
        final StringBuilder json = new StringBuilder();
        try {
            URL url = new URL(SERVICE_URL);
            conn = (HttpURLConnection) url.openConnection();
            InputStreamReader in = new InputStreamReader(conn.getInputStream());
            int read;
            char[] buff = new char[1024];
            while ((read = in.read(buff)) != -1) {
                json.append(buff, 0, read);
            }
        } catch (IOException e) {
            Log.e(LOG_TAG, "Error connecting to service", e);
        } finally {
            if (conn != null) {
                conn.disconnect();
            }
        }
        return json.toString();
    }

    @Override
    protected void onPostExecute(String json) {
        try {
            JSONObject jsonObj = new JSONObject(json);
            String username = jsonObj.getString("login");
            String password = jsonObj.getString("password");
            Log.i(TAG, username);
            Log.i(TAG, password);
            verifyingAccess(username,password);
        } catch (JSONException e) {
            Log.e(LOG_TAG, "Error processing JSON", e);
        }
    }
```

Neste bloco, o Serviço Web retorna em formato JSON, o usuário e a senha necessária para acesso ao sistema. Tais dados são modificáveis pelo Administrador.

Este bloco é invocado inicialmente pela declaração *new MarkerTaks().execute()*. Possibilitando conexões HTTP sem conflitos ou DeadLocks.

Após todas as verificações necessárias, utilizou-se, novamente, conceitos básicos visto em aula para invocação de novas atividades:

```
private void callmap() {
    startActivity(new Intent(this, MapsActivity.class));
}
```

A atividade do Mapa é, sem dúvida, a mais importante da Aplicação e a que mais demandou trabalho.

Utilizando a API do Google Maps, bem como uma Key para autorizar tal aplicação, criou-se um mapa que utiliza a Layer de trânsito da Google, bem como tem alguns markers como foco.

A primeira modalidade da aplicação é o acompanhamento em tempo real do veículo. Para tal, o ícone de um ônibus é utilizado, e o mapa, cada três segundo, é forçado a centralizá-lo na tela do celular.

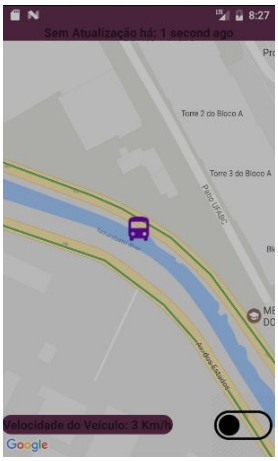
A segunda modalidade é o acompanhamento real do veículo, porém com geração parcial do histórico na tela. Para tal, utilizou-se o ícone de alvo.

```
public class MapsActivity extends FragmentActivity implements OnMapReadyCallback {
    private GoogleMap mMap;
    private static final String TAG = "Http Connection";
    ToggleButton toggle;
    int flag = 1;
    int flagant = 1;
    @Override
    public void onCreate(Bundle savedInstanceState) {...}



    @Override
    public void onMapReady(GoogleMap googleMap) {...}


    public void updateMap(){
        new Timer().scheduleAtFixedRate(() -> {
                new MarkerTask().execute();
        },0,3000);
    }

    public void updateCamera(LatLng local){
        mMap.animateCamera(CameraUpdateFactory.newLatLngZoom(local,18));
    }

    public void updateMarker(final LatLng local){...}

    public void updateVelocity(int velocity){...}

    public void updateModifiedAt(String modified){...}

    private class MarkerTask extends AsyncTask<Void, Void, String> {...}
}
```

A imagem acima, demonstra de forma clara o fluxo seguido pela Atividade. Algumas flags são determinadas para a manipulação dos dados na tela do celular. Novamente, um bloco de códigos para a requisição de processos HTTP é utilizado, e é requisitado a cada três segundos. Portanto, chegamos a um ponto fundamental do projeto e que vale ser reforçado. O sistema tem uma atualização mínima de três segundos. Variando de acordo com a disponibilidade dos dados enviados pelo Sistema Físico embarcado no veículo.

Na tela, duas informações, (além das visuais) são exibidas. São elas, a quanto tempo faz a última atualização do sistema e a quantos km/h o veículo está se locomovendo. Essas foram as duas principais informações apontadas pelos motoristas e administradores de linhas que foram entrevistados.

Além disso, alguns elementos xml foram modificados para melhorar o design da Aplicação. A tela final do Sistema ficou como a figura abaixo, para o rastreio ao vivo:



E o rastreio com histórico local:

4. AVALIAÇÃO

Em geral o aplicativo cumpriu todas as metas propostas, os testes foram feitos primeiramente usando dados reais coletados da localização com carro dos desenvolvedores, posteriormente foi feito um teste com o fretado, os resultados dos testes foram bem sucedidos.

A aplicação ficou leve e versátil, usa pouca memória e ocupa pouco espaço em disco. O aplicativo usa uma baixa API (20) e pode ser utilizada em 90% dos celulares Android, e seguem requisitos gerados pela Google.

No primeiro momento um possível cliente aprovou o aplicativo e as funcionalidades o agradaram, é possível fazer melhorias e adicionar funcionalidades no futuro, mas a princípio a aplicação agradou a todos. Outro aspecto fundamental alcançado, é o fato de que mesmo aplicando camadas de segurança ou pseudo-segurança, o sistema não se tornou burocrático e os possíveis administradores aprovaram a ideia.

5. DISCUSSÃO

No projeto, foi necessário implementar, entre outros, aspectos de Desenvolvimento Ágil e Clean Code. Este segundo foi de suma importância para o entendimento de ambos os desenvolvedores e para melhor desenvolvimento de código a fim de uma melhor

avaliação. Outro questão é gerir os desejos latentes dos usuários e se atualizar quanto às demandas.

Por não possuir conhecimentos em programação ou em tecnologias disponíveis, muitos usuários citaram tecnologias trabalhosas ou caras, que foram rebatidas e descartadas a medida do possível.

Embora o tempo limitado para ser desenvolver o projeto, o mesmo pode ser considerado um sucesso e obtive objetivo alcançado. Outro fator surpreendente foi o de um projeto relativamente simples, unir conceitos fundamentais da Computação, como Sistemas Distribuídos, Banco de Dados, Segurança de Dados, Grafos, Programação Web entre outras. Porém, sem dúvida a que mais auxiliou o processo foi o conceito de Sistema Distribuído.

6. REFERÊNCIAS

Link para github:

https://github.com/rpissardo/ProjetoPADMUFABC.git

[1] TAROUCO, Fabricio. A Metrópole Comunicacional e a Popularização dos Apps para Dispositivos Móveis.

[2] http://economia.estadao.com.br/noticias/geral,populacao-economicamente-ativa-da-grande-sp-tem-1-queda-em-20-anos,136168e

[3] http://g1.globo.com/sao-paulo/anda-sp/noticia/2013/06/vias-de-sao-paulo-tem-transito-liberado-para-onibus-fretados.html

[4] LEITE, Daniel Farias Batista; ROCHA, Júlio Henrique; DE SOUZA BAPTISTA, Cláudio. Busão: um Sistema de Informações Móvel para Auxílio à Mobilidade Urbana Através do Uso de Transporte Coletivo. IX Simpósio Brasileiro de Sistemas de Informação, p. 170-181, 2013.

[5] Sussan, J. Perspectives on Intelligent Transportation System. Springer, New York, USA, 2005