Ementas e Parques da

Universidade de Aveiro

Desenvolvimento de Aplicações  
para Dispositivos Móveis

Escola Superior de Tecnologia

e Gestão de Águeda

Universidade de Aveiro

2º semestre

2017-2018

Rui Duarte nº 29979

Índice

[1. Introdução 2](#_Toc516472736)

[2. Atividades 3](#_Toc516472737)

[2.1. MainActivity 4](#_Toc516472738)

[2.2. Cantinas 7](#_Toc516472739)

[2.3. Ementas 8](#_Toc516472740)

[2.4. Parques 11](#_Toc516472741)

[3. Manifest.xml 14](#_Toc516472742)

[4. Ícone da aplicação 15](#_Toc516472743)

[5. Considerações finais 16](#_Toc516472744)

[5.1. A melhorar/Implementar: 16](#_Toc516472745)

1. Introdução

A aplicação foi desenvolvida recorrendo ao *Android Studio* e usado o *GIT* como controlo de versões e gestão de código fonte, alojado em <https://github.com/rduarte83/DADM/commits/master>.

Tem como principais funcionalidades:

* Acesso às ementas do dia das várias cantinas da UA
* Ementas sensíveis à hora de visualização (Almoço/Jantar)
* Disponibilidade de lugares de estacionamento dos vários parques da UA
* Ordenação automática por proximidade (baseada no GPS)
* *Timeout* personalizável para arranque automático dos parques
* Suporte para tablets
* Compatibilidade com Android 4.4+ (API 19)

1. Atividades

De seguida são apresentadas as atividades da aplicação. De referir que todas elas, com exceção da ***MainActivity*** (por motivos óbvios), estão munidas de navegação para a atividade inicial, assim como o respetivo *back button*, implementado com o código abaixo indicado.

getSupportActionBar()).setDisplayHomeAsUpEnabled(true);

@Override  
public boolean onOptionsItemSelected(MenuItem item) {  
 //back button  
 if (item.getItemId() == android.R.id.*home*) {  
 this.finish();  
 return true;  
 }  
 return super.onOptionsItemSelected(item);  
}

Entretanto acabei por descobrir que tal implementação era bastante mais simples se implementado pelo manifest.xml com o código abaixo indicado.

<activity android:name=".AMinhaActividade"  
          android:parentActivityName=".MainActivity">  
    <!-- The meta-data tag is required if you support API level 15 and lower -->  
    <meta-data  
        android:name="android.support.PARENT\_ACTIVITY"  
        android:value=".MainActivity" />  
</activity>

Os layouts foram criados recorrendo aos *RelativeLayouts* que simplificam bastante a tarefa de posicionar os elementos no ecrã e se adaptam a vários tipos de resoluções.

De referir, ainda que as mensagens de alertas foram inicialmente implementadas usando *Toasts* sendo, à posteriori, após sugestão dos *guidelines* da Google, substituídas por *SnackBars*, uma forma bastante mais atrativa e moderna de apresentar o mesmo tipo de informação.

* 1. *MainActivity*



Figura 1 - Ecrã Inicial

A ***MainActivity***, é a atividade principal que é inicializada quando a aplicação inicializa (fig. 1)

É constituída por:

* uma *image\_view* no topo
* uma *text\_view* no centro
* um botão para acesso às ementas
* um botão para acesso aos parques
* uma *progress\_bar* com um contador

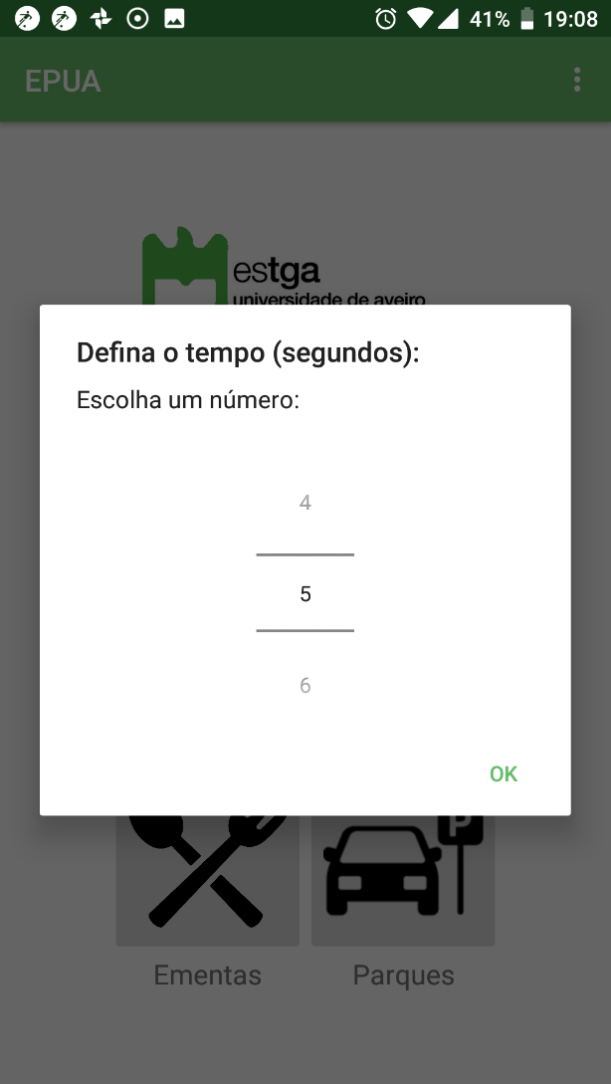
 

Figura 2 – Opção Arranque Figura 3 – Caixa de diálogo Arranque

O *timeout* predefinido da aplicação são 5 segundos podendo esse tempo ser alterado a qualquer momento acedendo à opção “Arranque” do menu (fig. 2) que invoca a caixa de diálogo *AlertDialog* (fig. 3). O novo valor é armazenado num ficheiro MyPrefs.xml recorrendo ao *SharedPreferences*. O *timeout* é interrompido e a respetiva *progressBar* e contador são ocultados sempre que o utilizador aceder a qualquer uma das opções disponíveis. O código dessa implementação está disponibilizado abaixo:

cdt.cancel();  
pbar.setVisibility(View.*GONE*);  
tv\_progress.setVisibility(View.*GONE*);

De notar que o *timeout*, inicialmente, foi implementado recorrendo a um *Handler* e a um *Runnable* mas, por uma questão de simplicidade, acabei por recorrer ao *CountDownTimer* que torna o processo de implementação bastante mais facilitado, como demonstrado abaixo:

Implementação inicial:

final Handler mHandler = new Handler();  
final Runnable runnable = new Runnable() {  
 @Override  
 public void run() {  
 Intent it = new Intent(ctx, Parques.class);  
 startActivity(it);  
 }  
};

mHandler.postDelayed(runnable, timeout);

Implementação final:

cdt = new CountDownTimer (timeout, 1000) {  
 public void onTick(long millisUntilFinished) {  
  
 int delta = (int) (timeout - millisUntilFinished)/(timeout/100);  
 pbar.setProgress(delta);  
 String progress = getString(R.string.*progress*)+ String.*valueOf*((millisUntilFinished / 1000)+1);  
 tv\_progress.setText(progress);  
 }

* 1. Cantinas

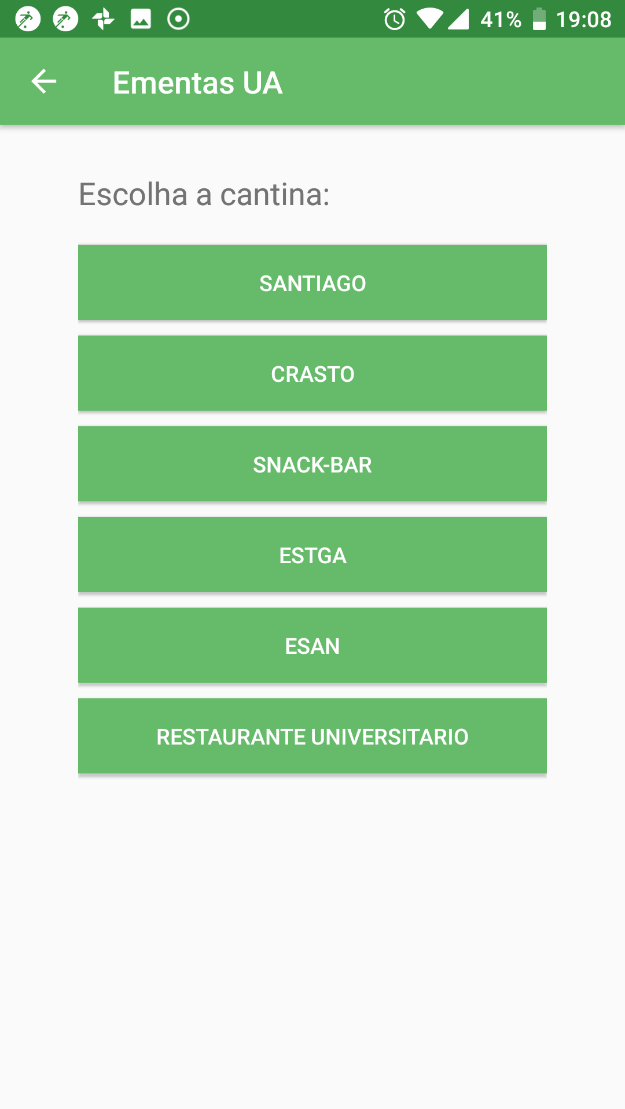


Figura 4 - Ecrã de seleção das cantinas

A atividade ***Cantinas*** permite ao utilizador escolher a cantina a que pretende aceder (fig. 4).

* 1. Ementas

Figura 5 - Menu Almoço Figura 6 - Menu Jantar

A atividade ***Ementas*** apresenta ao utilizador os pratos do dia e respetivas opções consoante a cantina selecionada na atividade anterior. Apresenta ainda um botão no canto superior direito que permite alternar entre almoço e jantar (fig. 5 e 6).

A apresentação da informação é feita através de cardviews que permitem um efeito visual bastante mais apelativo que a tradicional lista.

O método *verificaMeal* tem a simples função de alternar a variável *mealType* entre 0 e 1, de forma a conseguir alternar a vista entre o almoço e o jantar.

O método *verificaHora* usa o *GregorianCalendar* que permite obter a data e hora atual. Com essa informação verifica a hora e, se a mesma se encontrar entre as 15 e as 21, apresenta de forma predefinida o jantar.

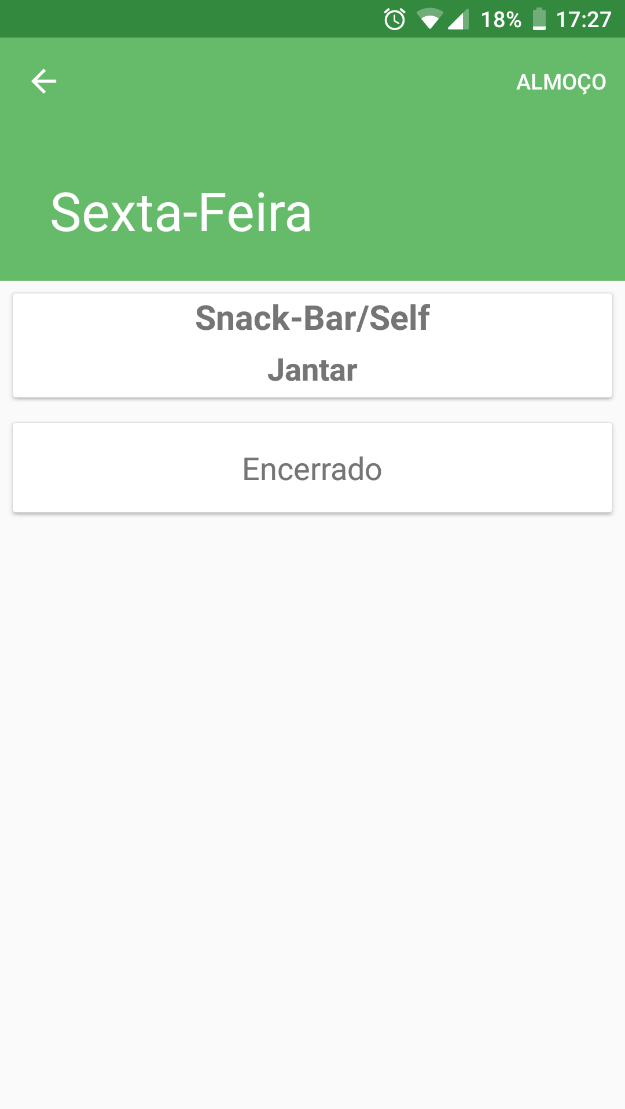


Figura 7 - Cantina Encerrada

O método *parseJSON* efetua um pedido de ligação à API da UA em causa usando o URL <http://services.web.ua.pt/sas/ementas?date=week&place=all&format=json> e que recorre ao *Volley* que é uma forma bastante simplificada de implementação.

Como referido em <https://developer.android.com/training/volley> recebe um *JsonObject*. Considerando que o *Json* em causa tinha uma estrutura pouco normalizada, o tratamento do mesmo teve de ser individualizado, o que fez com que o código ficasse de todo otimizado.

O metodo *insereJson*” começa por traduzir os dias da semana para Português e, de seguida, insere as variáveis criadas pelo *parseJson* nos *cardviews* correspondentes. Durante esta inserção faz a importante tarefa de “filtrar” as possíveis falhas na normalização acima referidas, de forma a evitar possíveis erros, como exemplificado na figura 7.

O excerto de código abaixo indicado exemplifica a estrutura do *Json*:

SANTIAGO  
Menus.[0].Menu.[0] - Almoco  
Menus.[0].Menu.[1] - Jantar  
Sopa 0 Carne 1 Peixe 2 Dieta 3 Veget 4 Opcao 5 Salada 6 Diversos 7 Sobremesa 8  
CRASTO  
Menus.[0].Menu.[2] - Almoco  
Menus.[0].Menu.[3] - Jantar  
Sopa 0 Carne 1 Peixe 2 Dieta 3 Veget 4 Opcao 5 Salada 6 Diversos 7 Sobremesa 8  
SNACK-BAR  
Menus.[0].Menu.[4] - Almoco  
Menus.[0].Menu.[5] - ND  
Sopa 0 Carne 1 Peixe 2 Dieta 3 Veget 4 Opcao Salada Diversos Sobremesa 5  
ESTGA  
Menus.[1].Menu.[0] - Almoco  
Menus.[1].Menu.[1] - Jantar  
Sopa 0 Carne 1 Peixe 2 Dieta 3 Veget 5 Opcao Salada 4 Diversos Sobremesa 6  
ESAN  
Menus.[2].Menu.[0] - Almoco  
Menus.[1].Menu.[1] - Jantar  
Sopa 0 Carne 1 Peixe 2 Dieta 3 Veget 5 Opcao Salada 4 Diversos Sobremesa 6  
RESTAURANTE UNIVERSITARIO  
Menus.[3].Menu.[0] - Almoco  
Menus.[3].Menu.[1] - ND  
Sopa 0 Carne 1 Peixe 2 Dieta Veget Opcao 3Salada 4 Diversos Sobremesa 5

* 1. Parques



Figura 8 – Ecrã Parques

A atividade ***Parques*** (fig. 8), apresenta informações relacionadas com os parques de estacionamento da Universidade de Aveiro (número e nome do parque, distancia que se encontra do utilizador, indicador gráfico de taxa de ocupação e lugares disponíveis). Recorre a uma API semelhante à usada nas Ementas, mas bastante mais estruturada e organizada.

O tratamento de erros aqui efetuado, quando comparado com as ***Ementas***, foi bastante mais simples, evitando apenas incoerências de informação, como lugares livres negativos e lugares livres acima da capacidade total do parque.

O Indicador gráfico facilita a visualização da informação.  
  
Legenda:  
 - Taxa de ocupação = 100% (0 lugares livres)  
 - Taxa de ocupação <100% e >= 75%  
 - Taxa de ocupação <75%

Esta atividade foi implementada recorrendo a uma *RecyclerView*, uma versão bastante mais flexível e avançada da *ListView* (versão inicialmente implementada).   
Por uma questão de performance e, por a lista em causa ser fixa, foi utilizado o seguinte código.

recyclerView.setHasFixedSize(true);

Esta lista é ordenada por distância, recorrendo à localização atual do dispositivo, utilizando o seguinte código:

Collections.*sort*(parqueList, new Comparator<Parque>() {  
 public int compare(Parque p1, Parque p2) {  
 return Float.*compare*(p1.getDistancia(), p2.getDistancia());  
 }  
});

O método *getLocation* implementa o *FuseLocationClient*, uma forma bastante simples de receber a última localização. O método verifica se a aplicação tem permissões para aceder à localização e, se tal não acontecer, requisita-as.



Figura 9 – AlertDialog invocado quando a localização não está ativa

De seguida vai verificar se o serviço de localização está ativo e pede a última localização. Se tal não acontecer chama o método *gpsPrompt* que cria um *AlertDialog* que encaminha o utilizador para ativação desse mesmo serviço (fig. 9).

Apesar de bastante funcional, este sistema não está livre de alguns bugs. O facto de o método não requisitar um novo pedido de localização, mas sim a última localização faz com que, ocasionalmente esse valor seja *null*. A forma de contornar essa situação é abrir um software que usa a localização como, por exemplo o Google Maps, aguardar alguns segundos e, de seguida, voltar a abrir a aplicação.

Para finalizar, de forma a atualizar (refrescar) a informação, foi implementada a opção atualizar no menu e ainda o *swipeToRefresh* que de uma forma muito resumida, volta a chamar os métodos *getLocation* e *parseJson*.

1. *Manifest.xml*

No ***manifest.xml*** foram requisitadas as permissões que permitissem a funcionabilidade da aplicação como o acesso à internet ou ao GPS como o código abaixo indicado indica.

<uses-permission

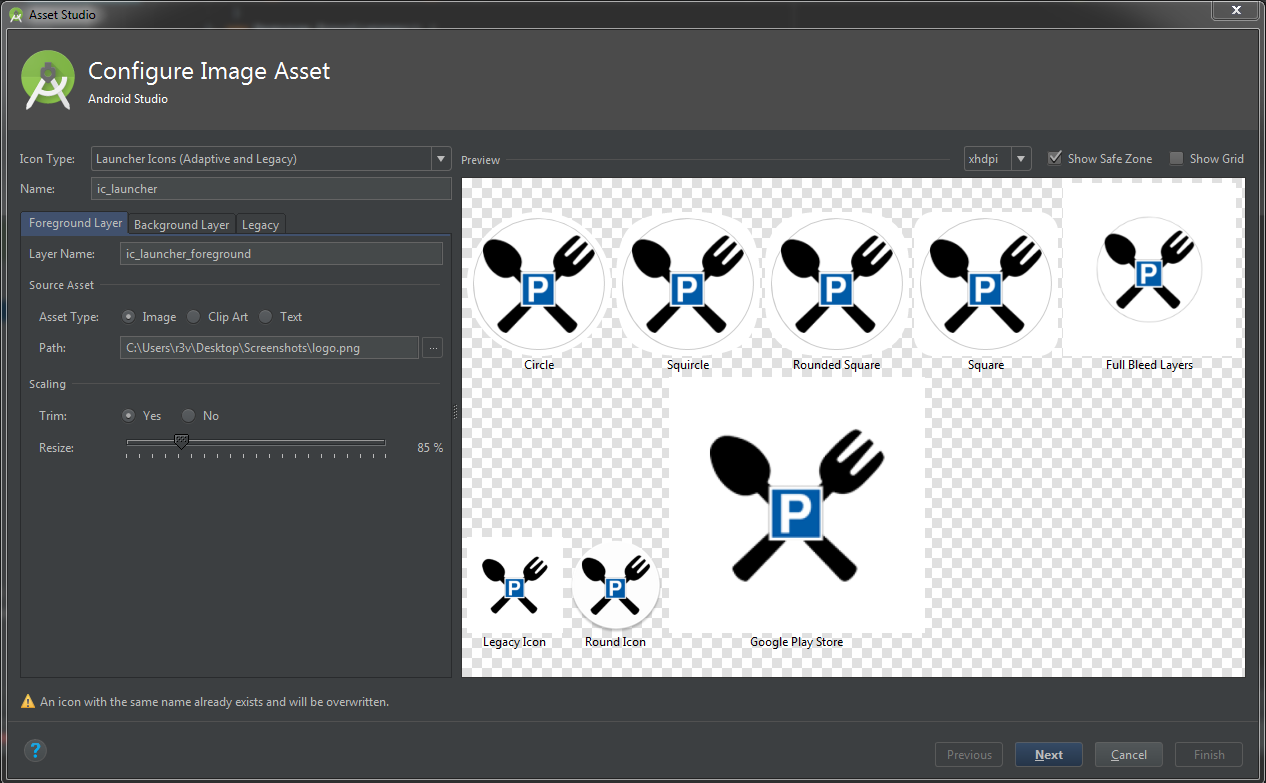
android:name="android.permission.INTERNET" />  
<uses-permission android:name="android.permission.ACCESS\_FINE\_LOCATION" />

De referir ainda que, de forma a otimizar a compatibilidade com tablets, a orientação do ecrã foi bloqueada através da seguinte linha de código.

android:screenOrientation="sensorPortrait"

Ao contrário do *portrait* de anteriores API’s, este permite a visualização em modo *landscape* com barras pretas nas laterais ao invés de bloquear a rotação de ecrã na totalidade.

1. Ícone da aplicação



O ícone da aplicação foi personalizado utilizando a ferramenta *Asset Studio*, incluída no *Android Studio*. De notar o detalhe na compatibilidade com API’s mais antigas.

1. Considerações finais

Por uma questão de performance, todas as imagens foram tratadas usando o website <https://tinypng.com/> que permite reduzir o tamanho das imagens, reduzindo o respetivo tempo de carregamento da aplicação.

Foi também efetuado o *Lint* (*Optimize Code*) do *Android Studio* efetuando as sugestões do mesmo e mantendo a compatibilidade com API’s anteriores usando o abaixo indicado.

tools:targetApi="26"

De referir também o cuidado na extração de todas as *Strings* para ficheiro strings.xml, de forma a facilitar futuras traduções para outros idiomas.

* 1. A melhorar/Implementar:
* Trabalhar com *fragments* de forma a implementar sistemas mais complexos como *Tabs* com *swipe*, que seria bastante útil para apresentar as ementar de forma semanal ao invés de diário
* Converter os *layouts* para *CoordinatorLayouts* para otimizar a performance da aplicação
* Um menu de definições que permitisse um controlo mais refinado da aplicação como por exemplo o suporte a várias línguas, um switch para ligar e desligar o arranque automático, personalização da hora de apresentação das ementas, entre outros.
* Performance relacionada com o *clearcache* (tive de recorrer a este método porque o *Json* não voltava a ser pedido o que fazia com que as ementas não fossem atualizadas para o dia atual)
* *Spinners* e/ou *progressBars* como forma visual de apresentar os tempos de carregamento
* Otimizar compatibilidade para tablet através da criação de layouts específicos
* Implementar outros serviços da UA como as senhas dos Serviços Académicos, Biblioteca, entre outros)
* Criar um tutorial de arranque a descrever a funcionalidade da aplicação e respetivas instruções para a configuração e utilização de aplicação
* Publicar a aplicação na Google PlayStore
* Implementar o sistema de localização de forma alternativa ou corrigir o atual

(<https://google-developer-training.gitbooks.io/android-developer-advanced-course-practicals/unit-4-add-geo-features-to-your-apps/lesson-7-location/7-1-p-use-the-device-location/7-1-p-use-the-device-location.html>