Instituto Superior de Engenharia de Lisboa

Projeto Final de Curso

Elderoid

Interface Android adaptada a indivíduos de 3ª Idade

Relatório Final

Licenciatura em Engenharia Eletrónica e Telecomunicações e de Computadores

Pedro Gonçalves, 45890

Orientador: Professor Vitor Vaz da Silva
do Departamento de Engenharia Eletrónica e Telecomunicações e de Computadores

Presidente do Júri: Professor Jorge Martins

Arguente: Professora Paula Graça

6º Semestre, 2020/2021

Agradecimentos

Ao orientador Vítor Vaz da Silva, pela disponibilidade, auxílio e dicas.

Ao meu primo David Gonçalves, pela conceção do Nétie.

A todos os que, de alguma forma, contribuíram com sugestões e para o enriquecimento do projeto.

Resumo

Observa-se no mundo real que, em muitos casos, os idosos assutam-se com a complexidade de um simples **smartphone**[1], e deixam de lado um aparelho que lhes pode ser muito útil. Este projeto procurará disponibilizar aos indivíduos de 3ª idade uma forma (acessível por qualquer um) de usufruirem de um dispositivo Android conforme as suas necessidades. Se a única necessidade for realizar chamadas telefónicas, o dispositivo não incomodará o utilizador com a meteorologia, disponibilizando apenas a funcionalidade pretendida. Será desenvolvido um **software** para o Sistema **Android** com uma interface[2] adequada e flexível nesse sentido.

Palavras Chave: Android Launcher, Aplicativo para Idosos, Chamadas, Elderoid, Interativo, Interface Adaptada, Meteorologia, Nétie, Telemóvel, Tablet.

[1] Traduz-se para "telefone inteligente". Telemóvel com diversas funcionalidades.

[2] Aspeto visual do produto. Componentes do ecrã com que o utilizador interagirá, por exemplo botões.

Abstract

In the real world, older individuals often get scared of the complexity a single **smartphone** holds, leaving aside a device that can be of many uses, even to them. This project seeks to offer the elderly an accessible way to take advantage of these **Android** devices, according to their needs. If the user's only need is to make phone calls, for example, the smartphone will not bother with other functionalities, nor they will even be available, in the first place. The object is to develop an **Android Software** that is flexible in that sense and has an adequate UI[3].

Key Words: Adapted Interface, Android Launcher, App for the Elderly, Calls, Elderoid, Interactive, Nétie, Smartphone, Tablet, Weather.

[3] What the user sees on screen, and every visual component he may interact with.

Índice

Lista de Figuras

Lista de Tabelas

Lista de Programas Utilizados

Android Studio [https://developer.android.com/studio]

Postman [https://www.postman.com/]

Adobe Ilustrator [https://www.adobe.com/pt/]

Desmos [https://www.desmos.com/calculator]

Dia [http://dia-installer.de/index.html.en]

1. Introdução

A tecnologia tem avançado, em vários campos, ao longo das últimas décadas. No campo da comunicação, o progresso tem sido igualmente imensurável. Hoje em dia, andamos com um autêntico computador no bolso, capaz de navegar na internet, reproduzir música e outras formas de multimédia, tirar fotos e gravar vídeos de grande qualidade, obter coordenadas geográficas e muitas outras funcionalidades.

No entanto, nem todos os indivíduos conseguem acompanhar esta evolução de forma tão espontânea, e para quem comunicar por telefone é absolutamente imprescindível. Indivíduos que utilizaram telefones de marcação de disco durante vários anos, e que até já possam ter utilizado telemóveis com teclas, têm muitas vezes dificuldades em lidar com as múltiplas funcionalidades que um simples smartphone oferece.

O objetivo deste projeto é tornar a vida dessas pessoas mais fácil, introduzindo-lhes uma interface mais adaptada à sua forma de compreensão, para que possam utilizar dispositivos Android de forma mais inconsciente, e apenas para o que necessitam.

O aspeto geral do aplicativo é apresentado na **figura 1**.

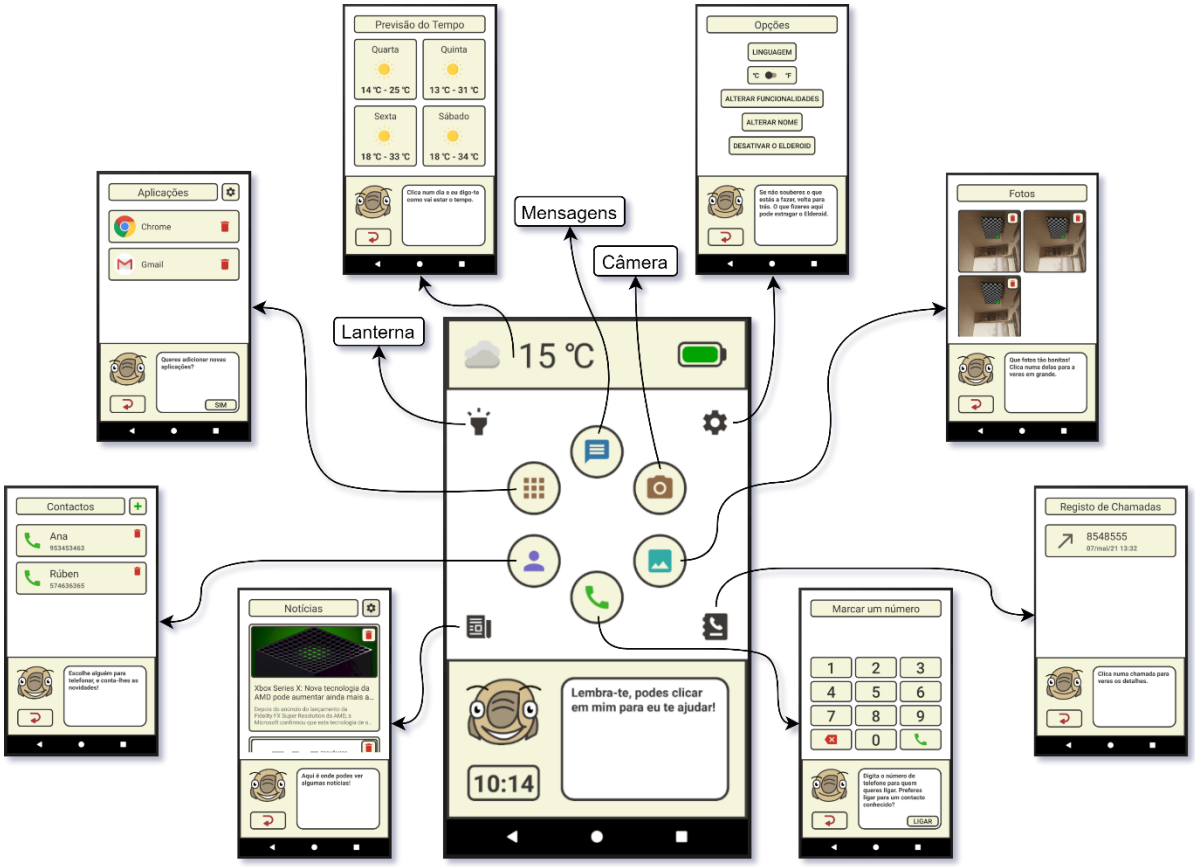

Figura 1 - Aspeto Geral do aplicativo e suas funcionalidades

A **figura 1** ilustra aquilo que será denominado a **fase de utilização**, posterior a uma **fase de configuração**. A **fase de configuração** (abordada com frequência ao longo deste documento) será algo por que o utilizador terá de passar apenas uma vez, de forma a configurar o aplicativo conforme as suas preferências. Esta fase será também utilizada para configurar outros aspetos importantes para o seu bom funcionamento. Todos estes conceitos serão devidamente explicados e discutidos, em capítulos subsequentes.

2. Projeto

O software irá diferir de uma aplicação comum no sentido em que, depois de obter todas as permissões necessárias do utilizador, apenas será visível a sua interface, aquando da utilização do smartphone. Chamar-se-á **Elderoid.**

Assim sendo, surgem imediatamente três conceitos que devem ser abordados o quanto antes:

- **Abstração:** O problema da abstração deriva precisamente da necessidade de manter o utilizador no aplicativo. Isto é, quando o dispositivo for utilizado, seja para o que for, o sistema operativo deve redirecionar o utilizador diretamente para a interface a desenvolver neste projeto. Existem algumas formas de atingir este objetivo, como será discutido no capítulo **2.1**;
- **Ambiente de Desenvolvimento:** O Elderoid terá de ser projetado com recurso a alguma plataforma, de forma a facilitar o seu desenvolvimento. A plataforma de desenvolvimento escolhida, irá depender diretamente do método de **Abstração** concluído no capítulo **2.1**, e será brevemente abordada no capítulo **2.2**.
- **Interface:** A aparência do aplicativo e a forma como as funcionalidades estarão organizadas serão o aspeto mais importante do projeto, visto que será o que estará sempre em contato com o utilizador. Este conceito será abordado constantemente, ao longo do documento. No entanto, alguns pontos de partida serão estabelecidos no capítulo **2.3**.

Depois de estabelecer estes três pontos de partida, avançar-se-á para questões mais detalhadas, como os menus associados a cada funcionalidade e a forma como se implementou cada uma dessas funcionalidades. Abordar-se-á também a questão da compatibilidade entre dispositivos (nomeadamente ***smartphones*** e ***tablets***), muito importante para este projeto, visto que o objetivo é disponibilizar esta interface a todos os que precisem, qualquer que seja o dispositivo **Android** a utilizar.

2.1 Abstração

A abstração será a forma como o aplicativo se tornará independente do sistema **Android** a que os utilizadores frequentes estão habituados. Ou seja, é necessário arranjar uma forma de manter o aplicativo ativo, mesmo quando o dispositivo é utilizado depois de ser encerrado ou suspendido. O objetivo é que quando o utilizador fizer uso do dispositivo com o aplicativo instalado, apenas verá a interface desenvolvida.

Ora, de forma a atingir esse objetivo, pensou-se em 3 diferentes opções:

- Criar uma **App** convencional, mas com algumas características que a tornem **uncloseable**[4], isto é, inibir as várias formas possíveis de encerrar a aplicação, e iniciá-la sempre que o dispositivo for utilizado;
- Desenvolver um "**Android Launcher**" capaz de reorganizar a interface inicial do dispositivo;
- Reaproveitar o código fonte do Android para reprojetar um Sistema Operativo simples, apenas com as funcionalidades desejadas. Aquilo a que se chama uma "**ROM Personalizada**".

[4] Traduz-se para "algo que não pode ser fechado".

2.1.1 Uncloseable App

A primeira consideração terá sido desenvolver um aplicativo comum, mas com algumas características especiais, que não permitam que o utilizador o abandone. Contudo, por diversos motivos de segurança, o **SDK**[5] **Android** não fornece formas de tornar um aplicativo "**uncloseable**", visto que isso abriria possibilidades a aplicativos mal intencionados e a **malware**[6].

Contudo, com alguma pesquisa, conclui-se que existem formas de contornar algumas dessas limitações impostas pelo **Android**. Por exemplo, iniciando o aplicativo com alguns parâmetros específicos, declarados no ficheiro **AndroidManifest.xml**[7], é possível ocultar a barra superior de notificações (**figura 2**), como descreve o Guia de Desenvolvimento [1].

Figura 2 - Barra de Notificações (Status Bar)

Com a ajuda de alguma pesquisa, arranjar-se-iam soluções para manter o utilizador no aplicativo, quer o dispositivo seja suspendido, encerrado ou o botão **Home** (botão central que leva o utilizador à página inicial do ***smartphone*** ou ***tablet***) seja pressionado. No entanto, esta opção não será a mais viável, visto que não é eticamente correto forçar o utilizador a fazer algo ou a ficar numa certa janela.

O ideal seria criar um aplicativo que solicita, inicialmente, o consentimento do utilizador para modificar a janela inicial do Android. Como veremos no sub-capítulo **2.1.2**, o **Android** permite que uma aplicação seja desenvolvida nesse âmbito.

[5] **Software Development Kit**, traduz-se para Kit de Desenvolvimento de Software

[6] **Software** malicioso. Programa ou código prejudicial para o sistema.

[7] Ficheiro **XML** fundamental para qualquer aplicativo **Android**, que descreve informações essenciais sobre o mesmo.

2.1.2 Android Launcher

Um **Android Launcher** é semelhante a um aplicativo, no sentido em que será instalado a partir da **App Store**, e não fará nenhumas alterações permanentes. Quando o aplicativo for desinstalado, todas as alterações serão revertidas. Contudo, difere de um aplicativo convencional, visto que o intuito é alterar o modo como as funcionalidades já existentes são disponibilizadas e organizadas, e não acrescentar funcionalidades.

Como pode ser observado na **figura 3**, um **Android Launcher** reorganiza a página inicial do dispositivo, e permite que o utilizador apenas veja aquilo que necessita, quando fizer uso do mesmo. É exatamente isso que se pretende com este projeto, e por isso, esta será a opção mais viável.

Figura 3 - Exemplo de aspeto de uma página inicial criada pelo Android Launcher "Themer"

Outra grande vantagem desta solução é a sua simpática implementação. Segundo Adam Sinicki, da **Android Authority** [2], tudo o que é necessário acrescentar a um aplicativo para que funcione como **launcher**, são duas simples linhas, ao ficheiro **AndroidManifest.xml**:

```
<category android:name="android.intent.category.HOME" />
<category android:name="android.intent.category.DEFAULT" />
```

Estas duas linhas farão com que o aplicativo seja reconhecido como **launcher**, pelo dispositivo Android. Será necessário, na **fase de configuração** (capítulo **2.5**), redirecionar o utilizador para as definições, de forma a selecionar o **Android Launcher** padrão do seu dispositivo, visto que não há forma de o fazer programaticamente. Assim, garante-se que o utilizador tem perfeita consciência do que está a fazer, e que a aplicação não se "apodera" do dispositivo sem que ele tenha conhecimento. Relembra-se que tudo isto será feita na **fase de configuração**, aconselhavelmente por alguém com mais experiência, antes de se passar à **fase de utilização**.

2.1.3 ROM Personalizada

Uma das características do Sistema Operativo **Android**, é que o seu código fonte está aberto ao público, incluindo *kernel*, *UI* e bibliotecas. Isto significa que qualquer desenvolvedor pode criar a sua própria versão do Android, sendo que várias empresas ou até desenvolvedores individuais até distribuem as suas próprias versões aos utilizadores. Estas versões designam-se "**Custom ROMs**" ou "**Custom Firmware**".

Esta solução difere da anterior (**Android Launcher**) porque permite muito mais do que, em princípio, será necessário. O objetivo do projeto é projetar uma interface, e não um sistema operativo.

Como é descrito pela página disponibilizada por Gary Sims, da Android Authority [3], existem alguns pré-requisitos, entre os quais, a posse de um sistema **Linux** ou **Mac**. Em princípio, não haveria grande problema, sendo que na ausência de ambos, poder-se-ia utilizar uma máquina virtual. Contudo, Gary Sims descreve a sua experiência ao utilizar máquinas virtuais, e afirma que se deparou com imensos problemas, dos quais alguns insuperáveis.

Além do problema dos pré-requisitos, esta solução com base em "**Custom ROMs**", será muito mais consumidora de tempo, visto que exige experiência com terminais Linux e Sistemas Operativos. Exige também grande paciência e uma boa máquina, visto que para testar uma mínima alteração, é necessário fazer "build" de todo o sistema, o que pode demorar até 20 minutos, dependendo do hardware. Gary Sims aconselha **130 GB** de disco livres e mais de **8 GB** de **RAM**.

Para piorar, o acesso dos utilizadores a este tipo de **software** seria mais complicado, visto que não se trata apenas de um aplicativo transferível na **App Store**.

Em suma, uma **ROM Personalizada** apresenta imensas vantagens e uma grande flexibilidade. Contudo, essa opção não é a mais viável para este projeto, visto que os pré-requesitos são muito exigentes, o processo de instalação será longo e estonteante, e as coisas podem tornar-se bastante complexas.

2.1.4 Conclusão sobre o Ambiente de Desenvolvimento e Método de Abstração

A solução que o **Elderoid** adotará será a implementação de um **Android Launcher,** como descrito no sub-capítulo **2.1.2**, com recurso ao **IDE**[8] **Android Studio** [4], e utilizando a linguagem de programação **Java**. Ao longo deste documento, o Elderoid será referido como aplicativo, apesar de ser mais do que isso. Contudo, como já vimos, um **Launcher** não difere assim tanto de um aplicativo comum.

[8] **Integrated Development Environment** (Ambiente de Desenvolvimento Integrado)

2.2 Introdução ao Ambiente de Desenvolvimento

Como referido anteriormente, utilizar-se-á o **Android Studio** como plataforma de desenvolvimento do aplicativo. Contudo, antes de passar a questões mais relevantes, como a **Interface**, será boa ideia estabelecer alguns pontos de partida quanto ao ambiente em que o Elderoid será desenvolvido. Aspetos como:

- **Organização do projeto:** Esta questão está diretamente relacionada com o **IDE** que se utilizará. O **Android Studio** organiza os conteúdos associados ao desenvolvimento de uma aplicação de uma certa e determinada maneira, a qual será abordada brevemente no sub-capítulo **2.2.1**.
- **Introdução ao Desenvolvimento Android:** É importante adquirir algumas noções no âmbito do Desenvolvimento de uma aplicação **Android**, imprescindíveis para o desenvolvimento deste projeto. Estas noções serão introduzidas no sub-capítulo **2.2.2**.
- **Organização de dados:** O sub-capítulo **2.2.3**, abordará a forma como dados importantes serão guardados e organizados no dispositivo do utilizador, de forma a não serem eliminados, mesmo quando o aplicativo fechar.

2.2.1 Organização do Projeto

A diretoria de um projeto de uma aplicação Android pode tornar-se confusa. Contudo, há alguns locais onde será passada a maior parte do tempo, e é muito importante perceber o seu propósito. No caminho **/app/src/main** (**figura 4**) encontram-se três ficheiros/pastas fundamentais para qualquer aplicação, e contêm grande parte da informação mais importante para o funcionamento e exclusividade da mesma:

Figura 4 - Pasta /main no explorador do Windows 10

- **/java:** Esta pasta contará com todos os ficheiros **.java** associados ao Elderoid. Toda a lógica por trás do aplicativo será encontrada neste local.
- **/res:** Todos os recursos, quer sejam imagens, botões, cores, frases, estilos, serão armazenados nesta pasta.
- **/AndroidManifest.xml:** Como já foi abordado anteriormente, este ficheiro XML conta com vários elementos descritivos da aplicação, pelo que é imprescindível.

2.2.2 Introdução ao Desenvolvimento Android

Ao longo deste documento, há duas palavras que serão usadas recorrentemente – **activity** e **layout**. Estes dois conceitos andam frequentemente de mão dada, visto que cada **activity** apresenta **layout**.

Mas o que é uma **activity**? Uma **activity**, no ambiente **Android**, é algo que o utilizador pode fazer. Em geral, uma **activity** interage com o utilizador, e apresenta uma componente visual (nomalmente um ficheiro **.xml**) responsável pela forma como as coisas são organizadas na janela e apresentadas ao utilizador (designa-se o **layout** da **activity**), e uma componente lógica (ficheiro **.java**, para a linguagem **Java** ou ficheiro **.kt**, para a linguagem **Kotlin**[9]) responsável pelas funcionalidades e código associado à **activity**. Uma **activity** é a complementação destas duas componentes.

Assim, todas as componentes lógicas de todas as **activities** poderão ser encontradas na pasta **/java**, enquanto todas as componentes visuais (**layouts**) estarão contidas na pasta **/res/layout**. Com recurso a bibliotecas fornecidas pelo **Android**, é possível referenciar as componentes visuais programaticamente, a partir da linguagem **Java**, interligando, mais uma vez, estas duas componentes.

Outro conceito importante acerca de uma **activity**, é o seu ciclo de vida. Como é referido no Guia de Desenvolvimento Android [5] acerca do ciclo de vida de uma **activity**, à medida que o utilizador navega pela aplicação, as **activities** transitam entre diferentes estados do seu ciclo de vida. A classe **Activity** fornece um conjunto de **callbacks**[10] associados a cada um desses estados, os quais estão representados no diagrama da **figura 5**.

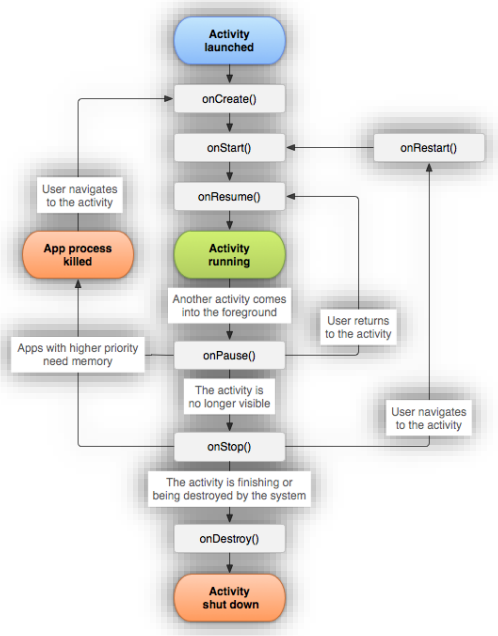

Figura 5 - Ciclo de vida de uma atividade (Retirado do Guia de Desenvolvimento Android)

Outro conceito que será referido múltiplas vezes neste documento, é a ideia de **Intent** [6]. No ambiente Android, um **Intent** é, sobretudo, um objeto que serve para comunicar entre componentes, como **activities**. Apresenta três aplicações fundamentais - iniciar **activities**, iniciar serviços[11] e enviar/receber transmissões[12];

Por fim, o layout de cada **activity** será muito importante para este projeto, e tentar-se-á que sejam compatíveis com o máximo de ***smartphones*** e ***tablets*** possível. Para isso utilizar-se à uma biblioteca de organização visual (fornecida pelo **Android**) chamada **ConstraintLayout** [7]. Com o auxílio desta biblioteca, todos os componentes visuais serão posicionados e dimensionados de forma relativa, sem nunca se utilizarem valores absolutos. Existem várias bibliotecas de organização visual em Android, mas concluiu-se que, para este projeto, esta será a mais adequada, dada a sua flexibilidade e o nível de compatibilidade que atribuirá ao aplicativo.

[9] Linguagem de programação que combina características de programação tanto funcional, como orientada a objetos.
[10] Função que é definida previamente, e é suposto ser executada após um determinado evento.
[11] Componente do aplicativo que realiza operações em segundo plano.
[12] Mensagem que qualquer aplicativo pode receber. Por exemplo, o sistema fornece várias transmissões relativas a eventos do próprio.

2.2.3 Organização dos Dados

A capacidade de armazenar informação no dispositivo do utilizador, será uma competência do Elderoid muito importante. Esta necessidade deriva do conceito de ciclo de vida de uma **activity**, abordado no sub-capítulo anterior, visto que todas as **activities**, eventualmente, serão suspendidas ou encerradas, perdendo-se toda a informação. Alguma desta informação pode ser relevante para o funcionamento do aplicativo, no sentido em que outras **activities** poderão precisar dela. Será fundamental armazenar dados persistentes no dispositivo, associados ao Elderoid.

Existem algumas forma de atingir esse objetivo. A solução a utilizar neste projeto, tira partido das **Preferências Partilhadas** (Shared Preferences) [8]. Permite que uma aplicação armazene dados que persistem, mesmo quando a aplicação é encerrada, no formato **chave-valor**[13].

Para facilitar o acesso às Preferências Partilhadas, utilizar-se-á uma biblioteca, adicionando uma camada de abstração entre o aplicativo e as preferências. Desenvolvida por *Farruhha Bibullaev*, **SimplePrefs** [9] é uma **API**[14] que facilita e melhora a gestão de dados nas preferências. Tem uma característica estática, pelo que basta inicializar, e passa a poder ser utilizada e acedida pelas várias **activities** ao longo do seu ciclo de vida.

Os dados armazenados nas preferências serão perdidos quando o utilizador desinstalar a aplicação, ou limpar os dados associados à mesma, nas definições do Android.

[13] A cada chave está associado um valor, seja de que tipo for. Estrutura semelhante a um **Mapa**, em programação.
[14] Application Programming Interface – Software Intermediário de comunicação com uma certa funcionalidade ou aplicação.

2.3 Estudo da Interface

A questão da interface é um pouco mais delicada, visto que será aquilo que estará em constante contacto com o utilizador.

Desde já, a interface a desenvolver terá de cumprir alguns critérios:

- **Sobriedade:** De forma a poder ser utilizado por indivíduos pouco familiarizados com este tipo de tecnologia, o aplicativo terá de apresentar uma interface espontânea e sóbria, de forma a que os utilizadores se mantenham conscientes do que fazem ao longo da sua utilização;
- **Flexibilidade:** Como já foi referido, um dos objetivos deste projeto é, numa fase de configuração, prévia à utilização casual do dispositivo, escolher-se quais as funcionalidades a serem disponibilizadas. Isto significa que, dependendo das funcionalidades que forem selecionadas nessa fase, o aspeto da tela inicial terá de se reorganizar automaticamente;
- **Contraste:** Todos os pontos de interesse na tela terão de ser claramente identificados, facilitando a navegação a indíduos com dificuldades visuais. Por exemplo, botões, ícons e caracteres terão de apresentar um tamanho razoavelmente grande e as cores terão de ser minimamente contrastantes. Ter-se-ão de utilizar todos os recursos possíveis para manter o utilizador consciente daquilo que está a fazer, e de onde se encontra.

Nos próximos sub-capítulos serão abordados alguns exemplos de aplicativos que também apresentam interfaces facilitadas, cujo público alvo representa a 3ª Idade, e destacar-se-ão as suas vantagens e desvantagens, concluindo-se sobre alguns aspetos a ter em conta no projeto.

2.3.1 Doro

Os dispositivos *smartphone* produzidos e distribuídos pela empresa **Doro** [10] são um belíssimo exemplo de uma interface adapatada para idosos. Apresenta algumas opções, as quais a Doro optou por organizar em "**Chamar**", "**Ver**" e "**Enviar**", como pode ser observado na página inicial do *smartphone* da **figura 6**. A secção "**Ver**" inclui navegar pela galeria de fotos, ver as mensagens recebidas e ver publicações em redes sociais, por exemplo. A secção "**Enviar**", por sua vez, representa o contrário, como enviar mensagens ou enviar um foto para as rede sociais.

Trata-se de uma organização interessante e eficaz, visto que o utilizador sabe sempre para onde ir e não tem muito por onde escolher. Claro que o botão "**Enviar**", por exemplo, levaria a um novo menu em que o utilizador selecionaria que tipo de envio quer realizar. Note-se também os contactos mais importantes disponibilizados acima dos botões principais.

Por outro lado, a parte superior da tela assemelha-se bastante a um dispositivo Android convencional, apresentando a barra de notificações, hora e uma barra de pesquisa no **Google**.

Figura 6 - Smartphone Doro

A **Doro** tem mão não só no *software*, como também no *hardware* embutido no *smartphone*. Como é possível verificar na **figura 7**, neste modelo existem três botões físicos na parte inferior do *smartphone*, e possuem as mesmas funções a que estamos acostumados em qualquer dispositivo **Android**. Contudo, nenhum deles é em *software*, ao contrário da maioria dos dispositivos mais recentes. A barra de navegação[15] pode gerar muita confusão, visto que tem tendência a desaparecer e a reaparecer com alguma frequência, sendo essa a razão que levou a **Doro** a tomar esta decisão.

Figura 7 - Barra de navegação Android

2.3.2 Big Launcher

O aplicativo **Big Launcher** (disponível na **App Store**) também apresenta uma interface muito simples e intuitiva, repleta de cores contrastantes e ícons grandes, o que beneficia a navegação para indivíduos com dificuldades visuais.

Como descrevem os desenvolvedores [11], o aplicativo apresenta diversas funcionalidades e possui alguns aspetos configuráveis, como por exemplo, o tamanho dos caracteres e símbolos.

Além dos 6 botões principais expostos em grelha, a página inicial do aplicativo Big Launcher apresenta uma secção que disponibiliza algumas informações (data, bateria, hora e qualidade da conexão), como pode ser verificado na **figura 8**. Mais à direita, ainda nessa secção, situa-se um pequeno botão com o símbolo ⋮ que leva a um menu de preferências. Nesse menu é possível acede às definições do dispositivo, de forma a reconfigurar o **Launcher Android** padrão, caso se queira desinstalar o aplicativo.

Figura 8 - Página inicial do Big Launcher

[15] Barra que contém o botão de "Retroceder", "Página Inicial" e "Gestão de Aplicativos", em software.

2.3.3 Análise e Conclusões acerca da Interface

Relativamente à interface desenvolvida pela **Doro** (apresentada em **2.2.1**), existem alguns aspetos negativos que terão de ser tidos em conta no desenvolvimento de **Elderoid**. Por exemplo, a disponibilização da barra de pesquisa no **Google**. O **Google** é uma ferramenta poderosa com grande alcance e complexidade, podendo por isso, gerar grande confusão e até algum perigo. Contudo, a maior desvantagem da interface **Doro** é que não se encontra disponível na **App Store** para utilizar em qualquer dispositivo.

Por outro lado, o **Big Launcher** encontra-se disponível para instalar em qualquer dispositivo. No entanto, apesar de apresentar alguns aspetos configuráveis, o aplicativo não apresenta grande flexibilidade a nível de escolha de funcionalidades a disponibilizar. Como se verifica na **figura 8**, a página inicial apresenta sempre os mesmos 6 botões, quer o utilizador queira, quer não.

O objetivo deste projeto é colocar ao alcance de todos os utilizadores não familiarizados com o **Android**, uma interface flexível ao ponto de disponibilizar apenas aquilo que for solicitado numa fase de configuração prévia.

A respeito da interface, há alguns objetivos que se tencionam cumprir, se o tempo assim permitir:

- **Menus:** Implementação dos **Menus** que serão apresentados ao utilizador, isto é, zonas da navegação no dispositivo associadas a uma ou mais funcionalidades.
- **Feedback[16] em Tempo Real:** Implementação de uma forma de informar o utilizador acerca das suas ações, e acerca dos locais onde se encontra. Para isso, pensou-se em utilizar um personagem interativa que fará companhia ao utilizador durante a navegação (**figura 9**). A expressão facial do personagem e os balões de fala que lhe estaram atribuídos ajudarão o utilizador a ter sempre uma noção do sítio onde se encontra e das suas ações.
- **Tutoriais:** Implementação de um tutorial para todos os **Menus**. Em cada menu de navegação, pretende-se adicionar um botão que inicia um toturial, com o intuito de explicar ao utilizador os diferentes pontos de interesse que se encontrem na tela, destacando-os e descrevendo-os.

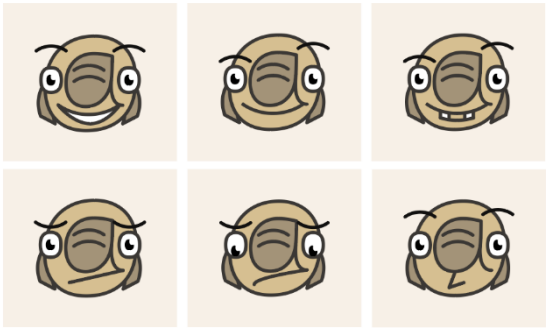

Figura 9 - Personagem de companhia (Nétie)

O personagem ilustrado na **figura 9** é original de **David Miguel Gonçalves**, licenciado em Design Gráfico, cujo projeto [12] de Gestão de Projeto na Escola Superior de Artes e Design (Caldas da Rainha) consistiu em imaginar uma aplicação adaptada a idosos. O projeto não envolveu qualquer componente de ***software***. Limitou-se a uma organização visual e desenho de vários componentes e funcionalidades de um ***smartphone*** ou ***tablet***, considerando que seria um indivíduo de 3ª idade a utilizá-lo. Consistiu essencialmente na sua conceção, gestão e estudo do possível impacto no mundo real. Contudo, a pesquisa associada a esse projeto revelou-se extremamente interessante, pelo que alguns conceitos serão considerados para este projeto, sobretudo a mascote que está sempre presente no ecrã. Essa ideia é especialmente interessante. Com autorização do autor, o personagem será a mascote do Elderoid, e chamar-se-á Nétie.

[16] Avaliar as ações do utilizador, e informá-lo com o que for necessário.

2.4 Feedback em Tempo Real

Visto que o feeback em tempo real é um dos aspetos mais interessantes deste projeto, e sendo essa uma característica que estará presente em todos os menus do aplicativo (salvo algumas excepções), começou-se por desenhar essa componente. Tratar-se-á de uma janela que ocupa cerca de um terço do ecrã, na parte inferior. Alojará o personagem *Nétie*, bem como o seu balão de fala. No caso da **figura 10**, verifica-se que o espaço restante foi atribuído a um botão de **Retroceder**.

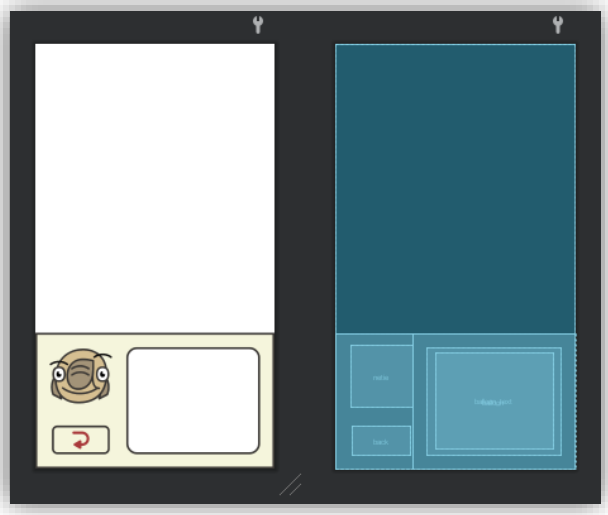

Figura 10 - Janela de Feedback

A forma como as componentes serão organizadas nesta aplicação, será muito importante. Pretende-se obter compatibilidade com o máximo de dispositivos possível, **telemóveis** e ***tablets***. Posto isto, é estritamente proibido atribuir dimensões fixas às componentes de uma janela.

No caso da **figura 10**, a janela principal (a bege), está dividida em duas sub-janelas, na horizontal. A primeira contém o *Nétie* e o botão de **Retroceder**, e ocupa **32%** da janela principal, independentemente do tamanho do ecrã. Assim, tanto a imagem do personagem, como o botão, jogarão com o espaço disponível, e adaptar-se-ão consoante o dispositivo. Da mesma forma, a segunda sub-janela (associada ao balão de fala) estará referenciada ao final da primeira, e jogará com os **68%** do espaço restante. Estas percentagens são conseguidas em XML, através de um atributo disponibilizado pela biblioteca ConstraintLayout, e que será abordado em capítulos subsequentes.

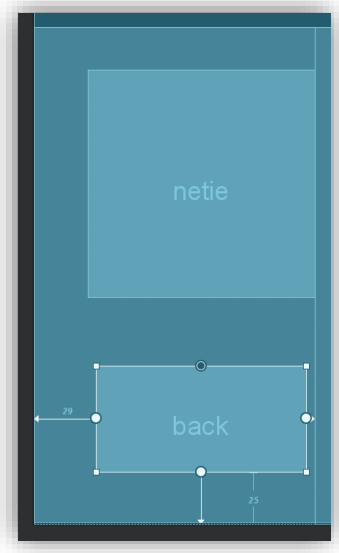

Figura 11 - Aproximação à primeira sub-janela

Contudo, como pode ser observado na **figura 11**, nem todos os aspetos do dimensionamento dos componentes visuais são altamente dinâmicos. O botão de Retroceder, por exemplo, apresenta uma margem em relação ao início do ecrã. Por enquanto, essa margem apresenta um valor fixo (29 dp[17]), e pode representar uma dimensão diferente, de dispositivo para dispositivo. Este assunto será abordado com mais detalhe no capítulo de **Compatibilidade** (capítulo **2.8**).

Ainda no contexto do feedback, desenharam-se mais duas janelas, ligeiramente diferentes da janela apresentada na **figura 10**, mas com dimensões semelhantes. Na **figura 12**, apresenta-se uma janela de feedback com um relógio digital, em vez de um botão de **Retroceder**, e a **figura 13** apresenta uma janela de feedback sem mais nenhum componente, apenas o **Nétie** e o seu balão de fala.

[17] **Density-independent Pixel**, isto é, Pixel independente da Densidade (este conceito será abordado no capítulo de **Compatibilidade**)

Estas 3 janelas foram projetadas de forma a poderem ser incluídas no **layout** das diversas **activities** do aplicativo. Ou seja, todos os componentes visuais associados à **activity** em questão, aparecerão acima da janela que contém o *Nétie*, no restante espaço em branco.

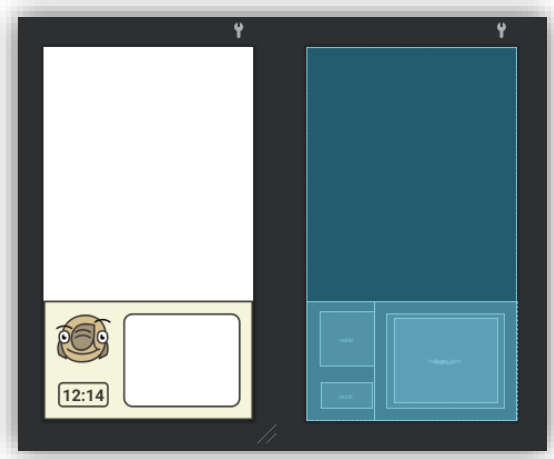

Figura 12 - Janela de feedback com relógio digital

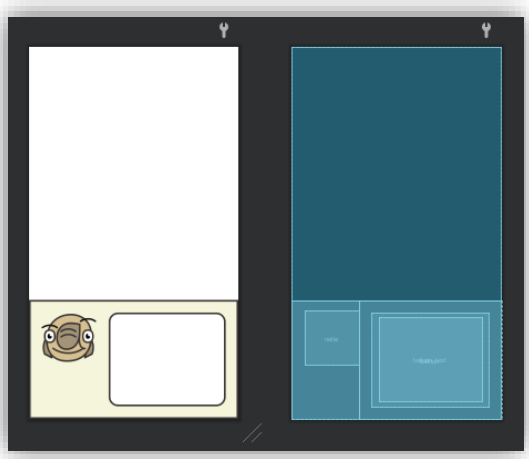

Figura 13 - Janela de feedback vazia

Para implementar um relógio digital em tempo-real, utilizou-se uma classe que extende **TextView**[18], implementada por **Jossy Paul**, e pode ser utilizada como qualquer outro componente na tela. O nome da classe é **DigitalClock** [13].

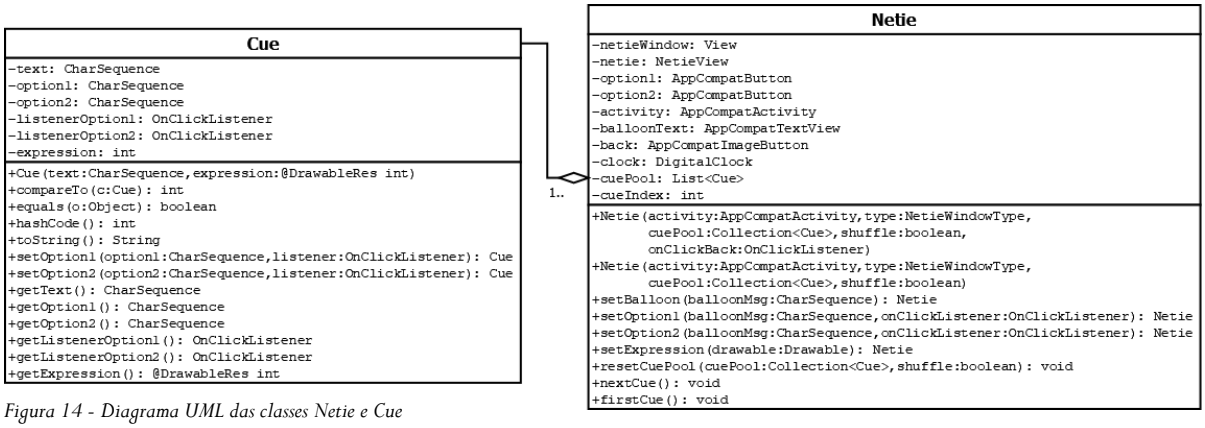

Figura 14 - Diagrama UML das classes Netie e Cue

Do ponto de vista programático, elaborou-se a classe descrita na **figura 14**. A classe **Netie** será útil sempre que for necessário atribuir uma nova expressão facial, balão de fala ou opções de diálogo, em qualquer **activity**. Basta, na criação de cada atividade, instanciar um **Netie**, e quando conveniente, invocar as funções disponibilizadas pela classe sob o objeto instanciado.

Na **figura 14** está também representada a classe **Cue**. Uma **Cue** representa algo que o **Nétie** poderá dizer ou aconselhar (mensagem no balão de fala), e pode estar acompanhado por uma expressão facial e/ou opções de dialógo. Cada instância do nétie apresenta um conjunto de **Cues** disponíveis. Isso significa que, numa determinada **activity** (i.e. num determinado menu/janela), o Nétie possui um leque de **Cues** que pode manifestar ao utilizador. Note-se que é obrigatório que cada instância de **Netie** possua pelo menos uma **Cue** na chamada ***cuePool*** (conjunto de **Cues**), visto que o balão de fala nunca poderá estar vazio.

[18] Componente que representa uma caixa de texto em **Android**.

Implementou-se também uma **View**[19] personalizada o **Nétie** (apenas a cara). Esta **View** (denominada **NetieView**) permite que, sempre que se clica na mascote, seja dispertada uma animação (serve essencialmente para informar o utilizador de que o clique foi bem-sucedido). Esta animação foi implementada em vários locais do Elderoid, principalmente nos elementos clicáveis que não levam a outra janela (por exemplo, o relógio digital).

Todo este procedimento, tanto programático como de organização visual, tornará o desenvolvimento de cada **activity** do Elderoid mais abstraído e independente da janela do **Nétie**.

[19] Componente visual em **Android** (pode ser uma caixa de texto, uma imagem, um botão ou qualquer outra coisa que apareça na tela).

2.5 Fase de Configuração

Dependendo dos objetivos do utilizador para com o aplicativo pretende-se que este passe primeiro por uma **fase de configuração**, em que se poderão escolher as funcionalidades que serão acessíveis. Assim, o aplicativo Elderoid será composto por duas fases, após a sua instalação a partir da **App Store** – a **fase de configuração** (capítulo presente) e a **fase de utilização** (capítulo **2.6**).

De forma a atribuir ao utilizador o referido poder de escolha de funcionalidades a utilizar, é necessária esta **fase de configuração** prévia em que isso será feito. Esta fase também será utilizada para pedir as permissões necessárias (para que o aplicativo consiga utilizar certas funcionalidades do sistema), para configurar o nome e outros dados do utilizador e para configurar o Elderoid como **Launcher** padrão.

Na **figura 15**, apresenta-se o fluxograma desta fase. Concentra-se em cinco principais ações – a configuração do utilizador, das funcionalidades, das permissões e da app inicial.

Note-se que quando todas as configurações já foram efetuadas, o aplicativo salta imediatamente para a **fase de utilização**, aquando do sua iniciação. Se o utilizador abandonar o aplicativo a meio da fase de configuração, não terá de configurar tudo novamente, visto que começará no sítio onde estava, podendo sempre voltar a trás sem causar nenhum problema.

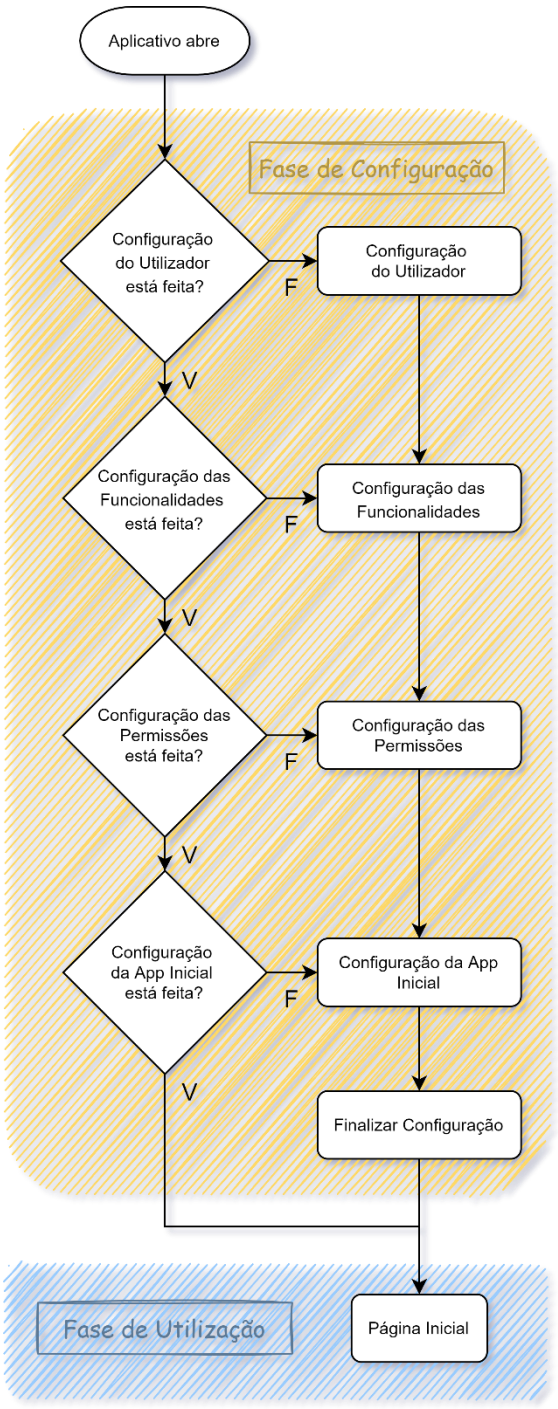

Figura 15 - Fluxograma da Fase de Configuração

2.5.1 Configuração do Utilizador

Na configuração do utilizador, o aplicativo aproveitará para dar a escolher a linguagem do Elderoid, explicar o seu funcionamento e obter alguns dados pessoais, como o nome do utilizador.

Na **figura 16** (escolha da linguagem), verifica-se a utilização da janela de feedback desenvolvida no capítulo **2.4**, exemplificando a forma como será utilizada esta componente do Nétie.

Em todas as janelas desta fase de configuração, o balão de fala apresentará uma frase muito breve a descrever o que se pretende. Todas as frases e palavras disponibilizadas no ecrã pelo Elderoid, estarão arrumadas no ficheiro **strings.xml**, de forma a facilitar uma possível tradução para outras línguas, além do português.

Depois de escolher a linguagem, o **Nétie** dará as boas-vindas ao utilizador e explicar-lhe-á que, sempre que quiser, poderá clicar nele para receber auxílio e conselhos em como utilizar o aplicativo (**figura 17**, janela da esquerda). Assim, resta apenas inserir o nome (como indica a janela à direita da **figura 17**), o qual será utilizado pelo **Nétie** para se dirigir ao utilizador.

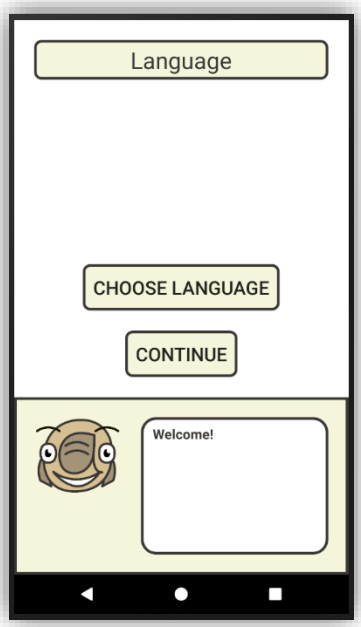

Figura 16 - Layout da escolha da linguagem

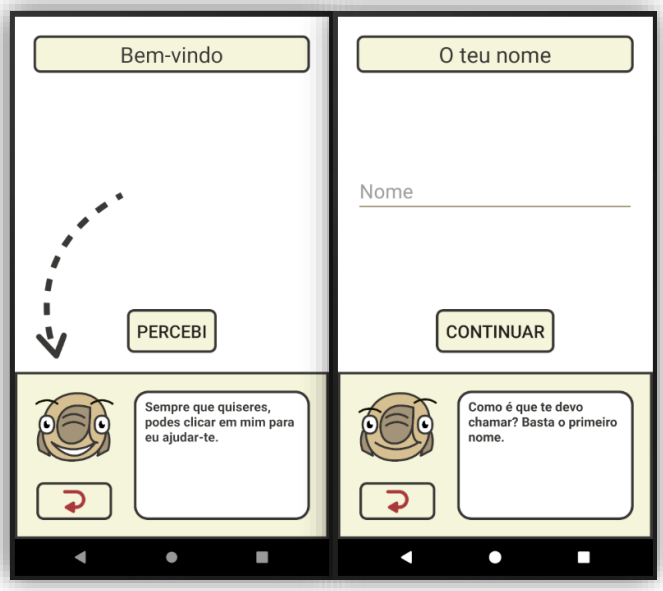

Figura 17 - Layout da Configuração do Utilizador

Na janela da esquerda da **figura 17**, é possível clicar no Nétie para receber uma confirmação de que o mecanismo de **Cues** (abordado no capítulo) funciona.

Note-se que em qualquer momento desta fase de configuração, é possível clicar no botão abaixo do **Nétie** para retroceder para a configuração anterior, bem como clicar no próprio **Nétie** para receber aconselhamento.

2.5.2 Configuração das Funcionalidades

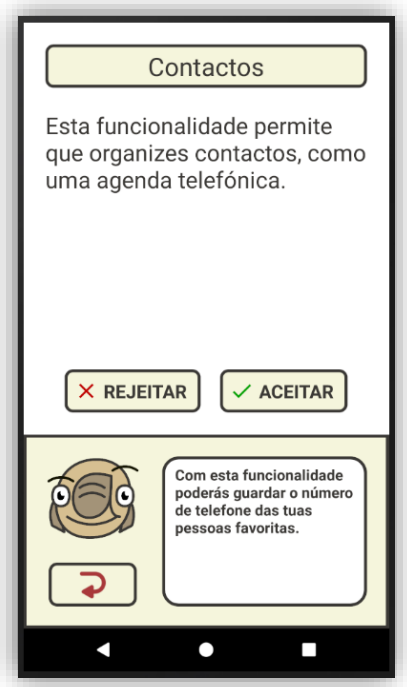

Figura 18 - Layout da configuração dos contactos

Para configurar as funcionalidades que estarão acessíveis no aplicativo, o utilizador terá de aceitá-las ou rejeitá-las. Neste menu de configuração, aparecerão as várias funcionalidades que o Elderoid fornece (uma de cada vez), acompanhadas sempre por um botão de **'Aceitar'** e um botão de **'Rejeitar'**, como exemplifica a **figura 18**. Depois de se aceitar ou rejeitar uma funcionalidade, o aplicativo salva para **SimplePrefs** a escolha do utilizador, de forma a que apenas aquilo que foi aceite pelo mesmo, ser disponibilizado na **fase de utilização**.

Nota: Apesar de, na fase de configuração, o Nétie se estar a dirigir ao utilizador de 3ª idade, que tirará partido da fase de utilização, aconselha-se que a fase de configuração seja realizada por alguém que possua alguma experiência com ***smartphones***. É apenas pura simpatia do personagem. No entanto, a fase de configuração foi projetada para que qualquer indivíduo fosse capaz de a realizar.

A lista de funcionalidades disponíveis é a seguinte: **Chamadas**, **contactos**, **mensagens**, **aplicações**, **meteorologia**, **notícias**, **fotos**, **galeria** e **lanterna**.

Como ilustra a **figura 19**, algumas das funcionalidades, caso sejam aceites, apresentam outra janela para que sejam configuradas algumas opções. No caso das aplicações, o utilizador terá de escolher quais as aplicações que o Elderoid disponibilizará. Com o botão da engrenagem (canto superior direito da **figura 19**), é possível filtrar as aplicações instaladas por categorias, como jogos ou redes sociais.

Quando satisfeito com a seleção, assim que o utilizador prosseguir, o nome do **Package**[20] de todas as aplicações selecionadas, será guardado em **SimplePrefs**.

Obter uma lista das aplicações instaladas e disponibilizá-la ao utilizador, não é tarefa fácil. Para listar as várias aplicações utilizar-se-á uma biblioteca fornecida pelo **Android** chamada **RecyclerView**. Como está descrito no Guia de Desenvolvimento Android [14], o RecycleView facilita e torna eficiente a exibição de grandes conjuntos de dados. Só é necessário fornecer os dados, definir a aparência de cada item, e a biblioteca, quando necessário, cria os elementos de forma dinâmica.

Figura 19 - Layout da configuração das aplicações

[20] **String** necessária para obter o nome e logotipo de uma aplicação, e para executá-la.

Como o nome indica, quando um item é deslizado para fora da tela (através de scroll), o **RecyclerView** não o destrói. Recicla-o. Reutiliza-o para disponibilizar um novo elemento que acabou de aparecer na tela. Para utilizar o RecyclerView é necessário, definir o layout de cada elemento da lista, e implementar duas classes que trabalharão juntas para definir como os dados serão mostrados ao utilizador:

- **ViewHolder:** Funciona como um contentor para cada item individual da lista.
- **Adapter:** Criará objetos **View Holder** conforme seja preciso, e define os dados que serão apresentados no item individual associado ao objeto.

O layout de cada elemento individual da lista é o que se apresenta na **figura 20**. Contém o logotipo e nome da aplicação, e uma caixa de seleção associada.

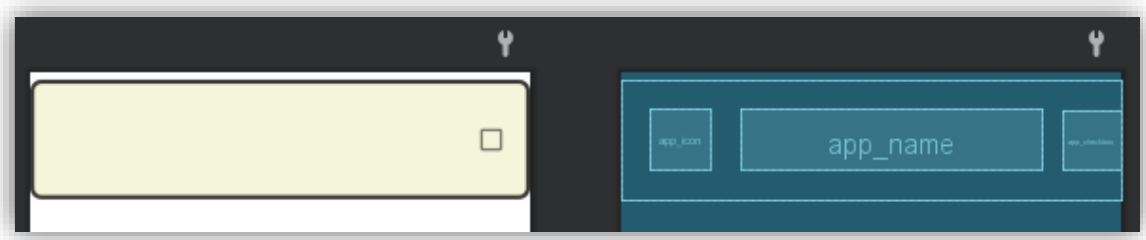

Figura 20 - Layout de um item da lista de aplicações

Contudo, existe um pequeno conflito em utilizar o mecanismo **RecyclerView** juntamente com caixas de seleção. Como cada elemento é reciclado, a memória da caixa de seleção ilustrada na **figura 20** é perdida. Ou seja, se o utilizador selecionar uma aplicação, por exemplo o **Gmail** (presente na **figura 19**), deslizar na lista de forma a colocar o **Google Maps** fora da tela (reciclando-o), e voltar a deslizar para o mesmo local, a caixa de seleção poderá já não estar selecionada.

A solução é guardar o estado de cada caixa de seleção programaticamente, sem que o **RecyclerView** se tenha de preocupar com elas. Quando um elemento for colocado na lista, a caixa de seleção será colocada com o estado guardado.

Recuando à **figura 19**, observa-se do lado direito do botão **"CONTINUAR"**, um ***joystick*** clicável, que leva a uma janela de sugestões de jogos. Como se ilustra na **figura 21**, o Elderoid sugere alguns jogos para Android, apropriados a indivíduos de terceira idade. Para disponibilizar esta breve lista, utilizou-se novamente o mecanismo **RecyclerView**, mas desta vez, cada item individual tem o layout da **figura 22**. Ao clicar num item da lista, o Nétie pergunta ao utilizador se quer instalar e redireciona-o para a página do jogo no Google Play Store.

Figura 21 - Menu de sugestão de jogos

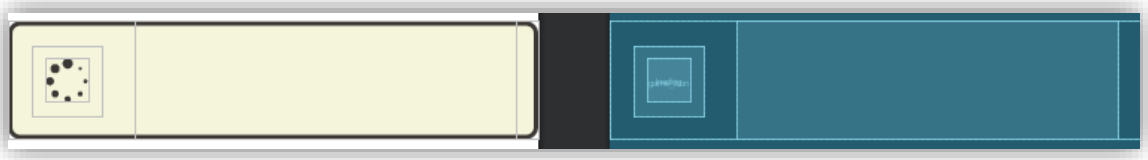

Figura 22 - Item da lista de jogos sugeridos

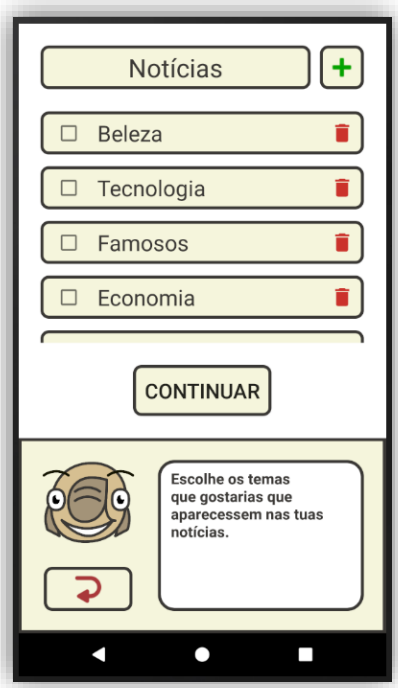

Figura 23 - Layout da configuração dos temas das notícias

Além das aplicações, outra funcionalidade que exige uma configuração adicional é a das notícias. Como se apresenta na **figura 23**, caso se aceite a funcionalidade das notícias, a janela que surgirá a seguir serve para configurar os tópicos das próprias. O Elderoid filtrará as notícias consoante as preferências temáticas do utilizador. Este processo será abordado num próximo capítulo.

Nesta janela (**figura 23**), o aplicativo apresenta tópicos padrão, dos quais o utilizador poderá selecionar os que preferir. Além disso, ao clicar no botão com um **'+'** verde, é possível adicionar tópicos personalizados.

Ao continuar, o aplicativo guarda os temas selecionados para **SimplePrefs**, e quando for necessário procurar notícias, a solicitação será feita com base nos tópicos guardados.

Para disponibilizar os vários tópicos na tela, utilizou-se a mesma estratégia que no menu de configuração de aplicações. Com recurso à biblioteca **RecyclerView**. Contudo, desta vez, o layout utilizado para cada item individual foi o da **figura 24**. Contém apenas uma caixa de seleção, uma caixa de texto e um botão para eliminar o tópico.

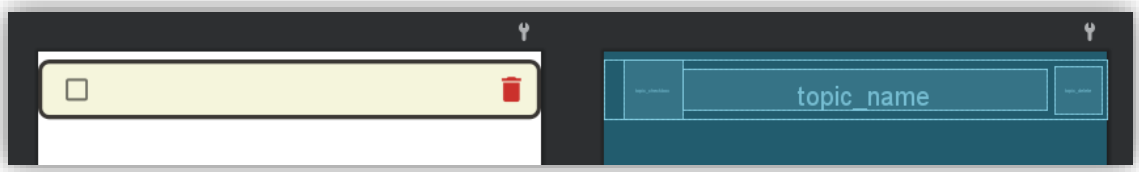

Figura 24 - Layout de um item da lista de tópicos de notícias

Caso o dispositivo não suporte uma determinada funcionalidade (por ausência de câmera ou lanterna, ou por ter uma versão muito antiga do sistema operativo), o **Nétie** lançará um aviso, quando o utilizador tentar aceitar dita funcionalidade. No caso da **figura 25**, o dispositivo não apresenta lanterna.

Por fim, existem funcionalidades que exigem conexão à internet (notícias e meteorologia) ou acesso à localização do dispositivo (meteorologia). Quando o utilizador tenta aceitar estas funcionalidades, o Elderoid lançará uma caixa de diálogo com um aviso a respeito.

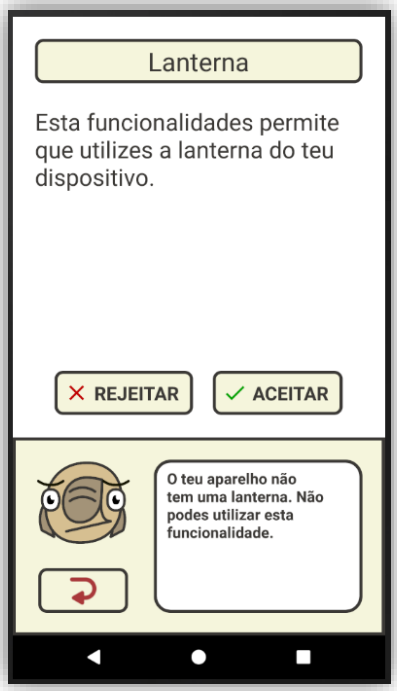

Figura 25 - Funcionalidade não suportada

2.5.3 Configuração das Permissões

Para que o Elderoid possa funcionar corretamente, e para que possa tirar partido de algumas funcionalidades do dispositivo, têm de ser pedidas algumas permissões ao utilizador. O Sistema Android obriga a aplicação a obter permissão diretamente do utilizador, quando se trata de algo que pode comprometer a segurança do mesmo. Por exemplo, aceder à localização do dispositivo ou efetuar chamadas.

Todas as permissões necessárias para o bom funcionamento do aplicativo deverão ser colocadas no ficheiro **AndroidManifest.xml**, ao qual esta **activity** recorrerá para obter as permissões que deverá solicitar ao utilizador. Normalmente, os aplicativos comuns solicitam as permissões aquando da sua necessidade. Contudo, o Elderoid terá de o fazer previamente, de forma a não confundir o utilizador na **fase de utilização**. Logo, nesta **activity**, as permissões serão solicitadas todas de uma vez, e terão de ser obrigatoriamente aceites, para se poder prosseguir. O **layout** desta **activity** é muito simples, como mostram as **figuras 26** e **27**.

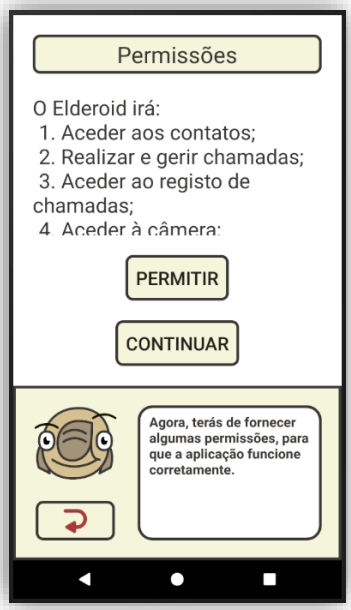

Figura 26 - Layout da Configuração de Permissões

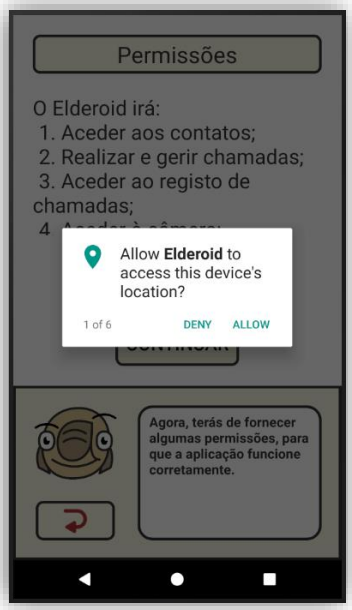

Figura 27 - Solicitação de uma permissão

Deverá também ser referido que o Nétie está atento a todos os eventos (não só nesta **activity**, mas em toda a fase de configuração), e instrui o utilizador daquilo que deve fazer e do que já fez. Por exemplo, no contexto desta **activity**, a **figura 28** mostra a deixa de Nétie quando o utilizador clica em **Continuar** antes de aceitar as permissões, e a **figura 29** mostra quando o utilizador clica em **Permitir**, mas as permissões já foram aceites.

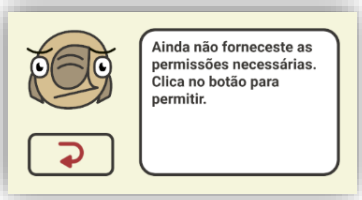

Figura 28 - Mensagem de Nétie quando não são fornecidas permissões

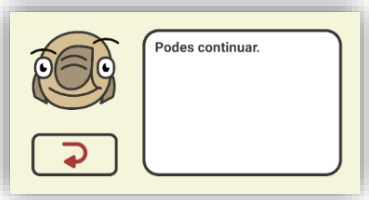

Figura 29 - Mensagem de Nétie quando já foram fornecidas permissões

O tipo de mensagem ilustrada nas **figuras 28** e **29**, são lançadas pelo Nétie recorrentemente, ao longo da navegação pelo Elderoid, conforme as ações do utilizador.

Nota: Apesar de parecer que o texto presente na tela da **figura 26** está cortado, este encontra-se contido num **scrollable**, pelo que é possível deslizar para ver o resto do texto.

2.5.4 Configuração da App Inicial

Este será o passo final da **fase de configuração**. Esta **activity** redirecionará o utilizador para as definições do Android associadas à configuração do App Inicial por defeito (***Default home app***), como mostra a **figura 31**. Nesse local, o Elderoid deverá ser selecionado, de forma a atuar como **Launcher** padrão no dispositivo. Infelizmente não existe forma de fazer isso programaticamente e de forma automática, pelo que terá mesmo de ser o utilizador a fazê-lo nas definições. Mais uma vez, a **activity** apresenta um layout muito minimalista, apresentado na **figura 30**.

Ao selecionar Elderoid, o utilizador é imediatamente redirecionado de volta para a aplicação.

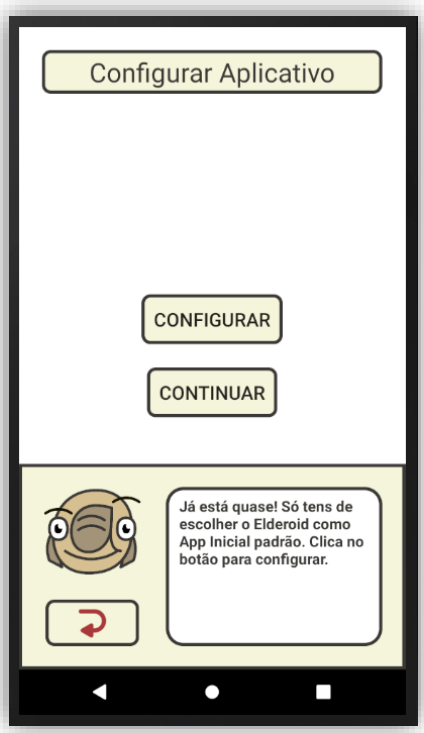

Figura 30 - Layout da Configuração do App Inicial

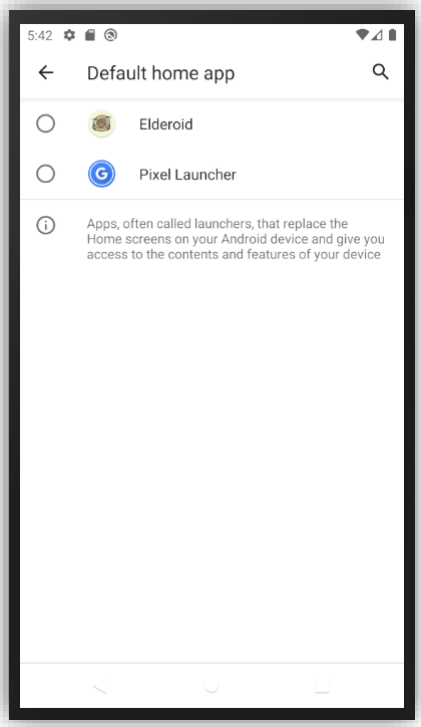

Figura 31 - Definições da app inicial

Dependendo da versão do sistema operativo e do dispositivo em utilização, pode não ser possível redirecionar o utilizador para o menu apresentado na **figura 31**. Logo, dependendo desses fatores, o método utilizado pelo Elderoid para levar o utilizador à configuração da App Inicial não será sempre o mesmo. No caso extremo, como último recurso (se o botão "configurar" não funcionar), a engrenagem no canto superior esquerdo leva às definições gerais do dispositivo, a partir das quais o utilizador terá de configurar a App Inicial manualmente. Isto apenas será necessário em versões do sistema operativo Android muito antigas.

Ao avançar, o aplicativo dará por finalizada a **fase de configuração**, o utilizador será enviado para a página inicial, e o Elderoid entrará na **fase de utilização**. Esta fase será abordada no capítulo **2.6**.

2.6 Menus da Fase de Utilização

Quando o aplicativo atinge esta fase, está pronto a ser utilizado casualmente. As funcionalidades selecionadas na **fase de configuração** estarão disponíveis e o utilizador poderá finalmente usufruir das mesmas. Nos próximos sub-capítulos abordar-se-ão cada um dos menus a desenvolver para esta fase.

2.6.1 Menu da Página Inicial

A página inicial do Elderoid terá de ser implementada muito cuidadosamente, visto que será a **activity** mais frequentada. O layout desta **activity** terá de apresentar todas as características discutidas no capítulo **2.3**.

Para começar, na parte inferior do ecrã, colocar-se-á a janela do Nétie. No entanto, não será uma janela qualquer. Note-se que até agora, na **fase configuração**, utilizaram-se apenas duas das três janels desenvolvidas para o Nétie. Na página principal, não faz sentido existir um botão de **Retroceder**, visto que ao clicá-lo, o Sistema Android assume que o utilizador quer sair da aplicação e redireciona-o para a página inicial padrão do dispositivo, que acontece ter sido configurada para o Elderoid, na fase de configuração. Ou seja, nada aconteceria. Também não faz sentido desperdiçar espaço ao simplesmente retirá-lo, por isso aproveitar-se-á para colocar um relógio digital, como mostra a **figura 12** do capítulo **2.4**, e a **figura 32** deste capítulo.

Figura 32 - Layout da Página Inicial

Além da janela inferior reservada para o Nétie, implementar-se-á uma barra superior, com o intuito principal de mostrar o nível da bateria. Por mais fácil que pareça disponibilizar um indicador do nível de bateria, fazê-lo em tempo real[21] implica algum conhecimento. Uma noção importante no âmbito de comunicação com o sistema em tempo real, é o conceito de ***Broadcast Receiver***[22] (**figura 33**). Esta funcionalidade permite que o aplicativo troque informações com o sistema, e até com outros aplicativos. Neste caso, tirar-se-á partido das informações transmitidas pelo sistema para obter alterações no nível da bateria. Para "ouvir" as transmissões enviadas pelo sistema, terá de ser criada uma instância da classe **BroadcastReceiver**, fornecida pelas bibliotecas **Android**, e o método **onReceive()** terá de ser reescrito. Sempre que o aplicativo recebe uma transmissão do sistema, este método será chamado, enviando o estado atual do sistema como parâmetro.

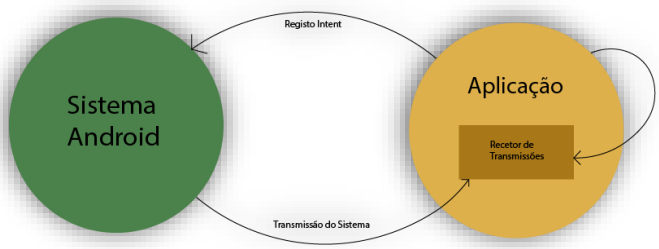

Figura 33 - Interação entre Aplicação e Sistema (Broadcast Receiver)

[21] Sempre que houver uma mudança na bateria.
[22] Traduz-se para Recetor de Transmissões.

Contudo, o Elderoid não se importará com todas as transmissões recebidas, por isso será aplicado um filtro a esta intância de BroadcastReceiver, de forma a executar **onReceive(...)** apenas quando há interesse do ponto de vista do aplicativo. O Elderoid apenas estará "à escuta" de alterações do nível de bateria, logo, após implementar o método de receção, a instância deverá ser registada com o seguinte filtro:

```
// Register battery broadcast receiver (mBatInfoReceiver being the instance of BroadcastReceiver)
this.registerReceiver(this.mBatInfoReceiver, new IntentFilter(Intent.ACTION_BATTERY_CHANGED));
```

O mais interessante acerca do indicador de bateria no canto superior direito da **figura 32**, é que foi desenhado utilizando apenas componentes visuais do próprio **Android Studio**. Ou seja, o comprimento da barra verde (a barra que indica quanta bateria ainda resta) é altamente dimensionável e dinâmica. Desta forma, não é necessário carregar várias imagens com vários comprimentos da barra, para representar diferentes conjuntos de percentagens da bateria. Dada a sua dinamicidade, sempre que onReceive(…) for executado, o seu tamanho é reconfigurado consoante a percentagem de bateria actual.

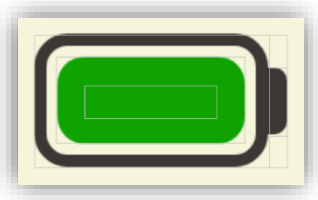

Figura 34 - Indicador da bateria a 100%

Esta característica dinâmica é conseguida com recurso a um atributo muito importante, que é utilizado com muita frequência nas várias componentes do Elderoid, e que fornece um elevado nível de compatibilidade entre diferentes resoluções dos dispositivos. O nome do atributo é **layout_constraintHeight_percent** e permite dimensionar o componente filho com base numa percentagem do componente pai[23].

Contido no retângulo exterior (a castanho escuro), encontra-se um retângulo mais pequeno (invisível, apenas de referência), que por sua vez conterá o retângulo interior (a verde), cujo comprimento apresentará uma percentagem do retângulo de referência. Quando a bateria está a **100%**, o retângulo verde apresentará um comprimento igual ao retângulo de de referência, como mostra a **figura 34**. Na **figura 35**, a bateria encontra-se a **30%**.

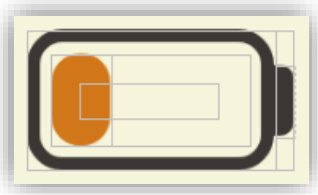

Figura 35 - Indicador da bateria a 30%

Figura 36 - Indicador da bateria a recarregar

Por fim, acrescentou-se um último componente associado ao indicador de bateria. Um raio, cuja função é indicar ao utilizador que a bateria está, de facto, a recarregar, como mostra a **figura 36**. Quando o dispositivo não está a receber energia, o raio desaparece. Note-se que os retângulos a cinzento não aparecerão no produto final, sendo que apenas representam as margens de cada componente no plano do **layout**.

Novamente, todos os referidos componentes associados a este indicador de bateria foram desenhados utilizando **XML**, à exceção do raio, que foi retirado de [15]. Como se observa na **figura 35**, o código que atualiza o retângulo interior consoante a percentagem da bateria, também atualiza a cor consoante a **tabela 1**.

Tabela 1 - Cor do Indicador de Bateria

Percentagem	Cor
60% - 100%	Verde
40% - 60%	Amarelo
20% - 40%	Laranja
0% - 20%	Vermelho

A **tabela 1** apresenta a gama de percentagens associada a cada cor do retângulo interior.

[23] O componente pai contém o componente filho. Por exemplo, a tela total contém um botão. Estas relações pai-filho podem ser aninhadas inúmeras vezes, conforme necessário.

Ainda no contexto da barra superior, se o utilizador escolher a meteorologia na fase de configuração, o canto esquerdo está reservado à indicação da temperatura e clima corrente. A forma como estes parâmetros serão obtidos será abordada no sub-capítulo do **Menu da Previsão do Tempo** (sub-capítulo **2.6.6**). Contudo, observando a **figura 32**, verifica-se que será necessário não só o valor de temperatura, como algum tipo de indicador do clima, de forma a disponibilizar uma pequena imagem que descreva o mesmo. Essa imagem encontrar-se-á encostada à margem esquerda da barra superior, seguida do valor de temperatura. Independentemente do clima, a imagem terá sempre o mesmo tamanho, e o valor de temperatura nunca sobreporá o indicador da bateria, visto que está referenciado ao mesmo.

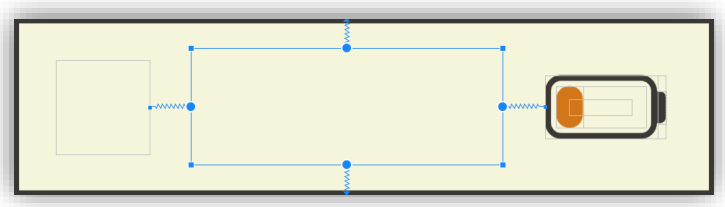

Figura 37 - Interação entre os componentes da barra superior

A **figura 37** mostra a forma como os componentes da barra superior interagem entre si. As linhas tremidas (com origem nos círculos azuis) representam as referências entre o valor de temperatura e os outros componentes, bem como as margens entre eles. A imagem do clima e o indicador da bateria encontrar-se-ão referenciados ao limite esquerdo e direito da barra, respetivamente. O valor de temperatura, como elemento central, estará referenciado a esse dois componentes. Neste caso, o texto contido no valor de temperatura terá de assumir um comportamento dinâmico. Isso é conseguido com recurso a alguns atributos disponibilizados pelas caixas de texto em Android, que permitem auto-redimensionamento, consoante o espaço disponível:

- **autoSizeTextType:** Apresenta duas opções – ***uniform*** e ***none***. A Opção ***uniform*** deverá ser selecionada para ativar a funcionalidade auto-redimensionamento do texto;
- **autoSizeMaxTextSize:** O tamanho máximo que o texto poderá assumir, aquando do auto-redimensionamento;
- **autoSizeMinTextSize:** O tamanho mínimo que o texto poderá assumir, aquando do auto-redimensionamento;
- **autoSizeStepGranularity:** O tamanho de incremento e decremento do auto-redimensionamento.

Tudo isto é necessário, dado que de dispositivo para dispositivo, a altura da barra poderá diferir, e por isso é importante redimensionar o texto (**AutoSize**) consoante o espaço que não estiver a ser ocupado pelos outros dois componentes. Contudo, as margens entre componentes são fixas, pelo que não atingem a dinamicidade desejada. Este assunto será abordado no capítulo de **Compatibilidade** (capítulo **2.8**).

Fora a barra superior e a janela do Nétie, o Menu da Página Inicial apresenta também um anel de botões. Estes serão os botões que redirecionarão o utilziador para os restantes Menus.

Esta disposição em anel é conseguida com recurso a uma das funcionalidades da **ConstraintLayout**. Essa funcionalidade dá pelo nome de ***Circular Positioning*** (Posicionamento Circular) e exige a criação de um elemento central, como mostra a **figura 38**. Neste caso, o elemento central estará invisível, e será colocado entre a janela do Nétie e a barra superior, de forma a ficar centrado no espaço disponível. Servirá apenas de referência.

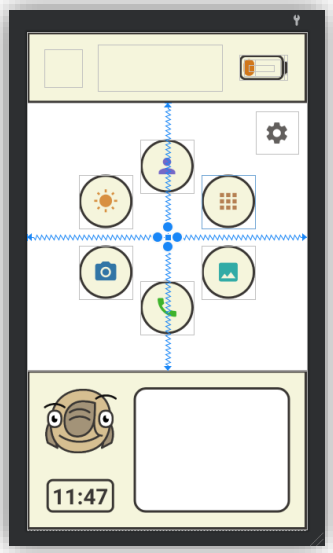

Figura 38 - Centro do anel de botões

Os elementos do anel terão de implementar os seguintes atributos:

- **layout_constraintCircle:** Referência ao elemento central.
- **layout_constraintCircleAngle:** Ângulo em que o elemento será posicionado
- **layout_constraintCircleRadius:** Distância entre o centro do elemento central e o centro do elemento do anel.

O anel de botões terá de ser flexível no sentido em que terá de se adaptar ao número de botões a disponibilizar. No exemplo da **figura 38**, o utilizador selecionou várias funcionalidades na **fase de configuração**, pelo que o Elderoid terá de disponibilizar os botões a elas associados. No caso de ser disponibilizado apenas um botão, por exemplo, o ideal seria colocá-lo no centro do ecrã, e não no próprio anel. Além disso, como haverá mais espaço livre, a dimensão também deverá ser incrementada.

Figura 39 - Página Inicial com apenas um botão

Para cada caso diferente, os botões apresentarão um **ângulo**, **raio** e **dimensão** diferente. No caso de ser necessário disponibilizar apenas um botão (**figura 39**), por exemplo, o raio será zero, pois será colocado no centro. Como o espaço será todo alocado para este único botão, a dimensão será maior do que nos outros casos, de forma a aproveitá-lo. O ângulo será indiferente, visto que o raio será zero. No caso de serem disponibilizados três botões, estes atributos terão de assumir valores diferentes.

Admitindo um máximo de seis botões possíveis para o anel, na **tabela 2**, apresentam-se estimativas testadas desses atributos para cada caso.

Tabela 2 - Estimativas do ângulo, raio e dimensão dos botões do anel

Número de botões	1	2	3	4	5	6
Ângulo	-	180º	120º	90º	72º	60º
Raio (rácio)	0	0.75	0.9	1	1	1
Dimensão (rácio)	0.34	0.3	0.27	0.24	0.22	0.2

Na **tabela 2**, o **Ângulo** representa o ângulo entre botões no anel. O **Raio** é relativo ao valor absoluto do caso **6**, que é o valor padrão e já está implementado pro defeito em **XML**. A **Dimensão** representa também um rácio, mas relativo ao espaço inteiro disponível para o anel, de forma a manter uma dependência entre o tamanho dos botões e o espaço disponível para os mesmos.

Uma solução seria usar a "força bruta" e especificar estes valores para cada caso. Contudo, tentar-se-á automatizar o processo.

Aquando da iniciação do menu da Página Inicial, obter-se-á a lista de botões a disponibilizar (**ArrayList<AppCompatImageButton> buttons**). A classe **AppCompatImageButton** representa uma instância de um botão que contém uma imagem (botões em questão). Para calcular o ângulo entre cada botão e obter os valores da **tabela 2**, efetua-se o seguinte cálculo:

```
float angle = (float) 360 / buttons.size();
```

Ao iterar pelos botões da lista **buttons**, o ângulo absoluto de cada botão é dado por:

```
((buttons.size() == 2) ? 180 - angle/2 : 180) + i*angle
```

Sendo **i** a iteração. À exceção do caso **2**, o ângulo do primeiro botão será sempre **180º** (o botão para efetuar chamadas, visto que, por defeito, este será sempre o primeiro a ser adicionado à lista).

Ao analisar os valores do **Raio** na **tabela 2**, verifica-se um comportamento semelhante à função inversa da tangente. Contudo, esta terá de ser deslocada no eixo das abcissas, de forma a se anular em **x=1**. Além disso, deverá tender para **1**, por valores positivos. Sabendo que o arco tangente tende para $\pi/2$, basta dividi-lo por esse valor. Como se não bastasse, para cumprir os valores da **tabela 2** minimamente, terá de apresentar também um declive mais acentuado junto à origem. Chega-se à seguinte fórmula:

$\frac{2}{\pi}\arctan\left(3\cdot(x-1)\right)$, sendo y o rácio do **Raio** e x o número de botões.

Em relação à **Dimensão**, verifica-se, na **tabela 2**, que existe um comportamente exponencial, visto que ao andar no eixo das abcissas, o valor do decremento das ordenadas diminui, isto é, diminuem cada vez menos. Para concluir uma expressão do tipo $y = a * b^x$ utilizaram-se os dois pontos extremos: **(1, 0.34)** e **(6, 0.2)**:

$\frac{0.34}{\left(\frac{10}{17}\right)^{\left(\frac{1}{5}\right)}}\cdot\left(\left(\frac{10}{17}\right)^{\left(\frac{1}{5}\right)}\right)x$, sendo y o rácio da **Dimensão** e x o número de botões.

Na **figura 40**, apresentam-se as funções concluídas para o **Raio** e **Dimensão**, utilizando a ferramenta Desmos para desenhá-las graficamente, a vermelho e azul, respetivamente.

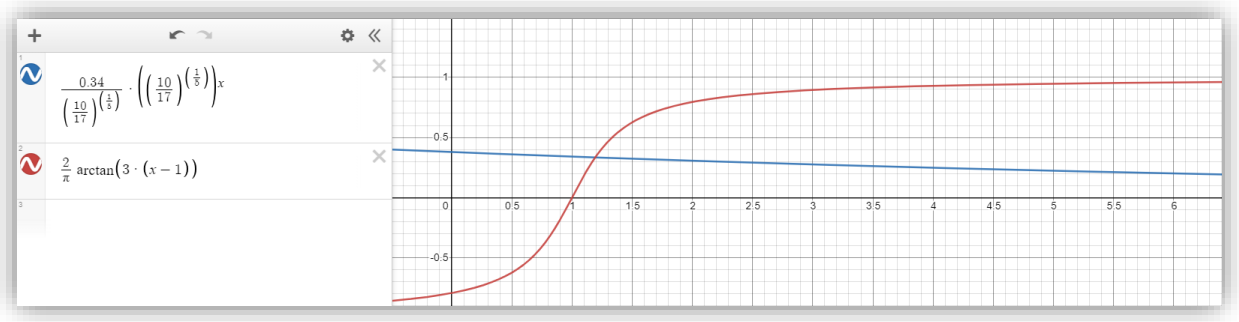

Figura 40 - Representação gráfica das funções do Raio e Dimensão

As curvas da **figura 40** apenas são relevantes, neste caso, para o intervalo de abcissas **[1, 6]**, e representam graficamente a variação dos parâmetros de **Raio** e **Dimensão** dos botões para cada cenário diferente. Relembra-se que o parâmetro **Raio** traduz a distância entre o centro do botão (qualquer um deles, desde que esteja presente, considerando o cenário) e o centro do anel.

Desta forma, não é necessário atribuir valores absolutos a cada botão, para cada caso, facilitando a escrita do código e reduzindo o número de linhas drasticamente.

Note-se que, apesar desta otimização, o Elderoid nunca aceitará mais do que 6 botões, visto que acabariam por se tornar demasiado pequenos e confusos. Um dos requisitos da interface deste aplicativo (como foi concluído no capítulo **2.3**) é manter as componentes visuais sóbrias e contrastantes.

Ainda no Menu da Página Inicial, recuando até à **figura 32**, observam-se 4 botões nas extremidades da secção central.

No canto superior direito, o botão com a engrenagem leva às definições do Elderoid, onde será possível alterar alguns aspetos do seu funcionamento. De forma a não ocorrerem cliques involuntários, o Nétie lança primeiro um aviso, antes que o utilziador aceda a este menu, visto que se trata de uma zona que pode comprometer o bom funcionamento do aplicativo. O aspeto da janela das definições está ilustrado na **figura 41**. Permite configurar a linguagem, a escala de temperatura, as funcionalidades e o nome. Permite também desativar o Elderoid, isto é, despromovê-lo de App Inicial padrão do dispositivo.

Figura 41 - Menu de definições

No canto superior esquerdo, um botão para ativar a lanterna do dispositivo. Se a lanterna não estiver disponível, esta funcionalidade também não será acessível.

No canto inferior esquerdo, o atalho para o menu de notícias, que será abordado no sub-capítulo **2.6.8**.

Por fim, no canto inferior direito, um botão que leva ao menu do registo de chamadas. Nesta janela (representada na **figura 42**) o utilizador poderá rever o seu histórico de chamadas efetuadas, recebidas e não-atendidas. Cada uma destas três apresenta um símbolo correspondente diferente: Seta para cima, seta para baixo e seta a fazer ricochete, respetivamente. Quando existirem chamadas não-atendidas recentes, o Nétie lançará um aviso na página inicial, como mostra a **figura 43**.

Figura 42 - Menu do registo de chamadas

Figura 43 - Aviso do Nétie de chamadas não-atendidas

2.6.2 Menu de Chamadas

Este é um dos menus mais importantes, visto que contem uma funcionalidade imprescindível, e que fará sempre parte do Elderoid. A de realizar chamadas.

Como se verifica na **figura 44**, o layout desta **activity** apresenta um teclado relativamente grande, com os algarismos de zero a nove, e mais dois botões – um de **Apagar** (a vermelho), que elimina o último algarismo inserido, e um de **Chamar** (a verde), que executa a chamada para o número de telefone escrito. Este número, à medida que o utilizador digita algarismos, aparecerá logo acima do teclado.

Figura 44 - Layout do Menu de Chamadas

Eis o código utilizado para executar uma chamada:

```
private void startCall(String number) {
    if (PhoneNumberUtils.isGlobalPhoneNumber(number)) {
        Intent intent = new Intent(Intent.ACTION_CALL);
        intent.setData(Uri.parse("tel:" + number));
        startActivity(intent);
    }
    else balloon_text.setText("Esse número de telefone não é válido.");
}
```

Ao criar um **Intent**, acrescenta-se uma referência **URI**[24] para o número de destino, antes de iniciar a **activity** para realizar a chamada (**activity** padrão do **Android**). O método **isGlobalPhoneNumber()** da classe **PhoneNumberUtils**, verifica se o número de telefone em number é válido. O atributo **balloon_text** representa a instância da caixa de texo do balão de fala do Nétie. No caso de o número não ser válido, o Nétie avisa o utilizador.

Tal como a maioria das **activities** do Elderoid, este layout apresenta também a janela do Nétie. No caso de a funcionalidade dos **Contactos** também tiver sido selecionada na **fase de configuração**, apareceram as duas frases da **figura 44** no balão de fala, bem como o botão Ligar. Senão, aparecerá apenas a primeira frase, e não aparecerá botão nenhum. O botão **Ligar**, no caso de existir, redireciona o utilizador para o **Menu de Contactos**, cujo layout e funcionamento será abordada no sub-capítulo **2.6.3**.

[24] **Uniform Resource Identifier** – String utilizada para identificar recursos.

2.6.3 Menu de Contactos

Esta é a **activity** que permite efetuar chamadas para contactos conhecidos, isto é, contactos armazenados no sistema Android, previamente. Como mostra a **figura 46**, utilizar-se-á um esquema muito semelhante ao layout da **Configuração das Aplicações**. Da mesma forma que se obtia uma lista das aplicações instaladas, para depois disponibilizá-la no ecrã com recurso à biblioteca RecyclerView, nesta **activity** obter-se-á um conjunto (**Set<>**, de forma a não permitir duplicados) dos contactos do sistema, através de um **Intent**.

Figura 46 - Layout do Menu de Contactos

Na **figura 45**, apresenta-se o aspeto visual de cada elemento da lista. Apresentará o nome e respetivo número de telefone, bem como um botão para realizar uma chamada.

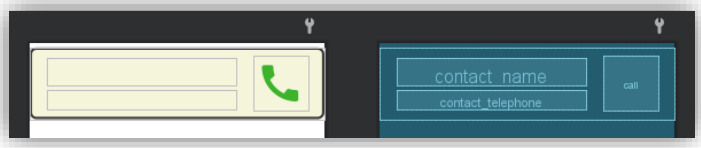

Figura 45 - Layout de um contacto individual

Através do botão verde do canto superior direito da **figura 46**, é possível adicionar contactos. Contudo, isso exige a competência de digitar conteúdo com recurso a um teclado, por parte do utilizador.

Para representar um contato no ambiente Elderoid, implementou-se uma classe chamada **ContactInfo**. Trata-se de uma classe muito simples, cujo diagrama **UML**[25] se encontra na **figura 47**. Implementa-se a Interface **Comparable<>**, de forma a facilitar a ordenação dentro de listas e conjuntos[26].

Programaticamente, as chamadas são efetuadas da mesma forma que a descrita no sub-capítulo **2.6.2**, verificando se o número de telefone é válido. Note-se que os números de telefone presentes na **figura 46** são aleatórios, visto que se trata de um emulador.

Figura 47 - Diagrama UML da classe ContactInfo

[25] **Unified Modeling Language** – Linguagem de modelação que simplifica a visualização de um sistema.
[26] Estruturas de dados em **Java** (**List<>** e **Set<>**)

2.6.4 Menu das Mensagens

No caso do menu das mensagens, a funcionalidade não foi implementada de raíz. O Elderoid simplesmente redireciona o utilizador para o aplicativo de **SMS**[27] padrão do dispositivo. Isto porque os aplicativos de **SMS** e de troca de mensagens em geral são bastantes simpels e de fácil compreensão.

Quando o utilizador receber uma ou mais mensagens, ser-lhe-ão notificadas na janela inicial, no próprio botão que redireciona para o aplicativo de **SMS** padrão (com um balão de fala azul), como indica a **figura 48**.

Figura 48 - Notificação de mensagem

[27] Short message service – Serviço de curtas mensagens

2.6.5 Menu das Aplicações

O menu das aplicações tem como finalidade disponibilizar ao utilizador as aplicações selecionadas na **fase de configuração**, caso essa funcionalidade tenho sido escolhida. Como foi descrito no capítulo **2.5.2**, o nome do pacote das aplicações selecionadas pelo utilizador, serão guardadas para **SimplePrefs**. Neste menu, tudo que há a fazer é recuperar essa informação, e disponibilizar as aplicações, tal como mostra a **figura 49**. Para isso tirou-se novamente partido da biblioteca **RecyclerView**, mas desta vez, utilizou-se um layout diferente para cada elemento individual da lista. Este layout é muito parecido com o da **figura 19**, do capítulo **2.5.2**, mas com duas pequenas diferenças:

- A caixa de seleção de cada elemento individual desapareceu, visto que já não é necessária;
- Cada elemento individual da lista é agora clicável, isto é, quando o utilizador clicar numa aplicação da lista, o Elderoid efetuará um pedaço de código previamente definido e atribuído.

Figura 49 - Layout do Menu das Aplicações

Neste caso, o código atribuído a cada elemento, e que responderá ao clique do utilizador, será o código necessário para executar a aplicação em questão. Utilizar-se-á um **Intent**, e, na posse do nome do pacote da aplicação, isso torna-se incrivelmente fácil:

```
app_item.setOnClickListener(v -> {
    Intent launchIntent = getPackageManager().getLaunchIntentForPackage(app.getPackageName());
    startActivity(launchIntent);
});
```

Sendo **app_item** uma instância de um elemento individual da lista, **launchIntent** a instância do **Intent** para executar a aplicação, e **app** uma instância da classe **AppInfo** (uma classe muito simples para representar as informações de uma aplicação).

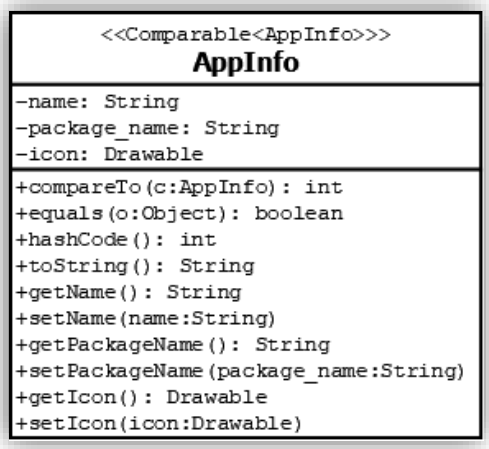

Figura 50 - Diagram UML da Classe AppInfo

Do ponto de vista do Elderoid, existem 3 atributos importantes associados a cada aplicação. O nome, o nome do pacote e um **drawable** associado. Um drawable (classe **Drawable**) é qualquer coisa que possa ser desenhada na tela, neste caso, o logotipo da aplicação em questão.

Como se verifica no diagrama **UML** da **figura 50**, implementou-se a interface **Comparable<>**, de forma a facilitar a ordenação dentro de listas e conjuntos, tendo de se sobrescrever o método **compareTo()**.

O botão com a engrenagem no canto superior direito da **figura 49**, leva à configuração das aplicações, onde o utilizador poderá escolher novas aplicações.

2.6.6 Menu da Previsão do Tempo

Figura 51 - Layout do Menu de Previsão do Tempo

De forma a conseguir obter informação sobre a meteorologia, o Elderoid necessitará de conexão à internet e da localização do dispositivo. Se algum destes não estiver disponível, um erro será apresentado.

No contexto da Meteorologia, o Elderoid apresentará duas funcionalidades. Na página inicial, abordada no capítulo **2.6.1**, a temperatura e clima corrente será apresentada no canto superior esquerdo, através de uma pequena imagem descritiva do clima, e de um valor em **ºC** (graus **Celsius**) ou **ºF** (graus **Fahrenheit**). Tanto a imagem como o valor, serão atualizados recorrentemente, desde que a página inicial esteja a ser mostrada ao utilizador. O período de tempo entre autalizações terá o nome de **WEATHER_REQUEST_DELAY**, e será algo como **10** segundos a **1** minuto.

Como segunda funcionalidade, a componente de meteorologia do Elderoid também disponibilizará uma previsão do tempo e clima, para, no máximo, os **4** dias seguintes. Essa informação poderá ser facultada no **Menu de Previsão do Tempo**, cujo layout está apresentado na **figura 51**.

Cada dia previsto, no layout da **figura 51**, contém a seguinte informação:

- **Dia da semana;**
- **Imagem que descreve o clima;**
- **Temperatura mínima e máxima em ºC ou ºF.**

Ao clicar num dos dias, o utilizador recebe uma mensagem descritiva do Nétie relativa ao clima do respetivo dia, como indica a **figura 52**.

Figura 52 - Descrição do clima do dia selecionado

Serão implementadas algumas classes que serão úteis na obtenção e estruturação dos dados acerca da temperatura e clima. O diagrama **UML** da **figura 53** mostra o conteúdo e relação entre essas classes.

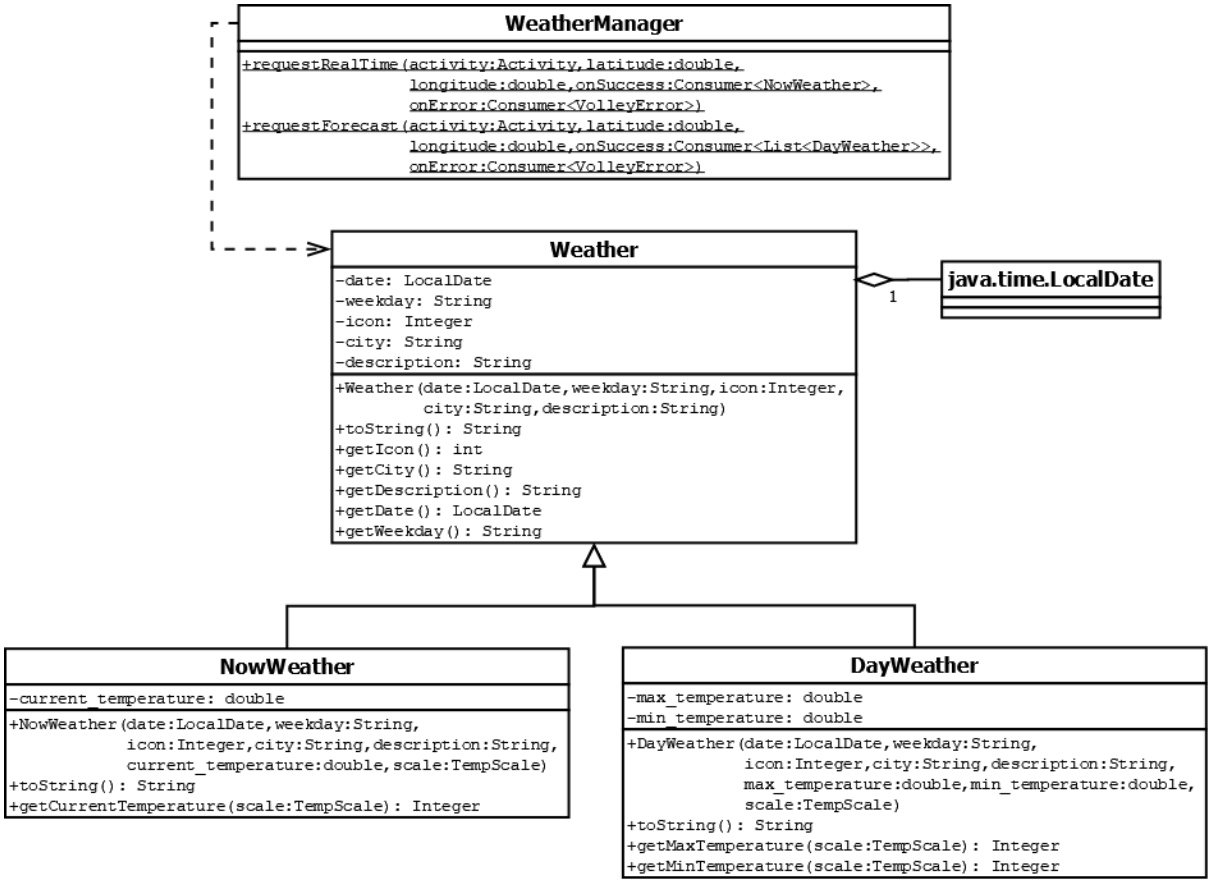

Figura 53 - Diagrama UML das classes que trocam informações relacionadas com a meteorologia

Como pode ser observado na **figura 53**, a classe **NowWeather** transporta informação acerca da temperatura e clima correntes, e será utilizada no método estático **requestRealTime()**, presente na classe **WeatherManager**. Por sua vez, a classe **DayWeather** representa a previsão da temperatura e clima para um determinado dia, e será utilizada no método estático **requestForecast()**. Ambas herdam da classe **Weather**.

A classe **WeatherManager** apresenta apenas dois métodos, públicos e estáticos, e ambos apresentam os seguintes parâmetros:

- **activity:** A **activity** que a solicitou;
- **latitude:** Valor de latitude do dispositivo;
- **longitude:** Valor de longitude do dispositivo;
- **onSuccess:** Função a executar, caso a informação seja obtida com sucesso;
- **onError:** Função a executar, no caso de erro.

A diferença entre os dois métodos, é que o método **requestRealTime()**, no caso de sucesso, executa **onSuccess** sobre uma instância da classe **WeatherNow**, enquanto **requestForecast()**, no caso de sucesso, executa **onSuccess** sobre uma lista de instâncias da classe **WeatherDay**.

Para lidar com as datas, utilizar-se-á a classe **LocalDate**.

De forma a obter a informação sobre Meteorologia, utilizar-se-á a **API OpenWeatherMap** [16] desenvolvida pela empresa **OpenWeather**. Esta **API** permite o acesso a vários dados meteorológicos disponibilizados pela empresa. Contudo, certas solicitações de informação exigem planos de subscrição pagos. A coleta de informação feita pelo Elderoid consistirá apenas no plano grátis. As duas funcionalidades disponíveis para este plano, que serão mais úteis neste projeto, dão pelo nome de:

- **Current Weather Data:** Solicita informação acerca da temperatura e clima correntes, aceitando como parâmetros os valores de latitude e longitude;
- **5 Day / 3 Hour Forecast:** Solicita informação da temperatura e clima a cada **3** horas, durante os próximos **5** dias, isto é, a resposta consistirá numa lista parecida com a **tabela 3**.

Tabela 3 - Exemplo da lista de informação recebida

Dia e hora	Temperatura e Clima
...	...
27/04/2021, 18:00	18ºC, 01n
27/04/2021, 21:00	19ºC, 02n
28/04/2021, 00:00	21ºC, 02n
28/04/2021, 03:00	20ºC, 03n
28/04/2021, 06:00	19ºC, 01n
...	...

Na segunda coluna da **tabela 3**, o clima é representado por um código do género "***01n***". No âmbito da **API**, este código representa um ícone, que descreve a condição climática. Todos os códigos e respetivo ícone e descrição, podem ser encontrados em [17]. Contudo, optou-se por utilizar ícones diferentes, retirados de [18], apresentados na **figura 54**.

Como pode ser verificado em [17], a **API** disponibiliza códigos tanto para as condições climáticas durante

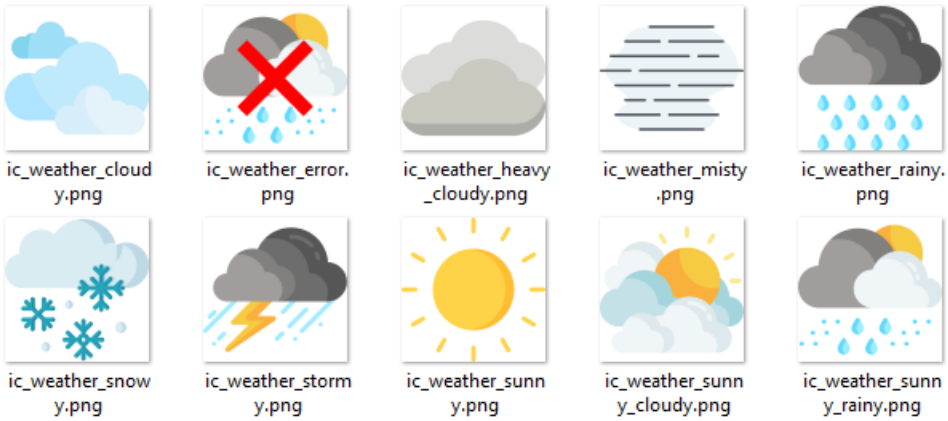

Figura 54 - Ícones de clima utilziados no Elderoid

o dia, como durante a noite. No entanto, o Elderoid não considerará essa distinção, generalizando os ícones para dia e noite.

O ícone **ic_weather_error** será utilizado quando o utilizador não estiver conectado à internet, ou o GPS estiver desligado, como mostra a **figura 55**. Quando isso acontecer, o botão da página inicial que leva ao menu de previsão do tempo deixa de funcionar e o Nétie lança um aviso.

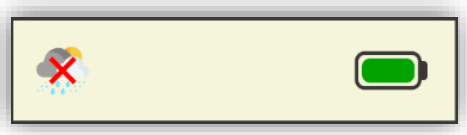

Figura 55 - Ícone de erro da meterologia

Os dados meteorológicos são obtidos através do protocolo **HTTP**[28], utilizando o método **GET**. Para experimentar, realizou-se uma solicitação manual com recurso ao software **Postman** [19], obtendo-se uma resposta em formato **JSON**[29], como apresenta a **figura 57**.

Na **figura 56**, estão apresentados os parâmetros utilizados na solicitação: o valor de latitude, o valor de longitude e o **appid**, que é uma chave oferecida pela **OpenWeather** de acesso aos seus servidores.

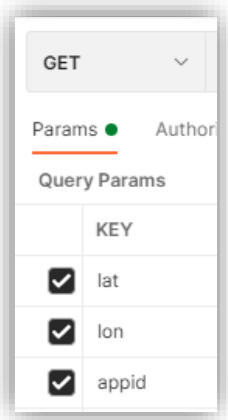

Figura 56 - Parâmetros da solicitação HTTP GET


```
{
    "coord": {
        "lon": -9.1017,
        "lat": 38.8262
    },
    "weather": [
        {
            "id": 803,
            "main": "Clouds",
            "description": "broken clouds",
            "icon": "04d"
        }
    ],
    "base": "stations",
    "main": {
        "temp": 292.49,
        "feels_like": 292.12,
        "temp_min": 292.04,
        "temp_max": 293.15,
        "pressure": 1008,
        "humidity": 63
    },
    "visibility": 10000,
    "wind": {
        "speed": 3.09,
        "deg": 310
    },
    "clouds": {
        "all": 75
    },
    "dt": 1619528147,
    "sys": {
        "type": 1,
        "id": 6901,
        "country": "PT",
        "sunrise": 1619502205,
        "sunset": 1619551462
    },
    "timezone": 3600,
    "id": 8014125,
    "name": "São João da Talha",
    "cod": 200
}
```


Figura 57 - Resposta JSON

A resposta obtida oferece diversos tipos de informação, desde o nome da cidade onde estão a ser feitas as medidas, até à velocidade do vento.

De toda esta informação, o Elderoid aproveitará o nó "***temp***" e "***icon***", essencialmente. Note-se que esta resposta foi obtida recorrendo à funcionalidade **Current Weather Data**, e por isso apresenta a informação referente à temperatura e clima correntes. A análise de uma resposta **JSON** da funcionalidade **5 Day / 3 Hour Forecast**, revelar-se-á um pouco mais complexa, visto que trará uma lista do com informação dos **5** dias (de **3** em **3** horas), como mostra a **tabela 3**.

[28] **Hypertext Transfer Protocol** – Protocolo de comunicação entre cliente e servidor do tipo solicitação-resposta.
[29] **Javascript Object Notário** – Forma de representar dados estruturados.

Para realizar solicitações **HTTP** em **Java**, utilizar-se-á uma biblioteca disponibilizada pelo Android, chamada **Volley**. Como descreve o Guia de Desenvolvimento Android em [20], o procedimento será criar uma fila de solicitações, e adicionar solicitações conforme necessário. Quando for obtida uma resposta, a função passada em **onResponse()** será executada, no caso de sucesso. Se ocorrer um erro, a função passada em **onErrorResponse()** será executada. É por esta razão que é importante que os dois métodos estáticos da classe **WeatherManager** aceitem como parâmetros duas funções para executar em cada caso: **onSuccess** e **onError**, no caso. Este mecanismo permite que as solicitações tenham um comportamento assíncrono.

No caso da previsão do tempo, terá de ser utilizado um algoritmo que traduza a informação recebida para cada dia, numa única temperatura máxima, uma única temperatura mínima e um único ícone.

Sabendo que, para cada dia, a resposta **JSON** enviará sempre 8 previsões de temperatura (o dia presente não será incluído), isto é, às **00:00**, às **03:00**, às **06:00**, às **09:00**, às **12:00**, às **15:00**, às **18:00** e às **21:00**, a ideia será verificar qual a temperatura máxima e mínima, e qual o ícone com mais ocorrências em todas as 8 medidas. Caso haja dois ícones com o maior número de ocorrências, o primeiro encontrado será selecionado. Este funcionamento encontra-se ilustrado na **figura 58**, para uma segunda-feira aleatória. O ícone selecionado é "*02*", porque, como já foi discutido, o "***n***" e "***d***" serão ignorados.

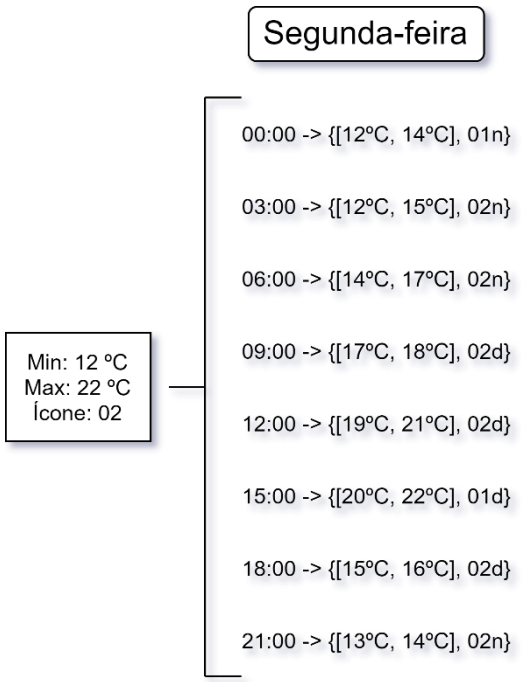

Figura 58 - Exemplo de previsão para um dia

Nota: O algoritmo utilizado está preparado para prever a temperatura e clima do dia corrente mais os 3 dias seguintes, ou dos 4 dias seguintes. Esse aspeto pode ser alterado através de uma constante estática booleana presente na classe **WeatherManager**, denominada **FORECAST_PRESENT_DAY**. Por defeito, está declarada como verdadeira, ou seja, prevê a temperatura e clima também do dia corrente.

2.6.7 Menu da Galeria

O **Menu da Galeria** será uma janela que, mais uma vez, tirará partido da biblioteca **RecyclerView**. Desta vez, organizar-se-ão os vários items individuais em grelha, e não de forma vertical, como foi implementado no **Menu de Aplicações** e no **Menu de Contatos**. Como se ilustra na **figura 59**, as fotografias serão dispostas numa grelha de 2 colunas, e cada uma delas estará acompanhada de um botão vermelho que servirá para removê-la.

Todas as fotografias serão clicáveis, e levarão o utilizador para uma janela com apenas a fotografia clicada, de forma a se poder efetuar aproximações e observar em tamanho grande (**figura 60**).

Programaticamente, para obter as fotos e disponibilizá-las na tela, é necessário realizar uma solicitação **SQL**[30] ao armazenamento interno do dispositivo. Mais informação em [21]. Por defeito, as fotos tiradas com a câmera integrada são guardadas em "***DCIM/Camera/***". Contudo, em alguns dispositivos isso não se verifica, por isso a solicitação terá de percorrer todas as diretorias do dispositivo à procura de ficheiros com extensão **.jpg**, **.jpeg**, **.png**, **.gif** ou **.bmp**. Desta forma todas as imagens serão encontradas (incluindo **screenshots**[31], por exemplo).

Figura 59 - Menu de Galeria

Figura 60 - Fotografia em tamanho grande

Para carregar uma imagem do armazenamento interno para a janela, recorreu-se a uma biblioteca que gere e manuseia imagens em ambiente Android. Dá pelo nome de **Glide** [22]**.** Esta biblioteca permite, com uma única linha de código, carregar uma imagem para uma **ImageView**, fornecendo apenas o seu caminho (**path**). Permite também acrescentar alguns detalhes como cantos recortados ou centerização. Para realizar aproximações, utilizou-se a biblioteca **PhotoView** [23].

Ao clicar no botão vermelho com um caixote do lixo, surgirá uma caixa de diálogo para o utilizador confirmar que quer realmente eliminar a fotografia. Se o utilizador confirmar, ela será eliminada do dispositivo totalmente.

No Elderoid, ao utilizar a câmera, garante-se que a fotografia será guardada "***DCIM/Camera/***". Logo a seguir, refresca-se o armazenamento interno, para que a fotografia apareça na galeria instantaneamente, visto que em alguns dispositivos é necessário esse procedimento.

[30] Structured Query Language.
[31] Fotografia tirada à tela do ecrã.

Contudo, o maior problema de compatibilidade no que toca à câmera e galeria, reside na tarefa de guardar uma fotografia acabada de tirar para o armazenamento do dispositivo. Para gerir o acesso à camera e o armazenamento de fotografias tiradas, criou-se a classe representada na **figura 61**.

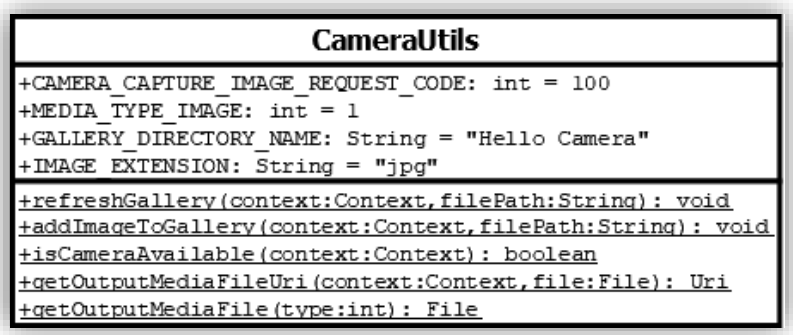

Figura 61 - Diagrama UML da classe CameraUtils

A classe **CameraUtils** (**figura 61**) apresenta métodos muitos úteis, todos eles estáticos, para detetar adisponibilidade da câmera, instanciar ficheiros, adicionar fotografias à galeria e refrescá-la (o que, como já foi abordado, será necessário efetuar com alguma frequência).

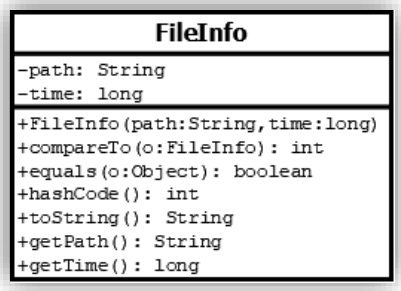

Figura 62 - Diagrama UML da classe FileInfo

Do outro lado da moeda, de forma a recuperar essas mesmas fotografias para, posteriormente, disponibilizá-las na galeria, implementou-se a classe da **figura 62**.

Observa-se que a classe **FileInfo** apresenta como atributos não só o caminho para o ficheiro, como também um valor **long** que representa a altura em que o ficheiro foi criado. Este valor é referente ao número de dias desde a data **Epoch** (00:00:00 **GMT**[32], 1 de Janeiro de 1970), e será importante para ordenar as fotografias pela data em que foram tiradas.

[32] Greenwich Mean Time.

2.6.8 Menu das Notícias

O Menu de Notícias responsabilizar-se-á por mostrar notícias ao utilizador, consoante os seus temas favoritos, selecionados na configuração dos tópicos de notícias. O aspeto desta janela será o ilustrado pela **figura 63**, em que se tirará partido, mais uma vez, da biblioteca **RecyclerView**, de forma a tornar a visualização das várias notícias o mais eficaz possível.

Figura 63 - Menu de Notícias

No caso da **figura 63**, o utilizador tem especial interesse por **Culinária** e **Receitas**, pelo que essas serão as temáticas principais das suas notícias. Caso seja necessário, será sempre possível alterar as temáticas das notícias, clicando no botão com uma engrenagem, ou com auxílio do **Nétie**.

É possível deslizar até um máximo de **50** notícias.

Com recurso à **API FreeNews**,disponibilizada pelo marketplace da **RapidAPI** [24], documentada em [25], e realizando solicitações do mesmo calibre das realizadas no sub-capítulo **2.6.6**. Uma solicitação de notícias pode envolver vários parâmetros, entre os quais:

- **Temáticas (q):** Podem ser mais que uma, e podem ser conjugadas de diferentes maneiras. Por exemplo, para procurar notícias relacionadas com Culinária **e** Tia Cátia, este parâmetro deverá ser **<q=Culinária && Tia Cátia>**. Por outro lado, se a intenção for procurar notícias relacionadas com Culinária **ou** Tia Cátia, deverá ser **<q=Culinária || Tia Cátia>**. O Elderoid apenas utilizará o concatenador **||**.
- **Linguagem (lang):** Linguagem das notícias (exemplo: **<lang=pt>**);
- **Número de notícias (page_size):** Número de artigos devolvidos pela API (exemplo: **<page_size=25>**;
- **Margem temporal (from):** O quão antigas as notícias poderão ser. Se a intenção for obter notícias até há 30 dias, então a data de há 30 dias deve ser passada (exemplo: **<from=2021-13-06>).**

Simulando uma solicitação no ***software* Postman**, a **API** devolve, na sua resposta **JSON**, um array de artigos. Cada artigo apresenta campos informativos da própria notícia. O Elderoid tirará partido de muitos desses campos e instanciará um objeto **New** para armazenar cada artigo. Na **figura 64**, está representada a classe **New** juntamente com a classe **NewsManager**, que se responsabilizará por realizar as solicitação necessárias à **API**

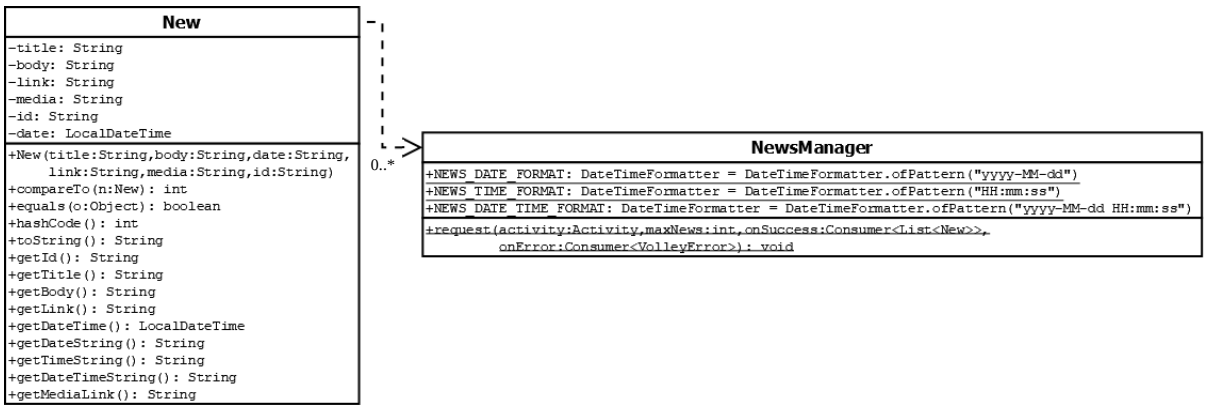

Figura 64 - Diagrama UML das classes New e NewManager

Figura 65 - Layout de uma Notícia

Cada item que a biblioteca **RecyclerView** disponibilizará na lista (cada artigo de notícia), terá o aspeto ilustrado na **figura 65**. Grande parte do **layout** é ocupado pela imagem que ilustra a notícia (devolvida também pela **API**). No canto superior direito da imagem, encontra-se um botão para eliminar a notícia em questão. Por fim, por baixo da imagem, apresentam-se o título (com um máximo de **2** linhas) e o corpo da notícia (com um máximo de **3** linhas).

Além disso, como acontece com grande parte dos componentes visuais que podem demorar algum tempo a renderizar, antes de a imagem aparecer, ocorre uma animação, com o intuito de mostrar ao utilizador que a imagem ainda está a ser renderizada. A animação consiste num símbolo de carregamento com círculos de diferentes tamanhos (ilustrado na **figura 66**) a realizar movimentos de rotação. Quando a imagem for renderizada e aparecer na janela, a animação termina e o símbolo de carregamento desaparece.

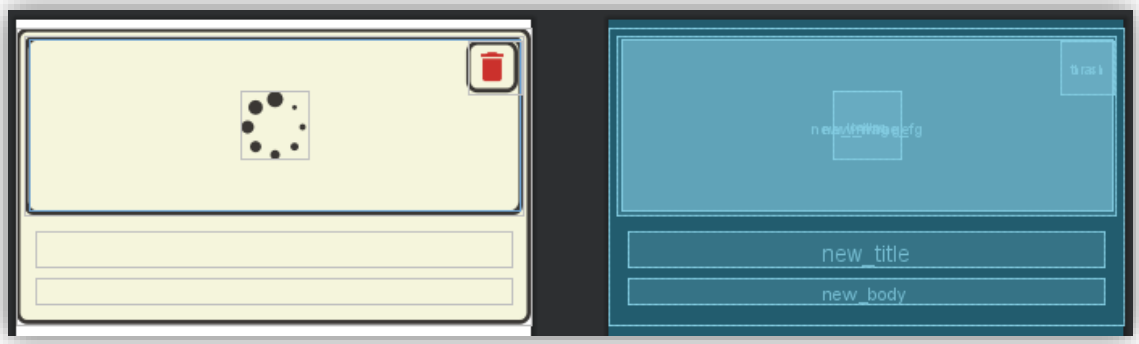

Figura 66 - Plano do Layout de uma Notícia

2.7 Estrutura Final do Aplicativo

É importante fazer uma ponderação final acerca da estrutura do aplicativo. A **figura 67** ilustra o aplicativo de um ponto de vista mais relacionado com a lógica e algumas das classes utilitárias envolvidas.

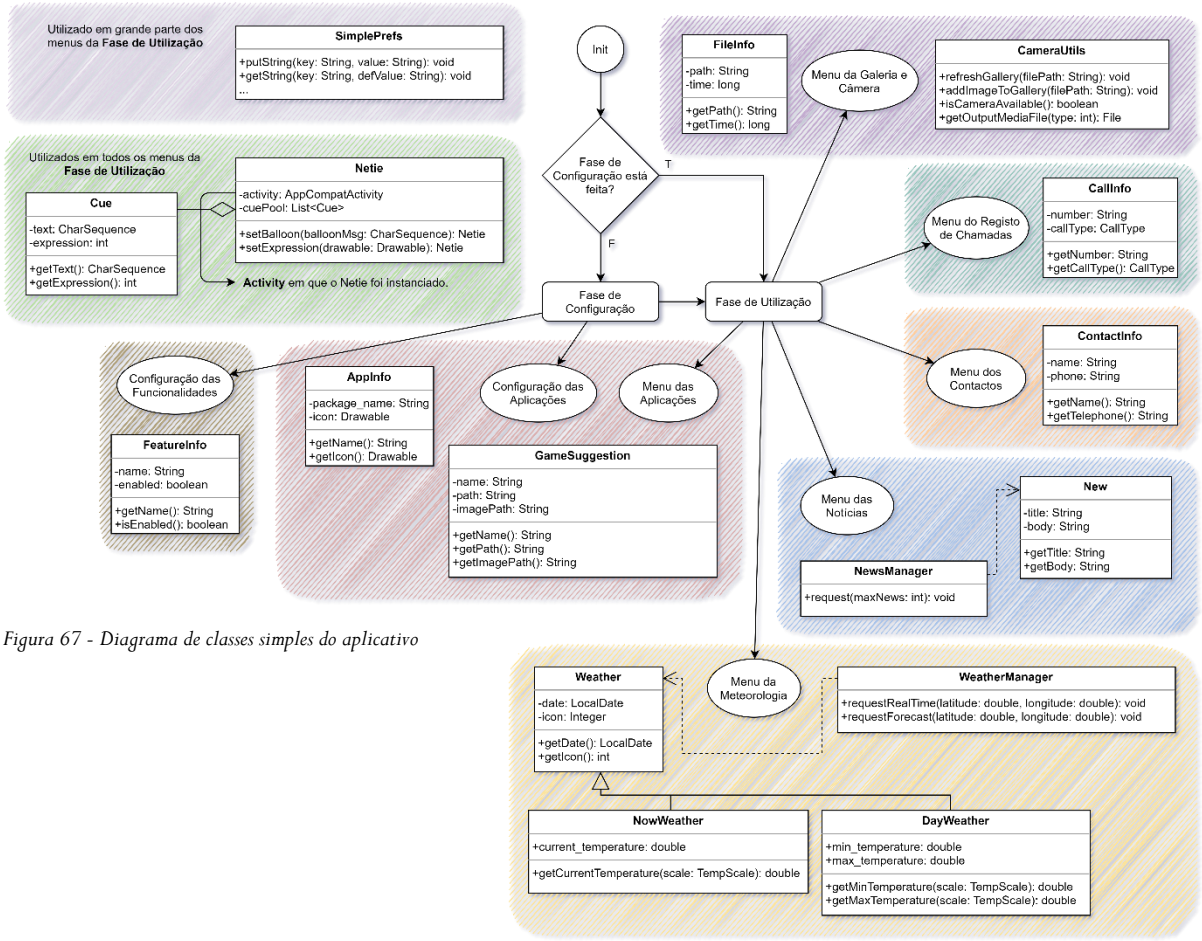

Figura 67 - Diagrama de classes simples do aplicativo

Descrevendo a **figura 67** de forma sucinta, quando o aplicativo inicia pela primeira vez, terá de passar pela **fase de configuração**. Uma vez configurado, o Elderoid transita para a **fase de utilização**, na qual o utilizador poderá navegar por diferentes menus, cada um com determinadas classes associadas, das quais tirará partido.

As classes **SimplePrefs**, **Netie** e **Cue** são classes utilizadas em várias **activities** do Elderoid, pelo que não estão diretamente associadas a nenhum menu do Elderoid. Todas as três classes já foram abordadas em capítulos precedentes.

O menu de Configuração das Aplicações e o menu das Aplicações partilham o acesso às classes **AppInfo** e **GameSuggestion**.

2.8 Compatibilidade

Como foi referido ao longo do documento, este projeto almeja um certo nível de compatibilidade entre diferentes ***smartphones*** e até ***tablets***. Neste capítulo, explicar-se-á as medidas e precauções necessárias para tentar, ao máximo, garantir esse nível de compatibilidade.

À medida que foram desenvolvidos os layouts das diferentes **activities** do Elderoid, a organização dos componentes visuais presentes na tela foi meticulosamente concebida, de forma a que a sua posição e dimensão seja flexível, e se adapte ao tamanho do ecrã do dispositivo. Para isso utilizou-se, em geral, a biblioteca **ConstraintLayout**, como referido no sub-capítulo **2.2.2**. Contudo, de forma a que os componentes não fiquem imediatamente encostados uns aos outros, acrescentam-se margens entre eles.

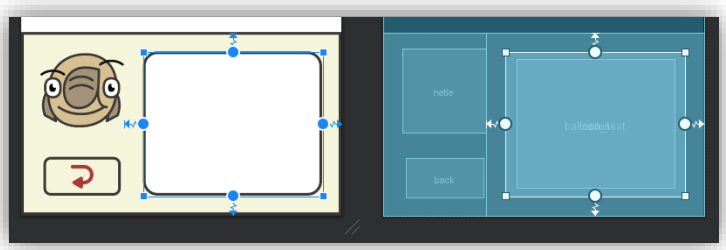

Figura 68 - Margens do balão de fala da janela do Nétie

Tomando como exemplo a janela do Nétie, observa-se na **figura 68**, que o balão de fala apresenta margens relativamente à própria janela. Isso é necessário, não só por motivos estéticos, mas para não confundir o utilizador. O problema é que o valor atribuído a essa margem é um valor constante, representado em **dp** [26]. Apesar de essa unidade de medida ser independente da densidade de pixéis da tela, não é independente do tamanho do ecrã. Isto é, para dispositivos com resoluções diferentes, essa margem parecerá maior ou menor, o que é inaceitável. Este problema surge novamente, por exemplo, quando se desenham as linhas que contornam componentes como a janela do Nétie ou o próprio balão de fala. Estas linhas apresentam uma certa grossura, dada também por um valor em **dp**.

Para solucionar este problema, utilizar-se-á uma unidade de dimensão diferente. Para isso, tirar-se-á partido de uma biblioteca que disponibiliza a unidade de dimensão **sdp** [27], que significa ***scallable density-independent pixel***. Como explica o desenvolvedor **Elhanan Mishraky**, esta alternativa ao **dp** adapta-se ao tamanho do ecrã, tal como se verifica na **figura 69**.

a) No emulador Nexus 5X (dp)

b) No emulador Nexus 5X (sdp)

c) No emulador Nexus 10 (dp)

d) No emulador Nexus 10 (sdp)

Figura 69 - Janela do Nétie em diferentes emuladores, utilizando dp e sdp

As **figuras 69 a)** e **69 b)** mostram a janela do Nétie utilizando **dp** como unidade de dimensão, e utilizando **sdp**, respetivamente. As **figuras 69 c)** e **69 d)** mostram o mesmo, mas para um tamanho de ecrã diferente. No primeiro caso, a diferença é nula, visto que o tamanho do ecrã é um dos mais comuns (e foi onde se realizaram a maioria dos testes). Contudo, na **figura 69 c)**, ao utilizar **dp** num emulador de um ***tablet*** (**Nexus 10**), verifica-se uma divergência enorme relativamente ao layout pretendido. Utilizando **sdp** (**figura 69 d)**), o layout apresenta-se bem mais consistente, garantindo um nível de compatibilidade aceitável para o Elderoid.

No caso do texto, em geral, utiliza-se **sp**[33] como medida de dimensão. Contudo, isso apresenta exatamente o mesmo problema, pelo que se utilizará uma biblioteca semelhante à descrita anteriormente. Além da biblioteca que disponibiliza **sdp** como medida de dimensão, **Elhanan Mishraky** desenvolveu uma biblioteca que introduz uma nova unidade de dimensão do texto, **ssp** [28]. Assim, também a dimensão do texto se adaptará ao tamanho do ecrã, tal como mostra o balão de fala da **figura 69 d)**, que tira partido também desta biblioteca.

Como foi referido ao longo do documento, de forma a atingir este nível de compatibilidade utilizaram-se imensas funcionalidades disponibilizadas pelo Android, desde a biblioteca ConstraintLayout, que permite posicionar componentes de forma relativa, até à funcionalidade de **AutoSize** disponível para as caixas de texto.

No contexto da compatibilidade, existe também um problema relacionado com a **fase de configuração**. Na **Configuração da App Inicial**, abordada no capítulo **2.5.4**, o utilizador será redirecionado para as opções do App Inicial padrão do dispositivo. No entanto, o método inicial utilizado para realziar esse redirecionamento, pode não funcionar corretamente em alguns dispositivos. A forma convencional de o fazer é lançar ao utilizador uma caixa de diálogo com os vários **Android Launchers** instalados no dispositivo, para que possa ser tomada essa decisão sem ser preciso navegar até às definições do **Android**. O problema é que essa decisão, em alguns dispositivos, não fica guardada, e a caixa de diálogo aparecerá sempre que o utilizador tentar voltar à página inicial. Noutros casos, aparece a opção "***Sempre***" e "***Apenas uma vez***", como mostra a **figura 70**. Nessa situação será necessário clicar uma vez em Elderoid, e, quando o utilizador voltar novamente à página inicial, o Elderoid já aparecerá em cima, com essas duas opções. Ao clicar "***Sempre***", a decisão fica guardada.

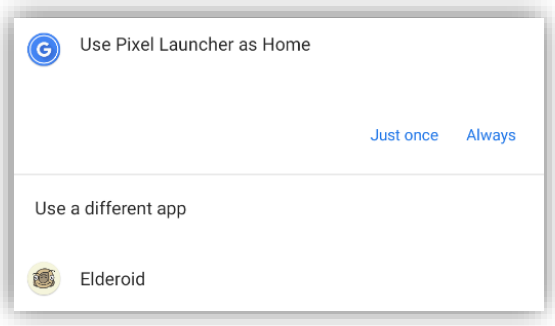

Figura 70 - Caixa de diálogo de configuração do App Inicial

Para dispositivos mais recentes, com versões do sistema operativo mais avançadas, esta tarefa é imensamente facilitada, pois existe maior suporte neste âmbito. Contudo, para dispositivos com sistema operativo abaixo da versão **21**, a única forma possível é a descrita pela **figura 70** ou recorrendo às definições do dispositivo manualmente (também não é possível redirecionar o utilizador para o menu das definições correto, pelo que terá de ser ele/a a encontrá-lo).

Dependendo da versão do sistema operativo que está a correr o aplicativo, é importante garantir que a tarefa a executar será compatível. No caso de o Elderoid necessita de recorrer a uma funcionalidade disponibilizada apenas a partir de uma determinada versão do sistema operativo, ter-se-á de escolher uma de duas opções:

- **Arranjar uma forma de atingir o mesmo objetivo, mas em versões mais antigas, sem acesso à funcionalidade em questão;**
- **Discartar a funcionalidade, visto que apenas será compatível com alguns dispositivos.**

[33] **Scale-independent Pixel –** Pixel independente da escala. Tem um comportamento semelhante a **dp**, mas utiliza-se para a dimensão das ***fonts*** de texto

Por exemplo, a funcionalidade da lanterna envolve algumas soluções programáticas que dependem da versão do sistema operativo. Para versões mais recentes, a tarefa de ativar e desativar a lanterna é relativamente fácil, dadas as bibliotecas e classes que apresentam suporte nesse âmbito. Para versões mais antigas, a história é outra. Existem soluções, pelo que foram devidamente implementadas, mas exigiram mais pesquisa e mais linhas de código.

Diversos blocos de código do aplicativo Elderoid dependem da versão do sistema operativo. Por isso, é comum verificarem-se instruções do seguinte calibre:

```
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.LOLLIPOP) {
    // Code Snippet 1
}
else {
    // Code Snippet 2
}
```

A constante **LOLLIPOP** representa uma versão do sistema operativo. Se o aplicativo estiver a correr num dispositivo Android com versão superior ou igual a **LOLLIPOP**, o bloco de código **1** será executado. Caso contrário, ser o bloco de código **2**.

Além disso, o termo compatibilidade, neste projeto, significa também compatibilidade com outras línguas. Como já foi referido, todo o texto utilizado terá sido colocado no ficheiro **strings.xml**, de forma a facilitar uma eventual tradução.

Por fim, o aplicativo não disponibilizará compatibilidade com rotação do ecrã, isto é, apenas funcionará em modo ***portrait***, e não em mode ***landscape*** (para não confundir o utilizador). Para garantir que o utilizador não rotaciona o ecrã enquanto navega no aplicativo, adicionar-se-á a seguinte linha a cada **activity**, no ficheiro **AndroidManifest.xml**:

```
android:screenOrientation="portrait"
```

3. Conclusões

O Elderoid apresenta características apelativas ao seu público alvo. A sua simplicidade, abstração e variadas funcionalidades são pontos muito importantes da sua natureza, e são as novidades que oferece ao mercado dos **Android Launchers**, principalmente em termos de flexibilidade e interação. Os aplicativos existentes, no âmbito da terceira idade, obrigam o utilizador, seja quem ele for, a ter acesso às mesmas funcionalidades. O Elderoid, acima de tudo, permitirá que o utilizador escolha de um conjunto de funcionalidades, aquilo que pretende fazer.

Contudo, a verdadeira essência do Elderoid reside no Nétie. A pequena mascote que estará sempre pronta para ajudar e acompanhar o utilizador nas suas façanhas virtuais, para que nunca se sinta sozinho ou desorientado.

Por outro lado, existem alguns aspetos em que o Elderoid, bem como os restantes Launchers adaptados à terceira idade, peca. Tomando como exemplo, a inserção de letras. O teclado padrão do sistema operativo Android não é o mais apropriado, pelo que se pode revelar confuso. Para resolver este problema, seria necessário desenvolver um teclado customizado. Outro exemplo, muito semelhante ao anterior, é a interface de uma chamada telefónica (tanto em chamada, como ao recebê-la). Esta interface varia de dispositivo para dispositivo, mas em alguns casos, o utilizador pode encontrar dificuldades em atender ou terminar chamadas. Infelizmente, não é possível substituir esta interface.

Por fim, espera-se que o Elderoid abra novas portas aos **Android Launchers** e à facilidade de acesso por parte de menos experientes e habilitados utilizadores.

4. Referências

[1] "Ocultar a barra de status, Desenvolvedores Android." https://developer.android.com/training/system-ui/status?hl=pt-br (accessed Mar. 02, 2021).

[2] A. Sinicki, "How to build a custom launcher in Android Studio - Part One - Android Authority," 2018. https://www.androidauthority.com/make-a-custom-android-launcher-837342-837342/ (accessed Mar. 02, 2021).

[3] G. Sims, "How to build your own custom Android ROM - Android Authority," 2016. https://www.androidauthority.com/build-custom-android-rom-720453/ (accessed Mar. 02, 2021).

[4] "Download Android Studio and SDK tools, Android Developers." https://developer.android.com/studio (accessed Mar. 02, 2021).

[5] "Entenda o ciclo de vida da atividade, Desenvolvedores Android." https://developer.android.com/guide/components/activities/activity-lifecycle?hl=pt-br (accessed Mar. 04, 2021).

[6] "Intents e filtros de intents, Desenvolvedores Android." https://developer.android.com/guide/components/intents-filters (accessed Apr. 10, 2021).

[7] "Criar uma IU responsiva com o ConstraintLayout, Desenvolvedores Android." https://developer.android.com/training/constraint-layout (accessed Mar. 30, 2021).

[8] "Salvar dados chave-valor, Desenvolvedores Android." https://developer.android.com/training/data-storage/shared-preferences (accessed Mar. 27, 2021).

[9] F. Bibullaev, "GitHub - farruhha/SimplePrefs: A simple preference management library for android," 2018. https://github.com/farruhha/SimplePrefs (accessed Mar. 27, 2021).

[10] "Doro mobile phones." https://www.doro.com/en-gb/products/mobile-phones/ (accessed Mar. 12, 2021).

[11] "BIG Launcher para idosos." https://biglauncher.com/pt/ (accessed Mar. 12, 2021).

[12] D. Gonçalves, "Nétie, um ciberneto para ciberavós," Caldas da Rainha, 2017.

[13] J. Paul, "Android - How to make digital clock display only hours and minutes - Stack Overflow," 2014. https://stackoverflow.com/questions/11143568/how-to-make-digital-clock-display-only-hours-and-minutes (accessed Apr. 02, 2021).

[14] "Criar listas dinâmicas com o RecyclerView, Desenvolvedores Android." https://developer.android.com/guide/topics/ui/layout/recyclerview (accessed Apr. 14, 2021).

[15] "Noun Project: Free Icons & Stock Photos for Everything." https://thenounproject.com/ (accessed Apr. 12, 2021).

[16] "Weather API - OpenWeatherMap." https://openweathermap.org/api (accessed Apr. 08, 2021).

[17] "Weather Conditions - OpenWeatherMap." https://openweathermap.org/weather-conditions (accessed Apr. 08, 2021).

[18] "Ícones vetoriais gratuitos – SVG, PSD, PNG, EPS e Icon Font – Milhares de ícones gratuitos." https://www.flaticon.com/br/ (accessed Apr. 09, 2021).

[19] "Postman, The Collaboration Platform for API Development." https://www.postman.com/ (accessed Apr. 09, 2021).

[20] "Enviar uma solicitação simples, Desenvolvedores Android." https://developer.android.com/training/volley/simple (accessed Apr. 12, 2021).

[21] "Acessar arquivos de mídia do armazenamento compartilhado." https://developer.android.com/training/data-storage/shared/media (accessed May 04, 2021).

[22] "GitHub - bumptech/glide: An image loading and caching library for Android focused on smooth scrolling." https://github.com/bumptech/glide (accessed May 01, 2021).

[23] "GitHub - Baseflow/PhotoView: Implementation of ImageView for Android that supports zooming, by various touch gestures." https://github.com/Baseflow/PhotoView (accessed May 03, 2021).

[24] "API Marketplace - Free Public & Open Rest APIs | RapidAPI." https://rapidapi.com/marketplace (accessed Jul. 13, 2021).

[25] "API Specs – API Reference." https://free-docs.newscatcherapi.com/#api-specs (accessed Jul. 13, 2021).

[26] "Compatibilidade com densidades de pixel diferentes, Desenvolvedores Android." https://developer.android.com/training/multiscreen/screendensities (accessed Apr. 22, 2021).

[27] E. Mishraky, "GitHub - intuit/sdp: An Android SDK that provides a new size unit - sdp (scalable dp). This size unit scales with the screen size.," 2020. https://github.com/intuit/sdp (accessed Apr. 19, 2021).

[28] E. Mishraky, "GitHub - intuit/ssp: Variant of sdp project based on the sp size unit.," 2020. https://github.com/intuit/ssp (accessed Apr. 19, 2021).