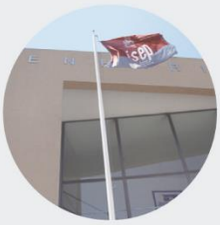
INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




Migração de aplicações móveis híbridas para Android nativo

HUGO FILIPE FERREIRA PEREIRA
Outubro de 2015

POLITÉCNICO
DO PORTO

Migração de aplicações móveis híbridas para Android nativo

Hugo Filipe Ferreira Pereira

Dissertação para obtenção do Grau de Mestre em Engenharia Informática, Área de Especialização em Arquiteturas, Sistemas e Redes

Orientador: Paulo Manuel Baltarejo De Sousa

Júri:

Presidente:

Paulo Alexandre Fangueiro Oliveira Maio

Vogais:

Luis Miguel Moreira Lino Ferreira

Paulo Manuel Baltarejo De Sousa

Porto, Outubro 2015

Resumo

Durante os últimos anos tem-se assistido a um crescimento exponencial da utilização dos dispositivos móveis. Atualmente, a presença destes dispositivos está tão consolidada na sociedade em que nos encontramos que é praticamente impossível estar num espaço público sem encontrar uma pessoa a utilizar um exemplar deste tipo de dispositivos, na forma de um *Smartphone* ou um *Tablet*. Nesse sentido, um grande número de organizações de desenvolvimento de *software* tem vindo a apostar na criação de aplicações para servir este mercado com enorme potencial.

De forma a iniciar o desenvolvimento aplicacional nesta área, cabe às organizações escolherem a metodologia de desenvolvimento que mais se adapta às suas necessidades e ao contexto que desejam explorar. Por vezes, a abordagem adotada acaba por ser tornar inadaptada a longo prazo, podendo as aplicações começar a apresentar níveis de desempenho e problemas comportamentais indesejados, de acordo com as alterações realizadas e as necessidades evidenciadas no seu processo de evolução, sendo necessário explorar outras alternativas metodológicas.

No contexto apresentado surge a temática da presente dissertação, que se propõe à criação de um plano de migração tecnológica genérico direcionado a um conjunto aplicações móveis em estudo, desenvolvidas através de uma metodologia híbrida, adaptada ao desenvolvimento de aplicações compatíveis com as diversas plataformas móveis. O plano devia então ser construído com o intuito de efetuar a migração das aplicações para uma metodologia nativa, adaptada ao desenvolvimento de aplicações móveis específicas a uma determinada plataforma móvel, que no presente caso seria sob a plataforma Android.

No decorrer do trabalho realizado foram desenvolvidos mecanismos e componentes genéricos que permitiram implementar um módulo de migração responsável por construir um projeto nativo base de Android através de um projeto híbrido, construído segundo as suas tecnologias específicas.

Finalizada a implementação, foram realizados testes comparativos entre a aplicação alvo de migração e o protótipo nativo conseguido e foi apresentada uma série de resultados que permitiram concluir a adequação da metodologia nativa para ultrapassar a problemática evidenciada. Além disso, foi também possível concluir que o sistema de migração genérico resultante do trabalho desta dissertação permite reduzir significativamente o tempo de desenvolvimento necessário em migrações de outras aplicações que se enquadrem na mesma problemática.

Palavras-chave: metodologias de desenvolvimento móveis, aplicações móveis híbridas, aplicações móveis nativas, plano de migração, desempenho e comportamento aplicacional.

Abstract

During the past few years it has been seen an exponential growth in the use of mobile devices. Currently, the presence of these devices is so integrated in the society in which we find that it is nearly impossible to be in a public place without finding a person using a mobile device such as a Smartphone or a Tablet. In this sense, a large number of software development organizations have been focusing on developing applications to serve this huge market.

In order to start the application development in such area, it is up to the organizations to choose the development methodology that best suits their needs and their target market context. Sometimes the chosen approach turns out to be unsuited over time, due to changed needs and requirements which can influence the application's performance and introduce undesirable application behavior, being essential to explore other methodological alternatives.

In the presented context emerges the theme of this dissertation, which proposes the creation of a generic technology migration plan applied to the mobile applications under study, developed through a hybrid methodology, which is adapted to the development of applications compatible with the different mobile platforms. The plan should then be constructed in order to execute the applications migration for a native methodology, which is adapted to the development of mobile applications under a specific mobile platform, which in this case would be under the Android platform.

The plan must then be constructed in order to effect the migration of applications for a native method, adapted to the development of mobile applications specific to a particular mobile platform, which in this case would be under the Android platform.

During the development process, mechanisms and generic components were created, enabling the development of a migration module responsible for building a base native Android project through a hybrid project, developed according to some specific technologies.

Completed the implementation process, a set of comparative tests between the migration target application and the resulted native prototype were performed, allowing to conclude the adequacy of the native methodology to overcome the highlighted problems. Moreover, it was also possible to conclude that the resulting generic migration system of this dissertation allows to significantly shortening the development time required in other similar application's migration process.

Keywords: mobile development methodologies, hybrid mobile applications, native mobile applications, migration plan, application performance and behavior.

Agradecimentos

Terminado o longo processo de desenvolvimento da dissertação, é importante deixar uma palavra de consideração e agradecimento a algumas pessoas sem as quais a realização deste trabalho não teria sido possível.

Primeiramente, à minha família e amigos pelo apoio fornecido e em especial à Tânia, por toda a compreensão e incentivo demonstrados, especialmente nos momentos mais complicados deste processo.

Ao meu orientador, o Professor Paulo Baltarejo, por me ter aceitado como seu orientando apesar do pedido tardio e por me ter apoiado na decisão do melhor caminho a seguir.

Por fim, à ANO, pelo acompanhamento do processo de desenvolvimento da dissertação, apesar de o mesmo se realizar, na sua grande maioria, no âmbito pós-laboral.

A todos os visados, um muito obrigado.

Índice

Lista de Figuras

Lista de Tabelas

Lista de Código Fonte

Acrónimos e Símbolos

Lista de Acrónimos

API	*Application Programming Interface*
APK	*Android Application Package*
CRUD	*Create, Read, Update and Delete*
CSS3	*Cascading Style Sheets 3*
HTML5	*Hyper Text Markup Language 5*
HTTP	*Hypertext Transfer Protocol*
IDE	*Integrated Development Environment*
JSON	*JavaScript Object Notation*
MVC	*Model View Controller Pattern*
SOAP	*Simple Object Access Protocol*
SQL	*Structured Query Language*
STC	*Software* de Trânsito e Contraordenações
XML	*Extensible Markup Language*

1. Introdução

Parece longínquo o tempo em que o telemóvel era um simples aparelho que tinha como único propósito a realização de chamadas de voz e o envio de mensagens de texto. Se voltarmos atrás na História verificamos que afinal foi apenas há pouco mais de 40 anos que surgiu o primeiro telemóvel.

Decorria o mês de Abril de 1973 quando um funcionário da organização Motorola, de seu nome Martin Cooper, desenvolveu o primeiro protótipo funcional de um telemóvel. Apenas 10 anos mais tarde este protótipo se traduziu em resultados comerciais uma vez que só em 1983 foi lançado o primeiro telemóvel oficial, designado Motorola DynaTAC 8000X [Richard Goodwin, 2015].

De volta à atualidade, percebe-se que nos 42 anos que passaram a evolução tecnológica da área de dispositivos móveis foi vertiginosa, não existindo perspetivas de abrandar dadas as constantes evoluções anunciadas de ano para ano. Tal foi a evolução, especialmente nos últimos anos, que a ideia de um simples aparelho utilizado pontualmente para efetuar chamadas parece o conceito de um filme antigo.

Hoje em dia é também comum deixar de ouvir falar no termo telemóvel mas sim *Smartphones* ou *Tablets*, que são os tipos de dispositivos móveis de utilização mais comum nos tempos que correm. Estes dispositivos permitem nos dias de hoje uma panóplia de funcionalidades tais como o acesso à *Internet*, a consulta de correio eletrónico, a realização de videochamadas, a visualização de um filme em alta definição ou o acesso às mais variadas redes sociais. Além do panorama pessoal, estes dispositivos são também cada vez mais utilizados pelas organizações para a execução das suas atividades profissionais. Exemplos disso são o registo de leituras dos contadores de eletricidade e de água no caso dos funcionários das organizações prestadoras desses serviços ou o registo de contraordenações por parte de agentes da autoridade pública.

Com um rol de possibilidades tão vasto e um mercado cada vez mais amplo e mais exigente, o investimento na criação de aplicações móveis de qualidade para os mais variados fins tem sido uma realidade cada vez mais presente na grande maioria das organizações de desenvolvimento de *software*. Com uma aposta tão elevada poderia existir o risco de saturação do mercado mas tal não se verifica, com tantas vertentes possíveis de ação.

O desenvolvimento de aplicações móveis não é um processo que possa ser encarado com leveza, especialmente numa fase em que o nível de exigência dos utilizadores é extremamente alto tendo em conta as capacidades técnicas dos componentes que os seus dispositivos apresentam. Para uma aplicação conseguir ter uma alta percentagem de aceitação do mercado deve apresentar um nível elevado de qualidade. Para isso a aplicação deve ter um comportamento rápido e fluído e um visual apelativo, proporcionar uma experiência de utilização simples e útil, ser adaptável à grande quantidade de dispositivos e às suas especificações técnicas, ser confiável e garantir a segurança dos dados utilizados e ser atualizada de forma regular. Além destes cuidados de desenvolvimento existe também outra vertente que as organizações devem

ter em especial atenção, tais como a promoção das aplicações através das redes sociais e outros meios e a capacidade de acompanhar constantemente a reação de mercado para conseguir agir consoante o sucesso ou insucesso conseguido. É através da conjugação destes fatores que as organizações poderão conseguir ser bem-sucedidas a longo prazo no mercado de aplicações móveis.

Uma das principais preocupações que uma organização deve ter ao desenvolver as suas aplicações é a definição da metodologia de desenvolvimento de aplicações móveis que mais se adequa ao seu contexto. Esta escolha nem sempre é fácil uma vez que existe uma combinação de variáveis que devem ser bem analisadas e conjugadas. Um exemplo de que este processo não é de todo fácil é o caso de organizações mundialmente reconhecidas como o caso do Facebook e Linkedin terem alterado a metodologia de desenvolvimento numa fase que as suas aplicações já se encontravam no mercado e se aperceberam com o tempo que a sua primeira abordagem não apresentava os resultados esperados [Mitch Pronschinske, 2014].

É precisamente sob a inadequação de uma metodologia de desenvolvimento de aplicações móveis e na procura de uma solução que simplifique a migração metodológica de um conjunto de aplicações alvo já implementadas no mercado que o trabalho da presente dissertação assenta.

1.1 Motivação

A presente dissertação surge da necessidade da organização ANO – Sistemas de Informática e Serviços solucionar os problemas das suas aplicações móveis, originados pela metodologia de desenvolvimento adotada.

Desenvolvidas sob a metodologia híbrida com o propósito de garantir a compatibilidade com os mais diversos dispositivos móveis e respetivos sistemas operativos, as aplicações começaram a apresentar níveis de desempenho longes do aceitável consoante a complexidade funcional adicionada ao longo do tempo.

Após as diversas intervenções e tentativas de solucionamento dessa problemática sem conseguir os resultados desejados, foi então decidido realizar a mudança estratégica de desenvolvimento das suas aplicações móveis para uma abordagem nativa e direcionada à plataforma Android, a mais utilizada pelo seu mercado alvo.

No entanto, surgiram duas questões fundamentais com esta decisão. Seria esta mudança metodológica eficaz na solução dos problemas evidenciados pelas aplicações atuais? Como se poderá migrar tantas aplicações com um custo de desenvolvimento aceitável?

É com o propósito de responder de forma eficaz a estas questões que a presente dissertação se apresenta. Para isso, deverá ser implementado o processo de migração a uma das suas aplicações e no final do trabalho realizado deverá ser possível analisar um comparativo de desempenho aplicacional segundo a metodologia de desenvolvimento escolhida. Além disso,

deverá ser também possível definir um plano de migração genérico que possa ser aplicado nas restantes aplicações e assim permitir a redução do custo de desenvolvimento em cada uma delas.

1.2 Contribuições

O trabalho desenvolvido nesta dissertação terá as seguintes contribuições:

- Estado da arte relativo às diferentes metodologias de desenvolvimento de aplicações móveis e principais tecnologias relacionadas;
- Migração de uma aplicação móvel híbrida desenvolvida com a combinação tecnológica de Sencha Touch e Apache Cordova para uma aplicação móvel nativa desenvolvida em Android e capaz de apresentar melhor desempenho e fluidez comportamental que a sua congénere.
- Plano de migração capaz de descrever as linhas gerais do processo de implementação dos módulos comuns às aplicações alvo e que possa ser aplicado no contexto de migração de qualquer uma delas;
- Módulo de migração responsável por criar um projeto Android base através dos automatismos, mecanismos e componentes genéricos desenvolvidos no decorrer do processo de implementação;

1.3 Estrutura do Documento

A presente dissertação é dividida em sete capítulos:

- Capítulo 1 – Apresenta de forma breve os conceitos base relacionados com a dissertação desenvolvida.
- Capítulo 2 – Realiza um estudo do meio envolvente do tema, sendo apresentadas as diferentes metodologias de desenvolvimento de aplicações móveis e sendo descrita a problemática que deu origem à dissertação e os seus conceitos e tecnologias relacionadas.
- Capítulo 3 – Descreve a análise dos requisitos para o cumprimento dos objetivos e a apresentação da estrutura alvo de migração.
- Capítulo 4 – Apresenta o ambiente de desenvolvimento utilizado e detalha o processo de implementação da solução proposta.
- Capítulo 5 – Apresenta os resultados obtidos pela solução desenvolvida através de testes de desempenho e análise de outras métricas.
- Capítulo 6 – Apresenta as conclusões da interpretação dos resultados obtidos em relação aos objetivos propostos e realiza uma sugestão de trabalho futuro que poderá ser desenvolvido no seguimento da solução conseguida no âmbito da presente dissertação.
- Capítulo 7 – Apresenta a lista de referências utilizadas no decorrer da dissertação.

2. Estado da Arte

Ao iniciar a elaboração de um projeto é necessário estudar o ambiente que rodeia o tema a abordar, aprofundando os seus conceitos inerentes e melhores práticas já implementadas. É a existência deste trabalho de investigação prévio acerca da área de ação que permite a um autor conseguir desenvolver uma solução adequada à problemática a que se propõe. Apenas é possível efetuar um trabalho de qualidade quando se conhece bem a realidade onde irá atuar e as soluções já existentes, aproveitando as boas práticas que delas resultam e evitando cair nos mesmos erros com que outros já se depararam. Este processo de investigação inicial é intitulado de "Observar o Estado da Arte".

Neste capítulo é apresentado um estudo sobre os principais temas abordados no âmbito desta dissertação, dando maior foco às metodologias de desenvolvimento de aplicações móveis existentes e suas principais tecnologias. É também realizada uma apresentação à problemática do projeto de forma a contextualizar o panorama que deu origem a esta dissertação.

2.1 Desenvolvimento de aplicações móveis

Num mundo em que o progresso tecnológico se encontra com um ritmo frenético, torna-se evidente o nível elevado de inovação e a procura constante de ultrapassar barreiras em diversas áreas como por exemplo na saúde, na área militar, nos transportes ou também nas telecomunicações.

Esse efeito é também visível no quotidiano das sociedades mais desenvolvidas, sendo que a forma de viver das pessoas é já dependente de toda esta evolução tecnológica.

Um dos exemplos mais flagrantes dessa dependência reside na utilização dos dispositivos móveis, que tem vindo a crescer exponencialmente, deixando de ser não só um simples equipamento de comunicação à distância recorrendo a chamadas telefónicas ou mensagens escritas, mas transformando-se num aparelho com presença nas mais diversas atividades do quotidiano dos seus utilizadores, como por exemplo, consultar as suas mensagens de correio eletrónico ou as notícias do dia durante a viagem para o trabalho nos transportes públicos, ouvir música ao praticar desporto ou efetuar uma videochamada de forma instantânea para o outro lado do globo.

Nesta gama de dispositivos móveis encontram-se os *Smartphones*, *Tablets* e os recentes *Smartwatch* e *SmartTV*.

De forma a ilustrar o aumento exponencial da utilização dos dispositivos é apresentado na Figura 1 um gráfico com base nos dados disponibilizados pela organização StatCounter Global Stats entre Janeiro de 2012 e Março de 2015 [StatCounter, 2015].

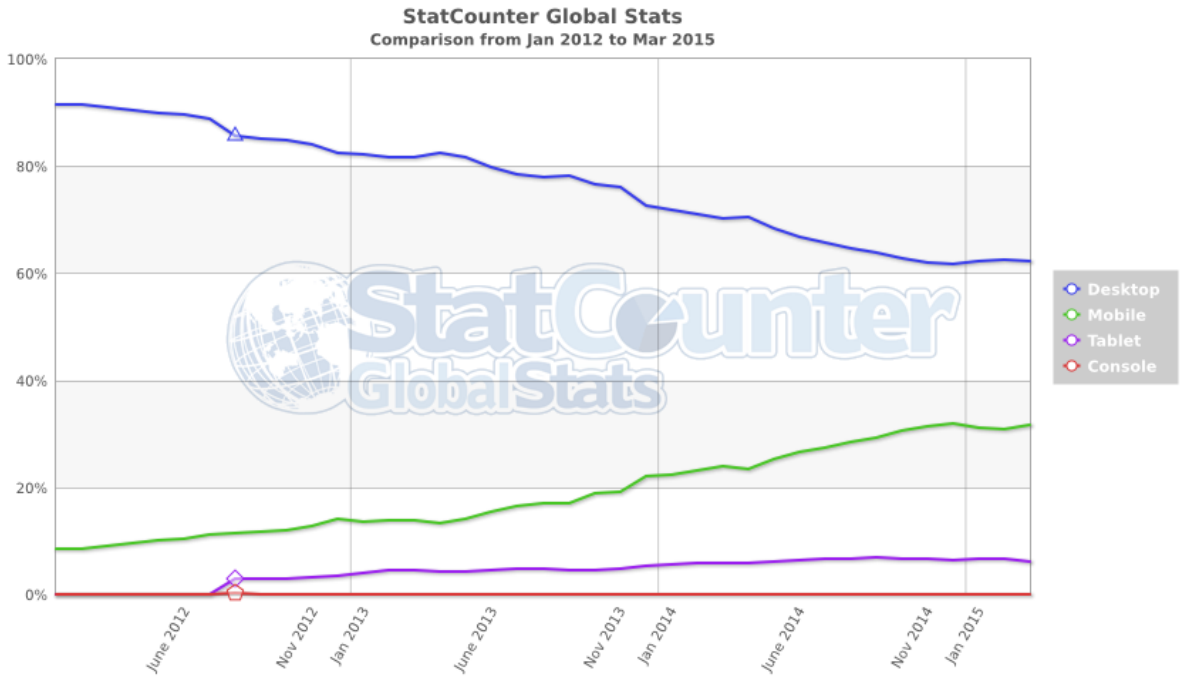


Figura 1- Evolução dos tipos de dispositivos tecnológicos [StatCounter, 2015]

Após uma análise dos dados da Figura 1 é possível verificar que nos últimos três anos a utilização de dispositivos móveis a nível global sofreu um aumento substancial, sendo mais significativo na área dos *Smartphones*, sensivelmente entre os 8% e 9% no início de 2012 e ultrapassando os 30% no primeiro trimestre de 2015. Em sentido inverso, como consequência, encontra-se uma quebra na utilização dos computadores pessoais, que têm dominado o mercado ao longo dos anos, passando de cerca de 90% até pouco mais de 60% no mesmo período de três anos.

A evolução referida anteriormente cria um mercado cada vez mais necessitado e exigente, sendo essa necessidade traduzida em aplicações inovadoras para dispositivos móveis e que facilitem cada vez mais o quotidiano dos seus utilizadores. Como consequência, existe uma grande oportunidade de negócio para as organizações da área tecnológica, o que tem criado uma elevada tendência para alterar o seu foco para um forte investimento no desenvolvimento de aplicações móveis.

2.2 Metodologias de desenvolvimento

Um dos principais requisitos a abordar no mundo de desenvolvimento de aplicações móveis é perceber as diferentes metodologias existentes e qual se poderá enquadrar nas suas necessidades e objetivos. Algumas das variáveis mais comuns a ter em conta na estratégia a definir são, por exemplo, o mercado alvo, o custo de desenvolvimento, as competências técnicas da equipa de desenvolvimento, as necessidades tecnológicas que advêm das funcionalidades a implementar, e a necessidade de interoperabilidade dos sistemas. Partindo do pressuposto que não existe uma solução perfeita para todas as necessidades, torna-se

fundamental conjugar as vantagens e desvantagens de cada abordagem de acordo com as variáveis identificadas de forma a perceber qual a mais indicada para a realidade a encarar [Pietro Saccomani, 2012].

Atualmente, as principais metodologias de desenvolvimento são agrupadas de acordo com três tipos: nativas, web e híbridas. Conhecer as principais características assim como as vantagens e desvantagens de cada uma das abordagens é o primeiro passo para uma decisão cuidada e consciente da mais apropriada aos requisitos.

2.2.1 Metodologia Nativa

Figura 2 – Processo de desenvolvimento da metodologia nativa [Mike Jacobs, 2011]

O processo de desenvolvimento de aplicações nativas (Figura 2) consiste numa abordagem tradicional de desenvolvimento segundo as linguagens e regras específicas de cada plataforma móvel (Figura 3). Exemplificando, para sistemas operativos Android é utilizada a linguagem de programação Java, para sistemas iOS a linguagem de programação Objective C e, para a sistemas Windows, as linguagens de programação nativas da Microsoft, como por exemplo C#.

Figura 3 - Principais sistemas operativos móveis

Além de se basear exclusivamente nas linguagens específicas, esta abordagem utiliza as bibliotecas e padrões de desenvolvimento específicos de cada plataforma, tornando-se numa das maiores vantagens, pois, quando corretamente implementada, as aplicações conseguem

retirar o máximo proveito dos equipamentos a que se destinam, conseguindo um nível de eficiência e desempenho superior às restantes metodologias [Mitch Pronshinske, 2014].

Por consequência direta advém a principal limitação desta abordagem, que está relacionada com o facto das aplicações apenas poderem ser executadas em dispositivos com o sistema operativo para o qual foi criada. Como tal, na perspetiva de mercado cria-se aqui uma restrição e, caso se pretenda lançar uma aplicação para as diversas plataformas, o custo de desenvolvimento é muito elevado, sendo necessário desenvolver a mesma aplicação em cada linguagem nativa que for pretendida [Pietro Saccomani, 2012].

O custo de desenvolvimento pode ser analisado segundo a combinações das seguintes variáveis:

- Tempo de desenvolvimento necessário na criação das aplicações, uma vez que dadas as diferenças de cada plataforma, o desenvolvimento em cada uma tem de ser feita praticamente de raiz;
- Constituição e especialização técnica das equipas de desenvolvimento. Sendo que cada plataforma apresenta a sua linguagem de programação específica e plataformas de desenvolvimento, tornando-se muito difícil ter uma única equipa especializada em todas as plataformas e responsável por gerir todas as aplicações. Uma das principais estratégias passa por definir equipas distintas para desenvolvimento e manutenção da aplicação em cada plataforma de forma a retirar o maior aproveitamento da especialização de cada uma;
- Complexidade de manutenção da mesma aplicação para cada plataforma. O conceito passa por ter uma aplicação semelhante, utilizando o mesmo nome comercial e as mesmas funcionalidades, mas com um nível de desenvolvimento e manutenção completamente distintos por cada plataforma para a qual se quer que seja compatível. Sempre que for necessário efetuar uma correção ou o lançamento de uma nova versão com mais funcionalidades tem de se repetir esse processo para cada plataforma.

Em suma, na abordagem de desenvolvimento para multiplataformas, a abordagem nativa apresenta custos significativamente elevados.

Pelo contrário, numa abordagem de desenvolvimento para apenas uma plataforma, esta abordagem apresenta diversas vantagens, como é o caso do maior aproveitamento das capacidades dos equipamentos e do próprio sistema operativo, como já foi referido anteriormente. Além dessa vantagem, esta abordagem permite também retirar proveito das constantes atualizações à plataforma, assim como a utilização dos seus ambientes de desenvolvimento integrado (IDE), que disponibilizam as melhores ferramentas e componentes para desenvolver e integrar com a mesma [Venkata Kiran, 2013].

2.2.2 Metodologia *Web*

Figura 4 - Processo de desenvolvimento da metodologia *web* [Mike Jacobs, 2011]

A metodologia *web* consiste, ao contrário da metodologia nativa, no desenvolvimento de aplicações *web* executadas na sua grande maioria através de um navegador de Internet, já incorporado ou instalável em qualquer dispositivo móvel recente. Alguns dos navegadores de Internet mais comuns para dispositivos móveis são o Google Chrome, Mozilla Firefox, Opera ou Safari. Na Figura 4 é ilustrada a representação do processo de desenvolvimento desta metodologia.

Tendo como base do seu desenvolvimento a utilização de *Hyper Text Markup Language 5* (HTML5), *Cascading Style Sheets 3* (CSS3) e JavaScript, tecnologias padrão (Figura 5) para a construção da camada de apresentação de aplicações em ambientes *web*, sejam eles móveis ou não, esta abordagem acaba por se tornar vantajosa para as equipas de desenvolvimento que já tenham experiência nesta vertente, facilmente se adaptando às principais plataformas existentes para criação de aplicações móveis *web*.

Figura 5 – Algumas plataformas de desenvolvimento de aplicações móveis *web*

Uma das vantagens da abordagem *web* é assim a capacidade de desenvolver uma única aplicação a servir as diferentes plataformas móveis reduzindo substancialmente o custo de desenvolvimento ao comparar com a metodologia nativa. É no entanto importante que as equipas de desenvolvimento testem devidamente as aplicações no maior número de ambientes e plataformas possíveis de forma a conseguir um nível de compatibilidade alto, uma vez que existem diferenças de requisitos ao nível da apresentação dos componentes visuais tanto nos diversos sistemas operativos móveis como também nos seus navegadores de internet. Conjuntamente com esses fatores, tal como acontece em todas as metodologias de desenvolvimento de aplicações móveis, é também importante ter em conta a compatibilidade com os diferentes tipos de dispositivos alvo, desde a resolução dos seus ecrãs às diferentes versões do seu sistema operativo.

Outros pontos positivos apresentados pela utilização desta metodologia passam pela facilidade da disponibilização das aplicações e a facilidade de manutenção das suas versões, sendo possível gerir todo o processo ao nível de um servidor *web*, ao invés de existir a necessidade de cada utilizador necessitar de descarregar e instalar uma aplicação ou respetivas atualizações no seu equipamento. [Rob Gravelle, 2015].

No entanto, apesar das vantagens apresentadas, as aplicações com base na metodologia *web* apresentam algumas limitações funcionais, nomeadamente ao nível da dificuldade em utilizar determinados componentes nativos dos dispositivos para a execução correta de tarefas mais complexas e ao nível da necessidade de uma conexão de internet ativa para o acesso às mesmas. Como consequência da obrigação da utilização de uma conexão à internet, existe uma dependência direta da qualidade da ligação com o desempenho conseguido na utilização da aplicação, podendo existir quebras constantes que afetem a experiência de interação dos utilizadores. Além do desempenho e estabilidade da aplicação, são de salientar também outras consequências, tais como o consumo de tráfego ao utilizar um pacote de dados de um fornecedor de serviços de telecomunicações e a dificuldade em guardar dados locais de forma segura no dispositivo.

Comparativamente com a metodologia nativa, a abordagem *web* torna-se mais apropriada para o desenvolvimento de aplicações simples e multiplataforma, no entanto apresenta diversas limitações e desvantagens que se salientam de acordo com o aumentar do nível de complexidade das funcionalidades desejadas.

2.2.3 Metodologia Híbrida

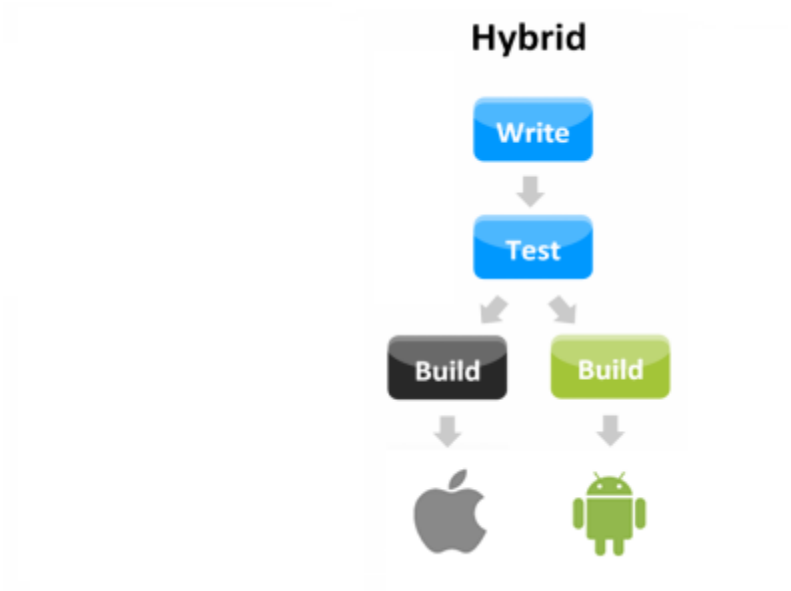


Figura 6 - Processo de desenvolvimento da metodologia híbrida

A metodologia híbrida (Figura 6) apresenta-se como uma conjugação das duas metodologias apresentadas anteriormente, numa tentativa de aproveitar e capitalizar o melhor entre as vantagens e desvantagens de cada uma.

Assenta em uma base tecnológica multiplataforma, recorrendo a linguagens como HTML5, CSS3 e JavaScript, como na metodologia *web*, no entanto, na perspetiva de conseguir uma melhor integração com os componentes do dispositivo, especialmente ao nível do seu *Hardware* e da sua capacidade de processamento, é utilizado também código nativo.

- Na grande maioria dos casos, o desenvolvimento das aplicações seguindo esta metodologia assenta em dois momentos distintos:
- Desenvolvimento da componente visual e todo o código-fonte aplicacional recorrendo a plataformas de desenvolvimento *web*, como por exemplo Angular.js [Angular, 2015], Sencha Touch [Sencha Touch, 2015], Kendo UI [Kendo, 2015] e jQueryMobile [jQueryMobile, 2015];
- Utilização de plataformas híbridas, como são os casos do PhoneGap [PhoneGap, 2015], Apache Cordova [Cordova, 2015] ou Ionic [Ionic, 2015], para compilação e encapsulamento do código-fonte produzido em uma aplicação nativa, tendo cada um deles um conjunto de bibliotecas que facilitam a interação nativa com os componentes de cada plataforma.

Uma das vantagens desta abordagem consiste no menor custo de desenvolvimento e manutenção das aplicações relativamente à metodologia nativa uma vez que apenas se torna necessário reescrever a parte de código nativo para adaptar a cada plataforma móvel, sendo o restante, a maior percentagem, reutilizável em qualquer uma delas dada a sua origem *web*.

Disponibilizando uma vertente de interligação nativa, com o apoio das bibliotecas das plataformas híbridas, esta metodologia ultrapassa algumas das limitações da abordagem *web*, como é o caso de permitir uma utilização *Offline*, sem necessitar de uma ligação à internet, e uma gestão segura dos dados do utilizador a nível local, ficando encriptados no próprio dispositivo, tal como acontece nas aplicações puramente nativas.

É também uma mais-valia desta metodologia a possibilidade de distribuição das suas aplicações no mesmo formato das aplicações nativas, utilizando os mesmos canais oficiais de cada plataforma móvel para o efeito, como por exemplo o Google Play da Google para sistemas operativos Android, a App Store da Apple para sistemas iOS ou a Windows Store da Microsoft para plataformas Windows [Patrick Rudolph, 2014].

Apesar de encapsuladas e distribuídas como aplicações nativas, as aplicações híbridas apresentam limitações relativamente às primeiras. O desempenho atingido pelas aplicações não consegue atingir um nível tão elevado como pelas aplicações nativas, tendendo a piorar consoante o grau de complexidade das funções desejadas. Outra limitação é a dependência das plataformas híbridas utilizadas na interligação com a componente nativa dos dispositivos respetivos sistemas operativos, pois nem todas as funcionalidades são asseguradas pelas suas bibliotecas.

Finalmente, importa ainda referir que em relação às outras abordagens, sendo esta uma metodologia que não desenvolve segundo os requisitos específicos de cada plataforma, o custo de desenvolvimento pode também aumentar substancialmente consoante a aproximação desejada a uma vertente nativa, pois a customização das interfaces para preencher os requisitos utilizando esta abordagem pode ser bastante complexa.

Apesar de todas as vantagens referidas, destacando-se a possibilidade de desenvolver uma aplicação multiplataforma a partir do mesmo código-fonte base permitindo atingir índices de desempenho e robustez elevados, existem ainda algumas sérias limitações nesta abordagem que não podem ser desconsideradas, mesmo que tenham sido implementados progressos ao longo dos últimos anos [Patrick Rudolph, 2014].

2.2.4 Sumário

Todas as metodologias abordadas apresentam vantagens e desvantagens, não sendo possível idealizar uma solução universal para o desenvolvimento de toda e qualquer aplicação móvel. A decisão para qual a metodologia a adotar deve ser ponderada de acordo com o tipo de aplicação pretendida, os seus requisitos técnicos e o mercado alvo desejado, sem esquecer os custos e tempos de desenvolvimento necessários. Todos estes fatores são apenas um exemplo entre muitos que podem e devem ser tidos em conta na escolha da abordagem a utilizar pelas organizações. O conceito de melhor metodologia existente no mundo tecnológico está longe de ser encontrado, pelo que o objetivo de cada organização deve pesar os fatores que mais se adequam à sua realidade para conseguir chegar a um resultado satisfatório.

Por exemplo, na perspetiva de desenvolvimento de aplicações direcionadas para um mercado focado numa plataforma móvel específica, a solução para apresentar melhores resultados, tanto no desempenho e fluidez como no aproveitamento dos recursos de *Hardware* e sistema operativos dos dispositivos, poderia ser baseada na metodologia nativa, no entanto, e mesmo tendo em conta os mesmos critérios de desempenho e estabilidade, se poderia aplicar a mesma metodologia a uma abordagem multiplataforma, sendo que neste âmbito, os custos de desenvolvimento e manutenção das aplicações seriam extremamente mais elevados. Se a organização dispusesse do tempo e recursos necessários para assumir o custo, então, o caminho nativo poderia até continuar a ser o mais apropriado para alcançar a máxima qualidade possível, caso contrário, já se deveriam analisar seriamente as outras opções, relacionando mais uma vez os pontos fortes e fracos de cada uma com a escalabilidade e complexidade desejadas.

A importância de uma reflexão cuidada na escolha da solução apropriada por parte das organizações é fundamental, pois caso contrário, poderá existir o risco da solução implementada não apresentar o resultado esperado a médio-longo prazo e o custo de realizar otimizações ou até reverter para uma nova abordagem pode ser alto. Um exemplo disso são o Facebook e o Linkedin, que na vertente móvel apostaram primeiramente numa aplicação híbrida, revertendo mais tarde para uma aplicação nativa, uma vez que à medida que as necessidades foram surgindo com o tempo e a complexidade aumentando, o desempenho diminuiu consideravelmente [Mitch Pronschinske, 2014].

É precisamente no âmbito de uma escolha apropriada de uma metodologia de desenvolvimento de aplicações móveis que surge a problemática a ser explorada nesta dissertação.

2.3 Apresentação da problemática

De forma a detalhar a motivação principal que originou o desenvolvimento desta dissertação, é apresentada de seguida uma contextualização da organização, da metodologia e tecnologias utilizadas no desenvolvimento das aplicações móveis assim como o estado atual da sua implementação.

2.3.1 A organização

Figura 7 – Logotipo da ANO Software

A ANO – Sistemas de Informática e Serviços, Lda. (Figura 7) foi fundada em Abril de 1994. A companhia tem sede na cidade do Porto e escritórios em São Paulo, no Brasil e a sua atividade

foca-se na conceção, desenvolvimento e comercialização de *Software*, e respetivos serviços de implementação e consultoria.

A organização tem como missão:

- Antecipar as necessidades dos seus clientes e transformá-las em resultados;
- Estabelecer um relacionamento de confiança com os seus clientes;
- Investir na melhoria contínua das suas competências e na qualidade dos seus serviços;
- Apostar na criação de um ambiente de satisfação e na valorização pessoal dos colaboradores;
- Ser uma referência na sua área de atuação.

O seu mercado principal é o setor público, onde realiza, aproximadamente, 70% do seu volume de negócio, representando o setor privado os restantes 30%, distribuídos entre grandes e médias empresas, advogados, bancos, seguradoras, serviços jurídicos, etc.

O desenvolvimento aplicacional da ANO é composto por oito principais áreas de negócio:

- Gestão Documental e Processual;
- Governo Eletrónico;
- Contratação Pública Eletrónica;
- Contraordenações e Multas de Trânsito;
- Gestão e Faturação;
- Monitorização do Território;
- Transcrição de Voz para Texto;
- *Outsourcing*.

Na perspetiva de acompanhamento da evolução das tecnologias e das necessidades do mercado, a ANO tem apostado na vertente móvel, tendo já algumas aplicações desenvolvidas e a serem atualmente utilizadas pelos seus clientes, entre as quais se destaca a aplicação STC (*Software* de Trânsito e Contraordenações), enquadrada na área de Contraordenações e Multas de Trânsito.

2.3.2 A metodologia

Como estratégia de desenvolvimento das aplicações móveis, a ANO tem apostado na metodologia híbrida.

A opção recaiu no fator híbrido devido principalmente à vantagem de permitir a interoperabilidade com os diversos sistemas operativos móveis, possibilitando, a partir do mesmo código-fonte compilar as suas soluções para Android, IOS e Windows, abrangendo assim as várias possibilidades de mercado com um menor custo de desenvolvimento, tanto em tempo necessário como recursos humanos associados. Juntando a estes fatores o facto de as suas aplicações terem, inicialmente, requisitos e funcionalidades simples, foi decidido, por todas as vantagens que apresentava, que a abordagem híbrida seria a mais indicada, sendo a que tem sido utilizada até ao momento atual no desenvolvimento das suas aplicações.

2.3.3 As tecnologias

Com o objetivo de aplicar uma abordagem híbrida, a escolha das tecnologias para o desenvolvimento de aplicações móveis da organização tem vindo a recair na combinação entre Sencha Touch [Sencha Touch, 2015] para a implementação da componente *web* e lógica das aplicações, e PhoneGap [PhoneGap, 2015] e Apache Cordova [Cordova, 2015], para a adaptação aos sistemas e componentes nativos.

2.3.3.1 Sencha Touch

Figura 8 - Sencha Touch [Sencha Touch, 2015]

Sencha Touch (Figura 8) é uma plataforma de desenvolvimento de aplicações móveis criada pela organização Sencha e tem como base da sua construção a plataforma JavaScript de desenvolvimento *web* Ext JS, pertencente à mesma empresa. A plataforma Sencha Touch herda os conceitos de Ext JS no desenvolvimento de aplicações *web* interoperáveis com interfaces personalizáveis, evoluindo o seu paradigma recorrendo às melhores práticas de HTML5, CSS3 e JavaScript e aplicando-o num contexto de desenvolvimento de aplicações móveis.

A plataforma Sencha Touch é organizada segundo o padrão estrutural Model-View-Controller, denominado por MVC. Além das três camadas base deste padrão, são também implementadas as camadas *Store* e *Profile*. A estrutura é representada na Figura 9.

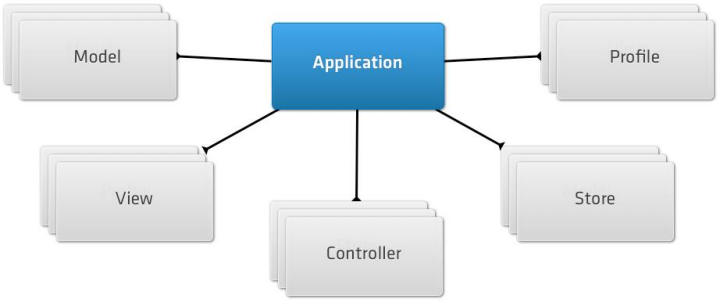


Figura 9 - Arquitetura de Sencha Touch [Sencha Touch, 2015]

A camada *Model* representa o modelo de dados da aplicação, responsável por definir a estrutura lógica e tratar a consulta e manipulação da informação associada. Em suma, esta camada é responsável pela comunicação entre a aplicação e a sua estrutura de dados, assim como pela realização das operações relacionadas com a sua informação. Já a camada *View* controla toda a apresentação visual com o utilizador final da aplicação, sendo ainda responsável

por qualquer interação visual com o mesmo através dos componentes que disponibiliza. Por seu lado, a camada *Controller* funciona como uma camada intermediária entre as duas anteriores, sendo responsável por controlar todo o fluxo de informação, gerindo os dados do *Model* a apresentar na *View* e a forma como os mesmos podem ser visualizados.

De forma a exemplificar o funcionamento destas três camadas, considere-se o exemplo de um utilizador efetuar o processo de autenticação numa aplicação. A camada *View* apresenta o ecrã de autenticação onde o utilizador terá de inserir as suas credenciais, como por exemplo o seu correio eletrónico e senha associada. A camada *Controller* é responsável por enviar os dados inseridos à camada *Model*, que por seu lado verifica se os dados estão corretos de acordo com a informação existente na estrutura de dados. Mediante a resposta do *Model*, o *Controller* dá ordem para a *View* encaminhar o utilizador para o ecrã de boas vindas em caso de sucesso ou uma mensagem de erro em caso de falha.

Como referido anteriormente, a plataforma acrescenta duas camadas adicionais além das três do modelo MVC tradicional, sendo elas a camada *Store* e a camada *Profile*.

A camada *Store* funciona como um repositório local responsável por armazenar uma coleção de informação relativa a instâncias da camada *Model*. Esta camada permite a filtragem, ordenação e agrupamento da sua informação, tornando-se ideal para a apresentação da informação em listagens ou outros formatos estruturais similares. Uma das vantagens da utilização desta camada é a possibilidade de carregar a informação do *Model* numa fase inicial, efetuando múltiplas operações de consulta e manipulação de forma local e com melhor desempenho. Por outro lado, torna-se necessário ter o cuidado de sincronizar periodicamente a informação com o *Model* de forma a atualizar os dados da estrutura de dados da aplicação. Utilizando o exemplo anterior, a *Store* seria responsável pelo carregamento e armazenamento prévio da informação dos utilizadores existentes no *Model*, bastando ao *Controller* validar as credenciais introduzida com os dados da *Store* ao invés de o fazer diretamente na estrutura de dados da aplicação. No Código Fonte 1 é exemplificada a representação do *Model* e *Store* de uma entidade e a sua relação.

```
Ext.define('ModelUtilizadores', {
    extend: 'Ext.data.Model',
    config: {
        idProperty : 'id',
        fields: ['id','nome','email','senha']
    }
});
```

MODEL

```
Ext.define('StoreUtilizadores', {
    extend: 'Ext.data.Store',
    config: {
        model  : 'ModelUtilizadores',
        storeId: 'utilizadores',
        data:
        [{
            id:    '1',
            nome:  'Utilizador 1',
            email: 'email1@mail.com',
            senha: '12345678',
        },{
            id:    '2',
            nome:  'Utilizador 2',
            email: 'email2@mail.com',
            senha: '87654321',
        }]
    }
});
```

STORE

Código Fonte 1 - Exemplo da representação de *Model* e *Store*

Finalmente, a camada *Profile* surge com o objetivo de facilitar a adaptação das aplicações aos diferentes tipos de dispositivos, como são exemplo os *Smartphones*, *Tablets*, etc. Para tal, permite a definição de um perfil para cada tipo de dispositivo. Cada perfil poderá ser responsável por gerir *Controllers* e *Views* distintos de acordo com o comportamento desejável no equipamento em questão, permitindo assim implementar diferentes experiências de utilização com um custo de desenvolvimento reduzido comparativamente com a alternativa de desenvolver a aplicação separadamente para cada tipo, dado que o restante código-fonte e a lógica aplicacional são partilhados. Importa ainda referir que a utilização desta camada no desenvolvimento das aplicações é opcional.

Permitindo uma forte organização estrutural do código-fonte e facilidade do seu reaproveitamento e uma customização dos componentes da aplicação de acordo com o tipo de dispositivo a utilizar, o modelo de cinco camadas utilizado pelo Sencha Touch torna-se uma combinação poderosa e vantajosa no desenvolvimento de aplicações móveis. No entanto, poderá existir alguma perda de desempenho dada a complexidade do motor responsável por manter esta estrutura, especialmente em aplicações cujo modelo possa não ser o mais adequado [Grgur Grisogono, 2014].

Além das característica supracitadas, Sencha Touch apresenta ainda as seguintes vantagens:

- Compatibilidade com todos os navegadores de internet móveis;
- Disponibilização de bibliotecas com temas e componentes visuais adaptados aos diversos sistemas operativos, conseguindo apresentar uma fluidez muito próxima do comportamento nativo;
- Flexibilidade na utilização dos seus componentes, possibilitando um nível de customização elevado, recorrendo a HTML5, CSS3 e JavaScript;
- Suporte de interfaces de comunicação RESTful e Simple Object Access Protocol (SOAP), as principais utilizadas na troca de informação entre serviços *web*;
- Compatibilidade com PhoneGap e Apache Cordova para integração com os componentes nativos dos dispositivos como por exemplo a câmara fotográfica. Com recurso à combinação com uma destas plataformas é também possível o encapsulamento de uma aplicação *web* em uma aplicação nativa, de forma a executar localmente no dispositivo;
- Apresenta uma documentação muito completa e detalhada, com recurso a exemplos e tutoriais. O desenvolvimento de melhorias é constante por parte da empresa Sencha e atualmente existe uma comunidade bastante ativa com um número de utilizadores cada vez mais elevado [Grgur Grisogono, 2014];
- Suporta a distribuição das suas aplicações nos principais mercados de aplicações móveis;
- A organização Sencha disponibiliza um compilador, Sencha Cmd, que, entre as diversas funcionalidades e vantagens que lhe são reconhecidas, é responsável por, conjuntamente com a plataforma Sencha Touch, comprimir e otimizar as aplicações desenvolvidas para que a sua leitura e execução sejam realizadas de forma mais eficiente pelos diferentes dispositivos.

No sentido oposto, apresenta também alguns pontos negativos e limitações, tais como:

- A retrocompatibilidade entre versões do Sencha Touch nem sempre tem sido assegurada, sendo que para aplicações com versões inferiores à versão 2 é necessária uma elevada reestruturação do seu código-fonte para evoluir o sistema e usufruir das funcionalidades das últimas versões da plataforma [Annie, 2012];
- O custo de aprendizagem é elevado comparativamente com outras tecnologias de desenvolvimento *web* móvel, dada a especificidade de estruturação apresentada pela plataforma;
- Dependendo da complexidade das aplicações, do método de desenvolvimento e das bibliotecas utilizadas, a aplicação poderá apresentar algumas limitações relativamente ao seu desempenho [Jim Cowart, 2013];
- A licença da plataforma é propriedade da organização Sencha, fator que implica uma dependência da sua continuidade de acordo com a aposta da organização no desenvolvimento e melhoria do projeto. No entanto, em contrapartida, de certa forma, este licenciamento acaba por poder ser considerado vantajoso na medida em que existe uma estrutura responsável por garantir apoio técnico e manutenção da plataforma.

2.3.3.2 PhoneGap / Apache Cordova

Figura 10 – PhoneGap [PhoneGap, 2015]

PhoneGap (Figura 10) apresenta-se como uma plataforma de desenvolvimento de aplicações móveis segundo uma metodologia híbrida. Inicialmente desenvolvida pela Nitobi em 2009 com o principal objetivo de conjugar tecnologias de desenvolvimento *web* puro como HTML5, CSS3 e JavaScript com comportamentos e componentes nativos na criação de aplicações móveis compatíveis com qualquer sistema operativo.

Em 2011, a Nitobi foi adquirida pela Adobe, juntamente com a marca PhoneGap, tendo doado o código-fonte da plataforma à Apache Software Foundation, que por sua vez alterou a sua designação por questões legais relativamente à marca, dando origem ao projeto de código-fonte livre Apache Cordova (Figura 11) [Cordova, 2015].

Figura 11 - Apache Cordova [Cordova, 2015]

Atualmente, apesar das duas plataformas serem fornecidas em diferentes pacotes de instalação, são as duas de código-fonte e utilização livres e a sua estrutura e funcionalidades são extremamente similares. Cordova acaba por ser o motor de desenvolvimento principal e PhoneGap é uma distribuição de Cordova com a adição de possibilitar uma integração com outros produtos desenvolvidos pela Adobe. Essa integração é aliás a única diferença funcional significativa entre as duas plataformas de desenvolvimento [Brian, 2012 e Mark, 2014].

Um exemplo da similaridade das duas plataformas é o facto de apesar de cada plataforma ter a sua página de internet para divulgação e descarregamento, ambas partilham a mesma fonte de documentação.

Existe ainda a particularidade de que a plataforma Cordova, com uma comunidade extensa, é atualizada com maior frequência, sendo as alterações transitadas para a suas distribuições como o PhoneGap em períodos posteriores.

Inicialmente, a ANO Software utilizava a plataforma PhoneGap nas suas aplicações móveis, tendo posteriormente transitado para Cordova, numa altura em que o fator de atualização constante da plataforma em períodos de tempo mais curtos se mostrava mais vantajoso em relação às funcionalidades de integração aplicacional da Adobe fornecida por PhoneGap, que para o âmbito específico não seriam sequer necessárias. Como tal, é dado maior ênfase a Cordova na análise tecnológica que se segue.

O principal foco da plataforma passa por disponibilizar ferramentas que permitam a utilização de componentes nativos dos equipamentos, independentemente do sistema operativo em que as aplicações sejam executadas [Cordova, 2015]. Este foco possibilita, em conjugação com uma plataforma de desenvolvimento *web* com recurso a tecnologias como HTML5, CSS3 e JavaScript como o Sencha Touch, o desenvolvimento de aplicações multiplataforma com acesso a recursos nativos dos equipamentos sem a dependência direta de utilização de código-fonte de cada sistema móvel. Na prática, tal como representado na Figura 12, com esta plataforma aplica-se a metodologia de desenvolvimento híbrido.

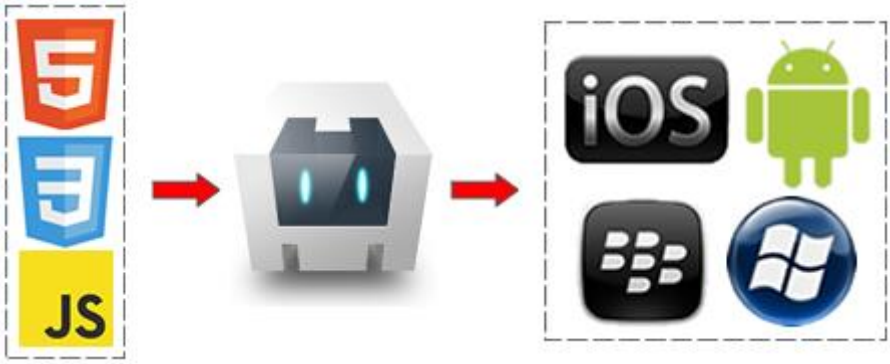
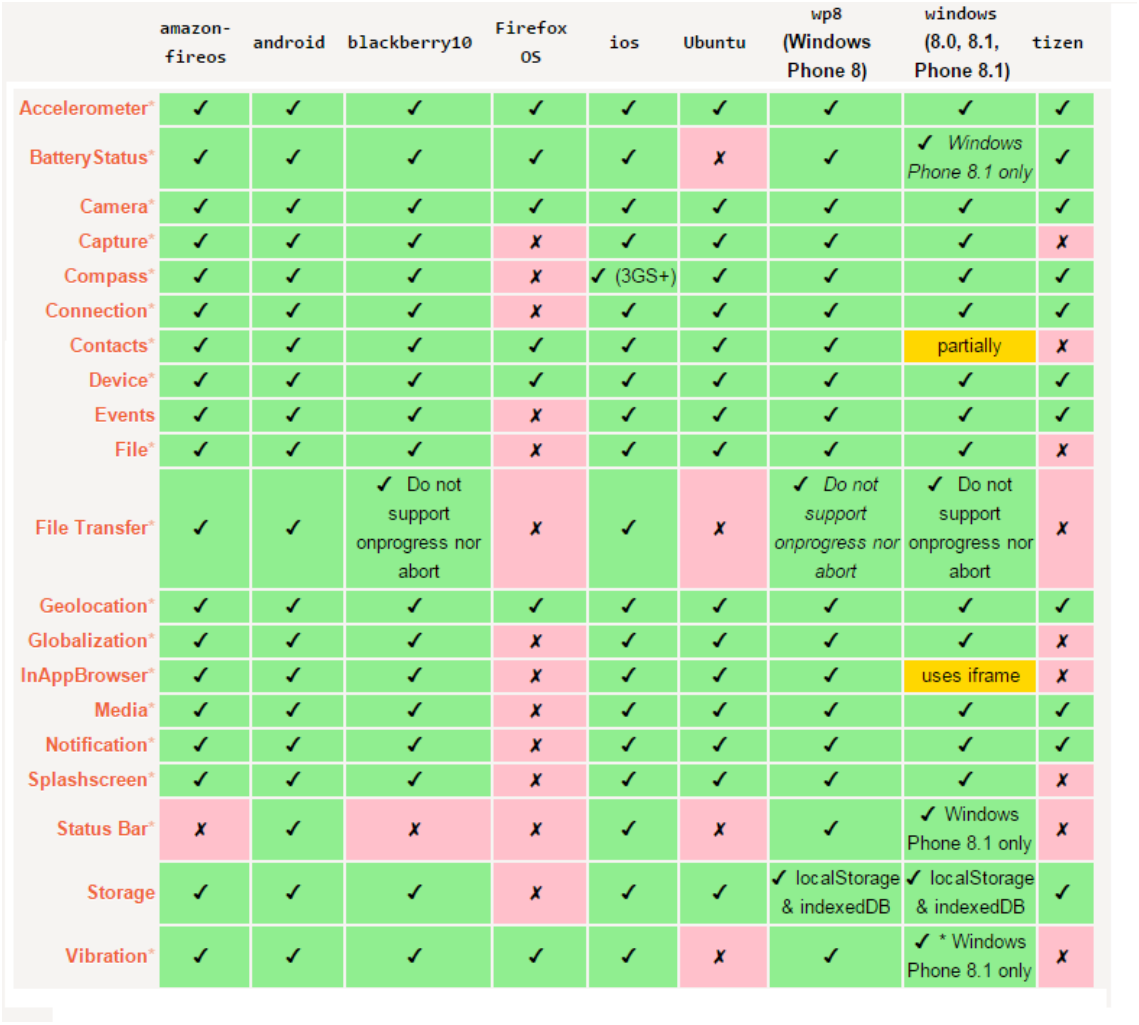


Figura 12 - Processo híbrido com Cordova

A plataforma Cordova suporta uma grande variedade de sistemas operativos, disponibilizando documentação específica para cada um deles com o intuito de auxiliar nas configurações dos ambientes necessários ao seu desenvolvimento. Os principais sistemas suportados são:

- Amazon Fire OS;
- Android;
- BlackBerry 10;
- Firefox OS;
- iOS;
- Ubuntu;
- Windows 8 e 8.1;
- Windows Phone 8;
- Tizen.

Dependendo dos sistemas operativos alvo, a plataforma disponibiliza *plugins* para acesso a alguns dos seus componentes nativos através da aplicação, como por exemplo, a câmara fotográfica, o acelerómetro ou o estado da ligação à internet do dispositivo. Os *plugins* são desenvolvidos em JavaScript de forma a potenciar o fator de multiplataforma e serem compatíveis com o maior número de sistemas possível, objetivo nem sempre possível em pleno dadas as diferenças significativas dos requisitos de cada um. A base documental para cada um dos *plugins* fornecidos pela plataforma é bastante completa, especificando o comportamento expetável em cada sistema, as limitações que podem ser encontradas e/ou a necessidade de efetuar alguma configuração adicional de forma a garantir a correta integração. Na documentação da plataforma [Cordova Docs, 2015] pode ser consultada uma matriz de compatibilidade entre os diversos *plugins* disponíveis e os diversos sistemas operativos suportados. A matriz é apresentada também na Figura 13.



	amazon-fireos	android	blackberry10	Firefox OS	ios	Ubuntu	wp8 (Windows Phone 8)	windows (8.0, 8.1, Phone 8.1)	tizen
Accelerometer*	✓	✓	✓	✓	✓	✓	✓	✓	✓
Battery Status*	✓	✓	✓	✓	✓	✗	✓	✓ *Windows Phone 8.1 only*	✓
Camera*	✓	✓	✓	✓	✓	✓	✓	✓	✓
Capture*	✓	✓	✓	✗	✓	✓	✓	✓	✗
Compass*	✓	✓	✓	✗	✓ (3GS+)	✓	✓	✓	✓
Connection*	✓	✓	✓	✗	✓	✓	✓	✓	✓
Contacts*	✓	✓	✓	✓	✓	✓	✓	partially	✗
Device*	✓	✓	✓	✓	✓	✓	✓	✓	✓
Events	✓	✓	✓	✗	✓	✓	✓	✓	✓
File*	✓	✓	✓	✗	✓	✓	✓	✓	✗
File Transfer*	✓	✓	✓ Do not support onprogress nor abort	✗	✓	✗	✓ *Do not support onprogress nor abort*	✓ Do not support onprogress nor abort	✗
Geolocation*	✓	✓	✓	✓	✓	✓	✓	✓	✓
Globalization*	✓	✓	✓	✗	✓	✓	✓	✓	✗
InAppBrowser*	✓	✓	✓	✗	✓	✓	✓	uses iframe	✗
Media*	✓	✓	✓	✗	✓	✓	✓	✓	✓
Notification*	✓	✓	✓	✗	✓	✓	✓	✓	✓
Splashscreen*	✓	✓	✓	✗	✓	✓	✓	✓	✗
Status Bar*	✗	✓	✗	✗	✓	✗	✓	✓ Windows Phone 8.1 only	✗
Storage	✓	✓	✓	✗	✓	✓	✓ localStorage & indexedDB	✓ localStorage & indexedDB	✓
Vibration*	✓	✓	✓	✓	✓	✗	✓	✓ * Windows Phone 8.1 only	✗

Figura 13- Matriz de compatibilidade de plugins base de Cordova [Cordova Docs, 2015]

Caso exista a necessidade de aceder a um componente nativo para o qual não existe um *plugin* base ou este não é compatível para o sistema alvo, a plataforma permite a utilização de *plugins* criados e customizados pelos seus utilizadores. No entanto, a complexidade do seu desenvolvimento é elevada, uma vez que requer conhecimento aprofundado acerca do funcionamento técnico do componente em causa. No caso de uma aplicação necessitar de aceder a vários componentes para os quais não existem *plugins* de integração disponíveis, o custo de desenvolvimento pode aumentar significativamente, dependendo da complexidade do componente e do número de sistemas alvo, tornando-se num fator desvantajoso para as organizações [Jim Cowart, 2013].

A aplicação dos *plugins* pode ser facilmente realizada através do Cordova *Command-Line Interface* (CLI). Introduzida na versão 3.0, consiste numa interface de linha de comandos que possibilita as seguintes ações:

- **Criação de uma nova aplicação –** cria a estrutura base que servirá o projeto da aplicação e gera o ficheiro de configurações gerais *config.xml*, onde ficam armazenadas os parâmetros básicos tais como o nome, descrição e autores da aplicação, bem como

as referências aos *plugins* utilizados. A estrutura base do projeto é constituída pelos diretórios *www*, *platform* e *plugins*. O diretório *www* é o diretório principal do projeto uma vez que a sua função é armazenar os ficheiros de código-fonte da aplicação, ou seja, o resultado da compilação de plataformas de desenvolvimento *web* como o Sencha Touch, por exemplo. Já o diretório *platform* tem como finalidade armazenar a aplicação híbrida final, isto é, o resultado de compilação do Cordova, sendo criado um subdiretório por cada sistema desejado. Importa assinalar que o conteúdo destes subdiretórios é gerado pelo processo de compilação, pelo que não deve ser alterado manualmente. Finalmente, o diretório *plugin* é utilizado para armazenar os *plugins* Cordova a utilizar na aplicação;

- **Adição de sistemas móveis suportados pela aplicação –** é necessário executar este comando por cada sistema móvel que se deseja suportar na aplicação, uma vez que é ele que cria o subdiretório específico para cada uma no diretório *platform*;
- **Adição de *plugin* à aplicação –** permite adicionar ou retirar um *plugin* para que a aplicação consiga aceder a determinado componente nativo do equipamento onde será executada;
- **Compilação de uma aplicação –** interpreta o código fonte do diretório *www,* compila-o e gera a aplicação final de acordo com o sistema desejado, ajustando as configurações específicas, adaptando os *plugins* selecionados e copiando os recursos visuais como os ficheiros de estilo e imagens para o subdiretório relacionado. Este processo gera ainda um executável não assinado para instalação num dispositivo móvel, bastando assiná-lo posteriormente com recurso a uma *keystore* para possibilitar a sua distribuição nos mercados de aplicações móveis dos diferentes sistemas.
- **Execução de uma aplicação –** permite executar e testar a aplicação num dispositivo móvel disponível ou num emulador instalado na máquina de desenvolvimento, evitando assim a necessidade de um ambiente de desenvolvimento aplicacional específico para essa tarefa.

Todos os pontos apresentados revelam uma plataforma que possibilita diversas vantagens no âmbito da criação de aplicações multiplataforma com um custo de desenvolvimento baixo, no tanto, existem alguns aspetos que devem ser levados em grande atenção aquando da sua utilização.

O ponto mais crítico a ter em conta nesta plataforma de desenvolvimento é o desempenho das aplicações, limitação inerente às aplicações móveis de origem *web* e híbrida quando comparadas com as suas homónimas nativas. Por muito bem que os componentes nativos sejam integrados na aplicação, continua a ser um motor intermediário que realiza essa ponte, pelo que o mesmo poderá ser restringido por determinados sistemas e/ou equipamentos. Outros dos pontos críticos a ter cuidado e que resultam em parte do desempenho, é a fluidez dos componentes visuais da aplicação. Apesar de serem desenvolvidos com tecnologias *web* genéricas de grande maturidade e de conseguirem apresentar um comportamento muito próximo dos componentes nativos, não o são, e como tal, a experiência da sua utilização pode tornar-se indesejável, como por exemplo ocorrerem atrasos na execução de ações banais como pressionar um botão ou selecionar um elemento de uma lista.

Estes potenciais problemas podem ser minimizados pelas equipas de desenvolvimento se forem tidos em conta como uma preocupação constante no ciclo de vida da criação e manutenção da aplicação, uma vez que existe um grande número de meios de analisar e monitorizar as variáveis em questão.

2.3.4 O contexto atual

Inicialmente, a escolha da metodologia teve efeitos práticos positivos, sendo eficaz no cumprimento dos objetivos, nomeadamente, a criação de aplicações móveis multiplataforma simples com um custo de desenvolvimento relativamente baixo.

Mesmo que se considere um baixo custo de desenvolvimento, se comparado com o custo de realização do mesmo objetivo seguindo outras abordagens, tornou-se um processo mais dispendioso que o esperando uma vez que se verificou o seguinte:

- A configuração do ambiente de desenvolvimento é mais complexa do que na abordagem nativa, sendo necessária a utilização de mais ferramentas e tecnologias de desenvolvimento e compilação dos projetos.
- O tempo de aprendizagem das tecnologias utilizadas é elevado, especialmente, devido à especialização dos elementos das equipas de desenvolvimento da organização que assenta essencialmente na linguagem Java.

Apesar deste custo inicial mais elevado, o resultado final apresentou-se bastante satisfatório, ainda que se tenha percebido, ao longo do desenvolvimento, que seria necessário ter em especial atenção a criticidade do tratamento do desempenho da aplicação e o comportamento dos seus componentes. No entanto, dada a simplicidade de funcionalidades que as aplicações teriam, este possível problema ainda não era verificado.

Com a necessidade do mercado alvo das aplicações móveis da organização a aumentar exponencialmente, tornou-se necessária a inclusão de novas funcionalidades, cada vez mais complexas, o que começou a desencadear as mais diversas dificuldades, das quais se destaca a redução significativa do desempenho e da fluidez das aplicações, passando a apresentar tempos de resposta elevados nas ações e comportamentos visuais indesejáveis, desde componentes fragmentados a bloqueios momentâneos de ecrã ao apresentar a informação.

Na tentativa de resolução destas questões foi realizada uma reestruturação e otimização da construção das aplicações, que, conjuntamente com a utilização das versões mais recentes lançadas das plataformas de desenvolvimento híbridas, permitiu reduzir substancialmente a ocorrência dos problemas encontrados, tornando as aplicações mais estáveis mas ainda sem os valores de desempenho e fluidez ideais quando comparadas com outras aplicações com grau de complexidade semelhante. Além disso, um dos pontos focados nessa reestruturação passou por bloquear determinadas funcionalidades das aplicações, o que se torna uma grande limitação a médio-longo prazo, uma vez que levará a uma estagnação das aplicações que não poderão acompanhar as necessidades e pedidos crescentes dos clientes da organização, podendo trazer como consequência a desistência dos mesmos na sua utilização.

Uma vez que o tempo e esforço despendidos na reestruturação são elevados e não apresentam uma solução viável para os objetivos atuais da organização, torna-se necessária a adoção de uma nova estratégia que permita não só colmatar as limitações encontradas nas aplicações móveis, mas que, ao mesmo tempo, permita também que as mesmas possam evoluir funcionalmente de forma robusta e sustentada.

Na análise da questão é importante ter em conta alguns fatores importantes que foram evidenciados através da experiência de atuação da organização na área:

- Apesar de inicialmente se considerar uma vantagem a possibilidade de apresentar aplicações para os mais diversos sistemas operativos em simultâneo, a verdade é que a base de clientes tem apresentado necessidades exclusivas na plataforma Android. As razões apresentadas para esta escolha passam pela maior variedade de escolha de dispositivos, especialmente no que toca à relação das margens de preços. Uma vez que a organização atua sob o mercado de organizações públicas, prevê-se que a tendência da escolha dos dispositivos continuará a recair na plataforma Android;
- Como referido anteriormente, as equipas de desenvolvimento da organização são especializadas na linguagem Java e nas suas plataformas *web*, uma vez que a sua aposta para a grande maioria dos seus produtos recai nessa escolha tecnológica. Como tal, sempre que é necessário integrar novos elementos para efetuar correções e ou melhorias nas aplicações móveis, o tempo de aprendizagem é elevado e uma intervenção que deveria ser rápida acaba por se tornar mais dispendiosa.

Tendo em conta a problemática encontrada, conjuntamente com os fatores que evidenciam que a abordagem de desenvolvimento de aplicações multiplataforma utilizada no âmbito da organização não é justificável nem adequada, é então definido como estratégia alternativa de resolução, a migração das aplicações móveis da organização para Android passando assim de uma metodologia multiplataforma para uma metodologia nativa.

Uma vez que a organização tem várias aplicações móveis implementadas, é um dos principais objetivos que a migração tecnológica seja realizada com o menor custo de desenvolvimento possível, uma vez que adaptar cada uma de raiz pode ser um processo demorado e penoso. Uma vantagem em relação a este ponto é o facto de as aplicações serem estruturadas e moduladas de forma similar entre si, diferenciando apenas na lógica de negócio em que se enquadram.

No cenário apresentado surgem os objetivos a que esta dissertação se propõe. Por um lado, apresentar um plano de migração genérico que procure definir um processo de desenvolvimento que possa ser aplicado a qualquer das aplicações móveis estruturadas com as tecnologias utilizadas. Um outro objetivo será demonstrar que a estratégia de migração para a abordagem nativa permite ultrapassar os obstáculos atuais, nomeadamente, a melhoria de desempenho e fluidez comportamental das aplicações.

De forma a cumprir estes objetivos será utilizada uma das aplicações da organização como objeto de estudo. A aplicação escolhida para o propósito é intitulada Sistema de Trânsito e Contraordenações (STC).

2.3.5 Sistema de Trânsito e Contraordenações

O STC apresenta-se como uma solução *web* dinâmica e simplificada para o tratamento de todo o processo de contraordenações de trânsito. É um sistema completo e integrado, que promove a desmaterialização dos processos e a agilidade da organização.

O STC permite aos agentes de autoridade fiscalizadora do trânsito deixarem de utilizar o papel como suporte de registo para o tratamento das infrações cometidas pelos transeuntes da via pública. Desta maneira, todo o processo de tratamento de informação de contraordenações passa a ser feito em formato eletrónico tanto através da componente de *BackOffice* como uma componente de *FrontOffice*.

A componente de *BackOffice* é utilizada nas instalações da organização e tem como propósito o registo de contraordenações, a verificação do estado de determinada ocorrência, o registo de um condutor assim como as contraordenações a ele associado.

A componente de *FrontOffice* corresponde a uma aplicação móvel e é utilizada através de um dispositivo suportado como um *Smartphone* ou um *Tablet*, sendo de fácil transporte para os agentes da organização, permitindo-lhes consultar e registar contraordenações e infrações de forma imediata, facultando uma atuação mais eficaz e ágil no terreno.

Para o cumprimento dos objetivo propostos no âmbito desta dissertação será utilizada a componente de *FrontOffice* da aplicação, que tem como principais funcionalidades:

- Sincronização de dados com o *BackOffice* da aplicação, alojado num servidor *web*;
- Registo de novas contraordenações;
- Possibilidade de captação de registos fotográficos para arquivar ao processo de uma contraordenação;
- Impressão de contraordenações via comunicação com impressoras *Bluetooth*;
- Pesquisa *Online* e *Offline* dos dados de contraordenações registadas e respetiva informação de utentes.

3. Análise

Posteriormente a ter sido apresentado o estudo do Estado da Arte e ter sido realizada uma introdução à problemática que originou esta dissertação, neste capítulo será descrita a análise realizada com o intuito de desenvolver uma solução adequada.

A fase de análise é fundamental antes de iniciar o processo de implementação da solução. Na fase de análise desta dissertação é realizado o levantamento de requisitos através dos objetivos propostos e é descrita a estrutura da aplicação que servirá de piloto para o processo de migração.

3.1 Levantamento de requisitos

O processo de levantamento de requisitos consiste na análise e recolha das necessidades associadas a uma problemática. O resultado deste processo permite estabelecer uma base de objetivos a cumprir na fase de implementação, ou seja, pode ser considerado como um guião do desenvolvimento a efetuar.

No capítulo anterior foi apresentada a problemática desta dissertação e o seu contexto atual. Através dessa análise podem-se agrupar os objetivos principais a cumprir como:

- Efetuar um plano de migração genérico para as aplicações móveis multiplataforma da organização de forma a reduzir os custos de desenvolvimento na passagem de cada uma delas para uma abordagem nativa Android;
- Demonstrar que a alteração de estratégia aplicacional de uma abordagem multiplataforma para uma abordagem nativa apresenta resultados positivos na resolução dos problemas atuais relacionados com o comportamento das aplicações, como o seu desempenho e a melhoria da sua usabilidade.

Como tal, de forma a conseguir responder a estes pontos será necessário implementar um processo de migração de uma aplicação móvel multiplataforma da organização para a plataforma Android, servindo assim de piloto para a estruturação do plano de migração pretendido.

O plano de migração deverá apresentar o resultado conseguido através da aplicação do processo referido anteriormente, permitindo definir as linhas de desenvolvimento para os vários módulos comuns entre as diferentes aplicações.

Atualmente, apesar de funcionarem segundo lógicas de negócio distintas, as aplicações móveis da organização têm uma estrutura base e fluxo funcional semelhantes. Seguidamente, na Figura 14 é ilustrado um diagrama representativo da sua estrutura, dividida em módulos principais que deverão ser focados no processo de migração.

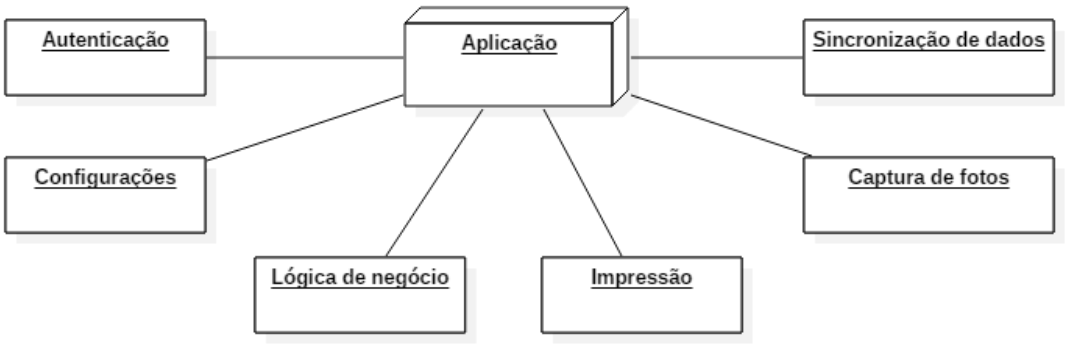


Figura 14 – Estrutura base das aplicações móveis da organização ANO

Como é apresentado na Figura 14, as aplicações podem ser divididas segundo seis módulos diferentes do ponto de vista funcional:

- **Autenticação**: módulo que permite aos utilizadores autenticarem-se através de credenciais privadas de forma a conseguirem aceder aos dados e interagir com as restantes áreas da aplicação. Este módulo, apesar de ser comum em todas as aplicações da organização, é também obrigatório, garantindo a segurança das mesmas, invalidando o acesso a informação restrita a utilizadores sem permissões.
- **Configurações:** módulo que representa as configurações gerais da aplicação, tais como a linguagem, o identificador do dispositivo ou o endereço para ligar com o BackOffice da aplicação. Este módulo é o único independente do módulo de autenticação, uma vez que é destinado aos administradores do sistema que têm a função de configurar as aplicações antes de entregar os dispositivos aos utilizadores finais. Apesar de não ser necessário realizar a autenticação na aplicação, este módulo tem o seu próprio mecanismo de segurança, sendo apenas acessível após introdução de uma senha. Tal como o módulo de autenticação, este módulo é comum e obrigatório em todas as aplicações atuais.
- **Sincronização de dados**: permite sincronizar os dados da aplicação móvel com o componente *BackOffice* através de um serviço *web*, cujo endereço deverá estar devidamente configurado nas configurações da aplicação. Em suma, o fluxo passa por numa primeira instância descarregar os dados para manipular na aplicação e enviar de volta para o *BackOffice*, de forma aos dois lados terem a sua informação coerente e atualizada entre si. Todas as aplicações atuais da organização utilizam também este módulo.
- **Captura de fotos**: módulo que tem como objetivo a captura de multimédia, mais especificamente fotos, recorrendo para tal à câmara fotográfica dos dispositivos. A ativação deste módulo, assim como a personalização dos seus parâmetros como por exemplo as dimensões das imagens e repositório onde as guardar é realizada através das configurações gerais da aplicação.
- **Impressão**: possibilita a impressão de informação através de impressoras móveis com recurso a uma ligação *Bluetooth*. Este módulo é responsável pela procura e emparelhamento com as impressoras, sendo possível definir nas configurações da

aplicação quais os modelos autorizados e o logotipo a utilizar nos documentos impressos.

- **Lógica de negócio**: ao contrário dos restantes módulo apresentados, representa todas as funcionalidades específicas de cada aplicação. Este componente traduz-se num conjunto de ecrãs e operações que variam de aplicação para aplicação e que depende diretamente dos requisitos funcionais de cada lógica de negócio e necessidades dos clientes associados.

Como referido anteriormente, a migração das aplicações deverá ser efetuado segundo a metodologia nativa para a plataforma Android. Como tal, deverá ser utilizada a ferramenta de desenvolvimento Android Studio [Android Studio, 2015], desenvolvida pela conceituada Google, de forma a implementar o processo para a aplicação escolhida. Sendo uma ferramenta específica para desenvolvimento em Android, disponibiliza os melhores e os mais atuais componentes e bibliotecas da plataforma que possibilitam aplicar os padrões recomendados com o objetivo de criar aplicações robustas e otimizadas para os seus dispositivos suportados.

Relativamente à compatibilidade aplicacional, para o processo de migração deverá ter-se o cuidado de abranger várias versões do sistema operativo, mantendo a retrocompatibilidade com dispositivos que utilizem versões mais antigas do sistema. De forma a cumprir esse objetivo, deverão utilizar-se sempre que possível as bibliotecas de compatibilidade fornecidas pela plataforma Android.

Quanto às especificações de Hardware a utilizar no processo de migração, deverão ser utilizados diferentes dispositivos no processo de desenvolvimento e teste da aplicação. Para tal, o Android Studio disponibiliza um leque de possibilidades para criar simuladores de alguns dos principais dispositivos existentes no mercado, com diferentes dimensões e versões de sistema operativo. Uma limitação dos simuladores é que consomem muitos recursos das máquinas de desenvolvimento e apesar de ser possível visualizar o enquadramento das aplicações em diferentes ecrãs, as aplicações apresentam um nível comportamental mais lento e limitado do que em dispositivos reais. Como tal, para o processo de depuração da aplicação deverão ser utilizados de preferência dispositivos reais de forma a garantir os melhores resultados do desenvolvimento.

No processo de desenvolvimento da migração, deverá ser dada especial atenção às questões do desempenho e usabilidade, sendo capaz de criar uma aplicação com interface responsiva e tempos de resposta rápidos e fluídos, sem qualquer evidência de paralisação de componentes ao carregar dados ou ecrãs que deixam de responder momentaneamente. Para funcionalidades que tenham tempos de processamento mais elevados, deverá ser dada a indicação clara ao utilizador que a aplicação se encontra a processar a informação de forma a este perceber o que está a acontecer e que é expectável a demora apresentada. Essa informação deverá ainda ser dinâmica para mostrar algum tipo de progresso da tarefa ao invés de criar a ilusão que poderá estar a processar em ciclo infinito.

Ainda com o intuito de ultrapassar os problemas comportamentais das aplicações, obtendo os melhores resultados possíveis, o processo de migração deverá cumprir as melhores práticas e

padrões específicos da plataforma de desenvolvimento Android, tal como é o caso do padrão MVC. Este padrão, além de manter o código-fonte da aplicação mais estruturado, facilita a sua manutenção e reutilização, tornando-se uma grande vantagem para o âmbito desta dissertação.

Como referido anteriormente nesta dissertação, a aplicação a utilizar como piloto na migração será o STC e como tal é importante efetuar uma análise prévia à sua estrutura e funcionalidades.

3.2 Arquitetura do STC

De seguida são apresentados alguns diagramas com o intuito de descrever e compreender o funcionamento e arquitetura da aplicação STC que irá ser utilizada na fase de implementação.

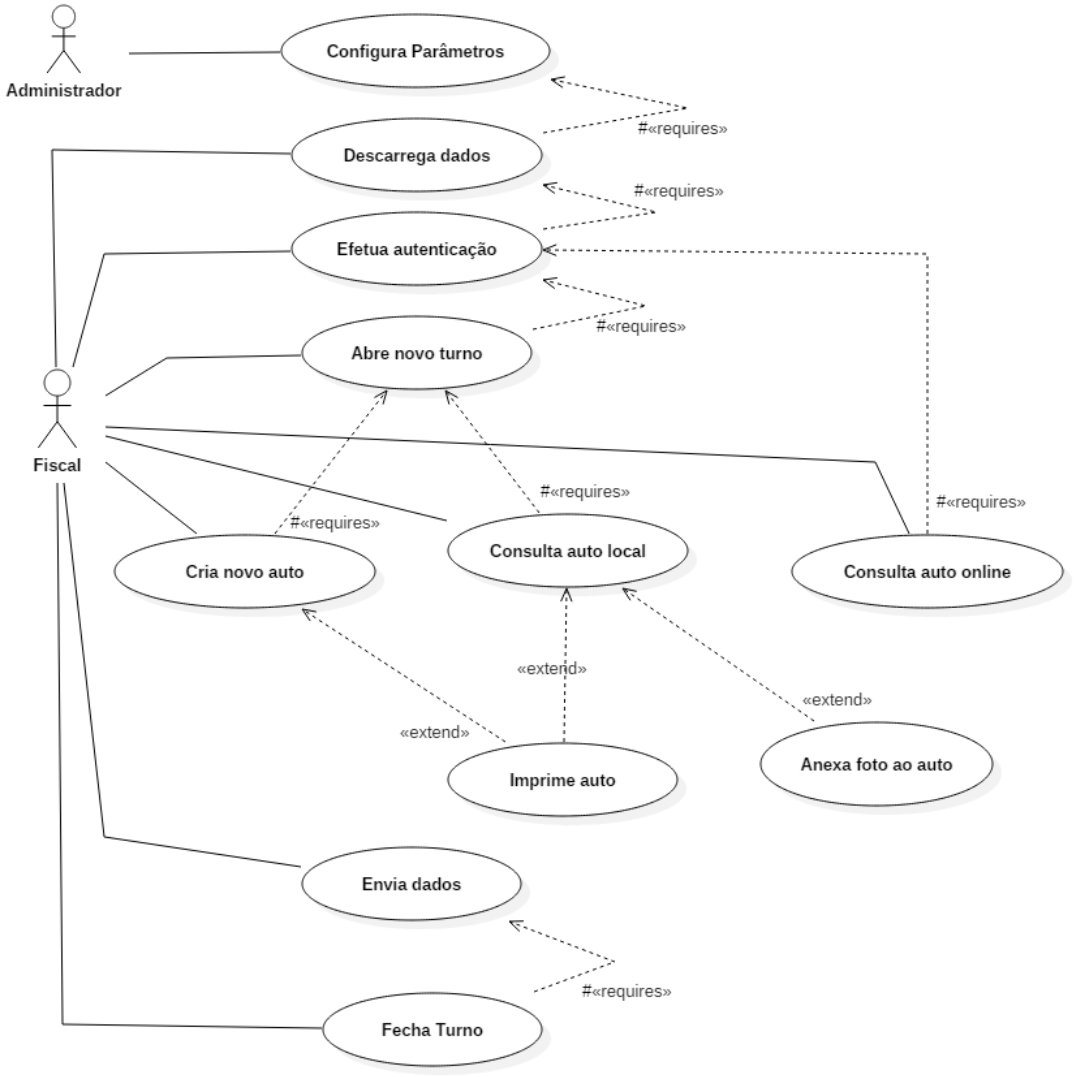


Figura 15 – Diagrama de Casos de Uso do STC

Na Figura 15 é ilustrado um diagrama de casos de usos simples com o propósito de apresentar as principais funcionalidades do sistema, nomeadamente as ações que os utilizadores podem despoletar na aplicação e a relação que apresentam entre si.

Como se pode constatar existem dois tipos de utilizadores da aplicação, Administrador e Fiscal. O utilizador do tipo Administrador apenas tem como função possível configurar a aplicação, consoante inserção de uma senha própria, enquanto o Fiscal corresponde ao perfil de utilizador que interage com a aplicação no seu todo, visto ser este tipo de utilizador que andará no terreno com o dispositivo móvel a verificar possíveis contraordenações.

Tal como referido, uma das premissas representadas é a de que apenas um utilizador do tipo Administrador tem acesso às configurações da aplicação de forma a preparar os parâmetros necessários à sua correta utilização. Esta ação é obrigatória visto que se os parâmetros como o endereço do serviço de BackOffice ou o identificador do dispositivo não estiverem configurados, o Fiscal não conseguirá descarregar os dados base e aceder ao dispositivo para efetuar as suas tarefas.

Após uma configuração correta, o Fiscal pode então utilizar a aplicação, sendo que para conseguir o acesso necessita de sincronizar os dados. O processo de sincronização é dividido no diagrama em duas ações: “Descarrega dados” e “Envia dados”. A nível aplicacional este processo é despoletado numa só ação em que é feito o descarregamento dos dados atualizados da componente de BackOffice e, caso tenha dados novos na aplicação móvel, trata de os enviar. Apesar disso, a nível conceptual, é feita essa divisão para melhor entendimento do seu enquadramento no fluxo.

Concluído o processo de descarregados os dados, o utilizador pode então aceder à aplicação através do processo de autenticação. O processo de autenticação é realizado de acordo com as credenciais introduzidas e com o identificador do dispositivo configurado pelo Administrador. Esta relação de validação só é possível após o processo de descarregamento dos dados do BackOffice, pois é lá que são definidos quais os identificadores dos dispositivos autorizados, quais os fiscais com acesso a cada dispositivo e quais as suas credenciais válidas.

Garantindo o acesso através da autenticação, é realizada uma verificação da existência de algum turno em curso. Um turno representa um período temporal da atividade de um fiscal, sendo que todas as contraordenações registadas ou editadas terão de estar obrigatoriamente relacionadas com um turno. Como tal, se não existir um turno ativo na aplicação, o fiscal, além de ter a possibilidade de dar início a um novo turno, apenas poderá efetuar a ação “Consulta auto *online*” representada no diagrama, isto é, consultar contraordenações que já se encontrem processadas no BackOffice da aplicação.

Existindo um turno aberto, o fiscal pode então proceder ao registo de novos autos ou à consulta de autos locais, criados no decorrer do turno atual. De notar que na fase de consulta das contraordenações locais, pode ser adicionada informação adicional e efetuada anexação de uma foto capturada pela câmara do dispositivo para arquivar junto do auto. Outra funcionalidade que deriva destas duas ações é a possibilidade de o fiscal conseguir imprimir as

contraordenações a qualquer momento após a sua criação, desde que detenha uma impressora com ligação *Bluetooth* e que esta esteja emparelhada com o dispositivo.

Por fim, o fiscal pode fechar o seu turno, sendo que para o conseguir necessite de enviar todos os novos dados registados para o BackOffice da aplicação para que todas as contraordenações sejam devidamente processadas.

Encontrando-se apresentado o fluxo comportamental da aplicação, é também necessário para esta fase de análise compreender um pouco da estrutura de dados do STC antes de iniciar a implementação da sua migração. Para cumprir esse objetivo, é representado na Figura 16 um diagrama de classes simplificado, contendo apenas os principais atributos necessários para conseguir ilustrar quais as diferentes peças estruturais e como se interligam entre elas.

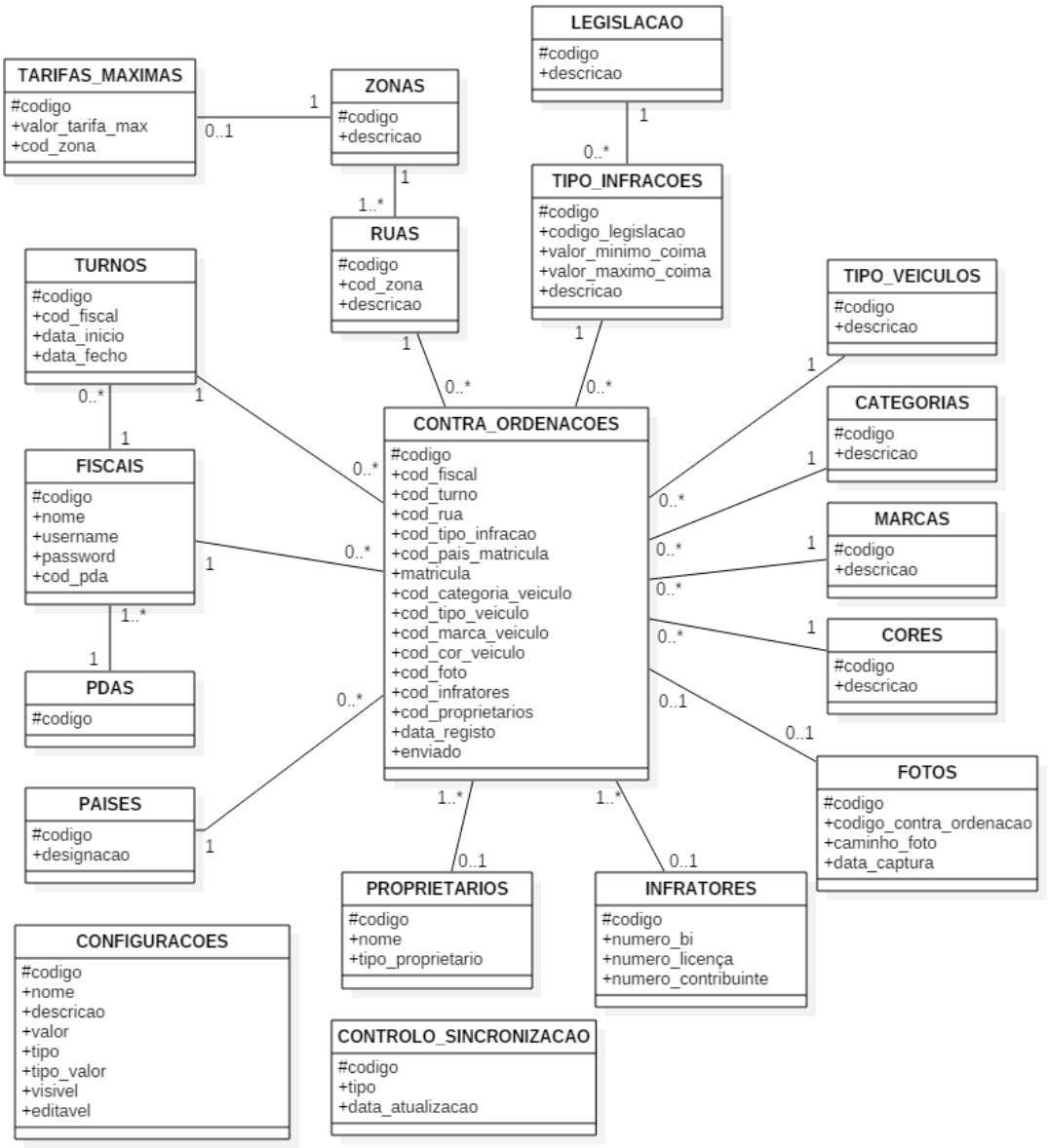


Figura 16 – Diagrama de Classes simplificado do STC

Um diagrama de classes tem como principal intuito representar a estrutura de uma aplicação e a forma como os seus elementos concetuais podem ser divididos e relacionados entre si.

Como se pode constatar, o conceito central da aplicação STC foca-se nas contraordenações, o que vai de acordo com o que foi apresentado até este ponto. O que não tinha sido ainda referido, mas pode ser analisado no diagrama de classes apresentado, é toda a estrutura que envolve este conceito, as peças necessárias para o seu correto processamento.

De acordo com o conhecimento geral e segundo o Dicionário de Língua Portuguesa, o termo contraordenação representa uma infração ou fato ilícito punível com coima. No caso da aplicação STC, o conceito de contraordenações encontra-se relacionado com as infrações de trânsito, como por exemplo, o estacionamento em zonas indevidas, o desrespeito pela sinalização, o incumprimento das normas básicas de segurança rodoviária ou a não apresentação dos documentos legais obrigatórios por parte de um cidadão. Como tal, uma contraordenação tem obrigatoriamente associada a si um tipo de infração, que por sua vez, tem como base uma legislação vigente e dá origem a uma coima entre um intervalo previamente definido.

Uma vez que a aplicação trata contraordenações rodoviárias, é necessário, no momento do seu registo, associar o local da sua ocorrência. A representação desse local é ilustrada através da classe Ruas, que por sua vez pertence a uma determinada zona geográfica que pode ter ou não uma tarifa máxima de coima específica para tipos de infrações muito específicos, como por exemplo o estacionamento indevido em parques de pagamento obrigatório.

Também associado a uma contraordenação rodoviária encontra-se um veículo. No diagrama de classes e a nível estrutural esse conceito não está explícito à primeira vista, encontrando-se a sua informação modularizada em diversas classes. Poderá ser uma oportunidade de melhoria à estrutura da aplicação a especificação de uma classe denominada “Veículo” que faça a agregação de todos os seus módulos e efetue uma ligação direta à contraordenação ao invés do modelo atual. Como se pode verificar no diagrama, é registada a matrícula do veículo na própria contraordenação, existindo depois relação a uma categoria (ligeiro, pesado, motociclo), um tipo de veículo (passageiros, mercadorias), uma marca e uma cor. É importante também referir a existência de uma validação do formato da matrícula relativamente ao seu país de origem.

Existem dois conceitos que se podem encontrar registados numa contraordenação e que também se encontram relacionados com um veículo, que são os infratores e os proprietários. Um infrator representa o cidadão que cometeu a contraordenação, enquanto o proprietário representa o cidadão ou organização que detém o registo comercial do veículo envolvido. Assim sendo, o infrator pode ser ou não o proprietário do veículo autuado. O registo desta informação pode não ser obrigatório, dependendo do tipo de infração, visto que existem contraordenações que podem ser registadas sem o infrator se encontrar presente, como é caso mais comum, um estacionamento indevido.

Relativamente à entidade representativa de uma contraordenação, resta referir a ligação ao fiscal que a registou, assim como ao turno em que foi efetuada. Quanto aos seus atributos principais, é fundamental mencionar também o armazenamento da data exata do seu registo e o seu estado de sincronização com o sistema de BackOffice.

Quanto ao fiscal, que como já foi referido, representa o utilizador da aplicação, tem obrigatoriamente um turno associado para conseguir registar e consultar as suas contraordenações e, para além disso, está também obrigatoriamente relacionado com um dispositivo válido no qual utiliza a aplicação. A classe representativa do dispositivo é ilustrada no diagrama de classes como “PDAS”.

Finalmente, resta mencionar as duas classes que se encontram isoladas deste esquema específico da lógica de contraordenações. Por um lado temos as configurações gerais da aplicação e por outro uma classe de controlo de sincronização. Esta última é responsável pelo armazenamento e gestão da informação sincronizada e o seu processo será apresentado em maior detalhe na fase de implementação da dissertação. Importa referir que estas duas classes são comuns à estrutura de todas as aplicações móveis das aplicações da organização.

3.3 Conclusão da Análise

Para além de evidenciar os principais objetivos a alcançar no decorrer desta dissertação, o processo de análise teve como intuito a compreensão do contexto envolvente, identificando os requisitos e a estrutura base que se encontra atualmente implementada.

Através da análise de requisitos foi apresentada como uma necessidade fundamental o estabelecimento de um plano de migração tecnológico genérico que facilmente pudesse ser posto em prática em qualquer das aplicações móveis multiplataforma da organização. Para tal, foram identificados os diferentes módulos comuns às aplicações e que serão o foco principal do processo de implementação.

Com a perspetiva de resposta eficaz a este ponto, além de uma referência concetual e descritiva do processo de migração de cada módulo, será desenvolvido um sistema que permita criar automatismos de geração de código fonte que visem a simplificação de todo o processo em qualquer uma das aplicações. Este sistema, designado MobileMigration, consistirá num módulo intermédio, desenvolvido na linguagem Java, que apresente a capacidade de interpretar os diferentes módulos de uma aplicação móvel multiplataforma desenvolvida com base na metodologia híbrida e na estrutura de Sencha Touch e construa a base de uma aplicação móvel nativa Android, consoante é ilustrado na Figura 17.

É expetável que esta aplicação apenas terá como foco os módulos comuns entre aplicações, sendo que a componente específica de cada lógica de negócio não se correlaciona e como tal, o seu processo de migração poderá ter de ser realizado de forma isolada. No entanto, com o intuito de amenizar esta provável limitação, serão utilizadas, no decorrer do processo de migração da aplicação piloto, algumas metodologias de desenvolvimento que foquem a

abstração de forma a criar mecanismos e componentes genéricos que possam ser incluídas neste sistema.

Figura 17 – Esquema de funcionamento da aplicação MobileMigration

Em suma, o módulo intermédio a desenvolver será um ponto fulcral desta dissertação, pois será a partir dele que poderão ser atingidos os resultados de generalização que possam permitir a simplificação e a redução do tempo de desenvolvimento necessário à implementação do processo de migração das restantes aplicações alvo da organização a que se destina este trabalho.

4. Plano de Migração

Findo o processo de análise é de seguida apresentado o processo de implementação desenvolvido no decorrer desta dissertação. Primeiramente é feita uma descrição do ambiente de desenvolvimento utilizado no trabalho efetuado. Seguidamente, de forma a modularizar o plano de migração e conseguir uma melhor compreensão por parte do leitor, foi realizada uma divisão aos principais módulos de ação de acordo com as funcionalidades destacadas no processo de análise. Para cada um destes módulos é realizada uma descrição da metodologia implementada, bem como as dificuldades encontradas ao realizar o seu processo de migração. Além disso, é também apresentada de que forma o módulo foi utilizado no processo de desenvolvimento da aplicação MobileMigration, seja na criação de um processo automático, ou através da inclusão de mecanismos ou componentes auxiliares genéricos.

4.1 Ambiente de Desenvolvimento

Antes de se proceder à especificação do processo de implementação desta dissertação é importante efetuar uma breve referência às tecnologias e ferramentas utilizadas, de forma a contextualizar o leitor quanto a alguns conceitos presentes da fase seguinte.

Uma vez que o processo de migração foi idealizado no âmbito plataforma Android, a principal utilização tecnológica para o seu desenvolvimento recaiu para a linguagem de programação Java e a ferramenta Android Studio.

O ambiente de desenvolvimento Android foi configurado para a *Application Program Interface* (API) 22, proporcionando também a retrocompatibilidade até à API 17. Em suma, uma API representa um conjunto de bibliotecas de desenvolvimento que permitem a criação e compilação de aplicações móveis com as versões do sistema operativo de forma otimizada, conseguindo assim retirar o máximo proveito das suas funcionalidades. Como tal, quando é lançada uma nova versão do sistema operativo, é também, por norma, disponibilizada uma nova API de desenvolvimento de forma a garantir o correto funcionamento nos novos requisitos aplicacionais [Android Developers-1, 2015]. Neste caso, ao englobar uma compatibilidade entre a API 17 e a API 22, garante-se o correto funcionamento das aplicações em dispositivos que suporte desde a versão 4.2 até à versão 5.1 de Android, a mais atual no momento de início do processo de implementação desta dissertação.

Ainda relativamente à configuração do ambiente de desenvolvimento, foi utilizada a versão 22.0.1 das *build tools*, disponibilizada em Março de 2015. As *build tools* representam um conjunto de componentes ferramentas que têm como propósito auxiliar o processo de compilação das aplicações móveis Android.

Quanto ao processo de compilação utilizado, uma das vantagens da ferramenta Android Studio é a disponibilização de um sistema de compilação automatizado e independente da utilização da tradicional linha de comandos. Este sistema tem como base tecnológica a integração com o

Gradle [Gradle, 2015], que necessitando apenas da parametrização de um ficheiro de configuração executa todo o processo de compilação da aplicação de forma personalizada e automática. Além da facilidade de utilização e customização do processo de compilação, este mecanismo facilita também a reutilização do código-fonte e recursos da aplicação e a disponibilização de múltiplos executáveis, designados *Android Application Packages* (APK) da mesma aplicação com diferentes funcionalidades e aspetos [Android Developers-2, 2015].

Na estrutura de projetos Android, podem existir vários ficheiros de configuração Gradle, especificados como *build.gradle*, um de nível hierárquico superior, ao nível do projeto, e um por cada módulo integrante do projeto. É neste último ficheiro que se encontram as configurações específicas à compilação do módulo aplicacional, como por exemplo, a versão das *build tools* a utilizar no processo, a versão da API alvo que se pretende, os diferentes tipos de compilação que podem ser aplicados e as dependências de bibliotecas a incluir para garantir o correto funcionamento da aplicação [Android Developers-3, 2015]. No caso da aplicação desenvolvida no âmbito desta dissertação existe apenas um módulo no projeto, cujo ficheiro *build.gradle* é o seguinte:

```
apply plugin: 'com.android.application'

android {
    compileSdkVersion 22
    buildToolsVersion "22.0.1"

    defaultConfig {
        applicationId "com.example.hugo.stc_android"
        minSdkVersion 17
        targetSdkVersion 22
        versionCode 1
        versionName "1.0"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'),
'proguard-rules.pro'
        }
    }
}

dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile 'com.android.support:appcompat-v7:22.2.0'
    compile 'com.android.support:recyclerview-v7:22.2.0'
    compile files('libs/ksoap2-android-assembly-3.4.0-jar-with-dependen-
cies.jar')
    compile files('libs/woosimprinter_bt.jar')
}
```

Código Fonte 2 – Ficheiro Gradle de configuração

No início do ficheiro encontram-se definidas as versões de compilação e das build tools a utilizar, que, tal como referido anteriormente, correspondem à versão 22 e 22.0.1 respetivamente.

Dentro do elemento *defaultConfig* encontra-se configurado o identificador da aplicação ou módulo e as versões alvo e mínima da API a que se destina, que neste caso, tal como já indicado,

correspondem às versões 22 e 17. Por vezes, a garantia de retrocompatibilidade de versões apenas é possível se forem utilizadas as bibliotecas de compatibilidade disponibilizadas pelas API. Essas bibliotecas são conhecidas como *appcompat* e disponibilizam adaptações de novos componentes que sejam compatíveis com versões anteriores do sistema operativo Android. A sua inclusão nas aplicações é considerada uma boa prática uma vez que permite alargar o número de dispositivos suportados, mantendo o nível de desempenho e o comportamento aplicacional dos componentes nas suas versões compatíveis [Android Developers-4, 2015]. Neste caso, é utilizada a versão v7 pertencente à revisão 22.2.0 da biblioteca de suporte, lançada em Maio de 2015, tal como se encontra referenciado no elemento *dependencies*. Importa ainda referir que ao compilar a aplicação, o conteúdo do elemento *defaultConfig* é injetado no ficheiro *AndroidManifest.xml*, que representa o ficheiro de configuração obrigatório nas aplicações Android e onde era tradicionalmente, no sistema de compilação anterior, especificada esta informação.

No elemento *buildTypes* são especificados os tipos de compilação que podem ser aplicados a um módulo. Por defeito, são definidos os tipos *debug* e *release*, sendo o primeiro indicado para gerar versões de desenvolvimento com anotações que permitem uma melhor depuração do seu comportamento. Por outro lado, a versão *release* tende a ser mais apropriada para compilar as aplicações para ambiente produtivo [Android Developers-3, 2015]. Na aplicação em causa apenas foi configurado o tipo de configuração *release* de forma a efetuar um processo de compilação mais adequado ao ambiente produtivo desde início. Existe ainda nesta área a possibilidade de inclusão da ferramenta ProGuard no processo de compilação de determinado tipo. Esta ferramenta protege o código-fonte da aplicação, dificultando a sua interpretação por parte de elementos externos. Para cumprir esse objetivo utiliza mecanismos de remoção de código não utilizado e alteração de nomes de classes, métodos e atributos para otimizar e ofuscar o código-fonte aplicacional [Android Developers-5, 2015].

Por fim, de forma a concluir a apresentação do ficheiro Gradle, no elemento *dependencies* são definidas as bibliotecas necessárias ao correto funcionamento da aplicação. Além da definição do repositório onde se encontram as dependências internas, é possível definir quais as bibliotecas adicionais a incluir no processo de compilação e empacotamento da versão. Neste contexto, além da biblioteca de compatibilidade *appcompat*, já referida anteriormente, foram definidas as seguintes dependências:

- **RecyclerView** – tal como a biblioteca de compatibilidade, este componente encontra-se inserido no módulo de bibliotecas de suporte do Android. Apresentando-se como uma extensão às componentes de listagens, é indicada para a definição de listas dinâmicas, isto é, cujo conteúdo seja construído de acordo com determinadas dependências, não sendo fixo [Android Developers-6, 2015].
- **Ksoap2** – esta biblioteca deriva de um projeto de código livre e disponibiliza mecanismos que permitem a ligação de aplicações Android a serviços *web* recorrendo ao protocolo de comunicação Simple Object Access Protocol (SOAP) de forma fácil e eficiente [Ksoap2, 2015]. A inclusão desta dependência é necessária para a correta implementação do módulo de sincronização de dados com a aplicação BackOffice.

- **Woosim Printer** – a inclusão desta dependência foi um requisito necessário de forma a garantir a compatibilidade com impressoras móveis da marca Woosim. Através da sua utilização, além de proporcionar uma correta ligação às impressoras, permite aceder a funcionalidades específicas e formatar devidamente o conteúdo a imprimir de acordo com as bibliotecas que a Woosim disponibiliza. Esta biblioteca é fundamental para o desenvolvimento do módulo de impressão da aplicação.

Continuando o processo de apresentação tecnológica, para efetuar o armazenamento de dados e a gestão da sua estrutura recorreu-se à utilização de bases de dados SQLite [SQLite, 2015], suportadas nativamente pelos sistemas Android [Android Developers-7, 2015].

As tecnologias e ferramentas apresentadas até este ponto representam o *software* utilizado no processo migração e respetivo desenvolvimento da aplicação em ambiente Android.

Quanto ao desenvolvimento da aplicação MobileMigration, foi utilizada a linguagem Java com o intuito de garantir uma maior compatibilidade com o Android, tendo-se optado pela ferramenta Netbeans 8.0.2 para o seu desenvolvimento. Ainda em relação a esta aplicação, recorreu-se a *JavaScript Object Notation* (JSON) de forma a garantir uma interpretação e conversão de ficheiros utilizando uma notação genérica.

Passando para a especificação do *hardware* utilizado, foi utilizada uma máquina com o sistema operativo Windows 7 de 64 bits para o desenvolvimento dos dois módulos. Já os equipamentos móveis utilizados para acompanhamento de desenvolvimento e a realização de testes foram um *Smartphone* Bq Aquaris E4 e um *Tablet* Samsung Galaxy Tab2 de 10 polegadas com as versões 4.4.2 e 4.2 de Android, respetivamente. Foi também utilizada uma impressora Woosim PORTI-S30/40 com tecnologia *Bluetooth* para efetuar os testes ao módulo de impressão.

4.2 Estrutura do Projeto

Esta secção tem como intuito descrever o início do processo de implementação da migração aplicacional. Em primeira instância, foi necessário criar um projeto na nova tecnologia que estivesse dotado de uma estrutura adaptada ao seu contexto original. Para tal, inicialmente será realizada uma apresentação da estrutura do projeto segundo a plataforma Sencha Touch. No contexto desta dissertação, entende-se como estrutura de um projeto a sua organização de diretórios consoante a divisão lógica implementada, assim como os seus ficheiros de configuração base. Efetuada a apresentação estrutural será apresentado o processo de passagem dessa mesma estrutura para a criação de um projeto na plataforma Android. Uma vez que a estrutura das aplicações alvo desta dissertação partilham uma estrutura comum, o processo de migração que irá ser descrito poderá ser rapidamente aplicado a qualquer uma. De forma a auxiliar este processo, serão também apresentados os mecanismos de automação desenvolvidos para incluir na aplicação Java.

Como referido aquando da apresentação tecnológica de Sencha Touch, a sua base estrutural assenta na aplicação do padrão MVC. Ao contrário das implementações tradicionais deste padrão, esta plataforma apresenta ainda a particularidade de acrescentar duas camadas

adicionais à sua estrutura: a camada *Store* e a camada *Profile*. Este padrão é transposto para a estrutura de uma aplicação Sencha através da divisão de cada camada em diretórios agregados num módulo principal, designado *app*, que corresponde ao diretório principal do desenvolvimento da aplicação, ou seja, onde se localizam os ficheiros com o código-fonte produzido. A transposição referida encontra-se ilustrada na Figura 18, juntamente com a restante estrutura base de um projeto Sencha.

```
│   app.js
│   app.json
│   cordova.js
│   cordova_plugins.js
│   index.html
│   packager.json
├───app
│   ├───controller
│   ├───model
│   ├───profile
│   ├───store
│   └───view
├───build
├───lib
├───resources
└───touch
```

Figura 18 – Estrutura base do projeto Sencha Touch

Além do módulo *app*, existem outros módulos fundamentais na estrutura de uma aplicação desta tecnologia. O diretório *resources* é um desses exemplos e é nele que são armazenados os ficheiros dos recursos da camada visual das aplicações, como por exemplo, os ficheiros de estilos, as fontes de texto, os ícones dos botões e as imagens a utilizar [Ed Spencer, 2012]. O diretório *lib*, por sua vez, tem como propósito armazenar as bibliotecas relativas às dependências externas de uma aplicação. Já as bibliotecas específicas disponibilizadas pela plataforma Sencha para apoiar o desenvolvimento de aplicações na tecnologia encontram-se localizadas sob o diretório *touch*. Finalmente, resta referir o diretório *build*, que é utilizado para armazenar a própria aplicação, ou seja, o resultado obtido após o processo de compilação do código-fonte desenvolvido.

Ainda relativamente à estrutura do projeto em Sencha, fazem parte da sua constituição determinados ficheiros de configuração responsáveis por tarefas tais como empacotar a aplicação de forma a poder ser lançada em meios de distribuição oficiais de plataformas nativas ou como otimizar a execução de uma aplicação. Destes ficheiros destaca-se o app.js, onde são definidos alguns aspetos fundamentais da aplicação, como por exemplo o seu nome, os componentes utilizados, as dependências necessárias e o método que especifica o processamento a realizar no lançamento da aplicação.

Tendo por base a informação recolhida, como primeiro passo de automatização do processo de migração da estrutura foi desenvolvido um método responsável por interpretar o ficheiro app.js e recolher a informação relativa ao nome da aplicação para o incluir na geração do projeto Android. Este método foi desenvolvido na aplicação MobileMigration e a sua definição encontra-se representada no Código Fonte 3.

```
public static String readAppNameFromFile(String senchaFolderPath);
```

Código Fonte 3 – Assinatura do método que lê o nome da aplicação Sencha

Umas das decisões assumidas para o processo de migração foi manter a utilização do padrão de desenvolvimento MVC de forma a aplicar uma boa prática coerente e também compatível com a estrutura anterior, à exceção das camadas *Store* e *Profile* que não se adaptam à plataforma Android. Como tal, a construção da estrutura base do projeto Android teria de cumprir este requisito.

No contexto da aplicação STC, o processo de definição da estrutura do seu projeto Android foi realizado de forma manual. Em primeira instância, foi criado um projeto vazio na ferramenta Android Studio, tendo-se de seguida adaptado a sua estrutura ao padrão de desenvolvimento MVC pretendido. Este processo foi realizado desta forma para que o projeto fosse criado com uma base robusta e otimizada de acordo com a ferramenta de desenvolvimento nativa de Android. A estrutura que resultou deste processo encontra-se ilustrada na Figura 19.

```
│   build.gradle
│   gradle.properties
│   gradlew
│   gradlew.bat
│   local.properties
│   settings.gradle
├──app
│   │   build.gradle
│   │   proguard-rules.pro
│   ├──build
│   ├──libs
│   └──src
│       ├──androidTest
│       └──main
│           │   AndroidManifest.xml
│           ├──java
│           │     ├──Model
│           │     │__ViewController
│           └──res
│               ├──drawable
│               ├──layout
│               ├──menu
│               ├──mipmap
│               ├──raw
│               └──values
├──build
└──gradle
```

Figura 19 – Estrutura base do projeto Android

À semelhança da estrutura Sencha, os ficheiros de configuração do projeto encontram-se no diretório de raiz. No entanto, existe uma diferença considerável neste ponto, uma vez que esses ficheiros assentam nas configurações gerais do projeto ao invés das configurações da aplicação. Na estrutura do projeto Android, as configurações específicas da aplicação encontram-se ao nível do diretório *app* e dividem-se em dois ficheiros. Um desses ficheiros é o manifesto da aplicação, designado *AndroidManifest.xml*, onde se definem as suas configurações principais, como o nome, o ícone de apresentação, as atividades que lhe serão executadas e as permissões

que necessita para interagir com determinados componentes do dispositivo. O outro ficheiro, representado na estrutura como *build.gradle*, representa o ficheiro de configurações Gradle relativo ao processo de compilação da aplicação. O seu conteúdo encontra-se apresentado no Código Fonte 2.

Além das configurações, é o diretório *app* que assenta toda a restante estrutura da aplicação desde o código-fonte desenvolvido até ao seu executável gerado como resultado do processo de compilação. É precisamente no diretório *build* que são armazenados os diferentes resultados de compilação segundo as diferentes variantes definidas na sua configuração. Podem existir vários tipos de compilação, sendo a variante de teste e a variante de produção os mais comuns. Quanto ao diretório *libs*, é nele que se localizam as bibliotecas das dependências externas da aplicação. Já o diretório *src* contém todo o processo de desenvolvimento da aplicação. Esta área é dividida em dois subdiretórios: o subdiretório *androidTest*, responsável por armazenar os ficheiros de teste unitários produzidos, e o subdiretório *main*, que por sua vez contém o manifesto da aplicação, o código-fonte aplicacional, estruturado segundo o padrão MVC no subdiretório *java*, e os seus recursos auxiliares, que se encontram sob o subdiretório *res* [Android Developers-9,2015].

O subdiretório *res* pode ser modularizado segundo os diferentes tipos de recursos utilizados. Os recursos do tipo imagem e outros ficheiros XML que representem formas desenháveis de base Android encontram-se sob o subdiretório *drawable*. No subdiretório *layout*, por sua vez, são armazenados os ficheiros XML que definem determinados componentes visuais ou até mesmo as estruturas integrais das páginas a apresentar pela aplicação. Ainda relativo aos recursos de construção, os que são associados à construção de menus encontram-se armazenados no subdiretório *menu*. Já os recursos como os textos, as cores e os estilos a utilizar na construção das páginas são armazenados no subdiretório *values*. Quanto aos restantes recursos, o ícone de execução da aplicação fica localizado no subdiretório *mipmap*, e os recursos com formato externo, como os ficheiros JSON, devem ser colocados no subdiretório *raw* [Android Developers-9,2015].

Com o intuito de finalizar a apresentação da estrutura do projeto Android, importa realizar uma menção aos diretórios *build* e *gradle* definidos no seu diretório de raiz. No diretório *build* são armazenados os resultados de compilação de todos os módulos aplicacionais pertencentes ao projeto, enquanto no diretório *gradle* encontra-se a biblioteca responsável pela definição e execução do processo de compilação do projeto e dos seus módulos segundo o sistema Gradle [Android Developers-9, 2015].

Através da aprendizagem obtida com o processo realizado, foi desenvolvido um método para a aplicação MobileMigration que permite automatizar a geração de um projeto Android de acordo com a estrutura base apresentada. A definição do método produzido encontra-se representada no Código Fonte 4 e o principal objetivo da sua implementação foi tornar a aplicação MobileMigration capaz de estabelecer mecanismos de auxílio a migrações futuras desde a base de construção da estrutura das aplicações.

```
public static boolean generateAndroidBaseStructure(String appName,
                                                   String destinationFolderPath);
```

Código Fonte 4 – Assinatura do método que gera estrutura de projeto Android

4.3 Camada de Apresentação

Concluída a fase de construção da estrutura do projeto em Android, foi iniciado o processo de definição da camada de apresentação da aplicação como passo seguinte no processo de migração tecnológica. Entende-se por camada de apresentação a estrutura visual e respetiva forma de interação que uma aplicação proporciona ao seu utilizador.

Nas aplicações Sencha Touch, esta camada é construída recorrendo unicamente à combinação de componentes de construção *web* como é o caso de HTML5, CSS3 e JavaScript. HTML5 é utilizado para a definição da estrutura das diferentes páginas, CSS3 para a customização do seu aspeto e dos seus componentes, e JavaScript para a definição de validações e comportamentos aplicacionais dinâmicos.

Em sentido oposto, a tecnologia Android disponibiliza componentes nativos próprios e já estruturados para cumprir o objetivo de construção de uma camada de apresentação consistente e otimizada para os seus dispositivos. O mesmo se aplica ao processo de customização dos seus componentes e respetivo comportamento. A base tecnológica utilizada consiste na combinação entre a definição e customização visual através de estruturas *Extensible Markup Language* (XML) e implementação dos aspetos comportamentais através da linguagem Java [Android Developers-10, 2015].

Existindo uma diferença tecnológica substancial na construção da camada de apresentação entre as duas plataformas de desenvolvimento, procedeu-se a uma investigação de ferramentas existentes que pudessem auxiliar no processo de migração. Durante o processo de estudo foram encontradas algumas soluções que permitem a criação de aplicações nativas Android através de tecnologias *web*. Uma das soluções encontradas foi a ferramenta NativeCSS [NativeCSS, 2015], que permite a definição dos estilos visuais a implementar através de CSS3. Uma das limitações desta ferramenta é que se foca exclusivamente na vertente de customização visual, ou seja, apenas permite criar estilos CSS3 para uma estrutura Android já desenvolvida. Apresentando-se como mais completa, foi também encontrada a ferramenta NativeScript [NativeScript, 2015], que além de permitir a customização dos componentes via CSS3, permite também a adição de código JavaScript para implementar o seu comportamento desejado. Esta ferramenta perfila-se como um motor bastante interessante e robusto, sendo necessário no entanto, construir a página segundo a estrutura XML, nativa de Android. Além destas soluções de vertente *web*, a própria plataforma Android disponibiliza a possibilidade de definir a sua camada de apresentação através da invocação de páginas HTML5 com todos os seus conteúdos, pela componente nativa WebView [Android Developers-11, 2015].

Apesar das diversas possibilidades encontradas no decorrer do estudo, optou-se por realizar uma transição manual, isto é, a definição da estrutura visual da aplicação Android através dos

seus componentes nativos, aplicando as melhores práticas de construção recomendadas pela plataforma. A principal razão que deu origem a esta decisão foi conseguir responder ao objetivo de conseguir o melhor desempenho possível da aplicação, sendo que a utilização de qualquer uma das restantes alternativas estudadas não davam a garantia de conseguir esse nível desejado. Além disso, mesmo pesando o custo de desenvolvimento como fator de decisão, não era certo que o tempo de aprendizagem fosse mais baixo do que a implementação cem por cento nativa, visto que ainda seria necessário adaptar a estrutura do código-fonte de Sencha Touch, baseada em Ext JS.

No decorrer do processo de construção da camada de apresentação base da aplicação Android foram tidos em conta dois pontos fundamentais. Em primeiro lugar, não foi tido em conta o aspeto da aplicação mas sim a sua qualidade de construção, uma vez que no âmbito da dissertação, o desenvolvimento do protótipo em Android tem como propósito a definição de processos e mecanismos que visem potenciar o desempenho aplicacional e facilitar a migração das aplicações alvo e não a construção de uma aplicação pronta a entrar em produção. Com esse objetivo bem assente, a decisão de melhoria do aspeto visual das aplicações deverá ser realizado como trabalho individual em cada aplicação migrada pelo seu responsável. No sentido oposto, durante esta fase de desenvolvimento foi realizada com a preocupação de construir mecanismos genéricos no projeto Android que possam ser reutilizados em processos de migração futuras, permitindo a redução do custo desenvolvimento e colmatando assim o ponto referido anteriormente, que teria sempre de acontecer dadas as diferenças visuais entre cada uma das aplicações alvo.

A principal funcionalidade genérica desenvolvida foi a construção de uma atividade base com uma estrutura e comportamentos pré-definidos e que pode ser utilizada como extensão para qualquer outra atividade a implementar no futuro. Esta atividade é designada BaseActivity e estende ela própria da atividade AppCompatActivity da biblioteca de suporte de Android, de forma a garantir a compatibilidade dos seus componentes com versões anteriores do sistema operativo.

Seguindo as tendências de apresentação das aplicações móveis mais recentes, um dos componentes implementados na atividade base desenvolvida foi um painel de navegação lateral com base no componente nativo NavigationDrawer de Android [Android Developers-12, 2015]. Este painel define a apresentação de uma lista de opções, ficando responsável por funcionar como menu principal da aplicação e por defeito encontra-se oculto, permitindo uma gestão mais eficaz do espaço disponível em ecrã. Para aceder ao painel, basta simplesmente que o utilizador deslize o dedo da ponta esquerda do ecrã para o meio, sendo necessário o movimento oposto para voltar a ocultá-lo. Adicionalmente, foi implementado outro sistema de acesso ao painel, através de um ícone incluído na barra de topo do ecrã, responsável por mostrar e ocultar o painel consoante a interação do utilizador. A barra de topo foi outro dos componentes incluídos na atividade e além de apresentar o ícone de acesso ao painel, é também definido o título da página, que pode ser dinamicamente alterado ao estender a atividade base.

Para a construção do painel foi estabelecido um *layout* Android que define uma listagem de itens correspondentes às opções de possível desencadeamento. Por sua vez, cada um dos itens da listagem é constituído por um ícone e uma designação. Os valores de cada um dos itens foram definidos no ficheiro *strings.xml*, conforme a estrutura representada no Código Fonte 5.

```
<!-- Menu Lateral(Navigation Drawer) -->
<string-array name="nav_options_items">
    <item>Sincronização</item>
    <item>Turnos</item>
    <item>Configurações</item>
    <item>Impressora</item>
    <item>Sobre</item>
</string-array>
<array name="nav_options_icons">
    <item>@mipmap/icone_sync</item>
    <item>@mipmap/icone_turnos </item>
    <item>@mipmap/icone_config</item>
    <item>@mipmap/icone_printer</item>
    <item>@mipmap/icone_about</item>
</array>
```

Código Fonte 5 – Definição das opções a apresentar no painel de navegação

Após a fase de construção visual, foi implementado na atividade base um método, cujo excerto se encontra no Código Fonte 6, responsável por inicializar e definir o comportamento do painel.

```
public void set(String[] navMenuTitles, TypedArray navMenuIcons) {
    mTitle = mDrawerTitle = getTitle();

    // Inicia o layout do painel de navegação
    mDrawerLayout = (DrawerLayout) findViewById(R.id.drawer_layout);
    mDrawerList = (ListView) findViewById(R.id.left_drawer);

    ArrayList<NavDrawerItem> navDrawerItems = new ArrayList<>();

    // Adiciona itens ao painel de navegação
    if (navMenuIcons == null) {
        for (String navMenuTitle : navMenuTitles) {
            navDrawerItems.add(new NavDrawerItem(navMenuTitle));
        }
    } else {
        for (int i = 0; i < navMenuTitles.length; i++) {
            navDrawerItems.add(new NavDrawerItem(navMenuTitles[i],
                                    navMenuIcons.getResourceId(i, -1)));
        }
    }

    // Define o listener a executar quando seleciona uma opção do painel
    mDrawerList.setOnItemClickListener(new SlideMenuClickListener());

    // Define o adaptador do painel
    NavDrawerListAdapter adapter = new NavDrawerListAdapter(
                                    getApplicationContext(), navDrawerItems);
    mDrawerList.setAdapter(adapter);
    ...
```

Código Fonte 6 – Excerto do algoritmo de inicialização do painel de navegação

Como se encontra apresentado, a inicialização do painel é efetuada através do *layout* criado e dos valores recebidos como parâmetro no método, os mesmos que se encontram ilustrados no

Código Fonte 5. Para efetuar o preenchimento destes valores no painel foram criadas as estruturas auxiliares NavDrawerItem e NavDrawerListAdapter. A classe NavDrawer é responsável por estruturar cada item do painel de acordo com o ícone e designação, enquanto a classe NavDrawerListAdapter cumpre o propósito de popular a coleção de itens obtida no painel. A classe NavDrawerListAdapter tem ainda a particularidade de estender a classe nativa BaseAdapter, que possibilita obter níveis de desempenho otimizados na construção de listas e na interação com os seus elementos.

Também no Código Fonte 6 é representada a definição de um *listener* que executa consoante a opção que for selecionada no painel. Assim, é neste *listener* que é definido o comportamento aplicacional do painel, mais especificamente, a ação que cada opção despoleta, como se pode verificar no excerto transcrito no Código Fonte 7. Algumas das ações implementadas no contexto da dissertação poderão ser reutilizadas como é o caso da opção de sincronização de dados ou de acesso às configurações da aplicação, no entanto, dependendo da lógica de negócio de cada aplicação, decerto existirá uma reestruturação das opções do painel. Sendo esse o caso, deve ser tida em especial consideração a dependência direta da posição do item no painel com a especificação da sua ação.

```
private class SlideMenuClickListener implements ListView.OnItemClickListener {
    @Override
    public void onItemClick(AdapterView<?> parent,
                                        View view, int position, long id) {
        // Efetua uma ação dependendo da posição da opção selecionada.
        switch (position) {
           case 0:
                   SynchronizeInfo synchronization = new SynchronizeInfo(this);
                   synchronization.execute(this);
                   break;
           case 1:
                   Intent turnos = new Intent(this, TurnosActivity.class);
                   startActivity(turnos);
                   finish();
                   break;
           ...
    }
}
```

Código Fonte 7 – Excerto do método que executa ação de cada opção do painel

Além dos componentes visuais referidos, foi também incluída na atividade base a possibilidade de apresentação de uma janela sobreposta com uma mensagem simples e informativa, cujo conteúdo possa variar de acordo com a ação que se encontrar a ser processada. Para a criação dessa funcionalidade, foi desenvolvida a *interface* representada no Código Fonte 8 e que define um método que deverá receber como parâmetros o título e mensagem a utilizar na construção da janela.

```
public interface DialogListeners {
    void showDialogMessage(String title, String message);
}
```

Código Fonte 8 – Interface para despoletar a apresentação de mensagens

Na atividade base da aplicação foi implementada a *interface* criada, isto é, foi definido o processo de construção e respetivo comportamento desejado, através do método apresentado no Código Fonte 9. Como se pode verificar, recorre à classe AlertDialog de Android para apresentar um *popup* simples contendo apenas um título, uma mensagem e um único botão de confirmação, responsável por desvanecer a área acabada de criar. Com esta funcionalidade implementada na atividade base, passa a ser possível a qualquer processo associado despoletar a apresentação deste tipo de janelas, bastando para isso enviar os parâmetros referidos de acordo com a informação que se pretende que o utilizador visualize.

```
@Override
public void showDialogMessage(String title, String message) {
    AlertDialog.Builder builder = new AlertDialog.Builder(this);
    builder.setTitle(title)
            .setCancelable(false)
            .setMessage(message)
            .setPositiveButton(R.string.btn_ok, new
                                    DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialog, int id) {
                    dialog.dismiss();
                }
            });
    AlertDialog messageDialog = builder.create();
    messageDialog.show();
}
```

Código Fonte 9 – Método genérico que apresenta uma mensagem informativa

Relativamente à customização dos componentes de apresentação da aplicação, foi estruturada uma divisão de alguns tipos de recursos em diferentes ficheiros no diretório *values*. O conjunto de cores base da aplicação encontra-se representado no ficheiro *colors.xml*. Seguindo as boas práticas de desenvolvimento de não incluir texto fixo espalhado pela aplicação, foi definido o ficheiro *strings.xml* para definir e agregar todos os valores de texto necessários. Estando todos os textos localizados no mesmo ficheiro, torna-se mais simples alterar o seu conteúdo, assim como permite facilitar o trabalho de implementação de um sistema de internacionalização para a aplicação, se necessário, bastando para tal traduzir todo o texto deste ficheiro. Finalmente, os estilos definidos para a apresentação dos componentes encontram-se localizados no ficheiro *styles.xml*. Esta prática foi implementada para visar facilitar qualquer customização que se deseje implementar em momento posterior.

Além dos mecanismos implementados no projeto Android, foi também desenvolvido um método simples para a aplicação MobileMigration com o intuito de migrar as imagens do projeto Sencha para a diretoria *drawable* no projeto Android. Este processo de transferência de ficheiros é direto e visa unicamente completar a construção base da aplicação através deste módulo, o que não descarta uma intervenção manual posterior, dado que, dependendo do nome dos recursos de cada aplicação, poderá ser necessário efetuar alguma renomeação dos ficheiros ou um ajuste à aplicação através do Android Studio para invocar a imagem correta no código-fonte. A assinatura do método desenvolvido encontra-se definida no Código Fonte 10.

```
public static boolean processResourcesToAndroidProject(String resOriginPath,
                                                        String resDestPath);
```

Código Fonte 10 – Assinatura do método que migra os recursos para o projeto Android

4.4 Estrutura e Persistência de dados

Nesta secção é descrito o processo de implementação relativamente às alterações efetuadas à representação das entidades do modelo e ao processo de armazenamento e manipulação dos seus dados.

Como foi descrito anteriormente no Estado da Arte, na estrutura de Sencha Touch os dados são representados e acedidos através das camadas *Model* e *Store*, respetivamente. Nessa área descritiva poderão ser recordados os conceitos, assim como as estruturas dos ficheiros relativos às duas camadas através do exemplo ilustrado no Código Fonte 1.

A camada *Store* não existe por defeito nem seria implementada na estrutura de Android, uma vez que a interação entre a camada *Model* e a camada *Controller* aplicação será realizada sempre através de uma base de dados SQLite. Assim sendo, o processo de implementação foi realizado em duas fases distintas. Em primeiro lugar foi efetuada a migração das entidades da camada *Model* e numa segunda fase foi efetuado o tratamento das *Stores* com dados pré-definidos.

Como em qualquer migração de estruturas de dados aplicacionais, uma das primeiras problemáticas que são apresentadas é o facto de ser necessário repetir o processo de migração por cada entidade existente no modelo. Numa estrutura com um número de entidades elevado este processo pode tornar-se exaustivo e demorado, aumentando significativamente o tempo de desenvolvimento.

Por outro lado, uma preocupação comum é a migração dos dados em histórico de uma estrutura para a outra. Como os dados destas aplicações se encontram sincronizados com a aplicação BackOffice que é responsável pela garantia da sua integridade e preservação, esta problemática não se aplica no contexto deste processo de migração.

De forma a ultrapassar o problema da migração manual das entidades, foi desenvolvido um processo automático na aplicação auxiliar de migração, capaz de converter as entidades de Sencha em classes Java. Como primeiro passo para implementar esse automatismo, foi necessário perceber a estrutura de origem em Sencha, apresentada no Código Fonte 11, comum a todas as entidades.

```
Ext.define('STC.store.StoreConfiguracoes', {
    extend: 'Ext.data.Store',
    config: {
        model  : 'STC.model.ModelConfiguracoes',
        storeId: 'configuracoes',
        …
    }
}
```

Código Fonte 11 – Excerto da estrutura da store da entidade Configurações em Sencha

Para cada entidade, a partir do seu ficheiro da *store* é possível identificar dois elementos fundamentais: a sua designação, através do valor do atributo “storeId”, e a localização do ficheiro que representa o seu modelo.

```
Ext.define('STC.model.ModelConfiguracoes', {
    extend: 'Ext.data.Model',
    config: {
         idProperty : 'id',
         fields:
['id','nome','descricao','valor','tipo','editavel','visivel','tipo_valor']
  }
});
```

Código Fonte 12 – Estrutura do modelo da entidade Configurações em Sencha

A partir do modelo de uma entidade, ilustrado no Código Fonte 12, é possível identificar os seus atributos e qual deles representa o seu identificador único ou chave primária na nomenclatura de base de dados. No caso das aplicações visadas, todos os valores dos atributos das entidades são definidos com o tipo de dados alfanumérico.

Com a informação base recolhida, o próximo passo consistiu em criar um modelo representativo da mesma entidade, mas adaptada à tecnologia Android. Para cumprir esse objetivo, com a perspetiva de generalização, foi estruturado um *template* que pudesse ser utilizado para criar uma classe Java a partir da interpretação dos dados acima ilustrados.

```
package <PACKAGE>.Model.Persistence;

import android.provider.BaseColumns;

public final class <CLASS_NAME> {

    public <CLASS_NAME>() {}

    // Schema
    public static abstract class <CLASS_SCHEMA> implements BaseColumns {
        public static final String TABLE_NAME = "<TABLE_NAME>";

        public static final String <COLUMN_VAR> = "<COLUMN_NAME>";
    }
    ...
}
```

Código Fonte 13 – Excerto do *template* de migração de uma entidade (Estrutura)

No excerto do *template*, representado no Código Fonte 13, designado *ModelTemplate.txt*, foi criada a sintaxe base de uma classe Java representativa de uma entidade, com a especificidade de se terem definido marcadores com recurso aos caracteres especiais <>. Estes marcadores são utilizados para identificar as zonas do *template* que irão substituídas de acordo com os valores retirados do ficheiro de Sencha.

Os marcadores identificados permitem definir dinamicamente o *package* Java onde irá ser armazenada a classe, a sua designação e o seu *schema*, isto é, uma classe abstrata responsável por representar a sua estrutura, nomeadamente, os seus atributos. No desenvolvimento desta estrutura recorreu-se à implementação da *interface* BaseColumns de Android, que tem a principal vantagem de definir por defeito um atributo designado "_ID", que representa a chave primária de uma entidade.

De forma a efetuar a conversão entre as duas estruturas distintas foi desenvolvido um algoritmo na aplicação MobileMigration responsável por interpretar os ficheiros da *store* e respetivo

modelo de cada entidade pertencente ao projeto Sencha, e, com recurso aos dados lidos e ao *template* especificado, construir as classes Java que irão representar as entidades no projeto Android. Para tal, o método desenvolvido recebe três parâmetros: o diretório onde se encontram os ficheiros da *store* do projeto Sencha, o diretório onde se encontram os ficheiros do modelo do projeto Sencha e o diretório destino pertencente ao projeto Android, onde serão criados os novos ficheiros Java. Este último diretório tem de estar diretamente relacionado com o *package* declarado nas classes de forma a garantir um correto processo de compilação do projeto Android.

```
public static void migrateModelFilesFromFolder(String folderModelPath, String
folderStorePath, String destinationFolder)
```

Código Fonte 14 – Assinatura do método de automação da migração de entidades

Após aplicação do algoritmo representado no Código Fonte 14, todas as entidades de Sencha são migradas para o projeto Android de forma automática com a seguinte representação:

```
package com.example.hugo.stc_android.Model.Persistence;

import android.provider.BaseColumns;

public final class ModelConfiguracoes {

    public ModelConfiguracoes() {}

    // Schema
    public static abstract class ModelConfiguracoesSchema implements BaseColumns
{
        public static final String TABLE_NAME = "configuracoes";

        public static final String COLUMN_NOME = "nome";
        public static final String COLUMN_DESCRICAO = "descricao";
        public static final String COLUMN_VALOR = "valor";
        public static final String COLUMN_TIPO = "tipo";
        public static final String COLUMN_EDITAVEL = "editavel";
        public static final String COLUMN_VISIVEL = "visivel";
        public static final String COLUMN_TIPO_VALOR = "tipo_valor";
    }
    …
}
```

Código Fonte 15 - Estrutura do modelo da entidade Configurações em Android

Tendo completado o processo de representação das entidades no projeto Android, foi necessário logo de seguida pensar no armazenamento, acesso e manipulação dos seus dados. Para cumprir esse objetivo recorreu-se à utilização de uma base de dados SQLite. De todas as formas de armazenamento possíveis que a plataforma Android disponibiliza, a escolha recaiu na utilização da base de dados para o armazenamento de dados locais, pois garante uma elevada privacidade e segurança dos dados, dispõe de um motor que permite atingir níveis elevados de desempenho no acesso aos dados, permite uma organização dos dados que facilita a sua interpretação e utilização e possibilita a utilização de código nativo SQL na obtenção e persistência dos dados [Android Developers-8, 2015].

De acordo com o objetivo de generalização dos procedimentos de migração e após o estudo das melhores práticas relacionadas com a utilização de bases de dados SQLite no sistema Android, foi implementada a seguinte classe no decorrer do processo de desenvolvimento:

```
public class DatabaseHelper extends SQLiteOpenHelper {

    private static DatabaseHelper sInstance;

    public static final int DATABASE_VERSION = 1;
    public static final String DATABASE_NAME = "STC.db";

    public DatabaseHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    public static synchronized DatabaseHelper getInstance(Context context) {
        if (sInstance == null) {
            sInstance = new DatabaseHelper(context.getApplicationContext());
        }
        return sInstance;
    }

    public void onCreate(SQLiteDatabase db) {}

    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {}

    public void insertRecord(String table, Map<String, String> record) {...}

    public void updateRecord(String table, Map<String, String> values,
                             String restrictionExpression,
                             String[] restrictionValues) {...}

    public void deleteRecord(String table, String restrictionExpression,
                             String[] restrictionValues) {...}

    public List<String> getDistinctColumnRecordsFromTable(String table,
                               String[] columns, String restrictionExpression,
                               String[] restrictionValues, String order) {...}

    public Cursor getOrderedRecordsByFilter(String table, String[] columns,
                              String restrictionExpression,
                              String[] restrictionValues, String order) {...}

    public String getSingleColumnValueFomTable(String table, String[] columns,
                              String restrictionExpression,
                              String[] restrictionValues) {...}

    public boolean checkTableExists(String tableName) {...}

    public int getCountRecordsFromTable(String tableName) {...}

    public int getCountRecordsFromTableByFilter(String tableName,
                            String filterExpression, String[] filterValues) {...}

    public boolean executeSqlStatement(String sqlStatement) {...}

    public boolean executeBatchInserts(List<String> listStatements) {...}
}
```

Código Fonte 16 – Classe de interação com a base de dados SQLite

Estendendo a classe SQLiteOpenHelper, disponibilizada pela plataforma Android para auxiliar a criação e gestão de versões de bases de dados SQLite, esta classe permite a customização do comportamento da base de dados na criação e na atualização da sua estrutura. Além disso, utilizando as bibliotecas nativas de Android, o processo de gestão da base de dados é realizado de forma mais eficiente.

Na construção desta classe foi utilizado o padrão de desenvolvimento *Singleton,* que tem como principal objetivo possibilitar a criação de uma única instância de uma classe, sendo utilizada a mesma sempre que for efetuada uma invocação durante o ciclo de vida da aplicação [Avinash Zala, 2015]. No contexto específico, a utilização deste padrão previne que exista mais que uma ligação em simultâneo à base de dados local do dispositivo, que por seu lado, impede a existência de conexões que não foram devidamente fechadas, assim como uma melhor gestão do seu funcionamento.

O objetivo desta classe foi focalizar a interação direta com a base de dados num só local e como tal, foram definidos os métodos de persistência, desde as operações básicas *Create, Read, Update e Delete* (CRUD) a operações mais complexas e cuja necessidade de utilização foi surgindo no processo de desenvolvimento, como por exemplo a pesquisa de dados com recurso a filtros, a execução de instruções nativas de SQL ou a inserção de dados em série. A implementação destes métodos foi realizada de forma genérica para a classe poder ser reaproveitada em qualquer projeto. Sempre que for necessário num novo projeto efetuar uma nova operação com a base de dados e esta não esteja disponível na classe, deve ser adicionada de forma a garantir a sua evolução constante.

Relativamente à utilização de boas práticas, na chamada desta classe deve ser tido em consideração o tempo de processamento necessário. Uma vez que a classe interage diretamente com a base de dados local, sempre que for efetuada uma operação mais complexa, poderão existir tempos de processamento elevados que podem provocar bloqueios da aplicação, pois esta fica a aguardar a resposta. De forma a evitar esta situação, a chamada das operações mais complexas desta classe deve ser realizada através de uma AsyncTask, responsável por criar uma tarefa paralela de fundo que irá efetuar o processamento desejado de forma assíncrona e que, após a sua conclusão, irá comunicar com o processo principal da aplicação, que se encontra também ativo, e lhe enviará a resposta obtida para ser interpretada [Lars Vogel, 2015].

Ainda relativamente à persistência dos dados, importa referir que existe outra fonte de armazenamento neste contexto que é o armazenamento em rede. Uma vez que os dados são sincronizados com a área *web* de BackOffice e a sua base de dados, é nesse componente que a gestão final dos dados é garantida. Dado que esse módulo não se encontra no âmbito da migração, não foi realizado qualquer desenvolvimento nesse sistema, no entanto, fica a referência a esse comportamento, que apresenta uma taxa de significância elevada no que toca à segurança dos dados.

Além da classe DatabaseHelper apresentada, foi também realizado desenvolvimento no módulo de migração Java no que toca à persistência dos dados. De forma a complementar a

estrutura já apresentada das entidades, foi adicionado o seguinte conteúdo ao *template* de migração *ModelTemplate.txt*:

```
public final class <CLASS_NAME> {
    ...
    // SQL
    public static final String SQL_CREATE_QUERY =
            "CREATE TABLE IF NOT EXISTS " + <CLASS_SCHEMA>.TABLE_NAME +
               " (" + <CLASS_SCHEMA>._ID + DBUtils.TYPE_INTEGER +
DBUtils.PRIMARY_KEY + DBUtils.COMMA_SEP + <CLASS_SCHEMA>.<COLUMN> +
DBUtils.TYPE_TEXT + DBUtils.COMMA_SEP + <CLASS_SCHEMA>.<COLUMN_LAST> +
DBUtils.TYPE_TEXT + " )";

    public static final String SQL_DELETE_QUERY =
            "DROP TABLE IF EXISTS " + <CLASS_SCHEMA>.TABLE_NAME;

    public static String sqlInsertQuery(List<String> recordValues) {
        StringBuilder builder = new StringBuilder();

        if(recordValues.size() > 2) {
            for (int i = 1; i < recordValues.size() - 1; i++) {
                builder.append(DBUtils.STRING_DELIMITER)
                        .append(recordValues.get(i).contains("'") ?
recordValues.get(i).replaceAll("'", "''") : recordValues.get(i))
                        .append(DBUtils.STRING_DELIMITER).append(DBUtils.COMMA_S
EP);
            }
        }
        builder.append(DBUtils.STRING_DELIMITER)
                .append(recordValues.get(recordValues.size() -
1).contains("'") ? recordValues.get(recordValues.size() - 1).replaceAll("'",
"''") : recordValues.get(recordValues.size() - 1))
                .append(DBUtils.STRING_DELIMITER);

        return "INSERT INTO " + <CLASS_SCHEMA>.TABLE_NAME +
                " VALUES(" +
                Integer.parseInt(recordValues.get(0)) + DBUtils.COMMA_SEP +
                builder.toString() + ")";
    }
}
```

Código Fonte 17 - Excerto do *template* de migração de uma entidade (Persistência)

Os três atributos adicionados representam três instruções de SQL nativo e foram implementados com o intuito de tornar mais genérico e simples o processo de execução das operações que representam. Dado que a criação e remoção de tabelas não é feita no mesmo momento de utilização da aplicação e que por vezes este processo é repetido várias vezes, como no caso da sincronização de dados, torna-se mais simples invocar os atributos estáticos SQL_CREATE_QUERY e SQL_DELETE_QUERY para efetuar as operações necessárias ao invés de repetir constantemente o mesmo código-fonte pela estrutura da aplicação. Já o último atributo foi adicionado de forma a simplificar a inserção de um registo da entidade através de uma lista de valores, processo que visa aumentar o desempenho aplicacional na receção dos dados via módulo de sincronização.

Por fim, de forma a terminar o processo de migração da estrutura das entidades, resta abordar o processo de passagem dos seus dados pré-definidos. Em Sencha, é possível definir na *store*

de uma entidade quais os dados iniciais que a mesma possuirá por defeito. Estes valores podem ser fixos ou podem ser modificados consoante a utilização da aplicação.

```
Ext.define('STC.store.StoreConfiguracoes', {
    extend: 'Ext.data.Store',
    config: {
        model  : 'STC.model.ModelConfiguracoes',
        storeId: 'configuracoes',
        data:
        [{
            id:'0',
            nome: 'Endereço Web Service',
            descricao:'Indicar o endereço de ligação do Web Service',
            valor: 'http://teste/stcws/infobase.asmx',
            tipo:'Ligação Web Service',
            editavel: 'S',
            visivel: 'S',
            tipo_valor: '1'
        },{
            id:'1',
            nome: 'Número PDA',
            descricao:'Indicar o número do Terminal.',
            valor: '1',
            tipo:'Ligação Web Service',
            editavel: 'S',
            visivel: 'S',
            tipo_valor: '1'
        },
        ...
    }
});
```

Código Fonte 18 – *Store* da entidade Configurações com dados pré-definidos

Uma vez que esta camada não existe em Android e os dados têm de ser associados à entidade via base de dados, foi desenvolvido o seguinte algoritmo na aplicação MobileMigration de forma a migrar os dados para um formato genérico e compatível:

```
public static void migrateStoreFilesFromFolder(String storeFolder, String
destinationFolder)
```

Código Fonte 19 – Assinatura do método de migração dos dados das *stores*

Este método é responsável por converter os ficheiros da *store* de Sencha que tenham dados preenchidos para ficheiros no formato *standard* JSON, reconhecido pela plataforma Android, onde será realizada a sua interpretação e inserção na base de dados correspondente.

```
[{
   "id":"0",
   "nome": "Endereço Web Service",
   "descricao":"Indicar o endereço de ligação do Web Service",
   "valor": "http://192.168.10.185/stcws/infobase.asmx",
   "tipo":"Ligação Web Service",
   "editavel": "S",
   "visivel": "S",
   "tipo_valor": "1"
  },{
   ...
}]
```

Código Fonte 20 – Ficheiro JSON com os dados da entidade Configurações

4.5 Módulo de Configurações

Na atual secção será abordado o processo de migração do módulo de configurações da aplicação. Como foi referido anteriormente nesta dissertação, o módulo de configurações é genérico a todas as aplicações alvo, tendo uma estrutura comum entre si e sendo a sua construção realizada de forma dinâmica, uma vez que depende dos dados definidos na *store* da sua entidade representativa em cada aplicação. Também no decorrer desta dissertação, nomeadamente na secção anterior, foi apresentada a forma como se encontram representados esses dados e o processo de migração aplicado segundo uma estrutura JSON.

Reunidos os pressupostos referidos, como primeiro passo do processo de migração deste módulo, foi necessário desenvolver um mecanismo capaz de interpretar o ficheiro JSON com os dados iniciais das configurações da aplicação. Assim sendo, foi implementada a classe InsertConfiguracoesDefault que se pode caracterizar como uma *AsyncTask* responsável por ler o ficheiro dos atributos presentes no ficheiro JSON e inserir a informação na base dados, onde passa a ser gerida em exclusivo essa informação.

```
@Override
protected Void doInBackground(Object... params) {

    DatabaseHelper mDbHelper = DatabaseHelper.getInstance(mContext);

    try {
        // Inicializa o stream do ficheiro JSON de configurações
        InputStream in = mContext.getResources()
                                 .openRawResource(R.raw.configuracoes);

        // Inicializa reader de JSON para interpretar o ficheiro
        JsonReader reader = new JsonReader(new InputStreamReader(in, "UTF-8"));
        reader.beginArray();

        while (reader.hasNext()) {
            Map<String, String> configuracao = new HashMap<>();
            reader.beginObject();
            while (reader.hasNext()) {
                String name = reader.nextName();
                /* Conjunto de instruções "if" encadeadas que comparam cada
                   chave encontrada no reader JSON com o nome do atributo
                   da entidade e adiciona a combinação ao mapa que irá ser
                   inserido posteriormente na base de dados */
                ...
            }
            reader.endObject();
            // Insere os dados na base de dados
            mDbHelper.insertRecord(ModelConfiguracoes
                                   .ModelConfiguracoesSchema.TABLE_NAME, configuracao);
        }
        reader.close();
    } catch (Exception e) {
        Log.e("ERRO", "Ocorreu um erro ao popular as configuracoes iniciais: "
                                                                + e.getMessage());
    }
    return null;
}
```

Código Fonte 21 – Excerto do método principal da classe InsertConfiguracoesDefault

Tratada a parte de inserção dos dados das configurações na base de dados, foi inicializado o processo da sua apresentação. Porém, antes de detalhar esse processo, é necessário ter em conta o requisito da área de configurações apenas ser acessível consoante introdução de uma senha própria. A especificidade da senha necessária é que ela própria é uma das configurações pré-definidas, sendo necessária a validação com o valor correspondente que se encontra na base de dados. Por norma, esta senha é disponibilizada pelo fornecedor de serviços ao administrador do sistema no processo de configuração inicial da aplicação, que a pode alterar posteriormente ao aceder a esta área. Uma vez que as configurações da aplicação são geridas localmente ao nível de cada dispositivo, é, por base, definida a mesma senha no contexto aplicacional para todos eles, ficando à responsabilidade do administrador a sua gestão em cada um.

De forma a cumprir este pressuposto, foi desenvolvida a classe PasswordDialog, responsável por criar e controlar o comportamento do seguinte componente visual:

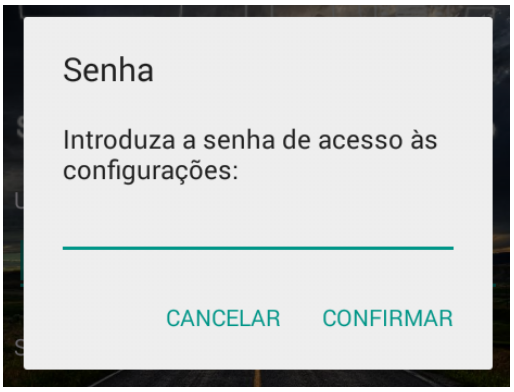


Figura 20 – Fragmento de introdução de senha para acesso às configurações

De forma a possibilitar a criação de um fragmento sobreposto à janela da atividade principal, foi estendida a classe DialogFragment da plataforma Android. O termo fragmento em Android refere-se a um componente visual que é invocado por uma atividade, no entanto o seu controlo e gestão do ciclo de vida é totalmente independente, o que permite a sua reutilização a partir de diversas atividades de uma aplicação. A classe DialogFragment permite a criação de um fragmento sobreposto à atividade que o invocou, apresentando um comportamento visual idêntico a um *popup* em ambientes *web*. Esta classe tem também a vantagem de estar incluída na biblioteca de suporte *appcompat* de Android, pelo que é assegurada a retrocompatibilidade deste componente com versões anteriores do sistema operativo [Mike James, 2015].

Quanto ao conteúdo do fragmento foi utilizado um AlertDialog, um componente nativo de Android que representa uma *interface* simples, de rápida implementação e apropriada à apresentação de informação simples e disponibilização de opções limitadas de interação. Exemplos da sua utilização são janelas de confirmação simples de uma mensagem de erro e a confirmação ou rejeição de uma ação por parte do utilizador. Para a construção da sua estrutura, este componente dispões de um *Builder* interno que permite a definição de um título, uma mensagem e no máximo até três botões com comportamento distinto: um para definir uma

ação positiva, um para representar uma ação negativa e outro para despoletar uma ação neutra. Além destes atributos, é também possível acrescentar elementos adicionais através da injeção de um ficheiro *layout*. No caso do AlertDialog utilizado, além do título, mensagem explicativa e botões de ação positiva e negativa para validar os dados e para fechar o fragmento, respetivamente, foi adicionado um *layout* com uma caixa de texto para o utilizador efetuar a introdução da senha.

```
public class PasswordDialog extends DialogFragment {
    @Override
    @NonNull
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        // Layout customizado
        LayoutInflater inflater = getActivity().getLayoutInflater();

        AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
        builder.setTitle(R.string.dialog_passwd_title)
               .setMessage(R.string.dialog_passwd_msg)
               .setView(inflater.inflate(R.layout.dialog_password, null))
               .setPositiveButton(R.string.btn_ok, new
                                        DialogInterface.OnClickListener() {
                    public void onClick(DialogInterface dialog, int id) {
                       // Verifica password e abre página de configurações se
                          estiver correta
                       ...
                    }
               })
               .setNegativeButton(R.string.btn_cancel, new
                                        DialogInterface.OnClickListener() {
                    public void onClick(DialogInterface dialog, int id) {
                       // Fecha a dialog sem efetuar qualquer acao
                       dialog.dismiss();
                    }
               });
        return builder.create();
    }
}
```

Código Fonte 22 – Excerto da classe PasswordDialog

Desenvolvido o processo de introdução de senha, restou implementar a migração do processo de apresentação das configurações. Tendo em conta que o processo de definição dos valores das configurações não é fixo, a construção da página que as iria apresentar teria também de ser efetuada de forma dinâmica no sistema Android.

Ao invés da habitual definição dos componentes visuais através de um ficheiro *layout* de Android com estrutura XML, a construção da página foi realizada exclusivamente através de código-fonte, produzido na atividade ConfiguracoesActivity. Esta construção foi conseguida através da produção de um fragmento interno à atividade, ConfiguracoesFragment e à sua definição como conteúdo a apresentar na atividade através da seguinte instrução:

```
getFragmentManager().beginTransaction().replace(android.R.id.content, new
ConfigurationsFragment()).commit();
```

Código Fonte 23 – Instrução que define um fragmento como o conteúdo de uma atividade

A plataforma Android disponibiliza um componente cuja estrutura facilita a definição de uma lista de valores que representem as preferências de uma aplicação, apresentando um formato

idêntico às configurações dos dispositivos Android [Rishabh, 2015]. A implementação desse componente é possível através da classe PreferenceFragment, recorrendo à sua extensão para o desenvolvimento do fragmento ConfiguracoesFragment.

Através da utilização da estrutura da classe PreferenceFragment foi possível criar uma hierarquia de configurações agrupada por categorias, o que era o pretendido, uma vez que as configurações em causa, tal como se pode relembrar no Código Fonte 18, têm uma propriedade designada "tipo", que representa o grupo de configurações a que pertence. Analisando ainda as propriedades da entidade representativa das configurações, foi necessário ter em conta, para efeitos de apresentação, os atributos "visível" e "editável". O atributo "visível" é utilizado como filtro ao obter as configurações da base de dados para apenas apresentar os que são marcados com o valor "S". O atributo editável marca a possibilidade de alteração do seu valor, pelo que para cada componente de configuração foi adicionada a propriedade *enabled* de forma a bloquear a edição consoante o definido.

Genericamente, um módulo de configurações é constituído por tipos de estruturas de valores. O presente caso não é distinto, sendo definida a propriedade "tipo_valor" na sua estrutura. Uma vez que se encontra especificada em valores numéricos, foi necessário interpretá-los e adaptá-los para uma estrutura compatível. O valor 0 corresponde a uma *checkbox* e como tal foi definido o componente CheckBoxPreference de Android. Para o valor 1, que corresponde a introdução de texto, foi definido o componente EditTextPreference. Finalmente, para o valor 2, que representa uma lista de valores, foi definido o componente ListPreference.

```
do {
    String tipoComponente = configuracoes.getString(COLUMN_TIPO_VALOR);
    switch (tipoComponente) {
        case "0":
            CheckBoxPreference checkBox = new CheckBoxPreference(getActivity());
            checkBox.setKey(configuracoes.getString(ID));
            checkBox.setTitle(configuracoes.getString(COLUMN_NOME));
            checkBox.setSummary(configuracoes.getString(COLUMN_DESCRICAO));
            checkBox.setEnabled(configuracoes.getString(COLUMN_EDITAVEL)
                                                        .equals("S"));
            checkBox.setDefaultValue(configuracoes.getString(COLUMN_VALOR)
                                                         .equals("1"));
            checkBox.setOnPreferenceChangeListener(preferenceListener);
            if (pos < limite)
                categoria.addPreference(checkBox);
            else
                opcaoSubNivel.addPreference(checkBox);
            break;
        // Restantes tipos de valores
        ...
    }
    pos++;
} while (configuracoes.moveToNext());
```

Código Fonte 24 – Excerto da criação dos componentes de apresentação das configurações

Por fim, após a construção dinâmica do fragmento com todas as configurações da aplicação, foi desenvolvido um mecanismo com comportamento semelhante a um *listener*, responsável por

atualizar o valor de uma configuração na base de dados sempre que este for alterado, de forma a garantir que as edições dos dados são repercutidas no imediato.

```
private Preference.OnPreferenceChangeListener preferenceListener = new
Preference.OnPreferenceChangeListener() {

    @Override
    public boolean onPreferenceChange(Preference preference, Object newValue) {
        // Altera os valores visualizados
        if(preference instanceof ListPreference) {
            ListPreference listPreference = (ListPreference) preference;
            listPreference.setValue(newValue.toString());
            listPreference.setSummary(listPreference.getEntry());
        } else if(preference instanceof EditTextPreference) {
            preference.setSummary(newValue.toString());
        }

        // Guarda novo valor na BD
        DatabaseHelper mDbHelper = new DatabaseHelper(getActivity());

        Map<String,String> valueToUpdate = new HashMap<>();
        valueToUpdate.put(ModelConfiguracoes.ModelConfiguracoesSchema
                                            .COLUMN_VALOR, newValue.toString());

        mDbHelper.updateRecord(
                    ModelConfiguracoes.ModelConfiguracoesSchema.TABLE_NAME,
                    valueToUpdate,
                    ModelConfiguracoes.ModelConfiguracoesSchema._ID + "=?",
                    new String[]{preference.getKey()});

        return true;
    }
};
```

Código Fonte 25 – Algoritmo de alteração de valores das configurações

O resultado final do processo de migração deste módulo traduz-se num ecrã com construção totalmente dinâmica e com código-fonte genérico que pode ser reaproveitado em qualquer uma das aplicações alvo desta dissertação. Um excerto desse ecrã é ilustrado na Figura 21.

Figura 21 – Excerto do ecrã de configurações na aplicação Android

4.6 Módulo de Sincronização

O módulo que se segue neste processo de descrição é o de sincronização de dados com o componente servidor. Como já foi mencionado, um dos pontos comuns entre as aplicações alvo desta dissertação é a comunicação com o componente de BackOffice alojado num servidor. Por sua vez, este componente consiste em uma aplicação *web* que representa um sistema central responsável pelo armazenamento e gestão da informação que circula entre o mesmo e os diversos dispositivos móveis que controlam a aplicação de FrontOffice. Assim sendo, é responsabilidade da aplicação móvel sincronizar devidamente os seus dados com a aplicação *web*. Este processo de sincronização pode ser dividido em duas tarefas:

- Descarregamento dos dados base que são definidos na aplicação *web* e que são fundamentais para o seu correto funcionamento por parte do utilizador. No caso da aplicação STC esses dados consistem na informação relativa aos leitores autorizados, às zonas de atuação ou aos tipos de contraordenações aplicáveis e respetiva legislação em vigor, por exemplo;
- Envio dos dados recolhidos a nível local através da utilização da aplicação móvel para a aplicação *web* de forma a serem armazenados e processados de acordo com a área de negócio em vigor. No contexto da aplicação STC, essa tarefa implica o envio do turno diário de um leitor juntamente com as contraordenações registadas no seu decorrer.

Uma vez que este módulo implica uma interação elevada com a estrutura de dados, a premissa de uma implementação genérica do seu processo de migração é dificultada, dado que foi necessário o desenvolvimento de uma quantidade significativa de lógica específica, especialmente no que diz respeito ao tratamento dos dados, tanto na parte da sua interpretação e armazenamento no momento da receção, como na parte de tratamento necessário para o seu envio de forma a ser compatível com a estrutura que a aplicação *web* espera. No entanto, de forma a contornar esta limitação, foram implementados mecanismos genéricos que podem ser reaproveitados, nomeadamente ao nível das configurações do cliente que garante a comunicação e outros componentes utilizados para auxiliar uma correta integração. Além disso, será descrito o processo implementado de forma conceptual de forma a facilitar uma adaptação do mesmo a uma lógica de negócio diferente.

No contexto das aplicações alvo da migração, como base de comunicação é disponibilizado um serviço *web* pelo sistema de BackOffice que assenta no protocolo SOAP para garantir a troca de informação. Tal como o sistema em si, o serviço também não se encontra no âmbito da dissertação, pelo que foi necessário desenvolver uma solução que cumprisse esse requisito protocolar. Uma vez que a plataforma Android não disponibiliza nenhuma ferramenta nativa com suporte a SOAP, foi necessário iniciar um processo de investigação com o intuito de encontrar uma solução que fosse robusta e compatível com os requisitos técnicos. Como resultado do processo surgiu o projeto ksoap2-android que disponibiliza uma biblioteca de comunicação SOAP leve e adaptada para sistemas Android [Ksoap2, 2105]. Algumas das razões que motivaram a sua escolha foram o facto de ser um projeto que continua a ter desenvolvimento ativo, tendo sido lançadas novas versões da biblioteca ao longo do corrente

ano, e, o facto de se encontrar diversa informação relacionada e tutoriais de utilização em diversas fontes *online*.

Adicionada a dependência da biblioteca ksoap2 ao ficheiro de configuração Gradle, consoante apresentado no Código Fonte 2, foi necessário adicionar algumas permissões ao manifesto do projeto Android de forma que a aplicação fosse capaz de comunicar com um serviço *web*:

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

Código Fonte 26 – Permissões de acesso à rede *web* e ao seu estado

As permissões em causa permitem o acesso à rede *web,* normalmente também denominada como *Internet*, e o acesso ao estado da conectividade do dispositivo móvel de forma a saber se existe alguma ligação disponível para efetuar a comunicação.

Reunidos os pressupostos técnicos, como primeira fase de implementação, foi desenvolvida a classe WebserviceSoapClient para efetuar todas as comunicações com o serviço *web*. É nesta classe que são definidas as configurações para aceder ao serviço e são também implementados os métodos de consumo necessários para a sincronização dos dados. No Código Fonte 27 é apresentado o código-fonte desenvolvido para efetuar a configuração de um cliente genérico para consumir o serviço *web*. Nele são estruturados os métodos responsáveis por construir uma mensagem SOAP e estabelecer a ligação com o serviço *web* para a conseguir transmitir.

```
public class WebserviceSoapClient {

    private String SERVICE_URL;
    private int MAX_TIMEOUT;

    public WebserviceSoapClient(String service_url, String timeout) {
        SERVICE_URL = service_url;
        MAX_TIMEOUT = Integer.parseInt(timeout);
    }

    private SoapSerializationEnvelope getSoapSerializationEnvelope(
                                                        SoapObject request) {
        SoapSerializationEnvelope envelope = new
                                SoapSerializationEnvelope(SoapEnvelope.VER11);
        envelope.dotNet = true;
        envelope.implicitTypes = true;
        envelope.setAddAdornments(false);
        envelope.setOutputSoapObject(request);
        return envelope;
    }

    private HttpTransportSE getHttpTransportSE() {
        HttpTransportSE ht = new HttpTransportSE(Proxy.NO_PROXY, SERVICE_URL,
MAX_TIMEOUT);
        ht.setXmlVersionTag("<?xml version=\"1.0\" encoding=\"utf-8\"?>");
        ht.debug = DEBUG;
        return ht;
    }
    ...
}
```

Código Fonte 27 – Classe com configuração para ligação ao serviço *web*

Em primeiro lugar, foi definido um único construtor para a classe, para que a sua instanciação apenas seja realizada se forem previamente definidos o endereço do serviço e o tempo máximo de tentativa de conexão. Neste caso, estes dois parâmetros correspondem a duas configurações da aplicação e como tal, são utilizados os seus valores que se encontram armazenados na base de dados, tal como demonstrado no Código Fonte 28.

```
DatabaseHelper mDbHelper = DatabaseHelper.getInstance(mContext);

String serviceUrl = mDbHelper.getSingleColumnValueFomTable(
        ModelConfiguracoes.ModelConfiguracoesSchema.TABLE_NAME,
        new String[]{ModelConfiguracoes.ModelConfiguracoesSchema.COLUMN_VALOR},
        ModelConfiguracoes.ModelConfiguracoesSchema._ID + "=?",
        new String[]{"0"});

String operationTimeout = mDbHelper.getSingleColumnValueFomTable(
        ModelConfiguracoes.ModelConfiguracoesSchema.TABLE_NAME,
        new String[]{ModelConfiguracoes.ModelConfiguracoesSchema.COLUMN_VALOR},
        ModelConfiguracoes.ModelConfiguracoesSchema._ID + "=?",
        new String[]{"3"});

WebserviceSoapClient soapClient = new WebserviceSoapClient(serviceUrl,
                                                           operationTimeout);
```

Código Fonte 28 – Exemplo de instanciação do cliente para consumo do serviço

A preocupação seguinte na configuração do cliente traduziu-se na estruturação das mensagens. As mensagens SOAP são representadas através de uma estrutura XML e transmitidas através do protocolo *Hypertext Transfer Protocol* (HTTP). As mensagens podem ser constituídas por quatro elementos principais: *Header*, *Body*, *Fault* e *Envelope*. O elemento *Header* é de utilização opcional e permite a definição de informação adicional, como por exemplo, a especificação de credenciais de autenticação quando o serviço assim o exige para autorizar a receção de mensagens por parte dos seus clientes. O elemento *Body*, de utilização obrigatória, é o elemento responsável por armazenar os dados que se pretendem transmitir nas mensagens SOAP. O elemento *Fault* é definido dentro do elemento *Body* e deve ser apenas utilizado para representar a existência de erros na troca de mensagens. Por último, o elemento *Envelope* é de utilização obrigatória e é responsável por encapsular os elementos XML anteriores num envelope que representará mensagem SOAP a transmitir [SOAP, 2015].

De forma a cumprir o requisito de encapsulamento da mensagem SOAP foi implementado o método getSoapSerializationEnvelope, descrito no Código Fonte 27. Recorrendo a classes disponibilizadas pela biblioteca ksoap2, este método é responsável por criar um *Envelope* através de um objeto SOAP, que representa o conteúdo da mensagem. Além do seu conteúdo foram também definidos outros parâmetros adicionais de forma a auxiliar o correto funcionamento da transmissão da mensagem, tal como a identificação que o serviço com que se pretende comunicar se encontrar disponibilizado através da plataforma .NET. Importa referir ainda que na inicialização do envelope é definida a versão de SOAP a utilizar, que no caso concreto equivale à versão 1.1. Caso seja utilizada a versão 1.2 de SOAP, é necessário alterar este parâmetro. No Código Fonte 29 é apresentado um exemplo da chamada deste método, bem como a necessária definição prévia de um objeto SOAP para incorporar na mensagem com

as devidas propriedades, como o método do serviço a consumir e os seus parâmetros e respetivos valores pretendidos.

```
String metodoWs = "metodo_1";

SoapObject request = new SoapObject(NAMESPACE, metodoWs);
request.addProperty("parametro_1", valor_1);
request.addProperty("parametro_2", valor_2);

SoapSerializationEnvelope envelope = getSoapSerializationEnvelope(request);
```

Código Fonte 29 – Exemplo de criação da mensagem SOAP para envio

Concluída a parte da criação de mensagens SOAP, foi necessário desenvolver o componente que permitisse efetuar a comunicação com o serviço e efetuar a sua transmissão. Para o conseguir, foi aproveitada a classe HttpTransportSE de ksoap2, que define um pedido de transporte HTTP. Para facilitar a sua utilização foi definido o método getHttpTransportSE, apresentado no Código Fonte 27, responsável por criar um pedido HTTP para o endereço *web* definido previamente no construtor da classe. Este método utiliza também o parâmetro relativo ao tempo máximo de espera de resposta por parte do serviço para a criação do pedido. No Código Fonte 30 é exemplificada a invocação deste método e consequente processo de tratamento efetuado após estabelecimento do pedido de transporte HTTP. Este processo inclui o consumo do método do serviço *web* pretendido através da transmissão da mensagem SOAP criada, e também, a interpretação e processamento da resposta fornecida pelo serviço, na forma de um objeto SOAP.

```
try {
    // Cria o pedido de transporte HTTP
     HttpTransportSE ht = getHttpTransportSE();

     // Processa o pedido
    ht.call(SOAP_ACTION + metodoWs, envelope);

    // Trata mensagem SOAP de resposta
    SoapObject resultado = (SoapObject)envelope.getResponse();
    ...
} catch (Exception e) {
    // Trata exceção ocorrida no processo
    ...
}
```

Código Fonte 30 – Exemplo de consumo do serviço e obtenção da resposta

Concluído o processo de configuração genérica para comunicação com o serviço *web*, procedeu-se à implementação da lógica necessária para efetuar a migração do processo de sincronização. A implementação foi realizada da seguinte forma: implementação dos métodos de comunicação com o serviço e toda a lógica de tratamento dos dados na classe WebserviceSoapClient e aplicação da lógica do processo de sincronização, incluindo a invocação dos métodos implementados, numa AsyncTask intitulada SynchronizeInfo.

Para a implementação dos métodos de consumo do serviço foi utilizada como base o processo genérico exemplificado no Código Fonte 29 e Código Fonte 30 para o envio e receção dos dados. Já o processo de tratamento dos dados poderá variar de aplicação para aplicação, dependendo

de como os dados são enviados pelo serviço *web* correspondente. Para colmatar esta limitação, foi aplicada uma filosofia de desenvolvimento com o intuito de criar mecanismos de apoio ao tratamento dos dados que pudessem ser utilizados ou adaptados entre aplicações. Um exemplo desses mecanismos, o mais genérico, foi a implementação de um método na classe WebserviceSoapClient responsável pela conversão dos dados de resposta mais complexos, nomeadamente no processo de descarregamento, para uma estrutura genérica que facilitasse a sua leitura e interpretação. A assinatura do método e a classe representativa da estrutura referida encontram-se representadas no Código Fonte 31.

```
private ResponseObjectWS parseResposta(SoapObject response);

public class ResponseObjectWS {
    private String tabela;
    private int numeroRegistos;
    private int numeroCampos;
    private List<LinkedHashMap<String,String>> registos;
}
```

Código Fonte 31 – Estrutura do mecanismo de conversão das respostas SOAP

Por fim, resta apresentar de forma conceptual a migração da lógica aplicacional relativa ao processo de sincronização. Tal como referido anteriormente, foi utilizada uma AsyncTask para implementar este processo dada a complexidade e o tempo de processamento necessário para o realizar, especialmente na primeira sincronização da aplicação, que necessita de descarregar os dados base disponibilizados pelo servidor.

O processo encontra-se dividido de forma faseada, tendo sido efetuadas algumas melhorias ao longo do processo de migração, especialmente na fase de validação de cada fase. Inicialmente, é realizada uma validação à existência das configurações relativas ao endereço do serviço *web* e ao tempo máximo de espera de resposta, que como foi já referido, são fundamentais para a definição do cliente de consumo. Se não existir o parâmetro relativo ao endereço do serviço, a aplicação apresenta uma mensagem de erro ilustrativa. Já no caso do segundo parâmetro, se não existir é definido um valor por defeito, neste caso, de 60000 milissegundos. De seguida, é verificado o estado da ligação do dispositivo móvel à rede *web*, através do Código Fonte 32.

```
ConnectivityManager connectivityManager = (ConnectivityManager)
                        mContext.getSystemService(Context.CONNECTIVITY_SERVICE);
NetworkInfo activeNetworkInfo = connectivityManager.getActiveNetworkInfo();
if(activeNetworkInfo == null) {
    return mContext.getString(R.string.error_internet_connection);
}
```

Código Fonte 32 – Algoritmo que permite verificar estado da ligação do dispositivo à web

Verificando-se a disponibilidade de uma ligação válida à rede, é invocado o método do serviço para validação do dispositivo. Caso a resposta seja afirmativa, são enviados os dados recolhidos localmente que não tenham sido ainda sincronizados. Após o processo de envio, é realizada uma verificação às datas de controlo de descarregamento. Este processo consiste no descarregamento das últimas datas de descarregamento de cada módulo que se encontram no servidor e compara-as com os dados que se encontram na base de dados. Se existirem módulos cujas datas não correspondam, efetua o descarregamento dessa informação e substitui as

tabelas da base de dados de forma a representar os novos dados. Isto é possível, pois o descarregamento de cada módulo é realizado por cada entidade de dados definida no serviço.

Importa ainda referir que no decorrer de todo o processo de sincronização é utilizada uma ProgressDialog. Este componente nativo de Android permite a apresentação de uma janela sobreposta à principal com informação de um processo a correr em segundo plano. Uma especificidade deste componente é a possibilidade do conteúdo apresentado ser dinâmico, ou seja, permitir a sua alteração ao longo do processo. Neste caso, por cada fase do processo de sincronização é apresentada uma informação de sucesso ou erro do passo para que o utilizador tenha conhecimento de forma imediata do que está a ser processado. É também uma forma de o utilizador ter a noção de que a aplicação se encontra a funcionar devidamente e não se encontra bloqueada.

4.7 Módulo de Autenticação

Depois de implementado o processo de migração aos módulos de configurações e de sincronização, encontravam-se estabelecidos os requisitos necessários para se proceder ao processo de migração do módulo de autenticação. Este é um módulo central da aplicação, pois é o que permite garantir um correto nível de acesso aos utilizadores do dispositivo e impedir que pessoas sem autorização não possam efetuar ações restritas da aplicação.

A página de autenticação é a primeira página com que o utilizador se depara. Através dela apenas tem acesso a um conjunto restrito de funcionalidades. Esse leque de funcionalidades inclui a gestão das configurações da aplicação, mediante introdução de senha de acesso, a sincronização de dados com o servidor e o emparelhamento de impressoras móveis via *Bluetooth*. As restantes funcionalidades das aplicações apenas são desbloqueadas ao utilizador a partir do momento em que se autentica com sucesso.

Com uma estrutura que estende a atividade base, BaseActivity, apresentada na secção 4.3, a atividade de autenticação, designada LoginActivity, é constituída por uma página com um formulário simples com duas caixas de texto para introdução das credenciais de acesso e um botão de confirmação para processar toda a lógica inerente à validação dos dados. Além deste formulário, é também apresentada nesta área uma imagem de fundo, específica da aplicação e da sua imagem comercial.

Como já referenciado, um dos primeiros requisitos para o funcionamento das aplicações alvo desta dissertação é a existência das suas configurações. Como a página de autenticação é a primeira a ser carregada sempre que uma aplicação é executada, foi adicionado um algoritmo na criação da atividade de autenticação que verifica a existência da tabela de configurações na base de dados e, caso não se verifique a sua existência, é imediatamente realizada a sua criação e importação do ficheiro JSON com os dados pré-definidos. Este algoritmo é apresentado no Código Fonte 33 e como se poderá perceber, é realizada a execução de uma AsyncTask, previamente apresentada no Código Fonte 21 e que é responsável pelo tratamento do ficheiro com os dados pré-definidos das configurações.

```
// Cria e popula tabela de configurações se não existir
DatabaseHelper mDbHelper = DatabaseHelper.getInstance(this);
if(!mDbHelper.checkTableExists(ModelConfiguracoes.ModelConfiguracoesSchema
                                                        .TABLE_NAME)) {
    mDbHelper.executeSqlStatement(ModelConfiguracoes.SQL_CREATE_QUERY);
    InsertConfiguracoesDefault asyncInsertConfig =
                                        new InsertConfiguracoesDefault(this);
    asyncInsertConfig.execute(this);
}
```

Código Fonte 33 – Algoritmo de verificação da existência de configurações da aplicação

Efetuada a verificação das configurações da aplicação, procedeu-se à migração do processo de validação dos dados de autenticação. Todo o processo de validação original foi respeitado na migração deste módulo, tendo sido implementadas todas as suas fases. As fases de validação são inerentes a qualquer das aplicações alvo, no entanto, as tabelas onde são verificados os dados poderão ter designações ou atributos diferentes, pelo que pode requerer alguma análise e adaptação específica a esse nível em cada caso. Um exemplo dessa possível necessidade é logo a primeira fase de validação, que consiste na verificação da existência da tabela de utilizadores. No caso do STC, a tabela de utilizadores é representada pela tabela Fiscais, ao contrário do que acontece no contexto de outras aplicações. Uma solução para esta problemática poderia passar por uniformizar o modelo das aplicações, passando a definir uma tabela de Utilizadores de estrutura comum a todas elas. No âmbito desta dissertação tal não foi possível uma vez que o modelo estava previamente definido e é dependente da aplicação de BackOffice, que se encontra fora do contexto de ação.

Caso o primeiro passo da validação das credenciais falhe, isto é, caso não seja encontrada a tabela onde estão definidos os utilizadores da aplicação e respetivas credenciais válidas, então é porque não foi efetuada qualquer sincronização dos dados. Sendo esse o caso, é despoletada uma mensagem de erro a utilizador com essa informação. Pelo contrário, se a tabela de utilizadores existir, é realizada uma validação à versão da aplicação. Esta informação é interna da aplicação e é realizada uma comparação com a versão enviada pelo servidor no descarregamento dos dados. Se as versões não forem idênticas então é apresentada uma mensagem ao utilizador para proceder à atualização da aplicação para a última versão no dispositivo móvel em questão de forma a conseguir efetuar o processo de autenticação.

Após as verificações de fundo, são então validadas as credenciais introduzidas pelo utilizador de acordo com os dados existentes na base de dados. Se a combinação entre nome de utilizador e senha existir, é ainda realizada uma validação adicional à autorização do utilizador introduzido para o dispositivo móvel em questão. Para isso, é utilizado o identificador do dispositivo definido nas configurações da aplicação pelo administrador do sistema.

Por fim, caso todas as validações sejam bem-sucedidas, é despoletada uma última verificação, à existência de turnos ativos para o utilizador autenticado. Caso não exista é criado um novo e só depois dessa ação o utilizador consegue prosseguir para a atividade seguinte. É importante salientar que esta última validação e respetiva ação é específica da aplicação STC. Para as restantes aplicações alvo deverá ser adaptado o comportamento a seguir após validação dos dados de autenticação, de acordo com os requisitos de cada caso.

4.8 Outros módulos

Nesta secção da fase de implementação será descrita uma abordagem conceptual ao restante processo de migração implementado. Dentro desse contexto serão incluídos o módulo de captura de fotos e o módulo de impressão. Além desses dois módulos, todos os componentes restantes da aplicação incidem sobre a lógica de negócio específica. Nesse sentido, uma vez que o objetivo do plano de migração estabelecido era a generalização de processos, não será dado um foco descritivo ao desenvolvimento manual realizado fora do seu âmbito. No entanto, importa referir que mesmo fora do âmbito do plano de migração, esse processo de desenvolvimento foi também importante para conseguir efetuar os testes comparativos ao desempenho e comportamento das aplicações nas duas tecnologias. Sem esses dados não seria possível apresentar conclusões relativamente à estratégia tecnológica adotada para responder à problemática essencial que deu origem a esta dissertação.

4.8.1 Módulo de Captura de Fotos

Uma das limitações iniciais evidenciadas pelo desenvolvimento de aplicações móveis através de tecnologias multiplataforma era a dificuldade de comunicação com os componentes nativos do dispositivo. Um dos exemplos dessas dificuldades encontradas era conseguir controlar a câmara fotográfica de um dispositivo através da aplicação de forma capturar uma simples foto

Com o aparecimento de tecnologias como o Phonegap e o Apache Cordova, algumas das limitações encontradas anteriormente foram colmatadas. Para o conseguir, estas tecnologias disponibilizam bibliotecas ou *plugins* próprios capazes de efetuar a comunicação com determinados componentes dos dispositivos móveis, conseguindo em alguns casos atingir o mesmo nível de controlo que uma aplicação nativa consegue por defeito. No caso das aplicações multiplataforma alvo desta dissertação, essa foi a metodologia utilizada para garantir uma correta integração com alguns componentes nativos dos dispositivos, como é o caso da câmara fotográfica.

No âmbito do desenvolvimento de aplicações móveis recorrendo a tecnologias nativas não é necessária a utilização de qualquer biblioteca adicional para garantir o controlo dos componentes nativos dos seus dispositivos alvo. No caso da plataforma Android, para garantir o acesso e o controlo da câmara fotográfica basta definir as configurações do Código Fonte 34 ao manifesto da aplicação.

```
<uses-permission android:name="android.permission.CAMERA" />
<uses-feature android:name="android.hardware.camera"
              android:required="false" />
```

Código Fonte 34 - Configurações para utilização da câmara fotográfica

Ao contrário de outros componentes em que apenas é necessária a definição da permissão de utilização, para garantir uma correta integração com a câmara, é necessário assinalar essa funcionalidade como requisito aplicacional. Através do atributo *required* desta propriedade é definida a obrigatoriedade de o aparelho deter uma câmara fotográfica para conseguir instalar

a aplicação [Android Developers-13, 2015]. Neste contexto, a utilização da câmara não é uma função crítica nem obrigatória, muito pelo contrário, e como tal, qualquer dispositivo poderá utilizar a aplicação mesmo não incluindo uma câmara fotográfica nas suas especificações técnicas. No entanto, para manter a coerência aplicacional e seguir as boas práticas recomendadas, foi implementada a validação do Código Fonte 35 à funcionalidade associada à captura de fotos, que só permite a sua utilização se existir alguma câmara traseira no dispositivo.

```
if(getPackageManager().hasSystemFeature(PackageManager.FEATURE_CAMERA)) {
    // O dispositivo tem uma câmara traseira disponível
    ...
} else {
    // Apresenta dialog com mensagem de erro
    ...
}
```

Código Fonte 35 – Excerto do algoritmo de validação de existência de câmara

Ultrapassada a questão da configuração da câmara, procedeu-se à implementação da funcionalidade de captura de fotos, tendo em conta os parâmetros aplicacionais definidos para este módulo. Um desses parâmetros é o repositório onde ficam armazenadas as fotos. Uma vez que a sua localização é externa ao repositório da aplicação, foi necessário acrescentar uma nova permissão, representada no Código Fonte 36, ao manifesto para permitir a escrita no repositório em questão.

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

Código Fonte 36 – Permissão para escrita de ficheiros em repositório externo

Antes de terminar, importa ter em atenção uma possível oportunidade de melhoria a implementar neste módulo. As dimensões das fotos recolhidas estão a ser definidas através dos parâmetros estabelecidos nas configurações da aplicação e assim continua de forma a manter a coerência de lógica com a aplicação origem e continuar a possibilitar a escolha dinâmica dos valores por parte do utilizador. No entanto, nem todas as câmaras permitem a mesma resolução de imagens e desse modo, para evitar possíveis problemas aplicacionais, a escolha da resolução poderia ser realizada tendo em conta os tamanhos de imagem suportados pela câmara em utilização ao invés de um valor introduzido manualmente. Essa validação poderia ser conseguida com a criação de uma lista de valores preenchida com o resultado do algoritmo especificado no Código Fonte 37.

```
Camera camera = Camera.open();
Parameters params = camera.getParameters();
List<Camera.Size> sizes = params.getSupportedPictureSizes();
```

Código Fonte 37 – Algoritmo que permite obter os tamanhos de fotos suportados

4.8.2 Módulo de Impressão

À semelhança do módulo de captura de fotos, também o módulo de impressão foi implementado nas aplicações de origem multiplataforma com o auxílio do *plugin* de Cordova para interligação com o componente de *Bluetooth* dos dispositivos. É através deste

componente que é possível detetar e comunicar com impressoras que suportem a mesma tecnologia. Na plataforma Android, mais uma vez, não é necessário recorrer a qualquer *plugin* ou biblioteca, sendo que para aceder ao *Bluetooth* de um dispositivo basta definir as suas permissões de acesso e controlo necessárias, como é assinalado no Código Fonte 38, ao manifesto da aplicação.

```
<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
```

Código Fonte 38 – Permissões para utilização da tecnologia *Bluetooth*

Após configurar as permissões necessárias, foi implementado um mecanismo capaz de procurar por aparelhos com a tecnologia *Bluetooth* ligada nas proximidades do dispositivo, permitir a seleção e respetivo emparelhamento de um deles e possibilitar o envio dos dados para impressão. Para o conseguir recorreu-se à biblioteca nativa de Android que possibilita a interação com esta tecnologia, mais especificamente, às suas classes BluetoothAdapter, BluetoothDevice e BluetoothSocket. A classe BluetoothAdapter permite controlar o *Bluetooth* do dispositivo que se encontra a executar a aplicação. Através desta classe é possível descobrir outros dispositivos com a mesma tecnologia, representados programaticamente pela classe BluetoothDevice. Por fim, a classe BluetoothSocket permite criar um canal de comunicação com um outro dispositivo para efetuar o envio de informação [Android Developers-14].

No Código Fonte 39 é apresentado um excerto do algoritmo genérico desenvolvido para listar os aparelhos com *Bluetooth* ativa encontrados nas proximidades do dispositivo móvel em causa.

```
BluetoothAdapter mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();

// Se Bluetooth do dispositivo estiver desligado, liga-o automaticamente
if (!mBluetoothAdapter.isEnabled()) {
    Intent enableBluetooth = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
    startActivityForResult(enableBluetooth, 0);
}

// Lista os aparelhos com Bluetooth já reconhecidos pelo dispositivo
Set<BluetoothDevice> pairedDevices = mBluetoothAdapter.getBondedDevices();
// Se existirem, apresenta-os na lista para seleção
if (pairedDevices.size() > 0) {
    for (BluetoothDevice device : pairedDevices) {
        listDevices.add(device.getName() + "\n" + device.getAddress());
    }
} else {
   // Caso contrário, inicia processo de descoberta de novos dispositivos
   ...
}

listView.setAdapter(new ArrayAdapter<String>(this,
                                    R.layout.printer_list_items, listDevices));
```

Código Fonte 39 – Excerto de código para listar dispositivos Bluetooth encontrados

Depois de listados os dispositivos e efetivando-se a seleção e respetivo emparelhamento de um deles, o passo seguinte de implementação passou pelo envio dos dados para impressão quando necessário. Em primeiro lugar, foi necessário desenvolver um mecanismo capaz de estabelecer um canal de ligação ao dispositivo emparelhado. Um excerto do mecanismo conseguido

encontra-se no Código Fonte 40. Como é possível verificar, na criação do canal de comunicação através de uma instância da classe BluetoothSocket é definido um identificador. O identificador em questão é específico de dispositivos periféricos como impressoras e *scanners.* Desta forma é possível identificar o tipo de dispositivo com que se pretende comunicar.

```
UUID uuid = UUID.fromString("00001101-0000-1000-8000-00805F9B34FB");
BluetoothSocket mSocket = mDevice.createRfcommSocketToServiceRecord(uuid);
mSocket.connect();
OutputStream mmOutputStream = mSocket.getOutputStream();
InputStream mInputStream = mSocket.getInputStream();
```

Código Fonte 40 – Excerto de código para estabelecer uma ligação Bluetooth

Como passo final, depois de estabelecido o canal de comunicação, foi necessário enviar o conteúdo formatado segundo os requisitos desejados e com conteúdo que fosse reconhecido e processado convenientemente pela impressora. Na aplicação multiplataforma encontra-se implementado um *plugin* customizado, designado WoosimPlugin, que trata estas questões recorrendo a funções da biblioteca disponibilizada pelo fabricante das impressoras utilizadas, a organização Woosim. Uma vez que o *plugin*, tal como todos os restantes de Cordova, se encontra desenvolvido em Java, facilmente foi adaptado e migrado para a aplicação Android, bastando para isso uma pequena revisão do código-fonte e a importação da biblioteca específica das impressoras para as dependências do seu projeto. De notar que a classe migrada poderá ser reaproveitada no processo de migração das outras aplicações alvo à exceção da definição e formatação dos dados enviados, que deverá variar de acordo com a área de negócio.

4.9 Conclusão do Plano de Migração

No capítulo que agora termina foi apresentado o processo de implementação desenvolvido no decorrer da dissertação com o intuito de conseguir cumprir os objetivos propostos.

O processo de implementação consistiu na definição das linhas gerais necessárias à migração das aplicações alvo de uma metodologia híbrida para uma metodologia nativa, nomeadamente para a plataforma Android.

Com o intuito de conseguir o melhor desempenho possível para resolver os problemas comportamentais apresentados pelas aplicações multiplataforma, foram utilizadas sempre que possível as melhores práticas e componentes nativos recomendados.

Além dos pontos anteriores, foi desenvolvida a aplicação MobileMigration com o propósito de automatizar o processo de migração que, conjuntamente com os componentes e mecanismos genéricos implementados no projeto Android, poderá possibilitar uma redução significativa do custo de desenvolvimento do processo de migração das restantes aplicações alvo.

Conjugando toda a informação descrita, esta aplicação, cuja camada visual se encontra representada na Figura 22, apresenta-se no final do processo de implementação com as seguintes funcionalidades:

- Interpretação do nome da aplicação a migrar através do ficheiro de configuração do projeto Sencha Touch;
- Geração da estrutura base do projeto Android. Na geração da estrutura são agregados e incluídos todos os mecanismos e componentes genéricos desenvolvidos para o projeto Android no decorrer do processo de migração desta dissertação. O objetivo desta funcionalidade foi conseguir gerar um projeto Android com todas as funcionalidades migradas e outros módulos genéricos desenvolvidos de forma a potenciar ao máximo a reutilização dos procedimentos e reduzir significativamente o tempo de desenvolvimento de migrações futuras;
- Migração das entidades da aplicação alvo no formato de Sencha Touch para entidades Java e inclusão das mesmas no projeto Android;
- Migração dos dados das entidade da aplicação, quando pré-definidos nas *stores* do projeto de Sencha Touch, para ficheiros JSON de forma a serem processados na base de dados através dos mecanismos do projeto Android.

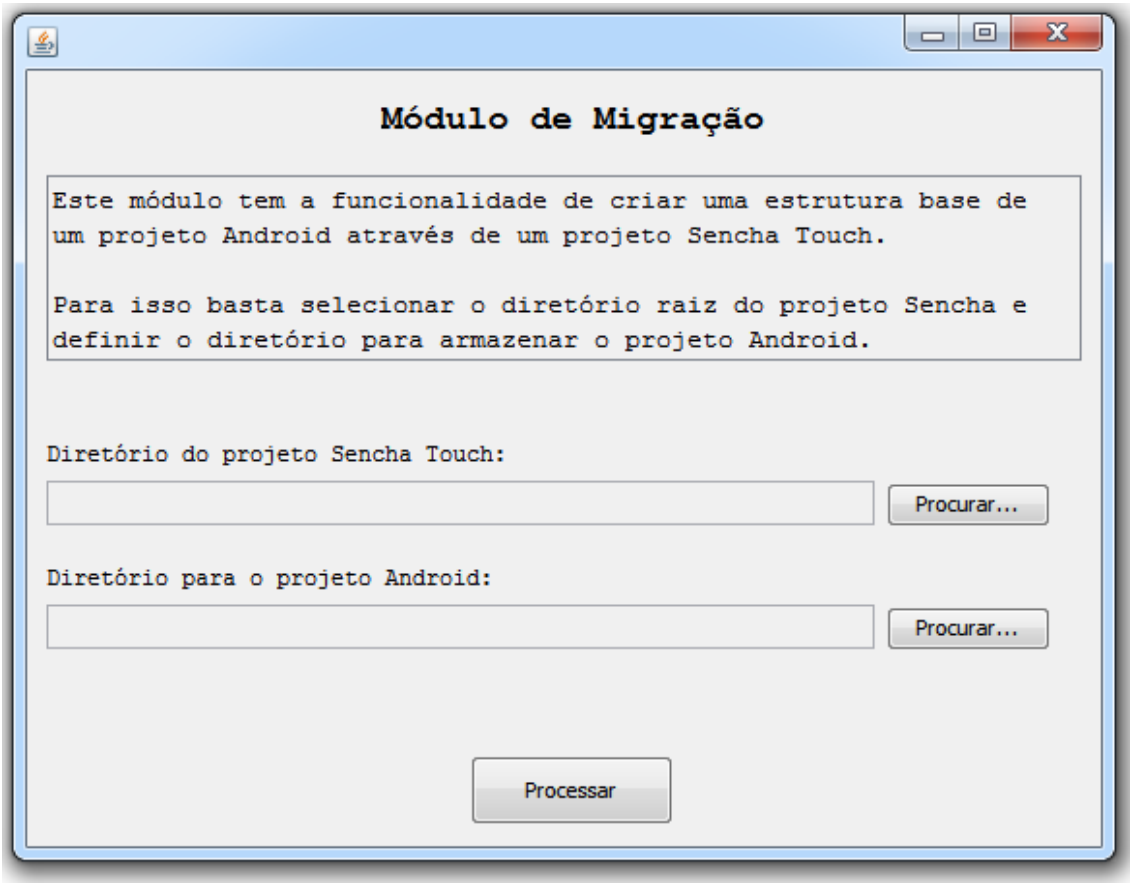


Figura 22 – *Interface* da aplicação MobileMigration

5. Apresentação de Resultados

Na fase inicial da presente dissertação foi descrito o contexto tecnológico encontrado e as suas dificuldades associadas, tendo sido delineada uma estratégia para ultrapassar essa problemática. A estratégia definida passou pela implementação de um plano de migração tecnológica para aplicações móveis desenvolvidas sob uma vertente multiplataforma híbrida para uma vertente nativa em Android.

A implementação da estratégia seria aplicada a uma aplicação piloto, a aplicação STC, e teria de ser realizada tendo em conta dois objetivos principais. Um desses objetivos passava por conseguir resolver os problemas de baixo desempenho e comportamentos visuais indesejados. Por outro lado, o processo teria de ser realizado segundo uma filosofia genérica e de reaproveitamento de forma a conseguir reduzir os tempos de desenvolvimento aquando da migração tecnológica das restantes aplicações da organização.

Concluído o processo de implementação, nesta secção serão apresentados os resultados conseguidos que evidenciarão o sucesso ou insucesso da estratégia desenvolvida para atingir os resultados esperados.

Para a verificação do objetivo relacionado com o comportamento da aplicação móvel, a melhor forma de analisar os resultados conseguidos é através da utilização do protótipo resultante do processo de implementação, comparativamente com a aplicação híbrida. Importa relembrar que o protótipo desenvolvido no decorrer do processo de implementação foi construído segundo a mesma lógica funcional que a aplicação de origem. Isto quer dizer que, apesar de o seu desenvolvimento ter sido realizado segundo as melhores práticas e utilizando os componentes nativos recomendados pela plataforma Android, o seu comportamento e os seus dados são exatamente os mesmos. Foi precisamente para conseguir um comparativo mais coerente que essa estratégia foi adotada.

5.1 Critérios de Comparação

Para se realizar o comparativo entre as duas aplicações, foram tidas em conta duas variáveis fundamentais: o nível de desempenho e o comportamento dos componentes visuais. Uma vez que a medição de comportamento visual é dependente da utilização e não é transponível para dados concretos, será mais à frente realizada uma breve descrição acerca das diferenças sentidas relativamente a esse ponto.

Inicializando então o processo de apresentação de resultados para o primeiro objetivo, foi realizado um teste comparativo ao desempenho de determinadas áreas da aplicação. Entenda-se como teste de desempenho o tempo que a aplicação demora a efetuar determinada tarefa, desde o despoletar da ação pelo utilizador, até à apresentação do resultado. Para efetuar o registo dos tempos de resposta foi utilizada um relógio com cronómetro.

O método utilizado para este teste consistiu na realização de cinco medições do tempo de resposta de cada área em cada dispositivo, tendo sido recolhido o melhor resultado em cada uma das combinações. Este método foi o escolhido, pois os valores conseguidos nas cinco medições de cada combinação variavam entre si por milissegundos, pelo que a variação dos piores resultados conseguidos com os melhores foi bastante baixa, optando assim pela apresentação do melhor ao invés de uma média, por exemplo.

Importa ainda referir que para a execução de cada medição foram limpos os dados e a *cache* de cada aplicação para garantir que os tempos eram cronometrados de acordo com as mesmas condições e o mesmo volume de dados.

As áreas selecionadas para este teste tiveram em conta fatores como a criticidade da funcionalidade, a complexidade de processamento da tarefa ou o volume de dados envolvidos. Como tal, foram então selecionadas as seguintes áreas:

- **Arranque** – representa o processo de arranque da aplicação, desde a sua seleção no menu de aplicações até à apresentação efetiva do ecrã principal da aplicação;
- **Configurações** – área de configurações da aplicação. Nesta área foi medido o tempo de construção dinâmica dos componentes e da sua apresentação;
- **Sincronização** – representa o processo de sincronização de dados, tendo sido cronometrado o processo de descarregamentos dos dados base com a aplicação sem qualquer informação. A medição desta área visou testar uma operação de alta complexidade, com múltiplas validações e responsável pelo tratamento de uma quantidade elevada de dados.
- **Autenticação** – representa a totalidade do processo de autenticação, validação das credenciais e apresentação do menu principal. Apesar de ser uma área com um nível de processamento acessível, é uma área crítica e como tal foi considerada para os testes.
- **Listagem de Marcas** – como última área alvo de cronometragem foi necessário incluir uma listagem de dados carregada a partir da base de dados. Uma das áreas onde se detetava alguma lentidão e comportamentos indesejados consistia precisamente na seleção e consulta de listagens com um grande volume de dados, especialmente se acedida várias vezes no mesmo ecrã. No contexto da aplicação STC, a listagem que representava o maior volume de elementos foi a lista de marcas de veículos.

5.2 Resultados

Os resultados obtidos na execução dos testes comparativos ao desempenho das duas aplicações encontram-se na Tabela 1, estando os tempos representados em segundos.

	STC Híbrido	STC Android
Arranque	4,85	1,78
Configurações	1,83	0,58
Sincronização	8,15	6,03
Autenticação	1,19	0,46
Listagem de Marcas	2,85	0,1

Tabela 1 – Comparativo de desempenho das duas metodologias

Como referido anteriormente neste capítulo, a outra variável a ter em conta para o cumprimento do primeiro objetivo seria o comportamento visual da aplicação. Sumariamente, na aplicação híbrida, ao despoletar diversas ações sob uma mesma área, a aplicação começa a apresentar um comportamento lento, dando por vezes a ideia que os seus componentes visuais ficam bloqueados e não reagem às ações dos utilizadores. Apesar do nível de desempenho da aplicação não ser o ideal e o comportamento apresentado também depender desse índice, essa razão não é só por si justificável, uma vez que essa perda de fluidez de navegação é visível mesmo em áreas em que os dados já deveriam ter sido todos carregados.

No decorrer dos testes de utilização do protótipo nativo da aplicação a problemática comportamental não foi detetada. A navegação apresentou-se fluída e os componentes nativos responderam de forma rápida e coerente às ações do utilizador, mesmo efetuando diversas interações no mesmo ecrã. Poder-se-á contestar que se trata de um protótipo e que os componentes visuais se encontram construídos de forma mais leve que na aplicação híbrida, no entanto, mesmo tendo em conta essa variável, não justifica uma diferença comportamental tão grande como a existente, especialmente em áreas que foram já implementadas de forma robusta, como o caso do módulo de configurações. Além disso, importa reforçar que o volume de dados e o processamento das ações despoletadas foram exatamente as mesmas na realização deste comparativo comportamental.

Finalizado o processo de apresentação de resultados para o objetivo de resolução de problemas aplicacionais, restam descrever os resultados conseguidos com o intuito de reduzir o custo de implementação de um processo de migração semelhante para as restantes aplicações móveis da organização.

Nesse sentido, como foi detalhadamente descrito no decorrer do processo de implementação, foram utilizados padrões e desenvolvidos mecanismos que possibilitassem o reaproveitamento de código. Um desses mecanismos foi a construção de componentes visuais base para serem utilizados em todas as aplicações, mediante alguma customização de acordo com o estilo pretendido em cada uma delas. Foram também desenvolvidas classes Java genéricas

responsáveis por tratar operações comuns a todas as aplicações e que podem ser reutilizadas integralmente em migrações futuras.

Além dos padrões e mecanismos desenvolvidos, foi também realizada uma abordagem descritiva ao processo de migração dos módulos comuns a todas as aplicações, como evidenciado no desenrolar do capítulo 4. Além do processo descritivo apresentado, desta abordagem resultaram mais mecanismos e componentes reutilizáveis, que culminaram no desenvolvimento da aplicação MobileMigration.

Tendo essa informação como base e para terminar, resta apresentar uma breve descrição ao tempo gasto no processo de migração implementado. Para isso devem-se ter em conta algumas variáveis importantes. O tempo despendido no processo de implementação define-se em aproximadamente cinco meses. Nesses cinco meses inclui-se o tempo de investigação e aprendizagem da tecnologia Android e das suas metodologias de boas práticas. Deve ser também tido em conta nesse tempo o processo de desenvolvimento de mecanismos e componentes genéricos e também da aplicação MobileMigration, sendo que neste último caso foram implementados mecanismos de automação que permitiram encurtar o tempo de desenvolvimento, nomeadamente no processo de migração das entidades e dos seus dados pré-definidos.

Em suma, ao implementar mecanismos que permitissem a redução do tempo de migração futura de outras aplicações, o tempo de desenvolvimento desta dissertação aumentou consideravelmente. Obstante isso, o resultado final de desenvolvimento é um protótipo nativo inacabado da aplicação híbrida original, uma vez que nem todas as funcionalidades específicas da aplicação STC foram incluídas por estar fora do objetivo de especificação de um plano genérico. No entanto, estima-se que essas funcionalidades necessitem de um tempo de desenvolvimento entre duas a três semanas, tendo em conta o número de funções em falta e o seu baixo nível de complexidade inerente.

Conjugando as diferentes variáveis e tendo em consideração a mais-valia da metodologia de implementada no decorrer da presente dissertação e do módulo de migração conseguido como base de desenvolvimento, estima-se que o tempo necessário para migrar cada uma das restantes aplicações móveis se poderá apresentar entre um a um mês e meio de implementação, podendo esse tempo variar consoante a experiência do programador na plataforma Android e das funcionalidades específicas desejadas. Importa referir que o tempo estimado tem em conta apenas um único recurso humano alocado à tarefa.

Desta forma é dado por terminado o processo de apresentação dos resultados. Com base na interpretação da informação apresentada, conjuntamente com os objetivos iniciais, serão retiradas as conclusões desta dissertação no capítulo que se segue.

6. Conclusões e Trabalho Futuro

Nos últimos anos a utilização de dispositivos móveis tem aumentado de forma acentuada, acompanhando a constante evolução tecnológica da área, que tem sido também realizada a um ritmo vertiginoso. Como é normal em áreas com tanta procura, a oportunidade de negócio é ampla para as organizações conseguirem tentar encontrar o seu espaço no mercado. No caso das organizações da área de desenvolvimento de *software*, o potencial que emana da área das aplicações móveis é extremamente apetecível.

Nesse contexto, como muitas outras organizações, a ANO decidiu investir no desenvolvimento de aplicações móveis para o seu mercado alvo. Com o intuito de conseguir abranger o máximo número de plataformas móveis possíveis foi adotada uma metodologia de desenvolvimento híbrida. A estratégia implementada resultou e as aplicações desenvolvidas foram aceites pelo mercado. No entanto, com o aumentar das necessidades dos clientes e da complexidade das funcionalidades implementadas, começaram a surgir os problemas de desempenho e comportamentais das aplicações. Esgotadas as tentativas de solucionar os problemas segundo a metodologia híbrida, foi necessário proceder à alteração da estratégia a implementar.

Com esse intuito surgiu a presente dissertação de mestrado, cujo resultado final consiste no desenvolvimento de mecanismos genéricos que permitam efetuar a migração das aplicações para a plataforma nativa Android. A solução conseguida visa não só a resolução dos problemas de desempenho e comportamentais das aplicação como também permitir definir um processo de implementação que possa ser aplicado em todas as aplicações alvo com tempos de desenvolvimento baixos.

6.1 Conclusões

Através da interpretação dos resultados obtidos no âmbito da presente dissertação de mestrado podem-se retirar algumas conclusões interessantes. Em primeiro lugar, é possível verificar que em todos os testes de desempenho realizados a aplicação nativa conseguiu os melhores tempos de execução. Aliando esse fator ao nível superior de fluidez comportamental apresentado pelos componentes da aplicação nativa, é também possível concluir que a estratégia de alteração da metodologia de desenvolvimento das aplicações móveis foi uma aposta acertada. Neste seguimento, também se pode concluir que com a adoção desta metodologia para as restantes aplicações móveis alvo, a percentagem de os seus problemas comportamentais e de desempenho também ficarem resolvidos é bastante elevada. Apesar do sucesso conseguido neste ponto e dos resultados serem esclarecedores e conclusivos, devido a questões relacionadas com limitação de tempo, os testes de cronometragem ao desempenho da aplicação não foram realizados da forma exaustiva que se pretendia inicialmente. Idealmente, o número de medições realizadas deveria ser superior e o número de áreas testadas também.

Juntamente com a verificação dos resultados comportamentais da aplicação foi realizado um estudo acerca do tempo despendido no processo de implementação. Conjugando os dados apresentados nesse estudo com a metodologia pormenorizadamente descrita e as funcionalidades conseguidas através aplicação MobileMigration desenvolvida, pode-se concluir que o trabalho desta dissertação resultou na construção de uma base robusta e eficaz que permite facilitar e reduzir substancialmente o tempo de implementação de migrações tecnológicas a aplicações móveis desenvolvidas sob a mesma base estrutural e funcional.

Em suma, tendo como base os resultados apresentados, é possível concluir que a solução resultante da presente dissertação de mestrado permite atingir todos os objetivos propostos.

6.2 Limitações

Mesmo alcançando os objetivos propostos, a verdade é que a solução conseguida apresenta algumas limitações, sendo a de maior destaque o facto de não se ter conseguido implementar um processo de migração integral. Isto quer dizer que, mesmo utilizando toda a base de construção aplicacional conseguida e os mecanismos de automação desenvolvidos, não é suficiente para concluir o processo de migração total das diferentes aplicações com sucesso. Para conseguir finalizar esse processo é necessário proceder a adaptações ao tratamento dos dados e à implementação de novas funcionalidades de acordo com a variação apresentada pela lógica de negócio de cada aplicação alvo.

Essa principal limitação ocorre devido à combinação de diversos fatores verificados. Por um lado, as diferenças conceptuais de construção e desenvolvimento aplicacional entre as duas tecnologias em estudo são imensas, o que dificulta a possibilidade de criar mecanismos para executar uma migração direta do código fonte. Outro fator que justifica a limitação enunciada passa pela metodologia assumida para a implementação da tese. Sendo o seu intuito a criação de um plano de migração genérico e a procura de utilização dos melhores componentes nativos para resolver os problemas comportamentais das aplicações, o foco principal passou pela adaptação dos módulos genéricos utilizando as metodologias que conseguissem o melhor desempenho possível, ficando a implementação das funcionalidades específicas da lógica de negócio da aplicação para segundo plano.

Mesmo tendo em conta as limitações apresentadas e assumidas, pode-se concluir que as mesmas não influenciam a taxa de sucesso da solução conseguida no que toca ao atingir dos objetivos propostos.

6.3 Trabalho Futuro

Tendo em conta as limitações encontradas e tendo em conta a base metodológica conseguida, considera-se que existem alguns pontos de possível evolução que poderiam ser alvo de análise e de implementação.

Um dos pontos a ter em conta para a melhoria da solução seria a continuidade de construção de componentes visuais genéricos e classes de apoio, de forma a tornar a base do projeto Android mais robusta e mais adaptável às funcionalidades específicas de cada aplicação. Com a implementação deste ponto poderia ser possível reduzir ainda mais o tempo de desenvolvimento necessário para as migrações futuras.

Como ponto de evolução, desviando do âmbito tecnológico da dissertação, seria um trabalho interessante a adaptação do processo de implementação para as restantes plataformas móveis nativas, como o caso de iOS e Windows.

6.4 Apreciação Final

Para terminar, a título pessoal, considera-se que a realização da presente dissertação foi importante para o seu autor segundo a perspetiva de evolução profissional, especialmente no que toca à capacidade de análise e de estruturação de processos genéricos para reutilização futura. Também foi importante para melhorar a qualidade de escrita que por vezes fica para segundo plano na área de desenvolvimento aplicacional.

7. Referências

[Android Developers-1, 2015] What is API Level?
http://developer.android.com/guide/topics/manifest/uses-sdk-element.html#ApiLevels
[Último acesso: 23 Set 2015]

[Android Developers-2, 2015] Build System Overview
https://developer.android.com/sdk/installing/studio-build.html
[Último acesso: 23 Set 2015]

[Android Developers-3, 2015] Configuring Gradle Builds
https://developer.android.com/tools/building/configuring-gradle.html
[Último acesso: 23 Set 2015]

[Android Developers-4, 2015] Support Library Features
https://developer.android.com/tools/support-library/index.html
[Último acesso: 24 Set 2015]

[Android Developers-5, 2015] ProGuard
http://developer.android.com/tools/help/proguard.html
[Último acesso: 24 Set 2015]

[Android Developers-6, 2015] RecyclerView
https://developer.android.com/reference/android/support/v7/widget/RecyclerView.html
[Último acesso: 24 Set 2015]

[Android Developers-7, 2015] Saving Data in SQL Databases
http://developer.android.com/training/basics/data-storage/databases.html
[Último acesso: 27 Set 2015]

[Android Developers-8, 2015] Storage Options
http://developer.android.com/guide/topics/data/data-storage.html
[Último acesso: 27 Set 2015]

[Android Developers-9, 2015] Managing Projects Overview
https://developer.android.com/tools/projects/index.html
[Último acesso: 3 Out 2015]

[Android Developers-10, 2015] Layouts
http://developer.android.com/guide/topics/ui/declaring-layout.html
[Último acesso: 4 Out 2015]

[Android Developers-11, 2015] WebView
http://developer.android.com/reference/android/webkit/WebView.html
[Último acesso: 4 Out 2015]

[Android Developers-12, 2015] NavigationDrawer
https://developer.android.com/training/implementing-navigation/nav-drawer.html
[Último acesso: 4 Out 2015]

[Android Developers-13, 2015] Taking Photos Simply
https://developer.android.com/training/camera/photobasics.html
[Último acesso: 16 Out 2015]

[Android Developers-14 2015] Bluetooth
http://developer.android.com/guide/topics/connectivity/bluetooth.html
[Último acesso: 16 Out 2015]

[Android Studio, 2015] Android Studio
https://developer.android.com/tools/studio/index.html
[Último acesso: 5 Out 2015]

[Angular, 2015] Angular.js
https://angularjs.org/
[Último acesso: 3 Jun 2015]

[Annie, 2012] Sencha Touch
http://www.vensi.com/sencha-touch/
[Último acesso: 23 Jun 2015]

[Avinash Zala, 2015] Design Patterns: The Singleton Pattern
http://code.tutsplus.com/tutorials/design-patterns-the-singleton-pattern--cms-23073
[Último acesso: 27 Set 2015]

[Brian, 2012] PhoneGap, Cordova, and what's in a name?
http://phonegap.com/2012/03/19/phonegap-cordova-and-what%E2%80%99s-in-a-name/
[Último acesso: 24 Jun 2015]

[Cordova, 2015] Apache Cordova
https://cordova.apache.org/
[Último acesso: 5 Jun 2015]

[Cordova Docs, 2015] Platform Support
https://cordova.apache.org/docs/en/5.0.0/guide/support/index.html
[Último acesso: 5 Jun 2015]

[Ed Spencer, 2012] Anatomy of a Sencha Touch 2 App
http://edspencer.net/2012/03/19/anatomy-of-a-sencha-touch-2-app/
[Último acesso: 3 Out 2015]

[Gradle, 2015] Build System Overview
https://gradle.org/
[Último acesso: 23 Set 2015]

[Grgur Grisogono, 2014] 5 Best Mobile Web App Frameworks: Sencha Touch
http://moduscreate.com/5-best-mobile-web-app-frameworks-sencha-touch/
[Último acesso: 22 Jun 2015]

[Ionic, 2015] Ionic Framework
http://ionicframework.com/
[Último acesso: 6 Jun 2015]

[Jim Cowart, 2013] Pros and Cons of the Top 5 Cross-Platform Tools
http://www.developereconomics.com/pros-cons-top-5-cross-platform-tools/
[Último acesso: 24 Jun 2015]

[jQueryMobile, 2015] jQuery Mobile
https://jquerymobile.com/
[Último acesso: 3 Jun 2015]

[Kendo, 2015] Kendo UI
http://www.telerik.com/kendo-ui/
[Último acesso: 3 Jun 2015]

[Ksoap2, 2015] ksoap2-android - lightweight, efficient SOAP on Android
http://simpligility.github.io/ksoap2-android/index.html
[Último acesso: 24 Set 2015]

[Lars Vogel, 2015] Android Threads, Handlers AsyncTask
http://www.vogella.com/tutorials/AndroidBackgroundProcessing/article.html
[Último acesso: 27 Set 2015]

[Max, 2014] The Last Word on Cordova and PhoneGap
http://blog.ionic.io/what-is-cordova-phonegap/
[Último acesso: 24 Jun 2015]

[Mike Jacobs, 2011] Living on the Edge of Mobile Development
http://br.sys-con.com/node/1719019
[Último acesso: 3 Jun 2015]

[Mike James, 2014] Android Adventures - Custom Dialogs Using DialogFragment
http://www.i-programmer.info/programming/android/7426-android-adventures-custom-dialogs-using-dialogfragment.html
[Último acesso: 29 Set 2015]

[Mitch Pronshinske, 2014] The State of Native vs. Web vs. Hybrid
http://java.dzone.com/articles/state-native-vs-web-vs-hybrid
[Último acesso: 11 Jun 2015]

[NativeCSS, 2015] NativeCSS
https://nativecss.com/
[Último acesso: 4 Out 2015]

[NativeScript, 2015] NativeScript
https://www.nativescript.org/
[Último acesso: 4 Out 2015]

[Patrick Rudolph, 2014] Hybrid Mobile Apps: Providing a Native Experience with Web Technologies
http://www.smashingmagazine.com/2014/10/21/providing-a-native-experience-with-web-technologies/
[Último acesso: 11 Jun 2015]

[Pietro Saccomani, 2012] Native, Web or Hybrid Apps? What's The Difference?
http://www.mobiloud.com/blog/2012/06/native-web-or-hybrid-apps/
[Último acesso: 10 Jun 2015]

[Phonegap, 2015] PhoneGap
http://phonegap.com/
[Último acesso: 5 Jun 2015]

[Richard Goodwin, 2015] The History of Mobile Phones From 1973 To 2008: The Handsets That Made It ALL Happen
http://www.knowyourmobile.com/nokia/nokia-3310/19848/history-mobile-phones-1973-2008-handsets-made-it-all-happen
[Último acesso: 18 Out 2015]

[Rishabh, 2015] Saving User Settings with Android Preferences (PreferenceActivity, PreferenceFragment, Headers)
http://codetheory.in/saving-user-settings-with-android-preferences/
[Último acesso: 1 Out 2015]

[Rob Gravelle, 2013] Comparing Mobile Development Framework Types
http://www.htmlgoodies.com/html5/mobile/comparing-mobile-development-framework-types.html#fbid=Mvdr2weWSPh
[Último acesso: 10 Jun 2015]

[Sencha Touch, 2015] Sencha Touch
http://www.sencha.com/products/touch/
[Último acesso: 4 Jun 2015]

[SOAP, 2015] XML Soap
http://www.w3schools.com/xml/xml_soap.asp
[Último acesso: 2 Out 2015]

[SQLite, 2015] SQLite
https://www.sqlite.org/
[Último acesso: 27 Set 2015]

[StatCounter, 2015] StatCounter Global Stats
http://gs.statcounter.com/#all-comparison-ww-monthly-201201-201503
[Último acesso: 10 Abr 2015]

[Venkata Kiran, 2013] Native vs Mobile Web vs Hybrid applications
http://codingsquare.blogspot.pt/2013/08/native-vs-mobile-web-vs-hybrid.html
[Último acesso: 12 Jun 2015]