UNIVERSIDADE DE LISBOA

FACULDADE DE CIÊNCIAS

DEPARTAMENTO DE INFORMÁTICA

UNIVERSAL WINDOWS APPLICATION – APOIO À INSPEÇÃO

David Manuel da Costa Berto

Mestrado em Engenharia Informática
Especialização em Engenharia de Software

Versão Pública

Trabalho de Projeto orientado por:
Professora Doutora Ana Paula Pereira Afonso
e Engenheiro Eduardo Miguel Neto Alves Silvério

2018

Agradecimentos

A todos os elementos da Accenture Consultores de Gestão SA. que estavam presentes no cliente onde decorreu o projeto, pelo excelente ambiente de trabalho.

Aos elementos da minha equipa de projeto, com especial atenção para o Vítor Carabineiro e o Filipe Cardoso que me acolheram desde o início e estiveram sempre prontos para me ajudar.

Ao Pedro Couto e ao Rui Sequeira pela simpatia e ajuda prestada tanto na escrita da tese como no desenvolvimento da aplicação.

Ao Diogo Antão pela companhia nas pausas para o lanche.

Ao meu colega de faculdade Rui Poeiras por me ter acompanhado nesta experiência.

Aos meus pais e irmã que sempre me apoiaram e deram tudo para que tivesse sucesso na minha formação académica.

Ao meu avô Ernesto e à minha avó Arlete todo o apoio e acompanhamento durante esta fase da minha vida.

À minha namorada Inês pela paciência que tem para me aturar e por estar presente aconteça o que acontecer.

Ao meu orientador da empresa Eduardo Silvério pela ajuda a envergar no mundo do trabalho.

À minha orientadora da faculdade, professora Ana Paula Afonso por toda a ajuda prestada no desenvolvimento desde documento.

Aos meus amigos e todos os que me ajudaram direta ou indiretamente na realização do projeto de mestrado, o meu obrigado.

Aos meus avós.

Resumo

O processo de inspeção efetuado pelo organismo público onde decorreu este projeto, engloba atualmente diversos tipos de atividades. Cada uma destas atividades tem a sua prória forma de ser executada consoante as equipas envolvidas, não existindo uma uniformização estabelecida.

Para além disso, as equipas necessitam de uma grande quantidade de documentos cada vez que se deslocam ao exterior, o que torna o processo de inspeção uma tarefa mais morosa e complexa. Todo este processo encontra-se desenquadrado com as tecnologias atuais, sendo por isso necessário uma reformulação ao setor. Surge aqui uma oportunidade de fazer uso das tecnologias móveis, que têm tido um crescimento ao longo dos últimos anos.

O presente relatório documenta a solução tecnológica realizada, com recurso a *tablets* Windows 10 através do uso do *Universal Windows Platform* (UWP), ao longo dos nove meses de projeto no cliente da empresa Accenture Consultores de Gestão SA. Tem como objetivo a desmaterialização dos documentos físicos usados atualmente, a possibilidade de criar uma nova "*toolbox*"para os inspetores trabalharem e finalmente o melhoramento das diversas atividades de inspeção.

Neste documento vão ser destacados os conceitos mais importantes, as tecnologias utilizadas durante o projeto, assim como a investigação feita no começo do estágio às alternativas usadas atualmente no mercado. Nos restantes capítulos é exposto o trabalho desenvolvido, as conclusões obtidas, assim como o significado de algumas decisões tomadas.

Palavras-chave: Processo de Inspeção, Tecnologias móveis, *Tablets*, Windows 10, UWP, Desmaterialização

Abstract

Currently, the inspection process done by the public organization where this project was performed, encompasses a wide variety of activities. Each one has its own flow of execution. In the same activity, inspections are performed very differently between teams. There are no specific guide lines.

To complicate things further, the teams need a huge amount of documents each time they go out to perform an inspection. All this is very outdated and error prone. That's why a reformulation of the sector is necessary. Here the mobile technology emerges as an opportunity to takle this issue.

This report documents the technological solution developed for Windows 10 tablets using the Universal Windows Platform (UWP) Framework during the 9-month internship at the client of Accenture Consultores de Gestao SA. The mais goals of this project are: the dematerialization of physical documents, the possibility of a new "toolbox" for the inspectors to work and finally to further improvement the inspection activities.We will start by introducing the most important concepts and used technologies. Followed by the research of market applications that inspired the developement of this application.

Keywords: Inspection process, Mobile Technology, Tablets, Windows 10, UWP, Dematerialization

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Acrónimos

ACID Atomicidade, Consistência, Isolamento e Durabilidade. 15

API Application Programming Interface. 11, 13, 19–24, 58

ASCII American Standard Code for Information Interchange. 63

DI Departamento de Informática. 1

DPEI Dissertação/Projeto de Engenharia Informática. 1, 77

FCUL Faculdade de Ciências da Universidade de Lisboa. 1, 3, 77

HTML HyperText Markup Language. 13

HTTP Hypertext Transfer Protocol. 45, 49, 50, 53

HTTPS Hypertext Transfer Protocol Secure. 49, 52

IDE Integrated Development Environment. 11, 12

IoT Internet of Things. 14

ISO International Organization for Standardization. 70

IT Information Technology. 14

JSON JavaScript Object Notation. 24, 51, 63, 80

LINQ Language-Integrated Query. 12

MEI Mestrado em Engenharia Informática. 1

MVC Model View Controller. 7

MVVM Model View ViewModel. 3, 7–9, 46

PM Presentation Model. 7

REST Representational State Transfer. 16, 50, 52

SDK Software Development Kit. 16

SOAP Simple Object Access Protocol. 16, 44, 45, 49–53

SSL Secure Socket Layer. 49

UI User Interface. 7, 8, 11, 13

UWP Universal Windows Platform. 3, 10–13, 16, 51, 58, 60, 63, 64, 66, 75, 78–80

WinRT Windows Runtime. 13

WPF Windows Presentation Foundation. 10

WS-Security Web Services Security. 44, 50, 51, 53

WSDL Web Services Description Language. 50–52

XAML Extensible Application Markup Language. 12, 13, 63, 66, 67

XML Extensible Markup Language. 12, 24, 49–51

Capítulo 1

Introdução

Este relatório documenta o trabalho realizado no 2º ano do Mestrado de Engenharia Informática (MEI), que consiste na realização de um trabalho autónomo original, de natureza profissionalizante, no âmbito da disciplina de Dissertação/Projeto de Engenharia Informática (DPEI) do Departamento de Informática (DI), da Faculdade de Ciências da Universidade de Lisboa (FCUL) no ano letivo 2017/2018. O Projeto decorreu na empresa Accenture Consultores de Gestão S.A. sob a supervisão do Engenheiro Eduardo Silvério, como orientador da empresa, e da Professora Doutora Ana Paula Afonso, como orientadora da faculdade.

Neste capítulo é dado um contexto ao projeto, assim como a motivação que o levou a ser realizado tal como os seus objetivos. É descrito tanto a instituição como a equipa presente nele. Também será apresentada a forma como se encontra estruturado este documento.

1.1 Contexto

Num estudo recente [1], feito pelo jornal do Observador pode ler-se: "Há cada vez mais pessoas a usar o *tablet* como instrumento de trabalho". Depois da conquista do meio profissional por parte dos computadores e *smartphones*, eis que surgem os *tablets* como uma nova ferramenta de trabalho quotidiano.

O universo empresarial tem vindo a transformar-se ao longo das últimas décadas. O aparecimento dos computadores mudou radicalmente a forma como as pessoas trabalhavam. Depois, chegaram os *smartphones*, que na prática se tornaram pequenos computadores de bolso que vieram acrescentar à voz a capacidade de trocar emails e partilhar documentos em tempo real. Sendo que agora o uso dos *tablets* como ferramenta de trabalho também tem ganho força nos últimos anos, prometendo agregar as funcionalidades das duas plataformas, isto é, a capacidade de desempenho e trabalho presente num computador, com a versatilidade móvel e aplicacional presente num *smartphone*.

Este tipo de aparelho tem-se tornado muito popular, ganhando rapidamente a capaci-

dade de substituir os *laptops* nas tarefas menos complexas, como ver o email, consultar as redes sociais ou ficar a par das notícias. Com este aumento de procura surgiu também a necessidade de criação de aplicações móveis que satisfizessem o desejo e necessidade de quem utiliza estes dispositivos, o que levou a um grande aumento de pedidos por parte das empresas em disponibilizar os seus conteúdos/serviços num formato deste tipo.

Num estudo promovido pela Samsung [1], foi possível concluir que mais de 80% dos portugueses usam estes aparelhos para aceder ao *e-mail* e que quase 35% usa o *tablet* para consultar a conta bancária através das soluções de *home banking*. Mais de 29% assume ainda usá-lo para realizar operações financeiras como pagamentos de serviços, bens ou mesmo transferências bancárias. Esta amostra baseou-se num grupo de 304 portugueses que usam estes dispositivos mais de uma hora por dia.

Com resultados tão apelativos é normal que as empresas comecem a pensar nesta alternativa como mais uma ferramenta de apoio aos seus funcionários, assim como a criação de aplicações internas que os ajudem nas suas tarefas. Foi neste contexto que foi realizada a proposta do projeto descrito neste documento, uma aplicação interna usada em *tablet* que permitirá ajudar os trabalhadores do cliente da empresa de acolhimento.

1.2 A Accenture

A empresa acolhedora deste projeto foi a Accenture [2], uma organização internacional com instalações em vários pontos do mundo, líder em serviços profissionais que oferece uma ampla gama de serviços e soluções em estratégia, consultoria, digital, tecnologia e operações. Através da combinação de uma vasta experiência profissional e uma forte especialização em várias indústrias com diferentes tipos de negócio, suportada pela maior rede de *delivery* do mundo, a Accenture trabalha na interseção entre negócio e tecnologia para ajudar os clientes a melhorar o seu desempenho assim como a forma como o mundo trabalha e vive. O projeto sobre o qual este documento incide tem como destinatário um cliente da Accenture.

1.3 O Cliente

Devido a motivos de confidencialidade, não é possível especificar o cliente onde foi realizado o projeto, nem abordar por completo o modo como funciona. No entanto, trata-se de um dos clientes da Accenture, um organismo público que, entre outras atividades, é responsável por ações de inspeção realizadas durante saídas exteriores. Estas ações são praticadas por funcionários na categoria de inspetores que usarão a aplicação desenvolvida neste projeto, como ferramenta principal de trabalho durante as deslocações. Desta forma, são recolhidas informações que mais tarde serão tratadas pelo organismo responsável. A equipa responsável pelo desenvolvimento do projeto, contem 6 elementos com diferentes

níveis de senioridade. Dois programadores seniores, um *manager*, um *project manager* e dois alunos da FCUL.

1.4 Motivação

Dentro das atividades que o cliente fornece existem quatro setores responsáveis pelo processo de inspeção, como foi pedido o máximo de confidencialidade possível na escrita deste documento, estes foram referidos como Tarefa de Recolha de Informação, Tarefa de Assinatura, Tarefa de Inspeção e Tarefa de Controlo explicados no Capítulo 5. Estes setores pretendem que as suas atividades levem uma reformulação na forma como são executadas diariamente, em especial, pretendem o desaparecimento do papel em todas as suas tarefas externas, iniciando assim o seu processo de digitalização.

Face à grande importância que esta atividade tem para o cliente, surge a necessidade de uma modernização e melhoramento dos processos e tecnologias envolvidas. Para tal, é necessário promover uma investigação de quais as melhores opções a tomar de modo a tornar todo este trabalho simples de executar, organizado e eficiente, promovendo uma melhoria da atividade em questão. Assim sendo, a escolha de tecnologias móveis surge como algo natural, desejável e estratégico de forma a acompanhar a evolução atual, aproveitando-as da melhor maneira possível.

1.5 Objetivos

O projeto foca-se no desenvolvimento de uma aplicação móvel com destino a ser executada em *tablets* Windows 10, pertencentes aos funcionários do cliente responsáveis pelas ações de inspeção. Para tal será produzida uma solução composta no *Universal Windows Platform* (UWP), usando *Model View ViewModel* (MVVM). A elaboração deste projeto tem como principal objetivo acabar com o uso de papel nas atividades de inspeção, bem como a modernização da forma como os inspetores executam as suas tarefas diárias levando aos poucos à sua uniformização. Deste modo, pretende-se obter um aumento considerativo na organização e planeamento das atividades realizadas, assim como uma diminuição na quantidade de material físico que transportam diariamente, que passa a ser transformado numa "caixa de ferramentas" digitais ao seu dispor.

Para além da participação no desenvolvimento da aplicação móvel, este projeto tem como objetivo particular estudar a conciliação de documentos de transporte e a melhor forma de o fazer, assim como tratar os diferentes tipos de documentos que o sujeito passivo se faça acompanhar, seja uma fatura do material ou uma guia de transporte.

1.6 Estrutura do Documento

O relatório está dividido em 6 capítulos:

- O Capítulo 1 introduz o projeto, dando um contexto geral do que será tratado, falando sobre a empresa de acolhimento, a motivação para a sua realização, os objetivos e como este documento se encontra estruturado.

- O Capítulo 2 descreve alguns dos conceitos importantes aplicados ao desenvolvimento do projeto, que permitiram desenhar e implementar a solução através dos padrões de qualidade existentes no mercado. São enunciadas as tecnologias que foram escolhidas tanto pela equipa, como pelo cliente, para serem aplicadas no desenvolvimento da aplicação. É realizado um enquadramento das tecnologias utilizadas hoje em dia na elaboração de aplicações móveis, sendo feita uma comparação entre os seus pontos positivos e negativos. Também são investigadas aplicações com funcionalidades semelhantes, que possam contribuir com ideias para o desenvolvimento deste projeto.

- No Capítulo 3 é descrita a metodologia aplicada durante o trabalho efetuado, assim como o planeamento do trabalho a realizar durante os 9 meses do projeto. É feita também uma análise aos potenciais riscos que possam acontecer e como devem ser encarados.

- No Capítulo 4 é abordada a forma como foi planeada a construção da aplicação, descrevendo o modelo de dados criado, a forma de comunicação com os serviços e os mecanismos de segurança aplicados. São apresentadas várias vistas arquiteturais tanto da aplicação móvel como do sistema que a engloba.

- No Capítulo 5 é descrito o trabalho realizado durante o projeto, como funciona a aplicação, quais as suas funcionalidades e o que elas representam, sendo dado destaque à atividade que o aluno ficou encarregue por estudar e realizar. No final são apresentados os testes efetuados à aplicação durante e após o término do projeto.

- Por fim, no Capítulo 6 são feitas as considerações finais ao projeto realizado, tirando conclusões do que foi feito, das dificuldades encontradas, das alternativas que poderiam ter sido efetuadas e do trabalho futuro possível.

Capítulo 2

Enquadramento: Conceitos e Tecnologias Utilizadas

Neste capítulo serão descritos os principais conceitos utilizados neste projeto que permitiram desenhar e implementar a solução através dos padrões de qualidade existentes no mercado, de forma a garantir qualidade e eficiência do produto final. Serão também apresentadas as tecnologias e ferramentas utilizadas no decorrer do projeto, que foram escolhidas após um estudo de várias alternativas possíveis. Os requisitos pretendidos pelo cliente também tiverem impacto nestas escolhas.

2.1 Metodologia *Agile*

Segundo Larman e Basili [3], a origem das metodologias ágeis está ligada às metodologias iterativas e incrementais de desenvolvimento de *software*. Estes acreditam que a Metodologia em Cascata, associada a longas etapas de desenvolvimento de projetos será substituída por ciclos iterativos de menor dimensão, criando uma versão incremental do produto no fim de cada ciclo.

Em 2001 é criado um pilar na evolução da Metodologia Ágil, quando dezassete pioneiros desta metodologia se reuniram em Utah, Estados Unidos da América, para discutir os valores e princípios fundamentais que deveriam aplicar, criando assim o Manifesto Ágil [4], que estabelece quatro valores orientadores:

- São encorajados todos os elementos da equipa a ter um papel ativo no desenvolvimento do plano de atividade. Desta forma são partilhadas ideias que formam um produto suportado por uma visão comum a cada membro do processo;
- Entrega do produto em ciclos incrementais, como o propósito de produzir um produto livre de *bugs*, continuo, até alcançar o estado final;
- Como os requisitos do produto podem não estar disponíveis no início do projeto devido a inúmeros fatores, este deve começar a ser desenvolvido quase de imediato,

sendo apresentado várias vezes ao cliente para questões de validação;

- A capacidade de responder a alterações é mais importante que seguir um plano.

Estes valores não pretendem negar o valor dos procedimentos, ferramentas, documentação, contratos e planos. Em vez disso, pretendem estabelecer um escala de prioridades que introduza flexibilidade nos projetos.

O ciclo de vida de desenvolvimento de software Ágil [5], baseia-se nos modelos de processos iterativos e incrementais. Foca-se na adaptabilidade à mudança de requisitos de produtos, no aprimoramento da satisfação do cliente, através da entrega rápida de produtos funcionais e na participação do cliente durante o desenvolvimento. Consoante os requisitos, estes serão "partidos" em requisitos de menores dimensões, facilmente desenvolvidos através de ciclos incrementais conhecidos como "*Sprints*".

No início de cada *sprint* são realizadas reuniões para definir quais os requisitos a serem feitos, sendo que no início de cada dia é realizada uma reunião, apenas de 15 minutos, para ter uma visão dos requisitos já estabelecidos e possíveis atrasos. O objetivo é no fim de cada iteração ser possível acabar com uma versão incremental do produto, com os requisitos programados para esse *sprint* a serem satisfeitos. Este processo encontra-se representado na Figura 2.1.

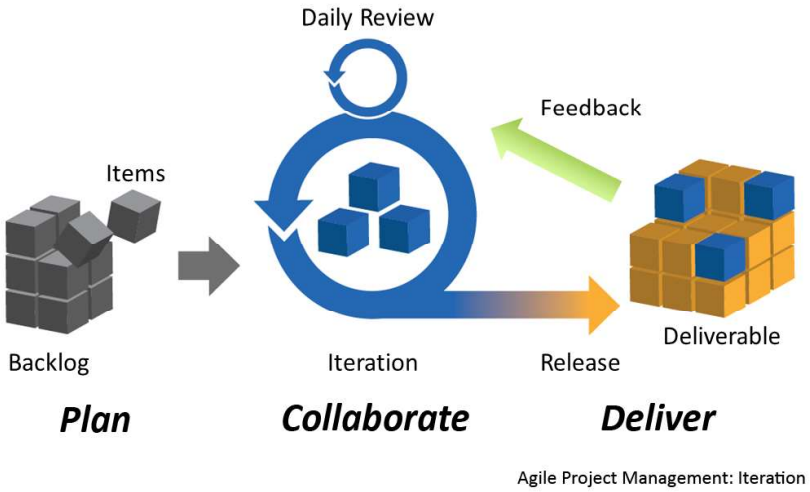

Figura 2.1: Metodologia Ágil [6]

O desenvolvimento do projeto seguiu a metodologia ágil, que costuma ser a metodologia adotada nos projetos realizados pelas equipas da Accenture neste cliente, dada a constante alteração dos requisitos durante as várias iterações. Os detalhes serão abordados no Capítulo 3.

2.2 Padrões de Desenho

Na engenharia do *software*, um padrão de desenho é uma solução geral para um problema que ocorre com frequência dentro de um determinado contexto num projeto de *software*. Apesar de não ser um projeto finalizado que pode ser traduzido diretamente para código fonte, representa um *template* de como resolver um problema recorrente em diferentes situações [7].

2.2.1 *Model View ViewModel*

O padrão *Model View ViewModel* (MVVM)[8] foi introduzido por John Gosmann em 2005 e é uma especialização do padrão *Presentation Model* (PM) introduzido em 2004 por Martin Fowler. Um dos objetivos principais do padrão *Presentation Model* é a separação e abstração da *View* da camada de desenvolvimento, deste modo é possível tornar a interface do utilizador testável e reduzir o acoplamento com o resto do código.

Estruturalmente uma aplicação que usa o padrão MVVM [9] consiste em três componentes principais: o *Model*, a *View* e o *ViewModel*. O *Model* é a implementação do modelo de domínio da aplicação, que inclui os dados utilizados pela camada de negócio, por exemplo a informação relativa a um cliente. A *View* é o controlador da *interface* gráfica responsável pela apresentação do modelo de dados ao utilizador. Define a estrutura, o *layout* e a aparência do que o utilizador vê no ecrã. O *ViewModel* contém a lógica da *View*, os comandos, os eventos, e uma referência para o modelo, sendo responsável pelo tratamento dos seus dados. No MVVM, o *ViewModel* não é responsável por atualizar os dados exibidos na *User Interface* UI, ao contrário do padrão *Model View Controller* (MVC), em que o controlador passa informação às *Views*. Por sua vez, o MVVM, usa *binders* para expor propriedades e comandos utilizados. O *ViewModel* funciona como mediador da comunicação entre as *Views* e os *binders*, podendo converter cada valor em algo apresentável para o utilizador. Na Figura 2.2 é possível observar as relações explicadas entre os três componentes.

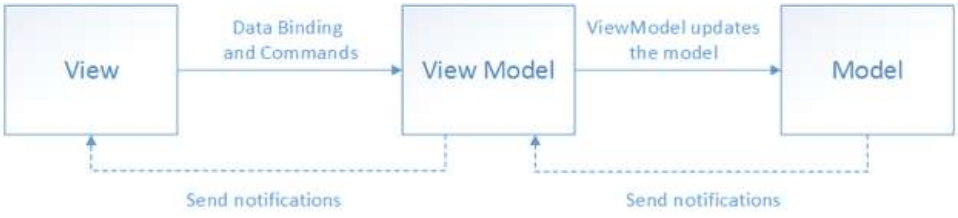

Figura 2.2: Padrão *Model View ViewModel* [10]

Após uma análise aos padrões atuais que são aplicados a tecnologias móveis, os programadores concluíram que este padrão é o mais indicado. O principal motivo desta escolha deveu-se ao objetivo proposto pela equipa de desenvolvimento, em tornar o código

o mais simples, *decoupled* e fácil de alterar possível. Desta forma, as alterações futuras ou testes serão mais fáceis de realizar em detrimento de padrões onde a lógica do negócio e a *interface* gráfica não se encontram tão separadas.

Binders

Os *Binders* são o core do MVVM [11]. São os responsáveis pela separação da camada de desenvolvimento, do resto do padrão. Ao criar esta separação as *Views* nunca estão dependentes dos objetos que lhe são passados, sendo que caso existam alterações frequentes a nível da interface numa fase mais avançada do projeto, não será necessário alterar a lógica de negócio envolvida. Os *Binders* dividem-se em dois tipos, *OneWay* e *TwoWay*. No primeiro caso uma alteração do valor do recurso na parte do código, faz alterar o valor apresentado pela *View*. No segundo caso, verifica-se o acontecimento anterior com a adição de, se o valor for alterado através do *input* do utilizador na sua interação com o UI, essa alteração será efetuada também no valor da variável do lado do código. Estas alterações só são visíveis dinamicamente graças à interface *INotifyPropertyChanged*.

Property Change Notification

De forma a dar suporte aos *Binders*, a *INotifyPropertyChanged* [12], permite atualizar os valores dos elementos presentes na *View* que implementam a *interface* em questão. De forma a obedecer ao padrão MVVM, estes elementos podem ser definidos tanto no *View Model* da *View* correspondente, ou no modelo de dados associado. Assim, quando ocorrer alguma alteração aos seus valores, estes serão modificados dinamicamente, sem ser necessário recarregar a página.

DataContext

DataContext é um conceito onde os objetos herdam informação dos seus objetos pais numa relação hierárquica [13]. Costuma ser usado tipicamente em coleções de elementos. A informação contida nos objetos presente na coleção é traduzida conforme o tipo de objeto que é dado no seu contexto. Por exemplo, uma lista apresentada pelo UI se tiver como *source* uma coleção de objetos do tipo Pessoa, irá necessitar de um contexto do mesmo tipo, de modo a apresentar os atributos pretendidos.

2.2.2 *Dependency Injection*

A ideia básica da *Dependency Injection* consiste em ter um objeto separado, um *assembler*, que injeta instâncias de variáveis aquando da criação de um novo objeto [14]. Imaginando, um objeto X que aquando da sua criação necessita de um objeto Y, em vez de ser feita a criação do objeto Y, este é passado quando é feita a criação do objeto X. Este

tipo de padrão é bastante usado em MVVM de forma a tornar o código cada vez mais *decoupled* e testável de forma independente [15, 16].

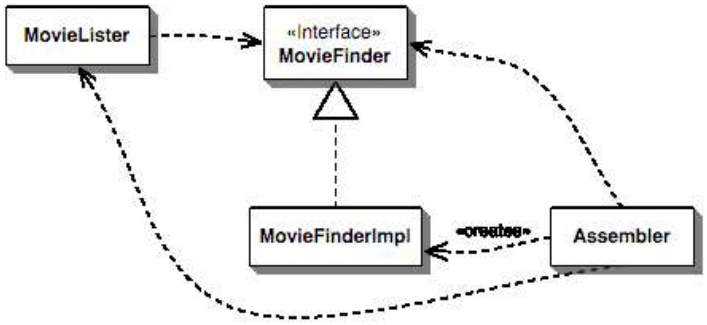

Figura 2.3: As dependências usando *Dependency Injection* [14]

Na Figura 2.3 de Martin Fowler é possível observar a associação do *MovieFinderImpl* à sua respetiva *interface* através do uso do *Assembler* [14] . Após esta atribuição, o *MovieLister*, quando for criado vai receber a injeção da *interface* do *MovieFinder*.

2.2.3 *Service Locator*

A ideia básica do *Service Locator* é ter um objeto, um *locator*, que sabe como obter todos os serviços que uma aplicação necessita [14, 17]. Desta forma, as interfaces dos serviços podem ser encapsuladas no *locator*, de modo a que todas as chamadas que sejam efetuadas, sejam redirecionadas para a respetiva implementação.

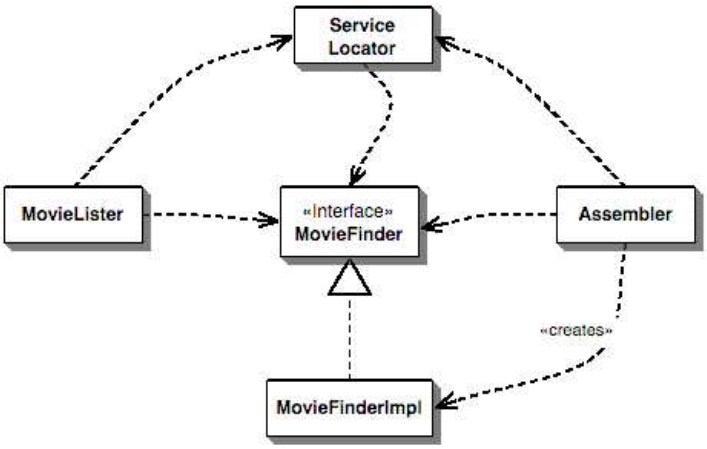

Figura 2.4: As dependências usando *Service Locator* [14]

Na Figura 2.4 de Martin Fowler é possível observar a criação do *Service Locator* e a forma como o *MovieLister* comunica com ele quando necessita de usar a *interface*

MovieFinder [14]. Estão presentes tanto os conceitos de *Dependency Injection* como de *Service Locator*, apesar de ser possível usar ambos os padrões em separado.

2.2.4 *Inversion of Control Container*

Como observado em 2.2.2 e 2.2.3, ambos os padrões usam um *Assembler* responsável pela gestão dos componentes da aplicação, no caso de serviços, é o responsável por associar as *interfaces* aos respetivos serviços. O *Inversion of Control Container* é uma *framework* que simplifica a criação do *Assembler*, fazendo uso de código específico [18]. Na Figura 2.5 está representado um exemplo de como as *interfaces* são registas no projeto.

Novamente a escolha destes padrões teve em conta o objetivo principal da equipa de desenvolvimento em facilitar alterações futuras ao código do projeto. Desta forma se algum serviço necessitar de ser alterado, desde que o contrato da *interface* seja cumprido é possível substituir o módulo de forma direta.

```
Container.RegisterType<IDatabase, Database>();
Container.RegisterType<IWSPerfilUtilizador, WSObterDadosInspetor>();
```

Figura 2.5: Registo das interfaces usando um *container*

2.2.5 *Singleton Pattern*

Singleton Pattern [19], é um padrão de desenho bastante usado quando pretendido criar uma única instância de um determinado objeto. O padrão garante que todas as chamadas a este objeto referem-se à mesma instância. O seu uso é bastante útil quando é necessário um ponto de acesso global a um determinado recurso, sendo mais apropriado que criar uma variável global. Esta pode ser copiada, levando a múltiplos pontos de acesso, o que tornaria os duplicados diferentes do recurso original [20].

Como será explicado com mais detalhe na secção referente aos testes de memória realizados, foi necessário recorrer a este padrão de desenho para criar uma única instância de um determinado objeto. Esta solução veio resolver o problema das múltiplas instâncias criadas no ciclo de vida da aplicação, cada vez que o objeto era necessário.

2.2.6 *Desktop Bridge*

Desktop Bridge [21], é uma infraestrutura que tal como o nome indica permite converter aplicações clássicas como Win32, *Windows Forms* e WPF [1] em aplicações *Universal Windows Platform*. Após convertidas, estas aplicações são empacotadas, servidas e feitas *deployed* no formato UWP, visando Windows 10. Com isto, não só é possível permitir que as aplicações clássicas façam uso das novas *Application Programming Interfaces*

[1] Windows Presentation Foundation [22], *framework* responsável pelo desenvolvimento de aplicações Windows para *desktop*. Utiliza o .NET Framework como fonte de bibliotecas.

(APIs) do UWP, mas também permite que este possa recorrer a bibliotecas que apenas se encontram disponíveis nas plataformas clássicas.

Como foi necessário recorrer ao uso de uma biblioteca que não se encontra disponível para UWP, a equipa de desenvolvimento teve necessidade de recorrer ao uso desta infraestrutura.

2.2.7 *Asynchronous Programming*

Usar programação assíncrona ajuda as aplicações a manterem-se *responsive* quando determinada atividade demora mais tempo do que o esperado [23]. Por exemplo, quando é necessário consultar um serviço de modo a obter determinado conteúdo, caso fosse usada programação síncrona, a aplicação iria bloquear até obter a resposta. Graças à assincronia enquanto são esperados os novos recursos, é possível continuar a fazer uso da aplicação, sem que a *thread*[2] principal do UI bloqueie. Desta forma é possível à aplicação desenvolvida neste projeto fazer uso dos serviços do cliente, obtendo informação que será apresentada na *interface* sem que esta bloqueie durante a utilização do *tablet* pelo inspetor.

2.3 Tecnologias e Ferramentas

Após o esclarecimento efetuado pelo cliente de quais as suas ideias para o projeto e que tipo de aplicação móvel desejava, coube à equipa de desenvolvimento encontrar as tecnologias que melhor se adequavam ao problema. As seguintes tecnologias utilizadas constituem o "*core*" do projeto sendo através delas que foi possível obter um produto final de acordo com os desejos iniciais.

2.3.1 Microsoft Visual Studio

O *Integrated Development Environment* (IDE) utilizado neste projeto foi o Microsoft Visual Studio 2017 [24]. Este IDE suporta diversas linguagens de programação (C, C#, C++ e Visual Basic), assim como diversas extensões de recursos (SQLite por exemplo). Inclui um editor de código, que além de suportar *IntelliSense*, suporta também *code refactoring*, ou seja, é possível fazer refatoração ao código. O *debugger* incluído, suporta tanto código fonte das aplicações como código máquina. É possível elaborar o desenvolvimento gráfico das aplicações através de ferramentas de desenho, que incluem elementos visuais nativos comuns às aplicações atuais. Atualmente está disponível para Windows e Mac e permite desenvolver aplicações para Android, iOS, Mac, Windows, Web e Cloud.

[2]Thread, encadeamento de um conjunto de tarefas que formam um processo. Várias *threads* podem ser formadas para um determinado processo, desta forma as tarefas são divididas e podem ser executadas em paralelo.

2.3.2 C#

C# é uma linguagem de programação orientada a objetos, baseada em C, C++ e Java que permite a construção de uma variedade de aplicações que usam o .NET. [25]. A sua sintaxe simplifica a complexidade existente no C++ contribuindo com um conjunto de ferramentas poderosas, como por exemplo, tipos de valores nulos, expressões *lambda*, enumerados e acessos diretos à memória que não é possível em Java. Faz uso das expressões LINQ (*Language-Integrated Query*) como principal construtor de *queries*.

O LINQ veio simplificar a necessidade dos programadores aprenderem todas os tipos de linguagens orientadas a dados, oferecendo um modelo consistente através dos mais diversos tipos de fontes e formatos. Numa *query* deste tipo trabalha-se sempre com objetos, usando os mesmos tipos de padrões quer sejam usadas base de dados SQL, documentos escritos em *Extensible Markup Language* (XML) ou coleções .NET.

É nesta linguagem de programação que a aplicação é desenvolvida. Esta escolha baseia-se no fato de conseguir executar tudo o que a linguagem nativa do Windows consegue. C# permite também programação assíncrona, explicada na Secção 2.2.7, conceito importante em *mobile*, pois desta forma a aplicação não fica "bloqueada" à espera de resposta, continuando a funcionar enquanto outra tarefa executa em paralelo.

2.3.3 XAML

Extensible Application Markup Language vulgo XAML [26], é uma linguagem declarativa desenvolvida pela Microsoft e baseada em XML. É responsável pela representação visual da *interface* gráfica das aplicações desenvolvidas para Windows, através da criação de estruturas e objetos. É possível graças ao uso de IDEs como o Visual Studio que possuem ferramentas de desenho, gerar XAML através do simples *drag and drop* de elementos visuais, desta forma não é necessário os programadores saberem como funciona a linguagem. As regras de sintaxe são quase as mesmas que para o XML, sendo que um documento deste tipo é sempre um documento XAML, apesar de o inverso não ser necessariamente verdadeiro.

2.3.4 *Universal Windows Platform*

Universal Windows Platform (UWP) é uma plataforma comum a cada dispositivo que executa o Windows 10 [27]. O conjunto dos diferentes tipos de equipamentos tais como *xbox*, computadores ou *tablets* forma uma família [28] que partilha a mesma API. Desta forma é possível a criação de um único pacote aplicacional, que pode ser distribuido a qualquer membro do grupo. Ainda assim, é possível especificar diferentes capacidades de cada dispositivo usando uma extensão de um *SDK* [3] próprio. As aplicações criadas

[3]SDK, também conhecido como Software Development Kit [29], é um conjunto de ferramentas de desenvolvimento de software que permite a criação de aplicações para um certo pacote de software.

em UWP podem chamar não só as APIs do WinRT (*Windows Runtime*) que são comuns a todos os dispositivos, mas também as APIs (incluindo APIs do Win32 e .NET) que são específicas da classe de dispositivos na qual a aplicação está sendo executada.

As aplicações UWP são distribuídas a partir da mesma *store* para todos os tipos de dispositivos disponíveis a recebê-las. O pacote aplicacional usa o formato .AppX, reconhecido como uma instalação de confiança pelo mecanismo responsável.

A nível visual, uma aplicação UWP pode correr numa ampla gama de equipamentos com diferentes formas de *input*, resoluções e outras características especificas de *hardware*. Graças ao Windows 10 existem vários controlos, ferramentas e *layouts* que ajudam a adaptar o UI ao aparelho do utilizador. Além destes, existem ainda outros elementos que se adaptam automaticamente sem exigir intervenção do programador, como é o caso dos botões ou dos *sliders*. É possível usar várias linguagens na elaboração da *interface* gráfica, sendo que, estas são escolhidas consoante a linguagem de *back-end* preferida. Em caso de uso de C++, C# ou *Visual Basic* é utilizado XAML de modo a garantir uma boa *user experience* nativa, no caso do C++ também é possível escolher *DirectX* de forma exclusiva ou misturado com XAML. Para *JavaScript* a camada de apresentação está associada ao *HyperText Markup Language* (HTML).

Como o UWP é executado numa *sandbox*, tem acesso limitado ao *system* assim como aos dados do utilizador. Ao contrário das aplicações em Win32, que correm muitas vezes em modo "Administrador", o UWP tem menos acessos que um utilizador "normal", sendo que nunca consegue correr em modos privilegiados. Por exemplo, não consegue aceder à maioria dos ficheiros do sistema, mas consegue ler da pasta onde foi instalado ou da pasta de ficheiros temporários [30].

Em suma, esta plataforma do Windows vem permitir desenvolver uma aplicação que pode ser executada em múltiplos dispositivos da mesma família como pode ser observado na Figura 2.6, com as mesmas APIs partilhadas, através do uso de um conjunto de linguagens com que o programador se sinta familiarizado a desenvolver.

Figura 2.6: Universo Windows [27]

2.3.5 .NET

.NET é uma plataforma de desenvolvimento criada pela Microsoft [31]. Em 2000 foi disponibilizada a primeira versão beta, o .NET 1.0. Atualmente, forma um ecossistema com várias *frameworks* que sofreram *fork* do .NET original como pode ser observado na Figura 2.7. Linguagens como C#, F# ou Visual Basic tal como bibliotecas para construção em web, *mobile*, *desktop* e *Internet of Things* (IoT), são usadas para desenvolvimento aplicacional. Fazendo uso destas bibliotecas/linguagens o programador deixa de escrever código para um sistema ou dispositivo específico, passando assim a desenvolver para a plataforma .NET. O código gerado será executado de forma nativa seja qual for o sistema operativo (Windows, Linux, MacOs, iOS, Android e Windows 10 UWP).

.NET Framework

.NET Framework foi criado em 2002 e é a evolução do .NET original [31]. Consiste na plataforma principal de desenvolvimento de aplicações cliente/servidor para Windows. É uma *framework* bastante evoluída e madura, com uma grande quantidade de bibliotecas que suportam uma grande variedade de aplicações e soluções.

O .NET Framework é servido como parte do sistema operativo Windows sendo por isso uma *framework* monolítica, ou seja, tem de ser instalado por completo na máquina de destino. Os *updates* são recebidos através das atualizações automáticas do Windows ou então podem ser instaladas diretamente a partir de um executável.

.NET Core

.NET Core é a nova versão do .NET, *open source*, modular e *cross-platform* [31]. Com o crescimento e evolução do mercado de *Information Technology* (IT) a Microsoft sentiu necessidade de atualizar o seu produto, de modo a não ficar presa apenas ao seu sistema operativo. Ao contrário da tradicional *framework*, o .NET Framework, que como dito em 2.3.4 é instalado num único pacote e específico do Windows, o .NET Core é distribuído num formato *decoupling*.

Em vez de ser necessário instalar um pacote gigante de bibliotecas (400mb), que se encontra dependente do sistema operativo, o .NET Core usa o sistema de pacotes *Nuget* [4] que são instalados de forma independente do sistema e diretamente no pacote da aplicação. Desta forma, ao contrário do .NET Framework, só instalamos o que é necessário de modo a não termos bibliotecas que não vão ser usadas.

[4] Arquivo ZIP com a extensão .nupkg, usado pelos programadores no qual podem criar, partilhar e consumir código. Este código é agrupado em *packages* que contêm código compilado (DLLs), juntamente com outro conteúdo necessário aos projetos [32].

.NET Standard Library

.NET Standard foi criado de forma a ser possível partilhar e reutilizar código entre múltiplas plataformas .NET [31]. No caso do desenvolvimento aplicacional não é dado *target* diretamente à biblioteca, no entanto todas as aplicações usufruem destes benefícios.

Desta forma, se for criada uma aplicação em WPF que mais tarde possa ser migrada para UWP, não será necessário grandes alterações ao projeto. Isto, caso as bibliotecas .NET Framework usadas, estejam também presentes no .NET Standard.

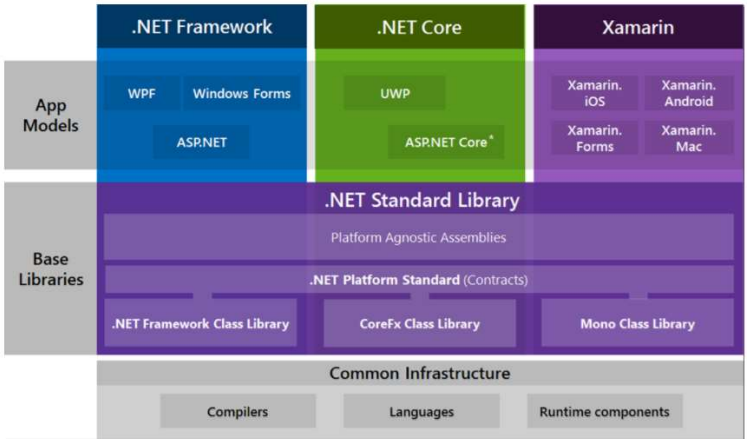

Figura 2.7: Ecossistema .NET [31]

2.3.6 SQLite

SQLite é uma biblioteca escrita em C que implementa uma base de dados relacional [33]. O seu código está no domínio publico, livre para qualquer uso seja comercial ou não. Atualmente é encontrado num número variado de dispositivos. Em contraste com outros sistemas de gestão de base de dados, o SQLite não é uma base de dados cliente-servidor, escreve e lê dados diretamente de um disco. A sua popularidade deve-se à sua simplicidade de administração, implementação e manutenção, o que a torna uma mais valia em aplicações que necessitam de armazenar dados para mais tarde sincronizar, como é o caso de aplicações móveis que se se deslocam constantemente para zonas sem acesso à *internet*.

O SQLite suporta transações de informação com as propriedades de Atomicidade, Consistência, Isolamento e Durabilidade (**ACID**) [34], mesmo que sejam interrompidas por *crashes* ou falhas de energia do sistema. **Atomicidade** significa que "é tudo ou nada", se parte de uma transação falha, toda a transação falha e o estado da base de dados mantém-se inalterado. **Consistência** garante que qualquer transação muda o estado válido de uma base de dados para outro igualmente válido, obedecendo a regras previamente definidas. **Isolamento** a propriedade que garante que várias operações feitas em simultâneo, provocam um estado à base de dados que seria obtido caso fossem feitas de

forma sequencial, ou seja, uma após uma. **Durabilidade** assegura que após uma transação tenha sido efetuada com sucesso, esta fica guardada permanentemente.

SQLite é uma biblioteca compacta, com todos os seus recursos ativados o seu tamanho pode ser inferior a 500KiB, dependendo da plataforma de destino e das configurações de otimização do compilador, o que a torna ideal para dispositivos com memória limitada. O ficheiro criado pelo SQLite é uma compilação de dados estruturados, organizados e armazenados de forma constante a partir de múltiplas tabelas. Foi usado o SQLiteStudio como ferramenta gráfica para visualização/edição do conteúdo da base de dados.

2.3.7 Sketch

Sketch foi a aplicação sugerida para efetuar a prototipagem da *interface* gráfica da aplicação. Encontra-se disponível apenas para Mac. Desta forma aproxima o cliente do desenvolvimento do projeto e torna-se mais fácil chegar a um consenso do que é pretendido. Esta aplicação costuma ser utilizada pelo gestor do projeto quando necessário preparar demonstrações de protótipos ao cliente.

2.3.8 SoapUI

SoapUI é uma ferramenta *open-source* que serve para criação, simulação e testes de *mocks* de serviços na web. Funciona para protocolos SOAP (*Simple Object Access Protocol*) e para o estilo arquitetural REST (*Representational State Transfer*). Foi utilizada de modo a simular um ambiente de execução na qual a aplicação obtinha dados de vários serviços.

2.4 *Frameworks*

O principal objetivo deste projeto é desenvolver uma aplicação móvel capaz de desmaterializar o processo de inspeção utilizado atualmente . Por este motivo, é preciso investigar as tecnologias que foram surgindo com o crescimento do mercado, escolhendo aquelas que se adaptam melhor para o problema deste projeto. Esta pesquisa consistiu sobretudo em *frameworks* alternativas, bases de dados e aplicações existentes atualmente.

Partindo do requisito que a aplicação desenvolvida seria para funcionar em *tablets*, foi feita uma investigação às possíveis *frameworks* existentes no mercado que podiam satisfazer este requisito. De notar que a decisão final coube ao cliente, que optou que o projeto seria desenvolvido para dispositivos Windows, através do uso do UWP.

2.4.1 *Native Apps*

Uma aplicação nativa é uma aplicação que vai de encontro os requisitos de um sistema operativo particular, através do uso de um SDK específico. Como foi escrita para uma plataforma específica, não pode ser executada noutro sistema, sem ser aquele para o qual

foi concebida. Tendo isto em consideração, este tipo de aplicação tem a habilidade de usar o *hardware* e *software* do dispositivo no qual opera, tirando assim vantagem da última tecnologia disponível no aparelho como é o caso do GPS ou da câmara (algo que aplicações web não conseguem).

iOS

Criado em 2007, iOS é o sistema operativo para dispositivos Apple, tais como iPhone, iPad, iPod ou Apple TV. Apesar não ser o *software* móvel dominante no mundo, tem uma cota de mercado de 60% na América do Norte, segundo estudos de mercado de 2010. [35]. O iOS deriva do Mac OSX, contendo quatro camadas cada uma com o seu conjunto de *frameworks*, onde as que ocupam o nível superior são as mais abstratas e as de nível inferior as mais próximas do *hardware* [36]:

- *Core OS Layer:* Responsável por funcionalidades de baixo nível assim como *frameworks* de segurança e interação com *hardware* externo.
- *Core Services Layer:* Responsável por fornecer serviços às camadas de cima.
- *Media Layer:* Responsável pelas tecnologias necessárias para os gráficos, vídeo e áudio.
- *Coca Touch Layer:* Responsável pela maioria das *frameworks* utilizadas pelos programadores. Situa-se no topo da *stack* do iOS.

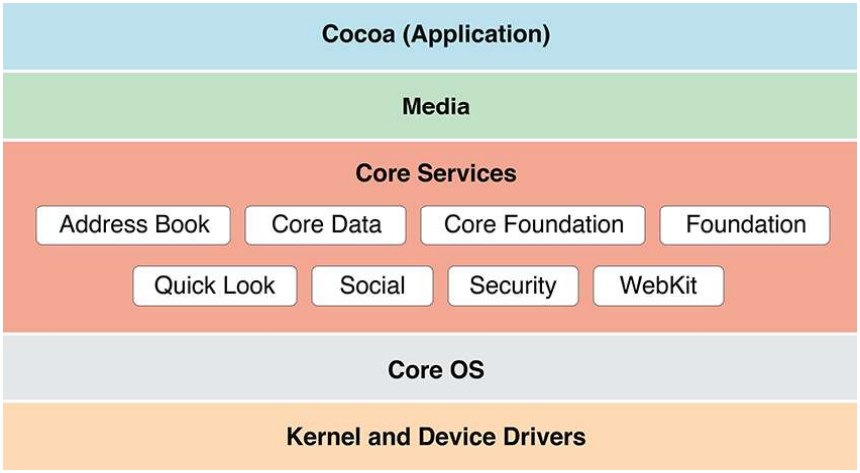

Figura 2.8: Stack do iOS [36]

Como é possível observar pela Figura 2.8, existe ainda outra secção, *Kernel e Drivers*, responsável pelo nível de abstração do *hardware*. Consiste nas *drivers* essenciais ao *hardware* e o núcleo de funcionamento do sistema.

Tanto para Mac como para aplicações iOS, é possível fazer uso do Xcode para desenvolvimento. Um IDE gráfico, que permite escrever código para as novas versões do sistema, incorporando a nova linguagem de programação da Apple, o Swift. Este veio substituir a linguagem usada até à data, Objective-C, mas continua a ser possível desenvolver aplicações usando uma mistura das duas, ou apenas uma, sendo uma escolha do programador [37].

Android

Inicialmente desenvolvido pela Android Inc. e comprado pela Google em 2005 [38]. Android é um sistema operativo móvel, baseado numa versão modificada do *kernel* do Linux e desenhado para dispositivos como telemóveis, *tablets*, relógios ou televisões Android. Domina atualmente o mercado com 85% da cota contra 14,7% do iOS, segundo estudos feitos em 2017 [39]. O sistema operativo Android é composto tal como iOS por 4 camadas de abstração [40], tendo ambos em comum a quinta secção correspondente ao *kernel* e *drivers* como é possível observar na Figura 2.9:

- *Libraries:* Conjunto de bibliotecas responsáveis pelo desenvolvimento específico de Android, persistência de dados (SQLite) e segurança.
- *Runtime:* Responsável por um elemento chamado Dalvik Virtual Machine que é um género de Java Virtual Machine[5], especialmente desenhada e otimizada para Android. Faz uso das principais funcionalidades do Linux como gestão de memória e *multi-threading*. Também contem um conjunto de bibliotecas que permite aos programadores desenvolverem as suas aplicações usando Java.
- *Application Framework:* Responsável por fornecer serviços de alto-nível às aplicações em formato de classes Java. Serviços tais como sistema de notificações, sistema de recursos(cores, *layouts*) e serviços de partilha de dados.
- *Applications:* Camada onde é instalada a aplicação. Situa-se no topo da *stack* do Android.

[5]Aplicação virtual que executa programas Java, convertendo os *bytecodes* em código máquina executável

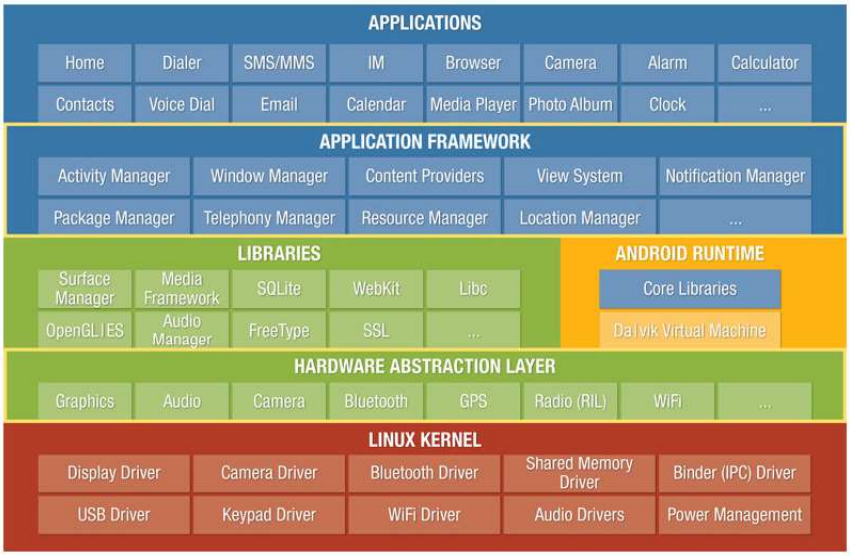

Figura 2.9: Stack do Android [40]

É usado o Android Studio como IDE gráfico oficial [41], que vem suportando Java como principal linguagem de desenvolvimento ao longo dos anos. Em 2017 durante uma conferência, a Google oficializou o Kotlin como alternativa, passando a ser a segunda linguagem oficial.

2.4.2 *Cross-Platform Apps*

Uma aplicação cross-platform é uma aplicação compatível com múltiplos sistemas operativos, que usa uma linguagem não nativa do sistema [42]. Isto significa que é possível fazer partilha de código através de múltiplas plataformas, como por exemplo Android e iOS.

React Native

React Native é uma *framework open-source* mantida pelo Facebook e baseada na biblioteca React que faz uso de JavaScript para a criação de aplicações para Android e iOS [43]. Permite a integração de componentes nativos quando necessário e a visualização de mudanças sem ser necessário recompilar. Desta forma é possível o desenvolvimento rápido de aplicações, partindo como ideia base "*learn once, write anywhere*".

Diferente das outras alternativas que utilizam *webviews* para renderização, o React Native invoca APIs nativas em Objective-C para iOS, ou Java para Android [44]. Desta forma a *interface* gráfica será gerada com componentes nativos da plataforma de destino, isto acontece devido à existência de uma "*bridge*" que converte os elementos React em elementos nativos, como por exemplo uma <View> é transformada numa UiView no iOS

e numa View no Android. Este comportamento pode ser visível na Figura 2.10. É possível fazer desenvolvimento para outras plataformas desde que alguém escreva esta "*bridge*".

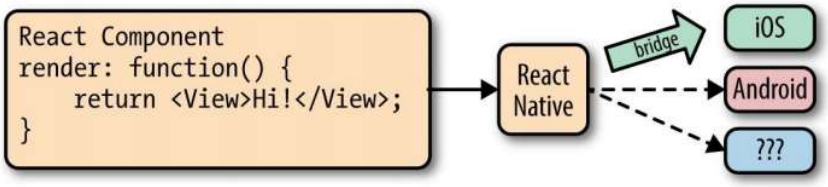

Figura 2.10: Renderização do React [43]

Caso seja necessário usar alguma API que não esteja contemplada pela *framework*, é possível instalar *plugins* feitos por programadores externos que permitem aceder aos recursos pretendidos. Em última instância, caso seja pretendido algum componente que não esteja presente na *framework* nem sob a forma de *pulgin*, é possível implementar módulos nativos em Java ou Objective-C.

Xamarin

Comprada pela Microsoft, Xamarin é uma plataforma de desenvolvimento que serve para distribuir aplicações nativas em Android, iOs, MacOS e Windows usando as propriedades e características do .NET, assim como C# como linguagem de desenvolvimento [31]. É baseado no Mono, a plataforma *open source* e *cross-platform* original implementada pela Microsoft. Segue o progresso do .NET Framework e não do .NET Core.

As aplicações criadas em Xamarin fornecem desenvolvimento semelhante às criadas de forma nativa com Objective-C/Swift para iOs ou Java para Android [45] . É possível criar aplicações para os vários dispositivos partilhando o mesmo código, não só a nível lógico de negócio, como a nível da *interface* do utilizador. No entanto, o Xamarin permite a utilização de mecanismos de desenvolvimento que levam à criação de *interfaces* gráficas nativas para cada plataforma, o que apesar de ir contra o objetivo principal da *framework*, permite aos programadores um maior controlo sobe a personalização individual que é pretendida em cada sistema operativo. Caso o programador não se importe com as personalizações da aplicação e queira o aspeto nativo gerado por um só código, pode recorrer ao Xamarin.Forms, que encapsula as APIs nativas de cada sistema, gerando assim uma *interface* gráfica comum. Na Figura 2.11 é visível os dois tipos de mecanismos de desenvolvimento.

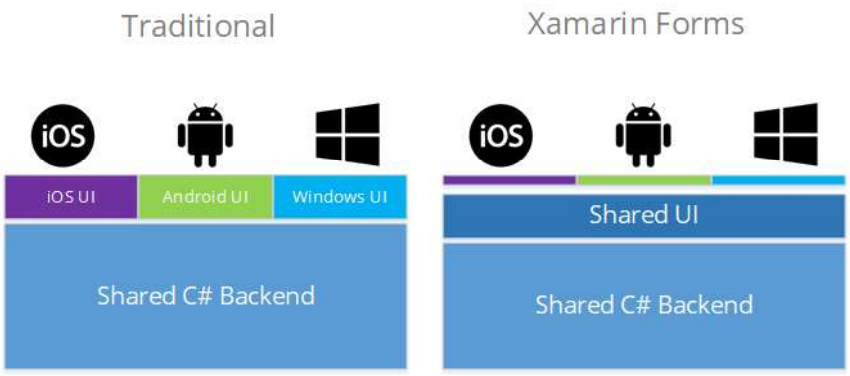

Figura 2.11: Xamarin Tradicional vs Xamarin Forms [46]

O Xamarin tem a sua própria de loja de componentes que são desenvolvidos tanto por especialistas como por programadores independentes. Alguns destes componentes são desenvolvidos para *cross-platform* enquanto que outros dão *target* a plataformas específicas. A sua produção pode ser feita através de dois ambientes de desenvolvimento, o *Xamarin Studio* e o *Visual Studio*.

2.4.3 *Hybrid Apps*

Uma aplicação híbrida é uma aplicação *cross-platform* que recorre ao uso de componentes *web* para renderização da *interface* gráfica do utilizador com HTML5, CSS e JavaScript [42]. Em vez de terem como *target* um *browser*, têm uma *webView* que se encontra dentro de um *container* nativo. Desta forma é possível acederem a capacidades do *hardware* presentes nos dispositivos.

Apache Cordova

Apache Cordova é uma *framework open-source* que permite o desenvolvimento *mobile cross-platform* recorrendo ao uso de tecnologias *web* [47]. As aplicações implementam uma *web page* que contêm referências CSS, Javascript e imagens. A aplicação é executada numa *webView* com um *wrapper* [6] nativo da plataforma de destino.

A *webView* presente na *interface* gráfica do utilizador é carregada quando a aplicação inicia, sendo que à medida que o utilizador interage com o conteúdo disponível, os *links* ou código JavaScript carregam nova informação empacotada na *internet* ou em *web services*.

Antes da versão 3.0 do Cordova, as APIs foram implementadas numa única *interface* de JavaScript exposta para todas as aplicações. Desta forma, mesmo que a aplicação não fizesse uso de todos os recursos, o código seria integrado no total, não sendo possível

[6] *Wrapper* é uma *class* que contem uma instância para outra *class*, mas que não expõe essa instância diretamente. Ou seja, pode expor diferentes funcionalidades da do *wrapped object*.[48]

removê-lo. A partir da versão 3.0 as APIs passaram a ser disponibilizadas no formato de *plugins*.

Os *plugins* são uma parte integral do ecossistema do Cordova. Eles fornecem uma *interface* que permite a comunicação entre o Cordova e os componentes nativos do dispositivo. Desta forma é possível invocar as APIs nativas necessárias através de Javacript. Estes *plugins* encontram-se numa das camadas da arquitetura do Cordova, chamada *Core Plugins*. Assim é possível usar capacidades específicas dos dispositivos como a câmara, bateria ou contactos. Na Figura 2.12 é possível observar toda a sua arquitetura e a forma como os diversos componentes se relacionam entre si.

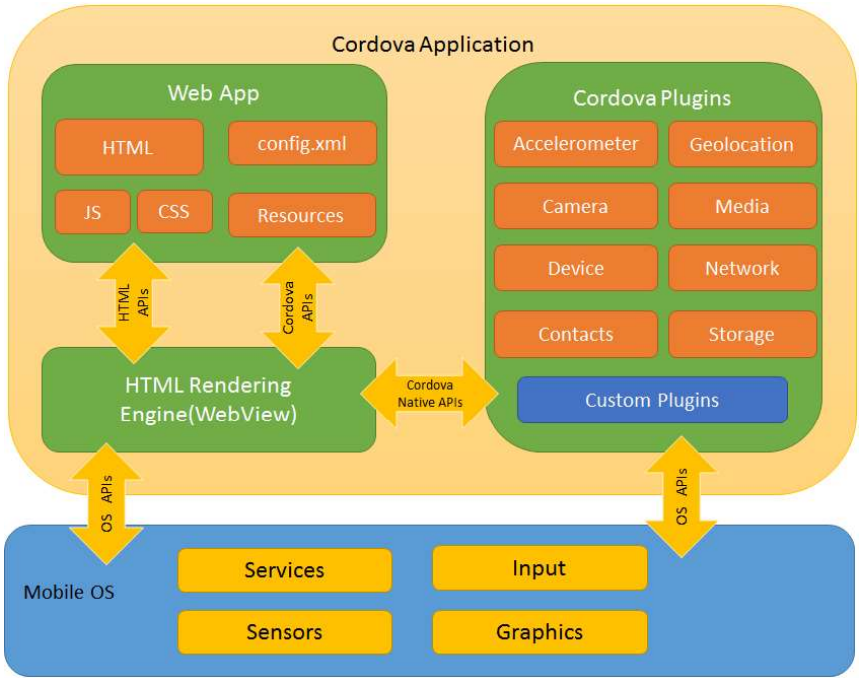

Figura 2.12: Arquitetura Cordova [47]

2.4.4 *Cross-Platform vs Native vs Hybrid*

	Cross-Platform	**Native**	**Hybrid**
Tecnologias	C# com .Net framework + bibliotecas nativas (Xamarin), Javascript (React).	Diferente consoante a plataforma (Android - Java, iOS - Swift/Objective-c).	Javascript, HTML5 e CSS.
Partilha de Código	Sim, ate 96% com Xamarim.Forms.	Não, código base diferente.	Sim, 100%.

UI/UX	Comum para cada plataforma com possibilidade de personalização individual.	Específico de cada plataforma.	Comum para cada plataforma (personalizações limitadas).
Performance	Boa, perto do nativo.	Excelente.	Médio - Fraca comparado com nativo.
Capacidades do *Hardware*	Alto - São usadas as APIs específicas de cada plataforma assim como bibliotecas nativas.	Alto - Ferramentas nativas têm suporte completo das capacidades do sistema	Médio - As capacidades podem ser acedidas através de APIs de terceiros e *plugins*, no entanto existem alguns riscos devido à pobre qualidade e fiabilidade das maioria das ferramentas.

Tabela 2.1: *Cross-Platform vs Native vs Hybrid* [49]

Ao observar a Tabela 2.1 é possível identificar algumas **vantagens** e **desvantagens** sobre o tipo de aplicação a desenvolver. Se o objetivo for desenvolver uma aplicação para **diferentes sistemas operativos**, o caminho a tomar será algo do tipo ***cross-platform*** ou **híbrida**. Dentro destas escolhas, caso a aplicação a desenvolver seja complexa com um UI rico em personalizações, seja necessário alto desempenho e com possibilidade de necessitar de aceder aos componentes nativos do *hardware*, será preferível usar algo do tipo *cross-platform* como o Xamarin, dado que garante uma *performance*/personalização ao nível do nativo. Por outro lado, se a aplicação for **simples** sem grandes pormenores gráficos e com recurso médio às capacidades do *hardware*, então algo como o **Cordova** vai bastar, garantindo uma aplicação de boa qualidade quase ao nível do nativo. Além destas considerações, há que ter em conta as capacidades das equipas de desenvolvimento de se encontrarem familiarizadas com as tecnologias necessárias. Uma equipa que esteja familiarizada com o **desenvolvimento *web*** terá mais facilidade em desenvolver uma **aplicação híbrida**. Caso a aplicação vise apenas **um tipo de sistema operativo**, como por exemplo Windows 10, não vale a pena desenvolver código que seja partilhado por vários dispositivos, perdendo as vantagens obtidas através das aplicações **nativas**.

2.5 *Mobile DataBases*

Como os dispositivos atribuídos aos inspetores podem não ter acesso à *internet* enquanto estes estejam fora das instalações do cliente, foi necessário analisar bases de dados locais que poderiam servir como armazenamento interno da aplicação. A escolha realizada pela equipa do projeto acabou por recair sobre o SQLite descrito na Secção 2.3.6. Esta escolha

deveu-se à familiarização da tecnologia por parte do aluno, assim como pelas sugestões dos programadores seniores pertencentes à equipa do projeto.

2.5.1 Couchbase Lite

É uma base de dados orientada a documentos armazenados em formato de *JavaScript Object Notation* (JSON) [50]. Cada documento pode ter um ou mais anexos, que são transformados em dados binários não interpretados e guardados de forma separada do documento. Juntamente com *Couchbase Sync Gateway* e *Couchbase Server*, formam o *Couchbase Mobile*, uma base de dados para dispositivos móveis que permite a sincronização (*Couchbase Sync Gateway*) entre a base de dados local (*Couchbase Lite*) e a *cloud* (*Couchbase Server*), desta forma o programador não tem de desenvolver o seu próprio código de sincronização. *Couchbase Lite* fornece uma API nativa para Android e iOS, assim como *plugins* para duas *cross-platform* PhoneGap e Xamarin.

2.5.2 SnappyDB

É uma base de dados chave-valor para o Android que funciona como uma alternativa ao SQLite caso seja necessário uma abordagem NoSQL [51]. Permite armazenamento de dados primitivos e serialização de objetos ou *arrays* de forma *type-safe*. Consegue ter melhor performance em operações de escrita e leitura que o SQLite.

2.5.3 Realm

É uma base de dados orientada a objetos em que a informação armazenada é feita na forma de objetos [52]. Foi a primeira base de dados a ser construida de raiz para funcionar em dispositivos móveis, tendo suporte para os mais diversos tipos de sistemas operativos incluindo Android, iOS, Mac e UWP. Realm é *thread safe* logo é possível aceder aos dados de forma concorrente através de múltiplas *threads* sem problemas. As suas operações mais comuns são mais rápidas que SQLite.

2.5.4 Berkeley DB

Permite a gestão da informação de forma relacional através do uso de APIs do SQLite, no formato chave-valor através de *arrays* de *bytes*, no formato de objetos Java ou ainda em documentos XML [53]. Idenpendetemente da API escolhida, a *Berkeley Database* garante funcionalidades como multi acesso de dados, recuperação automática de falhas, replicação e cifra dos dados. Pode ser implementada em variados sistemas operativos incluindo Android e iOS, apesar de no caso deste último as aplicações não poderem ser usadas na loja devido à incompatibilidade da licença. De notar que derivado ao facto

de fornecer funcionalidades expectáveis nas tradicionais bases de dados cliente/servidor, esta base de dados é considerada uma solução relativamente "pesada".

2.6 Aplicações de Inspeções

Foi feita de forma autónoma uma investigação às aplicações existentes no mercado, com o intuito de encontrar produtos com funcionalidades em comum aos requisitos do projeto. Desta forma, poderiam ser formadas novas ideias que proporcionassem um enriquecimento ao trabalho que seria realizado.

2.6.1 iAuditor

iAuditor é uma aplicação desenvolvida para Windows, Android e iOS, que permite a realização de inspeções e auditorias. É distribuída gratuitamente juntamente com as funcionalidades básicas, ou com um plano de pagamento caso o utilizador queira acesso total [54]. Permite a criação de formulários de inspeção consoante o sector de atividade, tendo já alguns *templates* incorporados. Após início da inspeção é possível para além do preenchimento do formulário, tirar fotografias com anotações, adicionar o local onde se está a realizar a atividade e recolher assinaturas. Após a finalização, é possível gerar um PDF com um resumo da informação de forma a ter um comprovativo do trabalho realizado e enviá-lo por email. Inclui funcionalidades de análise ao desempenho das inspeções realizadas. Existe a possibilidade de realização personalizada dos formulários a consumir pelo utilizador, através do *website* da aplicação. A Figura 2.13 representa a página inicial da aplicação.

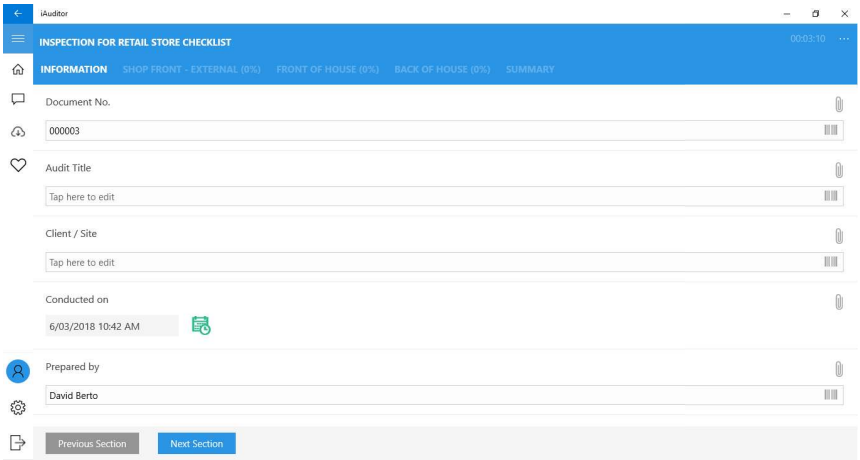

Figura 2.13: iAuditor

2.6.2 Inspection Apps

Inspection Apps é um *software* disponível para iOS com o objetivo de criar formulários à semelhança da aplicação anterior, para atividades de inspeção ou auditoria [55]. Permite vários tipos de inspeções dentro da mesma aplicação, ou seja, consoante a atividade profissional é gerado um grupo de *checklists* com combinações únicas de perguntas. Através do sistema de gestão de dados é possível guardar o trabalho realizado *offline* e sincronizá-lo posteriormente com a *Cloud*, assim como capturar fotografias para apresentar como provas. Possuí ainda funcionalidades como planeamento de inspeções através de lembretes, responder a alertas de *email*, enviar uma cópia do relatório efetuado e controlo de tráfego em tempo real.

2.6.3 Inspect THIS!

Inspect THIS! é um *software* nativo desenvolvido para Android e iOS que também permite o aumento da *performance* no desenvolvimento de inspeções, auditorias e serviços [56]. Além da componente móvel onde é possível a realização das tarefas também já referidas nas duas aplicações anteriores, possui uma componente *web* gerida pelos administradores do sistema onde é realizada a gestão das inspeções, os custos associados às deslocações e o controlo dos utilizadores que vão para o terreno fazer uso da aplicação móvel. Na Figura 2.14 é possível observar a versão da aplicação para iPad assim como o modo de planeamento presente na *web*.

(a) Versão *mobile* (b) Planeamento *Web*

Figura 2.14: Inspect THIS! [56]

Como é possível observar, já existem diferentes tipos de soluções tanto para o problema de desmaterialização de documentos, como para o processo de inspeção e auditoria realizado por diversas organizações. No entanto, ainda é possível notar alguma ausência do uso destas aplicações no mercado nacional, continuando a serem usados os métodos tradicionais com recurso a vários documentos físicos. Esta relutância à mudança pode ser explicada por vários fatores, tais como, o elevado preço de aquisição e manutenção destes serviços e o facto de na maioria das vezes, o cliente procurar uma solução mais adaptada e personalizada ao seu problema. Também há que ter em conta a complexidade do negócio.

Capítulo 3

Metodologia e Planeamento

Este capítulo encontra-se omisso por confidencialidade.

Capítulo 4

Análise e Desenho

Este capítulo encontra-se omisso por confidencialidade.

Capítulo 5

Desenvolvimento e Testes

Este capítulo encontra-se omisso por confidencialidade.

Capítulo 6

Considerações Finais

Este capítulo apresenta uma abordagem geral sobre o período de desenvolvimento do projeto que decorreu no cliente da Accenture. Será feita uma análise ao trabalho realizado na aplicação móvel, identificando os principais desafios e obstáculos que foram surgindo e a forma como estes foram resolvidos. Por fim é feita uma reflexão crítica do que poderia ter corrido melhor, ou o que poderia ter sido utilizado durante a fase de desenvolvimento, assim como a sugestão de melhoria futura.

6.1 Trabalho Realizado

Este projeto foi efetuado no âmbito da realização do DPEI da FCUL na empresa Accenture e tem como objetivo a realização de um projeto em ambiente empresarial. Foi possível por em prática os conhecimentos de Engenharia de *Software* adquiridos durante o mestrado, através do desenvolvimento de uma aplicação móvel para o cliente da empresa de acolhimento. Sendo assim, este projeto foi dividido em dois semestres.

O primeiro semestre foi dedicado ao planeamento e começou com o estudo das tecnologias que viriam a ser utilizadas, através da criação de aplicações *mockup* que mais tarde foram úteis para o desenvolvimento do projeto. Foram desenvolvidos também protótipos de alta fidelidade, através da análise de requisitos obtidos a partir de reuniões com o cliente. Estes protótipos serviram não só para aproximar as ideias do cliente com as dos programadores, como para a realização de testes de usabilidade aquando das apresentações. Juntamente com a análise de requisitos foi feita uma análise funcional e técnica, recorrendo ao desenho de estilos arquiteturais. Estes assuntos foram abordados na disciplina de *Design de Software*.

O segundo semestre foi mais orientado para o desenvolvimento da aplicação móvel. Apesar de como explicado no decorrer do documento, o fluxo e os requisitos ainda estarem a ser definidos, foi feita a consolidação da fase anterior, com a definição de modelo de dados e serviços necessários, e posterior implementação e configuração da solução encontrada. Foram ainda elaborados alguns testes ao sistema, apesar de que, devido a al-

guns atrasos e motivos alheios ao aluno, a aplicação acabou por não entrar em produção, acabando por ficar como prova de conceito. Durante o decorrer do segundo semestre foi escrito o relatório de tese.

6.2 Principais Obstáculos

A demora da recolha dos requisitos funcionais, assim como a indefinição do fluxo de atividade das tarefas a serem desmaterializadas, representaram o principal obstáculo ao desenvolvimento deste projeto. Este atraso ocorreu pois existiam outros projetos com maior prioridade que necessitavam de ser concluídos, assim como algumas dúvidas de como a aplicação iria funcionar. Deste modo, foi necessário realizar várias reuniões até se obter uma decisão final por parte do cliente.

Durante o desenvolvimento da aplicação foram encontrados alguns desafios interessantes que necessitaram de uma maior dedicação e estudo do problema. O facto do UWP ser executado num ambiente controlado, com uma plataforma de desenvolvimento recente (.NET Core) e ser necessário o uso de bibliotecas mais antigas presentes na plataforma do .NET Framework para corresponder aos requisitos do cliente, contribuiu para uma maior aprendizagem do universo de desenvolvimento Microsoft. O uso das ferramentas presentes nestas bibliotecas, tais como a estampagem de assinaturas em documentos assinados digitalmente, também levantou preocupações de validade, que mais tarde foram resolvidas após uma investigação à documentação da Adobe.

Dada a falta de experiência do aluno, a falta de uniformização da forma com as tarefas de inspeção são atualmente realizadas, assim como a dimensão do projeto onde se engloba esta aplicação móvel, ocorreram alguns atrasos no desenvolvimento das funcionalidades, levando assim ao atraso da sua entrada em produção.

6.3 Reflexão Crítica

O principal objetivo deste projeto foi proporcionar ao aluno a nível académico uma experiência no mercado de trabalho, onde fosse possível fazer uso das principais matérias aprendidas no decorrer do mestrado em Engenharia de *Software*. Do lado da empresa, o objetivo era desenvolver uma aplicação móvel em UWP, que desmaterializasse os principais setores de inspeção do cliente e que pudesse ser integrada em *tablets*. Do lado académico considerou-se que o objetivo foi cumprido, pois no decorrer do projeto foi possível trabalhar com as mais diversas tecnologias e ferramentas nunca antes exploradas, assim como, aplicar conhecimentos teóricos adquiridos em algumas disciplinas da faculdade. Do ponto de vista da aplicação, apesar desta não ter entrado em produção como era pretendido, foram cumpridos os requisitos possíveis e o cliente ficou satisfeito com a prova de conceito alcançada, ficando bastante entusiasmado para o futuro e evolução da

mesma. Dada a dimensão de todo o projeto, que inclui a aplicação móvel, este deveria ter sido melhor planeado e tido um acompanhamento mais presente do cliente de forma a não só diminuir os atrasos, como proporcionar um maior apoio aos programadores a nível de conhecimento do negócio. No entanto, uma vez que é necessário uniformizar as atividades que são elaboradas de maneiras diferentes há vários anos, é normal que surjam algumas dificuldades também por parte do cliente em transmitir o necessário à equipa de desenvolvimento.

Analisando algumas das tecnologias utilizadas e tendo em conta a pesquisa efetuada no início do projeto, existem alguns pontos que poderiam ter sido diferentes. Desde do primeiro dia que foi dito por parte do cliente, que esta aplicação seria integrada em *tablets* e que faria uso de mapas, dado existirem deslocações ao exterior, sendo estes uma mais valia na orientação do inspetor. Partindo deste requisito, como a aplicação é restrita a um tipo de aparelho e de modo a ser possível fazer uso de todo o seu potencial, surge de forma natural a escolha de uma *framework* nativa. No entanto, dada a variedade de *tablets* Android existente no mercado e de este sistema operativo fazer uso dos mapas da Google, que são bastante ricos em pontos de interesse e funcionalidades disponíveis, seria o tipo de sistema operativo possivelmente ideal para o desenvolvimento da aplicação. Tanto o sistema operativo Windows como os iOS não dispõe de tanta variedade de dispositivos com o Android, os seus custos são mais elevados e ambos os mapas nativos não são tão bons e ricos como os da Google[1]. No entanto, compreende-se a decisão tomada em usar UWP, pois até agora os inspetores só trabalharam com um tipo de sistema operativo, Windows, sendo que a ideia do cliente passaria por adquirir *tablets* híbridos que não só pudessem ser levados para o exterior, como substituiriam as máquinas que os inspetores atualmente possuem no seu posto de trabalho. Desta forma, o que se perde por não usar mapas da Google, ganha-se na adaptação ao novo método de trabalho.

Por fim, o tipo de base de dados local que deveria ser usado também foi alvo de investigação. Esta escolha ficou a cargo da equipa de programadores e tendo em conta a sua experiência com base de dados relacionais, acabou por ser esse o modelo escolhido. Como a Microsoft não oferece suporte oficial a mais nenhuma base de dados local para UWP sem ser o SQLite (Secção 2.3.6), e como o aluno encontra-se familiarizado com esta tecnologia acabou por ser esta a base de dados escolhida. No entanto, durante a pesquisa efetuada, o Realm apareceu como alternativa, tendo também suporte não oficial para UWP. Apesar das vantagens enumeradas na Secção 2.5.3, na construção dos objetos da solução é necessário que estes herdem a propriedade *RealmObject*. Como a equipa de desenvolvimento traçou como objetivo reduzir o maior número de dependências do projeto ao tipo de base de dados usado, caso fosse necessário alterá-la no futuro, não viu como boa prática ter estas dependências ao longo dos ficheiros, abandonando assim esta

[1] `https://www.macworld.co.uk/review/reference-education/apple-maps-vs-google-maps-3464377/`, `https://blog.capterra.com/bing-maps-vs-google-maps/`

hipótese.

6.4 Trabalho Futuro

Como referido ao longo deste documento, devido a atrasos alheios ao aluno o projeto acabou por não entrar em produção, precisando ainda de alguns meses de desenvolvimento. Uma das tarefas necessárias é a implementação, do lado do sistema central, de uma aplicação que crie os formulários a serem consumidos em formato JSON pela aplicação móvel. Atualmente existem protótipos desenvolvidos em UWP que através do JSON geram os campos para a Tarefa de Inspeção. Na Tarefa de Controlo ainda não foi implementada a validação de documentos de transporte, estando também em discussão a implementação ou não de formulários. Quanto à segurança inerente à aplicação, esta deve ser revista de forma a ser possível cifrar a base de dados local SQLite, apesar desta não ser acessível ao utilizador comum. A *framework* de comunicação também necessita de uns ajustes, dado que inicialmente fora produzida para outras aplicações do cliente. Desta forma até ao final da data proposta para desenvolvimento do projeto, foi possível desenvolver uma prova de conceito com os principais requisitos realizados, sendo que a Tarefa de Recolha de Informação e a Tarefa de Assinatura encontram-se finalizadas, a Tarefa de Inspeção fica a aguardar a produção dos formulários e a Tarefa de Controlo o fluxo de validação dos documentos. Também ainda não foi decidido se o comprovativo da inspeção será responsabilidade do *tablet* ou do sistema central.

Bibliografia

[1] Observador. *Há cada vez mais pessoas a usar o tablet como instrumento de trabalho*. 2015. URL: http://observador.pt/2015/10/20/ha-cada-vez-mais-pessoas-a-usar-o-tablet-como-instrumento-de-trabalho/ (acedido em 28/10/2017).

[2] Accenture Portugal. *Sobre nós*. URL: https://www.linkedin.com/company/3780787/ (acedido em 28/10/2017).

[3] Scrum Portugal. *Agile Manifesto*. 18-04-2011. URL: http://www.scrumportugal.pt/agile-manifesto/ (acedido em 04/11/2017).

[4] Quickscrum. *What Is Agile software development life cycle?* 28-03-2016. URL: https://www.quickscrum.com/Article/ArticleDetails/2031/3/What-Is-Agile-Software-Development-Life-Cycle (acedido em 04/11/2017).

[5] Craig Larman e Victor R Basili. "Iterative and incremental development: A brief history". Em: *Computer*. 2003, 36.6: 47–56.

[6] *Don't Doom Your Business With Mishandled Projects! Avoid These Mistakes*. 19-01-2017. URL: http://www.kristymlopez.com/2017/01/ (acedido em 04/11/2017).

[7] Sourcemaking. *Design Patterns*. URL: https://sourcemaking.com/design_patterns (acedido em 30/05/2018).

[8] ivom1. *Model-View-ViewModel (MVVM) Applications: General Introduction*. 17-03-2012. URL: https://blogs.msdn.microsoft.com/ivo_manolov/2012/03/17/model-view-viewmodel-mvvm-applications-general-introduction/ (acedido em 05/11/2017).

[9] Microsoft. *The MVVM Pattern*. 10-02-2012. URL: https://msdn.microsoft.com/en-us/library/hh848246.aspx (acedido em 25/02/2018).

[10] Microsoft. *The MVVM Pattern*. 8-07-2017. URL: https://docs.microsoft.com/en-us/xamarin/xamarin-forms/enterprise-application-patterns/mvvm (acedido em 25/02/2018).

[11] JohnGossman. *Introduction to Model/View/ViewModel pattern for building WPF apps*. 8-10-2005. URL: https://blogs.msdn.microsoft.com/johngossman/2005/10/08/introduction-to-modelviewviewmodel-pattern-for-building-wpf-apps/ (acedido em 05/11/2017).

[12] Microsoft. *How to: Implement Property Change Notification*. 30-03-2017. URL: https://docs.microsoft.com/en-us/dotnet/framework/wpf/data/how-to-implement-property-change-notification (acedido em 26/02/2018).

[13] Microsoft. *DataContext*. URL: https://docs.microsoft.com/en-us/uwp/api/windows.ui.xaml.frameworkelement#Windows_UI_Xaml_FrameworkElement_DataContext (acedido em 01/03/2018).

[14] Martin Fowler. *Inversion of Control Containers and the Dependency Injection pattern*. 23-02-2004. URL: https://martinfowler.com/articles/injection.html (acedido em 25/02/2018).

[15] Laurent Bugnion. *MVVM - IOC Containers and MVVM*. 2-2013. URL: https://msdn.microsoft.com/en-us/magazine/jj991965.aspx (acedido em 25/02/2018).

[16] Joel Abrahamsson. *Inversion of Control – An Introduction with Examples in .NET*. 29-03-2010. URL: http://joelabrahamsson.com/inversion-of-control-an-introduction-with-examples-in-net/ (acedido em 25/02/2018).

[17] Microsoft. *The Service Locator Pattern*. URL: https://msdn.microsoft.com/en-us/library/ff648968.aspx (acedido em 25/02/2018).

[18] Loredana Crusoveanu. *Intro to Inversion of Control and Dependency Injection with Spring*. 25-02-2018. URL: http://www.baeldung.com/inversion-control-and-dependency-injection-in-spring (acedido em 26/02/2018).

[19] Jon Skeet. *Implementing the Singleton Pattern in C#*. 9-2013. URL: http://csharpindepth.com/Articles/General/Singleton.aspx (acedido em 26/02/2018).

[20] Richard Carr. *Singleton Design Pattern*. 25-06-2008. URL: http://www.blackwasp.co.uk/Singleton.aspx (acedido em 26/02/2018).

[21] Microsoft. *Desktop app bridge to UWP Samples*. 8-12-2017. URL: https://github.com/Microsoft/DesktopBridgeToUWP-Samples (acedido em 26/02/2018).

[22] Microsoft. *Software Architecture in Practice*. 25-01-2018. URL: https://docs.microsoft.com/en-us/dotnet/framework/wpf/ (acedido em 09/03/2018).

[23] Michael Satran Norm Estabrook e Alexander Koren. *Asynchronous programming*. 2-08-2017. URL: https://docs.microsoft.com/en-us/windows/uwp/threading-async/asynchronous-programming-universal-windows-platform-apps (acedido em 26/02/2018).

[24] Microsoft. *Visual Studio*. URL: https://www.visualstudio.com/pt-br/?rr=https%3A%2F%2Fwww.google.pt%2F (acedido em 04/11/2017).

[25] Microsoft. *C# Language*. 20-07-2015. URL: https://docs.microsoft.com/en-us/dotnet/csharp/getting-started/introduction-to-the-csharp-language-and-the-net-framework (acedido em 04/11/2017).

[26] Microsoft. *What is XAML?* URL: https://msdn.microsoft.com/en-us/library/cc295302.aspx (acedido em 19/03/2018).

[27] Microsoft. *Intro to the Universal Windows Platform*. 27-10-2017. URL: https://docs.microsoft.com/en-us/windows/uwp/get-started/universal-application-platform-guide (acedido em 05/11/2017).

[28] Microsoft. *Device families*. 27-10-2017. URL: https://docs.microsoft.com/pt-pt/windows/uwp/get-started/universal-application-platform-guide (acedido em 11/11/2017).

[29] Wikipedia. *Software development kit*. 10-10-2017. URL: https://en.wikipedia.org/wiki/Software_development_kit (acedido em 05/11/2017).

[30] Chuck Walbourn. *UWP limitations in desktop apps*. 29-03-2016. URL: https://stackoverflow.com/questions/36286806/uwp-limitations-in-desktop-apps (acedido em 09/03/2018).

[31] Cesar de la Torre. *.NET Core, .NET Framework, Xamarin – The "WHAT and WHEN to use it"*. 27-06-2016. URL: https://blogs.msdn.microsoft.com/cesardelatorre/2016/06/27/net-core-1-0-net-framework-xamarin-the-whatand-when-to-use-it/ (acedido em 01/03/2018).

[32] Kraig Brockschmidt e Alfred Myers. *An introduction to NuGet*. 10-01-2018. URL: https://docs.microsoft.com/en-us/nuget/what-is-nuget (acedido em 02/03/2018).

[33] Sqlite. *About SQLite*. 1-11-2017. URL: https://www.sqlite.org/about.html (acedido em 05/11/2017).

[34] IBM. *ACID properties of transactions*. URL: https://www.ibm.com/support/knowledgecenter/en/SSGMCP_5.3.0/com.ibm.cics.ts.productoverview.doc/concepts/acid.html (acedido em 05/11/2017).

[35] Techopedia. *iOS*. URL: https://www.techopedia.com/definition/25206/ios (acedido em 05/03/2018).

[36] Apple. *About Developing for Mac*. 16-09-2015. URL: https://developer.apple.com/library/content/documentation/MacOSX/Conceptual/OSX_Technology_Overview/CoreServicesLayer/CoreServicesLayer.html (acedido em 05/03/2018).

[37] Thorin Klosowski. *I Want to Write iOS Apps. Where Do I Start?* 10-10-2014. URL: https://lifehacker.com/i-want-to-write-ios-apps-where-do-i-start-1644802175 (acedido em 05/03/2018).

[38] JOHN CALLAHAM. *The history of Android OS: its name, origin and more*. 14-01-2018. URL: https://www.androidauthority.com/history-android-os-name-789433/ (acedido em 05/03/2018).

[39] IDC. *Smartphone OS Market Share, 2017 Q1*. 5-2017. URL: https://www.idc.com/promo/smartphone-market-share/os (acedido em 05/03/2018).

[40] Tutorialspoint. *Android - Architecture*. URL: https://www.tutorialspoint.com/android/android_architecture.htm (acedido em 05/03/2018).

[41] Google. *Android Studio Release Notes*. 11-2017. URL: https://developer.android.com/studio/releases/index.html (acedido em 05/03/2018).

[42] Chris Rickard. *Choosing the right mobile app for your project: Native vs cross-platform vs hybrid*. 21-01-2016. URL: http://inoutput.io/articles/development/choosing-the-right-mobile-app-for-your-project-native-vs-cross-platform-vs-hybrid (acedido em 01/03/2018).

[43] Hramos. *React Native*. 5-03-2018. URL: https://github.com/facebook/react-native (acedido em 05/03/2018).

[44] Bonnie Eisenman. "Learning React Native: Building Mobile Applications with JavaScript". Em: *O'Reilly Media, Inc*. California, 2015, pp. 27,35.

[45] Microsoft. *Learn about mobile development with Xamarin*. 2015. URL: https://msdn.microsoft.com/en-us/library/mt488768.aspx (acedido em 19/03/2018).

[46] James Montemagno. *Getting Started with Xamarin and Xamarin.Forms*. 23-02-2018. URL: https://montemagno.com/getting-started-with-xamarin-q-a/ (acedido em 19/03/2018).

[47] Dblotsky. *Architectural overview of Cordova platform*. 22-04-2016. URL: https://cordova.apache.org/docs/en/latest/guide/overview/index.html (acedido em 05/03/2018).

[48] Alex Humphrey. *Where and how is the term used WRAPPER in programming, what does it help to do?* 21-07-2010. URL: https://stackoverflow.com/questions/3293752/where-and-how-is-the-term-used-wrapper-in-programming-what-does-it-help-to-do (acedido em 05/03/2018).

[49] Andrew Klubnikin. *Cross-platform vs. Native Mobile App Development: Choosing the Right Dev Tools for Your App Project*. 24-05-2017. URL: https://medium.com/all-technology-feeds/cross-platform-vs-native-mobile-app-development-choosing-the-right-dev-tools-for-your-app-project-47d0abafee81 (acedido em 06/03/2018).

[50] KATERINA ROUKOUNAKI. *Five popular databases for mobile*. 10-09-2014. URL: https://www.developereconomics.com/five-popular-databases-for-mobile (acedido em 16/03/2018).

[51] Nabil HACHICHA. *SnappyDB*. URL: http://www.snappydb.com/ (acedido em 15/03/2018).

[52] Realm. *Realm Database*. URL: https://realm.io/products/realm-database (acedido em 16/03/2018).

[53] KATERINA ROUKOUNAKI. *Five popular databases for mobile*. 10-09-2014. URL: https://www.developereconomics.com/five-popular-databases-for-mobile (acedido em 16/03/2018).

[54] safetyculture. *Build checklists, conduct inspections, file reports*. URL: https://safetyculture.com/iauditor/ (acedido em 06/03/2018).

[55] Inspection Apps. *Inspection Apps*. 2013. URL: http://www.inspectionapps.com (acedido em 19/03/2018).

[56] Inspect THIS! *Mobile Inspection App*. URL: https://inspectthis.net/ (acedido em 19/03/2018).

[57] Microsoft. *What is .NET?* URL: https://www.microsoft.com/net/learn/what-is-dotnet (acedido em 05/11/2017).

[58] Engadget Kris Naudus. *Here's what our readers think of Windows 10*. 8-2015. URL: https://www.engadget.com/2015/08/20/windows-10-reader-review-roundup/ (acedido em 11/11/2017).

[59] Usability.gov. *User Interface Design Basics*. URL: https://www.usability.gov/what-and-why/user-interface-design.html (acedido em 11/11/2017).

[60] Shivprasad koirala. *Can you explain Lazy Loading?* 16-09-2013. URL: https://www.codeproject.com/Articles/652556/Can-you-explain-Lazy-Loading (acedido em 19/11/2017).

[61] SinficWeb. *Método de Avaliação do Risco em Software*. 2005. URL: http://www.sinfic.pt/SinficWeb/displayconteudo.do2?numero=24252 (acedido em 07/03/2018).

[62] N. Lavanya e T. Malarvizhi. *Risk analysis and management: a vital key to effective project management. Paper presented at PMI® Global Congress 2008—Asia Pacific, Sydney, New South Wales, Australia. Newtown Square, PA: Project Management Institute.* 2008. URL: https://www.pmi.org/learning/library/risk-analysis-project-management-7070 (acedido em 07/03/2018).

[63] Mountain Goat Software. *User Stories*. URL: https://www.mountaingoatsoftware.com/agile/user-stories (acedido em 07/03/2018).

[64] Google Sites. *Software Architecture in Practice*. URL: https://sites.google.com/site/softwarearchitectureinpractice/9-documenting-software-architecture/b-module-views/1-decomposition-view (acedido em 09/03/2018).

[65] w3schools. *XML SOAP*. URL: https://www.w3schools.com/xml/xml_soap.asp (acedido em 15/03/2018).

[66] *HTTP*. 14-12-2017. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP (acedido em 15/03/2018).

[67] InstantSSL. *What is HTTPS?* URL: https://www.instantssl.com/ssl-certificate-products/https.html (acedido em 15/03/2018).

[68] w3schools. *XML WSDL*. URL: https://www.w3schools.com/xml/xml_wsdl.asp (acedido em 15/03/2018).

[69] Margaret Rouse. *REST (REpresentational State Transfer)*. 12-2017. URL: http://searchmicroservices.techtarget.com/definition/REST-representational-state-transfer (acedido em 15/03/2018).

[70] w3schools. *JSON - Introduction*. URL: https://www.w3schools.com/js/js_json_intro.asp (acedido em 15/03/2018).

[71] *ISO 9241-11: Guidance on Usability (1998)*. URL: http://www.usabilitynet.org/tools/r_international.htm#9241-11 (acedido em 22/03/2018).

[72] IBM. *WS-Security*. 21-07-2017. URL: https://www.ibm.com/support/knowledgecenter/en/SSMKHH_9.0.0/com.ibm.etools.mft.doc/ac55630_.htm (acedido em 02/03/2018).

Apêndice A

Metodologia e Planeamento

Este apêndice encontra-se omisso por confidencialidade.