DM

Apresentação e Análise do Ecossistema para Desenvolvimento Mobile em Multiplataforma

DISSERTAÇÃO DE MESTRADO

Ricardo Lucas Jardim
MESTRADO EM ENGENHARIA INFORMÁTICA

UNIVERSIDADE da MADEIRA
A Nossa Universidade
www.uma.pt

outubro | 2021

Apresentação e Análise do Ecossistema para Desenvolvimento Mobile em Multiplataforma

DISSERTAÇÃO DE MESTRADO

Ricardo Lucas Jardim
MESTRADO EM ENGENHARIA INFORMÁTICA

ORIENTAÇÃO
Filipe Magno de Gouveia Quintal

FCEE

Mestrado em Engenharia Informática

Apresentação e Análise do Ecossistema para Desenvolvimento Mobile em Multiplataforma

Ricardo Lucas Jardim

supervisionado por

Filipe Quintal

13 de dezembro de 2021

Agradecimentos

Após cinco anos, a minha jornada académica chega ao fim. Durante estes anos conheci algumas pessoas importantes a quem devo agradecer por fazerem parte desta grande caminhada, em que sempre me incentivaram a continuar e nunca desistir.

Começo assim por agradecer aos meus pais e à minha namorada por me apoiarem, confiando que iria chegar ao fim de mais uma etapa da minha vida com sucesso. Agradeço também a todos os meus amigos que me ajudaram ao longo destes cinco anos, tornando este caminho mais fácil e alegre nas situações mais difíceis.

Por fim, gostaria de agradecer a todos os professores que fizeram parte desta jornada, particularmente ao meu orientador Filipe Quintal, por toda a dedicação, paciência, empenho, todas as críticas e pelo conhecimento transmitiu, tornando, assim possível a concretização desta dissertação.

Ricardo Lucas Jardim

Resumo

Esta tese foca-se em problemas identificados no mercado das tecnologias Cross-Platform; na enorme variedade de soluções disponíveis; no conhecimento dos diferentes comportamentos, estruturas, custo e esforço de implementação em comparação com o desempenho desejado; e por fim, a falta de guias analíticos não só a nível de implementação, mas também de compatibilidade e escalabilidade tornando-se difícil para um *developer* escolher a melhor solução a utilizar.

Diversos investigadores empenharam-se em criar soluções para alguns destes problemas, comparando tecnologias de desenvolvimento mobile, identificando as suas vulnerabilidades e problemas que possam limitar alguns aspetos no desenvolvimento de aplicações. Do mesmo modo, dedicaram-se em criar métodos para avaliar uma tecnologia de desenvolvimento mobile, desde questionários ao público alvo, ao desenvolvimento de provas de conceito para realização testes de desempenho e *benchmarks*.

Nesta tese, tentamos juntar contribuições de diversas áreas de estudo na avaliação de ferramentas e tecnologias para a criação de aplicações mobile Cross-Platform. Desenvolvemos, uma revisão de literatura extensiva de forma a identificar as principais diferenças entre as tecnologias mobile nativas e Cross-Platform; uma análise individual das várias tecnologias Cross-Platform; e uma análise comparativa entre as tecnologias, em conjunto com uma componente quantitativa abordando os *developers*. Os nossos resultados demonstram que a tecnologia que providenciou um melhor desempenho nos casos de teste foi o Xamarin. Observou-se ainda que a tecnologia Cordova tem vantagens a nível de consumo de recursos. Por outro lado, os *developers* para o desenvolvimento de aplicações mobile, têm preferência nas tecnologias Ionic e React Native. Verificou-se, também, que o consumo de recursos depende da solução criada pela tecnologia e não da abordagem seguida. As tecnologias analisadas apresentam semelhanças, na escolha de linguagens, estruturas de código, entre outros. Porém, também possuem limitações idênticas, sendo que o maior problema localiza-se na área da manutenção de uma aplicação.

Palavras-chaves: Dispositivo, Multiplataforma, Desenvolvedor, Benchmarks, Plug-in's, Browser.

Abstract

This thesis focuses on problems identified in the market of mobile Cross-Platform technologies, the huge variety of solutions available, the knowledge of the different behaviors, structures, implementation cost and effort in comparison with the desired performance, and finally, the lack of analytical guides not only in terms of implementation but also of compatibility and scalability, making it difficult to choose the best solution to use. Several researchers have worked to create solutions for some of these problems, creating comparisons between mobile development technologies, identifying their vulnerabilities and problems that may limit some aspects of application development. In the same way, they have dedicated themselves to creating methods to evaluate a mobile development technology, from questionnaires to the development of proofs of concept to performance tests and benchmarks.

In this thesis we tried to bring together contributions from several areas of study in the evaluation of tools and technologies for creating Cross-platform mobile applications. Namely, we developed, an extensive literature review in order to identify the main differences between native and Cross-Platform mobile technologies, an individual analysis of the various Cross-Platform technologies, a comparative analysis between various Cross-Platform technologies together with a quantitative component addressing developers. We hope that with these objectives, we can solve the problems identified, facilitating the choice of a technology depending on the proposed situation, as well as pointing out its constraints and problems that may be encountered. Our results show that the developers for mobile application development have a preference for the Ionic and React Native technologies. We also observed that Cordova has advantages in terms of resource consumption. The technology that provided the best performance in the test cases was Xamarin. It was also found that resource consumption depends on the solution created by the technology and not on the approach it follows.

Keywords: Mobile · Cross-Platform.· Developer · Benchmarks · Plug-in's · Browser

Conteúdo

Lista de Figuras

Lista de Tabelas

Nomenclatura

API Interface de Programação de Aplicações

AppStore Loja de aplicações

AVD Dispositivo Virtual Android

Bridge Padrão de desenho utilizado para não causar dependências entre objetos

CDN Distruição de conteúdo online

CLI Interface de Linhas de Comandos

Cross − Platform Multiplataforma

DSL Linguagem de domínio específico

HTTPS Hyper Text Transfer Protocol Secure

IDE Ambiente de desenvolvimento integrado

MVC Model-View-Controller

POO Programação Orientada por Objetos

Progressivo Aplicação que funciona independentemente do *browser* escolhido

PWA Progressive Web App

Responsivo Aplicação que se adequa a qualquer plataforma: Desktop, Mobile e Tablet

SDK Kit de desenvolvimento de software

Smartphone Telemóvel inteligente

SO Sistema Operativo

URL Uniform Resource Locator

WebView Componente nativo que permite aceder a conteúdo web

Runtime Tempo de execução

1 Introdução

O aparecimento do computador, em 1946, permitiu uma grande evolução a nível tecnológico, e de computação. Com este grande poder computacional, surgiu uma nova era, que originou várias tecnologias que todos nós interagimos diariamente. Com a rápida evolução da humanidade, as tecnologias tornaram-se cada vez mais portáteis e acessíveis, ao ponto que, hoje em dia, acomodam-se discretamente no nosso bolso, oferecendo um enorme poder computacional [74].

Estes dispositivos móveis estão rapidamente a tornar-se cada vez mais presentes na vida quotidiana de todos nós, graças às suas capacidades informáticas e amplo conjunto de diferentes sensores. Desempenhando assim, tanto um papel de local de trabalho móvel, como também de comunicação, utilização pessoal e lazer, capaz de conhecer o ambiente e monitorizar as atividades dos utilizadores [29]. Devido à sua rápida evolução, o mercado de smartphones cresceu de forma exponencial, pequenas e grandes empresas começaram a desenvolver os seus smartphones e sistema operativos (SO), como por exemplo, a Nokia e BlackBerry foram pioneiras ao produzir um SO capaz de suportar aplicações realizadas por terceiros, utilizando os seus próprios dispositivos. Com esta evolução, o mercado tornou-se assim, um mercado mais acessível e abrangente, tanto a nível monetário como a nível computacional. Durante esta evolução, três empresas emergiram como líderes do mercado com os seus SO's, sendo estas, a Google com o Android, Apple com o iOS e Microsoft com o Windows Phone [42]. Neste momento, o mercado é liderado pela Google com 86,7% do mercado em 2019 [69], logo de seguida está a Apple com 13,3% em 2019 e Microsoft com a menor percentagem do mercado, cada uma com o seu mercado privado de aplicações [107]. O SO Android lidera o mercado, isto porque, não está bloqueado a um fabricante de hardware e poderá ser licenciado por qualquer fabricante que deseje oferecer um smartphone com este sistema, enquanto o SO iOS só pode ser utilizado em dispositivos Apple [69].

As diferenças entre estes SO's, demonstram que o desenvolvimento de aplicações nativas é, também, distinto. Utilizando-se o IDE Android Studio em conjunto com a linguagem JAVA e/ou Kotlin para construir aplicações para o SO Android, o IDE Xcode em conjunto com Objetive-C e/ou Swift para aplicação iOS e por fim, o IDE Visual Studio, juntamente com a linguagem C# e/ou C++ para produzir aplicações para o Windows Phone [69] [57]. Através destas tecnologias, inúmeras aplicações foram construídas. Contudo esta fragmentação revelou-se também problemática.

Esta diversidade de IDE e linguagens, gerou alguns problemas aos *developers*. Os conteúdos das plataformas são diferentes e comportam-se de forma diferente, a mesma aplicação terá que ser construída em várias linguagens e tecnologias para ser suportada em diferentes plataformas. Este facto poderá tornar o processo de construção de aplicações demoroso e dispendioso. De modo a solucionar este problema diversas empresas desenvolveram abordagens Cross-Platform [114]. O aparecimento de tecnologias Cross-Platform permitiu a redução do custo e esforço de desenvolvimento, redução de recursos, aumento da velocidade de construção e reutilização de código, utilizando apenas uma linguagem [22] [35]. Algumas destas tecnologias Cross-Platform são baseadas em conteúdo Web, possibilitando uma grande redução no custo e esforço de desenvolvimento, graças à grande familiarização destas linguagens.

1.1 Descrição do Problema

Nos dias de hoje, existe uma enorme variedade de soluções para cada tipo de aplicação e *feature* a desenvolver, tornando difícil a escolha da melhor tecnologia a utilizar. Além disso, o suporte destas tecnologias é também fragmentado, visto que algumas não têm a versatilidade de apoiar uma vasta variedade de versões de SO's, limitando-se apenas a algumas versões mais antigas desses SO's, consequentemente, geram grandes problemas em termos de manutenção, suporte e segurança [89].

Visto que o mercado das tecnologias Cross-Platform cresceu exponencialmente, estas começaram a oferecer diferentes abordagens para o mesmo objetivo Cross-Platform, desde PWA's (Progressive Web App) a tecnologias que compilam para código nativo [42]. A escolha de uma tecnologia para o desenvolvimento de uma aplicação é muito importante, em função do facto de, todas estas possuírem diferentes comportamentos, estruturas, linguagens e o mais importante, custo e esforço de implementação em comparação com o desempenho desejado [89]. Todas estas tecnologias contêm também limitações, inerentes à sua arquitetura, ambiente de desenvolvimento ou mesmo limitações relativas ao seu âmbito.

Este fracionamento poderá gerar grandes dificuldades para empresas e *developers* na escolha da melhor tecnologia para o seu problema inicial. A escolha da tecnologia certa é de grande importância, como por exemplo, se uma empresa escolhe uma tecnologia aleatória, posteriormente poderá encontrar graves problemas de desempenho, falta de suporte, entre outros problemas inesperados, sendo obrigada a recomeçar todo o processo de desenvolvimento em outra tecnologia [33], ou oferecer um produto que não vai ao encontro dos requisitos iniciais.

Devido a estes problemas, e à falta de guias analíticos, não só a nível de implementação, mas também de compatibilidade, torna-se difícil escolher qual é a melhor solução para um problema proposto. Muitas destas tecnologias têm problemas de escalabilidade, tornando-as apenas adequadas para projetos pequenos, restringindo a evolução do sistema [19]. Por este motivo, e pelo facto de ser necessário investir tempo para analisar e compreender as limitações de uma tecnologia Cross-Platform, muitas empresas continuam a ter equipas de desenvolvimento distintas para implementações nativas em cada um dos sistemas operativos [35].

1.2 Solução Proposta

De forma a resolver os problemas detalhados anteriormente, alguns investigadores e empresas começaram por comparar soluções Cross-Platform, utilizando apenas alguns parâmetros comuns, como o, consumo de bateria, comunicação com servidores, SO's suportados, linguagem, licenças e acesso às API's do sistema [89][33][8].

Dito isto, esta dissertação irá se basear numa revisão de literatura, sobre tecnologias para construção de aplicações móveis, além dos métodos e métricas utilizadas para as avaliar. A principal contribuição desta dissertação será uma análise comparativa com objetivos académicos e empresariais, entre as várias soluções disponíveis, selecionadas por critérios de inclusão e exclusão, de modo a fornecermos uma avaliação válida e completa. Esta análise tem como objetivos contribuir para uma melhor escolha da tecnologia Cross-Platform para solucionar um problema proposto, expondo as suas limitações.

Para construir uma análise comparativa completa e válida, iremos utilizar um caso de estudo, construindo uma aplicação móvel em todas as tecnologias selecionadas. O objetivo será realizar uma análise individual de cada uma, testando e recolhendo métricas como consumo de recursos, custo,

esforço de desenvolvimento, limitações impostas pela arquitetura, desempenho, segurança, entre outros parâmetros posteriormente selecionados. Além disso, esta avaliação irá também incluir uma componente qualitativa, em que serão realizados questionários ao público alvo, de forma a aferir como estas ferramentas são utilizadas no dia a dia, por equipas que desenvolvem aplicações *mobile*. Com isto, esperamos recolher informação técnica e qualitativa sobre as tecnologias escolhidas.

Através desta análise, será possível futuramente facilitar a construção de um sistema de recomendação de tecnologias para desenvolvimento Cross-Platform.

De uma forma mais especifica a solução proposta pretende contribuir com os seguintes pontos:

CONT1 : Revisão de literatura sobre o desenvolvimento *mobile* Cross-Platform, avaliando e comparando as diferentes tecnologias.

CONT2 : Análise individual de várias tecnologias Cross-Platform, utilizando um estudo de caso e ferramentas *benchmark*.

CONT3 : Análise comparativa entre várias tecnologias Cross-Platform.

CONT4 : Elicitar uma série de recomendações para a seleção de uma tecnologia de desenvolvimento *mobile* para um problema proposto.

CONT5 : Propor um sistema futuro para, dado um problema, recomendar uma tecnologia Cross-Platform adequada.

1.3 Estrutura do Documento

A estrutura da dissertação será a seguinte:

O primeiro capítulo, a introdução, descreve uma iniciação ao assunto principal, de modo a possibilitar uma melhor compreensão do tema ao leitor. Além disso, são apresentadas descrições do problema, a solução proposta e os objetivos e contribuições desta dissertação. No segundo capítulo, a revisão de literatura, representa o estado da arte, indicando as tecnologias de desenvolvimento *mobile*, em conjunto com as diversas abordagens de desenvolvimento, sendo também, apresentada a motivação para as métricas e métodos para a avaliação das tecnologias escolhidas. No terceiro capítulo, a metodologia, expomos claramente os métodos seguidos para as avaliações e comparações, tornando todas as comparações válidas, contendo também a aplicação proposta para caso de estudo e a sua avaliação. No quarto capítulo, os resultados, de uma forma qualitativa e quantitativa apresentamos os resultados obtidos das avaliações, comparando o comportamento das tecnologias, com gráficos e tabelas. O quinto capítulo, a discussão, representa uma análise e reflexão sobre os vários temas expostos nos capítulos revisão de literatura, metodologia e resultados. Por fim, o último capítulo, a conclusão apresenta uma reflexão das contribuições inicialmente propostas, os resultados globais da tese e os trabalhos futuros propostos.

2 Revisão de Literatura

Este capítulo será utilizado para apresentar a base temática desta dissertação, deste modo, será apresentada uma contextualização das tecnologias de desenvolvimento *mobile* e as diversas abordagens de desenvolvimento Mobile. Apresentaremos também como surgiu a motivação para a construção de métricas, parâmetros e métodos utilizados na avaliação das tecnologias escolhidas.

2.1 Diversas Abordagens de Desenvolvimento *mobile*

A expansão do desenvolvimento de aplicações *mobile* evoluiu de forma a resolver as necessidades dos *developers*, solucionando problemas como separação de equipas por SO's, aprendizagem de várias linguagens, entre outros, levando assim, à criação de diversas abordagens de desenvolvimento de aplicações *mobile*, como podemos observar na Figura 1 [21]. Independentemente do tipo de desenvolvimento, é sempre necessário ter em atenção aos desafios e às boas práticas de desenvolvimento *mobile*, proporcionando uma aplicação final agradável ao utilizador e, uma facilidade na manutenção e adição de funcionalidades [5].

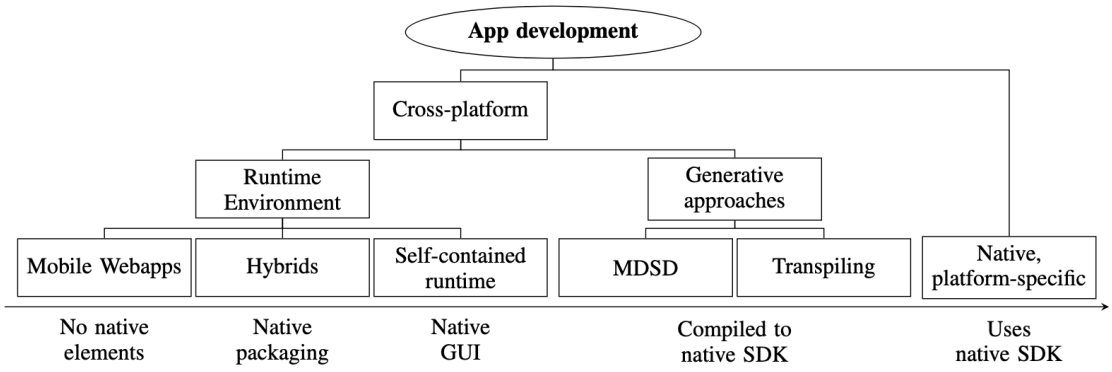

Figura 1. Esquema das possíveis abordagens de desenvolvimento mobile [76]

O desenvolvimento de aplicações *mobile* pode ser separado entre duas abordagens, as aplicações nativas que são desenvolvidas individualmente para cada SO, utilizando o SDK fornecido, e o desenvolvimento Cross-Platform, que utiliza uma linguagem base que pode ser executada em diferentes SO's [37].

O principal objectivo das tecnologias Cross-Platform's é possibilitar o desenvolvimento de aplicações a partir do mesmo código. Torna-se assim necessário ter em conta alguns requisitos, com o intuito de proporcionar uma tecnologia abrangente e com um bom desempenho [33][75].

Estes são os requisitos necessários de uma tecnologia Cross-Platform [108] [33] [75]:

- **Suporte a várias plataformas:** Uma tecnologia deve suportar múltiplas plataformas, em particular as mais utilizadas no mercado (iOS e Android).
- **Interface agradável:** O sucesso de uma aplicação depende em grande parte da experiência do utilizador logo, é necessário ter incorporado uma interface rica e com apoio a animações, multimédia e gráficos 2D e 3D.
- **Segurança:** É essencial as aplicações possuírem o máximo de segurança exequível.
- **Manutenção:** É crucial possibilitar a manutenção da aplicação, sem comprometer com funcionalidades anteriormente desenvolvidas.

– **Consumo de recursos:** As aplicações desenvolvidas devem de ser otimizadas de modo a consumir menos recursos possível.

– **Ambiente de desenvolvimento:** A tecnologia deve prover um bom ambiente de desenvolvimento, em que integre os compiladores necessários e simuladores dos diferentes SO's que suporta.

2.1.1 Desenvolvimento Nativo

As aplicações *mobile* nativas são concebidas para apenas serem executadas num SO específico, considerando apenas o tipo de dispositivo e a versão a ser utilizada. Deste modo, o *developer* tem uma seleção limitada de linguagens que diferem de SO e IDE, sendo assim necessário utilizar vários SDK's e IDE's para desenvolver a mesma aplicação para várias plataformas. Por exemplo, o Android suporta Java e Kotlin , iOS o Objective-C e Swift e, por fim, o C# para Windows Phone. Todos os exemplos anteriormente mencionados, geram código binário diferente, obtendo assim um executável, semelhante ao processo utilizado para as aplicações Desktop [37] [75].

Pelo motivo de este dispor de acesso direto as API's fornecidas pelo SO (como GPS, lista de contactos, bibliotecas HTTP, entre outras), como podemos visualizar na Figura 2, apresentará sempre um melhor desempenho quando comparado com as aplicações *mobile* desenvolvidas em tecnologias Cross-Platform [26]. No entanto, o desenvolvimento nativo torna difícil reutilizar o código, pois, cada plataforma utiliza uma linguagem diferente, dificultando manter as mesmas alterações em todas as plataformas [108].

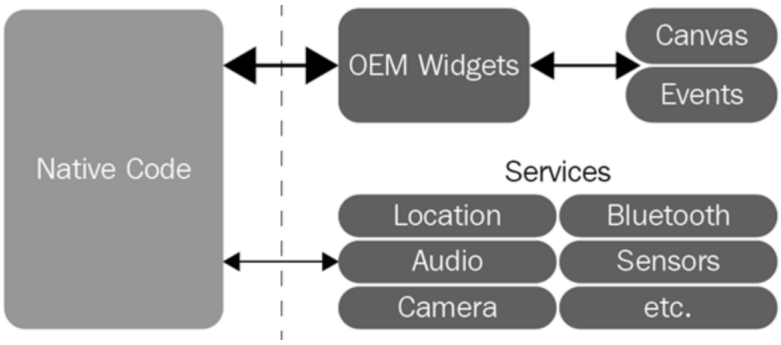

Figura 2. Arquitetura Native App

No momento de distribuir as aplicações, cada uma terá de ser transferida para as *app stores* de cada SO, sendo que, cada loja tem um processo para avaliar se a aplicação cumpre com os requisitos da plataforma [26].

Esta abordagem de desenvolvimento *mobile* é a que envolve mais custos, uma vez que é necessário utilizar diferentes tecnologias e linguagens, sendo necessário realizar processos de codificação, manutenção e testes em cada plataforma [37].

2.1.2 Desenvolvimento Cross-Platform

Atualmente, das diversas abordagens *mobile*, o desenvolvimento Cross-Platform é a mais utilizada, pelo motivo de nos possibilitar a construção de aplicações utilizando apenas uma linguagem, para várias plataformas [108].

Esta abordagem pode ser dividida em dois conceitos principais: abordagens em runtime e abordagens generativas, ambas cujo objetivo é construir uma aplicação *mobile* semelhante a uma aplicação nativa. A abordagem em runtime encontra-se dividida em três sob conceitos (WebApps, aplicações híbridas e *interpreted*), todas estas processam o código em runtime. Por outro lado, a abordagem generativa encontra-se dividida em dois sob conceitos (Cross-Compiled e MDSD), onde utilizam geradores para gerar o código em código nativo específico do SO [69][20].

2.1.2.1 Desenvolvimento WebApp (PWA)

O conceito de desenvolvimento PWA iniciou-se em 2015, pela Google, em que pretendia juntar uma aplicação web com as aplicações *mobile*, permitindo desempenhar funções em modo *offline* em conjunto com notificações. Este conceito consiste numa aplicação web responsiva e progressiva, no qual se adapta ao dispositivo que está a ser executado, em que é construída a partir de linguagens Web, JavaScript, HTML e CSS, como podemos visualizar na Figura 3 [20].

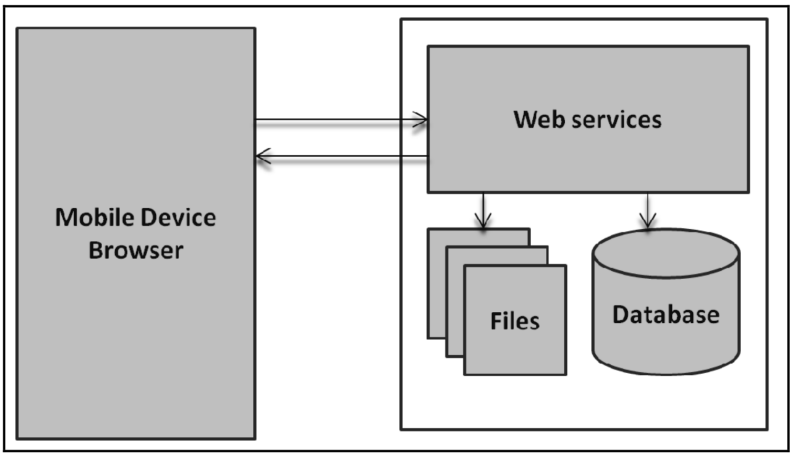

Figura 3. Arquitetura Web Application [93]

Esta abordagem distingue-se da abordagem nativa pela capacidade de ser executada em várias plataformas, independentemente do SO, pois, é processada pelo *browser*. Pelo facto de a aplicação ser executada no *browser*, não é possível instalar pelas *app stores*, só podem ser executada através de um URL, ou instalada pelo próprio *browser* [93] [37].

O desenvolvimento PWA contém um conjunto de características que a torna uma abordagem adequada para desenvolvimento web e *mobile* [115][48]:

1. O seu desenvolvimento progressivo, torna-o independente do *browser*.
2. É responsivo, adapta-se a qualquer dispositivo (dimensão de ecrã).
3. Tem funcionalidades *offline* através da utilização de *workers*.
4. Pode ser descarregado do *browser* e apresentado como um ícone no dispositivo.
5. Utiliza protocolos de comunicação seguros. (HTTPS).

A vantagem da abordagem PWA é que esta comporta-se de forma semelhante em todos os *browsers* e em todas as plataformas, possuindo interfaces idênticas [41]. Contudo, não permite o acesso às funcionalidades nativas do SO, como por exemplo, Gps, câmara, entre outros, limitando assim algumas funcionalidades que o *developer* pode vir a necessitar. Uma outra desvantagem

é, pelo facto de utilizar um *browser*, é necessário considerar que o tempo que renderização das páginas web poderá ser demorando e dispendioso, sendo necessário considerar vários padrões de implementação [75] [48].

2.1.2.2 Desenvolvimento Híbrido

Através da abordagem de desenvolvimento WebApp, permitiu-se criar um outro conceito, a abordagem de desenvolvimento híbrida, que possibilitou os *developers* de aplicações web, criar aplicações *mobile* utilizando apenas linguagens de desenvolvimento web em conjunto com uma API em JavaScript [21], onde esta API permite realizar chamadas às funcionalidades do SO e também utilizar os sensores disponíveis do dispositivo.

Com base no mesmo conceito de desenvolvimento WebApp, esta abordagem também é processada por um *browser*, mas, é encapsulado numa base nativa chamada de WebView. A base nativa será encarregue de renderizar esta componente, tornando assim possível aceder aos recursos do SO [107], como podemos observar na Figura 4.

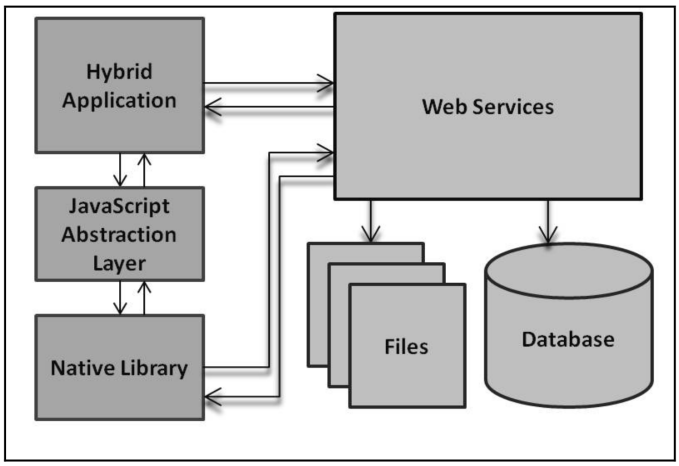

Figura 4. Arquitetura Hybrid [93]

A combinação do acesso às funcionalidades do SO, o comportamento nativo, e por último, a facilidade de implementação da interface à custa das linguagens conhecidas, tornam esta abordagem muito popular [37]. De forma oposta às aplicações PWA, esta abordagem permite distribuir as aplicações pelas *app stores*, obtendo características nativas. Contudo estas aplicações poderão ter um desempenho inferior, uma vez que os processos são realizados pelo *browser*, estando também sujeitas a vulnerabilidades na camada de abstração de JavaScript. Como estas aplicações utilizam apenas tecnologias Web, torna-se mais fácil realizar engenharia inversa para extrair informações armazenadas ou dados sensíveis e, como são processadas pelo *browser*, permite facilmente a um *hackers* recorrer a técnicas de ataques remotos [75].

2.1.2.3 Abordagem *Interpreted*

Em contraste com a abordagem de desenvolvimento híbrida, as aplicações produzidas com esta abordagem contêm um runtime autónomo, isto é, não utilizam um interpretador externo (como um *browser*), desta forma o código é interpretado posteriormente por um próprio interpretador. Assim, o código é interpretado por diferentes plataformas sem utilizar um *browser* [93], melhorando o seu desempenho e utilizando as API's através da camada de abstração (Figura 5).

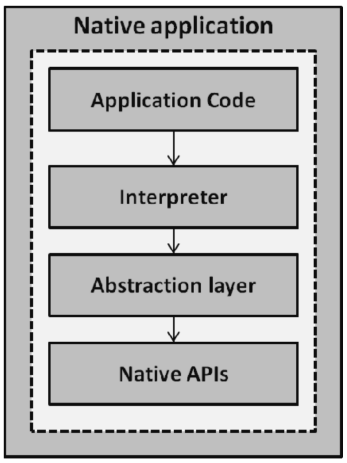

Figura 5. Arquitetura Interpreted [93]

Esta abordagem fornece um aspeto e sensação de uma aplicação nativa, sendo possível manter a mesma funcionalidade e estrutura de interface, mas poderá apresentar um mau desempenho devido à interpretação em runtime [93]. As aplicações implementadas através desta abordagem são construídas a partir de um único projeto, em que a maior parte é transposto para código nativo e apenas uma pequena parte é interpretada. Assim, a sua implementação é independente da plataforma e do SO, permitindo a utilização de componentes de interface nativos [37] [21].

2.1.2.4 Abordagem *Cross-Compiled*

A abordagem *Cross-Compiled* consiste em criar uma aplicação completamente nativa utilizando a mesma linguagem entre várias plataformas, através do compilador que converte o código utilizado na tecnologia para código binário nativo do SO. O principal componente desta abordagem é o compilador que é responsável pela produção de código que possa ser executado numa determinada plataforma, Figura 6.

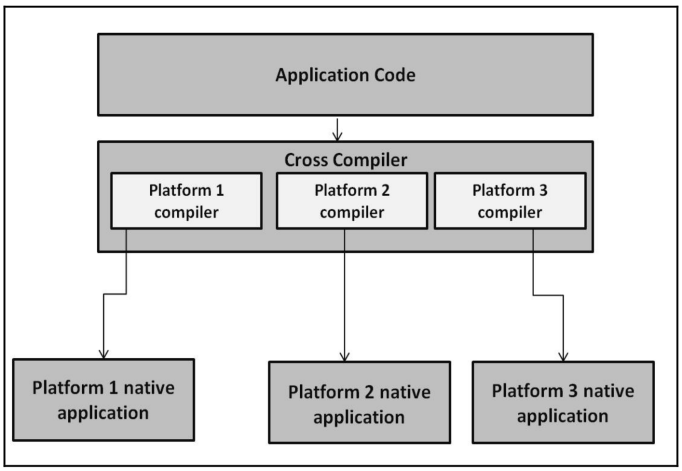

Figura 6. Arquitetura Cross Compiled [93]

Através do compilador, o código produzido torna-se nativo, desta forma todas as chamadas e funcionalidades do sistema tornam-se mais eficientes e fiáveis, como não depende de uma camada abstrata, toda a aplicação consegue ter um melhor desempenho [21].

A principal vantagem desta abordagem é o facto de as aplicações serem capazes de atingir um desempenho nativo, fornecendo todas as características das aplicações nativas, juntamente com os componentes nativos. Mas por outro lado, é extremamente difícil de identificar e corrigir erros na fase de compilação e, algumas funcionalidades terão de ser rescrevidas devido ao funcionamento diferente entre plataforma [93][42].

2.1.2.5 Abordagem *Model Driven Software Development*

Por fim, a abordagem *Model Driven Software Development* é a menos popular de todas, pois, tem uma complexidade muito superior. Esta abordagem originou-se na engenharia de software orientada por modelos, que consiste numa metodologia de desenvolvimento que se foca principalmente na criação e exploração de modelos. Modelos estes que possam representar problemas abstratos ou específicos [109]. Desta forma, o foco da abordagem orientada a modelos consiste em aumentar a produtividade, maximizando a compatibilidade entre sistemas.

Esta abordagem orientada por modelos existe há muitos anos, com o objetivo de gerir a variabilidade e centrando-se no modelo como representação abstrata de um sistema real. Como abordagem *mobile*, esta permite o desenvolvimento de aplicações Cross-Platform utilizando um nível de abstração elevado, através de utilização da linguagens textuais, DSL ou modelações como UML [21].

A grande vantagem desta abordagem, passa pelo facto de utilizar atividades de modelação, assim, o *developer* pode construir uma aplicação em alto nível, sem ter de lidar com técnicas de baixo nível, como por exemplo, a forma de guardar dados, notificações de sistema, entre outros [35]. Toda a interface da aplicação é construída somente por componentes nativos e, da mesma forma que a abordagem *Cross-Compiled*, todo o código produzido é compilado para o código binário nativo específico da plataforma, resultando assim, num bom desempenho e todas as funcionalidades do SO [75].

2.1.2.6 Sumário

Como vimos anteriormente, existem diferentes abordagens de desenvolvimento *mobile*, onde todas têm as suas características e limitações, sendo impostas pela sua arquitetura.

Todas estas contêm um diferente custo de implementação, sendo que, as abordagens mais populares são as híbridas devido à possível integração de bibliotecas e frameworks conhecidas, como Vue e Angular, diminuindo assim o custo de implementação. Uma equipa de desenvolvimento web tem assim uma maior facilidade de implementação utilizando esta abordagem. Contudo, é sempre necessário entender e conhecer o modo de funcionamento das API's fornecidas pela tecnologia escolhida.

2.2 Métodos e Métricas de Avaliação

Na secção anterior foram apresentadas várias abordagens para desenvolvimento Cross-Platform, descrevendo e comparando as suas principais características e apontando as suas principiais diferenças. Considerando estes vários tipos de abordagens descritas, é de maior relevância compreender como poderão ser comparadas de uma forma analítica e rigorosa. Nesta secção será apresentada uma investigação das abordagens utilizadas pela indústria e pela academia para analisar as aplicações *mobile* Cross-Platform, os casos de estudo realizados, e formas de efetuar *benchmarks* para obter todos os dados relevantes sobre o consumo de recurso e outros aspetos relevantes para a avaliação de uma tecnologia Cross-Platform.

2.2.1 Métodos para avaliação

Atualmente existem vários métodos para avaliar e analisar tecnologias de desenvolvimento de software, desde a execução de um conjunto de questionários, à construção de outras tecnologias e ferramentas para realizar testes e medições [40]. Para avaliar as tecnologias de desenvolvimento de aplicações *mobile*, são propostos vários métodos e procedimentos, sendo que alguns destes são adaptações ou combinações de outros métodos.

Um dos métodos mais famosos na construção de estudos e artigos, são os questionários ao público alvo, que permitem a colheita de informações, utilizando sondagens ou inquéritos. O questionário consiste num método de investigação, composto por um número de questões, com o objetivo de adquirir uma coleção de dados, necessários para formar componente qualitativa [70]. Num estudo realizado por Catolino et al. [25] efetuaram várias etapas para extrair métricas relevantes (características, interface, custo, funcionalidades, tamanho de aplicação) que possam ser encontradas na fase inicial de desenvolvimento de aplicações móveis, com o objetivo de comparar as aplicações móveis com as aplicações web. A primeira etapa consiste na recolha de métricas através da análise das citações online e a segunda um questionário a quatro *developers* experientes na área de *mobile*, de forma a validar o conjunto inicial de métricas obtidos na primeira etapa. No trabalho de Biørn-Hansen et al.[19], os autores elaboraram várias etapas para recolher dados de profissionais sobre conhecimento, utilização, questões relacionadas com o desenvolvimento Cross-Platform e os resultados da abordagem (Figura 7). Inicialmente criaram questões de investigação e de seguida um questionário de escolha múltipla, com um total de cinco perguntas de forma a recolher dados suficientes para responder às questões de investigação. Antes da escolha dos participantes, realizaram-se testes para assegurar a fiabilidade do questionário. Já, na dissertação de Santos [108], foi elaborado um questionário com questões de escolha múltipla, que se dividem em três grupos, informações pessoais, informação sobre a experiência do participante e informações acerca das tecnologias de desenvolvimento *mobile* Cross-Platform. O objetivo deste questionário foi obter dados relevantes para construir um sistema de recomendação baseado em conhecimento, através da escolha de 12 critérios relevantes aos participantes expostos na lista abaixo apresentada. Por fim, na investigação de Angulo et al.[8], depois de uma série de testes numa aplicação nativa e numa aplicação Cross-Platform, foi pedido aos participantes para preencherem uns questionários de comparação final, sendo estes o Escala de Usabilidade do Sistema (SUS) com o objetivo de recolher as ideias dos participantes sobre o grau de usabilidade do sistema, e o Questionário de Experiência do Utilizador (UEQ), focando nos únicos aspetos aplicáveis ao contexto de utilização, e um questionário final onde se compararam o aspecto e comportamento das tecnologias.

Lista de critérios obtidos pelos participantes, para a construção de um sistema de recomendação baseado em conhecimento elaborado na dissertação de Santos [108].

(i) Plataformas suportadas.
(ii) Linguagem de programação.
(iii) Tipo de aquisição (Grátis, Flexível ou Pago).
(iv) Licenças.
(v) Suporte a compras internas.
(vi) Facilidade na distribuição para as *app stores*.
(vii) Recursos suportados.
(viii) Geração de código nativo.
(ix) Tempo de adopção da versão mais recente das plataformas suportadas.
(x) Atualizações constantes da tecnologia.

(xi) Suporte a criação de interface.
(xii) Utilização de IDE de desenvolvimento.

Framework *(alphabetic order)*	Familiarity *(Q#1)* Totals	Familiarity *(Q#1)* Averages *(max 5)*	Interest *(Q#2)* Totals	Interest *(Q#2)* Averages *(max 5)*	Usage *(Q#3-4)* *N (Hobby)*	Usage *(Q#3-4)* *N (Prof.)*
Cordova *	-	-	-	-	2	0
Fuse by Fusetools	124	1.23	162	1.6	3	1
Intel XDK	132	1.31	147	1.46	2	0
Ionic Framework	289	2.86	281	2.78	37	17
jQuery Mobile	280	2.77	164	1.62	12	9
Meteor *	-	-	-	-	13	12
NativeScript by Telerik	159	1.57	199	1.97	5	1
PhoneGap	338	3.35	306	3.03	47	32
React Native by Facebook	302	2.99	432	4.28	46	26
Reapp	135	1.34	174	1.72	4	2
Sencha Touch	174	1.72	138	1.37	1	1
Tabris.js	112	1.11	145	1.44	1	0
Titanium by Appcelerator	173	1.71	149	1.48	3	3
Touchstone.js	123	1.22	164	1.62	2	1
Xamarin Forms	166	1.64	166	1.64	6	2
None		-	-	-	14	30

Figura 7. Resultados dos questionários na investigação de Biørn-Hansen et al.[19] realizados em várias etapas para recolher dados de profissionais sobre popularidade, utilização e questões relacionadas com o desenvolvimento Cross-Platform

A revisão sistemática é uma investigação cientifica que agrega vários estudos relevantes à investigação permitindo adquirir vários dados úteis para a construção de uma análise comparativa [18]. Através da revisão de literatura, vários investigadores conseguiram obter dados suficientes para construir listas de critérios, atribuindo pesos a cada item escolhido, avaliando assim uma abordagem ou tecnologia. Na investigação de Rieger et al.[102], construíram uma lista de critérios essenciais para caracterizar e avaliar as abordagens e tecnologias de desenvolvimento *mobile*, no qual os pesos de 1 a 5 foram atribuídos por investigadores académicos e experientes na área. A lista de critérios foi criada através de dados de vários artigos e separada em 4 categorias, infraestrutura, perspectiva de desenvolvimento, perspectiva de aplicação e perspectiva de utilização. Um estudo similar de Heitkötter et al.[59], utilizou uma lista de critérios semelhante, mas apenas separou a lista em 2 categorias, perspectiva de infraestrutura e perspectiva de desenvolvimento, avaliando cada critério através de conclusões de vários estudos. Xanthopoulos et al.[132], da mesma forma que os anteriores, propuseram um conjunto de critérios para avaliar e comparar as abordagens de desenvolvimento *mobile*, sendo os critérios distribuição no mercado, tecnologias generalizada, hardware e acesso a dados, interface do utilizador e percepção de desempenho pelo utilizador. Outras contribuições de Bernardes et al.[51], Amatya et al.[6] , Charkaoui et al.[27] e Ahmad et al.[4] analisaram e compararam as várias abordagens de desenvolvimento *mobile* Cross-Platform através das informações obtidas de vários artigos. As investigações de Bernardes et al.[51] e Charkaoui et al.[27] indicaram quais os pontos fracos e fortes de cada abordagem de desenvolvimento, e os estudos de Ahmad et al.[4] e Amatya et al.[6] assinalaram os desafios encontrados em cada uma das abordagens de desenvolvimento.

De um modo mais prático, o desempenho e o consumo de recursos de uma aplicação têm uma enorme importância na escolha de uma tecnologia, sendo que existem atualmente muitos estudos e artigos que analisam vários parâmetros do dispositivo ao executar uma determinada aplicação,

como podemos visualizar na Tabela 1. Dito isto, os métodos de teste são realizados de forma a obter dados significativos sobre o desempenho da aplicação e o consumo de recursos. O estudo desenvolvido por Dorfer et al.[39], utilizou dois cenários de teste controlados, o primeiro media o consumo de CPU e RAM, e o segundo o consumo de bateria, executando os dois cenários 10 vezes. Todos os testes foram criados utilizando a ferramenta UI *Automator*, que permite executar comandos de interface autonomamente, e seguindo um conjunto de passo definidos pelos autores, mostrados na Figura 8. Dalmasso et al.[33] realizaram testes de desempenho de forma a comprar a aplicação criada com o Titanium e outras com o PhoneGap, utilizando apenas o sistema Android. Representando uma aplicação empresarial, tinha vários botões na interface, cada um com um tipo de pedido web (AJAX, REST e SOAP) e analisava e exibia diferentes formatos de respostas (Texto, XML, JSON). A investigação de Biørn-Hansen et al.[20], realizou testes em seis dispositivos móveis, de modo a abranger várias gamas de dispositivos, efetuando cinco testes com focos diferentes, analisando os parâmetros, CPU, RAM em estado de inactividade e ocupação e tempo de conclusão de uma certa tarefa. Os testes foram escolhidos para serem executáveis na sua maioria isoladamente, evitando configurações complexas que dependem de fatores externos, tais como a qualidade da rede.

Steps	CPU and memory test	Battery test
Step 1	Start the measurement tool (Recording time: 3 min. 16 sec.).	Start the measurement tool (Recording time: 15 min. 16 sec.).
Step 2	Press the home screen button and wait 2 seconds.	Press the home screen button and wait 2 seconds.
Step 3	Start of the installed app by an intent.	Start of the installed app by an intent.
Step 4	Run the app for 3 minutes.	Run the app for 15 minutes.
Step 5	Press the home screen button and wait 2 seconds.	Press the home screen button and wait 2 seconds.
Step 6	Open recently opened apps and wait 2 seconds.	Open recently opened apps and wait 2 seconds.
Step 7	Close the app.	Close the app.

Figura 8. Lista de passos utilizados nos dois cenários de teste controlados no estudo de Dorfer et al.[39]

De uma forma diferente dos estudos anteriores, Delia et al.[37] elaborou testes em todas as abordagens de desenvolvimento *mobile*, e em ambos os SO's (Android e iOS). Os testes foram realizados em seis dispositivos, num total de 42 casos de teste, e estes testes consistiam num cálculo matemático incluindo várias iterações, avaliando assim a velocidade de processamento e o tempo de execução necessário para efetuar cálculos matemáticos intensivos. Para cada caso de teste, foram realizadas 30 execuções separadas, obtendo em cada caso uma amostra de tempo, e para caracterizar cada uma das amostras obtidas, foram calculadas estatísticas, como a média da amostra e o desvio padrão da amostra. De forma, a conseguir medir o consumo de energia de um telemóvel e a energia consumida por cada aplicação sem a utilização de ferramentas de software, Ciman et al.[29] decidiu utilizar dispositivos que tivessem acesso direto à bateria para usar um Monsoon PowerMonitor8 externo. A principal função do PowerMonitor8 seria medir a energia do telemóvel, controlando o nível exigido de energia ao realizar as funcionalidades das aplicações *mobile*.

Outros estudos e contribuições propostas por Willocx et al.[130], Willocx et al.[131], Rösler et al.[105], Jia et al.[65], Bekkhus et al.[17] e Corbalan et al.[32], focaram-se numa aplicação empresarial, sendo que os teste de desempenho consideraram apenas as funcionalidades da aplicação, sem seguir um processo específico ou algum padrão para recolher e analisar os dados obtidos pelos testes.

Um outro modo de avaliar o desempenho de uma tecnologia ou abordagem de desenvolvimento *mobile*, consiste na criação de um software ou uma tecnologia que realiza testes automáticos, semelhantes entre várias tecnologias. O estudo de Dhillon et al.[38], apresentou uma framework (Figura

Tabela 1. Parâmetros de desempenho avaliados pelos estudos

Estudo	Parâmetros
Bekkhus et al.[17]	CPU, RAM, Tempo médio de execução, Bateria
Biørn-Hansen et al.[20]	CPU, RAM, API's(Aceletometro, Contactos, Sistema de ficheiros, Localização)
Corbalan et al.[32]	CPU, RAM e Bateria
Ciman et al.[29]	Bateria
Dalmasso et al.[33]	CPU, RAM e Bateria
Dorfer et al.[39]	CPU, RAM e Bateria
Jia et al.[65]	Tempo de renderização, RAM, Tempo de resposta, Espaço no disco
Que et al.[92]	CPU, RAM, Bateria, Tempo de instalação, Tempo para iniciar, Fluxo de rede
Rawassizadeh et al.[94]	CPU, RAM, Espaço no disco, Bateria, Rede
Ryan et al.[104]	CPU, RAM e Bateria, Fluxo de rede, ECS, SOS, OMS, EMS, NEI, SSO, NI
Willocx et al.[130]	CPU, RAM, Tempo para iniciar, Espaço no disco, Tempo de pausa e resumo
Willocx et al.[131]	CPU, RAM, Espaço no disco, Tempo de resposta, Bateria

9), com o objetivo de realizar testes de desempenho para avaliar as tecnologias de desenvolvimento *mobile* Cross-Platform. A framework de testes é limitada a aplicações que não são graficamente intensivas, não permitindo aplicações com o foco em jogos ou processamento gráfico intensivo (aplicações de realidade aumentada, realidade virtual e realidade mista). A avaliação consiste em três fases, a primeira centra-se na determinação das capacidades da estrutura e caracteristicas das ferramentas (ambiente de desenvolvimento, acesso ao telemovel, sensores, segurança, entre outros), a segunda fornece um conjunto de parâmetros de referencia que devem ser implementados utilizando as tecnologias *mobile* Cross-Platform (*Benchmarks* de desempenho e usabilidade) e, por fim a terceira fornece os dados da ferramenta estudada, onde são discutidos itens de experiências de desenvolvimento a fim de compreender o resultado.

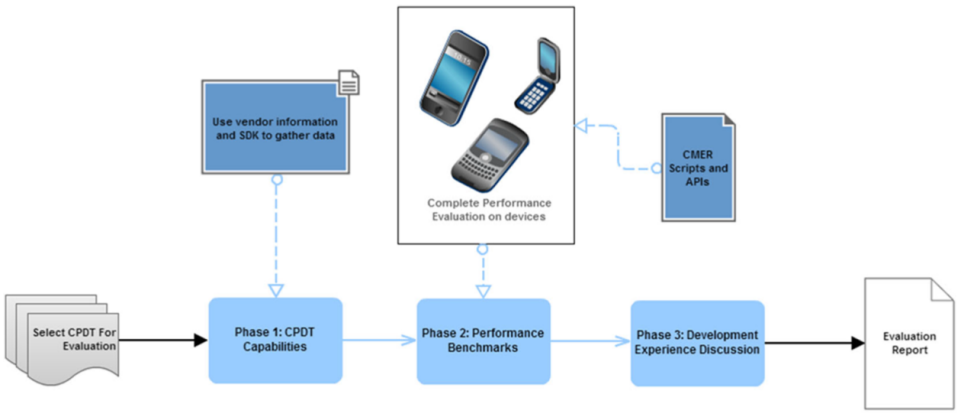

Figura 9. Framework apresentada por Dhillon et al.[38] para avaliar certos tipos de tecnologias de desenvolvimento mobile Cross-Platform

Os testes de desempenho têm uma grande influencia na avaliação de uma tecnologia. Muitos investigadores também dão importância aos testes de usabilidade e funcionalidade. O estudo realizado por Hussain et al.[61], reuniu 26 artigos para obter orientações sobre a construção de um modelo de avaliação de usabilidade. Para assegurar que o modelo construído pelos autores é fiável, realizaram experiências para testar a usabilidade das aplicações em dois telemóveis diferentes. Por outra lado, o trabalho de Angulo et al.[8] focou-se na avaliação do impacto da interface gráfica, entre uma aplicação nativa e outra criada pelo Titanium. Para decidir qual a abordagem que tem

uma melhor usabilidade, realizaram dois estudos com 37 participantes. O primeiro estudo era laboratorial, que avaliava a primeira impressão dos utilizadores, realizando cinco tarefas especificas que abrangiam todas as funcionalidades da aplicação. O segundo, um estudo longitudinal, onde os participantes utilizavam a aplicação durante cinco dias em circunstancias reais, utilizando os seus próprios telemóveis. Por ultimo, Sommer et al.[114] realizaram uma investigação, onde criaram cinco aplicações idênticas, três com tecnologias Cross-Platform e duas em tecnologias nativas, com o foco de conhecer a utilidade das soluções Cross-Platform para o *developer* e as vantagens para as pequenas e médias empresas que desenvolvem aplicações móveis. Utilizando o modelo FURPS+ (Funcionalidade, usabilidade, fiabilidade, desempenho, suporte ao desenvolvimento, implementação, suporte, custos) avaliaram cada aplicação dando um peso entre uma a cinco valores.

Category	*Weight*	*Titanium*	*Rhodes*	*PhoneGap and Sencha Touch*	*Android SDK*	*iOS SDK*
Functionality	4	3	2	3	**4**	**4**
Usability features	3	4	3	3	**5**	**5**
Developer support	2	**4**	3	**4**	**4**	**4**
Reliability & Performance	3	3	2	2	4	**5**
Deployment, Supportability, Costs	5	**4**	**4**	**4**	**4**	3
Rounded final score:		**3.6**	**2.9**	**3.2**	**4.2**	**4.1**

Figura 10. Aplicação do modelo FURPS+ para avaliar cada aplicação, realizado na investigação de Sommer et al.[114]

2.2.2 Provas de Conceito

Provas de conceito podem ser a base de toda a análise de uma comparação, sendo possível investigar ao pormenor, eliminando possíveis ambiguidades [52]. Esta abordagem tem uma grande importância pelo facto de ser possível visualizar as diferenças e as limitações entre tecnologias. Os casos de estudo são muito utilizados para a realização de testes no desenvolvimento de software, da mesma forma, são também utilizados no desenvolvimento *mobile*, principalmente para garantir que a tecnologia suporta todos os requisitos necessários para resolver o problema inicialmente proposto e se a linguagem e estrutura de código não dificultam a manutenção e adição de funcionalidades [103].

Um simples caso de estudo foi realizado por Xanthopoulos et al.[132] que construiu uma aplicação utilizando a tecnologia Titanium, com o objetivo que esta realizasse pedidos HTTP, mostrando os dados em uma lista, utilizando menos que 150 linhas de código JavaScript. Através deste caso de estudo, conseguiu demonstrar que apenas com uma linguagem e poucas linhas de codigo, era possível criar a mesma aplicação para dois sistemas diferentes (Android e iOS). Um caso de estudo semelhante foi realizado por Smutný[113], com o objetivo de comprovar que as linguagens nativas podiam ser substituídas por linguagens Web. Utilizando a tecnologia jQuery Mobile, construiu uma aplicação (tradutor automático de termos das línguas Inglês-Checo) utilizando código já existente da aplicação web (Figura 11), apenas ajustando o tamanho da janela e modificando os eventos do rato para para os eventos customizados (tap, swipe, orientação, etc...).

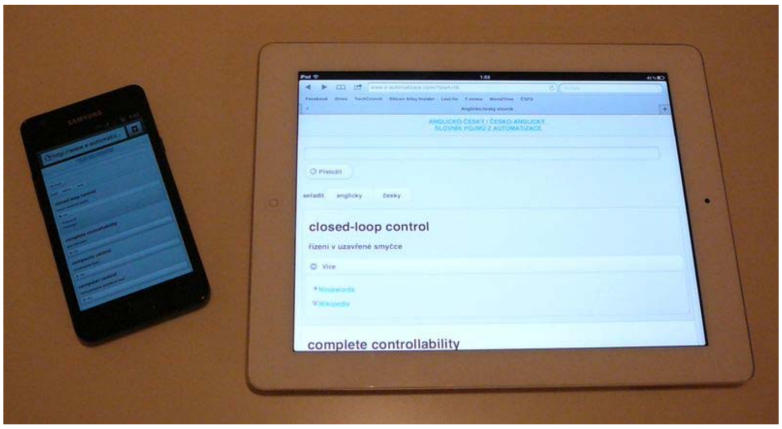

Figura 11. Caso de estudo realizado por Smutný[113] utilizando código já existente da aplicação web, transformando numa aplicação mobile

Uma das formas de comparar as diferenças entre as tecnologias de desenvolvimento de aplicações nativas e Cross-Platform consiste na construção de duas aplicações em tecnologias diferentes com o mesmo intuito. Angulo et al[8], realizou um estudo em que construiu dois protótipos, o prototipo preferido, seria depois utilizado pelos estudantes para estes terem acesso à informação universitária em tempo real, incluindo dados dos professores, entre outras informações. A primeira aplicação foi criada com tecnologias nativas e a segunda com a tecnologia Titanium. Um estudo semelhante conduzido por Dorfer et al.[39], desenvolveram a mesma aplicação duas vezes, uma delas utilizando a abordagem nativa do Android, e a outra utilizando a tecnologia React Native (Figura 12). Esta aplicação consiste na procura de restaurantes, utilizando o GPS para determinar a posição atual, a API do Google Places para encontrar os restaurantes próximos (2000 metros) e mostrando-os num mapa e numa lista, e ligação a Internet para realizar pedidos HTTP. O objetivo destas duas aplicações era obter resultados de qual a que consumia mais recursos. O artigo realizado por Que et al.[92], construiu um caso de estudo similar aos anteriores, utilizando a tecnologia Cordova e tecnologias nativas, com o foco de testar as funcionalidades do sistema, como a camara, GPS, entre outros, de forma a obter uma indicação de qual a melhor abordagem.

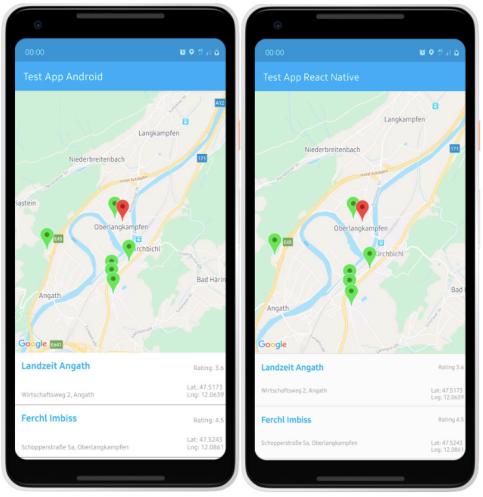

Figura 12. Caso de estudo construído por Dorfer et al.[39], com o objetivo de comparar uma aplicação nativa com uma Cross-Platform

Da mesma forma, como os estudos anteriores, os casos de estudo também são utilizados para realizar comparações entre diferentes tecnologias de desenvolvimento de aplicações *mobile* Cross-Platform. Biørn-Hansen et al.[20] desenvolveu um estudo sobre as aplicações WebApp, no qual construiu três aplicações como caso de estudo (Figura 13), a primeira aplicação construída com a tecnologia Ionic (hibrida), a segunda criada com o React Native (*Interpreted*) e por fim, a terceira com o React (PWA), com o objetivo de realizar comparações técnicas entre as tecnologias que utilizam linguagens web. Um outro estudo realizado por Sommer et al.[114], construiu como caso de estudo, uma aplicação comercial "MobiPrint"para apoiar os clientes de lojas de impressão fotográfica, criando a mesma aplicação em três tecnologias, Titanium, Rhoads e PhoneGap. Entre outros pontos, a aplicação testa, a integração com serviços web (HTTP), a localização (GPS), acesso ao sistema de ficheiros e por ultimo, funcionaliades de usabilidade. Em comparação aos artigos anteriormente descritos, existem outros mais abrangentes, como os estudos de Ciman et al.[29], Corbalan et al.[32], Delia et al.[36] e Willocx et al.[130], em que construíram um caso de estudo nas várias abordagens de desenvolvimento de aplicações Cross-Platform em conjunto com a abordagem nativa, seguindo um conjunto de requisitos impostos pelos autores. A aplicação de Ciman et al.[36], consiste numa versão *mobile* de um ambiente virtual de ensino (WebUNLP) que não estava adaptado para dispositivos móveis. Após a construção dos casos de estudo, os autores construíram uma análise com as limitações e problemas de cada abordagem ao encontro com os requisitos. Por outro lado, Corbalan et al.[32] construíram três aplicações diferentes com objectivos, processamento intensivo, reprodução de vídeo e reprodução de áudio, a fim de medir o consumo de energia causado por cada abordagem.

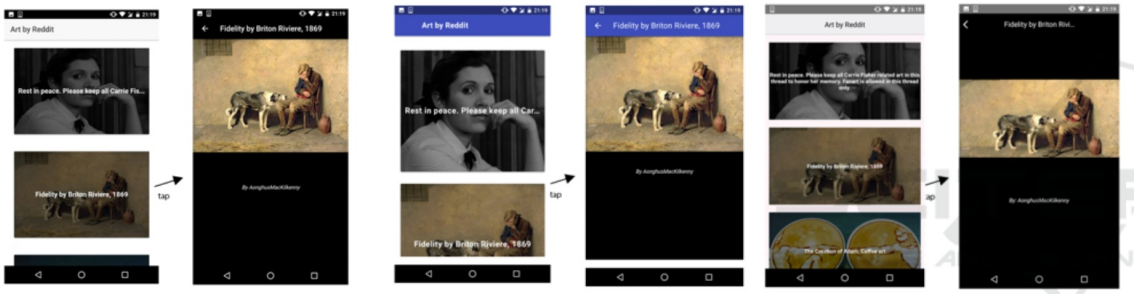

Figura 13. Caso de estudo desenvolvido por Biørn-Hansen et al.[20], com o objetivo de realizar comparações técnicas entre as tecnologias que utilizam linguagens web (Ionic, PWA e React Native)

2.2.3 *Benchmarks*

Sendo que os casos de estudo podem ser utilizados para analisar as limitações de uma tecnologia, muitas vezes é necessário avaliar as características de desempenho do software. Dito isto, o *Benchmarking* consiste numa avaliação de características de desempenho de um software em um hardware específico, como por exemplo, o consumo de recursos [112]. Nos dias de hoje, existem muitas ferramentas para avaliar o desempenho de um software, como a ferramenta ADB do Android que permite verificar a utilização do CPU, memória, entre outros [39]. Para algumas empresas, a escolha de uma tecnologia depende grande parte, dos resultados dos *benchmark*, geralmente quando necessitam de um software rápido e estável.

Sendo o SO Android um sistema Open-Source, o processo de *benchmark* torna-se mais controlado e detalhado em comparação com outros SO's, possuindo uma grande variedade de ferramentas

para analisar os sensores e componentes de hardware [32]. Através da ferramenta ADB (Android Debug Bridge) é possível verificar aspetos de desempenho de um software num específico hardware. O trabalho realizado por Biørn-Hansen et al.[20] utilizou esta ferramenta para monitorizar o tempo de lançamento da atividade [1]. Os estudos de Dorfer et al.[39], Dalmasso et al.[33], Willocx et al.[130], Willocx et al.[131] e Bekkhus et al.[17] utilizaram esta ferramenta para executar medições contínuas do CPU, memória e utilização de bateria, sendo que os resultados foram produzidos pelos casos de teste em intervalos de segundo, reinicializando a ferramenta antes de cada teste. Uma outra ferramenta fornecida pelo Android é o Android Profiler que apresenta dados mais descritivos, utilizada pelo artigos de Bekkhus et al.[17], Biørn-Hansen et al.[21], Willocx et al.[131] e Dalmasso et al.[33], que é semelhante à anterior, mas permite uma análise pormenorizada da actividade do CPU, memória, tempos de resposta e tráfego de rede, em conjunto com uma interface gráfica. Contudo, existem outras ferramentas de *benchmark* criadas por terceiros, utilizadas em artigos, como:

- Power Tutor, utilizado na investigação de Dalmasso et al.[33], para medir o consumo de energia de aplicações individuais.
- Google Devtools Devices [16], utilizado para aplicações PWA's e híbridas, permite analisar o consumo de recursos remotamente, em qualquer dispositivo móvel.
- Trepn Profiler da Qualcomm, utilizado no trabalho de Corbalan et al.[32], para medir e analisar o consumo de energia em cada um dos componentes de hardware.
- Little Eye4, utilizado no artigo de Willocx et al.[130], para monitorizar um conjunto de parâmetros relacionados com desempenho para qualquer processo em execução no dispositivo.
- Uma ferramenta de monitorização de recursos desenvolvida por Rawassizadeh et al.[94], que rastreia e regista o consumo de recursos de um determinado processo, sendo executado em paralelo como uma aplicação alvo.

Por outro lado, sendo o iOS um sistema mais restrito, torna processo de *benchmark* mais dificil em comparação ao SO Android, possuindo apenas algumas ferramentas para analisar o comportamento de hardware. Os estudos de Willocx et al.[130] e Willocx et al.[131] utilizam a ferramenta Activity Monitor (Monitor de atividade) fornecida pela Apple, para monitorizar alguns aspetos de desempenho em certos componentes de hardware, como o CPU e memória. Uma outra ferramenta fornecida pela Apple, é o Time Profiler do Xcode, que permite analisar o desempenho de uma aplicação, encontrar problemas de memória, tais como fugas e zombies [62].

Por fim, o SO do Windows Phone como também consiste num sistema Close-Source, possui também apenas algumas ferramentas de *benchmark*, como a Windows Phone Power Tools. Esta ferramenta possibilita a monitorização de algumas componentes de hardware, no qual é utilizada no estudo de Willocx et al.[131], para analisar a utilização de CPU e memória.

2.3 Sumário

Neste capítulo foi realizado uma revisão de literatura com o objetivo apresentar-vos uma contextualização sobre as diversas abordagens e tecnologias de desenvolvimento *mobile*, em conjunto com as suas principais diferenças e métodos utilizados por investigadores para as avaliar e analisar. De modo a concluir o capítulo, será apresentado um sumário sobre o que foi descrito, com o foco nas

[1] Uma atividade fornece a janela na qual a aplicação desenha a própria interface gráfica. Essa janela normalmente preenche o ecrã, mas pode ser menor do que o ecrã e flutuar sobre outras janelas. [56]

abordagens e tecnologias de desenvolvimento *mobile* Cross-Platform, juntamente com as métricas e métodos utilizados para as avaliar.

2.3.1 Desenvolvimento Nativo

O desenvolvimento nativo contêm diversas implementações, sendo que todas estas são individualizadas por SO's, nomeadamente o Android, iOS e Windows Phone, tendo como foco, a construção de uma aplicação apenas na plataforma escolhida. Para o desenvolvimento em iOS utiliza-se as linguagens Objective-C ou Swift, mas só é possível desenvolver aplicações iOS num computador Macintosh com o Mac OS, em conjunto com a plataforma Xcode. Já para o desenvolvimento nativo de Android, não existe restrições a nível tecnológico, sendo possível utilizar qualquer SO atual, proporcionando assim uma grande flexibilidade em termos de hardware necessários. Este desenvolvimento utiliza as linguagens Java ou Kotlin, através da plataforma Android Studio. Por fim, o desenvolvimento nativo de Windows Phone utiliza as linguagens C++ ou C# em conjunto com XAML através da plataforma Visual Studio, em que apenas é possível criar estas aplicações utilizando o SO Windows, devido à tecnologia .NET fornecida pela Microsoft. Todos estes fornecem SDK's que contêm as ferramentas necessárias para desenvolver, instalar, testar e executar aplicações nativas, no entanto, o desenvolvimento nativo de Android é o mais flexível, pois não é restrito ao SO e os seus emuladores têm todos os módulos de um telemóvel real.

2.3.2 Desenvolvimento Cross-Platform

As implementações nativas e as ferramentas disponibilizadas (SDK's e dispositivos virtuais) pelo Android Studio, Xcode e Visual Studio, vieram a permitir as diversas abordagens Cross-Platform. A abordagem Cross-Platform tem o principal objetivo de possibilitar o desenvolvimento de aplicações *mobile* para as diferentes plataformas a partir do mesmo código, e está dividida em dois conceitos principais, abordagens runtime e abordagens generativas. A abordagem runtime encontra-se subdividida em três conceitos (WebApps, híbridas e *Interpreted*), gerando e processando o código em runtime, e por outro lado, a abordagem generativa encontra-se subdividida em dois conceitos (*Cross-Compiled* e MDSD), onde utilizam compiladores para gerar o código em código binário nativo específico do SO.

A abordagem WebApps, é a única que não utiliza os SDK's ou ferramentas fornecidas pelos SO's. Apenas utiliza o *browser* para interpretar e processar toda a aplicação. Por este motivo, não consegue aceder as funções do sistema, sensores, nem ser publicada nas *app stores*. Uma das suas principais características é a utilização de *service workers*, que em comparação a uma aplicação Web, permite ser acedida em modo *offline* utilizando a memória cache, e permitindo também a utilização de notificações, através da *App Shell*.

De uma forma semelhante à anterior, a abordagem híbrida também utiliza um *browser* para processar e interpretar toda a aplicação, mas ao contrário da implementação WebApp, a aplicação é encapsulada numa WebView, concedendo o acesso às funções e sensores fornecidos pelo SOs, através de uma Bridge construída em JavaScript. Através dos SDK's, torna-se disponível grande parte todos os componentes nativos. Contudo com a utilização de uma camada abstracta construída em JavaScript e por ser processada por um *browser* a aplicação fica sujeita a vulnerabilidades.

Em contraste com as aplicações *mobile* híbridas, as aplicações produzidas pela abordagem *Interpreted* contêm um *runtime* autónomo, isto é, não utilizam um interpretador externo. Desta forma o código é interpretado posteriormente por um próprio interpretador. Uma parte do código

é interpretado por diferentes plataformas sem utilizar um *browser*, permitindo utilizar todos os componentes nativos e utilizando as API's através de uma camada de abstrata.

A abordagem *Cross-Compiled* consiste um criar uma aplicação completamente nativa através de um compilador que converte o código utilizado pela tecnologia para código binário nativo do SO. O principal componente desta abordagem é o compilador, que é responsável pela produção de código que possa ser executado numa determinada plataforma. Através do compilador, o código produzido torna-se nativo, desta forma todas as chamadas e funcionalidades do sistema tornam-se mais eficientes e fiáveis, como não depende de uma camada abstrata, toda a aplicação consegue ter um melhor desempenho.

Por fim, a abordagem *Model Driven Software Development* tem uma complexidade muito superior, com foco em aumentar a produtividade, maximizando a compatibilidade entre sistemas, utilizando um nível de abstracção elevado através de linguagens textuais, DSL ou modelações como UML. A sua grande vantagem, é o facto de possibilitar a construção de uma aplicação *mobile* sem conhecimentos de programação e sem ter de lidar com técnicas de baixo nível (estrutura de dados, entre outros).

2.3.3 Métodos de Avaliação

Considerando estas várias abordagens e implementações, torna-se necessário avalia-los e compara-los de forma analítica e rigorosa. Existem vários métodos para avaliar e analisar as tecnologias e abordagens de desenvolvimento *mobile*, nomeadamente, questionários, revisões sistemáticas, métodos para casos de testes de desempenho, e usabilidade e tecnologias de automatização de testes.

Os questionários ao público alvo, permitem uma colheita de grandes quantidades de dados, apenas com a utilização de sondagens e inquéritos, formando um componente qualitativa. Estes são realizados em várias etapas (normalmente informações pessoais, informação sobre a experiência do participante e informações acerca das tecnologias) para extrair métricas relevantes para a análise das tecnologias e abordagens de desenvolvimento *mobile* Cross-Platform. Estes questionários são usualmente criados pelos investigadores, porém existe alguns questionários predefinidos como o SUS (System Usability Scale), SUMI (Software Usability Measurement Inventory), o QUIS (Questionnaire for User Interaction Satisfaction) e o UEQ (User Experience Questionnaire).

A revisão sistemática faculta a aquisição de vários dados úteis para a construção de uma análise comparativa. Através da revisão de literatura, consegue-se obter dados suficientes para construir uma listas de critérios, atribuindo pesos a cada item escolhido, avaliando assim uma abordagem ou tecnologia.

Os métodos para casos de testes são realizados de forma a obter dados significativos sobre o desempenho da aplicação, o consumo de recursos e usabilidade. Um dos métodos consiste numa lista de passos separados em vários cenários de teste controlados, sendo executados um número indefinido de vezes e isolados, evitando configurações complexas que dependem de fatores externos, tais como a qualidade da rede, entre outros. Um outro método baseia-se nos testes das funcionalidades da aplicação sem realizar padrões específicos. Mas de um modo diferente, os testes podem consistir num cálculo matemático incluindo várias iterações, avaliando assim a velocidade de processamento e o tempo de execução necessário para efetuar cálculos intensivos. Por ultimo um modo de avaliar o desempenho de uma tecnologia ou abordagem de desenvolvimento *mobile*, consiste na criação de um software que realiza testes automáticos, semelhantes entre várias tecnologias.

2.3.4 Provas de Conceito

As provas de conceito possibilitam uma investigação ao pormenor, eliminando possíveis ambiguidades, visualizando as limitações das tecnologias, permitindo verificar se a tecnologia suporta todos os requisitos necessários e se a linguagem ou estrutura de código não dificultam a manutenção e adição de funcionalidades. Os casos de estudo podem ser simples, de modo a entender como a tecnologia funciona e quais os seus benefícios, ou podem ser um pouco mais complexos de forma a comparar quais as diferenças entre as tecnologias de desenvolvimento de aplicações nativas e Cross-Platform, construindo duas aplicações com o mesmo intuito. Da mesma forma, os casos de estudo também são utilizados para realizar comparações entre diferentes tecnologias de desenvolvimento de aplicações *mobile*, sendo necessário criar uma aplicação para tecnologia. Dito isto, os casos de estudo podem ser utilizados como uma aplicação a nível empresarial ou apenas para testes, porém para um estudo completo, deve ser criado um caso de estudo que consiga abordar as API's dos SO's e realize funções de CPU intensivo para se conseguir obter vários parâmetros sobre o consumo de recursos, desempenho e usabilidade.

2.3.5 *Benchmarks*

Muitas vezes é necessário avaliar as características de desempenho do software no hardware. Dito isto, o *Benchmarking* em conjunto com os metodos para realização de testes, permite avaliar as características de desempenho de um software em um específico hardware, como o consumo de recursos, desempenho, entre outras características.

Sendo o Android um sistema Open-Source, o processo de *benchmark* é mais controlado e detalhado em comparação com outros sistemas, possuindo uma grande variedade de ferramentas para analisar os sensores e componentes de hardware. A ferramenta ADB (Android Debug Bridge) possibilita verificar aspetos de desempenho de um software num específico hardware, como por exemplo CPU, memória, utilização de bateria, entre outros, em intervalos de tempo fornecido pelas ferramentas do CLI. Uma outra ferramenta do Android é o Android Profiler, que fornecido pelo Android Studio, apresenta dados de perfil, semelhante ao anterior, mas permite uma análise pormenorizada da actividade do CPU, memória, tempos de resposta e tráfego de rede, em conjunto com uma interface gráfica. Contudo, existem outras ferramentas de *benchmark* criadas por terceiros, como por exemplo, o Power Tutor, Trepn Profiler da Qualcomm e Little Eye4.

Por outro lado, sendo o iOS um sistema Close-Source, o processo de *benchmark* é mais restrito em comparação ao Android, possuindo apenas algumas ferramentas para analisar o comportamento de hardware. A ferramenta Activity Monitor (Monitor de atividade) fornecida pela Apple, permite monitorizar alguns aspetos de desempenho em certos componentes de hardware, como o CPU e memória. Uma outra ferramenta fornecida pela Apple, é o Time Profiler do Xcode, que analisa o desempenho de uma aplicação e encontra problemas de memória, tais como fugas e zombies.

Por ultimo, o Windows Phone como também consiste num sistema Close-Source, possui apenas algumas ferramentas de *benchmark*, como a Windows Phone Power Tools. Esta ferramenta possibilita a monitorização de algumas componentes de hardware, como utilização de CPU, memória, entre outros.

3 Metodologia

Este capítulo expõe os métodos seguidos para a avaliação e comparação entre as tecnologias escolhidas, em conjunto com uma revisão de literatura extensiva de forma a identificar as principais diferenças entres as tecnologias *mobile* nativas e Cross-Platform. Todos os parâmetros para a avaliação foram escolhidos de forma a tornar essa avaliação válida e mais precisa possível. De igual forma, todos estes métodos resultaram da avaliação às abordagens realizadas por investigadores, identificados e discutidos na secção 2.2 do capítulo 2.

3.1 Revisão de literatura extensiva

Após a identificação e discussão das abordagens de desenvolvimento *mobile* nativo e Cross-Platform nas secções 2.1 e 2.3 do capítulo 2, torna-se necessário especificar os tipos de tecnologias e ferramentas que são utilizados nas diversas abordagens de desenvolvimento e as diferentes implementações nativas e Cross-Platform, realizando assim a contribuição 1 (CONT1). Nesta secção serão apresentados vários tipos de tecnologias e ferramentas e, de seguida, os diversos tipos de implementações de tecnologias de desenvolvimento *mobile*, identificando as principais características e diferenças.

3.1.1 Tipos de Tecnologias e Ferramentas

Na secção 2.1 do capítulo 2 foram realçados os padrões e abordagens de desenvolvimento *mobile* nativo e Cross-Platform. Contudo, existem uma enorme variedade de estruturas e ferramentas que permitem a implementação desses padrões. De forma a tornar mais clara a distinção entre tecnologias, torna-se útil realizar uma distinção nítida entres os tipos de tecnologias e ferramentas, dando ênfase nas suas características e capacidades [58]. Deste modo, as tecnologias estão separadas em quatro entidades, bibliotecas, frameworks, produto e serviço, e por fim, plataformas.

3.1.1.1 Bibliotecas

Uma biblioteca consiste num conjunto de ferramentas de pequena dimensão que oferece funcionalidades muito específicas para o *developer*. Estas são normalmente utilizadas e incluídas em conjunto com outras ferramentas e bibliotecas (Figura 14), compondo assim a aplicação. Podemos observar estas características na biblioteca React, este oferece uma biblioteca em JavaScript que permite incluir outras bibliotecas, formando assim uma aplicação web completa, facilitando o desenvolvimento web [106] [58]. As bibliotecas são extremamente utilizadas em sistemas operativos, em que estas implementam grande parte dos serviços do sistema, permitindo ser partilhadas por vários processos e programas simultaneamente.

Figura 14. Esquema de utilização de uma biblioteca

3.1.1.2 Frameworks

As frameworks são compostas por conjuntos de bibliotecas; componentes de software e compiladores, ligados através de uma arquitetura de software, fornecendo um conjunto variado de ferramentas necessárias para construir uma aplicação (Figura 15). Normalmente, estas impõem mais restrições em comparação com as bibliotecas [41], tendo como principal objetivo resolver problemas recorrentes através de uma abordagem genérica. Assim possibilita ao *developer* focar-se apenas na resolução do problema, permitindo um desenvolvimento rápido e seguro. O React Native é uma das frameworks mais conhecidas. Esta framework é composta por várias bibliotecas, incluindo o próprio React, compiladores e ferramentas necessárias para a construção de aplicações.

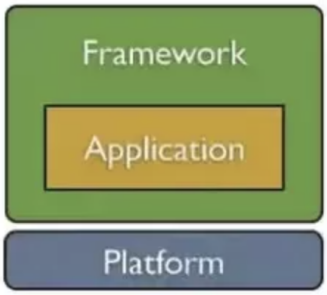

Figura 15. Esquema de utilização de uma framework

3.1.1.3 Serviço

Os serviços podem ser encontrados em todas as áreas de mercado, representando de uma forma geral, um serviço de uma empresa a ser vendido a um cliente, de forma periódica ou por venda única (Figura 16). Em termos de software, um serviço representa funcionalidades ou serviço específico pronto a ser utilizado e integrado num sistema ou aplicação. Estes são construídos através de uma combinação de bibliotecas, estruturas ou plataformas, em que num sistema ou aplicação, não são visíveis para o utilizador [79]. Um serviço pode ser encontrado, como por exemplo, numa aplicação que forneça dados periódicos sobre as ações de várias empresas, em que o utilizador paga periodicamente por esse serviço.

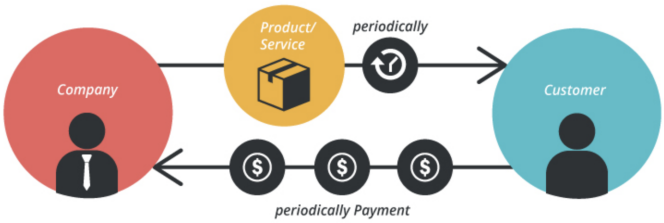

Figura 16. Esquema de geral do produto e serviço

3.1.1.4 Plataformas

As plataformas podem ser vistas como modelos de negócios, que funcionam por meio de tecnologias, sendo internas ou externas. Estas consistem num ambiente, que possui um conjunto de ferramentas, estruturas e serviços que não só permitem ao utilizador construir uma aplicação completa, como também configurá-la e, por exemplo, distribuí-la pela Cloud (Figura 17). Este tipo de tecnologia é altamente utilizada por grandes empresas, ajudando os utilizadores na recolha de informações necessárias, modificações ou inclusão de novos serviços. Assim permite uma gestão completa do sistema. Como exemplo, podemos considerar o Firebase da Google, que permite a criação e gestão de base de dados incluindo um tipo de desenvolvimento integrado para facilitar a construção de aplicações, com ferramentas de suporte e documentação [116].

Figura 17. Ilustração de uma plataforma

3.1.2 Implementação Nativa

Os quatro tipos de tecnologias e ferramentas anteriormente descritos foram utilizados nas implementações *mobile*, permitindo que fossem criadas as tecnologias de desenvolvimento que atualmente conhecemos. Através das bibliotecas, as grandes empresas de SO's criaram os SDK's. Estes SDK's permitiram um maior aproveitamento dos dispositivos móveis sem recorrer a bibliotecas externas, utilizando o mesmo ambiente de desenvolvimento.

Dito isto, o mercado de dispositivos móveis é dominado atualmente por três diferentes plataformas tecnológicas, que incluem diferentes SO's e plataformas de desenvolvimento. O desenvolvimento nativo para estas plataformas é distinto, com requisitos únicos a cada plataforma, como mostra a Figura 18 [1]. A característica mais importante das aplicações nativas é a possibilidade de interagir com todas as capacidades oferecidas pelo dispositivo, sem depender de outros *plugin-in's* ou camadas abstratas.

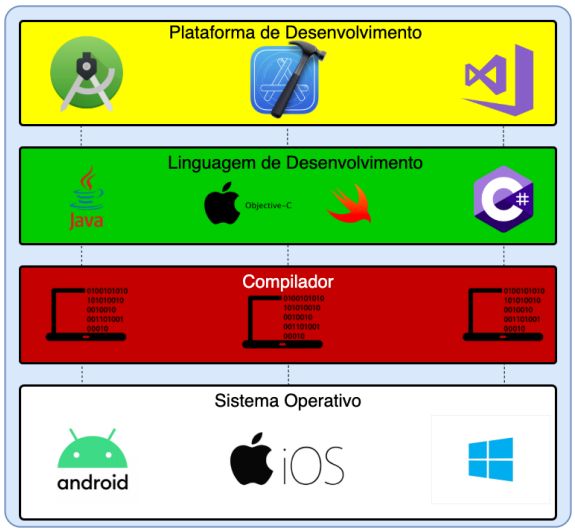

Figura 18. Esquema de desenvolvimento nativo

Neste subcapítulo só serão revistos os sistemas iOS, Android e Windows Phone, pela sua grande popularidade, bem como, a falta de suporte e manutenção dos outros SO's, como por exemplo, Blackberry OS e Symbian OS.

iOS

O sistema iOS é um SO para plataformas móveis desenvolvido pela Apple Inc. e distribuído exclusivamente para hardware Apple. Este sistema possui um código Closed-source e é utilizado no iPhone, iPad, Apple TV e Apple Watch. A plataforma alvo é derivada do MacOS X, assim patilha os mesmos princípios e fundações básicas [1] [23].

A arquitetura do iOS é dividida em camadas, no qual cada uma tem a sua função, como mostra a Figura 19, encapsulando assim cada camada e, permitindo que as camadas interajam apenas com as mais próximas. Este sistema utiliza como base o *Kernel* UNIX, de forma semelhante ao MacOS, utilizado em sistemas desktop [69].

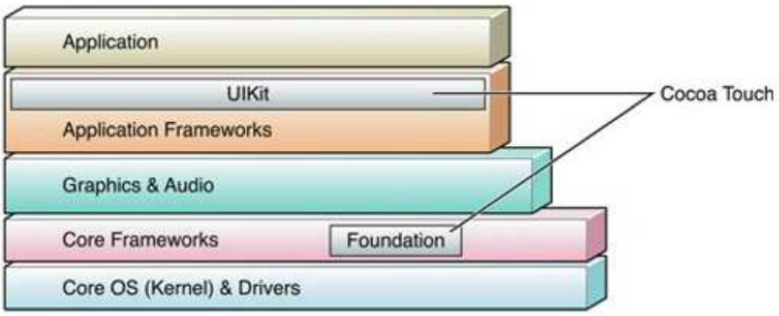

Figura 19. Arquitetura do SO iOS

O desenvolvimento nativo de iOS, só é possível num computador Macintosh com o MacOS, em conjunto com a plataforma Xcode, onde utiliza um SDK que contém ferramentas e interfaces necessárias para desenvolver, instalar, testar e executar aplicações nativas[12], oferecendo escolhas de especificações entre telemóveis, tablets ou smartwatchs [60]. Esta plataforma utiliza um IDE capaz de compilar código e realizar *debug*, sendo também capaz de expor os consumos de recursos, entre outras propriedades [53].

As aplicações nativas são construídas utilizando as estruturas fornecidas pelo sistema e as linguagens Objective-C ou Swift, em conjunto com a opção de utilização de simuladores oferecidos pelo SDK. Estas aplicações comunicam com o hardware através de um conjunto de interfaces que protegem a aplicação das alterações de hardware [60].

Android

O SO Android é um sistema desenhado para suportar todos os dispositivos móveis, independentemente do hardware escolhido. Este é desenvolvido pela Open Handset Alliance, liderada pela Google, que divulgou a distribuição Android em Novembro de 2007 [1].

Grande parte do sistema Android é Open-Source, sob a licença Apache, mas todas as aplicações da Google, como por exemplo, Play Store, Google Music, entre outros, são licenciados e Closed-Source [23]. O Android utiliza API's de alto nível escritas em C e as aplicações são executadas através de uma máquina virtual DVM, oferecendo características seguras, como a gestão eficiente de memória partilhada, identificadores de utilizador e processos Unix e, permissões de ficheiro [1].

Como está representado na Figura 20, o Android utiliza um *Kernel* Linux, que constitui a primeira camada do sistema, funcionando como uma abstração entre o hardware e as outras camadas. Da mesma forma que o sistema iOS as camadas só comunicam com as mais próximas [69].

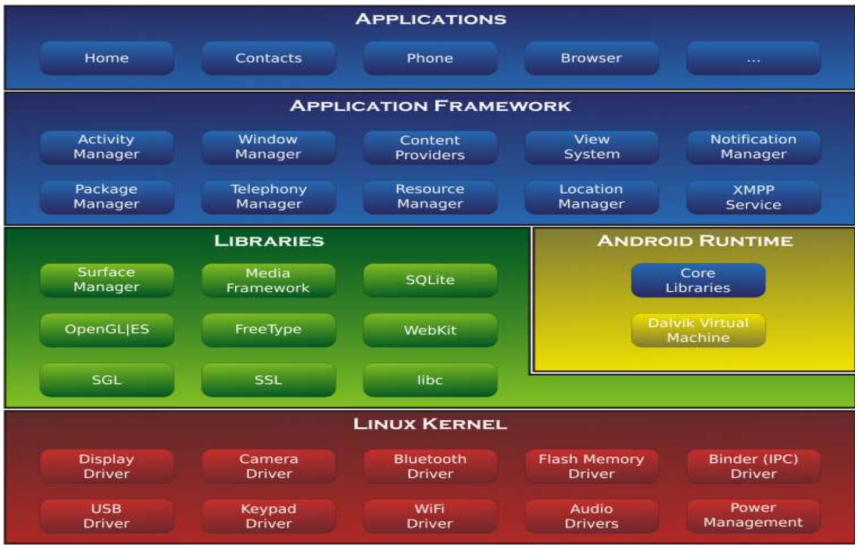

Figura 20. Arquitetura do SO Android

Contrariamente ao iOS, que o seu desenvolvimento é restrito ao MacOS, o desenvolvimento Android por ser realizado utilizando qualquer SO atual, Windows, MacOS, e sistemas Linux, proporcionando assim uma grande flexibilidade em termos de hardware necessários [69].

O desenvolvimento nativo Android, utiliza o Android Studio como plataforma e IDE, fornecendo dispositivos virtuais (AVD), que podem ser configurados de modo a representar qualquer dispositivo físico, diferentes resoluções, recursos SDK's , permitindo assim, realizar testes nas funcionalidades sem a utilização de um dispositivo real. A linguagem suportada pelo Android Studio é Java e Kotlin, permitindo incluir vários *Plug-in's* [7].

Windows Phone

O Windows Phone consiste num SO proprietário da Microsoft, sucessor do Windows Mobile, lançado em 2010 sob o nome de Windows Phone 7. Embora incompatível com a plataforma anterior, vários fabricantes de hardware, como por exemplo, Samsung, LG, Nokia, entre outros, desenvolveram dispositivos com este sistema. A Nokia foi o primeiro fabricante de dispositivos a utilizar este sistema como SO principal. A falta de funcionalidades deste SO, tornou-se-se rapidamente clara e a Microsoft sentiu necessidade de criar uma segunda geração, o Windows Phone 8 [1].

O Windows Phone possui, tal como o Android e iOS, uma arquitetura por camadas, apresentada na Figura 21. Começando pela camada de hardware e, de seguida, a camada do *Kernel* (baseado no *Kernel* do Windows CE) [69]. Esta arquitetura ao contrário das anteriores, exigia uma camada de hardware que cumprisse com os requisitos mínimos estabelecidos pela Microsot: CPU ARM7, GPU Directx, RAM de 256MB e 8GB de memória, ecrã *multi-touch*, câmara de 5 Mpx, sensores comuns e seis botões físicos [57].

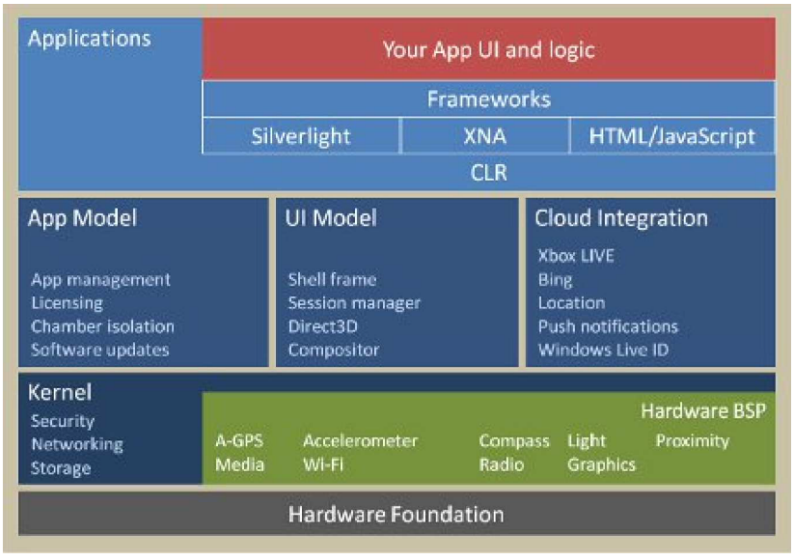

Figura 21. Arquitetura do SO Windows Phone

O desenvolvimento nativo de Window Phone, introduziu a capacidade de criar uma aplicação com C++ e/ou C# dentro de uma aplicação XAML, através da utilização da plataforma e IDE Visual Studio. As aplicações desenvolvidas nesta plataforma utilizam uma API em conjunto com um SDK .NET que contém todo o ambiente de desenvolvimento necessário [60] [82].

Esta plataforma permite o desenvolvimento apenas no SO Windows e, da mesma forma que as anteriores, promove um simulador com características necessárias para realização de testes e *debug* [77].

Comparação

Cada SO contém a sua plataforma e ambiente de desenvolvimento, cada uma permite otimizar o desempenho e consumo de recursos das aplicações nativas.

A Windows Phone perdeu valor no mercado devido ao crescimento do Android, levando assim ao reduzido suporte do sistema [69]. Em comparação com o iOS e o Android, a escalabilidade é muito menor e a percentagem de utilizadores também, consequentemente, existe pouca utilização desta plataforma [57].

O desenvolvimento nativo de iOS, em comparação com o Android, contém muitas restrições, isto por ser necessário um computador Apple para utilizar o Xcode, por outro lado, o Android Studio consegue ser executado em qualquer computador [57]. Relativamente aos dispositivos virtuais, o fornecido pela Xcode é considerado mais restrito comparado com o AVD do Android Studio, isto negligenciar módulos essenciais, como por exemplo, acesso à câmara, galeria, sensores e acesso encapsulado do Gps e Rede [11].

Por fim, em termos de licenças, o desenvolvimento nativo de Android e Windows Phone são completamente gratuitos. Fornecem todas as tecnologias necessárias para o desenvolvimento completo de uma aplicação, no entanto, o desenvolvimento nativo de iOS requer uma licença de *developer* [9] caso queiramos executar a aplicação num dispositivo móvel real ou até publicar a aplicação na loja AppStore da Apple.

Tabela 2. Características das tecnologias de desenvolvimento nativo

	Linguagem	**Licenças**	**IDE**	**Tipo de Tecnologia**	**Suporte**
iOS	Objective-C e/ou Swift	Flexível*	Xcode	Plataforma	Apple
Android	Java e/ou Kotlin	Grátis	Android Studio	Plataforma	Google
Windows Phone	C# e/ou C++	Gratis	Visual Studio	Plataforma	Microsoft

*Opções pagas e gratuitas

3.1.3 Implementação Cross-Platform

Através das implementações nativas e das ferramentas disponibilizadas (SDK's e dispositivos virtuais) pelo Android Studio, Xcode e Visual Studio, permitiram diversas outras implementações Cross-Platform, na qual têm como foco principal, o funcionamento da mesma aplicação em várias plataformas e dispositivos sem realizar alterações no código.

De forma a tornar mais claro quais as diferenças em termos de implementação entre as várias tecnologias Cross-Platform, nos seguintes subcapítulos será descrito como consiste a implementação Cross-Platform e como é que estas tecnologias utilizam as ferramentas disponibilizadas para a criação de aplicações, em conjunto com as suas arquiteturas.

3.1.3.1 Implementação WebApp (PWA)

Ao contrário de todas as outras implementações Cross-Platform, este é o único tipo de implementação que não utiliza qualquer tipo de SDK's ou ferramentas fornecidas pelos SO's, na qual apenas utiliza um *browser* para interpretar e processar toda a aplicação, como podemos observar na Figura 22. Por este motivo, não consegue aceder às funções do sistema, sensores, nem ser publicada numa *app store* [76][48].

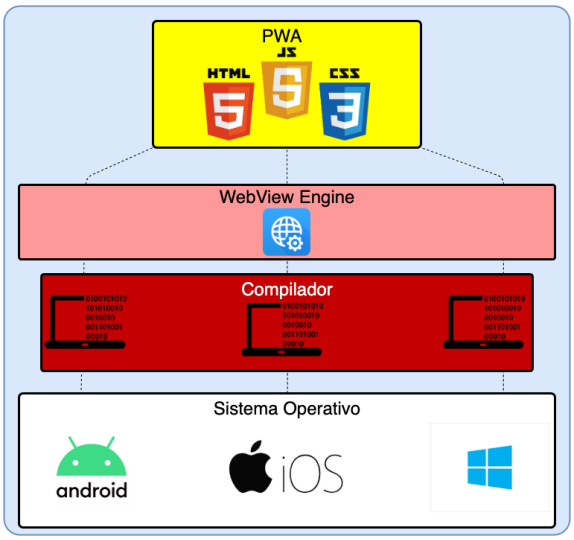

Figura 22. Esquema de desenvolvimento PWA

Pelo motivo de ser processado pelo *browser* é necessário considerar vários padrões de implementação, como por exemplo, o PRPRL (Push – Render – Pre-cache – Lazy-load) que permiti tornar a aplicação mais dinâmica e interactiva. Uma outra consideração a ser tomada é a falta de ligação à internet ou falhas de ligação, isto porque, compromete a interação e funcionamento da aplicação [76][115].

Uma das características do PWA é a utilização de *service workers*, em comparação a uma aplicação Web, permite ser acedida em modo offline, utilizando a memória cache e possibilitando, também, a utilização de notificações, através da *App Shell* [20]. Através da *App Shell*, a aplicação PWA pode ser instalada pelo *browser*, sendo que, o *web app manifest* informa através de um ficheiro JSON, quais as informações básicas da aplicação (nome, autor, icon, descrição, entre outros) proporcionando assim aos utilizadores um acesso mais rápido e uma experiência mais rica [115].

Neste subcapítulo serão descritos sucintamente as tecnologias, Angular, Vue, React e jQuery Mobile, sendo que, são consideradas as tecnologias Web e PWA mais populares do mercado e com manutenção periódica. De seguida será apresentada uma comparação entre estas tecnologias

Angular

Angular é uma framework de desenvolvimento web construída com módulos e bibliotecas de Node.JS, que posteriormente, foi comprada pela Google, sendo a primeira tecnologia a suportar o conceito de PWA. Esta tecnologia foi inicialmente criada para facilitar o desenvolvimento web e garantir que as aplicações criadas sejam responsivas e progressivas [48]. A primeira versão do Angular utilizava as linguagens HTML, CSS e JavaScript, porém, após a aquisição por parte da Google, originou-se uma segunda versão, substituindo o JavaScript pelo TypeScript, permitindo assim utilizar conceitos de POO, tornando o código mais flexível e com a possível adição de padrões de desenho [55].

A Angular 2 apresenta uma arquitetura com base em MVC derivada por módulos, componentes, metadata e templates como podemos visualizar na Figura 23, que permite adicionar bibliotecas, proxys, e *third-party apps* [54].

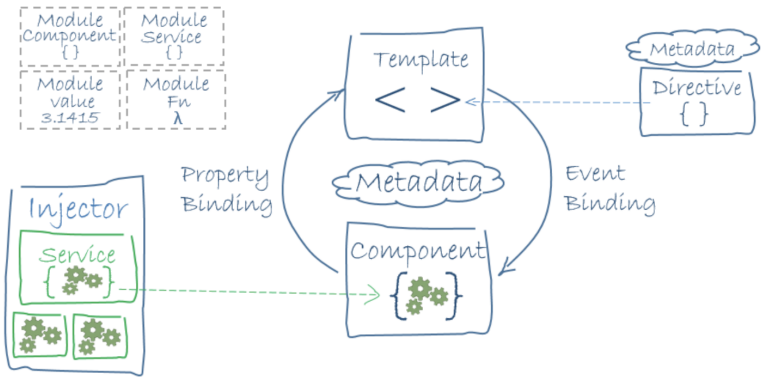

Figura 23. Arquitetura abstrata do Angular 2

Sendo esta framework Open-Source, possibilitou a manutenção e desenvolvimento de código em conjunto com a comunidade. Atualmente existe um ramo do repositório que tem o foco do *mobile* chamado de Mobile Angular UI [24].

Vue

O Vue consiste numa tecnologia de desenvolvimento Web que utiliza como linguagens HTML, CSS e JavaScript, criada com o intuito de organizar e simplificar o desenvolvimento, centrando-se na renderização declarativa e na composição de componentes [48]. Esta tecnologia foi criada pelo Evan You utilizando apenas o Node.js, mas com o crescimento exponencial da popularidade, o código fonte passou a ser Open-Source facilitando assim a manutenção e a criação de novas versões [126].

Atualmente, o Vue encontra-se na terceira versão e, em conjunto com a comunidade, conseguiu-se integrar novas funcionalidades e ferramentas [129], como a escolha entre TypeScript ou JavaScript, a renderização do lado do servidor, adição do VueX (biblioteca para a gestão do estados e partilha entre componentes) [125], e por fim, a adição de bibliotecas para a realização de testes, sendo estes testes unitários, testes nas componentes e teste *End-To-End* para verificar as ligações e comunicações entre componentes [128].

A estrutura de uma aplicação é feita em componentes, possibilitando assim ter filhos e decidir se estes são criados, modificados ou eliminados. Toda a interação é feita por eventos, modificando os dados dos componentes, no qual o Vue utiliza um DOM Virtual e um Watcher para renderizar e verificar essas mudanças [127], como exemplifica a Figura 24.

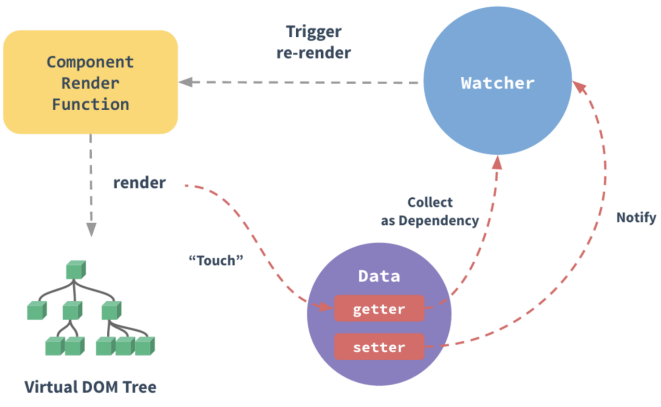

Figura 24. Esquema de renderização do Vue

Esta tecnologia pode ser vista como uma biblioteca permitindo ser incluída por um link de CDN, assim podendo ser incluída em qualquer outra tecnologia web ou pode ser considerada como uma framework, instalada através da ferramenta do Node.js (NPM), permitindo utilizar o CLI fornecido pelo Vue, uma interface de gestão de projetos Vue e, facilita a integração de outros módulos e bibliotecas no projeto [126].

React

De todas as tecnologias de desenvolvimento web, a mais conhecida é o React, tendo um grande crescimento no número de *developers* a escolher esta tecnologia [48]. O React foi criado pela Facebook, mantendo-o Open-source, criou-se uma manutenção comunitária em conjunto com as equipas do Facebook [100].

Esta tecnologia consiste numa biblioteca de módulos Node.js, que permite juntar HTML, CSS e JavaScript numa linguagem chamada de React que tem como objetivo principal facilitar o controlo de dados e estado da aplicação [101]. Esta linguagem, React é declarativa e flexível, possibilitando a integração de outras bibliotecas de JavaScript sem criar conflitos [99].

O React possui uma estrutura semelhante ao Vue, é também composto por componentes, renderizados por um DOM Virtual. Uma das grandes vantagens da utilização desta tecnologia é o facto da integração do Redux e middlewares, que facilita a gestão e comunicação de dados entre componentes e servidores, como observamos na Figura 25 [95].

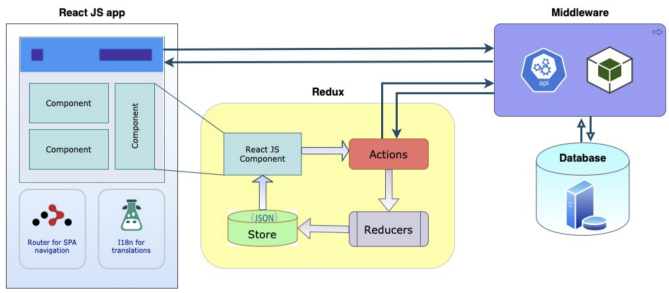

Figura 25. Esquema de uma aplicação React

Através da interação da comunidade, criou-se vários ramos no repositório principal que gerou novas tecnologias, como por exemplo, o React Native que será descrito nos próximos capítulos.

jQuery Mobile

A tecnologia mais antiga, das descritas neste capítulo, de desenvolvimento web é o jQuery, lançada em 26 de agosto de 2006, com o foco de simplificar o desenvolvimento web [111]. Esta tecnologia Open-source durante muitos anos obteve muita popularidade, isto porque, simplificava em larga escala o modo de utilização do JavaScript na manipulação com o DOM e, no uso de animações [30]. Esta biblioteca facilita as comunicações entre servidores através do uso do Ajax, sendo estas comunicações síncronas ou assíncronas [110].

Com o passar dos anos esta tecnologia caiu em desuso, isto devido ao surgimento e desenvolvimento de tecnologias mais eficientes. Contudo uma versão de *mobile* foi criada pela comunidade e os *developers*, o jQuery Mobile que utiliza as mesmas linguagens que o seu predecessor (HTML, CSS e JavaScript) [87]. A framework jQuery Mobile pode ser vista como uma continuação da tecnologia principal, mas com o foco de desenvolvimento *mobile* utilizando os conhecimentos e técnicas da biblioteca jQuery [88], como podemos observar na Figura 26.

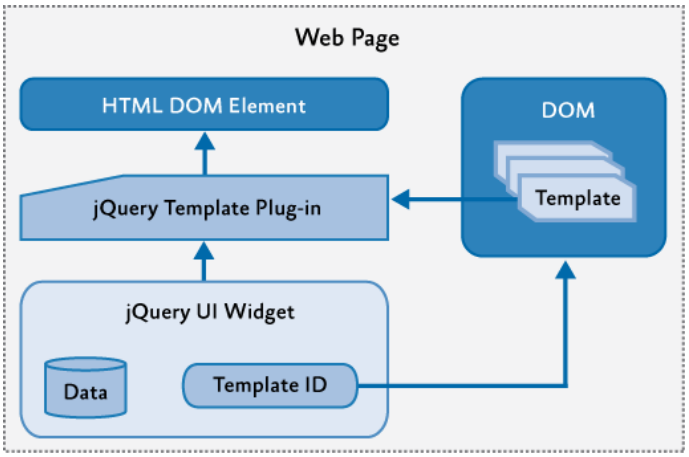

Figura 26. Esquema de renderização do jQuery Mobile

Sendo esta framework a mais antiga de desenvolvimento WebApp, suporta todas as plataformas e diapositivos, desde os mais antigos (BlackBerry OS) aos mais recentes (iOS), permitindo incluir várias bibliotecas através de links CDN[110].

Comparação

Independentemente da tecnologia escolhida é necessário para o desenvolvimento WebApp possuir bons conhecimentos nas linguagens HTML, CSS e JavaScript, onde estas tecnologias apenas virão a simplificar tarefas complexas ou aumentar a reutilização código entre aplicações (Tabela 3). Todas estas tecnologias comportam-se de forma diferente e têm os seus benefícios e restrições, sendo necessário avaliar primeiro quais as necessidades do projeto.

Em termos de popularidade, o React obteve grande crescimento (Figura 27) em comparação ao angular, vue e jQuery Mobile, devido à manutenção constante do Facebook e à grande variedade de bibliotecas criadas pela comunidade e próprio Facebook. Pelos dados fornecidos pela Google Trends (Figura 28), é possível verificar que o jQuery Mobile não tem muita procura em comparação com as restantes tecnologias anteriormente descritas.

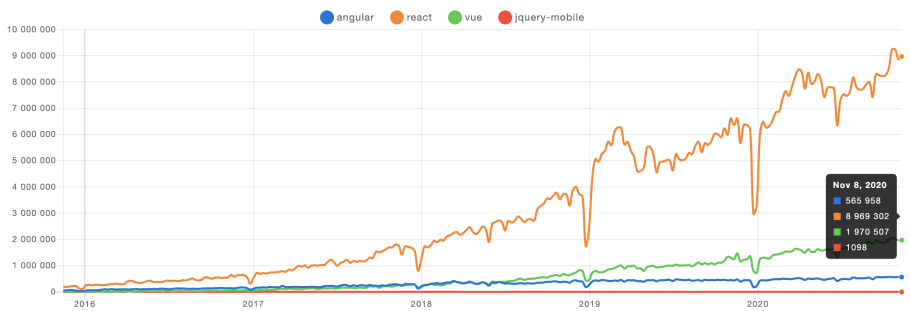

Figura 27. Número de downloads realizados na ferramenta do Node.js (NPM) entre 2016 e final de 2020

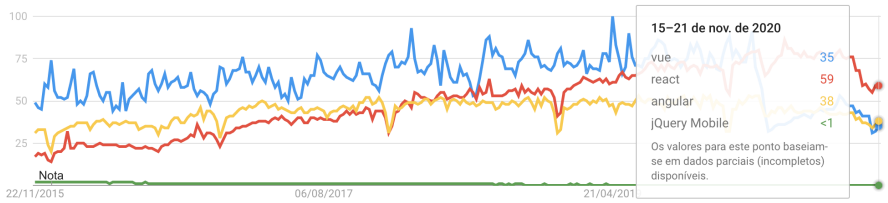

Figura 28. Número de pesquisas realizadas no Google entre 2015 e final de 2020, utilizando as palavras chave (vue/angular/react/jQuery Mobile)

Em termos de performance todas estas são processadas pelo *browser*, logo grande parte das limitações de performance são impostas pelo próprio *browser*. O Angular, por utilizar uma arquitetura por MVC, permite dividir tarefas em pedaços lógicos, reduzindo o tempo de renderização. Por outro lado, ambas as tecnologias React e Vue utilizam um DOM virtual para melhorar o desempenho e renderizar apenas o que é necessário, deste modo, torna menor o esforço de otimização. Ao contrário destas tecnologias, o jQuery Mobile apenas modifica o elemento diretamente no DOM, levando mais tempo para o modificar. Um grande ponto positivo do React e Vue é que podem ser utilizados como um SPA (Single Page Application), assim, em vez de processar uma página completa na mudança de url, processa apenas o necessário, aumentando o desempenho no lado do cliente.

Em suma, podemos concluir que o Angular organiza a sua estrutura de forma mais escalável, facilitando a construção de aplicações em grande escala e, através do TypeScript proporciona uma abordagem orientada a objetos, permitindo utilizar padrões de desenho. Por outro lado, a biblioteca React tem um grande apoio e soluções comunitárias, sendo muito utilizando quando necessário desenvolver aplicações empresariais leves e na atualização de funcionalidades de outras aplicações. O Vue como é muito recente, é utilizado apenas por pequenas empresas, em que o foco passa por desenvolver aplicações rápidas e de alto desempenho mas, com recursos limitados. Por fim, o jQuery Mobile apresar de não possuir muita popularidade, permite aos *developers*, que já conheçam a tecnologia jQuery, criar aplicações com muito menos custo de desenvolvimento.

Tabela 3. Características das tecnologias de desenvolvimento PWA

	Linguagem	**Licenças**	**IDE**	**Tipo de Tecnologia**	**Suporte**	**Plataformas**
Angular	HTML, CSS e TypeScript	Grátis	Não	Framework	Open-Source + Google	iOS, Android e Windows Phone
Vue	HTML, CSS e JavaScript	Grátis	Não	Framework	Open-Source	iOS, Android e Windows Phone
React	React	Grátis	Não	Biblioteca	Open-Source + Facebook	iOS, Android e Windows Phone
jQuery Mobile	HTML, CSS e JavaScript	Grátis	Não	Framework	Open-Source + jQuery	iOS, Android e Windows Phone

3.1.3.2 Implementação Híbrida

De uma forma semelhante à implementação anterior, a implementação híbrida também utiliza um *browser* para processar e interpretar toda a aplicação. Porém contrariamente da implementação WebApp, a aplicação é encapsulada numa WebView e, como podemos observar na Figura 29, esta

consegue aceder às funções e sensores fornecidos pelo SO através de uma bridge construída em JavaScript, fornecendo as API's do sistema [66].

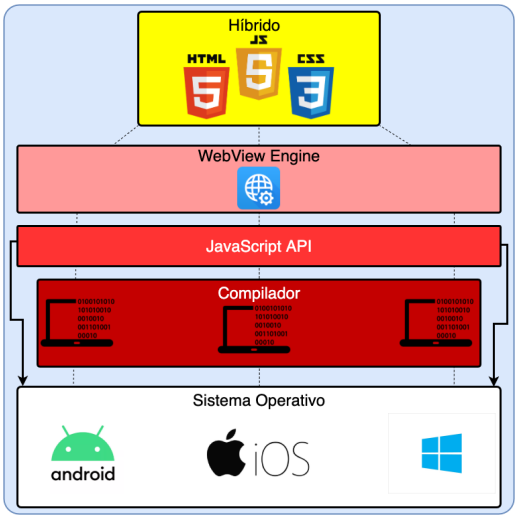

Figura 29. Esquema de desenvolvimento Híbrido

Através da utilização do SDK fornecido pelos SO's, permite-se a utilização de componentes e funcionalidades nativas, deste modo, a aplicação fica encapsulada através de uma WebView, tornando-se mais segura que as PWA. Contudo, com a utilização de uma camada abstracta de Hardware construída em JavaScript, a aplicação fica sujeita a vulnerabilidades [93].

Este SDK irá fornecer ferramentas para realização de testes e funcionalidades que os PWA não permitem, mantendo a utilização de tecnologias e linguagens web conhecidas, como por exemplo, HTML, CSS, JavaScript, entre outros. Grande parte das tecnologias híbridas permitem a integração de bibliotecas e frameworks populares, desta forma um *developer* pode integrar uma tecnologia que já conheça, consequentemente, reduzindo os custos de implementação [92].

Em seguida serão descritas, de forma sucinta, as tecnologias, Ionic, PhoneGap, Cordova e Framework7, uma vez que são consideradas as tecnologias híbrido mais estáveis do mercado. Por fim, será apresentada uma comparação entre estas tecnologias.

Cordova

A tecnologia Cordova consiste numa framework Open-Source de desenvolvimento híbrido construída por Nitobi e adquirida mais tarde pela Apache, que permite utilizar linguagens web, HTML5, CSS3 e JavaScript para a construção de aplicações Cross-Platform [49]. As aplicações são encapsuladas numa WebView de forma a serem compatíveis com as várias plataformas e SO's. Por utilizar as linguagens web, um *web-developer* consegue reaproveitar grande parte do seu código e implementá-lo na aplicação móvel [50].

A sua arquitetura, apresentada na Figura 30, é composta pela aplicação web, parte onde reside todo o código da aplicação contendo todos os recursos necessários para a construção de uma WebApp em conjunto com config.xml (ficheiro de configuração) [50], uma WebView fornecida pelo SDK que irá renderizar e processar todo o conteúdo web e, por fim, os *Plug-in's* que são a parte mais importante da framework. Estes fornecem uma interface para que a framework e os componentes nativos comuniquem entre si, permitindo invocar funções nativas a partir do JavaScript [50].

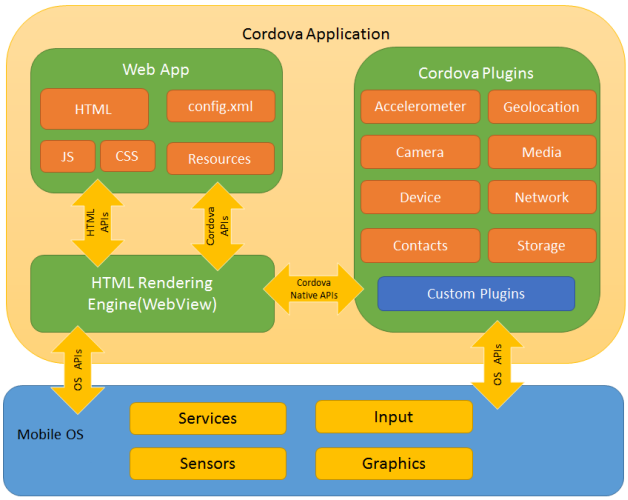

Figura 30. Arquitetura geral do Cordova [50]

Quando uma aplicação é criada, não possui quaisquer *Plug-in's* instalados, estes têm de ser explicitamente adicionados, desta forma o *developer* apenas inclui o que necessita, diminuindo o tamanho final da aplicação [50]. O Cordova apenas mantém um conjunto de *Plug-in's*, chamados de Core *Plug-in's*, que permitem aceder às capacidades dos dispositivos como bateria, câmara, contactos, entre outros. Para além deste *Plug-in's*, a framework permite incluir *Plug-in's* de terceiros que fornecem ligações adicionais a funcionalidades não necessariamente disponíveis em todas as plataformas [49] [50].

Esta framework permite, na criação do projeto, a escolha entre um fluxo de trabalho em Cross-Platform que possibilita um correto funcionamento entre vários dispositivos e SO's e o fluxo de trabalho centrado na plataforma, que permite realizar modificações de baixo nível, como por exemplo, misturar componentes nativos personalizados com componentes baseados em web [50]

Ionic

A plataforma Ionic construída em 2013, fornece vários produtos e serviços para desenvolvimento *mobile* Cross-Platform e desenvolvimento de *Plug-in's* com funcionalidades nativas [63].

Esta plataforma fornece uma framework Open-Source para desenvolvimento *mobile* Cross-Platform que utiliza a framework Cordova acima descrita como base. Apesar de se basear e utilizar o Cordova, esta framework construída em Node.JS fornece recursos que simplificam a construção das aplicações, em conjunto com as linguagens HTML, CSS e TypeScript [107], permite a integração de frameworks web populares, em específico, o Vue, React e Angular [64], utilizando as API's fornecidas pelo Cordova para comunicar com o SO.

Através da integração de uma framework, como o Angular (Figura 23), o desenvolvimento torna-se mais rápido e com menos custos, isto devido a utilização de uma tecnologia conhecida pelo *developer*. Como esta framework não fornece um IDE, todas as funcionalidades são incluídas pela CLI, permitindo incorporar ferramentas de outras tecnologias, como por exemplo o Router do Angular [64] e ferramentas do Cordova, como por exemplo, o Core *Plug-in's*. Este CLI, possui ainda ferramentas de compilação em tempo real, diminuindo o tempo de construção de uma aplicação e, ferramentas de debug utilizando um browser, clarificando os erros e consumo de recursos.

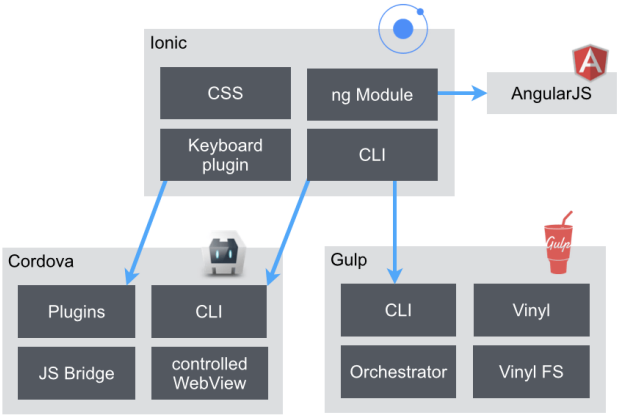

Figura 31. Arquitetura geral do Ionic

PhoneGap

PhoneGap é uma framework de desenvolvimento Open-Source construída e mantida pela Adobe, que permite a utilização das linguagens HTML, CSS e JavaScript para a criação de aplicações *mobile* [107]. Esta tecnologia é muito popular entre os *developers* principalmente devido à sua flexibilidade e facilidade de utilização sendo adequada quando já existe uma aplicação web, que se pretende converter para um ambiente móvel [58].

Esta tecnologia utiliza como base arquitetural o Cordova, de uma forma muito similar a tecnologia Ionic, como podemos observar na Figura 32, facilitando o desenvolvimento Cross-Platform e garantindo um correto funcionamento nas plataformas Android, iOS e Windows Phone [89]. Da mesma forma que o Cordova, permite também a integração de *Plug-in's* criados por terceiros, sendo que todas as funções nativas são feitas pela API realizada em JavaScript pelo Cordova [3].

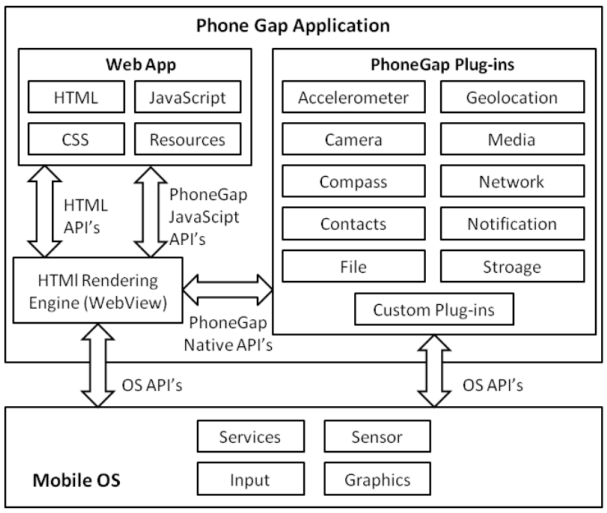

Figura 32. Arquitetura geral do PhoneGap [89]

O PhoneGap não fornece um IDE para a construção de aplicações, mas impede a criação de ambientes de desenvolvimento centralizados, sendo necessário utilizar ferramentas CLI para a compilação da aplicação [30]. Contudo, na última versão foi lançada uma aplicação para *desktop* como alternativa as ferramentas CLI, facilitando a instalação da framework, criação e compilação de aplicações [2].

Framework7

De uma forma semelhante às tecnologias anteriormente descritas, a Framework7 consiste numa framework Open-Source construída em Node.Js, de desenvolvimento *mobile* híbrido, web e desktop utilizando as linguagens HTML, CSS e JavaScript, e além disso, pode ser utilizado como ferramenta de prototipagem [122]. Para além da utilização destas linguagens, permite incluir frameworks conhecidas como Vue, React e Svelte, facilitando o desenvolvimento caso o *developer* já conheça estas tecnologias, mas em termos de design foca-se no design do iOS e Google Material para trazer uma maior simplicidade [124].

Como a Framework7 não fornece um IDE, todas as ferramentas estão disponíveis pelo CLI. Ao utilizar o CLI, podemos escolher qual a framework a usar (Framework7 Core, Framework7-Vue, Framework7-React, Framework7-Svelte) e qual a abordagem (PWA, híbrida ou desktop) [123]. As ferramentas para a construção de aplicações híbridas são obtidas através da integração do Cordova, através da utilização de *Plug-in's* para aceder as API's, sendo também possível criar *Plug-in's* em JavaScript facilitado a construção e reutilização de código.

Comparação

Todas estas tecnologias utilizam linguagens web, nomeadamente, HTML, CSS e JavaScript, sendo assim necessário ter algum conhecimento nestas linguagens antes se começar a utilizar-las. No entanto, para equipas de desenvolvimento web, a utilização destas tecnologias torna-se ideal para a construção de aplicações *mobile* Cross-Platform.

Estas tecnologias comportam-se de forma semelhante, a grande diferença está no facto de algumas integrarem tecnologias de desenvolvimento web (Tabela 4), sendo este o fator decisivo caso o *developer* tenha um conhecimento prévio nessas tecnologias.

Em termos de popularidade, o Cordova e o Ionic sempre tiveram uma grande popularidade em comparação com o PhoneGap e o Framework7 (figuras 33 e 34). Isto porque, o Cordova é o pilar de todas estas tecnologias, sempre foi muito utilizado, mesmo de forma independente, no entanto, o Ionic ganhou popularidade por utilizar o Cordova e permitir a utilização de tecnologias web, como o Vue, React, entre outros.

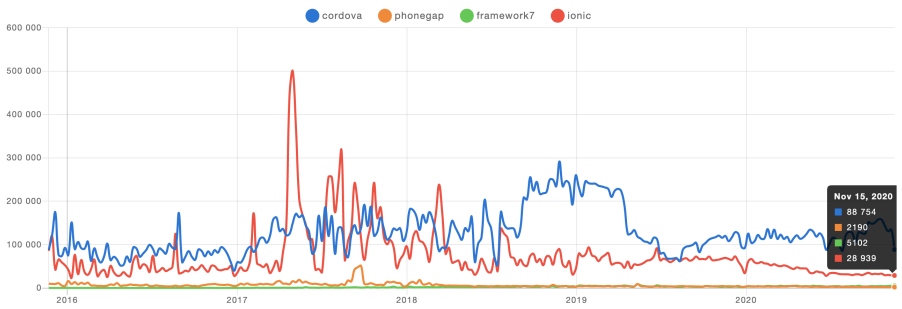

Figura 33. Número de downloads realizados na ferramenta do Node.js (NPM) entre 2016 e final de 2020

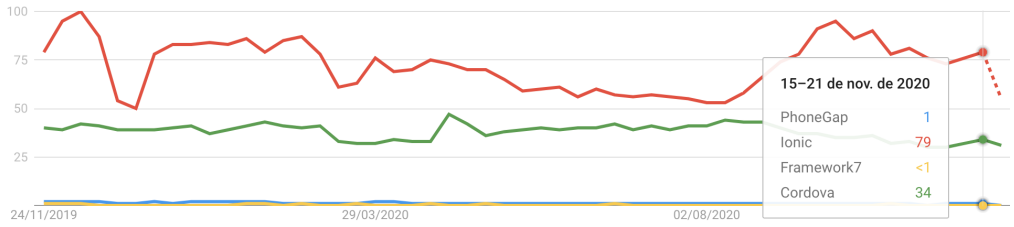

Figura 34. Número de pesquisas realizadas no Google entre 2015 e final de 2020, utilizando as palavras chave (cordova/ionic/phonegap/framework7)

Já em termos de performance, todas estas utilizam uma WebView como forma de processar o conteúdo web, portanto as limitações continuam a ser impostas pelo *browser*. Por utilizarem o Cordova como base arquitetural, todas as falhas de segurança e vulnerabilidades são criadas pela camada abstrata construída em JavaScript. Qualquer falha na camada abstrada é da responsabilidade do Cordova e, caso este tenha a necessidade de se atualizar, será necessária a atualização de todas as outras tecnologias, para beneficiar desta atualização da versão do Cordova.

Em resumo, podemos concluir que sendo o Cordova a base arquitetural de todas estas tecnologias, é a primeira a realizar atualizações no maior ponto de possíveis falhas que é a camada abstrata em JavaScript, porém apenas suporta as linguagens HTML, CSS e JavaScript. Por outro lado, o Ionic tem todas as funcionalidades do Cordova e é capaz de incorporar mais tecnologias de desenvolvimento web tornando-se mais flexível e facilita o transporte de uma aplicação web para aplicação móvel, obtendo todas as funcionalidades e vantagens dessa tecnologia. O PhoneGap contém todas as funcionalidades do Cordova, mas com o apoio da comunidade, oferece uma grande variedade de *Plug-in's*. Por último a Framework7 permite o desenvolvimento PWA, desktop e *mobile* híbrido através do mesmo código, permitindo também a utilização de tecnologias web, como o Vue, React, entre outros, apenas utilizando o Cordova para desenvolvimento híbrido.

Tabela 4. Características das tecnologias de desenvolvimento híbrido

	Linguagem	**Licenças**	**IDE**	**Tipo de Tecnologia**	**Suporte**	**API's**	**Plataformas**
Cordova	HTML, CSS e JavaScript	MIT	Não	Framework	Open-Source + Apache	Todas	iOS, Android e Windows Phone
Ionic	HTML, CSS e TypeScript/ ++**	Flexível*	Não	Plataforma	Open-Source + Ionic	Todas	iOS, Android e Windows Phone
PhoneGap	HTML, CSS e JavaScript	Grátis	Não	Framework	Open-Source + Adobe	Todas	iOS, Android e Windows Phone
Framework7	HTML, CSS e JavaScript/ ++**	MIT	Não	Framework	Open-Source + Vladimir Kharlampidi	Não***	iOS, Android e Windows Phone

*Opções pagas e gratuitas

** Opção de adição de frameworks ou bibliotecas de desenvolvimento web

*** Necessário a integração do Cordova

3.1.3.3 Implementação Interpreted, Cross-Compiled e MDSD

Tal como as implementações anteriores, estas também focam-se no desenvolvimento *mobile* Cross-Platform utilizando apenas um ambiente e uma ou mais linguagens de programação, garantindo

o bom funcionamento da aplicação entre diferentes plataformas e dispositivos [75]. No entanto, ao contrário das anteriores, estas implementações não utilizam um *browser* para processar a aplicação, como podemos observar na Figura 35 [132].

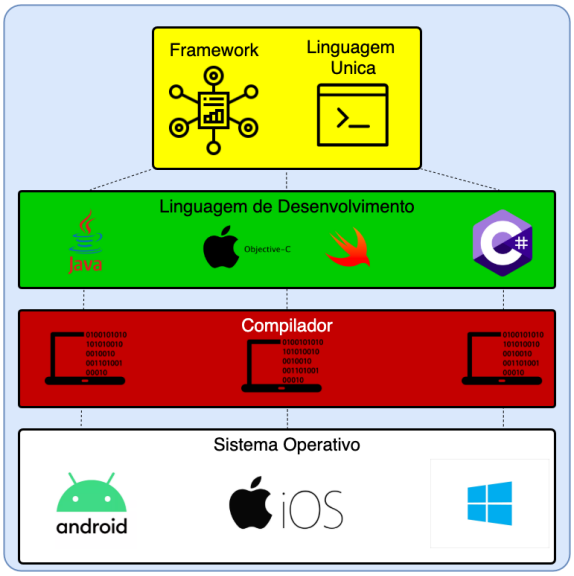

Figura 35. Esquema de desenvolvimento Cross-Platform

Todas estas implementações utilizam os SDK's fornecidos pelos SO's, permitindo aceder a todas as funcionalidades do dispositivo e aceder às ferramentas disponibilizadas pelo SDK, como por exemplo, ferramentas de debug, distribuição para as *app stores*, entre outras.

As implementações *Interpreted* e *Cross-Compiled* focam-se em a partir de um código escrito numa determinada linguagem de programação, geram um programa nativo para cada plataforma, sendo este interpretado em runtime ou compilado directamente [69]. Entretanto, a implementação MDSD utiliza outra estratégia, foca-se apenas em modelos de representação de tarefas obrigatórias antes de iniciar o desenvolvimento de cada aplicação [132].

Dito isto, serão descritas as tecnologias React Native, Flutter, Appcelertor Titanium, Xamarin, NativeScript, Rhomobile, Unity, Kivy e Microsoft Power Apps, sendo que são as mais populares do mercado e com um considerável nível de manutenção. Por último, será apresentada uma comparação entre estas tecnologias.

React Native

O React Native consiste numa framework Open-Source, criada pela Facebook em 2015, construída em Node.JS, que utiliza como base a biblioteca React para criar aplicações Cross-Platform [98]. Como este utiliza a biblioteca React, facilita a reutilização de código e transporte do código da aplicação web para a aplicação *mobile* [108]. Isto permitiu que os *Plug-in's* e componentes criados para uma certa aplicação web pudessem ser utilizados na aplicação *mobile*, baixando assim os custos de desenvolvimento.

A grande diferença entre o React Native e o React, é que como React Native utiliza os SDK's dos SO's, todos os componentes nativos são acedidos diretamente na framework, como mostra a Figura 36 através de código JavaScript e inclusão dos *Plug-in's* necessários [97].

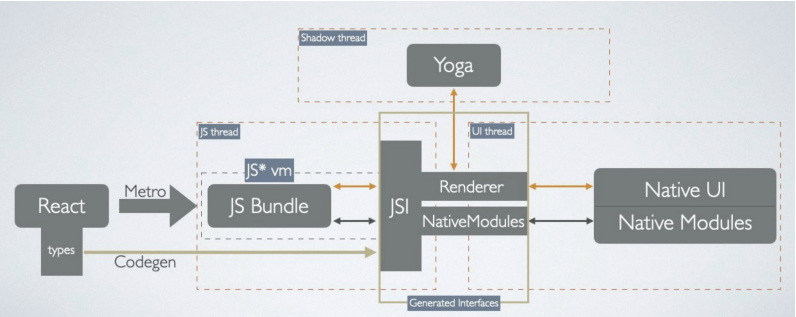

Figura 36. Esquema arquitetural do React Native

Como o React Native possui uma arquitetura *Interpreted*, para além de conseguir aceder diretamente aos componentes e funcionalidades nativas, permite modificá-los de forma a apenas utilizar as funcionalidades necessárias, aumentando assim o seu desempenho [96].

Pelo grande êxito desta framework, a sua popularidade aumentou, tal como as contribuições da comunidade. Muitas módulos e bibliotecas foram criados pela comunidade, desde bibliotecas para interface, como otimização de *Plug-in's* e API's. Atualmente, com o crescimento da framework, é possível substituir o JavaScript pelo TypeScript, modificando o modo de construção, passando de funcional para POO [97].

Uma das ferramentas mais conhecidas, criada em torno do React Native é o Expo, que consiste num SDK que fornece um conjunto de ferramentas que simplificam o desenvolvimento e o teste da aplicação criada em React Native e, todas as API's dos SO's [44]. Este traz muitas vantagens, como por exemplo, não necessita do Xcode ou Android Studio para executar e testar a aplicação no dispositivo móvel, oferece uma coleção de soluções testadas para utilização de API's dos SO's mantidas pela equipa da Expo [43]. No entanto, tem muitas limitações, como por exemplo, não permite adicionar módulos nativos escritos em Objective-C, Swift, Java ou Kotlin, torna a aplicação com um enorme tamanho, como contém todas as soluções do SDK, mesmo que não sejam necessárias para a aplicação em questão, entre outros problemas dependentes da sua arquitetura [45].

Flutter

A framework Flutter construída pela Google, utiliza como linguagem principal o Dart e tem como objetivo, a criação de aplicações *mobile*, web e desktop Cross-Platform [46]. Esta framework possui um conjunto de ferramentas de interface gráfica, concebido para permitir a reutilização de código em SO's como o iOS e o Android, facilitando as chamadas das API's do sistema [47].

Durante o desenvolvimento de uma aplicação, estas correm numa máquina virtual que oferece um modo *hot reload*, recompilando apenas a parte necessária da aplicação, baixando os tempos de espera de recompilação completa. Quando as aplicações são compiladas para o lançamento nas *app stores*, este transforma o código Dart em código de máquina, seja instruções em x86 ou ARM, fazendo com que o código final seja completamente nativo à plataforma [47].

O Flutter foi construído de forma a ser um sistema extensível e por camadas, como podemos observar na Figura 37. Este baseia-se numa arquitetura *Cross-Compiled*, existindo uma série de bibliotecas que dependem cada uma da camada inferior, sendo que, cada parte do nível da estrutura está concebida para ser opcional e substituível. [47]. No núcleo está o motor (*engine*) escrito em C++ que suporta as primitivas necessárias para apoiar todas as aplicações (Entrada de dados, gráficos, subsistemas de renderização de texto, entre outros), o engine é também responsável por

garantir o correto funcionamento da framework (compilação das ferramentas, inclusão de *Plug-in's*, entre outros). De seguida, está o *Embedder* construído de forma que o Flutter possa ser integrado numa aplicação como um módulo [47].

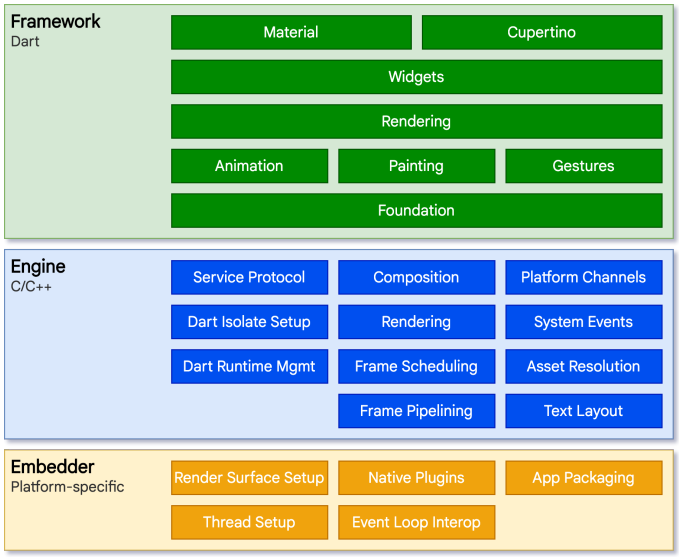

Figura 37. Arquitetura do Flutter

As aplicações são construídas através de *Widgets*, que são os blocos de construção da interface de uma aplicação, podendo ser *StateFull*, *StateLess*, entre outros. Por fim, estes seguem um tipologia de árvore, podendo aceder aos dados do pai e modificar os dos filhos [47].

Appcelerator Titanium

A plataforma Appcelerator construída pela Axway [13], apresenta produtos para a construção de aplicações *mobile*, especificamente uma plataforma para desenvolvimento *mobile*, uma API para a criação de End-Points e, por fim, uma ferramenta de análise de dados [14]. A plataforma Open-Source chama-se Titanium, construída em 2006, oferece a possibilidade de desenvolver aplicações *mobile* Cross-Platform em JavaScript [30].

O Titanium oferece várias ferramentas para desenvolvimento *mobile* Cross-Platform, sendo as mais importantes: um próprio SDK baseado em Node.js que possui ferramentas de apoio que funcionam com as ferramentas nativas; a ferramenta Hyperloop que dá ao utilizador acesso direto às API's nativas, permitindo modificá-las; a framework Alloy que permite o desenvolvimento de aplicações *mobile* Cross-Platform utilizando o padrão MVC e as linguagens JavaScript, XML e TSS (Titanium Style Sheets); uma API criada em JavaScript que fornece acesso a vários componentes nativos; e um IDE chamado de Appcelerator Studio, que permite criar um ambiente de desenvolvimento integrado, facilitando a gestão de SDK's e módulos de Node.JS [15].

A arquitetura da framework do Titanium (Figura 38) é *Interpreted*, compilando o código JavaScript em runtime e comunicando com as API's nativas através de uma bridge que consiste no SDK do Titanium [15]. Quando inicia-se o processo de compilação, uma pré-compilação recolhe todo o código de JavaScript otimizando-o (reduzir espaços em branco, otimiza funções, entre outros), e de seguida, cria uma hierarquia de dependências de todas as API's utilizadas, aumentando o desempenho da aplicação [58].

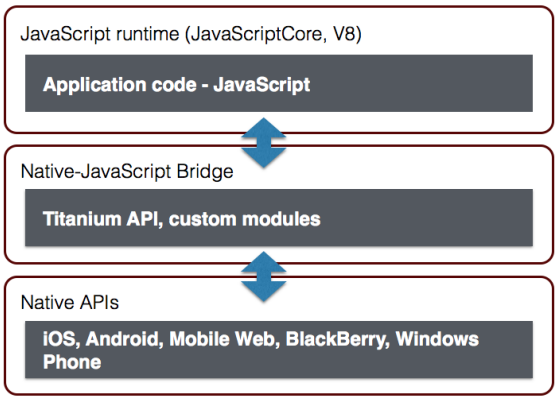

Figura 38. Esquema simplificado da arquitetura do Titanium

O Titanium utiliza como linguagem principal o JavaScript, em que os *developers* necessitam de conhece-la, pelo motivo, que todas as API's são chamadas a partir desta linguagem [30].
Inicialmente as plataformas suportadas eram o iOS, Android, Blackberry, Tizen e WindowsPhone [30], contudo os novos SDK's do Titanium apenas suportam o sistemas iOS e Android [15]. Esta plataforma fornece ferramentas grátis mas com uso limitado, no entanto, muitos outros produtos e ferramentas podem ser adquiridas [14].

Xamarin

A plataforma Xamarin possibilita ao *developer* criar aplicações *mobile* Cross-Platform, utilizando apenas a linguagem C#, usufruindo de todas as componentes nativas disponibilizadas pelo dispositivo [107]. Construída pela Microsoft, esta framework Open-Source fornece as ferramentas necessárias para a construção de uma aplicação apenas com o uso de C# e .Net, suportando as plataformas iOS, Android e WindowsPhone [90]. Esta torna-se assim como uma camada de abstração, que gere a comunicação do código partilhado com o código da plataforma, permitindo aos *developers* a partilha de cerca de 90% do código da sua aplicação entre as várias plataformas [90].

A arquitetura geral de uma aplicação *mobile*, como mostra a Figura 39, permite construir uma interface nativa em cada plataforma e usa o C# para construir as funcionalidades da aplicação. Utilizando como base a arquitetura *Cross-Compiled*, transforma todo o código escrito em C# em código nativo do dispositivo [22] [90].

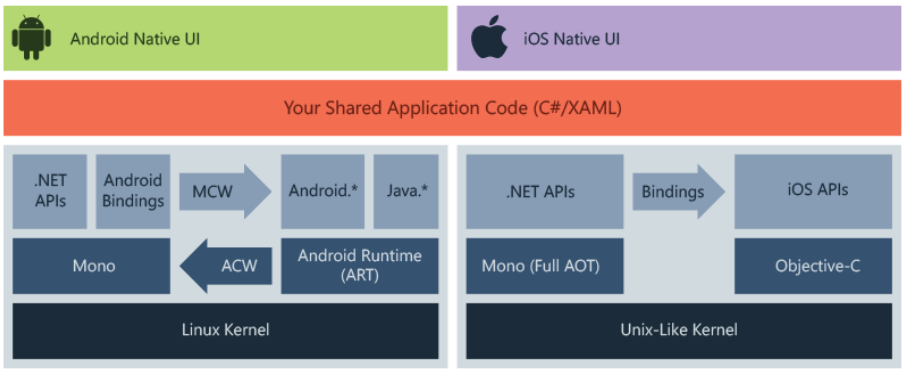

Figura 39. Esquema da arquitetura MVC do Xamarin

O Xamarin oferece várias ferramentas para a construção de aplicações *mobile* Cross-Plataform, sendo as principais: o IDE Visual Studio, que inclui várias ferramentas de suporte na construção de código e inclusão de *Plug-in*'s; a framework Xamarin.Android que utiliza a compilação *Just-in-Time* (JIT) para criar aplicações para Android, que irão funcionar no ambiente de uma máquina virtual; a framework Xamarin.ios que utiliza a compilação *Ahead-of-Time* (AOT) transformando o código C# em código ARM nativo; a framework Xamarin.Forms que permite a construção de aplicações *mobile* Cross-Platform num único projecto, utilizando-se as linguagens XAML e C#; e uma biblioteca Xamarin.Essentials que fornece as API's simplificando o acesso às funcionalidades nativas [90] [31] [34] [78].

NativeScript

O NativeScript é uma framework Open-Source de desenvolvimento *mobile* Cross-Platform que permite a construção de aplicações utilizando tecnologias web [84], para as plataformas iOS e Android. Construída em 2015 pela Progress, utiliza como base a arquitetura *Interpreted*, que suporta as linguagens JavaScript e TypeScript compilando o código em runtime [84]. O foco principal desta framework é a integração de tecnologias web para o desenvolvimento de aplicações *mobile*, como o Vue, Angular, entre outros, facilitando assim o desenvolvimento *mobile* para equipas de desenvolvimento web [86].

Como podemos observar na Figura 40, os módulos principais da arquitetura do NativeScript são: o NativeScript runtimes que permitem utilizar as API's fornecidas pelos sistemas Android e iOS; o Core Modules que forenecem as abstrações necessárias para aceder às componentes nativas, como gestos, entre outros; a ferramenta CLI que permite criar, construir e executar as aplicações NativeScript; e os NativeScript *Plug-in's* que são os blocos de construção que encapsulam algumas funcionalidades, que na maioria são construídas pela comunidade [86].

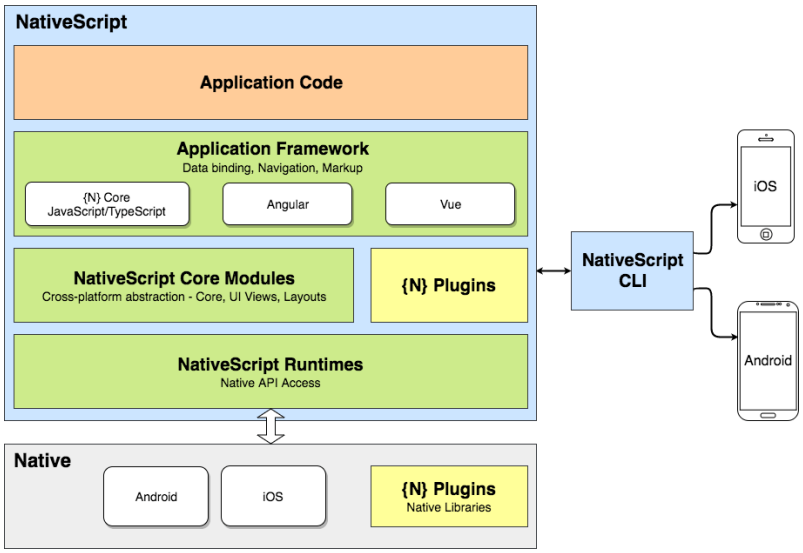

Figura 40. Esquema arquitetural do NativeScript

Para além da ferramenta CLI do NativeScript que permite a construção de aplicações em qualquer IDE [83], este fornece um próprio IDE construído para simplificar a construção de aplicações, em que todo o processo é realizado no browser, não sendo necessário instalar dependências no computador [85].

Rhomobile

A Rhomobile fornece várias ferramentas para apoiar a construção de aplicações *mobile*, em que uma delas é a framework Rhoads criada em 2008, que utiliza a arquitetura *Interpreted* [58]. Esta framework é Open-Source e possui a Ruby como principal linguagem de programação e linguagens web como secundárias, seguindo o padrão MVC (Model, View, Controller) [89].

Inicialmente construída pela Motorola Solutions Inc. permitia o desenvolvimento para todos os SO's, nomeadamente, iOS, Android, BlackBerry, WindowsPhone e Symbian. Contudo atualmente apenas permite os SO's iOS, Android, BlackBerry, WindowsPhone [80]. Podemos observar na Figura 41 a arquitetura da framework Rhoads, que permite modificar a parte do visual com as linguagens web e as comunicações com API's do sistema e *Plug-in's* com a linguagem Ruby.

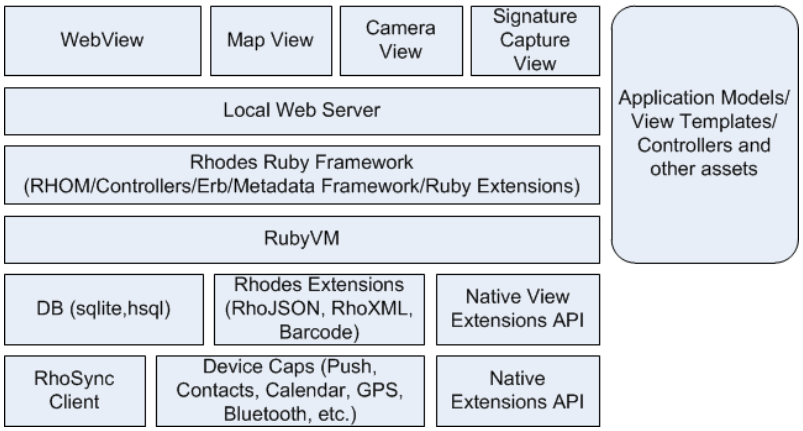

Figura 41. Arquitetura generalizada do Rhoads

Esta plataforma fornece um IDE, para ajudar na construção de aplicações *mobile*, podendo ser instalada nos sistemas Linux, MacOS e Windows mas, com a utilização do CLI é possivel utilizar qualquer IDE da preferência do *developer* [89].

Unity

Uma das tecnologias mais conhecidas de desenvolvimento de jogos é Unity, que permite criar jogos e aplicações para *mobile* e desktop [121]. Este possui uma vasta gama de ferramentas e recursos para a criação de jogos (2D e 3D), filmes, animações e integração de realidade virtual e realidade aumentada, sendo este um dos ponto fortes desta tecnologia [118].

O Unity fornece um próprio IDE, para facilitar todo o desenvolvimento, desde a modificação de elementos 3D, à integração de ficheiros C#. Apesar de se focar em jogos, permite criar aplicações Cross-Platform muito idênticas às nativas, sendo que compila o código escrito em C# e transforma-o em código binário nativo da plataforma escolhida [120]. A sua arquitetura é baseada em *Cross-Compiled*, como mostra a Figura 42, utiliza uma camada de abstração para converter as API's construídas em C# para as nativas [119].

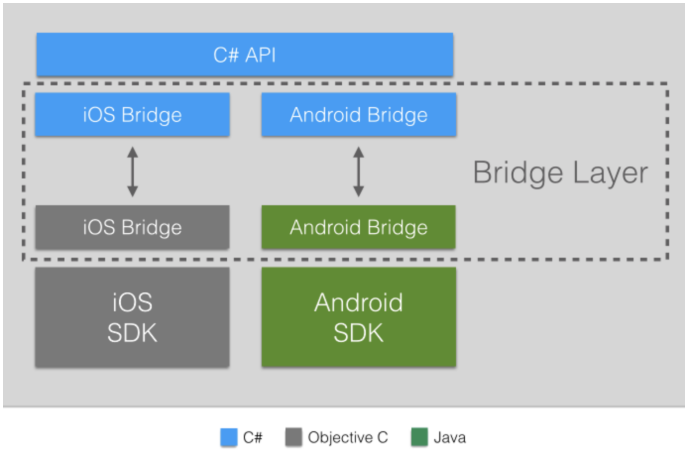

Figura 42. Esquema arquitetural do Unity

Apesar de grande parte do seu conteúdo ser gratuito, existem ferramentas que só podem ser acedidas caso tenhamos planos pagos [117].

Kivy

Com a grande popularidade que a linguagem Python possui, já era de esperar a sua utilização para a construção de uma ferramenta de desenvolvimento *mobile*. Muito recentemente, uma nova framework de desenvolvimento *mobile* foi criada, chamada de Kivy. Esta foi primeira framework de desenvolvimento *mobile* Cross-Platfom a utilizar como linguagem principal Python. O Kivy é Open-Source e é capaz de ser executado em Linux, Windows, OS X, Android, iOS e Raspberry Pi, podendo correr o mesmo código em todas as plataformas suportadas [68].

A arquitetura do Kivy (Figura 43) é baseada em *Interpreted* e possui compiladores para interpretar o código em Python e os módulos separados pelas suas funcionalidades e características. De acordo com a sua forma de construção, o *developer* apenas necessita de conhecer as camadas superiores da arquitetura para conseguir compreender como otimizar e gerir os recursos da aplicação [67].

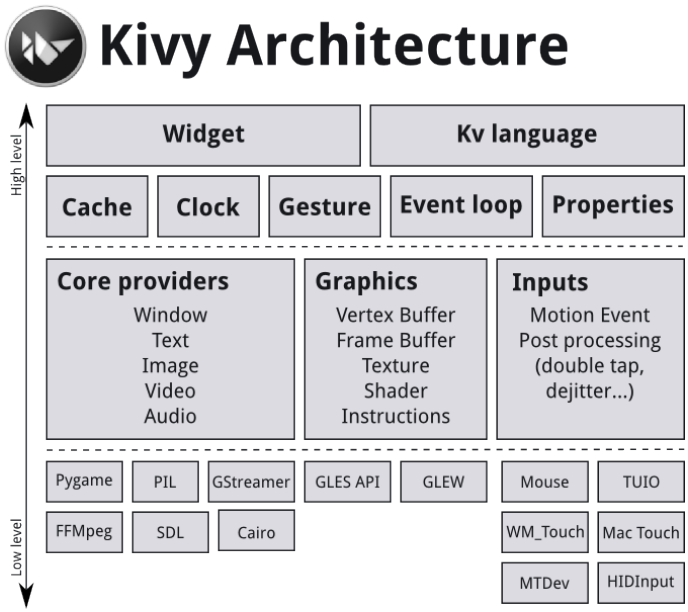

Figura 43. Arquitetura de alto nivel a baixo nivel do Kivy

Para a construção de aplicações utilizando esta framework é necessário possuir bons conhecimentos em Python, pelo motivo de este permitir a construção de aplicações em modo funcional, declarativo e POO em conjunto com a criação de Widgets [67].

Microsoft Power Apps

Ao contrário de todas as outras plataformas e frameworks, a plataforma Power Apps segue uma abordagem muito diferente. Este utiliza uma arquitetura derivada por modelos (*Model Driven Software Development*), não sendo necessário utilizar um linguagem de programação. No entanto, atualmente, só possui apenas licenças pagas e restritas a empresas [73].

Esta plataforma, consiste num conjunto de aplicações, serviços, conectores e plataformas de dados que fornece um ambiente de programação rápida para que a criação de aplicações seja mais rápida e eficiente (Figura 44) [71]. Com o foco em empresas, as aplicações podem ser ligadas aos dados armazenados em qualquer serviço da Microsoft [73].

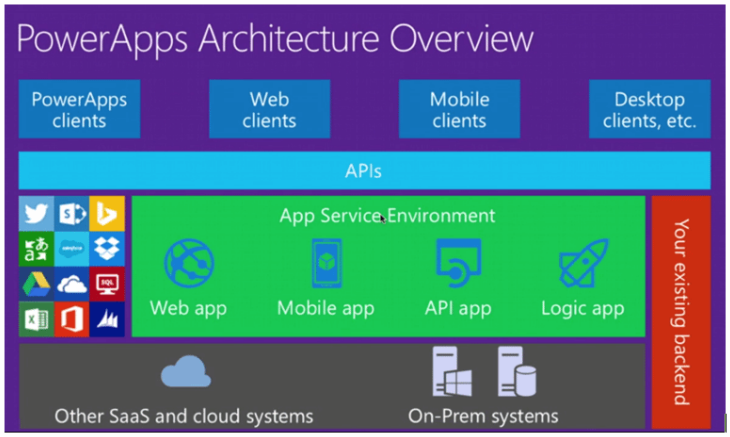

Figura 44. Esquema geral da platafora Power Apps

As aplicações criadas a partir do Power Apps possuem um design responsivo, podendo ser executadas num *browser*. O foco principal desta plataforma é a construção de uma aplicação sem a utilização de código, desta forma qualquer pessoa sem conhecimentos de programação pode criar uma aplicação a nível empresarial, no qual apenas é necessário alguns conhecimentos na área de engenharia de software [81]. Uma plataforma extensível também é fornecida para permitir aos *developers* interagir com dados, aplicar funcionalidades, criar conectores personalizados e integrar dados externos através de código [72].

Comparação

As tecnologias anteriormente descritas são todas diferentes e possuem formas de desenvolvimento diferenciados, podemos observar as suas características principais na Tabela 5. No entanto todas estas têm o mesmo foco e objetivo final, que passa por permitir a criação e desenvolvimento de uma aplicação *mobile* Cross-Platform que se assemelham a uma aplicação nativa e apresenta todas as características nativas. Desta forma, todas estas tecnologias pretendem baixar os custos de desenvolvimento, reduzindo a necessidade de existir duas equipas de desenvolvimento separadas por SO's para criar aplicações *mobile* com comportamento nativo.

Em termos de popularidade da arquitetura *Interpreted*, o React Native sempre contou com uma grande popularidade devido à comunidade da tecnologia principal, o React, que por sua vez, permitir a reutilização de código utilizado num aplicação web para a construção de uma aplicação *mobile*

nativa. As outras tecnologias *Interpreted* sempre mantiveram um nível de popularidade constante, sendo que o NativeScript conseguiu manter-se superior às outras, pois, conseguiu incorporar várias tecnologias web para o desenvolvimento de aplicações *mobile* nativas.

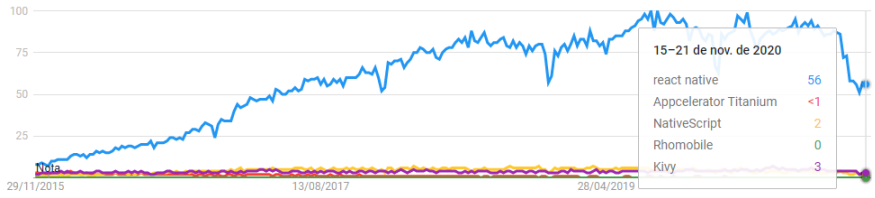

Figura 45. Número de pesquisas realizadas no Google entre 2015 e final de 2020, utilizando as palavras chave(React Native/ Appcelerator Titanium/ NativeScript/ Rhomobile/ Kivy)

Já na popularidade da arquitetura *Cross-Compiled*, o Unity possuiu um amplo nível de popularidade, contudo a utilização desta tecnologia não consiste apenas no desenvolvimento *mobile*, permite, também, a construção de aplicações *mobile* com realidade aumentada, que é o fator de diferenciação em relação às restantes. O Xamarin desde a sua criação, manteve um constante nível de popularidade, pelo motivo de utilizar a linguagem C#, mas com o aparecimento do Flutter, perdeu popularidade concedendo o seu lugar. Por fim, o Power Apps, nunca teve muita procura por ser considerado uma tecnologia empresarial, com o foco em ter apenas utilizadores empresariais.

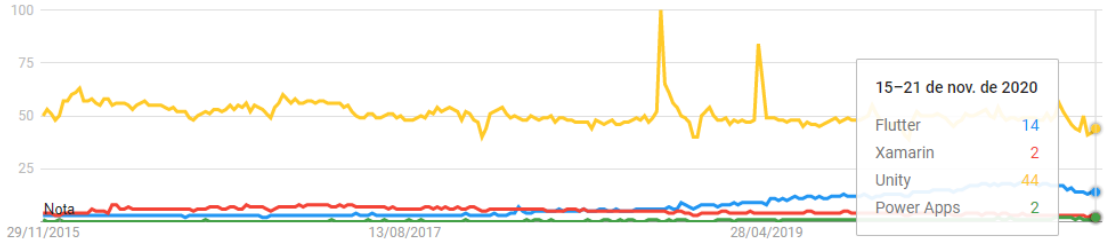

Figura 46. Número de pesquisas realizadas no Google entre 2015 e final de 2020, utilizando as palavras chave (Flutter/Xamarin/Unity/Power Apps)

Em termos de performance, as tecnologias distinguem-se pela forma como a aplicação final é processada, isto é, se é processada em runtime (*Interpreted*), se compila para código binário nativo (*Cross-Compiled*) ou se é realizado por modelação (*MDSD*). Uma aplicação *mobile* que esteja compilada para código binário nativo poderá ter uma melhor performance, pelo motivo de não ser necessário compilar código em runtime. Contudo, atualmente, as aplicações realizadas por modelação, são às menos eficientes, sendo que ainda estão a ser estudadas e trabalhadas.

Em suma, as grandes diferenças entre estas, são as linguagens de programação e a comunidade que as suporta. Caso uma empresa esteja disponível a ter uma equipa de *developers* para a criação e manutenção de aplicações *mobile*, necessita de ter em conta, quais as linguagens favoritas dos *developers* e se a tecnologia tem uma comunidade que suporta, facilitando a resolução de erros

imprevistos. Da mesma forma, é necessário ter em conta o nível de formação e conhecimentos dos *developers*, visto que o processo de formação de novos conceitos, linguagens e tecnologias é dispendioso. No entanto, existem opções que utilizam tecnologias web, facilitando o desenvolvimento de uma aplicação *mobile*, caso estes queiram investir numa nova linguagem.

Tabela 5. Características das tecnologias de desenvolvimento Cross-Platform

	Linguagem	**Licenças**	**IDE**	**Tipo de Tecnologia**	**Suporte**	**Arquitetura**	**Plataformas**
React Native	React	Grátis	Não	Framework	Open-Source + Facebook	Interpreted	iOS e Android
Flutter	Dart	Grátis	Não	Framework	Open-Source + Google	Cross Comp.	iOS e Android
Appc Titanium	XML, TSS e JavaScript	Flexível*	Sim	Framework / Plataforma	Appcelerator	Interpreted	iOS e Android
Xamarin	C#	Flexível*	Sim	Framework / Plataforma	Open-Source + Microsoft	Cross Comp.	iOS, Android e Windows Phone
NativeScript	TypeScript ou JavaScript /XML e CSS / ++**	Grátis	Sim	Framework	Open-Source	Interpreted	iOS e Android
Rhomobile	Ruby, JavaScript, HTML e CSS	Grátis	Sim	Plataforma	Open-Source	Interpreted	iOS, Android, Windows Phone e BlackBerry
Unity	C#	Flexível*	Sim	Framework	Unity	Cross Comp.	iOS e Android
Kivy	Python	MIT	Não	Framework	Open-Source + Kivy	Interpreted	iOS, Android e Windows Phone
Power Apps	Modelação + XML	Pago	Sim	Plataforma / Product & Service	Microsoft	MDSD	iOS, Android e Windows Phone

*Opções pagas e gratuitas

** Opção de adição de frameworks ou bibliotecas de desenvolvimento web

3.2 Métodos de Avaliação

De forma a tornar o processo de avaliação o mais completo possível é fundamental escolher quais os métodos que serão utilizados para avaliar individualmente cada tecnologia escolhida, em conjunto com as formas e parâmetros para criar comparações legítimas. Existem vários métodos para avaliar e analisar as tecnologias e abordagens de desenvolvimento *mobile*, desde a construção de questionários à criação de casos de estudo para a realização de testes de usabilidade e desempenho, descritos e discutidos na secção 2.2 e 2.3 do capítulo 2. Para clarificar a metodologia escolhida, serão descritos quais as tecnologias escolhidas, os questionários escolhidos e construídos, os casos de testes propostos, e por fim, o caso de estudo a realizar nas tecnologias escolhidas.

3.2.1 Escolha de Tecnologias

Nas secções 2.1 e 3.1.3 do capítulo 2 e 3, foram apresentadas e discutidas diversas abordagens e implementações de desenvolvimento de aplicações *mobile* Cross-Platform. Devido à falta de tempo para realizar uma avaliação a cada uma das tecnologias apresentadas, é necessário selecionar apenas algumas delas. Para isto, foram propostos alguns critérios de inclusão e exclusão considerando apenas tecnologias atualmente utilizadas e evitando tecnologias ultrapassadas, descontinuadas e similares. Desta forma, será possível criar uma amostra significativa e abrangente.

Os critérios de inclusão e exclusão consistiram no seguinte: (i) Tecnologias de livre acesso (licenças gratuitas); (ii) Tecnologias que não estejam dependentes de outras; (iii) Tecnologias com

um constante nível de manutenção e suporte; (iv) Tecnologias populares entre *developers* e com suporte da comunidade; (v) Tecnologias que não sejam centralizadas em aplicações de processamento gráfico intensivo. Sendo estes os critérios propostos, muitas das tecnologias apresentadas foram excluídas pelo motivo de serem muito semelhantes e de terem um comportamento idêntico, assim sendo, as tecnologias selecionadas foram as seguintes:

Tabela 6. Tecnologias selecionadas através dos critérios de inclusão e exclusão

	Linguagem	**Licenças**	**IDE**	**Tipo de Tecnologia**	**Suporte**	**Arquitetura**	**Plataformas**
Angular	HTML, CSS e TypeScript	Grátis	Não	Framework	Open-Source + Google	WebApp (PWA)	iOS, Android e Windows Phone
Vue	HTML, CSS e JavaScript	Grátis	Não	Framework	Open-Source	WebApp (PWA)	iOS, Android e Windows Phone
Cordova	HTML, CSS e JavaScript	MIT	Não	Framework	Open-Source + Apache	Híbrida	iOS, Android e Windows Phone
Ionic	HTML, CSS e TypeScript/ ++**	Flexível*	Não	Plataforma	Open-Source + Ionic	Híbrida	iOS, Android e Windows Phone
Appc Titanium	XML, TSS e JavaScript	Flexível*	Sim	Framework / Plataforma	Appcelerator	Interpreted	iOS e Android
React Native	React	Grátis	Não	Framework	Open-Source + Facebook	Interpreted	iOS e Android
NativeScript	TypeScript ou JavaScript /XML e CSS / ++**	Grátis	Sim	Framework	Open-Source	Interpreted	iOS e Android
Kivy	Python	MIT	Não	Framework	Open-Source + Kivy	Interpreted	iOS, Android e Windows Phone
Flutter	Dart	Grátis	Não	Framework	Open-Source + Google	Cross-Compile	iOS e Android
Xamarin	C#	Flexível*	Sim	Framework / Plataforma	Open-Source + Microsoft	Cross-Compile	iOS, Android e Windows Phone

*Opções pagas e gratuitas

** Opção de adição de frameworks ou bibliotecas de desenvolvimento web

3.2.2 Parâmetros de avaliação

Ao longo da execução dos métodos de avaliação seguidamente propostos, para cada tecnologia escolhida, serão observados vários parâmetros de avaliação. Estes parâmetros de avaliação apresentados na Tabela 7, foram obtidos e selecionados através da realização da revisão de literatura descrita na secção 2 e revisão de literatura extensiva apresentada na secção 3.1, verificando quais os pontos mais importantes numa tecnologia de desenvolvimento *mobile* Cross-Platform. Estes parâmetros estão separados por categorias e sub categorias, que serão importantes para realizar análise completa de cada tecnologia escolhida e, por fim, uma comparação entre tecnologias.

Tabela 7. Parâmetros de avaliação propostos

Categoria	Sub Categoria	Parâmetro
Aplicação	-	Tamanho final da aplicação (pronto para distribuição nas *app stores*)
		Estabilidade da aplicação entre plataformas
	Consumo de Recursos	CPU
		Bateria
		memória (RAM)
	Desempenho	Tempo médio da realização de testes
		Recursos utilizados para renderizar a aplicação
		Tempo médio para renderizar a aplicação
Ambiente	Desenvolvimento	Tamanho final do projeto
		Tempo de compilação
		Suporte da comunidade
		Suporte na resolução e demonstração de erros
		Dificuldades de ambiente e desenvolvimento
		Limitações e restrições da tecnologia
		Ferramentas disponibilizadas para desenvolvimento
		Segurança e Encriptação
	Dependências	Dependências de outras tecnologias
		Ferramentas disponibilizadas para gestão de dependências e aplicações

3.2.3 Questionários

O primeiro método proposto será um questionário ao público alvo, incluindo na análise uma componente quantitativa. Esta componente tem um grande nível de importância, pois irá permitir recolher informações relevantes sobre a utilização das tecnologias escolhidas no dia a dia, por equipas de desenvolvimento de aplicações *mobile*.

O questionário proposto será baseado no questionário realizado por Biørn-Hansen et al.[19], pois esta investigação possuía objetivos semelhantes aos esperados nesta componente (dados de profissionais sobre conhecimento, utilização, questões relacionadas com o desenvolvimento Cross-Platform) e, antes de escolherem os participantes, realizaram testes para assegurar a fiabilidade do questionário. Assim sendo, o questionário será composto por seis perguntas de escolha múltipla, divido em três categorias, conhecimento, interesse e utilização.

- Conhecimento

Q#1: O quão familiarizado está com a seguinte tecnologia?

- Resposta: 1-5
- Escala: (1) Não é familiar. (2) Ligeiramente familiar. (3) Moderadamente familiarizado. (4) Muito familiar. (5) Extremamente familiar.
- Opções pré-definidas: (X) tecnologias selecionadas. (X+1) Nenhuma. (X+2) Outras.

- Interesse

Q#2: Qual o nível do seu interesse em explorar as seguintes tecnologias?

- Resposta: 1-5

- Escala: (1) Nada interessado. (2) Ligeiramente interessado. (3) Moderadamente interessado. (4) Muito interessado. (5) Extremamente interessado.
- Opções pré-definidas: (X) tecnologias selecionadas. (X+1) Nenhuma. (X+2) Outras.

- Utilização

Q#3: Quais das seguintes tecnologias que tem preferência?

- Resposta: Escolha múltipla
- Opções pré-definidas: (X) tecnologias selecionadas. (X+1) Nenhuma. (X+2) Outras.

Q#4: Quais das seguintes tecnologias que utiliza a nível profissional?

- Resposta: Escolha múltipla
- Opções pré-definidas: (X) tecnologias selecionadas. (X+1) Nenhuma. (X+2) Outras.

Q#5: Se existir, qual das seguintes opções pode relacionar com a tecnologia que tem preferência?

- Resposta: Escolha múltipla
- Opções pré-definidas: (i) Perda de desempenho em comparação com aplicações nativas. (ii) Experiência do utilizador óptima (UX). (iii) Opção óptima para criar boas interfaces de utilizador (IU). (iv) Comunidade imatura (demasiado novas/baixa actividade). (v) Difícil de integrar com API's dos dispositivos. (vi) Difícil de testar. (vii) Problemas de segurança. (viii) Boas funcionalidades pré-definidas. (ix) Difícil estrutura de código e projeto. (x) Outros

Q#6: Se existir, qual das seguintes opções pode relacionar com a tecnologia que utiliza a nível profissional?

- Resposta: Escolha múltipla
- Opções pré-definidas: (i) Perda de desempenho em comparação com aplicações nativas. (ii) Experiência do utilizador óptima (UX). (iii) Opção óptima para criar boas interfaces de utilizador (IU). (iv) Comunidade imatura (demasiado novas/baixa actividade). (v) Difícil de integrar com API's dos dispositivos. (vi) Difícil de testar. (vii) Problemas de segurança. (viii) Boas funcionalidades pré-definidas. (ix) Difícil estrutura de código e projeto. (x) Outros

3.2.4 *Benchmarks*

Sendo que a prova de conceito será utilizada para analisar as limitações das tecnologia tanto a nível de aplicação *mobile* como nível de ambiente de construção, torna-se necessário avaliar as características de performance da aplicação num específico hardware. Deste modo, um outro método a ser realizado será o *Benchmarking*, que irá monitorizar o consumo de recursos e características de desempenho durante a execução dos casos de teste descritos na subsecção seguinte. Os parâmetros a monitorizar estão descritos na Tabela 7.

Para a realização dos *benchmarks*, na prova de conceito em Android, serão utilizadas as ferramentas, ADB (Android Debug Bridge) que irá monitorizar os consumos de recursos da prova de conceito num específico hardware, como CPU, memória e utilização de bateria, em intervalos de tempo fornecidos pelas ferramentas do CLI, o Android Profiler será utilizando, quando suportado, para obter uma análise pormenorizada da actividade do CPU, memória, tempos de resposta e tráfego de rede, em conjunto com uma interface gráfica.

Já para a prova de conceito em iOS, será utilizado um conjunto de ferramentas chamado de *Instruments* fornecidas pela Apple no Xcode. Este conjunto de ferramentas irão permitir monitorizar certos componentes de hardware, como CPU, memória e consumo de bateria.

Foram escolhidas estas ferramentas, pois, são fornecidas pelos SDK's dos sistemas Android e iOS, não dependendo de sistemas criados por terceiros, tornando o processo de monitorização do consumo de recursos da prova de conceito mais preciso e com uma menor percentagem de erro. Um outro aspecto que levou a escolha destas ferramentas foi pelo facto de que os sistemas criados por terceiros terem um nível de precisão desconhecido, invalidando assim alguns dados recolhidos.

3.2.5 Prova de conceito

Para realizar os cenários de teste acima descritos, será construída uma prova de conceito, que irá possibilitar uma investigação ao pormenor. Esta prova de conceito ajudará a eliminar possíveis ambiguidades entre as tecnologias, visualizando as suas limitações, permitindo verificar se a tecnologia suporta todos os requisitos necessários e se a linguagem e estrutura de código não dificultam a manutenção e adição de funcionalidades.

Dito isto, a prova de conceito proposta consistirá na construção de uma aplicação móvel em todas as tecnologias selecionadas realizando assim uma análise individual. Através dos dados obtidos pela construção da prova de conceito, será possível realizar uma análise comparativa completa, entre as várias tecnologias de desenvolvimento *mobile* Cross-Platform.

A aplicação terá de ir ao encontro com os seguintes requisitos, de modo a permitir realizar os casos de testes descritos na próxima secção.:

1. **R1:** Janela principal com várias opções de escolha.
2. **R2:** Área para executar um cálculo matemático com várias iterações, expondo o valor final e com o tempo total de processamento.
3. **R3:** Área para comunicação com servidor, consumindo vários dados obtidos por este.
4. **R4:** Área para realizar acesso as API's fornecidas pelo sistema operativo e dispositivo.
5. **R5:** Área para gerar vários botões e janelas.

3.2.6 Caso de Teste

Para construir uma análise individual de cada tecnologia escolhida, é necessário realizar métodos de teste, de forma a obter dados significativos sobre o desempenho da aplicação, o consumo de recursos e usabilidade. Este método será executado sobre a prova de conceito realizada em cada tecnologia. Este caso de teste consiste em dois cenários de teste controlados, sendo executados um número definido de vezes e isolados, evitando configurações que dependam de fatores externos, tais como a qualidade da rede, entre outros.

O primeiro cenário será um cálculo matemático (semelhante ao utilizado no artigo de Delia et. al [37], apresentado na Figura 47) incluindo várias iterações, avaliando assim a velocidade de processamento e o tempo de execução necessário para efetuar cálculos intensivos, sempre monitorizando o consumo de recursos. Através do primeiro cenário, é possível obter dados significativos sobre como a tecnologia processa os dados e utiliza a memória para guardá-los.

$$serie = \sum_{j=1}^{5} \sum_{k=1}^{100000} (\log_2(k) + \frac{3k}{2j} + \sqrt{k} + k^{j-1})$$

Figura 47. Cálculo matemático utilizado no artigo de Delia et. al [37], de forma a intensificar o consumo de recursos

O segundo cenário consistirá numa lista de passos que serão realizados sequencialmente. Este cenário será capaz de exigir o máximo da tecnologia, monitorizando o consumo de recursos e estabilidade da aplicação. A lista de passos a seguir será baseada na investigação de Dorfer et al.[39], isto porque, conseguiu obter dados tecnológicos significativos para efetuar uma comparação entre tecnologias.

A lista de passos será a seguinte:

1. **Passo 1:** Iniciar a ferramenta de monitorização
2. **Passo 2:** Clicar no botão *home screen* e esperar 2 segundos
3. **Passo 3:** Abrir a aplicação
4. **Passo 4:** Realizar os seguintes testes:
   (a) **Passo 4.1:** Abrir janela de CPU intensivo, esperar pelo resultado e depois fechar.
   (b) **Passo 4.2:** Abrir janela de consumo de API's do SO, esperar pelos resultados e depois fechar.
   (c) **Passo 4.3:** Abrir janela de consumo de recursos, esperar pelo resultado e depois fechar.
5. **Passo 5:** Fechar aplicação
6. **Passo 1:** Terminar a monitorização e fechar a ferramenta

4 Resultados

Neste capítulo apresentamos os resultados obtidos da execução da metodologia anteriormente apresentada, de forma qualitativa e quantitativa. Deste modo, será exposto o ambiente de avaliação utilizado para a implementação da prova de conceito, os casos de teste e os *benchmarks* apresentados no capítulo anterior. De seguida, serão apresentados os resultados obtidos da execução da metodologia, para cada tecnologia escolhida, em conjunto com os problemas encontrados durante a fase de desenvolvimento e testes.

4.1 Questionários

Com o propósito de incluir uma componente quantitativa sobre dados e informações relevantes da utilização das tecnologias escolhidas no dia a dia e por equipas de desenvolvimento de aplicações mobile, foi realizado um questionário baseado no artigo realizado por Biørn-Hansen et al.[19], apresentado no capítulo 3, secção 3.2.3.

Este questionário foi realizado através da plataforma *Profilic* [91], que permite realizar um questionário a nível mundial, apenas indicando qual o público-alvo. Para este questionário o público-alvo foram programadores de aplicações mobile, tanto a nível empresarial como pessoal, contando com número inicial de 70 participantes. Contudo, foram realizadas várias perguntas para verificar a elegibilidade dos participantes, de forma a excluir participantes que no fundo não conheciam estas tecnologias ou não tinham qualquer conhecimento sobre o desenvolvimento de aplicações mobile, fazendo com que o número final de participantes descesse para 41.

O questionário possuía seis perguntas principais, divididas em três categorias. A categoria Conhecimento para recolher dados sobre o conhecimento, Interesse para recolher dados sobre o interesse nas tecnologias e Utilização para recolher dados sobre a utilização das tecnologias preferidas. A análise dos resultados foi feita separando cada pergunta e construindo gráficos individuais por tecnologia, no entanto para sumarizar os dados, foi realizado um gráfico geral para cada pergunta. Assim sendo, os resultados foram os seguintes:

O quão familiarizado está com a seguinte tecnologia?

A primeira pergunta pertence à categoria de Conhecimento, no qual pretende recolher dados sobre o conhecimento dos participantes das tecnologias escolhidas, avaliando-as numa escala de 1 a 5 valores. De seguida, estes dados foram transformados no gráfico representado na Figura 48, identificando a tecnologia a cores, a escala de 1 a 5 e a quantidade de participantes que avaliaram cada tecnologia.

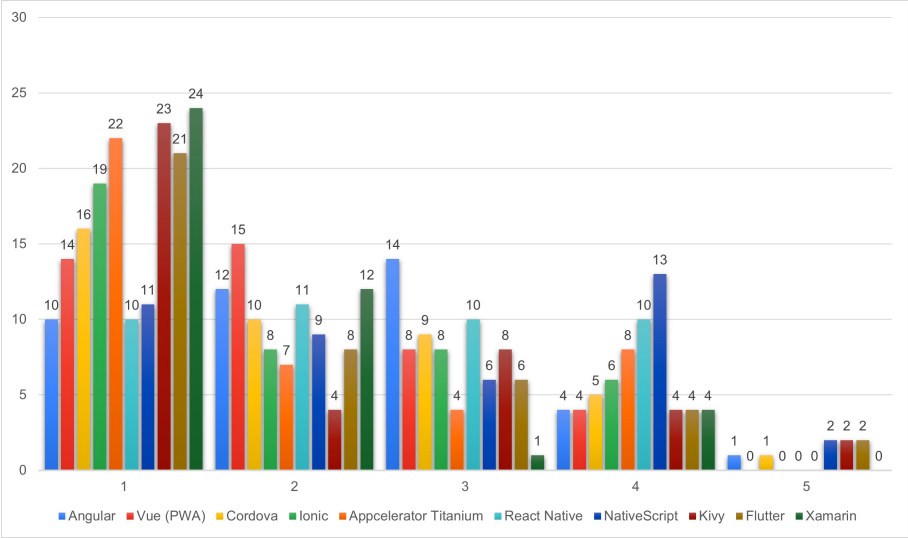

Figura 48. Dados recolhidos sobre a primeira questão: O quão familiarizado está com a seguinte tecnologia?

Qual o seu nível de interesse em explorar as seguintes tecnologias?

A segunda pergunta pertence à categoria de Interesse, que pretende recolher dados sobre o interesse que os participantes têm em explorar as tecnologias apresentadas, avaliando-as numa escala de 1 a 5 valores. De seguida, estes dados foram transformados no gráfico representado na Figura 49, identificando a tecnologia, a escala de 1 a 5 a cores e a quantidade de participantes que avaliaram cada tecnologia.

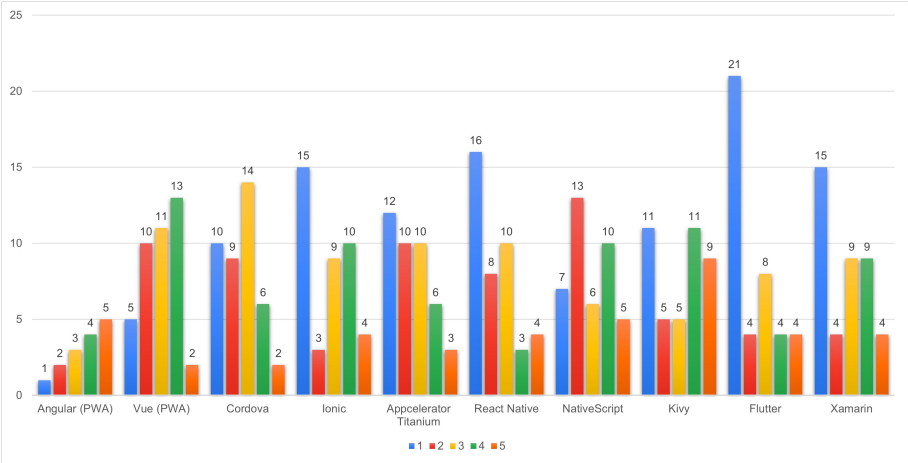

Figura 49. Dados recolhidos sobre a segunda questão: Qual o seu nível de interesse em explorar as seguintes tecnologias?

Quais das seguintes tecnologias que tem preferência?

A terceira pergunta pertence à categoria de Utilização, que pretende recolher dados sobre as tecnologias preferidas e mais utilizadas por cada participante, numa resposta de escolha múltipla. De seguida, estes dados foram transformados no gráfico representado na Figura 50, identificando a tecnologia e o número de vezes, em percentagem, que cada tecnologia foi selecionada.

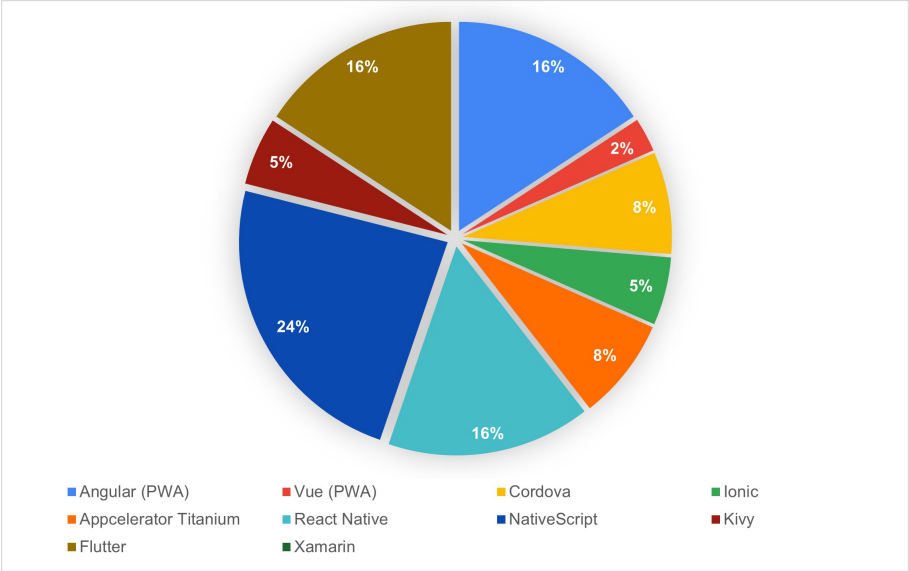

Figura 50. Dados recolhidos sobre a terceira questão: Quais das seguintes tecnologias que tem preferência?

Se existir, qual das seguintes opções pode se relacionar com a tecnologia que tem preferência?

A quarta pergunta pertence à categoria de Utilização, que pretende recolher dados sobre alguns aspetos positivos ou negativos de cada tecnologia anteriormente escolhidas pelo participante. De seguida, estes dados foram transformados no gráfico representado na Figura 51, identificando a tecnologia e as opções escolhidas pelos participantes sobre as tecnologias escolhidas anteriormente.

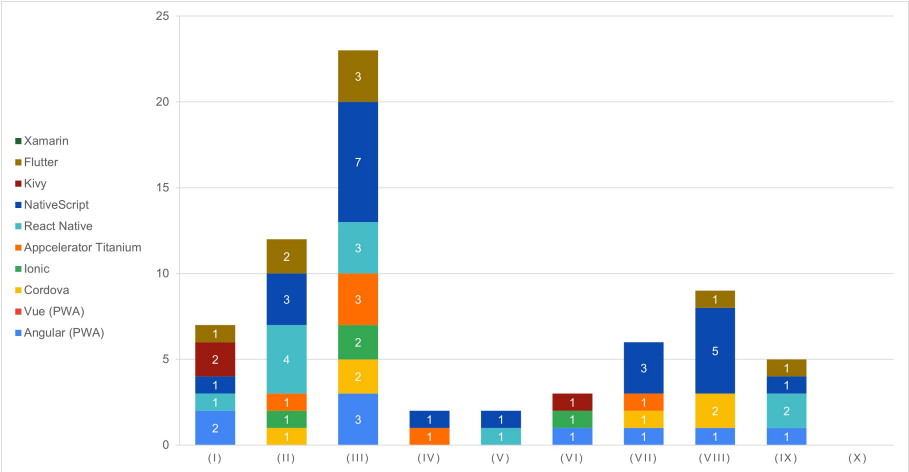

Figura 51. Dados recolhidos sobre a quarta questão: Se existir, qual das seguintes opções pode relacionar com a tecnologia que tem preferência?

Opções predefinidas no questionário: (i) Perda de desempenho em comparação com aplicações nativas. (ii) Experiência do utilizador óptima (UX). (iii) Opção óptima para criar boas interfaces de utilizador (IU). (iv) Comunidade imatura (demasiado novas/baixa actividade). (v) Difícil de integrar com API's dos dispositivos. (vi) Difícil de testar. (vii) Problemas de segurança. (viii) Boas funcionalidades pré-definidas. (ix) Difícil estrutura de código e projeto. (x) Outros

Quais das seguintes tecnologias utiliza a nível profissional?

A quinta pergunta pertence à categoria de Utilização, onde pretende recolher dados sobre as tecnologias utilizadas a nível profissional pelos participantes, numa resposta de escolha múltipla. De seguida, estes dados foram transformados no gráfico representado na Figura 52, identificando a tecnologia e o número de vezes, em percentagem, que cada tecnologia foi selecionada pelos participantes.

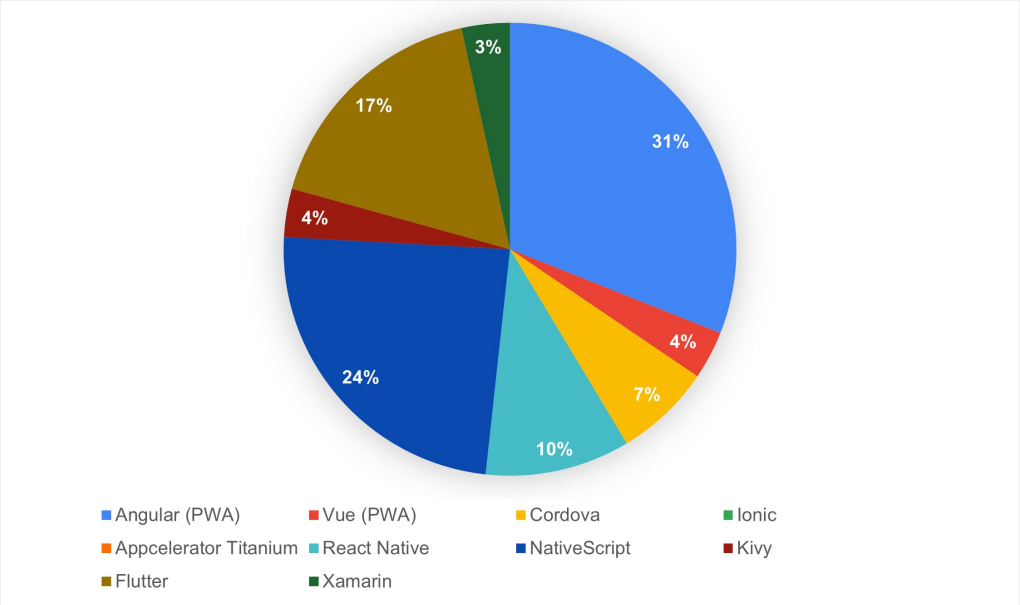

Figura 52. Dados recolhidos sobre a quinta questão: Quais das seguintes tecnologias utiliza a nível profissional?

Se existir, qual das seguintes opções pode se relacionar com a tecnologia que utiliza a nível profissional?

A sexta e última pergunta pertence à categoria de Utilização, onde pretende recolher dados sobre alguns aspetos positivos ou negativos de cada tecnologia anteriormente escolhidas pelo participante. De seguida, estes dados foram transformados no gráfico representado na Figura 53, identificando a tecnologia e as opções escolhidas pelos participantes sobre as tecnologias escolhidas anteriormente.

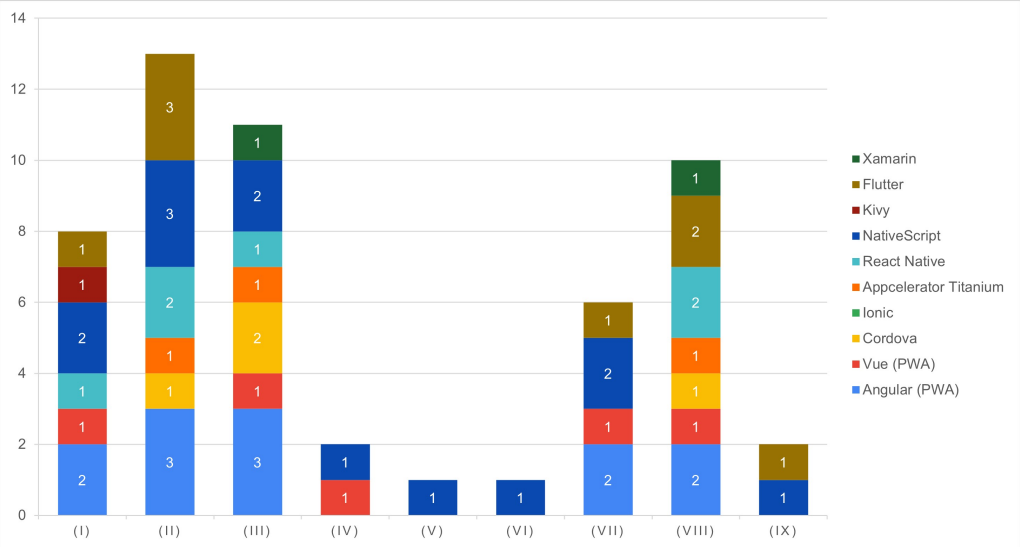

Figura 53. Dados recolhidos sobre a sexta questão: Se existir, qual das seguintes opções pode relacionar com a tecnologia que utiliza a nível profissional?

Opções predefinidas no questionário: (i) Perda de desempenho em comparação com aplicações nativas. (ii) Experiência do utilizador óptima (UX). (iii) Opção óptima para criar boas interfaces de utilizador (IU). (iv) Comunidade imatura (demasiado novas/baixa actividade). (v) Difícil de integrar com API's dos dispositivos. (vi) Difícil de testar. (vii) Problemas de segurança. (viii) Boas funcionalidades pré-definidas. (ix) Difícil estrutura de código e projeto. (x) Outros

4.2 Ambiente de Avaliação

Após a análise de uma componente quantitativa focada nos *developers* e a utilização das tecnologias mobile no dia a dia e por equipas de desenvolvimento, foi decidido avaliar as tecnologias a nível do implementação e desempenho. Para isso foi criado um ambiente de avaliação para ser utilizando durante o desenvolvimento da prova de conceito, realização de testes e *benchmarks*.

Para criar um ambiente estável durante todo o processo de desenvolvimento, correção de erros e testes, foi utilizado sempre o mesmo ambiente descrito abaixo, minimizando assim os impactos de fatores externos e internos à prova de conceito. Dito isto, segue-se o ambiente utilizado, as ferramentas *benchmarks* e as principais características da prova de conceito.

4.2.1 Dispositivos utilizados

De forma a manter uma consistência no desenvolvimento e testes das provas de conceito, foram utilizados sempre os mesmos dispositivos, diminuindo a margem de erro entre cada tecnologia utilizada e permitindo comparar os resultados obtidos sem ser necessário ter em conta as diferenças entre dispositivos. Dito isto, para o processo de desenvolvimento, correção de erros e testes foram utilizados os seguintes dispositivos:

Tabela 8. Especificação dos dispositivos móveis utilizados

Detalhe	**Huawei mate 20 lite**	**iPhone 7**
Sistema Operativo	Android 10	iOS 14
RAM	4 GB	2 GB
CPU	Kirin 710 (8 core, 2Ghz)	Apple A10 Fusion (4 Core, 2Ghz)
memória Interna	64 GB	64 GB
Bateria*	3750 mAh (100%)	1960 mAh (100%)
Resolução de Ecrã	2340x1080 (60 fps)	750 x 1334 (60 fps)
Tamanho de Ecrã	6.3	4.7
Rede (Wi-Fi)**	600/100 Mbps	450/50 Mbps

* Nenhuma aplicação a correr em background, apenas as aplicações essenciais ao SO

** Velocidade máxima de download / upload

Tabela 9. Especificação dos dispositivos utilizados

Detalhe	**Computador para o desenvolvimento**	**Servidor**
Sistema Operativo	Big Sur 11.4	Raspberry Pi OS Lite
RAM	16 GB (3733 MHz LPDDR4X)	1 GB
CPU	Intel Core i5 (4 Core, 2GHz)	Broadcom BCM2837 (4 Core, 1.2GHz)
GPU	Grafico Intel Iris Plus (1536 MB)	-
Rede (Wi-Fi)	1 Gbps	100 MBps

4.2.2 Servidor NodeJs

De modo a minimizar impactos e dependências externas à prova de conceito, durante os testes que necessitam de consumir dados de serviços externos, optou-se por utilizar um servidor local utilizando um Raspberry Pi, que retorna objetos JSON, em vez de realizar pedidos HTTP a servidores externos. Esta decisão foi uma forma de controlar o estado de rede e a precisão e consistência da resposta do servidor, pelo motivo que não é possível verificar a contínua qualidade de rede interna e externa. Desta forma foi possível manter um certo nível consistência entre pedidos HTTP e entre provas de conceito.

Para a construção do servidor local foi utilizado o NodeJs em conjunto com o Express, devido ao seu rápido desenvolvimento e consistência entre respostas ao cliente. Este tem como principal função resolver dois pedidos GET, em que ambos lêem um ficheiro JSON com 100 objetos distintos.

4.2.3 Ferramentas *Benchmarks*

Como foi apresentado no capítulo 3, foram utilizas as ferramentas Android Profiler e XCode *Instruments* para captar o consumo de recursos durante os casos de teste. No entanto, não foi possível utilizar estas ferramentas para as aplicações PWA, pelo facto da tecnologia não produzir um ficheiro instalável no dispositivo Android e iOS, e por ser executada inteiramente num *browser*. Para resolver este problema, foi escolhida a ferramenta *Google Chrome DevTools* para monitorizar as aplicações PWA, que permite verificar o consumo de recursos, como CPU, RAM, entre outros, sem comprometer a aplicação e os seus resultados, tendo um grande suporte pela Google [28].

Assim sendo, é apresentado abaixo como é que estas ferramentas foram utilizadas durante os casos de teste.

Android Profiler

O Android Profiler fornece várias ferramentas para encontrar e visualizar vários problemas, desde problemas de desempenho a alocações de memória durante a execução da aplicação. Para monitorizar o consumo de recursos de cada tecnologia durante a execução dos testes foram utilizadas as ferramentas: *CPU Profiler*, que monitoriza o consumo de CPU; *Memory Profiler*, para controlar a memória RAM; *Network Profiler* para monitorizar o tráfico de rede; *Energy Profiler*, que permite supervisionar o consumo de bateria. Para todas estas ferramentas é permitido filtrar por processo ou aplicação que está a ser executada, mostrando os dados em tempo real, como mostra a Figura 54.

Figura 54. Monitorização do consumo de recursos utilizando a ferramenta Android Profiler. Segundo caso de teste na tecnologia React Native

Após a criação do ficheiro instalável para cada uma das tecnologias analisadas em modo de produção, seguiu-se os seguintes passos:

1. Iniciar o Android Studio e escolher a opção *Profile or debug APK*.
2. Escolher o ficheiro instalável criado pela tecnologia
3. Escolher o dispositivo-alvo para executar a aplicação
4. Iniciar a monitorização e realizar o primeiro teste
5. Encerrar a monitorização e verificar os tempos de resposta e o consumo de recursos durante o teste.

6. Iniciar a monitorização e realizar o segundo teste
7. Encerrar a monitorização e verificar os tempos de resposta e o consumo de recursos durante o teste.

Google Chrome DevTools

As ferramentas Performance e Performance Monitor do Chrome DevTools [28] fornecidas pela Google, permitem verificar o consumo de recursos em tempo real para aplicações PWA e websites em qualquer dispositivo móvel. De forma a associar a ferramenta de monitorização ao dispositivo móvel, foi utilizada a ferramenta DevTools Devices, que permite associar remotamente o DevTools com a dispositivo-alvo [16]. As figuras 55 e 56 mostram um exemplo da sua utilização.

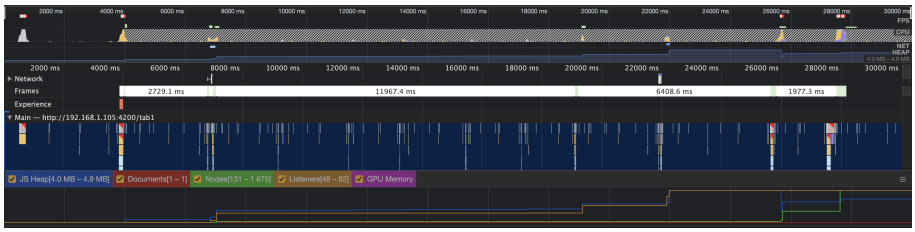

Figura 55. Vista geral da monitorização do consumo de recursos utilizando a ferramenta Performance do Chrome DevTools. Segundo caso de teste na tecnologia Angular

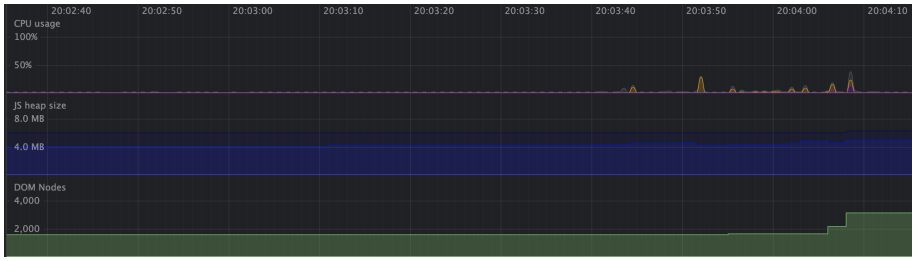

Figura 56. Vista pormenorizada da monitorização do consumo de recursos de cpu e memória utilizando a ferramenta Performance Monitor do Chrome DevTools. Segundo caso de teste na tecnologia Vue

Após a tecnologia gerar o servidor local em modo de produção, seguiu-se os seguintes passos:

1. Iniciar o Google Chrome DevTools Devices.
2. Iniciar a aplicação no dispositivo-alvo.
3. Iniciar as ferramentas Performance e Performance Monitor do Google Chrome DevTools
4. Iniciar a monitorização e realizar o primeiro teste
5. Encerrar a monitorização e verificar os tempos de resposta e o consumo de recursos durante o teste.
6. Iniciar a monitorização e realizar o segundo teste
7. Encerrar a monitorização e verificar os tempos de resposta e o consumo de recursos durante o teste.

XCode Instruments

O XCode Instruments possui uma grande variedade de ferramentas, como por exemplo, *Activity monitor*, *Allocations*, *Energy log*, *Network*, *Leaks* entre outros. Para monitorizar o consumo de recursos durante a execução dos testes foram utilizadas as ferramentas: *Activity monitor*, que permitem verificar o utilização do CPU, RAM e Network; *Energy Log e CPU Activity Log* que monitorizam o consumo de bateria. Para todas estas ferramentas é permitido filtrar por processo ou aplicação que está a ser executada, mostrando os dados em tempo real, como demonstra a Figura 57.

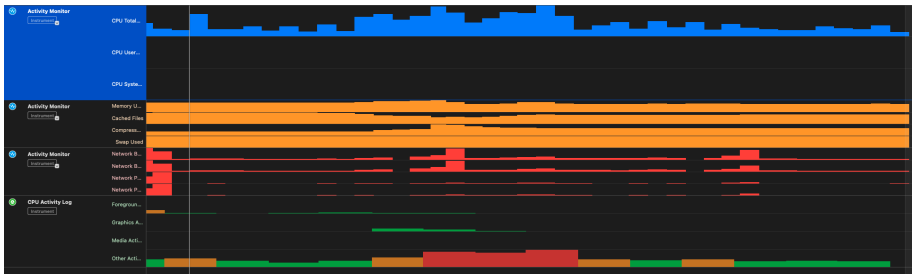

Figura 57. Monitorização do consumo de recursos utilizando as ferramenta Activity monitor, Energy Log e CPU Activity Log do XCode Instruments. Segundo caso de teste na tecnologia Ionic

Após a tecnologia gerar o projeto XCode em modo de produção, seguiu-se os seguintes passos:

1. Iniciar o XCode e abrir projeto criado pela tecnologia
2. Escolher o dispositivo-alvo para executar a aplicação
3. Iniciar a ferramenta Instruments
4. Escolher a ferramenta Activity Monitor com três janelas (CPU, RAM e Network), o Energy Log
5. Escolher a aplicação que será o Target Process
6. Iniciar a monitorização e realizar o primeiro teste
7. Encerrar a monitorização e verificar os tempos de resposta e o consumo de recursos durante o teste.
8. Iniciar a monitorização e realizar o segundo teste
9. Encerrar a monitorização e verificar os tempos de resposta e o consumo de recursos durante o teste.

4.2.4 Prova de conceito

A fim de realizar os cenários de teste propostos na secção 3.2.6, do capítulo 3 foi realizada uma prova de conceito para cada tecnologia escolhida, que permitiu-nos recolher métricas como consumo de recursos, custo de desenvolvimento, problemas encontrados, entre outros parâmetros representados na Tabela 7.

A realização da prova de conceito não só permitiu recolher as várias métricas acima descritas, mas também foram construídas de forma a serem úteis para a realização dos casos de teste e *benchmarks*, pois estes foram executados sobre a aplicação resultante da prova de conceito. Desta

forma, conseguiu-se avaliar a tecnologia não só a nível de custo de desenvolvimento mas também a nível de desempenho utilizando a mesma solução.

Dito isto, e respeitando os requisitos impostos no capítulo anterior, secção 3.2.5, a prova de conceito foi realizada da seguinte forma.

- Barra de navegação inferior contendo três página (**R1**)
- Primeira página
  - Botão que permite executar o algoritmo apresentado no primeiro cenário. (**R2**)
  - Texto que apresenta o resultado do algoritmo.
- Segunda página
  - Área de disposição em pilha.
  - Botão que realiza dois pedidos HTTP assíncronos ao servidor local, e quando obtém resposta, escreve a resposta num ficheiro de texto no dispositivo.(**R3**)
  - Botão que cria 500 *views* dinamicamente e insere-os na área. (**R5**)
  - Botão que cria 500 botões dinamicamente e insere-os na área. (**R5**)
- Terceira página
  - Botão que realiza um acesso à câmara do dispositivo e devolve o caminho da foto tirada. (**R4**)
  - Botão que realiza um acesso à galeria do dispositivo e devolve o caminho da imagem escolhida.(**R4**)
  - Botão que realiza um acesso ao serviço de localização do dispositivo.(**R4**)
  - Botão que realiza um acesso ao calendário do dispositivo e devolve os eventos do calendário .(**R4**)
  - Botão que realiza um acesso aos contactos do dispositivo e devolve o nome de todos os contactos.(**R4**)

Foi possível construir a prova de conceito para grande parte das tecnologias escolhidas, porém não foi possível construir na totalidade para as tecnologias PWA, devido a estas não possuírem acesso às API's do sistema, como discutido na secção 2.1.2.1 do capítulo 2. Por fim, não foi possível construir a prova de conceito para a tecnologia *Kivy*, devido a esta tecnologia gerar muitos problemas durante a fase de configuração e desenvolvimento, fazendo com que não fosse possível ser executada num dispositivo Android ou iOS através de um computador Windows ou MacOS. Após vários testes e tentativas, verificou-se que só é possível construir e executar a aplicação através da utilização de uma máquina *Linux* ou no *Google Colab*, dificultando assim as fases de desenvolvimento e testes. Devido a estes problemas e à falta de suporte para computadores com o sistema MacOS, foi tomada uma decisão que esta tecnologia seria excluída.

4.3 Prova de conceito e avaliação das tecnologias

Para cada tecnologia escolhida foi realizada uma prova de conceito, a fim de obter dados significativos. Todos os dados apresentados nas secções abaixo foram obtidos durante e depois do desenvolvimento da prova de conceito. Estes dados serão discutidos ao pormenor no capítulo 5,

a Discussão. Dito isto, para cada tecnologia recolheu-se dados sobre a forma de desenvolvimento, parâmetros apresentados na Tabela 7, problemas encontrados e capturas de ecrã da aplicação final.

4.3.1 Angular (PWA)

A prova de conceito utilizando a tecnologia Angular foi realizada com as versões mais recentes até à data 20 de julho de 2021. As versões utilizadas foram as seguintes: Angular: 12.1; Rxjs: 6.6.7; TypeScript: 4.3.4;

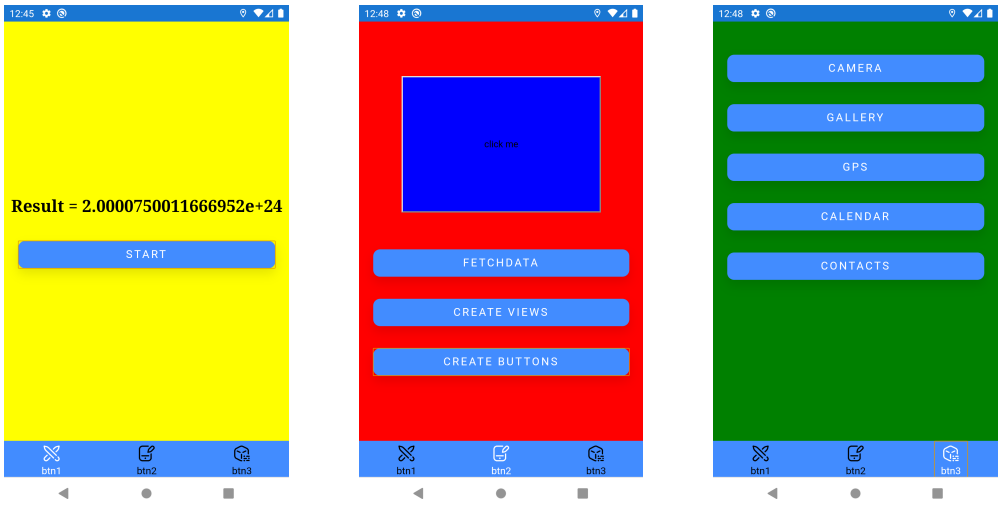

Figura 58. Prova de conceito com a tecnologia Angular

A tecnologia Angular, permite-nos criar uma aplicação web responsiva e progressiva [48] em que possa ser transformada numa aplicação PWA. Esta tecnologia utiliza a linguagem TypeScript, permitindo assim utilizar conceitos de POO, criando automaticamente ficheiros de testes unitários [55].

A sua estrutura de código está dividida principalmente em Páginas e Componentes, em que para cada componente existe um conjunto de ficheiros:

- Routing: Ficheiro que fornece e cria as rotas ou subrotas da aplicação
- HTML: Ficheiro que cria a estrutura de UI do Componente ou Página
- Css ou SASS: Ficheiro individual ou geral para os estilos
- Spec: Ficheiro de testes unitários
- Component: Ficheiro que contém a funcionalidade do Componente ou Página
- Módulo: Ficheiro que agrega todos os ficheiros anteriores em um único módulo

Por fim, o Angular possibilita a utilização de Services, em que são ficheiros com funcionalidades insoladas, que podem ser injetados em Módulos de Componentes e Páginas, minimizando a replicação de código. Estes Services são geralmente serviços HTTP, Proxy's, Logger, entre outros, facilitando a injeção de dependências dos Módulos.

Parâmetros de avaliação

Os parâmetros de avaliação propostos na Tabela 7 foram avaliados com sucesso durante e após o desenvolvimento da prova de conceito utilizando a tecnologia Angular. Estes dados estão apresentados na Tabela 10, identificados pelo seu parâmetro de avaliação.

Tabela 10. Parâmetros de avaliação propostos sobre a tecnologia Angular

Parâmetro	Valor
Tamanho final da aplicação (pronto para distribuição nas *app stores*)	Não é possível obter este parâmetro devido ao facto de, não ser possível distribuir a aplicação nas *app stores*
Estabilidade da aplicação entre plataformas	Estável entre plataformas
Tempo médio da realização de testes	Primeiro Cenário: 28 segundos Segundo Cenário: 30 segundos
Recursos utilizados para renderizar a aplicação	CPU: 4% a 16% (Android) / 1 % a 10,1 % (iOS) RAM: 4 MB a 5 MB (Android) / 9,91 MB a 11,03 MB (iOS) Bateria: Low (Android) / Medium (iOS)
Tempo médio para renderizar a aplicação	Android:1,65 segundos iOS: 1,10 segundos
Tamanho final do projeto	395 405 994 bytes (494,4 MB em disco)
Tempo de compilação	13 segundos (média)
Suporte da comunidade	Excelente suporte da comunidade, tanto na construção de UI como na resolução de erros.
Suporte na resolução e demonstração de erros	Bom suporte na demonstração de erros.
Dificuldades de ambiente e desenvolvimento	Estrutura de codigo um pouco complexa
Limitações e restrições da tecnologia	Não permite aceder às API's dos SO's, sendo apenas possível aceder ao GPS.
Ferramentas disponibilizadas para desenvolvimento	LiveView: Permite recompilar um ficheiro sempre que se realize uma alteração, não sendo necessário recompilar toda a aplicação Browser *Plug-in*: Permite desconstruir a aplicação e encontrar erros
Segurança e Encriptação	Permite a utilização de https, mas para tipos de encriptação é necessário ser instalado separadamente
Dependências de outras tecnologias	Angular; Rxjs; TypeScript;
Ferramentas disponibilizadas para gestão de dependências e aplicações	Ferramentas CLI

Considerações e problemas

Durante a fase de desenvolvimento encontrou-se muitos problemas, principalmente em descobrir quais as boas práticas da tecnologias e a sua estrutura de implementação. Inicialmente a prova de conceito encontrava-se um pouco instável devido a falta de conhecimento sobre as diferenças entre componentes, páginas, módulos, entre outros, mas consoante o desenvolvimento foram adquiridos conhecimentos sobre a sua estrutura e decidiu-se reconstruir a prova de conceito. Após a reconstrução da prova de conceito, a aplicação ficou fácil de manter e escalar, pelo motivo de utilizar corretamente os componentes, páginas, módulos, routers e principalmente os Services. Para um *developer* com pouco conhecimento de tecnologias Web e TypeScript poderá encontrar vários problemas durante o desenvolvimento, principalmente na utilização dos service-workers.

4.3.2 Vue (PWA)

A prova de conceito utilizando a tecnologia Vue foi realizada com as versões mais recentes até à data 20 de julho de 2021. As versões utilizadas foram as seguintes: Vue: 3.1; Rxjs: 6.6.7; TypeScript: 4.1.5;

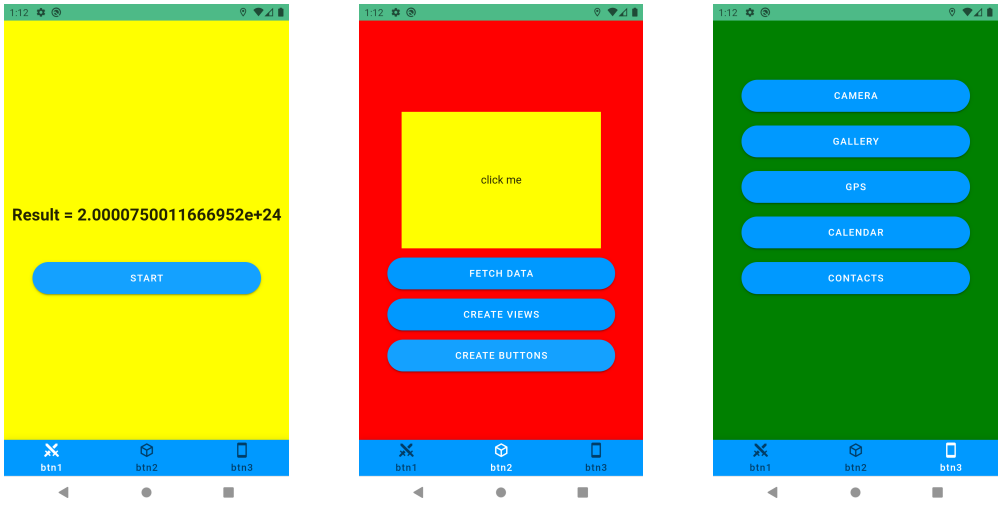

Figura 59. Prova de conceito com a tecnologia Vue

A tecnologia Vue consiste numa framework que permite-nos criar uma aplicação web muito rapidamente, com o intuito de organizar e simplificar o desenvolvimento, centrando-se na renderização declarativa e na composição de componentes [48]. Esta tecnologia permite-nos escolher entre as linguagens JavaScript e TypeScript, no qual a estrutura de código se adequa consoante a linguagem e possibilita a escolha de uma aplicação PWA, facilitando o processo de instalação e injeção de dependências.

Para esta prova de conceito foi utilizada a linguagem TypeScript, permitindo utilizar o conceito de POO e dividindo a estrutura de código em Componentes, router, views e ServiceWorker. A estrutura de código está divida em várias partes:

- Components: Local onde contém vários ficheiros .vue que se comportaram como componentes
- Views: Local onde encontra-se ficheiros .vue que se comportaram como páginas
- Router: Ficheiro individual onde se registam as rotas e páginas
- registerServiceWorker: Ficheiro para realizar o registo e gestão do service-worker.

Apartir desta estrutura de código, torna-se muito fácil de criar uma aplicação e mantê-la, no entanto, em certos casos é necessários utilizar uma *store*, como o VueX, para manter um estado global da aplicação devido a este ser estruturado por uma árvore de componentes.

Parâmetros de avaliação

Os parâmetros de avaliação propostos na Tabela 7 foram avaliados com sucesso durante e após o desenvolvimento da prova de conceito utilizando a tecnologia Vue. Estes dados estão apresentados na Tabela 11, identificados pelo seu parâmetro de avaliação.

Tabela 11. Parâmetros de avaliação propostos sobre a tecnologia Vue

Parâmetro	Valor
Tamanho final da aplicação (pronto para distribuição nas *app stores*)	Não é possível obter este parâmetro devido ao facto de, não ser possível distribuir a aplicação nas *app stores*
Estabilidade da aplicação entre plataformas	Estável entre plataformas
Tempo médio da realização de testes	Primeiro Cenário: 19 segundos Segundo Cenário: 24 segundos
Recursos utilizados para renderizar a aplicação	CPU: 3% a 18% (Android) / 4,3 % a 5,6 % (iOS) RAM: 3 MB a 4,3 MB (Android) / 10,09 MB a 10,74 MB (iOS) Bateria: Low (Android) / Low (iOS)
Tempo médio para renderizar a aplicação	Android:1,56 segundos iOS: 0,89 segundos
Tamanho final do projeto	395 405 994 bytes (494,4 MB em disco)
Tempo de compilação	12,80 segundos (média)
Suporte da comunidade	Excelente suporte da comunidade. Comunidade muito ativa, no entanto, grande parte da comunidade utiliza a segunda versão do Vue
Suporte na resolução e demonstração de erros	Bom suporte, tanto em runtime como durante a compilação.
Dificuldades de ambiente e desenvolvimento	Nenhuma dificuldade encontrada
Limitações e restrições da tecnologia	Não permite aceder às API's dos SO's, sendo apenas possível aceder ao GPS.
Ferramentas disponibilizadas para desenvolvimento	LiveView: Permite recompilar um ficheiro sempre que se realize uma alteração, não sendo necessário recompilar toda a aplicação Browser *Plug-in*: Permite desconstruir a aplicação e encontrar erros
Segurança e Encriptação	Permite a utilização de https
Dependências de outras tecnologias	NodeJs; TypeScript (Opcional);
Ferramentas disponibilizadas para gestão de dependências e aplicações	Ferramentas CLI, que permite a opção de instalar um servidor local para gerir e criar aplicações do vue

Considerações e problemas

A construção da prova de conceito nesta tecnologia foi muito rápida e simples. A estrutura de código é muito objetiva, apenas é necessário ter em conta a forma como os componentes funcionam e as funcionalidades que estes trazem. A prova de conceito foi inicialmente construída utilizando o Vue 2, mas como foi decidido que seria utilizado para a prova de conceito as versões mais recentes de cada tecnologia, foi então necessário realizar uma atualização para o Vue 3. Contudo, esta atualização gerou muitos problemas especialmente nas suas dependências e grande parte da resolução de erros fornecidos pela comunidade estavam solucionados utilizando o Vue 2, fazendo com que a resolução de futuros erros fosse mais demorada.

Um ponto importante a considerar é que ao utilizar uma estrutura de código em árvore de componentes, é necessário ter em conta a escalabilidade, isto é, quando vários componentes modificam os mesmos dados, torna-se necessário utilizar uma *store* como o VueX, devido às dependências entre componentes (pai-filho) e a troca de dados constantes entre vários componentes. Desta forma, os dados que serão modificados em vários componentes estarão guardados num único local, diminuindo assim a complexidade de estrutura de código.

4.3.3 Cordova

A prova de conceito utilizando a tecnologia Cordova foi realizada com as versões mais recentes até à data 20 de julho de 2021. As versões utilizadas foram as seguintes: Cordova: 10; Java: 1.8; Xcode 12.5;

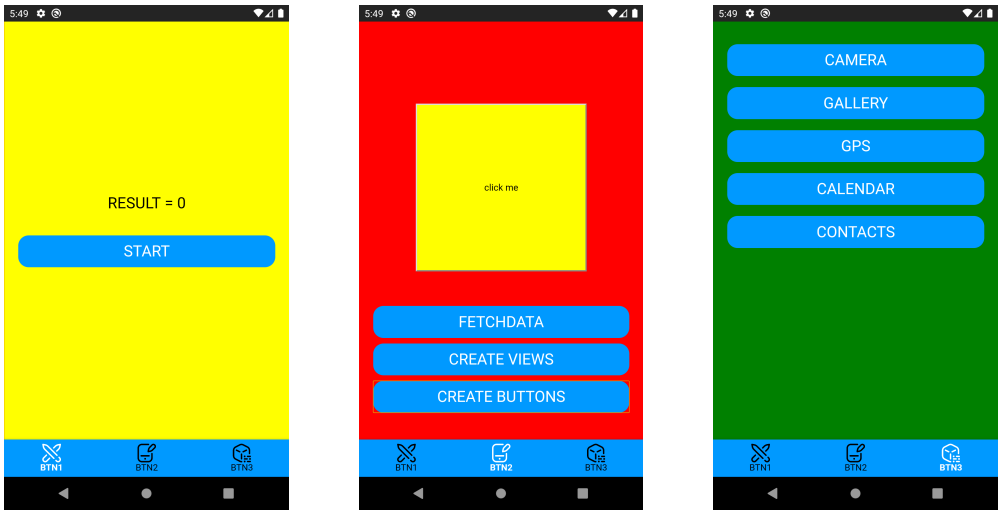

Figura 60. Prova de conceito com a tecnologia Cordova

A tecnologia Cordova consiste numa framework de desenvolvimento híbrido, que permite utilizar linguagens web, HTML5, CSS3 e JavaScript para a construção de aplicações Cross-Platform [49]. As aplicações são encapsuladas numa WebView de forma a serem compatíveis com as várias plataformas e SO's. Esta tecnologia possui uma estrutura de código muito simples, no qual o seu comportamento e forma de desenvolvimento é igual a uma aplicação web, tornando-se fácil transportar uma aplicação web para uma aplicação híbrida.

Para a construção da prova de conceito na tecnologia Cordova, foi utilizado uma arquitetura MVC em conjunto do JQuery Mobile, proposto pela tecnologia, facilitando assim a criação das vistas e as chamadas às API's do dispositivo. Desta forma, o código está dividido em 3 partes:

- Views: Local onde se encontram ficheiros HTML que formam a estrutura das páginas
- Css: Local onde se encontram ficheiros CSS que fornecem estilos as páginas
- JS: Local onde se encontram ficheiros de JavaScript que irão funcionar como *model*, *controller* ou funcionalidades das vistas.

Com a inclusão do JQuery Mobile tornou o de desenvolvimento mais simples, porque este já oferece funcionalidades para dispositivos móveis, ficheiros CSS com adaptações de temas e tamanhos de ecrã. Para aceder às funcionalidades nativas do dispositivo, foi necessário instalar vários *Plug-in's*, cada *Plug-in* para um acesso específico a uma API, desta forma permite incluir apenas as funcionalidades extras necessárias.

Parâmetros de avaliação

Os parâmetros de avaliação propostos na Tabela 7 foram avaliados com sucesso durante e após o desenvolvimento da prova de conceito utilizando a tecnologia Cordova. Estes dados estão apresentados na Tabela 12, identificados pelo seu parâmetro de avaliação.

Tabela 12. Parâmetros de avaliação propostos sobre a tecnologia Cordova

Parâmetro	Valor
Tamanho final da aplicação (pronto para distribuição nas *app stores*)	Android: 2 MB iOS: 2,8 MB
Estabilidade da aplicação entre plataformas	Boa estabilidade entre plataformas
Tempo médio da realização de testes	Primeiro Cenário: 22 segundos Segundo Cenário: 38 segundos
Recursos utilizados para renderizar a aplicação	CPU: 12% a 24% (Android) / 8,1 % a 17,1 % (iOS) RAM: 30 MB a 163,9MB (Android) / 8,23 MB a 8,94 MB (iOS) Bateria: Medium (Android) / Medium (iOS)
Tempo médio para renderizar a aplicação	Android:1,8 segundos iOS: 0,87 segundos
Tamanho final do projeto	51 433 062 bytes (69,8 MB em disco)
Tempo de compilação	10 segundos (média)
Suporte da comunidade	Pouco suporte da comunidade, sendo um difícil resolver erros sobre a sintaxe ou assuntos sobre a tecnologia. Contudo, tem bastante suporte sobre assuntos de UI
Suporte na resolução e demonstração de erros	Bom suporte na resolução de erros, no qual acontecem durante a compilação. No entanto caso o erro aconteça durante a utilização da aplicação, torna-se difícil saber onde o erro aconteceu devido a este apontar para o local do ficheiro compilado.
Dificuldades de ambiente e desenvolvimento	Utilização de uma versão do Java mais antiga que as outras tecnologias. Para realizar testes no dispositivo android ou iOS é necessário utilizar as ferramentas do browser, dificultando assim todo processo.
Limitações e restrições da tecnologia	Apresenta os mesmos limites que uma tecnologia web e do JQuery
Ferramentas disponibilizadas para desenvolvimento	Serve: Cria um webserver localmente, para tornar a implementação do UI mais rápida, contudo só é possível ser utilizado no browser.
Segurança e Encriptação	Permite a utilização de https e bloqueia naturalmente pedidos externos (fora do domínio, ou que não inscritos na whitelist)
Dependências de outras tecnologias	NodeJs; Java; Xcode
Ferramentas disponibilizadas para gestão de dependências e aplicações	Ferramentas CLI

Considerações e problemas

Durante a fase de desenvolvimento da prova de conceito nesta tecnologia encontrou-se vários problemas. No começo, originou-se problemas em organizar as várias versões do Java, pois esta tecnologia requer o Java 1.8, enquanto todas as outras tecnologias utilizam o Java 15. Foram encontrados também vários problemas em realizar debug e corrigir erros, devido à tecnologia utilizar uma WebView para encapsular e interpretar o código de JavaScript, HTML e CSS, o debug só é possível utilizando as ferramentas do browser, como por exemplo, o Google DevTools, para verificar erros nos logs. Um outro problema é que esta tecnologia não permite realizar testes unitários ou qualquer outro tipos de teste.

Por fim, um dos problemas que possam surgir ao manter a aplicação são os *Plug-in's*. Por motivo da necessidade de utilizar *Plug-in's* para aceder as API's do SO, certas funcionalidades têm um limite de versão mínima do SO e até poderão ficar obsoletas ou desatualizadas, fazendo com que novas versões dos SO's não sejam suportadas.

4.3.4 Ionic

A prova de conceito utilizando a tecnologia Ionic foi realizada com as versões mais recentes até à data 20 de julho de 2021. As versões utilizadas foram as seguintes: Ionic: 10; Angular 11.2.11; Cordova: 10; Capacitor:3.0.1; Java: 15; Xcode 12.5;

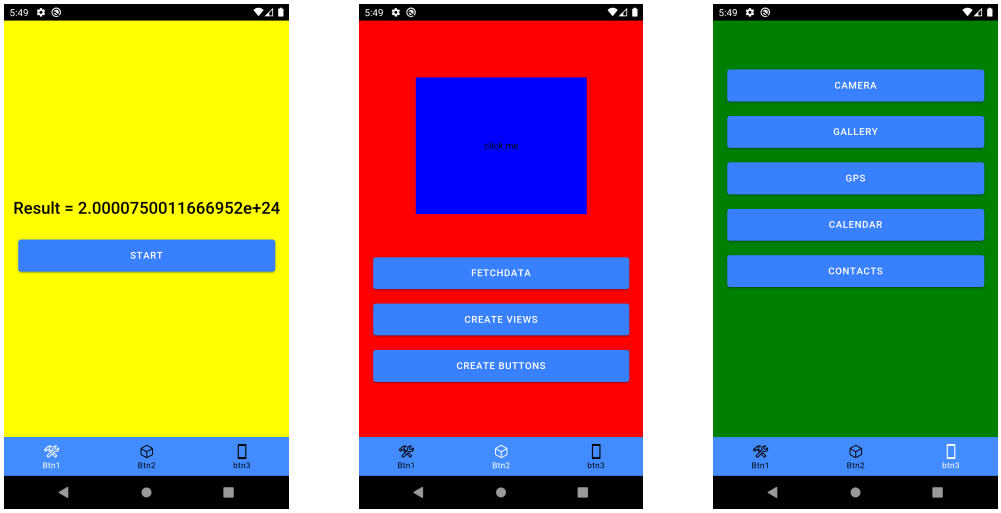

Figura 61. Prova de conceito com a tecnologia Ionic

A plataforma Ionic fornece uma framework para desenvolvimento mobile Cross-Platform, em que pode utilizar como a tecnologia Cordova e/ou Capacitor, fornecendo a opção da utilização de tecnologias populares como TypeScript, Vue, React e Angular [64]. As aplicações são encapsuladas numa WebView e utilizam a tecnologia Cordova e/ou Capacitor para aceder às API's do sistema.

Para a construção da prova de conceito nesta tecnologia, ao principio foi utilizado Vue como tecnologia auxiliar, mas durante o desenvolvimento dos acessos às funcionalidades do sistema, verificou-se que não era possível. Esta funcionalidade só estava disponível com a utilização da tecnologia Angular, foi assim necessário recomeçar a prova de conceito utilizado o Angular para a construção da UI e estrutura geral de código. Desta forma, também foi possível reaproveitar grande parte do código utilizado na prova de conceito em Angular, facilitando o processo de construção da interface e estruturação de código. Esta framework traz vários componentes criados para cada tecnologia a usar (Vue, React e Angular), facilitando o processo de criação do UI. Dito isto, o código está dividido de uma forma semelhante ao Angular, estando as chamadas as API's do sistema no Services.

Para o acesso às API's do sistema utilizamos a tecnologia Capacitor, que consiste numa framework muito idêntica ao Cordova, permitindo criar aplicações Cross-Platform híbridas utilizando tecnologias de desenvolvimento web. Esta tecnologia é suportada pela comunidade do Ionic e tem um grande suporte de como fazer o acesso às API's, no entanto verificou-se que faltavam algumas funcionalidades, sendo necessário adicionar alguns *Plug-in's* do Cordova para concluir a prova de conceito.

Parâmetros de avaliação

Os parâmetros de avaliação propostos na Tabela 7 foram avaliados com sucesso durante e após o desenvolvimento da prova de conceito utilizando a tecnologia Ionic. Estes dados estão apresentados na Tabela 13, identificados pelo seu parâmetro de avaliação.

Tabela 13. Parâmetros de avaliação propostos sobre a tecnologia Ionic

Parâmetro	Valor
Tamanho final da aplicação (pronto para distribuição nas *app stores*)	Android: 6,9 MB iOS: 24,1 MB
Estabilidade da aplicação entre plataformas	Estavel entre plataformas
Tempo médio da realização de testes	Primeiro Cenário: 20 segundos Segundo Cenário: 38 segundos
Recursos utilizados para renderizar a aplicação	CPU: 4% a 16% (Android) / 3,4 % a 21,5 % (iOS) RAM: 76,3 MB a 193,3 MB (Android) / 19,47 MB a 19,50 MB (iOS) Bateria: Light/Medium (Android) / High (iOS)
Tempo médio para renderizar a aplicação	Android: 2,76 segundos iOS: 1,24 segundos
Tamanho final do projeto	606 166 596 bytes (843,7 MB em disco)
Tempo de compilação	22,5 segundos (média)
Suporte da comunidade	Bom suporte da comunidade, principalmente para a ajuda de criação de Layouts
Suporte na resolução e demonstração de erros	Apenas demonstra os erros na fase de compilação. Para descobrir os erros que ocorrem depois dessa fase, é necessário realizar o debug do erro pelo Android studio, Xcode ou pelo Chrome DevTools.
Dificuldades de ambiente e desenvolvimento	É necessário instalar o Cordova e o Capacitor caso queiramos utilizar todos os acessos às API's do sistema. Como o Capacitor não tem todas as funcionalidades de acesso às API's do sistema, é necessário instalar *Plug-in's* do Capacitor e do Cordova.
Limitações e restrições da tecnologia	Não é possível utilizar o Capacitor com o Vue e React. Necessário ter o Android Studio e Xcode para compilar a aplicação Muitas funcionalidades são pagas
Ferramentas disponibilizadas para desenvolvimento	Ferramentas opcionais: Cordova; Capacitor; Angular; Vue; React; Esta Framework traz vários componentes criados para cada tecnologia, facilitando o processo de criação do UI
Segurança e Encriptação	Permite a utilização de https, encriptação de ficheiros e base de dados local.
Dependências de outras tecnologias	NodeJs; Java; Xcode; Tecnologia escolhida (Angular, Vue ou React)
Ferramentas disponibilizadas para gestão de dependências e aplicações	Ferramentas CLI; Cordova; Capacitor;

Considerações e problemas

Ao realizar a prova de conceito na tecnologias Ionic, foram encontrados vários problemas, fazendo com que a aplicação tivesse que ser refeita várias vezes. Ao utilizar o Vue como tecnologia de UI encontrei vários problemas, principalmente em testar a aplicação no dispositivo, pois não permitia utilizar o Cordova como a camada de interligação entre o SO e a WebView, apenas deixava utilizar o Capacitor, mas este ainda está na fase de desenvolvimento tendo apenas algumas API's testadas. Após utilizar o Capacitor verificou-se que muitas das API's não funcionavam para as tecnologias React e Vue, foi então necessário recomeçar a aplicação utilizando o Angular, sendo que neste momento é a única tecnologia que permite utilizar o Capacitor. Ao contrário da tecnologia Cordova, o Capacitor não permite compilar o código diretamente no dispositivo, é necessário utilizar o Android Studio ou Xcode para instalar e testar a aplicação num dispositivo real.

Um outro grande problema foi encontrado durante a fase de desenvolvimento das chamadas às API's, alguns dos *Plug-in's* não suportavam as versões mais recentes do Android e iOS, sendo que foi necessário atualizar para a nova versão do Capacitor. Durante esta atualização obteve-se muitos problemas com dependências e referências, sendo necessário recomeçar novamente a construção da prova de conceito.

4.3.5 Appc Titanium

A prova de conceito utilizando a tecnologia Appcelerator Titanium foi realizada com as versões mais recentes até à data 20 de julho de 2021. As versões utilizadas foram as seguintes: Titanium Version: 9.3.1.GA; Java: 15; Xcode 12.5;

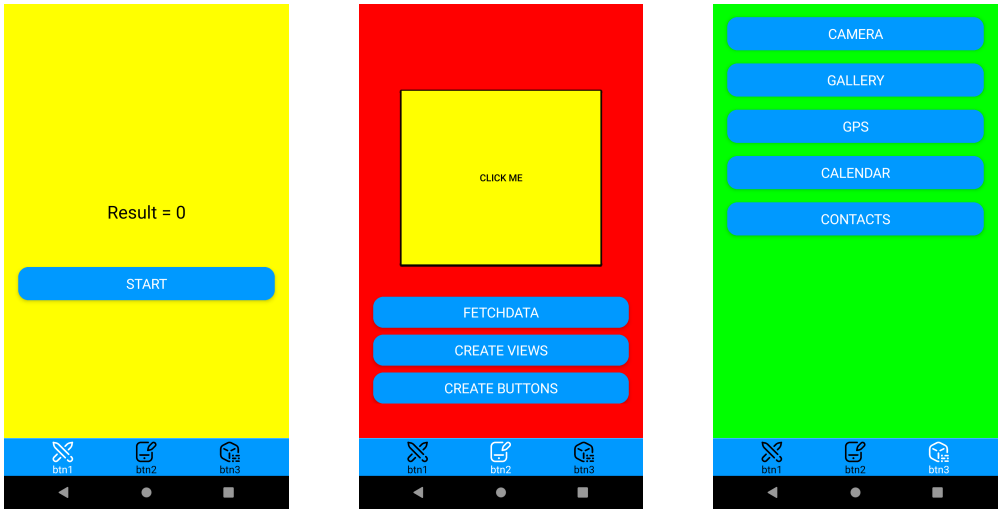

Figura 62. Prova de conceito com a tecnologia Appcelerator Titanium

Appcelerator Titanium é uma plataforma que oferece a possibilidade de desenvolver aplicações mobile Cross-Platform em JavaScript e XML[30], incluindo várias ferramentas como o seu próprio SDK que possui ferramentas nativas e a framework Alloy que utiliza um desenvolvimento MVC com as linguagens JavaScript, XML e TSS (Titanium Style Sheets).

A construção da prova de conceito nesta tecnologia foi realizada utilizando a framework Alloy, que estrutura o código em três categorias, o Controller, o Model e a View simplificando a construção de uma aplicação e facilitado a manutenção da mesma. Dito isto, o código está separado da seguinte forma:

- Views: Local onde estão as vistas em XML
- Controllers: Local que contém a funcionalidade das vistas
- Models: Local que contém a estrutura de dados das vistas
- Lib: Local utilizado para funcções que possasm ser importadas em vários controllers
- Styles: Local onde se encontram todos os ficheiros de estilos, utilizando a linguagem TSS

Todas as chamadas às API's do sistemas são realizadas utilizando a camada abstracta fornecida pelo SDK do Appcelerator Titanium, facilitando assim a adição de novas funcionalidades, não sendo necessário instalar bibliotecas para fazerem esse acesso. Todas as funcionalidades ligadas ao SO estão incluídas no SDK, fazendo com que estejam presentes caso sejam necessárias, no entanto, caso não sejam utilizadas, continuam a ocupar espaço no produto final.

Parâmetros de avaliação

Os parâmetros de avaliação propostos na Tabela 7 foram avaliados com sucesso durante e após o desenvolvimento da prova de conceito utilizando a tecnologia Appcelerator Titanium. Estes dados estão apresentados na Tabela 14, identificados pelo seu parâmetro de avaliação.

Tabela 14. Parâmetros de avaliação propostos sobre a tecnologia Appcelerator Titanium

Parâmetro	Valor
Tamanho final da aplicação (pronto para distribuição nas *app stores*)	Android: 20,3 MB iOS: 38 MB
Estabilidade da aplicação entre plataformas	Estável entre plataformas, no entanto para funcionalidades nativas é necessário criar condições para Android e iOS. Grandes problemas em manter a mesma interface entre plataformas e sistemas operativos
Tempo médio da realização de testes	Primeiro Cenário: 25 segundos Segundo Cenário: 41 segundos
Recursos utilizados para renderizar a aplicação	CPU: 11% a 22% (Android) / 14,2% a 58% (iOS) RAM: 30,1 MB a 172,9 MB (Android) / 31,83 MB a 57,71 MB (iOS) Bateria: Medium (Android) / Medium (iOS)
Tempo médio para renderizar a aplicação	Android: 2,54 segundos iOS: 1,04 segundos
Tamanho final do projeto	322 252 874 bytes (324,2 MB em disco)
Tempo de compilação	40 segundos (média)
Suporte da comunidade	Pouco suporte da comunidade, sendo difícil resolver erros sobre a sintaxe ou questões sobre a tecnologia
Suporte na resolução e demonstração de erros	Bom suporte na resolução de erros, no qual acontecem durante a compilação. No entanto caso o erro aconteça durante a utilização da aplicação, torna-se difícil saber onde o erro aconteceu devido o erro apontar para o local do ficheiro compilado
Dificuldades de ambiente e desenvolvimento	Difícil de realizar o Setup do ambiente da tecnologia no PC Não permite realizar testes unitários ou qualquer outro tipo de testes
Limitações e restrições da tecnologia	Algumas funcionalidades da plataforma são pagas
Ferramentas disponibilizadas para desenvolvimento	LiveView: Permite recompilar um ficheiro sempre que se realize uma alteração, não sendo necessário recompilar toda a aplicação Package:Facilita a criação do APK para debug ou release.
Segurança e Encriptação	Permite a utilização de https, encriptação de ficheiros e base de dados local.
Dependências de outras tecnologias	NodeJs; Java; Xcode;
Ferramentas disponibilizadas para gestão de dependências e aplicações	Ferramentas CLI;

Considerações e problemas

Para construir aplicações utilizando esta plataforma é necessário criar uma conta na plataforma Appcelerator, em que apenas algumas ferramentas são gratuitas. Um grande problema que esta tecnologia possui, é o facto de que o código é compilado utilizando uma chave num servidor da empresa Axway, compilando totalmente o código escrito em um servidor e então depois devolvido ao computador para ser executado no dispositivo. Esta forma de compilação traz uma enorme segurança para o criador da aplicação, pois sem o ID específico não é possível compilar a aplicação, no entanto, o grande problema é que não é possível compilar o código sem ligação à Internet. Caso o utilizador perca a ligação à Internet, já não consegue testar a sua aplicação, e todo o processo de desenvolvimento terá de ser feito sem realizar debug nem testar num dispositivo até voltar a ter ligação à Internet.

Um outro problema é que algumas funcionalidades nativas comportam-se de forma diferente dependente do dispositivo e SO. Ao utilizar esta tecnologia, é necessário ter em conta os dois casos, a aplicação Android que as funcionalidades nativas são realizadas de uma forma específica, e a aplicação iOS onde as funcionalidades nativas são executadas de uma forma diferente. Para

alguma parte dos casos, estas funcionalidades são idênticas, contudo, para grande parte é necessário verificar primeiro qual é o SO e depois executar o fluxo pretendido.

4.3.6 React Native

A prova de conceito utilizando a tecnologia React Native foi realizada com as versões mais recentes até à data 20 de julho de 2021. As versões utilizadas foram as seguintes: React-Native: 0.64; Java: 15; Xcode 12.5;

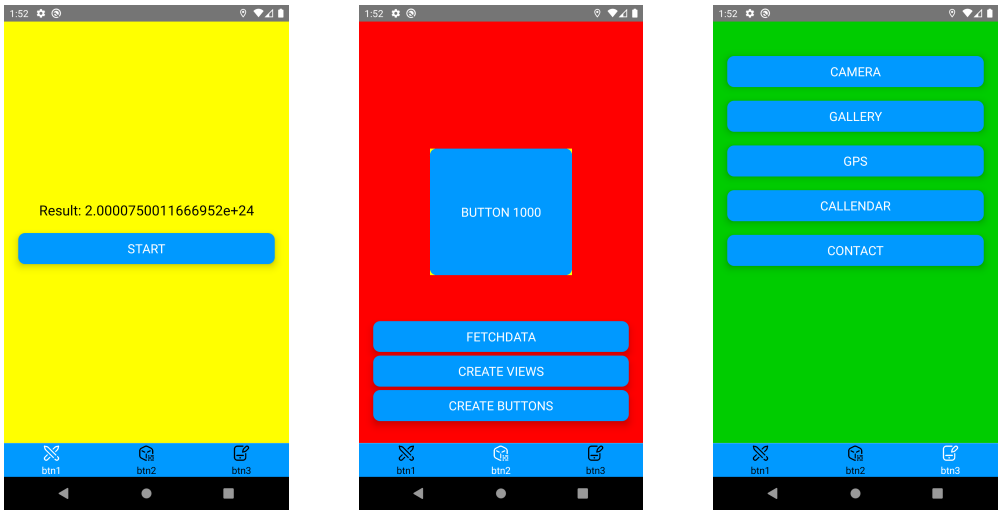

Figura 63. Prova de conceito com a tecnologia Angular

O React Native consiste numa framework construida em Node.JS, que utiliza como base a biblioteca React para criar aplicações mobile [98]. Como este utiliza a biblioteca React, a estrutura de código é muito semelhante, facilitando a construção de uma aplicação mobile caso tenha conhecimento prévio desta tecnologia. Esta framework utiliza então como base o React, permitindo um estilo de programação funcional e por objetos, usando uma linguagem específica, o JSX (JavaScript Extended), que em um único ficheiro, engloba o Javacript e HTML.

Como o React Native possui uma arquitetura em árvore de componentes, para a construção da prova de conceito nesta tecnologia, a estrutura de código foi separada em *Components* e *Screens*, em que cada *Screen* representa uma página completa e os *Components* um objeto de UI que poderá ser incluído em várias páginas. Assim sendo, o código está separado da seguinte forma:

- *Components*: Local onde se encontram objetos de UI.
- *Screens*: Local onde se encontram as Páginas.
- Lib: Local onde estão todas a funcionalidades que possam ser acedidas por *Components* e *Screens*

Ao criar a prova de conceito nesta framework, surgiu a possibilidade de utilizar o Expo, que consiste num SDK que fornece um conjunto de ferramentas para o desenvolvimento e todas as API's dos SO's num único local [44]. No entanto, a utilização do Expo resultava numa aplicação muito grande por este englobar todas as soluções nativas no SDK, mesmo que algumas destas não seriam necessárias. Por este motivo, e outros problemas dependentes da sua arquitetura [45], foi decidido não o utilizar. Todas as chamadas às API's do sistema foram realizadas utilizando um

conjunto de *Plug-in's* criados pelo próprio Facebook ou pela comunidade que a suporta, fazendo com que apenas sejam instalados os *Plug-in's* necessários.

Parâmetros de avaliação

Os parâmetros de avaliação propostos na Tabela 7 foram avaliados com sucesso durante e após o desenvolvimento da prova de conceito utilizando a tecnologia React Native. Estes dados estão apresentados na Tabela 15, identificados pelo seu parâmetro de avaliação.

Tabela 15. Parâmetros de avaliação propostos sobre a tecnologia React Native

Parâmetro	Valor
Tamanho final da aplicação (pronto para distribuição nas *app stores*)	Android: 27,5 MB iOS: 34,9 MB
Estabilidade da aplicação entre plataformas	Estável entre plataformas, no entanto para funcionalidades nativas é necessário criar condições para Android e iOS.
Tempo médio da realização de testes	Primeiro Cenário: 23 segundos Segundo Cenário: 37 segundos
Recursos utilizados para renderizar a aplicação	CPU: 10% a 26% (Android) / 32,7% a 79,8% (iOS) RAM: 37,5 MB a 182,2 MB (Android) / 9,63 MB a 36,05 MB (iOS) Bateria: Medium (Android) / Medium (iOS)
Tempo médio para renderizar a aplicação	Android: 2,03 segundos iOS: 1,02 segundos
Tamanho final do projeto	1 943 938 028 bytes (2,15 GB em disco)
Tempo de compilação	28 segundos (média)
Suporte da comunidade	Gigante suporte da comunidade, facilitando a construção de uma aplicação, tanto a nível de UI (React) como a nível mobile
Suporte na resolução e demonstração de erros	Bom suporte na resolução de erros, no qual acontecem durante a compilação e em runtime.
Dificuldades de ambiente e desenvolvimento	Dificuldade em iniciar o projeto.
Limitações e restrições da tecnologia	Não apresenta restrições nem limitações.
Ferramentas disponibilizadas para desenvolvimento	HotReload:Permite recompilar um ficheiro sempre que se realize uma alteração, não sendo necessário recompilar toda a aplicação Expo: Ao criar uma aplicação react native, é possível juntar com o SDK do Expo facilitando a integração de API's dos SO's
Segurança e Encriptação	Permite a utilização de https, encriptação de ficheiros e base de dados local.
Dependências de outras tecnologias	NodeJs; Java; Xcode;
Ferramentas disponibilizadas para gestão de dependências e aplicações	Ferramentas CLI; Expo;

Considerações e problemas

Durante a fase de desenvolvimento, após excluir a utilização do Expo, tornou-se um pouco difícil de encontrar *Plug-in's* que fossem fiáveis e mantidos pela comunidade, porque muitos destes, ou não suportavam as novas versões dos SO's, ou então tinham problemas de estabilidade e de suporte. Um outro problema é que a maioria não implementavam o pedido das permissões para o acesso à API em questão, sendo necessário incluir um outro *Plug-in's* para fazer esse pedido, fazendo com que aumentasse a complexidade de código.

Um grande problema que esta tecnologia possui é a grande diferença entre versões do React e React Native, dificultando manter a aplicação entre novas versões. Como as versões são muito diferentes, as soluções apresentadas pela comunidade ou já não funcionavam, ou então era necessário

adaptá-las para a versão a ser utilizada. Este foi um grande problema encontrado, porque como a curva de aprendizagem é muito grande, o desenvolvimento inicial da aplicação foi muito demorado, fazendo com que esta tecnologia na minha opinião, não seja indicada para aplicações simples.

4.3.7 Native Script

A prova de conceito utilizando a tecnologia Native Script foi realizada com as versões mais recentes até à data 20 de julho de 2021. As versões utilizadas foram as seguintes: NativeScript: 8.0.0; Java: 15; Xcode 12.5;

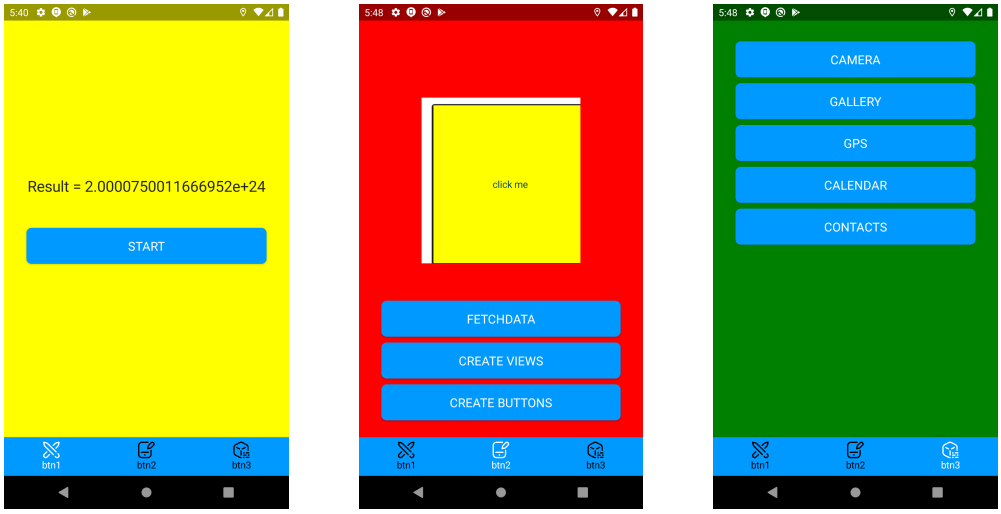

Figura 64. Prova de conceito com a tecnologia Native Script

O NativeScript é uma framework de desenvolvimento mobile que permite a construção de aplicações utilizando tecnologias web [84] como JavaScript ou TypeScript, para as plataformas iOS e Android. Esta tecnologia utiliza como arquitetura o MVVM (Model, View, View-Model), em que para cada View temos um View-Model e caso necessário um Model. Desta forma permite-nos que para cada Vista exista um ficheiro que representa a sua estrutura e possa interagir com ela através de *observers*. Para registar cada View-Model é necessário fazer um *binding* através de um ficheiro JavaScript ou TypeScript, mas no entanto, caso o *developer* não goste desta abordagem, pode utilizar um ficheiro que interaja diretamente com a View.

Para a construção da prova de conceito nesta framework, foi utilizado a abordagem VMVM, pelo motivo que separa a funcionalidade pela responsabilidade, em conjunto com a linguagem TypeScript por esta ser orientada a objetos. Já para a construção das View é utilizado XML. Dito isto, a estrutura de código foi a seguinte:

- App-Resources: Local onde se encontram as configurações do Android e iOS
- Css: Local onde se encontram os estilos para cada vista
- Models: Local onde residem os Modelos e estrutura de dados.
- Páginas: Local onde que contém cada página, com um conjunto de ficheiros:
  - Ficheiro Page.XML: Ficheiro que representa a vista
  - Ficheiro Page.TS: Ficheiro que contém algumas funcionalidades da vista e o *binding* ao View-Model

- Ficheiro Page-View-Model.TS: Ficheiro que representa o Model da vista e interage utilizando *observers*

Parâmetros de avaliação

Os parâmetros de avaliação propostos na Tabela 7 foram avaliados com sucesso durante e após o desenvolvimento da prova de conceito utilizando a tecnologia Native Script. Estes dados estão apresentados na Tabela 16, identificados pelo seu parâmetro de avaliação.

Tabela 16. Parâmetros de avaliação propostos sobre a tecnologia Native Script

Parâmetro	Valor
Tamanho final da aplicação (pronto para distribuição nas *app stores*)	Android: 25,2 MB iOS: 33,1 MB
Estabilidade da aplicação entre plataformas	Pouca estabilidade entre plataformas, especialmente em termos de UI
Tempo médio da realização de testes	Primeiro Cenário: 26 segundos Segundo Cenário: 38 segundos
Recursos utilizados para renderizar a aplicação	CPU: 12% a 26% (Android) / 15,8% a 57% (iOS) RAM: 52,8 MB a 161 MB (Android) / 21,83 MB a 54,84 MB (iOS) Bateria: Medium/High (Android) / Medium (iOS)
Tempo médio para renderizar a aplicação	Android: 2,83 segundos iOS: 1,78 segundos
Tamanho final do projeto	1 447 868 521 bytes (1,5 GB em disco)
Tempo de compilação	21,6 segundos (média)
Suporte da comunidade	Suporte da comunidade apenas na utilização da *Plug-in's* para o acesso as API's.
Suporte na resolução e demonstração de erros	Bom suporte na resolução de erros, no qual acontecem durante a compilação e em runtime.
Dificuldades de ambiente e desenvolvimento	Dificuldade na estrutura de código e em aplicar as boas práticas de programação Não tem ferramentas de recompilação de código rápido, é necessário recompilar todo o código para verificar alterações
Limitações e restrições da tecnologia	Não permite realizar testes unitários ou qualquer outro tipo de testes Apresenta restrições entre versões, caso não tenhamos a versão mais atualizada da tecnologia, alguns *Plug-in's* deixam de funcionar. Muitos dos *Plug-in's* necessitam de estabelecer limite mínimo para versões Android e iOS
Ferramentas disponibilizadas para desenvolvimento	Playground: IDE online que possibilita praticar e criar a UI da aplicação sem ser necessário realizar Setup no PC ou tecnologias como Android Studio e XCode.
Segurança e Encriptação	Permite a utilização de https
Dependências de outras tecnologias	NodeJs; Java; Xcode;
Ferramentas disponibilizadas para gestão de dependências e aplicações	Ferramentas CLI;

Considerações e problemas

Ao realizar a prova de conceito nesta tecnologia, notou-se que esta tem muitos problemas a nível de manutenção, documentação e *Plug-in's*. As versões têm enormes diferenças, fazendo com que quando realizamos uma atualização de versão, muitas das funcionalidades implementadas deixam de funcionar, principalmente os *Plug-in's*, no qual originam muitos conflitos e problemas. Durante o desenvolvimento da prova de conceito foi necessário realizar uma atualização para uma versão mais recente do Native Script, mas durante essa atualização, surgiram grandes dificuldades na resolução de dependências, como NativeScript/Core, NativeScript/ios, NativeScript/Android,

entre outras, no qual foi necessário atualizar as dependências uma por uma. Esta atualização foi necessária devido a alguns *Plug-in's* de acesso às API's não funcionarem corretamente nas versões que estavam inicialmente a utilizar.

Um outro problema devido à grande diferença entre versões é a documentação fornecida pela Native Script. As funcionalidades são implementadas de forma diferente, levando a diferentes versões de documentação. Muitas vezes ao procurar documentação sobre alguma funcionalidade, verificava que só havia documentação disponível para versões mais antigas, fazendo com que fosse necessário verificar várias vezes qual das versões estava a procurar, pois muitas das vezes estava a utilizar código apresentado na documentação que não funcionava na prova de conceito por ser versões diferentes.

4.3.8 Flutter

A prova de conceito utilizando a tecnologia Flutter foi realizada com as versões mais recentes até à data 20 de julho de 2021. As versões utilizadas foram as seguintes: Flutter: 2.2.2; Dart 2.13.3; Java: 15; Xcode 12.5;

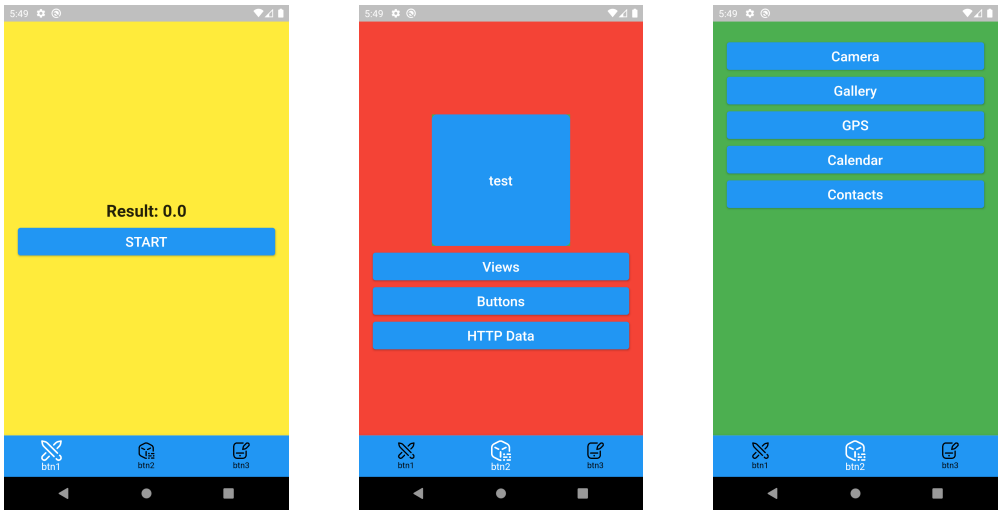

Figura 65. Prova de conceito com a tecnologia Flutter

A framework Flutter utiliza como única linguagem o Dart, que permite a criação de aplicações mobile, web e desktop Cross-Platform [46] através do mesmo código, sendo apenas necessário implementar algumas funcionalidades específicas para cada aplicação. Esta framework possui um conjunto de ferramentas que transforma o código escrito em Dart para código nativo dos sistemas iOS e o Android.

O grande foco desta está nos *Widgets* que são blocos de construção da interface gráfica, em que os mais utilizados são os *StateFull* e *StateLess*, formando no geral uma tipologia de árvore de *Widgets*. Para a construção da prova de conceito na tecnologia Flutter, foi utilizado a abordagem de programação orientada a objetos e *Widgets*, tirando os bons aspetos de ambas as partes. Desta forma, a estrutura de código foi separada da seguinte forma:

– Ficheiro Main.dart utilizado como ponto de entrada, em que neste caso é utilizado para criar a navegação

– Ficheiro Pubspec.ymal: ficheiro de configuração para controlo de dependências, *Plug-in's* e mapeamento de assets

– *Widgets*: Local onde residem os *Widgets*

– *Utils*: Local onde estão as classes para realizar os acessos às API's

– Test: Local para realizar os testes unitários ou qualquer outro tipo de testes.

Para realizar chamadas às API's do sistema é necessário instalar *Plug-in's* escritos em Java ou Objective-C que foram criados pela Google ou pela comunidade que suporta o Flutter, fazendo com que apenas sejam instalados os *Plug-in's* necessários, no entanto, existem uma grande variedade das mesmas funcionalidades, tornando difícil a escolha do melhor *Plug-in* a utilizar.

Parâmetros de avaliação

Os parâmetros de avaliação propostos na Tabela 7 foram avaliados com sucesso durante e após o desenvolvimento da prova de conceito utilizando a tecnologia Flutter. Estes dados estão apresentados na Tabela 17, identificados pelo seu parâmetro de avaliação.

Tabela 17. Parâmetros de avaliação propostos sobre a tecnologia Flutter

Parâmetro	Valor
Tamanho final da aplicação (pronto para distribuição nas *app stores*)	Android: 17,5 MB iOS: 55,6 MB
Estabilidade da aplicação entre plataformas	Estável entre plataformas.
Tempo médio da realização de testes	Primeiro Cenário: 23 segundos Segundo Cenário: 34 segundos
Recursos utilizados para renderizar a aplicação	CPU: 12% a 18% (Android) / 34,6% a 55,2% (iOS) RAM: 90 MB a 139,2 MB (Android) / 24,56 MB a 113,08 MB (iOS) Bateria: Medium (Android) / High (iOS)
Tempo médio para renderizar a aplicação	Android: 1.8 segundos iOS: 0.9 segundos
Tamanho final do projeto	864 826 246 bytes (887,4 MB em disco)
Tempo de compilação	Primeira vez: 40 segundos (média) Segunda vez: 14 segundos (média)
Suporte da comunidade	Grande suporte da comunidade, tanto na resolução de erros como na criação de *Plug-in's* para facilitar a construção de Widget's de UI e API's (nativas ou externas)
Suporte na resolução e demonstração de erros	Excelente suporte na resolução de erros. Quando ocorre um erro durante ou depois da compilação, retorna-nos a posição correta do erro, e qual o seu problema.
Dificuldades de ambiente e desenvolvimento	Poderá trazer problemas de escalabilidade, devido à estrutura de código É necessário ter conhecimento de POO
Limitações e restrições da tecnologia	Não apresenta restrições nem limitações
Ferramentas disponibilizadas para desenvolvimento	LiveView: Permite recompilar um ficheiro sempre que se realize uma alteração, recompilando apenas o ficheiro modificado
Segurança e Encriptação	Permite a utilização de https, encriptação de ficheiros e base de dados local.
Dependências de outras tecnologias	Dart; Java; Xcode;
Ferramentas disponibilizadas para gestão de dependências e aplicações	Ferramentas CLI; Integração na tecnologia Android Studio, facilitando a gestão de dependências e tecnologias envolvidas no projeto;

Considerações e problemas

Durante o desenvolvimento da prova de conceito notou-se que a estrutura de código poderá causar problemas na escalabilidade da aplicação, devido à enorme quantidade de código necessário para criar um *Widgets* e a forma como estes são escritos, fazendo com que o número de ficheiros cresça muito rapidamente e torne-se um pouco confuso de manter e escalar.

Como o Flutter não possui acesso às API's nativas no seu SDK, é necessário utilizar *Plug-in's* criados pela comunidade. Isto traz pontos positivos como reduz o tamanho da aplicação e apenas incluindo as API's necessárias, no entanto, existe uma grande variedade de *Plug-in's* que torna difícil a escolha de qual iremos utilizar, sendo que muitos destes deixaram de ser mantidos podendo não suportar novas versões dos SO's ou possuem restrições. Devido a este problema é necessário rever bem o *Plug-in* a utilizar e ter em conta as suas restrições.

4.3.9 Xamarin

A prova de conceito utilizando a tecnologia Xamarin foi realizada com as versões mais recentes até à data 20 de julho de 2021. As versões utilizadas foram as seguintes: Xamarin.Forms: 5.0; Visual Studio 2019; Java: 15; Xcode 12.5;

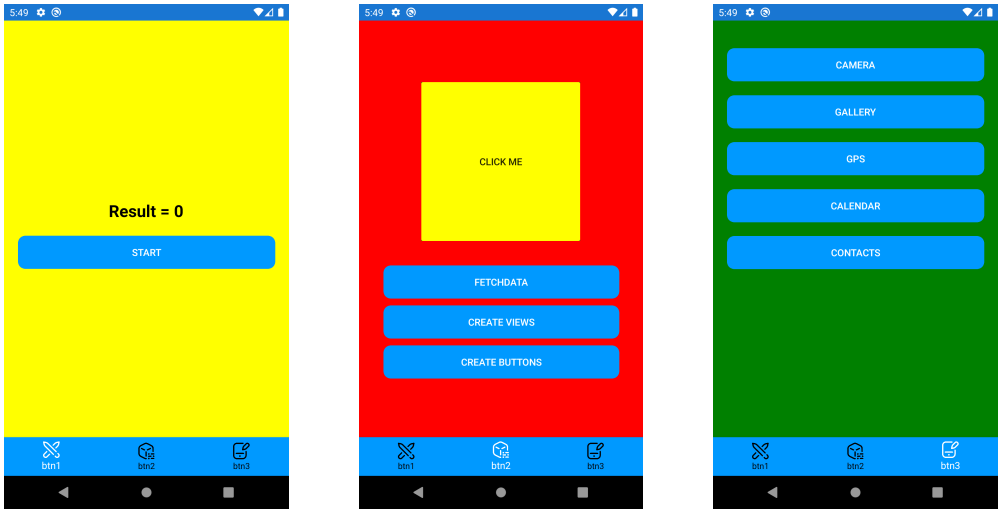

Figura 66. Prova de conceito com a tecnologia Xamarin

A plataforma Xamarin utiliza como base o .Net, permitindo criar aplicações mobile Cross-Platform, utilizando apenas a linguagem C#, usufruindo de todas as componentes nativas disponibilizadas pelo dispositivo [107], permitindo a *developers* de Asp.Net criar aplicações mobile muito rapidamente. Esta plataforma fornece três opções para o desenvolvimento de aplicações mobile, o desenvolvimento de aplicações apenas para o iOS (Xamarin.iOS), o desenvolvimento de aplicações apenas para o Android (Xamarin.Android) e o desenvolvimento de aplicações mobile Cross-Platform (Xamarin.Forms), no qual a opção escolhida foi o Xamarin.Forms.

A solução do Xamarin.Forms inclui três projetos, o Xamarin.Android que contém especificações do Android, o Xamarin.iOS para especificações do iOS e o Xamarin.Forms que consiste no projeto que o código será partilhado para ambos os SO's. Caso seja necessário inserir alguma funcionalidade específica para o SO, esta será realizado no projeto Xamarin.Android ou Xamarin.iOS. O projeto Xamarin.Forms possui uma arquitetura MVVM (Model, View, View-Model) idêntica à apresentada na tecnologia NativeScript, secção 4.3.7, facilitando assim a manutenção e escalabilidade da

aplicação, separando a estrutura de código pelas suas responsabilidades. Dito isto, a estrutura de código está separada da seguinte forma:

- Services: Local onde contém os serviços e chamadas as API's isoladas.
- Models: Local onde se encontram as estruturas e modelos de dados
- View: Local que contém as páginas e componentes da aplicação, em que para cada página ou componente existe:
  - XAML, Utilizados para a construção do UI, que representa a estrutura da página ou componente e, os estilos a utilizar
  - CS, Utilizados para realizar uma ligação entre o View-Model e XAML, também utilizado para inicializar a página/componente e integrações simples ou animações)
- View-Model: Local específico para os View-Model, que funciona à base de *Observer's*, *Command's* e *Task's*.

As chamadas às API's do sistema são feitas através da utilização de um package chamada de Xamarin.Essentials que é uma a biblioteca essencial que fornece todas as chamadas às API's do sistema. No entanto esta biblioteca não tem todos os acessos, sendo necessário instalar alguns *Plug-in's* .

Parâmetros de avaliação

Os parâmetros de avaliação propostos na Tabela 7 foram avaliados com sucesso durante e após o desenvolvimento da prova de conceito utilizando a tecnologia Xamarin. Estes dados estão apresentados na Tabela 18, identificados pelo seu parâmetro de avaliação.

Tabela 18. Parâmetros de avaliação propostos sobre a tecnologia Xamarin

Parâmetro	Valor
Tamanho final da aplicação (pronto para distribuição nas *app stores*)	Android: 12,6 MB iOS: 53,2 MB
Estabilidade da aplicação entre plataformas	Estável entre plataformas.
Tempo médio da realização de testes	Primeiro Cenário: 21,2 segundos Segundo Cenário: 42,4 segundos
Recursos utilizados para renderizar a aplicação	CPU: 11% a 20% (Android) | 13,9% a 32,8% (iOS) RAM: 35,7 MB a 168,7 MB (Android) | 7,03 MB a 39,11 MB (iOS) Bateria: Medium (Android) | Medium (iOS)
Tempo médio para renderizar a aplicação	Android: 2,89 segundos iOS: 1,8 segundos
Tamanho final do projeto	259 655 274 bytes (274 MB em disco)
Tempo de compilação	18,3 segundos (média)
Suporte da comunidade	Pouco suporte da comunidade Difícil encontrar soluções para certos problemas
Suporte na resolução e demonstração de erros	Muito bom suporte na demonstração de erros, tanto a nível de compilação como em runtime
Dificuldades de ambiente e desenvolvimento	Grande parte dos acessos às API's estão na package Xamarin.Essentials, no entanto para algumas funcionalidades foi necessário instalar *Plug-in's* pelo Nuget.

Limitações e restrições da tecnologia	Não foi possível aceder ao calendário pessoal do dispositivo. Poucos layouts de UI disponíveis.
Ferramentas disponibilizadas para desenvolvimento	Visual Studio; Nugets;
Segurança e Encriptação	Permite a utilização de https, encriptação de ficheiros e base de dados local.
Dependências de outras tecnologias	Xamarin.Android; Xamarin.iOS; Xcode; Java;
Ferramentas disponibilizadas para gestão de dependências e aplicações	Visual Studio

Considerações e problemas

Para construir aplicações utilizando esta tecnologia, é necessário que o *developer* tenha bons conhecimentos em programação orientada a objetos e também .NET, devido a esta tecnologia utilizar o .NET como base torna-se complicado de compreender alguns passos e formas de escrever código. Um *developer* sem estes conhecimentos poderá encontrar vários problemas durante a fase de desenvolvimento, porque existem funcionalidades que são muito específicas para o .NET.

Um outro problema encontrado durante o desenvolvimento foi a necessidade de integrar um *Plug-in* para aceder à Câmara e Galeria do dispositivo, porque esta funcionalidade não estava integrada no Xamarin. Como existe uma grande variedade de *Plug-in's* no Nuget, foi muito difícil de encontra um *Plug-in* que fosse mantido e que funcionasse com as versões mais recentes do Android e iOS. Mesmo depois de encontrar um bom *Plug-in* verificou-se que este tinha uma versão mínima permitida apartir do Android 8, fazendo assim com que a aplicação não funcionasse nos dispositivos com o Android anterior à versão 8.

Um outro grande problema centra-se no facto de que não foi possível aceder à API do Calendário, pois a tecnologia não disponha desse acesso, nem existem *Plug-in's* que permitiam o acesso a essa API. Este foi o maior problema que foi encontrado na prova de conceito, pois não permitiu a realização todos os passos requeridos.

4.4 Casos de Teste

Todos os casos de teste foram realizados sobre a aplicação que resultou da prova de conceito criadas e descritas acima. Estes casos de testes, como discutidos e apresentados na secção 3.2.6, do capítulo 3 foram realizados de forma isolada, individual e num ambiente controlado de forma a minimizar o impacto de fatores externos, tais como a qualidade da rede, sinal GPS, entre outros. Cada teste foi realizado no mesmo ambiente, no mesmo local, fazendo assim com que os impactos de fatores externos se mantivessem semelhantes, não prejudicando os dados obtidos pelas ferramentas de monitorização.

Cada caso de teste foi realizado três vezes em cada tecnologia, fazendo com que os dados abaixo apresentados sejam uma média de valores dos dados obtidos.

4.4.1 Primeiro Caso de Teste

O primeiro cenário foi um cálculo matemático (semelhante ao utilizado no artigo de Delia et. al [37]), apresentado na secção 3.2.6, do capítulo 3, incluindo várias iterações, avaliando assim a velocidade de processamento e o tempo de execução necessário para efetuar cálculos intensivos, sempre monitorizando o consumo de recursos.

Para cada caso de teste, o cálculo matemático foi executado dez vezes, fazendo com que os dados apresentados sejam uma média das trinta execuções para cada SO's.

4.4.2 Consumo de recursos - CPU

Para a monitorização do consumo de recursos do CPU no primeiro cenário, foram avaliados vários parâmetros, como apresentados na Tabela 19. O primeiro parâmetro, Tempo de Resposta, consiste na média de tempo desde o clique no botão para iniciar o processo do cálculo matemático até a aplicação mostrar o resultado desse cálculo na vista. O segundo parâmetro, Main Thread, equivale à média de tempo em que o processo de cálculo esteve na *Thread* principal do CPU, verificando assim quando tempo este permanece no CPU para resolver o cálculo e devolver à vista. O terceiro, quarto e quinto parâmetro correspondem respetivamente à percentagem média, máxima e mínima da utilização de CPU de pico de cada execução.

Tabela 19. Consumo de recursos - CPU - Primeiro Caso de Teste

Tecnologia	Tempo de Resposta (ms)		Main Thread (ms)		Média (%)		Máximo (%)		Mínimo (%)	
	Android	iOS	Android	iOS	Android	iOS	Android	iOS	Android	iOS
Angular	670 ms	230 ms	296,5 ms	1070 ms	29,3%	6,9%	32%	8,2%	28,7%	6,7%
Vue	546 ms	220 ms	390 ms	1020 ms	38,8%	9,1%	42,3%	10,1%	36,2%	7,3%
Cordova	475 ms	140 ms	579 ms	459,09 ms	7%	4,6%	12%	5,7%	5%	3,1%
Ionic	612 ms	210 ms	818 ms	536,80 ms	19%	8,1%	22%	12,2%	15%	7,8%
Appc Titanium	440 ms	160 ms	503 ms	1050 ms	13%	18,5%	14%	21,1%	10%	12,8%
React Native	580 ms	156 ms	781 ms	1250 ms	13%	18,1%	16%	21,4%	11%	15,7%
Native Script	390 ms	173 ms	540 ms	1140 ms	13%	18,8%	15%	21,3%	10%	13,1%
Flutter	400 ms	248 ms	764 ms	1070 ms	15%	21%	16%	31%	13%	12%
Xamarin	350 ms	180 ms	586 ms	1060 ms	17%	10,4%	19%	11,6%	14%	7,8%

4.4.3 Consumo de recursos - RAM

Para a monitorização do consumo de recursos de RAM no primeiro cenário, foram considerados dois parâmetros, como apresenta a Tabela 20, no qual o primeiro representa a média da memória RAM utilizada em *MegaBytes* durante a execução do cálculo matemático e o segundo representa o valor de pico registado da memória RAM em *MegaBytes*.

Tabela 20. Consumo de recursos - RAM - Primeiro Caso de Teste

Tecnologia	**Memória utilizada (MB)**		**Pico de memória utilizada (MB)**	
	Android	**iOS**	**Android**	**iOS**
Angular	6,5 MB	10,45 MB	7 MB	10,82 MB
Vue	5,8 MB	9,3 MB	6 MB	9,84 MB
Cordova	89,7 MB	8,98 MB	92,2 MB	9,47 MB
Ionic	182,3 MB	19,47 MB	197,1 MB	19,61 MB
Appc Titanium	97 MB	59,98 MB	118 MB	68,3 MB
React Native	83,2 MB	34,8 MB	88,5 MB	36,33 MB
Native Script	89,8 MB	54,84 MB	97,7 MB	59,11 MB
Flutter	63,4 MB	113 MB	70,3 MB	126 MB
Xamarin	98,8 MB	39,63 MB	103 MB	39,63 MB

4.4.4 Consumo de recursos - Bateria

Para a monitorização do consumo de recursos da Bateria no primeiro cenário, foi avaliado qual o seu consumo desde o início da aplicação até ao fim das dez execuções. Os níveis de bateria mostrados na Tabela 21 variam entre *Low*, *Medium*, *High* e *Very High*, indicados pelas ferramentas de *benchmark* em tempo real, e de seguida, a quantidade de bateria utilizada no final da realização dos testes [2] .

Tabela 21. Consumo de recursos - Bateria - Primeiro Caso de Teste

Tecnologia	Níveis de bateria utilizado		Quantidade de bateria utilizada	
	Android	**iOS**	**Android**	**iOS**
Angular	Low	Low	2,36 mAh	1/20
Vue	Low	Low	2,19 mAh	1/20
Cordova	Low	Low	0.86 mAh	1/20
Ionic	Low	Low	1,3 mAh	1/20
Appc Titanium	Low/Medium	Low	0,58 mAh	1/20
React Native	Low/Medium	Low	1,8 mAh	1/20
Native Script	Low/Medium	Low	0,67 mAh	1/20
Flutter	Low	Low	1,03 mAh	1/20
Xamarin	Low	Low	1,71 mAh	1/20

4.4.5 Segundo Caso de Teste

O segundo cenário consistiu na execução de uma sequência de vários passos, apresentados na secção 3.2.6 do capítulo 3. Esta lista de passos foi baseada na investigação de Dorfer et al.[39]. Este caso de teste foi executado de forma fluida, assemelhando-se a uma utilização de um utilizador normal, isto é, sem verificar os *Logs* e o término da tarefas, simulando assim um ambiente de utilização diário.

4.4.6 Consumo de recursos - CPU

Para a monitorização do consumo de recursos do CPU no segundo cenário, foram avaliados três parâmetros, como apresentados na Tabela 22. O primeiro parâmetro, consiste na percentagem média da utilização de CPU de pico de cada tarefa ou passo executado. O segundo parâmetro, corresponde à percentagem máxima da utilização de CPU de pico registado durante todo o cenário. E por fim, o terceiro parâmetro, equivale à percentagem mínima da utilização de CPU de pico de cada tarefa ou passo executado.

[2] Para o sistema iOS não é possível obter dados concretos como o sistema Android, a ferramenta indica um nível entre 0 a 20, no qual mostra a quantidade de energia que a aplicação está a usar, sendo assim números subjetivos, mas no entanto são considerados válidos [10].

Tabela 22. Consumo de recursos - CPU - Segundo Caso de Teste

Tecnologia	Média (%)		Máximo (%)		Mínimo (%)	
	Android	**iOS**	**Android**	**iOS**	**Android**	**iOS**
Angular	20,4 %	6,9 %	42,1 %	8,5 %	9 %	5,1 %
Vue	34,6 %	9,2 %	57,2 %	10,2 %	25 %	7,8 %
Cordova	12 %	4,8 %	17 %	7,2 %	5 %	2,3 %
Ionic	18 %	26,5 %	32 %	73,2 %	12 %	8,9 %
Appc Titanium	18 %	19 %	25 %	82 %	6 %	7 %
React Native	27 %	20,5 %	43 %	134 %	12 %	5,7 %
Native Script	20 %	31,2 %	25 %	99,8 %	9 %	10 %
Flutter	16 %	42 %	22 %	115 %	4 %	4 %
Xamarin	15 %	18,9 %	28 %	101,6 %	10 %	10,5 %

4.4.7 Consumo de recursos - RAM

Para a monitorização do consumo de recursos de RAM no segundo cenário, foram considerados quatro parâmetros, como mostra a Tabela 23. O primeiro parâmetro regista a quantidade de memória RAM em *MegaBytes* utilizada após a aplicação estar carregada no dispositivo, de forma a obter uma base durante todo o caso de teste. O segundo, terceiro e quarto parâmetro correspondem respetivamente à quantidade média, máxima e mínima de memória RAM em *MegaBytes* utilizada durante cada tarefa ou passo executado.

Tabela 23. Consumo de recursos - RAM - Segundo Caso de Teste

Tecnologia	Memória antes do teste (MB)		Média (MB)		Maximo (MB)		Mínimo (MB)	
	Android	**iOS**	**Android**	**iOS**	**Android**	**iOS**	**Android**	**iOS**
Angular	3,5 MB	10,49 MB	5,2 MB	14,74 MB	7,2 MB	18,52 MB	4 MB	14,56 MB
Vue	4,1 MB	10,34 MB	6,1 MB	15,12 MB	6,8 MB	19,27 MB	4,4 MB	14,42 MB
Cordova	32,4 MB	6,91 MB	73,4 MB	12,89 MB	104,5 MB	14,30 MB	43,6 MB	8,48 MB
Ionic	35,6 MB	19,42 MB	144,8 MB	40,2 MB	203,5 MB	167,88 MB	119,8 MB	21,36 MB
Appc Titanium	40,6 MB	51,64 MB	117 MB	70,14 MB	145,1 MB	98,82 MB	88,7 MB	59,98 MB
React Native	38,8 MB	37,78 MB	96,1 MB	41,20 MB	136,5 MB	83,80 MB	70,5 MB	40,6 MB
Native Script	40,9 MB	58,14 MB	105,6 MB	67,64 MB	146,3 MB	96,02 MB	97,4 MB	64,30 MB
Flutter	46,6 MB	109,30 MB	76,4 MB	137,44 MB	93 MB	183,2 MB	64,4 MB	131,67 MB
Xamarin	34,8 MB	39,25 MB	99,4 MB	49,22 MB	121,2 MB	73,55 MB	72,3 MB	46,25 MB

4.4.8 Consumo de recursos - Bateria

Para a monitorização do consumo de recursos da Bateria no segundo cenário, foi avaliado qual o seu consumo durante cada tarefa ou passo executado. Os valores mostrados na Tabela 24 variam entre *Low*, *Medium*, *High* e *Very High*, indicados pelas ferramentas de *benchmark* em tempo real, e de seguida, a quantidade de bateria utilizada no final da realização dos testes [3] .

Tabela 24. Consumo de recursos - Bateria - Segundo Caso de Teste

Tecnologia	Nível de bateria utilizado		Quantidade de bateria utilizada	
	Android	**iOS**	**Android**	**iOS**
Angular	Low	Low	3,58 mAh	1/20
Vue	Low	Low	3,64 mAh	1/20
Cordova	Low/Medium	Low/Medium/High	1,41 mAh	3/20
Ionic	Low/Medium	Low/Medium/High	1,89 mAh	3/20
Appc Titanium	Medium	Medium/High	1,96 mAh	4/20
React Native	Low/Medium/High	Low/Medium	2,91 mAh	2/20
Native Script	Low/Medium	Low/Medium/High	2,87 mAh	4/20
Flutter	Low/Medium	Low/Medium/High	2,16 mAh	3/20
Xamarin	Low/Medium	Low/High	3,04 mAh	3/20

[3]Para o sistema iOS não é possível obter dados concretos como o sistema Android, a ferramenta indica um nível entre 0 a 20, no qual mostra a quantidade de energia que a aplicação está a usar, sendo assim números subjetivos, mas no entanto são considerados válidos [10].

5 Discussão

Este capítulo será apresentada uma discussão sobre os vários temas expostos nos capítulos anteriores e decisões tomadas ao longo da tese. Assim sendo, serão discutidos os vários métodos utilizados para a avaliação das tecnologias, desde a utilização de questionários à prova de conceito, em conjunto com as ferramentas *benchmark* utilizadas durante o processo dos casos de testes e monitorização de consumo de recursos. De seguida, os dados obtidos durante os testes serão discutidos à luz do estado da arte revisto, bem como, os problemas gerais que as tecnologias avaliadas apresentaram.

5.1 Métodos Utilizados

Durante a análise dos artigos revistos, (capítulo 2), verificou-se que muitos investigadores aplicaram diferentes metodologias para avaliar uma ou mais tecnologias, sendo que as mais utilizadas foram questionários, revisões sistemáticas, provas de conceito, métodos para casos de teste de desempenho, usabilidade, automatização de testes.

A maioria dos investigadores optam por utilizar questionários em vez de provas de conceito ou casos de teste, por estes permitirem uma colheita de grandes quantidades de dados sem a necessidade de realizar muito esforço de implementação. Usualmente, estes questionários são criados pelos autores, porem já existe uma vasta quantidade de questionários predefinidos, diminuindo assim o custo total da investigação. Contudo, as provas de conceito e casos de teste permitem obter dados mais significativos sobre a tecnologia escolhida, como por exemplo, o desempenho da aplicação, consumo de recursos, usabilidade, entre outros, proporcionando assim uma análise mais focada no desenvolvimento de uma aplicação e no produto final que a tecnologia é capaz de produzir, isto é, a aplicação que será utilizada no dispositivo. Para realizar uma análise individual e comparativa de várias tecnologias *mobile* Cross-Platform, como proposto nas contribuições desta dissertação (CONT2 e CONT3), seria necessário após investigar os vários métodos de análise utilizados em investigações, selecionar metodologias de avaliação que fosse em direção com os objetivos propostos. Assim sendo, foi decidido que seriam incluídos questionários, provas de conceito e casos de teste.

Através da implementação de um questionário, recolhemos uma grande quantidade de dados e informações relevantes sobre a utilização das tecnologias no dia a dia, que são importantes para conhecer quais são as tecnologias preferidas dos *developers* e quais os problemas que tenham encontrado durante toda a fase de desenvolvimento. Este método possibilitou identificar vários parâmetros que não seriam possíveis utilizando outro método de análise, como por exemplo, as provas de conceito. Contudo, apenas estas informações não eram suficientes para descobrir dados sobre métricas como forma de desenvolvimento, limitações e restrições impostas pela tecnologia, estabilidade da aplicação entre dispositivos e plataformas, entre outras métricas. Para conseguirmos adquirir estes dados foi realizada para cada tecnologia uma prova de conceito, que se baseava no desenvolvimento de uma aplicação *mobile* com vários requisitos, retirando métricas desde a fase de construção do ambiente de desenvolvimento até à criação do executável em modo de produção. A utilização deste método foi muito importante, pois veio a permitir recolher as métricas necessárias para uma avaliação sobre todo o desenvolvimento de uma aplicação utilizando aquela tecnologia, e também um conjunto de dificuldades e problemas que a tecnologia possui, sendo estes específicos ou presentes em várias tecnologias. Ainda assim, somente com a utilização do questionário e prova de conceito não foi possível retirar dados sobre o consumo de recursos, um ponto muito importante numa análise de uma tecnologia, sendo que demonstra como a aplicação criada pela tecnologia se

comporta num dispositivo real. Utilizando as aplicações finais que resultaram da prova de conceito, implementamos os casos de teste, que vieram a permitir obter dados significativos sobre o desempenho da aplicação, o consumo de recursos e usabilidade.

5.2 Ferramentas *Benchmark* Utilizadas

Para avaliar as características de desempenho de um software em um específico hardware, como o consumo de recursos, desempenho, entre outros, é necessário utilizar alguma ferramenta que permita monitorizar estes parâmetros. Ao realizar a revisão de literatura, capítulo 2, sobre o tema de análise de desempenho e ferramentas de monitorização de recursos, para o sistema Android, confirmou-se que existe uma grande variedade de ferramentas, fornecidas pela própria Google ou criadas por terceiros, no entanto para o sistema iOS, as escolhas são mais limitadas, sendo apenas utilizadas ferramentas nativas. Como o sistema Android possui um sistema Open-Source, o processo de *benchmark* é mais controlado e detalhado em comparação com outros sistemas, permitindo-se utilizar ferramentas criadas por terceiros, como por exemplo, o Power Tutor, Trepn Profiler da Qualcomm, entre outros, que facilita o processo de monitorização. Já para o sistema iOS que é Close-Source, o processo de *benchmark* é mais restrito em comparação ao Android, possuindo apenas algumas ferramentas para analisar o comportamento de hardware.

De forma a monitorizar o consumo de recursos, foram escolhidas as ferramentas Android Profiler e Google Chrome DevTools para monitorizar o sistema Android e a ferramenta XCode Instruments para os sistema iOS. Foram selecionadas ferramentas nativas para monitorizar o consumo de recursos pelo motivo que são fornecidas pelos SDK's dos sistemas Android e iOS, não dependendo assim de sistemas criados por terceiros, tornando assim todo o processo mais preciso e com menor percentagem de erro. No entanto, visto que as tecnologias PWA não produziam um ficheiro instalável, foi necessário utilizar uma ferramenta não nativa (Google Chrome DevTools).

A ferramenta Android Profiler possui um conjunto de ferramentas para encontrar problemas de desempenho, alocações de memória, entre outros problemas para aplicações do sistema Android. Durante a execução dos testes foram utilizadas as ferramentas CPU Profiler, Memory Profiler, Network Profiler e Energy Profiler que permitem visualizar diferentes recursos em tempo real e separando o consumo por aplicação que está a ser executada. Esta ferramenta fornece estes dados em forma de gráfico, no qual podemos escolher intervalos de tempo e um tempo exato. O XCode Instruments têm um comportamento muito semelhante ao Android Profiler, este também possui uma grande variedade de ferramentas para monitorizar o consumo de recursos. Para realizar esta monitorização foram utilizadas as ferramentas Activity monitor, Energy Log e CPU Activity Log que apresentam os dados também em tempo real, separando o consumo de recursos por um específico processo ou aplicação. No entanto, o XCode Instruments não necessita do ficheiro instalável para iniciar o processo de monitorização, enquanto a ferramenta Android Profiler precisa de um ficheiro que possa ser instalado no dispositivo. Devido a este problema que a ferramenta Android Profiler apresenta, foi necessário utilizar o Google Chrome DevTools para monitorizar o consumo de recursos para as aplicações das tecnologias PWA no sistema Android. A ferramenta Google Chrome DevTools, embora não seja tão precisa a monitorizar o consumo de recursos como as ferramentas nativas, até permite monitorizar de forma simples, vários tipos de recursos no dispositivo móvel em tempo real.

Um grande problema que todas estas ferramentas apresentam, é a falta de precisão do consumo de bateria em valores numéricos, isto é, valores em miliamperes. A precisão do consumo de bateria é bastante fiável em tempo real, no entanto, os valores expostos nos gráficos são valores subjetivos

como, Low, Medium, High, Very High que representam uma quantidade de energia que a aplicação está a utilizar naquele exato momento. Estes valores para o *developer* são importantes para verificar onde ocorrem os fluxos com maior carga e intensidade, no entanto torna-se difícil obter a quantidade de bateria utilizada durante todo o processo de teste. De forma a contornar este problema, como o Android permite verificar na área das aplicações o consumo de bateria por aplicação, foi utilizado esse valor numérico. Já o iOS como não permite verificar dados mais concretos como o Android, a ferramenta Xcode Instruments indica um nivel entre 0 a 20, que representa a quantidade de energia que a aplicação está a usar, sendo assim novamente números subjetivos, mas são considerados válidos pela Apple. Apesar destes problemas e limitações, consideramos que as ferramentas de *benchmark* escolhidas foram as mais adequadas para os objetivos da investigação propostos.

5.3 Resultados

Através da realização de uma revisão de literatura extensiva, conseguimos obter alguns dados sobre cada tipo de implementação e tecnologias antes de executar a metodologia. A revisão pormenorizada veio a permitir realizar uma comparação entre tecnologias, expondo alguns defeitos e particularidades que estas possuem, avaliando-as de uma forma geral sobre a sua utilidade, arquitetura, linguagens, licenças e suporte. Estes parâmetros muitas vezes são suficientes para uma equipa de *developers* tomar uma decisão sobre qual a tecnologia que vai utilizar, principalmente as linguagens que a tecnologia utiliza, pois estas têm um grande peso na decisão de uma tecnologia, pelo motivo que se os *developers* não dominarem estas linguagens todo o processo de desenvolvimento poderá se atrasar, causando enormes custos à empresa e ao projeto. No entanto, outras características também são importantes para se tomar uma decisão, como aspetos de qualidade que as tecnologias possuem, provenientes da sua arquitetura e organização de código. Uma das formas de verificar estes aspetos de qualidade foi executando a metodologia apresentada, retirando dados sobre várias categorias e parâmetros nos quais foram obtidos e selecionados através da realização da revisão de literatura, verificando quais os pontos mais importantes numa tecnologia de desenvolvimento *mobile* Cross-Platform, descritos no capítulo 3, secção 3.2.2.

Com a execução do questionário, conseguiu-se recolher dados sobre conhecimento, interesse e utilização das tecnologias mobile de *developers* à volta do mundo, que as utilizam no dia-a-dia. Muitos *developers* estão a utilizar cada vez mais e tem vindo a ter preferência na tecnologia NativeScript, como podemos verificar nos gráficos da Figura 48 e 49, no capítulo 4. No entanto, uma das tecnologias mais utilizadas para o desenvolvimento mobile é o Ionic (gráfico da Figura 34), pois este permite a integração de uma tecnologia Web, como o Angular, Vue ou React, para o desenvolvimento de uma aplicação mobile, facilitando assim todo o processo de criação do UI e reutilização de código de uma aplicação web para mobile. As questões mais importantes do questionário consistiam em recolher dados sobre alguns aspetos positivos ou negativos de cada tecnologia escolhida pelo participante. Com estas verificamos que grande parte das tecnologias forneciam uma boa experiência de utilizador e eram uma opção ótima para criar boas interfaces de utilizador, permitindo afirmar assim que qualquer destas tecnologias fornecem uma boa interface gráfica, porem algumas destas têm uma perda de desempenho em comparação com aplicações nativas.

Ao realizar a prova de conceito para cada tecnologia, conseguimos extrair dados objetivos e métricas sobre todo o desenvolvimento de uma aplicação *mobile* Cross-Platform em cada tecnologia, em conjunto com as dificuldades e problemas que a tecnologia possui. Muitos destes resultados já eram esperados, pois com a realização da revisão de literatura extensiva tornou-se possível já conhecer algumas limitações e problemas que poderiam surgir durante o desenvolvimento da prova

de conceito. Todas as provas de conceito foram construídas utilizado as boas práticas de programação e desenvolvimento de software, contudo algumas provas de conceito foram mais complicadas de desenvolver, devido à estrutura de código, ou ao enorme esforço de configuração do ambiente necessário para a tecnologia ser executada. Este também foi um enorme problema, porque é extremamente difícil manter na mesma máquina vários ambientes de desenvolvimento para várias tecnologias, pelas diferenças de versões entre o Java, XCode, entre outras dependências.

Sendo que os casos de teste foram aplicados na prova de conceito em cada tecnologia, com o intuito de obter dados significativos sobre o desempenho e consumo de recursos, todos estes testes foram executados de forma isolada, individual e num ambiente controlado, minimizando assim o impacto de fatores externos. Visto que os dados apresentados no capítulo 4, secção 4.4 são uma média de valores obtidos de cada execução do teste, conseguimos verificar que não existe grandes diferenças entre abordagens de arquitetura (*Webapp*, *Hybrid*, *Interpreted*, *Cross-Compiled*). Como podemos verificar, embora a abordagem *Cross-Compiled* compilar para código nativo, não apresenta grandes diferenças de consumos de recursos em comparação com as outras abordagens, pois este código nativo é gerado pela tecnologia, sendo o responsável por gerar um código eficiente. Apesar de a abordagem *Cross-Compiled* produzir uma aplicação muito semelhante a uma aplicação nativa e com um desempenho muito idêntico, verificou-se que o consumo de recursos depende da solução criada pela tecnologia e não da abordagem seguida.

5.4 Considerações sobre as tecnologias Avaliadas

Cada tecnologia escolhida foi avaliada através de várias metodologias de trabalho, na qual a metodologia que teve mais impacto foi a prova de conceito, onde permitiu recolher (na primeira pessoa) dados sobre todo o esforço de desenvolvimento necessário para criar uma aplicação utilizando aquela tecnologia, em conjunto com outras métricas que vieram a possibilitar criar uma avaliação individual completa elicitando uma série de recomendações para a seleção de uma tecnologia de desenvolvimento (CONT4).

5.4.1 Abordagem WebApp

A tecnologia Angular utiliza uma estrutura separada por módulos e componentes, que facultou uma melhor separação de código durante a prova de conceito. Esta apresenta muitos pontos positivos, tem um grande suporte da comunidade, faz uma recompilação de código muito rápida, facilita a criação de testes unitários e devido a sua estrutura permite escalar facilmente. No entanto, achamos que apresenta uma curva de aprendizagem muito longa, pois foi necessário aprender muitos conceitos sobre as diferenças entre componentes, páginas, módulos, services, entre outros, fazendo com que não fosse aconselhada um *developer* com pouco conhecimento em tecnologias web e/ou TypeScript.

Já a tecnologia Vue centra-se na renderização declarativa e na composição de componentes, fazendo com que a criação da prova de conceito fosse muito mais simples e ágil, tornando-se uma tecnologia muito útil para *developers* com pouca experiência. Contudo, notou-se que como esta tecnologia utiliza uma estrutura de código em árvore de componentes, ao escalar a aplicação, poderá ser necessário utilizar uma *store* como o VueX, devido as trocas de dados constantes entre vários componentes.

Estas tecnologias tornam muito rapidamente uma aplicação web em uma aplicação mobile, tudo através do mesmo código base, reutilizando inteiramente o código da aplicação. Se a aplicação não necessita de acessos das API's do dispositivo, estas tecnologias são uma boa escolha, sendo apenas

necessário incluir os service-workers, para permitir o funcionamento da aplicação sem ligação à Internet, contudo muitas funcionalidades nativas são perdidas, assemelhando-se a uma aplicação web normal.

5.4.2 Abordagem Híbrida

A construção da prova de conceito utilizando a tecnologia Cordova foi muito simples, pelo motivo que a estrutura de código e forma de desenvolvimento é bastante semelhante a uma aplicação web. No entanto, consideramos que esta tecnologia possui pouco suporte da comunidade, dificultando a resolução de erros sobre a sintaxe ou assuntos sobre a tecnologias, e da mesma forma, os erros de runtime são também difíceis de rastrear. O grande ponto positivo desta tecnologia é que cria uma solução extremamente pequena, e que consume poucos recursos do dispositivo *mobile*, porém a fase de debug num dispositivo *mobile* é muito problemática, sendo necessário utilizar outras ferramentas como o Google DevTools para verificar erros nos logs, não permitindo realizar testes unitários.

A tecnologia Ionic ao fornecer a opção da utilização de tecnologias web populares como TypeScript, Vue, React e Angular, permitiu transportar facilmente o código da prova de conceito em Angular para esta tecnologia, utilizando-se assim toda a estrutura de código já implementada sobre a interface de utilizador. Um outro ponto positivo é que esta tecnologia, ao trazer vários componentes de interface gráfica, manteve-se uma maior consistência entre vistas e componentes. O grande problema que foi encontrado ao realizar a prova de conceito foi que não existe qualquer tipo de documentação nem suporte da tecnologia Capacitor em conjunto com as tecnologias Vue e React, sendo apenas possível utilizar por agora em conjunto com o Angular.

Estas tecnologias permitem realizar uma aplicação mobile muito rapidamente, caso o *developer* tenha experiência com tecnologias web, contudo os grandes problemas desta abordagem é que estão sujeitas a vulnerabilidades na camada de abstração em JavaScript sendo necessário se preocupar com a segurança, e que estas como interpretam as tecnologias web numa base nativa chamada de WebView, perdem funcionalidades nativas de interface gráfica, como toques, navegação, entre outros, perdendo a sensação de uma aplicação *mobile* e assemelhando-se a uma aplicação web.

5.4.3 Abordagem *Interpreted*

A tecnologia Appcelerator Titanium ao facultar um SDK que possui várias funcionalidades nativas e uma estrutura MVC simplifica o desenvolvimento e manutenção de código. O ponto positivo que esta tecnologia apresenta é que todas as funcionalidades nativas como chamadas as API's do dispositivo, interfaces nativas, navegação, entre outras, estão incluídas num único SDK, fazendo com que não seja necessário incluir *Plug-in's* ou dependências para cada funcionalidade. No entanto, traz grandes problemas como: dificuldades em realizar Setup do ambiente de desenvolvimento; muito pouco suporte da comunidade na resolução de erros; necessário ter sempre em conta as diferenças entre SO's; e o maior problema é que o código é compilado utilizando uma chave num servidor, fazendo com não seja possível compilar nem testar a aplicação sem ligação à Internet.

Já a tecnologia React Native possui uma estrutura de código árvore de componentes, separada por *Components* e *Screens*, permitindo um estilo de programação funcional e por objetos. Esta tecnologia apenas traz as ferramentas necessárias e elementos de interface nativos, no qual para incluir aspetos de navegação e chamadas às API's do dispositivo é necessário incluir vários *Plug-in's*. Contudo, apesar de se tornar uma framework muito modelar, é necessário incluir muitos *Plug-in's* fazendo com que o tamanho do projecto fique extremamente grande, e devido a extensa

variedade, fica difícil de encontrar *Plug-in's* que fossem fiáveis e suportados. Caso o *developer* tenha conhecimento com a biblioteca React, esta tecnologia é a boa opção, mas caso contrário é uma tecnologia que não é indicada para aplicações simples.

Por fim, a tecnologia NativeScript separa os ficheiros de código pela sua responsabilidade e efetuando *bindings* para realizar referencias e dependências entre ficheiros. Notou-se que esta tecnologia possui enormes problemas a nível de manutenção e documentação, pelo motivo que as versões têm uma enorme diferença, fazendo com que seja extremamente difícil manter a aplicação entre versões, principalmente a nível de dependências e conflitos dos Plug-in's que deixam de funcionar, ou por não funcionarem correctamente em alguns dispositivos e/ou versões dos SO's. Apesar disso, é flexível por utilizar tecnologias web (como o Angular, Vue, React, TypeScript, entre outros) e possui uma área online onde permite criar e testar uma aplicação, facilitando a construção de protótipos e tutoriais.

Em contraste com a abordagem anterior, como estas tecnologias contêm um runtime autónomo, todas as funcionalidades nativas estão presentes na aplicação, permitindo criar uma aplicação com o aspecto e com funcionalidades totalmente nativas, desde a interface gráfica até as chamadas as API's do dispositivo. Um outro ponto positivo é que permitem ao *developer* incluir os seus próprios módulos nativos, caso seja necessário incluir uma funcionalidade extra que não esteja incluída na tecnologia.

5.4.4 Abordagem *Cross-Compile*

A framework Flutter têm uma estrutura de código é baseada em *Widgets*, formando assim uma arquitetura de árvore de *Widgets*. O Flutter possui um grande suporte da comunidade tanto na resolução de erros como na utilização de Plug-in's e é excelente no suporte de resolução de erros. Esta tecnologia é ideal para uma equipa que necessita de criar várias aplicações através do mesmo código base, fazendo com que a aplicação seja muito idêntica entre plataforma, contudo a estrutura de código poderá causar problemas de escalabilidade, devido à enorme quantidade de código necessário e a forma como os Widget's são construídos (código em cascata).

Por último, a tecnologia Xamarin permite a *developers* de Asp.NET criar aplicações *mobile* muito rapidamente. O Xamarin apresenta muitos pontos positivos, traz uma biblioteca que contém grande parte das funcionalidades nativas, permite utilizar conceitos de engenharia de software, proporcionando uma maior reutilização e escalabilidade da solução, e também, possibilita a utilização nativa de multi-threading, tirando maior partido dos recursos do dispositivo e aumentado o desempenho da aplicação. O grande problema encontrado durante o desenvolvimento é que não foi possível aceder a API do Calendário, devido a que a tecnologia não disponha esse acesso, nem existem *Plug-in's* com que permitiam o acesso a essa API.

Todas estas tecnologias tornam a solução criada numa aplicação completamente nativa. Todas as funcionalidades nativas tornam-se mais eficientes e fiáveis, não dependendo de uma camada abstrata como a abordagem anterior. No entanto, o principal componente desta abordagem é o compilador que produz o código que possa ser executado numa determinada plataforma, fazendo assim com que este seja o responsável por criar um código eficiente e com um bom desempenho.

5.4.5 Sumário

Pela grande variedade de tecnologias estudadas, consegue-se verificar que existe duas grandes diferenças a nível arquitetural, a escolha entre a utilização de um SDK que fornece todas as funcionalidades nativas num único lugar, ou o uso de *Plug-in's* que separa as funcionalidades em diferentes

bibliotecas. Podemos verificar que existem muitas tecnologias que utilizam os *Plug-in's*, como o Cordova, Ionic, React Native, NativeScript e Flutter, pelo motivo que torna a aplicação e o desenvolvimento modelar sendo apenas incluídas as funcionalidades necessárias na aplicação, contudo dificulta a manutenção da aplicação. Já o Appc Titanium utiliza um SDK que contém todas as funcionalidades necessárias, e por fim, o Xamarin que permite incluir um SDK e também vários *Plug-in's*. Um ponto comum entre várias tecnologias é a utilização de estruturas de código em árvores de componentes, facilitando a construção das interfaces de utilizador. Um outro ponto que está presente em grande parte das tecnologias estudadas, é a utilização de tecnologias e linguagens web, possibilitando assim a um *web developer* construir uma aplicação *mobile* sem muitos custos e sem ser necessário aprender uma nova linguagem.

5.4.6 Problemas

Um dos maiores problemas na área de software está na manutenção, e os impactos que estes têm na nossa solução ou aplicação. Encontraram-se vários problemas neste tópico, como por exemplo, a manutenção da prova de conceito em Native Script, que causou enormes dificuldades na resolução de dependências e até levando a novas alterações de código. Durante a realização da prova de conceito, algumas tecnologias introduziram novas versões, as quais dificultaram e atrasaram todo o processo de desenvolvimento. Este é um tópico que tem uma enorme importância porque é extremamente difícil manter uma aplicação *mobile* quando as novas versões implicam grandes modificações de código e estrutura, principalmente devido as novas versões dos SO's dos dispositivos.

É sempre necessário realizar atualizações nas tecnologias, principalmente pelos novos aspetos de segurança e para aceitar as novas versões dos SO's, no entanto quando estas alterações são demasiado rápidas, alguns módulos ou Plug-ins deixam de funcionar. Embora algumas tecnologias adoptem os *Plug-in's* por tornarem o desenvolvimento modular e flexível, incluindo assim apenas as funcionalidades necessárias, alguns problemas são evidenciados. Devido as novas versões e por grande parte dos *Plug-in's* serem criados pela comunidade, certas funcionalidades têm um limite de versão mínima do SO, podendo ficar obsoletas ou desatualizadas, fazendo com que novas versões dos SO's não sejam suportadas. Isto faz com que para tecnologias que utilizem uma base arquitetural modelar (com utilização de *Plug-in's*) seja extremamente importante antes de escolher o módulo a utilizar verificar a credibilidade do criador, quais as limitações que este possui e se irá ser mantido durante as novas versões da tecnologia e dos SO's.

6 Conclusão

Este capítulo será utilizado para apresentar um conclusão sobre esta dissertação, discutindo as contribuições inicialmente propostas, em conjunto com uma reflexão sobre todo o trabalho realizado incluindo as suas limitações. Por fim, será proposto o trabalho futuro.

6.1 Contribuições

Através da solução proposta na secção secção 1.2 do capítulo 1 para resolver os problemas identificados na secção 1.1 do capítulo 1, foi possível propor um total de cinco contribuições. Todas estas contribuições foram concluídas com sucesso, e serão discutidas de seguida.

Contribuição 1: Revisão de literatura sobre o desenvolvimento mobile Cross-Platform, avaliando e comparando as diferentes tecnologias.

A revisão de literatura forneceu uma base teórica necessária para que toda a metodologia pudesse ser implementada e discutida. Após a identificação e discussão das abordagens de desenvolvimento *mobile* nativo e Cross-Platform nas secções 2.1 e 2.3 do capitulo 2, identificamos e comparamos um conjunto de tecnologias que estão fortemente presentes no mercado, nas secções 3.1.2 e 3.1.3, do capitulo 2. Através desta avaliação a nível teórica, conseguimos expor alguns defeitos e particularidades que estas tecnologias possuem, avaliando-as de uma forma geral sobre a sua utilidade, arquitetura, linguagens, licenças e suporte. Estes parâmetros muitas vezes são suficientes para uma equipa de *developers* tomar uma decisão sobre qual a tecnologia que vai utilizar.

Contribuição 2: Análise individual de várias tecnologias Cross-Platform, utilizando um estudo de caso e ferramentas *benchmark*.

De forma a obter uma análise completa das tecnologias escolhidas, foi necessário incluir uma análise individual através de uma metodologia extensa, em conjunto com vários parâmetros de avaliação. Após o estudo de diversas metodologias de avaliação e análise, apresentadas nas secções 2.2 do capitulo 2, foi proposto nas secções 3.2.3, 3.2.5, 3.2.6 no capitulo 3, realizar um questionário sobre a utilização das tecnologias, uma prova de conceito a ser aplicada em cada tecnologia, casos de teste para avaliar a aplicação final criada pela tecnologia e ferramentas de monitorização para avaliar o consumo de recursos de cada aplicação, nos sistemas Android e iOS. Através desta metodologia conseguimos apresentar uma análise individual sobre cada tecnologia, sendo exposta no capitulo 4.

Contribuição 3: Análise comparativa entre várias tecnologias Cross-Platform.

Com a realização de uma análise individual completa das tecnologias escolhidas, tornou-se possível realizar comparações entre tecnologias. Como apresentado nos capítulos 4 e 5, recolhemos dados da implementação da metodologia, identificando e comparando assim várias métricas entre tecnologias. Sendo que o foco não é como a tecnologia funciona no momento, mas sim como esta se comporta e foi construída, esta abordagem permitiu-nos recolher aspetos positivos e negativos, nos quais são dependentes da sua arquitetura e forma de implementação, garantindo assim uma viabilidade dos dados durante várias versões.

Contribuição 4: Elicitar uma série de recomendações para a seleção de uma tecnologia de desenvolvimento mobile para um problema proposto.

Após a conclusão das contribuições anteriores, onde através da prova de conceito, permitiu recolher dados sobre todo o esforço de desenvolvimento necessário, em conjunto com outras métricas, conseguimos elicitar, indicar e reunir um conjunto de recomendações. Estas recomendações,

apresentadas na secção 5.4 do capitulo 5, são um conjunto de opiniões sobre os pontos positivos que as tecnologias apresentam, apontando também quais os problemas e limitações que o *developer* poderá encontrar ao utilizar esta tecnologias.

Contribuição 5: Propor um sistema futuro para, dado um problema, recomendar uma tecnologia Cross-Platform adequada

Através de todos estes dados recolhidos e de toda a análise efetuada, expostos nos capítulos 4 e 5, torna-se possível propor um sistema futuro que irá permitir através de uma simples descrição do problema, decidir qual a tecnologia Cross-Platform adequada. Dito isto, e utilizando todos estes dados, um sistema de recomendação com base no conhecimento e necessidades do *developer* ou da equipa de *developers* seria o ideal para indicar a melhor decisão perante um específico problema.

6.2 Reflexão

Ao finalizar esta dissertação, conseguimos obter dados significativos para resolver os problemas inicialmente propostos na secção 1.1 do capitulo 1, concluindo com sucesso todas as contribuições apresentadas anteriormente. Com toda esta informação sobre as várias tecnologias de desenvolvimento mobile estudadas, consideramos que este trabalho irá ajudar muitos *developers* a escolher uma tecnologia para um determinado problema. Ao apresentar todos os pontos positivos, problemas e limitações que as tecnologias apresentam, o *developer* que irá ter a responsabilidade de tomar uma escolha informada entre as tecnologias. Ao ler esta dissertação, um *developer* conseguirá entender quase tudo sobre cada tecnologia, deste a forma de programar, aos consumos de recursos, tornando assim essa escolha mais informada, conhecendo também os problemas e limitações que irá encontrar.

Ao realizar esta dissertação ganhei muitas competências na área de investigação, desde a enorme pesquisa de várias outras investigações e artigos, e efetuando comparações entre metodologias de avaliação para escolher qual a melhor abordagem para avaliar uma tecnologia. Posso afirmar que também aprendi imenso sobre a área de engenharia de software: utilizando e explorando todas as tecnologias estudadas; analisando em detalhe a arquitetura de todas as tecnologias; implementando uma gestão e separação de conteúdo de cada tecnologia estudada.

Mas também encontramos vários problemas, principalmente durante a implementação da prova de conceito. É extremamente difícil manter um ambiente de desenvolvimento para a grande variedade das tecnologias estudadas, as versões das dependências que as tecnologias utilizam são diferentes entre elas, fazendo com que seja necessário como por exemplo, três versões do Java. Uma outra dificuldade que tivemos foi conseguir conciliar a forma de desenvolvimento e estrutura de código de cada tecnologia. Muitas destas tecnologias são completamente diferentes, fazendo com que seja necessário estudar a tecnologia lendo a sua documentação e praticando com pequenas aplicações, tentando esquecer assim como funcionava a tecnologia anterior, de forma a seguir as boas práticas da tecnologia em questão.

6.2.1 Limitações

Esta dissertação apresenta algumas limitações, principalmente devido ao fator do tempo que não permitiu investigar e analisar mais profundamente sobre as tecnologias de desenvolvimento mobile estudados. Existiram algumas falhas durante a aplicação da metodologia proposta, as tecnologias de monitorização não permitem obter dados em tempo real da quantidade de bateria a ser consumida em miliamperes, fornecendo apenas valores subjetivos. Isto poderia ser melhorado adicionando uma outra ferramenta de monitorização, apenas focando-se no consumo de bateria em tempo real.

A principal limitação foi a prova de conceito ser muito pequena e pouco complexa, isto fez com que não fosse possível avaliar a escalabilidade da cada uma das tecnologias. Por fim, a última limitação que esta dissertação apresenta é que o consumo de recursos foi analisado utilizando as versões das tecnologias de desenvolvimento mobile Cross-Platform mais recentes até à data 20 de julho de 2021, fazendo que com futuras versões poderão alterar os dados do consumo de recursos.

6.3 Trabalho Futuro

Com todos dados obtidos nesta dissertação, poderão ser criados guias analíticos e estudos sobre estas ou outras tecnologias. Como trabalho futuro gostaria de criar um sistema de recomendação, que apenas utilizando uma descrição do problema e outros dados que fossem críticos para a equipa de *developers*, decidia qual a tecnologia de desenvolvimento mobile Cross-Platform mais adequada. Utilizando apenas algumas perguntas, como por exemplo, *É necessário utilizar alguma API nativa?*, *É necessário que a aplicação tenha uma UI com aspecto nativo?*, *Tem alguma linguagem de programação em preferência?*, entre outras perguntas, conseguiríamos propor ao utilizador qual a tecnologia mais adequada para o problema e critérios importantes. Dito isto, um sistema de recomendação com base no conhecimento e necessidades da equipa de *developers* seria o ideal para indicar a melhor decisão perante um específico problema.

Referências

[1] Adesina, G.: Mobile Operating Systems and Application Development Platforms: A Survey. International Journal of Advanced Networking and Applications **6**, 2195–2201 (Jan 2014)

[2] Adobe, I.: PhoneGap, https://phonegap.com/

[3] Adobe, I.: PhoneGap Explained Visually, https://phonegap.com/blog/2012/05/02/phonegap-explained-visually/

[4] Ahmad, A., Feng, C., Tao, M., Yousif, A., Ge, S.: Challenges of mobile applications development: Initial results. In: 2017 8th IEEE International Conference on Software Engineering and Service Science (ICSESS). pp. 464–469 (Nov 2017). https://doi.org/10.1109/ICSESS.2017.8342956, iSSN: 2327-0594

[5] Aldayel, A., Alnafjan, K.: Challenges and Best Practices for Mobile Application Development: Review Paper. In: Proceedings of the International Conference on Compute and Data Analysis. pp. 41–48. ICCDA '17, Association for Computing Machinery, New York, NY, USA (May 2017). https://doi.org/10.1145/3093241.3093245, https://doi.org/10.1145/3093241.3093245

[6] Amatya, S., Kurti, A.: Cross-Platform Mobile Development: Challenges and Opportunities. vol. 231, pp. 219–229 (Jan 2014)

[7] Android, D.: Arquitetura da plataforma | Desenvolvedores Android, https://developer.android.com/guide/platform?hl=pt-br

[8] Angulo, E., Ferre, X.: A Case Study on Cross-Platform Development Frameworks for Mobile Applications and UX. In: Proceedings of the XV International Conference on Human Computer Interaction. pp. 1–8. Interacción '14, Association for Computing Machinery, New York, NY, USA (Sep 2014). https://doi.org/10.1145/2662253.2662280, https://doi.org/10.1145/2662253.2662280

[9] Apple: Choosing a Membership - Support - Apple Developer, https://developer.apple.com/support/compare-memberships/

[10] Apple: Energy Efficiency Guide for iOS Apps: Measure Energy Impact with Instruments, https://developer.apple.com/library/archive/documentation/Performance/Conceptual/EnergyGuide-iOS/MonitorEnergyWithInstruments.html

[11] Apple: Running Your App in the Simulator or on a Device | Apple Developer Documentation, https://developer.apple.com/documentation/xcode/running_your_app_in_the_simulator_or_on_a_device

[12] Apple: Xcode, https://developer.apple.com/xcode/

[13] Axway: Home - Appcelerator, https://www.appcelerator.com/

[14] Axway: Products, https://www.appcelerator.com/mobile-app-development-products/

[15] Axway: Titanium Platform Overview, https://docs.axway.com/bundle/Titanium_SDK_allOS_en/page/titanium_platform_overview.html

[16] Basques, K.: Remote debug Android devices, https://developer.chrome.com/docs/devtools/remote-debugging/

[17] Bekkhus, M., Arvidsson, L.: Resource utilization and performance : A comparative study on mobile crossplatform tools (2020), http://urn.kb.se/resolve?urn=urn:nbn:se:hj:diva-50431

[18] Biolchini, J., Mian, P., Candida, A., Natali, C.: Systematic review in software engineering (01 2005)

[19] Biørn-Hansen, A., Grønli, T.M., Ghinea, G., Alouneh, S.: An Empirical Study of Cross-Platform Mobile Development in Industry. Wireless Communications and Mobile Computing **2019**, 1–12 (Jan 2019). https://doi.org/10.1155/2019/5743892

[20] Biørn-Hansen, A., Majchrzak, T.A., Grønli, T.M.: Progressive Web Apps: The Possible Web-native Unifier for Mobile Development. pp. 344–351 (Jan 2017). https://doi.org/10.5220/0006353703440351

[21] Biørn-Hansen, A., Rieger, C., Grønli, T.M., Majchrzak, T.A., Ghinea, G.: An empirical investigation of performance overhead in cross-platform mobile development frameworks. Empirical Software Engineering **25**(4), 2997–3040 (Jul 2020), https://doi.org/10.1007/s10664-020-09827-6

[22] Boushehrinejadmoradi, N., Ganapathy, V., Nagarakatte, S., Iftode, L.: Testing cross-platform mobile app development frameworks. In: Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering. pp. 441–451. ASE '15, IEEE Press, Lincoln, Nebraska (Nov 2015). https://doi.org/10.1109/ASE.2015.21, https://doi.org/10.1109/ASE.2015.21

[23] Böhm, S., Adam, F., Farrell, W.: Impact of the mobile operating system on smartphone buying decisions: A conjoint-based empirical analysis. vol. 9228, pp. 198–210 (08 2015)

[24] Casimirri (http://github.com/mcasimir), M.: Mobile Angular UI - Angular JS Mobile UI framework with Bootstrap 3 Css, http://mobileangularui.com/

[25] Catolino, G., Salza, P., Gravino, C., Ferrucci, F.: A set of metrics for the effort estimation of mobile apps. In: 2017 IEEE/ACM 4th International Conference on Mobile Software Engineering and Systems (MOBILESoft). pp. 194–198 (2017). https://doi.org/10.1109/MOBILESoft.2017.31

[26] Charaf, H.: Developing mobile applications for multiple platforms. In: 2011 Second Eastern European Regional Conference on the Engineering of Computer Based Systems. pp. 2–2 (2011). https://doi.org/10.1109/ECBS-EERC.2011.43

[27] Charkaoui, S., Adraoui, Z., Benlahmar, E.H.: Cross-platform mobile development approaches. In: 2014 Third IEEE International Colloquium in Information Science and Technology (CIST). pp. 188–191 (Oct 2014). https://doi.org/10.1109/CIST.2014.7016616, iSSN: 2327-1884

[28] Chrome, D.: Chrome DevTools, https://developer.chrome.com/docs/devtools/

[29] Ciman, M., Gaggi, O.: An empirical analysis of energy consumption of cross-platform frameworks for mobile development. Pervasive and Mobile Computing **39**, 214–230

(Aug 2017). https://doi.org/10.1016/j.pmcj.2016.10.004, http://www.sciencedirect.com/science/article/pii/S1574119216303170

[30] Ciman, M., Gaggi, O., Gonzo, N.: Cross-Platform Mobile Development: A Study on Apps with Animations (Mar 2014). https://doi.org/10.1145/2554850.2555104

[31] conceptdev: Xamarin documentation - Xamarin, https://docs.microsoft.com/en-us/xamarin/

[32] Corbalan, L., Fernandez, J., Cuitiño, A., Delia, L., Cáseres, G., Thomas, P., Pesado, P.: Development frameworks for mobile devices: a comparative study about energy consumption. In: Proceedings of the 5th International Conference on Mobile Software Engineering and Systems. pp. 191–201. MOBILESoft '18, Association for Computing Machinery, New York, NY, USA (May 2018). https://doi.org/10.1145/3197231.3197242, https://doi.org/10.1145/3197231.3197242

[33] Dalmasso, I., Datta, S.K., Bonnet, C., Nikaein, N.: Survey, comparison and evaluation of cross platform mobile application development tools. In: 2013 9th International Wireless Communications and Mobile Computing Conference (IWCMC). pp. 323–328 (Jul 2013). https://doi.org/10.1109/IWCMC.2013.6583580, iSSN: 2376-6506

[34] davidortinau: Building Cross Platform Applications Overview - Xamarin, https://docs.microsoft.com/en-us/xamarin/cross-platform/app-fundamentals/building-cross-platform-applications/overview

[35] Dehlinger, J., Dixon, J.: Mobile Application Software Engineering : Challenges and Research Directions (2011), /paper/Mobile-Application-Software-Engineering-%3A-and-Dehlinger-Dixon/3ffc4f867e45f8c8e9019726b351172ed77aca4f

[36] Delia, L., Galdámez, N., Thomas, P., Corbalán, L., Pesado, P.: Multi-platform mobile application development analysis. pp. 181–186 (May 2015). https://doi.org/10.1109/RCIS.2015.7128878

[37] Delía, L., Galdamez, N., Corbalan, L., Pesado, P., Thomas, P.: Approaches to mobile application development: Comparative performance analysis. In: 2017 Computing Conference. pp. 652–659 (Jul 2017). https://doi.org/10.1109/SAI.2017.8252165

[38] Dhillon, S., Mahmoud, Q.H.: An evaluation framework for cross-platform mobile application development tools. Software: Practice and Experience **45**(10), 1331–1357 (2015). https://doi.org/10.1002/spe.2286, https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.2286, _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/spe.2286

[39] Dorfer, T., Demetz, L., Huber, S.: Impact of mobile cross-platform development on CPU, memory and battery of mobile devices when using common mobile app features. Procedia Computer Science **175**, 189–196 (Jan 2020). https://doi.org/10.1016/j.procs.2020.07.029, http://www.sciencedirect.com/science/article/pii/S1877050920317099

[40] Dujmovi'c, J.: A Method For Evaluation And Selection Of Complex Hardware And Software Systems. In: CMG 96 Proceedings. pp. 368–378 (1996)

[41] Egli, P.: Software framework for web-based applications (Jun 2009), https://patents.google.com/patent/US7546576B2/en

[42] El-Kassas, W.S., Abdullah, B.A., Yousef, A.H., Wahba, A.M.: Taxonomy of Cross-Platform Mobile Applications Development Approaches. Ain Shams Engineering Journal **8**(2), 163–190 (Jun 2017). https://doi.org/10.1016/j.asej.2015.08.004, http://www.sciencedirect.com/science/article/pii/S2090447915001276

[43] Expo: Guides to get things done, https://docs.expo.io//guides

[44] Expo: Introduction to Expo, https://docs.expo.io//

[45] Expo: Limitations, https://docs.expo.io//introduction/why-not-expo

[46] Flutter, D.: Flutter - Beautiful native apps in record time, https://flutter.dev/

[47] Flutter, D.: Flutter architectural overview, https://flutter.dev/docs/resources/architectural-overview

[48] Fortunato, D., Bernardino, J.: Progressive web apps: An alternative to the native mobile apps. In: 2018 13th Iberian Conference on Information Systems and Technologies (CISTI). pp. 1–6 (2018). https://doi.org/10.23919/CISTI.2018.8399228

[49] Foundation, T.A.S.: Apache Cordova, https://cordova.apache.org/

[50] Foundation, T.A.S.: Architectural overview of Cordova platform - Apache Cordova, https://cordova.apache.org/docs/en/latest/

[51] Freitas Bernardes, T., Miyake, M.: Cross-platform Mobile Development Approaches: A Systematic Review. IEEE Latin America Transactions **14**, 1892–1898 (Apr 2016). https://doi.org/10.1109/TLA.2016.7483531

[52] Gerring, J.: What is a case study and what is it good for? The American Political Science Review **98**(2), 341–354 (2004), http://www.jstor.org/stable/4145316

[53] Goadrich, M.H., Rogers, M.P.: Smart smartphone development: iOS versus android. In: Proceedings of the 42nd ACM technical symposium on Computer science education. pp. 607–612. SIGCSE '11, Association for Computing Machinery, New York, NY, USA (Mar 2011). https://doi.org/10.1145/1953163.1953330, https://doi.org/10.1145/1953163.1953330

[54] Google: Angular, https://v2.angular.io/docs/ts/latest/guide/architecture.html

[55] Google: Angular - Introduction to the Angular Docs, https://angular.io/docs

[56] Google, D.: Introdução a atividades | Desenvolvedores Android, https://developer.android.com/guide/components/activities/intro-activities?hl=pt-br

[57] Grønli, T., Hansen, J., Ghinea, G., Younas, M.: Mobile application platform heterogeneity: Android vs windows phone vs ios vs firefox os. In: 2014 IEEE 28th International Conference on Advanced Information Networking and Applications. pp. 635–641 (2014). https://doi.org/10.1109/AINA.2014.78

[58] Gustavo Hartmann, Geoff Stead, A.D.: Cross-platform mobile development (2011), https://wss.apan.org/jko/mole/Shared%20Documents/Cross-Platform%20Mobile%20Development.pdf

[59] Heitkötter, H., Heierhoff, S., Majchrzak, T.A.: Evaluating Cross-Platform Development Approaches for Mobile Applications. vol. 140, pp. 120–138 (Jan 2013)

[60] Hernández, I.M.T., Viveros, A.M., Rubio, E.H.: Analysis for the design of open applications on mobile devices. In: CONIELECOMP 2013, 23rd International Conference on Electronics, Communications and Computing. pp. 126–131 (Mar 2013). https://doi.org/10.1109/CONIELECOMP.2013.6525772

[61] Hussain, A., Hashim, N.L., Nordin, N., Tahir, H.M.: A METRIC-BASED EVALUATION MODEL FOR APPLICATIONS ON MOBILE PHONES. Journal of Information and Communication Technology **12**, 55–71 (Apr 2013), http://e-journal.uum.edu.my/index.php/jict/article/view/8137

[62] Inc, A.: Instruments Overview - Instruments Help, https://help.apple.com/instruments/mac/8.0/#/dev7b09c84f5

[63] Ionic: Ionic - Cross-Platform Mobile App Development, https://ionicframework.com/

[64] Ionic: Ionic Framework - Ionic Documentation, https://ionicframework.com/docs/

[65] Jia, X., Ebone, A., Tan, Y.: A performance evaluation of cross-platform mobile application development approaches. In: Proceedings of the 5th International Conference on Mobile Software Engineering and Systems. pp. 92–93. MOBILESoft '18, Association for Computing Machinery, New York, NY, USA (May 2018). https://doi.org/10.1145/3197231.3197252, https://doi.org/10.1145/3197231.3197252

[66] Khandeparkar, A.: An Introduction to Hybrid Platform Mobile Application Development

[67] Kivy, D.: Architectural Overview — Kivy 1.11.1 documentation, https://kivy.org/doc/stable/guide/architecture.html

[68] Kivy, D.: Kivy: Cross-platform Python Framework for NUI, http://kivy.org/

[69] Korchi, A., Khachouch, M.K., Lakhrissi, Y., Moumen, A.: Classification of existing mobile cross-platform approaches. In: 2020 International Conference on Electrical, Communication, and Computer Engineering (ICECCE). pp. 1–5 (Jun 2020). https://doi.org/10.1109/ICECCE49384.2020.9179222

[70] Krosnick, J.A.: Survey research. Annual Review of Psychology **50**(1), 537–567 (1999). https://doi.org/10.1146/annurev.psych.50.1.537, https://doi.org/10.1146/annurev.psych.50.1.537, pMID: 15012463

[71] KumarVivek: Documentação do Microsoft Power Apps - Power Apps, https://docs.microsoft.com/pt-pt/powerapps/

[72] KumarVivek: Model-driven apps developer documentation - Power Apps, https://docs.microsoft.com/en-us/powerapps/developer/model-driven-apps/

[73] KumarVivek: O que é o Power Apps? - Power Apps, https://docs.microsoft.com/pt-pt/powerapps/powerapps-overview

[74] Kushlev, K.: Digitally connected, socially disconnected : can smartphones compromise the benefits of interacting with others? Ph.D. thesis, University of British Columbia (2015). https://doi.org/10.14288/1.0166492, https://open.library.ubc.ca/cIRcle/collections/ubctheses/24/items/1.0166492

[75] Latif, M., Lakhrissi, Y., Nfaoui, E.H., Es-Sbai, N.: Cross platform approach for mobile application development: A survey. pp. 1–5 (03 2016). https://doi.org/10.1109/IT4OD.2016.7479278

[76] Majchrzak, T.A., Biørn-Hansen, A., Grønli, T.M.: Progressive Web Apps: the Definite Approach to Cross-Platform Development? Hawaii International Conference on System Sciences 2018 (HICSS-51) (Jan 2018), https://aisel.aisnet.org/hicss-51/st/mobile_app_development/7

[77] Microsoft: Mobile App Development | Visual Studio, https://visualstudio.microsoft.com/vs/features/mobile-app-development/

[78] Microsoft: Xamarin | Open-source mobile app platform for .NET, https://dotnet.microsoft.com/apps/xamarin

[79] Mikusz, M.: Towards an Understanding of Cyber-physical Systems as Industrial Software-Product-Service Systems. Procedia CIRP **16**, 385–389 (2014). https://doi.org/https://doi.org/10.1016/j.procir.2014.02.025, https://www.sciencedirect.com/science/article/pii/S2212827114001462

[80] Motorola, S.: Rhomobile | Mobile API Compatibility, http://docs.rhomobile.com/en/2.2.0/rhoelements/apicompatibility

[81] tapanm MSFT: Descrição geral de criação de aplicações - Power Apps, https://docs.microsoft.com/pt-pt/powerapps/maker/

[82] olprod: Ferramentas de programação, documentação técnica e exemplos de código, https://docs.microsoft.com/pt-pt/

[83] OpenJs, F.: CLI Setup - NativeScript Docs, https://docs.nativescript.org/angular/start/quick-setup

[84] OpenJs, F.: Native mobile apps with Angular, Vue.js, TypeScript, JavaScript - NativeScript, https://www.nativescript.org/

[85] OpenJs, F.: NativeScript Playground, https://play.nativescript.org/

[86] OpenJs, F.: Technical Overview - NativeScript Docs, https://docs.nativescript.org/core-concepts/technical-overview

[87] Openjs.foundation, J.F.: jQuery Mobile, https://jquerymobile.com/

[88] OpenJs.foundation, J.F.: jQuery Mobile Demos, https://demos.jquerymobile.com/1.4.0/

[89] Palmieri, M., Singh, I., Cicchetti, A.: Comparison of cross-platform mobile development tools. In: 2012 16th International Conference on Intelligence in Next Generation Networks. pp. 179–186 (Oct 2012). https://doi.org/10.1109/ICIN.2012.6376023

[90] profexorgeek: What is Xamarin? - Xamarin, https://docs.microsoft.com/en-us/xamarin/get-started/what-is-xamarin

[91] Prolific: Prolific | Online participant recruitment for surveys and market research, https://prolific.co/

[92] Que, P., Guo, X., Zhu, M.: A Comprehensive Comparison between Hybrid and Native App Paradigms. In: 2016 8th International Conference on Computational Intelligence and Communication Networks (CICN). pp. 611–614 (Dec 2016). https://doi.org/10.1109/CICN.2016.125, iSSN: 2472-7555

[93] Raj, C.P.R., Seshu Babu Tolety: A study on approaches to build cross-platform mobile applications and criteria to select appropriate approach. In: 2012 Annual IEEE India Confe-

rence (INDICON). pp. 625–629 (Dec 2012). https://doi.org/10.1109/INDCON.2012.6420693, iSSN: 2325-9418

[94] Rawassizadeh, R.: Mobile Application Benchmarking Based on the Resource Usage Monitoring. IJMCMC **1**, 64–75 (Oct 2009). https://doi.org/10.4018/jmcmc.2009072805

[95] React, C.: Design Principles – React, https://reactjs.org/docs/design-principles.html

[96] React, C.: Direct Manipulation · React Native, https://reactnative.dev/docs/direct-manipulation

[97] React, C.: React Fundamentals · React Native, https://reactnative.dev/docs/intro-react

[98] React, C.: React Native, https://reactnative.dev/

[99] React, C.: React – A JavaScript library for building user interfaces, https://reactjs.org/

[100] React, C.: Team – React, https://reactjs.org/community/team.html

[101] React, C.: Tutorial: Intro to React – React, https://reactjs.org/tutorial/tutorial.html

[102] Rieger, C., Majchrzak, T.A.: Towards the definitive evaluation framework for cross-platform app development approaches. Journal of Systems and Software **153**, 175–199 (Jul 2019). https://doi.org/10.1016/j.jss.2019.04.001, http://www.sciencedirect.com/science/article/pii/S0164121219300743

[103] Runeson, P., Höst, M.: Guidelines for conducting and reporting case study research in software engineering. Empirical Software Engineering **14**(2), 131 (Dec 2008), https://doi.org/10.1007/s10664-008-9102-8

[104] Ryan, C., Rossi, P.: Software, performance and resource utilisation metrics for context-aware mobile applications. In: 11th IEEE International Software Metrics Symposium (METRICS'05). pp. 10 pp.–12 (Sep 2005). https://doi.org/10.1109/METRICS.2005.44, iSSN: 1530-1435

[105] Rösler, F., Nitze, A., Schmietendorf, A.: Towards a Mobile Application Performance Benchmark (Jul 2014)

[106] Saied, M., Ouni, A., Sahraoui, H., Kula, R., Inoue, K., Lo, D.: Improving reusability of software libraries through usage pattern mining. Journal of Systems and Software **145** (08 2018). https://doi.org/10.1016/j.jss.2018.08.032

[107] dos Santos, D.S., Nunes, H.D., Macedo, H.T., Neto, A.C.: Recommendation System for Cross-Platform Mobile Development Framework. In: Proceedings of the XV Brazilian Symposium on Information Systems. pp. 1–8. SBSI'19, Association for Computing Machinery, New York, NY, USA (May 2019). https://doi.org/10.1145/3330204.3330279, https://doi.org/10.1145/3330204.3330279

[108] Santos, D.S.d.: Sistema de recomendação de frameworks para desenvolvimento multiplataforma em dispositivos móveis (Aug 2018), https://ri.ufs.br/jspui/handle/riufs/10685, accepted: 2019-03-15T12:50:37Z Publisher: Pós-Graduação em Ciência da Computação

[109] Sendall, S., Kozaczynski, W.: Model transformation: the heart and soul of model-driven software development. IEEE Software **20**(5), 42–45 (2003). https://doi.org/10.1109/MS.2003.1231150

[110] Silva, M.S.: jquery mobile. Brasil: Novatec (2012)

[111] Silva, M.S.: jQuery - A Biblioteca do Programador JavaScript - 3ª Edição: Aprenda a criar efeitos de alto impacto em seu site com a biblioteca JavaScript mais utilizada pelos desenvolvedores web. Novatec Editora (Dec 2013), google-Books-ID: 7qujAwAAQBAJ

[112] Sim, S.E., Easterbrook, S., Holt, R.C.: Using benchmarking to advance research: a challenge to software engineering. In: 25th International Conference on Software Engineering, 2003. Proceedings. pp. 74–83 (2003). https://doi.org/10.1109/ICSE.2003.1201189

[113] Smutný, P.: Mobile development tools and cross-platform solutions. In: Proceedings of the 13th International Carpathian Control Conference (ICCC). pp. 653–656 (May 2012). https://doi.org/10.1109/CarpathianCC.2012.6228727

[114] Sommer, A., Krusche, S.: Evaluation of cross-platform frameworks for mobile applications (Mar 2013)

[115] Tandel, S., Jamadar, A.: Impact of progressive web apps on web app development (09 2018). https://doi.org/10.15680/IJIRSET.2018.0709021

[116] Taudes, A., Feurstein, M., Mild, A.: Options analysis of software platform decisions: A case study. MIS Quarterly **24**, 227–243 (06 2000). https://doi.org/10.2307/3250937

[117] Unity, T.: Powerful 2D, 3D, VR, & AR software for cross-platform development of games and mobile apps., `https://store.unity.com/`

[118] Unity, T.: Solutions | Unity, `https://unity.com/solutions`

[119] Unity, T.: Unity - Manual: Unity User Manual (2019.4 LTS), `https://docs.unity3d.com/Manual/index.html`

[120] Unity, T.: Unity for mobile | Unity, `https://unity.com/features/mobile`

[121] Unity, T.: Unity Real-Time Development Platform | 3D, 2D VR & AR Engine, `https://unity.com/`

[122] Vladimir, K.: Framework7 - Full Featured Framework For Building iOS, Android & Desktop Apps, `https://framework7.io/`

[123] Vladimir, K.: Framework7 Command-Line Interface, `https://framework7.io/cli/`

[124] Vladimir, K.: Introduction to Framework7 | Framework7 Documentation, `https://framework7.io/docs/introduction.html`

[125] Vue, D.: Introduction — Vue.js, `https://vuejs.org/v2/guide/`

[126] Vue, D.: Meet the Team — Vue.js, `https://vuejs.org/v2/guide/team.html`

[127] Vue, D.: Reactivity in Depth — Vue.js, `https://vuejs.org/v2/guide/reactivity.html`

[128] Vue, D.: Testing — Vue.js, `https://vuejs.org/v2/guide/testing.html`

[129] Vue, D.: Vue.js, `https://vuejs.org/`

[130] Willocx, M., Vossaert, J., Naessens, V.: A Quantitative Assessment of Performance in Mobile App Development Tools. In: 2015 IEEE International Conference on Mobile Services. pp. 454–461 (Jun 2015). https://doi.org/10.1109/MobServ.2015.68, iSSN: 2329-6453

[131] Willocx, M., Vossaert, J., Naessens, V.: Comparing performance parameters of mobile app development strategies. In: Proceedings of the International Conference on Mobile Software Engineering and Systems. pp. 38–47. MOBILESoft '16, Association for Computing Machinery, New York, NY, USA (May 2016). https://doi.org/10.1145/2897073.2897092, https://doi.org/10.1145/2897073.2897092

[132] Xanthopoulos, S., Xinogalos, S.: A comparative analysis of cross-platform development approaches for mobile applications. In: Proceedings of the 6th Balkan Conference in Informatics. pp. 213–220. BCI '13, Association for Computing Machinery, New York, NY, USA (Sep 2013). https://doi.org/10.1145/2490257.2490292, https://doi.org/10.1145/2490257.2490292