Desenvolvimento móvel em Swift, Java e React Native: uma avaliação experimental em áudioguias

Mobile development in Swift, Java and React Native: an experimental evaluation in audioguides

Hugo Brito, Álvaro Santos, Jorge Bernardino, Anabela Gomes
Coimbra Polytechnic - ISEC
Coimbra, Portugal
a21220118@isec.pt, ans@isec.pt, jorge@isec.pt, anabela@isec.pt

Resumo — **O desenvolvimento móvel está cada vez mais integrado nas empresas de desenvolvimento de *software*. Atualmente os clientes esperam aplicações com bom desempenho em menos tempo e com menor custo. Já não é novidade a procura por uma solução híbrida que permita o desenvolvimento de aplicações para ambas as plataformas com o mesmo código. O React Native é uma *framework* recente que surgiu com o intuito de colmatar as lacunas apresentadas por soluções anteriores, aliando o desenvolvimento em JavaScript, que neste momento é considerada a melhor solução para desenvolvimento *frontend*. Neste artigo é feita uma análise comparativa do tempo de desenvolvimento das principais funcionalidades de uma aplicação do tipo áudioguia, para Java (Android), Swift (iOS) e React Native (Android e iOS). Este trabalho indica que o Swift é a solução que apresenta o tempo de desenvolvimento mais baixo, enquanto o React Native e o Java apresentam tempos de desenvolvimento totais muito idênticos, contudo com a particularidade do React Native dar suporte aos dois principais sistemas operativos móveis.**

Palavras Chave – JavaScript, Android, iOS, React Native.

Abstract — **Mobile development is increasingly integrated into software development companies. Customers are currently expecting applications with good performance in less time and at lower cost. The demand for a hybrid solution which allows the development of applications for both platforms with the same code is no longer new. React Native is a recent framework that has emerged to bridge the gaps presented by previous solutions, combining JavaScript development, which is now considered the best solution for frontend development. In this article, a comparative analysis of the development time of the main functionalities of an audio-guide application for Java (Android), Swift (iOS) and React Native (Android and iOS) is made. This work shows that Swift is the solution with the lowest development time, while React Native and Java have very similar overall development times, however with the particularity that React Native supports both the two main mobile operating systems.**

Keywords - JavaScript, Android, iOS, React Native.

I. INTRODUÇÃO

Uma aplicação móvel é um software projetado para ser executado em *smartphones*, *tablets* ou outros dispositivos móveis [1]. No entanto, para que seja bem sucedida, uma aplicação deverá permitir um envolvimento interativo com o utilizador de forma o mais expedita, intuitiva e portátil possível [2]. Originalmente, a maior parte das aplicações móveis foi concebida para fins informativos e de produtividade como, por exemplo, *e-mail*, calendário, contatos ou calculadora. Com a rápida evolução da tecnologia, o desenvolvimento das aplicações móveis foi expandido para outras categorias, tais como, jogos, localização (GPS) ou redes sociais. Atualmente é possível realizar quase todas as operações do dia-a-dia através de aplicações móveis, as quais oferecem funcionalidades idênticas ou superiores às aplicações tradicionais usadas nos PC, mas podendo ser acedidas em qualquer lugar, tirando ou não partido de uma ligação à Internet.

O número de aplicações publicadas vai aumentando a cada semestre [3] e, devido ao crescimento do mercado das aplicações móveis, as empresas querem construir aplicações mais eficientes e com mais funcionalidades, mas ao mesmo tempo querem manter a mesma sensação de desempenho de qualquer outra aplicação atualmente no mercado.

O grande problema surge quando se inicia o desenvolvimento e se percebe que é necessário configurar vários ambientes de desenvolvimento e aprender a programar duas ou mais linguagens totalmente distintas, Java ou Kotlin para Android e Objective-C ou Swift para iOS. Isto leva a que muitas empresas tenham equipas distintas de desenvolvimento Android e iOS, sendo o conhecimento dos membros normalmente limitado a uma plataforma. Embora o nível de especialização que se obtém nessas equipas seja benéfico em muitas situações, se fosse possível desenvolver aplicações Android e iOS numa única equipa e num único projeto isso poderia reduzir substancialmente o tempo e custos de desenvolvimento. Numa tentativa de mitigar estes problemas e agilizar o processo de desenvolvimento, tem-se assistido ao aparecimento de soluções híbridas e *cross-platform* que permitem atenuar alguns destes problemas. Em alguns casos ajudando na agregação de equipas para desenvolvimento conjunto num único projeto, o qual permitirá ter a aplicação disponível em vários sistemas operativos de plataformas móveis.

Neste contexto, já não são novidade as *frameworks* híbridas baseadas nas tecnologias *web* que permitem desenvolver

2019 14th Iberian Conference on Information Systems and Technologies (CISTI)
19 – 22 June 2019, Coimbra, Portugal
ISBN: 978-989-98434-9-3

aplicações para sistemas diferentes, mas utilizando o mesmo código-fonte. O Phone Gap, Appcelerator ou Ionic, já existem há algum tempo, mas normalmente ficam aquém do exigido para os programadores, quer em termos de desempenho quer de experiência com o utilizador [4], [5], [6]. Esta situação surge porque existe uma *webview* intermédia que faz uma adaptação do código em HTML e CSS para executar no dispositivo móvel.

Num estudo anterior [6] foi realizada uma análise comparativa entre soluções híbridas JavaScript, nomeadamente React Native, NativeScript e Ionic, baseada em sete critérios considerados mais relevantes para o desenvolvimento de aplicações móveis. Este estudo concluiu que o React Native apresentava-se como a solução mais vantajosa para o desenvolvimento de aplicações móveis [6]. Assim sendo, a *framework* escolhida para esta avaliação experimental recaiu sobre o React Native. Devido à *framework* ser recente, não foram encontrados estudos relacionados que comparem o tempo de desenvolvimento em React Native relativamente às duas linguagens atualmente mais utilizadas para desenvolvimento nativo (Java em Android e Swift em iOS).

O React Native [7], criado pelo Facebook, tem como objetivo fornecer uma experiência de utilização similar à das aplicações nativas, tornando a execução possível em Android e iOS, sem ter de recorrer a uma *webview* intermédia para fazer toda a adaptação do código-fonte. Esta *framework* é bastante recente, mas vista como a solução de desenvolvimento híbrido mais emergente atualmente.

Antes de aparecer o React Native foi criado o ReactJS, que é uma *framework* de JavaScript construída pelo Facebook, a qual foi lançada no ano de 2013 [8]. O ReactJS é uma *framework* que vai ao encontro da visão principal de desenvolvimento do Facebook [9], que é a de reduzir o tempo de desenvolvimento, optando pela criação de código reutilizável e eficiente. Desta forma o ReactJS tem por objetivo principal a divisão de todas as vistas em diferentes componentes para ser possível a reutilização de código para outras vistas do projeto ou até mesmo em projetos totalmente diferentes, levando a um ciclo de desenvolvimento mais rápido. Em 2015 e após dois anos do lançamento do ReactJS, na React.js Conf 2015, Tom Occhino [10] anunciou que, com o sucesso que o ReactJS estava a ter no desenvolvimento de aplicações para a *web*, iria ser preparada uma solução para desenvolvimento de aplicações móveis, o React Native. A ideia era a de acabar com a desconexão total no desenvolvimento entre Android e iOS, mas sem a necessidade de recorrer à utilização de *webviews* [10]. O React Native teve a sua primeira aparição em 2015, contudo, em versões com algumas lacunas. Esta situação deve-se ao facto de existir uma maior dificuldade em criar uma *framework* de desenvolvimento híbrido sem recorrer às já mencionadas *webviews*. Atualmente ainda não existe uma versão estável da *framework*, contudo a maior parte dos problemas já foi ultrapassada, estando a *framework* a entrar numa fase de estabilização e com poucas alterações. A lógica da aplicação é gravada em JavaScript enquanto a vista é totalmente nativa, havendo sempre a possibilidade de implementação de código nativo para cada uma das plataformas.

Atualmente, com o grande ecossistema que existe de JavaScript em *frontend web* [11], são cada vez mais procuradas soluções para *backend*, *mobile* ou *desktop* para JavaScript, daí a terem surgido novas alternativas para desenvolvimento móvel de forma híbrida em JavaScript como é o caso deste estudo que compara o React Native com as linguagens as duas principais de desenvolvimento nativo tanto para Android como para iOS.

O estudo realizado é reportado neste documento da seguinte forma. Na seção II é apresentado o tipo de aplicação em que se vai basear a comparação e a metodologia utilizada para a comparação entre propostas de soluções. Na seção III são apresentados os resultados do desenvolvimento das funcionalidades e tarefas para cada uma das plataformas de desenvolvimento e na seção IV é feita uma discussão dos resultados. Por fim na seção V são apresentadas as conclusões e propostas algumas ideias para trabalho futuro.

II. METODOLOGIA DA AVALIAÇÃO EXPERIMENTAL

Para ser feita uma comparação das vantagens e desvantagens do desenvolvimento híbrido em React Native com o desenvolvimento nativo em Java e Swift foram criados três projetos distintos para a mesma solução, neste caso concreto a criação de áudioguias.

No caso do desenvolvimento nativo foi criada uma aplicação em Swift, para o sistema operativo iOS, e a mesma aplicação em Java, para executar no sistema operativo Android. Na criação das aplicações esteve sempre presente o objetivo de replicar ao máximo as aplicações para oferecer as mesmas funcionalidades, independentemente do sistema operativo usado pelo utilizador final da aplicação. Este é um cuidado normalmente associado ao desenvolvimento nativo que no desenvolvimento híbrido não acontece. Posteriormente, foi desenvolvida em React Native uma aplicação para o mesmo modelo de negócio, com as mesmas funcionalidades das anteriores. A escolha do tipo de aplicação, áudioguias, justifica-se pelo facto de já estarem a ser trabalhadas num contexto empresarial. Justifica-se também pelo facto de estar a ser realizada uma mudança de paradigma do desenvolvimento da própria empresa, de soluções nativas para uma solução híbrida. Assim, a escolha das áudioguias para projeto inicial permite fazer uma comparação das vantagens e desvantagens de implementação nas diferentes linguagens/*frameworks*.

Uma áudioguia tem como principal objetivo auxiliar um turista de uma região ou visitante de museus/exposições a ter uma visita guiada de forma autónoma. Estes serviços podem ser totalmente gratuitos ou, em alguns casos, com custos mais baixos em relação a um guia turístico tradicional. As funcionalidades mais comuns de uma áudioguia são:

- **Mapa**: serve para localizar e/ou dar orientações gerais ao utilizador;
- **Pontos de interesse (POI)**: permite identificar zonas de interesse para o utilizador, recorrendo a descrições, galeria de imagens ou áudio;
- **Visitas ou orientações**: permite a obtenção de orientações de navegação até um determinado POI;

ISBN: 978-989-98434-9-3

- **Favoritos**: possibilita o regresso a um determinado POI de forma mais simples e direta;
- **Pesquisa**: permite efetuar pesquisas através de texto ou informações contextuais (p. ex., QRCode), dando a possibilidade de aceder aos POI mais rapidamente.

Para realizar a comparação entre desenvolvimento nativo e híbrido em React Native foram selecionadas algumas das funcionalidades mais importantes na implementação, para o modelo de negócio em questão. Posteriormente, foi feito o desenvolvimento em todas as linguagens/*frameworks* consideradas neste estudo de forma a serem comparados os tempos de desenvolvimento e as vantagens e desvantagens associadas ao desenvolvimento de cada uma das aplicações. As funcionalidades selecionadas foram divididas em tarefas mais pequenas. Assim, as funcionalidades e tarefas analisadas para a comparação foram:

- **Galeria e *swipe* de imagens**: os aspetos analisados nesta funcionalidade são os relativos à manipulação de imagens, desde o *download* a partir de um servidor, até à criação da vista de galeria de um conjunto de imagens em lista. As tarefas associadas a esta funcionalidade são:
    - Obtenção de uma lista de imagens através de uma REST API;
    - Armazenamento local de imagens para posterior utilização em modo *offline*;
    - Criação de galeria de imagens que permita a navegação através de gestos do tipo *swipe*.

- **Leitura de QRCodes e navegação:** nesta funcionalidade são analisados os aspetos relativos à gestão de permissões da câmara, para leitura e descodificação de QRCodes para texto, e navegação para uma vista relacionada com os dados lidos através do QRCode, sendo esta funcionalidade dividida nas seguintes tarefas:
    - Configuração de permissões associadas à câmara;
    - Leitura de QRCode para formato texto;
    - Navegação para a vista associada ao QRCode identificado.

- **Gestão de favoritos:** com a implementação dos favoritos é possível analisar a facilidade de mudança de uma vista consoante a ação do utilizador (tirar ou remover dos favoritos) e, também, a facilidade de implementar essas repercussões de alteração de dados globais a outras vistas que utilizam os mesmos dados, sendo que as duas tarefas analisadas associadas a esta funcionalidade são:
    - Atualização da vista após mudança de estado de favoritos;
    - Atualização de vistas globais após mudança de estado numa vista particular.

- **Carregamento e execução de áudio:** nesta funcionalidade é observada a forma de manipulação de ficheiros de áudio, desde o seu *download* até à sua execução e controlo, dando origem à análise das seguintes tarefas:
    - Obtenção de ficheiro de áudio através de uma REST API;
    - Execução de ficheiro de áudio;
    - Controlo da execução de áudio;
    - Vista associada ao estado de reprodução do ficheiro de áudio.

- **Lista dinâmica de dados:** com esta funcionalidade é possível verificar a facilidade de implementação de listas de dados, que incluam uma imagem alusiva de um ponto e texto com o nome do ponto. As tarefas analisadas foram:
    - Criação de lista com itens de imagem e texto;
    - Reutilização da mesma lista para implementação numa outra vista distinta.

- **Obtenção de dados e armazenamento:** nesta funcionalidade é investigada a interação com uma REST API, para obtenção de dados e facilidade de armazenamento dos dados para posterior utilização em modo *offline*, dando origem à análise das seguintes tarefas:
    - Preparação do ambiente para suportar armazenamento de modelos de dados;
    - Criação de um primeiro modelo de dados referente a pontos de interesse;
    - Obtenção de dados através de uma REST API.

III. RESULTADOS

O desenvolvimento das aplicações foi realizado seguindo a metodologia SCRUM, que é uma estrutura que permite abordar problemas adaptativos de forma iterativa e incremental [12]. Durante o desenvolvimento, todos os tempos associados às várias tarefas foram registados logo após o seu término. Assim, foi possível obter os tempos para todas as soluções analisadas na comparação. Os resultados em tempo, vantagens e desvantagens para cada uma das soluções foram os seguintes:

- **Galeria e *swipe* de imagens**: na criação de uma galeria de imagens o React Native obteve vantagens na configuração do método de obtenção de ficheiros via REST API, devido a todo o desenvolvimento em React Native ser baseado em objetos JSON, ao contrário das soluções nativas que apresentam maior dificuldade no mapeamento de dados em JSON. Enquanto as soluções nativas apresentam facilidade em realizar *caching* de imagens o React Native necessita de uma biblioteca externa (react-native-fs) que complementa o *caching* de imagens de forma a ser compatível com as duas plataformas (Android e iOS). Na criação da vista de lista de imagens, com possibilidade de navegação através de gestos de *swipe*, os tempos obtidos foram aproximados, tal como demonstra a Tabela 1. Para as três soluções consideradas existe bastante documentação associada ao componente de *swipe* de imagens.

ISBN: 978-989-98434-9-3

TABELA 1. TEMPOS DE DESENVOLVIMENTO ASSOCIADOS À FUNCIONALIDADE DE GALERIA E SWIPE DE IMAGENS

Tarefa	Tempo de desenvolvimento por solução		
	Swift	***Java***	***React Native***
Obtenção de uma lista de imagens através de uma REST API	4h	5h	1h
Armazenamento local de imagens para posterior utilização	0h 30m	0h 30m	3h
Criação de vista dinâmica de galeria de imagens com funcionalidade de *swipe*	3h	3h 45m	4h

- **Leitura de QRCodes e navegação:** na gestão das permissões para acesso à câmara fotográfica do dispositivo, no Swift basta configurar/disponibilizar uma *string* com o pedido a ser apresentado ao utilizador e o sistema operativo gere as permissões diretamente em *runtime*. O Android a nível de permissões exige um processo mais complicado, sendo tratadas de diferentes formas consoante a versão. No caso de configuração de permissões, o desenvolvimento híbrido em React Native não apresenta vantagens em relação ao desenvolvimento nativo devido a ter de tratar os dois sistemas operativos de forma separada conforme as suas particularidades. A descodificação de um QRCode para formato de texto e a navegação para a vista associada ao QRCode revelaram tempos de desenvolvimento idênticos para cada uma das soluções. Os tempos finais de desenvolvimento para esta funcionalidade são apresentados na Tabela 2.

TABELA 2. TEMPOS DE DESENVOLVIMENTO ASSOCIADOS À FUNCIONALIDADE DE LEITURA DE QRCODES E NAVEGAÇÃO

Tarefa	Tempo de desenvolvimento por solução		
	Swift	***Java***	***React Native***
Configuração de permissões associadas à câmara	0h 15m	2h	3h 30m
Leitura e descodificação de QRCode para formato texto	3h	4h	3h 30m
Navegação para a vista associada ao QRCode identificado	2h	0h 30m	0h 45m

- **Gestão de favoritos:** o React Native apresenta maior rapidez na criação de métodos que utilizam diretamente o modelo de dados, pois armazena diretamente os dados em JSON. As linguagens nativas, após uma boa definição da arquitetura do projeto, permitem que a atualização global das vistas após a mudança num modelo de dados seja feita de forma agilizada e automática. No caso do React Native é necessário utilizar o ciclo de vida do estado do componente para comparação dos dados que vêm do modelo (*store*). O React Native ao basear-se numa visão de desenvolvimento de máxima reutilização, apresenta a desvantagem de ser difícil em certos momentos controlar a imutabilidade dos dados entre modelo e vistas que já foram instanciadas. Os tempos finais de desenvolvimento para esta funcionalidade são apresentados na Tabela 3.

TABELA 3. TEMPOS DE DESENVOLVIMENTO ASSOCIADOS À FUNCIONALIDADE DE GESTÃO DE FAVORITOS

Tarefa	Tempo de desenvolvimento por solução		
	Swift	***Java***	***React Native***
Atualização da vista após mudança de estado de favoritos	2h	2h 30m	1h 30m
Atualização global após mudança de estado numa vista particular	0h	0h	2h 30m

- **Carregamento e execução de áudio:** o desenvolvimento nativo apresentou maior lentidão na obtenção de ficheiros de áudio devido a algumas extensões de áudio. Por exemplo, os ficheiros ".wav" obrigam à realização de configurações extras dos metadados da API, dificuldade essa que o React Native não apresentou. A manipulação de áudio apresentou-se mais rápida em Swift e em React Native, devido a permitirem trabalhar de modo simplificado com a *Event Based API* (arquitetura de cliente/servidor com interações baseadas em eventos), algo que foi mais lento de implementar em Java. A nível de controlo da vista consoante o progresso do áudio, o desenvolvimento em React Native foi mais demorado devido a uma maior dependência de bibliotecas externas, relativamente às plataformas nativas, e também pela menor documentação fornecida. Os tempos finais de desenvolvimento para esta funcionalidade são apresentados na Tabela 4.

TABELA 4. TEMPOS DE DESENVOLVIMENTO ASSOCIADOS À FUNCIONALIDADE DE CARREGAMENTO E EXECUÇÃO DE ÁUDIO

Tarefa	Tempo de desenvolvimento por solução		
	Swift	***Java***	***React Native***
Obtenção de ficheiros de aúdio através de uma REST API	8h	6h	3h 30m
Execução de ficheiro de aúdio	1h	2h	1h 30m
Controlo da execução do ficheiro de áudio	2h	2h 30m	2h 30m
Vista associada ao estado de reprodução do ficheiro de aúdio	6h	7h	9h

- **Lista dinâmica de dados:** o sistema operativo iOS com o Swift permite reutilização e criação de *rows* (item de uma lista de dados iterativa), sendo rápido implementar listas dinâmicas. No Java, através dos adaptadores, também é rápido implementar a lista, contudo para o mesmo tipo de lista tornou-se um processo ligeiramente mais demorado em comparação com os tempos do Swift. O React Native apresentou o desenvolvimento de listas dinâmicas mais rápido devido a utilizar um componente de *flatlist* que executa o mesmo item da lista (*row*) como um componente independente, fazendo com que seja apenas necessário fornecer um JSON como propriedade da *flatlist* e o componente que a mesma vai instanciar. A nível de reutilização, todas as soluções permitiram uma reutilização da mesma lista noutra vista, de forma rápida e eficaz. Os

ISBN: 978-989-98434-9-3

tempos finais de desenvolvimento para esta funcionalidade são apresentados na Tabela 5.

TABELA 5. TEMPOS DE DESENVOLVIMENTO ASSOCIADOS À FUNCIONALIDADE DE LISTAS DINÂMICAS DE DADOS

Tarefa	Tempo de desenvolvimento por solução		
	Swift	*Java*	*React Native*
Criação de lista com itens de imagens e texto	1h	1h 30m	0h 45m
Reutilização da mesma lista para implementação em outra vista distinta	0h 15m	0h 15m	0h 15m

- **Obtenção de dados e armazenamento:** o Swift e o Java apresentaram tempos de desenvolvimento idênticos na preparação de ambiente de persistência e obtenção de dados através das bibliotecas já utilizadas, existindo bastante documentação associada. A principal dificuldade, não só no Swift como também no Java, foi o mapeamento dos dados de uma REST API em JSON para as devidas estruturas. Pelo contrário, o React Native apresentou tempo de desenvolvimento inverso, pois o React é focado no desenvolvimento de vistas e isso leva a ter de existir instalações e integrações para persistência de dados, como por exemplo o Redux, levando a um maior tempo de preparação do ambiente de armazenamento de dados. A obtenção de dados através de uma REST API e o seu mapeamento são feitos de forma rápida, pois os modelos de dados (*reducers*) em React Native são armazenados no formato de JSON diretamente. Os tempos finais de desenvolvimento para esta funcionalidade são apresentados na Tabela 6.

TABELA 6. TEMPOS DE DESENVOLVIMENTO ASSOCIADOS À FUNCIONALIDADE DE OBTENÇÃO DE DADOS E ARMAZENAMENTO

Tarefa	Tempo de desenvolvimento por solução		
	Swift	*Java*	*React Native*
Preparação do ambiente para suportar armazenamento de modelos de dados	6h	5h 30m	9h
Criação de um primeiro modelo de dados referente a pontos de interesse	0h 15m	0h 30m	1h 15m
Obtenção de dados referentes através de uma REST API	3h 30m	4h	1h 30m

IV. DISCUSSÃO DOS RESULTADOS

Neste trabalho fornecemos a comparação em termos de tempo de implementação de três diferentes soluções para programação de aplicações móveis. As soluções foram analisadas através do tempo de implementação do mesmo tipo de projeto (áudioguias).

Os resultados obtidos neste estudo estão de acordo com as conclusões alcançadas em outros estudos comparativos realizados entre as soluções nativas, Java e Swift, em que relatam que o desenvolvimento em Java para Android pode ser mais lento entre 20% a 30% em relação ao Swift para iOS [13]. Na análise realizada o Java apresentou-se cerca de 11.1% mais lento do que o Swift.

Como foi constatado, a solução que apresentou menor tempo de desenvolvimento foi o Swift, notando-se também, a par com o React Native, que a quantidade de código feito era consideravelmente menor comparando com o Java, levando a maior facilidade de manuseamento em classes com complexidade de lógica maior. Esta situação justifica em parte a pretensão da Google em transferir o desenvolvimento Android para Kotlin, uma linguagem muito mais próxima em termos de sintaxe com o JavaScript do React Native e o Swift do iOS.

Na Tabela 7 são apresentados os tempos totais das funcionalidades desenvolvidas para todas as soluções.

TABELA 7. TEMPOS DE DESENVOLVIMENTO TOTAL DE TODAS AS FUNCIONALIDADES PARA CADA SOLUÇÃO

Tarefa	Tempo de desenvolvimento por solução		
	Swift	*Java*	*React Native*
Total	42h 45m	47h 30m	48h

V. CONCLUSÕES E TRABALHO FUTURO

O desenvolvimento de aplicações móveis encontra-se em enorme evolução e atrai um grande interesse de várias organizações a par do que tem acontecido com todo o mercado dos *smartphones*.

No estudo realizado para comparar o desenvolvimento nativo (Swift e Java) relativamente ao desenvolvimento híbrido (React Native) concluiu-se que o desenvolvimento híbrido em React Native apresentou o pior tempo de desenvolvimento, mas, contudo, similar ao Java. O resultado final do desenvolvimento em React Native foi uma aplicação multiplataforma, para Android e iOS, enquanto no Swift e no Java apenas foi uma aplicação para cada um dos sistemas operativos, iOS e Android respetivamente. Daí também a implementação em React Native ter um tempo maior. Neste desenvolvimento com o React Native não foi necessário recorrer à implementação em código nativo, embora tenham surgido configurações individuais para cada sistema operativo, como por exemplo no que concerne às permissões. Na instalação de bibliotecas no React Native também foi necessário realizar configurações diferentes para cada um dos sistemas operativos (não apenas as configurações do *gradle* no Android em Java ou nos *pods* no iOS para Swift).

Para pequenas e médias empresas, onde a existência de equipas individuais especializadas no desenvolvimento nativo para iOS e Android é financeiramente inviável, o React Native pode ser uma solução vantajosa para o desenvolvimento móvel, já que é possível fazer aplicações para ambos os sistemas operativos em quase metade do tempo. O maior tempo de desenvolvimento que poderá ser constatado nos primeiros projetos realizados por programadores com experiência apenas em *mobile* (Swift e Java) é rapidamente ultrapassado após os programadores perceberem a arquitetura associada ao React

ISBN: 978-989-98434-9-3

Native, principalmente se já tiverem experiência em Swift pois é uma linguagem com uma sintaxe mais próxima do JavaScript.

De referir que, através da arquitetura utilizada pelo React Native, as diferenças de desempenho e de utilização são quase impercetíveis entre as aplicações híbridas e as nativas, daí a vasta quantidade de aplicações já criadas em React Native como, por exemplo, Instagram, Skype, UberEats, Tesla [14]–[17].

Para empresas que pretendam mudar o seu desenvolvimento móvel para uma solução de desenvolvimento móvel JavaScript, o React Native apresenta-se como uma das melhores soluções atuais [6]. Esta opção permite criar uma equipa mais homogénea com experiência em desenvolvimento móvel, possivelmente também com experiência em desenvolvimento *frontend* JavaScript, caso este seja realizado em ReactJS. A construção de uma equipa com estas valências tornará o desenvolvimento mais eficiente e com maior qualidade, pois o conhecimento dos aspetos do desenvolvimento móvel aliado ao conhecimento de todas as arquiteturas e boas práticas utilizadas no React Native dará uma curva de aprendizagem melhor à equipa de desenvolvimento. Esta foi uma das técnicas usadas na constituição da equipa de desenvolvimento móvel hibrido realizada pelo Airbnb [18].

Como trabalho futuro, propomos uma avaliação experimental a utilizadores comuns de aplicações móveis, em que são apresentadas quatro versões para uma mesma aplicação, duas para cada plataforma (Android e iOS): uma versão nativa em Java (Android), uma versão nativa em Swift (iOS), uma versão em React Native para Android e outra desenvolvida em React Native para iOS (as versões React Native para Android e iOS tem por base o mesmo projeto). A ideia será a de aferir a satisfação do utilizador tanto a nível de utilização como de resposta à utilização para cada uma das versões, de forma a perceber se são encontradas diferenças entre uma implementação híbrida e uma implementação nativa.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] V. N Inukollu, D. D. Keshamon, T. Kang, and M. Inukollu, "Factors Influncing Quality of Mobile Apps: Role of Mobile App Development Life Cycle," *Int. J. Softw. Eng. Appl.*, vol. 5, no. 5, pp. 15–34, 2014.

[2] I. Malavolta, S. Ruberto, T. Soru, and V. Terragni, "Hybrid Mobile Apps in the Google Play Store: An Exploratory Investigation," *2015 2nd ACM Int. Conf. Mob. Softw. Eng. Syst.*, pp. 56–59, 2015.

[3] Statista, "App Download and Usage Statistics 2017," 2017. [Online]. Available: http://www.businessofapps.com/data/app-statistics/. [Accessed: 14-Jan-2018].

[4] M. Palmieri and A. Cicchetti, "Comparison of Cross-Platform Mobile Development Tools," in *16th International Conference on Intelligence in Next Generation Networks*, 2012, pp. 179–186.

[5] N. C. Zakas, "The Evolution of Web Development for Mobile Devices," *ACM Digit. Libr.*, pp. 1–10, 2017.

[6] Brito Hugo, Gomes Anabela, Santos Álvaro, Jorge Bernardino, "JavaScript em aplicações móveis: React Native vs Ionic vs NativeScript vs desenvolvimento nativo," *CISTI*, 2018.

[7] Facebook, "React Native Documentation," 2019. [Online]. Available: https://facebook.github.io/react-native/docs/getting-started.html. [Accessed: 15-Dec-2019].

[8] Facebook, "React," 2018. [Online]. Available: https://reactjs.org/. [Accessed: 15-May-2018].

[9] I. Suzdalnitski, "React.js: a better introduction to the most powerful UI library ever created," 2018. .

[10] T. Occhino, "React JS conf," 2015. [Online]. Available: https://code.facebook.com/videos/786462671439502/react-js-conf-2015-keynote-introducing-react-native-/. [Accessed: 15-May-2018].

[11] AlexSoft, "JavaScript Ecosystem," 2018. [Online]. Available: https://www.altexsoft.com/blog/engineering/javascript-ecosystem-38-tools-for-front-and-back-end-development/. [Accessed: 10-Feb-2019].

[12] SCRUM, "What is SCRUM?," 2018. [Online]. Available: www.scrum.org/. [Accessed: 10-Jun-2018].

[13] TomislavCar, "Android development is 30% more expensive than iOS. And we have the numbers to prove it!," 2015. [Online]. Available: https://infinum.co/the-capsized-eight/android-development-is-30-percent-more-expensive-than-ios. [Accessed: 31-Dec-2017].

[14] N. Singh, "An comparative analysis of Cordova Mobile Applications V/S Native Mobile Application," *Int. J. Recent Innov. Trends Comput. Commun.*, pp. 3777–3782.

[15] H. Heitk, S. Hanschke, and T. A. Majchrzak, "Evaluating Cross-Platform Development Approaches for Mobile Applications," *Lect. Notes Bus. Inf. Process.*, vol. 140, pp. 120–138, 2013.

[16] S. Xanthopoulos and S. Xinogalos, "A comparative analysis of cross-platform development approaches for mobile applications," *Proc. 6th Balk. Conf. Informatics - BCI '13*, p. 213, 2013.

[17] R. Native, "Who's using React Native?," 2017. [Online]. Available: http://facebook.github.io/react-native/showcase.html. [Accessed: 30-Dec-2017].

[18] G. Peal, "Building a Cross-Platform Mobile Team," 2018. [Online]. Available: https://medium.com/airbnb-engineering/building-a-cross-platform-mobile-team-3e1837b40a88. [Accessed: 10-Feb-2019].

ISBN: 978-989-98434-9-3