Universidade do Minho
Escola de Engenharia
Mestrado em Engenharia Informática

Unidade Curricular de
Tópicos de Desenvolvimento de Software

Ano Letivo de 2022/2023

Análise de Pesquisa
Tecnologias de Desenvolvimento Cross-Platform

Grupo
a83630 - Duarte Serrão
pg50457 - Joana Alves

março 2023

Conteúdo

1 Introdução

Este trabalho de pesquisa foi realizado no âmbito da unidade curricular de Tópicos de Desenvolvimento de *Software* do Mestrado em Engenharia Informática e tem como objetivo aprofundar o conhecimento no tema das tecnologias de desenvolvimento multiplataforma ou *cross-platform*.

No contexto de desenvolvimento de *software*, com o aparecimento de múltiplas plataformas, desenvolver uma aplicação para cada uma tornou-se inviável, uma vez que há demasiadas plataformas ou sistemas operativos, cada um utilizando diferentes *stacks* de desenvolvimento.

Desta forma, o desenvolvimento *cross-platform* tornou-se bastante popular, sendo muito utilizado na atualidade. Esta técnica permite que as aplicações possam ser desenvolvidas e executadas em diferentes plataformas, quer sejam diferentes sistemas operativos, como *Windows* e *Linux*, ou diferentes dispositivos, como dispositivos móveis e computadores, sem a necessidade de reescrever o código para cada plataforma.

Assim, ao longo deste trabalho vai-se começar por apresentar em concreto o que é desenvolvimento *cross-platform*, as suas vantagens e desvantagens, acompanhado de alguns exemplos de modo a auxiliar a perceção do tema. De seguida, o grupo realizará uma pesquisa detalhada sobre uma das *frameworks* utilizadas para este tipo de desenvolvimento, nomeadamente a *framework* Ionic acompanhada por Cordova ou Capacitor, com o objetivo de aprofundar o conhecimento sobre a mesma.

2 Desenvolvimento *Cross-Platform* vs. Nativo

Como referido na introdução deste trabalho, este capítulo é responsável pela definição de desenvolvimento *cross-platform* e sua respetiva exemplificação. No entanto, para auxiliar na perceção, será preciso começar por diferenciá-lo do típico desenvolvimento nativo.

2.1 Desenvolvimento Nativo

O oposto de desenvolvimento *cross-platform* é o desenvolvimento nativo, onde as aplicações são criadas especificamente para uma única plataforma, como *iOS*, *Android*, *Windows* ou *MacOS*. O desenvolvimento nativo envolve a escrita de código separado para cada plataforma, usando as ferramentas e tecnologias específicas da mesma.

Embora o desenvolvimento nativo possa oferecer uma experiência de utilizador mais rica e personalizada, também pode ser mais caro e demorado do que o desenvolvimento *cross-platform*, já que o programador precisa de criar aplicações separadas para cada plataforma.

O desenvolvimento nativo é geralmente escolhido para projetos que exigem alto desempenho, integração com recursos específicos da plataforma, como notificações *push*, mapas ou câmaras, ou para projetos em que a experiência do utilizador é crucial e precisa de ser perfeitamente ajustada para a plataforma em que a aplicação está a ser executada.

Alguns exemplos de aplicações nativas incluem:

- **Whatsapp**
- **Microsoft Word**
- **Spotify**

2.2 Desenvolvimento *Cross-Platform*

O desenvolvimento *cross-platform* é uma abordagem de desenvolvimento de *software* em que o objetivo é criar aplicações ou programas que possam ser executados em múltiplas plataformas, como sistemas operativos diferentes, dispositivos móveis e *browsers*.

Para além disto, permite que os programadores criem aplicações para diversas plataformas usando uma única base de código, em vez de escrever código separado para cada plataforma, tendo como consequência uma redução do tempo e do custo do desenvolvimento, já que o código pode ser reutilizado em diferentes plataformas.

Para auxiliar no desenvolvimento de aplicações *cross-platform*, existem várias *frameworks* desenhadas para o mesmo. Algumas *frameworks* focadas para *desktops* são *Qt*(6) e *Electron*(2) e para desenvolvimento em dispositivos móveis são *React Native*(5) e *Ionic*(3), apesar de estas duas últimas também poderem ser usadas para desenvolvimento *web*. Existem muitas outras *frameworks*, cada uma oferecendo diferentes recursos e abordagens para o desenvolvimento de aplicações, permitindo que os programadores escolham a melhor opção para o seu projeto específico.

Assim, apresentam-se algumas aplicações *cross-platform*:

- **Slack**
- **Adobe XD**
- **Discord**
- **Mozilla Firefox**

2.2.1 Vantagens

De acordo com o que foi dito nas secções anteriores, esta técnica de desenvolvimento possui várias vantagens. Assim, enumera-se algumas de modo a sintetizar a informação:

- **Redução de Tempo e Custo:** É possível criar uma aplicação para várias plataformas com um único código-fonte, o que reduz significativamente o tempo e os custos de desenvolvimento.
- **Alcance:** É possível alcançar um público maior, já que a aplicação pode ser executada em várias plataformas, incluindo dispositivos móveis, *desktops* e *web*.
- **Facilidade de Manutenção:** Como a aplicação é desenvolvida com um único código-fonte, as atualizações e correções podem ser aplicadas facilmente em todas as plataformas, sem a necessidade de reescrever o código para cada plataforma individualmente.
- **Consistência de Design:** É possível manter uma aparência e comportamento consistentes em todas as plataformas, o que ajuda a criar uma experiência de utilizador mais coesa.
- **Acesso a Recursos Nativos:** As aplicações podem ter acesso aos recursos nativos de cada plataforma, como câmara, GPS, entre outros, proporcionando ao utilizador uma experiência mais rica.
- **Flexibilidade:** Os programadores têm a flexibilidade de escolher a linguagem e ferramentas de desenvolvimento mais adequadas para o projeto, o que permite uma maior produtividade e satisfação no trabalho.

2.2.2 Desvantagens

No entanto, esta técnica de desenvolvimento, não possui apenas traços vantajosos, havendo bastantes desvantagens, tais como:

- **Desempenho:** Podem ter um desempenho inferior em comparação com as aplicações nativas, especialmente em tarefas que exigem muitos recursos do sistema, como jogos.
- **Limitações de Recursos:** Em alguns casos, os recursos nativos de uma plataforma podem não estar disponíveis ou serem limitados em aplicações *cross-platform*, o que pode afetar a funcionalidade e a experiência do utilizador.
- **Dependência de Terceiros:** Algumas *frameworks* podem depender de bibliotecas e ferramentas de terceiros, o que pode criar dependências adicionais e aumentar a complexidade do projeto.
- **Curva de Aprendizagem:** Aprender uma nova linguagem ou ferramenta de desenvolvimento pode levar tempo, o que pode afetar o tempo de desenvolvimento e a produtividade dos programadores.
- **Limitações de Personalização:** Algumas plataformas podem ter limitações em relação à personalização de recursos nativos, como ícones ou animações, o que pode afetar a aparência da aplicação e a experiência do utilizador.
- **Adaptação:** Em alguns casos, pode haver uma certa demora na adaptação das novas tecnologias pelas ferramentas *cross-platform*, o que pode levar a limitações em termos de recursos disponíveis para desenvolvimento.

3 Ionic + Cordova/Capacitor

Esta secção irá detalhar a informação de uma *framework* em específico, neste caso *Ionic* em junção com *Cordova* ou *Capacitor*. Estas duas últimas *frameworks* funcionam essencialmente da mesma forma, pelo que se decidiu focar-se em *Cordova*. Escolheu-se *Ionic*, uma vez que é uma das *frameworks* mais conhecidas na área.

3.1 Contextualização

A *framework* **Ionic** é uma tecnologia *open source* que permite o desenvolvimento de aplicações móveis híbridas e *cross-platform*. Esta utiliza tecnologias *web* como *HTML, CSS* e *JavaScript* para criar aplicações que funcionam em várias plataformas, incluindo *iOS*, *Android* e *Windows*.

O aparecimento da *framework* Ionic foi motivado pela necessidade de criar aplicações móveis com uma abordagem mais rápida e económica, aproveitando as capacidades dos programadores que já possuíam conhecimento em tecnologias *web*. Desta forma, a Ionic trouxe uma nova perspetiva para o desenvolvimento de aplicações móveis, permitindo que empresas e programadores independentes criassem soluções inovadoras de forma mais ágil.

A Ionic foi lançada em 2013 e desde então tem sido uma das *frameworks* mais populares para o desenvolvimento de aplicações móveis. Esta conta com uma ampla gama de recursos, como bibliotecas de componentes pré-fabricados, integração com outras tecnologias, como o *Angular, React* ou *Vue*, além de oferecer suporte para desenvolvimento em equipa e integração com serviços *cloud*.

A *framework* **Cordova** é, também, uma tecnologia *open source* para desenvolvimento de aplicações móveis *cross-platform*. Assim, utiliza *plugins* para permitir que as aplicações acedam a recursos nativos do dispositivo para que as aplicações pareçam e funcionem como aplicações nativas. Para além disto, também fornece um conjunto de APIs para aceder a recursos comuns, como o armazenamento de dados. Desta forma, se uma aplicação desenvolvida em Ionic necessitar de funcionalidades nativas, esta irá utilizar Cordova.

Com a crescente procura por aplicações móveis de alta qualidade e com funcionamento em diversas plataformas, a *framework* Ionic tem-se tornado cada vez mais popular entre empresas e programadores. Portanto, estudar e entender esta tecnologia pode ser muito útil para quem procura destacar-se no mercado de desenvolvimento de aplicações móveis.

3.2 Ionic vs Cordova/Capacitor

Tanto Ionic como Cordova/Capacitor são *frameworks* utilizadas para a construção de uma aplicação, apesar de que ambas têm papeis bastante diferentes. Ionic foca-se na interação com o utilizador, enquanto que Cordova ou Capacitor já se focam em fornecer funcionalidades nativas (como a câmara), para que a aplicação a possa utilizar.

Na lista abaixo encontram-se algumas das funcionalidades que cada *framework* oferece.

Ionic	**Cordova/Capacitor**
• Navegação	• Câmara
• Formulários	• Geolocalização
• Listas	• Giroscópio
• Animações	• Sistema de ficheiros
• Botões	• Notificações *push*
• etc.	• etc.

3.3 Conhecimento Prévio

Embora não seja necessário possuir conhecimento prévio sobre outras *frameworks* ou linguagens de programação para começar a utilizar Ionic, é importante que o programador tenha conhecimento de tecnologias *web*, como *HTML, CSS* e *JavaScript*. Além disso, possuir uma compreensão básica de conceitos de programação, como os seus diversos paradigmas, estruturas de dados, funções e variáveis, pode ajudar no processo de aprendizagem.

A *framework* Ionic é baseada na *framework Angular*, portanto, ter conhecimento básico em *Angular* pode, também, ajudar o programador a entender melhor a estrutura e funcionamento de Ionic.

No entanto, a documentação é bastante abrangente e conta com diversos tutoriais e exemplos práticos para facilitar o processo de aprendizagem para aqueles que estão a começar do zero. Além disso, a comunidade é bastante ativa e pode ser uma fonte valiosa de suporte e ajuda em caso de dúvidas ou problemas.

Em resumo, embora não seja necessário ter um conhecimento prévio específico, ter uma base em tecnologias *web* e uma compreensão básica de programação pode facilitar o processo de aprendizagem da *framework*.

3.4 Arquitetura

A arquitetura da *framework* Ionic é baseada numa variedade de componentes e serviços que trabalham em conjunto para criar a interface do utilizador e a lógica das aplicações móveis. A estrutura é construída sobre tecnologias *web*, como *HTML, CSS* e *JavaScript*, e usa a *framework* Angular para *data binding* e injeção de dependências.

Como se pode observar pela figura 1, esta *framework* é composta por três grandes componentes:

- **Tecnologias *Web*:** Este componente, tal como o nome indica, inclui todas as tecnologias *web* disponibilizadas pela aplicação, como por exemplo: *HTML, CSS* e *JavaScript*.
- **Aplicação Ionic:** Este componente engloba toda a estrutura da aplicação, como *frameworks* auxiliares (*React*, *Vue* ou *Angular*) e tecnologias *web* (ponto acima). Para além disto, fornece um conjunto de *plugins* nativos que permitem aos programadores aceder a recursos do dispositivo, como câmara, armazenamento, entre outros.
- **Capacitor/Cordova:** Este componente fornece uma ponte entre a aplicação *web* e o código nativo, permitindo que os programadores acedam a APIs nativas e criem aplicações híbridos.

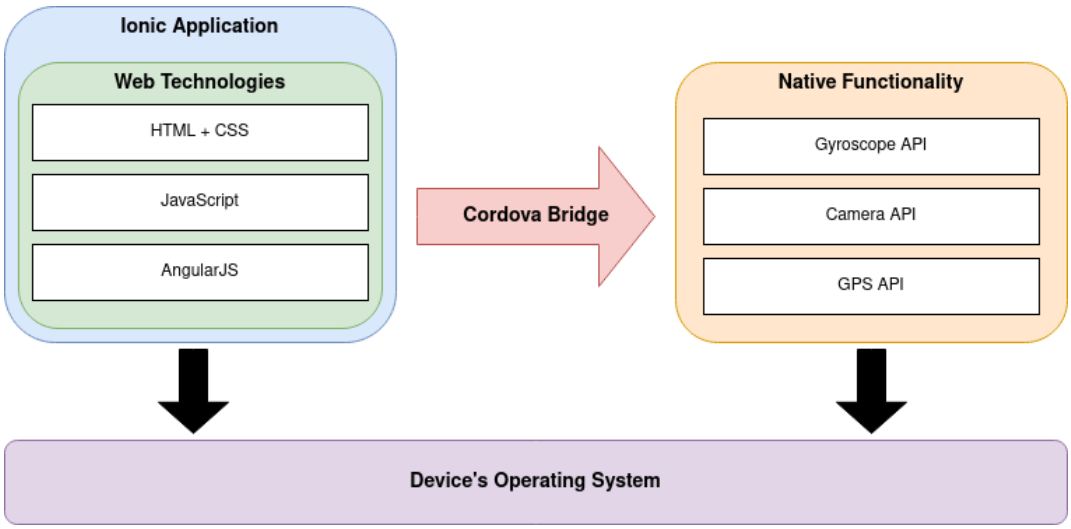

Figura 1: Arquitetura Ionic + Cordova.

No geral, a arquitetura desta *framework* foi projetada para ser modular e extensível, permitindo que os programadores construam e personalizem facilmente aplicações móveis usando tecnologias *web*.

3.5 Vantagens e Limitações

De acordo com o que foi referido nas secções anteriores, é possível sintetizar as características desta *framework* e até distingui-las em vantagens e limitações da mesma. Assim, enumera-se as suas **vantagens** comparativamente a outras *frameworks cross-platform*:

- **Desenvolvimento Rápido e Económico:** Utilizando tecnologias *web* (*HTML, CSS, JavaScript*) é possível desenvolver aplicações móveis de forma mais rápida e económica, aproveitando as capacidades dos programadores que já possuem conhecimento nessa área.
- **Curva de Aprendizagem:** Tendo em conta que qualquer *framework* nova para um programador demore o seu tempo a ser aprendida e conhecida, esta usa já ferramentas que grande parte da comunidade já conheça previamente. Isto leva a que uma grande percentagem de programadores consigam ter uma curva de aprendizagem reduzida.
- **Biblioteca de Componentes:** A *framework* Ionic oferece uma ampla gama de componentes pré-fabricados, como botões, menus, listas, entre outros, que permitem construir rapidamente a interface do utilizador da aplicação.
- **Integração:** Esta *framework* pode ser integrada com outras tecnologias, como o *Angular*, *React*, *Vue*, entre outros, para aprimorar ainda mais o desenvolvimento de aplicações.
- ***Plugins*:** Ionic utiliza *plugins* para aceder a recursos nativos de dispositivos móveis, como câmara, GPS e armazenamento local. Isso permite que os programadores criem aplicações móveis que aproveitem ao máximo os recursos do dispositivo.
- **Comunidade Ativa:** Possui uma comunidade ativa de programadores e utilizadores, que fornecem suporte e recursos para aqueles que estão a iniciar-se na *framework*.

No entanto, como qualquer outra tecnologia, também possui as suas **limitações** ou desvantagens. Desta forma, enumera-se os traços avaliados como negativos desta *framework*:

- **Desempenho:** Embora a Ionic seja eficiente em termos de desenvolvimento rápido e económico, algumas aplicações desenvolvidos com a *framework* podem apresentar problemas de desempenho, especialmente em dispositivos mais antigos.
- **Personalização:** Embora a biblioteca de componentes da Ionic seja ampla, algumas personalizações mais específicas podem ser difíceis de serem implementadas.
- **Dependência de *Plugins*:** Para aceder a recursos nativos de dispositivos móveis, a Ionic depende de *plugins* que podem ser limitados em termos de funcionalidades.

Em resumo, a Ionic oferece diversas vantagens para o desenvolvimento de aplicativos móveis, como rapidez, economia e suporte a várias plataformas, mas também apresenta algumas limitações, como desempenho, personalização e dependência de plugins.

3.6 Aplicabilidade

O *Ionic* é aplicável para principalmente a criação de aplicações móveis, tanto para *iOS*, como *Android*. Sendo uma *framework* de desenvolvimento *cross-platform*, é possível criar uma aplicação com o mesmo código para as duas plataformas mencionadas.

Mais especificamente, é possível criar aplicações móveis híbridas que combinam tecnologias de aplicações nativas e da *Web*, resultando numa experiência rápida e responsiva para o utilizador.

Apesar de ser focada em aplicações móveis, também é possível desenvolver na *Web*.

Também é possível utilizar o *Ionic* para prototipagem, já que permite um desenvolvimento rápido de uma aplicação funcional e teste da mesma em dispositivos reais.

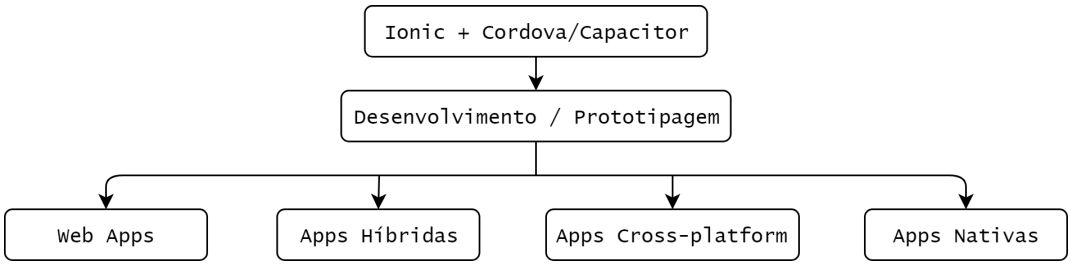

Figura 2: Esquema das aplicabilidades de *Ionic + Cordova/Capacitor*

É de notar que para se criar uma aplicação que utilize funcionalidades nativas, será preciso utilizar Cordova ou Capacitor.

3.7 Componentes Básicos UI

De forma a se familiarizar com a ferramenta, encontram-se abaixo alguns componentes básicos(1):

- **Content:** Sendo este um componente fulcral, é o que permite conter outros componentes para interação e navegação pela aplicação.
- **Navigation:** A navegação é como os utilizadores se movem entre diferentes páginas na aplicação.
- **Item:** Os itens são um *container* de interface de utilizador genérico que pode ser usado como parte de uma lista.
- **Input:** Este componente fornece uma forma para os utilizadores inserirem dados na aplicação.
- **Button:** Os botões permitem que seus utilizadores realizem ações. Eles são uma maneira essencial de interagir e navegar pela aplicação.
- **Toast:** Este componente é usado para mostrar uma notificação por cima do conteúdo da aplicação. Pode ser temporário ou descartável

A partir da documentação de Ionic, existem bastantes mais componentes. A junção de todos eles torna possível uma criação bastante personalizada de uma aplicação.

3.8 Aplicações que utilizam Ionic

Com base na plataforma mdevelopers(4), e uma pesquisa mais aprofundada das aplicações escolhidas, encontram-se abaixo três aplicações que utilizaram a *framework* Ionic para desenvolver uma aplicação *cross-platform*:

- **MarketWatch:** Esta aplicação ajuda a descobrir notícias, análises e dados do mercado de ações mais recentes. Isso significa que os clientes podem utilizar a aplicação para obter aconselhamento financeiro pessoal e notícias da empresa, bem como cotações do mercado de ações.
- **Sanvello:** Esta aplicação ajuda os seus utilizadores a gerir o stress, a ansiedade e a depressão, recebendo atendimento profissional de terapeutas e psiquiatras. A aplicação foi criada tendo como base a Terapia Cognitivo Comportamental.
- **Sporkit:** Esta aplicação de *fitness* oferece um plano personalizado para cada utilizador, com base nas suas necessidades, tempo e nível físico.

Existem muitas outras, o que indica que uma grande quantidade de programadores que preferem desenvolvimento *cross-platform* em vez de nativo.

4 Conclusão

Concluindo, ferramentas de *cross-platform* como o *Ionic* tornaram-se cada vez mais populares devido à sua capacidade de criar aplicações móveis que podem ser executadas em várias plataformas usando uma única base de código. O *Ionic* oferece uma gama de benefícios em relação a outros *frameworks* de *cross-platform*, incluindo a sua sintaxe fácil de aprender, arquitetura modular, desempenho semelhante ao nativo e interface do utilizador personalizável.

Ao usar *Ionic*, programadores podem economizar tempo e recursos enquanto fornecem aplicações móveis de alta qualidade que oferecem uma experiência de utilizador perfeita em diferentes dispositivos. O *Ionic* pode ser aplicado a uma variedade de casos de uso em vários setores, incluindo saúde, comércio eletrónico, educação, redes sociais e finanças.

No entanto, é importante observar que ferramentas de *cross-platform* como *Ionic* nem sempre são a melhor solução. Dependendo dos requisitos do projeto, o desenvolvimento nativo de aplicações ou até outros *frameworks* de *cross-platform* podem ser mais apropriados. Portanto, é importante que os programadores avaliem cuidadosamente as suas opções e escolham a ferramenta que melhor se adapta às suas necessidades e objetivos específicos.

Referências

[1] Componentes de Ionic, https://ionicframework.com/docs/components

[2] Eletron *homepage*, https://www.electronjs.org.

[3] Ionic *homepage*, https://ionicframework.com.

[4] Aplicações desenvolvidas com Ionic, https://mdevelopers.com/blog/14-best-ionic-apps-of-2022

[5] React *homepage*, https://reactnative.dev.

[6] Qt *framework*, https://www.qt.io/product/framework.