# React Native Multi-Platform Mobile Apps Development

André Fial de Almeida

(Licenciado em Engenharia Informática e Multimédia)

Relatório de Estágio para obtenção do grau de Mestre em

Engenharia Informática e Multimédia

Orientadores:

Doutor João Beleza Sousa

Mestre Nuno Ribeiro

Júri:

Presidente: Doutor António Teófilo

Vogais: Doutor Carlos Gonçalves

Doutor João Beleza Sousa

Outubro de 2024

# React Native Multi-Platform Mobile Apps Development

André Fial de Almeida

(Licenciado em Engenharia Informática e Multimédia)

Relatório de Estágio para obtenção do grau de Mestre em
Engenharia Informática e Multimédia

Orientadores:

Doutor João Beleza Sousa, ISEL

Mestre Nuno Ribeiro, *Bliss Applications*

Júri:

Presidente: Doutor António Teófilo, ISEL

Vogais: Doutor Carlos Gonçalves, ISEL

Doutor João Beleza Sousa, ISEL

Outubro de 2024

# Declaração de integridade

Declaro que esta(e) dissertação / trabalho de projeto / relatório de estágio é o resultado da minha investigação pessoal e independente. O seu conteúdo é original e todas as fontes listadas nas referências bibliográficas foram consultadas e estão devidamente mencionadas no texto. Mais declaro que todas as referências científicas e técnicas relevantes para o desenvolvimento do trabalho estão devidamente citadas e constam das referências bibliográficas.

O autor

Lisboa, 24 de Setembro de 2024

# Agradecimentos

Este relatório de estágio representa a conclusão de um período de aprendizagem e crescimento pessoal e profissional, que não teria sido possível sem o apoio e colaboração de várias pessoas e instituições, às quais gostaria de expressar a minha gratidão.

Em primeiro lugar, gostaria de agradecer à *Bliss Applications* pela oportunidade de realizar este estágio e pela confiança depositada em mim. A todos os membros da equipa e ao orientador Nuno Ribeiro, o meu sincero agradecimento pela partilha de conhecimentos, pela orientação constante e pelo ambiente de trabalho acolhedor e amigável que tornaram esta experiência tão importante e enriquecedora. Um agradecimento especial à equipa de desenvolvedores, composta por Sérgio Fontora e Rui Rigueira, pelo acompanhamento ao longo do projeto e pelo *feedback* construtivo. Agradeço também e desejo tudo do melhor ao Nuno Pereira que infelizmente não esteve presente durante todo o projeto, mas que contribuiu para a minha introdução neste. A vossa orientação foi essencial para o sucesso do estágio e para o meu desenvolvimento como programador.

Gostaria também de expressar a minha gratidão à equipa de *mobile* da *Worten*, desde líderes do projeto aos *designers*, com um agradecimento especial à *product owner* Catarina Veloso, pela oportunidade de contribuir num projeto desta dimensão. Esta experiência proporcionou-me um contexto real de desenvolvimento de *software* num ambiente profissional, mas também dinâmico e cooperativo, o que foi fundamental para o meu crescimento.

Agradeço ainda à minha instituição de ensino, com destaque ao orientador João Beleza, cujo suporte e sugestões transmitidas ao longo do projeto foram essenciais para a realização deste relatório.

A todos, o meu sincero obrigado.

# Resumo

As tecnologias móveis têm desempenhado um papel cada vez mais importante no dia a dia das nossas vidas, facilitando a comunicação, o entretenimento, o acesso e a partilha de informação e o consumo. Tudo isto na palma das nossas mãos, e a qualquer momento. Por isso, o desenvolvimento de aplicações móveis tornou-se numa área crucial, exigindo conhecimentos específicos de forma a criar experiências eficientes e atrativas para o utilizador.

Neste contexto cada vez mais empresas têm-se dedicado ao desenvolvimento destas aplicações, entre as quais a ***Bliss Applications***, onde o autor teve a oportunidade de estagiar.

O desenvolvimento desta área depende também da escolha das tecnologias utilizadas, existindo várias tecnologias dedicadas ao desenvolvimento de aplicações móveis, como *Flutter*, *Xamarin*/*.NET MAUI*, *Ionic* ou ***React Native***. No caso deste estágio a tecnologia utilizada foi ***React Native***. Esta tecnologia tem sido cada vez mais utilizada devido à sua eficiência, versatilidade e suporte à criação de aplicações para múltiplas plataformas (*multi-platform application*), como *iOS* e *Android*.

A aplicação móvel em questão foi desenvolvida para a empresa ***Worten***, sendo concebida visando proporcionar uma experiência intuitiva aos clientes, simplificando o processo de compra e interação. As tarefas de desenvolvimento incluíram manutenção, a otimização de código e o desenvolvimento de novas funcionalidades.

Este relatório oferece uma visão abrangente das atividades realizadas durante o estágio, destacando as contribuições para o projeto, os *insights* adquiridos e reflexões sobre a experiência adquirida. Espera-se que este relatório possa servir como uma fonte de conhecimento e aprendizagem para qualquer interessado no desenvolvimento de aplicações móveis.

**Palavras-chave:** Tecnologias móveis, Desenvolvimento de aplicações móveis, Bliss Applications, React Native, Multi-platform application, Worten.

# Abstract

Mobile technologies are playing an increasingly important role in our daily lives, facilitating communication, entertainment, access to and sharing of information and consumption. All this in the palm of our hands at any time. That's why the development of mobile applications has become a crucial area, requiring specific knowledge in order to create efficient and attractive user experiences.

In this context, more and more companies have dedicated themselves to developing these applications, including ***Bliss Applications***, where the author had the opportunity to do an internship.

The development of this area also depends on the choice of technologies used. There are several technologies dedicated to the development of mobile applications, such as *Flutter*, *Xamarin*/*.NET MAUI*, *Ionic* or ***React Native***. In the case of this internship, the technology used was ***React Native***. This technology has been increasingly used due to its efficiency, versatility and support for creating applications for multiple platforms (*multi-platform application*), such as *iOS* and *Android*.

The mobile application in question was developed for the company ***Worten***, and was designed with the aim of providing an intuitive experience for customers, simplifying the purchasing and interaction process. Development tasks included maintenance, code optimization and the development of new features.

This report provides a comprehensive overview of the activities carried out during the internship, highlighting the contributions to the project, the *insights* acquired and reflections on the experience gained. It is hoped that this report can serve as source of knowledge and learning for anyone interested in mobile application development.

**Keywords:** Mobile technologies, Mobile application development, Bliss Applications, React Native, Multi-platform application, Worten.

# Acrónimos

***API*** *Application Programming Interface*. 16, 19

***AR*** *Augmented Reality*. 83

***CLI*** *Command Line Interface*. 17, 26

***CSS*** *Cascading Style Sheets*. 16, 19, 30

***DS*** *Design System*. 19, 38, 39, 56, 57, 60, 79

***FNAC*** *Fédération Nationale d’Achats des Cadres*. vii, ix, 8–10, 82

***HTML*** *Hypertext Markup Language*. 16, 19

***HTTP*** *Hypertext Transfer Protocol*. 19

***IDE*** *Integrated Development Environment*. vii, 19, 25, 60

***JSON*** *JavaScript Object Notation*. 19, 55, 56, 78

***JSX*** *JavaScript XML*. 39, 40, 51

***QA*** *Quality Assurance*. 20, 23, 25

***SDLC*** *Software Development Life Cycle*. 22

***SVG*** *Scalable Vector Graphic*. 58, 65

***UI*** *User Interface*. 17, 23, 60, 62, 67, 71

***UX*** *User Experience*. 23, 60, 62, 73

***XML*** *eXtensible Markup Language*. 39, 58

***iOS*** *iPhone Operating System*. iv, v, 14–16, 20, 23, 26

***npm*** *Node Package Manager*. 25, 26

***npx*** *Node Package Execute*. 26

# Índice de Conteúdos


**1 Introdução** **1**

1.1 Objetivos . . . . . 4

1.2 Planeamento . . . . . 5

1.3 Organização do Projeto . . . . . 6

**2 Trabalho Relacionado** **8**

2.1 Aplicações Semelhantes . . . . . 8

2.1.1 *FNAC* . . . . . 8

2.1.2 *MediaMarkt* . . . . . 10

2.1.3 Amazon . . . . . 11

2.1.4 eBay . . . . . 13

2.2 Tecnologias Multi-Plataforma Semelhantes . . . . . 14

2.2.1 Flutter . . . . . 15

2.2.2 *Xamarin* . . . . . 16

2.2.3 Ionic . . . . . 16

**3 Tecnologias Utilizadas** **18**

3.1 *Frameworks* . . . . . 18

3.2 Bibliotecas . . . . . 18

3.3 *IDE* (*Integrated Development Environment*) . . . . . 19

3.4 Ferramentas de Teste . . . . . 20

3.5 Ferramentas de Gestão de Projeto e Controlo de Versões . . . . . 21

**4 Metodologia** **22**

**5 Trabalho Realizado** **25**

5.1 Curso React Native . . . . . 25

5.1.1 *Setup* . . . . . 25

5.1.2 Noções Básicas . . . . . . 27
5.1.3 Navegação . . . . . . 34
5.1.4 Flatlist . . . . . . 37
5.2 *Design System* . . . . . . 38
5.2.1 Criação e Desenvolvimento de Novos Componentes . . . . . . 39
5.2.2 Manutenção de Componentes . . . . . . 54
5.2.3 Iconografia . . . . . . 56
5.2.4 Ecrãs de Componentes . . . . . . 59
5.3 Aplicação de Produção . . . . . . 60
5.3.1 *Setup* e Introdução à Aplicação . . . . . . 60
5.3.2 Jogo . . . . . . 62
5.3.3 Área de Cliente . . . . . . 72
5.3.4 Traduções . . . . . . 77
5.3.5 Migração de Componentes . . . . . . 79

**6 Conclusões** **82**

# Índice de Figuras


1.1 *Bliss Applications* . . . . . . . . . . 2
1.2 *Worten* . . . . . . . . . . 3
1.3 *Software Jira* . . . . . . . . . . 5
1.4 *Figma* . . . . . . . . . . 6
1.5 Plataforma *GitLab* . . . . . . . . . . 6

2.1 *FNAC* . . . . . . . . . . 9
2.2 *Blog* de sugestões/*reviews* da *FNAC* . . . . . . . . . . 9
2.3 *MediaMarkt* . . . . . . . . . . 10
2.4 Páginas de produto . . . . . . . . . . 11
2.5 *Amazon* . . . . . . . . . . 12
2.6 Serviços *Amazon* . . . . . . . . . . 12
2.7 *eBay* . . . . . . . . . . 13
2.8 Leilão no *eBay* . . . . . . . . . . 14
2.9 *React Native* . . . . . . . . . . 15
2.10 *Flutter* . . . . . . . . . . 15
2.11 *Xamarin* e *.NET MAUI* . . . . . . . . . . 16
2.12 *Ionic* . . . . . . . . . . 17

3.1 *WebStorm* . . . . . . . . . . 19
3.2 *Firebase App Distribution* . . . . . . . . . . 20
3.3 Aplicações de Teste no *Firebase App Distribution* . . . . . . . . . . 20
3.4 *TestFlight* . . . . . . . . . . 20

4.1 Ciclo de Vida do Desenvolvimento de Software . . . . . . . . . . 22

5.1 *Expo Go* . . . . . . . . . . 26
5.2 Propriedade *flexDirection* . . . . . . . . . . 32
5.3 Propriedade *justifyContent* . . . . . . . . . . 33

5.4 Propriedade *alignItems* . . . 33
5.5 Diferentes tipos de navegação . . . 34
5.6 Navegação em pilha (*Stack Navigation*) . . . 35
5.7 Navegação em abas (*Tab Navigation*) . . . 36
5.8 Navegação em gaveta (*Drawer Navigation*) . . . 37
5.9 Componente *Link Chip* . . . 41
5.10 Ecrã exemplo do componente *Link Chip* . . . 42
5.11 Ecrã exemplo do componente *Link Chip Container* . . . 44
5.12 Ecrã exemplo do componente *Radio Button Group* . . . 47
5.13 Componente *Info Blocks* . . . 49
5.14 *Badge Flags* no componente *Product Card* . . . 50
5.15 Componente *Badge Flag* . . . 51
5.16 Ecrã exemplo do componente *Action List Item* . . . 53
5.17 Componente *Action List Item* . . . 53
5.18 Botão com *lottie* de carregamento (*loading state*) . . . 56
5.19 Biblioteca de *design* . . . 57
5.20 *Icon Brand 033* . . . 58
5.21 *Icons* dos idiomas suportados . . . 59
5.22 Ecrã exemplo antigo do componente *Button* . . . 59
5.23 Ecrã exemplo atual do componente *Button* . . . 60
5.24 Página de pesquisa antiga . . . 61
5.25 Página de pesquisa atual . . . 61
5.26 Menu do jogo . . . 62
5.27 Variante *play* do botão do jogo . . . 63
5.28 Página da classificação do jogo . . . 64
5.29 *Drawers* informativas do jogo . . . 64
5.30 Ecrã de jogo . . . 65
5.31 Elementos do jogos . . . 66
5.32 Ecrã de fim de jogo (*game over*) . . . 66
5.33 Ecrã de pausa do jogo . . . 67
5.34 Cartões de classificação . . . 68
5.35 *Design* dos níveis . . . 70
5.36 Exemplo retirado de um tapete . . . 71
5.37 Menu principal antes da campanha *Ayvens* . . . 72

5.38 Área de cliente antes da remodelação . . . . . . . . . . . . . . . . . . . . . . . . 73
5.39 Menu da conta . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 74
5.40 *Worten Resolve* na área cliente . . . . . . . . . . . . . . . . . . . . . . . . . . . . 75
5.41 Cartões de saldo e cupões (*Worten* e Continente) . . . . . . . . . . . . . . . . . . 75
5.42 Compras e favoritos do utilizador . . . . . . . . . . . . . . . . . . . . . . . . . . . 76
5.43 *Item* de mudança de idioma . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 76
5.44 *Drawer* de mudança de idioma . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 77
5.45 Cabeçalho do componente *Express Filters* . . . . . . . . . . . . . . . . . . . . . . 79
5.46 *Drawer* principal do componente *Express Filters* . . . . . . . . . . . . . . . . . . 80
5.47 *Drawers* de entregas/levantamentos do componente *Express Filters* . . . . . . . 80
5.48 Lista de cartões de produtos sugeridos . . . . . . . . . . . . . . . . . . . . . . . . 81
5.49 *Drawers* de *onboarding* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 81

# Índice de Listagens


5.1 View . . . . . 27
5.2 Text . . . . . 27
5.3 Image . . . . . 27
5.4 ScrollView . . . . . 28
5.5 Button . . . . . 28
5.6 TouchableOpacity . . . . . 28
5.7 TextInput . . . . . 28
5.8 Props no componente pai . . . . . 29
5.9 Props no componente filho . . . . . 29
5.10 Output com Props . . . . . 29
5.11 States . . . . . 30
5.12 *StyleSheet* . . . . . 31
5.13 Estilos *inline* . . . . . 31
5.14 Estilos com *StyleSheet* . . . . . 31
5.15 *Stack Navigator* . . . . . 35
5.16 *Bottom Tab Navigator* . . . . . 36
5.17 *Drawer Navigator* . . . . . 37
5.18 *Flatlist* . . . . . 38
5.19 *Link Chip* . . . . . 40
5.20 Utilização do *Link Chip* . . . . . 41
5.21 *Link Chip Container* . . . . . 42
5.22 Utilização do *Link Chip Container* . . . . . 43
5.23 *Radio Button Group* . . . . . 45
5.24 Utilziação do *Radio Button Group* . . . . . 46
5.25 Info Blocks . . . . . 48
5.26 Utilização do *Info Blocks* . . . . . 48
5.27 Badge Flags . . . . . 50

5.28 Utilização do *Badge Flags* . . . . . 50
5.29 Action List Item . . . . . 52
5.30 Utilização do *Action List Item* . . . . . 53
5.31 Antes da refatoração dos estilos . . . . . 54
5.32 Utilização dos estilos antes da refatoração . . . . . 54
5.33 Depois da refatoração de estilos . . . . . 55
5.34 Utilização dos estilos depois da refatoração . . . . . 55
5.35 *Loading lottie* . . . . . 56
5.36 Ícone *Bra033* . . . . . 58
5.37 Animações . . . . . 68
5.38 Função de animação . . . . . 68
5.39 Utilização da função de animação . . . . . 69
5.40 Utilização da animação num componente . . . . . 69
5.41 Implementação de um tapete . . . . . 71
5.42 Implementação sem tradução . . . . . 77
5.43 Implementação com tradução . . . . . 78
5.44 JSON com traduções - PT . . . . . 78
5.45 JSON com traduções - EN . . . . . 79

# Capítulo 1

# Introdução

Nos últimos anos, as aplicações móveis têm desempenhado um papel fundamental na forma como interagimos com o mundo digital. Desde o surgimento dos primeiros *smartphones* até à expansão dos dispositivos móveis por todas as áreas do quotidiano, as aplicações móveis têm-se tornado numa grande parte do nosso dia a dia.

As aplicações móveis, abreviadas como *apps* (*applications*), são programas desenvolvidos especificamente para serem executados em dispositivos móveis, como *smartphones* e *tablets*, abrangendo um espetro variado de funcionalidades, desde redes sociais e jogos até aplicações de trabalho e comércio *online*. Estas facilitam a interação do utilizador e proporcionam uma experiência personalizada e mais conveniente quando comparada com a interação de páginas *web* comuns. Com estas interfaces mais intuitivas e acesso rápido a informação e serviços, as aplicações móveis tornaram-se essenciais para diversos contextos da vida moderna.

O mercado de aplicações móveis tem crescido exponencialmente, tal como observado em [24 (Statista, 2024)], ajudado pelo desenvolvimento de soluções inovadoras e pela constante evolução das tecnologias móveis, criando investimento neste mercado por parte das empresas de forma a alcançar os clientes de maneira mais intuitiva.

No contexto do comércio *online*, as aplicações móveis desempenham um papel crucial, oferecendo uma plataforma interativa e acessível para compras e interação com os clientes. Com a adoção de dispositivos móveis em todo o mundo, as empresas focaram os seus esforços e a sua atenção cada vez mais no desenvolvimento e melhoramento das plataformas móveis como parte fulcral das estratégias de *e-commerce*.

À medida que as aplicações móveis se tornam uma parte indispensável do nosso quotidiano, a necessidade de conseguir suportar todas as plataformas e dispositivos tornou-se cada vez mais importante. O desenvolvimento de aplicações multiplataforma surge como uma estratégia para lidar com os desafios da diversidade do mercado e aumentar as possibilidades

de alcance das aplicações. Esta estratégia surge como uma resposta eficaz e económica para as complexidades do mercado atual, alcançando vários dispositivos, sistemas operativos e resoluções de ecrã.

Em vez da aplicação ser implementada separadamente para cada plataforma, são utilizadas *frameworks* que permitem reutilizar o que foi implementado em diferentes sistemas operativos, reduzindo o tempo e recursos necessários para desenvolver e manter as aplicações para cada plataforma individualmente.

Esta abordagem amplia também o alcance dentro do mercado das aplicações, permitindo que atinjam uma maior audiência ao serem disponibilizadas para múltiplas plataformas. Além disso, ao consolidar o desenvolvimento numa única implementação, o processo de manutenção (atualização e correção de *bugs*, novas funcionalidades, etc.) pode ser significativamente simplificado, garantindo uma interação consistente para os utilizadores. Com esta crescente utilização de aplicações móveis, o desenvolvimento multiplataforma tornou-se numa estratégia essencial para aplicações que procuram maximizar o seu impacto no mercado e alcançar sucesso sustentável a longo prazo.

É possível observar como esta abordagem tem sido adotada por várias empresas de forma a criar experiências eficientes e generalizadas. Um exemplo notável é a ***Bliss Applications***, uma empresa destacada no cenário tecnológico pela sua abordagem centrada no utilizador e o seu compromisso com soluções tecnológicas eficazes e dinâmicas.

Figura 1.1: *Bliss Applications*

A *Bliss Applications*, tal como pode ser consultado em [2 (Applications, 2024)], é uma empresa inovadora fundada em Lisboa no final de 2009, tendo-se tornado líder em Portugal no desenvolvimento de software centrado no *design* (com foco no desenvolvimento de produtos digitais) em apenas dois anos com uma equipa de mais de 180 pessoas. Esta empresa possui atualmente escritórios localizados em Lisboa, Porto e Boston, entregando produtos de qualidade a todo o mundo. Faz parte de uma empresa maior, o ***WYgroup***, o maior grupo nacional de *marketing* digital em Portugal. O *WYgroup* foi criado em 2001 e é atualmente uma *holding* de 7 empresas independentes e emprega mais de 400 pessoas. Para um melhor entendimento sobre o *WYgroup*, consultar [25 (WYgroup, 2024)].

Com mais de 100 produtos digitais lançados na última década, a *Bliss Applications* possui

uma equipa experiente de *designers* e engenheiros que trabalham em conjunto com cada cliente para enriquecer a experiência de cada utilizador que interage com as experiências digitais desenvolvidas, traduzindo em valor acrescentado para o cliente. O seu processo de *design* de produto, inspirado em metodologias como *Google Design Sprint* e *Design Thinking*, é facilitado por profissionais e adaptado às necessidades de cada cliente, sempre em estreita colaboração com a equipa de engenharia, garantindo que é escolhida a melhor abordagem técnica para uma qualidade superior.

Ao analisar as práticas de desenvolvimento adotadas pela *Bliss Applications*, é de salientar as soluções implementadas em colaboração com clientes líderes no seu mercado. Dentro de vários projetos apresentados em [3 (Applications, 2024)], um exemplo desta parceria é a ***Worten***, marca de renome no setor de retalho de tecnologia.

Figura 1.2: *Worten*

A *Worten* é reconhecida pela sua vasta gama de produtos de qualidade e pelo atendimento eficiente ao cliente. Com uma presença consolidada em vários mercados, a *Worten* continua a evoluir e a adaptar-se às mudanças no cenário do comércio eletrónico, procurando constantemente novas formas de oferecer conveniência e valor aos seus clientes.

Como uma das marcas mais notáveis do portfólio da *Sonae*, um dos maiores grupos de retalho e serviços de Portugal, a *Worten* tem uma forte presença tanto *online* como física, oferecendo aos clientes múltiplos pontos de contacto para adquirir os seus produtos. Ao combinar uma rede de lojas físicas bem estabelecida com uma plataforma de *e-commerce* robusta, a *Worten* oferece aos clientes a flexibilidade de escolher como e quando desejam interagir com a marca, garantindo uma experiência de compra mais conveniente.

Com a sua visão inovadora, a *Worten* está constantemente à procura das tecnologias mais recentes de forma a melhorar a experiência do utilizador. Ao adotar uma abordagem centrada neste e colaborar com parceiros como a *Bliss Applications*, a *Worten* tem explorado o potencial das aplicações móveis implementadas em *React Native* de modo a atingir uma aplicação multiplataforma para oferecer uma experiência de compras mais intuitiva. Esta sinergia entre tecnologia de ponta e um compromisso com a satisfação do cliente posiciona a *Worten* como líder no setor de retalho, elevando os padrões de excelência no mercado.

## 1.1 Objetivos

No início do estágio foram delineados cinco objetivos importantes a cumprir durante a sua duração. Estes objetivos representam contextos de aprendizagem pessoal e profissional mas também de conclusão de etapas académicas.

1. **Aprendizagem de React e Expansão de Conhecimentos em Programação:** Um dos principais objetivos é expandir o conhecimento em linguagens de programação, especificamente em *React Native*, utilizando esta *framework* no desenvolvimento de aplicações *web* e móveis.

2. **Colaborar com Designers e Melhorar Competências em Figma:** Reconhecendo a importância da colaboração em paralelo entre programadores e *designers*, é considerado como segundo objetivo poder trabalhar com uma equipa de *design*, permitindo melhorar competências em *Figma* (descrito na secção seguinte - **1.2 Planeamento**) e fortalecer a integração entre o *design* e o desenvolvimento.

3. **Desenvolver Habilidades de Trabalho em Equipa e Comunicação:** Dada a relevância da comunicação no desenvolvimento de projetos de *software*, é pretendido aprimorar as habilidades de trabalho em equipa que têm sido desenvolvidas no percurso académico, contribuindo ativamente para discussões construtivas, fornecendo *feedback* útil e comunicando de forma eficiente com colegas de diferentes áreas.

4. **Aplicar Conhecimentos Académicos num Contexto Profissional:** Durante o percurso académico é adquirido um conjunto de várias competências. Um dos objetivos seria poder aplicar esses conhecimentos num ambiente profissional, colaborando com colegas com diferentes conhecimentos e experiências, onde esta sinergia pode ser enriquecedora.

5. **Concluir o Mestrado e Ganhar Experiência Profissional:** Por último, durante o estágio, é pretendido conciliar o trabalho com a conclusão do mestrado, aplicando simultaneamente os conhecimentos adquiridos nas atividades académicas e profissionais.

## 1.2 Planeamento

No contexto de desenvolvimento de *software*, a gestão eficaz de projetos é fundamental para o sucesso e a entrega de produtos de qualidade dentro dos prazos estabelecidos. Para tal, pode ser utilizada a ferramenta ***Jira***, desenvolvida pela *Atlassian* e descrita em [4 (Atlassian, 2024)], que passa por uma ferramenta no mercado de gestão de projetos ágeis muito utilizada em contexto profissional, disponibilizando uma variedade de recursos para auxiliar as equipas a colaborar nos seus projetos. Desde a criação e atribuição de tarefas até ao acompanhamento do progresso e à gestão de *bugs*, o *Jira* oferece uma plataforma centralizada para organizar e acompanhar todas as etapas do ciclo de vida do desenvolvimento de *software*.

Figura 1.3: *Software Jira*

Uma das características distintivas do *Jira* é a sua flexibilidade e adaptabilidade às metodologias ágeis, como *Scrum* e *Kanban*. O *Scrum* é uma *framework* para gestão de projetos que enfatiza a colaboração, a entrega iterativa e a adaptação contínua. No *Scrum*, o trabalho é organizado em iterações chamadas “*sprints*”, em que cada um começa com uma reunião de planeamento, onde as tarefas a serem realizadas durante o *sprint* são definidas. Durante o *sprint*, a equipa realiza reuniões diárias rápidas (“*daily*”) de forma a discutir o progresso e identificar quaisquer obstáculos. No final de cada *sprint*, a equipa realiza uma revisão deste para avaliar o trabalho concluído e uma retrospetiva para identificar maneiras de melhorar o processo. Por outro lado, *Kanban* é um sistema de gestão visual concentrado na visualização do fluxo de trabalho, onde as tarefas são representadas por cartões (*tickets*) movidos por colunas num quadro *Kanban*, que representam diferentes etapas do processo. Cada coluna pode representar uma etapa específica do processo, como, por exemplo “Para fazer”, “Em progresso”, “A Testar”, “Concluído”, entre outros. Ao limitar o número de tarefas que podem estar em progresso simultaneamente, o *Kanban* ajuda a evitar a sobrecarga de trabalho e a manter um fluxo de trabalho mais consistente e eficiente, para além de também proporcionar uma melhoria contínua, incentivando as equipas a identificar e resolver problemas à medida que surgem. Com funcionalidades destas, o *Jira* permite que as equipas implementem processos ágeis de forma eficaz e que acompanhem o progresso de forma transparente. Além disso, o *Jira* integra-se com outras várias ferramentas e serviços utilizados no desenvolvimento de software, como *Bitbucket*, *Confluence* e *Slack*, proporcionando uma experiência de trabalho

mais dinâmica e completa para as equipas.

No entanto, para que nesta plataforma as tarefas possam ser averiguadas de forma a que os desenvolvedores as implementem, é feito um trabalho prévio pelos *designers*, posteriormente disponibilizado em cada cartão. Este trabalho inclui o *design* de qualquer componente ou funcionalidade a implementar já aprovado pelo cliente, dando aos desenvolvedores uma base para a implementação e consistência na interface da aplicação. Estes *designs* são realizados no ***Figma***, um editor gráfico e uma ferramenta de prototipagem colaborativa que permite criar protótipos interativos e *layouts* responsivos, oferecendo componentes reutilizáveis e bibliotecas de design. Esta ferramenta, muito utilizada por diversas empresas, centraliza a criação de *design* e melhora a comunicação entre as equipas de um projeto. Para mais informações sobre este editor, consultar [10 (Figma, 2024)].

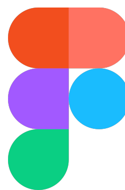


Figura 1.4: *Figma*

## 1.3 Organização do Projeto

Num contexto onde a colaboração e a eficiência são fundamentais para o desenvolvimento de *software*, a gestão de projetos e a coordenação nas equipas tornam-se elementos essenciais. É crucial simplificar e integrar todo o processo, desde a conceção até à implementação, de modo a garantir a qualidade e o sucesso do produto. É neste cenário que surge o ***GitLab*** ([12 (GitLab, 2024)]), uma plataforma de desenvolvimento de *software* que disponibiliza múltiplas ferramentas para gestão de código e colaboração em equipa.

Figura 1.5: Plataforma *GitLab*

Uma das características centrais do *GitLab* é o seu sistema de controlo de versões baseado em *Git*, que é um sistema que permite aos programadores gerir e controlar as alterações feitas no código-fonte de um projeto. O *GitLab* proporciona, portanto, um ambiente centralizado onde o código-fonte de um projeto é armazenado e gerido, significando que todas as alterações feitas no código ao longo do tempo podem ser acompanhadas e documentadas de forma

organizada. Com esta característica, os elementos da equipa podem criar funcionalidades ou corrigir *bugs* através da criação de ramificações (*branches*) no repositório do *GitLab*. Cada uma destas ramificações representa uma versão do código em desenvolvimento, permitindo que as alterações sejam feitas de forma isolada antes de serem integradas no código principal que está localizado na ramificação mestre (*branch master*).

Dentro do contexto de controlo de versões, a plataforma facilita ainda a colaboração entre os membros da equipa por meio de pedidos de fusão (*merge requests*), permitindo que estes revejam e discutam as alterações realizadas antes de serem importadas para o código principal. Estes pedidos promovem uma cultura de revisão e *feedback* entre os membros da equipa, contribuindo para a qualidade e consistência do código.

Para além destas características referidas, o *GitLab* oferece ainda uma variedade de ferramentas para gestão de projetos, dando a possibilidade das equipas poderem organizar e priorizar certas tarefas, acompanhar o progresso do projeto e automatizar o processo de implantação e testes do código.

É ainda importante realçar que todo o conteúdo do projeto está situado num ambiente seguro, garantindo que o código realizado e as informações do projeto encontram-se protegidos e acessíveis apenas para os elementos autorizados da equipa. Esta segurança e controlo de acesso são fundamentais de forma a garantir a integridade e confidencialidade do projeto.

# Capítulo 2

# Trabalho Relacionado

## 2.1 Aplicações Semelhantes

Neste capítulo, é realizada uma análise comparativa entre a aplicação móvel da Worten e outras aplicações semelhantes no mercado. Para isso, foram selecionadas aplicações de empresas que competem diretamente com a Worten no setor do comércio eletrónico, tais como a ***FNAC***, ***MediaMarkt***, ***Amazon*** e ***eBay***.

### 2.1.1 *FNAC*

A ***FNAC*** é uma cadeia de retalho de origem francesa fundada em 1945, especializada em produtos culturais e eletrónicos, que tem expandido a sua presença internacional, incluindo uma forte presença em Portugal. À semelhança da *Worten*, a *FNAC* destaca-se pela diversidade de produtos, que abrange desde livros, música e filmes até aos produtos de tecnologia. A *FNAC* é um concorrente direto da *Worten* no setor de retalho de eletrónica, competindo pela preferência dos consumidores tanto nas plataformas físicas como *online*.

Analisando as semelhanças entre a *FNAC* e a *Worten* é possível observar que ambas atuam no setor de retalho de eletrónica e eletrodomésticos, oferecendo uma ampla gama de produtos.

No entanto, analisando as diferenças, observa-se que enquanto a *Worten* tem um maior foco em eletrónica e eletrodomésticos, a *FNAC* investe também bastante na venda de produtos culturais, tendo por vezes eventos culturais, como lançamentos de livros, sessões de autógrafos, concertos e exposições. Apesar do serviço de assistência técnica da *FNAC* ser eficiente, a *Worten* possui um dos mais conhecidos serviços de apoio ao cliente com o nome de *Worten Resolve*, onde se destaca a capacidade de resolver problemas técnicos e prestar um serviço pós-venda de alta qualidade.

Figura 2.1: *FNAC*

A aplicação móvel da *FNAC* tem funcionalidades semelhantes à da *Worten*, permitindo aos utilizadores navegar, procurar e comprar produtos de uma vasta gama de categorias. Possuem ainda uma pesquisa avançada onde oferecem funcionalidades como filtros personalizados e sugestões de produtos adequados a cada utilizador. É também permitido em ambas as aplicações consultar a página da conta, podendo fazer a gestão desta ou visualizar o histórico de compras e lista de favoritos. Para além destas semelhanças, ambas as aplicações permitem consultar o estado do plano de fidelização (Cartão *FNAC* e Cartão *Worten* Resolve).

No entanto, a interface da *Worten* é mais focada em promoções e ofertas especiais, enquanto a da *FNAC*, para além de campanhas exclusivas, foca-se também em sugestões culturais ou tecnológicas com disponibilização de *reviews*. A *FNAC* também oferece a opção de compra de uma maior diversidade de bilhetes para eventos culturais através da aplicação, facilitando o acesso a estes. Por fim, e de realçar, a *FNAC* possui também uma secção dedicada a *blogs* e *reviews* de utilizadores, promovendo uma comunidade mais ativa com discussões sobre produtos e eventos.

Figura 2.2: *Blog* de sugestões/*reviews* da *FNAC*

### 2.1.2 *MediaMarkt*

A ***MediaMarkt*** é uma das maiores cadeias de retalho de eletrónica da Europa, fundada na Alemanha em 1979. Com uma presença global, incluindo várias lojas em Portugal, a *MediaMarkt* é reconhecida pela sua seleção de produtos eletrónicos e eletrodomésticos com preços competitivos e frequentes promoções. Compete diretamente com a *Worten*, oferecendo uma gama similar de produtos eletrónicos e eletrodomésticos tal como a *FNAC*, existindo assim dois concorrentes semelhantes.

Relativamente às semelhanças com a *Worten*, temos o setor de retalho em que ambas operam e as aplicações móveis que ambas disponibilizam. Além disso, ambas operam num mercado focado em tecnologia e equipamentos eletrónicos.

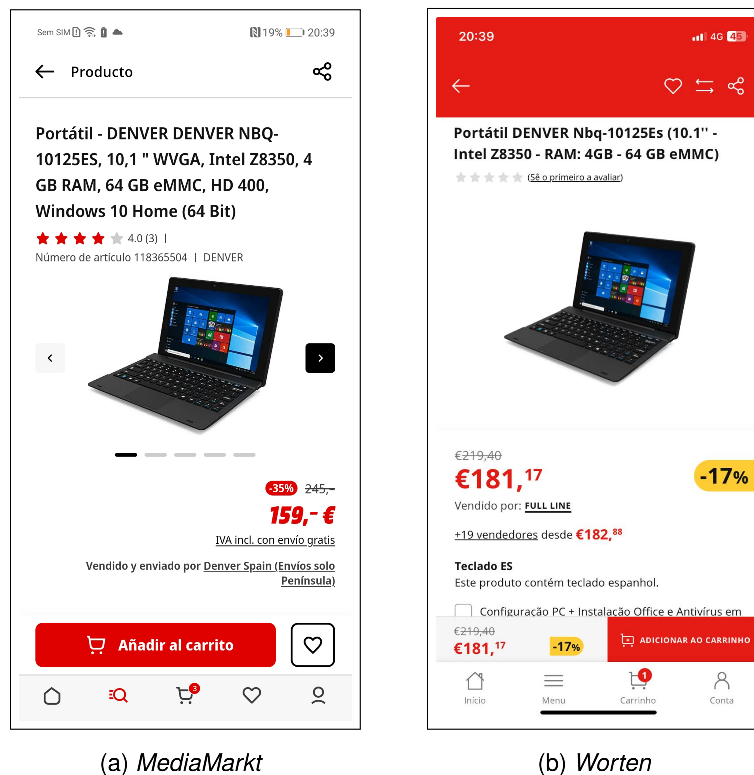
Analisando para as diferenças, vê-se que a expansão de ambas diferem, onde a *Worten* apostou no mercado ibérico (Portugal e Espanha), adaptando as suas estratégias de *marketing* às preferências locais. A *MediaMarkt*, por outro lado, apostou num mercado europeu, ou seja, mais generalizado.

Figura 2.3: *MediaMarkt*

A aplicação móvel da *MediaMarkt*, apesar de não disponível em Portugal, é a que mais se assemelha à da *Worten* entre todos os concorrentes. Esta facilita a navegação e compra de produtos, acompanhamento de encomendas e acesso a ofertas exclusivas. Para além destas semelhanças, ambas as aplicações possuem uma página de produtos semelhantes e várias categorias de pesquisa, destacando as promoções e descontos na página principal da aplicação. Esta aplicação oferece também, à semelhança da *Worten*, um localizador de lojas físicas, permitindo que os utilizadores encontrem as lojas mais próximas e verifiquem a disponibilidade de produtos. As duas aplicações oferecem também funcionalidades de suporte ao cliente, seja por *chat* ou por pedido de assistência telefónica.

Além da interface bastante semelhante, a aplicação da *MediaMarkt* sempre esteve muito focada na comparação e descrição de produtos para que o utilizador tenha a decisão de compra facilitada com base em especificações técnicas e avaliações, algo que, durante o estágio, foi desenvolvido na aplicação da *Worten*, comparando no máximo quatro produtos da mesma categoria, superior quando comparado com o limite máximo de 3 produtos na aplicação da *MediaMarkt*. Esta ênfase nas avaliações técnicas dos produtos fornece informações críticas aos utilizadores, ajudando-os a compreender de uma forma mais clara as características e o

desempenho dos produtos.




(a) *MediaMarkt* (b) *Worten*

Figura 2.4: Páginas de produto

### 2.1.3 Amazon

A ***Amazon***, fundada por *Jeff Bezos* em 1994, começou como uma livraria online e rapidamente se transformou numa das maiores plataformas de comércio *online* do mundo. Presente em vários mercados globais, incluindo Portugal, a *Amazon* oferece uma grande diversidade de produtos em múltiplas categorias, desde livros e eletrónica até moda e produtos para o lar. A *Amazon* é uma concorrente significativa da *Worten*, não apenas pela sua vasta oferta de produtos eletrónicos, mas também pela conveniência e rapidez dos seus serviços de entrega e pela amplitude do seu catálogo.

A *Amazon* assemelha-se à *Worten* na presença significativa no mercado eletrónico, apesar de não ser a área de foco da *Amazon*. Tanto uma como a outra proporcionam um serviço de entregas rápidas para os seus clientes, podendo entregar produtos dentro do mesmo dia.

No entanto, a *Amazon* é uma plataforma de comércio eletrónico que também vende produtos de terceiros, enquanto a Worten é uma cadeia de retalho físico e *online* que vende principalmente produtos próprios. Além disso, a *Amazon* oferece uma variedade de serviços adicionais, como *Amazon Prime*, *Amazon Music* e *Amazon Web Services*, que não são ofere-

cidos pela *Worten*.

Figura 2.5: *Amazon*

A aplicação móvel da *Amazon* proporciona aos utilizadores uma experiência de compra conveniente, com funcionalidades como recomendações personalizadas, acompanhamento de encomendas e acesso ao serviço de subscrição *Amazon Prime*, que inclui entregas rápidas e outros benefícios. Tal como a aplicação da *Worten*, oferecem uma vasta gama de produtos desde a eletrónica até à moda, cosmética e muito mais. Ambas utilizam algoritmos de recomendação para sugerir novos produtos com base na atividade do utilizador. E, para além destas semelhanças, as aplicações permitem comprar um produto com entrega rápida, indo desde a *Worten* que oferece uma entrega em 15 minutos na loja ou uma entrega de 2 horas em casa, até à *Amazon* que oferece uma entrega entre o próprio dia e o seguinte.

No entanto, a aplicação da *Amazon* oferece uma variedade de serviços adicionais que a *Worten* não disponibiliza, como *Prime Video*, *Amazon Music* e *Prime Reading*. Apesar da *Worten* possuir diversos vendedores, tem um *marketplace* mais virado para vendas próprias, enquanto o *marketplace* da *Amazon* é centrada num modelo com múltiplos vendedores. Além disso, a aplicação fornece a opção de alterar a região onde é pretendido realizar as compras.

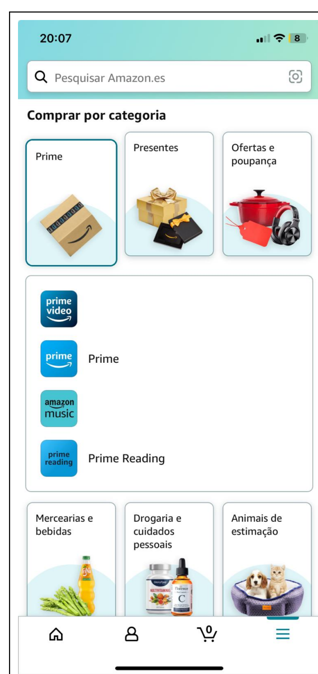



Figura 2.6: Serviços *Amazon*

### 2.1.4 eBay

O ***eBay*** é uma plataforma de comércio eletrónico fundada em 1995 por *Pierre Omidyar*, semelhante à *Amazon*, que permite a compra e venda de produtos novos e usados entre utilizadores individuais e empresas, tal como a licitação ou venda de produtos em leilões *online*. O *eBay* é um concorrente da *Worten* devido à sua capacidade de atrair consumidores que procuram tanto produtos novos quanto usados, oferecendo uma alternativa diversificada e frequentemente mais económica para a compra de eletrónica e outros bens. Esta marca assemelha-se à *Amazon* quando comparada com a *Worten*, onde esta também atua como plataforma de compra e venda de produtos de várias categorias (abrangendo mais diversidade que a *Worten*).

Observando as diferenças, entende-se que, à semelhança com a *Amazon*, o *eBay* é uma plataforma de *marketplace* que conecta compradores e vendedores individuais e empresariais, enquanto a *Worten* é uma cadeia de retalho que vende principalmente produtos próprios. As políticas de compra e venda podem também ser diferentes entre as duas plataformas, com o *eBay* dando mais ênfase à negociação e ao leilão de produtos. Ao contrário da *Worten* e referido na subsecção **2.1.3 *Amazon***, o *eBay* não possui nenhuma forma de entrega rápida. Esta comparação entre a *Amazon* e o *eBay* pode ser aprofundada ao consultar [23 (Slade, 2024)].

Figura 2.7: *eBay*

A aplicação móvel do *eBay* facilita a participação em leilões ao contrário da aplicação da *Worten*, mas assemelha-se com esta no acompanhamento de encomendas e a gestão de vendas, oferecendo uma boa experiência de comércio *online*. Dada a existência de leilões, esta plataforma permite que vendedores terceiros listem e vendam os seus produtos, algo que, por diferentes razões, a plataforma da *Worten* também possibilita. No entanto, as diferenças das plataformas prendem-se muito com esta diferença de modelos de vendas, onde o *eBay* está também direcionado para a venda de leilões, enquanto a da *Worten* tem uma navegação mais orientada para um *marketplace* comum com eventos de promoções e ofertas.

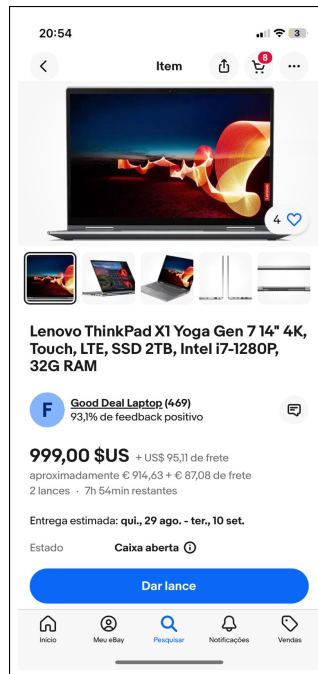



Figura 2.8: Leilão no *eBay*

Esta aplicação assemelha-se mais com a da *Amazon* do que com a da *Worten*, sendo a que menos de assemelha a esta. A principal razão prende-se, portanto, com o modelo de negócio e funcionalidades oferecidas.

## 2.2 Tecnologias Multi-Plataforma Semelhantes

***React Native*** é uma *framework* de desenvolvimento de aplicações móveis criado pelo *Facebook* em 2015. Baseado na biblioteca *JavaScript React*, esta *framework* permite criar aplicações nativas para *Android* e *iOS* utilizando o mesmo código. Uma das grandes vantagens do *React Native* é a sua capacidade de renderizar componentes nativos utilizados a partir de componentes *React*, proporcionando uma elevada eficiência e uma boa experiência de utilizador, para além da diversidade de bibliotecas e ferramentas que facilitam o desenvolvimento de aplicações móveis. Este modelo híbrido permite que as aplicações *React Native* ofereçam um desempenho quase igual ao das aplicações nativas, mas mantendo ao mesmo tempo, a flexibilidade do desenvolvimento *web*. A familiaridade com *JavaScript*/*TypeScript* e *React* torna-o uma boa escolha para programadores *web* que tencionam aumentar as suas competências no desenvolvimento de aplicações móveis. Para uma explicação mais detalhada e exemplos de aplicações que fazem uso desta tecnologia, consultar [8 (Budziński, 2024)].

Figura 2.9: *React Native*

Esta necessidade de alcançar utilizadores em múltiplas plataformas, como *Android* e *iOS*, levou ao aparecimento de diversas linguagens e *frameworks* de desenvolvimento multi-plataforma. Para além de *React Native*, existem outras opções bastante populares no mercado, como *Flutter*, *Xamarin* (novo *.NET MAUI*) e *Ionic*. Pode ser consultada uma breve comparação destas três linguagens e *React Native* em [1 (AgiliWay, 2022)].

### 2.2.1 Flutter

***Flutter***, tal como pode ser observado em [13 (Google, 2024)], é uma *framework* de desenvolvimento de aplicações móveis criado pela *Google* em 2017. Esta *framework* permite aos programadores criar aplicações nativas para *Android* e *iOS* tal como o *React Native*, mas utilizando a linguagem de programação *Dart*. A sua maior diferença é a utilização do seu próprio motor gráfico chamado *Skia*, que fornece uma interface consistente para ambas as plataformas. Com recursos como *Hot Reload*, o *Flutter* facilita o desenvolvimento rápido ao permitir ver as alterações realizadas no código em tempo real, facilitando o desenvolvimento e a correção de erros. A popularidade do *Flutter* tem crescido rapidamente, tornando-se uma escolha atrativa para programadores que procuram uma solução moderna para desenvolvimento multi-plataforma, com um desempenho elevado visto que a compilação é feita diretamente para código nativo.

No entanto, podem existir alguns pontos negativos na utilização desta linguagem, tal como as aplicações em *Flutter* poderem ser mais pesadas devido à utilização do motor de renderização *Skia*. Além disso, apesar da comunidade estar a crescer rapidamente, ainda é muito menor que a de *React Native*, podendo não existir tanto suporte. *Dart* é também uma linguagem menos utilizada que *JavaScript*/*TypeScript*, o que poderá exigir mais tempo de aprendizagem.

Figura 2.10: *Flutter*

### 2.2.2 *Xamarin*

***Xamarin*** (novo *.NET MAUI*) é uma *framework* de desenvolvimento de aplicações móveis adquirido pela *Microsoft* em 2016, onde o suporte desta foi finalizado em Maio de 2024 quando foi realizada a migração para *.NET MAUI*. Utilizando *C#* e a plataforma *.NET*, esta *framework* permite o desenvolvimento de aplicações nativas para *Android* e *iOS* onde uma das principais vantagens é a sua integração profunda com o ambiente da *Microsoft*, tornando-o ideal para empresas que já utilizam tecnologias como *Azure* e *.NET*, podendo aceder diretamente às *API* (*Application Programming Interface*) nativas de cada plataforma, garantindo uma boa experiência de utilizador. Esta linguagem continua a ser uma escolha sólida para o desenvolvimento de aplicações empresariais e comerciais com ajuda da alta reutilização de código entre plataformas, sendo possível chegar a uma taxa de 90%. A partilha de código em *React Native* também é alta, mas geralmente não atinge os níveis de *Xamarin* devido à necessidade de escrever código nativo para algumas funcionalidades específicas. Pode ser encontrada uma descrição mais detalhada sobre esta *framework* e a sua sucessora *.NET MAUI* em [16 (Kathiresan, 2024)].

No entanto, esta opção pode não ser tão eficiente quanto soluções como *React Native* que compilam diretamente para código nativo. Existe também uma curva de aprendizagem mais acentuada devido ao uso de *C#* e *.NET* o que pode tornar o desenvolvimento inicial mais complicado e demorado. Além disso, a comunidade de *Xamarin* é ainda menor que *Flutter* e por consequente de *React Native*.

(a) *Xamarin* (b) *.NET MAUI*

Figura 2.11: *Xamarin* e *.NET MAUI*

### 2.2.3 Ionic

***Ionic*** é uma *framework* de desenvolvimento de aplicações híbridas, lançado em 2013, que permite a criação de aplicações móveis utilizando tecnologias *web* familiares como *HTML*, *CSS* e *JavaScript*. Inicialmente construído sobre *Angular*, o *Ionic* oferece suporte a outras *frameworks* populares como *React* e *Vue*. A grande vantagem de *Ionic* é a sua familiaridade para programadores *web*, permitindo uma rápida transição para o desenvolvimento móvel. *Io-*

*nic* utiliza componentes pré-construídos que se enquadram no *design* de cada plataforma, proporcionando uma experiência de utilizador consistente. Esta tecnologia, com o auxílio de vastas bibliotecas de componentes *UI* que se adaptam ao *design* nativo e de outras ferramentas como *Ionic CLI* e a integração com *Capacitor* (antigo *Cordova*), tornou-se uma escolha popular para *startups* e empresas que procuram um desenvolvimento acessível.

*Ionic CLI* é uma linha de comandos que facilita o processo de desenvolvimento de aplicações com *Ionic*. Fornece uma série de comandos que permitem iniciar novos projetos, adicionar componentes, compilar a aplicação para diferentes plataformas, executar emuladores, e muito mais. Por outro lado, o *Capacitor*, é uma plataforma de *runtime* desenvolvida pela equipa de *Ionic* que permite criar aplicações *web* modernas que podem ser implementadas nativamente em dispositivos móveis. Permite que os programadores acedam funcionalidades nativas dos dispositivos móveis usando *JavaScript*, mantendo a experiência de desenvolvimento *web*.

No entanto, como se trata de uma solução híbrida, o desempenho geralmente é inferior ao de *frameworks* nativas ou que compilam diretamente para código nativo. Isto ocorre porque as aplicações *Ionic* são renderizadas numa *WebView*, o que pode introduzir limitações e inconsistências de desempenho. Em situações onde o desempenho e a fluidez são pontos críticos, como em aplicações com altos requisitos gráficos, essas diferenças tornam-se ainda mais evidentes. Para consultar mais informação sobre esta *framework*, pode ser utilizada a documentação disponível em [15 (Ionic, 2024)].

Figura 2.12: *Ionic*

# Capítulo 3

# Tecnologias Utilizadas

No desenvolvimento do projeto, são usadas várias tecnologias desde a área da programação propriamente dita, à infraestrutura informática que o suporta, aos testes da aplicação, como ainda à gestão do projeto. Este capítulo apresenta estas principais tecnologias e *frameworks* utilizados.

## 3.1 *Frameworks*

Tal como já referido anteriormente, o *framework* utilizado neste projeto foi o *React Native*, um dos *frameworks* mais populares para a criação de aplicações móveis multi-plataforma. Foram utilizadas diversas bibliotecas integradas no *React Native* para melhorar e simplificar a aplicação. O *React Navigation* foi utilizado para a navegação dentro da aplicação, oferecendo uma solução robusta e flexível para gerir a navegação em aplicações *React Native*. O *React Native Elements* foi utilizado de forma a construir interfaces consistentes e eficientes para os utilizadores, fornecendo um conjunto de componentes já previamente criados, para além da grande customização oferecida em cada um. Foi também utilizado o *React Native Reanimated* para criar animações fluídas e, consequentemente, melhorar a experiência do utilizador na aplicação.

## 3.2 Bibliotecas

Este projeto integra diversas bibliotecas que disponibilizam várias das funcionalidades existentes na aplicação. Tenta-se, dentro do possível, integrar aquilo que se encontra disponível para tornar o desenvolvimento mais rápido e ágil.

Como primeira e mais significativa, foi utilizada uma biblioteca de nome *Design System*

(*DS*), realizada pela equipa da *Worten* para garantir consistência no *design* de todas as interfaces da aplicação, facilitando a manutenção e a escalabilidade do projeto. A determinada altura foi também integrada a biblioteca *i18next*, sendo esta essencial para a internacionalização da aplicação, permitindo que esta suportasse múltiplos idiomas de forma dinâmica. Para renderizar animações em *JSON*, foi integrada no projeto a biblioteca *lottie-react-native*, que simplifica a integração de animações complexas (como, por exemplo, animações de *loading*, como será visto no capítulo **5 Trabalho Realizado**).

Para além destas, existem outras bibliotecas utilizadas no projeto, mas que não fizeram parte do desenvolvimento deste durante o estágio. Para a gestão de estado, é utilizada a biblioteca *MobX*, que facilita a manutenção e a atualização do estado da aplicação, melhorando a eficiência e o desempenho. Além disso, é também utilizada a biblioteca *Axios* para realizar pedidos *HTTP* que facilita a comunicação com *API*s. Por fim, o *Firebase* foi utilizado em vários contextos, como a monitorização e análise do uso da aplicação através do *Firebase Analytics*, o relato de falhas e erros com o *Firebase Crashlytics*, a edição da aparência e comportamento da aplicação sem a necessidade de lançamentos de novas versões com o *Firebase Remote Config*, a implementação de notificações *push* com o *Firebase Messaging* e outras funcionalidades.

## 3.3 *IDE* (*Integrated Development Environment*)

Para o desenvolvimento do projeto foi aconselhada a utilização do *WebStorm*, um *IDE* desenvolvido pela *JetBrains*. Este *IDE* oferece uma excelente integração com *JavaScript* e *TypeScript*, e *frameworks* associados, além de ferramentas de *debugging*, *autocomplete* e suporte a várias tecnologias *web* como *HTML/CSS*, *Node.js*, *React* e outras, possibilitando um desenvolvimento integrado.

Figura 3.1: *WebStorm*

## 3.4 Ferramentas de Teste

Para garantir a qualidade da aplicação, foram utilizadas diversas ferramentas de teste. Para começar, ao testar cada alteração de código realizada nas tarefas diárias, é utilizado o emulador do *Android Studio* para testar a aplicação num ambiente virtual *Android*, permitindo simular e testaras em diferentes dispositivos.

Por outro lado, o *Firebase App Distribution* facilitou a distribuição de versões de teste da aplicação para um grupo selecionado de utilizadores, permitindo *feedback* e identificação de problemas antes do lançamento da aplicação de produção para as lojas *online*.

Figura 3.2: *Firebase App Distribution*

Nesta interface são disponibilizadas dois tipos de aplicação, *QA* (*Quality Assurance*), onde a aplicação disponibilizada contém as melhorias realizadas na nova versão, mas ainda não se encontra no estado de produção, e *Dev* (*Developer*), onde a aplicação disponibilizada encontra-se num estado possível de efetuar encomendas e testar outras dinâmicas não possíveis em *QA*.

(a) *Worten QA*

(b) *Worten Dev*

Figura 3.3: Aplicações de Teste no *Firebase App Distribution*

Para dispositivos *iOS*, foi também utilizado a aplicação *TestFlight*, essencial para obter *feedback* de utilizadores *iOS* e garantir o funcionamento na aplicação em estado de produção.

Figura 3.4: *TestFlight*

## 3.5 Ferramentas de Gestão de Projeto e Controlo de Versões

Para a gestão de projetos, a ferramenta escolhida foi o *Jira*, já explicada anteriormente na secção **1.2 Planeamento**. Tal como referido, o *Jira* é uma ferramenta para planeamento, acompanhamento e gestão de projetos, onde se podem organizar todas as tarefas, facilitando a colaboração dentro deste.

Como sistema de controlo de versões, foi utilizado o *GitLab*, também já mencionado anteriormente na secção **1.3 Organização do Projeto**, essencial para o desenvolvimento colaborativo. O *Git* permite rastrear todas as mudanças no código, gerir diferentes versões do projeto e colaborar com outros desenvolvedores, estando integrado nas plataformas *GitHub* e *GitLab*, também utilizadas para gestão de repositórios.

No contexto deste estágio, o *Jira* desempenhou um papel fundamental na organização e gestão das tarefas, permitindo que a equipa acompanhasse o progresso, identificasse obstáculos e alcançasse os objetivos de forma eficaz e colaborativa.

# Capítulo 4

# Metodologia

O ciclo de vida do desenvolvimento de software (*SDLC* - *Software Development Life Cycle*) é um processo cíclico utilizado no desenvolvimento de *software* de maneira que seja possível planear, implementar e implantar *software* de forma organizada. Este processo define uma série de fases sequenciais e atividades que guiam o desenvolvimento de um projeto de *software* desde a sua conceção até ao seu lançamento, garantindo que todos os aspetos do desenvolvimento sejam abordados de maneira sistemática.

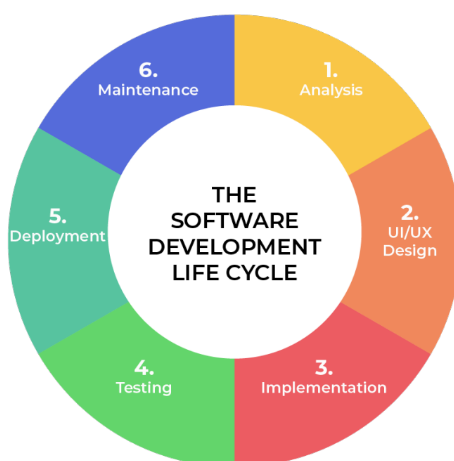



Figura 4.1: Ciclo de Vida do Desenvolvimento de Software

Analisando o ciclo da **figura 4.1** e as informações apresentadas em [14 (Harris, 2021)], pode ser observado que o desenvolvimento, para além do planeamento, implementação e implantação, divide-se em seis blocos que representam fases distintas do processo.

Como primeiro bloco tem-se a **Análise** das tarefas/objetivos a realizar, onde são coletados e documentados os requisitos funcionais e não funcionais do sistema em desenvolvimento. Os requisitos funcionais (requisitos ativos) referem-se às funcionalidades que o sistema deverá fornecer, ou seja, o que os utilizadores poderão efetuar na aplicação para atender às

suas necessidades, enquanto os requisitos não funcionais (requisitos passivos) descrevem as características ou possibilidades do sistema, como, por exemplo, o desempenho, segurança, usabilidade ou escalabilidade da aplicação. Portanto, é preciso entender as necessidades dos utilizadores, do cliente e de outras partes interessadas para definir o que é necessário desenvolver. No caso deste projeto, os requisitos são estabelecidos pelo cliente à medida que o projeto vai crescendo, para que depois sejam transmitidos de forma clara aos *designers* no segundo bloco do ciclo.

O segundo bloco refere-se ao **Design de *UI*/*UX***, onde o objetivo é criar uma arquitetura da aplicação com base nos requisitos documentados no bloco anterior. É então projetada uma estrutura detalhada de cada tarefa a implementar, com a interface e os componentes necessários. Na *Worten*, os *designers* formam a interface em *Figma* detalhadamente e com medidas já definidas para que esta seja transmitida aos desenvolvedores para dar início ao terceiro bloco.

Neste ciclo de desenvolvimento, existe agora um dos blocos fundamentais, o **Desenvolvimento**, que descreve a implementação do *software* conforme a arquitetura feita no bloco anterior (**Design de *UI*/*UI***). Existe, portanto, a criação de funcionalidades conforme os requisitos estipulados pelo cliente e desenhados pelos *designers*, garantindo um desenvolvimento conforme o esperado de cada componente. Este bloco, para além de fundamental, é o mais importante no contexto deste estágio, visto que é conduzido pela equipa de desenvolvedores *mobile*, onde foi investido o maior tempo de todos os blocos, tanto na aprendizagem como na concretização.

O quarto bloco deste ciclo - **Testes** - também se destaca como um dos mais importantes para o contexto do estágio. Neste é verificado se o *software* realizado no bloco do **Desenvolvimento** satisfaz os requisitos especificados e se não possui nenhum defeito ou *bug*. Os testes são realizados em várias etapas, desde testes unitários para componentes individuais, até testes gerais da aplicação aquando de um lançamento de uma nova versão, garantindo que esta funciona corretamente. Este bloco, no seu formato normal, será responsabilidade dos *testers* (*QA*), no entanto, no contexto deste estágio, grande parte dos testes foram realizados pelos mesmos desenvolvedores do bloco anterior. Caso sejam testes unitários, os componentes são testados através dos emuladores tanto de *iOS* como de *Android*, visto que a versão da aplicação disponibilizada no *TestFlight* ou nas aplicações de *QA* e *Dev* da *Worten* ainda não possui as novas alterações. Isto leva-nos, então, ao próximo bloco.

O próximo bloco trata da **Implantação**/**Distribuição** da aplicação, onde o objetivo é colocar o novo *software* implementado em operação num ambiente de teste (*QA*, *Dev* e *TestFlight*) e

produção. É feita a atualização da aplicação com novas funcionalidades e melhorias recorrendo a versões incrementais, incluindo a disponibilização inicial em ambientes de teste e, após validação, a disponibilização para os utilizadores. Portanto, no contexto do estágio, este bloco deveria existir duas vezes, estando situado antes e depois do bloco de **Testes**, para existir a distribuição para teste e distribuição para produção. Esta implantação/distribuição é realizada por um desenvolvedor da equipa.

Por fim, como último bloco do ciclo, surge o bloco **Manutenção**, onde, como o nome indica, é garantido que o *software* desenvolvido até à altura continue em funcionamento de forma eficaz e com o mínimo de problemas possíveis. Este processo envolve corrigir quaisquer *bugs* existentes, realizar atualizações de melhoria de desempenho ou implementar novas funcionalidades e fornecer suporte contínuo aos utilizadores. Dado que a aplicação desenvolvida é uma aplicação em constante mudança, pode ser observado que este ciclo é constantemente reiniciado, em que a cada atualização ou funcionalidade nova, é feita uma análise de requisitos, os *sketchs*, a implementação em si, os testes, a distribuição de uma nova versão com estas atualizações/funcionalidades e, como passo final do ciclo, a manutenção da aplicação com as novas tarefas implementadas.

# Capítulo 5

# Trabalho Realizado

Durante este estágio, foram desenvolvidas várias tarefas da aplicação móvel da *Worten*, desde a configuração inicial e desenvolvimento de funcionalidades específicas até à realização de testes e gestão de tarefas. Este capítulo detalha estas tarefas, distribuídas por diferentes áreas do desenvolvimento da aplicação.

Inicialmente, foi realizado um curso de *React Native* de forma a adquirir competências iniciais necessárias para o desenvolvimento do projeto, onde foi possível entender pontos fundamentais como a configuração do ambiente de desenvolvimento, navegação, utilização de bibliotecas, alguns componentes essenciais e outros tópicos que serão falados a seguir.

Com o conhecimento adquirido através do curso, o autor foi integrado no desenvolvimento do *Design System*, reconstruindo a arquitetura de estilos e tipos de componentes existentes, criando componentes e melhorando a documentação. Foi também integrado mais tarde na aplicação principal, onde conseguiu contribuir no desenvolvimento de componentes, na implementação de novas funcionalidades, em campanhas realizadas durante o período do estágio, na nova área de cliente, em traduções para diferentes idiomas, entre muitas outras tarefas.

O capítulo também aborda o trabalho realizado em testes de qualidade (*QA*) e a utilização do *Jira* para a gestão das tarefas do projeto. Estes dois contextos foram importantes para garantir a qualidade e consistência do desenvolvimento da aplicação.

## 5.1 Curso React Native

### 5.1.1 *Setup*

Ao iniciar o curso comecei por realizar a instalação do ambiente de desenvolvimento, incluindo a configuração do *Android Studio* e do *WebStorm* (*IDE*) e a instalação do ***npm*** e ***yarn*** (gestor de dependências). Tanto um como o outro são gestores de pacotes de *JavaScript*, fer-

ramentas essenciais no desenvolvimento em *React Native*. Apesar do *yarn* ser uma alternativa desenvolvida para ser mais rápida e segura do que o *npm*, durante o estágio foram utilizados os comandos *npm* e ***npx*** ($npx\ expo\ start$) para a execução do *Design System*, e os comandos do tipo *yarn* ($yarn\ android - win$) para a execução da *app* de produção. O *npx* (*Node Package Execute*) vem com o *npm* quando este é instalado e é um executor de pacotes do *npm* que pode executar qualquer um do registo do *npm* sem o instalar. Para mais informações sobre estas ferramentas de gestão e execução de pacotes *JavaScript*, pode ser consultado o conteúdo apresentado em [6 (Braga, 2023)].

Foram instaladas também duas ferramentas, o ***React Native CLI*** e o ***Expo***, com a descrição e a comparação entre elas disponíveis em [11 (Flatirons, 2024)]. O *React Native CLI* foi instalado para a realização deste curso, visto que esta é a interface de linha de comandos oficial para criar e gerir projeto do *React Native*, possuindo maior flexibilidade do que outras ferramentas e maior acesso a bibliotecas nativas. No entanto, possui uma configuração mais complexa quando comparada com alternativas como o *Expo*, um conjunto de ferramentas e serviços visando simplificar o desenvolvimento em *React Native*, abstraindo muitas das configurações nativas e fornecendo um ambiente mais simplificado. Este *Expo*, apesar de mais limitado e com menor flexibilidade, foi utilizado não só neste curso, mas também no projeto desenvolvido durante o estágio. Para tal, foi necessária a instalação do ***Node.js*** no sistema e a instalação de um gestor de pacotes já descritos anteriormente.

Para a conclusão do *setup* do curso, foram criadas duas aplicações em que uma utilizou os comandos nativos da linha de comandos *React Native CLI* (*noexpoapp*) e outra utilizou os comandos do *Expo* (*expoapp*), mostrando a sua simplicidade. Ambas as aplicações foram executadas no emulador do *Android Studio* e concluiu-se que a aplicação criada com *Expo*, quando é iniciada através do comando "*expo start*", é executada e visualizada através da plataforma ***Expo Go***, aplicação disponível para *iOS* e *Android* que atua como um "contentor" de aplicações, permitindo testar código no dispositivo/emulador sem a necessidade de compilar o código nativo. Esta plataforma permite um desenvolvimento mais rápido com a possibilidade de visualizar as mudanças em tempo real à medida que o código é alterado.

Figura 5.1: *Expo Go*

### 5.1.2 Noções Básicas

Após concluído o *setup*, foi possível começar a entender os conceitos básicos do *React Native*, começando pelos componentes nativos de base apresentados abaixo. É de notar que o código apresentado de seguida é em *TypeScript*.

- ***View*:** Um contentor básico para outros componentes, suporta *layout* com *flexbox* e estilos. Serve para agrupar outros componentes. Exemplo:

Listing 5.1: View

```
<View style={{width: '100%', justifyContent: 'center', alignItems:
    'center'}}>
    {/* Outros componentes */}
</View>
```

- ***Text*:** Componente utilizado para exibir qualquer tipo de texto. Exemplo:

Listing 5.2: Text

```
<Text style={{fontSize: 20}}>
    Hello, World!
</Text>
```

- ***Image*:** Componente para exibir imagens locais ou remotas. Exemplo:

Listing 5.3: Image

```
<Image
source={{uri: 'https://example.com/image.jpg'}}
style={{width: 100, height: 100}}
/>
```

- ***ScrollView*:** Um contentor que permite fazer *scroll* pelo conteúdo que excede o tamanho deste componente. Tem uma função semelhante ao componente *View* mas com a possibilidade de realizar *scroll*. Exemplo:

Listing 5.4: ScrollView

```
<ScrollView>
    <Text>Componente 1</Text>
    <Text>Componente 2</Text>
    {/* Outros componentes */}
</ScrollView>
```

- ***Button***: Um componente básico de botão que gere os cliques realizados. Realiza uma ação aquando do clique. Exemplo:

Listing 5.5: Button

```
<Button title="Press Me" onPress={() => alert('Button Pressed!')} />
```

- ***TouchableOpacity***: Um contentor que faz o seu conteúdo clicável. É um componente alternativo ao componente ***Button***. Exemplo:

Listing 5.6: TouchableOpacity

```
<TouchableOpacity onPress={() => alert('Pressed!')}>
    <Text>Press Me</Text>
</TouchableOpacity>
```

- ***TextInput***: Componente que permite ao utilizador introduzir texto. Exemplo:

Listing 5.7: TextInput

```
<TextInput
placeholder="Escrever aqui"
style={{height: 40, borderColor: 'gray', borderWidth: 1}}
/>
```

Depois da introdução a estes componentes nativos, foi introduzida a noção de ***props*** e ***states***.

### ***Props***

As *props* são **argumentos/propriedades** que permitem que sejam transmitidos dados de um componente pai para um componente filho, tornando estes componentes dinâmicos e reutilizáveis. Para um melhor entendimento destas propriedades, pode ser consultado o conteúdo em [20 (Platforms, 2024)] e, observado o código apresentado abaixo, onde estão presentes o componente pai "*MainComponent*" (onde faz uso do componente filho e lhe passa certos argumentos) e o componente filho "*SubComponent*" (utiliza os argumentos vindos do componente pai).

Desta forma, o componente pai consegue renderizar dois textos representados pelo mesmo componente:

Listing 5.8: Props no componente pai

```
const App = () => {
  return (
    <View>
      <SubComponent name="Charles" />
      <SubComponent name="Lewis" />
    </View>
  );
};
```

O componente filho recebe apenas uma *prop* que será o nome de cada pessoa, utilizado no componente *Text*:

Listing 5.9: Props no componente filho

```
const SubComponent = (props) => {
  return <Text>Hello {props.name}!</Text>;
};
```

Com a utilização destes dois componentes obtém-se o seguinte resultado aquando da renderização do componente pai:

Listing 5.10: Output com Props

```
Hello Charles!
Hello Lewis!
```

### States

Os *states* são objetos que permitem que os componentes façam uma **gestão de dados** e sejam **responsivos a alterações** em tempo real, tal como referido em [21 (Platforms, 2024)]. Ao contrário das *props*, um *state* é normalmente gerido dentro do componente e pode ser modificado dentro deste. Além disso, um *state* visa armazenar dados que podem ser alterados ao longo do tempo e influenciam a renderização do componente, enquanto uma *prop* só procura passar dados de um componente para o outro.

Um *state* tem a estrutura mostrada abaixo, onde é identificado por um nome e uma função modificadora do valor do *state*. Pode ainda ser definido um valor inicial para o *state* caso este seja utilizado na renderização inicial do componente:

$$const\,[stateName, setStateName] = useState(< initial_value >);$$

A seguir é apresentado um código exemplo do componente "*Counter*", onde é renderizado um botão que a cada clique incrementa o valor do *state count* através da função *setCount*. Este valor é inicializado a 0.

Listing 5.11: States

```
const Counter = () => {
  const [count, setCount] = useState(0);

  return (
    <View>
      <Text>Count: {count}</Text>
      <Button title="Increment" onPress={() => setCount(count + 1)} />
    </View>
  );
};
```

Após entender a gestão e passagem de dados dos componentes, foi introduzido o uso de estilos utilizando o objeto ***StyleSheet***.

### Estilos com *StyleSheet*

Os estilos em *React Native* são implementados de forma semelhante ao ***CSS*** no desenvolvimento *web*, no entanto, são utilizados objetos *JavaScript* em vez de arquivos *CSS* e os estilos não são herdados automaticamente, é necessário especificar para cada componente.

Para tal, é utilizado o componente ***StyleSheet*** apresentado em [22 (Platforms, 2024)], que permite criar estilos de uma forma estruturada.

Para a utilização deste componente, após a sua importação através do pacote "*react-native*", é utilizada a função *create()* de forma a criar estilos evitando carregá-los a cada renderização.

Listing 5.12: *StyleSheet*

```
const styles = StyleSheet.create({
  container: {
    justifyContent: 'center',
    alignItems: 'center',
    backgroundColor: 'lightblue',
  },
  text: {
    fontSize: 20,
    color: 'darkblue',
  },
});
```

Neste código, o estilo *container* será utilizado num componente *View* e o estilo *text* num componente *Text*. Assim, em vez dos estilos estarem definidos em cada componente de forma ***inline*** como

Listing 5.13: Estilos *inline*

```
style={{justifyContent: 'center', alignItems: 'center', backgroundColor:
    'lightblue',}}
```

são definidos utilizando o objeto criado na *StyleSheet*, ou seja, $style = \{styles.container\}$.

Listing 5.14: Estilos com *StyleSheet*

```
const StylesExample = () => {
  return (
    <View style={styles.container}>
      <Text style={styles.text}>Hello, World!</Text>
    </View>
  );
};
```

Para finalizar as noções básicas foi ainda introduzida um modelo de *layout* denominado ***Flexbox***.

### ***Layouts* com *Flexbox***

O *Flexbox* é um **modelo de *layout*** com o propósito de construir interfaces responsivas e flexíveis. É utilizado para alinhar e distribuir espaço entre componentes num contentor, adaptando-os para diferentes tamanhos de ecrã.

Antes da introdução das propriedades do *Flexbox*, é necessário definir os eixos em que o *Flexbox* opera:

- **Eixo Principal:** Eixo ao longo do qual os componentes vão sendo posicionados. Este eixo, por omissão, está configurado como vertical (***column***).
- **Eixo Transversal:** Eixo perpendicular ao eixo principal.

Tendo conhecimento dos eixos operáveis, podemos entender de que forma é que os componentes são adaptados através das múltiplas propriedades. As propriedades são passadas aos estilos de cada componente como exemplificado em **5.1.2 Estilos com *StyleSheet***. A seguir são detalhados três exemplos amplamente utilizados durante o estágio:

1. ***flexDirection*:** Define a direção do **eixo principal** e por consequente a orientação dos componentes dentro do contentor (componente pai):
    - ***row*:** Componentes alinhados horizontalmente da esquerda para a direita.
    - ***column*:** Componentes alinhados verticalmente de cima para baixo (**padrão**).
    - ***row-reverse*:** Componentes alinhados horizontalmente da direita para a esquerda.
    - ***column-reverse*:** Componentes alinhados verticalmente de baixo para cima.

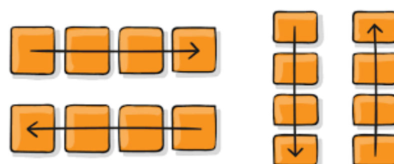


Figura 5.2: Propriedade *flexDirection*

2. ***justifyContent*:** Define o posicionamento dos componentes e o seu espaçamento ao longo do eixo principal:
    - ***flex-start*:** Componentes alinhados no início do eixo principal (**padrão**).

- ***flex-end***: Componentes alinhados no final do eixo principal.
- ***center***: Componentes alinhados no centro do eixo principal.
- ***space-between***: Espaçamento igual entre os componentes.
- ***space-around***: Espaçamento igual ao redor os componentes.
- ***space-evenly***: Espaçamento igual entre os componente e as extremidades do contentor.

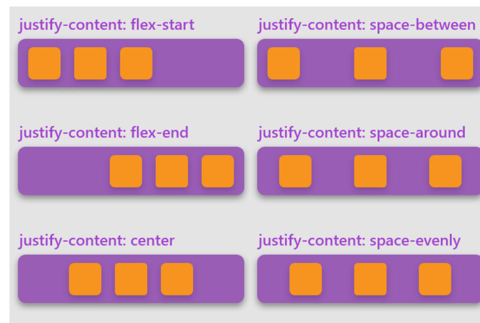



Figura 5.3: Propriedade *justifyContent*

3. ***alignItems***: Define o posicionamento dos componentes e o seu espaçamento ao longo do eixo transversal:

   - ***flex-start***: Componentes alinhados no início do eixo transversal.
   - ***flex-end***: Componentes alinhados no final do eixo transversal.
   - ***center***: Componentes alinhados no centro do eixo transversal.
   - ***stretch***: Componentes esticados para preencher o contentor ao longo do eixo transversal (**padrão**).
   - ***baseline***: Componentes alinhados pelas linhas de base do texto.

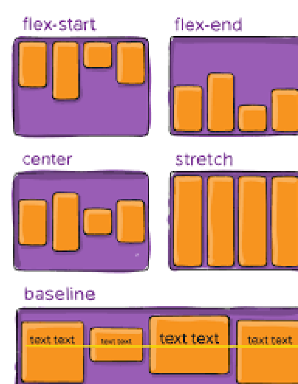



Figura 5.4: Propriedade *alignItems*

No início do estágio foi recomendado um guia do *Flexbox* para consultar o funcionamento de cada propriedade e os seus valores. Este guia pode ser consultado em [9 (Coyier, 2024)].

### 5.1.3 Navegação

Para gerir a navegação de uma aplicação pode ser utilizada a biblioteca ***React Navigation***, uma das mais populares e introduzida em [19 (Platforms, 2024)]. Com esta biblioteca, é possível implementar diferentes tipos de navegação, como navegadores em pilha (***stack***) que permitem transições lineares entre ecrãs, navegadores em abas (***tabs***) que facilitam a mudança entre diferentes secções da aplicação, e navegadores em gaveta (***drawer***) que fornecem um menu lateral com as diferentes secções. É possível ainda misturar as diferentes navegações, implementando uma navegação mais geral (***nested***). Também permite a **passagem de parâmetros** entre ecrãs, a **gestão de estados** de navegação e a customização das **animações de transição**.

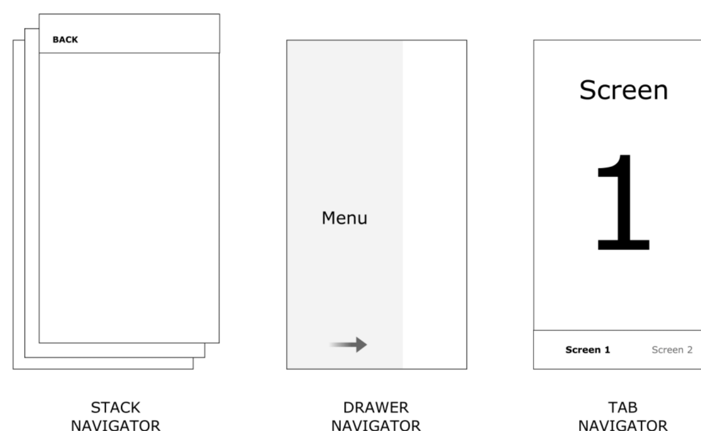



Figura 5.5: Diferentes tipos de navegação

#### *Stack Navigator*

Este tipo de navegação permite empilhar ecrãs uns nos outros e navegar entres eles numa ordem linear. Cada ecrã adicionado à pilha fica posicionado no topo desta, mas, quando o utilizador retrocede, este ecrã no topo é removido e é renderizado o ecrã que estava posicionado em segundo lugar. Esta navegação pode ser utilizada em fluxos de navegação sequenciais como um processo de compra (*checkout*), formulários com várias categorias/etapas ou numa situação geral onde o utilizador deverá seguir um fluxo linear sem a possibilidade de ecrãs alternativos a meio da interação.

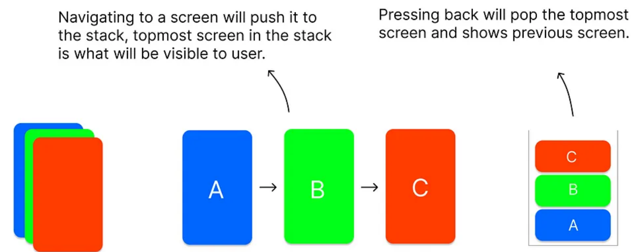



Figura 5.6: Navegação em pilha (*Stack Navigation*)

Esta navegação é possível recorrendo à inicialização do navegador *stack* utilizando a constante ***Stack***. Esta constante implementa o tipo de navegação pretendida pelos ecrãs criados.

Listing 5.15: *Stack Navigator*

```
const Stack = createNativeStackNavigator();

const App = () => {
  return (
    <NavigationContainer>
      <Stack.Navigator
        initialRouteName="Home"
        screenOptions={{
          headerTitle: props => LogoTitle(props),
        }}>
        <Stack.Screen name="Home" component={Home} />
        ...
      </Stack.Navigator>
    </NavigationContainer>
  );
};
```

### *Tab Navigator*

Este tipo de navegação visa criar uma navegação baseada em abas, onde o utilizador pode alternar rapidamente entre as secções ou funcionalidades principais da aplicação. De notar que cada aba (*tab*) corresponde a um ecrã distinto. Pode ser utilizada para uma organização mais eficiente e intuitiva da estrutura principal de uma aplicação, como acontece com aplicações de redes sociais que possuem uma barra inferior com os ecrãs mais utilizados (*feed*, procurar, perfil, etc.).

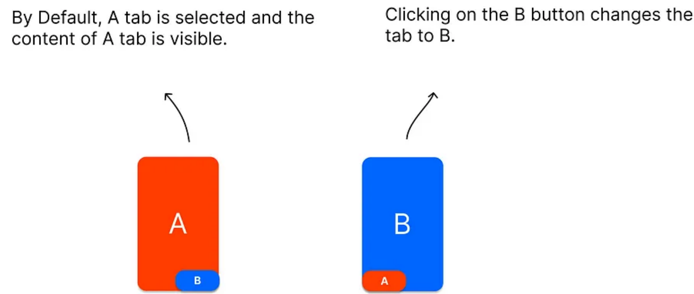



Figura 5.7: Navegação em abas (*Tab Navigation*)

Esta navegação é possível recorrendo à inicialização do navegador *tab* utilizando a constante ***Tab***. Esta constante implementa o tipo de navegação pretendida pelos ecrãs criados. O código para criar esta navegação é idêntico ao da navegação *Stack*, onde apenas muda a constante e o nome do componente de navegação e de ecrã.

Listing 5.16: *Bottom Tab Navigator*

```
const Tab = createBottomTabNavigator();

const AppTabNavigation = () => {
  return (
    <NavigationContainer>
      <Tab.Navigator (...) >
        <Tab.Screen name={'Home'} component={Home} />
        (...)
      </Tab.Navigator>
    </NavigationContainer>
  );
};
```

### *Drawer Navigator*

Este tipo de navegação permite criar uma navegação baseada num menu lateral que pode ser aberto ou fechado (normalmente denominado por **menu hambúrguer** pela sua forma semelhante ≡). Este menu, ou gaveta (*drawer*), contém as ligações para as diferentes secções da aplicação. Pode ser utilizada em aplicações que possuem um espaço de interface reduzido, que possuem várias categorias a destacar, ou que requerem um menu de navegação acessível em todos os momentos, mas sem estar constantemente visível.

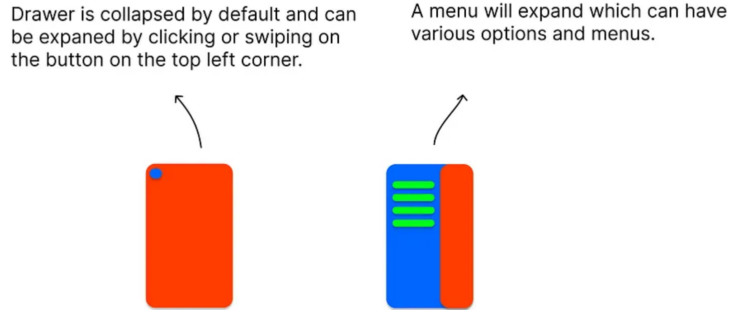



Figura 5.8: Navegação em gaveta (*Drawer Navigation*)

Esta navegação é possível recorrendo à inicialização do navegador *drawer* utilizando a constante ***Drawer***. Esta constante implementa o tipo de navegação pretendida pelos ecrãs criados. O código para criar esta navegação é idêntico ao da navegação *Stack* e *Tab*, onde apenas muda a constante e o nome do componente de navegação e de ecrã.

Listing 5.17: *Drawer Navigator*

```
const Drawer = createDrawerNavigator();

const AppDrawerNavigation = () => {
  return (
    <NavigationContainer>
      <Drawer.Navigator (...) >
        <Drawer.Screen name={'Home'} component={Home} />
        (...)
      </Drawer.Navigator>
    </NavigationContainer>
  );
};
```

### 5.1.4 Flatlist

O componente ***Flatlist*** é um dos fundamentais no desenvolvimento em *React Native*, tendo como objetivo renderizar listas de dados de forma organizada e eficiente. É ideal para renderizar conjuntos de dados de dimensão substancial, mostrando-se superior quando comparado com componentes nativos como ***ListView*** ou ***ScrollView***. Este componente utiliza uma técnica de desempenho chamada de "virtualização", que renderiza apenas os componentes visíveis no ecrã, economizando também memória quando é necessário lidar com listas grandes.

Este componente é apresentado em [18 (Platforms, 2024)].

Para implementar este componente são necessárias algumas propriedades, das quais duas a referir: um ***array data*** que possui os dados a listar e uma **função *renderItem*** que define como renderizar cada componente (através duma propriedade *item* que representa o componente atual a ser renderizado). Abaixo é apresentado um exemplo prático.

Listing 5.18: *Flatlist*

```
const App = () => {
  const DATA = [
    { id: '1', title: 'Item 1' },
    { id: '2', title: 'Item 2' },
    { id: '3', title: 'Item 3' },
  ];
  const renderItem = ({ item }) => <Text>{item.title}</Text;

  return (
    <FlatList
      data={DATA}
      renderItem={renderItem}
      keyExtractor={(item) => item.id}
    />
  );
};
```

## 5.2 *Design System*

No desenvolvimento de aplicações é essencial destacar a importância de uma consistência visual e funcional de forma a garantir uma experiência de utilizador intuitiva. Para conseguir esta consistência é utilizado um ***design system*** (*DS*) que funciona como uma biblioteca de componentes, padrões de *design* e valores pré-definidos que podem ser reutilizados em diferentes contextos. Um *design system* tem o potencial de acelerar bastante tanto a manutenção de código como a sua implementação, garantindo que todos as novas funcionalidades na aplicação são desenvolvidas de forma coesa com o resto da aplicação.

Esta biblioteca vai sendo desenvolvida à medida que a aplicação cresce, onde todos os novos componentes são implementados nesta e mais tarde utilizados na aplicação. Para tal, é necessária uma aplicação de teste do *DS* que tenha atualizações e novas versões. Na *app*

é também atualizada, a cada atualização do *DS*, a versão da biblioteca. O *design system*, ao possuir esta aplicação de teste, permite que se descubram erros e sejam corrigidos e testados numa só plataforma. Ao ter uma biblioteca centralizada, como é o caso, permite que todos os erros sejam resolvidos de forma eficiente e geral, resolvendo o problema de um componente para toda a aplicação.

### 5.2.1 Criação e Desenvolvimento de Novos Componentes

Durante o estágio existiram várias tarefas de criação de novos componentes, que mais tarde seriam introduzidos na aplicação de produção. Quando um componente novo é criado, em condições normais, são utilizados cinco ficheiros:

- **Ficheiro do componente** ou **ficheiro principal**, onde é implementado o código funcional do componente. O nome deste ficheiro é **componente.tsx** (*.tsx* representa ficheiros *TypeScript* que contêm *JSX* - *JavaScript XML*);
- **Ficheiro de estilos**, onde são definidos os estilos que serão utilizados no ficheiro principal. Utilizado para simplificar a implementação do componente. O nome deste ficheiro é **componente.styles.ts** (*.ts* representa ficheiros *TypeScript*);
- **Ficheiro de tipos**, onde são definidos os tipos de constantes e *props* utilizados no ficheiro principal. Utilizado para simplificar a implementação do componente. O nome deste ficheiro é **componente.types.ts**;
- **Ecrã exemplo do componente**, onde é utilizado o componente criado como se tratasse da aplicação de produção. Serve para mostrar e exemplificar o seu funcionamento. O nome deste ficheiro é **ComponenteScreen.tsx**;
- **Documentação**, onde é descrito o componente e os seus atributos. É também mostrado um exemplo em código da sua utilização. O nome deste ficheiro é **componente.md**

Desta forma podemos ter um *design system* organizado com o código separado em vários ficheiros. Com isto em mente, durante o estágio, foram criados cinco componentes.

#### Componente *Link Chip*

Este componente é definido como um pequeno botão (*chip*) utilizado para aceder a ligações (*link*). Pode conter um texto, um ícone ou ambos.

O componente possui cinco *props*, onde apenas uma é obrigatória:

- ***text*** **(*String*):** Define o texto do *chip*;
- ***icon*** **(*JSX*):** Define o *icon* do *chip*;
- ***typeCompact*** **(*Boolean*):** Ativar ou desativar o tipo compacto. Quando ativo, o *chip* fica mais pequeno/compacto.
- ***backgroundColor*** **(*String*):** Definir a cor de fundo do *chip*. Apenas podem ser as cores *neu01* (#ffffff), *neu02* (#f6f6f6) ou *neu03* (#ededed), visto que no ficheiro de tipos (*componente.types.ts*), esta *prop* está limitada a estes três valores.
- ***onPress*** **(*Function*):** Define a função *onPress*, ou seja, a função executada quando o utilizador pressiona o componente. **Obrigatório**.

Como só a função *onPress* é que é obrigatória, é feita uma verificação no início da implementação deste componente, onde é confirmado se existe uma das *props text* ou *icon* para ser possível renderizar o *chip*. Caso não exista nenhum, o componente não é renderizado, ficando a garantia que a aplicação não dará erro por falta de propriedades.

Listing 5.19: *Link Chip*

```
const LinkChip = (props: LinkChipProps) => {
  const {text, icon, typeCompact, backgroundColor, onPress, style = {}} = props;
  const contentColor = backgroundColor ? getColor(backgroundColor) : neu11;

  if (!text && !icon) {
    return null;
  }

  return (
    <View
      style={[WRAPPER, ...]}>
      <Touchable onPress={onPress} style={CONTAINER}>
        {text && (
          <Text
            ...>
            {text}
          </Text>
        )}
        {icon &&
          React.cloneElement(icon, {
```

```
            nopad: true,
            ...
          })}
        </Touchable>
      </View>
    );
  };
```

Este componente é composto por um contentor *View* que inclui um *Touchable* como forma de registar cliques. Dentro deste *Touchable* é então renderizado o conteúdo, ou seja, o texto ou o ícone. É feita uma verificação em cada um destes (text && <Text ...), percebendo se existem e se podem ser renderizados. Esta verificação significa que se a constante atrás de && existir e não for vazia, então o componente que está à frente é renderizado. Caso a constante seja vazia ou não exista, nada é executado.

Podem ser observados vários estilos como *WRAPPER*, *CONTAINER_COMPACT* e *TEXT_PADDING_RIGHT*, implementados no ficheiro ***link-chip.styles.ts*** e utilizados no código principal. Da mesma forma, o tipo de *props* utilizado (***LinkChipProps***) é implementado no ficheiro ***link-chip.types.ts*** e utilizado no início do código apresentado acima. O componente é posteriormente utilizado da seguinte forma:

Listing 5.20: Utilização do *Link Chip*

```
<LinkChip
    text={'Far far away'}
    icon={true}
    typeCompact={true}
    backgroundColor={'neu01'}
    onPress={() => console.log('Pressed chip')}
/>
```

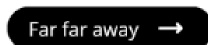



Figura 5.9: Componente *Link Chip*

No ecrã exemplo (***LinkChipScreen.tsx***) é então implementado o componente com *toggles* para permitir que o seu comportamento seja alterado, testando diferentes valores para as propriedades do componente:

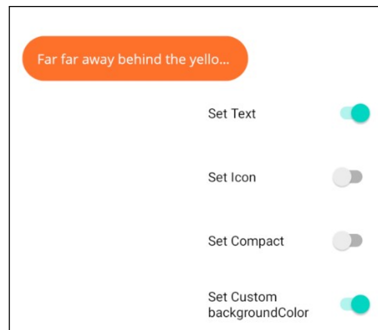



Figura 5.10: Ecrã exemplo do componente *Link Chip*

### Componente *Link Chip Container*

Este componente agrega múltiplos componentes *Link Chip* (**5.2.1 Componente *Link Chip***) em forma de grelha ou *slider*.

Existem duas propriedades a ter em atenção neste componente:

- ***data*** **(*Array*):** Lista de componentes *Link Chips* a representar. Contém as propriedades de cada *Link Chip*. **Obrigatório**.
- ***typeSlider*** **(*Boolean*):** Ativar ou desativar o modo de *slider*. Quando desativo, será definido o modo de grelha.

Caso se queria representar a lista de componentes num *slider*, é utilizado um contentor do tipo *Flatlist* como descrito em **5.1.4 *Flatlist***, economizando memória com a técnica de virtualização caso existam *itens* da lista que não caibam no ecrã. Caso se queria representar a lista em modo de grelha, então é utilizado um contentor *View* comum.

Listing 5.21: *Link Chip Container*

```
<>
  {typeSlider ? (
    <FlatList
      data={data}
      horizontal={typeSlider}
      showsHorizontalScrollIndicator={false}
      keyExtractor={(_item, index) => `link-chip-${index}`}
      renderItem={({item, index}) => {
        return (
          <LinkChip {...item} key={`link-chip-${index}`} style={CHIP} />
        );
      }}
```

```
      style={SLIDER_CONTAINER}
      contentContainerStyle={style}
    />
  ) : (
    <View style={[GRID_CONTAINER, style]}>
      {data.map((item, index) => {
        return (
          <LinkChip
            {...item}
            key={`link-chip-${index}`}
            style={CHIP_GRID}
          />
        );
      })}
    </View>
  )}
</>
```

O componente é posteriormente utilizado no ecrã exemplo como demonstrado abaixo. Existem duas utilizações do *Link Chip Container*, onde a única diferença é no valor da propriedade *typeSlider*.

Listing 5.22: Utilização do *Link Chip Container*

```
const data = [
{
  text: "Hello world, I'm a Link Chip",
  icon: <Bas020 />,
  typeCompact: true,
  backgroundColor: add01,
  onPress: () => {
    console.log('Pressed chip 1');
  },
},
{
  text: "Hello",
  icon: <Bas020 />,
  typeCompact: true,
  backgroundColor: add02,
  onPress: () => {
```

```
      console.log('Pressed chip 2');
    },
  },
  ...
];
const props = {data};

return (
  <Layout>
    <Text style={styles.title}>Link Chip Container Grid</Text>
    <LinkChipContainer {...props} style={styles.linkChip} />
    <Text style={styles.title}>Link Chip Container Slider</Text>
    <LinkChipContainer {...props} typeSlider={true} style={styles.linkChip} />
  </Layout>
);
```

Com esta implementação obtém-se o ecrã com o seguinte aspeto:

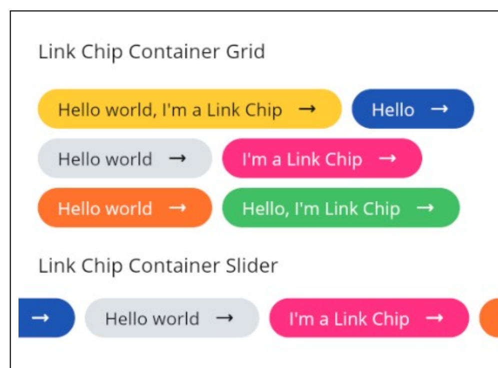



Figura 5.11: Ecrã exemplo do componente *Link Chip Container*

### Componente *Radio Button Group*

Este componente é um grupo de botões de opção (botões *radio*) para selecionar uma de várias opções num grupo. O componente pode ser utilizado com apenas um botão que devolve verdadeiro caso esteja selecionado e falso caso contrário, ou vários botões onde é devolvido o nome do botão selecionado.

O componente possui apenas duas propriedades a descrever:

- ***buttons*** **(*Array*):** Os botões que serão utilizados no grupo. **Obrigatório.**

- ***direction*** **(*String*):** Define a direção em que os botões serão apresentados. *row* significa

que os botões serão apresentados horizontalmente e *column* significa que os botões serão apresentados verticalmente. **Obrigatório**.

O *array* de botões contém, para cada um, as propriedades necessárias para a sua criação e funcionamento:

- ***onPress* (*Function*):** Define a função *onPress*, ou seja, a função executada quando o utilizador pressiona o componente. **Obrigatório**.
- ***isSelected* (*Boolean*):** Define se o botão está selecionado ou não.
- ***children* (*Object*):** Elementos filhos do componente, como texto.

Para a implementação deste *Radio Button Group*, foi necessário utilizar a função ***map*** onde lhe é passado um *item* e um *index* simbolizando o elemento atual a ser renderizado. A partir destes dois parâmetros é possível criar o grupo tendo em consideração qual dos botões é que se encontra selecionado.

Listing 5.23: *Radio Button Group*

```
<View style={[GROUP_CONTAINER, {flexDirection: direction}]}>
  {buttons.map((item, index) => (
    <View
      key={index}
      ...>
      <RadioButton
        onPress={() => handleRadioButtonPress(item.value)}
        children={React.cloneElement(...)}
        isSelected={selectedButton === item.value}
        valid={valid}
      />
    </View>
  ))}
</View>
```

Portanto, pode ser observado que para a utilização deste componente no ecrã exemplo, será necessário providenciar uma lista com botões caso seja pretendido ter mais do que um botão. No ecrã exemplo são criados dois casos, um com apenas um botão e outro com uma lista de três botões.

Listing 5.24: Utilziação do *Radio Button Group*

```
const singleButton = [{value: 'button0', label: <Text>Single button</Text>}];
const multipleButton = [
    {value: 'button1', label: <Text>Button 1</Text>},
    ...
];

return (
    <Layout>
        <Text style={styles.title}>Single Button</Text>
        <View style={styles.singleButtonContainer}>
            <View>
                <RadioButtonGroup
                    buttons={singleButton}
                    direction="row"
                    ref={singleButtonRef}
                />
            </View>
            ...
        </View>
        <Text style={styles.title}>Multiple Buttons</Text>
        <View style={styles.multipleButtonContainer}>
            <View style={styles.multipleButtons}>
                <RadioButtonGroup
                    buttons={multipleButton}
                    ...
                />
            </View>
            ...
        </View>
    </Layout>
);
```

Este código exemplo produz o seguinte resultado:

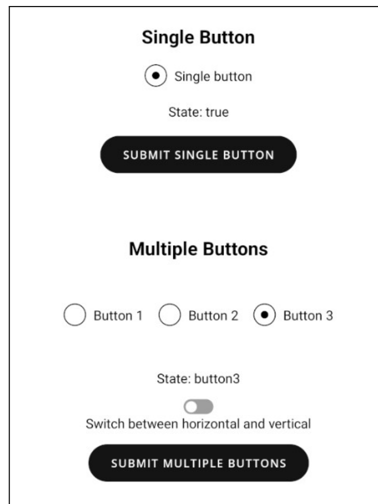



Figura 5.12: Ecrã exemplo do componente *Radio Button Group*

### Componente *Info Blocks*

Este componente agrega vários cartões informativos e interativos com um ícone, um texto e um valor. Estes cartões não só são informativos como detetam cliques e, portanto, podem conter uma ligação para se adequar à funcionalidade e contexto pretendido. O contentor (componente pai) também pode ser modificado para ser enquadrado em diversos cenários.

O componente pai possui três propriedades:

- ***items*** **(*Array*):** Conjunto de cartões a apresentar no contentor. **Obrigatório**.
- ***title*** **(*String*):** O título do contentor. Indica o assunto dos cartões.
- ***background*** **(*String*):** Cor de fundo do contentor. **Obrigatório**.

Cada cartão do *array*, que serve como propriedade do componente principal, possui cinco propriedades necessárias para a sua criação e funcionamento:

- ***text*** **(*String*):** A descrição do cartão, indica o assunto deste. **Obrigatório**.
- ***icon*** **(*Element*):** O ícone do cartão. **Obrigatório**.
- ***value*** **(*Object*):** O valor do cartão, possui um texto e um ícone de informação. **Obrigatório**.
- ***background*** **(*String*):** Cor de fundo do cartão. **Obrigatório**.
- ***onPress*** **(*Function*):** A ação a realizar quando o cartão é premido. **Obrigatório**.

Este componente possui um *design* dinâmico, onde consoante o número de cartões, o contentor ajusta-se e organiza o seu conteúdo de forma diferente para acomodar toda a sua informação. É utilizado também, nesta implementação, a função *map* para renderizar cada cartão e várias condições para ser possível ajustar todos os cartões ao seu conteúdo e o contentor aos cartões.

Listing 5.25: Info Blocks

```
...
    {items.map((item, index) => (
      <TouchableOpacity
        key={index}
        onPress={() => item.onPress()}
        style={[CARD, {backgroundColor: item.background}]}>
        {item.icon && <View style={CARD_ICON}>{item.icon}</View>}
        <View
          style={[...]}>
          {item.text && (...)}
          {item.value.text && (
            ...
            {item.value.hasInfoIcon && (
                ...
            )}
          )}
        </View>
      </TouchableOpacity>
    ))}
...
```

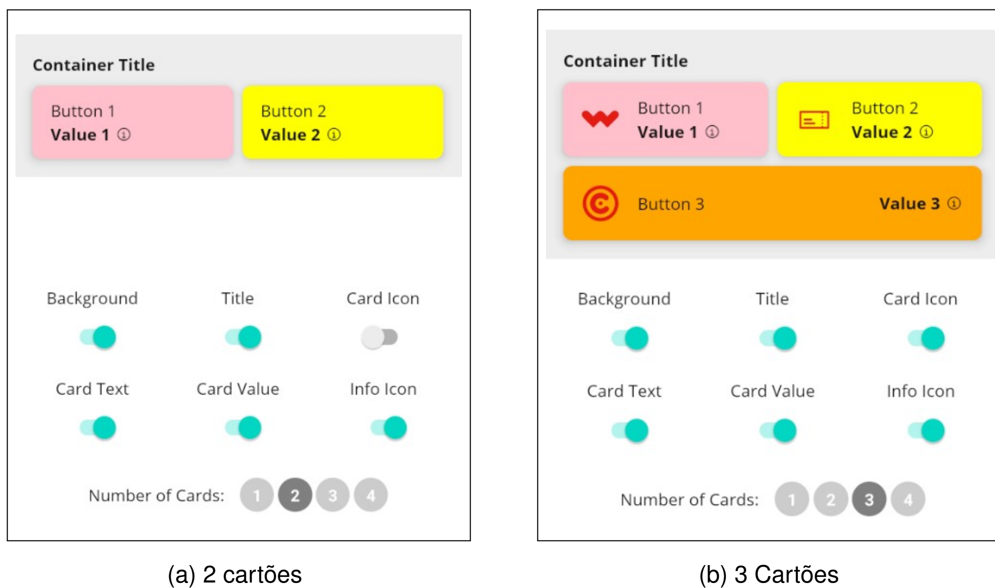
Este componente foi então utilizado no ecrã exemplo com várias opções de customização para ser possível observar diferentes comportamentos. Para tal, foi criado um conjunto de cartões a renderizar (no código seguinte apenas é mostrado um cartão).

Listing 5.26: Utilização do *Info Blocks*

```
let propItems = [
    { background: 'pink',
      ...(hasCardIcon && {
        icon: <Log009 nopad={true} size="S" color={main} />,
      }),
      ...(hasCardText && {text: 'Button 1'}),
```

```
        value: {
          ...(hasCardValue && {text: 'Value 1'}),
          hasInfoIcon: hasInfoIcon,
          onPressInfo: () => console.log('Press info 1'),
        },
        onPress: () => console.log('Press button 1'), },
      ...
  ];
```

Com estes cartões, são então apresentados dois cenários a partir do ecrã exemplo, o primeiro onde existem dois cartões com todo o conteúdo menos o ícone, e o segundo com três cartões com todo o conteúdo possível.




(a) 2 cartões (b) 3 Cartões

Figura 5.13: Componente *Info Blocks*

### Componente *Badge Flags*

Este componente é implementado visando fornecer ao utilizador informações rápidas sobre um produto. A bandeira será normalmente posicionada por cima de um cartão de produto (por exemplo, produtos relacionados), com diversas cores e textos consoante o seu propósito.

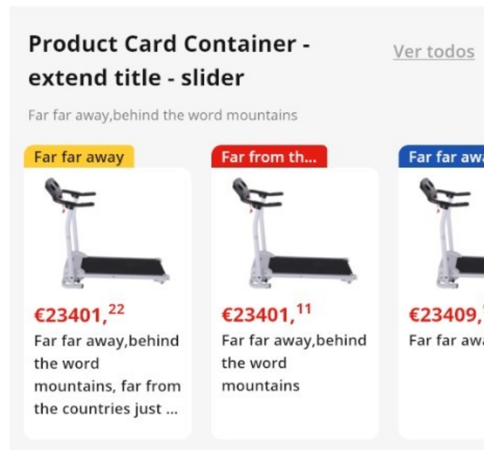



Figura 5.14: *Badge Flags* no componente *Product Card*

O componente, tal como mencionado acima, possui apenas duas propriedades:

- ***text*** **(*String*):** Define o texto da bandeira. **Obrigatório**.
- ***background*** **(*String*):** Define a cor de fundo da bandeira. **Obrigatório.**

Com a sua simplicidade, o componente é meramente implementado recorrendo a um contentor *View* e um texto *Text*.

Listing 5.27: Badge Flags

```
<View style={[CONTAINER, {backgroundColor: background}]}>
    <Text
        preset={3}
        size="S2"
        numberOfLines={1}
        weight="semiBold"
        style={{color: getColor(background)}}>
        {text}
    </Text>
</View>
```

Desta forma, a sua utilização passa por uma única linha de código com:

Listing 5.28: Utilização do *Badge Flags*

```
<BadgeFlag text="Worten Recomenda" background={Colors.m01}/>
```

Produzindo o seguinte resultado:

Worten Recomenda

Figura 5.15: Componente *Badge Flag*

### Componente *Action List Item*

Este componente é utilizado para listar um item servindo o propósito de um botão com uma descrição, um ícone ou uma imagem. Pode servir como ligação para uma nova página ou uma *drawer*.

Existem várias customizações possíveis sobre o componente, tal como se pode observar pelas propriedades que este recebe:

- ***label*** **(*String*):** Descrição do *item*. **Obrigatório**.
- ***icon*** **(*JSX Element*):** Ícone à esquerda. Representativo do *item*.
- ***image*** **(*JSX Element*):** Substituto do ícone à esquerda. Representativo do item.
- ***hideActionIcon*** **(*Boolean*):** Mostrar ou esconder o ícone de seta à direita.
- ***density*** **(*String*):** Define o tamanho dos ícones. Pode ser “*default*” ou “*compact*”.
- ***scheme*** **(*String*):** Define o esquema de cores do *item*. Pode ser “*default*”, “*gray*” ou “*lighter*”.
- ***status*** **(*String*):** Define o estado do *item* e, por consequência, as cores do *item* são alteradas. Pode ser “*default*” ou “*active*”.
- ***dividers*** **(*String*):** Divisores para separar os *items*. Para cada um, existe a opção de não haver divisores (“*none*”), de estar um posicionado na parte superior (“*top*”) ou dois posicionados na parte superior e inferior (“*topBottom*”).
- ***nested*** **(*Boolean*):** Aumenta o espaço (*padding*) entre o limite esquerdo e o conteúdo do *item*.
- ***horizontalSpacing*** **(*String*):** Define o espaço horizontal. Pode ser “*none*”, “*narrow*” ou “*spacious*”.
- ***verticalSpacing*** **(*String*):** Define o espaço vertical. Pode ser “*none*”, “*narrow*” ou “*spacious*”.
- ***onPress*** **(*Function*):** Define a função a executar quando o item é premido. **Obrigatório**

Com um número elevado de propriedades a definir, o código implementado requer várias condições e verificações de forma a criar o componente decentemente para ser utilizado na aplicação. De notar que a linha $\{image\,?\,(...)\ :\ icon?.icon\,?\,(...)\ :\ (<></>)\}$ é traduzida em “Se houver image então ..., caso só haja icon então ..., no caso de não haver nenhum então $<></>$ (vazio)”.

Listing 5.29: Action List Item

```
<>
  <Touchable
    style={[...]}
    onPress={onPress}>
    <View style={CONTENT_CONTAINER}>
      <View style={ICON}>
        {image ? (...) : icon?.icon ? (...) : (<></>)}
      </View>
      <View style={LABEL}>
        <Text weight="bold" size="M" numberOfLines={2}>
          (...)
        </Text>
      </View>
    </View>
    {!hideActionIcon && (...)}
  </Touchable>
  {dividers === 'topBottom' && (
    <View style={{backgroundColor: dividerColor}} />
  )}
</>
```

De forma a contemplar estas possibilidades todas, no ecrã exemplo foram implementados vários botões de forma a alterar o comportamento do componente:

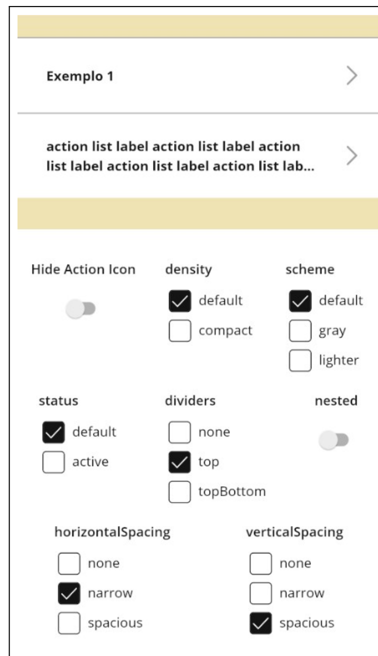



Figura 5.16: Ecrã exemplo do componente *Action List Item*

Foi realizado um pequeno exemplo da utilização deste componente com apenas um *item*:

Listing 5.30: Utilização do *Action List Item*

```
<ActionListItem
    icon={<Icons.Bra009 />}
    label="action list label action list label action list label action list label"
    hideActionIcon={false}
    density="default"
    scheme="default"
    status="default"
    dividers="topBottom"
    nested={false}
    horizontalSpacing="narrow"
    verticalSpacing="narrow"
    onPress={handleOnPress}
/>
```

Com esta utilização do componente *Action List Item* foi observado o seguinte resultado:

Figura 5.17: Componente *Action List Item*

### 5.2.2 Manutenção de Componentes

À medida que os componentes eram criados, outros necessitavam de atualização, podendo ser esta realizada num contexto de reformulação da arquitetura de código, alteração de estilos de forma a modificar a aparência ou alteração do código para uma mudança de comportamento/funcionalidade. Esta atualização serve para corrigir *bugs* e melhorar componentes para uma melhor interface e experiência de utilizador.

No início do estágio foi feita uma refatoração dos estilos e tipos dos componentes visando uma melhoria de legibilidade de código. Esta refatoração serviu também como uma integração no *Design System* antes da passagem para a criação de componentes, facilitando a aprendizagem de *React Native* e o comportamento e arquitetura dos componentes.

Como exemplo desta refatoração, abaixo é apresentado um pequeno exemplo da alteração feita a todos os ficheiros de estilos dos componentes. Antes da mudança, os estilos eram armazenados numa *StyleSheet* como descrito em **5.1.2 Estilos com *StyleSheet***.

Listing 5.31: Antes da refatoração dos estilos

```
const styles = StyleSheet.create({
  image: {
    overflow: 'hidden',
  },
  caption_text: {
    marginTop: space_XXS,
    color: neu06,
  },
});
export {styles};
```

Este código requeria uma utilização do estilo da seguinte forma:

Listing 5.32: Utilização dos estilos antes da refatoração

```
<View style={styles.image}>...</View>
```

Foi então feita uma reformulação deste código para passarem a existir dois estilos em que ambos são exportados individualmente. Além disso, passou-se a especificar a que tipo de estilo é que cada um pertencia, como ***ViewStyle*** ou ***TextStyle***.

Listing 5.33: Depois da refatoração de estilos

```
export const IMAGE: ViewStyle = {
  overflow: 'hidden',
};

export const CAPTION_TEXT: TextStyle = {
  marginTop: space_XXS,
  color: neu06,
};
```

Desta forma, a legibilidade dos estilos aumenta e facilita-se a sua utilização, que passa a ser a seguinte:

Listing 5.34: Utilização dos estilos depois da refatoração

```
<View style={IMAGE}>...</View>
```

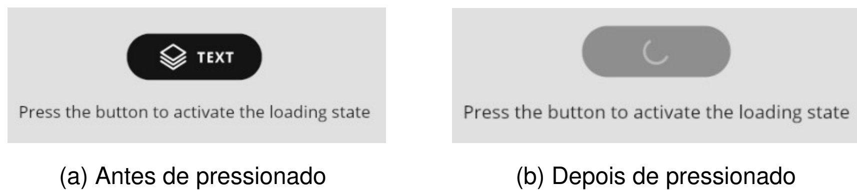
Para além da implementação e refatoração de estilos, foi alterado também código nos ficheiros principais dos componentes, onde foram corrigidos dois como forma de inicialização aos componentes.

- **Seta *Go Back* no componente *Bottom Drawer*:** Ao alterar-se o comportamento do componente, em *Android* a seta com a função de retroceder ficou perto do canto da *drawer*, causando inconsistência com o resto do *design* da aplicação.
- **Limite mínimo de *itens* no componente *Related Cards*:** Este componente é utilizado para mostrar produtos em cartões, inicialmente utilizado principalmente no *dashboard* (página inicial). No entanto, este componente possuía um limite mínimo de três cartões a apresentar e, quando foi desenvolvida uma nova área cliente na aplicação, este componente foi utilizado para apresentar os favoritos. Portanto, foi alterado o limite mínimo de três para um de forma a que o componente conseguisse apresentar menos que três favoritos.

Por último, após a implementação de estilos e correção de componentes, ainda existiu a atualização de um componente devido à introdução de ***Lotties*** na aplicação. *Lotties* são animações vetoriais utilizadas em *websites* e aplicações móveis para melhorar a experiência de utilizador. São gerados em formato *JSON*, permitindo escalabilidade sem perda de qualidade, otimizando também a *performance* e o *design* da interface.

Foi então implementado um *lottie* com o objetivo de informar o utilizador de que algo conti-

nua a carregar e ainda não acabou. Este ***loading lottie*** apresentado abaixo continua a rodar infinitamente até que o processo em questão seja finalizado. Desta forma, a aplicação e os processos realizados de forma oculta ao utilizador são mais intuitivos, fornecendo informação sobre o seu estado.




(a) Antes de pressionado (b) Depois de pressionado

Figura 5.18: Botão com *lottie* de carregamento (*loading state*)

Este *loading lottie* é implementado com o apoio do componente ***LottieView***, onde é necessário destacar as suas três propriedades:

- ***autoPlay* (*Boolean*):** Define se o *lottie* começa automaticamente assim que é carregado.
- ***source* (*JSON*):** Ficheiro *JSON* com o conteúdo para desenhar e animar o *lottie*.
- ***loop* (*Boolean*):** Define se a animação do *lottie* é reproduzida em *loop*, ou seja, se é reproduzida mais que uma vez.

Portanto, como exemplo de implementação, tem-se o seguinte código:

Listing 5.35: *Loading lottie*

```
<LottieView
    ref={animation}
    autoPlay
    source={LOTTIE_SOURCE[color]}
    loop
    style={sizesNoPad[size]}
/>
```

### 5.2.3 Iconografia

Para além da criação de componentes, foram criados alguns *icons* em falta no *Design System*. A *Worten* possui uma biblioteca gráfica *online* com todos os estilos, icons, temas e outros elementos de *design* para que a implementação seja facilitada e direta.

Ao consultar o *link* que dá acesso à biblioteca foi confirmado que existiam vários *icons* em falta no *DS* e outros que possuíam um nome diferente para o *website* e para a *app*. Foi

então realizada uma remodelação desta, criando os *icons* necessários e editando os que precisam de remodelação. Desta forma, ambas as plataformas da *Worten* passaram a possuir os mesmo elementos de *design*, permitindo uma implementação partilhada de componentes entre um e outro.

Na imagem seguinte está apresentada uma parte da biblioteca como exemplo. Para além dos *icons base* que estão presentes, existem ainda mais quatro categorias, sendo elas *brand icons* que servem como uma extensão da identidade da marca, *special icons* que se destinam a representar serviços específicos ou a servir fins ilustrativos, *logo icons* que oferecem uma coleção de ícones elaborados por empresas, e por fim, *Worten icons* que desempenham um papel crucial sendo que são a personificação visual da identidade da marca.

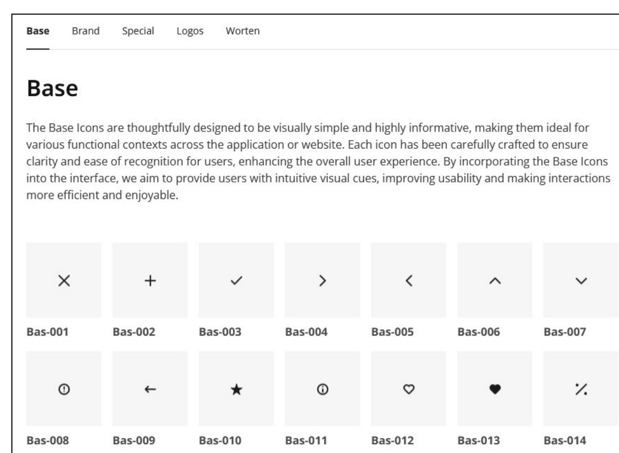



Figura 5.19: Biblioteca de *design*

Foram então realizadas alterações na pasta de *icons* do *DS* com base na biblioteca de forma a que este ficasse atualizada e em sincronia com o *website*. Para tal, foram comparados os *icons* entre a biblioteca e o *DS* e verificados quais os que estavam em falta e os que necessitavam de remodelação ou reorganização.

Para a criação de um *icon* é apenas necessário um ficheiro, onde são fornecidas as coordenadas para que se possa delinear o conteúdo e o tamanho original do *icon*. Como propriedades, este ficheiro possui:

- ***size*** **(*String*):** Tamanho do *icon* com base em medidas previamente estipuladas.
- ***color*** **(*String*):** Cor do *icon*.
- ***nopad*** **(*Boolean*):** Se esta propriedade estiver ativa, o *icon* deixa de possuir espaço (*padding*) à sua volta.
- ***style*** **(*Object*):** Qualquer outro estilo que necessário adicionar.

Listing 5.36: Ícone *Bra033*

```
export const Bra033 = (props: IconTypes) => {
  const {size = 'M', color = neu11, nopad, style} = props;

  return (
    <View
      style={style ? [getStyles(size, nopad), style] : getStyles(size, nopad)}>
      <Svg.Svg viewBox="0 0 40 40">
        <Svg.Path
          d="M27.276 32a3.038 3.038 0 01-3.034-3.035 3.037 3.037 0 013.034-3.034
              3.038 3.038 0 013.035 3.034A3.038 3.038 (continua...)"
          fill={color}
          fillRule="nonzero"
        />
      </Svg.Svg>
    </View>
  );
};
```

Este código produz o seguinte *icon*:

Figura 5.20: *Icon Brand 033*

Assim, utilizando um *link* do *Figma* que continha os *SVG*s de todos os *icons*, foram criados os que estavam em falta na *app* utilizando uma ferramenta de "transformação" de *SVG* para código de *React Native* (*.tsx*) - [17 (Kumar, 2019)]. ***Scalable Vector Graphic*** (*SVG*) é uma linguagem *XML* para descrever de forma vetorial desenhos e gráficos bidimensionais como o apresentado na figura **5.20 *Icon Brand 033***.

Para além desta reestruturação da biblioteca de *icons* na *app*, mais tarde foram ainda criados mais dois com o objetivo do desenvolvimento da opção multilíngua. Foram então implementados vários *icons* em forma de bandeiras de Portugal, Reino Unido, Espanha, França e Alemanha. A implementação destas foi semelhante à implementação do *icon Bra033* apresentada anteriormente.

(a) Portugal (b) Reino Unido (c) Espanha (d) França (e) Alemanha

Figura 5.21: *Icons* dos idiomas suportados

### 5.2.4 Ecrãs de Componentes

Tal como a manutenção dos componentes que preenchem a *app* é importante, também a criação e manutenção dos ecrãs de exemplo é. Estes ecrãs, vistos na aplicação do *Design System*, servem para demonstrar os possíveis comportamentos de um componente, que pode diferir consoante o contexto. Este ecrã exemplo é referido no início do capítulo **5.2.1 Criação e Desenvolvimento de Novos Componentes**.

No início do estágio, para além da introdução aos estilos e aos componentes, existiu também uma introdução aos ecrãs que servem de exemplo, sendo então criados os ecrãs em falta correspondentes aos componentes que já existiam. Após esta criação de vários ecrãs, foi feita uma alteração total num dos componentes mais importantes do *Design System*, o ***Button***. Este componente é criado a partir do componente nativo de *React Native* mas adaptado à aplicação. Esta alteração deveu-se à complexidade do ecrã que continha todos os tipos de botão apresentados por categoria, produzindo um ecrã de grande dimensão e pouco intuitivo.

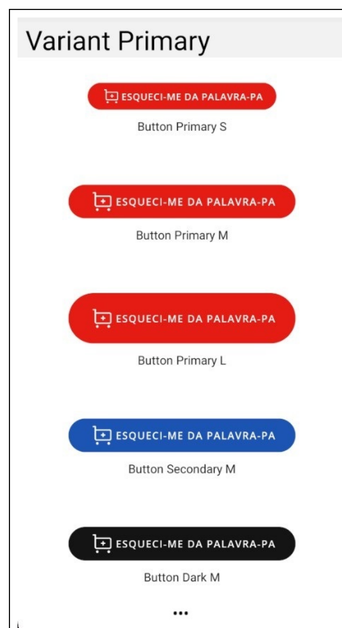



Figura 5.22: Ecrã exemplo antigo do componente *Button*

Para evitar esta arquitetura, foram criados um conjunto de botões de forma a controlar a aparência do botão e ser possível testar vários cenários num só botão.

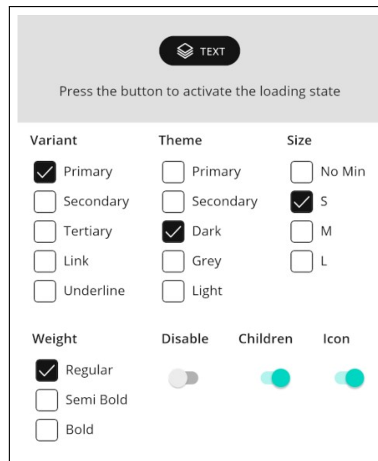



Figura 5.23: Ecrã exemplo atual do componente *Button*

## 5.3 Aplicação de Produção

Após a criação e refatoração do *Design System*, o foco do desenvolvimento passou para a implementação prática desses elementos na aplicação de produção. Enquanto a biblioteca de componentes (*DS*) estabeleceu a base para a consistência visual e funcional, o trabalho na aplicação de produção envolveu a adaptação, integração e expansão desses componentes para atender às necessidades dos utilizadores e do cliente.

Neste capítulo, são detalhadas as diversas tarefas realizadas na aplicação de produção, com ênfase em três áreas principais, sendo elas o desenvolvimento de um jogo promocional, a implementação de uma nova área de cliente e a realização de melhorias de *UI*/*UX*. Além disso, existiu ainda a integração de traduções para uma aplicação multilíngua e a migração de componentes do *Design System* para a aplicação.

### 5.3.1 *Setup* e Introdução à Aplicação

Em primeiro lugar, de forma a que a aplicação fosse executada no emulador corretamente, foi necessário realizar a configuração desta, efetuando ajustes no *IDE* e instalando certas dependências necessárias no projeto.

Depois da configuração, o desenvolvimento da aplicação foi introduzido com a ajuda na realização de uma nova página de pesquisa. Antes, a pesquisa de produtos na aplicação apenas possuía a barra de pesquisa e, caso fosse introduzido algum texto, eram apresentadas as sugestões relativas a este texto.

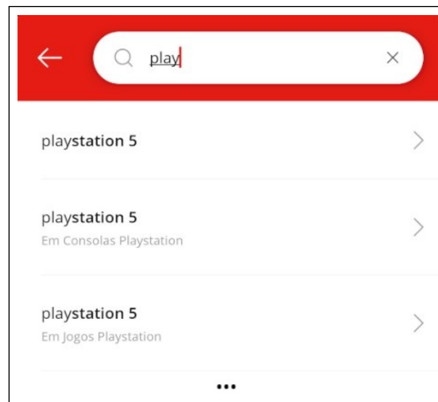
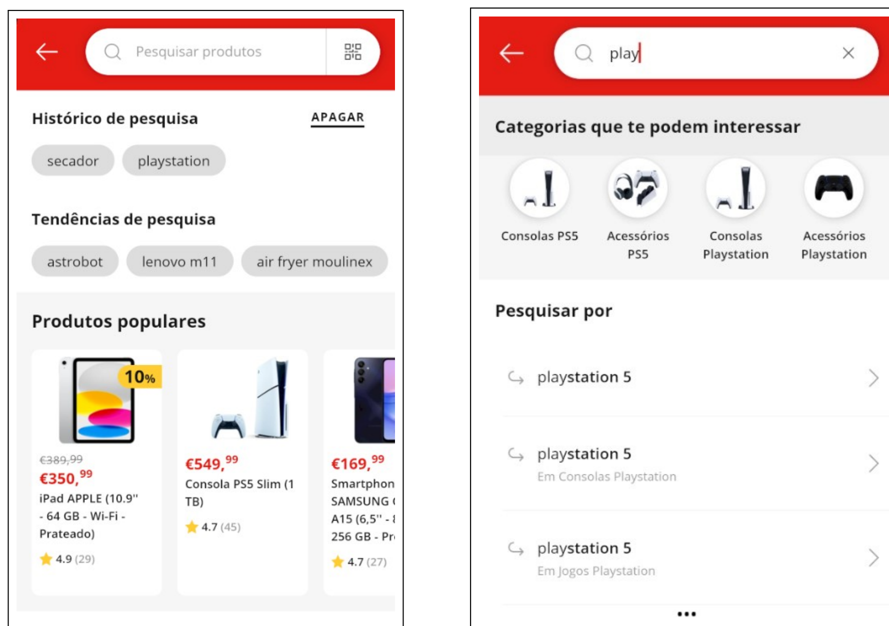



Figura 5.24: Página de pesquisa antiga

A nova pesquisa, por outro lado, possui um novo *design* onde existem duas páginas. A primeira é mostrada quando não é inserido nenhum texto, onde é apresentado um histórico de pesquisa e uma secção de tendências de pesquisa, dando uso ao componente *Link Chip Container* desenvolvido no *Design System* e referido em **5.2.1 Componente *Link Chip Container***, e uma secção de produtos populares apresentados em cartões, dando uso ao componente *Product Card Container* apresentado na figura **5.14 *Badge Flags* no componente *Product Card***. A segunda página é apresentada quando se insere texto na barra de pesquisa, onde o conteúdo é semelhante à página de pesquisa antiga (figura **5.24 Página de pesquisa antiga**) com uma secção adicional de categorias relacionadas com a pesquisa, que poderão interessar ao utilizador.




(a) Sem Texto (b) Com Texto

Figura 5.25: Página de pesquisa atual

### 5.3.2 Jogo

Uma das tarefas mais desafiantes foi o desenvolvimento de um jogo integrado na plataforma. Este jogo teve como objetivo angariar mais utilizadores para a aplicação visto que o jogo só seria possível jogar na *app*. Para esta angariação ser mais rápida foi feita uma campanha em cooperação com a *Ayvens* (antiga *LeasePlan*) em que seria oferecido ao vencedor um *Tesla* durante 3 anos. Desta forma foi tentado equilibrar minimamente a quantidade de utilizadores na aplicação e no *website*.

É então descrito o processo de desenvolvimento do jogo, desde a criação dos componentes da interface até a implementação de animações e ajustes de *UI/UX*. Este projeto incluiu a construção de páginas essenciais (como a inicial ou a de ranking), a criação de elementos interativos e a otimização de elementos visuais para garantir um desempenho fluido. Além disso, foram realizados diversos ajustes e correções para garantir que a experiência de jogo fosse intuitiva e com o mínimo de falhas possível.

Em primeiro lugar, foram implementados certos ecrãs para o funcionamento básico do jogo.

#### Desenvolvimento de Componentes e Páginas

Na primeira fase do desenvolvimento do jogo foram repartidas tarefas, umas relativas ao motor e funcionamento deste e outras às interfaces principais.

Foi então realizada a página principal do jogo, o menu, onde tiveram de ser implementados novos componentes que posteriormente serão utilizados noutros contextos.

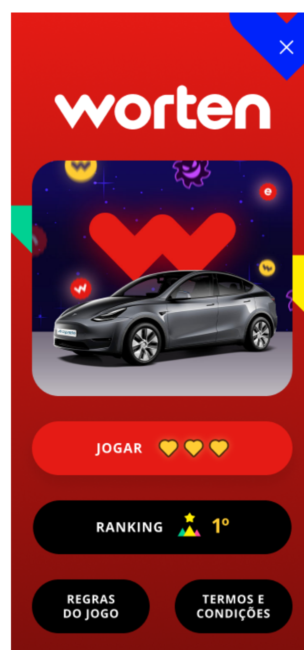



Figura 5.26: Menu do jogo

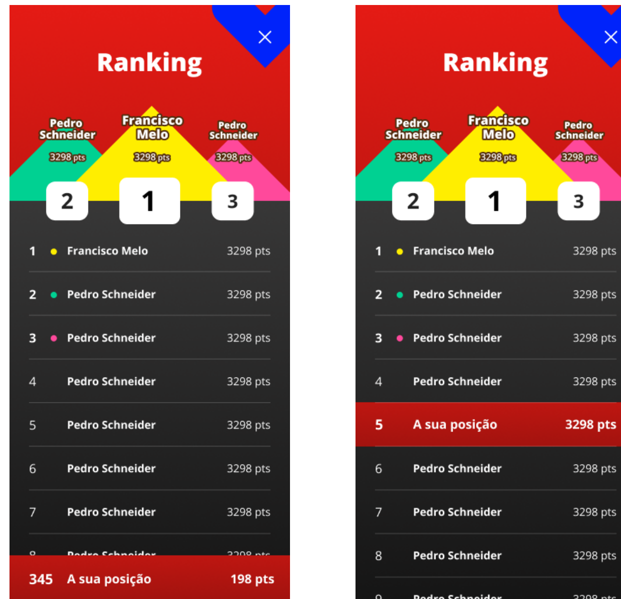
Para realizar esta página foi então implementado um novo botão, específico para o jogo. Este novo componente difere do botão apresentado nas figuras **5.22** e **5.23** em termos de *design*, onde o botão original serve sempre o mesmo propósito. No entanto, este botão do jogo apresenta vários estilos e funcionalidades, podendo ser usado como apresentado na figura acima, como botão de jogar (variante *play*), de *ranking*, ou de regras e de termos e condições. Mas, para além destes, é usado como botão de *ranking* no menu de final de jogo (figura **5.32**) e também como botão "principal" (*primary*) e "secundário" (*light*) do jogo (figura **5.33**).

Estas variantes deste componente existem para permitirem que um botão possa acionar ou gerir certas propriedades do jogo. Na figura **5.26 Menu do jogo**, é possível observar que a variante *play* contém as vidas que o utilizador ainda possui (três por dia). Esta variante tem um processamento distinto dos outros, onde regista as tentativas e vai descontando-as à medida que o utilizador joga até já não restarem nenhuma.

Figura 5.27: Variante *play* do botão do jogo

A outra variante presente é a de *ranking*, onde é visto onde é que o utilizador se encontra na tabela de classificação para poder apresentar um *icon* coerente com a sua posição. Estas duas variantes são variantes que apresentam comportamentos dinâmicos, ao contrário da variante das regras do jogo e dos termos e condições, onde o processamento passa apenas por realizar uma ação quando o botão é premido.

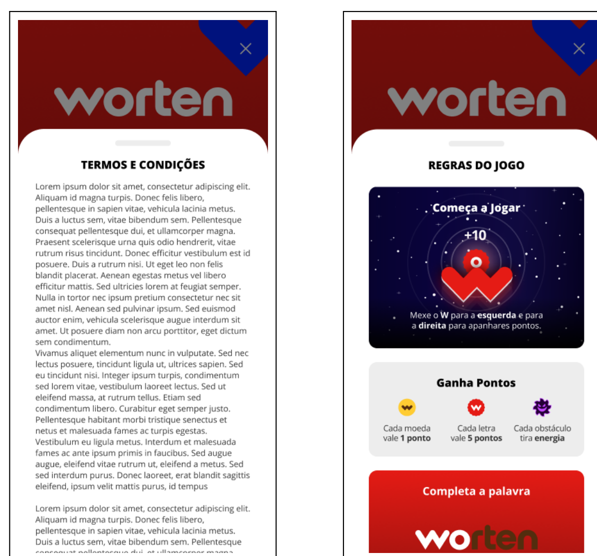
Com este ecrã principal realizado, foi desenvolvido o ecrã da classificação de utilizadores. Este ecrã não apresenta nenhuma interação possível para o utilizador, o seu objetivo é apenas mostrar as posições de todos os utilizadores que pontuaram. Esta classificação é apresentada publicamente apenas até à posição 50 (cinquenta). Caso o utilizador se encontre fora da tabela, apenas ele poderá ver a sua posição, tal como apresentado na figura abaixo (**5.28 Página da classificação do jogo**). Este ecrã foi desenvolvido sem auxílio de componentes exteriores, ao contrário do ecrã principal observado na figura **5.26 Menu do jogo**.




(a) Fora do *top* 50 (b) Dentro do *top* 50

Figura 5.28: Página da classificação do jogo

Na continuação do desenvolvimento dos ecrãs, foram criados os "ecrãs" das regras do jogo e dos termos e condições. Estes "ecrãs" são apresentados em forma de gavetas inferiores (*bottom drawers*), dando a opção de deslizar para baixo para facilitar a saída destas. Na *drawer* dos termos e condições foi apenas inserido texto, enquanto a *drawer* das regras do jogo utiliza várias imagens descrevendo como se joga e, no final desta, um botão da variante *primary* para que o utilizador possa visualizar o *ranking* atual.




(a) Termos e condições (b) Regras

Figura 5.29: *Drawers* informativas do jogo

Foi então desenvolvido o ecrã de jogo ainda sem as mecânicas, começando pelo cabeçalho cuja função é fornecer informações rápidas e relevantes.

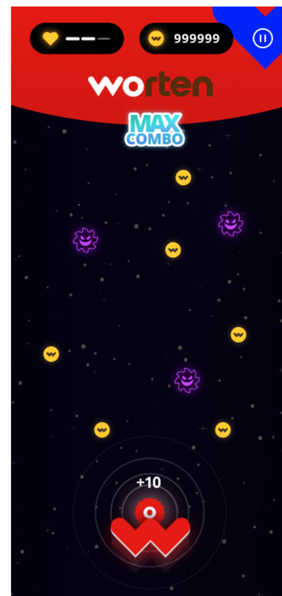



Figura 5.30: Ecrã de jogo

Este cabeçalho informa o utilizador das vidas que tem dentro de cada tentativa, as moedas (pontuação), as letras apanhadas e, por fim, o multiplicador atual. Cada moeda conta como 1 ponto, enquanto as letras da palavra *Worten* contam como 5 pontos. Por outro lado, os obstáculos/inimigos retiram uma vida dentro da tentativa em questão. Para aumentar o multiplicador, é necessário "apanhar" todas as letras da palavra, no entanto, caso o utilizador colida contra um inimigo, o multiplicador é reiniciado. Este pode ir de 1 até 6 (*max combo* representado na figura anterior). Portanto, o objetivo do utilizador é mover horizontalmente o "**W**" (controlador) apanhando as moedas e as letras enquanto se desvia dos inimigos. A cada nível os elementos descem cada vez mais depressa, tornando o jogo mais difícil.

Estes elementos foram, numa primeira fase, implementados como *SVG*s, mas posteriormente alterados para um ficheiro de *TypeScript* que faz uso dos *SVG*s, permitindo um melhor desempenho do jogo ao requerer menos processamento quando os elementos são renderizados (à medida que estas alterações foram feitas, foram também alteradas as imagens do tipo *.png* para *SVG*s pelo mesmo motivo). Para esta transformação foi utilizada a ferramenta apresentada em [17 (Kumar, 2019)] referida anteriormente em **5.2.3 Iconografia**.

(a) Controlador (b) Moeda (c) Inimigo (d) Letra w (e) Letra o (f) Letra r

(g) Letra t (h) Letra e (i) Letra n

Figura 5.31: Elementos do jogos

No seguimento deste ecrã foi desenvolvido ainda o ecrã de fim de jogo (*game over*), onde se deu uso ao botão específico para o jogo. É utilizada uma variante do botão ainda não apresentada anteriormente, *rankingGameOver*, que não apresenta a posição do utilizador visto que esta já é referida no ecrã, mas continua a reencaminhá-lo para o ecrã de classificação apresentado na figura **5.28 Página de classificação do jogo**. É ainda utilizada uma variante do botão já vista anteriormente, a variante *play*, que permite ao utilizador começar uma nova tentativa, caso possa.

Figura 5.32: Ecrã de fim de jogo (*game over*)

Por fim, foi ainda criado mais um ecrã, o ecrã de pausa. Este permite que o utilizador, a meio do jogo, possa pausar e voltar ao jogo ou desistir da tentativa, perdendo uma vida. É neste contexto que as duas variantes do botão *primary* e *light* referidas anteriormente são utilizadas.

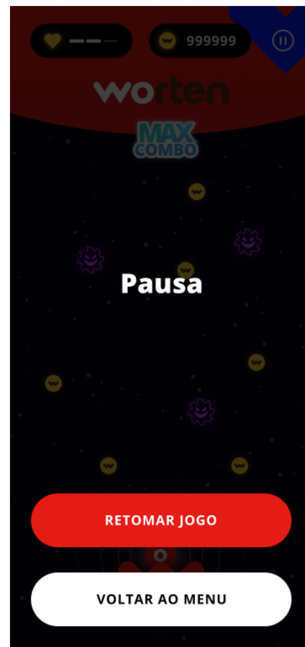



Figura 5.33: Ecrã de pausa do jogo

### Animações

Com a base do jogo construida, tendo os ecrãs necessários ao seu funcionamento, foram efetuadas melhorias de *UI* para tornar o jogo mais apelativo. Uma destas melhorias foi a introdução de animações.

Foram implementadas três animações em contextos diferentes:

- **Multiplicador:** Cada vez que o utilizador completa a palavra *worten*, o multiplicador aumenta. Nesta transição de multiplicadores, é realizada uma animação de *fade in* e *scale down* no novo multiplicador, fazendo o antigo simplesmente desaparecer ao deixar de o renderizar.

- **Pontuação:** A cada tentativa, o utilizador terá a sua pontuação a ser atualizada constantemente. Nesta atualização, a pontuação será alterada à semelhança de uma *slot machine*, como observado em [5 (Bloom, 2010)].

- **Cartões de classificação:** Antes do ecrã de classificação mostrado na figura **5.28**, tinha sido implementado outro diferente que acabou por ser substituído. Este ecrã antigo, para destacar os três primeiros lugares, possuía uns cartões em vez dos triângulos presentes no novo.

Figura 5.34: Cartões de classificação

Quando o ecrã era aberto, os nomes dos três primeiros lugares possuíam uma animação de *scale down* que fazia parecer que caíam no seu cartão, criando um destaque maior para quem estava melhor posicionado na tabela de classificação.

Para dar um melhor entendimento destas animações, é dada a animação do multiplicador como exemplo. É necessário, em primeiro lugar, definir as duas animações com o seu valor inicial. Na animação de *fade in*, o seu valor inicial é 0 visto que vai começar transparente, enquanto na animação de *scale down*, o seu valor inicial será 2 para que depois possa diminuir para o valor normal de 1.

Listing 5.37: Animações

```
const fadeInAnimation = useRef(new Animated.Value(0)).current;
const scaleAnimation = useRef(new Animated.Value(2)).current;
```

A seguir é necessário definir a função que controla a animação. Neste exemplo, ambas as animações levarão 750 milissegundos, uma a transitar do valor 0 para 1 e outra de 2 para 1. Por questões de robustez de código, é assegurado de que os valores estão estabelecidos para o pretendido e de seguida, as animações são iniciadas em paralelo.

Listing 5.38: Função de animação

```
const animateMultiplier = useCallback(() => {
  const fadeIn = Animated.timing(fadeInAnimation, {
    toValue: 1,
    duration: 750,
    useNativeDriver: true,
  });

  const scaleDown = Animated.timing(scaleAnimation, {
    toValue: 1,
```

```
    duration: 750,
    useNativeDriver: true,
  });

  fadeInAnimation.setValue(0);
  scaleAnimation.setValue(2);

  Animated.parallel([fadeIn, scaleDown]).start();
}, [fadeInAnimation, scaleAnimation]);
```

É então utilizada esta função de forma a que a animação seja efetuada no contexto pretendido, neste caso quando o multiplicador for apenas maior que 1.

Listing 5.39: Utilização da função de animação

```
useEffect(() => {
  if (multiplier > 1) {
    animateMultiplier();
  }
}, [animateMultiplier, multiplier]);
```

Por fim utiliza-se um contentor *View* da biblioteca ***Animated***, que permite que componentes tenham os valores das suas propriedades alterados em tempo real. Neste caso, a opacidade do componente ficou designado com o valor da animação de *fade in* e foi feito um escalamento com o valor da animação de *scale down*.

Listing 5.40: Utilização da animação num componente

```
return (
  <Animated.View
    style={{
      opacity: fadeInAnimation,
      transform: [{scale: scaleAnimation}],
    }}>
    {multiplierIcon[multiplier - 1]}
  </Animated.View>
);
```

### Mecânicas

No que toca ao motor do jogo, foi apenas realizada uma tarefa, implementar os padrões dos elementos de forma a que todas as tentativas de qualquer utilizador sejam iguais, fazendo com que a pontuação apenas seja influenciada pela forma de jogar, diminuindo o fator de "sorte". Estes padrões são chamados de tapetes ou carruagens e foram criados quinze, para poder haver reutilização de uns em diferentes níveis.

Estes tapetes foram criados pelos *designers* para serem transformados numa lista de valores, com coordenadas e o tipo de elemento. Assim, foi necessário implementar um código que renderizasse os elementos conforme os dados disponibilizados na lista, permitindo que caso fosse preciso realizar alguma alteração às posições ou aos elementos, a única a ser feita era na lista.

Os dez níveis do jogo foram construídos consoante a figura abaixo, onde os números representam tapetes que não possuem letras e as letras representam tapetes que possuem. Por exemplo, no primeiro nível, existirão três tapetes que possuem letras, dando a possibilidade ao utilizador de ficar com "wor" da palavra "worten". De notar que neste primeiro nível existem dois tapetes iguais, o tapete 1.

| Order | module |
|---|---|
| level 1 | 1,A,3,B,1,D |
| level 2 | 4,A,D,2,B,3,1 |
| level 3 | C,2,1,A,B,2,2,4 |
| level 4 | 4,D,3,C,2,B,1,3,4 |
| level 5 | 1,2,3,4,A,B,C,2,4,1 |
| level 6 | B,2,3,4,C,3,1,3,2,D,2 |
| level 7 | A,1,1,B,2,2,2,C,3,3,3,3 |
| level 8 | D,2,2,1,3,B,1,2,3,D,3,2,1 |
| level 9 | 4,3,2,1,A,1,2,3,4,B,2,2,C,3 |
| level 10 | 2,4,C,3,1,A,D,4,1,3,3,2,4,1,4 |

Figura 5.35: *Design* dos níveis

Considera-se o seguinte exemplo de *design* do tapete A do nível 1, desenhado na figura seguinte. De notar que a primeira linha do tapete é a de baixo.

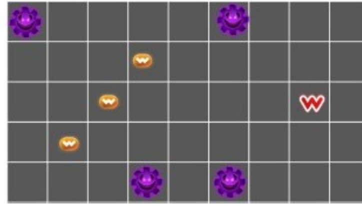


Figura 5.36: Exemplo retirado de um tapete

Com este exemplo de disposição de elementos, a lista de coordenadas e tipos de elementos ficaria com a seguinte estrutura:

Listing 5.41: Implementação de um tapete

```
const config = [
    {posX: 4, posY: 1, type: enemy},
    {posX: 6, posY: 1, type: enemy},
    {posX: 2, posY: 2, type: coin},
    {posX: 3, posY: 3, type: coin},
    {posX: 8, posY: 3, type: letter},
    {posX: 4, posY: 4, type: coin},
    {posX: 1, posY: 5, type: enemy},
    {posX: 6, posY: 5, type: enemy},
];
```

#### ***Bugfixes* e Ajustes de *UI***

Para além destas tarefas desenvolvidas, o jogo foi sofrendo algumas mudanças tanto a nível de *UI* como de correções de erros.

Durante o seu desenvolvimento foram realizadas algumas alterações onde se destacam estas:

- **Correção de elementos de *UI*:**
  - Correção dos botões de *game over* que não mostravam qualquer *feedback* visual, fazendo parecer com que o utilizador não o tivesse premido.
  - Correção do espaçamento inferior das *drawers* do menu principal (figura **5.29**). O final do conteúdo de cada *drawer* estava cortado mesmo quando feito o *scroll* todo da página.
- **Correção do comportamento do ecrã de jogo:** Durante uma tentativa, caso o ecrã de jogo fosse puxado para baixo a partir do cabeçalho (à semelhança de uma *drawer*), o

jogo era fechado como se o utilizador tivesse desistido da tentativa, perdendo uma vida.

- **Correção do comportamento das vidas:** Caso um utilizador fechasse a aplicação enquanto estava numa tentativa, este não perdia uma vida visto que só lhe era retirada quando perdesse o jogo, ou seja, quando fosse apresentado o ecrã de final de jogo (figura **5.32**). Para corrigir isso, a vida passou a ser retirada assim que uma tentativa começava.

- **Alteração de *design* geral:** No início do desenvolvimento, o jogo teve um *design* diferente, onde ainda não era confirmado que este seria criado para a campanha da *Ayvens*. Quando se obteve esta confirmação, existiu uma mudança de certos ecrãs. Por exemplo, antes da alteração, o menu principal (figura **5.26 Menu do jogo**) tinha o seguinte aspeto:

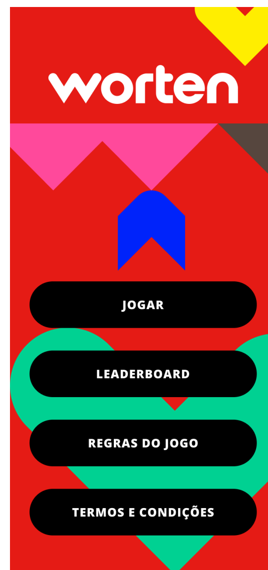
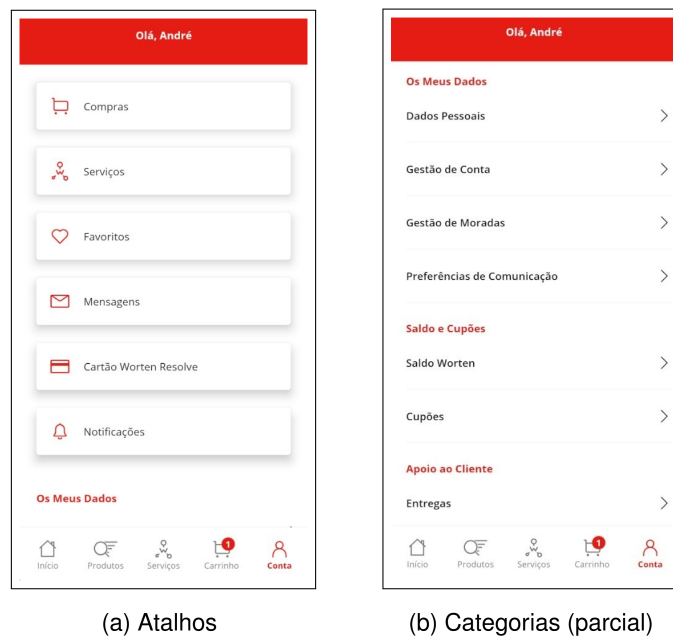



Figura 5.37: Menu principal antes da campanha *Ayvens*

### 5.3.3 Área de Cliente

Para além do projeto do jogo, existiu outro projeto dividido em várias tarefas, uma total remodelação da área de cliente, tornando esta mais apelativa e intuitiva ao passar a apresentar novas informações na página principal de cliente, as quais serão descritas a seguir. Antes da remodelação, a área cliente tinha o *design* apresentado na figura abaixo, onde existiam atalhos em forma de cartões horizontais e categorias como “Os Meus Dados”, “Saldo e Cupões”, “Apoio ao Cliente”, “Sobre Nós” e “Sobre a App”. Esta área cliente funcionava como uma lista de páginas a aceder, sem apresentar nenhuma informação diretamente ao cliente, ao contrário do que refere a lei de *Paul Fitts*, onde uma das suas aplicações é a minimização de cliques e interações, que refere que é preferível que uma ação seja realizada com o menor número de

cliques possíveis, evitando menus ou interações complexas. Para mais informações sobre a aplicação da lei de *Paul Fitts* em *UX*, consultar [7 (Budiu, 2022)].




(a) Atalhos (b) Categorias (parcial)

Figura 5.38: Área de cliente antes da remodelação

Seguindo este princípio, este ecrã começou a ser remodelado a partir do cabeçalho, que passou a ter informações e possíveis interações. Para visualizar a conta, o botão foi movido para perto do nome do utilizador onde é aberta uma *drawer* com o mesmo menu de conta que existia anteriormente. Apesar de se tornar o menu da conta acessível com dois toques em vez de um, na área de cliente antiga era necessário fazer *scroll* para visualizar as opções, para além de que foi posicionado junto ao nome da conta, o que o torna mais intuitivo.

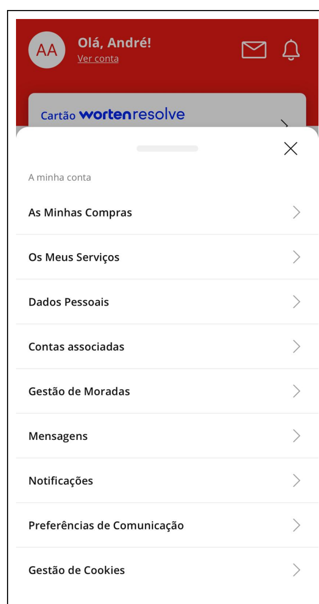
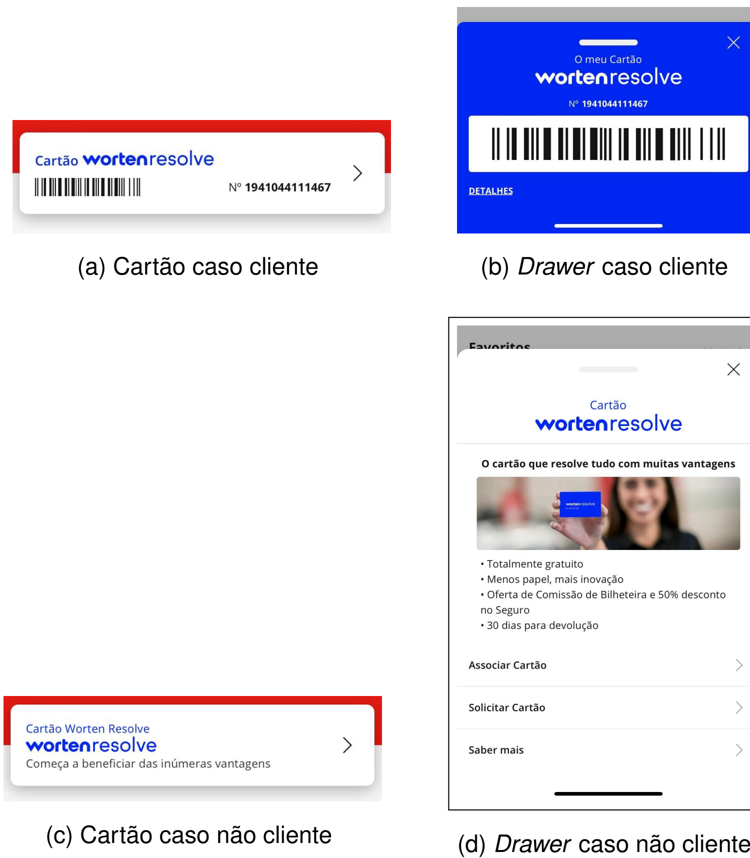



Figura 5.39: Menu da conta

Como é possível observar na figura anterior, para além do botão do menu da conta, passam a estar presentes mais dois botões, mensagens e notificações, representados apenas por um ícone cada um.

Para além destas alterações, foi criado um cartão “flutuante” para acesso rápido ao cartão do serviço *Worten Resolve*. Este cartão pode ter dois *designs*/estados, dependendo se o utilizador já aderiu ao serviço ou não. No entanto, qualquer um dos estados abre uma *drawer* seja com a informação do cartão ou com informação para aderir ao serviço.




(a) Cartão caso cliente

(b) *Drawer* caso cliente

(c) Cartão caso não cliente

(d) *Drawer* caso não cliente

Figura 5.40: *Worten Resolve* na área cliente

Foram também desenvolvidos, mas não utilizados, quatro cartões utilizando o componente *info blocks* descrito em **5.2.1 Componente *Info Blocks***. Assim, a visualização do saldo e cupões da *Worten* e do Continente seria mais fácil e rápida. No entanto, esta implementação ficou sem efeito e não está presente na aplicação.

Figura 5.41: Cartões de saldo e cupões (*Worten* e Continente)

Logo a seguir ao cartão *Worten Resolve*, foram implementadas duas listas com cartões utilizando o componente *Product Card Container* apresentado na figura **5.14**, uma para as compras feitas pelo utilizador e outra para os seus favoritos. Desta forma consegue-se cumprir o objetivo proposto anteriormente quando referido que o utilizador deverá efetuar o menor

número de toques possíveis para realizar uma ação, neste caso visualizar as suas compras ou favoritos. De notar que na figura abaixo não se vê o cartão *Worten Resolve* mostrado na figura **5.40** visto que este desaparece quando é feito *scroll* para baixo, de forma a deixar mais espaço do ecrã disponível para outros conteúdos.

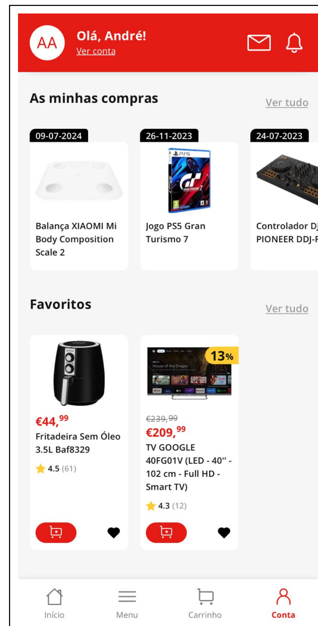


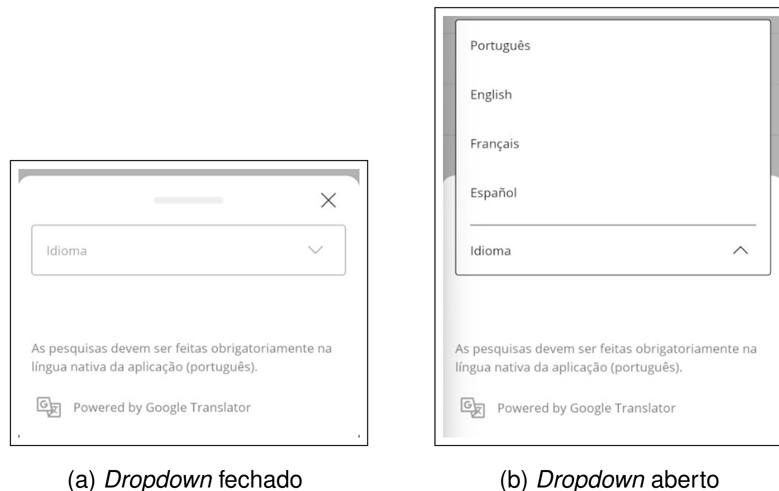

Figura 5.42: Compras e favoritos do utilizador

Nesta nova área de cliente foi também implementada a opção de mudança de idioma, contendo português (pt), inglês do Reino Unido (en), espanhol (es), francês (fr) e alemão (de), onde são utilizados os *icons* apresentados na figura **5.21** para identificar cada um. Para mostrar esta opção de troca de idioma, foi utilizado o componente *Action List Item* apresentado em **5.2.1 Componente Action List Item**.

De notar que esta funcionalidade de mudança de idioma não se encontra ainda contemplada na aplicação, visto que ainda não foi totalmente desenvolvida.

Figura 5.43: *Item* de mudança de idioma

Ao selecionar esta opção, seria aberta uma *drawer* com informação e uma lista de idiomas em forma de *dropdown* dos quais o utilizador pode escolher.




(a) *Dropdown* fechado (b) *Dropdown* aberto

Figura 5.44: *Drawer* de mudança de idioma

Por fim, no que toca à área de cliente, foi ainda realizada mais uma tarefa, mas desta vez de correção de uma tarefa previamente realizada. Os cartões de favoritos apresentados na figura **5.42** não estavam completos visto que estes tinham sido implementados de forma semelhante dos cartões das compras do utilizador presentes na mesma figura, onde apenas eram apresentados os produtos, sem qualquer botão adicional. Estes cartões foram então corrigidos e testados de forma a apresentarem mais dois botões, "adicionar ao carrinho" (botão vermelho no canto inferior esquerdo) e "remover dos favoritos" (botão preto no canto inferior direito).

### 5.3.4 Traduções

A *Worten*, ao possuir cada vez mais utilizadores na aplicação móvel, decidiu implementar uma aplicação multi-idioma de maneira a incluir os países vizinhos e os mais significativos da Europa. Os idiomas implementados foram os já referidos e apresentados anteriormente na figura **5.21**, ou seja, português, inglês, espanhol, francês e alemão.

Assim, em vez de se possuir um texto diretamente introduzido num componente *Text* como observado abaixo,

Listing 5.42: Implementação sem tradução

```
<Text size="M" style={TEXT}>
    Seguir as tuas compras online
</Text>
```

passa-se a utilizar o componente ***TKey***, construído para auxiliar a implementação das traduções:

Listing 5.43: Implementação com tradução

```
<Text size="M" style={TEXT_ITEM}>
    <TKey keyPath="confirm-delete.order-tracking" ns="account" />
</Text>
```

Este componente baseia-se na biblioteca ***i18next*** já falada em **3.2 Bibliotecas**, que pode ser descrita como uma *framework* de internacionalização desenvolvida em *JavaScript*. O componente possui dois parâmetros, ***ns*** (*namespace*) e ***keyPath***, que simbolizam uma *string* específica a ser utilizada, tal como é feito no exemplo acima.

Os *caminhos* especificados em *ns* e *keyPath* são utilizáveis graças à criação de múltiplos ficheiros *JSON*, cada um relativo a um idioma específico. Estes ficheiros possuem uma estrutura onde existem diferentes nós principais, como $ns = "account"$, que se dividem em várias ramificações, como *confirm-delete*, que, por sua vez, se desdobram em outras, como *order-tracking*.

Listing 5.44: JSON com traduções - PT

```
"account": {
    "main": {
        (...)
    },
    "confirm-delete": {
        "elimination-process": "Vais iniciar o processo de eliminação de conta, ao
            avançar deixas de conseguir:",
        "order-tracking": "Seguir as tuas compras online",
        "history-access": "Aceder ao histórico das tuas compras e consultar
            respetivas faturas",
        (...)
    },
}
```

Dependendo então do idioma que o utilizador escolhia no menu *dropdown* da figura **5.44**, é usado um ficheiro *JSON* diferente para chegar à *string* pretendida do idioma em questão. Por exemplo, caso o utilizador escolha inglês, o ficheiro *JSON* usado passa a ter o texto em inglês, mantendo a estrutura e as chaves iguais:

Listing 5.45: JSON com traduções - EN

```
"account": {
    "main": {
      (...)
    },
    "confirm-delete": {
            "elimination-process": "You will start the process of deleting your
                account, as you go along you will no longer be able to:",
            "order-tracking": "Track your online orders",
            "history-access": "Access your order history and check your invoices",
            (...)
    },
}
```

### 5.3.5 Migração de Componentes

Por fim, foi realizada a migração de dois componentes do *Design System* para a *app* de produção, os componentes ***Express Filters*** e ***Cookies***. Esta decisão foi tomada por uma questão de organização dos projetos, tanto do *DS* como da aplicação principal, visto que os componentes que estão implementados no *DS* são utilizados mais do que uma vez na *app*, economizando recursos e promovendo a otimização do código ao não repetir a implementação de componentes iguais. No entanto, estes dois componentes são apenas utilizados uma vez, o que faz com que não correspondam aos requisitos do *DS*. Portanto, estes foram movidos para a *app* para serem utilizados apenas no contexto pretendido.

#### Componente *Express Filters*

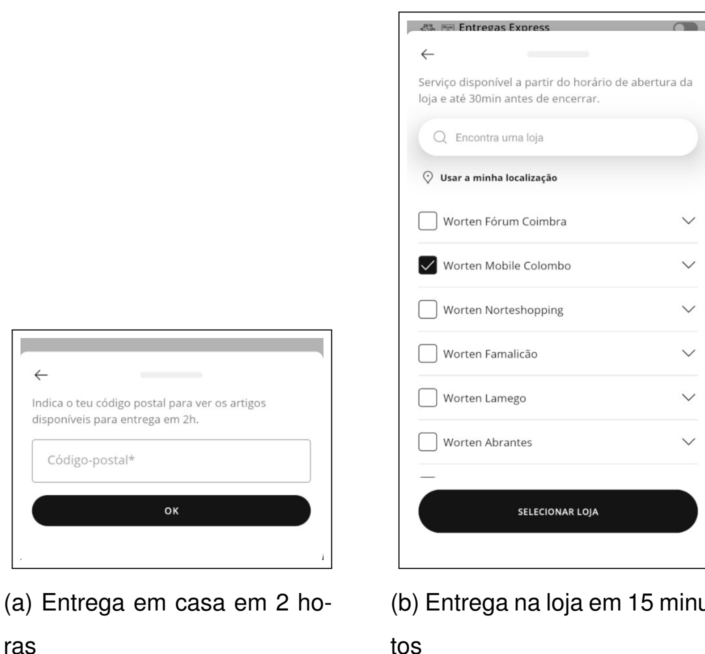
Este componente tem como objetivo apresentar opções de entrega/levantamento de produtos e está presente no topo dos ecrãs referentes a produtos (página de categoria de produtos ou produto individual). Possui um *design* semelhante ao componente *Action List Item* descrito em **5.2.1 Componente *Action List Item***.

Figura 5.45: Cabeçalho do componente *Express Filters*

Quando o utilizador clica nos filtros será aberta uma *drawer* geral que dará duas opções, de entrega em casa dentro de 2 horas ou de levantamento numa loja à escolha em 15 minutos.

Figura 5.46: *Drawer* principal do componente *Express Filters*

Nas *drawers* de entregas/levantamento, o utilizador terá de fornecer o código postal ou escolher uma das lojas que deseja e, quando concluído, o filtro será aplicado à sua pesquisa.




(a) Entrega em casa em 2 horas

(b) Entrega na loja em 15 minutos

Figura 5.47: *Drawers* de entregas/levantamentos do componente *Express Filters*

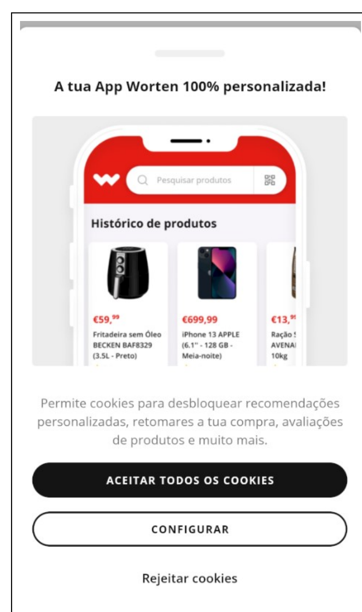
A *drawer* principal tal como as *drawers* para cada opção estão incluídas na implementação e migração do componente e, dado que são *drawers*/modais muito específicas para serem utilizadas noutro contexto, foi realizada a sua migração para a *app*.

#### Componente *Cookies*

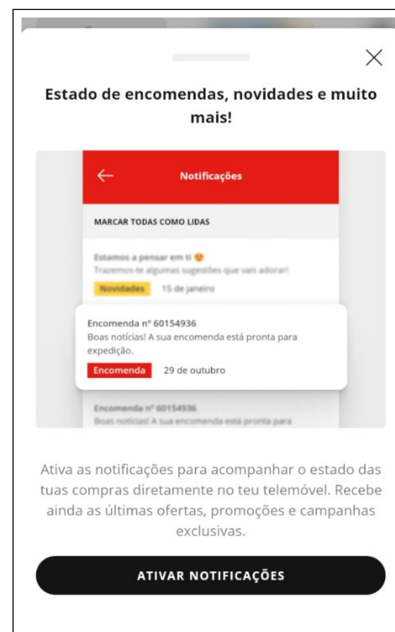
O mesmo acontece no cenário do componente *Cookies*, utilizado na *dashboard* da aplicação ao apresentar produtos semelhantes às pesquisas do utilizador e que podem ser do seu interesse. No entanto, estes produtos só são apresentados caso os *cookies* sejam aceites.

Figura 5.48: Lista de cartões de produtos sugeridos

Os *cookies* podem então ser aceites através do primeiro cartão mostrado na figura anterior ou ao iniciar pela primeira vez a aplicação, onde são apresentadas ao utilizador duas modais, uma relativa aos *cookies* e outra às notificações.




(a) *Cookies*




(b) Notificações

Figura 5.49: *Drawers* de *onboarding*

# Capítulo 6

# Conclusões

Este relatório teve como objetivo descrever o trabalho realizado durante o estágio na empresa ***Bliss Applications***, consistindo no desenvolvimento de uma aplicação móvel em ***React Native*** para uma das maiores lojas de retalho portuguesas, a ***Worten***. Este estágio forneceu a oportunidade de colaborar em várias frentes de desenvolvimento, desde a implementação de funcionalidades até à manutenção e otimização da aplicação já desenvolvida.

Ao longo deste estágio, foram abordadas diversas áreas essenciais para a compreensão e execução de um projeto de *software* com estas características. No capítulo **2 Trabalho Relacionado**, foi feita uma análise comparativa com outras aplicações de retalho semelhantes, como a *FNAC*, *MediaMarkt*, *Amazon* e *eBay*, permitindo avaliar os pontos fortes e os que necessitavam de uma melhoria com base em tendências e funcionalidades do mercado. Além disso, foram comparadas algumas tecnologias de desenvolvimento multiplataforma, nomeadamente *Flutter*, *Xamarin*/*.NET MAUI* e *Ionic*, onde foi analisada a escolha de *React Native* quanto à sua flexibilidade, desempenho e integração no ecossistema de desenvolvimento de aplicações.

No capítulo **3 Tecnologias Utilizadas**, exploraram-se as várias ferramentas e bibliotecas que foram utilizadas ao longo do projeto. A escolha de *React Native* revelou-se acertada pela possibilidade de reutilização de código entre plataformas, acelerando o desenvolvimento e garantindo uma experiência uniforme para o utilizador. É referido e descrito o uso de certos *frameworks* e bibliotecas, que facilitaram a implementação de funcionalidades mais complexas, e o uso de ferramentas como o ***Jira*** e o ***Git***, que permitiram uma organização e um controlo eficaz do projeto, essenciais para a coordenação entre e dentro das equipas.

O capítulo **5 Trabalho Realizado** reflete o contributo dado durante o estágio, onde é abordado o curso de ***React Native***, que forneceu as bases necessárias para a rápida integração no projeto, o trabalho realizado no ***Design System***, onde colaborei no desenvolvimento e manu-

tenção de componentes para garantir a consistência visual na aplicação, e o desenvolvimento de tarefas na aplicação de produção, como o jogo, onde foram aplicados os conhecimentos adquiridos em animações, navegação e *design* de interfaces, ou a nova área de cliente, onde foram aplicados os mesmos conhecimentos num contexto diferente, aumentando a curva de aprendizagem no que toca ao ***React Native***.

Foi ainda destacada, no capítulo **4 Metodologia**, a importância de uma abordagem iterativa e ágil no desenvolvimento de *software*. Mediante ciclos curtos de desenvolvimento, teste e revisão, garantiu-se que a aplicação evoluiu de forma eficiente e alinhada com os objetivos do cliente.

No entanto, apesar do progresso significativo alcançado durante o estágio e o desenvolvimento da aplicação móvel, continuam a existir várias áreas que poderão ser exploradas e melhoradas no futuro de forma a garantir uma evolução contínua e melhoria da experiência do utilizador.

Com o mercado em constante evolução, será importante que a aplicação incorpore novas funcionalidades de *e-commerce*, como realidade aumentada (*AR*) para a visualização de produtos, ou ainda melhorias na personalização da experiência do utilizador, recorrendo à **inteligência artificial** e ***machine learning***, onde estas tecnologias poderão ser usadas para uma recomendação mais eficiente de produtos.

Outra linha de trabalho será a internacionalização da aplicação. Embora já tenha sido implementado um sistema de traduções para diferentes línguas, este ainda não foi publicado em qualquer versão da aplicação devido à necessidade de melhorias. Além disso, o crescimento da *Worten* para novos mercados poderá necessitar de **ajustes culturais específicos**, garantindo que o conteúdo está adaptado a diferentes públicos-alvo.

De forma concisa, este estágio foi uma oportunidade enriquecedora para aplicar os conhecimentos adquiridos ao longo do mestrado e desenvolver novas competências, nomeadamente em ***React Native*** e no desenvolvimento de aplicações móveis. O trabalho realizado e descrito não só contribuiu para a melhoria do projeto, como também proporcionou uma visão das práticas e desafios que existem no desenvolvimento de *software* nesta indústria.

# Bibliografia


[1] AgiliWay. *Flutter, Ionic, React Native, and Xamarin*. 2022. URL: https://www.agiliway.com/popular-open-source-frameworks-flutter-ionic-react-native-and-xamarin.

[2] Bliss Applications. *About Us*. 2024. URL: https://www.blissapplications.com/about-us.

[3] Bliss Applications. *Work*. 2024. URL: https://www.blissapplications.com/work.

[4] Atlassian. *Welcome to Jira*. 2024. URL: https://www.atlassian.com/software/jira/guides/getting-started/introduction.

[5] Eliana Bloom. *Slot Machine Animation*. Vídeo no YouTube. 2010. URL: https://www.youtube.com/watch?v=EX-uzP6BdjI.

[6] Renan Braga. *Package Managers: Understanding npm, npx and yarn*. 2023. URL: https://dev.to/azhariel/package-managers-understanding-npm-npx-and-yarn-3j0c.

[7] Raluca Budiu. *Fitts's Law and Its Applications in UX*. 2022. URL: https://www.nngroup.com/articles/fitts-law.

[8] Maciej Budziński. *What Is React Native? Complex Guide for 2024*. 2024. URL: https://www.netguru.com/glossary/react-native.

[9] Chris Coyier. *CSS Flexbox Layout Guide*. 2024. URL: https://css-tricks.com/snippets/css/a-guide-to-flexbox/.

[10] Figma. *About Us*. 2024. URL: https://www.figma.com/about.

[11] Flatirons. *Expo vs React Native CLI: Key Differences Explained*. 2024. URL: https://flatirons.com/blog/expo-vs-react-native.

[12] GitLab. *About GitLab*. 2024. URL: https://about.gitlab.com/company.

[13] Google. *Flutter FAQ*. 2024. URL: https://docs.flutter.dev/resources/faq.

[14] Richard Harris. *Software Development Life Cycle (SDLC)*. 2021. URL: https://www.linkedin.com/pulse/software-development-life-cycle-sdlc-tutorial-richard-harris.

[15] Ionic. *Ionic Docs*. 2024. URL: https://ionicframework.com/docs.

[16] Selva Ganapathy Kathiresan. *Xamarin Versus .NET MAUI*. 2024. URL: https://www.syncfusion.com/blogs/post/xamarin-versus-net-maui.

[17] Ritesh Kumar. *SVG to React Native*. 2019. URL: https://transform.tools/svg-to-react-native.

[18] Meta Platforms. *FlatList*. 2024. URL: https://reactnative.dev/docs/flatlist.

[19] Meta Platforms. *Getting started | React Navigation*. 2024. URL: https://reactnavigation.org/docs/getting-started.

[20] Meta Platforms. *Props*. 2024. URL: https://reactnative.dev/docs/props.

[21] Meta Platforms. *State*. 2024. URL: https://reactnative.dev/docs/state.

[22] Meta Platforms. *StyleSheet*. 2024. URL: https://reactnative.dev/docs/stylesheet.

[23] Simon Slade. *Amazon vs. eBay*. 2024. URL: https://www.salehoo.com/learn/crucial-differences-between-amazon-and-ebay.

[24] Statista. *App - Worldwide*. 2024. URL: https://www.statista.com/outlook/amo/app/worldwide.

[25] WYgroup. *About Us*. 2024. URL: https://www.wygroup.net/about-us/.