Desenvolvimento de Software para Marketplaces

Mestrado em Engenharia Informática – Computação Móvel

José Miguel Felizardo Pereira

Estudante nº 2190341

Leiria, janeiro de 2022

Desenvolvimento de Software para Marketplaces

Mestrado em Engenharia Informática – Computação Móvel

José Miguel Felizardo Pereira

Estudante nº 2190341

Estágio realizado sob a orientação do Professor Doutor Paulo Manuel Almeida Costa e sob supervisão do Sr. Engenheiro Luís Soares

Leiria, janeiro de 2022

Originalidade e Direitos de Autor

O presente relatório de estágio é original, elaborado unicamente para este fim, tendo sido devidamente citados todos os autores cujos estudos e publicações contribuíram para o elaborar.

Agradecimentos

A elaboração do presente relatório final de estágio não seria possível sem o apoio de alguns intervenientes. Assim sendo, pretendo agradecer a todos os que sempre me apoiaram e contribuíram para a concretização desta etapa final na minha formação, o Mestrado em Engenharia Informática – Computação Móvel.

Deste modo, agradeço primeiramente à minha mãe, por todo o apoio durante todo o meu percurso no ensino superior, em especial durante a realização do presente mestrado.

A todos os meus colegas e superiores na xgeeks, em especial ao Eng. Luís Soares por toda a atenção, compreensão e valiosos conhecimentos transmitidos, assim como pela amizade e espírito de comunidade com que me receberam durante o período de estágio.

Por fim, mas de igual importância, ao Professor Paulo Manuel Almeida Costa pela sua disponibilidade e compreensão, orientando no decorrer do meu trabalho e durante a escrita do presente relatório de estágio.

Resumo

Os *marketplaces*, ou lojas digitais, são umas das aplicações *web* mais comumente desenvolvidas. Porém, existem determinados desafios associados ao desenvolvimento de grandes plataformas de comércio digital na *web* moderna que requerem especial atenção. Na era de *frameworks* e bibliotecas de JavaScript, existe uma crescente necessidade de tomar decisões mais informadas, de modo a escolher uma pilha de tecnologias que melhor se adequa a este tipo de ambientes comerciais. Com o objetivo de explorar estas decisões, o presente relatório foca-se num projeto desenvolvido durante um estágio com a duração de 9 meses. Além destas decisões, este projeto apresentou também outros desafios, como a adaptação do conteúdo de uma aplicação *web* para a escrita árabe (da direita para a esquerda).

Palavras-chave: *Marketplaces*, *Web* Moderna, *Frontend*, Árabe

Abstract

Marketplaces, or online stores, are one of the most developed web applications, but there are certain challenges that come with big e-commerce platforms on the modern web that need special attention. In the era of JavaScript *frameworks* and libraries, there's a need to make more informed decisions and compromises, to choose a technology stack that better suits a professional and commercial environment. To shine light on some of these decisions, this report focuses on a particular project developed during a 9-month internship. Besides these technology decisions, this project posed some other challenges, like adapting a web application for Arabic content (written from right to left).

Keywords: Marketplaces, Modern Web, Frontend, Arabic

Índice

Lista de Figuras

Lista de Siglas e Acrónimos

API	*Application Programming Interface*
CD	*Continuous Delivery*
CDN	*Content Delivery Network*
CI	*Continuous Integration*
CMS	*Content Management System*
CSS	*Cascading Style Sheets*
CSV	*Comma-Separated Values*
DOM	*Document Object Model*
ETL	*Extract Transform Load*
FTP	*File Transfer Protocol*
GIF	*Graphics Interchange Format*
HTML	*HyperText Markup Language*
JPG/JPEG	*Joint Photographic Expert Group*
JSON	*JavaScript Object Notation*
JSX	*JavaScript Syntax Extension*
LRM	*Left-To-Right Mark*
LTR	*Left-To-Right*
MPA	*Multi-Page Application*
MVP	*Minimum Viable Product*
PNG	*Portable Network Graphics*
PWA	*Progressive Web App*

REST	*REpresentational State Transfer*
RTL	*Right-To-Left*
SaaS	*Software as a Service*
SDK	*Software Development Kit*
SEO	*Search Engine Optimization*
SKU	*Stock Keeping Unit*
SPA	*Single-Page Application*
SSG	*Static Site Generator*
SSR	*Server-Side Rendering*
URL	*Uniform Resource Locator*
XLSX	*Microsoft Excel Spreadsheet* (formato *Open Office XML*)

1. Introdução

A *web* moderna é um conceito que muitos ouvem falar, mas por vezes não sabem a que se refere. A maioria das aplicações de *web* modernas comumente utilizam algum tipo de *framework*/biblioteca JavaScript, bibliotecas estas que têm vindo a ser popularizadas nos últimos anos, como o React e o Vue.js. Estas tecnologias são extremamente poderosas para o desenvolvimento de aplicações bem otimizadas, responsivas, escaláveis e preparadas para diversos dispositivos. Contudo, trazem também um preço a pagar, que pode ser tão óbvio como o decréscimo de performance, comparado com uma tradicional aplicação desenvolvida apenas com recurso a JavaScript, HTML (*HyperText Markup Language*) e CSS (*Cascading Style Sheets*), ou desvantagens de SEO (*Search engine optimization*), resultantes do desenvolvimento de uma SPA (*Single-page application*), em vez de uma MPA (*Multi-page application*). Estas cedências são bastante importantes, e devem ser bem consideradas quando se pretende desenvolver uma aplicação como uma plataforma de comércio digital, devido ao facto destes fatores estarem diretamente ligados ao lucro gerado pelas mesmas. Isto é algo que se verifica, quando consideramos que, quando uma aplicação pública não é devidamente otimizada para motores de busca, esta não obterá a devida visibilidade, ou se a mesma não se adaptar corretamente para dispositivos móveis, os respetivos utilizadores/clientes optarão por não a utilizar nos seus *smartphones*. Contudo, a boa notícia é que, com o rápido avanço nas *frameworks* e bibliotecas JavaScript, já existem soluções que visam mitigar a maioria destes problemas, sem aumentar o esforço de desenvolvimento. *Frameworks* como Next.js e Gatsby são exemplos de soluções *open-source* e bastante completas para esse efeito.

O presente relatório descreve o trabalho realizado no decorrer de um estágio com uma duração de 9 meses (17 de Agosto de 2020 a 17 de Maio de 2021), realizado na empresa xgeeks Portugal, em Leiria, no âmbito do Mestrado em Engenharia Informática – Computação Móvel do Instituto Politécnico de Leiria.

1.1. Motivação e Objetivos

A escolha da realização de um estágio curricular, ao invés de um projeto ou dissertação, baseou-se em grande parte pela ambição de integração num ambiente profissional e

experienciação de desafios técnicos e reais como um engenheiro de software, numa empresa da área. A posição ocupada durante este estágio foi de Engenheiro de Software *Full Stack*, na qual tive a oportunidade de ser integrado numa equipa profissional e experiente de engenheiros, e trabalhar em projetos para clientes reais e de elevado valor para a empresa.

No decorrer do mesmo, o trabalho realizado focou-se primariamente num projeto de desenvolvimento *web*. Projeto esse, que teve como objetivo o desenvolvimento de um *marketplace* digital, utilizando um conjunto de tecnologias *web* moderna, para um numeroso e particular público-alvo. Este relatório atenta nessas mesmas tecnologias, particularmente no contexto de aplicações de comércio digital, com o objetivo de descrever os seus benefícios e os desafios enfrentados na sua utilização. Por motivos de confidencialidade, ao longo deste relatório referir-me-ei ao projeto em questão como "projeto XYZ".

1.2. Divisão de Capítulos

Este relatório encontra-se organizado em 6 capítulos:

1. **Introdução** – Descrição do tema a relatar e do trabalho realizado durante o presente estágio.

2. **Caracterização da Entidade de Acolhimento** – Detalhes da entidade acolhedora, tais como áreas de atuação, organização, localização, entre outros.

3. **Programa de Estágio** – Apresentação do programa de estágio estabelecido pela entidade acolhedora e descrição das diferentes fases decorridas desde o início ao final do mesmo.

4. **Processo de desenvolvimento** – Descrição do principal período que decorreu durante o estágio, no qual tive oportunidade de pôr em prática as minhas capacidades técnicas no desenvolvimento de *software* em projetos de clientes da entidade.

5. **Conclusão** – Descrição e reflexão sobre a conclusões obtidas durante e após o período de estágio, acerca do trabalho efetuado.

2. Caracterização da Entidade de Acolhimento

A entidade de acolhimento do presente estágio é denominada de XGEEKS PORTUGAL, LDA e encontra-se sediada no Edifício Moagem, Rua de São Francisco 14, Frações F-G 2400-230 Leiria. Foi fundada em 2019, em Leiria, e atua no setor de Atividades de programação informática. Esta faz parte do grupo alemão KI group, sediado em Colónia, que atua também na área de informática e tecnologias de informação, e conta com diversas divisões que atuam em diversas áreas, sendo a xgeeks uma das mais orientada para a vertente de engenharia. Esta já conta com diversos clientes de nome respeitável, como DAIMLER (grupo que detém a marca Mercedes), Porsche, CheMondis, saloodo, entre outros.

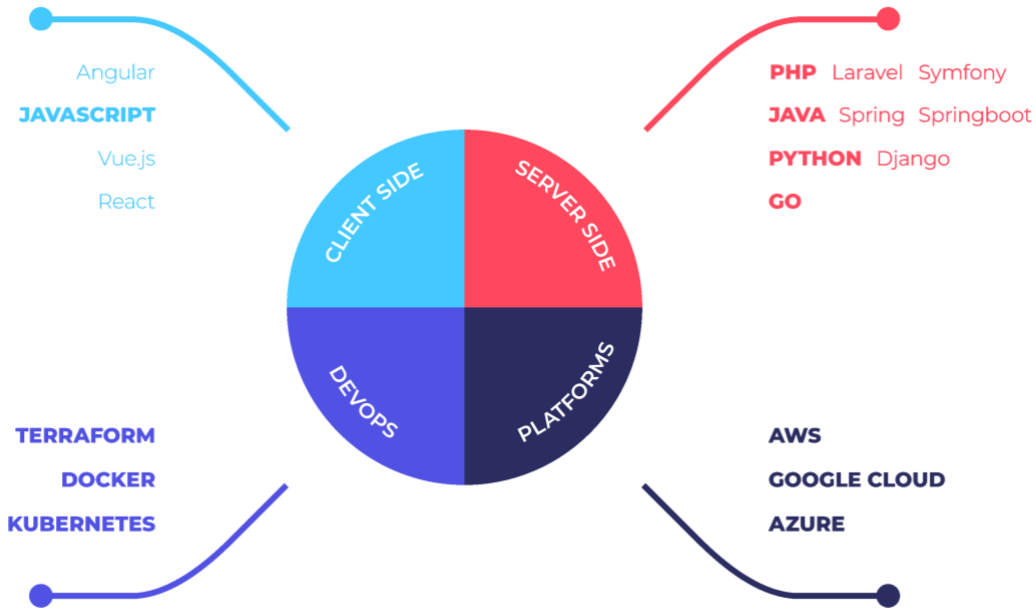

Figura 1 - Tecnologias utilizadas pelas equipas da xgeeks

O conjunto de tecnologias com que a xgeeks trabalha é bastante diverso e moderno, abrangendo áreas como *Frontend*, *Backend*, *DevOps*, *Data Science*, entre outras, como é possível observar na Figura 1. É também do interesse da mesma expandir a sua equipa para áreas emergentes específicas, como *Blockchain* e *Machine Learning*. Atualmente, esta conta com quase 100

engenheiros, com tendência a crescer rapidamente nos próximos anos, e com 3 escritórios em Portugal, nas localidades de Leiria, Lisboa e Viseu. Além dos escritórios da xgeeks, é possível contar também com os 6 escritórios do KI Group na Alemanha, nas localidades de Colónia, Estugarda (x2), Munique, Berlin e na Suíça, na localidade de Zurique.

2.1. Missão e Valores

Uma das principais missões da xgeeks é produzir software de qualidade, focando-se em tecnologias modernas e inovadoras, atuando num modelo semelhante ao de consultoria. Esta desenha e constrói soluções para *start-ups* e *scale-ups* em conjunto com uma equipa internacional, com experiência e *background* de algumas das maiores empresas da Europa. Os seus valores baseiam-se em inovação, impacto, crescimento, agilidade, excelência pessoal e espírito de equipa. É também uma empresa muito focada em *team building* e no bem-estar dos seus colaboradores, frequentemente realizando eventos informais de modo a promover a interação entre membros da equipa. Estes eventos variam entre festas de Natal e verão, paintball, *laser quest*, entre outros. Esta valoriza também a presença online e o contributo para a comunidade de desenvolvimento de software, demonstrando-o através de:

- Contribuições para *open-source*, como é possível ver na página de Github da mesma (@xgeekshq);
- Frequente realização de apresentações/*talks* com convidados externos e, por vezes internos. Estas apresentações são geralmente realizadas através da plataforma YouTube (@xgeeksio);
- Frequente publicação de artigos técnicos na plataforma Medium (@xgeeks), escritos por engenheiros da equipa, assim como de anúncios e sugestões nas redes sociais Twitter (@xgeeksio), Facebook (@xgeeksio) e Instagram (@xgeeks);
- Patrocínio de eventos externos de desenvolvimento de software, tais como *require.lx* e *JNation*;
- Participação em eventos externos de elevada dimensão, tais como a *React Conference Live*, por parte de engenheiros da equipa;
- Realização de eventos públicos de desenvolvimento de software, como a Geekathon.

2.2. Organograma da empresa

Como referido anteriormente, a xgeeks faz parte do grupo KI, sendo o organograma da mesma constituído por membros de ambos.

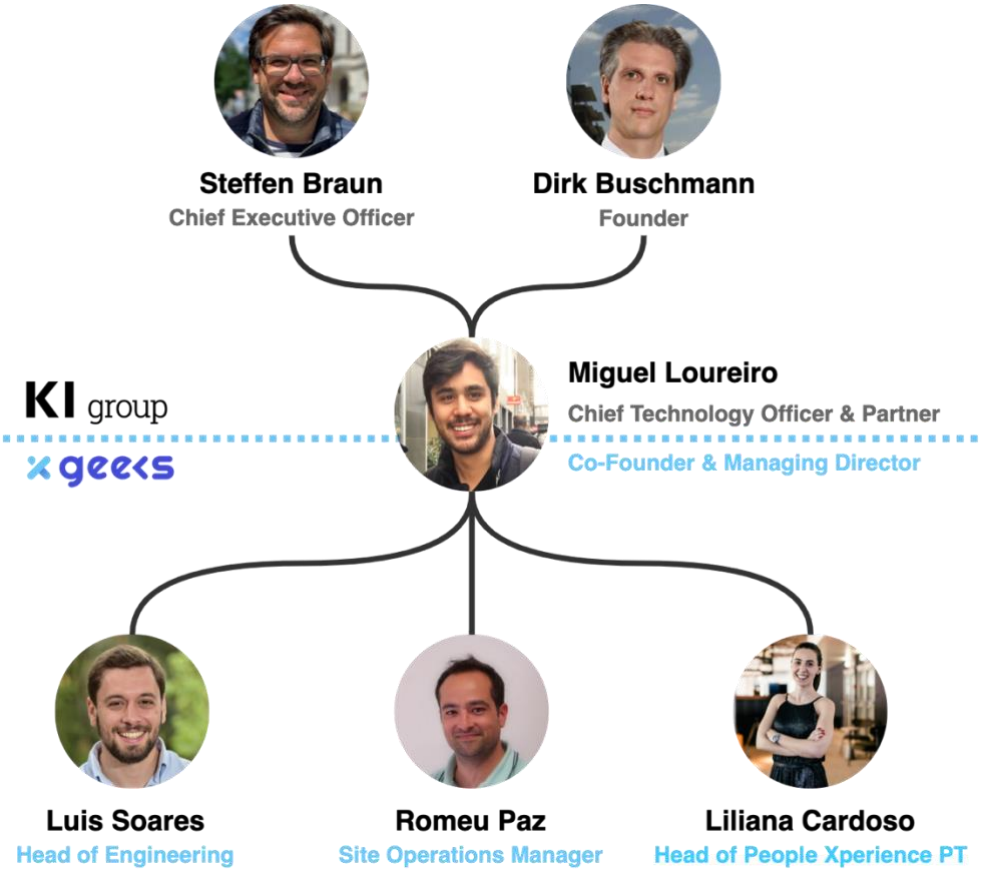

Figura 2 - Organograma do grupo KI & xgeeks

Na Figura 2 é possível verificar os elementos-chave da equipa xgeeks, tendo a grande maioria dos mesmos feito parte do meu processo de integração na mesma.

3. Programa de Estágio

A proposta efetuada pela entidade empregadora tinha como descrição "Participação em projetos dos nossos clientes com integração nas equipas existentes", como é possível observar no Anexo A: Proposta de Estágio. Dada a natureza genérica desta proposta, o programa de estágio foi maioritariamente definido consoante as necessidades dos clientes da empresa.

O processo de integração na empresa é de elevada organização, sendo constituído por um período de *onboarding*, no qual são introduzidos os vários componentes, práticas e ferramentas utilizadas pela empresa, seguido de um período de formação até a alocação do engenheiro a um projeto.

3.1. Integração

No período de *onboarding*/integração são designadas as primeiras tarefas do engenheiro. Durante o mesmo são também concedidos acessos a todas a ferramentas tecnológicas e plataformas utilizadas, introduzidos os projetos existentes, ideologias, alguma história da entidade, entre outras informações introdutórias da mesma.

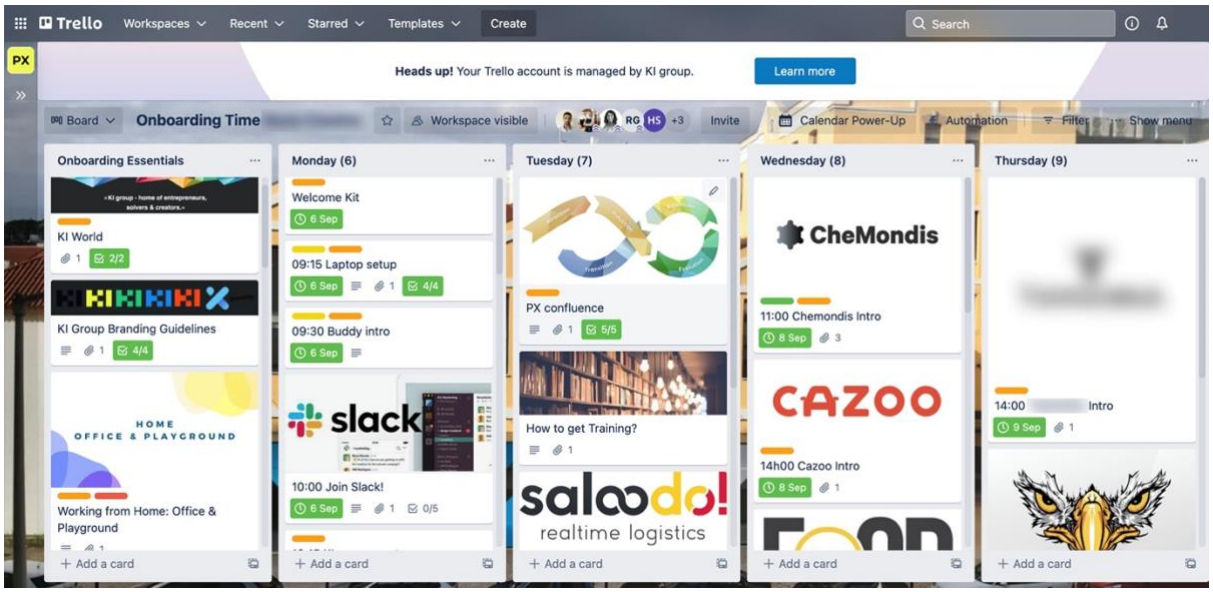

Figura 3 - Trello de *onboarding* de novos membros da equipa

Este período tem a duração de 1 semana e é acompanhado por um quadro de Trello com todas as tarefas a efetuar, reuniões a comparecer e o respetivo horário, como é possível observar na Figura 3.

De modo a facilitar a integração durante este período, é escolhido e atribuído ao novo membro um dos engenheiros já empregado pela entidade e com alguma experiência, denominado de *buddy*. Este é responsável por introduzir o espaço físico, neste caso, o escritório de Leiria, e apresentar outros membros da equipa. São também apresentados diversos canais de comunicação formais e informais, como os canais existentes no Slack da empresa, e um grupo de WhatsApp utilizado para outro tipo de convivências. De um modo geral, o principal papel do *buddy* atribuído aos novos membros é de os integrar e fazer papel de amigo, de modo a acolher o novo membro integrante.

3.2. Formação Tecnológica

Após o período de integração, é dada a oportunidade de usufruir de um período de formação numa determinada tecnologia, com acesso a diversas plataformas, e consoante o plano de alocação do engenheiro a um projeto. Este período varia consoante a necessidade do empregado e a brevidade da alocação do mesmo a um projeto. Por norma, a tecnologia sobre a qual o engenheiro faz formação, depende da previsão do projeto no qual o mesmo será integrado e do seu nível de experiência.

Durante este período, após ter sido informado do possível projeto onde viria a ser integrado, tive a oportunidade de efetuar diversas formações online na linguagem de programação Go e outras ferramentas comumente utilizadas na empresa, como Docker e Kubernetes. Esta formação teve como recurso as plataformas Udemy, Frontend Masters e Plural Sight, e teve uma duração de aproximadamente 4 semanas. Os cursos mais relevantes completados foram os seguintes:

- Udemy – Go: The Complete Developer's Guide (Golang) – 25/08/2020 - 9 horas
- Udemy – Kubernetes Certified Application Developer (CKAD) with Tests – 28/08/2020 – 9.5 horas

No entanto, devido à necessidade de alocação de novos engenheiros num outro projeto de elevada importância, após essas 4 semanas, fui informado de que seria integrado nesse mesmo

projeto. Neste projeto iria ter uma posição de programador *frontend*, utilizando mais especificamente a tecnologia React. Tendo em conta a minha experiência prévia com esta tecnologia, não houve tanta necessidade de formação, tendo efetuado apenas um curso online sobre a *framework* React, de modo a entrar no projeto mais familiarizado com a tecnologia.

3.3. Desenvolvimento

O processo de desenvolvimento decorre após a alocação do engenheiro empregado a um projeto da empresa, sendo caracterizado pelo desenvolvimento de software em conjunto com a equipa do projeto em questão.

O projeto em que fui colocado, ao qual me referirei como **projeto/cliente XYZ**, consiste num *marketplace* digital para o médio oriente, no qual fui introduzido como engenheiro de software, maioritariamente orientado a *frontend*. O período de integração neste projeto teve início no dia 23 de Setembro de 2020, sendo estreado por um período de 2/3 de dias destinado à familiarização com os objetivos, equipa, práticas e base de código, assim como a comparência nos eventos diários e semanais com a equipa. O processo de desenvolvimento foi iniciado a 25 de Setembro de 2020. A partir desta data, todas as tarefas realizadas não constam no programa de estágio previamente estabelecido pela entidade, sendo que o trabalho efetuado foi planeado consoante a necessidade do cliente, caracterizando o trabalho normal de um engenheiro de software contratado pelo mesmo.

Durante este período tive oportunidade de aplicar os diversos conhecimentos obtidos durante a licenciatura em Engenharia Informática, o Mestrado em Engenharia Informática – Computação Móvel, assim como nos cursos online efetuados durante o período de Formação Tecnológica. Mais especificamente, conhecimentos de desenvolvimento *web* gerais, noções básicas e avançadas de JavaScript, utilização de *frameworks*/bibliotecas de desenvolvimento *web* em JavaScript (e.g. Vue.js), CSS, HTML, entre outros. Estes conhecimentos foram cruciais durante o desenvolvimento do projeto em questão.

A descrição detalhada das tarefas efetuadas durante o período de desenvolvimento é efetuada no capítulo de **Processo de desenvolvimento**.

Em conclusão, apesar da falta de definição de objetivos e tarefas concretas para o plano de estágio ser algo que dificulta a avaliação e realização de relatório, este é um cenário bastante

comum no contexto de desenvolvimento de software. Isto é algo que acontece comumente na área de consultoria, pois os projetos variam, assim como os seus requisitos, e um engenheiro de software deve adaptar-se aos mesmos. No caso do estágio realizado, tendo em conta a proposta realizada pela xgeeks, o plano foi traçado não pela mesma, mas sim pelos requisitos e visão do cliente do projeto em que fui integrado, sendo estes descritos com maior detalhe mais abaixo, na secção de Requisitos.

4. Processo de desenvolvimento

Como referido, o projeto onde fui integrado teve como objetivo o desenvolvimento de um *marketplace* digital que tem como público-alvo a população do médio oriente, sendo inicialmente lançado em exclusivo para o Kuwait, um país situado no Golfo Pérsico. Na altura em que fui alocado ao projeto, este encontrava-se numa fase de desenvolvimento pré-lançamento, ainda com alguns processos de decisão a decorrer.

Figura 4 - Cronologia dos principais marcos do processo de desenvolvimento

O processo de desenvolvimento iniciou-se no dia 25 de julho de 2020, 2 dias após a integração na equipa existente, como é possível observar na Figura 4. Nesta figura encontram-se também os principais marcos de desenvolvimento dos 3 subprojetos em que tive oportunidade de trabalhar no projeto em questão, sendo estes os seguintes:

- ***Storefront*** – Principal projeto a ser desenvolvido. Consiste na principal plataforma *web* desenvolvida para o *marketplace* digital a ser criado;
- ***Seller Lab*** – Projeto interno que tem como objetivo o desenvolvimento de uma plataforma *web* para suportar as marcas e equipas que estão em contactos com as marcas comercializadas pelo *marketplace* no processo de gestão dos seus produtos;
- **Aplicação móvel** – Aplicação desenvolvida para as plataformas iOS e Android como complemento à plataforma *web* desenvolvida.

Na data de escrita desta secção, este projeto ainda se encontrava a decorrer (com a intervenção da xgeeks), no entanto, o período de estágio terminou em Maio de 2021, sendo esta a data considerada como final de desenvolvimento.

4.1. Requisitos

O levantamento de requisitos é uma das partes mais importantes do processo de desenvolvimento de software, pois é uma fase crucial para analisar e entender os objetivos do cliente, de modo a idealizar o produto final, de acordo com os mesmos. Apesar de, numa metodologia ágil, esta fase não ser uma das mais importante, devido ao constante envolvimento do cliente ao longo do processo de desenvolvimento, é sempre importante delinear os objetivos desde o início.

Embora não tenha feito parte do processo inicial de levantamento de requisitos, devido ao facto de ter sido integrado no projeto em questão já numa fase de desenvolvimento, fui inteirado dos principais requisitos levantados. Sendo estes os seguintes:

1. Desenvolvimento de um *marketplace* para o Kuwait (com a possibilidade de eventual expansão para os Emirados Árabes Unidos), com o objetivo de comercializar artigos de moda e ser uma referência na região do Golfo Pérsico;
2. Integração com marcas detidas pelo grupo responsável pelo projeto em questão;
3. Conteúdo em Inglês e Árabe;
4. Foco em dispositivos móveis.

No contexto de desenvolvimento de software, este conjunto de requisitos foram traduzidos para os seguintes:

⇒ Otimização da plataforma para motores de busca. Este é um requisito **quase obrigatório** no desenvolvimento de um *marketplace*;

1. Necessidade de utilização de uma solução de *e-commerce*, de modo a agilizar o processo;
2. Criação de um processo de integração automática de novas marcas;
3. Suporte para multilingue, com adaptação de conteúdo para línguas escritas da direita para a esquerda e a utilização de uma plataforma de tradução;
4. Utilização de uma *framework*/biblioteca *frontend* que permita uma boa otimização para dispositivos móveis e a facilite a criação de uma PWA (*Progressive Web App*).

Estes requisitos permitiram à equipa escolher um conjunto de tecnologias para iniciar o desenvolvimento. Fatores como a experiência da equipa e a data-limite para o lançamento de um Produto Viável Mínimo (*MVP - Minimum Viable Product*) foram também tidos em conta na escolha de tecnologias.

4.2. Tecnologias

O projeto XYZ foi desenvolvido com recurso a tecnologias *web* modernas, com especial foco na escalabilidade e otimização para motores de busca. Este capítulo descreve as tecnologias utilizadas no desenvolvimento das plataformas *web*, assim como na gestão de tarefas e documentação ao longo do projeto. Tendo desenvolvido apenas para *frontend*, os seguintes capítulos explicam mais detalhadamente as tecnologias utilizadas neste contexto.

4.2.1. TypeScript

TypeScript é um *superset open-source* da linguagem de programação JavaScript [1]. Esta é desenvolvida pela Microsoft desde 2012 e destaca-se pela adição de tipos de dados e outras ferramentas ao JavaScript, apelando aos programadores acostumados a linguagens fortemente tipadas (com tipos de dados), tornando uma das principais linguagens de desenvolvimento *web* mais verbosa e previsível. Uma das grandes vantagens desta linguagem é a facilidade de transição vindo do JavaScript, devido à sua inferência de tipos e permitir a utilização de tipos não específicos (e.g. any [2]), promovendo uma transição suave e incremental. No entanto, ao contrário do JavaScript, o TypeScript não é interpretado nativamente pelos *browsers*, tendo de passar por um processo de transpilação [3], em que é efetuada a conversão para JavaScript. Este processo é relativamente simples, e é efetuado pelo próprio compilador.

Diversas ferramentas e *frameworks* têm vindo a ser reescritas ou adaptadas para JavaScript, de modo a serem facilmente utilizadas em projetos desenvolvidos em TypeScript. Este processo é geralmente efetuado através da definição de ficheiros de declarações (.d.ts), ou através da definição de um *package*, instalável como qualquer outra biblioteca (e.g. utilizando um gestor de *packages* como o npm ou yarn), que define apenas os tipos utilizados na biblioteca. Os ficheiros de declarações funcionam de forma semelhante a um ficheiro de headers (.h) na linguagem C, e permitem especificar estruturas de dados, tipos de parâmetros e de valores de retorno de funções, entre outros. A definição de todos os tipos de uma biblioteca escrita em JavaScript pode ser feita com recurso a apenas um ficheiro, mantendo a biblioteca original intacta, sem necessidade de adaptar o código existente. Os *packages* de definição de tipos devem ser instalados em conjunto com aquele a que este corresponde, e geralmente possuem um prefixo @types (e.g. react [4] + @types/react [5]). A grande maioria destes *packages* de tipos pode ser encontrada no repositório comunitário DefinitelyTyped [6]. A

utilização destes é totalmente opcional, sendo possível usufruir de todas a funcionalidade de uma biblioteca sem saber os tipos que esta utiliza. No entanto, a utilização de bibliotecas não tipadas em projetos desenvolvidos em TypeScript é contraprodutivo e propício a *bugs*.

4.2.2. React

React é uma biblioteca JavaScript, *open-source* e utilizada em desenvolvimento *web* [7]. Foi lançada em 2013 pelo Facebook e é atualmente desenvolvida e mantida pelo mesmo e por uma vasta comunidade de indivíduos e empresas. É uma das bibliotecas mais conhecidas para desenvolvimento *frontend*, e destaca-se pela facilidade que oferece no desenvolvimento de SPAs e aplicações móveis, com recurso a linguagens e tecnologias *web*. Semelhante a outras bibliotecas de *frontend* (e.g. Vue,js), o React possui funcionalidades como a fácil gestão e renderização de estado no DOM (*Document Object Model*), controlando quando e o que deve renderizar. Para tal, esta utiliza um DOM virtual, de modo a ter maior controlo sobre os elementos visuais, atualizando-os de forma eficiente no DOM do *browser*. A declaração destes elementos visuais é efetuada com recurso a uma linguagem de *templating*, denominada de JSX (*JavaScript Syntax Extension*), que permite definir elementos HTML em JavaScript, e também a utilização direta de lógica de negócio para a apresentação dos mesmos. Atualmente, combinar React com TypeScript é um processo cada vez mais fácil, devido aos *packages* de declaração de tipos existentes [4.2.1].

Apesar de bastante completa, a biblioteca React foca-se apenas na construção de componentes de interfaces para o utilizador, não possuindo funcionalidades como roteamento de páginas, otimização de página estáticas, suporte para internacionalização, entre outros. Sendo necessário implementar manualmente tais funcionalidades ou recorrer a bibliotecas externas existentes. Estes tipos de funcionalidades são cruciais para aplicações em produção, especialmente num ambiente comercial, em que é importante otimizar as aplicações para motores de busca, minimizar tempos e tamanhos de carregamento, entre outros. Com o objetivo de incluir este tipo de funcionalidades num só conjunto, têm vindo a surgir diversas *frameworks* como o Next.js e Gatsby.

4.2.3. Next.js

Next.js é uma *framework* desenvolvida em JavaScript e TypeScript, sobre a biblioteca React [8]. Esta foca-se em facilitar o desenvolvimento de aplicações preparadas para ambientes de produção e otimizadas para motores de busca, através da aplicação de técnicas como roteamento baseado em sistema de ficheiros [9] e pré-renderização de páginas [10]. Ao contrário do que é comum nas aplicações desenvolvidas utilizando apenas React, as aplicações que utilizam esta *framework* não são, por defeito, uma SPA, pois cada rota é tratada como uma página diferente. Este é o paradigma original de desenvolvimento para a *web* e, ao contrário de SPA, permite uma melhor otimização para redes sociais e motores de busca, pois estes vêem uma aplicação com várias páginas distintas, e não apenas uma em que as páginas são totalmente geridas por JavaScript no lado do cliente (*browser*). Este comportamento é possível devido ao facto desta *framework* fazer a pré-renderização de páginas do lado do servidor, facilitando parte do trabalho que normalmente é efetuado pelo *browser*. Esta pré-renderização está disponível em duas formas, *SSG (Static Generation)* e *SSR (Server-Side Rendering)*, cada uma com as suas vantagens e desvantagens.

Static Site Generation (SSG)

Static Site Generation (ou apenas *Static Generation*) permite efetuar a geração de páginas em *build time*, ou seja, quando a *framework* é executada, reutilizando a mesma página em cada pedido efetuado pelo cliente, sendo semelhante à abordagem tradicional de desenvolvimento *web* em puro JavaScript (Vanilla JS) e HTML. No entanto, esta funcionalidade permite mais do que gerar páginas estáticas uma só vez, sendo também possível obter dados dinâmicos externos em *build time*, podendo também especificar um período de revalidação destes dados (e.g. voltar a gerar a página com novos dados a cada 1h) - ***Incremental Static Regeneration*** [11]. Esta funcionalidade é disponibilizada pelo Next.js através das funções `getStaticProps` e `getStaticPaths`, onde é possível executar um excerto de código JavaScript (e.g. pedidos assíncronos a uma API - *Application Programming Interface*) e utilizar quaisquer dados obtidos na geração da página.

Mais recentemente (Fevereiro de 2022), foi adicionada a possibilidade de manualmente despoletar a regeneração de uma determinada página, ao invés de especificar um tempo de revalidação. Esta funcionalidade abre a porta a inúmeras possibilidades de otimização de páginas (e.g. despoletar a regeneração de uma página estática apenas após a alteração do seu conteúdo num CMS).

Server-Side Rendering (SSR)

Server-Side Rendering ou *Dynamic Rendering* permite a geração, cálculo ou obtenção de dados dinâmicos em cada pedido efetuado pelo cliente, sendo ideal para páginas que apresentem dados provenientes de uma determinada fonte, e que sejam frequentemente atualizados. Tal como o SSG, esta funcionalidade é disponibilizada pelo Next.js através de uma função executada do lado do servidor - `getServerSideProps`. Esta prática deve ser devidamente acomodada, pois os pedidos de páginas renderizadas do lado do servidor tornam-se mais demorados, podendo resultar numa degradação da experiência do utilizador. De modo a evitar que tal aconteça, o código executado do lado do servidor deve ser pouco bloqueante, evitando sempre que possível chamadas a fontes não controladas, sendo também aconselhado um alojamento da aplicação em servidores devidamente capacitados.

4.2.4. CSS em JS e Styled Components

CSS em JavaScript é uma prática muito comum no desenvolvimento *web* moderno que permite a escrita e interpretação de CSS diretamente em código JavaScript/TypeScript. Uma das principais vantagens desta prática é a capacidade de manipulação e composição dinâmica de estilos utilizando variáveis, condições e ciclos, sendo mais fácil a geração de estilos em *runtime*, em vez de definir os mesmos separadamente em ficheiros CSS. Styled Components [12] é uma das mais comuns bibliotecas *open-source* que permite esta prática, sendo frequentemente combinada com React, facilitando todo o processo de escrita de estilos.

4.2.5. Contentful

Contentful é um *headless* CMS *(Content Management System)* de código fechado e disponibilizado como *SaaS (Software as a Service)*.

O que é um *headless CMS*?

Um CMS é um tipo de software dedicado à gestão e criação de conteúdo digital, normalmente, para a apresentação do mesmo numa página *web [13]*. Certos CMS permitem a criação de páginas *web* e a gestão do conteúdo das mesmas sem necessitar de extenso conhecimento técnico, utilizando interfaces interativas, que não requerem manipulação de

código fonte, como é o caso do WordPress. No entanto, quando nos referimos a um CMS como sendo *headless*, isto significa que o conteúdo gerido não se encontra acoplado a próprio sistema (e.g. geração de páginas *web*), gerando apenas dados sem um formato final definido. Isto faz com que seja possível utilizar o conteúdo em qualquer plataforma, tal como uma base de dados, utilizando métodos convencionais de obtenção de dados, como pedidos HTTP a uma API REST (*Representational state transfer*), com respostas em formato JSON (*JavaScript Object Notation*).

O Contentful disponibiliza todo o conteúdo gerido através de uma API REST [14], assim como o através do seu próprio SDK (*Software development kit*) para JavaScript [15], facilitando o processo de obtenção de dados.

4.2.6. Lokalise

O Lokalise é um serviço de gestão de traduções baseado em *cloud*. Este permite centralizar todos os pedaços de texto presentes numa aplicação numa só plataforma, de modo a serem traduzidos para outras línguas. Normalmente, este processo é efetuado por uma equipa dedicada de tradutores, sendo uma mais valia ter uma plataforma independente para tal, não sendo necessária a intervenção da equipa de engenharia. As traduções podem ser posteriormente integradas na aplicação em questão, através de diversos métodos disponibilizados pelo Lokalise. Estes métodos incluem bibliotecas disponibilizadas para diversas linguagens/plataformas, extração de um ficheiro comprimido (em diversos formatos), entre outros.

4.2.7. GraphQL

GraphQL é uma linguagem de código aberto para consulta e manipulação de dados em APIs, lançada em 2015 pelo Facebook, sendo uma das principais, mais eficientes e flexíveis, alternativas ao REST [16]. Esta linguagem oferece a habilidade de modelação de estruturas de dados utilizando *schemas* – uma descrição dos dados que os clientes podem pedir – sobre os quais são executadas *queries* (pedidos de informação) e mutações (alterações de informação). Esta utiliza tipos primitivos (*Scalar Types*) como String, Int, Float, ID e Boolean, que são a base de criação das estruturas de dados disponibilizadas. O GraphQL é também uma abstração

do protocolo HTTP, pois tradicionalmente, consumindo uma API REST, são efetuados pedidos a vários *endpoints*, utilizando vários métodos como GET, POST e PUT, enquanto numa API de GraphQL, é apenas utilizado um "*endpoint*" e o método POST. Isto deve-se ao facto de, ao contrário de uma API REST, a especificação do pedido encontra-se do lado do cliente e não do lado da API, sendo este que decide o que quer obter ou manipular, eliminando a necessidade de vários *endpoints* e métodos para especificar o tipo de operação a efetuar. Esta especificação da operação é feita através da inclusão da *query* ou mutação a efetuar no corpo do pedido, de acordo com a sintaxe definida pela linguagem (*QL – Query Language*).

Figura 5 - Exemplo de *query* GraphQL

Esta é uma das principais vantagens do GraphQL, do ponto de vista de um consumidor, pois permite a construção de pedidos (*queries*) à medida, ou seja, apenas com os dados necessários, como é possível observar na Figura 5. Por exemplo, se necessitar de obter um determinado produto de uma base de dados, cujo modelo possui 20 campos de dados, mas eu apenas preciso de 7 destes campos para apresentar na minha página, eu consigo escrever uma *query* que pede apenas estes 7 campos, resultado num campo de mensagem muito mais pequeno. Todos estes fatores, tornam o GraphQL uma escolha atraente para otimização de APIs, dada a possibilidade de encurtamento de mensagens, assim como pelo facto de oferecerem mais controlo ao consumidor, eliminando a necessidade de desenvolvimento de diversas operações que satisfaçam as necessidades do mesmo.

4.2.8. Git

Git é um sistema de controlo de versões distribuído usado maioritariamente no desenvolvimento de software [17]. Neste contexto, este sistema é útil para manter um histórico de alterações de código efetuadas por elementos de uma equipa de desenvolvimento, assim como realizar diversas ações sobre este mesmo histórico, prevenindo perdas e promovendo a agilidade de trabalho em equipa. O Git não é o único sistema de controlo de versões existente, existindo também outros sistemas, como o SVN (*Subversion*), no entanto o Git é o mais utilizado profissionalmente, especialmente por equipas distribuídas, e oferece um maior número de integrações, sendo a escolha mais acertada no presente contexto.

4.2.9. GitLab

GitLab é uma plataforma *open-source* de desenvolvimento de software que fornece funcionalidades como a hospedagem de repositórios Git e o seu controlo de versões, rastreamento de problemas, revisões de código, *CI/CD (Continuous Integration/Continuous Development)*, entre outras.

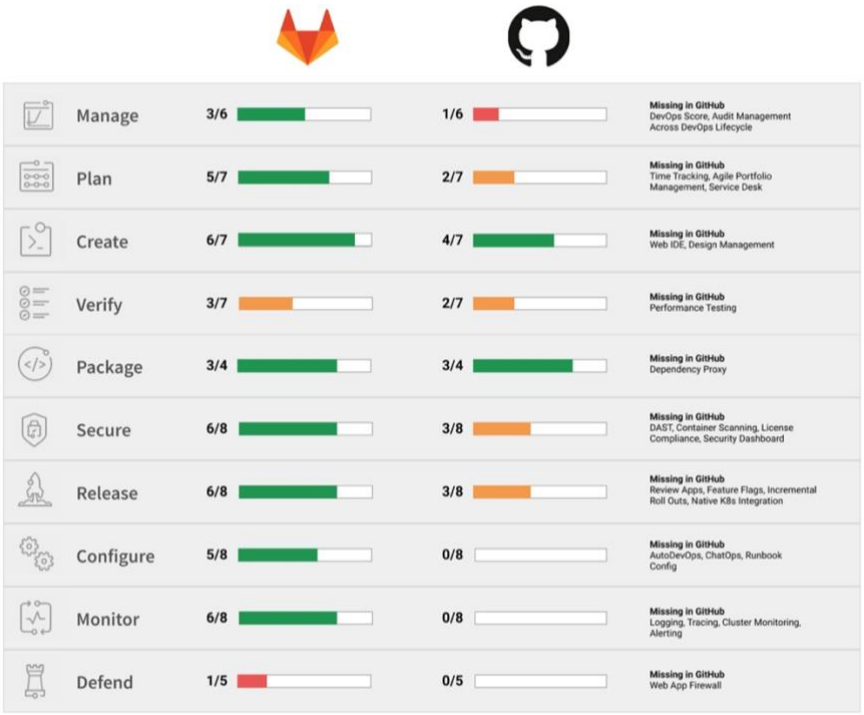

Figura 6 - Funcionalidades GitLab vs GitHub

Esta plataforma é comumente escolhida em contextos profissionais, pois oferece um conjunto de funcionalidades mais completo do que a sua principal concorrência, o GitHub, como é possível observar na Figura 6, especialmente em ferramentas de DevOps [18]. No contexto do presente projeto, o GitLab é utilizado para a hospedagem de todos os repositórios existentes, assim como para todo o processo de CI e CD.

4.2.10. Jira

Jira é uma plataforma de gestão de projetos, desenvolvida pela Atlassian, que permite a atribuição e monitorização de tarefas entre diversas equipas. Esta permite a criação de diversos projetos e a divisão de equipas em quadros que podem ser utilizados tanto numa metodologia Scrum como Kanban.

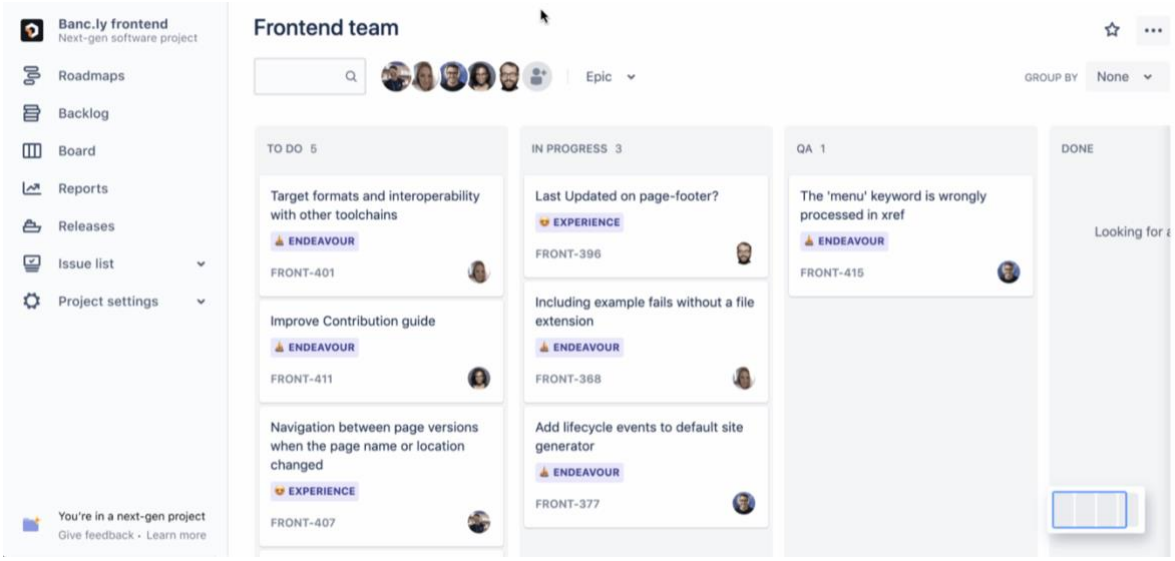

Figura 7 - Exemplo de um quadro Jira

Como é possível observar na Figura 7, cada quadro possui diversos cartões, que representam tarefas que podem ser rotuladas e classificadas consoante o seu tipo e importância, atribuídas a elementos da equipa, estimadas consoante o nível de esforço, entre outros. Esta plataforma possui também diversas integrações com outras ferramentas e plataformas como o GitLab, que permite o reconhecimento automático de identificadores de tarefas e conversão para hiperligação, a gestão do estado das tarefas (e.g. mover de “A fazer” para “Feito” ao abrir

um *Merge Request*), entre outras. Todas estas funcionalidades têm como finalidade oferecer uma melhor gestão e organização de projetos, mantendo um histórico do trabalho realizado.

4.2.11. Confluence

Confluence é uma plataforma *web*, desenvolvida pela Atlassian, utilizada para o armazenamento e organização de documentação. Esta oferece a possibilidade de criação de páginas de conteúdo rico (com formatação de texto e conteúdo multimédia), assim como a sua contextualização e organização utilizando espaços e pastas. Sendo uma plataforma da Atlassian, esta permite a integração com a plataforma Jira, permitindo a inclusão de identificadores de tarefas, de modo a obter hiperligações, assim como o seu estado atual de desenvolvimento. No contexto do presente projeto, esta plataforma é utilizada para documentar decisões tecnológicas (e.g. análise e escolha de um determinado serviço a subscrever), criar diversos guias de desenvolvimento (e.g. board práticas e estilo de código utilizado no projeto) e utilização de determinadas ferramentas, apoio a *onboarding* de novos membros numa determinada equipa, formulação de planos de lançamento de produtos, entre outros.

4.2.12. Outras tecnologias

Ao longo do desenvolvimento do projeto XYZ tive oportunidade de trabalhar ativamente com as tecnologias referidas acima. No entanto, foram utilizadas muitas outras mais, de igual ou maior importância, mais no âmbito de *backend* e infraestrutura.

Como tecnologias de *backend*, este projeto utilizou a *framework* Java Spring Boot em conjunto com a plataforma Broadleaf Commerce. Esta plataforma permitiu o rápido desenvolvimento de funcionalidades de comércio digital, devido às suas funcionalidades pré-concebidas para esta função.

Como tecnologias de infraestrutura, este foi alojado na Azure Cloud, sobre *clusters* de Kubernetes provisionados através de ficheiros de configuração Terraform.

4.3. Escolha de tecnologias

A escolha de tecnologias é um dos passos mais importantes de qualquer processo de desenvolvimento de software. Estas escolhas são importantes dado ao facto de impactarem diretamente o processo de desenvolvimento, as dificuldades enfrentadas e a escalabilidade do software a desenvolver. No contexto de *frontend*, durante este processo é escolhida a linguagem, a *framework* (caso de aplique), as principais bibliotecas, os serviços externos, entre outras tecnologias. São também tidos em conta fatores como os conhecimentos técnicos dos membros da equipa, o tempo de mercado (TTM - *Time To Market*), as tecnologias utilizadas noutros níveis (como o *backend*), o público-alvo e o volume de tráfego expectável e, acima de tudo, os requisitos do cliente. No contexto de um *marketplace*, é também tido em conta o tipo de produto e negócio cujo este software suporta (e.g. venda de peças de vestuário).

4.3.1. Tecnologias de *web* moderna

Os métodos de desenvolvimento de aplicações *web* têm vindo a mudar, sendo que, atualmente, são raros os casos que justificam o tradicional desenvolvimento de um *website* utilizando ficheiros estáticos de HTML e CSS. Isto deve-se maioritariamente ao facto de ser difícil e trabalhoso desenvolver aplicações dinâmicas, de forma rápida, utilizando apenas estas ferramentas. Atualmente, a tarefa de geração dos ficheiros interpretados nativamente pelos *browsers* (e.g. HTML, CSS e JavaScript) é delegada a *frameworks* e bibliotecas especializadas em traduzir sintaxes mais dinâmicas (e.g. JSX) e poderosas para estes. Como é o caso das bibliotecas React e Vue. Este tipo de sintaxe é geralmente uma combinação de HTML, CSS e maioritariamente JavaScript.

No contexto do projeto XYZ, indo de encontro à utilização de tecnologias de *web* moderna referida anteriormente, foi considerado um conjunto de pontos chave que influenciaram a opção pelas tecnologias referidas acima (4.2. Tecnologias). Pontos como a necessidade de fácil otimização da plataforma para motores de busca (SEO), adaptação para várias línguas, possibilidade de edição de conteúdo por parte de equipas externas (e.g. equipas de marketing), entre outros.

Framework de *frontend*

Um dos pontos mais importantes neste processo foi a otimização para motores de busca, que fundamentou em grande parte a escolha da *framework* Next.js. Esta *framework fullstack*

moderna oferece pré-renderização das páginas criadas por defeito e, manualmente, nos seguintes formatos:

- Renderização de páginas do lado do servidor (*Server-Side Rendering - SSR*);
- Geração de páginas estáticas (*Static Site Generation - SSG*);
- Regeneração estática incremental (*Incremental Static Regeneration ISR*).

Funcionalidades descritas em maior detalhe em 4.2.3. Next.js

A facilidade de utilização destas funcionalidades é uma característica que diferencia a *framework* Next.js das outras. Estas permitiram otimizar diversas páginas chave para motores de busca, algo que é de extremo valor no contexto de uma plataforma de comércio digital, pois a visibilidade em motores de busca tem um elevado impacto nos lucros obtidos. Tendo em conta que, atualmente, a grande maioria das aplicações *web* são construídas utilizando maioritariamente JavaScript, utilizando diversas abstrações que lidam automaticamente com a geração das páginas HTML, a técnica de pré-renderização é uma funcionalidade cada vez mais necessária.

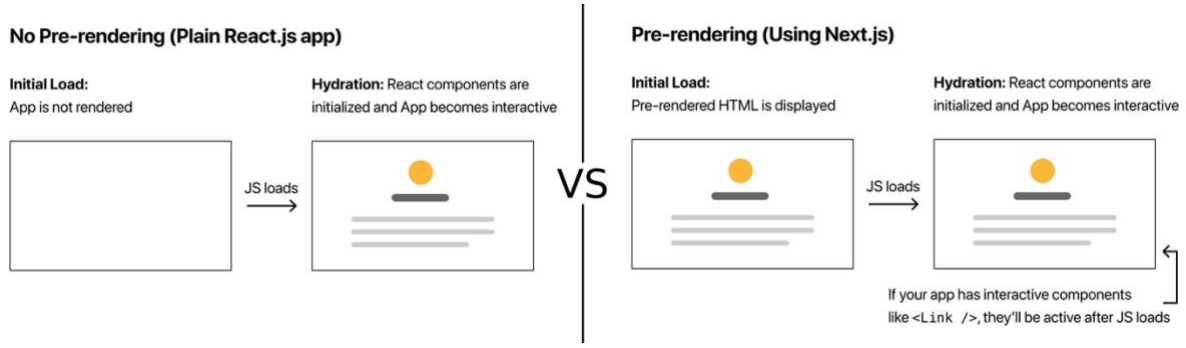

Figura 8 - Comparação Next.js sem e com *pré-renderização*

Esta técnica garante que a página é preparada, ou seja, o conteúdo gerado por JavaScript é pré-obtido, de modo a poder responder ao *browser* com uma página HTML com algum conteúdo relevante, e não apenas um esqueleto à espera de conteúdo dinâmico, como é possível observar na Figura 8. Isto faz com que os motores de busca consigam indexar conteúdo relevante, de modo a mostrar em resultados de pesquisas, e também com que as páginas sejam carregadas mais rapidamente, melhorando o feedback fornecido ao utilizador final.

No contexto do projeto XYZ, estes tipos de funcionalidades são úteis para diversos tipos de páginas. Por exemplo, começando logo pela página inicial de uma loja digital, onde são apresentadas promoções, produtos, categorias de produtos, entre outros. Por norma, este conteúdo não muda diariamente, sendo possível tornar a página estática com um período fixo de regeneração, diminuindo drasticamente o tempo de carregamento do principal ponto de entrada da aplicação. A vantagem da pré-renderização de páginas é óbvia no contexto de uma loja digital.

Adicionalmente, a *framework* Next.js possui uma numerosa comunidade e um largo conjunto de funcionalidades pré-incluídas e extensões que facilitam diversos processos, como a adaptação para várias linguagens (árabe e inglês, no caso do projeto XYZ).

Tendo em conta todos os pontos referidos anteriormente, é possível concluir que a escolha da *framework* Next.js foi uma das que mais agilizou o processo de desenvolvimento e melhorou a experiência da plataforma do projeto XYZ.

Gestão de conteúdo

Como referido anteriormente, e de modo a complementar a tradicional base de dados, houve a necessidade de permitir a edição do conteúdo apresentado na aplicação, por parte de uma equipa externa. Para tal, é geralmente utilizado um CMS (*Content Management System*). Estes sistemas atuam como uma base de dados de conteúdo rico e dinâmico, geralmente de fácil edição, através de uma interface amigável. Este conteúdo pode posteriormente ser obtido por aplicações num formato standard de troca de dados (e.g. JSON), geralmente no caso dos CMS *headless*.

No contexto do projeto XYZ, os requisitos existentes para esta plataforma não eram extensivos, os principais sendo o suporte para traduções, funcionalidades de publicação, draft, eliminação e arquivamento de conteúdo e disponibilização através de uma API REST. Opções como Directus, Strapi, Contentful e Broadleaf foram consideradas, tendo sido o Contentful a escolha final. Esta escolha baseou-se maioritariamente no bom suporte para traduções, tendo sido reforçada pelo facto de ser um SaaS, reduzindo o esforço de infraestrutura para disponibilizar uma aplicação *web* aos editores de conteúdo, assim como a existência de uma CDN para distribuição de imagens, suporte para pré-visualização de conteúdo, entre outros.

4.4. Metodologia de desenvolvimento

Como metodologia de desenvolvimento, foi maioritariamente utilizado um híbrido da *framework* ágil Scrum com o método Kanban ("Scrumban"). Este método desenvolvimento tinha como eventos o *Sprint Planning* (geralmente à sexta-feira) de *Sprints* de tempo variável (por norma de 2 semanas), *Daily Scrum*, ocasionais *Sprint Retrospective*, e outros como *Backlog Grooming*. Como artefactos, foram apenas gerados o *Product Backlog* e *Sprint Backlogs*, gerados pelos *Sprint Plannings*. Este projeto teve também responsáveis com os papéis de *Product Owner*, *Scrum Master* e equipa de desenvolvimento. Como referido, o método de desenvolvimento utilizado era híbrido, devido à necessidade de ter uma metodologia flexível e cumprir com os curtos prazos estabelecidos.

No último projeto para o cliente XYZ do qual fiz parte, o desenvolvimento da aplicação móvel, o método de desenvolvimento utilizado foi totalmente Kanban, devido ao facto de ser um projeto novo, havendo uma maior necessidade de flexibilidade, e poucos objetivos definidos.

4.4.1. Estratégia de ramificação e controlo de versões

No projeto em questão, é utilizada a ferramenta Git para controlo de versões. A estratégia utilizada para tal é semelhante ao GitFlow [19], utilizando 3 principais ramos de desenvolvimento - `master`, `staging` e `develop` – correspondentes aos diferentes ambientes de desenvolvimento e ramos de suporte para implementação de novas funcionalidades. São também utilizadas convenções para a efetuação de *commits* e criação de ramos.

Commits

Apesar de não ser uma regra obrigatória, devem ser efetuados *commits* frequentes, consoante o contexto da tarefa, separando estrategicamente cada *commit* em alterações/funcionalidades estáveis e corretamente identificadas, de modo a facilitar a leitura de histórico e a navegação entre *commits*. A mensagem de cada *commit* tem de seguir a convenção de *conventional commits* [20], que estrutura a mesma da seguinte forma:

```
<tipo>(escopo (opcional)): <breve-descrição>
[corpo do commit (opcional) ]
[rodapé (opcional)]
```

- Tipo – O tipo de *commit* varia consoante o código implementado e é inspirado pela convenção de Angular [21], podendo ter valores como **feat**, **fix**, **chore**, **refactor**, **test**, entre outros;
- Escopo – O escopo/*scope* do *commit* fornece contexto adicional à descrição. Este deve ser curto, normalmente apenas composto por uma ou duas palavras separadas por traços (e.g. shopping-bag);
- Descrição – A descrição é a parte mais informativa da mensagem de *commit*, e deve ser breve e sucinta e ter um formato imperativo ("*add feature*" em vez de "*added a feature")*. A primeira linha do *commit* é a mais importante, pois é que será destacada, devendo ser informativa e bem estruturada, de modo a facilitar a leitura e navegação na árvore de *commits*;
- Corpo e rodapé – No corpo e rodapé do *commit* podem ser incluídas outras mensagens secundárias, descrições contextuais ou qualquer outra informação relevante ao *commit* em questão. Este conteúdo é normalmente oculto em diversas ferramentas e plataformas (e.g. GitHub e GitKraken), sendo necessário selecionar ou inspecionar a mensagem para ler todas as linhas.

Estas regras impostas pela convenção utilizada são verificadas a cada *commit*, através da utilização de Git *hooks*, exceto quando é efetuada alguma ação para ignorar esta verificação. Os Git *hooks* são ações/*scripts* que são despoletados através de uma ação Git, como um *commit* ou um *push*. Estes são geralmente utilizados para efetuar verificações sobre o código que está a ser submetido (e.g. verificação de erros de sintaxe).

Figura 9 – Script *commit hook* utilizando a ferramenta husky (commit-msg)

Para a verificação da mensagem de *commit*, é utilizado um *commit hook*, que é configurado pela ferramenta `husky` [22], através de um ficheiro com o nome `commit-msg`, como é possível observar na Figura 9. Neste ficheiro é especificado um ou mais comandos a executar, no caso da Figura 9, é utilizada a ferramenta *open-source* `commitlint` [23], utilizando a configuração `config-conventional` [24], responsável por validar a mensagem de *commit*.

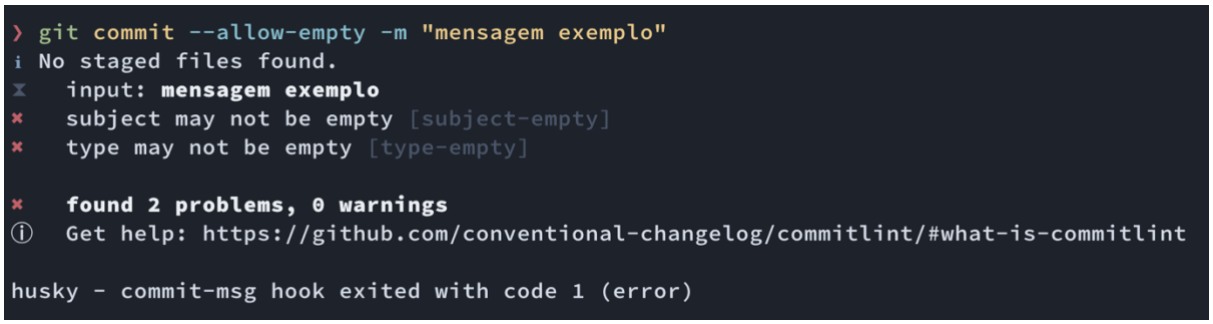

Figura 10 - Exemplo de um *commit* rejeitado pelo *commit hook* de validação de mensagem

Caso a mensagem não cumpra as regras de formato impostas, o *commit* é rejeitado, sendo necessário corrigir a mensagem para submeter com sucesso. Na Figura 10 é possível visualizar um exemplo de um *commit* rejeitado por este *commit hook*, por não cumprir o formato. Estas ferramentas são essenciais para manter a consistência de mensagens entre membros da equipa de desenvolvimento.

Ramificação

Como estratégia de ramificação/*branching*, com recurso à ferramenta Git, são utilizados ramos de suporte (e.g. *feature branches*), que têm como origem e destino os principais ramos de desenvolvimento, consoante o tipo de tarefa. Este tipo de ramos é caracterizado pelo seu

curto tempo de vida, sendo criados geralmente por um único indivíduo com o objetivo de implementar uma determinada funcionalidade ou corrigir um defeito na base de código de desenvolvimento. O nome dos ramos segue também um formato bem definido, para efeitos de consistência e legibilidade, sendo o seguinte:

- `[tipo-ramo-de-suporte]/JIRA-TICKET-breve-descrição`

Dependendo do ramo de origem/destino, os ramos de suporte podem ter os seguintes tipos:

- *Feature* e *Bug* – Com origem e destino no ramo de `develop`. Utilizados para desenvolver novas funcionalidades ou corrigir defeitos na base de código de desenvolvimento. Este tipo de ramo geralmente tem no nome um prefixo `feat/feature` ou `fix`. Ocasionalmente, também podem ser criados ramos que não correspondem a uma funcionalidade que seja visível pelo utilizador (e.g. mudanças de configurações ou melhoria de sintaxe), podendo ser utilizado um prefixo como `chore` ou `refactor`, inspirado na convenção de *conventional commits* [20];
- *Hotfix branches* – Estes têm como origem e destino no ramo `master` ou `staging`. São utilizados para corrigir defeitos diretamente na base de código estável e que se pode encontrar em produção. Este tipo de ramo é normalmente utilizado quando é necessário corrigir um defeito de elevada gravidade. Após a junção de código de um ramo `hotfix` com o ramo de destino, é também feito uma junção posterior para os ramos de desenvolvimento (e.g. `develop` e `staging`), de modo a não criar divergências de código. Este tipo de ramo geralmente tem no nome um prefixo `hotfix`.

O identificador do cartão/*ticket* Jira correspondente à tarefa atual deve também ser adicionado antes da descrição, caso exista. A descrição da mesma deve ser breve, sucinta, ter um formato imperativo e ser composta por palavras separadas por traços (“-“).

- Exemplos de nomes de ramos:
  - `feat/PAL-1234-add-user-login`
  - `hotfix/PAL-4321-fix-home-crash`
  - `chore/PAL-5678-setup-eslint-config`

Como referido anteriormente, estes ramos resultam num pedido de junção do mesmo com o respetivo ramo principal (e.g. `feat` é junto com `develop`), através da criação de um *pull*/*merge request*, que é submetido a um processo de revisão e aprovação (*code review*) por parte de um ou mais elementos da equipa, dependendo da preferência da equipa e da importância da tarefa. Este *merge request* deve também ter um título com o seguinte formato:

- `[JIRA-TICKET/NI] Breve descrição`

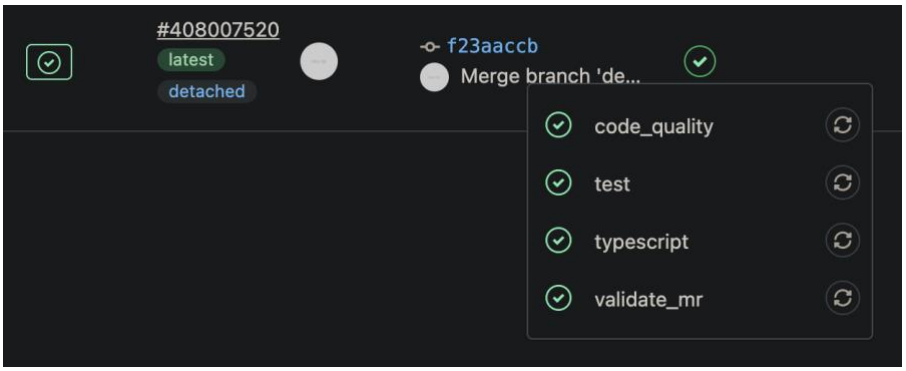

Figura 11 - Passos da pipeline GitLab CI executados num merge request

Semelhante ao formato utilizado para mensagem de *commit*, este também é verificado com recurso a um *script*, neste caso executado num passo da pipeline de GitLab CI (`validate_mr`), como é possível observar na Figura 11. Este *script* é responsável por simplesmente verificar se o título do *merge request* começa pelo número identificador da tarefa Jira ou NI (*No Identifier*).

4.5. Desenvolvimento da plataforma *Storefront*

O objetivo deste projeto consistiu no desenvolvimento de uma plataforma *web* para um *marketplace*, com o objetivo de comercializar produtos, maioritariamente de moda.

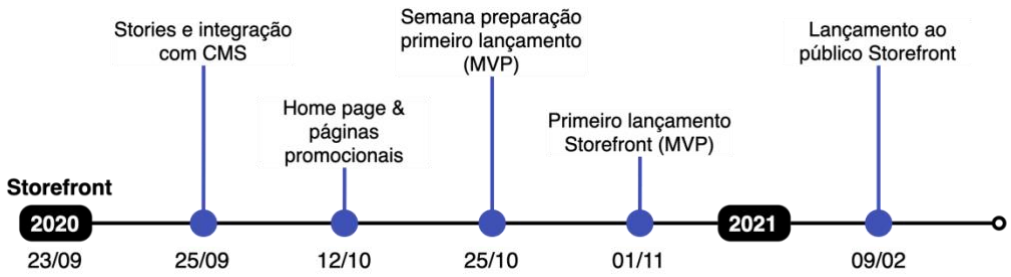

Figura 12 - Cronologia dos principais marcos do processo de desenvolvimento do projeto *Storefront*

O processo de desenvolvimento após a integração na equipa iniciou-se a 25 de Setembro de 2020, como é possível observar na Figura 12. De modo a familiarizar-me com as tecnologias utilizadas e com a base de código desenvolvida até à altura, foram-me atribuídas tarefas com um moderado nível de abrangência tecnológica, que me permitiram investigar diversos componentes já desenvolvidos.

Esta plataforma utilizou a seguinte pilha tecnológica de *frontend*:

- **TypeScript** – Linguagem utilizada para desenvolvimento *frontend*, em conjunto com a *framework* Next.js;
- **Next.js (React)** – Principal *framework* utilizada para o desenvolvimento *frontend* da plataforma *web*;
- **CSS em JS e Styled Components** – Biblioteca de CSS em JS utilizada para desenvolver todos os componentes React presentes na plataforma *web*;
- **Contentful** – CMS (*Content Management System*) utilizado para o armazenamento e distribuição de conteúdo na plataforma *web*.

4.5.1. Integração com um CMS

Uma das primeiras tarefas realizadas no projeto em questão, foi a integração com um CMS (*Content Management System*). Tradicionalmente, o conteúdo de uma página *web* é controlado pela equipa de engenharia, sendo necessário conhecimento técnico para alterar ou criar novo conteúdo numa página *web*. Esta integração teve como objetivo permitir a criação e organização do conteúdo apresentado em diversas páginas da *Storefront*, por parte de equipas especializadas, como editores de conteúdo, especialistas de marketing, ou qualquer outra equipa. A utilização de um CMS para este efeito é ideal, pois, geralmente, estas plataformas

permitem a edição de conteúdo através de uma *interface* mais visual, sem a necessidade de desenvolvimento de código.

A plataforma selecionada para o efeito foi o Contentful, um *SaaS Headless CMS* de código fechado. A escolha desta plataforma foi maioritariamente inspirada pelo completo suporte de internacionalização e bom *workflow* de agendamento e publicação de conteúdo, algo que é vantajoso para a equipa de editores.

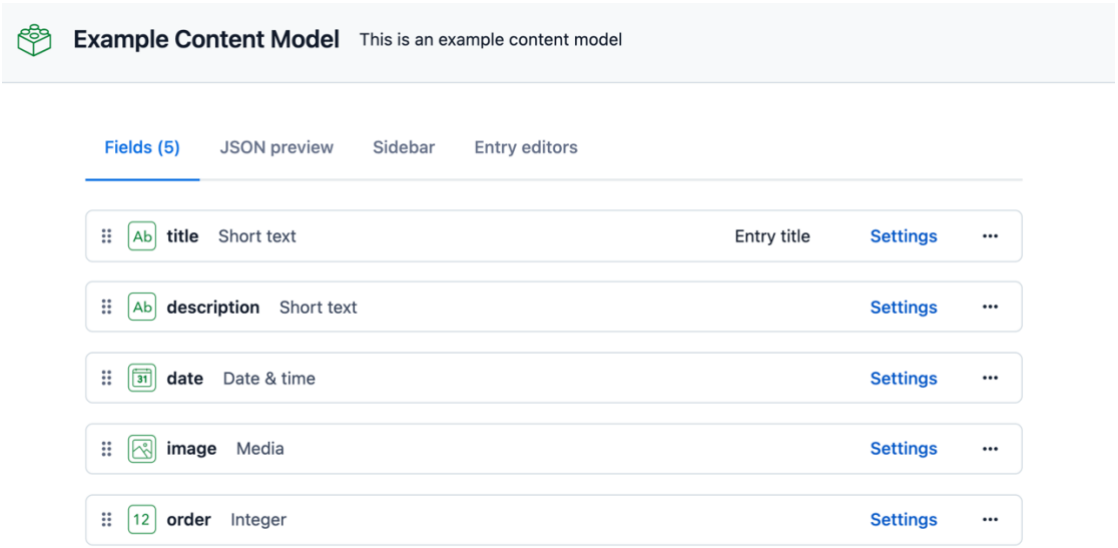

Figura 13 - Página de edição de um *content model* no Contentful

A criação de conteúdo no Contentful é efetuada através da construção de modelos de dados - *Content Models* [25] - com um conjunto de propriedades e um tipo associado, que podem ser posteriormente preenchidos com os dados adequados, como é possível observar na Figura 13. Semelhante a uma base de dados estes campos podem ter diversos tipos de dados, tais como texto, números, valores booleanos, imagens, tipos mais dinâmicos como *arrays* e JSON, entre outros [25]. É também possível criar relações complexas, através de campos de referências, que permitem a inclusão de um determinado modelo noutro. Estes campos são especiais, pois o conteúdo do modelo associado é incluído no corpo do conteúdo do modelo pai, semelhante a um *join* numa base de dados relacional.

Obtenção de conteúdo do CMS

Como referido anteriormente, o Contentful é um CMS *headless*, tendo naturalmente uma forma de obtenção/distribuição de conteúdo, neste caso, um SDK. Assim sendo, a integração do serviço com o projeto foi efetuada através da utilização do SDK oficial para JavaScript, que por si utiliza a API REST também disponibilizada pela plataforma [14]. Esta permitiu obter qualquer conteúdo criado na mesma, assim como utilizar mecanismos de paginação, ordenação, manipulação de resultados de pesquisa (*querying*), entre outros.

```
import { createClient } from 'contentful';

const client = createClient({
  space: process.env.CONTENTFUL_SPACE_ID,
  accessToken: process.env.CONTENTFUL_TOKEN,
  // optional:
  environment: process.env.CONTENTFUL_ENV, // default: master
  host: process.env.CONTENTFUL_HOST, // default: cdn.contentful.com
});

const data = await client.getEntries({
  content_type: 'example'
});
```

Figura 14 - Excerto de código de configuração e utilização do SDK do Contentful em JavaScript

Este processo de obtenção de dados no *frontend* foi relativamente fácil de configurar devido à utilização do SDK, sendo apenas necessário especificar o *token* de autenticação, o identificador do espaço a ser utilizado e, opcionalmente, o nome do ambiente e o endereço do *host*, entre outros parâmetros [26], como é possível observar na Figura 14. Após a configuração, os pedidos à API podem ser efetuados utilizando os métodos disponibilizados, como o `getEntry` e `getEntries` [27], como é possível observar na Figura 14.

Pré-visualização de conteúdo do CMS

Uma das funcionalidades fornecidas pelo Contentful é a pré-visualização de conteúdo antes de ser publicado. Devido ao facto de este ser um CMS *headless*, não estando associado a nenhuma aplicação em específico, esta funcionalidade é possível através da associação de um URL a cada tipo de conteúdo.

Figura 15 - Contentful: Associação de URLs de *preview* a tipos de conteúdo

Como é possível ver Figura 15, cada tipo de conteúdo pode ter um URL (*Uniform Resource Locators*) associado para pré-visualização. No caso de um tipo de conteúdo dinâmico, que possa precisar um campo do Contentful no URL, como um ID (e.g. página de detalhes de produto), é possível especificar o nome deste campo através da utilização de chavetas (e.g. https://example.com/preview/product/{entry.id}).

No contexto de uma aplicação *web*, esta funcionalidade é possível através da utilização de um diferente *host* e token de autenticação ao efetuar os pedidos à API, ambos disponibilizados pelo Contentful para esta função em específico.

```
export const CONTENTFUL_CONFIG = {
  space: process.env.CONTENTFUL_SPACE_ID,
  environment: process.env.CONTENTFUL_ENV,
  accessToken: process.env.CONTENTFUL_CDN_TOKEN,
  host: 'cdn.contentful.com',
};
```

```
export const CONTENTFUL_CONFIG = {
  space: process.env.CONTENTFUL_SPACE_ID,
  environment: process.env.CONTENTFUL_ENV,
  accessToken: process.env.CONTENTFUL_PREVIEW_TOKEN,
  host: 'preview.contentful.com',
};
```

Figura 16 - Excerto de código de configuração do modo preview do Contentful

Como é possível observar na Figura 16, a configuração utilizada para obter conteúdo público (do lado esquerdo), utiliza o endereço `cdn.contentful.com`, ao contrário do modo de pré-visualização, que utiliza o endereço `preview.contentful.com` e um *token* de

autenticação diferente, para obter conteúdo público e não público. Esta configuração permitiu disponibilizar o modo de pré-visualização através de um ambiente de desenvolvimento, com um endereço público, de modo a ser acessível pela equipa responsável pela edição de conteúdo.

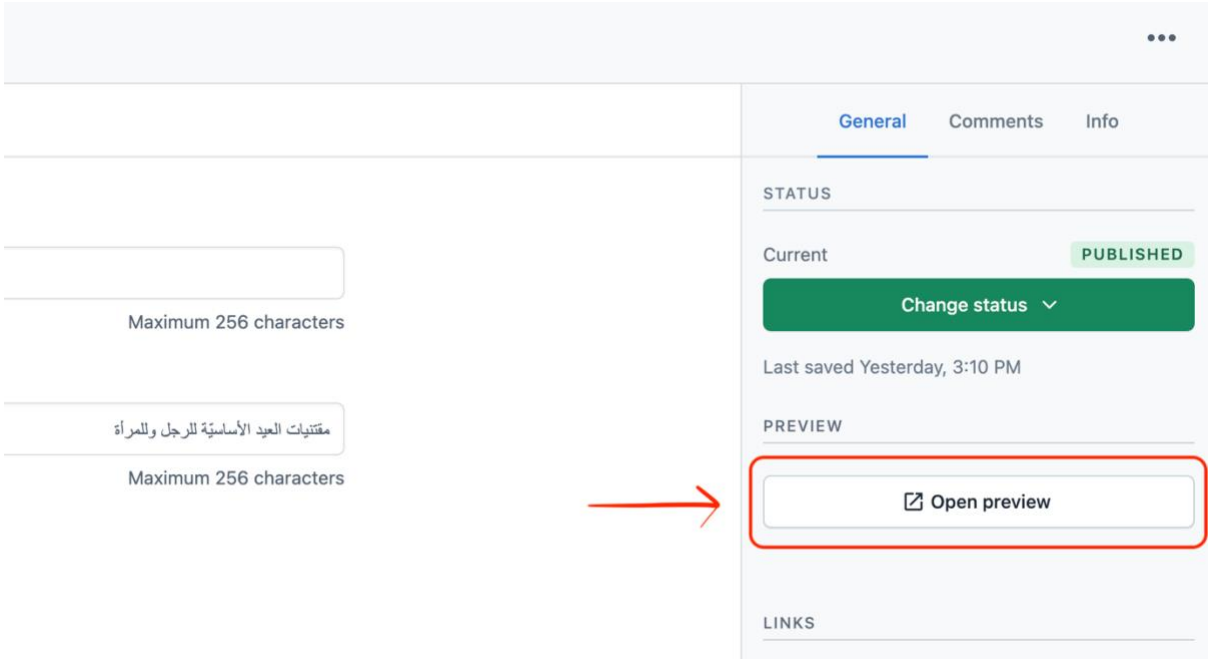

Figura 17 - Botão de pré-visualização de conteúdo Contentful

O conteúdo que tem um lugar previsível na aplicação, como é o caso das Páginas de *stories*, possui um botão de "Abrir pré-visualização" no modo de edição do Contentful, como é possível observar na Figura 17. Este botão redireciona o utilizador para o respetivo conteúdo num ambiente de desenvolvimento protegido (e.g. development-xyz.com), onde o modo de pré-visualização está ativo. Enquanto o editor não publicar o conteúdo, este não se encontra disponível no ambiente de produção (xyz.com). A implementação desta funcionalidade foi de elevado valor para as equipas de edição de conteúdo, pois a alteração de conteúdo em produção pode ser arriscada, conseguindo assim pré-visualizar o conteúdo antes de o publicar.

Abstração de plataforma

A mudança de CMS numa fase avançada de desenvolvimento de um software é algo indesejável, mas que pode acontecer a qualquer altura de um projeto, seja por razões monetárias ou falta de funcionalidades necessárias. De modo a reduzir o esforço de uma eventual migração, é importante seguir um conjunto de práticas que permitem a abstração da plataforma

que fornece este serviço (e.g. Contentful). Ao longo do desenvolvimento do projeto em questão, esta foi uma preocupação tida em conta pela equipa, que utilizou os seguintes métodos para o efeito:

- Mapear o resultado obtido pelos pedidos HTTP efetuados ao CMS para os próprios modelos, evitando sempre lidar com campos estabelecidos pelo *payload* das respostas do mesmo. Com esta abordagem, a mudança para outro CMS requer apenas mudanças no código de mapeamento;
- Evitar mencionar o nome do CMS no código (e.g. nomes de variável `cmsData` em vez de `contentfulData`).

A migração de um CMS numa fase avançada de um projeto é algo indesejável, mas, caso aconteça, estas pequenas práticas ajudam no processo.

4.5.2. Páginas de *stories*

Desenvolvida em conjunto com a tarefa de Integração com um CMS, esta consistiu no desenvolvimento de um conjunto de páginas, denominadas de *Stories* que têm como objetivo apresentar texto e imagens publicitárias de determinados produtos comercializados pela loja. Estas páginas podem ser utilizadas para destacar determinadas marcas, coleções de produtos para uma determinada estação ou ocasião, produtos utilizados por determinadas celebridades, entre outros.

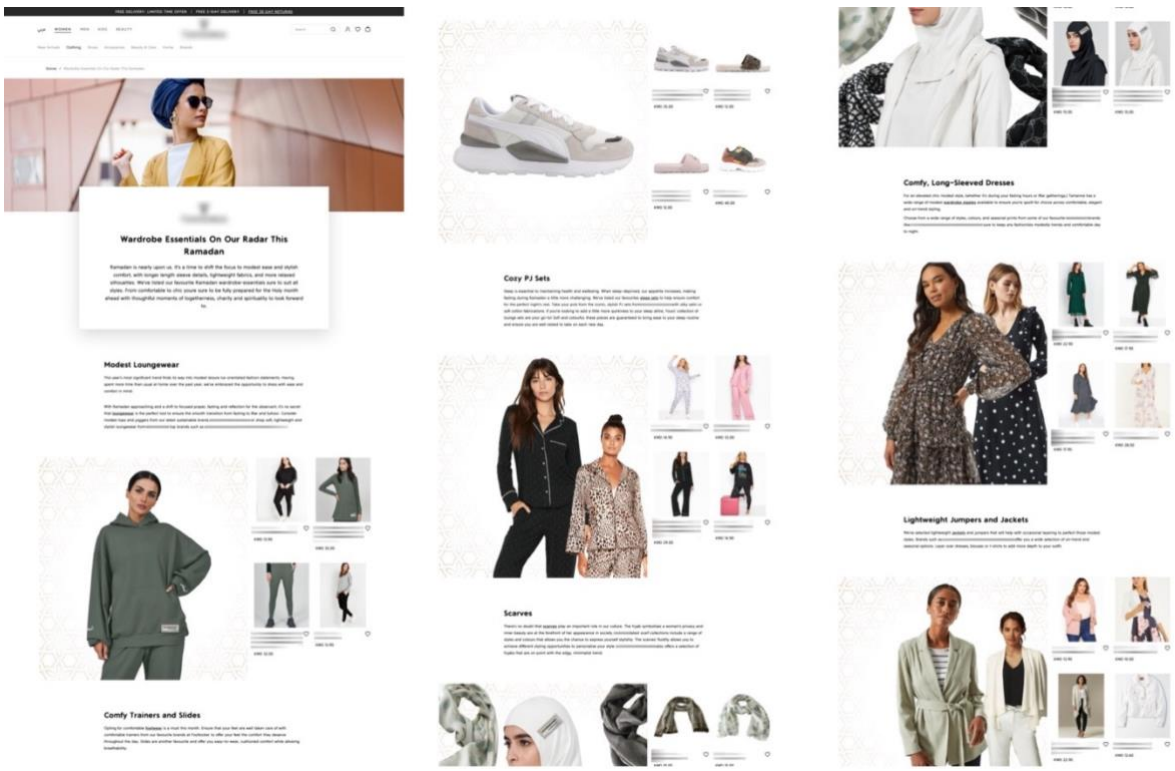

Figura 18 - Exemplo de uma página *Story* com o primeiro formato

Na Figura 18 é possível ver um exemplo de uma *Story* na qual são publicitados produtos para uma determinada ocasião, neste caso para o Ramadão.

Estas páginas foram inicialmente desenvolvidas em **dois formatos** distintos que permitem compor uma história com excertos de texto, imagens e os produtos a publicitar. Estes formatos, que partilham componentes, foram desenvolvidos em conjunto com outro elemento da equipa, a quem foram atribuídas tarefas semelhantes.

Formato/Template 1

O primeiro formato, presente na Figura 18, foi desenhado com o principal objetivo de publicitar produtos através da apresentação de imagens que demonstrem a utilização dos mesmos, geralmente peças de vestuário utilizadas por modelos ou outras personalidades.

Figura 19 - Componente "Story Look" de uma Story com o primeiro formato

Como é possível observar na Figura 19, estas imagens são normalmente acompanhadas pelos produtos em questão, do lado direito, com os quais o utilizador pode interagir, adicionando-o à sua lista de favoritos, ou clicando para ser redirecionado para a respetiva página do produto. Este componente, que agrupa uma imagem demonstrativa e os respetivos produtos, foi denominado de *Story Look*, devido ao facto de geralmente apresentar um *look* ou visual de um(a) determinado(a) modelo. A gestão de conteúdo deste componente foi organizada de uma maneira simples, sendo apenas necessário introduzir os IDs dos produtos a apresentar no CMS. Estes Ids, são depois traduzidos em produtos, através de um pedido efetuado à API de *e-commerce* (*backend*).

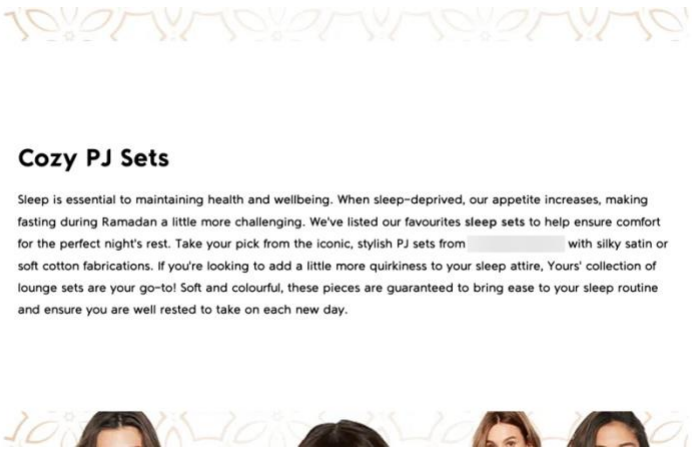

Figura 20 – Componente de bloco de texto de uma *Story* com o primeiro formato

Ao utilizar este formato, além do componente *Story Look*, é também possível incluir blocos de texto formatável, como é possível observar na Figura 20. Estes são blocos de texto livre e são geralmente utilizados para enquadrar os produtos numa determinada história, mencionar marcas, entre outros. Este componente, tal como o anterior, é formatável, sendo possível editar o texto apresentado através do CMS. O texto introduzido pode ainda incluir símbolos de *markdown*, que são posteriormente interpretados pelo código de *frontend* e traduzidos para os respetivos elementos HTML, com recurso à biblioteca `react-markdown` para o efeito [28]. Esta funcionalidade permitiu à equipa de conteúdo incluir mais do que simples blocos de texto, mas também links, texto formatado (negrito, itálico, etc), usar diferentes tamanhos de letra, entre outros.

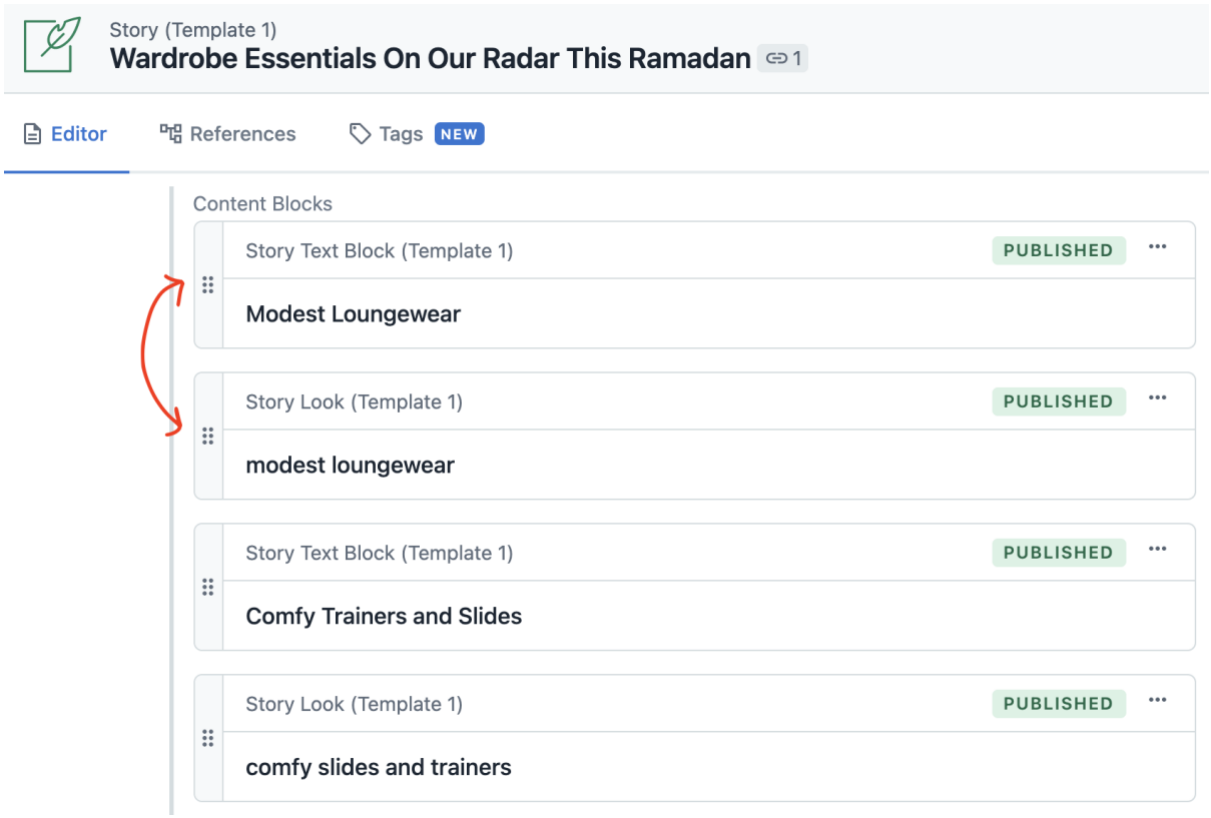

Figura 21 - Gestão dos blocos de conteúdo de uma Story através do Contentful

De modo a promover a modularidade do formato, no ato de criação de conteúdo através da interface *web* do Contentful, é possível gerir a disposição dos diferentes blocos, alternando entre texto e imagens com produtos, como é possível observar na Figura 21. Esta funcionalidade trouxe alguma modularidade a esta página, pois o seu conteúdo não teria uma ordem fixa, permitindo ao editor intercalar entre componente, consoante necessário.

Formato/Template 2

O segundo formato de *Story* foi desenvolvido com o objetivo de incluir um maior número de produtos (em formato de lista), sem necessidade de inclusão de muitas imagens ou texto de apresentação. Apesar deste formato ter sido desenvolvido maioritariamente por outro elemento da equipa, possui elementos comuns a ambos os formatos, nomeadamente o cabeçalho da página, o componente de um produto individual, o interpretador de *markdown* para os blocos de texto, os componentes responsáveis pela obtenção de dados do CMS, entre outros.

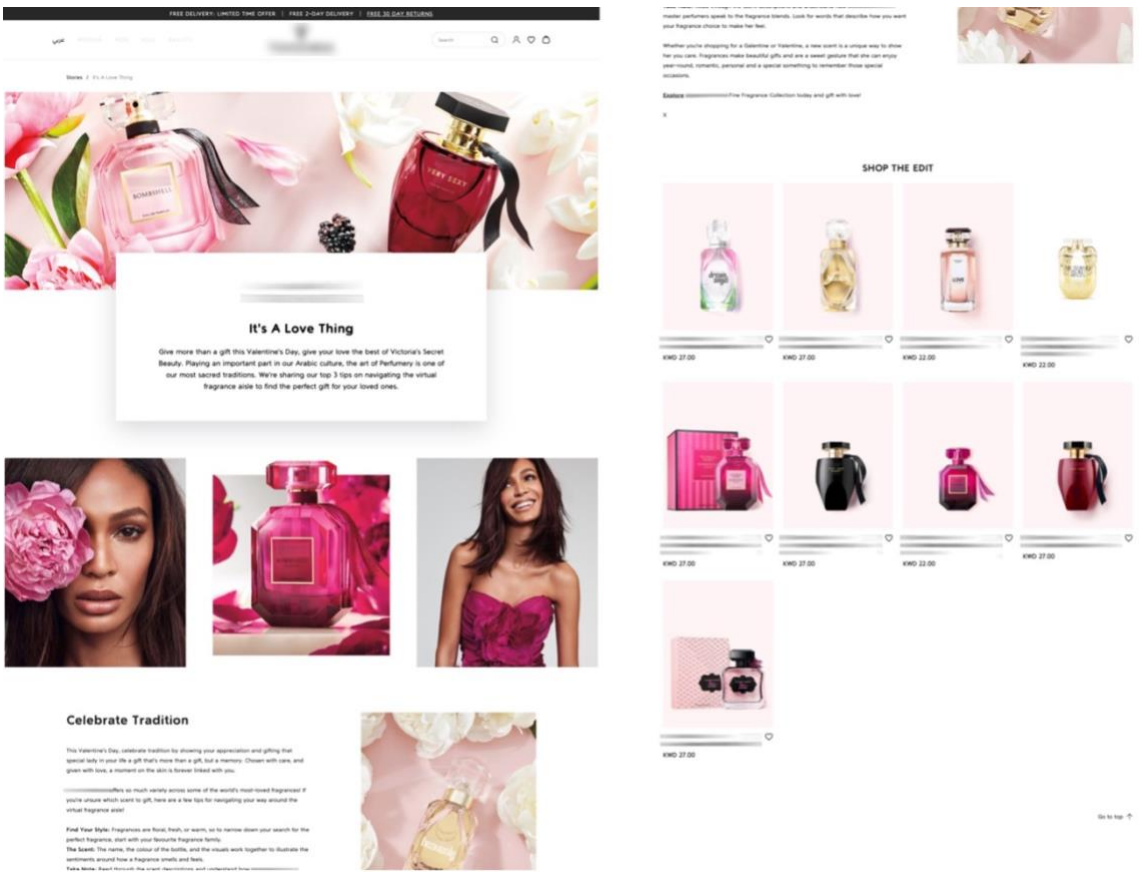

Figura 22 - Exemplo de uma página Story com o segundo formato

Este formato é comumente utilizado para a apresentação de um grande número de produtos numa lista, como é possível observar na Figura 22, não existindo limite para tal, ao contrário do primeiro formato, que permite um máximo de 4 produtos por bloco, acompanhado por uma imagem. Este é também composto por uma imagem tipo *banner* no topo, assim como um pequeno campo de texto introdutório, acompanhado por uma única imagem lateral.

Estas características fazem com que este segundo formato seja mais fácil de desenvolver (por parte da equipa de marketing), sendo ideal para uma grande coleção de produtos que necessite de ser destacada rapidamente.

Todo os elementos das páginas de *Stories* foram também cuidadosamente desenvolvidos, de modo a se adaptarem corretamente a dispositivos móveis

4.5.3. Página inicial e promocionais

A página inicial de um *marketplace* é uma das mais importantes, pois deve reunir uma boa seleção de conteúdo, de modo a cativar o maior número de visitantes possível e a aumentar a taxa de conversão (métrica utilizada para medir o desempenho de ações específicas num *website* de comércio digital). Assim sendo, o conteúdo desta página deve ser controlado por uma equipa especializada em marketing. Para tal, foi necessário integrar esta página com o CMS já utilizado no projeto, o Contentful.

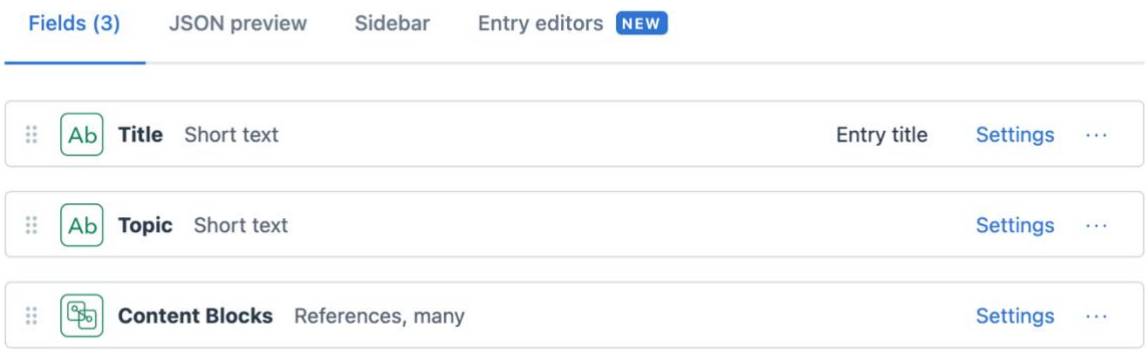

Figura 23 - Contentful Content Model do conteúdo da página inicial

A abordagem seguida para esta página assemelha-se à das Páginas de *stories*, em que os vários componentes de conteúdo são separados em blocos, permitindo aos editores decidir a disposição dos elementos da página, assim como escolher o componente que mais se adequa ao conteúdo que pretendem apresentar. Na Figura 23 é possível visualizar o modelo criado para esta página, sendo este bastante simples, apenas com 3 campos, agregando os vários blocos de conteúdo no campo “Content Blocks”.

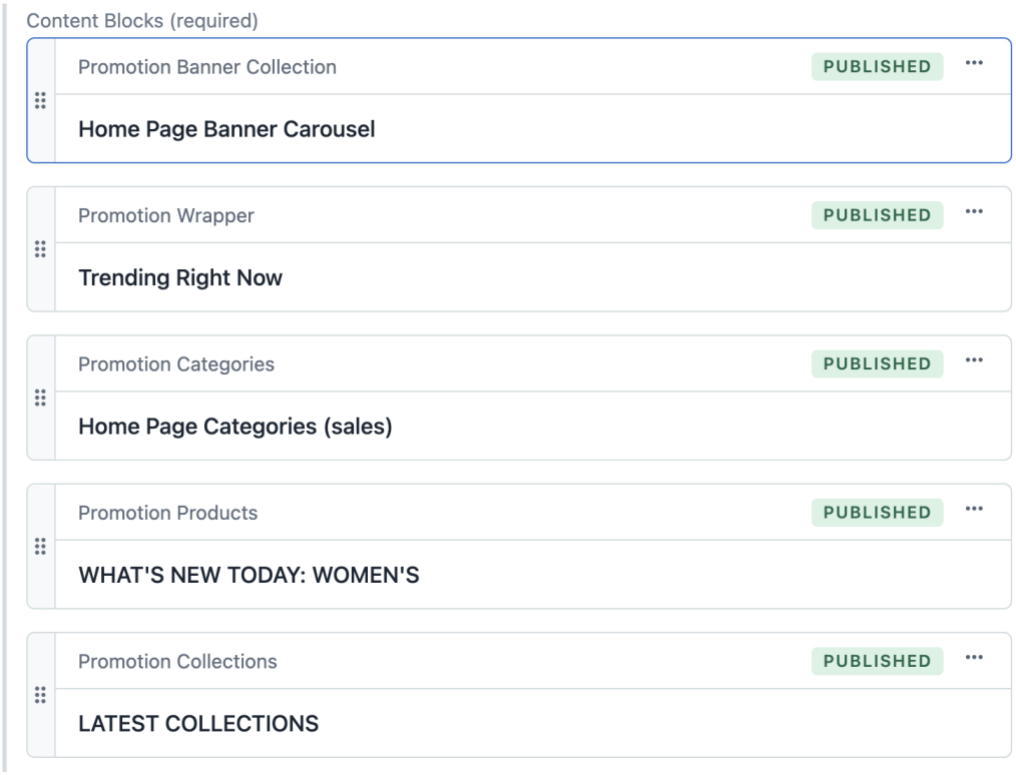

Figura 24 - Conteúdo da página inicial gerido no Contentful

Como é possível visualizar na Figura 24, estes vários blocos de conteúdo correspondem a componentes como uma coleção de imagens tipo *banner*, um *carousel* de elementos a destacar, entre outros. Estes diversos componentes têm todos o objetivo comum de estrategicamente promover, consoante a estação, ocasião ou oportunidade, produtos e coleções, marcas, publicações de *Stories*, categorias, entre outros. Estes dados são depois traduzidos em componentes e elementos visuais no *frontend*, compondo dinamicamente uma página completa.

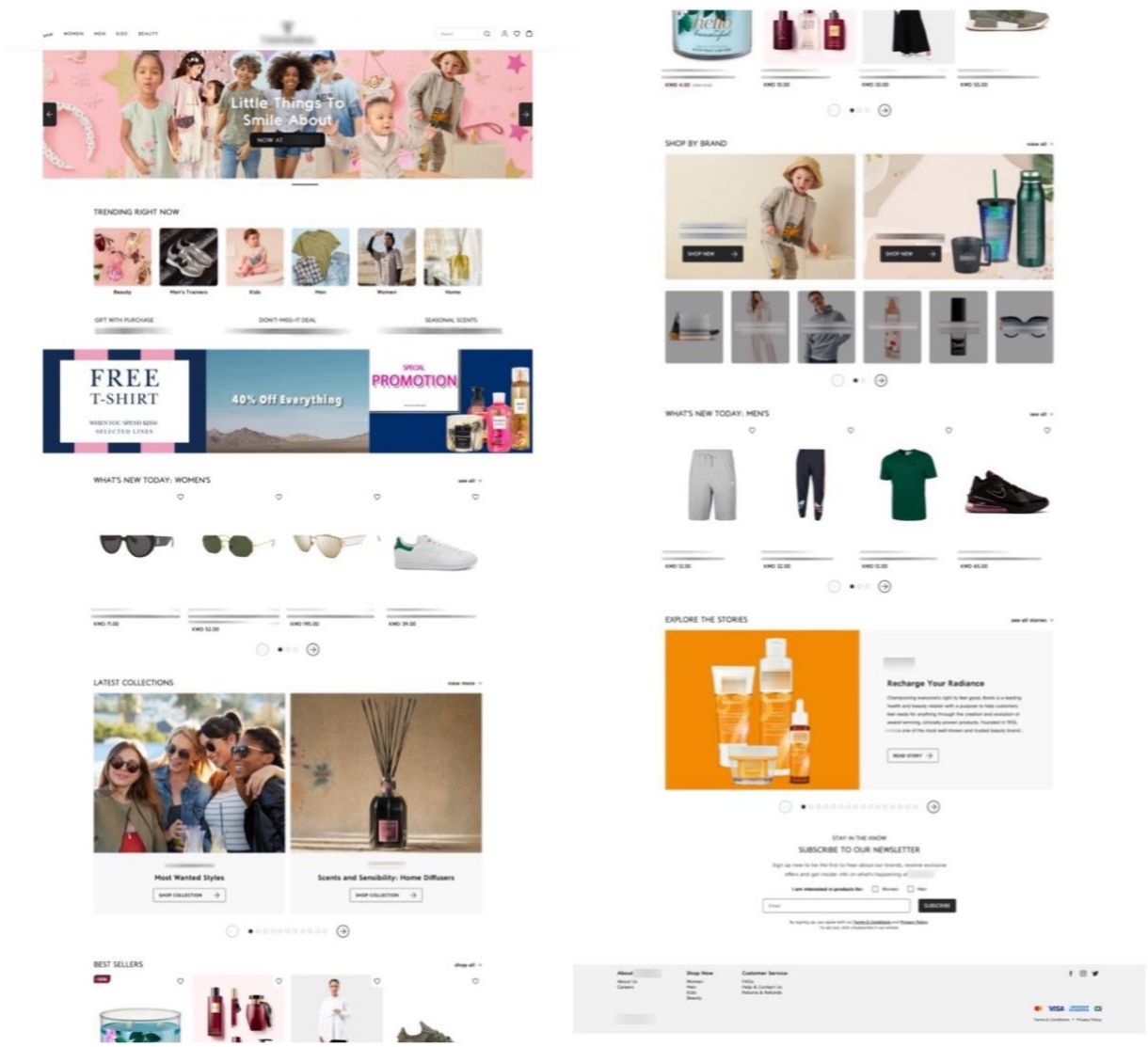

Figura 25 - Página inicial da Storefront

Na Figura 25 é possível visualizar uma página inicial totalmente composta pelos componentes referidos anteriormente. Cada componente tem o seu propósito e nesta página podemos observar alguns dos quais tive oportunidade de desenvolver e/ou reestruturar.

Banner

Este componente é normalmente posicionado no topo da página, sendo o principal elemento visual utilizado para chamar à atenção do utilizado, publicitando os produtos de maior interesse consoante a ocasião.

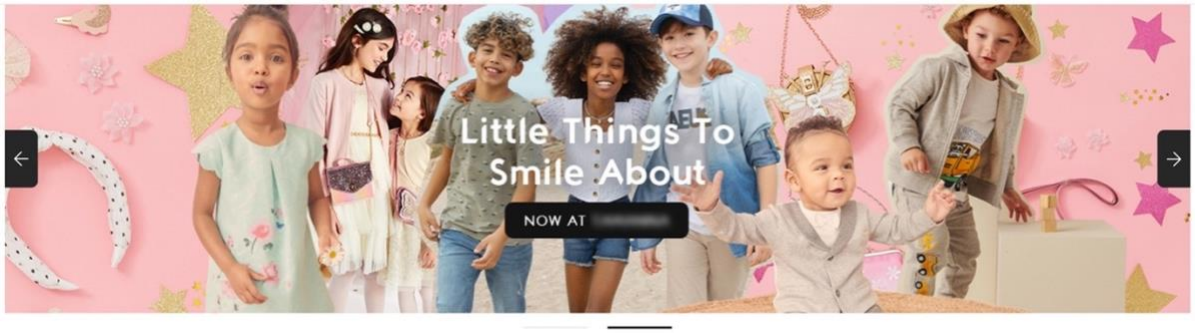

Figura 26 - Banner da página inicial da Storefront

É composto por um *carousel* de imagens interativo, em que é possível interagir com as mesmas, assim como com os elementos de navegação (visíveis na Figura 26), de modo a visualizar as várias imagens, ou aguardar pela transição efetuada pela reprodução automática.

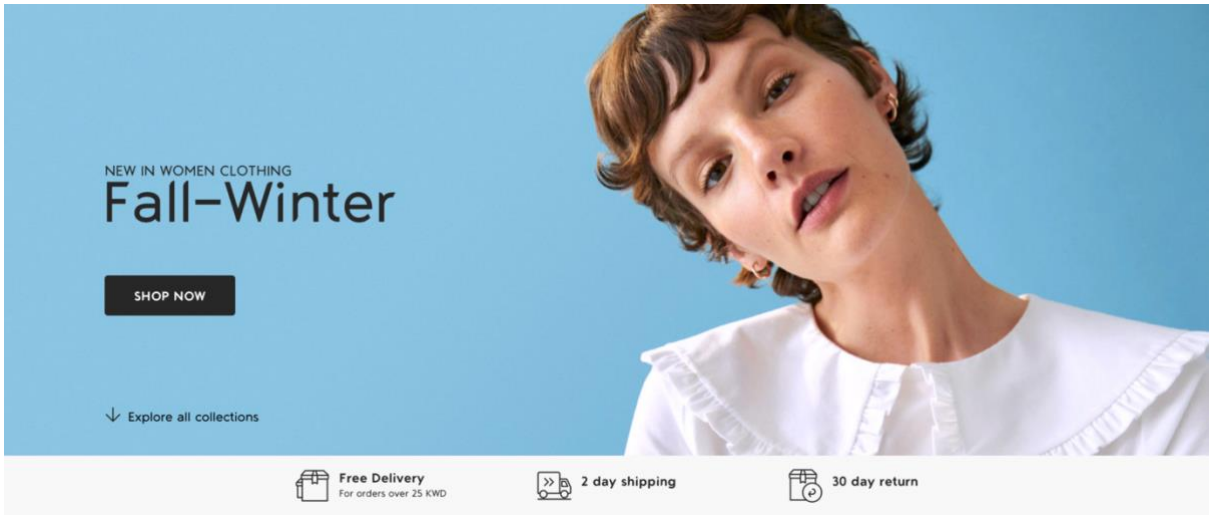

Figura 27 – Versão anterior do Banner da página inicial

Este elemento visual sofreu diversas alterações ao longo do processo desenvolvimento, a fim de ter mais controlo sobre a apresentação e posicionamento de elementos clicáveis. Na Figura 27 é possível ver um design anterior deste elemento, em que os diversos elementos têm uma posição fixa, limitando as possibilidades dos editores de conteúdo na escolha de imagens. Este design foi alterado para o que é possível ver Figura 26, em que toda a imagem é selecionável, deixando o posicionamento dos elementos ao critério dos editores. Tendo em conta que a plataforma em questão é também disponibilizada na língua árabe, houve uma

necessidade de tomar especial atenção a este componente, devido ao facto dos componentes *carousel* tenderem a apresentar comportamentos inesperados, quando sujeitos a uma alteração de direção. Os desafios enfrentados na adaptação deste tipo de componentes para uma alteração de direção são descritos com maior detalhe em Desafios (4.8).

Carousel de produtos

O *carousel* de produtos é essencialmente uma galeria de produtos interativa. Este é um dos componentes mais estratégicos da página, pois publicita um conjunto de produtos, sendo importante estudar os produtos a apresentar.

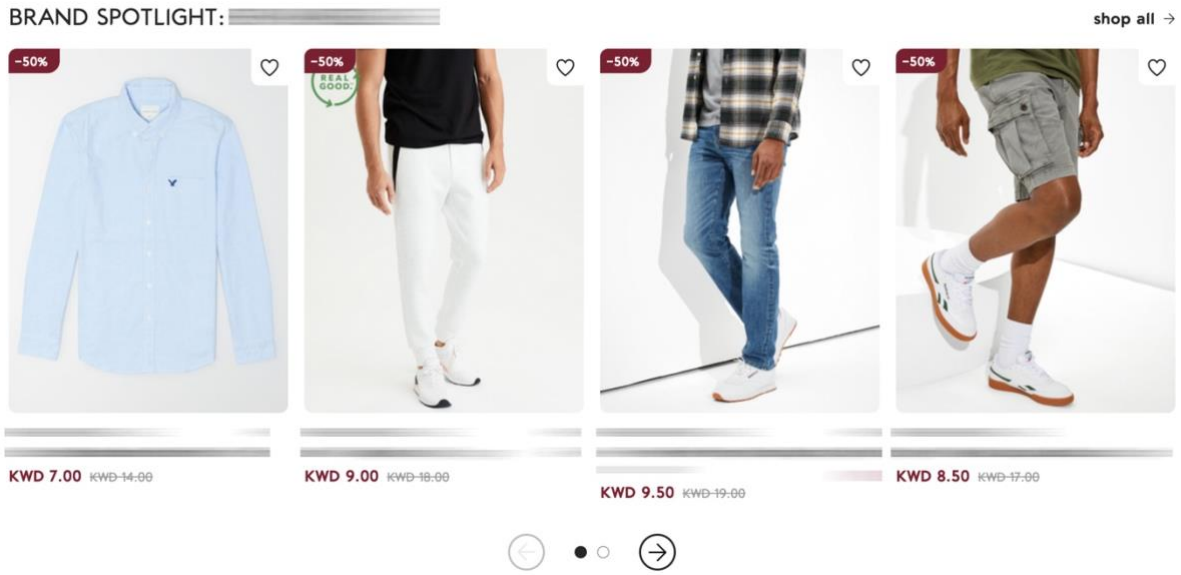

Figura 28 - Carousel de produtos

Como é possível observar na Figura 28, este componente é constituído por um título do lado esquerdo, um link/elemento clicável do lado direito, vários slides com um grupo de produtos e elementos de navegação em baixo. A grande maioria destes elementos é personalizável, sendo possível escolher o título, o texto e o destino do link e os produtos a apresentar, tudo através do CMS Contentful. Os produtos, correspondendo a itens reais e adquiríveis em loja, são especificados através de códigos SKU (*Stock Keeping Unit*) no CMS

e posteriormente cruzados com dados provenientes da API, de modo a adquirir a informação de cada um, a fim de apresentar campos como a o nome, marca, preços, imagem, entre outros.

Figura 29 - Animação de carregamento de esqueleto do *carousel* de produtos

Durante o período de desenvolvimento, tive oportunidade de complementar este componente de diversas maneiras, tais como:

1. Implementação de *lazy loading* com uma animação de carregamento de esqueleto, como é possível observar na Figura 29, o que contribuiu para uma melhor experiência do utilizador e redução de *layout shifts* [29];
2. Apresentação de uma percentagem de desconto e preço anterior de cada item;
3. Desacoplamento de uma página em específico e integração do componente com o CMS. Inicialmente este componente encontrava-se associado apenas à página inicial e presente numa posição específica.

Carousel de coleções

Este componente tem como objetivo apresentar diversos cartões que publicitam coleções de produtos. Coleções estas que possuem produtos selecionados pela equipa de marketing e visam relacionar os mesmos através da associação a uma determinada ocasião, estação, marca, causa, entre outros.

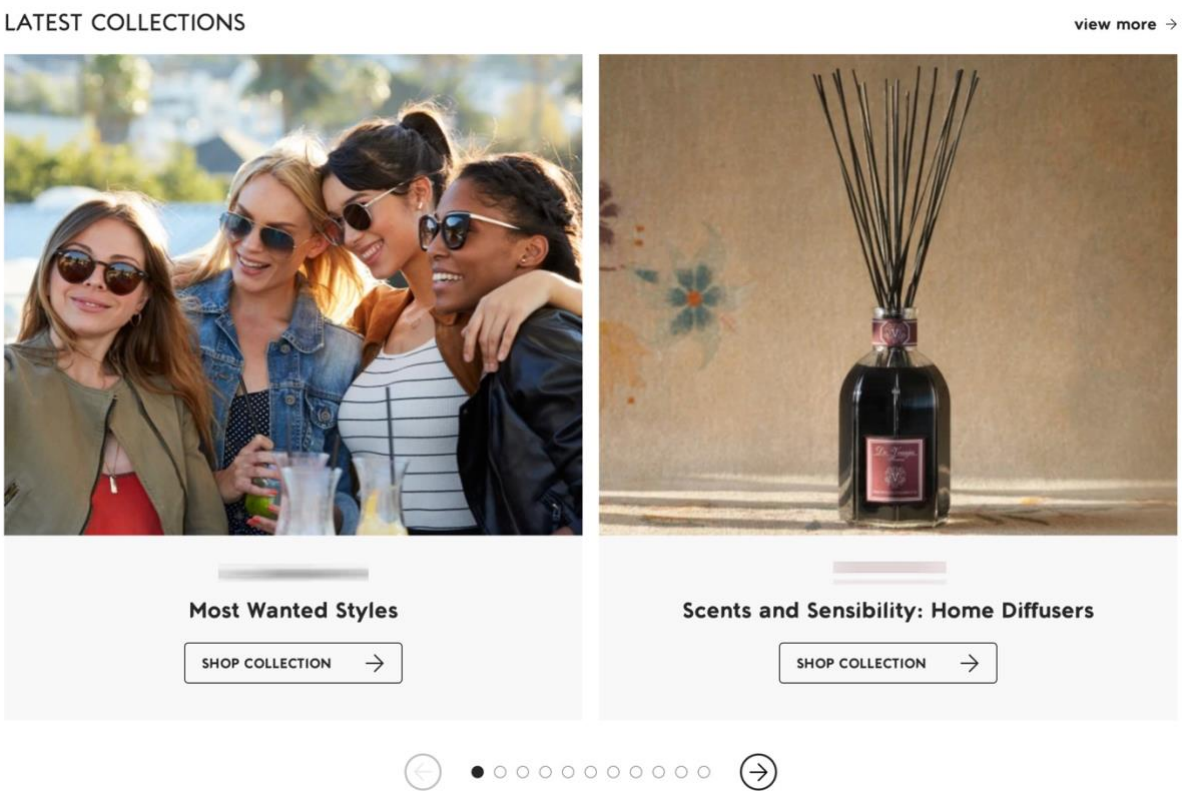

Figura 30 - Carousel de coleções de produtos

Neste componente é utilizada a mesma base de *carousel* presente no Carousel de produtos, como é possível observar na Figura 30, promovendo a reutilização de componente, assim como a consistência da interface e experiência do utilizador. Semelhante ao carousel de produtos, todos os elementos de texto, à exceção do texto do botão ("*SHOP COLLECTION*"), assim como as imagens, são totalmente personalizáveis através do CMS Contentful.

Carousel de stories

Este componente tem como objetivo agrupar e destacar cartões que correspondem a Páginas de *stories* de *stories* publicadas. Tem também como base o mesmo componente de *carousel* utilizado no Carousel de produtos e Carousel de coleções.

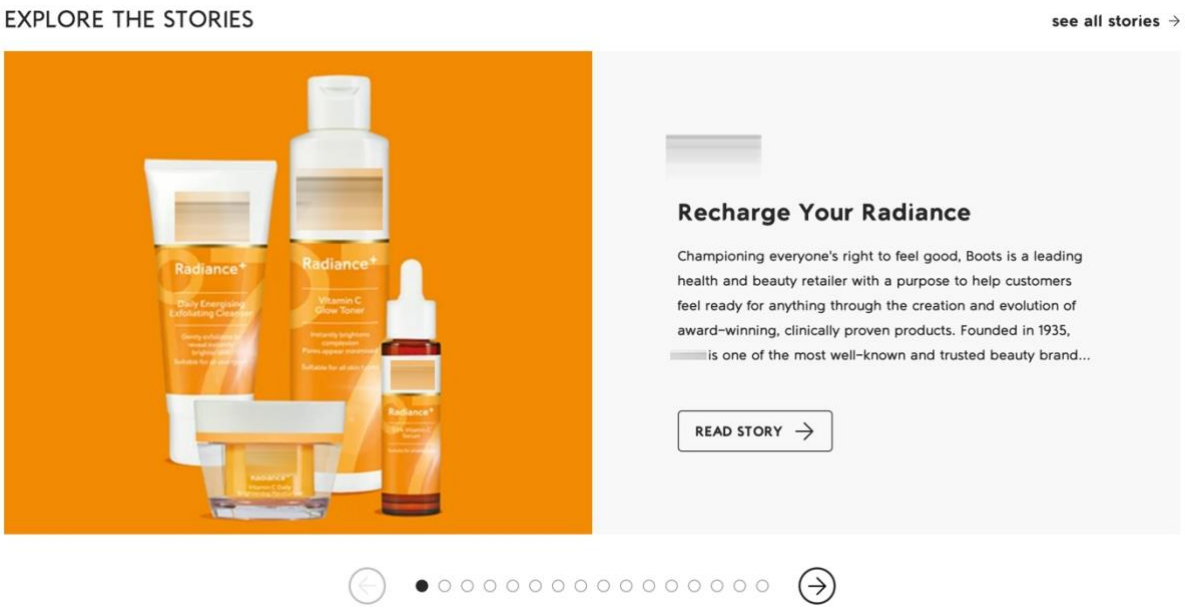

Figura 31 - Carousel de stories

Como é possível observar na Figura 31, os cartões agrupados por este *carousel* apresentam um breve resumo do conteúdo presente na *Story* em questão, de modo a cativar o utilizador, a fim de levar o mesmo a aceder à página completa desta *Story*.

Páginas promocionais de categorias

Um dos requerimentos da tarefa de desenvolvimento da página inicial, além da modularidade de componentes, possibilidade de ordenação dos mesmos e total controlo através do CMS, era a capacidade de reutilizar este tipo de página em vários contextos.

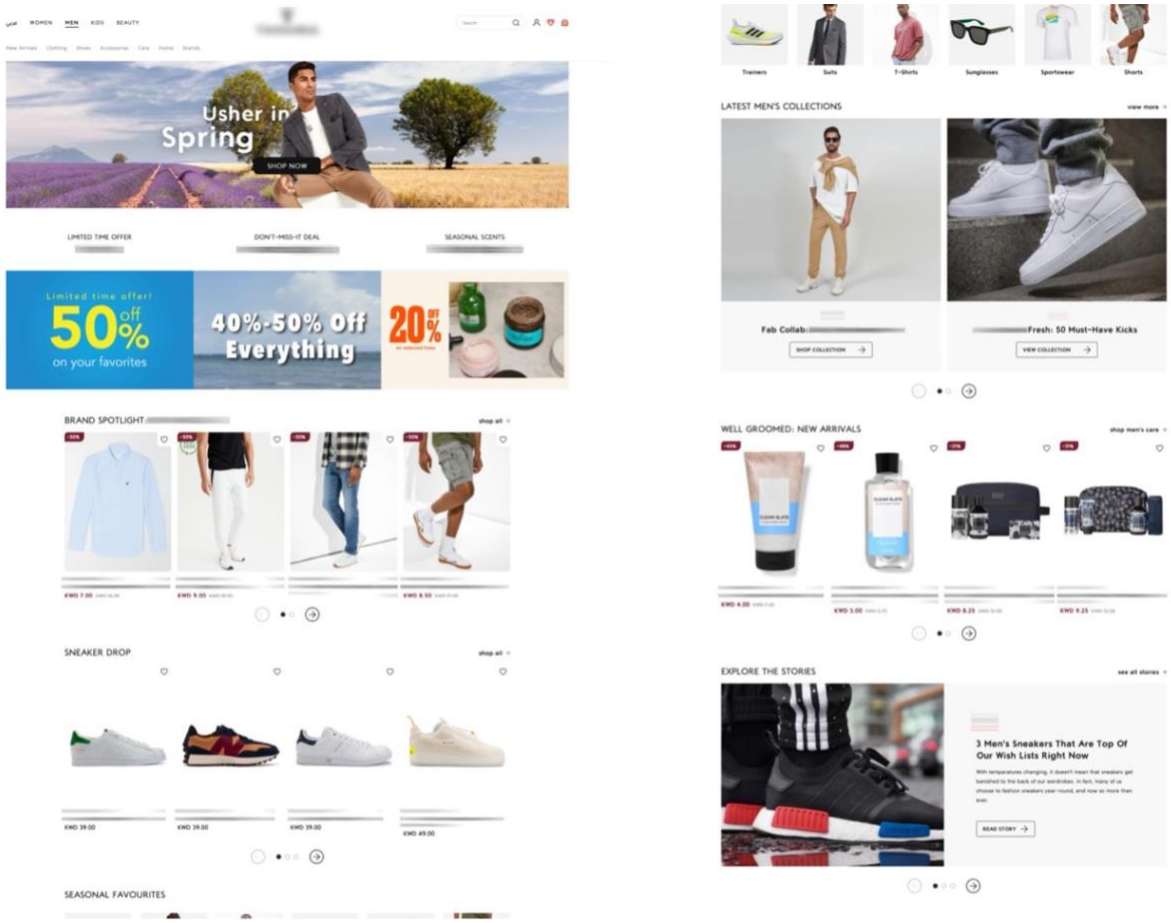

Figura 32 - Página promocional da categoria de homem

Além da página inicial, pretendia-se ter também páginas associadas a cada categoria principal de produtos, como as categorias da mulher, homem, criança, entre outras. Na Figura 32 é possível visualizar a página promocional da categoria de homem. Cada uma destas categorias teria uma página semelhante à página inicial, com o objetivo de promover determinados produtos e conteúdo associado à mesma. Estas páginas foram denominadas de páginas promocionais ou *overview*, devido ao facto de darem uma visão geral ao utilizador dos produtos relevantes de uma determinada categoria.

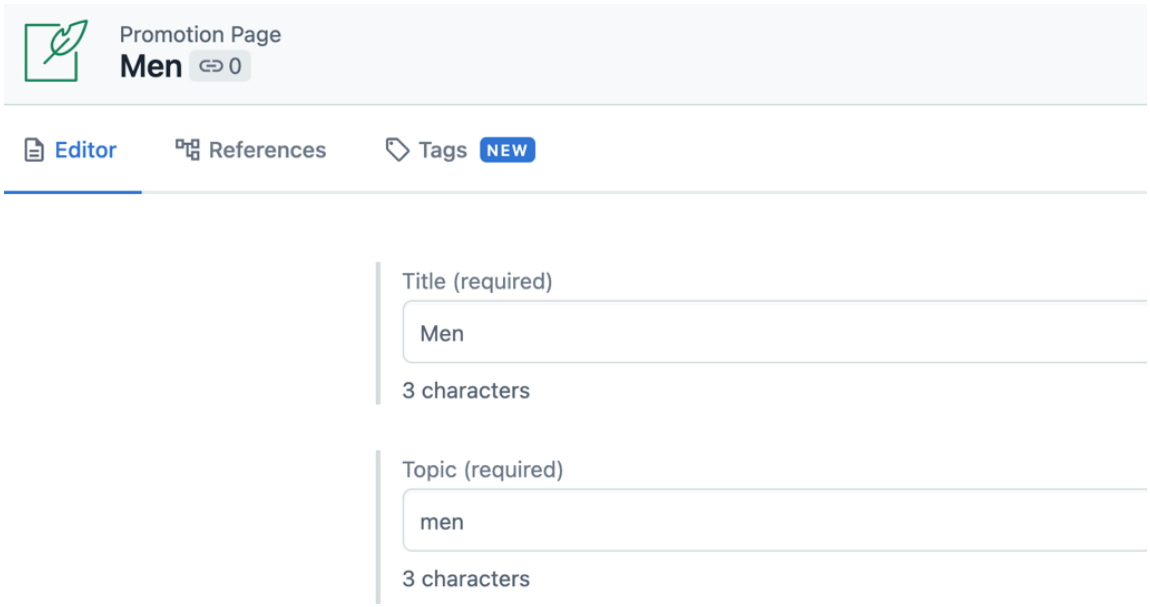

Figura 33 - Criação de uma página promocional através do Contentful

A criação destas páginas promocionais associadas a uma determinada categoria foi conseguida através da especificação do nome da categoria (*topic*) no Contentful, no ato de criação das mesmas. Após criação de uma página, e especificação de uma categoria, ao aceder a uma página de *overview* utilizando o nome especificado no Contentful (no caso da Figura 33, o endereço da página seria www.xyz.com/**men**/overview), o *frontend* realiza um pedido ao CMS, para verificar se o nome especificado no URL corresponde a uma página promocional criada e, se tal se verificar, obtém todo o conteúdo e apresenta a página desejada.

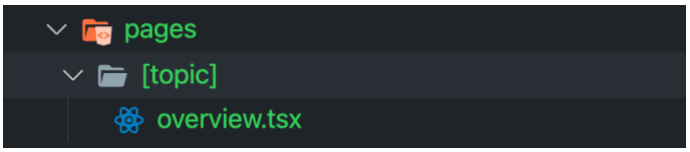

Figura 34 - Rota dinâmica utilizada para as páginas de *overview*

O dinamismo destas páginas foi conseguido através da utilização da funcionalidade de rotas dinâmicas, disponibilizada pelo Next.js [30]. Esta permitiu obter o campo *topic* dinamicamente numa só página, de modo a obter os dados armazenados no CMS com o *topic* correspondente. Como é possível observar na Figura 34, a definição desta rota dinâmica foi

efetuada através da criação de uma diretoria com o nome do parâmetro dentro de parênteses retos, indicando um parâmetro dinâmico, e da criação do ficheiro correspondente à página *overview* dentro dessa mesma diretoria.

Esta funcionalidade e dinamismo implementado permitiu agilizar o trabalho da equipa de marketing, possibilitando a rápida criação de novas páginas promocionais, mesmo não havendo uma categoria real associada, sendo ideal para determinadas ocasiões temporárias (e.g. Dia dos Namorados).

4.5.4. Página de convite

O primeiro lançamento da plataforma *web* do projeto em questão foi um *soft closed launch*, o que significa que foi lançado sem ser publicamente publicitado, e apenas para um conjunto de utilizadores internos à organização. Este tipo de lançamento é comumente utilizado a fim de testar a plataforma com utilizadores reais pela primeira vez, de modo a poder efetuar ajustes de preparação para o lançamento público. Tendo isto em conta, houve uma necessidade de disponibilizar a plataforma publicamente, mas restringir o seu acesso.

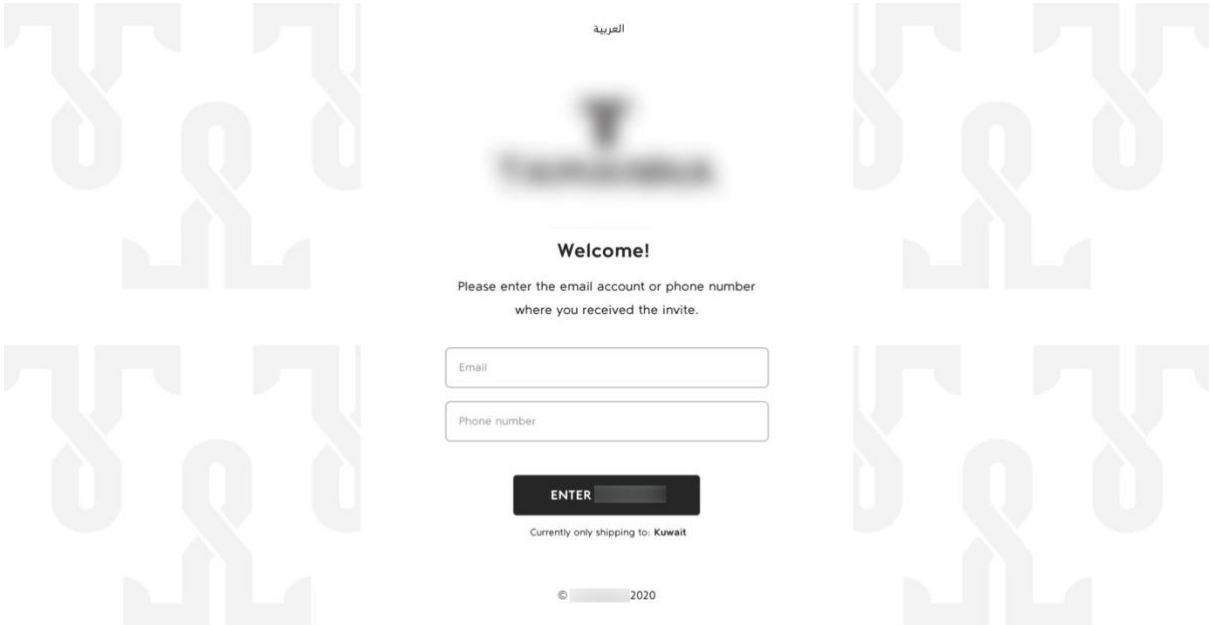

Figura 35 - Página de convite *soft launch*

Para tal, foi desenvolvida uma página de convite, para a qual todos os utilizadores seriam redirecionados ao aceder a qualquer página da plataforma. O mecanismo de convites foi efetuado através de email ou contacto telefónico, sendo necessário introduzir um dos dois na página de convite para poder aceder à plataforma, como é possível observar na Figura 35.

De um ponto de vista técnico, a funcionalidade de convite e redirecionamento do utilizador de qualquer página para a página de convite foi desenvolvida em três passos.

O primeiro passo consistiu no *upload* e disponibilização da lista de convites, em formato JSON, no CMS utilizado (Contentful), de modo a facilitar a atualização por parte de outros elementos da equipa. A decisão de utilizar o CMS para esta funcionalidade em parte influenciada pela necessidade de implementação de uma solução facilmente reversível (remoção da página de convite para lançamento público), invalidando a possibilidade de implementação de algum tipo mecanismo do lado do *backend* (API de *e-commerce*). Tendo em conta que o CMS em questão suporta alojamento e distribuição de ficheiros, foi a escolha que apresentou menor esforço por parte da equipa de engenharia.

O segundo passo, na base de código de *frontend*, consistiu na adição da função `getInitialProps` [31], disponibilizada pelo Next.js, no componente de raiz (`_app.ts`). Esta função permite a obtenção e análise de dados do lado do servidor, antes da página ser carregada no *browser* do cliente. Esta função foi utilizada para a lista de convites, transformar cada um para uma *hash* e enviar esta informação para o cliente, para que este possa carregar a página com a lista de convites disponível.

```
// -- SERVER SIDE ---------------------------------------------------------------------------------
const inviteList = ['john@email.com', 'test@email.com'];
const inviteHashList = inviteList.map((invite) => crypto.createHash('md5').update(invite).digest('hex');
// inviteHashList: ['33bdb35e6703454586f1ec426407d7de', '93942e96f5acd83e2e047ad8fe03114d'];

// -- CLIENT SIDE ---------------------------------------------------------------------------------
const providedInvite = 'test@email.com'; // user input
const providedInviteHash = crypto.createHash('md5').update(providedInvite).digest('hex');
// providedInviteHash: '93942e96f5acd83e2e047ad8fe03114d';
const isInviteValid = inviteHashList.includes(providedInviteHash); // true
```

Figura 36 - Excerto de código de verificação do convite de um utilizador

O passo de *hashing* de cada uma das credenciais de convite foi necessário devido ao facto do Next.js incluir a informação que é passada do servidor para o cliente no DOM, em formato JSON, sendo possível, qualquer utilizador, inspecionar o código fonte da página e obter um convite de acesso. Desta forma, o convite que o utilizador introduz é convertido para uma *hash* e comparado com cada uma das *hashes* da lista de convites existentes, como é possível observar no exemplo da Figura 36. Este método de ofuscação da lista de convites não é de todo o que oferece mais segurança, mas tendo em conta a sensibilidade da informação utilizada para os convites e o baixo risco de acessos indesejados, foi a solução escolhida, de modo a reduzir o esforço e tempo de desenvolvimento.

Tendo em conta que os utilizadores convidados poderiam voltar a aceder novamente à plataforma, tornar a introduzir a credencial de convite seria um processo aborrecido. Tendo isto em conta, após a primeira introdução da credencial de convite, esta era guardada com recurso a uma *cookie* local, associada ao domínio da plataforma. Esta *cookie* era verificada a cada acesso, fazendo com que a um utilizador que já tivesse introduzido o seu convite, em condições ideais, nunca mais fosse apresentada a página de convite.

De modo a facilitar o acesso à plataforma com convite, foi também implementada uma funcionalidade de introdução da credencial de convite através de URL (e.g. https://www.xyz.com?invite=test@mail.com).

O terceiro passo de implementação da funcionalidade de convite, ainda na base de código de *frontend*, consistiu no envolvimento de toda a aplicação num componente que apresentasse a página de convite caso o utilizador nunca tenha introduzido o seu convite ou o conteúdo da aplicação caso contrário.

```tsx
type State = 'validating' | 'valid' | 'invalid';

const InviteWrapper: React.FC = ({ children }) => {
  const [inviteStatus, setInviteStatus] = useState<State>('validating');

  useEffect(() => {
    const inviteValue = getInviteCookie();
    const isInviteValid = validateInvite(inviteValue);
    setInviteStatus(isInviteValid ? 'valid' : 'invalid');
  }, []);

  if (inviteStatus == 'validating') return <LoadingAnimation />;
  if (inviteStatus == 'invalid') return <InvitePage onSuccess={() => setInviteStatus('valid')} />;
  return <>children</>;
};
```

Figura 37 - Excerto de código de um componente React para validar um convite

No contexto do React, esta implementação é relativamente simples, sendo apenas necessário criar um componente que, condicionalmente, apresente a página de convite, caso o utilizador não tenha um convite ou este não seja válido, ou o conteúdo normal da aplicação, caso o convite seja válido, como é possível observar no excerto de código de exemplo da Figura 37.

Mais tarde, o método de verificação de convites foi adaptado para utilizar um sistema de *caching*, e não efetuar o pedido de obtenção da lista de convites a cada acesso. O ecrã de convites foi removido antes do lançamento público, a 8 de Fevereiro de 2020.

4.5.5. Otimização de imagens

O processo de melhoria da performance de uma aplicação *web*, assim como de otimização para motores de busca (SEO), inclui a otimização de conteúdos multimédia, como as imagens, pois estas são geralmente o conteúdo mais pesado de uma página *web*. A performance de uma página *web* pode ser medida utilizando diversas ferramentas, sendo o Lighthouse uma das mais utilizadas, encontrando-se atualmente embutida nas ferramentas de programador do Google Chrome.

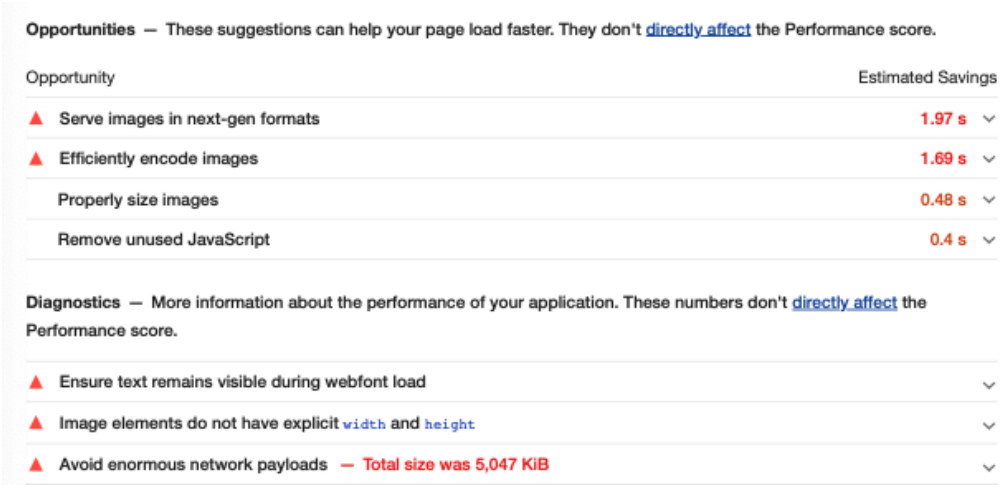

Figura 38 - Oportunidade de melhoria apontada pela ferramenta Lighthouse

Na altura em que foi efetuada uma primeira análise à performance e outras métricas da aplicação, um dos problemas apontados por esta ferramenta, foi a utilização de imagens de elevada dimensão e formatos pouco eficientes, como é possível observar na Figura 38. Este tipo de problema pode ser difícil de resolver quando o conteúdo não é redimensionado na origem (e.g. no ato de importação de produtos), que é o caso no projeto em questão. No entanto, na altura, as imagens utilizadas já se encontravam a ser servidas por CDNs (*Content Delivery Networks*), mais especificamente pela Cloudflare [32] e Contentful [33] CDN, que forneciam opções de redimensionamento e manipulação de imagens através de parâmetros. Com recurso a estas duas CDNs, foi possível otimizar o tamanho de quase todas as imagens apresentadas na aplicação.

Otimização de formato

Como verificado na Figura 38, um dos problemas apontados pela ferramenta Lighthouse, era a utilização de formatos pouco eficientes, como JPG ou JPEG (*Joint Photographic Expert Group*), PNG (*Portable Network Graphics*) ou GIF (*Graphics Interchange Format*). Estes formatos são muito comuns, no entanto existem formatos mais eficientes, que permitem a redução de tamanho sem comprometer a qualidade da imagem, como o WebP, JPEG 2000 ou JPEG XR.

Figura 39 - Compatibilidade dos formatos de imagem WebP, JPEG 2000 e JPEG XR - caniuse.com

Após alguma análise, com recurso à página *web* caniuse.com, é claro que o formato WebP é o que apresenta melhor suporte por parte dos diferentes *browsers* existentes, como é possível observar na Figura 39, sendo esta a melhor escolha. A alteração de formato para WebP foi possível através da especificação da opção `format=auto` ou `f=auto` (e.g. https://xyz.com/image.jpg?format=auto), no caso da Cloudflare, e da opção `fm=webp` (e.g. https://images.ctfassets.net/image.jpg?fm=webp), no caso do Contentful. A obtenção de imagens no formato WebP utilizando a CDN da Cloudflare resulta apenas após a ativação da opção de *Polish* [34] e é gerida automaticamente, sendo apenas servido um formato suportado pelo *browser*.

Otimização de tamanho

O ajuste das dimensões das imagens utilizadas é uma das melhores otimizações de imagem a realizar, pois são raros os casos em que é necessária a utilização de dimensões que excedam o tamanho do ecrã do dispositivo.

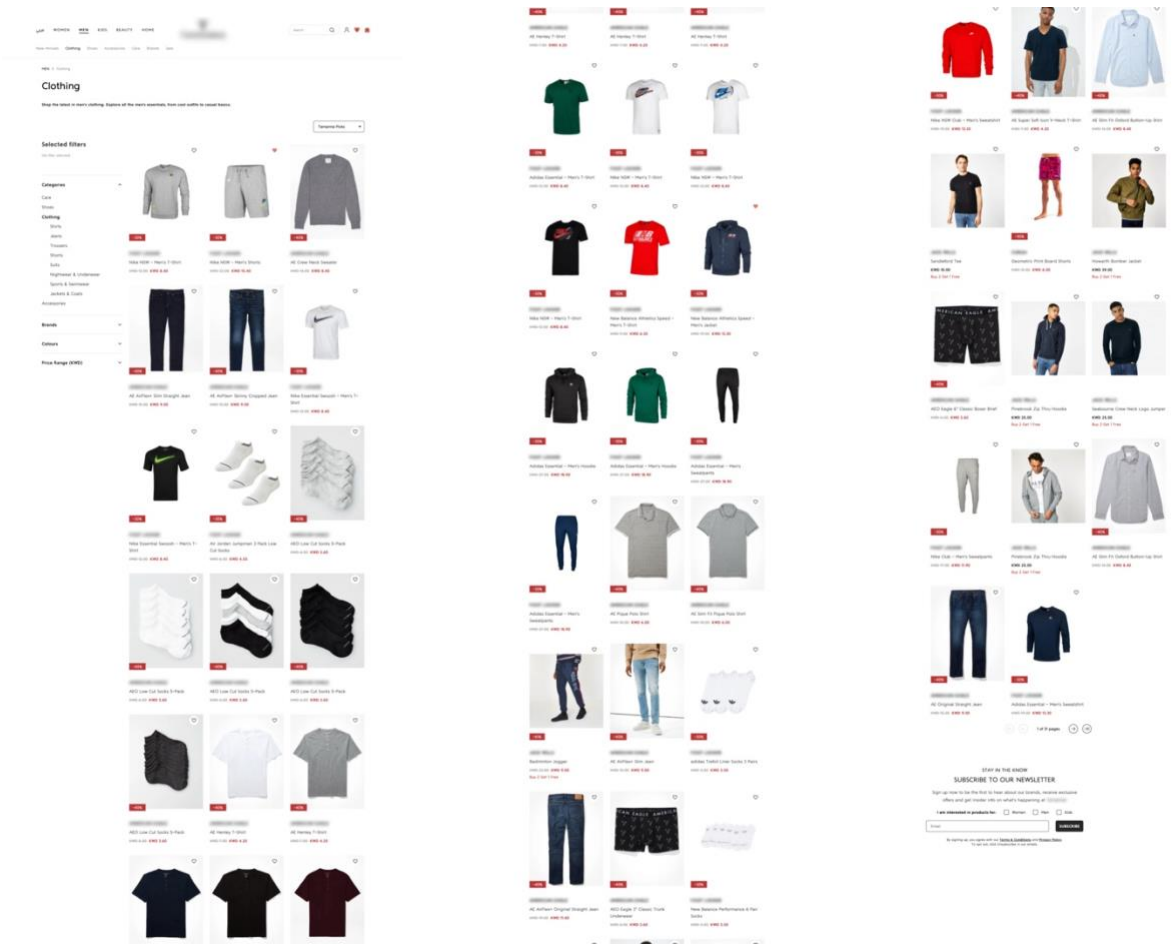

Figura 40 - Página de listagem de produtos do projeto XYZ

Uma das páginas mais importantes submetida a esta otimização foi a de listagem de produtos. Nesta são apresentados centenas de produtos, numa grelha de 3x17 (dependendo do tamanho do ecrã), com inúmeras páginas. Como é possível observar na Figura 40, dada a densidade da grelha de produtos, o tamanho de cada item é relativamente reduzido, reforçando a necessidade de redimensionamento da imagem na origem. Se todos os itens desta lista forem iguais ao primeiro, e apresentarem a sua imagem sem qualquer redimensionamento, estaríamos a carregar uma imagem com um tamanho de 1200x1200 píxeis (primeira imagem da lista da Figura 40), num elemento que tem um tamanho máximo de 261x356 píxeis. Tendo em conta que a grande maioria das imagens tem uma dimensão cerca 5x (por vezes mais) maior do que a do seu elemento, torna-se óbvia a necessidade de otimização. Para reforçar ainda mais a necessidade de otimização, com base em dados estatísticos recolhidos pela equipa de marketing do projeto em questão, cerca de 80% do tráfego da plataforma é proveniente de dispositivos móveis. Tendo em conta que os dispositivos móveis comumente utilizam dados móveis

(possivelmente finitos), transferir um elevado número de imagens de elevada dimensão numa só página, teria rapidamente um grande impacto no consumo.

```jsx
/* JSX */
<Image
  src={image.url}
  alt={image.title}
  options={{ height: 480, fit: 'crop' }}
  layout="fill"
  objectFit="cover"
  loading="eager"
  quality={100}
  priority
/>

/* next.config.js */
images = {
  deviceSizes: [640, 768, 1024, 1440],
  domains: ['images.ctfassets.net', 'cdn.xyz.com', 'cdn.stage-xyz.com'],
},
```

Figura 41 - Excerto de código de um componente de imagem React com parâmetros de otimização

De um ponto de vista técnico, a otimização na base de código da *Storefront*, foi efetuada com recurso aos componentes customizados, com base no componente de imagem do Next.js [35], que por ti já inclui funcionalidades de redimensionamento e integração com parâmetros de CDNs. Como é possível observar no exemplo da Figura 41, a utilização deste componente é bastante familiar (no contexto do React), tendo apenas duas propriedades responsáveis pela otimização da imagem, as propriedades `options` e `quality`. O componente é responsável por construir o URL com as opções fornecidas, sendo que no caso da Figura 41, o URL resultante seria algo como:

`https://www.xyz.com/imagem.jpg`**`?height=480&fit=crop&quality=100`**

Nesta mesma figura, é possível ver também um excerto de código de configuração para o Next.js (`next.config.js`), responsável por especificar os domínios a otimizar. Caso os URLs utilizados não coincidam com estes domínios, o componente de imagem não irá tentar otimizar a mesma. É também possível especificar os tamanhos de ecrã de dispositivo em que acontecem os redimensionamentos (*breakpoints*). Através destes valores, o componente de imagem do Next.js é capaz de gerar um parâmetro `srcset` para cada imagem [36], fazendo

com que esta carregue apenas o tamanho adequado, consoante a dimensão e densidade do ecrã do dispositivo.

Estas otimizações de tamanho foram aplicadas a todas as imagens com um tamanho previsível, ou seja, que têm uma largura ou altura limitado por alguma propriedade de CSS, quer seja fixa ou percentual.

Resultado obtidos

Após a alteração de formato e redução de dimensão, o tamanho das imagens transferidas no primeiro carregamento de cada página foi drasticamente reduzido. Os resultados foram analisados utilizando as ferramentas de programador do Google Chrome, para visualizar o tamanho e o formato das imagens transferidas pela rede no carregamento das páginas.

Figura 42 - Imagens transferidas na página de listagem de produtos antes de otimizações

Utilizando como base de teste a página de listagem de produtos (Figura 40), uma das páginas com mais imagens, podemos observar na Figura 42, as transferências de imagens de produtos efetuadas antes das otimizações efetuadas. Nesta é possível observar que a maior imagem transferida tem um tamanho de 118 kB e o formato JPEG. É também possível observar que o tamanho total de imagens transferido após visualizar todos os itens da página é de 2,1 MB.

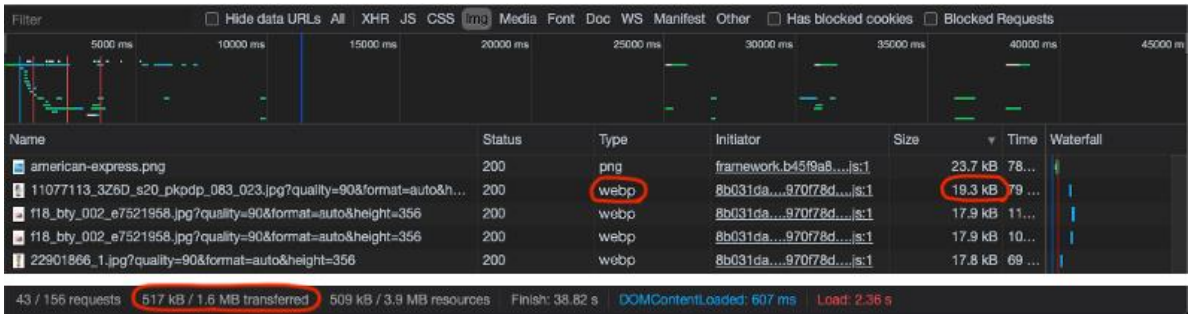

Figura 43 - Imagens transferidas na página de listagem de produtos após otimizações

Analisando os resultados obtidos após as otimizações de imagens, presentes na Figura 43, verificamos que a maior imagem de produto transferida na página tem um tamanho de 19.3 kB e o formato WebP. Observamos também que o tamanho total de imagens transferido após visualizar todos os itens da página também desceu para 517 kB. Estas diferenças de tamanho de imagens apresentam uma melhoria de cerca de 75%. Após esta análise, foi possível concluir que a otimização de imagens foi bem-sucedida.

4.5.6. Realização de outras tarefas

Durante o período de desenvolvimento são realizados diferentes tipos de tarefas, tais como implementação de novas funcionalidades e investigação, mas uma grande parte do tempo é também ocupado por tarefas de manutenção e correção de problemas. A correção de *bugs* ocupa naturalmente grande parte do tempo de desenvolvimento, pois o número de *bugs* correlaciona-se com a complexidade geral do software [37]. Assim sendo, grande parte do trabalho realizado ao longo do desenvolvimento da aplicação *Storefront* foi dedicado à correção de problemas que surgiram.

4.6. Desenvolvimento da plataforma Seller Lab

O processo de importação de produtos e introdução de novos vendedores e marcas num *marketplace* é talvez o mais importante de todos, pois sem marcas para comercializar produtos não existiriam clientes ou lucro associado. Este processo é de elevada complexidade, devido ao facto de cada marca ter as suas especificidades, diferentes tipos de produtos, maneira de trabalhar e tecnologias utilizadas. Todos estes fatores reforçam a necessidade de uniformizar o

processo de introdução de novas marcas em loja e facilitar a importação de novos produtos e inventário.

Inicialmente, o processo de importação de produtos era efetuado através de uma pipeline ETL (*Extract Transform Load*), que consiste num conjunto de processos responsáveis por extrair (*Extract*), transformar (*Transform*) e carregar (*Load*) dados provenientes de uma determinada fonte, para um determinado destino. Este processo é ideal para tratar dados provenientes de várias fontes, que possam diferir em formato, devido ao facto de se focar no processo de transformação, limpando e uniformizando os dados, e também pela sua modularidade, sendo possível acrescentar outros módulos transformadores de dados consoante a necessidade.

No entanto, este processo apresentou diversos desafios, devido ao facto de ser desenvolvido numa tecnologia diferente dos restantes componentes do *marketplace*, utilizar métodos pouco viáveis ou tradicionais, como um servidor de FTP (*File Transfer Protocol*) como fonte de dados e organização hierárquica de marcas, mas principalmente por ser totalmente operado pela equipa de engenharia. A operação deste processo pela equipa de engenharia demonstrou ser pouco escalável, pois com o crescer do *marketplace* e a constante necessidade de introduzir novas marcas, este delicado processo passou a ocupar muito tempo e conhecimento das diversas marcas, por parte da equipa. Este não era um cenário ideal, pois o contacto com as marcas já era efetuado por uma equipa dedicada – a equipa de *Seller Success* e cargos como *Brand Executive* e *Brand Acquisition Manager*– que já tinha conhecimento de como as marcas operam, as suas preferências de apresentação de produtos, entre outros. Tendo em conta que o processo de importação de produtos requer algum conhecimento técnico e não estava preparado para ser operado por pessoas menos técnicas, como a equipa de *Seller Success*, houve uma grande necessidade de reformulação do mesmo.

O plano formulado para combater este problema consistiu no desenvolvimento de uma plataforma *web* interna, com uma pilha tecnológica semelhante à utilizada no desenvolvimento do *marketplace*, com o objetivo de fornecer uma interface simples e acessível para ser utilizada pela equipa de *Seller Success*. Toda a lógica de negócio utilizada na importação de produtos, inicialmente presente na pipeline de ETL utilizada, passou para uma API, denominada de *Seller API*, desenvolvida em Java Spring Boot, com o objetivo de suportar as ações efetuadas nesta mesma plataforma *web*.

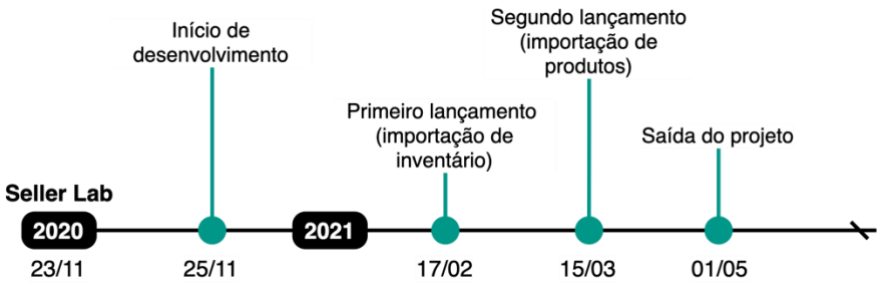

Figura 44 - Cronologia dos principais marcos do desenvolvimento do Seller Lab

Esta plataforma foi denominada de *Seller Lab*, e o processo de desenvolvimento da mesma teve início a 25 de Novembro de 2020, como é possível verificar na Figura 44.

O processo de desenvolvimento desta plataforma foi iterativo, mantendo o processo antigo ativo, enquanto não seriam implementadas todas as funcionalidades para substituir a pipeline ETL original.

Como primeira iteração (MVP) esta plataforma teria funcionalidades como login por vendedor (detentor de várias marcas), listagem de produtos presentes em loja, **importação de ficheiros de inventário (stock de produtos)**, apresentação do estado atual e erros encontrados em cada importação (caso se aplique) e exportação de ficheiros de inventário de produtos existentes e de erros de importação. A funcionalidade de importação de ficheiros de inventário era a ação mais importante e requisitada pelas marcas. Estas primeiras funcionalidades permitiram agilizar o processo de atualização de stock por parte das marcas, diminuindo a quantidade de pedidos à equipa de engenharia.

Como segunda iteração da plataforma, foi desenvolvida a funcionalidade de **importação de novos produtos**, uma das mais importante de qualquer plataforma de gestão de produtos. Esta foi a funcionalidade que apresentou mais desafios, pois cada produto possui um elevado número de propriedades, havendo mais espaço para erros.

Ambas as funcionalidades de importação (inventário e produtos) era efetuada através de ficheiros CSV (*Comma-separated values*) e XLSX (*Microsoft Excel Spreadsheet*), pois este era um método familiar para a equipa de *Seller Success*, facilmente manipulável por pessoas com pouco conhecimento tecnológico, e por ser também um formato muito utilizado pelas próprias marcas, no processo de extração de produtos dos seus sistemas de gestão. Estes ficheiros tinham de cumprir um determinado formato definido pela equipa de engenharia em

conjunto com as marcas, e era da responsabilidade da equipa de *Seller Success* garantir que este formato era cumprido. O formato deste ficheiro era também validado pela API desenvolvida, o que significa que quando algo não se encontrava corretamente preenchido, o utilizador da plataforma seria informado de um erro no ficheiro utilizado, e teria de o corrigir antes de voltar a importar.

4.7. Desenvolvimento da aplicação móvel

O último projeto no qual tive oportunidade de trabalhar, foi o desenvolvimento da aplicação móvel, como complemento à plataforma *web* (*Storefront*) desenvolvida. A ideia do desenvolvimento desta aplicação móvel foi suportada por dados estatísticos recolhidos durante aproximadamente 6 meses após o primeiro lançamento da plataforma *web*. Estes dados indicavam que, no período de um mês, cerca de 90% do tráfego (junho de 2021) existente na plataforma era proveniente de dispositivos móveis. Estes dados não evidenciaram nenhuma novidade, tendo em conta os acessos à internet em todo o mundo já são liderados por dispositivos móveis há alguns anos, dada a sua conveniência e baixo custo, em comparação com computadores portáteis ou fixos. Mesmo sem estes dados estatísticos, já era do conhecimento da equipa que a larga maioria dos grandes *marketplaces* digitais existentes dispõem de aplicações móveis ou pelo menos de uma plataforma *web* bem-adaptada para plataformas móveis. Tendo estes factos em conta, houve uma necessidade de alocar engenheiros para formar uma equipa de desenvolvimento móvel, a fim de desenvolver a aplicação móvel para o projeto XYZ. O tempo de desenvolvimento estabelecido para esta aplicação foi cerca de 6 meses e foram delineadas as funcionalidades a implementar numa primeira iteração. Sendo que a grande maioria das funcionalidades presentes na plataforma *web*, teriam de ser incluídas na aplicação móvel.

De um ponto de vista técnico, dada a exigência deste tipo de mercado, houve uma necessidade de agilizar o processo de desenvolvimento, sendo que, na escolha da pilha tecnológica do projeto, foram tidos em conta os seguintes fatores:

- Estatísticas recolhidas, que indicavam que cerca de 70% do tráfego era proveniente de dispositivos de iOS, e o restante de Android, indicando que existem uma necessidade de abranger ambas as plataformas (num curto espaço de tempo);

- Tecnologias base utilizadas nos restantes projetos (*Storefront* e *Seller Lab)*, sendo estas React e TypeScript. A pilha tecnológica desde projetos fundamentou a escolha da tecnologia React Native para o desenvolvimento da aplicação móvel. O desenvolvimento com esta tecnologia assemelha-se bastante ao desenvolvimento *web* utilizando React, o que faz com que a grande maioria dos membros da equipa de *frontend* se sintam confortáveis em desenvolver para a aplicação móvel. Estes fatores contribuem também para um desenvolvimento mais acelerado;
- Problemas enfrentados durante o desenvolvimento dos restantes projetos, como a falta de documentação, inconsistência de tipos (TypeScript) e propriedades no corpo da mensagem nos pedidos à API de *e-commerce* (Broadleaf), assim como um elevado tamanho da mesma. Estes problemas inspiraram a escolha da tecnologia GraphQL para o desenvolvimento de uma API que atua como *middleware* entre a aplicação cliente e a API existente. A utilização desta tecnologia permitiu a redução de tamanhos de mensagem, dada a natureza de como o GraphQL funciona, melhor tipagem das mesmas e melhor documentação da API.

Após esta análise de fatores, a pilha tecnológica escolhida para o desenvolvimento do projeto da aplicação móvel foi a seguinte:

- Aplicação cliente
  - **React Native** como base de desenvolvimento, de modo a abranger tanto a plataforma iOS como Android;
  - **TypeScript** como a linguagem de desenvolvimento;
  - **Apollo Client** como o cliente de GraphQL utilizado para a efetuação de pedidos à API de *middleware* desenvolvida em conjunto com a aplicação, assim como desenvolvimento de mecanismos de persistência e cache;
  - **CSS em** JS e Styled Components para a criação de componentes estilizados.
- API de *middleware*
  - **Node.js** como base de desenvolvimento (*runtime*);
  - **TypeScript** como a linguagem de desenvolvimento;
  - **Apollo Server** (Express.js) como ferramenta de desenvolvimento da API.
- Automatização e distribuição
  - **App Center** como serviço de geração e distribuição de aplicações para ambas as plataformas, em especial para Android;

- **TestFlight** como serviço de distribuição de aplicações de teste específicas para a plataforma iOS.

Figura 45 - Cronologia dos principais marcos do desenvolvimento da Aplicação Móvel

O início do projeto deu-se a 3 de maio de 2021, e o início de desenvolvimento da aplicação a 5 de maio de 2021. Tendo em conta que o período de estágio terminou a 17 de maio de 2021, este processo não foi abrangido pelo mesmo, como é possível verificar na Figura 45.

Atualmente (19 de Novembro de 2021), a aplicação encontra-se lançada na App Store (iOS) e Play Store (Android).

4.8. Desafios de desenvolvimento

O desenvolvimento de uma loja de comércio digital (*marketplace)* apresenta sempre diversos desafios. Desafios estes que se devem em grande parte ao facto de ser uma plataforma potencialmente acedida por um elevado número de utilizadores, através de vários tipos de dispositivos. Esta característica faz com que esta plataforma requeira algum cuidado a nível de otimização e responsividade da página, assim como a nível de *SEO (Search Engine Optimization)*, de modo a maximizar a visibilidade das páginas perante potenciais clientes. Estes são desafios bastante comuns no desenvolvimento *web e-commerce*, mas existem outros desafios menos comuns, como a mudança de direção de uma página, como é o caso das páginas na língua árabe, que apresentam peculiaridades inesperadas neste contexto.

4.8.1. Adaptação do conteúdo para mudanças de direção

No contexto do projeto XYZ, um dos maiores desafios apresentados foi a adaptação da aplicação para árabe, uma linguagem escrita da direita para a esquerda (*RTL – Right-to-left*). Este foi um dos maiores desafios devido ao facto desta direção de escrita afetar, não só o texto apresentado, mas também a maneira de apresentação de conteúdo, havendo a necessidade de validação do mesmo por alguém familiarizado com este tipo de conteúdo, e que consiga ler e escrever árabe.

De uma perspetiva técnica, a adaptação para RTL, começa pela inversão da página. Esta tarefa é bastante facilitada pelos *browsers*, sendo apenas necessário adicionar a propriedade `dir="rtl"` ao elemento `html`, presente na raiz do documento. Esta propriedade trata de grande parte da inversão da direção da página, mas, como é natural, não cobre todos os casos, sendo necessário rever todos os componentes da página e adaptar consoante a necessidade.

Propriedades lógicas de CSS

Um aspeto importante de cada página são os espaços utilizados, através da aplicação de propriedades CSS, como *margin* e *padding*. Estas propriedades aplicam espaços verticalmente e horizontalmente, sendo possível aplicar especificamente ao lado esquerdo e direito. No entanto, devido ao facto de a página poder mudar de direção, os lados direito e esquerdo podem ser trocados. Isto significa que uma propriedade aplicada ao lado esquerdo (e.g. *margin-left*), com uma página da esquerda para a direita (*LTR - Left-to-right*) em mente, ficará do lado direito quando a página muda de direção (e.g. alteração de inglês para árabe), causando problemas de espaçamento.

Physical Property	Logical Property
margin-right	margin-inline-end
margin-left	margin-inline-start
padding-right	padding-inline-end
padding-left	padding-inline-start
text-align: left;	text-align: start;
text-align: right;	text-align: end;
left	inset-inline-start
right	inset-inline-end

Figura 46 - Propriedade CSS físicas e lógicas equivalentes [38]

De modo a resolver este problema, foi introduzido o módulo de *Logical Properties and Values* [39] na especificação de CSS, que trouxe propriedades lógicas, como alternativa às propriedades físicas existentes, como é possível observar na Figura 46. Por exemplo, propriedades como *margin-inline-end* em vez de *margin-right*, permitem aplicar uma margem no final de um elemento (lado direito, numa página da esquerda para a direita), independentemente da direção da página.

Conteúdo sujeito a alterações de direção

Numa aplicação com conteúdo da direita para a esquerda, grande parte do conteúdo está sujeito a uma alteração de direção.

Figura 47 - Exemplo de alteração de direção de uma imagem e texto em árabe

Conteúdo como texto, imagens, formulários (e.g. caixas de texto), entre outros elementos, são movidos para o lado contrário, devido ao facto de, tal como nós começamos a avaliar o conteúdo de uma página da esquerda para a direita, neste tipo de linguagens o conteúdo é avaliado da direita para a esquerda, como é possível verificar na Figura 47. Alguns tipos de conteúdo mais elaborado, como *carousels*, *sliders* e calendários, podem apresentar um desafio maior do que elementos simples como texto e imagens, devido ao facto de apresentarem lógica adicional que requeira algum tipo de adaptação para acomodar uma alteração de direção.

Carousels

Os *carousels*, essencialmente *slideshows* ou galerias de conteúdo interativas, são um exemplo de conteúdo que requer alguma adaptação para se comportarem corretamente numa mudança de direção, especialmente por ser comum recorrer a componentes/bibliotecas pré-desenvolvidas, de modo a ganhar algum tempo de desenvolvimento [40].

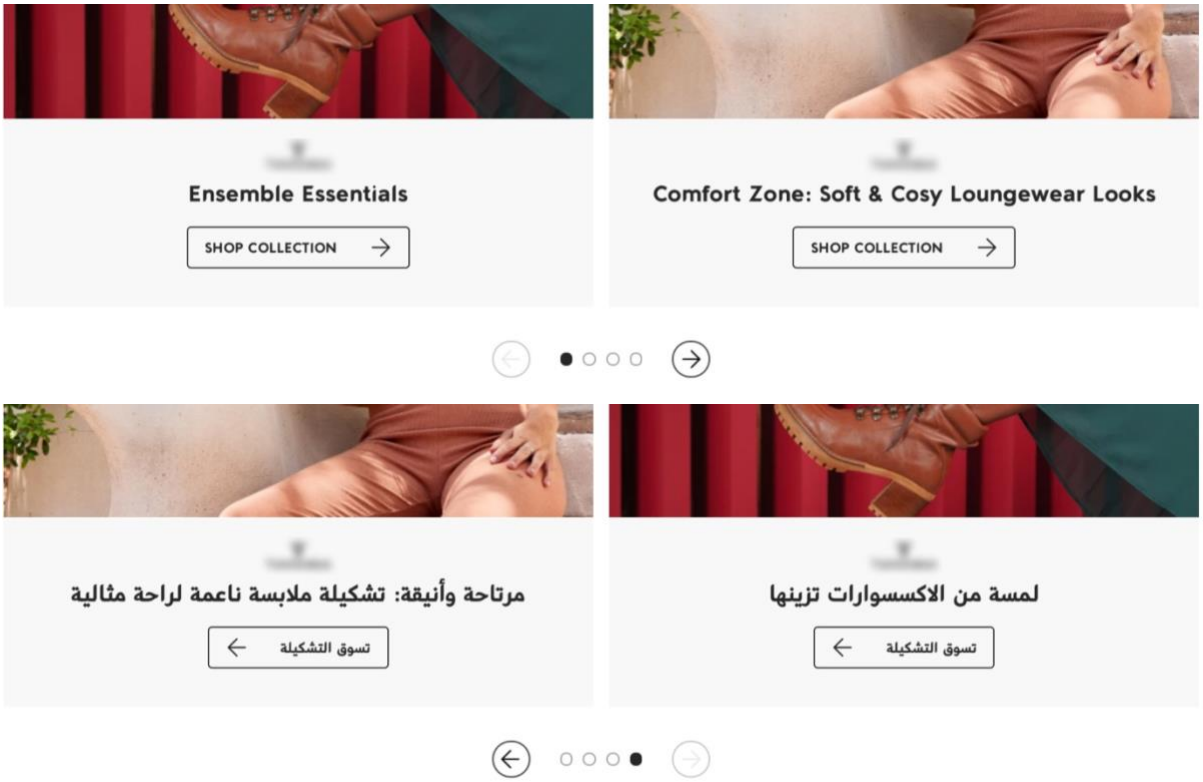

Figura 48 – Mudança de alteração num componente carousel

Como é possível observar na Figura 48, com uma alteração de direção, a ordem destes componentes é invertida, passando a ser necessário clicar para na seta que aponta para esquerda (caso se aplique), em vez de para a direita, para avançar para o próximo slide. Os próprios slides também são invertidos, tanto em ordem, como em conteúdo de texto e imagem. Esta adaptação apresenta alguns desafios, pois a lógica por trás da gestão de slides tem de ser adaptada, pois caso não seja, surgirão alguns comportamentos indesejados.

Conteúdo não sujeito a alterações de direção

Apesar da maioria do conteúdo de uma aplicação alterar de direção, existem diversos casos que fogem à regra, e precisam de especial atenção.

Números

Um dos casos mais importante são os números [41], pois nem todos devem mudar de direção e, sem a devida atenção, a grande maioria será incorretamente invertida. Vejamos um

exemplo importante, como um número de cartão de crédito, num formulário de pagamento. Este é um caso em que é comum ter uma caixa de texto que aceite números, mas também espaços ou outros caracteres utilizados como separadores, para auxiliar o utilizador no cumprimento de um determinado formato (e.g. xxxx xxxx xxxx xxxx).

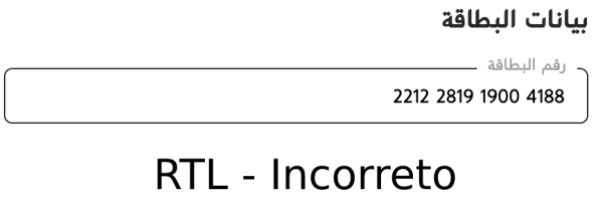

Figura 49 - Caixa de texto de números em RTL

Este caso em particular apresenta alguns desafios, pois, da direita para a esquerda, a caixa de texto apresenta um comportamento estranho, misturando números com símbolos (espaços), fazendo com o número seja apresentado com a ordem incorreta. Na Figura 49 é possível verificar este comportamento, em que o número 4188 1900 2819 2212 é apresentado na ordem incorreta. Vejamos outras possíveis soluções.

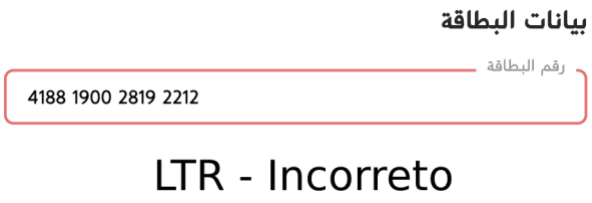

Figura 50 - Caixa de texto de números em LTR

A primeira possível solução é fazer com que a caixa de texto obedeça à ordem da esquerda para a direita, utilizando, por exemplo, a uma propriedade de CSS, como `direction:ltr`. No entanto, esta faz com que este elemento não obedeça à direção dos restantes elementos da página, o que não é o comportamento esperado, como é possível observar na Figura 50.

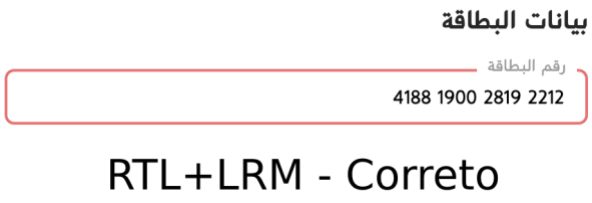

Figura 51 – Caixa de texto de números em RTL com LRM

A solução correta, para estes casos, é utilizar o código *unicode LRM (Left-To-Right Mark)* `U+200E` [42], concatenando-o com o texto presente na caixa de texto. Este código especial força o texto a obedecer à direção da esquerda para a direita (LTR), mantendo a direção da própria caixa de texto da direita para a esquerda, como é possível observar na Figura 51.

Figura 52 - Número de telefone LTR vs. RTL [41]

Outro exemplo de números que não se devem inverter são os números de telefone, pois a ordem dos mesmos é importante, como é possível observar na Figura 52. Este é um caso que gera bastante confusão, pois o número seria lido da direita para a esquerda. No entanto, sendo estes números, e não caracteres árabes, devem ser sempre apresentados da esquerda para a direita. Para tal, pode ser usada qualquer uma das estratégias mencionadas acima.

Ícones

Os ícones são comumente utilizados em aplicações para melhorar a experiência visual, salientar importância de mensagem ou ações e guiar o utilizador. Estes são elementos que alteram de direção com a página, no entanto, nem sempre faz sentido [43].

Figura 53 - Inversão horizontal de ícones

No caso de linguagens escritas da direita para a esquerda, sendo uma inversão horizontal, apenas os ícones que indicam uma direção ao utilizador (e.g. setas horizontais) devem ser invertidos, como é possível observar na Figura 53. Os restantes ícones devem manter a sua direção original. Tendo em conta que são poucos os ícones que devem ser invertidos (maioritariamente setas), uma boa solução técnica seria manter uma *whitelist* com os mesmos.

```
html[dir='rtl'] .flip-icon {
  transform: scaleX(-1);
}
```

Figura 54 - Classe CSS para inversão de ícones em RTL

Tendo em conta que os ícones são invertidos com a página, de modo restaurar a sua direção original, uma transformação através de CSS pode ser aplicada, como é possível observar na Figura 54.

4.9. Processo de contratação

Durante o período de desenvolvimento do projeto XYZ tive oportunidade de obter conhecimento em diversas tecnologias e obter uma alargada visibilidade da estrutura e lógica de negócio utilizada pelas aplicações desenvolvidas durante o mesmo. Tendo em conta este conhecimento obtido, tive a oportunidade de ser convidado a participar em 2 entrevistas a candidatos à equipa de engenharia deste mesmo projeto. A minha intervenção nestas entrevistas aconteceu na fase de entrevista técnica, que é, por norma, efetuada por dois engenheiros experientes e com visibilidade sobre as necessidades do projeto em questão. Esta entrevista acontece após o desenvolvimento de um desafio técnico, por parte do candidato, que é avaliado e analisado antes e durante a entrevista. O candidato, após a entrevista é classificado por ambos os entrevistadores, consoante um conjunto de parâmetros de engenharia pré-definidos, constituindo um cartão de classificação (*scorecard*) final que inclui a decisão dos mesmos. Após este processo, a candidatura segue para a frente no processo de contratação.

Ambas as entrevistas foram efetuadas a candidatos à posição de Engenheiro *Frontend*, focando-se em experiência com linguagens e tecnologias utilizada no projeto em questão, tais como JavaScript, TypeScript, HTML, CSS, React, Next.js, CMSs, Git, entre outras. É também obrigatório ter um bom inglês falado e escrito e de interesse a experiência com metodologias de desenvolvimento ágeis como o Scrum.

O resultado das mesmas foi positivo, tendo ambos os candidatos sido contratados e encontrarem-se atualmente a trabalhar na equipa de engenharia do projeto em questão.

4.9.1. Formato da entrevista

O formato de entrevista de engenharia utilizado é bastante aberto, ficando ao critério de cada engenheiro definir a ordem da mesma e que questões colocar ao candidato. Ambas as entrevistas efetuadas tiveram um formato relativamente idêntico, sendo constituídas pelas seguintes partes:

- **Diálogo de introdução** – Durante este diálogo, os entrevistadores apresentam-se candidato, e vice-versa, e são colocadas questões de cariz informal, num formato de conversa "corrida", sobre o percurso profissional do candidato e a sua

experiência. Durante o mesmo, por vezes, dependendo do candidato, grande parte das questões técnicas são colocas, o que influencia o restante formato da entrevista;

- **Questões técnicas** – Nesta fase são efetuadas questões técnicas mais objetivas, de modo ao candidato demonstrar o seu conhecimento de *frontend* com a tecnologias utilizadas na posição à qual se candidata. Como referido anteriormente, esta fase é bastante influenciada pelo diálogo inicial, pois grande parte das questões podem já ter sido colocadas ou o candidato pode, de alguma forma, já ter referido diversos tópicos de interesse, não fazendo sentido colocar perguntas já respondidas;
- **Desafios técnicos (opcional)** – Nesta fase são apresentados alguns desafios técnicos com recurso às tecnologias utilizadas na posição à qual o candidato concorre. Estes desafios são geralmente partilhados com recurso a plataformas *web* de desenvolvimento partilhado (e.g. codesandbox, replit e jsfiddle). Em ambas as entrevistas foram efetuados 2 desafios, um sobre JavaScript (sem *frameworks* ou bibliotecas) e outro sobre React. Neste foram abordados conceitos fundamentais de ambas as tecnologias, em que o candidato desenvolveu ou descreveu código ao vivo, a fim de responder às questões colocadas;
- **Diálogo final** – Neste diálogo final são abordados temas relativos à entidade empregadora, tais como o dia-a-dia de trabalho, áreas de atuação, composição de equipas, metodologia de trabalho, entre outros. É também a altura em que o candidato coloca questões que tenha aos entrevistadores.

Este formato não era de todo obrigatório, sendo também bastante comum optar por um formato mais orientado à conversa natural. Neste formato, o diálogo de introdução, em que se conhece o candidato e se fala sobre a sua experiência, acaba por ser diluído com a fase de questões e desafios técnicos. Isto acontece quando o candidato está à vontade, e ao longo da conversa, acaba por responder a perguntas chave de uma forma mais indireta. Este formato é geralmente preferível, pois o candidato encontra-se mais descontraído, comparado com um formato técnico de "pergunta resposta". Esta descontração apela também à sinceridade e clareza nas respostas às questões colocadas. Este formato resulta apenas se o candidato mantenha a conversa, caso não aconteça, o entrevistador pode sempre voltar para o formato original de perguntas diretas e desafios técnicos.

Em conclusão ao capítulo de desenvolvimento, as tarefas realizadas e desafios enfrentados foram cruciais na minha evolução como engenheiro de software *frontend*, permitindo-me

solidificar os meus conhecimentos de desenvolvimento *web*. O projeto em questão provou ter sido uma experiência bem-sucedida de tecnologias emergentes no mundo do desenvolvimento *frontend* (e.g. Next.js), assim como valiosa experiência em desenvolvimento de *marketplaces*, adquirida por diversos elementos da equipa da xgeeks.

5. Conclusão

O desenvolvimento de uma plataforma de comércio digital, ou *marketplace*, é algo que apresenta diversos desafios técnicos. Desafios estes, que podem ser minimizados por uma organizada fase de levantamento de requisitos e escolha de tecnologias. A escolha de tecnologias é uma das fases mais importantes, pois permite analisar pontos chave, cruciais na longevidade de um software. Pontos como flexibilidade, performance, e escalabilidade, devem ser considerados nesta fase, de modo a fundamentar a escolha da pilha tecnológica (e.g. linguagem, *frameworks* e bibliotecas, serviços externos, etc). Estas escolhas são de elevada importância, também devido ao facto de o esforço de transição numa fase avançada de desenvolvimento aumentar ao longo do tempo. É também do interesse de toda a equipa escolher ferramentas que apresentem sinais de desenvolvimento ativo, existência de suporte técnico ou uma comunidade madura.

Ao longo do projeto XYZ, no qual fui integrado no âmbito do presente estágio, tive oportunidade de enfrentar alguns dos desafios mencionados, desenvolvendo o meu conhecimento acerca dos mesmos, no contexto de desenvolvimento *web*. A otimização de performance e de páginas para motores de busca são exemplos de desafios que fundamentaram escolhas de tecnologias como Next.js. Estes desafios foram importantes para entender a importância de ferramentas modernas que agilizam processos deste tipo que, de outra forma, seriam mais demorados e exigiriam mais esforço de engenharia. Outros desafios como a adaptação de conteúdo para mudanças de direção foram também importantes para entender a importância do correto desenvolvimento de páginas *web*, de modo a estas estarem preparadas para diversos fatores externos.

É possível concluir que, o projeto no qual tive oportunidade de trabalhar foi bem-sucedido, tendo todas a aplicações desenvolvidas (*Storefront*, *Seller Lab* e a aplicação móvel) sido lançadas para produção, estando atualmente disponíveis a qualquer utilizador (à exceção de plataformas internas). As funcionalidades desenvolvidas cumpriram o seu objetivo, estando estas plataformas atualmente a gerar milhares de vendas, algo que era um dos principais objetivos a atingir. Foi também possível aplicar, durante o desenvolvimento deste projeto, grande parte dos conhecimentos obtidos durante o mestrado em questão, relativamente a desenvolvimento *web* e metodologias de desenvolvimento ágeis.

O modelo de trabalho que experienciei ao longo do presente estágio foi maioritariamente híbrido, algo que me permitiu escolher o meu lugar de trabalho. Este modelo traz diversas vantagens aos colaboradores de uma empresa, tanto a nível monetário como psicológico. Vantagens como a redução de stress, devido ao facto de não haver a necessidade de deslocação diária e, de um modo geral, a possibilidade de trabalhar num espaço familiar e de conforto. O trabalho a partir de casa permite também ter mais tempo livre, pois o tempo despendido na deslocação e outros preparativos para o trabalho em escritório já não fazem parte da equação. Apesar das vantagens do trabalho remoto serem diversas, é importante encontrar um balanço, de modo a manter o convívio entre equipas e a não perder outras *soft skills*, sendo o modelo híbrido ideal para tal.

Por fim, gostaria de salientar que um bom ambiente de trabalho e uma comunidade ativa e saudável são características que associo à xgeeks, e que contribuíram positivamente para a minha performance ao longo deste estágio. Agradeço a todos os elementos desta empresa que me ajudaram ao longo do mesmo, é com gosto que posso dizer foram muitos, e por me terem acolhido, e continuarem atualmente a acolher.

Referências Bibliográficas

[1] «TypeScript», *Wikipedia*. 28 de maio de 2022. [Em linha]. Disponível em: https://en.wikipedia.org/w/index.php?title=TypeScript&oldid=1090322494

[2] «Documentation - Everyday Types». https://www.typescriptlang.org/docs/handbook/2/everyday-types.html#any

[3] «Source-to-source compiler», *Wikipedia*. 15 de setembro de 2021. [Em linha]. Disponível em: https://en.wikipedia.org/w/index.php?title=Source-to-source_compiler&oldid=1044485864

[4] «react», *npm*. https://www.npmjs.com/package/react

[5] «@types/react», *npm*. https://www.npmjs.com/package/@types/react

[6] *Definitely Typed*. DefinitelyTyped, 2021. [Em linha]. Disponível em: https://github.com/DefinitelyTyped/DefinitelyTyped

[7] «React (JavaScript)», *Wikipédia, a enciclopédia livre*. 28 de abril de 2022. [Em linha]. Disponível em: https://pt.wikipedia.org/w/index.php?title=React_(JavaScript)&oldid=63480980

[8] «Next.js», *Wikipédia, a enciclopédia livre*. 28 de abril de 2022. [Em linha]. Disponível em: https://pt.wikipedia.org/w/index.php?title=Next.js&oldid=63480977

[9] «Routing: Introduction | Next.js». https://nextjs.org/docs/routing/introduction

[10] «Advanced Features: Automatic Static Optimization | Next.js». https://nextjs.org/docs/advanced-features/automatic-static-optimization

[11] «Incremental Static Regeneration», *Vercel Documentation*. https://vercel.com/docs/concepts/next.js/incremental-static-regeneration

[12] *styled-components/styled-components*. styled-components, 2021. [Em linha]. Disponível em: https://github.com/styled-components/styled-components

[13] «Headless content management system», *Wikipedia*. 6 de abril de 2022. [Em linha]. Disponível em: https://en.wikipedia.org/w/index.php?title=Headless_content_management_system&oldid=1081260367

[14] «Content Delivery API | Contentful». https://www.contentful.com/developers/docs/references/content-delivery-api/

[15] *contentful/contentful.js*. Contentful, 2021. [Em linha]. Disponível em: https://github.com/contentful/contentful.js

[16] «GraphQL | A query language for your API». https://graphql.org/

[17] «Git». https://git-scm.com/

[18] «GitLab CI vs GitHub Actions», *GitLab*. https://about.gitlab.com/devops-tools/github-vs-gitlab/ci-missing-github-capabilities/

[19] Atlassian, «Gitflow Workflow | Atlassian Git Tutorial», *Atlassian*. https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow

[20] «Conventional Commits», *Conventional Commits*. https://www.conventionalcommits.org/en/v1.0.0/

[21] *Angular - The modern web developer's platform.* Angular, 2021. [Em linha]. Disponível em: https://github.com/angular/angular/blob/22b96b96902e1a42ee8c5e807720424abad3082a/CONTRIBUTING.md

[22] typicode, *husky*. 2021. [Em linha]. Disponível em: https://github.com/typicode/husky

[23] «conventional-changelog/commitlint: Lint commit messages». https://github.com/conventional-changelog/commitlint

[24] «commitlint/@commitlint/config-conventional at master · conventional-changelog/commitlint», *GitHub*. https://github.com/conventional-changelog/commitlint

[25] «Content model | Contentful». https://www.contentful.com/developers/docs/concepts/data-model/

[26] «Home - Documentation». https://contentful.github.io/contentful.js/contentful/8.3.2/#configuration

[27] «Getting Started with Contentful and JavaScript | Contentful». https://www.contentful.com/developers/docs/javascript/tutorials/using-js-cda-sdk/

[28] *remarkjs/react-markdown*. remark, 2021. [Em linha]. Disponível em: https://github.com/remarkjs/react-markdown

[29] «Cumulative Layout Shift (CLS)», *web.dev*. https://web.dev/cls/

[30] «Routing: Dynamic Routes | Next.js». https://nextjs.org/docs/routing/dynamic-routes

[31] «Data Fetching: getInitialProps | Next.js». https://nextjs.org/docs/api-reference/data-fetching/getInitialProps

[32] «Cloudflare Image Optimization · Cloudflare Image Optimization docs». https://developers.cloudflare.com/images/

[33] «Images API». https://www.contentful.com/developers/docs/references/images-api/

[34] «Cloudflare Polish · Cloudflare Image Optimization docs». https://developers.cloudflare.com/images/polish

[35] «next/image | Next.js». https://nextjs.org/docs/api-reference/next/image

[36] «next/image - Device Sizes | Next.js». https://nextjs.org/docs/api-reference/next/image#image-sizes

[37] «Halstead complexity measures», *Wikipedia*. 13 de janeiro de 2022. [Em linha]. Disponível em: https://en.wikipedia.org/w/index.php?title=Halstead_complexity_measures&oldid=1065366941

[38] P. Figueiredo, «CSS Logical Properties», *xgeeks*, 4 de agosto de 2021. https://medium.com/xgeeks/css-logical-properties-507a8c04a21f

[39] «CSS Logical Properties and Values - CSS: Cascading Style Sheets | MDN». https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Logical_Properties

[40] P. Figueiredo, «External Components — Carousel», *xgeeks*, 26 de agosto de 2021. https://medium.com/xgeeks/external-components-carousel-81e330b71fd6 (acedido 31 de outubro de 2021).

[41] P. Figueiredo, «Stop fixing Numbers», *xgeeks*, 1 de setembro de 2021. https://medium.com/xgeeks/stop-fixing-numbers-96a0a1915719

[42] «Left-to-right mark», *Wikipedia*. 1 de maio de 2021. [Em linha]. Disponível em: https://en.wikipedia.org/w/index.php?title=Left-to-right_mark&oldid=1020908163

[43] P. Figueiredo, «Icons have meaning», *xgeeks*, 19 de agosto de 2021. https://medium.com/xgeeks/icons-have-meaning-bb222c485384

Anexos

Anexo A: Proposta de Estágio

Proposta de Estágio

Mestrado em Engenharia Informática – Computação Móvel (MEI-CM)

Escola Superior de Tecnologia e Gestão

Instituto Politécnico de Leiria

EMPRESA/ORGANIZAÇÃO PARCEIRA: XGEEKS PORTUGAL LDA

QUADRO A - DESCRIÇÃO DO ESTÁGIO

TEMA/TÍTULO	**Participação em projectos de Clientes**
ÁREA DE ESTUDO	Marketplaces
ÁREA DE ATIVIDADE ECONÓMICA	Desenvolvimento de Software (maioritariamente web)
OBJETIVOS	Integração na equipa
BREVE DESCRIÇÃO	
Participação em projectos dos nosso clientes com integração nas equipas existentes.	
Somos uma equipa de pessoas muito talentosas que desenvolvem soluções de ponta para as principais marcas da Europa. A nossa missão é levar capacidade e conhecimento de engenharia para grandes corporações.	
Os nosso Clientes: Dimler, DHL, Lanxess...	
COMPETÊNCIAS REQUERIDAS	Inglês Fluente
COMPETÊNCIAS OBTIDAS AO LONGO DO ESTÁGIO	Trabalho em equipa.

QUADRO B - TECNOLOGIAS ENVOLVIDAS NA REALIZAÇÃO DO ESTÁGIO

SISTEMAS OPERATIVOS	Indiferente
STACK	React, Symfony, Redux, Docker, Java, PHP, Kotlin, JS, Swift, Python, Kubernetes
METODOLOGIAS DE DESENVOLVIMENTO	Agile – Geralmente Scrum e Kanan (Depende do Cliente)
FERRAMENTAS DE GESTÃO DE PROJETO	Gira (Depende do Cliente)
OUTROS	

QUADRO C – PLANO DE TRABALHOS

PLANO DE TRABALHOS (**9 MESES**)		
ITENS DE AÇÃO (ACRESCENTAR LINHAS DE ACORDO COM A NECESSIDADE)	DATA ÍNICIO	DATA FIM
Onboarding Empresa	01/09/2020	08/09/2020
Onboarding Projecto Cliente		
Avaliação da continuidade na empresa		

QUADRO D – LOCAL DE TRABALHO

LOCALIDADE	Leiria
SÃO ESPERADAS DESLOCAÇÕES A CLIENTES/FORMAÇÃO NAS SEGUINTES LOCALIDADES	Qualquer lado no mundo
SÃO ESPERADOS PERÍODOS DE DESLOCAÇÕES DE (DIAS)	5

QUADRO E – RECURSOS ALOCADOS AO ESTAGIÁRIO

COMPUTADOR DE MESA	
COMPUTADOR PORTÁTIL	X
INTERNET	X
OUTROS	X
RECURSOS ACESSÍVEIS AO ESTAGIÁRIO NO LOCAL DE TRABALHO	
BAR	X
REFEITÓRIO	
MÁQUINA DE CAFÉ	X
OUTROS	X

QUADRO F – CONDIÇÕES PARA A REALIZAÇÃO DO ESTÁGIO

TEMPO CONCEDIDO AO ESTAGIÁRIO PARA	
ELABORAÇÃO DO RELATÓRIO DE ESTÁGIO (HORAS/SEMANA)	2
CONTACTO COM O ORIENTADOR NO IPLEIRIA (HORAS/SEMANA)	2

QUADRO G – FORMAÇÃO INTEGRADA

ESTÁGIO CONTEMPLA FORMAÇÃO NA EMPRESA/ORGANIZAÇÃO (DIAS)	15
ESTÁGIO CONTEMPLA FORMAÇÃO EXTERIOR À EMPRESA (HORAS/DIAS)	0

QUADRO H – CONDIÇÕES MATERIAIS

ESTÁGIO REMUNERADO (VALOR MENSAL)	700€ Brutos
APOIOS PARA DESLOCAÇÕES RESIDÊNCIA-TRABALHO	NÃO
SUBSÍDIO DE ALIMENTAÇÃO	7,63€ / Dia em Cartão Refeição
SEGUROS	SIM
OUTROS	

QUADRO I – SUPERVISOR NA EMPRESA/ORGANIZAÇÃO

NOME EMPRESA/ORGANIZAÇÃO	xgeeks Portugal Lda
MORADA POSTAL	Edifício Moagem, Rua Camilo Korrodi, 62 – 2400-111 Leiria
NOME SUPERVISOR	Luís Soares
MÓVEL SUPERVISOR	910166200 (Romeu Paz)
EMAIL SUPERVISOR	l.soares@xgeeks.io (**Candidaturas usar:** contact@xgeeks.io)

QUADRO J – POSSIBILIDADE DE PERMANÊNCIA NA EMPRESA/ORGANIZAÇÃO

EXISTE A POSSIBILIDADE DE PERMANECER NA EMPRESA/ORGANIZAÇÃO NO FINAL DO ESTÁGIO	SIM