

Universidade de Trás-os-Montes e Alto Douro

Desenvolvimento de front-end para aplicações web: estágio na empresa Mindera

Relatório de Estágio de Mestrado em Multimédia



Aluna:

Sofia Bárbara Pires Gomes

Orientador: Emanuel Soares Peres Correia

Coorientador: Telmo Miguel Oliveira Adão

Vila Real, 2019

Universidade de Trás-os-Montes e Alto Douro

Desenvolvimento de front-end para aplicações web: estágio na empresa Mindera

Relatório de Estágio de Mestrado em Multimédia



Aluna:

Sofia Bárbara Pires Gomes

Orientador: Emanuel Soares Peres Correia

Coorientador: Telmo Miguel Oliveira Adão

Vila Real, 2019

Agradecimentos

Após um longo percurso académico e o período de estágio, este será um marco importante na minha vida académica. Tenho a agradecer a todas as pessoas que passaram pela minha vida durante estes anos, pois, de alguma forma, contribuíram para me tornar no que sou hoje.

Quero deixar um agradecimento especial aos meus pais, por toda a paciência, ajuda e apoio incondicional que me tem dado ao longo da minha vida. Este agradecimento é, também, direccionado a minha irmã Cátia pelos inúmeros conselhos, o acompanhamento e a preciosa ajuda para todas as etapas da minha vida. Sem eles sei que não era possível ter chegado ao fim desta etapa.

Ao Aléxis Santos por ter que me ouvir com as minhas “dúvidas existenciais”, pelas longas horas de companhia durante a escrita deste relatório e, acima de tudo, por me fazer acreditar nas minhas capacidades.

À Mindera e aos seus fundadores por terem criado uma empresa excecional, que proporciona um bom ambiente e todos os meios para os colaboradores serem excelentes no seu trabalho.

À equipa da 106 por toda a ajuda e respostas as mil questões que fiz todos os dias, pelo esforço e pela dedicação que são postos no trabalho feito. Por me fazerem sentir parte da família desde o dia um, pela imensa vontade de fazer mais e melhor, por estarem dispostos a ensinar e ainda sempre pontos para aprender. Para além disso, por terem o grande trabalho de me mostrar o mundo de React.

Ao Tiago Nunes por acompanhar todo o meu percurso durante o estágio, pelas rápidas ajudas as questões que tive e pela exigente tarefa de me orientar no âmbito da empresa.

Aos meus orientadores Emanuel Peres e Telmo Adão, por toda a ajuda prestada, pela orientação e pela disponibilidade durante toda esta jornada. Ao professor Emanuel Peres, em especial, por me incentivar a seguir a área de desenvolvimento *web*.

Por fim, um obrigado, ao professor Pedro Mestre, que me fez descobrir que JavaScript “não é Japonês, apenas um bocadinho de Chinês”, e por, desta forma, me ter ensinado a gostar desta linguagem.

Sofia Gomes

Vila Real, 28 de Outubro de 2019

Resumo

A tecnologia tem evoluído, de forma rápida e diversificada, em todas as suas áreas de aplicação. Relativamente ao desenvolvimento *web*, cada vez mais os utilizadores pretendem obter informação o mais rápido possível, em qualquer lugar e em qualquer tipo de dispositivo.

Com base no estágio realizado na empresa Mindera foi possível perceber quais as técnicas e as metodologias utilizadas para o desenvolvimento *front-end* de aplicações *web* e de que forma é que este desenvolvimento acompanha a rápida evolução do mercado. Devido a cláusulas de confidencialidade contratuais, tanto o cliente como o nome do projeto não serão mencionados neste documento.

As aplicações *web*, têm-se vindo a transformar à medida do utilizador, existindo uma maior preocupação, quando é feito o desenvolvimento destas aplicações, com a forma como o utilizador vai ver e interagir com esta. Para além deste motivo influenciar a maneira como pensamos e executamos as aplicações, as equipas e os processos a que estas recorrem para o desenvolvimento são, também, afetados.

As equipas de desenvolvimento adaptaram-se ao mercado e utilizam formas ágeis de delinear e de executar o seu trabalho para que, se existir uma alteração de requisitos, esta não tenha um grande impacto no que foi delineado. Outra das formas de adaptação está relacionada com a integração partilhada do trabalho de todos os membros da equipa, de forma a que seja possível estar a par de todo o desenvolvimento feito até ao momento.

Neste documento será apresentada a metodologia de trabalho utilizada, bem como a *framework* com que foi aplicada no decorrer do meu estágio curricular na empresa Mindera, focando, sobretudo, no desenvolvimento de *front-end* para aplicações *web*. Para além, disso vão ser descritas algumas das ferramentas e tecnologias utilizadas no desenvolvimento da aplicação em questão. Por fim, será abordado o projeto em que fui inserida na duração do estágio e algumas das tarefas realizadas.

Palavras-chave: Desenvolvimento *front-end*, Agile, Scrum, JavaScript, React.

Abstract

Technology has evolved rapidly and diversely in all of fields of application. Regarding web development, users more and more want to get information as quickly as possible, anywhere and on any device.

Based on the internship at Mindera, it was possible to understand which techniques and methodologies are used for front-end web application development and how this development follows the fast evolution of the market. Due to contractual confidentiality clauses, neither the client nor the project name will be mentioned in this document.

Web applications have been changing to match the need of the user, and there is a greater concern when developing these applications, with the way the user will see and interact with it. Beyond this reason, the way we think and execute the applications, as well as the teams and the processes they use for development are also affected.

Development teams have adapted to the market and use agile ways of planning and execution of their work so that if there are requirements changing, it will not have a big impact on what was arranged. Another form of adaptation is related to the shared integration of the work of all team members, so that it is possible to be aware of all the development done so far.

In this document, I will present the working methodology used, as well as the framework with which it was applied during my internship at Mindera, focused mainly on the development of front end for web applications. In addition, some of the tools and technologies used in developing the application in question will be described. Finally, will be approached the project I was inserted in the duration of the internship and some of the tasks that were developed.

Keywords: Front-end development, Agile, Scrum, JavaScript, React.

Índice

AGRADECIMENTOS	I
RESUMO	III
ABSTRACT	V
ÍNDICE	VII
ÍNDICE DE FIGURAS	IX
ÍNDICE DE TABELAS	XI
GLOSSÁRIO	XIII

CAPÍTULO I

1	INTRODUÇÃO	1
1.1	Objetivos	3
1.2	Enquadramento	4
1.2.1	Mindera	4
1.2.2	Cliente	5
1.3	Estrutura do relatório	6

CAPÍTULO II

2	METODOLOGIAS E CONTINUOUS INTEGRATION	9
2.1	Metodologia Agile	9
2.2	Scrum	12
2.2.1	Equipa Scrum	13
2.2.2	Eventos	13
2.2.3	Artefactos	14
2.3	Mindera Scrum	15
2.4	Continuous Integration	19
2.5	Controlo de Versões	21
2.5.1	Git	23
2.5.2	GitLab	24

CAPÍTULO III **27**

3	DESENVOLVIMENTO DE FRONT-END	27
3.1	Ferramentas de apoio ao desenvolvimento web	27
3.1.1	Node.js	28
3.1.2	NPM	28
3.1.3	ESLint	29
3.2	Tecnologias de suporte ao desenvolvimento de front-end	29
3.2.1	HTML	29
3.2.2	CSS	30
3.2.3	JavaScript	30
3.2.4	React	31
3.2.5	Redux	33
3.2.6	Lodash	35
3.3	Testes	35
3.3.1	Jest	36
3.3.2	Enzyme	37

CAPÍTULO IV **39**

4	PROJETO	39
4.1	Enquadramento	39
4.2	Tarefas de integração	40
4.3	Bugs	47
4.4	Tasks	49
4.5	Acessibilidade	55

CAPÍTULO V **61**

5	CONCLUSÕES	61
	REFERÊNCIAS BIBLIOGRÁFICAS	63

Índice de figuras

<i>Figura 1 - Scrum process [13].</i>	15
<i>Figura 2 – User Story</i>	16
<i>Figura 3 - Votação PlanITPoker</i>	17
<i>Figura 4 – Sprint backlog em swimlane</i>	17
<i>Figura 5 – Mindera Scrum</i>	18
<i>Figura 6 – Controlo de versões</i>	22
<i>Figura 7 - Sistema de guardar dados em versão de diferenças (delta-based) [18].</i>	23
<i>Figura 8 - Sistema de guardar dados em snapshots [18].</i>	24
<i>Figura 9 – DevOps ciclo [19].</i>	24
<i>Figura 10 – GitLab tags</i>	25
<i>Figura 11 - CSS</i>	30
<i>Figura 12 – Render com a Virtual DOM [30].</i>	32
<i>Figura 13 – JSX vs JavaScript [32].</i>	33
<i>Figura 14 – Utilização de Redux no desenvolvimento de aplicações com JavaScript [33].</i>	34
<i>Figura 15 – Pirâmide de testes [36].</i>	35
<i>Figura 16 – Order Confirmation, Amazon</i>	42
<i>Figura 17 – Popular products inspired by this item, Amazon</i>	45
<i>Figura 18 - Componente em grandes breakpoints</i>	50
<i>Figura 19 – Componente em pequenos breakpoints</i>	51
<i>Figura 20 – Página dos detalhes de um produto, Amazon</i>	53
<i>Figura 21 – Headings Página UTAD</i>	58
<i>Figura 22 – Descrição do problema de contraste de cor na página da UTAD</i>	59
<i>Figura 23 – Resultado contraste de cor</i>	60

Índice de tabelas

<i>Tabela 1 - Cronograma</i>	40
------------------------------	----

Glossário

ADA – Americans with Disabilities Act
API – Application Programming Interface
B2C – Business to Consumer
CD – Continuous Delivery
CI – Continuous Integration
CSS – Cascading Style Sheets
DOM – Document Object Model
E-commerce – Electronic commerce
HTML – HyperText Markup Language
JS – JavaScript
JSX – JavaScript XML
Npm – Node Package Manager
PO – Product Owner
Props – Properties
QA - Quality Assurance
UI – User Interface
UTAD – Universidade de Trás-os-Montes e Alto Douro
UX – User Experience
VDOM – Virtual DOM
WCAG – Web Content Accessibility Guidelines
Web – World Wide Web.

Capítulo I

Introdução

O trabalho foi desenvolvido no âmbito da unidade curricular “Dissertação/Projeto/Estágio”, do Mestrado em Multimédia. Com efeito, este ciclo de estudos apresenta, no seu plano de estudos, a possibilidade de realizar um estágio curricular em empresa, com a duração máxima de seis meses.

O estágio teve lugar na empresa Mindera Software Engineering, sob a orientação do Professor Emanuel Peres, coorientação do Doutor Telmo Adão e o acompanhamento, na empresa, do Tiago Nunes.

O principal objetivo do estágio foi o desenvolvimento de *front-end* para aplicações *web*. Esta opção foi selecionada pois permite um contacto mais próximo com o mundo empresarial, proporcionando a aquisição de novos conhecimentos que podem ser aplicados a produtos digitais comumente desenvolvidos.

O desenvolvimento *web* consiste em considerar, numa aplicação *web*, vários elementos, tais como estrutura, dados, *design*, conteúdo e funcionalidade. Esta área pode ser subdividida em duas principais: o *front-end / client-side* e o *backend / server-side*, sendo que as duas têm igual importância para o funcionamento de uma aplicação *web*. [1]

No que respeita ao *front-end*, neste é desenvolvida a estrutura para a componente mais visual e interativa do sítio *web*.

Com base em designs desenvolvidos para um dado projeto processe-se à criação e ao desenvolvimento de cada elemento que os constitui e/ou dos comportamentos que se pretende disponibilizar aos utilizadores. Acresce, ainda, a adequação de conteúdos multimédia - imagens, vídeos, áudio, entre outros – ao que o designer pretende ver

implementado. Por fim o desenvolvimento de *front-end* tem, também de agregar todas as interações que os elementos desenvolvidos possam ter com os utilizadores, o tratamento dos dados recebidos e, ainda o tratamento de dados das interações que o utilizador realizou.

Cada componente desenvolvido tem de ser sujeito a um controlo de qualidade, nomeadamente no que concerne ao seu funcionamento sem erros e às questões de compatibilidade com navegadores, sistemas operativos e respetivas versões, assim como resoluções de ecrãs.

A aplicação web surge do desenvolvimento de vários elementos e de diferentes tecnologias de programação como, por exemplo, o HTML, as Css e o JavaScript. Estes elementos constituem a aplicação web: um produto digital que o utilizador vê e com a qual consegue interagir, num determinado contexto. Para que uma aplicação *web* possa ser considerada com qualidade, alguns fatores são essenciais:

- Um sítio *web* deve ser responsivo. Assim existe a possibilidade do seu conteúdo se adaptar aos vários tipos de dispositivos, tendo em conta o tamanho do ecrã, mas também à plataforma e à orientação do dispositivo que o utilizador está a utilizar para interagir com a aplicação;
- A interação que o utilizador tem com os vários componentes da interface, no sentido de a tornar mais intuitiva e rápida. Para isso os *designs* devem ser perceptíveis para ao utilizador alvo: dessa forma, este conseguirá identificar qual o comportamento que é suposto existir e como pode interagir com um determinado componente da aplicação;
- A usabilidade da interface. Esta mede o grau pelo qual os utilizadores conseguem realizar uma interação, se a rapidez com que o utilizador aprende a utilizar uma funcionalidade for grande. Assim, significa que a aplicação tem uma usabilidade muito relacionada ao comportamento humano típico.

A minha motivação para a realização deste estágio foi a de conhecer as empresas da área do desenvolvimento *web*, as suas necessidades e os requisitos que exigem aos seus profissionais. Pretendi, ainda, aprender e trabalhar com tecnologias e utilizar metodologias, muitas vezes recentes, com as quais o contacto tinha sido mínimo ou

inexistente ao longo do meu percurso académico, mas que considerava necessárias ao meu futuro no meio empresarial.

1.1 Objetivos

Os objetivos deste estágio foram:

- Entender e participar no desenvolvimento de aplicações para a web, desde a sua conceção até à entrega do produto final. Para isso, a Mindera disponibilizou-se a integrar-me num projeto que estava numa fase relativamente inicial, permitindo-me passar por quase todas as suas fases de inicialização e desenvolvimento.
- Utilizar a metodologia de trabalho Agile, com o recurso à *framework* Scrum. Numa primeira fase e para conseguir entender melhor a aplicação do Scrum e a sua integração na metodologia, apenas participei nos eventos de Scrum como membro não ativo. Isto permitiu-me entender a dinâmica da equipa em cada evento e também nos momentos de desenvolvimento, e ainda perceber em que consistia cada um dos eventos. Mais tarde, numa fase em que já compreendia melhor todo o funcionamento do Scrum com a equipa e conhecia melhor o projeto, passei a integrar os eventos como membro ativo, realizando, por exemplo, parte da votação do esforço de cada tarefa. Esta integração gradual foi muito benéfica.
- Aplicação de tecnologias como React e Redux, que foram o ponto principal no desenvolvimento do projeto. O React como biblioteca de JavaScript para a construção de interfaces para o utilizador e que está na base de quase todos os componentes que integram o sítio *web*. O Redux, como biblioteca de JavaScript para controlar o estado da aplicação, que permite saber qual o estado da aplicação e dos componentes que a compõem a qualquer momento. Numa fase inicial de contacto com estas tecnologias, foi necessária a leitura de documentação e realizar alguns tutoriais.
- Utilização da metodologia de desenvolvimento de aplicações “*code review*”. O *code review* permite que os membros pertencentes ao projeto analisem o código desenvolvido pelo elemento que pretende submeter o

seu código no repositório do projeto, para assim resolver uma determinada tarefa. Esta metodologia faz com que o código submetido tenha menos erros e, ainda, que se possam encontrar novas formas de pensamento ou de resolução de problemas, tornando, assim, o código mais eficiente.

- Utilização da metodologia de desenvolvimento de aplicações “*unit testing*” ou testes unitários. Trata-se de uma forma de testar o código criado, desenvolvendo testes com a tecnologia Jest. Estes vão verificar se o comportamento de um componente ou de uma unidade está a ser o esperado, considerando os requisitos de desenvolvimento. A utilização desta metodologia permite que sejam detetados problemas de código numa fase ainda prematura, o que resulta em que o comportamento de cada componente esteja conforme o pretendido.

1.2 Enquadramento

1.2.1 Mindera



A Mindera é uma empresa de desenvolvimento de tecnologia, fundada em 2014 por apenas 5 pessoas e sediada na cidade do Porto. Os fundadores - Bruno Lopes, Guilherme Almeida, José Fonseca, Paul Evans e Sofia Reis - agregaram a sua paixão pelas tecnologias e a sua motivação para criar produtos de qualidade com as pessoas que partilhem da mesma paixão. Assim, criaram uma empresa que tem uma cultura diferente da do mercado e que permite aos seus colaboradores crescerem e criarem produtos que os movem.

Atualmente, o grupo Mindera conta com dez empresas, entre as quais a Mindera PT, Mindera UK, Mindera IN, Mindera US, Lemonworld.

Durante os últimos cinco anos de rápido crescimento, da empresa aumentou a sua representação para 4 diferentes países (Portugal, Estados Unidos, Índia e Reino Unido) e conta um total de mais de 350 colaboradores.

Entre os seus valores encontram-se as pessoas e a paixão pela tecnologia. Um dos valores mais importantes da empresa é descrito pela seguinte frase:

“We use technology to build products we are proud of, with people we love.”

Esta empresa tem uma estrutura organizacional *flat*, que consiste num sistema em que a organização possui com poucos níveis de hierarquia. Ou seja, cada colaborador é menos supervisionado, promovendo, assim, o seu poder de decisão, eliminando algum nível excessivo de gestão e melhorando a coordenação e a rapidez do trabalho a desenvolver.

No centro do negócio da Mindera está o desenvolvimento de produtos para a *web* e o segmento móvel (designado, doravante, por *mobile*). A empresa tem a capacidade para o desenvolvimento de raiz de variados tipos de produtos, desde o conceito, desenho da interface, protótipo, desenvolvimento da plataforma, teste de qualidade, entrega final e a sua manutenção.

O funcionamento é baseado em equipas, onde os colaboradores estão agrupados de acordo com as áreas de trabalho em que operam e em que o(s) projeto(s) em que estão inseridos possam exigir.

No caso da equipa em que foi inserida, esta era constituída por programadores de front-end, de testes de certificação de qualidade (QA's) e ainda *Product Owners* (PO's).

O projeto desenvolvido ao longo do estágio destinou-se a um cliente que opera na área do e-commerce e será detalhado, em maior pormenor, na secção seguinte.

1.2.2 Cliente

Devido a cláusulas de confidencialidade contratuais, tanto o cliente como o nome do projeto não serão mencionados neste documento, sendo referidos sempre como o “cliente” e o “projeto”.

O cliente abordou a Mindera com o intuito de esta desenvolver um sítio *web e-commerce* para o seu negócio de forma a aumentar/reforçar a sua presença *online*. Esta metodologia facilita a compra dos seus artigos aos consumidores, tanto a nível dos países onde o cliente opera com lojas físicas, como a nível de outros países onde está representado e ainda numa escala mais global, onde eventualmente, pode ainda não ter presença.

O *e-commerce* (*electronic commerce*) é uma metodologia de negócio utilizada na venda e na compra de produtos e serviços, utilizando o meio eletrónico (internet) para executar as transações associadas com a compra ou venda [2]. Atualmente verifica-se

uma crescente evolução no *e-commerce* que facilita a descoberta dos produtos e a maneira de efetuar a transação com o utilizador. Uma das grandes vantagens que essa metodologia apresenta, relativamente ao comércio tradicional, é a sua globalização que permite uma escala de abrangência do negócio muito maior [3].

O cliente apresenta um tipo de modelo de negócio B2C (*Business to Consumer*), em que os produtos que estão apresentados no sítio *web* são vendidos diretamente ao consumidor, o processo do negócio consiste num pedido por parte do consumidor para adquirir um determinado produto / serviço / informação através do sítio *web*. Depois de o pedido ser realizado, este é enviado para o comerciante para proceder ao seu envio para o consumidor.

Algumas das vantagens que podemos encontrar nesta metodologia de negócio, para o consumidor são: o serviço de venda está disponível 24h por dia e em várias localizações do mundo; permite ao consumidor mais opções de escolha e possíveis comparações com outros produtos ou marcas.

Ao nível da organização que aplica o modelo, o *e-commerce* possibilita que o negócio seja expandido no mercado nacional e também internacional com um pequeno investimento; grande divulgação da marca da organização; reduzidos custos de distribuição; rápida gestão dos serviços devido a sua simplificação fazendo com que exista maior eficiência. Algumas das desvantagens dizem respeito ao investimento inicial necessário fazer em tecnologias e segurança, bem como, a rápida mudança do meio informático, que provoca a que o sistema rapidamente se torne obsoleto [4].

No capítulo seguinte será abordada a descrição da metodologia de trabalho utilizada para a organização dos projetos da empresa, bem como a framework utilizada para aplicar esta metodologia. Para além disso serão indicadas as adaptações feitas a esta metodologia pela empresa, de forma a que se adeque melhor na sua operação diária.

1.3 Estrutura do relatório

No primeiro capítulo é feita uma introdução ao trabalho que foi desenvolvido e descrito o contexto de trabalho, abordando os tópicos principais descritos como sendo os objetivos deste estágio, bem como, um enquadramento do projeto, da empresa onde realizado e do cliente.

No segundo capítulo são apresentadas as metodologias utilizadas no decorrer do estágio, como, por exemplo, a Agile, descrevendo os seus principais objetivos e de que

forma contribuíram para o processo. Será ainda abordado o tema de *continuous integration*, com a apresentação da técnica e a descrição da sua implementação para o desenvolvimento de aplicações *web*.

No terceiro capítulo descreve-se em que consiste o desenvolvimento de *front-end*, através de uma introdução ao tema. São apresentadas algumas das ferramentas aliadas ao projeto e que permitiram a criação de um ambiente para o desenvolvimento da aplicação para a *web*. De seguida, o enquadramento do desenvolvimento *front-end* e das tecnologias utilizadas durante o estágio.

No quarto capítulo é abordado o projeto desenvolvido durante este estágio, descrevendo-o, apresentando as tarefas implementadas e indicando quais as técnicas e as soluções desenvolvidas.

Por fim, no quinto capítulo, são apresentadas as conclusões deste trabalho, juntamente com a apresentação dos resultados alcançados face aos objetivos propostos e algumas considerações pessoais.

Capítulo II

Metodologias e Continuous Integration

Neste capítulo, será abordada a metodologia Agile, que constitui a base do método de trabalho utilizado na Mindera. Neste contexto, foi necessária a utilização da *framework Scrum*, no sentido de permitir que a estrutura descrita na Agile fosse implementada. Esta *framework* para além de ajudar na gestão da equipa e do projeto, também promove a resolução de problemas que possam existir ao longo de todo o processo de desenvolvimento.

2.1 Metodologia Agile

A metodologia Agile é, de uma forma muito geral, uma abordagem de desenvolvimento que suporta a flexibilidade e a rápida resposta à mudança [5].

Muitos projetos desenvolvidos no mundo tecnológico têm como base a metodologia Agile para o seu desenvolvimento: Agile é um conjunto de métodos e práticas baseadas em princípios e valores escritos no “Manifesto para o desenvolvimento de *software* Agile” de 2001, que teve dezassete assinantes [6].

Na base desta metodologia encontram-se quatro valores que a Agile utiliza para ajudar no desenvolvimento de *software*, seguindo melhores formas para a realização dos processos, e para que estes sejam valorizados. Os valores são os seguintes:

- **“Os indivíduos e suas interações** acima de procedimentos e ferramentas;
- **O funcionamento do *software*** acima de documentação abrangente;

- **A colaboração com o cliente** acima da negociação e contrato;
- **A capacidade de resposta a mudanças** acima de um plano pré-estabelecido;”

O texto apresentado a negrito seleciona os valores que tem mais importância e mais peso na equação da Agile, segundo os seus autores [6].

Isto é, a metodologia baseia-se nos valores referidos para que os processos sejam mais ágeis e aplicando-os permite que a flexibilidade e a colaboração de todos os envolvidos nos processos seja facilitada, relativamente a um planeamento clássico e rígido que tem de ser seguido e que não tem em conta os vários *inputs* que podem surgir ao longo do projeto.

O foco da Agile são as pessoas que desenvolvem uma determinada tarefa e que trabalham em equipa seguindo certas práticas para realizar um projeto.

Paralelamente também defende a importância de satisfazer os pedidos do cliente com a entrega de *software* funcional, tendo uma colaboração com o mesmo num espaçamento curto de períodos de tempo e aceitando alterações em requisitos previamente definidos [7].

Para além dos seus valores base, a Agile também é composta por princípios. Estes complementam a utilização desta metodologia enquadrando os valores e demonstrando como os podem ser aplicados. Os princípios da Agile são os seguintes:

1. Satisfação do cliente através da entrega atempada e contínua de *software* com valor. Isto significa que se deve ter a máxima atenção para entregar o que foi estimado para cada prazo.
2. Adaptação às modificações nos requerimentos durante o processo de desenvolvimento.
3. Entrega de *software* funcional frequentemente, com uma escala pequena de tempo.
4. Promover a partilha de informação com interação cara a cara na equipa de desenvolvimento.
5. Boa colaboração entre o parceiro comercial e a equipa de desenvolvimento durante o projeto.
6. Fazer o projeto com pessoas motivadas e com um ambiente de suporte e confiança para que as pessoas fiquem envolvidas no mesmo.
7. O fator final que mede o progresso alcançado é a entrega de *software* funcional.

8. Utilização dos processos Agile para assim suportar um ritmo de desenvolvimento constante.
9. Simplicidade.
10. Atenção ao detalhe técnico e de design para aumentar o nível de agilidade.
11. Equipas que se auto-organizam promovem o desenvolvimento de uma boa estrutura, requerimentos e design.
12. Reflexões regulares de como tornar o trabalho mais eficiente [8].

Os princípios enumerados ajudam a perceber que, para além de a Agile ser uma metodologia, acaba por ser, também, um tipo de mentalidade, porque ao aplicar estes princípios está-se a fazer com que a equipa modifique a sua mentalidade e que comece a pensar de uma forma em que os membros partilham a informação com a equipa, podendo, assim, tomar decisões juntos sobre a melhor forma para aplicar essas decisões. Numa metodologia tradicional cabe apenas ao gestor da equipa tomar estas decisões [9].

Algumas das vantagens que obtemos ao utilizar a metodologia Agile são:

- **Flexibilidade** – a Agile é baseada em conseguir adaptar o projeto as modificações necessárias. Durante o processo de construção e entrega do projeto, este deve conseguir reagir e adaptar-se ao mercado e ao resto das mudanças.
- **Velocidade do mercado** – com esta metodologia podemos, rapidamente, mostrar o conceito do projeto aos utilizadores, devido à entrega de *software* funcional no fim de cada *sprint*. Adapta-se facilmente o projeto à rápida mudança do mercado.
- **Qualidade** – Agile integra vários testes ao longo do processo. O software entregue é testado constantemente o que significa que a qualidade geral aumenta ao longo da construção do projeto, admitindo, assim, que não se despenda de tanto tempo, no fim, para certificar que a qualidade da aplicação é boa.

- **Produto certo** – tal como referido no ponto anterior, ao longo do processo são feitos testes para verificar a qualidade do mesmo. Através desta metodologia torna-se mais fácil identificar imperfeições e coisas que possam ser melhoradas, interagindo com o produto entregue.
- **Transparência** – Agile deixa ver, sentir e usar o projeto constantemente durante todo o seu processo. Não se vêem as coisas compartimentadas: ao longo do processo já se percebe como estas funcionam todas juntas [10].

2.2 Scrum

Scrum é uma *framework* que tem como base os métodos da Agile e, é utilizada na gestão e resolução de problemas complexos, com o intuito de ajudar: a (I) melhorar o produto que está a ser criado; (II) a equipa que está a desenvolver o mesmo e (III) o ambiente de trabalho desta equipa ao longo do tempo de desenvolvimento.

Existem 3 valores que estão na estrutura do processo Scrum, e que se pode verificar terem sido adotados à metodologia Agile. São eles a transparência, a inspeção e a adaptação:

- **Transparência**, consiste em que todos os elementos tenham um entendimento comum dos vários aspetos que compõem o projeto.
- **Inspeção** permite verificar se existe alguma variação entre os processos relativamente ao objetivo para a *sprint*.
- **Adaptação**, se algum dos processos se desviou relativamente ao objetivo da *sprint*, para retificar e ajustar o mais rapidamente possível de forma a minimizar os desvios do objetivo.

Esta *framework* é formada por 3 componentes: a equipa Scrum e as funções dos seus intervenientes, os artefactos e os eventos [11].

2.2.1 Equipa Scrum

A equipa Scrum é autossuficiente para se poder organizar e gerir a melhor forma para que, no final da *sprint*, o trabalho seja entregue, esta é constituída por 3 elementos de grande importância: o dono do produto / *product owner* (PO), o *Scrum master* e a equipa de desenvolvimento.

O **PO** é uma e apenas uma pessoa: ela representa o utilizador final e as suas necessidades. É responsável pela gestão do *backlog*, ordenando e reorganizando as tarefas conforme a sua prioridade, de forma que os objetivos do projeto ou da *sprint* sejam atingidos. Para além dos aspetos referidos anteriormente, esta pessoa também faz com que todos os itens do *backlog* sejam claros, de forma a serem entendidos por toda a equipa da mesma maneira. Caso isto não aconteça, cabe ao PO explicar à equipa o que se pretende de cada um dos itens e garantir que a definição é entendida por toda a equipa.

A **equipa de desenvolvimento** é uma equipa multifuncional e auto-organizada. O seu objetivo é trabalhar em conjunto para que, no final da *sprint*, as tarefas sejam entregues como “*done*”, entregando ao cliente um produto em que se podem visualizar as funcionalidades desenvolvidas, em funcionamento. Isto permite que o cliente siga o processo de desenvolvimento de uma forma mais envolvida, pois consegue visualizar as tarefas realizadas e propor alterações, nas diversas fases do projeto. Desta forma as alterações a aplicar serão mais fáceis de integrar no planeamento do projeto.

O último interveniente da equipa Scrum é o **Scrum Master** (mestre Scrum) que é quem transmite e ajuda os outros intervenientes a entenderem todos os conceitos do Scrum, este elemento ajuda a equipa Scrum a encontrar a melhor técnica para a gestão do *product backlog*, organizando reuniões e o acompanhando o trabalho da equipa de desenvolvimento.

2.2.2 Eventos

A **sprint** é o ponto central da *framework* Scrum. Trata-se de um período de tempo, definido no início do projeto, nunca superior a um mês. Durante esse tempo, a equipa Scrum tem que criar um produto que possa ser executado/funcional (incremento) e que cumpra os objetivos definidos para essa *sprint*. A nova *sprint* apenas começa quando a antiga for fechada.

Durante o tempo de uma *sprint* são realizados vários eventos: primeiro, o planeamento da *sprint* (**Planning**). Nesta reunião traça-se um plano para a nova *sprint*, definindo o que será entregue no incremento (objetivo da *sprint*/ *sprint goal*) e são determinados os itens que serão selecionados para a *sprint* (*sprint backlog*). A duração máxima desta reunião é definida no início do projeto e é mantida até à sua conclusão. Normalmente, a duração máxima é de 8 horas, para uma *sprint* com a duração de 1 mês. Esta duração deve ser adaptada tendo em conta a relação referida anteriormente (8 horas para 1 mês).

No decorrer da *sprint* a equipa tem uma reunião diária (**Daily**) que é limitada a um tempo máximo de 15 minutos. tem como grandes vantagens, uma melhor colaboração e um melhor desempenho da equipa. Durante a reunião, os elementos comunicam entre si, as tarefas que realizaram no dia anterior, o que vão fazer durante o dia e se tem algum impedimento, ou se precisam de alguma ajuda para a sua tarefa, de forma a que os membros da equipa possam contribuir, em caso de necessidade.

No final de cada *sprint*, é realizada uma revisão (**Review**), em que a equipa Scrum revê se o incremento está conforme o planeado para o *sprint goal* e adapta o *product backlog*, se for necessário. A *review* está limitada a 4 horas para a *sprint* de 1 mês, fazendo a adaptação para *sprints* mais pequenas.

O próximo evento é o momento em que a equipa faz uma introspeção e que define um plano para aperfeiçoar a próxima *sprint*, a retrospectiva (**Retrospective**), que tem um limite de tempo máximo de 3 horas para uma *sprint* de 1 mês, sendo ajustável, tal como os outros eventos já referidos. Nesta fase, examina-se o que correu bem e o que precisa de ser melhorado relativamente à última *sprint*, no que respeita às pessoas, relações, processos e ferramentas, e cria-se um plano para melhorar as *sprints* futuras [7].

2.2.3 Artefactos

Nos artefactos encontramos o **Backlog**. Este agrupa as tarefas e os requisitos do projeto e tem que ser constantemente priorizado. Existem dois tipos de *backlog*: o *product backlog* e o *sprint backlog*.

Product Backlog é uma lista de todas as funcionalidades do projeto, que pode ser atualizada por qualquer membro da equipa (adicionando mais funcionalidades), evoluindo paralelamente com o projeto.

O **Sprint Backlog** inclui todas as funcionalidades / requisitos selecionados do *product backlog* para realizar na *sprint* que irá decorrer que serão entregues no

incremento, apenas se podem fazer alterações ao *sprint backlog* caso estas não influenciem o *sprint goal* [12].

O último artefacto trata-se do **Incremento**, que é uma junção de todos os requisitos que foram desenvolvidos durante a *sprint*, juntamente com os outros incrementos já concluídos. Todas as tarefas têm de ter o estado de *done* para fazerem parte do incremento. Além disso, o incremento tem que estar funcional, e é apresentado ao cliente para este verificar se cumpre os requisitos pretendidos [11].

A Figura 1 é apresentado todo o processo Scrum, tal como descrito anteriormente.

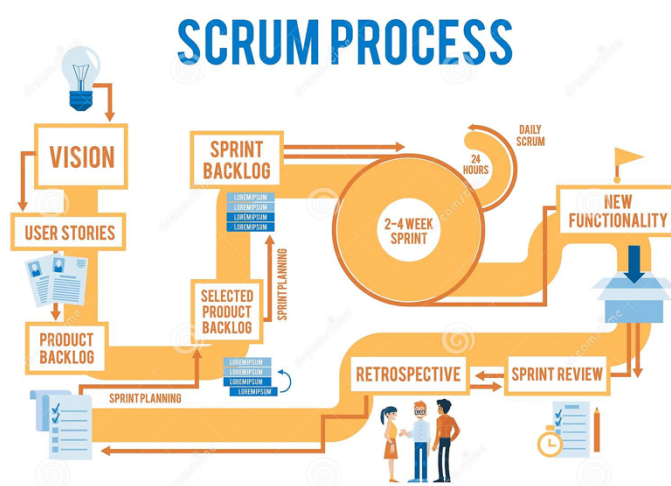


Figura 1 - Scrum process [13].

2.3 Mindera Scrum

Na Mindera verificou-se que o Scrum, tal como descrito pelos seus fundadores, não se enquadrava, completamente, nas necessidades da empresa e dos seus projetos. Por isso, foi necessário adaptar o Scrum dando origem ao “Mindera Scrum”.

No início de qualquer projeto os elementos, ou parte dos elementos que vão pertencer a equipa de desenvolvimento, fazem uma reunião com o PO, o Scrum *Master* e ainda com o cliente, com o propósito de dividir o projeto em *User Stories*. Para isso, utilizam um tipo de categorização chamado *t-shirt size*. Esta é uma técnica de estimativa relativa dos *story points*, através de tamanhos de *t-shirt* (XS, S, M, L, XL) tendo em conta o esforço necessário para desenvolver aquele requisito. A técnica é muito utilizada, porque faz com que o processo de estimar um item a longo termo seja mais rápido e mais fácil. No fim deste processo consegue-se obter uma estimativa do tempo necessário, quantas pessoas vão ter que compor a equipa e dos recursos necessário

para todo o processo de desenvolvimento. Além disso, consegue-se obter, também, as *User Stories* que vão compor o projeto.

Numa *user story*, como é apresentada na Figura 2, pode-se ver quem pretende esta tarefa (“As a”), o objetivo final da *story* (“I want”), onde vai ser aplicada (“So I”), os critérios que esta tarefa tem para ser aceite e finalizada (“Acceptance Criteria”) e ainda os *designs* que lhe estão associados, para quando este componente for desenvolvido saber-se o que é suposto existir a nível visual (“Designs”).

```
As a CMS user
I want to be able to add a double image component
So I can customize the      page

Acceptance Criteria

The double image block should contain 3 images:

• square image
• longer image
• taller image

The following visualisation properties should be added:

• Flipped or mirrored (depending on the breakpoint)

Note: ratios are forced

Designs:
https://app.zeplin.io/project/
```

Figura 2 – User Story

As *user stories*, do projeto encontram-se disponíveis na plataforma Jira. No que respeita aos *designs* fez-se uso do software Zeplin. Estas plataformas são partilhadas com toda a equipa, de forma a que todos os elementos estejam a par de: o que está a ser desenvolvido, o que ainda vai ser desenvolvido, quem está responsável por uma determinada tarefa, entre outros elementos. As plataformas ajudam a que todos os elementos da equipa entendam e estejam a par dos desenvolvimentos das várias tarefas, facilitando que se tenha o mesmo entendimento, tal como é descrito na Agile, a todo e qualquer momento do projeto.

Ao conjunto das *user stories* dá-se o nome de *Product Backlog*. Este contém todos os requisitos para o projeto. Uma vez por semana é feita uma *Backlog Refinement Session*, onde são estimados os pontos de esforço para as *user stories* definidas para aquela reunião. Nessa sessão são utilizadas as tecnologias: Jira e Zeplin (referidas anteriormente), que mostram a toda a informação necessária das *user stories*.

Utiliza-se ainda, um outro *software* em que podem ser estimados os pontos de esforço para uma determinada tarefa: PlanITPoker. Uma das vantagens em utilizar este

software é que todos os membros podem votar, de uma forma secreta. No final da votação revelam-se os votos e a maioria decide qual o número de pontos atribuído a essa tarefa (Figura 3). Caso não exista maioria procede-se a uma nova votação.

triple image	3
double image	3
Text components	3

Figura 3 - Votação PlanITPoker

Com todas as *user stories* entendidas da mesma forma por toda a equipa e estimados os pontos de esforço, ocorre a *Planning Session*. Trata-se de uma reunião realizada de 15 em 15 dias para atribuir, a cada elemento da equipa de desenvolvimento, as tarefas seleccionadas pelo PO do *backlog*. Nesta é apenas utilizado o *software* Jira. No final desta reunião obtém-se o *sprint backlog*, como pode ser observado na Figura 4.

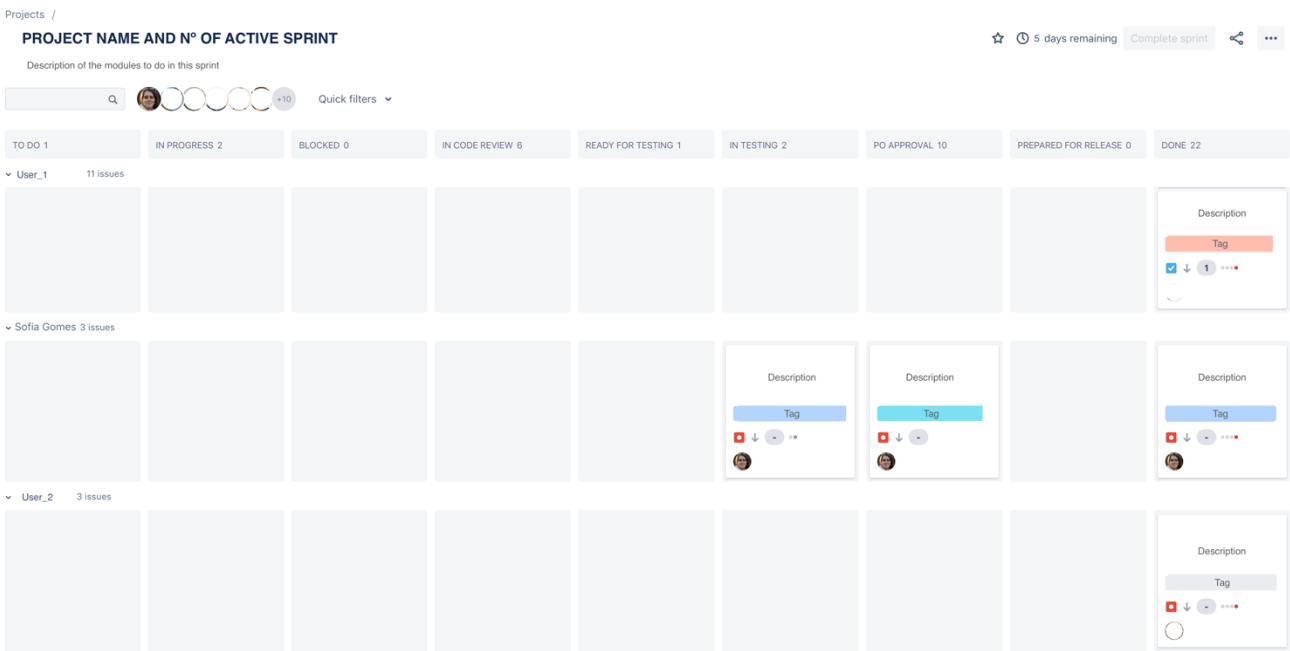


Figura 4 – Sprint backlog em swimlane

Na Figura 4 pode-se observar um *sprint backlog* que está configurado em *swimlane*, o que significa que se tem uma categorização horizontal das *user stories* que se encontram na *sprint* ativa do Scrum board. Esta categorização pode ser feita por *tags*, utilizador, *queries*, tipo de tarefa, entre outros.

Na figura vê-se uma *swimlane* categorizada por utilizador, a quem foi atribuída a *user story*. Desta forma é muito mais rápido de identificar claramente quais as tarefas que estão atribuídas a cada pessoa e conseguir seguir facilmente em que ponto do *workflow* se encontra cada tarefa sob a sua responsabilidade.

Dá-se então início à *sprint* que dura durante 2 semanas e diariamente realiza-se uma *Daily Standup*, onde todos os elementos da equipa Scrum falam sobre o que fizeram no dia anterior, o que vão fazer nesse dia, e se existe alguma dificuldade/impedimento no seu trabalho, sendo utilizado para esta reunião o *software zoom.us*.

No final da *sprint* a equipa desenvolveu um incremento que é mostrado ao cliente numa *Demo Session*. Nesta, o cliente tem a oportunidade de dar o seu *feedback* sobre o incremento e propor modificações que vão de encontro aos requisitos que pretende.

Após o *feedback* e a finalização da *sprint*, é feita uma *Retrospective Session* (normalmente realizada de 15 em 15 dias) para avaliar o que correu bem durante a *sprint* que passou, considerando os pontos melhores e piores dessa *sprint*, e quais as ações que devem ser aplicadas para melhorar as próximas *sprints*. Utiliza-se a tecnologia FunRetro para este efeito. A Figura 5 apresenta o Scrum adaptado às necessidades e à realidade da Mindera.

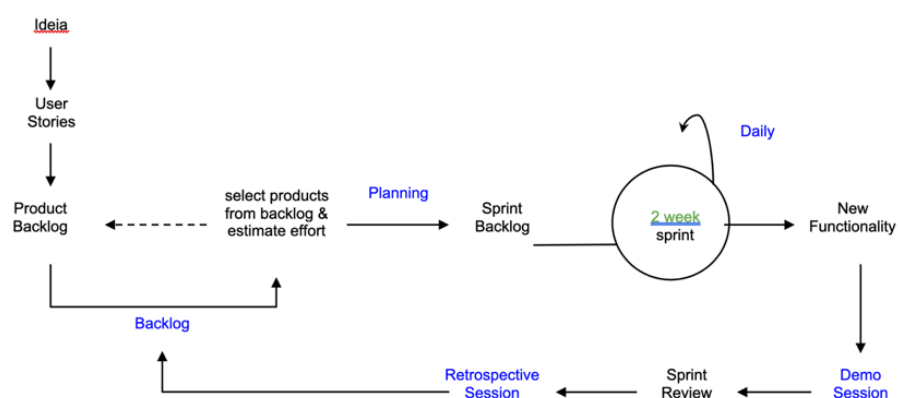


Figura 5 – Mindera Scrum

2.4 Continuous Integration

Continuous Integration (CI) consiste na validação do software mal este chega à versão de controlo que se encontra num repositório. Isto permite garantir que o software funciona e vai continuar funcional mesmo quando novo código for introduzido, mantendo a sua integridade ao longo de todo o processo de desenvolvimento e integração. Quando se faz um uso correto da CI, normalmente esta encontra-se aliada também a outra técnica que facilita a entrega de um produto funcional: a *Continuous Delivery*, que faz com que o software seja *deployed* de uma forma muito mais simples [14].

Alguns dos sinais da adoção da CI é a utilização de práticas, tais como:

- **Utilização de um repositório único e partilhado** – com toda a equipa e qualquer alteração que seja feita ao projeto será submetida nesse repositório. Normalmente, devem ser submetidas as alterações com uma periodicidade diária. Uma das vantagens de utilizar este método é que todos os membros estão a par do que está a ser feito e do que já foi feito podendo consultar a qualquer momento um determinado código. Mas para que este repositório partilhado não se torne confuso com o decorrer do projeto, recorre-se a utilização de ferramentas de controlo de versões que, de uma forma muito geral, registam as alterações que foram feitas, quando, onde e ainda por quem. Com esta ferramenta pode-se, mais facilmente, contornar qualquer erro que possa existir.
- **Automatização da construção e lançamento do produto** – processo automático das *builds* e *releases*, que consistem na conversão dos ficheiros e outros recursos num *software* correspondente a um produto que possa ser consumido.
- **Testes unitários** – os unitários referem-se a testes que são realizados, pela equipa de desenvolvimento, para poderem testar se os componentes desenvolvidos estão completamente operacionais (*pass*) ou se exigem correções em alguma parte que não está a funcionar em pleno (*fail*). Os testes que revelem o segundo resultado têm que ser revistos no momento em que se descobre essa falha para não se correr o risco de desenvolver mais valências com código que tem falhas [15].

Algumas das vantagens que se encontram com a utilização da CI são: a diminuição do esforço e da duração da integração, principalmente devido ao facto de que as essas integrações acontecerem muito frequentemente, fazendo com que não seja tão difícil integrar pequenas alterações em curtos espaços de tempo comparativamente a fazer grandes integrações em longos espaços de tempo. Utilizado o 2.º método descrito, a integração torna-se mais demorada, não apenas pela quantidade de alterações a integrar, mas, também, pela dificuldade em integrar essas mesmas alterações.

Com a CI é possível desenvolver um produto que pode apresentar vantagens relativamente ao método de desenvolvimento tradicional, seja uma grande diminuição de vários riscos, entre outras.

- **Falta de software para implementação (*deployment*)** – este risco pode acontecer devido aos membros desenvolverem as alterações necessárias ao projeto no seu ambiente local e criarem uma confiança na operacionalidade do produto desenvolvido que não corresponde a realidade, pois as alterações ainda não foram integradas com as outras alterações dos restantes elementos da equipa num ambiente externo. Este fator propicia à falta de *software* operacional que tenha as diferentes alterações integradas para o *deployment*.

Com a CI existe uma integração normalmente diária e o software que resulta da integração das diferentes alterações é confiável que está operacional, pois este já é testado individualmente e como um todo antes da sua integração.

- **Descoberta de defeitos numa fase avançada** – como referido anteriormente, quando adotado o método tradicional, devido as alterações locais dos vários elementos serem integradas no repositório poucas vezes, algum defeito que exista no ambiente local só vai ser detetado aquando da integração com o restante código ou possivelmente ainda mais tarde. Posteriormente, notam-se as consequências desta decisão, em que se terá de rever / refazer as alterações que podem ter um grande impacto para o *software* desenvolvido e que se tivessem sido detetadas numa fase mais primária seriam apenas pequenas correções.

Estes defeitos podem ocorrer mesmo que cada componente esteja a passar os seus testes unitários, ou seja que o componente tenha o comportamento esperado, devido a um fator muito importante, a integração dos vários componentes no mesmo repositório. Para se verificar que está a funcionar de acordo com os requisitos esperados, é necessário efetuar testes de integração que vão ditar se no final da integração existe ou não software funcional.

- **Falta de visibilidade do projeto** – pelo método tradicional, normalmente, são programadas poucas integrações das alterações para o repositório *master* ao longo de todo o desenvolvimento. Isto implica que os elementos só consigam ver as alterações de todos os elementos quando existe uma integração e que, apenas nesse momento, tenham, também, uma visão geral do projeto e do seu progresso. Como não existem muitas integrações, se um elemento estiver a desenvolver um componente que tenham uma lógica similar ao que está a ser desenvolvido por outro membro da equipa, estes dois elementos só vão ter noção de que podiam partilhar a implementação no final de terem desenvolvido os seus componentes e quando a implementação for feita [16].

2.5 Controlo de Versões

Num repositório existe uma versão controlada do projeto e, através de um diferencial do código da versão controlada com as alterações que vão ser submetidas, é criada uma versão do projeto que as agrega.

O controlo de versões (Figura 6) trata-se de um sistema que grava e gere as alterações que são feitas a um ou vários ficheiros, com a criação de várias versões ao longo do tempo para que, desta forma, seja sempre possível ver ou voltar a uma versão específica de um determinado momento.

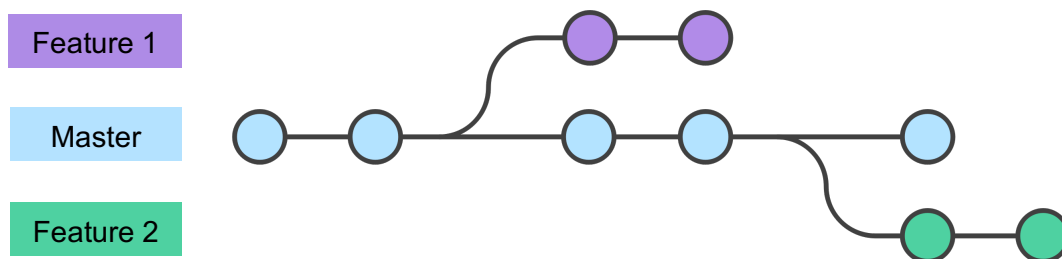


Figura 6 – Controlo de versões

Para que exista um controlo de versões confiável, a equipa de desenvolvimento tem um repositório partilhado entre todos os elementos onde são submetidas as alterações a acrescentar ao projeto. A versão *master* é a versão de controlo do projeto onde estão integradas as alterações aprovadas para submissão no projeto. As alterações que ainda estão a ser desenvolvidas, são feitas num *branch* que contém as alterações em *master* e as novas alterações, quando as novas modificações são aprovadas este *branch* é *merged* com a *master*.

Existem vários sistemas de versões de controlo:

- **Local** – neste sistema existe uma base de dados simples, que guarda o histórico das versões dos ficheiros que estão em controlo de revisão.
- **Centralizado** – existe um servidor que contém o histórico das versões e o nome da pessoa que modificou os ficheiros, em cada versão. Este sistema admite a alteração de ficheiros por várias pessoas, o que é uma grande vantagem relativamente ao sistema local.
- **Distribuído** – com este sistema existe um servidor que contém o histórico das versões dos ficheiros, mas a grande diferença e vantagem entre este sistema e o anterior é que com o sistema distribuído, quando alguém vê o projeto, o sistema faz uma cópia que espelha o total do repositório e o seu histórico. Assim, se por alguma razão o servidor que contém o histórico for corrompido ou deixar de funcionar, pode-se obter uma cópia do histórico dos ficheiros de qualquer pessoa que tenha o histórico atualizado e copiá-la, novamente, para servidor [17].

2.5.1 Git

Durante o estágio foi utilizado o Git como controlador de versões. Este baseia-se no sistema de controlo de versões distribuído, mas a sua maior vantagem é que, quando o Git guarda alterações de uma determinada versão, estas são guardadas em *snapshots* e não numa versão de diferenças.

No sistema distribuído, para cada alteração são guardadas novas versões dos ficheiros que apresentam diferenças (também conhecidas como *delta-based* da versão de controlo), tal como apresentado na Figura 7.

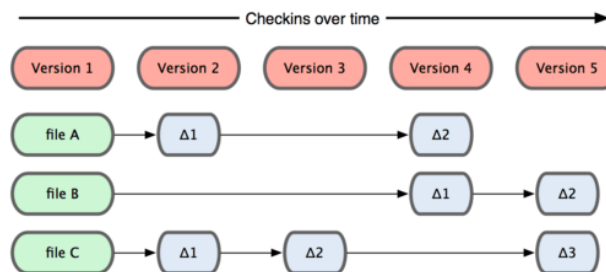


Figura 7 - Sistema de guardar dados em versão de diferenças (delta-based) [18].

O Git tem um sistema de guardar os dados baseado em *snapshots* ou seja, quando um ficheiro é alterado é “tirada uma fotografia” de como se encontram todos os ficheiros nesse momento e guardada uma referência desse *snapshot*. Se, futuramente, houver alguma alteração a um ficheiro, o Git vai analisar as diferenças entre o estado atual e o *snapshot* guardado. Se não houver alteração, o Git apenas referencia o *snapshot* tirado anteriormente aquele ficheiro [17]. A Figura 8 apresenta este sistema.

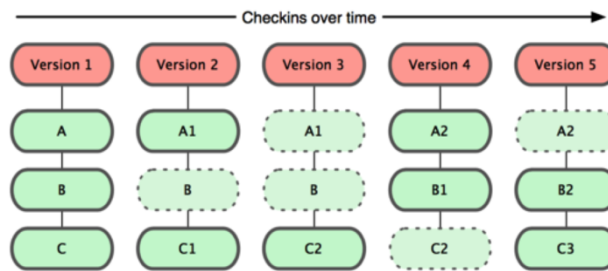


Figura 8 - Sistema de guardar dados em snapshots [18].

2.5.2 GitLab

O GitLab é uma *software* que permite ter todo o ciclo *DevOps* numa aplicação, ou seja, é possível ver, a qualquer momento, em que estado estão as tarefas que estamos a realizar e que os outros membros da equipa estão a fazer, tal como mostrado na Figura 9.

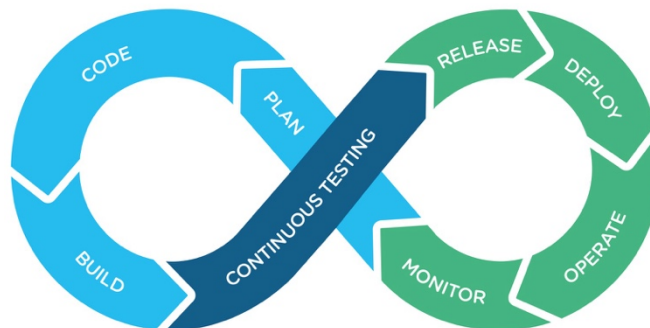


Figura 9 – DevOps ciclo [19].

Durante o projeto em participei no estágio, o GitLab tornou-se uma grande ajuda, pois ao agrupar várias funções que normalmente estão dispersas em várias aplicações, fez com que fosse muito fácil de ver em que estado estava cada *merge request* aberto, através da utilização de *tags*, apresentadas na Figura 10 e ainda fazer *code review* dos *merge requests* dos outros elementos da equipa.

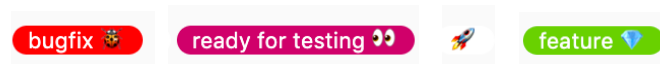


Figura 10 – GitLab tags

A utilização de *tags* identificativas, permitiu, tal como referido anteriormente, ser mais fácil seguir em que estado se encontrava o *merge request*, ficando acordado, no início do projeto, a identificação com a *tag* do tipo de tarefa - por exemplo *bugfix*, *feature* - na abertura do *merge request*, de forma a que os restantes elementos conseguissem identificar de que tipo era a *story* desenvolvida.

Para a continuação do processo, cada *merge request* tem que ser analisado (*code review*). Essa revisão de código pode ser feita diretamente no GitLab, pois este permite ver o código que foi identificado que teve alterações, facilitando o *code review*. Se existir alguma dúvida ou correção que seja necessária fazer para o código, é adicionado um comentário (*discussion*) pelo elemento que está a rever. O *merge request* só pode avançar para a próxima fase se todas as *discussions* estiverem fechadas e o *merge request* tiver sido aprovado por, pelo menos, três elementos.

Quando essas condições estiverem reunidas, o elemento que desenvolveu a tarefa coloca a *tag* “ready for testing”, indicando assim aos elementos de QA (*Quality Assurance*), que a tarefa seja testada mais a fundo e verificar se ainda existe algum erro que não tenha sido detetado nas fases anteriores. Se houver alterações a fazer, o QA notifica o elemento que desenvolveu para corrigir a tarefa. Caso não existam alterações para serem feitas, o QA coloca a *tag* “rocket” que identifica que esse *merge request* está pronto para ser *merged*.

Capítulo III

Desenvolvimento de Front-End

O *front-end* é a parte visível para o utilizador na forma de uma interface que usualmente inclui HTML, CSS, JavaScript, assim como tecnologias que lhes estão associadas [20]. O desenvolvimento de *front-end* (*client-side development*) consiste em produzir produtos digitais que vão de encontro as necessidades de cada utilizador e que o utilizador possa interagir com estes, no âmbito de um dado contexto [21].

O desenvolvimento *front-end* está também ligado à interação, também chamada de *User Interface* (UI), e a usabilidade do utilizador na interface, ou *User Experience* (UX), permitindo uma boa experiência com o produto desenvolvido. A experiência do utilizador é o efeito que a interação, num determinado sistema, dispositivo ou produto lhe provoca. [22]. Aliando a mesma à interface, o utilizador deixa de ver a interface apenas como cores e formas, mas plataforma como uma maneira de lhe apresentar as ferramentas certas para ele atingir os seus objetivos utilizando-a. A interface é a forma de conexão entre o utilizador e a usabilidade, ou seja, a interface deve guiar o utilizador durante a sua experiência a velocidade do seu pensamento [23].

3.1 Ferramentas de apoio ao desenvolvimento web

As ferramentas que vão ser referidas, foram utilizadas para a configuração do ambiente de desenvolvimento que suportou as atividades desenvolvidas durante o período do estágio.

3.1.1 Node.js

Node.js é um ambiente de execução assíncrona de *server-side*, livre e direcionado para múltiplas plataformas de JavaScript. Este executa o V8 JavaScript Engine, fazendo com que seja muito eficiente a execução dos processos. O V8 JavaScript Engine é a parte central que constitui o Google Chrome.

O node tem uma biblioteca que contém módulos de JavaScript, fazendo com que o desenvolvimento de muitas aplicações *web* serem facilitadas e fornece ainda uma interface para API's de baixo nível. Através de uma arquitetura de eventos, e utilizando um ciclo de eventos, este executa os eventos em *single-thread*, uma única chamada ao ciclo de eventos para executar o código, e assim utilizado menos recursos do que se fosse utilizada uma nova *thread*. Simplificando, este método permite executar uma ação sem que exista a necessidade das próximas chamadas ficarem em lista de espera, pois quando o ciclo de eventos recebe múltiplos pedidos, este executa uma das ações e em vez de esperar a resposta deste primeiro pedido, ele vai processando o próximo pedido. No momento em que a resposta do primeiro pedido é devolvida, um evento é disparado para que a função para dar resposta ao pedido correspondente a esta resposta seja executada, fazendo assim que um maior número de processos seja executado com menos recursos, como referido anteriormente.

3.1.2 NPM

O npm (*Node Package Manager*) é um repositório *online* que contém *software* livre, que permite o acesso fácil a inúmeros pacotes e módulos de *software* publicados nesta plataforma. Quando utilizado na linha de comandos, o npm permite operações como a instalação dos pacotes, gestão das suas versões e ainda a gestão das suas dependências.

Uma das vantagens da utilização do npm, para além do acesso a todos estes pacotes de *software*, é a criação de uma pasta denominada “node_modules”, onde se encontram instalados todos os pacotes relacionados com um determinado projeto. É através dessa pasta que é possível a utilização do “require()”, para proceder a utilização de um módulo, tal como se este fosse nativo.

Para além da criação da pasta “node_modules”, ocorre, também, a criação de um ficheiro (package.json) que tem como propósito listar todos os módulos instalados e,

ainda, a versão de cada um. Assim sendo, é possível fazer a instalação e a gestão dos módulos mais facilmente.

3.1.3 ESLint

ESLint é uma ferramenta de análise de código, que permite relatar problemas encontrados, tanto ao nível do código, como, também, de estilos que não cumpram as diretrizes definidas no ESLint.

Esta ferramenta possibilita a criação de regras que os programadores pensam ser mais adequadas ao seu projeto, bem como a personalização de regras já existentes, por omissão.

A importância desta ferramenta num projeto de grande dimensão é a normalização do código a nível de estilos e, também, o aviso de possíveis erros que existam, fazendo, assim, com que o código seja mais consistente em todo o projeto.

3.2 Tecnologias de suporte ao desenvolvimento de front-end

3.2.1 HTML

HTML (HyperText Markup Language) é o esqueleto do desenvolvimento para a *web*. Esta linguagem, tal como o nome indica é constituída por *markups*, que indicam ao *browser* qual a estrutura que o *sítio web* tem, tornando possível a apresentação da página com a estrutura pretendida.

Os elementos Html são compostos por *tags* e por conteúdo, como por exemplo **<p href="https://www.mozilla.org">Hello World</p>** que se traduz num parágrafo com o texto "Hello World". Neste exemplo existem 2 *tags*: uma de abertura (<p>) e outra de fecho (</p>). Estas tag identificam que a estrutura deste conteúdo é um parágrafo. Aliado à *tag* de abertura encontra-se o atributo (href="https://www.mozilla.org"), que corresponde à hiperligação para onde este texto vai redirecionar se for acionado pelo utilizador.

O conjunto de vários elementos dá origem ao documento HTML que vai ser lido e apresentado pelo *browser* [24].

3.2.2 CSS

CSS (Cascading Style Sheets) é uma linguagem utilizada para atribuir estilos e estrutura às páginas web como, por exemplo, o tamanho da fonte, a cor dos elementos, etc [24]. Mas o CSS permite fazer muito mais, como controlar a forma como os elementos da página vão ser apresentados e em que contexto. Por exemplo, organizar um menu verticalmente quando o tamanho do dispositivo é pequeno (*mobile* e *tablet*) e horizontalmente quando o tamanho é maior (*desktop* e *laptop*).

As regras de CSS são propriedades que estão aliadas aos elementos HTML, tornando assim possível afetar o que vai ser visualizado desse elemento. Para aplicar uma dada regra ao elemento é necessária a utilização de um seletor que vai selecionar o elemento onde se pretende ativar uma determinada propriedade e atribuir a esta propriedade o valor que se pretende alterar [25].

Como exemplo base, para ser alterada a cor de um texto para vermelho, no exemplo referido na subsecção anterior, poderia proceder-se como apresentado na Figura 11.


A dark rectangular box containing the CSS selector and rule 'p{ color: red;}' in a light-colored monospace font. The 'p' is pink, '{' is light blue, 'color:' is light blue, 'red;' is light blue, and '}' is light blue.

Figura 11 - CSS

3.2.3 JavaScript

JavaScript (JS) é uma linguagem de programação de *scripting* para a *web* de alto nível, que permite implementar páginas *web* complexas, com conteúdo dinâmico, como, por exemplo, o controlo de elementos multimédia (vídeos e imagens animadas), mapas interativos, entre outros. Através de *scripting*, consegue-se armazenar valores em variáveis, realizar operações complexas, desencadear ações a partir de certos eventos que ocorram na página web, etc [25].

Esta linguagem é bastante flexível. Com ela é possível controlar uma grande quantidade de funcionalidades extras: APIs criando conteúdo dinâmico de HTML, CSS entre outras linguagens; *frameworks* e bibliotecas para assim ter um desenvolvimento mais otimizado de aplicações *web*.

3.2.4 React

React ou ReactJS é uma biblioteca de JavaScript, que permite desenvolver interfaces para o utilizador. Foi pensada para aplicações de grande escala, em que os dados vão mudando ao longo do tempo.

“The only way to write complex software that won’t fall on its face is to hold its global complexity down—to build it out of simple parts connected by well-defined interfaces, so that most problems are local and you can have some hope of upgrading a part without breaking the whole.” por Eric Raymond em “The Art of Unix Programming” [26].

A ideologia do React está refletida no que foi escrito por Eric Raymond, pois o React baseia-se em componentes, que são as “partes simples conectadas” [27].

Um componente é um módulo que faz o *render* de um determinado resultado (*output*). Considerando que uma aplicação contém vários componentes que podem gerir o seu próprio estado, isto permite que os componentes possam ser reutilizáveis [28].

Uma das vantagens da utilização do React é a sua eficiência na atualização e no *render* dos componentes, pois esta tecnologia apenas efetua os processos referidos anteriormente quando um dos dados de um componente muda. Ou seja, apenas atualiza os campos onde os dados mudaram e os componentes em que mudaram. Isto deve-se ao React não atuar diretamente na DOM (*Document Object Model*), mas sim na virtual DOM [29].

Quando uma aplicação contém apenas a DOM, por cada modificação no estado da aplicação, a DOM vai mudar e atualizar aplicação integralmente, ou seja, a aplicação faz um *re-render*. Este fator cria pior desempenho, quanto maior for o tamanho da aplicação.

Por este motivo foi introduzida a virtual DOM (VDOM), apresentada na Figura 12. Tal como o próprio nome indica é uma representação virtual da DOM. Permite, quando

existe alguma alteração ao estado da aplicação, identificar o componente modificado e os seus filhos, que também serão afetados por esta modificação.

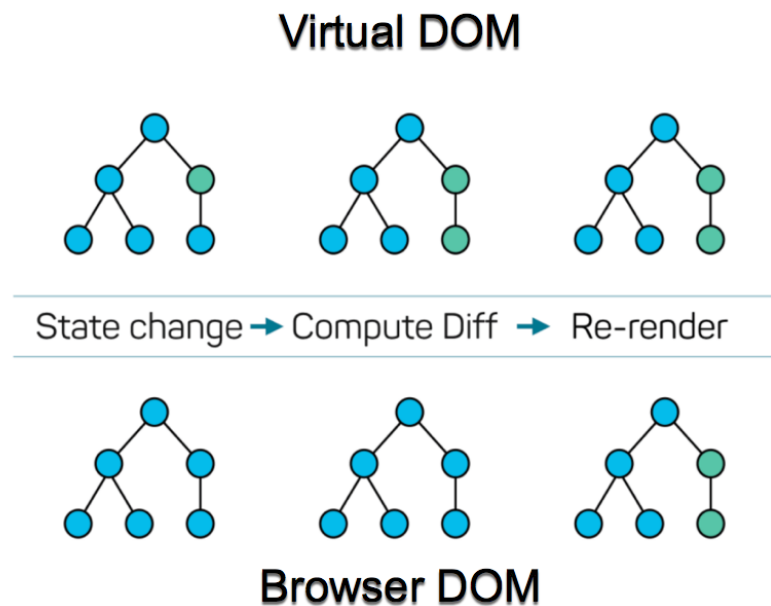


Figura 12 – Render com a Virtual DOM [30].

No final deste processo é atualizado o estado destes componentes na VDOM. De seguida, é comparado o estado atual da VDOM com o estado em que está a DOM. A partir deste processo é possível identificar as diferenças entre estes dois estados e, assim, proceder a atualização da DOM com este novo estado. A vantagem é a atualização dos componentes afetados pela modificação e não a aplicação completa, tal como acontece sem a existência da VDOM.

Importa, ainda, referir o JSX (JavaScript XML). Trata-se de uma extensão para JavaScript para a sintaxe ECMAScript, que permite produzir elementos React. Nela pode-se desenvolver a parte de *markup* e a parte lógica em conjunto, fazendo com que seja mais simples e identificativo desenvolver um determinado elemento / componente.

Esta extensão não é obrigatória quando o React é aplicado, mas é uma mais-valia no processo de desenvolvimento pois apresenta como vantagens a simplicidade que aporta ao desenvolvimento com funções JavaScript e ter uma linguagem familiar para as pessoas que já estão habituadas a fazer desenvolvimento *web* em *front-end*. Todos estes fatores contribuem para que a *markup* se torne melhor estruturada e com maior

significado [31]. A Figura 13 apresenta uma comparação de código desenvolvido com e sem JSX.

These two exemples with JSX	Are complied into this JavaScript
<pre>const element = (<h1 className="greeting"> Hello, world! </h1>);</pre>	<pre>const element = React.createElement('h1', {className: 'greeting'}, 'Hello, world!');</pre>
<pre><div className="sidebar" /></pre>	<pre>React.createElement('div', {className: 'sidebar'}, null)</pre>

Figura 13 – JSX vs JavaScript [32].

3.2.5 Redux

O Redux é um *container*, de estado para aplicações JavaScript, que tem como função gerir o estado e o fluxo dos dados de uma determinada aplicação. O facto de ser bastante simples e com um tamanho reduzido para que o desempenho da aplicação seja otimizado, fez com que rapidamente fosse difundido pela comunidade de programadores para aplicações com a utilização de JavaScript.

Quando se utiliza React sem Redux, os componentes apenas efetuam uma comunicação de estado com os diferentes componentes com que tem uma relação pai-filho (*parent-child*), fazendo com que exista uma grande dificuldade em que todos os componentes comuniquem diretamente entre si, independentemente da sua relação. A Figura 14 ilustra a comunicação entre componentes com e sem a utilização de Redux.

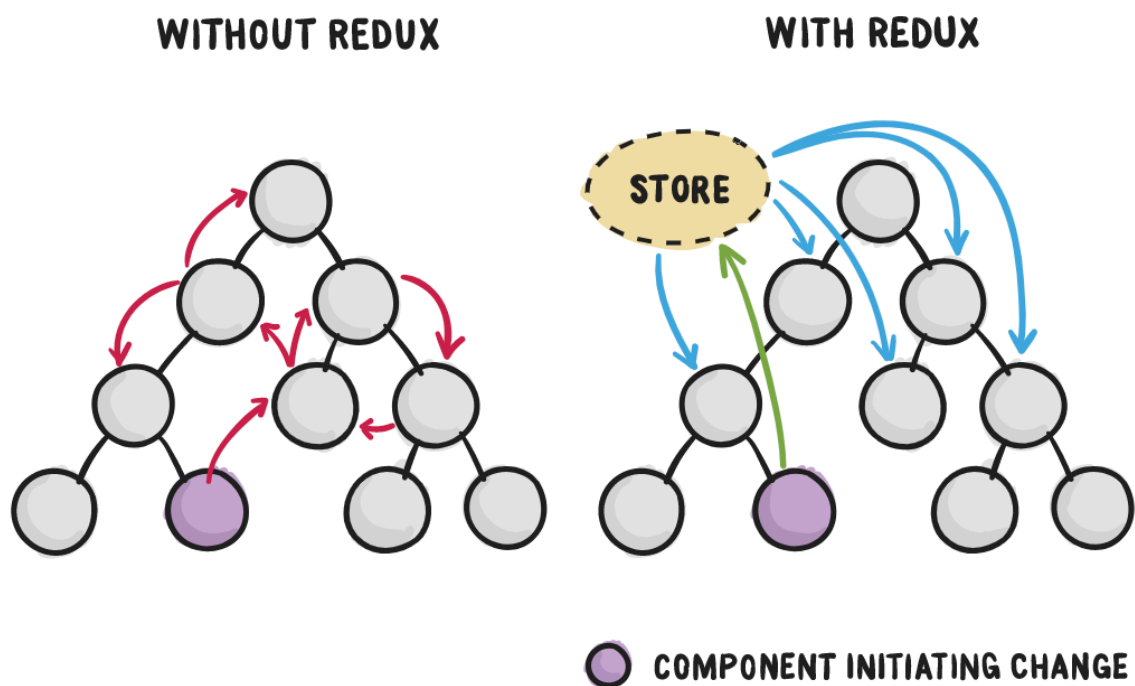


Figura 14 – Utilização de Redux no desenvolvimento de aplicações com JavaScript [33].

Com a utilização do Redux, o estado da aplicação é guardado na *Store* e todos os componentes conseguem comunicar com ela. Caso pretendam efetuar alguma ação no estado da aplicação efetuam um *dispatch* para a *store*, em vez de ser diretamente com outros componentes. Só depois é efetuado um *subscribe* das alterações, por parte dos componentes que necessitam desta modificação de estado, que será enviada pela *store*.

Importa, ainda, referir que o Redux tem uma única fonte de verdade: como o estado da aplicação é controlado por apenas uma entidade - a *store* - esta é a única fonte fidedigna que existe para saber o estado da aplicação. O estado da *store* é de *read-only*, o que faz com que nenhum componente possa alterar o estado da aplicação diretamente. Tal como referido anteriormente é sempre necessária uma ação (*dispatch* ou *subscribe*) para conseguir alterar o estado da aplicação [34].

3.2.6 Lodash

Lodash é uma biblioteca de JavaScript usada para simplificar a forma de lidar e de manipular objetos, *arrays*, *strings*, entre outros. É através da utilização de funções existentes nesta biblioteca, como por exemplo o `_.get(object, path, [defaultValue])`.

Esta função bastante utilizada no projeto desenvolvido no âmbito do estágio, para determinar o caminho de um dado objeto. O Lodash permite, assim, a iteração de *arrays*, objetos e *strings*; a manipulação e teste de valores e ainda a criação de funções compostas [35].

3.3 Testes

No âmbito da Mindera, existem três níveis de testes para aplicar a uma aplicação web (Figura 15): sendo o nível mais baixo o dos testes unitários; o segundo relativo a testes de integração; e o terceiro nível aborda testes de E2E (*end-to-end*).

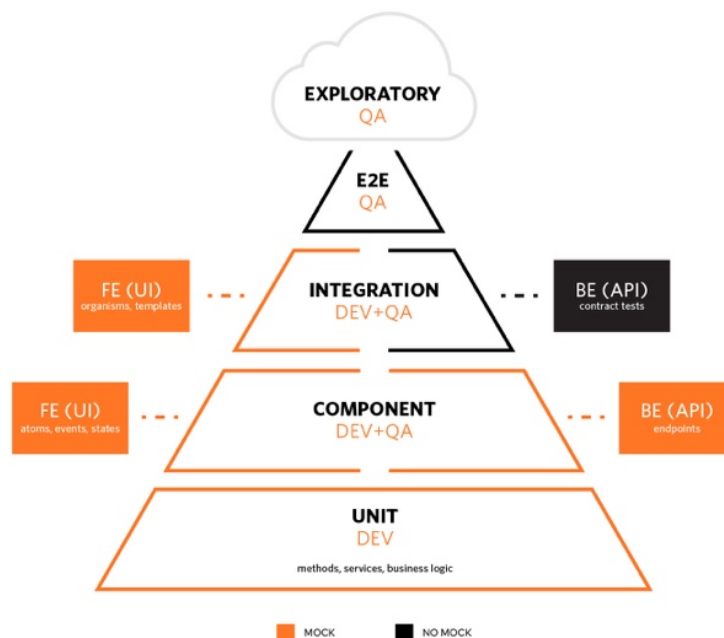


Figura 15 – Pirâmide de testes [36].

O primeiro nível de testes, **testes unitários** (*Unit testing*) respeita ao procedimento de testes de qualidade para verificar se os componentes, desenvolvidos para uma determinada tarefa, se comportam da forma esperada, cumprindo o objetivo para o qual foram projetados. Normalmente os testes unitários são desenvolvidos manualmente pelo programador. Ao testar individualmente cada componente pode-se garantir uma melhor detecção de eventuais erros de desenvolvimento, facilitando a sua posterior integração na aplicação [37].

Já os **testes de integração** (*Integration testing*), têm como propósito testar como é que os vários componentes unitários / isolados se comportam quando integrados com os componentes que vão estar agrupados. Neste tipo de testes, o mais importante é testar os elos de ligação / integração dos vários elementos e não focar no seu comportamento individual.

O último nível é dedicado aos testes de sistema que validam se os componentes integrados também cumprem os requisitos de comportamento quando são integrados com as interfaces / sistemas externos. Estes testes têm como ambiente o de produção. Através da utilização de dados e testes de simulação de comportamentos, é possível verificar se todos os requisitos são cumpridos para a entrega de um produto final.

3.3.1 Jest

Jest é uma *framework* de testes unitários de JavaScript. É bastante abrangente, funcionando em projetos que utilizem as tecnologias Babel, TypeScript, node, React entre outras [37].

Esta *framework* procura os ficheiros que contém testes unitários, executa-os e ainda escreve os resultados dos testes na consola, ou num ficheiro de *log*. Funciona, assim, como um *test runner*. Para além disso também é possível verificar esses resultados, ou seja, uma *assertion library*. Por último, permite a execução de *mocks*, ou seja, se um componente se encontra interligado com muitos dependentes, quando executando os testes unitários, a melhor forma de os fazer é criando um *mock* que irá substituir essas dependências. É possível, ainda, no teste desses componentes, requisitar como esperado que as dependências que este componente vai interligar-se sejam chamadas pelo teste que será executado.

Uma das vantagens da utilização do Jest para testes respeita ao recurso a *snapshots* para a verificação da saída do teste. Através de uma comparação de um *snapshot* com o guardado no teste anterior, é possível saber se o teste falha ou passa. Para o programador, os *snapshots* permitem detetar alterações esperadas ou não, sendo uma maneira simples de garantir que não se introduz erros inesperados.

3.3.2 Enzyme

O Enzyme é uma ferramenta de teste em JavaScript, possibilita a declaração, cruzamento e manipulação do *output* de componentes React [38].

Esta ferramenta dá a possibilidade ao programador de utilizar métodos de *render*, como por exemplo:

- **Mount**, que faz o *render* do componente que está a ser testado com as suas *props* (*default props*) e ainda os seus filhos com as suas *props*. Uma *prop* é o diminutivo utilizado para a palavra *property*. As *props* são parâmetros que permitem passar dados para um componente ou mudar o seu estado [28]. Este tipo de *render* é o indicado para testar componentes que têm de interagir, necessariamente com a DOM, ou com uma API.
- **Shallow**. Apenas permite o *render* do componente a ser testado, sendo bastante usual a utilização deste método para testes unitários, pois assim pode-se garantir que o comportamento testado é apenas e só proveniente do componente em questão.
- **Render**. Este método faz o *render* HTML estático e inclui os filhos desse componente. A nível de desempenho tem um custo menor, mas isso implica que tenha menos funcionalidades disponíveis para utilização dos testes.

Permite, também, a simulação de eventos, como, por exemplo, um *click* do rato, ou de uma tecla, entre outros. Através destes métodos é possível testar qual o comportamento de um componente, quando submetido a interações do utilizador.

Capítulo IV

Projeto

O projeto no qual estive envolvida durante o estágio tinha como objetivo o desenvolvimento de uma aplicação *web*, direcionada para o *e-commerce*, que renovasse a plataforma que o cliente possuía. Este *sítio web* foi construído com recurso às tecnologias já abordadas neste documento, tais como React, Redux, entre outras.

Para além de uma profunda renovação do *layout* do *sítio web*, foram, também, desenvolvidos novos tipos de conteúdos que o cliente pretendia divulgar, como, por exemplo, componentes de vídeo e de divulgação atualizada sobre a sua marca.

4.1 Enquadramento

No projeto onde estive integrada tive a oportunidade de executar variadas tarefas. Para melhor entendimento e explicação dessas tarefas, elas devem estar divididas pelo seu tipo: tarefas de integração, *bugs*, *tasks* e acessibilidade. Os termos referidos anteriormente são referentes a termos utilizados pela empresa para caracterizar os tipos de tarefas apresentadas no projeto. Segue-se uma caracterização sumária de cada tipologia de tarefa:

- **Tarefas de integração:** tarefas de acompanhamento do desenvolvimento, não tendo sido eu que diretamente as desenvolvi.

- **Bugs:** erros ou falhas encontradas no sítio *web*, que fazem com que o resultado esperado / planeado não seja o cenário que está a acontecer. Estes podem ser ao nível de design, de comportamento, acessibilidade, entre outros.
- **Tasks:** tarefas que são para desenvolver. Normalmente ainda se encontram no estado inicial, sendo preciso fazer toda a lógica pertencente aquele componente.
- **Acessibilidade:** problemas ou erros que não estão de acordo com as normas de acessibilidade para um sítio *web*.

Em cada uma destas tipologias serão abordadas apenas algumas das tarefas desenvolvidas. Selecionei, para o efeito, as que considere mais desafiantes, devido à sua complexidade, ou pela forma como tiveram de ser resolvidas.

A Tabela 1 apresenta a calendarização das tipologias de tarefas no decorrer do estágio e segundo a cronologia do projeto.

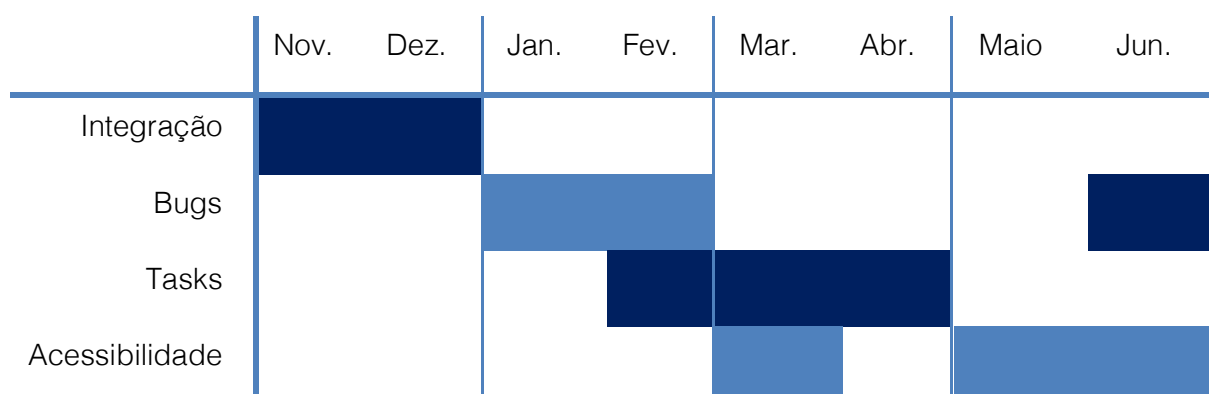


Tabela 1 - Cronograma

4.2 Tarefas de integração

Para realizar estas tarefas foi utilizada uma técnica de desenvolvimento de software Agile conhecida como *pairing* / *pair programming*. Esta fundamenta que estejam envolvidos dois programadores que vão trabalhar na mesma tarefa, partilhando o mesmo computador e colaborando para o desenvolvimento do mesmo código.

Um dos membros é o guia da tarefa (*driver*), que vai estar a escrever o código e orientar o desenvolvimento do trabalho e o outro é o seguidor do processo (*navigator*) que vai observar o trabalho que está a ser desenvolvido, verificar se existem erros ou alguma falha no trabalho e, ainda, pensar se existe uma forma melhor de resolver esta tarefa [39].

Seguindo a base do *pair programming*, numa fase inicial do estágio, fiz um total de seis tarefas recorrendo a esta técnica, em que acompanhava um dos elementos da equipa na tarefa que estavam a realizar e no decorrer do desenvolvimento, esse elemento explicava-me qual a sua tarefa, como estava a pensar abordar este problema, as tecnologias que estava a utilizar, a estrutura e a organização dos componentes que já estavam desenvolvidos e o funcionamento dos softwares de desenvolvimento e complementares ao trabalho, como, por exemplo, o Jira, o GitLab , entre outros.

1.ª Tarefa (Confirmação de compra):

Na base desta tarefa encontra-se a criação e o desenvolvimento dos componentes que constituem a página de confirmação de um pedido de uma encomenda.

Esta página é composta por um total de sete grandes componentes, o título (h1) da página, informações da encomenda (OrderInfo), a morada de envio (ShippingAddress), a morada para onde é enviado o recibo (BillingAddress), o método de pagamento (PaymentMethod), os itens que foram selecionados (OrderDetails) e um sumário dos custos da encomenda (OrderSummary).

Tendo como base a aplicação *web* da Amazon e a título de exemplo e de forma a ser mais ilustrativo para se poder entender melhor a tarefa realizada, a Figura 16, é um exemplo representativo da página que foi desenvolvida, bem como dos componentes criados nesta tarefa. Na sua descrição mais detalhada serão mencionados os componentes ilustrados na Figura 16.



Order Confirmation

Order # Number

Order place date Date

Email and message to user

Billing Address

Street, Number
State, Country

Shipping Address

Street, Number
State, Country

Payment Method

Method

Order Details



Sony S55A98060 55-Inch 1080p 120Hz LED Slim 3D
HDTV (Silver)
Electronics
In Stock

\$1,010.13

Item Subtotal: \$1,010.13
Shipping & Handling: \$0.00

Total Before Tax: \$1,010.13
Estimated Tax: \$0.00

Order Total: \$1,010.13

Figura 16 – Order Confirmation, Amazon

O componente que faz o *render* a página (OrderConfirmationPage) recebe, como *prop*, o id da encomenda e, através desse id, é possível fazer um pedido à API para que esta envie os dados correspondentes. Após a obtenção destes dados, a página transmite estes dados para os componentes filhos, que vão, apenas, utilizar os que necessitam de apresentar.

O primeiro componente da página permite o *render* do título, que é um h1 a nível de *tag* HTML (Order Confirmation). Esse título é um dado obtido através de uma tradução, na qual existe uma chave de tradução no componente. Terá uma

correspondência num texto definido num ficheiro de traduções. Este sistema é possível através da utilização do Redux *polyglot*. Uma das grandes vantagens deste sistema é a fácil possibilidade de modificação da tradução correspondente a cada chave e, ainda, o facto de ser possível definir textos alternativos a uma chave de tradução, conforme o idioma em que a aplicação se encontra.

Nos restantes componentes foi adotado o mesmo sistema para a apresentação das *labels* correspondentes aos dados a apresentar. Por exemplo, no componente *OrderInfo* é apresentada uma *label* (Order #) que se segue pelo número de encomenda (ex. 12345), ou, ainda, no componente *OrderSummary*, as *labels* (Item Subtotal, Shipping & Handling, ...) correspondentes a cada um dos montantes.

Tal como referido anteriormente, a representação dos dados de cada componente é possível através do contacto com uma API e através de *props* que são transmitidas ao componente em questão. No caso do componente *OrderInfo*, como referido anteriormente, são apresentados: o número da encomenda, a data em que foi efetuada e, ainda, o email utilizado pelo *user* para efetuar a encomenda, com uma mensagem de agradecimento definida pela marca.

É necessária uma grande atenção na utilização dos dados que chegam da API, pois estes podem ainda não estar preparados para a sua apresentação final no componente. Nos casos em que não estejam conforme o pretendido, é necessária a sua normalização. Esta normalização, no caso da data passa pela utilização da função *new Date()*, que recebe, opcionalmente, um ou mais argumentos e retorna uma data formatada de acordo com o que é pretendido apresentar ao utilizador. Neste caso o mês, o dia e o ano.

Para os componentes *ShippingAddress* e *BillingAddress* também foi feita uma normalização dos dados recebidos pela API, em que se verifica se todos os campos foram preenchidos, ou apenas os obrigatórios é que são recebidos. Após esta verificação, os dados são atribuídos aos campos e serão apresentados.

Quanto ao componente *PaymentMethod*, a API guarda qual o tipo de método de pagamento escolhido pelo utilizador e o número do cartão de crédito utilizado, quando são preenchidos os dados para efetuar a compra. Para a página de confirmação da encomenda, é utilizada a informação do tipo de cartão fornecida pela API para se apresentar o logótipo correspondente a esse tipo de cartão, através de uma comparação entre os dados recebidos e a lista de chaves de tipos de cartão. A cada valor desta lista de chaves é atribuída a imagem do logótipo correspondente, para poder ser apresentada no componente de *PaymentMethod*.

Relativamente ao número do cartão apenas são apresentados os últimos quatro números. Para isso foi utilizado o método *.slice()*, com o parâmetro -4, o que permite

retornar uma cópia do *array* em que é efetuado este método e guardadas, apenas, as últimas quatro posições deste *array*.

Relativamente ao componente *OrderDetails*, como apresentado na Figura 16, é composto por uma lista de o(s) artigo(s) escolhidos, em que cada um destes é um item na lista, bem como os detalhes do(s) mesmo(s). Nesta lista pode-se visualizar cada um dos itens e as informações que lhe estão atribuídas, devido a utilização do método *listaProdutos.map()*. Este método cria um novo *array*, com o resultado da chamada de cada um dos elementos para uma determinada função. Neste caso, a função chamada contém os parâmetros *item* e o *índice* de cada um destes no *array*.

Após o *map* da lista, são obtidos os itens, de qual o *render* será feito através de um componente (*OrderItem*) que é composto por uma imagem, agregado com um hiperligação para a página desse produto, o nome do produto que também é agregado a um hiperligação com o mesmo comportamento que a imagem. Para além disso, este componente recebe um dado do qual é feito o *render* condicionalmente: mostrar o *id* do produto. É feito o *render* deste texto apenas se a *prop* “*showId*” for recebida pelo componente com o valor de *true* (verdadeiro), caso contrário o texto não será mostrado.

Esta forma de *render* de informação é muito útil pois, assim, o mesmo componente pode ser utilizado em várias situações e locais da aplicação e através de uma *prop* mudar o seu comportamento, para respeitar o contexto onde está inserido. Este *render* condicional também pode ser aplicado quando se está perante diferentes *breakpoints*, através de uma função que vai verificar qual o *breakpoint* em que o utilizador se encontra e caso esse seja o da condição presente, é feito (ou não) o *render* de um determinado código. Em cada item é, ainda, apresentada a informação relativa a quantidade escolhida e, caso exista, o tamanho escolhido para aquele artigo.

Quanto ao último componente – o *OrderSummary* – este apresenta as *labels* de cada campo apresentado através de traduções, como referido anteriormente, e os dados respetivos a esse campo, que são enviados pela API para o componente, sendo necessária a verificação se existem dados para os campos que não são obrigatórios como, por exemplo, o valor de desconto que o utilizador obteve numa encomenda, quando introduz um código promocional.

Ao nível de testes, para o componente *OrderConfirmation*, foram feitos três grupos de testes: no primeiro grupo o objetivo foi testar se o componente estava a fazer o *render* corretamente dos comportamentos mais básicos como, por exemplo, o *render* de cada componente filho, através do método *shallow* e da verificação se cada data-teste destes componentes existe na árvore retornada pelo método *shallow*, tal como esperado.

No segundo grupo de testes é verificado se o componente consegue retornar os dados correspondentes a um *orderId* que lhe for fornecido. Além dessa verificação, é feita

uma verificação de que se não for fornecido nenhum *orderId* ao componente, este não é renderizado, sendo este comportamento um *fallback*.

No grupo seguinte, foram realizados testes verificando que o componente vai fazer o *re-render*, caso lhe seja fornecido um novo *orderId*.

Esta tarefa permitiu entender melhor as interações necessárias com a API para a obtenção dos dados e a forma de como tratarmos estes dados para ser possível apresentá-los ao utilizador. Para além disso foi uma boa introdução para a tecnologia utilizada para a tradução de *labels*

2.ª Tarefa (Carrossel de produtos recomendados):

Para esta tarefa foi desenvolvido um componente para integrar na página de produto. O objetivo é que o utilizador possa ter apresentados produtos recomendados, tendo como ponto de referência a página de produto que está a ser visualizada.

Considerando, novamente, a aplicação *web* da Amazon, a Figura 17 ilustra um componente similar ao desenvolvido.

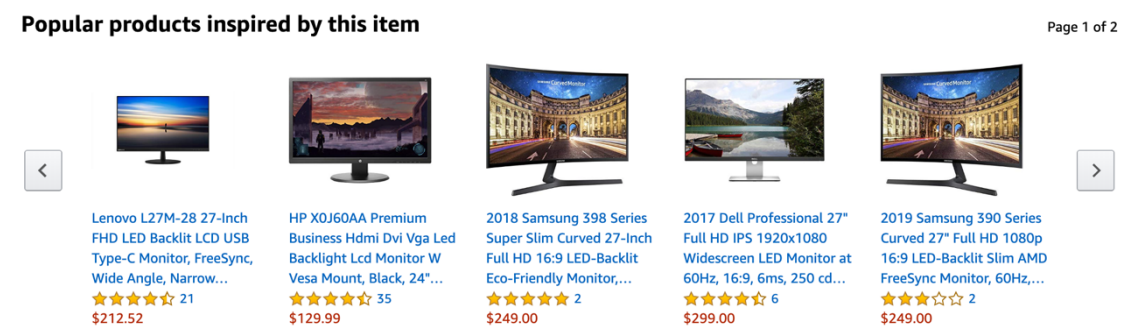


Figura 17 – Popular products inspired by this item, Amazon

O componente desenvolvido nesta tarefa é constituído por um carrossel de produtos, em que é possível encontrar produtos relacionados com o que se está a visualizar. Estes produtos são definidos pelo gestor de conteúdos da aplicação *web*, na plataforma da API. Quando a página é carregada, a API resolve, com os detalhes desse produto, se existirem alguns produtos definidos no *array* de produtos relacionados. O componente tem acesso ao *array* de produtos e é feito o *render* de forma condicional, comparando se o tamanho do *array* recebido é maior que 0 ou não. Caso o tamanho

seja maior que o valor 0, o componente é apresentado; caso contrário não é feito o *render* do componente.

No topo do componente é apresentado o título. No caso do exemplo (Figura 17), “Popular products inspired by this item”, que tem como *tag HTML* um `<h2>` visto ser o mais adequado a posição do componente na página e da semântica global da mesma. O texto do título é apresentado através de chave de tradução, à qual corresponde uma tradução possível de alterar. A tradução utiliza a técnica referida na 1.^a tarefa de integração.

Quanto ao produtos apresentados no carrossel, recorreu-se ao método de JavaScript `.map()`, para ser possível fazer o *render* de cada um dos produtos e os dados correspondentes, tal como já apresentado. Os dados exibidos são uma imagem do produto (que, quando clicada, permite o redirecionamento para a página deste produto), o nome do produto e o seu preço.

Para além dos dados referidos, existe um componente que não se pode visualizar na Figura 17: trata-se de uma barra que apenas aparece quando é feita a interação de *hover* sobre a imagem do produto.

Esta barra representa uma forma muito fácil do utilizador adicionar o produto à sua lista de favoritos ou, ainda, de escolher o tamanho do artigo, para que este possa ser adicionado ao seu carrinho. Este componente guarda, em estado, a escolha do utilizador. Os componentes da aplicação que tiverem como *prop* algum destes parâmetros irão atualizar o seu estado. Caso algum tamanho não esteja disponível para compra, este será apresentado com uma opacidade e *disabled*, de forma a que o utilizador não possa selecionar essa opção.

No caso de escolher adicionar o produto para lista de favoritos, será possível ver, no ícone dos favoritos, na barra do topo da aplicação, que o número de artigos que lá se encontra foi atualizado. O mesmo acontece caso se selecione um tamanho, para o ícone do carrinho.

Relativamente ao carrossel, é possível a interação do utilizador através do clique com o rato nas setas apresentadas nas laterais, para a visualização de mais artigos da lista de produtos. Também é possível fazer esta interação através de um *swipe* se o dispositivo onde a aplicação está a ser visualizada o permitir ou, ainda, através das setas presentes do teclado.

É permitido, também, visualizar um indicador do total de produtos que é possível ver e qual a posição do item ativo, ou seja, o item que se encontra no centro do carrossel. A quantidade destes indicadores é obtida através do tamanho do *array* de produtos recomendados e o item que se encontra no centro do carrossel é obtido através do *index* deste produto no *array* da lista de produtos.

4.3 Bugs

Os *bugs* são erros / falhas que levam o sistema a não produzir o resultado esperado. Estas falhas podem acontecer ao nível do comportamento de um determinado componente, ao nível de design, etc. Para estes *bugs* serem descobertos é necessário que seja feito um *debugging* ao sítio *web*. Após este processo, os *bugs* são reportados e se se continuarem a verificar, são adicionados ao *product backlog*, de forma a serem integrados numa *sprint* do projeto e poderem ser resolvidos.

A maior parte destas tarefas são pequenos ajustes ao comportamento de um componente, pois quando algum programador pretende submeter código, este tem que efetuar testes unitários que promovem a correção de algum comportamento inesperado.

Após esta fase, o programador pode abrir um *merge request* para integrar o seu código com o *master*, mas antes deste ser submetido, é revisto por diferentes *developers*, voltando a ser resolvido algum eventual problema, antes de passar para a seguinte fase. Já aprovado pelos programadores, este MR é submetido a testes por um QA, e só após a sua provação é que estão garantidos todos os fatores para efetuar o *merge* do código.

Devido a este fluxo de múltiplas verificações do código, realizadas por diferentes pessoas, nota-se que, de forma geral, os *bugs* que se encontram para corrigir no projeto são, apenas, pequenos pormenores de comportamento e nunca grandes modificações ao projeto.

Durante o estágio realizei a correção de um total de vinte e um *bugs*, mas serão apenas referidas as duas tarefas mais representativas.

1.^a Tarefa (Detalhes do produto ao adicionar a lista de favoritos):

A primeira tarefa de *bugs* foi relativa ao comportamento de um item, quando este se encontrava adicionado ao carrinho e era “movido” para a lista de itens favoritos.

Se o utilizador adicionasse um artigo ao carrinho para efetuar uma compra e, já com o artigo no carrinho, seleccionasse um tamanho para o item (por exemplo, M), quando de seguida clicasse no botão “Mover para a lista de favoritos”, o item era retirado do carrinho e movido para a lista de favoritos, mas o tamanho anteriormente selecionado

deixava de estar selecionado. Isto faria com que o utilizador tivesse que voltar a escolher o tamanho que já tinha sido selecionado, não proporcionando a melhor experiência e não sendo o comportamento pretendido para o componente.

Após se verificar que este *bug* ainda se encontrava na aplicação, a abordagem passou pela modificação do comportamento do clique no botão de “Mover para a lista de favoritos”. Para isso, alterou-se o componente que faz a apresentação do produto, que está inserido na página do carrinho. Em vez de o clique do botão automaticamente passar esse produto para a lista de favoritos, primeiro verifica-se se o produto tem algum tamanho associado nas suas *props*. Se for esse o caso, o tamanho vai ser guardado em estado. Apenas depois de ser efetuado este passo é que o componente irá registar o produto na lista de favoritos e retirar o mesmo do carrinho.

A correção deste *bug* foi possível quando se passou a considerar a *prop* tamanho selecionado como um parâmetro a incluir na passagem do carrinho para a lista de favoritos e deixou de se ir buscar o produto com os seus valores por omissão a partir do id do produto.

2.ª Tarefa (Ordem da lista do carrinho):

A segunda tarefa relacionada com *bugs* está relacionada, também, com a área do carrinho. Quando o utilizador acrescentava vários produtos ao seu carrinho e se esses produtos continham um tamanho selecionado, caso o utilizador fizesse uma alteração no tamanho escolhido, a ordem em que os produtos eram apresentados alterava-se.

Após investigação do comportamento que o componente tinha, foi descoberto que o *array* de artigos alterava a sua ordem caso fosse modificado algum parâmetro dos produtos, colocando, na primeira posição, o artigo modificado. Assim, a solução adotada para este problema passou por uma ordenação alfabética dos produtos contidos no carrinho de compras, antes ser feito o *render* dos mesmos.

Para pôr em prática a solução encontrada, foi necessária a utilização do método `sortBy()` do *Lodash*, que permite a organização dos elementos de um *array* por ordem ascendente, assim como do método `.map()`, retornando o um novo *array* com os elementos que foram iterados. Desta forma, quando o utilizador modificar algum parâmetro dos elementos presentes no seu carrinho, como estão organizados por ordem alfabética, não irão existir alterações na ordem dos produtos da lista.

4.4 Tasks

A *task* é uma tarefa que é necessária desenvolver para atingir um determinado objetivo. Nesse sentido, é preciso desenvolver *software* que cumpra os requisitos descritos na *user story* respetiva a essa tarefa. A *user story* serve de guia ao desenvolvimento desta tarefa, nomeadamente, é onde se encontra descrito o que é necessário desenvolver, para quê e quais os critérios obrigatórios para que esta tarefa seja concluída.

No final do seu desenvolvimento, esta *task* obtém o estado de *done* no *sprint backlog*. Cada *task* pode ter *subtasks* associadas, fazendo com que, sem estas estarem concluídas, a *task* também não pode ter o estado de concluída.

No âmbito das *tasks* serão referidas três tarefas que considere mais desafiantes, de um total de nove tarefas desenvolvidas no âmbito do meu estágio.

1.^a Tarefa (Componente editorial de quatro imagens):

A primeira tarefa foi relativa a um componente que apresenta quatro imagens. A sua organização vai mudar consoante o *breakpoint* em que estão a ser apresentadas. Neste componente, cada imagem pode ter associada, opcionalmente, uma pequena frase, que representa uma hiperligação.

A organização destas imagens é apresentada para os *breakpoints* maiores (1440px e 1024px) com 2 colunas. As colunas são alinhadas ao centro do componente e cada uma delas encontra-se à com a mesma distância do centro, tal como é apresentado na Figura 18.

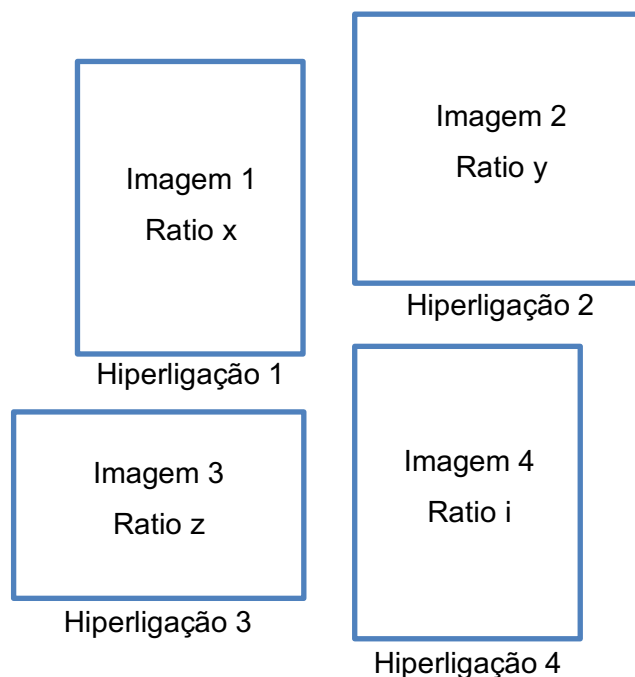


Figura 18 - Componente em grandes breakpoints

Um dos requisitos deste componente era que todas as imagens tinham de ser apresentadas com o tamanho definido no design. Para garantir que este requisito era cumprido, o componente foi desenvolvido de forma a que o tamanho destas imagens fosse dado pelo rácio definido para cada uma delas e, através de regras de CSS, transformar as imagens de forma a terem o tamanho que estava definido.

Para este componente era, ainda, obrigatório, o desenvolvimento de uma propriedade relacionada com a visualização do componente: a *prop mirrored*, que permite proceder a inversão das colunas (i.e. quando ativada, a coluna que se apresenta originalmente à direita passa para a esquerda, acontecendo o mesmo a coluna original da esquerda). Isto permite que facilmente se altere o aspeto do componente, sem existirem conflitos ao nível de espaçamentos e de design.

Para que seja possível o componente mostrar as imagens, é necessária uma comunicação com a API da aplicação para obter os dados correspondentes as imagens e aos textos que as podem acompanhar, bem como a hiperligação para onde o utilizador será redirecionado.

No caso dos *breakpoints* mais pequenos (768px e 320px), a disposição das imagens na aplicação passa de 2 colunas para, apenas, uma, onde as imagens são organizadas por ordem crescente, como se pode ver na Figura 19. Caso a *prop mirrored*

seja ativada, nestes *breakpoints* o comportamento que o componente irá realizar será uma inversão da ordem das imagens de crescente para decrescente, como apresentado na Figura 19.

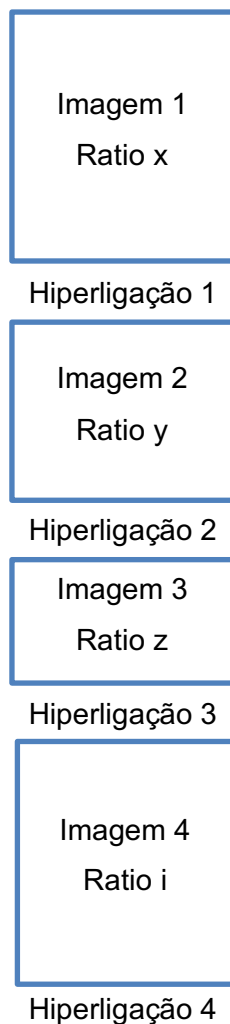


Figura 19 – Componente em pequenos breakpoints

Para ser possível apresentar duas colunas ou 1 coluna, consoante o *breakpoint* em questão, foi desenvolvido um *render* condicional, no qual se verifica qual o *breakpoint* em que o utilizador está a visualizar o componente e, caso este seja um dos pequenos, o *array* de imagens a apresentar vai corresponder ao total de imagens, ou seja quatro. Caso o *breakpoint* de visualização seja dos maiores, o *array* é filtrado e, na coluna da esquerda, apenas são apresentadas as imagens com o index ímpar, enquanto que na coluna da direita, vão surgir as imagens com index par.

Esta lógica de filtragem no *array* de imagens, e a divisão de imagens por colunas, permitiu facilitar o comportamento do componente, assim como não terem de ser feitas várias condições desnecessariamente. Com efeito, desta forma é sempre feito o *render* da coluna da esquerda, pois este pode conter, apenas as imagens ímpares ou todas as imagens, e o *render* é condicional apenas para a coluna da direita, apresentando as imagens, caso o utilizador se encontre nos maiores *breakpoints*.

Relativamente aos testes unitários, o primeiro teste consistiu em verificar, através do *render mount*, se o componente apresentava todos os dados como suposto (quatro imagens, os textos e as hiperligações). O segundo teste pretendeu testar se o componente apenas fazia *render* de uma coluna, através da verificação do seu data-teste, quando lhe é passado o parâmetro de um *breakpoint* pequeno.

Para garantir que o funcionamento do componente era o correto, também foi realizado um teste para verificar, caso o *breakpoint* seja um dos maiores, o componente apenas apresenta uma coluna. Para além destes testes, foram ainda feitos mais dois, para se verificar que, caso seja passado um texto, ele é apresentado, através da validação da existência do data-teste do texto e, caso não seja inserido nenhum texto, o componente não o apresenta.

2.^a Tarefa (Barra de controlo de compra):

Esta tarefa esteve relacionada com a página dos detalhes de um produto. Quando o utilizador acede a esta página, pode visualizar, junto da imagem do produto, os seus detalhes. Para além disto, ainda é possível, no componente de controlo da compra, adicionar o artigo ao carrinho ou à lista dos favoritos, tal como ilustra, como exemplo a Figura 20, a página dos detalhes de um produto da Amazon.

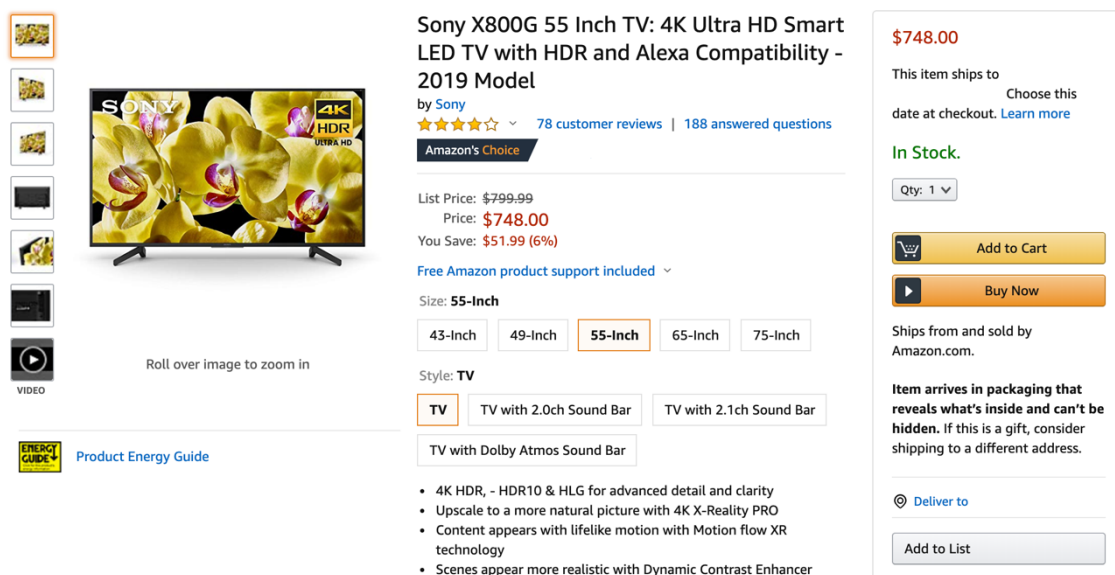


Figura 20 – Página dos detalhes de um produto, Amazon

No caso da aplicação *web* desenvolvida no âmbito deste projeto, e para facilitar a interação do utilizador, foi criado um componente que consiste numa barra com controlos para adicionar ou remover um produto da lista de favoritos. Possui, ainda, a possibilidade do utilizador seleccionar o tamanho pretendido para aquele artigo, e ao carregar no botão “Adicionar ao carrinho”, o artigo é adicionado ao carrinho.

Esta barra é mostrada apenas quando o utilizador deixa de conseguir visualizar o componente de controlo de compra, que contém os controlos na parte inicial da página de produto em que o utilizador se encontra.

Quanto às interações do utilizador com a barra e de forma a que este possa perceber o sucesso de uma ação, se a ação efetuada consiste na adição deste artigo ao carrinho, o número do ícone do carrinho é atualizado na barra do topo da aplicação e o texto do botão modifica de “Adicionar ao carrinho” para “Artigo adicionado ao carrinho”.

Quanto à lista de favoritos, após a interação do utilizador, o ícone da lista de favoritos que se encontra no componente deixa de ser um ícone apenas com contorno e passa a ser um ícone com uma cor de preenchimento. O mesmo acontece ao ícone de lista de favoritos no topo da aplicação. Para além desta modificação, acrescenta mais um ao número total de produtos existentes na lista de favoritos.

Para que este componente apenas fosse visível quando o utilizador deixa de ter apresentados os controlos referidos anteriormente, foi necessária a utilização de um

Intersection Observer. No caso desta aplicação foi utilizado o *React Intersection Observer*, que consiste num componente *React*, que contém a API do *Intersection Observer* encapsulada.

A razão para a utilização desta tecnologia consiste no facto desta API permitir a execução de um *callback* caso um determinado elemento – o *target* – interseção um certo ponto do *viewport*, ou outro elemento que o programador definira. Neste caso, foi definido que a barra desenvolvida para esta tarefa seria mostrada cada vez que o utilizador fizesse *scroll* suficiente, para que o controlo de compra deixe de ser visível no *viewport*.

Quando este evento acontece, o *Intersection Observer* tem, como *callback*, a mudança do estado da *prop* que determina se o componente desenvolvido está visível ou não. Através da mudança dessa *prop* para o estado de verdadeiro, o componente passa a estar presente no fundo do *viewport* do utilizador. Para ser possível que o utilizador realize o *scroll* pelo resto da página, sem que o componente deixe de aparecer no fundo do *viewport*, é utilizada a propriedade de CSS “position”, com o valor de “sticky”. Para além disso, foi definido que o “bottom” corresponde ao valor 0.

3.ª Tarefa (Fluxo de compra para utilizadores não registados):

Outra das tarefas que importa abordar está relacionada com a funcionalidade de um utilizador da plataforma ter a possibilidade de fazer uma compra mesmo que não se encontre registado (*Guest*).

Esta tarefa para além do desenvolvimento do componente que permite o *render* de um formulário para o utilizador preencher, terá de permitir, ainda, a alteração dos componentes de *Login* e do componente da página de encomenda dos produtos.

Relativamente a página de *Login*, o componente desenvolvido “Guest” é visível cada vez que o utilizador adiciona um artigo ao seu carrinho. Quando navega para a página do carrinho, após clicar no botão “Efetuar encomenda”, o utilizador é redirecionado para a página de *Login*, onde pode visualizar o componente. Só através do fluxo descrito anteriormente é que o componente é visível e não cada vez que o utilizador tem um produto no seu carrinho e navega para a página de *Login*.

Para garantir que o utilizador só consegue visualizar este componente com o fluxo descrito e não com o comportamento que foi descrito anteriormente como não desejado, a forma encontrada, a nível de desenvolvimento, foi através da verificação e da alteração da rota da página quando o utilizador navega de uma forma ou de outra. Quando o utilizador chega à página de *Login* através da página de encomenda de

produtos, a rota onde o utilizador vai visualizar o componente terá o parâmetro “encomenda”.

Caso o utilizador navegue até a página de *Login*, mesmo que o seu carrinho contenha algum produto, a rota será identificativa desse fluxo e terá o parâmetro “Login” em vez de encomenda, fazendo com que o componente *Guest* não possa ser visualizado nesta página.

Para além desta modificação no comportamento da página de *Login* também foram feitas alterações a página de encomenda de produtos. Assim, após o desenvolvimento deste componente, é possível a apresentação do email guardado em estado na página para que o utilizador consiga saber qual o email que escolheu para efetuar a compra. Nesta página também foram modificados os comportamentos de alguns componentes, para que os campos que permitiam guardar os dados na conta do utilizador não sejam mostrados, já que após a compra, o utilizador não fica com uma conta associada ao seu e-mail, na aplicação *web*.

4.5 Acessibilidade

ADA (*Americans with Disabilities Act*) é uma lei, no âmbito dos direitos civis Americanos, que proíbe a discriminação e garante que as pessoas que têm algum tipo de deficiência têm os mesmos direitos e as mesmas oportunidades que outras pessoas [40].

Esta norma foi escolhida pelo cliente como requisito, devido ao fator de os seus produtos terem muita procura no mercado americano. Para conseguir que o sítio *web* desenvolvido cumpra a lei que proíbe a discriminação (*ADA compliant*), este tem de seguir as normas WCAG (*Web Content Accessibility Guidelines*). Para cada uma das diretrizes que compõem o WCAG existem três níveis de conformidade que refletem o nível de prioridade:

A – Baixo. Caso não cumpra estas normas a tecnologia de acessibilidade pode nem conseguir ler, entender ou funcionar completamente com a página *web*.

AA – Médio. Respeita as normas para os utilizadores navegarem no sítio *web* e nos diferentes tipos de conteúdo que este apresenta.

AAA – Alto. Atinge facilmente todos os critérios descritos para que o sítio *web* e os seus diferentes conteúdos sejam acessíveis, para um variado tipo de utilizadores com deficiência [41].

No início do projeto, o cliente acordou com a Mindera que o nível do sítio *web* a ser desenvolvido tinha que cumprir o nível AA de conformidade para as normas WCAG. A obtenção deste nível de conformidade foi um dos grandes desafios apresentado à equipa, por um conjunto de razões.

A primeira respeito à falta de alguma experiência nesta área e com as várias tecnologias envolvidas no ato de navegação na *web* no contexto de pessoas com deficiência. Efetivamente, projetar e desenvolver um sítio *web* com o nível de conformidade exigido implica um conhecimento de como as pessoas com diferentes tipos de deficiência conseguem navegar num sítio *web*. Muitas vezes são utilizadas tecnologias como, os leitores de ecrã (*screen readers*), ampliadores de ecrã (*screen magnification*), comandos de voz para interagir com software (*speech input*), entre outros [41].

Acresce, ainda, que para se obter o nível de conformidade exigido, o sítio *web* era submetido a uma auditoria, por parte de uma empresa externa, num dado momento. Durante esse tempo, o desenvolvimento do projeto continuava, e quando mais tarde recebíamos os resultados da auditoria, algumas das situações descritas não estavam de acordo com o estado atual do desenvolvimento.

No fim do desenvolvimento do projeto parece-me que houve uma grande evolução da equipa quanto a sensibilização das questões de acessibilidade e um melhor entendimento sobre as mesmas, fazendo com que o pensamento para planear tarefas futuras seja diferente e agregando já o esforço necessário para que futuros projetos sejam *ADA compliant*.

Serão descritas três tarefas do total de dezasseis desenvolvidas no âmbito da acessibilidade por se terem demonstrado mais desafiantes.

1.ª Tarefa (Correção de várias áreas de acessibilidade):

Numa primeira auditoria, uma das tarefas que tive que realizar foi corrigir diversos problemas referentes a estrutura da informação. Os estavam descritos num documento muito extenso, onde era referido qual a situação encontrada e a área da aplicação onde surgia (e.g. *homepage*, *account*). O facto do documento estar dividido por áreas fez com que muitos dos problemas aparecessem repetidos. Por exemplo, a existência de um `<h4>` no *footer* era descrito no documento na área da *homepage*, mas como este se encontra em quase todas as páginas do sítio *web*, surgiu em praticamente todas. Foi necessário que a equipa de desenvolvimento fizesse a triagem de quais os problemas é que era realmente necessário corrigir. Para além disso, os problemas apareciam muito

concretos e como me encontrava numa fase inicial de contactar com as situações relacionadas com acessibilidade, procurei resolvê-los não tendo muito a noção se a resolução funcionaria com a estrutura do projeto total e com as resoluções aplicadas para outros problemas relacionados com ADA.

2.ª Tarefa (Correção de *headings* de acordo com a hierarquia):

Na segunda auditoria, um dos problemas encontrados implicou a correção dos *headings* de acordo com a profundidade que a estrutura do sítio *web* deve devia apresentar. Este problema requer uma boa compreensão da estrutura de um sítio *web* acessível, em que os *heading* devem estar encadeados com uma sequência lógica.

Para proceder à resolução desta situação, foi necessário percorrer todas as páginas da aplicação *web* e, com uma ferramenta adequada para mostrar o *outline* da página, verificar se a estrutura já se encontrava com a profundidade correta, de forma a cumprir a norma 1.3.1 “Info and Relationships” [41]. Depois de localizar onde o erro se encontrava e seleccionar a *markup* que melhor se adequa a esse elemento, foram utilizadas as técnicas descritas em “G141: *Organizing a page using headings*” [41] para resolver a situação.

Para se poder entender melhor este problema, a título de exemplo é apresentada a estrutura que a página da Universidade de Trás-os-Montes e Alto Douro contém na sua *Homepage* (Figura 21). As áreas representadas por um retângulo vermelho representam alguma incoerência relativamente as normas definidas pela ADA.

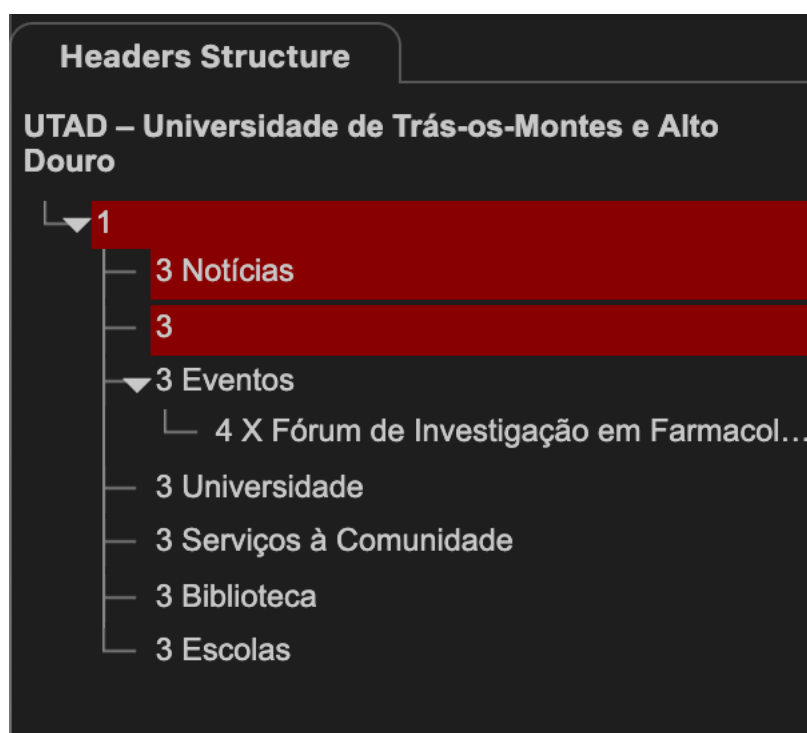


Figura 21 – Headings Página UTAD

No primeiro caso verifica-se que existe um elemento sintático `<h1>` mas que se apresenta sem texto. No segundo ponto verifica-se que os *headings* não estão organizados de acordo com a sua especificidade, já que para respeitar as normas o `<h3>` referente às “Notícias” deveria ser trocado por um `<h2>`. No seguinte *heading* verifica-se que não se encontra preenchido com texto e teria que ser revista a estrutura para decidir se este cabeçalho seria mesmo um `<h3>` ou se seria necessário mudá-lo para um `<h2>`.

Com a resolução desta situação, ficamos com a noção da importância que é pensar na estrutura de uma página web previamente e como um todo, para assim poderem ser evitados problemas com a regulação relativa à acessibilidade, desenvolver aplicações com qualidade para todos e evitar problemas de ADA relacionados com o *outline* no futuro.

3.ª Tarefa (Correção de cores para obtermos o contraste mínimo):

Outra inconformidade encontrada pela auditoria está relacionada com o contraste de cores entre elementos do sítio web. Se um elemento tem uma cor de fundo e

apresenta um texto com uma cor muito parecida, isto faz com que a leitura do texto deste elemento seja muito complicada para pessoas que tenham perda de visão, pois o contraste entre a primeira cor e a segunda é muito pequeno.

Para tornar o sítio *web* acessível a pessoas que tenham um baixo grau de visão, foi necessário analisar toda a aplicação *web* e alterar algumas cores que estavam definidas nos designs de forma a cumprir a norma 1.4.3 “Contrast (Minimum)” [41] da ADA, que está relacionada com o contraste mínimo. Este terá de ser numa relação, de pelo menos, 4.5:1 para textos e de, pelo menos, 3:1 para grande escala de texto (que tenha, pelo menos, 18 pontos de tamanho ou 14 pontos, se a fonte estiver a negrito) para que a aplicação esteja conforme as normas.

Tal como na situação anterior, apresenta-se o exemplo da *Homepage* da Universidade de Trás-os-Montes e Alto Douro, analisando os problemas de contraste que apresenta.

Na figura 22 está representado no título o tipo de problema que é necessário resolver, em subtítulo encontra-se o grau nível (neste caso, AA, a que a norma 1.4.3 *Contrast* [41] apresentada se refere). É, ainda, apresentado um pequeno texto que descreve a norma que está a ser violada.

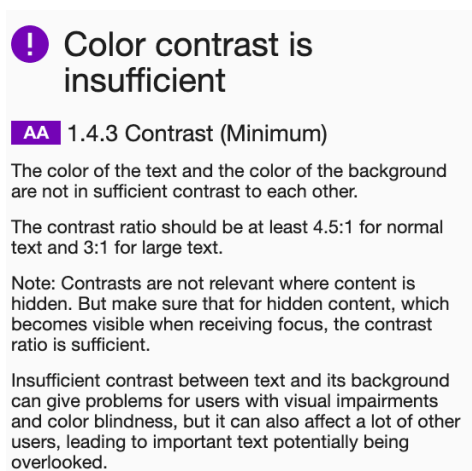


Figura 22 – Descrição do problema de contraste de cor na página da UTAD

Na Figura 23 estão representadas algumas das ocorrências da página em questão, quais as cores que foram comparadas (azul #364e72 e azul escuro #081b31), o resultado do contraste (2.05) e onde é que este se apresenta (Universidade). Neste caso, para que a página *web* fosse conforme o normativo de acessibilidade, seria

necessário substituir a cor por uma com que se conseguisse atingir, pelo menos, 4.5 em contraste.

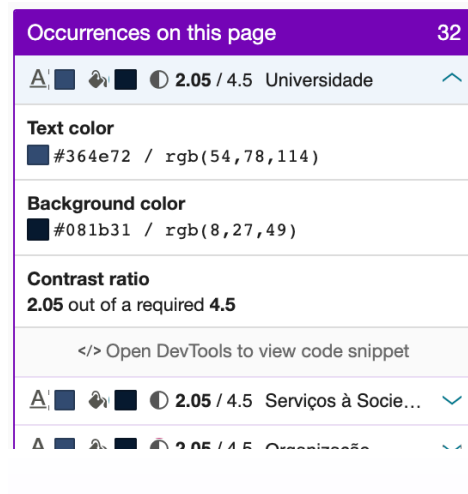


Figura 23 – Resultado contraste de cor

Capítulo V

Conclusões

Este relatório tem como base uma escolha pessoal das opções disponibilizadas para a conclusão do Mestrado em Multimédia da UTAD. Esta escolha foi influenciada pela evolução que pretendia ter na área das aplicações *web*, aproximada com o mundo empresarial. O estágio foi uma excelente forma de ter o primeiro contacto com este meio e sentir as suas exigências, bem como a sua rápida evolução e a procura pela utilização de tecnologias de ponta.

Relativamente aos objetivos propostos para este estágio, o facto de ter sido integrada num projeto que estava a ser iniciado pela empresa permitiu que tivesse contacto com todas as fases do desenvolvimento de uma aplicação *web* e, assim, ficar com conhecimento dos requisitos necessários para o resultado final.

Quanto à utilização da metodologia Agile, penso que é uma mais-valia para toda a organização e para a realização das tarefas de uma forma mais agilizada, fazendo com que o programador veja o seu trabalho facilitado. Ao nível pessoal, penso que, numa primeira fase, e para quem nunca teve contacto com a metodologia, o número de eventos e termos utilizados, relativos à Agile, causa alguma dificuldade na integração. Mas, devido à ajuda dos colaboradores da Mindera, esta integração tornou-se mais fácil.

É de destacar que a explicação das tarefas realizadas, das técnicas aplicadas, e qualquer assunto relacionado com o projeto ou com o cliente se tornou mais complexo do que previsto inicialmente, devido à não ser possível a divulgação de qualquer fator que possivelmente identificasse o projeto ou o cliente em questão.

Para finalizar, a nível pessoal penso que houve uma evolução enorme relativamente à forma do desenvolvimento de aplicações *web*, bem como do pensamento crítico relativamente às tarefas ou ainda a atenção que dou a cada tarefa

e aos seus requisitos, sendo que, no final do estágio, sentia que já conseguia pensar quais as implicações quando desenvolvendo uma tarefa de uma determinada forma, em comparação com outra forma diferente.

Concluo considerando que o estágio me foi extremamente útil neste primeiro contacto com a realidade empresarial na área do desenvolvimento *web*. No final do período de estágio, a Mindera sinalizou a sua vontade em que continuasse a colaborar na empresa, tendo sido contratada em 8 de Julho de 2019.

Referências bibliográficas

1. *Frontend vs Backend*. Retrieved 2019, from
<https://www.geeksforgeeks.org/frontend-vs-backend/>
2. Manzoor A. (2010) *E-Commerce: An Introduction*.
3. *Business encyclopedia – Ecommerce*. Retrieved 2019, from
<https://www.shopify.com/encyclopedia/what-is-ecommerce>
4. *E-Commerce*. Retrieved 2019, from
https://www.tutorialspoint.com/e_commerce/index.html
5. Islam, K. (2013) *Agile Methodology for Developing & Measuring Learning: Training Development for Today's World*.
6. *Manifesto for Agile Software Development*. Retrieved 2019, from
<http://agilemanifesto.org>
7. Littlefield, A. (2016). *The Beginner's Guide to Scrum and Agile Project Management*. Retrieved 2019, from
<https://blog.trello.com/beginners-guide-scrum-and-agile-project-management>
8. Eby, K. (2016) *Comprehensive Guide to the Agile Manifesto*. Retrieved 2019, from
<https://www.smartsheet.com/comprehensive-guide-values-principles-agile-manifesto>

9. Stellman, A., Greene, J. (2014) *Learning Agile: Understanding Scrum, XP, Lean and Kanban*.
10. Taymor, E. *Agile Handbook*. Retrieved 2019, from
<http://agilehandbook.com/agile-handbook.pdf>
11. Schwaber, K. Sutherland, J. (2017). *The Scrum Guide™ The Definitive Guide to Scrum: The Rules of the Game*. Retrieved 2019, from
www.scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-US.pdf#zoom=100
12. James, M. (2010). *Scrum Reference Card About Scrum A Management Framework*. Retrieved 2019, from
www.collab.net/sites/default/files/uploads/CollabNet_scrumreferencecard.pdf
13. *Newbies' Guide to Scrum Project Management 101*. Retrieved 2019, from
<https://www.ntaskmanager.com/blog/newbies-guide-to-scrum-project-management-101/>
14. Rossel, S. (2017) *Continuous Integration, Delivery, and Deployment: Reliable and faster software releases with automating builds, tests and deployment*.
15. (2019) *Agile Alliance – Agile Glossary*. Retrieved 2019, from
<https://www.agilealliance.org/agile101/agile-glossary>
16. Duvall, P., Matyas, S., Glover, A. (2007) *Continuous Integration: Improving software quality and reducing risk*.
17. Chancon, S., Straub, B. (2014) *Pro Git*

18. *Getting Started - Git Basics*. Retrieved 2019, from
<https://git-scm.com/book/en/v1/Getting-Started-Git-Basics>
19. *The Pursuit of Continuous Testing*. Retrieved 2019, from
<https://www.functionize.com/blog/the-pursuit-of-continuous-testing/>
20. Boise, C. (2016). *New to web development? Here are 25 common terms, defined*. Retrieved 2019, from
<https://medium.com/@BoiseCodeWorks/curious-about-web-development-here-are-25-common-terms-defined-2dd71c1d0d64>
21. Lindley, C. (2018). *Front-end Developer Handbook 2018*. Frontend Masters. Retrieved 2019, from
<https://frontendmasters.com/books/front-end-handbook/2018/>
22. Hartson, R, Pyla, P.S. (2012) *The Ux Book: Process and guidelines for ensuring a quality user experience*.
23. Bank, C, Cao, J. (2014) *Web UI design best practices*.
24. (Last update 2019). *Learn web development - MDN web docs*. Retrieved 2019, from
<https://developer.mozilla.org/en-US/docs/Learn>
25. Robbins, J. (2012) *Learning Web Design: A Beginner's Guide to HTML, CSS, JavaScript and Web Graphics*.
26. Raymond, E. (2004) *The Art of Unix Programming*.

27. Gackenheimer, C. (2015). *Introduction React*. Retrieved 2019, from
<https://pepa.holla.cz/wp-content/uploads/2016/12/Introduction-to-React.pdf>
28. (Last update 2019) *React – A JavaScript library for building user interfaces*. Retrieved 2019, from
<https://reactjs.org>
29. *30 Days of React – An introduction to React in 30 bite-size morsels*. Retrieved 2019, from
www.fullstackreact.com/assets/media/sGEMe/MNzue/30-days-of-react-ebook-fullstackio.pdf
30. *Demystifying React*. Retrieved 2019, from
<https://www.codemag.com/Article/1809041/Demystifying-React>
31. Singh, H., Bhatt, M. (2016) *Learning Web Development with React and Bootstrap*.
32. *Introducing JSX*
<https://reactjs.org/docs/introducing-jsx.html>
33. *Leveling Up with React: Redux*. Retrieved 2019, from
<https://css-tricks.com/learning-react-redux/>
34. Westfall, B. (2016) *Leveling Up with React: Redux*. Retrieved 2019, from
<https://css-tricks.com/learning-react-redux/>
35. *Lodash*. Retrieved 2019, from

<https://lodash.com>

36. Testing microservices: Challenges and Strategies. Retrieved 2019, from
<https://testsigma.com/blog/testing-microservices-challenges-and-strategies-testsigma/>
37. Jest. Retrieved 2019, from
<https://jestjs.io>
38. Enzyme. Retrieved 2019, from
<https://airbnb.io/enzyme/>
39. Salinger, S., Prechelt, L. (2013) Understanding Pair Programming: The Base Layer.
40. Introduction to the ADA: Information and Assistance on the Americans with Disabilities Act. Retrieved 2019, from
https://www.ada.gov/ada_intro.htm
41. W3C. Retrieved 2019, from
www.w3.org