INSTITUTO
UNIVERSITÁRIO
DE LISBOA

Metodologia de avaliação e comparação de ferramentas de desenvolvimento de *frontend* para *websites*

Rui de Jesus Figueiredo Ribeiro

Mestrado em Engenharia de Telecomunicações e Informática

Orientador:
Professor Doutor Jorge Manuel Anacleto Louçã, Professor Catedrático
ISCTE-IUL Instituto Universitário de Lisboa

Outubro, 2020

Agradecimentos

Aos meus colegas de curso e amigos, que me acompanharam ao longo da minha jornada académica.
Ao meu orientador, Professor Doutor Jorge Louçã, pelo apoio e disponibilidade nestes últimos meses.
À professora Noelma Viegas, pela grande ajuda na revisão e correção linguística.
Ao meu colega de trabalho e amigo, Dário Santos, por ter sido o meu mentor na área do desenvolvimento de *frontend*.
À minha namorada, Maria, pela motivação e ajuda, desde o início desta dissertação.
Aos meus pais, por me terem incentivado sempre a ir mais longe na minha formação académica e por serem o meu porto seguro.

Resumo

Esta dissertação tem como objetivo principal propor uma nova metodologia específica para avaliação e comparação de ferramentas de desenvolvimento de *frontend* para *websites*. A metodologia proposta caracteriza as ferramentas de desenvolvimento e permite, posteriormente, selecionar a melhor. Deste modo, são indicados os critérios mais relevantes para a escolha de ferramentas, o que resulta numa redução significativa do universo de ferramentas que se pretendem comparar. Após a seleção inicial, são apresentados os conceitos a ter em conta na atividade de desenvolvimento com as ferramentas. Terminadas as duas primeiras fases, são comparadas as ferramentas com melhor classificação do ponto de vista do desenvolvimento aplicacional, sendo realizados testes e comparados dados técnicos. De forma a comprovar a sua qualidade, esta metodologia é avaliada através de um caso de estudo, que compara as duas ferramentas de desenvolvimento de *frontend: React* e *Angular*.

Palavras chave: comparação; avaliação; ferramentas de desenvolvimento; *frontend*.

Abstract

The main objective of this dissertation is to propose a new specific methodology for evaluating and comparing frontend development tools for websites. The proposed methodology characterizes the development tools and allows the selection of the best of them. The most relevant criteria to choose the development tools are pointed, allowing a significant reduction of the tool universe to be contemplated. After the initial selection, the methodology presents which concepts should be considered on developing with the tools. After the previous phases, the tools with the best classification are compared from the development point of view, by implementing evaluation tests, as well as by comparing technical data. In order to prove its quality, the proposed methodology is evaluated via one case of study, where two development tools are compared: Angular and React.

Key words: comparation; evaluation; development tools; frontend.

Índice

Índice de figuras

Índice de tabelas

Lista de abreviaturas e siglas

FE – *Frontend*

UI – *User interface*

JS – *JavaScript*

TS – *TypeScript*

TDD – *Test Driven Development*

CLI – *Command line interface*

CAPÍTULO 1

Introdução geral

1.1. Enquadramento do tema

A evolução da tecnologia atingiu um ritmo tal, que, atualmente, é no dia a dia que nos apercebemos dessa realidade; para além disso, tornou-se fundamental para o desenvolvimento económico da sociedade. De facto, empresas de todo o tipo, quer no setor público quer no setor privado, estão dependentes da tecnologia para o sucesso dos seus objetivos. O ser humano, por sua vez, enquanto ser individual, usufrui da tecnologia para a sua própria comodidade. Mas este facto atingiu proporções inimagináveis, ao ponto de se apontar, entre a comunidade científica, para a existência de um fenómeno comportamental, que é a dependência dos produtos tecnológicos.

Entretanto, a capacidade de acesso, com apenas um clique, a um mundo interligado pela *Internet* é algo que proporciona uma experiência altamente benéfica ao ser humano, desde que haja sabedoria na sua utilização. Este "novo mundo" contribuiu, e continua a contribuir de forma crescente e positiva, para a realização do indivíduo, a nível pessoal e profissional. Porém, o utilizador depende de aplicações que permitam a sua interação com a *Internet*. Esta interação é conseguida através do acesso à informação pretendida, que será de facto proveitosa, caso seja facilmente consultada ou gerida. Por esta razão, surgiram as ferramentas de desenvolvimento de *frontend* (FE). Naturalmente, com o passar do tempo, associado ao rápido desenvolvimento tecnológico, por um lado, e, por outro, ao objetivo fulcral de bem servir o consumidor, foi sendo criada grande quantidade e variedade de ferramentas de FE. Surge assim a necessidade de saber escolher, de saber optar pela ferramenta que melhor se adequará ao negócio pretendido e ao serviço prestado.

1.2. Motivação e relevância do tema

No decorrer dos últimos cinco anos, foi-me dada a oportunidade de trabalhar com diversas tecnologias e ferramentas de desenvolvimento de aplicações *web*. Esta experiência permitiu-me expandir o meu gosto especial pelo desenvolvimento de FE, onde me tenho vindo a especializar nos últimos dois anos.

Como muitos programadores inexperientes, senti inicialmente a necessidade de encontrar as tecnologias e ferramentas mais recentes e conceituadas do mercado, por forma a ganhar o conhecimento necessário que me encaminhasse num percurso profissional de excelência.

Aliás, a determinação da melhor ferramenta de desenvolvimento *web* é tema de grande discussão e de sucessivas discórdias entre os programadores. Este facto, ao contrário de esclarecer os interessados,

tem dificultado a obtenção de informação inequívoca neste domínio, para além de transformar o simples ato de escolher uma ferramenta de desenvolvimento num obstáculo quase intransponível

O problema começa logo na diversidade de ferramentas de FE existentes no mercado, o que leva o utilizador a confrontar-se com diferentes conceitos e metodologias de desenvolvimento. Depois, a quase inexistente comparação teórica e prática destes conceitos e metodologias aumenta a dificuldade sentida pelos programadores, quando pretendem comparar e, por fim, eleger o produto pretendido.

1.3. Objetivo geral

Esta dissertação procura dar resposta ao problema acima descrito, comparando ferramentas de desenvolvimento de FE, de forma a facilitar a escolha final. A nossa hipótese de investigação partiu do princípio de que uma nova metodologia de avaliação e comparação de ferramentas de desenvolvimento de FE, a partir de critérios relevantes e bem definidos, permitirá a escolha da melhor ferramenta de FE.

1.4. Questões e objetivos de investigação

Como proceder à análise e comparação das ferramentas de desenvolvimento existentes no mercado, de forma fundamentada e adequada? Como proceder à eleição da ferramenta de desenvolvimento que melhor se adeqúe às finalidades apontadas pelo utilizador?

Inicialmente, o objetivo desta dissertação era a comparação das duas principais ferramentas de desenvolvimento de FE para *websites*, atualmente existentes no mercado. No entanto, devido à pouca quantidade, ou mesmo inexistência de material que viabilize tal estudo, este trabalho evoluiu para a satisfação de outro objetivo relacionado com o primeiro, que consiste na implementação de uma metodologia que permita a escolha da ferramenta mais indicada para o desenvolvimento de FE. Para tal, é necessário proceder à seleção dos critérios mais relevantes, de forma a obter uma fundamentação sólida e completa da comparação e escolha de tecnologias FE.

Definida a metodologia, ela será utilizada para levar a cabo a escolha das duas ferramentas de desenvolvimento de FE mais conceituadas. Após escolha das duas ferramentas, manter-se-á a metodologia, agora numa perspetiva mais profunda e técnica, para analisar e comparar as duas ferramentas, através de casos de estudo. Terminada essa análise, tornar-se-á mais clara a decisão sobre qual das duas ferramentas de desenvolvimento de FE escolher, validando-se deste modo a utilidade da metodologia aqui proposta.

CAPÍTULO 2

Revisão da Literatura

2.1. Introdução à revisão da literatura

O objetivo deste ponto é a exposição e descrição dos fundamentos do trabalho e das propostas atualmente existentes, disponíveis na literatura e nas mais variadas plataformas *web*. Esta revisão encontra-se organizada em secções sequenciais. Inicialmente, são referidas as fontes de informação; de seguida, é abordada a necessidade do recurso a ferramentas de desenvolvimento de FE; finalmente, são referidos os critérios de escolha de uma ferramenta de desenvolvimento. As fontes de informação utilizadas indicam quais os mecanismos e técnicas utilizadas para se proceder à pesquisa e à recolha de informação que suportam este estudo. A secção sobre a necessidade de utilização de uma ferramenta de desenvolvimento apresenta, de forma detalhada, os motivos e benefícios da ferramenta. Finalmente, a secção sobre os critérios de escolha de uma ferramenta de desenvolvimento expõe, de forma ordenada e detalhada, os critérios mais comuns a ter em conta na escolha de uma ferramenta de desenvolvimento.

2.2. Fontes utilizadas

As ferramentas de desenvolvimento de FE são tecnologias que se encontram em constante evolução, de forma a acompanhar as necessidades das mais recentes aplicações. O facto de serem novidade e de sofrerem constantes atualizações faz com que seja penoso localizar artigos válidos sobre estas tecnologias, devido à sua rápida desatualização. Ainda assim, localizar informação sobre este assunto não é impossível.

As informações fornecidas sobre as tecnologias de desenvolvimento de FE encontram-se essencial e maioritariamente em plataformas *online*, com o objetivo de facilitar a constante partilha de conhecimento. Estas informações encontram-se presentes em vários formatos, como por exemplo, os *sites* de documentação oficial das ferramentas existentes, conforme os pontos [47], [48] e [49] da bibliografia, respeitantes às ferramentas React, Angular e Vue. Para além destas, também encontramos informação em fontes externas, como por exemplo, em artigos científicos tais como "*React vs. Angular: The Complete Comparison*", de Mosh Hamedani [2] e "*Comparing the most popular frontend JavaScript frameworks*", de Debbie Otuagomah [4] ou aplicações como a "*js-framework-benchmark*", criada por Stefan Krause [19]; outros locais podem ser a plataforma YouTube e os fóruns, para tal bastando pesquisar o nome da tecnologia pretendida.

A divulgação de informação *online* é essencial quando se considera a pesquisa de dados relativos às tecnologias mais recentes do mercado. Contudo, é preciso ter especial atenção à sua seleção. A informação disponível *online* não é completamente fidedigna e, muitas vezes, é partilhada por

indivíduos sem experiência na área, o que pode levar a conclusões erradas. De facto, os autores de tais informações podem- mostrar-se como programadores experientes na utilização das ferramentas, mesmo quando tal não corresponde à realidade. Estes utilizam as plataformas *web* para partilhar as suas experiências e conhecimento empírico, de forma a que tais informações possam ajudar programadores nos mais variados assuntos, o mesmo ocorrendo em fóruns, onde existe espírito de entreajuda e partilha de informação sobre os mais variados assuntos relacionados com as ferramentas de desenvolvimento, porém sem qualquer rigor científico comprovado.

2.3. Vantagens das ferramentas de desenvolvimento

As ferramentas de desenvolvimento são caracterizadas por estruturas aplicacionais pré-definidas, que servem de base para o desenvolvimento aplicacional. Por "estrutura aplicacional" compreende-se uma estrutura de pastas e ficheiros já definidos, onde o programador irá adicionar os seus próprios ficheiros com código. Como explicado por M. Pekarsky no seu artigo *"Does your web app needs a frontend framework"* [1], as ferramentas de desenvolvimento têm a responsabilidade de ajudar o programador a construir aplicações, cuidando dos problemas base do desenvolvimento aplicacional.

As ferramentas de desenvolvimento *web* podem ser de dois tipos, *frameworks* ou *bibliotecas*. Apesar de apresentarem o mesmo objetivo, existem algumas diferenças entre elas. M. Hamedani explica-as no seu artigo *"React vs. Angular: The Complete Comparison"* [2], onde compara as ferramentas de desenvolvimento de FE, React e Angular, *biblioteca* e *framework*, respetivamente. Uma *framework* apresenta as três camadas MVC, modelo, visualização e controlador, representado no Anexo A; apresenta bastantes funcionalidades *out-of-the-box*, que não preocupam o programador com escolhas de *bibliotecas* para usufruto de funcionalidades específicas, tal como o roteamento entre páginas; também oferece sugestões sobre a estruturação de uma aplicação, o que permite ao programador o rápido início de desenvolvimento de código. Uma *biblioteca de desenvolvimento de FE*, por outro lado, oferece mais liberdade de escolha ao programador: pode apenas providenciar a camada visualização, o que obriga o programador a tomar decisões na escolha de *bibliotecas* externas a integrar para conseguir suportar as camadas de modelo e de controlo.

A diferença técnica encontra-se explicada por B. Wozniewicz, no seu artigo *"The Difference Between a Framework and a Library"* [3], onde afirma que, ao utilizar uma *biblioteca*, o programador torna-se responsável pelo fluxo da aplicação. Mas quando o programador utiliza uma *framework*, esta é responsável por este fluxo, devido às funcionalidades pré-existentes que ditam o comportamento do programador ao longo do desenvolvimento aplicacional.

As principais considerações no uso de uma *framework*, ou *biblioteca de desenvolvimento de FE*, são apresentadas por E. Normand, no seu artigo *"Why do we use web frameworks"* [5], que explica resumidamente quatro importantes aspetos a ter em conta. O primeiro relaciona-se com a complexidade da *web* e das linguagens necessárias para proceder ao seu desenvolvimento; o segundo

refere-se à necessidade de recriar funcionalidades básicas, repetidamente, quando estas são sempre necessárias, tal como formulários de autenticação; o terceiro salienta que a *web* tem muitas falhas de segurança; finalmente, alerta para a necessidade de tomada de várias pequenas decisões no desenvolvimento de uma aplicação. Todos estes quatro aspetos são facilmente resolvidos pela utilização de ferramentas de desenvolvimento.

M. Pekarsky explica a importância da utilização de uma ferramenta de desenvolvimento de FE fazendo a sua comparação com os modos de desenvolvimento anteriores à existência destas mesmas ferramentas. Para além disso, refere as mais-valias associadas ao uso das ferramentas e faz uso de um caso de estudo, que relembra a forma mais simplista e, ao mesmo tempo, mais datada de criar a camada de FE de aplicações *web*, feita através de três ficheiros, *JavaScript* (JS), HTML e CSS. Percebemos que, para uma aplicação pequena, é concebível uma arquitetura usando estes conceitos. No entanto, com o crescimento da aplicação é de imaginar a criação de mais três ficheiros, mais três ficheiros e por aí em diante. Tal arquitetura tornar-se-á demasiado complicada de gerir e de manter, devido à quantidade de ficheiros e, sobretudo, à sua ordenação. Em termos de funcionamento aplicacional neste formato já datado, surgem diversos problemas ao nível da *performance*. Ao serem consumidos dados do *backend* (BE), é necessário o seu processamento para atualizar as páginas HTML através de JS, adicionando novos elementos à DOM. Secções da aplicação idênticas obrigam a criação de ficheiros extra com duplicação de código. Com a necessidade de obtenção de dados inseridos na *user interface* (UI) pelo utilizador, existem apenas duas possibilidades: a procura pela DOM do elemento que foi atualizado ou a adição de eventos de escuta em cada um dos elementos de que se pretende obter o valor. Tudo isto torna o código de difícil interação e interpretação, podendo fazer com que qualquer alteração comprometa o correto funcionamento aplicacional. Torna-se inevitável a dificuldade de leitura, compreensão e manutenção do código desenvolvido [1].

Ora, a utilização de uma ferramenta de desenvolvimento de FE procura a resolução dos problemas suprarreferidos, começando logo por facilitar o desenvolvimento de código fácil de ler, compreender e manter. O desenvolvimento é feito em componentes, o que permite a reutilização do código sem ter de se reescrever os mesmos componentes para serem utilizados num local diferente. A representação e manipulação de dados da UI é feita de forma simplificada, através do uso de variáveis internas. Este procedimento permite a centralização de estilos CSS, para que exista coerência dentro do escopo do projeto. Uma ferramenta de desenvolvimento de FE ainda oferece diversas ferramentas e mecanismos que combatem os mais comuns problemas de FE [1].

2.4. Critérios de escolha de ferramentas de desenvolvimento

A existência de múltiplas ferramentas de desenvolvimento *web* torna a escolha de uma ferramenta numa tarefa difícil e complexa. Ultrapassar este constrangimento obriga à eleição de critérios a ter em conta na seleção de uma ferramenta de desenvolvimento de FE. O facto dos conceitos de *framework* e

biblioteca serem recentes faz com que não existam fontes únicas de informação relativas aos critérios mais corretos a considerar. De entre os mais variados critérios existentes, é possível fazer uma seleção dos mais relevantes para a escolha de uma ferramenta de desenvolvimento e, adicionalmente, dos critérios mais importantes a ter em conta, quando se trata de uma ferramenta de FE.

2.4.1. Documentação e conteúdos disponíveis

As ferramentas de desenvolvimento de aplicações *web* apresentam, maioritariamente, um formato de desenvolvimento aplicacional próprio, que deve ser respeitado. A utilização de uma ferramenta de desenvolvimento, de um modo geral, só faz sentido quando se tira daí o maior proveito, como referido por W. Koffel em *"Choosing a Web framework"* [10]. A disponibilização de variadas ferramentas no mercado faz com que exista disparidade entre metodologias, estruturas e conceitos. A existência de documentação torna-se assim num dos mais importantes fatores a ter em conta na escolha de uma ferramenta de desenvolvimento. Sem uma documentação apropriada, as ferramentas de desenvolvimento são extremamente difíceis de compreender, tornando as curvas de aprendizagem bastante largas e, portanto, resultando numa má experiência para novos programadores, tal como referido por Aguiar e David no seu *"A Minimalist Approach to Framework Documentation"* [6]. A documentação de uma ferramenta de desenvolvimento de *software* deve permitir ao programador a compreensão do seu propósito. Os dados apresentados na documentação devem considerar explicações, exemplos práticos e tutoriais de implementação, a fim de possibilitar a fácil compreensão e iniciação do programador. Adicionalmente, como publicado por Butler, G., Dénommé, P. em *"Documenting frameworks"* [7], estas mesmas informações devem estar organizadas por secções, que resultam num fácil e rápido acesso à informação pretendida. W. Koffel propõe a leitura da documentação das ferramentas [10], que, caso seja de difícil compreensão, pouco clara, significa que a aprendizagem de desenvolvimento com tal ferramenta não é aconselhada. É evidente que as ferramentas de desenvolvimento dependem fortemente de uma boa documentação, que permita ao programador uma fácil iniciação na tecnologia e um bom acompanhamento ao longo dos seus desenvolvimentos. Porém, nem sempre a documentação da tecnologia é suficiente, o que faz com que a existência e disponibilidade de informação sobre as ferramentas, para além da documentação oficial, seja um importante fator a ter em conta, como indicado por G. Bakradze, no seu artigo *"How to Choose the Best Frontend Framework"* [8]. O autor refere que a popularidade das novas ferramentas de desenvolvimento *web* é de tal forma considerável que, para além da documentação oficial anteriormente falada, existem várias fontes de informação alternativa. Tal informação é projetada e desenvolvida por programadores experientes na tecnologia, que procuram consolidar as funcionalidades e metodologias de desenvolvimento das tecnologias, através dos seus estudos e experiências profissionais. A divulgação de informação alternativa é feita de variados modos, a partir de vídeos, cursos, artigos e outros tipos de materiais, grátis ou pagos.

2.4.2. Comunidade

A ”comunidade de desenvolvimento” é o nome dado a um grupo de indivíduos que se reúnem para agir sobre aquilo que é importante, como refere o SCDC, no artigo *“What is community development”* [11]. Na maior parte das vezes, uma *comunidade de desenvolvimento* encontra-se maioritariamente em plataformas *web*. Estas plataformas permitem que programadores de todo o tipo, do aprendiz ao programador mais experiente, partilhem informações de âmbito pessoal ou profissional, relativas às tecnologias de desenvolvimento. No âmbito das ferramentas de desenvolvimento, C. Varisco, no seu texto publicado na página *web* da Hackermoon [14], afirma que quanto mais barulhentas forem e mais membros apresentarem as comunidades melhor. Isto porque uma comunidade ativa aumenta a documentação sobre a ferramenta, causando uma redução no tempo de pesquisa de informação, por parte dos programadores que a estão a utilizar. As comunidades ajudam a conexão entre indivíduos que se encontrem mais avançados ou mais recuados em relação a diversos temas. As comunidades permitem a discussão de assuntos, consulta de informação e pedidos de ajuda. Todos estes pontos fazem com que um programador, independentemente do seu nível de senioridade, seja capaz de tomar as decisões mais corretas, ou debloquear situações, a partir da interação com outros indivíduos que já passaram por situações iguais ou similares. Ainda, os criadores de uma determinada ferramenta de desenvolvimento nem sempre podem estar disponíveis para dar respostas às questões que possam surgir sobre a utilização da ferramenta. Tal torna a existência de uma comunidade algo bastante proveitoso, que ajuda a resolução de assuntos deste âmbito, como refere Timothy Chaves, no seu artigo na revista *Forbes* [13].

Um bom programador, mesmo apresentando uma elevada capacidade de análise da documentação de uma certa ferramenta de desenvolvimento, confrontar-se-á com situações em que irá questionar a sua interpretação da documentação, ou a sua utilização da ferramenta, algo que só pode ser respondido a partir de experiências de utilização da ferramenta, afirma Siddhartth, no seu artigo para a página *Tutsplus* [12]. Nestes e em muitos outros casos, a existência de uma grande comunidade é altamente benéfica para a partilha de dúvidas e de questões. Uma larga comunidade incluirá sempre programadores cuja experiência é suficiente para aconselhar e explicar conceitos menos percetíveis a outros programadores.

2.4.3. Popularidade

A popularidade é um fator com grande peso, que se pode relacionar com a comunidade acima descrita. Como indicado no artigo *“Ten criteria for choosing the correct framework”*, apresentado pela *Symphony* [16], quanto mais popular a ferramenta de desenvolvimento, maior será o seu tempo de vida, considerando a evolução e os níveis de competência. A popularidade de uma tecnologia ou ferramenta de desenvolvimento pode ser descrita por diversos parâmetros, tanto do ponto de vista individual como empresarial, refere o *site* DigiMantraLabs [15]. A popularidade torna-se importante quando a documentação disponível acerca da tecnologia de desenvolvimento não é completamente transparente; isto porque existem outros indivíduos com as mesmas questões e que se

podem ajudar mutuamente nas tomadas de decisão e conclusões a tirar. O facto de uma ferramenta de desenvolvimento não ser popular traduz-se num reduzido número de programadores disponíveis para oferecer assistência a nível de desenvolvimento aplicacional. Também do ponto de vista empresarial, a dificuldade de localização de indivíduos especializados numa determinada tecnologia de desenvolvimento é um fator muito relevante. A relevância está relacionada com a dificuldade de recrutamento de recursos, que pode ser fundamental para a conclusão de um projeto, por exemplo. Numa equipa, a existência de indivíduos experientes permite que decisões corretas sejam tomadas e que a informação seja passada, de forma mais segura, a programadores menos experientes, que, deste modo, não ficam apenas dependentes da documentação da tecnologia de desenvolvimento, o que poderia conduzir a variadas interpretações. G. Bakradze [8] partilha a sua opinião, partindo do seu caso pessoal , em que a especialização numa ferramenta de desenvolvimento popular o torna numa pessoa com um perfil ambicionado por parte das empresas. A popularidade faz com que o número de projetos que utilizam certa ferramenta aumente, e, com este aumento, também a procura de programadores especializados. Um dos principais motivos da popularidade de uma ferramenta de desenvolvimento é a dimensão da empresa que permitiu a sua criação e que a suporta, pois isso transmite segurança aos programadores que desenvolvem com a mesma ferramenta e também transmite segurança às empresas que escolhem a ferramenta para o desenvolvimento das suas aplicações [8]. Esta segurança encontra-se relacionada com a garantia de suporte na utilização da ferramenta e também na projeção da sua utilização futura.

2.4.4. Curva de aprendizagem

A teoria da curva de aprendizagem explica-se com a correlação entre a *performance* do aprendiz numa tarefa e o número de tentativas, ou o tempo necessário para completar essa mesma tarefa, que muitas vezes é representado em diagrama, como exemplificado no Anexo B, explica Valamis [16].

No caso de uma ferramenta de desenvolvimento, a curva de aprendizagem compreende-se no tempo despendido, necessário para o domínio da sua utilização. Existem vários fatores que determinam o alongamento ou encurtamento de uma curva de aprendizagem, explica Siddhartth [12]. Um dos principais fatores relaciona-se com as linguagens de programação necessárias para a utilização de uma ferramenta de desenvolvimento. Caso a linguagem de programação seja nova para o programador, o mesmo terá de despender tempo para a dominar, aumentando a curva de aprendizagem. O oposto ocorre quando o programador já se encontra familiarizado com as linguagens utilizadas pelas ferramentas de desenvolvimento. Outro dos principais fatores é a dimensão da tecnologia de desenvolvimento. Uma ferramenta grande, que apresente inúmeras funcionalidades, requer tempo de aprendizagem, para se conseguir tirar o maior proveito da ferramenta. A quantidade e qualidade de documentação da ferramenta é outro fator, permitindo ao programador um rápido e fácil acesso à informação desejada, sem que seja necessário proceder a pesquisas em meios alternativos. No entanto, além da existência de boa documentação, a disponibilização de informação em meios alternativos faz com que a curva de

aprendizagem diminua, pois facilita a consulta de informação e a obtenção de respostas . Do ponto de vista empresarial, a curva de aprendizagem é um importante fator, que permite a gestão dos recursos de um determinado projeto e uma análise mais realista da previsão de conclusão de desenvolvimentos.

2.4.5. Funcionalidades oferecidas

As funcionalidades oferecidas por parte de determinada ferramenta de desenvolvimento fazem parte dos principais fatores para determinar a escolha ou exclusão de uma ferramenta. Um dos motivos mais importantes que levam ao desenvolvimento aplicacional, com a utilização de ferramentas de desenvolvimento, é a implementação do produto desejado com o mínimo esforço possível, isto devido às funcionalidades já embutidas, que permitem ao programador concentrar-se nas suas tarefas mais importantes, como referenciado por C. Varisco [14]. Do ponto de vista das aplicações *web*, existem sempre funcionalidades cuja existência é imprescindível, como indicado extensamente por G. Bakradze [8] e pela página *DigiMantraLabs* [15].

2.4.6. Performance

A construção de *websites* e de aplicações que as pessoas gostem de utilizar, ou seja, que atraiam e retenham utilizadores, é apenas possível através de uma boa experiência de utilizador. Parte desta experiência de utilizador é permitida através do rápido carregamento de informação e da rápida resposta às interações do utilizador. A estes conceitos é dado o nome de *performance* na *web*, afirma a documentação do Mozilla [20].

A performance é um fator com elevado relevo a ser considerado na escolha de uma ferramenta de desenvolvimento. Porém, este fator não deve ser decisivo, afirma Timothy Chaves [13]. É previsível que as principais *frameworks* e *bibliotecas de desenvolvimento de* FE já se encontrem com um nível tão elevado de *performance* que seja difícil calcular este nível, mesmo usando ferramentas próprias como a ferramenta de desenvolvimento da *Google*. P. Krane partilha da mesma opinião no seu artigo, afirmando que as tecnologias estão de tal forma evoluídas, que os únicos tempos de espera consideráveis verificados são os de chamada ao BE [17].

2.4.7. Produtividade

Existem variados fatores a ter em conta para considerar a escolha de uma ferramenta de desenvolvimento, das quais se destacam o tamanho da aplicação, o tempo disponível para o seu desenvolvimento e a sua escalabilidade. Timothy Chaves [13] refere que a existência de regras de implementação rígidas, ou seja, a obediência a convenções próprias, faz com que os programadores se tornem mais produtivos na sua utilização, visto que tal não deixa muitas decisões para o programador tomar, o que pode ser benéfico. Ao mesmo tempo, faz com que o produto não seja flexível, estando sempre o programador obrigado a seguir as normas da ferramenta de desenvolvimento. O facto de não

ser flexível faz com que, por vezes, as funcionalidades não sejam as mais atualizadas e que a forma de implementação seja complexa, podendo causar atrasos no desenvolvimento aplicacional.

2.4.8. Flexibilidade

Para se proceder à implementação de uma aplicação, deve ser escolhida uma ferramenta de desenvolvimento cuja arquitetura se identifique com aquilo que se pretende desenvolver, ou então uma ferramenta que seja flexível, diz D. Schesselman, no seu artigo *"Pros & Cons of Frontend Frameworks"* [18]. Uma ferramenta de desenvolvimento deve então dispor de variadas opções que permitam a personalização, através de configurações simples. P. Crane [17] afirma que a existência destas opções é crucial para tornar a ferramenta suficientemente flexível para os mais diversos requerimentos de desenvolvimento.

CAPÍTULO 3

Proposta de uma nova metodologia de avaliação e de comparação de ferramentas de desenvolvimento de *frontend*

3.1. Estrutura da metodologia

A existência de um grande e variado leque de ferramentas de desenvolvimento de FE para aplicações *web* dificulta a escolha da ferramenta mais indicada, consoante os requisitos do projeto e o grau de experiência do programador. A informação disponibilizada, relativa à ajuda da escolha de ferramentas, apresenta diversas opiniões que, maioritariamente, vão ao encontro dos gostos e opiniões do indivíduo que expõe a informação. Tendo por base a minha experiência profissional, que me obriga a pesquisar informação adicional para o desenvolvimento do meu trabalho, posso afirmar , que, de um modo geral, o universo de fontes de informação obtidas na Internet, relativas a este tema ou a qualquer outro, apresentam, em parte, informação que não é coerente, devido ao facto de qualquer pessoa (profissional, académico, amador…) ter a liberdade de publicar o que quer que seja neste meio, fazendo com que o consumidor de tal informação seja induzido em erro, tomando decisões sem uma base sólida. Isto justifica a necessidade de uma nova metodologia, clara e coerente, para a escolha de ferramentas de desenvolvimento de FE para aplicações *web*, a qual me proponho a apresentar.

A metodologia apresentada é uma proposta totalmente original de avaliação e de comparação de ferramentas de desenvolvimento de FE. Esta metodologia proporciona a qualquer indivíduo o desenvolvimento das suas próprias conclusões, relativas à escolha da ferramenta mais apropriada a utilizar. É uma metodologia sequencial que apresenta os passos seguintes.

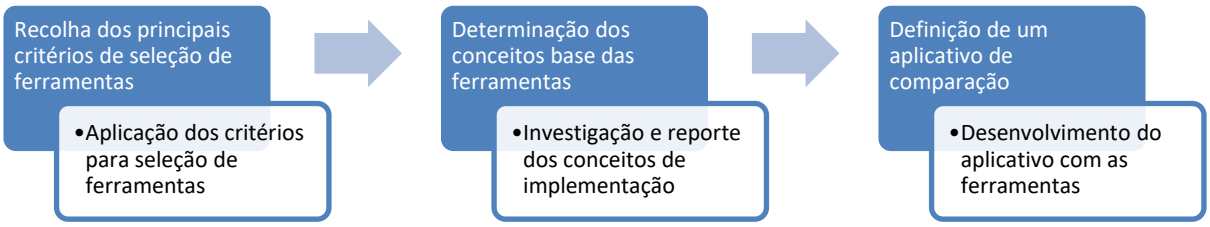


Figura 1-Sequência da metodologia proposta

3.2. Critérios de seleção de ferramentas de desenvolvimento

O primeiro passo a dar na construção da metodologia proposta é a eleição dos critérios com maior relevo, que devem ser considerados para fazer a escolha de uma ferramenta de desenvolvimento. O estado da arte permitiu a reunião de informação referente aos critérios a considerar, juntamente com a

sua justificação e depois de analisado cada critério, separadamente, foram selecionados os cinco mais importantes, a saber, popularidade, curva de aprendizagem, documentação e conteúdos disponíveis, comunidade e *performance*.

Estes critérios, apesar de distintos, encontram-se intimamente relacionados. A sua aplicação para seleção de uma ou mais ferramentas de desenvolvimento assegura um resultado sólido e fundamentado, com perspetivas de evolução futura. Deste modo, e como explicado de seguida, o uso destes cinco critérios de avaliação é o primeiro passo da metodologia proposta, para efetuar a seleção das ferramentas mais indicadas para o desenvolvimento de FE.

3.2.1. Popularidade

A popularidade é um ótimo indicativo a ter em conta, principalmente nas ferramentas de desenvolvimento de FE. A popularidade encontra-se muitas das vezes associada à origem das ferramentas, sendo que uma ferramenta desenvolvida por uma grande empresa transmite segurança aos programadores e empresas que a utilizem, visto indiciar a existência de um bom suporte, que se traduz numa constante atualização da ferramenta, disponibilização de recursos e expectativas de continuidade futura. Outro bom indicativo relaciona-se com o número e dimensão de empresas que utilizam a ferramenta para implementação do seu negócio. Apesar das aplicações serem desenvolvidas numa tecnologia atual, é importante considerar o seu futuro. Existirá sempre a necessidade de manutenção e por essa razão é de elevada importância o suporte da ferramenta no futuro. Seria difícil justificar o planeamento e desenvolvimento aplicacional, fazendo utilização de uma ferramenta suportada por empresas pequenas ou pouco reconhecidas, cujas atualizações sejam demoradas, com escassa documentação disponível e também sem perspetivas de futuro.

Ferramentas de desenvolvimento bem suportadas e com perspetivas de evolução futura tornam-se muito desejadas para se trabalhar. Assim, quando se planeia o desenvolvimento aplicacional, estas serão as ferramentas a ter em conta, fazendo com que o número de projetos utilizando tal ferramenta aumente. Por sua vez, o aumento do desejo de implementação com tais ferramentas proporciona uma maior procura de indivíduos com experiência nas mesmas, para integrarem os mais diversos projetos. Do ponto de vista individual, ou seja, do programador, é de todo o interesse a especialização numa ferramenta deste tipo, fazendo com que se tornem profissionais com um perfil ambicionado no mercado de trabalho, que se traduz numa elevada oferta profissional de qualidade.

A popularidade de ferramentas de desenvolvimento apresenta-se como um círculo vicioso, esquematizado na Figura 2. Quanto maior a popularidade, maior o seu tempo de vida, que se traduz numa maior competência e evolução, causando o aumento do desejo de trabalhar com a ferramenta e aumentando assim os recursos especializados na mesma, causando novamente o aumento da popularidade.

Figura 2- Ciclo da popularidade de uma ferramenta de desenvolvimento

3.2.2. Curva de aprendizagem

A curva de aprendizagem numa ferramenta de desenvolvimento explica-se pela correlação entre o tempo despendido para o seu domínio. A curva de aprendizagem relaciona-se com dois principais fatores: a dimensão da ferramenta de desenvolvimento e a quantidade e qualidade de informação disponível de apoio à mesma. Uma boa ferramenta de desenvolvimento requer que o programador despenda de um grande intervalo de tempo para proceder à leitura da documentação ou dos recursos alternativos, de forma a compreender e dominar a ferramenta. Este tempo pode ser encurtado, caso a qualidade de recursos seja boa. De forma generalizada, quanto maior a ferramenta maior a sua curva de aprendizagem. Para além dos recursos, é importante considerar as tecnologias requeridas para o desenvolvimento com a ferramenta. Neste caso, consideram-se as linguagens de programação. No caso de a ferramenta de desenvolvimento incluir uma ou mais linguagens de programação complexas, ou em que o programador não tenha experiência, irá observar-se um aumento do tempo para domínio da ferramenta. Mas, no mundo empresarial, tempo é dinheiro. E empresas de todo o tipo carecem cada vez mais da utilização de aplicações que proporcionem o aumento de produtividade dos seus colaboradores, da interação com clientes ou outros. Então, as aplicações são desenvolvidas de forma exclusiva, à imagem do negócio e necessidade da empresa. Deste modo, é necessário despender de tempo, dinheiro e recursos para o desenvolvimento destas aplicações por parte das empresas. É compreensível que as empresas ambicionem um desenvolvimento rápido e sólido, que apenas é permitido com a existência de programadores experientes. Caso não seja possível a disponibilização de recursos experientes, é importante que a curva de aprendizagem da tecnologia de desenvolvimento seja curta, o que torna recursos inexperientes em recursos experientes num curto período. A curva de aprendizagem também permite que as estimativas de tarefas dentro de um projeto sejam mais facilmente calculadas, facilitando assim a alocação de recursos por parte da empresa.

3.2.3. Documentação e conteúdos disponíveis

A existência de variadas ferramentas de desenvolvimento de FE faz com que existam diferentes metodologias e práticas de implementação. Cada ferramenta de desenvolvimento segue um formato próprio de implementação, que deve ser respeitado, de forma a tirar o maior partido da ferramenta de desenvolvimento. As metodologias e boas práticas de desenvolvimento encontram-se normalmente na documentação oficial das ferramentas. A existência de uma má documentação, ou simplesmente a sua inexistência, torna extremamente difícil a compreensão e implementação da ferramenta da forma mais correta, o que ainda conduz a um aumento da curva de aprendizagem, causando a desmotivação do programador e um difícil início com o desenvolvimento da tecnologia. Antes de escolher uma ferramenta, é de extrema importância a consulta da sua documentação oficial, de forma a perceber se os seus conteúdos são claros e se estão organizados. Não faz sentido optar por uma ferramenta de desenvolvimento cuja documentação seja pouco clara e de difícil consulta e compreensão. Além da documentação oficial, deve considerar-se a quantidade e qualidade de fontes de informação alternativa, de forma a conseguir desbloquear situações que não sejam consideradas pela documentação oficial.

3.2.4. Comunidade

A comunidade de uma ferramenta de desenvolvimento desempenha um importante papel no seu sucesso. A comunidade liga indivíduos que utilizam a mesma ferramenta de desenvolvimento. A existência de uma grande comunidade causa o aumento de quantidade e qualidade de documentação de funcionamento com a ferramenta. Estes benefícios são fruto de discussões ocorridas dentro da comunidade, relativos às metodologias e técnicas de implementação das ferramentas, o que causa o melhoramento e consolidação constante da tecnologia. A existência de documentação pouco clara, ou ausência de recursos alternativos, torna a existência de uma grande comunidade um fator ainda mais importante, sendo esta capaz de dar o maior e mais variado suporte aos programadores que desenvolvam com a ferramenta; permite também que assuntos pertinentes sejam discutidos e esclarecidos, facultando mais tarde esses resultados a outros programadores que passem pela mesma situação, poupando-lhes tempo.

3.2.5. Performance

As aplicações *web* apresentam o propósito de melhor servir o seu utilizador. Para tal, a interação das aplicações com o utilizador deve ser feita com a maior *performance* possível, causando uma excelente experiência de utilização. Uma aplicação lenta, ou que apresente muitos problemas, faz com que os seus utilizadores a abandonem rapidamente, procurando fontes alternativas. O abandono ocorre especialmente em plataformas de *e-commerce* ou de notícias, em que os utilizadores facilmente alternam entre páginas ou aplicações, encontrando aquilo que procuram de forma simples e rápida.

3.3. Conceitos a considerar em ferramentas de desenvolvimento

O desenvolvimento de aplicações *web*, a partir de ferramentas de desenvolvimento, envolve o conhecimento do programador relativamente aos conceitos mais básicos da sua implementação. Apresentam-se aqui alguns dos principais conceitos básicos a considerar.

3.3.1. Linguagens

A utilização de ferramentas de desenvolvimento requer a utilização de uma ou mais linguagens de programação. No caso de ferramentas de desenvolvimento de FE, as linguagens base são, neste momento, JS, HTML e CSS, podendo estas ser utilizadas a partir das suas extensões. As aplicações *web* requerem uma camada de UI, onde a informação é mostrada ao utilizador, recorrendo a HTML e CSS. A UI é utilizada não só para a apresentação de informação, mas também para a sua gestão a partir de ações do utilizador, sendo necessário o uso de JS.

As extensões destas linguagens de programação base podem apresentar metodologias de implementação diferentes e pouco comuns, o que eventualmente provocará uma maior dificuldade da sua implementação, causando assim um aumento da curva de aprendizagem para domínio da ferramenta.

3.3.2. Criação e execução aplicacional

Para o desenvolvimento de uma aplicação é necessária a criação de um projeto base, já com as configurações base definidas, que permitam ao programador o desenvolvimento imediato, ou quase imediato, das funcionalidades da aplicação. É de relevo considerar a *command line interface* (CLI), disponível em cada uma das ferramentas. A CLI representa um programa que permite aos programadores a execução de instruções específicas, através da escrita de comandos na consola do seu computador. Em ferramentas de desenvolvimento, a CLI tem como principal função a criação de um projeto base já com todas as configurações base definidas.

Após criação do projeto, é necessária uma fácil execução do mesmo, a partir de comandos inseridos na linha de comandos. Em desenvolvimento de FE, é muito comum a necessidade de atualização da parte visual imediata, de forma a ver os resultados obtidos pela implementação.

3.3.3. Estrutura aplicacional

A estrutura aplicacional exprime-se como a apresentação da aplicação a nível de estrutura de pastas e ficheiros. Uma boa estrutura define-se com a boa organização de pastas e ficheiros da solução da aplicação, permitindo a escalabilidade da mesma. A existência de uma boa estrutura torna clara a decisão de criação de novos componentes a facilita a navegação entre ficheiros e funcionalidades, causando uma ótima experiência de desenvolvimento a qualquer programador do projeto em questão.

As ferramentas de desenvolvimento podem apresentar estruturas aplicacionais rígidas pré-definidas, que pretendem o bom desenvolvimento aplicacional e a alta escalabilidade da mesma. Esta

pré-existência de estruturas faz com que não seja necessária a alocação de tempo no início do projeto para a definição da estrutura a utilizar, e faz com que qualquer programador se sinta à vontade com a estrutura do projeto, através de um bom estudo da documentação da estrutura definida pela ferramenta.

3.3.4. Componentes

Em ferramentas de desenvolvimento de FE orientadas a componentes, os componentes são responsáveis pela representação completa da aplicação. Entende-se então que haverá uma necessidade obrigatória de criação de diversos componentes, cada um com um propósito específico. Os componentes são normalmente agrupados por funcionalidades, sendo assim utilizados dentro de outros componentes maiores. A utilização de componentes embutidos dentro de outros componentes requer que a interação entre estes seja possível e implementada de forma simples, de modo a não tomar muito tempo ao programador. Sendo cada componente responsável por uma funcionalidade específica, é necessária a gestão do seu estado, podendo este ser feito de diversas formas, consoante a ferramenta utilizada.

3.3.5. Funcionalidades oferecidas

Qualquer aplicação *web* requer funcionalidades específicas para a sua implementação. Ao escolher uma ferramenta de desenvolvimento, é importante ter em conta as funcionalidades que esta permite ,, desenvolver. As funcionalidades mais usuais do desenvolvimento de aplicações *web* são as de comunicação HTTP, *routing*, construção e validação de formulários.

Uma ferramenta de desenvolvimento pode apresentar todas ou só apenas parte destas funcionalidades. A existência de tais funcionalidades numa ferramenta de desenvolvimento permite a sua fácil utilização, fazendo uso da respetiva documentação. Porém, nem sempre as funcionalidades podem estar disponíveis dentro de uma ferramenta de desenvolvimento, o que pode obrigar à importação de uma biblioteca externa, que apresente tal funcionalidade, ou até mesmo a criação de raiz da funcionalidade, algo complexo e demorado. Funcionalidades pré-existentes podem apresentar ainda a utilização de tecnologias pouco atuais, ou até mesmo formas de implementação que prejudiquem a *performance* da aplicação. A pré-existência de funcionalidades embutidas permite o imediato desenvolvimento aplicacional, enquanto a não existência provoca a procura das bibliotecas mais adequadas ou até mesmo o desenvolvimento da funcionalidade.

3.4. Desenvolvimento aplicacional de análise e comparação

A comparação do ponto de vista do desenvolvimento ocupa a última tarefa prática de comparação entre as ferramentas de desenvolvimento. De forma a possibilitar esta comparação, é necessário fazer o planeamento de uma pequena aplicação onde se possam colocar as ferramentas em prática. Pretende-se que a aplicação não seja muito complexa ou de longo desenvolvimento. A aplicação proposta deve ser simples e pequena o suficiente, de forma a permitir o seu desenvolvimento com ferramentas de

desenvolvimento totalmente novas para o programador, sem que lhe seja tomado demasiado tempo. Esta aplicação deve conter os principais conceitos utilizados em *websites* comuns, uma vez que serão estes os principais a ser desenvolvidos em qualquer aplicação deste tipo. Tornar-se-á então possível a aplicação da ferramenta em diferentes casos práticos, e assim fazer-se uma comparação entre eles de diversos fatores , recolhendo informação sobre quais os requisitos da sua implementação em casos reais, juntamente com a deteção de facilidades e dificuldades de utilização.

3.5. Síntese da metodologia

A metodologia de avaliação e comparação de ferramentas proposta é constituída por três fases.

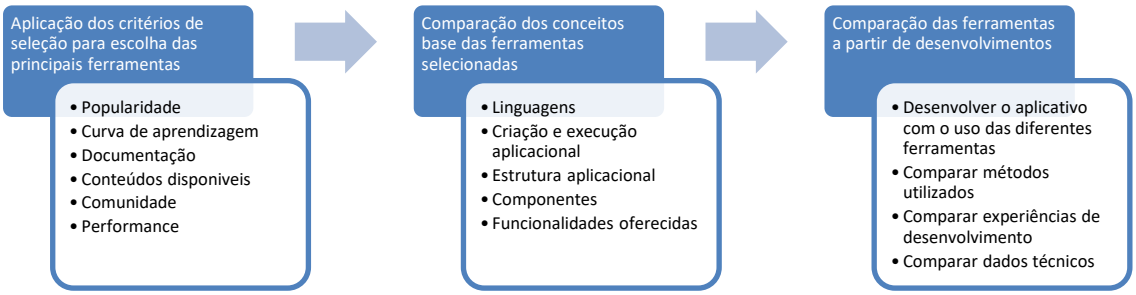


Figura 3- Esquema de aplicação da metodologia proposta

A primeira fase trata de aplicar os critérios de seleção de ferramentas a cada uma das ferramentas do universo escolhido, com vista a reduzi-lo a apenas duas ferramentas. Os critérios do primeiro passo são a popularidade, a curva de aprendizagem, a documentação e conteúdos disponíveis, a comunidade e a *performance*.

A segunda fase permite a análise e comparação de conceitos base de desenvolvimento para *websites*, em cada uma das ferramentas. Os conceitos são as linguagens de programação usadas, a criação e a execução aplicacional, a estrutura aplicacional, os componentes, e as funcionalidades oferecidas.

A terceira e última fase envolve o desenvolvimento de uma pequena aplicação *web*, previamente definida, com as funcionalidades normais de uma aplicação *web*. O desenvolvimento é feito em separado, com ambas as ferramentas, permitindo a sua comparação do ponto de vista do desenvolvimento.

CAPÍTULO 4

Avaliação da metodologia: casos de estudo

4.1. Aplicação dos critérios de seleção e filtragem das ferramentas

A aplicação dos critérios eleitos para a escolha das melhores ferramentas de desenvolvimento de FE é feita tendo por base um pequeno universo de ferramentas. A sua obtenção pode ser feita de diversas formas, como por exemplo, as mais utilizadas no mercado atual, as mais promissoras ou até mesmo aquelas que são do interesse do indivíduo que deseja fazer o estudo. O universo eleito contempla quatro ferramentas de desenvolvimento, Angular, React, Vue e Ember, às quais se aplicarão, de seguida, os critérios propostos no capítulo anterior.

4.1.1. Popularidade

Um bom e fiel indicativo da popularidade de uma ferramenta é transmitido pelo número de *downloads* da própria ferramenta de desenvolvimento, número este que pode ser obtido através da plataforma NPM.

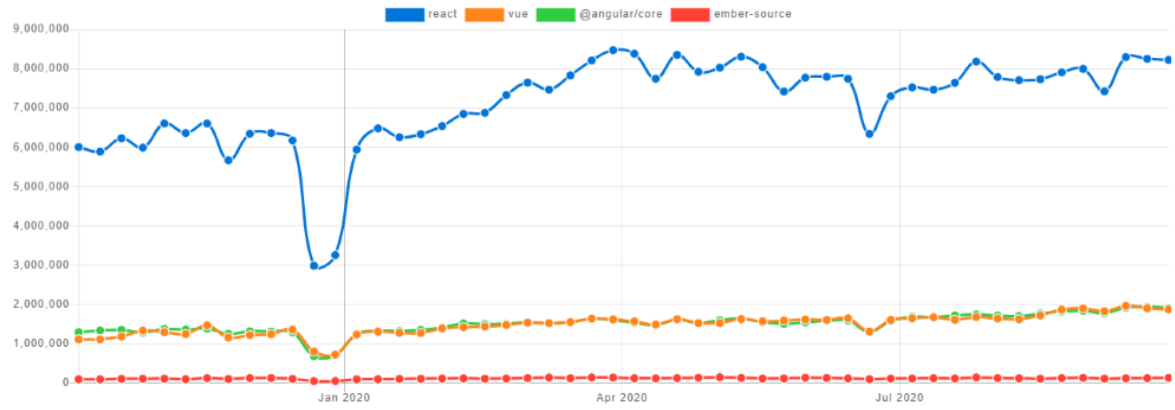


Figura 4- Número de downloads das ferramentas de desenvolvimento entre Outubro de 2019 e Outubro de 2020 [24]

Outro bom indicativo da popularidade de uma ferramenta pode ser efetuado a partir da análise do repositório das ferramentas existente no GitHub, que fornece alguns parâmetros de análise como as *Stars* e os *Forks*. Os utilizadores do GitHub utilizam as *Stars* para demonstrar o seu agrado pelo repositório, tendo assim a possibilidade de observar o seu progresso futuro. Os *Forks* representam o número de copias feitas ao repositório, de forma a que seja possível fazer o seu teste e até integrá-lo num projeto, se assim desejado.

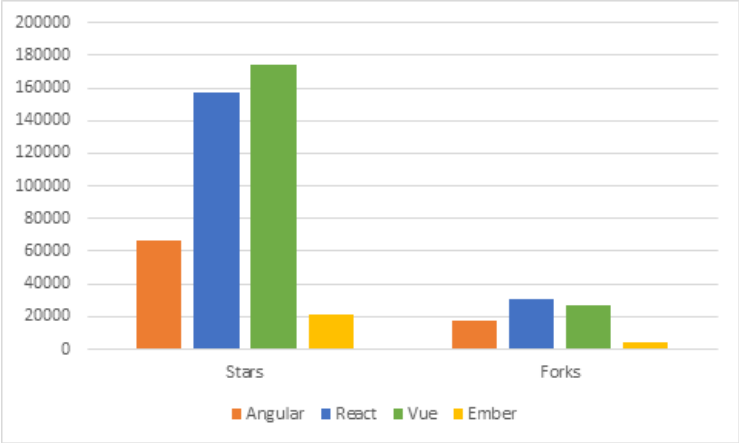


Figura 5- Comparação do número de Stars e Forks no GitHub entre React, Angular, Vue e Ember em 2020 [25][26][27][28]

O GitHub mostra também o número de dependências de cada ferramenta de desenvolvimento, podendo estas dependências ser repositórios ou pacotes, o que transmite a dimensão de utilização da ferramenta.

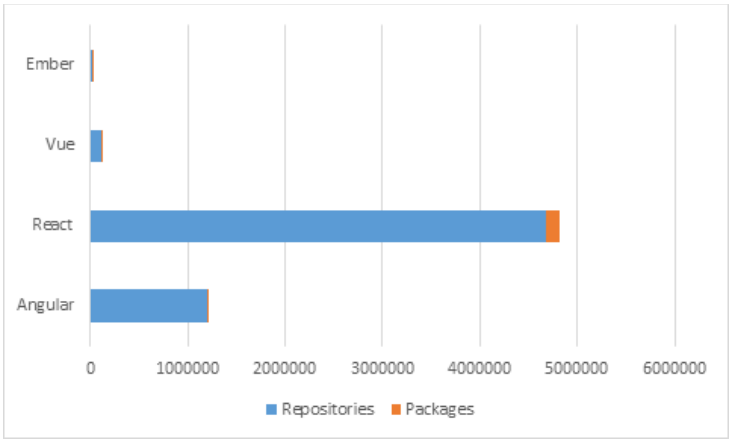


Figura 6- Comparação do número de dependências de repositórios e pacotes entre React, Angular, Vue e Ember em Outubro de 2020 [40][41][42][43]

StackOverflow, o maior site de perguntas e respostas exclusivamente para programadores, apresenta a opinião de um universo de programadores referentes à diferentes ferramentas de desenvolvimento, sendo possível a consulta sobre quais as ferramentas mais apreciadas e pretendidas.

Figura 7- Top 3 da percentagem de programadores que desenvolvem com a ferramenta de desenvolvimento web e expressam interesse de continuidade de desenvolvimento com a mesma em 2020, gráfico completo no anexo C [29]

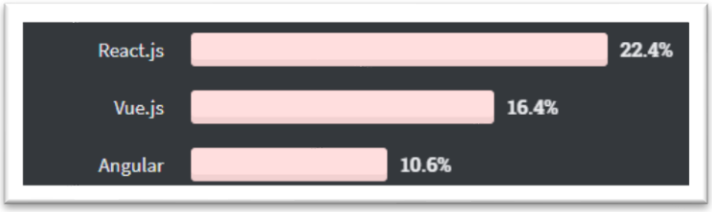


Figura 8- Top 3 da percentagem de programadores que não desenvolvem com a ferramenta de desenvolvimento web, mas expressam interesse de desenvolvimento com a mesma em 2020, gráfico completo no anexo D [30]

A oferta de empregabilidade representa também a popularidade de uma determinada ferramenta de desenvolvimento. Quanto maior a oferta de emprego para trabalhar com uma determinada ferramenta, maior é o número de projetos existentes com a mesma e, portanto, mais popular a ferramenta é.

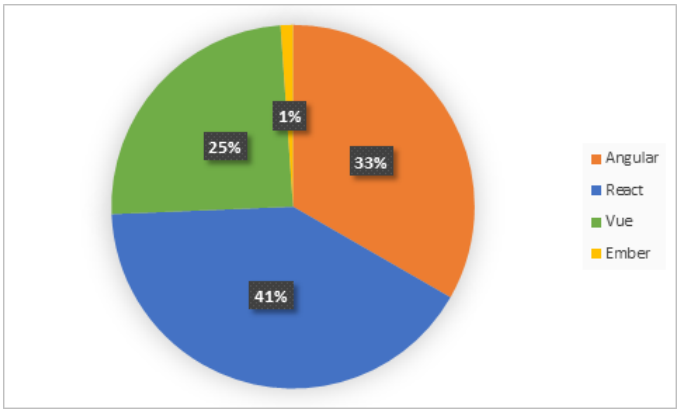


Figura 9-Gráfico de percentagem de oferta de empregabilidade entre Angular, React, Vue e Ember [31][32][33][34]

Angular	122.937
React	151.530
Vue	90.622
Ember	*4.073*

Tabela 1- Número de ofertas de empregabilidade no Linkedin em Outubro de 2020 para Angular, React, Vue e Ember a nível mundial [31][32][33][34]

Finalmente, para as ferramentas com maior nível de popularidade, pode-se proceder à análise da sua origem, tempo e empresas que a utilizam na implementação do seu negócio, que, como explicado anteriormente, influenciam a popularidade da ferramenta.

	Angular	React	Vue	Ember
Criador	Google	Facebook	Evan You	Yehuda Katz
Data de lançamento	2010	2013	2014	2011
Utilizado por	Google, Wix	Facebook, Uber	Alibaba, GitLab	LinkedIn, Yahoo!

Tabela 2- Informação sobre as ferramentas de desenvolvimento

Nas quatro ferramentas de desenvolvimento eleitas para aplicação dos critérios, verifica-se uma grande discrepância em relação à ferramenta Ember, em qualquer um dos pontos de popularidade analisada. Assim, os próximos critérios são aplicados apenas às outras três ferramentas.

4.1.2. Curva de aprendizagem

Angular apresenta-se como uma ferramenta de desenvolvimento MVC - *Model*, *View* e *Control*, o que faz com que existam mais funcionalidades disponíveis e, portanto, uma maior necessidade de tempo para a dominar. Angular utiliza TS, um subconjunto de JS, que permite a escrita de código JS, orientado a objetos. TS permite a utilização de tipos, que ajuda a prevenir os mais variados erros ocorridos durante o desenvolvimento de uma aplicação, como por exemplo, a passagem de um argumento inválido numa função. De forma geral, o uso de TS oferece a consolidação do desenvolvimento, porém a sua aprendizagem requer algum esforço e tempo. TS, após compilado, é convertido em JS puro, tornando possível a utilização de JS para o desenvolvimento, porém isso não oferece a mesma robustez de TS.

Angular apresenta inúmeras funcionalidades poderosas, com um modo de implementação útil para a construção de aplicações grandes e complexas. O facto de serem funcionalidades únicas requer a sua aprendizagem.

React apresenta-se como uma ferramenta de desenvolvimento da camada *View*. A apresentação apenas da camada de visualização cria a necessidade de importação de outras *bibliotecas*, de forma a permitir a implementação do MVC. React faz a introdução de JSX, que se resume na combinação de HTML e JS. A implementação de JSX é estranha e complicada no início da implementação, mas tende rapidamente a ficar mais simples. JSX torna tudo em JS, o que faz com que a manipulação do HTML seja feita de forma diferente ao que os programadores estão acostumados, podendo levar mais tempo a dominar.

Vue também se apresenta como uma ferramenta de desenvolvimento da *View*. Esta é considerada uma das ferramentas mais fáceis de aprender por parte dos programadores. Os seus conceitos encontram-se muito próximos de HTML e JS básicos, o que oferece a sensação de familiaridade ao programador. O início do desenvolvimento com esta ferramenta é muito simples, sendo apenas necessária a sua importação num ficheiro HTML. Vue é considerada uma ferramenta de simples implementação em aplicações pequenas, pelo que, com o aumento da dimensão aplicacional, torna-se

necessário o funcionamento com ficheiros específicos, o que requer uma configuração do projeto e padrões mais complexos.

4.1.3. Documentação e conteúdos disponíveis

Este ponto é de difícil análise comparação, a partir de dados técnicos. A partir da análise feita, compreendi que todas as ferramentas elegidas apresentam uma boa documentação oficial, que permite um bom suporte na criação e desenvolvimento de uma aplicação. Relativamente aos conteúdos disponíveis, considero que Angular e React são os que apresentam maior taxa de conteúdos alternativos, visto também serem os mais populares e apresentarem as maiores comunidades.

4.1.4. Comunidade

A comunidade de uma ferramenta de desenvolvimento pode ser medida na plataforma GitHub, a partir da consulta de *Watchers*, *Stars*, *Forks* e *Contributors*. *Stars* e *Forks*, como já explicado anteriormente, além de serem indicativos da popularidade de uma ferramenta, também podem ser considerados para medir a dimensão da sua comunidade. *Watchers* representa o número de utilizadores da plataforma que pediram para serem notificados relativamente à atividade do repositório da ferramenta.

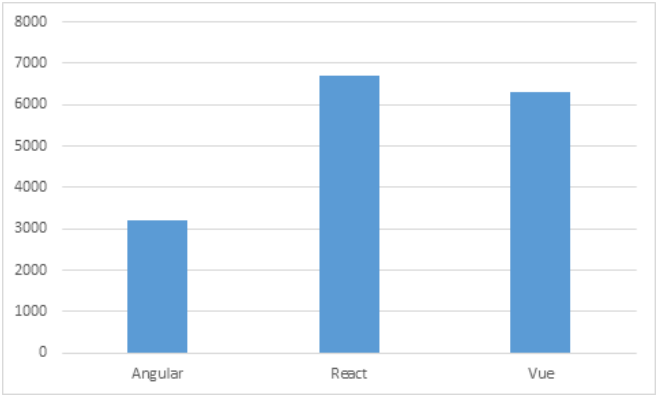


Figura 10- Gráfico de comparação do número de Watchers do GitHub entre Angular, React e Vue em Outubro de 2020 [25][26][27]

Contributors, tal como o nome indica, disponibiliza o número de utilizadores que já procederam a uma ou mais contribuições no repositório da ferramenta de desenvolvimento.

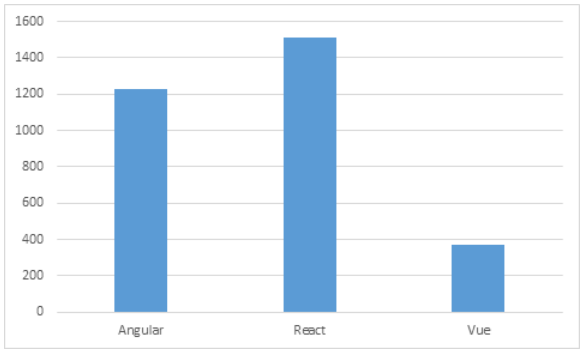


Figura 11- Gráfico de comparação do número de contribuidores de Angular, React e Vue em Outubro de 2020 [35][36][37]

Para além destes indicadores, ainda é possível consultar o número de utilizadores do repositório da ferramenta, o que faz deles membros da comunidade da ferramenta. Esta funcionalidade do GitHub ainda só está disponível para React e Vue; por esse motivo, Angular não é considerado na figura 12.

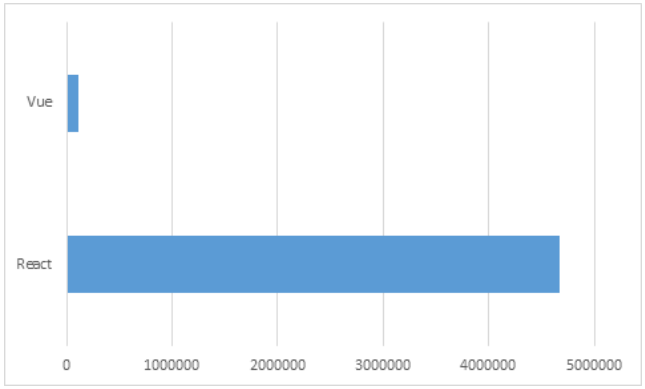


Figura 12- Comparação do número de utilizadores que utiliza a ferramenta entre React e Vue em Outubro de 2020 [25][27]

O StackOverflow Trends oferece a informação da percentagem de perguntas feitas relativas a uma determinada ferramenta. A quantidade de perguntas feitas indica a dimensão e o quão ativa é a sua comunidade.

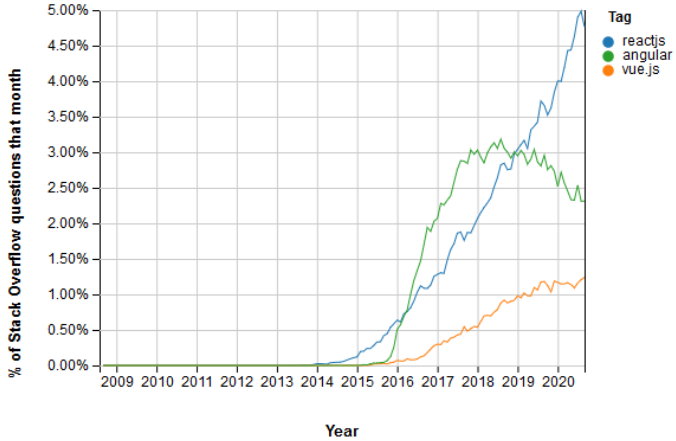


Figura 13- Gráfico de comparação da percentagem de perguntas feitas no StackOverflow entre Angular, React e Vue [38]

4.1.5. Performance

A comparação técnica entre ferramentas de desenvolvimento é feita considerando o tempo de execução e de resposta da aplicação, métricas de inicialização e a quantidade de memória alocada para execução aplicacional. A comparação técnica entre Angular, React e Vue pode ser feita a partir da aplicação *js-framework-benchmark* [19]. É uma aplicação que permite o estudo técnico de diferentes ferramentas de desenvolvimento de JS. Este estudo é feito com base numa tabela de dados aleatórios, em que as manipulações de renderização são medidas com base no tempo, iniciação e memória utilizada.

Para consulta da *performance* das ferramentas são utilizadas três secções. A primeira expõe a duração em milissegundos de determinadas ações na tabela, apresentada no Anexo E. A segunda expõe um conjunto de métricas de inicialização em cada uma das ferramentas, apresentado no Anexo F. A terceira apresenta a memória de cada ferramenta alocada em *Megabytes*, para algumas das ações efetuadas na tabela, e está disponível no Anexo G.

Apesar de algumas diferenças nos resultados, estes não suscitam qualquer preocupação. Todos os tempos das ações efetuadas são reduzidos e praticamente indistinguíveis através de observação pelo utilizador. Os valores de memória utilizados são também muito reduzidos, o que não provoca qualquer problema de *performance* numa aplicação.

4.1.6. Conclusões

A tabela 3 representa a comparação entre as diferentes ferramentas, de acordo com os critérios propostos.

	Angular	React	Vue
Popularidade	Segunda mais popular	Mais popular	Menos popular
Curva de aprendizagem	Maior	Média	Menor
Documentação e conteúdos disponíveis	Muito bons e idênticos entre ferramentas		
Comunidade	Segunda maior comunidade	Maior comunidade	Menor comunidade
Performance	Muito boa e idêntica entre ferramentas		

Tabela 3- Comparação dos critérios de seleção entre as ferramentas Angular, React e Vue

4.2. Análise e comparação de conceitos das ferramentas

4.2.1. Linguagens

Uma ferramenta de desenvolvimento requer uma ou mais linguagens de programação. No caso da maioria das ferramentas de desenvolvimento de FE, e no caso específico de Angular e React, existem pelo menos três linguagens de programação requeridas. A linguagem HTML que define a estrutura do componente a apresentar no ecrã, a linguagem CSS, que estiliza a estrutura HTML, e finalmente a linguagem JS, usada para lógica de negócio e para alterações na camada HTML do componente.

Angular utiliza TypeScript (TS), HTML e CSS, para a construção dos seus componentes. A implementação destas três linguagens é feita em ficheiros separados, que depois comunicam e funcionam entre si. TS é uma extensão de JS, que permite a utilização de tipos, o que reduz o tempo de programação, detetando erros e oferecendo soluções. TS, depois de compilado, é convertido para JS normal, possibilitando a escrita de componentes com JS.

React utiliza as mesmas três linguagens, porém existem alguma diferenças. React utiliza JSX, que é uma extensão de sintaxe para JS, cuja utilização permite a escrita de JS e de HTML no mesmo ficheiro. Assim, para a criação de um componente, são necessários apenas dois ficheiros, um ficheiro JSX e outro ficheiro CSS, que precisa de ser importado no ficheiro de JSX, para que a sua estrutura HTML o possa utilizar. JSX, depois de compilado, é convertido para HTML e JS, que correm em qualquer *browser*. Atualmente já é possível a utilização de TS em React, porém, são necessárias configurações adicionais na criação de um projeto.

4.2.2. Criação e execução aplicacional

4.2.2.1. Angular

Para a criação de aplicações em Angular é disponibilizada uma interface de linha de comandos, Angular CLI. Esta é utilizada para inicializar, desenvolver, escalar e manter aplicações Angular, diretamente a partir da consola de comandos. Esta interface Angular CLI tem de ser instalada a partir da linha de comandos, com o comando *npm install -g @angular/cli*. Após instalação do Angular CLI, pode proceder-se à criação do projeto a partir da linha de comando, *ng new my-app*. Ao correr tal comando, o programador é questionado sobre algumas configurações para a criação do seu projeto. A primeira questiona o programador se pretende incluir *routing* na sua aplicação e a segunda questiona sobre o formato de CSS que deseja utilizar.

O comando *ng new my-app* cria a estrutura inicial de um projeto Angular dentro da pasta chamada *my-app*, especificada no comando de criação do projeto. Dentro desta pasta, encontram-se também todas as dependências necessárias instaladas para a execução da aplicação. Angular ainda oferece a criação de um novo projeto em modo estrito, que ajuda o programador na melhoria da escrita do seu código e fluxo, adicionando --*strict* à linha de comando de criação do projeto. Apresenta-se a estrutura criada a partir da linha de comando *ng new my-app* no Anexo J.

De forma a executar a aplicação criada, é necessário entrar dentro da pasta criada, a pasta *my-app*, e executar o comando *ng serve*. A aplicação é então executada e lançada localmente na página *http://localhost:4200/*. Para evitar ao programador ter de abrir uma nova página *web*, pode adicionar-se o argumento *--open* ao comando, que irá abrir a página automaticamente no porto 4200, depois da aplicação já ter sido compilada, como se pode verificar no Anexo K.

4.2.2.2. React

A *Toolchain* de React para criação de aplicações *single page* denomina-se *Create React App*. Este é um ambiente confortável e propício ao desenvolvimento de uma aplicação *single-page*. É responsável por fazer o *set up* do ambiente de desenvolvimento para que o programador possa usufruir das funcionalidades JS mais recentes. Disponibiliza também uma boa experiência de desenvolvimento e otimiza a aplicação para produção. A criação de um projeto com *Create React App* é feita a partir da linha de comandos com o comando *npx create-react-app my-app*.

Após a execução do comando, é criada uma pasta na diretoria onde o comando foi executado. Neste caso, a pasta tem o nome *my-app*, que contém dentro todos os ficheiros e pastas necessárias à existência e execução de uma aplicação React. O Anexo L apresenta a estrutura criada a partir da linha de comando *npx create-react-app my-app*. A execução aplicacional é feita com o comando *yarn start*, dentro da pasta *my-app* criada. A aplicação irá arrancar na consola e quando a compilação inicial estiver concluída, uma nova página *web* será aberta, com a aplicação a correr localmente no *url* http://localhost:3000, como ilustrado no Anexo M.

4.2.3. Estrutura aplicacional

4.2.3.1. Angular

Angular apresenta-se como uma *framework* e contém na sua documentação a estrutura aplicacional a ser seguida no desenvolvimento de qualquer aplicação que utilize esta ferramenta.

Angular opta por apresentar os componentes da aplicação em duas pastas distintas. Os componentes podem pertencer a uma funcionalidade da aplicação, ou simplesmente ser um componente genérico, que pode ser utilizado em qualquer lado. Este componente genérico é partilhado e por esse motivo não pertence a nenhuma funcionalidade da aplicação, sendo assim colocado na pasta */src/app/shared*. Os outros componentes, que não são partilhados, fazem parte de uma funcionalidade específica da aplicação e são colocados numa pasta com o nome da funcionalidade, dentro da pasta */src/app/features*. Dentro desta pasta, os componentes são divididos em páginas ou componentes.

Além dos componentes, a aplicação precisa de outros ficheiros partilhados, que podem ser de diferentes tipos e apresentar diferentes funcionalidades, tais como serviços, modelos e utilidades transversais, entre outros. Angular sugere a sua localização na pasta */src/app/core*.

```
/src
    /app
        /core
            /services
                /TestServiceX
            /utils
                /TestUtilsX
        /features
            /TestFeatureX
                /pages
                    /TestPageX
                /componentes
                    /TestComponentX
        /shared
            /TestSharedX
```

Figura 14- Estrutura aplicacional sugerida por Angular

4.2.3.2. React

React, sendo uma *biblioteca* e não uma *framework*, não apresenta uma estrutura aplicacional sugerida na sua documentação. A estrutura aplicacional deve ser acordada entre os programadores do projeto, tendo em conta o produto a ser desenvolvido. É de elevada importância que a estrutura seja organizada e escalável de forma a facilitar o desenvolvimento de novas funcionalidades e alterações no código, se necessário. O facto de a estrutura ser aberta, facilita aos programadores a escolha da sua própria estrutura adaptada à aplicação, ou até mesmo a cópia de uma estrutura já existente de uma *framework* de desenvolvimento. Ainda assim, é sugerida alguma atenção com a organização dos ficheiros, no que diz respeito a componentes e não componentes, colocando-os em pastas separadas.

```
/src
    /components
        /features
            /TestFeatureX
        /pages
            /TestPageX
        /shared
    /shared
        /api
            /TestApiXTestApi2
        /utils
            /TestUtilX
```

Figura 15- Exemplo de estrutura aplicacional em React

4.2.4. Componentes

São apenas abordados os conceitos mais básicos dos componentes em Angular e React, visto existirem duas aplicações desenvolvidas, com cada uma das ferramentas, onde se poderá fazer a comparação de outros conceitos de forma prática.

4.2.4.1. Angular

Angular opta pela definição de um componente usando três ficheiros distintos. Esses ficheiros são os ficheiros de TS, HTML e CSS. O ficheiro de TS contém a lógica e todas as variáveis que o componente precisa para desempenhar as suas funções; o ficheiro de HTML contém a estrutura da UI desejada para

representar o componente, alimentando-se se necessário das variáveis do ficheiro TS; finalmente, o ficheiro de CSS apresenta todas as classes CSS que são utilizadas pelo ficheiro HTML do componente, de forma a estilizar a sua aparência. O facto de um componente ser definido a partir de três ficheiros distintos exige que dois destes sejam especificados no ficheiro TS, para que o componente identifique qual o seu ficheiro de HTML e de CSS.

Ficheiro TS	Ficheiro HTML

1 import { Component } from '@angular/core';
2
3 @Component({
4 selector: 'app-root',
5 templateUrl: './app.component.html',
6 styleUrls: ['./app.component.css'],
7 })
8 export class AppComponent {
9 message = 'Hello world!';
10 }
	1 {{ message }}

Tabela 4- Exemplo de um componente App na aplicação Angular

Para especificar um componente é necessário fazer a sua importação, como descrito no extrato de código acima. Ao inicializar o componente é necessário especificar três campos, o *selector*, o *templateUrl* e os *stylesUrls*. O *selector* é a *tag* HTML utilizada para inserir o componente numa página; o *templateUrl* é a diretoria do ficheiro HTML com a estrutura do componente; por fim, o *styleUrls* é um *array* de diretorias de ficheiros responsáveis por estilizar o HTML do componente. O *title* é uma variável criada que pode tanto ser utilizada para a lógica interna do componente, como ser exibida na UI do componente. A exibição da variável é feita a partir de interpolação, no ficheiro HTML *app.component.html*, como observado na linha 1 da tabela 4 no ficheiro HTML.

Angular oferece uma funcionalidade para ajudar na criação de novos componentes, uma vez que a criação de um só componente exige a criação de três novos ficheiros, como referido anteriormente, mais um ficheiro de testes. O comando *ng generate componente component-name* cria então uma pasta dentro da diretoria onde o comando é executado. Este comando cria os três ficheiros suprarreferidos, com o mesmo nome *component-name*, com extensões diferentes, TS, HTML e CSS, sendo que os ficheiros HTML e CSS já se encontram referenciados no ficheiro TS, estando assim interligados, como verificado nas linhas 5 e 6 da tabela 4, no ficheiro TS.

4.2.4.2. React

React opta pela definição de um componente, utilizando apenas dois ficheiros, um ficheiro JSX e outro de CSS. O ficheiro JSX é o que agrega o JS e o HTML, sem que seja necessário a existência de um ficheiro adicional. Desta forma, o ficheiro JSX apresenta a lógica do componente e também a sua

estrutura visual. O ficheiro CSS é utilizado para estilizar o HTML contido no ficheiro JSX, porém é necessário fazer a sua importação no ficheiro JSX, como se mostra abaixo, na linha 2.

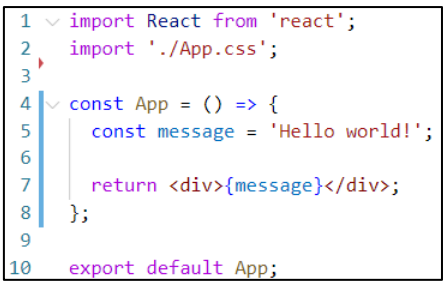


Figura 16- Exemplo de um componente App na aplicação React

Os componentes em React comportam-se como funções normais, fazendo com que o seu retorno inclua os elementos a serem mostrados, como verificado na linha 7 da figura 16. Na linha 5, é declarada uma variável *title*, que tanto pode ser utilizada para a lógica do componente, como para demonstração na estrutura HTML do componente, como exemplifica a linha 7 da figura 16. React não apresenta uma forma simples de criação de componentes, tendo esta de ser feita de forma manual, criando primeiro uma pasta com o nome do componente, seguido da criação de um ficheiro JSX e um ficheiro CSS, com o mesmo nome do componente, mas com extensões diferentes. Finalmente, de forma a conseguir utilizar o componente na aplicação, é necessário fazer a sua exportação, como exemplificado na linha 10 da figura 16.

4.2.5. Funcionalidades oferecidas

As funcionalidades oferecidas são um dos pontos de grande disparidade entre Angular e React, ou seja, entre *frameworks* e *bibliotecas de desenvolvimento de* FE, respetivamente. Angular apresenta-se como uma ferramenta totalmente equipada de funcionalidades próprias, que permitem a implementação completa de uma aplicação, sem que sejam necessárias *bibliotecas* extras. React, por sua vez, apenas apresenta as funcionalidades mais básicas para o desenvolvimento de FE, o que torna necessário a criação de novas funcionalidades e a utilização de *bibliotecas* externas, para a construção completa de uma aplicação.

Em Angular, as funcionalidades pré-existentes podem não apresentar a melhor *performance* comparativamente a *bibliotecas* externas. O facto de já existirem funcionalidades embutidas em Angular permite que o programador não despenda tempo a encontrar e comparar *bibliotecas* que apresentem as funcionalidades que pretende utilizar, partindo diretamente para o desenvolvimento. React obriga o programador a procurar *bibliotecas* extras, cuja *performance* pode ser de elevada qualidade. Além do uso de *bibliotecas* externas, as funcionalidades podem ser desenvolvidas de raiz, algo demorado e complexo, mas que pode ser benéfico, visto ser desenvolvido para um uso específico.

Sendo React uma *biblioteca* que apresenta apenas as funcionalidades básicas de desenvolvimento de FE, o seu tamanho é muito reduzido em comparação a Angular, que é uma *framework* que apresenta um grande leque de funcionalidades já embutidas. Com a necessidade de utilização de funcionalidades, novas *bibliotecas* podem ser adicionadas ao projeto de React, fazendo com que, no fim, apenas existam as *bibliotecas* utilizadas e nada mais. Já em Angular, o final do projeto poderá apresentar inúmeras funcionalidades internas, que não são utilizadas nem uma única vez, ocupando espaço desnecessário.

4.3. Desenvolvimento aplicacional de análise e comparação

4.3.1. Descrição aplicacional

O funcionamento da aplicação é demonstrado em vídeo, com o uso da aplicação React [44], que, a nível de visualização, é igual à aplicação Angular. As duas aplicações estão disponíveis para consulta e *download* no GitHub, no repositório de Angular [45] e de React [46].

A aplicação apresenta autenticação, mostrando uma preocupação com a segurança. A aplicação desenvolvida representa uma plataforma de gestão interna de equipas. Esta gestão de equipas traduz-se na visualização dos membros constituintes da equipa. Através da visualização geral dos membros da equipa, a aplicação permite a seleção de um membro, retornando a sua informação. A informação do membro escolhido pode ser alterada, caso desejado. Além da consulta e atualização de dados de membros individuais, é permitida a eliminação de membros. A criação de novos membros também é uma funcionalidade disponível, a partir da inserção da sua informação. Tal como em todas, ou em quase todas as equipas tecnológicas, existem diferentes áreas técnicas; como tal, cada membro da equipa tem associado ao seu perfil a sua área técnica. As áreas técnicas estão intimamente relacionadas com as tarefas que são planeadas para uma equipa, isto porque cada tarefa é referente a uma área. A apresentação das tarefas é uma outra funcionalidade da aplicação, onde a equipa pode consultar as tarefas pendentes e o seu estado mais atual. É possível a adição e edição de tarefas, porém tal só é possível caso a tarefa apresente a mesma área técnica que o membro utilizador da aplicação.

4.3.2. Fluxo aplicacional

A página de *login* é apresentada ao utilizador, pedindo as suas credenciais. O uso correto de credenciais, redirecionará o utilizador para o interior da aplicação, onde poderá consultar as páginas *members* e *tasks.* Uma vez autenticado, poderá proceder ao cessar de sessão, viajando novamente para a página de *login*.

A página inicial é a página *members* onde se apresentam listados todos os utilizadores que fazem parte da equipa do utilizador autenticado, incluindo o próprio. Aqui o utilizador pode selecionar e desselecionar membros específicos. A seleção de um membro mostra do lado direito da página os seus detalhes, algo puramente informativo. Nesse mesmo lado direito da página, o utilizador é capaz de atualizar membros ou até mesmo de os excluir da equipa. A opção de edição de membros redireciona o utilizador para a página de edição de membros, sendo que nessa mesma página serão apresentados

novamente os dados do utilizador para se fazer a alteração em cima desses mesmos dados. A partir da página de edição, o utilizador pode então proceder à atualização do membro, sendo de seguida redirecionado para a página *members*, onde a atualização já se encontra disponível. O utilizador, a partir da página de edição, tem também a opção de cancelar o pedido de edição, não causando qualquer atualização ao membro, sendo de seguida redirecionado para a página *members*. Na página *members*, o utilizador é capaz de efetuar o registo de um novo membro na equipa. Para tal, o utilizador é redirecionado para a página de criação de membros, onde terá de preencher os respetivos detalhes. Depois de terminado o preenchimento dos dados, o utilizador pode dar a ordem de criação do novo membro, sendo de seguida redirecionado para a página *members*, onde encontrará já listado o membro criado. Identicamente à página de edição de membros, na página de criação de membros também está disponível a opção de cancelamento da criação, redirecionando sempre o utilizador para a página *members*, sem causar qualquer alteração na listagem dos membros da equipa. Outra funcionalidade existe na página *tasks, que* apresenta todas as tarefas da equipa ainda não concluídas. As tarefas são apresentadas numa tabela inicialmente ordenada pela data limite de conclusão da tarefa, podendo ser ordenadas por qualquer coluna da tabela. É possível a criação de uma nova tarefa a partir do preenchimento dos seus detalhes, fazendo com que a tabela com todas as tarefas seja atualizada após o sucesso da adição. A seleção individual de uma tarefa permite a visualização dos seus detalhes e a partir deles a tarefa pode sofrer alterações relativas ao seu estado.

4.3.3. Estrutura

A aplicação desenvolvida apresenta três funcionalidades principais, *login, members* e *tasks*. A funcionalidade *login* permite ao utilizador autenticar-se na aplicação e aceder ao seu interior, onde estão apresentadas as funcionalidades *members* e *tasks*. A funcionalidade *members* inclui tudo o que está relacionado com a parte de visualização de membros da equipa, criação, edição e remoção de membros. A funcionalidade *tasks* inclui tudo o que está relacionado com a parte de visualização, criação e edição de tarefas. O uso destas funcionalidades só é permitido com o uso de outros ficheiros, para além dos componentes, tais como serviços, utilidades e outros.

Angular	React
/src /app /core /services /auth.service /member.service /task.service /models /member /task /utils /features /auth /pages /login-page /members /pages /members-page /member-add-page /member-edit-page /components /member-displayer /member-form /member-list /tasks /pages /task-page /components /task-form /shared /error-message /loading /table	/src /components /features /members /MemberDisplayer /MemberForm /MemberList /tasks /TaskForm /pages /LoginPage /MembersPage /MemberAddPage /MemberEditPage /TasksPage /shared /ErrorMessage /Loading /Table /shared /api /authApi /memberApi /taskApi /utils

Tabela 5- Comparação de estruturas da aplicação desenvolvida entre Angular e React

Em Angular, as funcionalidades são, então, divididas em duas secções, a secção de páginas e a secção de funcionalidades. As páginas são componentes responsáveis por apresentar as funcionalidades. No caso da funcionalidade *members*, a pasta *pages* irá conter os componentes das páginas de visualização, edição e criação de membros, enquanto a pasta *components* conterá todos os pequenos componentes que serão mostrados nas páginas, como por exemplo, o formulário de membro, a lista de membros e o painel de detalhes do membro. Todos os ficheiros que não representem componentes devem ser colocados no interior da pasta *core*.

React não sugere uma estrutura, sendo esta definida pelo programador. A estrutura tem de ser organizada e escalável, para facilitar o desenvolvimento de novas funcionalidades e alterações evolutivas ao código.

Na estrutura proposta em React, os componentes encontram-se todos dentro da mesma pasta */src/components* e são então divididos em páginas, funcionalidades e em componentes partilhados. Na pasta *shared*, apresentam-se todos os componentes partilhados da aplicação, que podem ser utilizados em qualquer contexto. A pasta *pages* apresenta os componentes das páginas da aplicação. A pasta *features* inclui no seu interior pastas com o nome de cada funcionalidade existente, neste caso *members* e *tasks*, e dentro de cada uma destas pastas estarão contidos todos os componentes

responsáveis pelo funcionamento da funcionalidade. Todos os ficheiros que não representem componentes devem ser colocados no interior da pasta *shared*.

4.3.4. Roteamento

A aplicação desenvolvida apresenta três funcionalidades principais, *login*, *tasks* e *members*. Para cada uma das funcionalidades, existem páginas específicas com componentes específicos. As páginas disponibilizadas para a aplicação apresentam assim diferentes rotas e diferentes objetivos, como descrito na tabela 6.

Rota	Objetivo
/login	Autenticar o utilizador e redirecionar o mesmo para a página principal *members*
/members	Visualizar lista dos elementos da equipa
/members/add	Adicionar um novo elemento à equipa
/members/edit/:id	Editar um elemento da equipa a partir do *id* recebido
/tasks	Visualizar, criar e editar tarefas
rota inválida	Redirecionar para a página principal *member*, se autenticado; caso contrário redirecionar para o *login*

Tabela 6- Rotas existentes na aplicação desenvolvida em Angular e React

Angular já inclui embutida a funcionalidade de roteamento. Quando a aplicação Angular é criada, um dos ficheiros automaticamente gerados é o ficheiro */src/app/app-routing.module.ts*, cuja responsabilidade é a configuração das rotas existentes, definindo o componente a ser devolvido.

```
10  const routes: Routes = [
11    { path: 'login', component: LoginPageComponent },
12    { path: 'tasks', component: TasksPageComponent, canActivate: [AuthGuard] },
13    {
14      path: 'members',
15      component: MembersPageComponent,
16      canActivate: [AuthGuard],
17    },
18    {
19      path: 'members/add',
20      component: MemberAddPageComponent,
21      canActivate: [AuthGuard],
22    },
23    {
24      path: 'members/edit/:id',
25      component: MemberEditPageComponent,
26      canActivate: [AuthGuard],
27    },
28    { path: '**', redirectTo: 'members' },
29  ];
```

Figura 17- Configuração de roteamento na aplicação Angular no ficheiro app-routing.module.ts

Na linha 24 da figura 17, é passado o parâmetro *id* do membro a ser editado. Na linha 28 da figura 18, é especificado para qual rota a aplicação se deve redirecionar no caso do *url* inserido não estar definido na lista de rotas existentes. Nas linhas 12, 16, 21 e 26 da figura 18, encontra-se um parâmetro de configuração *canActivate*, que, a partir de um *guard* passado, oferece um tipo de acesso especial à página. Neste caso, o *AuthGuard*, faz com que só seja possível o acesso a estas páginas, caso o utilizador esteja autenticado, ou seja, se os seus dados existirem na *localStorage*. O *guard* utilizado pode ser consultado no anexo G.

Este ficheiro faz com que seja disponibilizado automaticamente um componente chamado *router-outlet*. Este componente é utilizado para fazer a inserção das rotas e dos componentes na aplicação e é normalmente renderizado no componente de entrada da aplicação, o componente */src/app/app.component.ts*, como representado na linha 1 da figura 18.

```
1    <router-outlet></router-outlet>
```

Figura 18- Utilização do componente de roteamento em Angular

React utiliza uma *biblioteca* externa própria para conseguir utilizar roteamento no interior da aplicação, o React Router. Esta *biblioteca* oferece o componente *BrowserRouter*, cujo interior apresenta um *Switch*, responsável por analisar o *url* atual, e retornar o componente correspondente. Na linha 15 da figura 19, por exemplo, é verificado se o *url* corresponde a */tasks* e, caso isso se verifique, o componente devolvido será o *TasksPage*.

```
7   const Routes = () => {
8     return (
9       <BrowserRouter>
10        <Switch>
11          <Route path='/login' component={LoginPage} />
12          <PrivateRoute path='/members/edit/:id' component={MemberEditPage} />
13          <PrivateRoute path='/members/add' component={MemberAddPage} />
14          <PrivateRoute path='/members' component={MembersPage} />
15          <PrivateRoute path='/tasks' component={TasksPage} />
16          <Redirect from='/' to='/members' />
17        </Switch>
18      </BrowserRouter>
19    );
20  };
```

Figura 19- Configuração de roteamento em React

A linha 12 da figura 19 apresenta um parâmetro *id*, relativo ao membro a ser editado. Na linha 16 da figura 19, é especificado para qual rota a aplicação se deve redirecionar, no caso do *url* inserido não estar definido na lista de rotas existentes.

É de notar a utilização do componente *PrivateRoute*, entre a linha 12 e a 15 da figura 19, que é um componente desenvolvido de raiz, que faz a distinção entre páginas privadas e públicas. Este componente é responsável por redirecionar o utilizador para a página pretendida, verificando primeiro se o mesmo está autenticado, através da *localStorage*. Caso o utilizador não esteja autenticado, este é redirecionado para a página de *login*. O componente *PrivateRoute* está disponível no anexo M.

Para permitir o roteamento, há que fazer então a utilização do componente *Routes, da figura 20*, normalmente feita no componente "pai" da aplicação, no componente *App*, como representado na linha 6 da figura 21.

```
3   import Routes from './Routes';
4
5   function App() {
6     return <Routes />;
7   }
8
9   export default App;
```

Figura 20- Utilização do componente de roteamento em React no componente App

4.3.5. Funcionalidades e componentes

4.3.5.1. Layouts

Existem dois tipos de *layout* na aplicação desenvolvida, o *layout* autenticado e o não autenticado. Consideremos então estes *layouts* como uma moldura, onde serão inseridas as páginas. Desta forma, o *layout* não autenticado, disponível no Anexo N, mostra um título de boas-vindas à aplicação e a página de *login*. O *layout* autenticado, disponível no Anexo O, mostra um título da aplicação juntamente com o nome do utilizador autenticado, um botão de *logout* da aplicação, uma barra de navegação e ainda as páginas autenticadas no seu interior.

Apresentam-se os componentes de *layout* autenticado, nas figuras 21, 22 e 23.

```
9   export class AuthLayoutComponent {
10    isLoading: boolean = false;
11    authData = JSON.parse(localStorage.getItem('authData'));
12
13    constructor(private router: Router) {}
14
15    handleLogout = () => {
16      this.isLoading = true;
17      setTimeout(() => {
18        localStorage.clear();
19        this.router.navigate(['/login']);
20      }, 3000);
21    };
22  }
```

Figura 21- Componente do layout de páginas autenticadas em Angular, ficheiro TS

```
1  <app-loading *ngIf="isLoading"></app-loading>
2  <header class="auth-header">Team Manager - {{ authData.name }}</header>
3  <nav>
4    <a routerLink="/members">Team</a>
5    <a routerLink="/tasks">Tasks</a>
6    <button (click)="handleLogout()">Logout</button>
7  </nav>
8  <div class="app-container">
9    <ng-content></ng-content>
10 </div>
```

Figura 22- Componente do layout de páginas autenticadas em Angular, ficheiro HTML

```
6  const AuthLayout = ({ children }) => {
7    const [isLoading, setIsLoading] = useState(false);
8    const authData = JSON.parse(localStorage.getItem('authData'));
9    const history = useHistory();
10
11   const handleLogOut = () => {
12     setIsLoading(true);
13     setTimeout(() => {
14       localStorage.clear();
15       history.push('/');
16     }, 3000);
17   };
18
19   return (
20     <>
21       {isLoading && <Loading />}
22       <header className='auth-header'>Team Manager - {authData?.name}</header>
23       <nav>
24         <Link to='/tasks'>Tasks</Link>
25         <Link to='/members'>Team</Link>
26         <button onClick={handleLogOut}>Logout</button>
27       </nav>
28       <div className='app-container'>{children}</div>
29     </>
30   );
31 };
```

Figura 23- Componente do layout de páginas autenticadas em React

Com a inicialização de ambos os componentes, é criada a variável *authData* contendo os dados provenientes da *localStorage*, referentes ao utilizador autenticado, como observado na linha 11 da figura 21 e na linha 8 da figura 23. A variável *authData* é utilizada para renderizar o nome do utilizador juntamente com o título da aplicação, como apresentado na linha 2 da figura 22, para Angular, e linha 22 da figura 23, para React.

O botão de *logout*, quando clicado, é responsável por executar a função *handleLogout*. Pode observar-se a manipulação do evento de clique em ambos os componentes. Em Angular, a função utilizada para manipulação do clique do botão encontra-se especificada no ficheiro TS. A manipulação do evento de clique encontra-se na linha 6 do ficheiro 22 e na linha 26 da figura 23, para Angular e React. A variável *isLoading* serve para mostrar um ícone informativo de processamento, neste caso a eliminação da *localStorage* e o redireccionamento para a página de *login*. A variável *isLoading* é considerada para fazer uma renderização condicional na estrutura do componente, mostrando o componente *Loading* sempre que *isLoading* for afirmativo. A renderização condicional é feita em

Angular, através do **ngIf*, representado da linha 1 da figura 22. Em React, utiliza-se o JSX, para mostrar o componente *Loading*, através de um *&&* juntamente com a variável *isLoading*, como observa na linha 21 da figura 23. É de notar que a variável *isLoading* em React é uma variável de estado, cuja alteração de valor provoca uma nova renderização do componente, sendo visível os efeitos do seu valor.

Relembramos que os componentes de *layout* atuam como uma moldura das páginas. É utilizada contenção para mostrar a página no interior desta moldura, como verificado na linha 9 da figura 22 e na linha 28 da figura 23. Angular apresenta já definido um elemento interno, pronto a usar, o *ng-content*. Em React, é preciso fazer a receção do componente "filho", com a propriedade *children*, observado na linha 6 da figura 23, e utilização da mesma no retorno do componente, visto na linha 28 do ficheiro 23. A relação entre o componente de *layout* e as páginas é apresentada na linha 1 da figura 24 para Angular e na linha 40 da figura 26 para React, cujo interior é recebido por *ng-content* e *children*, respetivamente.

4.3.5.2. Login

O utilizador, se não autenticado, e como representado e explicado no ponto do roteamento, será sempre redirecionado para a página do *login*, responsável por apresentar um formulário com apenas duas entradas, o *email* e a *password*, utilizados para autenticar o utilizador. O formulário deve verificar se os campos foram inseridos corretamente, caso contrário mostra mensagens de erro. Após submissão do formulário com dados corretos, o serviço de *login* é chamado, e com a resposta positiva deste, a *localStorage* será atualizada com a informação do utilizador, redirecionando-o, de seguida, para o interior da aplicação. A resposta negativa do serviço informará sobre o erro ocorrido.

```
1  <app-not-auth-layout>
2    <h3>Login</h3>
3    <form [formGroup]="loginForm" (ngSubmit)="onSubmit()">
4      <label>Email:</label>
5      <input type="text" formControlName="email" class="login-input" />
6      <app-error-message *ngIf="email.errors && (email.dirty || email.touched)">
7        <span *ngIf="email.errors.required">Required</span>
8        <span *ngIf="email.errors.email">Invalid format</span>
9      </app-error-message>
10     <label>Password:</label>
11     <input type="password" formControlName="password" class="login-input" />
12     <app-error-message
13       *ngIf="password.errors && (password.dirty || password.touched)"
14     >
15       <span *ngIf="password.errors.required">Required</span>
16     </app-error-message>
17     <button type="submit">Login</button>
18   </form>
19 </app-not-auth-layout>
```

Figura 24- Componente da página de login em Angular, ficheiro HTML

```
12  export class LoginPageComponent {
13    constructor(
14      private fb: FormBuilder,
15      private authService: AuthService,
16      private router: Router
17    ) {}
18
19    loginForm = this.fb.group({
20      email: [null, [Validators.email, Validators.required]],
21      password: [null, Validators.required],
22    });
23
24    get email() {
25      return this.loginForm.get('email');
26    }
27
28    get password() {
29      return this.loginForm.get('password');
30    }
31
32    async onSubmit() {
33      if (this.loginForm.invalid) {
34        validateAllFormFields(this.loginForm);
35        return;
36      }
37
38      try {
39        const loginData = await this.authService.login(
40          this.loginForm.value.email,
41          this.loginForm.value.password
42        );
43        localStorage.setItem('authData', JSON.stringify(loginData));
44        this.router.navigate(['members']);
45      } catch (ex) {
46        alert(ex.message);
47      }
48    }
49  }
```

Figura 25- Componente da página de login em Angular, ficheiro TS

Na figura 24, é apresentada a estrutura renderizada para o formulário de *login* em Angular. Esta ferramenta precisa de especificar, no elemento formulário, a propriedade *formGroup*. O *formGroup*, recebe as configurações de validação do formulário do ficheiro TS do componente, a partir da variável *loginForm*, como mostrado na linha 19 da figura 25. A associação dos campos de escrita com os valores do formulários é feita através da propriedade *formControlName*, da linha 5 e 11 do ficheiro 24.

Para se proceder à verificação de erros de validação, é obrigatória a criação de uma variável de acesso aos campos no ficheiro TS, como verificado na linha 24 e 28 da figura 25. Com o uso destas variáveis, pode verificar-se a existência da propriedade *errors* no seu interior. Em caso afirmativo, *errors* conterá a propriedade com o nome do erro ocorrido, como *required* ou *invalid*, que será usada para mostrar mensagens de erro específicas. A utilização das variáveis do formulário, para apresentação das mensagens de erro, encontram-se nas linhas 6, 7 e 8 do ficheiro 24.

A associação da função de *submit* do formulário é feita com o uso da função pretendida, *onSubmit*, observado na linha 3 do ficheiro 24 e na linha 32 da figura 25. Na função *onSubmit*, é verificado se o formulário é válido. Se não for válido, a função *validateAllFormFields* é executada, de forma a despoletar as mensagens de erro, no formulário. Caso seja válido, será chamada a função de *login*, e

realizado o armazenamento no seu resultado na *localStorage*, finalizando com o redireccionamento do utilizador para a página *members*, como se observa na linha 44 da figura 25. Caso seja inválido, uma mensagem de alerta será lançada, com a descrição do erro ocorrido. Isto tudo pode ser observado na função *onSubmit*, na linha 32 da figura 26.

```
22  const LoginPage = () => {
23    const history = useHistory();
24    const { handleSubmit, errors, register } = useForm({
25      defaultValues,
26      resolver,
27    });
28
29    const handleLogin = async (formData) => {
30      try {
31        const loginData = await authApi.login(formData.email, formData.password);
32        localStorage.setItem('authData', JSON.stringify(loginData));
33        history.push('/members');
34      } catch (ex) {
35        alert(ex.message);
36      }
37    };
38
39    return (
40      <NotAuthLayout>
41        <h3>Login</h3>
42        <form onSubmit={handleSubmit(handleLogin)}>
43          <label>Email:</label>
44          <input type='text' ref={register} name='email' className='login-input' />
45          <div className='error'>{errors.email && errors.email.message}</div>
46          <label>Password:</label>
47          <input type='password' ref={register} name='password' className='login-input' />
48          <div className='error'>{errors.password && errors.password.message}</div>
49          <button type='submit'>Login</button>
50        </form>
51      </NotAuthLayout>
52    );
53  };
```

Figura 26- Componente da página login em React

React requer a importação de uma *biblioteca* externa, que permita o uso de formulários de forma avançada, para que seja equiparado com o formulário de Angular. Utiliza-se então a b*iblioteca* externa React-Hook-Forms, que apresenta uma implementação simples e uma *performance* soberba.

A *biblioteca* utilizada por React, é especificada com a função *useForm*, usada na linha 24 da figura 26. Os argumentos de *useForm* são um objeto de valores iniciais e outro de regras de validação do formulário, apresentados na linha 25 e 26 da figura 26, respetivamente, que podem ser consultados no Anexo T. Dessa mesma função é extraída a função de *handleSubmit*, e os objetos *errors* e *register*. O objeto *errors* pode ser diretamente utilizado na estrutura renderizada, de forma a mostrar a mensagem de erro de um campo, caso no seu interior exista uma propriedade com o mesmo nome do campo do formulário, como exemplificado na linha 45 e 48 da figura 26. A associação dos elementos dos campos

do formulário é feita com a propriedade *name* e também com a propriedade *ref*, que recebe o objeto *register*, proveniente da função *useForm.*

Finalmente, a função de *submit* do formulário é manipulada na linha 42 da figura 26, com a função *handleLogin* da linha 29 da figura 27. Contrariamente a Angular, a função de *submit* apenas é executada caso o formulário esteja correto; caso não esteja, as mensagens de erro apareceram automaticamente, ao clicar no botão de *login*, sem necessidade de uma função adicional. Idêntico a Angular, será chamado o serviço *login*, e com o seu sucesso, o armazenamento dos dados recebidos em *locaStorage*, seguido do redireccionamento do utilizador para a página *members*, a partir da linha 33 da figura 26.

4.3.5.3. Página *members*

A funcionalidade principal da aplicação, quando o utilizador está autenticado, encontra-se na página dos membros da equipa. Esta página é responsável por apresentar a lista dos elementos da equipa e os detalhes de cada elemento quando selecionado. Este componente "pai" apresenta assim dois componentes "filhos", o componente *MemberList*, que apresenta a lista e o componente *MemberDisplayer*, que apresenta os dados do membro selecionado.

O componente *MemberPage* tem a seu cargo a declaração e atualização de três variáveis: a variável *members*, que apresenta a lista de membros da equipa, a variável *selectedMember*, que armazena o membro selecionado, e ainda a variável *isLoading*.

```
11  export class MembersPageComponent implements OnInit {
12    isLoading: boolean = false;
13    members: Member[] = [];
14    selectedMember: Member = null;
```

Figura 27- Variáveis do componente MembersPage em Angular

```
10  const MembersPage = () => {
11    const [isLoading, setIsLoading] = useState(false);
12    const [members, setMembers] = useState([]);
13    const [selectedMember, setSelectedMember] = useState(null);
```

Figura 28- Variáveis do componente MembersPage em React

O *MembersPage* é responsável pela partilha destas duas variáveis pelos "filhos". A variável *members* é obtida através de chamada de serviço *getMembers*, que deve ser efetuada na inicialização do componente. As chamadas aos serviços não são instantâneas, requerem algum tempo até que se dê o retorno de resposta pelo BE; por esse motivo, é utilizada a variável *isLoading*, para mostrar um componente de espera. Angular permite a comunicação unidirecional instantânea do componente "pai" para o "filho", sendo que a atualização da variável *members* é refletida no componente *MemberList* sempre que sofre alterações. React, por sua vez, requer a utilização de variáveis de estado, para que a

alteração da variável *members* seja refletida no componente *MemberList*, ou em qualquer outro local que renderize o seu valor. A não utilização de uma variável de estado causa a inicial renderização de uma lista vazia, devido ao tempo de espera de resposta do BE. Com o retorno de resposta do BE, apesar da variável *members* ser atualizada, a DOM do componente não saberá que deve ser atualizada, causando a não exibição dos elementos recebidos.

```
18  ngOnInit(): void {
19    this.getMembers();
20  }
21
22  async getMembers() {
23    this.isLoading = true;
24    this.members = await this.memberService.getMembers();
25    this.isLoading = false;
26  }
```

Figura 29- Obtenção dos membros no componente MembersPage, em Angular

```
15  useEffect(() => {
16    getMembers();
17  }, []);
18
19  const getMembers = async () => {
20    setIsLoading(true);
21    const members = await membersApi.getMembers();
22    setMembers(members);
23    setIsLoading(false);
24  };
```

Figura 30- Obtenção dos membros no componente no componente MembersPage, em React

A variável *memberSelected* é obtida a partir do evento de seleção efetuado no componente "filho", *MemberList*. Para tal, o componente "pai" apresenta uma função a ser executada, quando este evento de seleção é verificado, o *handleSelectedMember*. O componente *MembersPage* tem então a obrigação de passar ao componente *MemberList* a variável de *input members* e a função de *output onSelectedMember*.

```
7       <app-member-list
8         [members]="members"
9         (onSelectMember)="handleSelectMember($event)"
10      ></app-member-list>
```

Figura 31- Integração do componente MemberList no MemberPage, em Angular

```
47      <div className='member-column'>
48        <MemberList members={members} onSelectMember={handleSelectMember} />
49      </div>
```

Figura 32- Integração do componente MemberList no MemberPage, em React

```
 9  export class MemberListComponent {
10    @Input() members: Member[];
11    @Output() onSelectMember = new EventEmitter();
```

Figura 33- Receção de argumentos de input e output no componente MemberList em Angular

```
4  const MemberList = ({ members, onSelectMember }) => {
```

Figura 34- Receção de argumentos de input e output no componente MemberList, em React

O componente "filho" por sua vez precisa de estar preparado para receber tais argumentos, como representado na linha 10 e 11 da figura 33 e na figura 34. O argumento de *input members* recebido é diretamente apresentado na UI, sem qualquer manipulação por parte do componente. Por sua vez, este cria uma lista usando o elemento *ul*, com os vários itens de lista *li*, que constituem cada elemento da lista de membros recebida. De forma a detetar qual o membro selecionado pelo utilizador, a função de *output* recebida é associada a cada elemento da lista, permitindo, assim, que a mesma seja executada quando o item da lista é selecionado, passando como argumento o elemento da lista em questão.

```
1  <ul class="members">
2    <li *ngFor="let member of members" (click)="onSelectMember.emit(member)">
3      <span class="badge">{{ member.id }}</span> {{ member.name }}
4    </li>
5  </ul>
```

Figura 35- Renderização da lista de membros no componente MemberList, em Angular

```
6      <ul className='members'>
7        {members.map((member) => (
8          <li key={member.id} onClick={() => onSelectMember(member)}>
9            <span className='badge'>{member.id}</span> {member.name}
10         </li>
11       ))}
12     </ul>
```

Figura 36- Renderização da lista de membros no componente MemberList, em React

Após a execução da função de output pelo componente "filho", a partir do clique num elemento da lista, o componente "pai" é avisado e executa uma função própria de resposta ao acontecido, a função *handleSelectedMember*, que irá atualizar a sua variável *selectedMember*.

```
39  handleSelectMember(member: Member) {
40    this.selectedMember = this.selectedMember === member ? null : member;
41  }
```

Figura 37- Função handleSelectedMember do componente MembersPage, em Angular

```
26  const handleSelectMember = (member) => {
27    setSelectedMember(selectedMember === member ? null : member);
28  };
```

Figura 38- Função handleSelectedMember do componente MembersPage, em React

A atualização da variável *selectedMember* permite que o componente *MemberDisplayer* apareça, mostrando a informação do membro selecionado. Caso a variável *selectedMember* não apresente um valor nulo, o componente *MemberDisplayer* é mostrado, caso contrário não é. A implementação de tal funcionalidade é obtida a partir de renderização condicional, observando a variável *selectedMember*, que pode ser observado nos anexos P e Q, para Angular e React, respetivamente.

O componente *MemberDisplayer*, disponível nos anexos R e S, para Angular e React respetivamente, recebem como argumento de *input* o *id* do membro selecionado. Este *id* é utilizado para a chamada do serviço *getMember*, no componente *MemberDisplayer*, que apresenta a informação completa do membro escolhido. Após retorno dos dados do membro, pelo serviço *getMember*, a variável *member* do componente *MemberDisplayer* é atualizada, permitindo a renderização dos dados do membro recebida na UI. A demora de resposta do serviço faz com que inicialmente a variável *members* seja nula, até a resposta ser recebida, causando erros na renderização inicial do componente. Estes erros devem-se à tentativa de aceder a parâmetros do membro que ainda não existem, sendo necessário o uso de *optional chaining*.

A renderização dos dados do membro selecionado é acompanhada por dois botões, que permitem o avanço para a página de edição do membro e a eliminação do membro selecionado, desempenhadas pelas funções *handleEdit* e *handleDelete*, respetivamente. A função *handleEdit* é responsável por redirecionar o utilizador para a página de edição do mesmo, enquanto a função *handleDelete* é responsável por eliminar o membro selecionado, atualizando assim a lista dos membros.

Uma vez apresentadas as principais metodologias de desenvolvimento com as ferramentas, torna-se desnecessário insistirmos na sua comparação, através dos restantes componentes e funcionalidades das aplicações, que se encontram disponíveis para consulta na página do GitHub, apresentadas nas referências [44] [45].

4.4. Comparação técnica

4.4.1. Número de linhas implementadas por componente

Apresenta-se a comparação do número de linhas implementadas por componente em ambas as aplicações. Como mencionado anteriormente, Angular exige dois ficheiros, TS e HTML, para a criação de um componente, enquanto React apenas precisa de um ficheiro JSX. Para além da comparação do número de linhas nos componentes das páginas da tabela 7, a mesma comparação é feita para os componentes das *features* e dos componentes *shared*, nos anexos U e V, respetivamente.

	Login	Members	MemberAdd	MemberEdit	Tasks	Total
Angular (TS)	49	45	22	37	53	206
Angular (HTML)	23	20	7	8	20	78
React (JSX)	59	60	27	36	79	261

Tabela 7- Número de linhas de código implementadas para os componentes das páginas em Angular e React

É notório que o desenvolvimento de componentes em React requer menos linhas de código comparado com Angular, o que pode ser traduzido em desenvolvimento mais rápido.

4.4.2. Espaço em disco ocupado pelas soluções em desenvolvimento

É de seguida apresentado o tamanho das soluções de ambas as aplicações em desenvolvimento. Em ambas as soluções, existe uma estrutura de pastas e ficheiros prontos a receber uma implementação. São considerados o espaço ocupado na pasta *src*, onde são feitos os desenvolvimento e também os espaços ocupados pelas aplicações com e sem os *node modules*, que são bibliotecas que oferecem um leque de funções, utilizadas no interior das aplicações.

	Src	Sem *node modules*	Com *node modules*
Angular	128 KB	6.70 MB	362 MB
React	96 KB	2.62 MB	215 MB

Tabela 8- Espaço em disco das aplicações de Angular e React, na src, com e sem os node modules

O espaço ocupado por React é bastante inferior ao de Angular. React apenas importa e utiliza as *bibliotecas* de que precisa estritamente, enquanto Angular apresenta no seu interior um conjunto enorme de *bibliotecas*, que permitem o desenvolvimento de qualquer funcionalidade, fazendo com que muitas delas não sejam usadas e ocupem espaço. Não considerando os *node modules*, o espaço ocupado pela solução React é praticamente metade do de Angular.

4.4.3. Duração e espaço em disco ocupado pela *build*

A *build* apresenta-se como uma versão compilada da aplicação, ou seja, é o resultado da conversão das linhas de código escritas para linguagem máquina. Os ficheiros da *build* podem ser utilizados para colocar uma aplicação a correr num servidor.

	Tamanho	Duração
Angular	6.05 MB	8831ms
React	1.39 MB	29280ms

Tabela 9- Duração e espaço em disco das aplicações de Angular e React, com e sem os node modules

O espaço em disco ocupado pela *build* de React é muito inferior ao de Angular. No entanto, o tempo demorado é três vezes superior ao de Angular.

4.4.4. Duração de execução de testes unitários

Foram implementados testes unitários apenas para o componente da página de *login*. Estes testes procuram avaliar o funcionamento do formulário, verificando a chamada do serviço de *login*, dependendo da validade dos campos inseridos no formulário. Os ficheiros de teste encontram-se nos Anexos W e X.

Angular	React
0.157s	3.859s

Tabela 10- Duração da execução de testes unitários da página de login

A duração de execução dos testes unitários de Angular é muito inferior à de React. Assim, caso a metodologia de desenvolvimento inclua *Test Driven Development* (TDD), Angular é uma melhor opção.

4.5. Comparação geral

Após o desenvolvimento aplicacional, com ambas as ferramentas, foi possível uma comparação entre formas de implementação, juntamente com a deteção de vantagens e desvantagens.

A estrutura considera-se boa em ambas as ferramentas. Não é possível fazer uma comparação mais aprofundada neste critério, uma vez que a aplicação desenvolvida é pequena e não encontra uma grande estrutura de ficheiros.

Os componentes de Angular são mais complicados e confusos de serem desenvolvidos, quando comparados com os componentes de React. Cada componente de Angular inclui três ficheiros, enquanto

um componente React apenas inclui dois. Com o desenvolvimento da aplicação, e com uma grande quantidade de ficheiros abertos em simultâneo, notou-se que em Angular se torna mais confusa e cansativa a navegação entre ficheiros. Estes fatores provocam quebras de raciocínio no programador, levando-o a tomar mais tempo para o desenvolvimento e integração dos componentes. A integração de componentes mostrou-se mais simples com a ferramenta React. Em React, os componentes comportam-se como autênticas funções, cujos argumentos são passados pelos componentes "pai", sendo utilizados diretamente no componente "filho". Em Angular, as passagens de variáveis e funções têm de ser declaradas no componente "filho", como *Inputs* e *Outputs* respetivamente, sendo que as funções de *output* não se comportam como funções, mas como *EventEmiters*, precisando de um *.emit()* para avisarem o componente "pai" sobre alguma ação.

As funcionalidades dos componentes são mais fáceis de desenvolver em React, uma vez que as variáveis e funções dos componentes se encontram no mesmo ficheiro onde está a estrutura HTML. A partilha do mesmo ficheiro permite que seja fácil e rápido o acesso às variáveis e funções, através de um clique em cima delas . Em Angular, a separação entre as variáveis e funções e a estrutura HTML faz com que seja preciso alternar constantemente entre ficheiros para concretizar um desenvolvimento, o que é cansativo e até desconcentrante. Ainda com Angular, a utilização destas variáveis e funções é feita com as mesmas aspas, ou seja, como texto. O facto de serem usadas como texto faz com que não seja possível navegar diretamente a partir do HTML e também não oferece a funcionalidade de *auto complete*, que ajuda o programador a selecionar as variáveis e funções existentes no ficheiro de TS.

Dentro do ficheiro de TS ou JS, a utilização das variáveis e funções é mais facilitada em React, visto que as suas utilizações são diretas. Angular, após declaração das mesmas, exige que os seus acessos sejam feitos a partir da variável *this*, que se refere ao componente onde se encontram. Porém, a utilização das variáveis e funções no HTML dispensa o *this*. Angular, ao utilizar TS, permite o uso de tipos nas variáveis, fazendo com que o código se torne mais fiável, detetando uma grande percentagem de erros, mesmo antes da execução aplicacional. React apresenta as variáveis de estado, que são bastante úteis no desenvolvimento de qualquer componente. Estas variáveis comportam-se como observadores sobre uma variável, que, quando é atualizada, informa a estrutura HTML que deve ser atualizada nos locais onde ela é usada. Esta funcionalidade é apenas permitida em React, que utiliza *Virtual* DOM. Esta apresenta-se como uma cópia da DOM, que a partir das variáveis de estados, registará em si as alterações feitas, após o que a *Virtual* DOM comparar-se-á com a DOM, atualizando nesta apenas o que foi alterado, permitindo assim uma ótima *performance*.

As renderizações condicionais e de listas consideram-se mais simples com a junção de JS e HTML em React, comparativamente à utilização das diretorias de Angular **ngFor e *ngIf*.

Os formulários, apesar de já incluídos na ferramenta Angular, exigem a escrita de muito código para a sua implementação, tornando tanto o ficheiro de TS como o de HTML grandes e confusos. React, não apresentando uma funcionalidade boa de formulários, exigiu que se optasse por uma *biblioteca*

externa. Sendo React uma ferramenta atual e grande, foi fácil obter-se uma boa ferramenta, que exigiu menos código desenvolvido e uma muito melhor *performanc*e.

Concluindo, a nível de desenvolvimento aplicacional, considero React a melhor ferramenta de desenvolvimento de FE. React apresenta uma metodologia de implementação muito simples, rápida e de fácil compreensão, que é uma mais-valia para qualquer programador.

CAPÍTULO 5

Conclusões

5.1. Principais conclusões

A realização desta dissertação permitiu o desenvolvimento de uma nova metodologia de avaliação e comparação de ferramentas de desenvolvimento de FE. Após desenvolvimento da metodologia, esta foi colocada em prática, comprovando assim a sua utilidade e vantagem.

O primeiro passo da metodologia permite que, através de critérios bem definidos e basilares, sejam selecionadas as ferramentas com as maiores e melhores bases e qualidades, reduzindo significativamente o universo, para continuação da aplicação da metodologia. O segundo passo, já com a filtragem feita, permite que as ferramentas de desenvolvimento selecionadas tenham os seus conceitos de desenvolvimento base analisados e comparados. Isto possibilita um conhecimento mais técnico do desenvolvimento com as tecnologias, oferecendo uma visão mais ampla daquilo que é presenteado por cada uma. O terceiro e último passo põe o desenvolvimento com as ferramentas em prática, com a implementação de uma pequena aplicação *web*. O facto da pequena aplicação escolhida apresentar uma grande parte de funcionalidades, existentes numa aplicação normal *web*, faz com que a experiência de desenvolvimento da aplicação seja sentida pelo programador. Desta forma, é transmitida a perceção de desenvolvimento com a ferramenta em projetos futuros, identificando as suas facilidades e dificuldades de implementação.

Após completar todas as fases da metodologia, o resultado é a apresentação da melhor ferramenta de desenvolvimento. Este resultado não é algo puramente universal, visto também considerar a opinião do programador que aplica a metodologia, fazendo com que a ferramenta seja uma escolha adaptada. Em conclusão, a metodologia proposta mostrou-se inovadora e útil: inovadora, porque sugere uma forma nova e única de fazer a seleção de ferramentas de desenvolvimento a partir de critérios e conceitos pré-definidos; útil, porque permite a qualquer indivíduo, com ou sem experiência, a aplicação da metodologia. A aplicação da metodologia torna possível a avaliação e comparação de quaisquer ferramentas de desenvolvimento, sendo o resultado uma maior compreensão das ferramentas e uma ideia clara sobre por qual optar para desenvolvimentos futuros.

5.2. Perspetivas de trabalhos futuros

A metodologia proposta pode ser aperfeiçoada com a prática, sendo adicionados ou substituídos alguns dos critérios, consoante a opinião dos programadores que fizerem uso dela. Ainda, esta metodologia poderá ser utilizada para fazer a avaliação e comparação de ferramentas de desenvolvimento de outras naturezas. Porém, nestes casos, a sua correta utilização implicará uma recolha de critérios e conceitos diferentes, que melhor se adaptem a ferramentas dessa natureza.

Referências Bibliográficas

[1] M. Pekarsky (2020), "Does your web app needs a front-end framework?", Stack Overflow, disponível em https://stackoverflow.blog/2020/02/03/is-it-time-for-a-front-end-framework/, consultado a 12/09/2020

[2] M. Hamedani (2018), "React vs. Angular: The Complete Comparison", Programming with Mosh, disponível em https://programmingwithmosh.com/react/react-vs-angular/, consultado a 12/09/2020

[3] B. Wozniewicz (2019), "The Difference Between a Framework and a Library", FreeCodeCamp, disponível em https://www.freecodecamp.org/news/the-difference-between-a-framework-and-a-library-bd133054023f/, consultado a 14/09/2020

[4] D. Otuagomah (2020), "Comparing the most popular frontend JavaScript frameworks", Educative.io, disponível em https://www.educative.io/blog/compare-popular-frontend-javascript-frameworks?aid=5082902844932096&utm_source=google&utm_medium=cpc&utm_campaign=blog-dynamic&gclid=Cj0KCQjwhvf6BRCkARIsAGl1GGiq0FmRW6FDrbcUdTOd_DVs0_WIuGP0VIj-DJYy0Fn38GlNFTuX5QsaAqvDEALw_wcB, consultado a 14/09/2020

[5] E. Normand (2017), "Why Do We Use Web Frameworks?", LispCast, disponível em https://lispcast.com/why-web-frameworks/, consultado a 14/09/2020

[6] A. Aguiar e G. David (2000), "A Minimalist Approach to Framework Documentation", Proceedings of the Conference on Object-Oriented Programming, Systems, Languages, and Applications, disponível em https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.135.5427&rep=rep1&type=pdf, consultado a 14/09/2020

[7] B., G., Dénommé, P. (1999), Documenting frameworks. In Building Application Frameworks: Object-Oriented Foundations of Framework Design, pp.495-504. John Wiley and Sons

[8] G. Bakradze, "How to Choose the Best Front-end Framework", Toptal, disponível em https://www.toptal.com/javascript/choosing-best-front-end-framework, consultado a 15/09/2020

[9] C. Lienert (2017), "How to Choose the Right Front-end Framework for Your Company", SitePoint, disponível em https://www.sitepoint.com/choose-the-right-front-end-framework-for-your-company/, consultado a 15/09/2020

[10] W. Koffel (2013), "Chosing a Web Framework", disponível em https://will.koffel.org/post/2013/choosing-a-web-framework/, consultado a 17/09/2020

[11] SCDC, "What is community development", disponível em https://www.scdc.org.uk/who/what-is-community-development, consultado a 12/09/2020

[12] Siddhartth (2009), “15 Important Considerations for Choosing a Web Dev Framework”, TutsPlus, disponível em https://code.tutsplus.com/tutorials/15-important-considerations-for-choosing-a-web-dev-framework--net-8035, consultado a 12/09/2020

[13] T. Chaves (2017), “How to Choose a Front-End Framework”, Forbes, disponível em https://www.forbes.com/sites/forbestechcouncil/2017/04/07/how-to-choose-a-front-end-framework/#132200f4bb93, consultado a 17/09/2020

[14] C. Varisco (2016), “How to choose a framework”, Hackemoon, disponível em https://hackernoon.com/how-to-choose-a-framework-ea8b5b1e1f44, consultado a 14/09/2020

[15] DigiMantraLabs, “How to Choose the Best Front-End Framework”, disponível em https://digimantralabs.com/best-front-end-framework/, consultado a 12/09/2020

[16] Valamis, “Learning Curve Theory”, disponível em https://www.valamis.com/hub/learning-curve#what-is-learning%20curve, consultado a 13/09/2020

[17] P. Krane (2019), “Wish to Choose the Best Frontend Framework for your Startup”, Creativetinge, disponível em https://creativetinge.com/peter/ui/choose-the-best-frontend-framework/, consultado a 12/09/2020

[18] D. Schesselman (2018), “Pros & Cons of Front-End Frameworks”, EnvisionIt, disponível em https://envisionitagency.com/blog/2018/04/pros-cons-front-end-web-frameworks/, consultado a 15/09/2020

[19] S. Krause, “js-framework-benchmark”, GitHub, disponível em https://github.com/krausest/js-framework-benchmark, consultado a 15/09/2020

[20] Mozilla, “What is web performance?”, disponível em https://developer.mozilla.org/en-US/docs/Learn/Performance/What_is_web_performance, consultado a 14/09/2020

[21] A. Gasparyan (2019), “Top 10 Things You Need to Know About Website Performance “, Monits, disponível em https://www.monitis.com/blog/top-10-things-you-need-to-know-about-website-performance/, consultado a 13/09/2020

[22] D. Miessler, modelo MVC, disponível em https://danielmiessler.com/study/mvc, consultado a 12/09/2020

[23] Valamis, exemplo de curva de aprendizagem, disponível em https://www.valamis.com/hub/learning-curve, consultado a 17/09/2020

[24] Npm, número de downloads NPM de Angular, React e Vue, disponível em https://www.npmtrends.com/react-vs-vue-vs-@angular/core-vs-ember-source, consultado a 17/09/2020

[25] GitHub, repositório React, disponível em https://github.com/facebook/react, consultado a 17/09/2020

[26] GitHub, repositório Angular, disponível em https://github.com/angular/angular, consultado a 17/09/2020
[27] GitHub, repositório Vue, disponível em https://github.com/vuejs/vue, consultado a 17/09/2020
[28] GitHub, repositório Ember, disponível em https://github.com/emberjs/ember.js, consultado a 17/09/2020
[29] StackOverflow (2020), "Most Loved, Dreaded, and Wanted", disponível em https://insights.stackoverflow.com/survey/2020#technology-most-loved-dreaded-and-wanted-web-frameworks-loved2, consultado a 13/09/2020
[30] StackOverflow (2020), "Most Loved, Dreaded, and Wanted", disponível em https://insights.stackoverflow.com/survey/2020#technology-most-loved-dreaded-and-wanted-web-frameworks-wanted2, consultado a 13/09/2020
[31] Linkedin, quantidade de ofertas de emprego de React, disponível em https://www.linkedin.com/jobs/search/?geoId=92000000&keywords=react&location=Mundialmente, consultado a 15/09/2020
[32] Linkedin, quantidade de ofertas de emprego de Angular, disponível em https://www.linkedin.com/jobs/search/?geoId=92000000&keywords=angular&location=Mundialmente, consultado a 15/09/2020
[33] Linkedin, quantidade de ofertas de emprego de Vue, disponível em https://www.linkedin.com/jobs/search/?geoId=92000000&keywords=vue&location=Mundialmente, consultado a 15/09/2020
[34] Linkedin, quantidade de ofertas de emprego de Ember, disponível em https://www.linkedin.com/jobs/search/?geoId=92000000&keywords=ember&location=Mundialmente, consultado a 15/09/2020
[35] GitHub , contribuidores de React, disponível em https://github.com/facebook/react/graphs/contributors, consultado a 12/09/2020
[36] GitHub, contribuidores de Angular, disponível em https://github.com/angular/angular/graphs/contributors, consultado a 12/09/2020
[37] GitHub, contribuidores de Vue, disponível em https://github.com/vuejs/vue/graphs/contributors, consultado a 12/09/2020
[38] StackOverflow, perguntas existentes em React, Angular e Vue, disponível em https://insights.stackoverflow.com/trends?tags=reactjs%2Cvue.js%2Cangular, consultado a 12/09/2020
[39] S. Krause, resultados da aplicação js-framework-benchmark, RawGit, disponível em https://rawgit.com/krausest/js-framework-benchmark/master/webdriver-ts-results/table.html, consultado a 14/09/2020
[40] GitHub, dependências React, disponível em https://github.com/facebook/react/network/dependents, consultado a 10/09/2020

[41] GitHub, dependências Angular, disponível em https://github.com/angular/angular/network/dependents, consultado a 10/09/2020

[42] GitHub, dependências Vue, disponível em https://github.com/vuejs/vue/network/dependents, consultado a 10/09/2020

[43] GitHub, dependências Ember, disponível em https://github.com/emberjs/ember.js/network/dependents, consultado a 10/09/2020

[44] Vídeo de demonstração da aplicação desenvolvida em React, disponível em https://iscteiul365-my.sharepoint.com/:v:/g/personal/rjfro_iscte-iul_pt/Ebq40ePE_p1EnDJakqnNOqABGNQ7Fn7WAeYmvQAdr6vlcA?e=zZVCYq

[45] Repositório da aplicação desenvolvida em Angular no GitHub, disponível em https://github.com/GIT-RR/angular-team-manager.git,

[46] Repositório da aplicação desenvolvida em React no GitHub, disponível em https://github.com/GIT-RR/react-team-manager.git

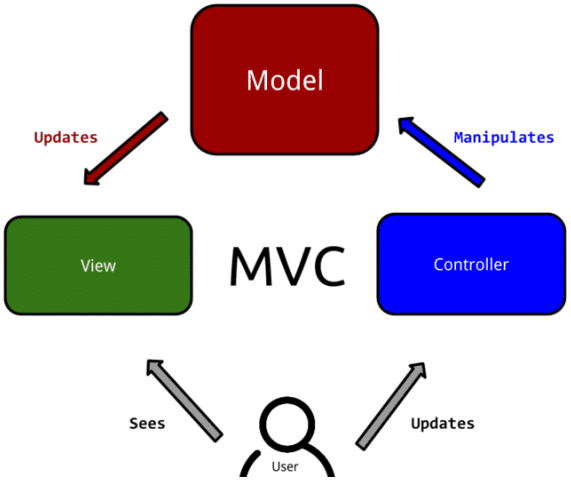
[47] *Site* oficial de React, disponível em https://reactjs.org/, consultado a 09/09/2020

[48] *Site* oficial de Angular, disponível em https://angular.io/, consultado a 09/09/2020

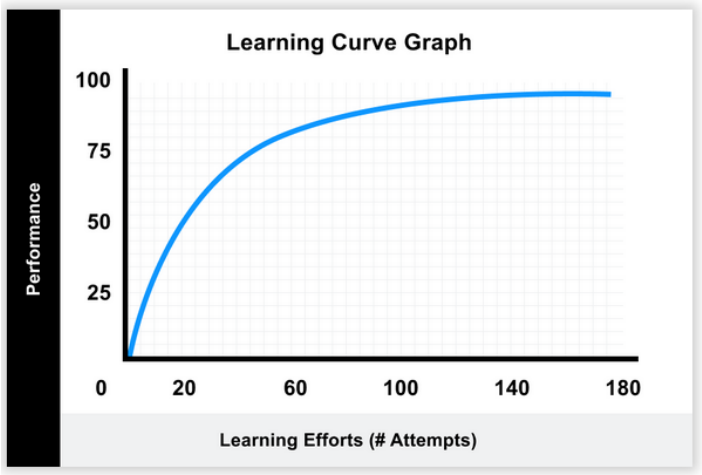
[49] *Site* oficial de Vue, disponível em https://vuejs.org/, consultado a 09/09/2020

Anexos

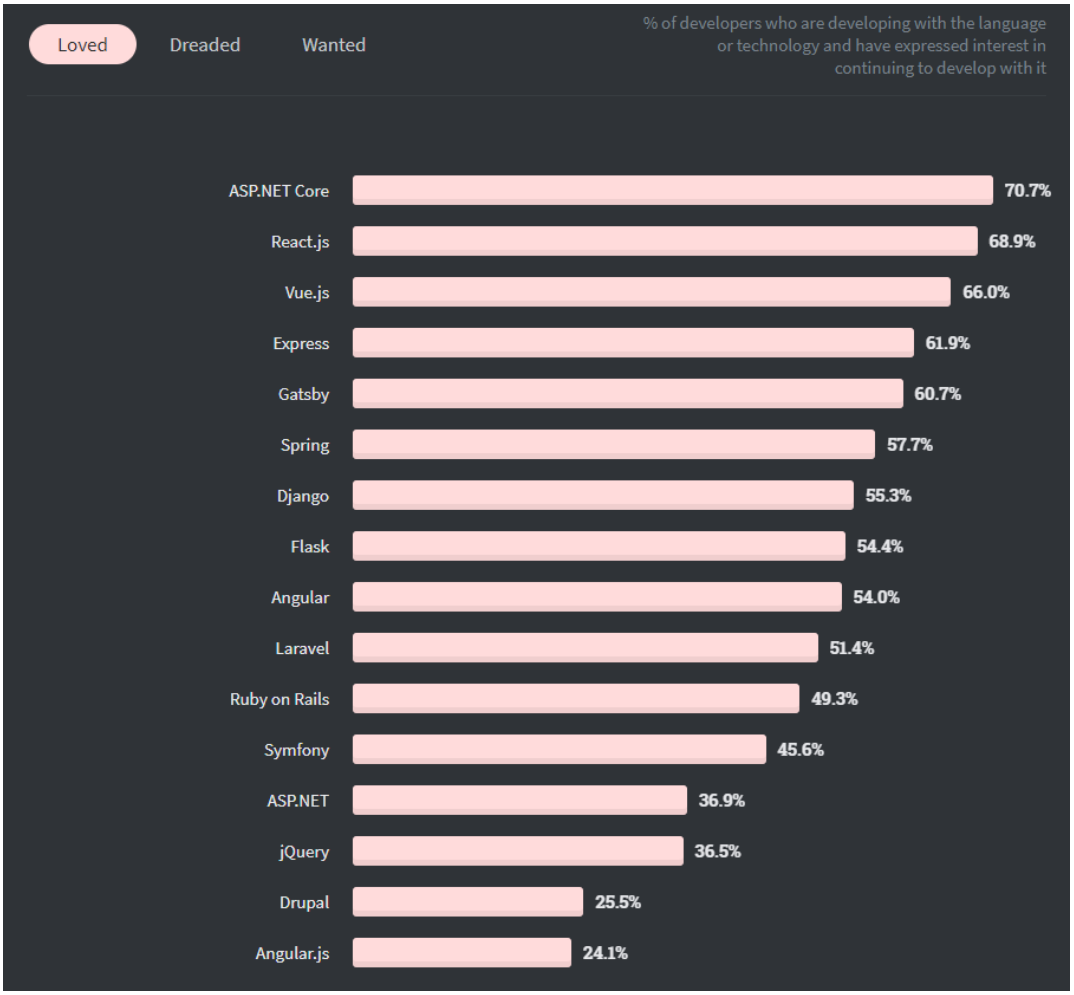
Anexo A – Modelo MVC – Modelo, visualização e controlo [22]



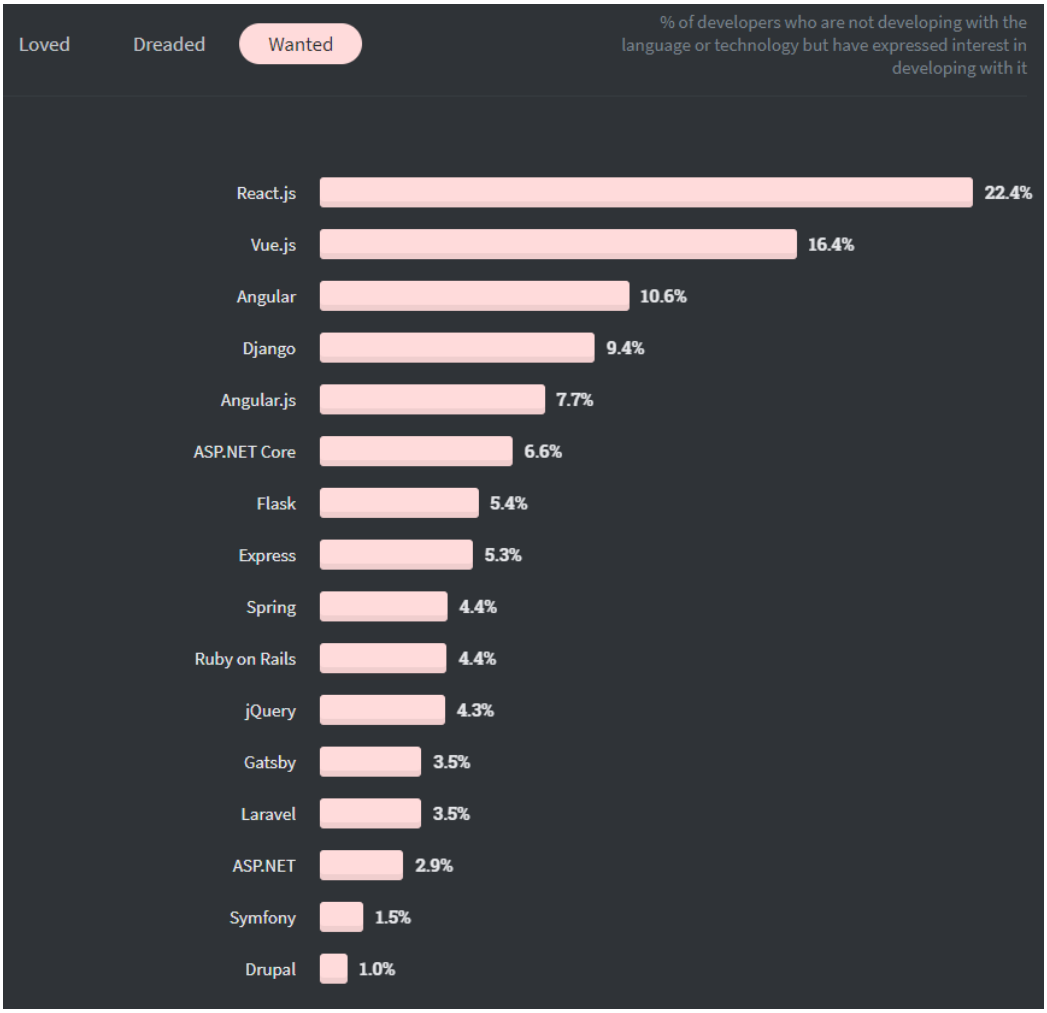
Anexo B – Exemplo de diagrama de curva de aprendizagem [23]



Anexo C – Percentagem de programadores que desenvolvem com a ferramenta de desenvolvimento *web* e expressam interesse de continuidade de desenvolvimento com a mesma, em 2020 [29]



Anexo D – Percentagem de programadores que não desenvolvem com a ferramenta de desenvolvimento web, mas expressam interesse de desenvolvimento com a mesma, em 2020 [30]



Anexo E – Duração em milissegundos de determinadas ações efetuadas na tabela da aplicação *js-framework-benchmark* [19]

Name Duration for...	angular-v8.0.1	react-v16.8.6	vue-v2.6.2
Issues Errors are red, cheats are yellow			
create rows creating 1,000 rows	152.1 ± 1.4 (1.00)	180.6 ± 1.9 (1.19)	167.0 ± 3.2 (1.10)
replace all rows updating all 1,000 rows (5 warmup runs).	40.5 ± 0.7 (1.00)	46.0 ± 1.1 (1.14)	46.8 ± 0.7 (1.16)
partial update updating every 10th row for 1,000 rows (3 warmup runs). 16x CPU slowdown.	148.6 ± 1.0 (1.00)	220.2 ± 3.9 (1.48)	232.1 ± 4.1 (1.56)
select row highlighting a selected row. (no warmup runs). 16x CPU slowdown.	76.3 ± 1.5 (1.00)	122.2 ± 3.7 (1.60)	310.8 ± 6.9 (4.07)
swap rows swap 2 rows for table with 1,000 rows. (5 warmup runs). 4x CPU slowdown.	30.6 ± 0.4 (1.00)	33.4 ± 0.2 (1.09)	47.1 ± 1.1 (1.54)
remove row removing one row. (5 warmup runs).	39.4 ± 0.4 (1.00)	45.6 ± 0.4 (1.16)	46.3 ± 0.2 (1.18)
create many rows creating 10,000 rows	1,462.3 ± 35.0 (1.09)	1,860.1 ± 53.1 (1.39)	1,340.9 ± 23.6 (1.00)
append rows to large table appending 1,000 to a table of 10,000 rows. 2x CPU slowdown	299.1 ± 3.9 (1.00)	341.3 ± 3.0 (1.14)	311.0 ± 3.7 (1.04)
clear rows clearing a table with 1,000 rows. 8x CPU slowdown	247.5 ± 2.6 (1.67)	148.3 ± 0.9 (1.00)	158.4 ± 1.5 (1.07)
geometric mean of all factors in the table	1.07	1.23	1.36
compare: Green means significantly faster, red significantly slower	compare	compare	compare

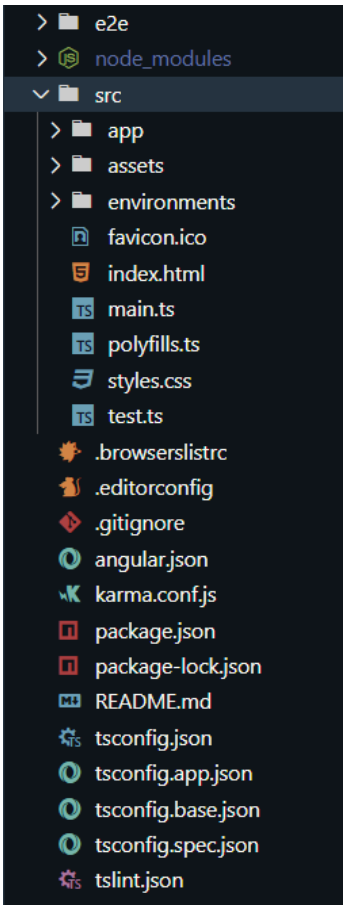
Anexo F – Métricas de inicialização de cada ferramenta na aplicação *js-framework-benchmark* [19]

Name	angular-v8.0.1	react-v16.8.6	vue-v2.6.2
consistently interactive a pessimistic TTI - when the CPU and network are both definitely very idle. (no more CPU tasks over 50ms)	2,867.2 ± 1.8 (1.26)	2,581.7 ± 0.5 (1.13)	2,275.9 ± 0.9 (1.00)
script bootup time the total ms required to parse/compile/evaluate all the page's scripts	113.9 ± 1.2 (7.12)	16.0 ± 0.0 (1.00)	16.0 ± 0.0 (1.00)
total kilobyte weight network transfer cost (post-compression) of all the resources loaded into the page.	295.2 ± 0.0 (1.40)	261.0 ± 0.0 (1.24)	210.8 ± 0.0 (1.00)
geometric mean of all factors in the table	2.32	1.12	1.00

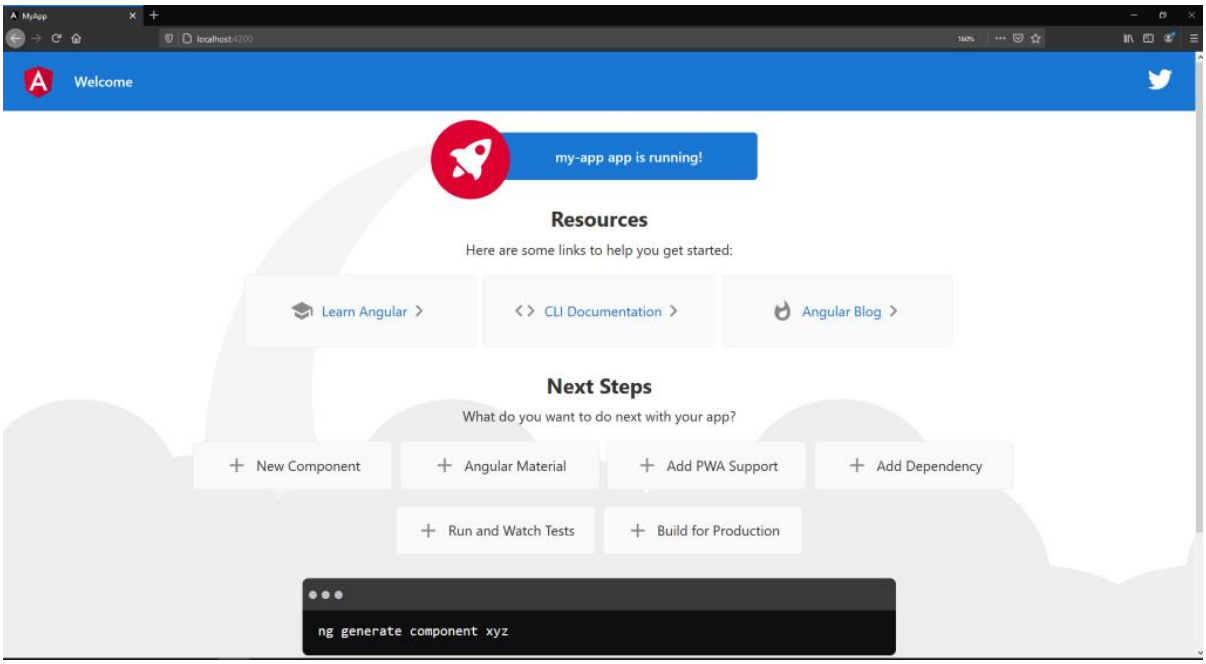
Anexo G – Alocação de memória em *Megabytes de* cada ferramenta, para algumas das ações efetuadas na tabela da aplicação *js-framework-benchmark* [19]

Name	angular-v8.0.1	react-v16.8.6	vue-v2.6.2
ready memory Memory usage after page load.	2.7 (2.20)	1.3 (1.06)	1.2 (1.00)
run memory Memory usage after adding 1000 rows.	5.1 (1.27)	4.3 (1.08)	4.0 (1.00)
update eatch 10th row for 1k rows (5 cycles) Memory usage after clicking update every 10th row 5 times	5.5 (1.25)	5.1 (1.17)	4.4 (1.00)
replace 1k rows (5 cycles) Memory usage after clicking create 1000 rows 5 times	5.2 (1.27)	6.5 (1.59)	4.1 (1.00)
creating/clearing 1k rows (5 cycles) Memory usage after creating and clearing 1000 rows 5 times	4.3 (1.62)	3.6 (1.38)	2.6 (1.00)
geometric mean of all factors in the table	1.49	1.24	1.00

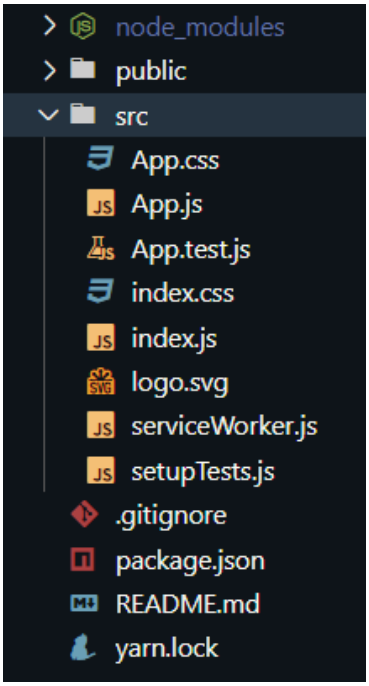
Anexo H – Estrutura aplicacional de um projeto criado em Angular, com o comando *ng new my-app*



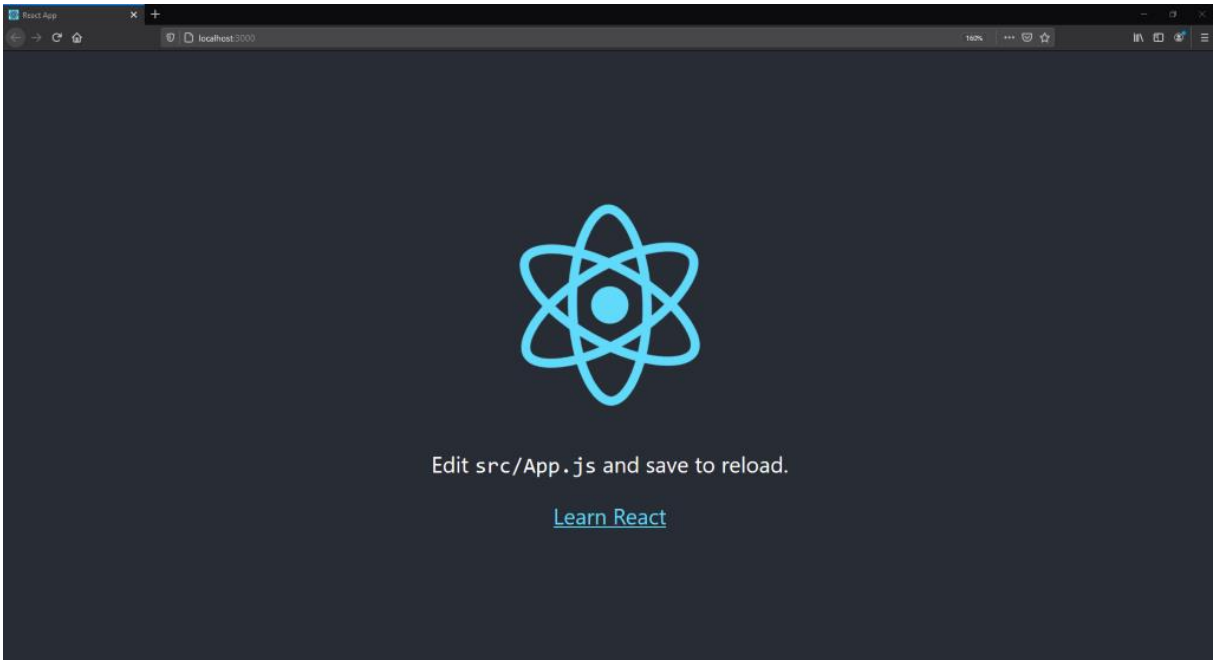
Anexo I – Página mostrada ao executar a aplicação Angular acabada de criar



Anexo J – Estrutura aplicacional de um projeto criado em React com o comando *npx create-react-app my-app*



Anexo K – Página mostrada ao executar a aplicação React acabada de criar



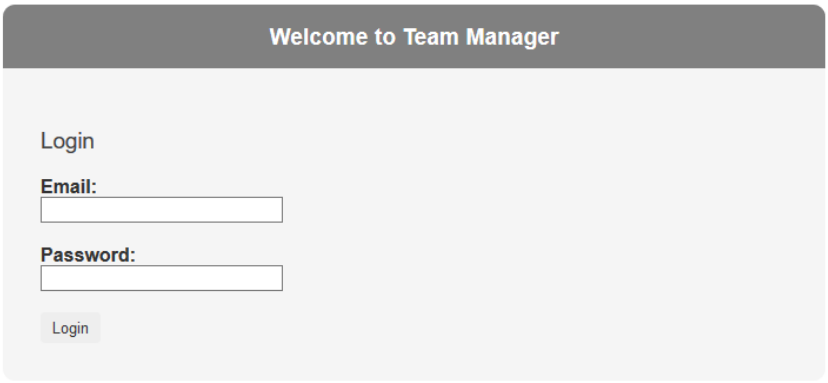
Anexo L – Ficheiro *guard* da aplicação Angular, responsável por redirecionar o utilizador para o *login*, caso este não esteja autenticado

```
14  export class AuthGuard implements CanActivate {
15    constructor(private router: Router) {}
16
17    canActivate(
18      next: ActivatedRouteSnapshot,
19      state: RouterStateSnapshot
20    ): Observable<boolean> | Promise<boolean> | boolean {
21      const authData = localStorage.getItem('authData');
22
23      //is authenticated?
24      if (authData) {
25        return true;
26      }
27
28      // if not authenticated >> go to login
29      this.router.navigate(['/login']);
30      return false;
31    }
32  }
```

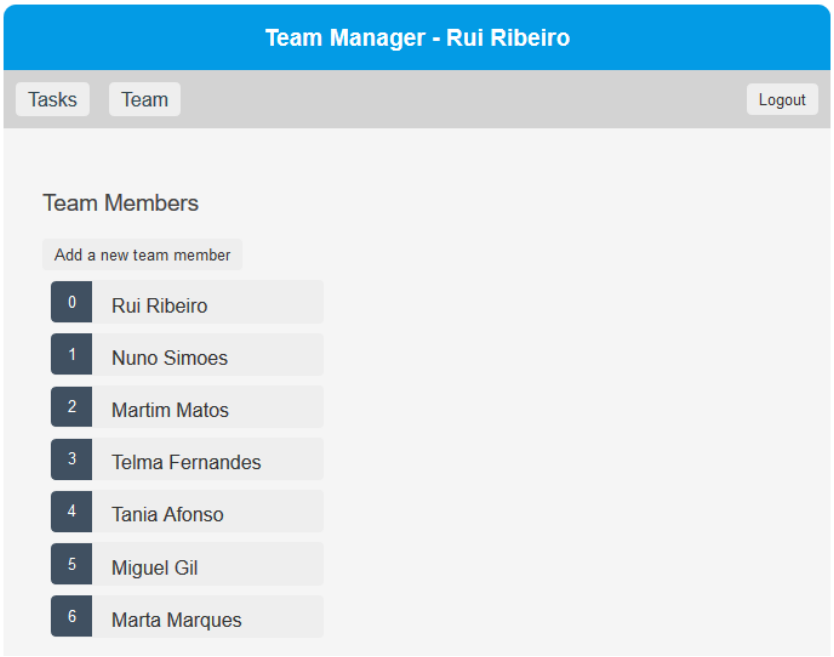
Anexo M – Componente PrivateRoute da aplicação React, responsável por redirecionar o utilizador para o *login*, caso este não esteja autenticado

```
22  const PrivateRoute = ({ component: Component, ...rest }) => {
23    const authData = localStorage.getItem('authData');
24
25    // ternary to check if is authenticated
26    return (
27      <Route
28        {...rest}
29        render={(props) => (
30          authData ?
31            <Component {...props} /> : <Redirect to='/login' />)}
32      />
33    );
34  };
```

Anexo N – *Layout* não autenticado da aplicação



Anexo O – *Layout* autenticado da aplicação



Anexo P – Renderização condicional do componente *MemberDisplayer* no componente *MembersPage* em Angular

```
12      <div class="member-column">
13        <app-member-displayer
14          *ngIf="selectedMember"
15          [id]="selectedMember.id"
16          (onDelete)="handleDeleteMember($event)"
17        ></app-member-displayer>
18      </div>
```

Anexo Q – Renderização condicional do componente *MemberDisplayer* no componente *MembersPage* em React

```
50              <div className='member-column'>
51                {selectedMember && (
52                  <MemberDisplayer id={selectedMember.id} onDelete={handleDeleteMember} />
53                )}
54              </div>
55            </div>
```

Anexo R – Componente *MemberDisplayer* em Angular, ficheiro TS e ficheiro HTML

```
20  export class MemberDisplayerComponent implements OnChanges {
21    @Input() id: number;
22    @Output() onDelete = new EventEmitter();
23    member: Member;
24
25    constructor(private memberService: MemberService, private router: Router) {}
26
27    ngOnChanges(): void {
28      this.getMember();
29    }
30
31    async getMember() {
32      this.member = await this.memberService.getMember(this.id);
33    }
```

```
1  <div>
2    <h2>{{ member?.name }} Details</h2>
3    <label>Email:</label>{{ member?.email }} <label>Role:</label
4    >{{ member?.roleDesc }} <label>Gender:</label>{{ member?.genderDesc }}
5
6    <button (click)="handleEdit()">Edit member</button>
7    <button (click)="onDelete.emit(this.id)">Delete member</button>
8  </div>
```

Anexo S – Componente *MemberDisplayer* em React

```
5  const MemberDisplayer = ({ id, onDelete }) => {
6    const history = useHistory();
7    const [member, setMember] = useState(null);
8
9    useEffect(() => {
10     const getMember = async () => {
11       const member = await membersApi.getMember(id);
12       setMember(member);
13     };
14     getMember();
15   }, [id]);
16
17   const handleEdit = () => {
18     history.push('/members/edit/' + id);
19     return;
20   };
21
22   return (
23     <div>
24       <h2>{member?.name} Details</h2>
25       <label>Email:</label>
26       {member?.email}
27       <label>Role:</label>
28       {member?.roleDesc}
29       <label>Gender:</label>
30       {member?.genderDesc}
31       <button onClick={handleEdit}>Edit member</button>
32       <button onClick={() => onDelete(id)}>Delete member</button>
33     </div>
34   );
35 };
```

Anexo T – Variáveis utilizadas no formulário de *login* da aplicação React

```
10  const defaultValues = {
11    email: null,
12    password: null,
13  };
14
15  const resolver = yupResolver(
16    yup.object().shape({
17      email: yup.string().email('Invalid format !').required('Required !'),
18      password: yup.string().required('Required !'),
19    })
20  );
21
```

Anexo U – Número de linhas de código implementadas para os componentes das *features* em Angular e React

	MemberDisplayer	MemberForm	MemberList	TaskForm	Total
Angular (TS)	42	76	12	68	198
Angular (HTML)	27	79	5	63	174
React (JSX)	57	102	16	91	266

Anexo V – Número de linhas de código implementadas para os componentes das *features* em Angular e React

	ErrorMessage	Loading	Table	Total
Angular (TS)	12	8	31	51
Angular (HTML)	1	3	19	23
React (JSX)	8	12	50	70

Anexo W – Ficheiro de testes do componente *LoginPage* na aplicação Angular

```
describe('LoginPageComponent', () => {
  let component: LoginPageComponent;
  let fixture: ComponentFixture<LoginPageComponent>;
  const authServiceSpy: jasmine.SpyObj<AuthService> = jasmine.createSpyObj(
    'AuthService',
    ['login']
  );

  beforeEach(async(() => {
    TestBed.configureTestingModule({
      declarations: [LoginPageComponent],
      imports: [
        FormsModule,
        ReactiveFormsModule,
        RouterTestingModule,
        HttpClientTestingModule,
      ],
      providers: [{ provide: AuthService, useValue: authServiceSpy }],
    });
  }));

  beforeEach(() => {
    authServiceSpy.login.and.returnValue(Promise.resolve({ email: 'rui' }));
    fixture = TestBed.createComponent(LoginPageComponent);
    component = fixture.componentInstance;
    fixture.detectChanges();
  });

  afterEach(() => {
    fixture.destroy();
  });

  it('should not call login with empty form', () => {
    authServiceSpy.login.and.returnValue(Promise.resolve({ email: 'rui' }));
    const button = fixture.debugElement.query(By.css('button[type="submit"]'));
    button.nativeElement.click();
    expect(authServiceSpy.login).not.toHaveBeenCalled();
  });

  it('should not call login with invalid email', async () => {
    authServiceSpy.login.and.returnValue(Promise.resolve({ email: 'rui' }));
    const email = fixture.debugElement.query(
      By.css('input[formControlName="email"]')
    ).nativeElement;

    email.value = 'invalid email';
    email.dispatchEvent(new Event('input'));
    const password = fixture.debugElement.query(
      By.css('input[formControlName="password"]')
    ).nativeElement;
    password.value = 'testpassword';
    password.dispatchEvent(new Event('input'));

    const button = fixture.debugElement.query(By.css('button[type="submit"]'));
    button.nativeElement.click();
    expect(authServiceSpy.login).not.toHaveBeenCalled();
  });

  it('should call login with valid form', () => {
    authServiceSpy.login.and.returnValue(Promise.resolve({ email: 'rui' }));
    const email = fixture.debugElement.query(
      By.css('input[formControlName="email"]')
    ).nativeElement;

    email.value = 'rui@mail.com';
    email.dispatchEvent(new Event('input'));
    const password = fixture.debugElement.query(
      By.css('input[formControlName="password"]')
    ).nativeElement;
    password.value = 'testpassword';
    password.dispatchEvent(new Event('input'));

    const button = fixture.debugElement.query(By.css('button[type="submit"]'));
    button.nativeElement.click();
    expect(authServiceSpy.login).toHaveBeenCalled();
  });
});
```

Anexo X – Ficheiro de testes do componente *LoginPage* na aplicação React

```
8   describe('LoginPage', () => {
9     let authApiLoginMock;
10
11    beforeEach(() => {
12      authApiLoginMock = jest.spyOn(authApi, 'login');
13      authApiLoginMock.mockRejectedValue(new Error('abort'));
14    });
15
16    it('should not call login with invalid form', async () => {
17      const { findByTestId } = render(<LoginPage />);
18
19      const button = await findByTestId('login');
20      fireEvent.click(button);
21      expect(authApiLoginMock).not.toHaveBeenCalled();
22    });
23
24    it('should not call login with invalid email', async () => {
25      const { findByTestId } = render(<LoginPage />);
26
27      const email = await findByTestId('email');
28      fireEvent.change(email, { target: { value: 'invalid email' } });
29
30      const password = await findByTestId('password');
31      fireEvent.change(password, { target: { value: 'password' } });
32
33      const button = await findByTestId('login');
34      fireEvent.click(button);
35      expect(authApiLoginMock).not.toHaveBeenCalled();
36    });
37
38    it('should call login with valid form', async () => {
39      const { findByTestId } = render(<LoginPage />);
40      const email = await findByTestId('email');
41      const password = await findByTestId('password');
42      const button = await findByTestId('login');
43
44      fireEvent.change(email, { target: { value: 'rui@mail.com' } });
45      fireEvent.change(password, { target: { value: 'password' } });
46      fireEvent.click(button);
47
48      wait(() => {
49        expect(authApiLoginMock).toHaveBeenCalled();
50      });
51    });
52  });
```