INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA

Framework de Geração de Código

TELMO RODRIGUES COELHO
Outubro de 2021

POLITÉCNICO
DO PORTO

Framework de Geração de Código

Telmo Rodrigues Coelho

Dissertação para obtenção do Grau de Mestre em Engenharia Informática, Área de Especialização em Engenharia de Software

Orientador: Ricardo Almeida

Co-orientador: Paulo Oliveira

Júri:

Presidente:

Dr. Nuno Escudeiro, Professor, DEI/ISEP

Vogais:

Dr. Constantino Martins, Professor, DEI/ISEP

Porto, outubro 2021

Dedicatória

A realização deste trabalho é dedicado especialmente ao meus pais que tornaram possível a realização de mais uma etapa na minha vida e que sempre acreditaram no meu esforço, apoiando-me em todo o percurso académico.

Telmo Coelho

Resumo

Este projeto foi desenvolvido num ambiente empresarial, onde o autor teve a possibilidade de realizar um estágio, numa organização designada Celfocus. Este projeto está inserido no ramo de geração de código, e este baseia-se na ideia de reaproveitamento de código.

A presente dissertação tem como objetivo documentar todo o processo de implementação de um protótipo de framework de geração e *React Journeys,* para que estas possam posteriormente ser integradas em projetos da organização. O objetivo é criar um padrão para diversos tipos de componentes utilizadas na organização, para que seja reduzida a intervenção dos desenvolvedores. É permitido ao utilizador configurar e definir as características das diversas componentes React que pretende gerar, através de uma interface gráfica.

Para além da geração de *React Journeys*, é permitido aos utilizadores o download de um ficheiro JSON com a informação relativa a uma *React Journey*, para que esta possa, mais tarde, ser importada novamente para o sistema, dando a possibilidade ao utilizador de a editar novamente.

O protótipo de framework de geração de código é composto por duas grandes aplicações, a aplicação do *frontend*, desenvolvida em ReactJs, e a aplicação responsável pela geração de código desenvolvida em NodeJs.

Em todo o processo de desenvolvimento desta aplicação, foram utilizados todos os processos de engenharia de software, nos quais se destacam a análise de requisitos e o desenvolvimento de todo o design do respetivo sistema.

Palavras-chave: Gerador de Código, Reaproveitamento de Código, React Journeys, ReactJs, NodeJs

Abstract

This project was developed in a business environment, where the author had the possibility of doing an internship, in an organization called Celfocus. This project is inserted in the code generation branch, and this one is based on the idea of code reuse. This dissertation aims to document the entire process of implementing a prototype of a React Journeys generator framework, so that they can later be integrated into the organization's projects. The objective is to create a standard for different types of components used in the organization, so that the intervention of developers is reduced. Users can configure and define the characteristics of the various React components they want to generate, through a graphical interface.

In addition to generating React Journeys, users can download a JSON file with information relating to a React Journey, so that it can later be imported back into the system, giving the user the possibility to edit it again.

The code generation framework prototype is composed by two great applications, the frontend application, developed in ReactJs, and the application responsible for code generation developed in NodeJs.

Throughout the development process of this application, all software engineering processes were used, in which the requirements analysis and the development of the entire design of the respective system stand out.

Keywords: Code Generator, Code Reuse, React Journeys, ReactJs, NodeJs

Agradecimentos

O primeiro agradecimento é dirigido aos meus pais, que sempre me apoiaram em tudo o que precisei, estando sempre dispostos e disponíveis para me ajudarem em todos os momentos. Para além disso, foram o meu maior suporte nos momentos mais difíceis do meu percurso académico, acreditando sempre nas minhas capacidades.

Agradeço também à instituição (ISEP) em especial ao departamento (DEI) pela passagem de conhecimento durante todo o percurso na Licenciatura no Mestrado em Engenharia de Software.

Agradeço ao meu orientador, Ricardo Almeida, e coorientador, Paulo Oliveira, por todo o suporte prestado. A sua disponibilidade para a revisão do trabalho, todas as críticas e sugestões, fornecidas pelo mesmo, foram cruciais para o resultado e qualidade do projeto.

Por fim, um agradecimento especial à Celfocus, ao meu supervisor Hugo Santos e a toda a sua equipa pelo tempo dedicado, e pelo conhecimento transmitido no decorrer do estágio curricular

Índice

Lista de Figuras

Lista de Tabelas

Acrónimos e Símbolos

Lista de Acrónimos

AHP	Análise Hierárquica de Processos
API	*Application Programming Interface*
DOM	*Document Object Model*
DSL	*Domain Specific Language*
HTML	*Hypertext Markup Language*
JSON	*JavaScript Object Notation*
MDE	*Model-Driven Engineering*
NCD	*New Concept Development*
SWOT	*Strengths*, *Weaknesses*, *Opportunities* and *Opportunities*
RBT	*Rule-Based Transformation*
RTCG	*Runtime Code Generation*
TBCG	Template-Base Code Generation
UML	*Unified Modelling Language*
XML	*Extensible Markup Language*

1 Introdução

No corrente capítulo, é descrita a visão geral do projeto "Framework de Geração de Código". Numa primeira fase, é descrito o contexto em que o projeto está inserido. Após a contextualização do projeto, é descrito o problema, apresentando também os objetivos associados. Neste capítulo, é ainda detalhada a abordagem utilizada para a resolução do problema e os resultados esperados com o desenvolvimento desta framework. Por fim, esta secção contém uma breve descrição de como o presente documento está estruturado.

1.1 Enquadramento/Contexto

O presente projeto foi desenvolvido em ambiente empresarial, numa organização denominada de Celfocus, onde o autor, pôde adquirir novas experiências e métodos de trabalho. A Celfocus é uma organização que desenvolve e implementa soluções tecnológicas, com o intuito de auxiliar os seus clientes na sua transformação digital. Atualmente, a Celfocus, é uma organização que se dedica ao desenvolvimento de soluções informáticas destinadas ao setor das telecomunicações, contendo vários projetos provedores de serviços nesse setor, maioritariamente para a Vodafone, em várias partes do mundo (Europa, África e médio oriente).

A geração de código tem cada vez mais um impacto significativo no domínio do desenvolvimento, esta, pode ser, uma abordagem muito importante para impulsionar a produtividade no processo de desenvolvimento de *software.* Com o crescimento progressivo da organização, torna-se cada vez mais relevante um sistema capaz de gerar código-fonte de qualidade.

Este projeto surge dessa mesma necessidade, e descreve todo o processo efetuado para a conceção de um protótipo de uma *framework* de geração de código, capaz de satisfazer as necessidades da organização.

1.2 Problema

A necessidade da realização deste projeto, surgiu devido à inexistência de um sistema capaz de gerar código-fonte de qualidade. A implementação de um sistema, responsável pela geração de um código-fonte robusto e escalável, pode ter um impacto significativo no funcionamento da organização em questão, visto que, esta pretende reduzir os custos de esforço das equipas de desenvolvimento.

Deste modo, a organização sentiu a necessidade de obter um software capaz de, ao iniciar novos projetos, gerar código-fonte de qualidade, garantindo consistência, portabilidade e escalabilidade.

O foco da criação desta ferramenta é a geração de componentes de *User Interface* (UI), uma vez que a organização utiliza muitos processos, onde algumas das componentes de UI podem ser reutilizadas. Para além disso, a organização tem como objetivo migrar alguns dos seus projetos para ReactJs[1]. Para isso, a organização sentiu a necessidade de uma ferramenta, que através de *templates* predefinidos, possa gerar a base das diversas componentes tendo apenas que adicionar a sua lógica. Mas para além da geração de componentes UI, esta ferramenta deve ser vista como algo mais genérico, visto que, futuramente devem ser integrados novos módulos de geração.

1.3 Objetivos

Face ao problema apresentado, o objetivo desta dissertação é analisar algumas ferramentas de geração de código para ReactJs, uma vez que, se pretende desenvolver um protótipo de uma framework com a capacidade de gerar React Journeys, que basicamente são um conjunto de componentes React. Para além disso, deve ser assegurada a escalabilidade do sistema, uma vez que, futuramente devem ser integrados novos módulos de geração (como por exemplo, geração de microsserviços).

Para isso, espera-se uma abordagem do tipo Spring Initializr [1], em que basicamente o utilizador, que neste caso, serão os *developers* da organização, terão de selecionar um conjunto de escolhas, através de uma interface gráfica, para gerar algum código. A Figura 1, representa a interface gráfica do Spring Initializar.

[1] https://reactjs.org/

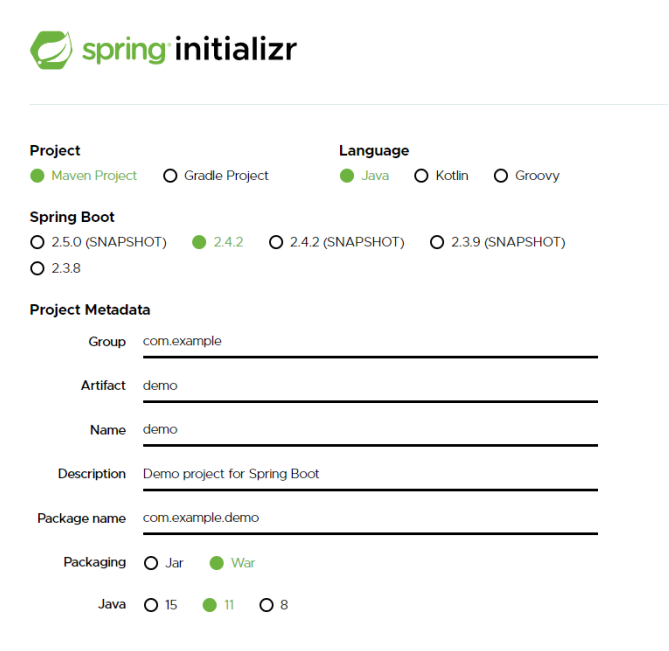


Figura 1 – Abordagem do Spring Initializr[2]

A interface desenvolvida deve ser simples e intuitiva, para que os utilizadores consigam facilmente familiarizarem-se com o sistema desenvolvido, bem como, com as suas funcionalidades.

Após o desenvolvimento do protótipo, pretende-se também realizar um caso de estudo que permita perceber o grau de satisfação dos futuros utilizadores.

1.4 Abordagem

A proposta visa o desenvolvimento de uma *framework* capaz de gerar um código base robusto capaz de suportar as necessidades diárias das equipas de desenvolvimento.

Numa primeira fase, deve ser realizado um estudo sobre a área de geração de código, com o objetivo de adquirir o conhecimento necessário para que, a integração com o problema a solucionar, seja mais facilitada.

Posteriormente, deve ser realizado o levantamento dos requisitos junto do cliente. Esta fase assume um papel fundamental no decorrer do projeto, pois todos os requisitos definidos serão utilizados para a validação do trabalho realizado.

Seguidamente, deve ser feita uma sistematização do problema de forma mais detalhada e perceber, no âmbito da organização, o processo que melhor se adequa para a criação de uma *framework* de geração de código.

[2] Retirado de: https://start.spring.io/

Para além disso, deve também ser realizada uma análise das tecnologias e abordagens existentes que melhor se adequam ao problema apresentado.

Na fase seguinte pretende-se a conceção e o desenho de uma arquitetura que permita a modularidade do sistema no que diz respeito à integração de novos componentes, tendo em conta as especificidades do negócio. Após o desenho, deverá ser implementada o protótipo de *framework* de geração de código, com a arquitetura previamente definida, garantindo a sua modularidade no que diz respeito à adição de novos componentes.

Posteriormente, deve ser realizado um caso de estudo que permita demonstrar e validar a utilidade do protótipo de *framework* desenvolvido no que diz respeito ao grau de satisfação dos possíveis utilizadores.

Numa última fase, os resultados provenientes do estudo devem ser analisados, para que, possam ser retiradas conclusões, no que diz respeito, ao impacto que a framework desenvolvida tem nas equipas de desenvolvimento.

Todo este processo é iterativo e incremental, por este mesmo motivo é que se denota uma diferença nos objetivos apresentados na primeira iteração. Posto isto, pode-se encontrar no Anexo B a pesquisa elaborada que acabou por não ser utilizada.

1.5 Estrutura da dissertação

Concluídos os pontos anteriores que têm como objetivo contextualizar e dar a perceber ao leitor o objetivo do trabalho realizado, é importante perceber a estrutura da dissertação.

No capítulo 2 encontra-se descrito o estado da arte, essencial para que se perceba quais as tecnologias mais utilizadas no contexto de geração de componentes React. Neste capítulo encontram-se também alguns trabalhos relacionados com a área de negócio do projeto desenvolvido.

O capítulo 3 descreve a análise de valor, onde é identificada a oportunidade que a organização pretende perseguir. Posteriormente, encontra-se uma breve análise de oportunidade, onde é realizada uma análise *SWOT (Strengths, Weaknesses, Opportunitues, Threats),* com o intuito de perceber até que ponto vale a pena perseguir a oportunidade identificada. Para além disso, encontra-se também descrita a proposta de valor, demostrando a correspondência entre as necessidades da organização e os benefícios do sistema desenvolvido. Por fim, ainda neste capítulo, é realizado o método AHP, que é um método de apoio à decisão.

No capítulo 4 encontra-se uma descrição detalhada da análise da solução a desenvolver. É neste capítulo onde é apresentado o domínio do problema, bem como os requisitos funcionais e não funcionais do problema apresentado anteriormente. Por fim, é apresentado todo o desenho da aplicação desenvolvida.

O capítulo 5 encontra-se descrita a implementação realizada para a resolução do problema desenvolvido.

O capítulo 6 destina-se à avaliação do sistema desenvolvido, onde foi realizado um questionário de satisfação do utilizador. Posteriormente, é feita uma interpretação dos resultados obtidos por parte do autor.

Por fim, no capítulo 7, são efetuadas algumas conclusões sobre o projeto implementado. Aqui são descritas as limitação da solução desenvolvida, possíveis pontos de melhoria e trabalho futuro no projeto.

2 Contexto e Estado da Arte

Neste capítulo, pode encontrar-se uma breve descrição teórica dos conceitos essenciais para a compreensão do âmbito em que o negócio está inserido, nomeadamente Gerador de código. Neste capítulo encontra-se também o estado da arte relativo ao tema da tese, onde se descrevem algumas das abordagens existentes para a construção de geradores de código, bem como, alguns geradores de código existentes tanto para componentes React como para microsserviços.

2.1 Gerador de Código

Como um dos principais objetivos da presente dissertação é a criação de uma *framework* de geração de código, é necessário entender o conceito de gerador de código.

Como o próprio nome indica, os geradores de código são responsáveis por produzir, automaticamente, código ou parte dele, a partir de especificações abstratas de um programa [2]. Essas especificações, que determinarão quais as partes de código a serem geradas, encontram-se descritas numa *domain specific language* (DSL).

Normalmente, as DSL fornecem notações de modulação que são representativas de alguns conceitos de domínio. Depois de escrita a DSL, o gerador é responsável pela codificação da semântica operacional da mesma, interpretando todas as especificações da DSL e gerando o código-fonte esperado. A Figura 2 demonstra de que maneira o processo de geração de código é efetuado.

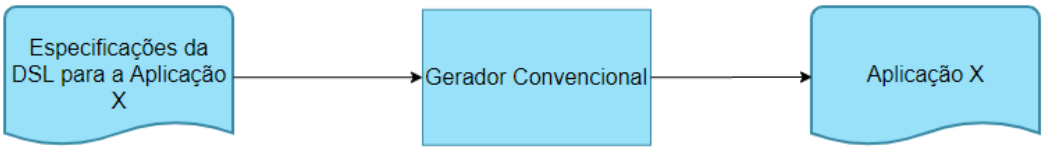


Figura 2 – Processo de Geração de Código

2.1.1 Impacto da utilização de geradores de código

Os geradores de código têm como objetivo elevar a programação a um nível de abstração mais alto. Com o aumento da abstração e com a reutilização consecutiva do código específico de domínio, os geradores possibilitam o aumento da produtividade de software[2].

Muitas organizações obtiveram sucesso e demonstraram, que a utilização de DSLs e de geradores de código, durante o processo inicial de desenvolvimento de software, têm um impacto significativo não só no custo de esforço das suas equipas de desenvolvimento, como também, conseguem reduzir o tempo de desenvolvimento das suas aplicações[2][3]. Estes dois fatores verificam-se, devido ao facto de que, as equipas de desenvolvimento apenas têm que escrever manualmente a parte do código que não é gerado.

Os geradores de código, com a diminuição da introdução de código pelos desenvolvedores, também implicam que as aplicações desenvolvidas sejam menos suscetíveis a erros, contribuindo, desta maneira para um código mais robusto e escalável.

Outra vantagem do uso de geradores de código é que, em muitos casos, as especificações das DSL desenvolvidas podem ser compreendidas pelos seus utilizadores, fazendo com que em muitos dos casos estes utilizadores possam desempenhar um papel na análise dos requisitos. Em casos muito específicos, os geradores de código fornecem a possibilidade a não desenvolvedores que construam aplicações ou pelo menos parte delas.

2.1.2 Baixa taxa de adoção

Apesar dos geradores de código, demostrarem alguns benefícios no processo de desenvolvimento, por vezes, as equipas de desenvolvimento e as organizações em geral optam por não os adotar.

Isto verifica-se em domínios menos estáveis e em evolução. Em domínios deste tipo, é mais difícil chegar a formalizações que produzam soluções de geração de código sofisticadas. Neste tipo de situações, geralmente, a DSL é menos perfeita e desconsidera detalhes importantes de uma aplicação. Para além disso, os clientes continuam a descobrir novos requisitos, que acabam por não estar descritos na DSL desenvolvida [2].

Mesmo com vários anos de investigação e pesquisa sobre geradores de código, estes, por sua vez, ainda não pertencem à prática, no que diz respeito ao processo de desenvolvimento, da maior parte das organizações. A constante evolução e reutilização do software exige, que as alterações ao mesmo seja feita de maneiras flexíveis. Deste modo, no âmbito descrito anteriormente, deduz-se que a adoção de geradores de código no processo de desenvolvimento é difícil de aplicar, pois têm que ser realizadas, constantemente, alterações manuais ao gerador utilizado.

2.2 Abordagens para a geração de código

O presente subcapítulo, visa analisar algumas das abordagens existentes na literatura para o desenvolvimento de ferramentas de geração de código. O conhecimento destas abordagens é fundamental para a conceção de uma ferramenta de geração de código. Para tal foram analisadas as seguintes abordagens: Runtime Code Generation [4], Template-Based Code Generation [5] e Rule-Based Transformation [3].

As subsecções seguintes, são destinadas para descrever essas abordagens, para uma melhor perceção do processo de geração de código de cada uma.

2.2.1 Runtime Code Generation

Esta técnica entrou em uso, em meados do ano 1940, e basicamente a Runtime Code Generation (RTCG), consiste na adição dinâmica de código, ao fluxo de instruções de um programa em execução[4]. Mas esta técnica, ao longo dos anos, tem caído em desuso, devido às várias evoluções tecnológicas tanto a nível de software como hardware, e a sua implementação *ad-hoc* fez com que esta técnica não fosse capaz de acompanhar esta evolução [4].

Uma implementação dinâmica de RTCG, compila dinamicamente as expressões convertendo-as em funções específicas, chamando essas funções de acordo com um determinado *input.* Por sua vez, para uma implementação estática de RTCG, uma implementação de código estático converte a expressão num estado intermédio, sendo este, posteriormente, interpretado por um interpretador compilado estaticamente.

Esta técnica é ainda utilizada para num grande número de sistemas de pesquisa e produção. É funcionalmente necessário, para compiladores incrementais, linkers dinâmicos e debuggers. O RTCG tem também um grande impacto no melhoramento do desempenho de máquinas virtuais de alto nível, sistemas interativos que exigem bom tempo de resposta e em operações de consulta à base de dados. Para além disso o RTCG, permite que um programa se adapte dinamicamente a implementações específicas dentro de uma família de arquiteturas.

2.2.2 Template-Based Code Generation

A Template-Base Code Generation (TBCG), surge, pela primeira vez em meados da década de 1990 [5]. Esta abordagem, para a construção de geradores de código, tem como principal objetivo, facilitar os esforços realizados pelos programadores na sua rotina.

A criação de geradores de código, com base nesta abordagem, utiliza os *templates* para a reutilização do código, seguindo o princípio, de que apenas é necessário escrever uma vez para que esse código possa ser reutilizado. Este conceito foi muito aplicado para gerar páginas web, visto que muitas UIs, a serem desenvolvidos, têm bastantes semelhanças [6]. Esta técnica é, também, muito popular em *Model-Driven Engineering* (MDE), visto que, ambas enfatizam a abstração e a automação.

Para a abordagem TBCG, são identificadas cinco componentes: o *Runtime input*, os *templates*, o *output*, o *template engine* e o *design-time input,* que corresponde à meta-informação na qual a lógica de geração do template depende.

O TBCG é uma técnica de síntese que usa *templates*, com o objetivo de gerar código-fonte, chamado *Output*. A Figura 3, é uma representação gráfica que esquematiza este processo.

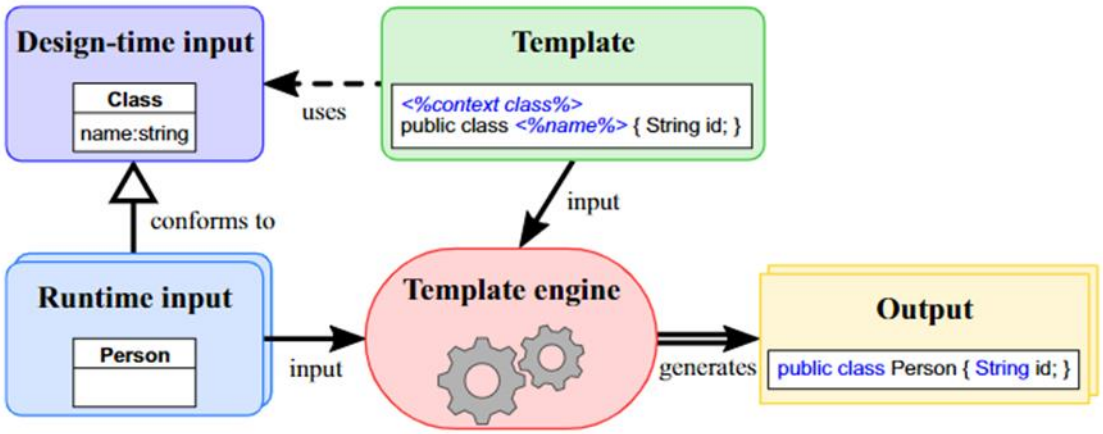


Figura 3 – Componentes do TBCG[3]

Um *template* é um modelo base que permite criar um *Output* textual que descreve. Estes *templates*, são constituídos por duas partes: uma parte estática que, basicamente, são fragmentos de texto que aparecem exatamente como estão no *Output*; e uma parte dinâmica que varia mediante a lógica de geração do meta código. Estes *templates* são executados pelo *Template Engine*, e este mecanismo, é responsável por identificar as partes dinâmicas e substituí-las por texto estático, de acordo com o *Runtime Input* fornecido.

O *Design-time input* é a componente responsável por conter a meta-informação, com a qual, o *Runtime input* está em conformidade. A parte dinâmica do *template* depende do *Design-time*

[3] Retirado de: https://www.sciencedirect.com/science/article/abs/pii/S1477842417301239

input, para consultar o *Runtime input*. Apenas desta maneira é possível filtrar as informações recuperadas e realizar expansões iterativas.

Resumidamente, o processo do TBCG começa com um *Runtime Input*, que é definido através do *Design-time input*. Este *Runtime input*, é aplicado ao template, produzindo desta maneira o *Output* esperado.

2.2.3 Rule-Based Transformation

Uma outra abordagem existente na literatura, para criação de um gerador de código, é a Rule-Based Transformation (RBT)[3]. Nesta abordagem, previamente, define-se um conjunto de regras que são responsáveis por retratar de que maneira cada elemento representado na linguagem de origem é transformado num elemento da linguagem esperada.

Num contexto real, para uma transformação, utilizando esta abordagem, o mecanismo de transformação é responsável por conter, interpretar e processar as regras previamente definidas, aplicando-as ao modelo de entrada descrito na linguagem de origem, transformando-o no modelo de saída esperado. Este tipo de abordagem é muitas vezes aplicado para a realização de *model-to-model Transformations* (transformações modelo para modelo) [7]. A Figura 4, demonstra sucintamente como este processo é realizado.

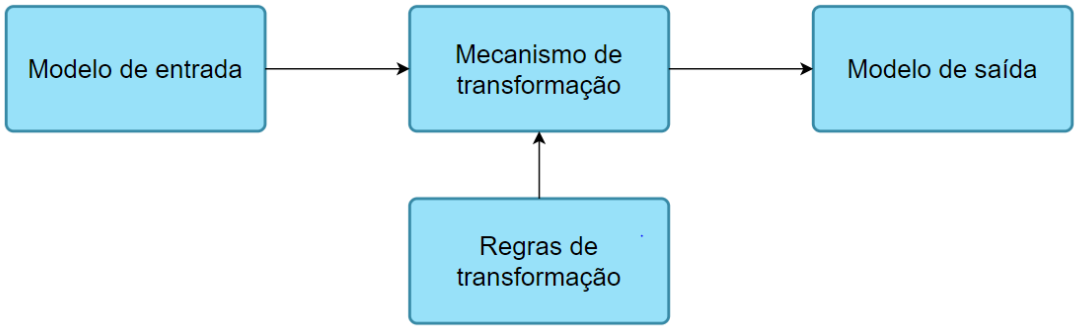


Figura 4 – Processo RBT

Quando estamos perante uma transformação bastante complexa, esta, por sua vez, é divida em transformações mais pequenas, com o objetivo de tornar todo este processo mais simples, e consequentemente, mais fácil de gerir.

Neste tipo de abordagens, em vez de expressar diretamente o modelo de entrada ou qualquer uma das representações intermediárias, ao longo das várias transformações, no correspondente da linguagem esperada, é apenas apresentada uma estrutura final da representação do modelo na linguagem esperada.

Uma das principais vantagens da utilização desta abordagem, é continuar a ter acesso à representação abstrata do código mesmo depois de todo o processo de geração. Isto permite, que este código possa ser reutilizado noutras etapas de processamento.

2.3 Frameworks JavaScript

Como um dos objetivos do presente projeto é a criação de uma interface simples intuitiva, é necessário que se realize um estudo das principais ferramentas para o desenvolvimento de frontend. O JavaScript tornou-se umas das principais linguagens de programação para o desenvolvimento Web, sendo esta uma das linguagens de programação mais populares da última década [8]. Por esse mesmo motivo, realizou-se um estudo de duas das principais *frameworks* JavaScript, sendo elas ReactJs e AngularJS.

2.3.1 ReactJs

Tendo em conta que cada vez mais a organização, Celfocus, adota o ReactJs[4] como tecnologia *front-end* para o desenvolvimento dos seus projetos, surgiu a necessidade de acelerar os processos de desenvolvimento recorrendo a um protótipo de geração de código. De seguida, é apresentado as conclusões do estudo realizado sobre ReactJs.

O ReactJs é uma biblioteca JavaScript implementada para desenvolver *componentes user interface* (UI) e recentemente tem ganho cada vez mais influência, tornando-se na ferramenta de JavaScript para a criação de frontend [9].

O React basicamente é utilizado para o desenvolvimento de aplicações, grandes e complexas, baseadas em Web, tornando todo o processo de criação de UIs simples e eficiente. Uma vez que, a atualização dos componentes só é realizada quando estas sofrem qualquer tipo de alteração. O servidor, utilizando NodeJs[5], é responsável pela renderização do React. Neste ambiente de desenvolvimento as componentes de React são utilizadas como View(V), numa arquitetura Model-View-Controller (MVC).

O componente é o principal bloco de construção de uma aplicação em React. Todas as UIs, são projetadas através da construção de uma árvore, responsável por mapear os vários componentes desenhados. O virtual DOM, que se encontra descrito na subsecção 2.1.1, é gerado por um método designado render() que se encontra em todas as componentes de React desenhadas. Posteriormente este Virtual DOM é convertido no DOM[6] do navegador.

Para a construção da árvore de componentes, como uma combinação de vários nós XML, o React utiliza uma linguagem designada por JavaScript XML (JSX). O JSX tem um papel fulcral na simplificação do processo de coordenação de eventos e na definição de propriedades, como atributos XML.

As próximas subsecções têm como principal objetivo ilustrar e detalhar os principais motivos que fazem do React, uma tecnologia cada vez mais influente no processo de desenvolvimento

[4] https://reactjs.org/
[5] https://nodejs.org/en/
[6] https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction

web, mostrando também de que maneira a sua utilização tem impacto no trabalho diário das equipas de desenvolvimento [9].

2.3.1.1 Virtual Document Object Model

Uma das principais características do React, é que este, fornece um DOM [9][10] bastante eficiente e que ao mesmo tempo é bastante leve. O React não interage diretamente com o DOM gerado pelo browser, mas sim com um DOM que se encontra armazenado em memória. Isto faz com que, a aplicação desenvolvida tenha um melhor desempenho sendo muito mais eficiente.

A maior parte das aplicações web que não utilizam React, faz interação direta com o DOM do navegador, o que faz com que a manipulação da DOM *tree*, seja direta e processada toda de uma vez, quando algum evento é despoletado. Ou seja, quando se está perante um grande aglomerado de dados que deve ser modificado, o desempenho é bastante afetado. Para resolver este problema, o React faz uso de uma virtual DOM.

O processo de funcionamento desta virtual DOM, é semelhante ao processo do DOM gerado pelo navegador, a grande diferença entre os dois, é que o virtual DOM é armazenado na memória. Sempre que alguma alteração é realizada sobre os conteúdos das páginas, essas alterações, primeiramente são registadas no virtual DOM, sendo depois comparado, com o DOM do navegador. Depois de encontradas as diferenças, apenas o conteúdo que não coincide é substituído na DOM do navegador, tornando todo este processo muito mais eficiente, uma vez que não é necessário renderizar todos os conteúdos novamente quando um evento é despertado [9].

2.3.1.2 JavaScript XML

Uma outra característica do React, que torna a sua utilização muito mais fácil, é a existência de uma linguagem designada por JSX [9][10], esta por sua vez, tem características muito idênticas às de XML. Para o desenvolvimento de uma aplicação com base em React não é obrigatório o uso desta linguagem. No entanto, esta linguagem é muito utilizada entre a comunidade de desenvolvedores, uma vez que, facilita não só o processo de escrever marcações dentro das componentes React como também facilita a ligação entre eventos.

Por estes mesmo motivos, é que a utilização desta linguagem, ao longo do tempo, se está a tornar cada vez mais popular entre a comunidade, pois a natureza do ser humano é optar por técnicas mais agradáveis e menos complexas [9].

2.3.1.3 Fluxo Unidirecional de dados

O React foi desenhado com o objetivo de suportar um fluxo de dados unidirecional [9]. Ao desenvolver uma aplicação em React as equipas de desenvolvimento devem preocupar-se em definir as componentes filho dentro das componentes pai de ordem superior.

O React foi construído sobre este principio, uma vez que, as componentes devem ser imutáveis e os seus dados não devem poder ser alterados em nenhuma circunstância. Deste modo, os dados apenas podem surgir num sentido, nunca no sentido contrário. Apenas desta maneira, é

que é possível manter e assegurar a sincronização entre as componentes pais e filhos correspondentes.

Isto é mais um motivo que torna esta tecnologia umas das mais utilizadas no processo de desenvolvimento de aplicações Web interativas. Se alguma alteração for realizada sobre dados upstream, as componentes que têm acesso ou utilizam esses mesmos dados são automaticamente renderizados, com o objetivo de efetuar as alterações registadas.

2.3.2 AngularJS

O AngularJS[7] é uma *framework* JavaScript *Open Source* (código aberto), que tem como objetivo ajudar os desenvolvedores na criação e desenvolvimento de aplicações *Single Page* (página única). O AngularJS é construído em cima do JavaScript e visa tornar todo o processo de desenvolvimento de páginas web muito mais simples.

A ideia por trás do uso do AngularJS é tornar as aplicações web modulares e consequentemente mais fáceis de manter. Esta é uma *framework* de apoio ao desenvolvimento de aplicações web com base no padrão Model-View-Controller (MVC), tornando todos os processo de desenvolvimento, manutenção e testagem mais fáceis [11].

O AngularJS é uma ferramenta de auxilia na criação de páginas web baseadas em HTML, CSS e JavaScript. Para além disso, permite aos desenvolvedores a criação de elementos personalizados de *Document Object Model (DOM).*

Uma das grandes vantagens do AngularJS é a sincronização automática entre o modelos e as componentes de visualização, esta sincronização dos dados é conhecida como *Two-way Binding* e é um recurso exclusivo do AngularJS.

2.3.2.1 Two-way Binding

A presente subsecção descreve de forma sucinta o recurso exclusivo do AngularJS, designado por *Two-way Binding.*

A vinculação dos dados no AngularJS é feita através da sincronização entre o modelo e as componentes de visualização. Quando os dados do modelo sofrem alterações, as componentes de visualização refletem essa mudança. Ao contrário o processo é semelhante, quando as componentes de visualização sofrem alterações, o modelo também é atualizado. Todo este processo é imediato e automático.

2.3.3 ReactJs VS AngularJS

Com um intuito de perceber qual a melhor framework de JavaScript, *Anurag Kumar* e *Ravi Kumar Singh* [11] realizaram uma breve análise comparativa entre ReactJs e AngularJS. Para tal

[7] https://angularjs.org/

elaboraram uma tabela comparativa, Tabela 1, entre as duas *frameworks* segundo alguns atributos.

Tabela 1 – Comparação de AngularJS e ReactJs pelos diferentes atributos

ATRIIBUTO	**ANGULARJS**	**REACTJS**
DOM	Regular DOM	Virtual DOM
LEARNING CURVE	Alta	Baixa
PACKAGING	Fraca	Forte
ABSTRACTION	Fraca	Forte
DEBUGGING	Bom para HTML/ Mau para JS	Mau para HTML/ Bom para JS
DEBUG LINE NO	Não	Sim
UNCLOSED TAG MENTIONED ?	Não	Sim
FAILS WHEN?	Tempo de execução	Tempo de compilação
BINDING	Duas direções	Unidirecional
TEMPLATING	Em HTML	Em JSX
COMPONENT MODEL	Fraco	Médio
BUILDING MOBILE?	Ionic Framework	React Native
MVC	Sim	Apenas componente de visualização
RENDERING	Lado do Cliente	Lado do Servidor

Após uma breve análise da tabela pode-se concluir que o React possuir uma performance muito mais eficiente que o AngularJS, uma vez que, o React faz uso de um DOM virtual. O DOM virtual permite ao React, quando alguma das componentes sofre alterações, renderizar apenas essas alterações. Enquanto que o Angular, que faz uso do DOM do navegador, tem que renderizar todas as componentes quando alguma delas sofre alterações.

Para além disso, pode-se verificar que o React tem uma curva de aprendizagem mais baixa que o AngularJS, dando a possibilidade aos desenvolvedores se familiarizarem-se mais facilmente com a framework.

Por todos estes motivos, entre as *frameworks* de JavaScript analisadas, optou-se pelo desenvolvimento do frontend em ReactJs.

2.4 Geradores para React

A presente secção reúne alguns geradores de código implementados no mercado, que são utilizados para geração de componentes React.

As subsecções seguintes destinam-se, a descrever com algum detalhe de que maneira esses geradores funcionam, para se perceber se algum se enquadra no problema a desenvolver, contendo também uma análise crítica para cada um onde são descritas as suas vantagens e

desvantagens. Para além disso, no final deste subcapítulo, encontra-se uma análise crítica entre os geradores apresentados.

2.4.1 Generate-React-Cli

O Generate-React-Cli (GRC) [12], é um gerador *Open-Source* desenhado especificamente para a geração de componentes React. O seu principal objetivo é ajudar as equipas de desenvolvimento, aumentando a produtividade no processo de construção de aplicações React, uma vez que, permite que os desenvolvedores parem de copiar e colar, e estarem constantemente a renomear os ficheiros, sempre que querem criar um novo componente. Este gerador tem por base *templates*, é através deles que o código é gerado. O utilizador indica o tipo de componente que quer gerar, tendo este que estar previamente descrito no ficheiro de configuração, e com base nos *templates* indicadas nesse ficheiro, para esse tipo de componente, o código é gerado.

Quando se corre pela primeira vez o Generate-React-Cli, através da consola, algumas perguntas são realizadas, criando desta maneira um ficheiro de configuração, contendo uma componente que será gerada por defeito, ou seja, sempre que não se indicar o tipo de componente que não se pretende gerar. O ficheiro de configuração, será designado por 'generate-react-cli.json', contendo a seguinte estrutura:

```
{
  "usesTypeScript": true,
  "usesCssModule": false,
  "cssPreprocessor": "css",
  "testLibrary": "Testing Library",
  "component": {
    "default": {
      "path": "src/components",
      "withStyle": true,
      "withTest": true,
      "withStory": true,
      "withLazy": true
    }
  }
}
```

Figura 5 – Ficheiro de configuração inicial do GRC

Pela a análise da Figura 5, pode-se observar que a "component" será gerado por defeito na pasta "src/components". Para além do ficheiro que contém a componente, serão também gerados para este componente um ficheiro de estilo (em css), um ficheiro de teste, um ficheiro *story* e um ficheiro *lazy*, visto que todos esses atributos se encontram a true no ficheiro de configuração. Para gerar um componente por defeito basta executar o seguinte comando 'npx generate-react-cli component Box' em que 'Box' será o nome da pasta e o nome dos ficheiros que serão gerados.

Apesar do componente definido por defeito, o GRC permite que algumas das configurações sejam alteradas aquando da geração desse mesmo componente, adicionando regras ao comando de geração. A Tabela 2 indica as regras que podem ser usadas no comando de geração.

Tabela 2 – Opções para a geração de componentes com GRC

Opções	**Descrição**	**Tipo de valor**	**Valor por default**
--path	Corresponde ao caminho onde se quer que a componente seja gerada	String	component.default.path
--type	Pode ser passado um tipo personalizado, que se encontre configurado no ficheiro de configuração	String	component.default
--withLazy	Cria um ficheiro Lazy para a componente	Boolean	component.default.withLazy
--withStory	Cria um ficheiro Story para a componente correspondente	Boolean	component.default.withStory
--withStyle	Cria um ficheiro de estilo para a componente correspondente	Boolean	component.default.withStyle
--withTest	Cria um ficheiro de teste para a componente correspondente	Boolean	component.default.withTest

Para além do componente definido por defeito, esta ferramenta permite a definição de tipos personalizados de componentes, no ficheiro de configuração, que podem fazer uso de *templates* para a criação dos seus ficheiros, indicando o caminho para esses mesmos *templates*. A Figura 6 mostra um exemplo de um tipo de componente personalizado com a indicação dos seus respetivos *templates*.

```
"Interface1": {
  "path": "src/interface1Components",
  "withStyle": true,
  "withTest": true,
  "withStory": false,
  "withLazy": false,
  "customTemplates": {
    "component": "templates/interface1/component.js",
    "test": "templates/interface1/test.js",
    "style": "templates/interface1/style.css"
  }
}
```

Figura 6 – Tipo personalizado com referência ao seus templates

Na Figura 7, podemos encontrar um exemplo de *template* definido para o componente personalizado. Para gerar este tipo de componente personalizado basta apenas correr o comando anteriormente descrito adicionando a regra “--type = Interface1.

```
import React from 'react';
import styles from './TemplateName.css';

const TemplateName = () => (
  <div className={styles.TemplateName} data-testid="TemplateName">
    <h1>TemplateName component</h1>
  </div>
);
```

Figura 7 – Template de uma componente

2.4.1.1 Sumário

Esta secção destina-se a compreender melhor quais as vantagens e desvantagens da utilização desta ferramenta, tendo em conta o problema a desenvolver. A Tabela 3 apresenta as vantagens e desvantagens do GRC.

Tabela 3 – Vantagens e desvantagens do GRC

Vantagens	Desvantagens
• É uma ferramenta Open-Source; • Suporta tipos de componente personalizados; • Suporta templates personalizados para as componentes; • Agrupa a componente e todos os seus ficheiros correspondentes na mesma pasta;	• Não tem editor visual integrado • Para além do nome não deixa passar qualquer tipo de parâmetros • Não permite a criação de funções personalizadas

2.4.2 Plop

O Plop[13] é uma ferramenta *Open-Source*, desenvolvida para oferecer uma maneira simples e consistente, para gerar código ou qualquer outro tipo de arquivo de texto. Embora esta ferramenta não seja diretamente desenhada para gerar código em React, o Plop é muito utilizado neste contexto[14].

O objetivo principal desta ferramenta é facilitar os processos habituais de desenvolvimento das equipas, uma vez que, esta ferramenta permite a criação de código com base em *templates*. Desta maneira, esta ferramenta facilita a reutilização de código, poupando tempo às equipas de desenvolvimento, pois estas, não precisam de procurar código que foi executado anteriormente, para ser adaptado aos seus novos projetos.

No contexto do React, esta ferramenta permite a reutilização de componentes anteriormente desenvolvidos, permitindo ao utilizador a passagem de vários parâmetros, tornando esta ferramenta totalmente personalizável. Uma simples chamada na linha de comandos, permite

economizar tempo e automatizar os processos de desenvolvimento das equipas da organização que adota este gerador.

Uma vez que o Plop se encontre instalado, o primeiro passo é a criação de um Plopfile, que é o ficheiro de configuração utilizado para esta ferramenta. A Figura 8 ilustra um exemplo de um ficheiro de configuração para esta ferramenta.

```
module.exports = function(plop) {
    // controller generator
    plop.setGenerator('controller', {
        description: 'application controller logic',
        prompts: [{
            type: 'input',
            name: 'name',
            message: 'controller name please'
        }],
        actions: [{
            type: 'add',
            path: 'src/{{name}}.js',
            templateFile: 'plop-templates/controller.hbs'
        }]
    });
};
```

Figura 8 – Exemplo de um Plopfile

Após uma análise da Figura 8, pode-se verificar que o gerador utiliza prompts, que basicamente funcionam como parâmetros de entrada para o utilizador e utiliza as actions para saber o que fazer quando um dos geradores é chamado.

Para comunicar com o utilizador o Plop faz uso de uma biblioteca inquier.js[15]. Existem duas maneiras de solicitar informação ao utilizador.

A primeira maneira é no momento da execução, as prompts definidas para esse gerador são solicitadas ao utilizador através da consola. O utilizador insere o seu valor mediante o tipo definido.

A segunda maneira que o utilizador tem para passar as prompts é diretamente na chamada do comando, como por exemplo:

```
## Bypassing both prompt 1 and 2
$ plop component "my component" react
$ plop component -- --name "my component" --type react
```

Figura 9 – Comandos Plop para a passagem de prompts[8]

Para além disso, o Plop disponibiliza um conjunto de métodos de apoio aos desenvolvedores, para criarem o Plopfile. Sendo os métodos mais utilizados [13] os que se encontram descritos na Tabela 4.

[8] Retirado de: https://plopjs.com/documentation/

Tabela 4 – Principais métodos do Plop

Método	Parâmetros	Descrição
setGenerator	String,GeneratorConfig	O objeto de configuração precisa de incluir prompts e actions. A matriz de prompts é passada ao inquirer. A matriz de actions define que ações devem ser executadas.
setHelper	String,Function	Este método permite definir funções auxiliares que não fazem parte da própria linguagem do Handlebars
setPartial	String,String	Este método permite a reutilização de partes de um template, podendo ser chamadas por outros templates.
setActionType	String,CustomAction	O Plop já dispõe de algumas actions, mas este método permite que o desenvolvedor possa criar as suas próprias actions. Tornando esta ferramenta totalmente personalizável.
setPrompt	String,InquirerPrompt	O inquirer disponibiliza ao desenvolvedor muitos tipos de prompts, mas também é permitido que os desenvolvedores criem os seus próprios tipos de prompts.
load	Array[String],Object,Object	Transporta toda a informação de outra Plopfile ou módulo npm.

Como já foi referido na Tabela 4, o Plop já dispõe de um conjunto de ações (*actions*) previamente definidas, que cobrem grande parte das necessidades por parte dos desenvolvedores, sendo elas:

- **Add** – A ação add é utilizada para adicionar ficheiros ao projeto. Nesta ação é possível definir um conjunto de propriedades de apoio, destacando-se, o *path* que indica o caminho onde o ficheiro deve ser gerado, e a propriedade *templateFile* que basicamente indica o caminho para o template através do qual o ficheiro deve ser gerado.
- **AddMany** – A ação addMany foi desenhada para adicionar vários ficheiros ao projeto com uma única ação. A propriedade *destination* é utilizada para descrever o caminho onde estes ficheiros devem ser gerados. Para além disso, nesta ação é possível definir o caminho para os diferentes *templates* dos ficheiros, através da propriedade *templateFiles*.
- **Modify** – A ação modify pode ser utilizada de duas maneiras. Uma delas é utilizando a propriedade *pattern* que basicamente permite encontrar o texto localizado no ficheiro, que se encontra definido na propriedade *path*, através de uma expressão regular. A outra maneira, é utilizar a propriedade *transform*, basicamente é uma função utilizada para transformar os conteúdos de um ficheiro.
- **Append** - A ação append é um subconjunto da ação modify. Esta ação é utilizada para anexar dados a um ficheiro, num local específico. Normalmente para identificar onde anexar esses dados é utilizada a propriedade *pattern*, utilizando uma expressão regular.

Para além das ações disponibilizadas pelo Plop, que já cobrem grande parte das necessidades dos desenvolvedores, através do Plop, é possível definir ações personalizadas. Para definir estas ações personalizadas é utilizado o método *setActionType*, permitindo desta maneira, que os desenvolvedores consigam executar qualquer tipo de operação utilizando o Plop. A Figura 10 demonstra um exemplo de uma ação personalizada que basicamente recebe um caminho onde se encontra um ficheiro JSON, com informação relativa às componentes que devem ser geradas.

```
plop.setActionType('customAction', function (answers, config, plop) {
        const {path} = answers;
        const {compType, props, childs} = require(path);

        const typeInfo = getTypeInfo(compType, props.name);

        createFile(plop, typeInfo.templateFile, typeInfo.path, props);

        handleChilds(plop, childs);

});
```

Figura 10 – Exemplo de uma ação personalizada

2.4.2.1 Sumário

A presente secção, tem o propósito de compreender melhor quais vantagens e desvantagens, que a utilização desta ferramenta pode trazer, tendo em consideração o problema a desenvolver. A Tabela 5 faz identifica as vantagens e desvantagens da ferramenta Plop.

Tabela 5 – Vantagens e desvantagens do Plop

Vantagens	**Desvantagens**
• É uma ferramenta Open-Source • Suporta tipos de componente personalizados • Suporta templates personalizados para as componentes • Permite definir a estrutura de como os ficheiros são gerados • Permite a passagem de vários parâmetros • Permite a criação de ações personalizadas	• Não tem editor visual integrado

2.4.3 Divjoy

O Divjoy[16] é uma ferramenta *Closed-Source* desenhada para criar código base para UIs em React. A utilização do Divjoy permite com que os desenvolvedores avancem rapidamente, economizando bastante tempo, podendo, desta maneira, focar-se apenas na lógica do negócio.

Para além disso, o Divjoy disponibiliza um conjunto de ferramentas, que são apresentadas na Figura 11, que permite ao desenvolvedor aceder a todos os tipos de recursos para que ele possa construir uma aplicação em React bastante completa. O Divjoy disponibiliza também um conjunto de templates, nomeadamente página inicial, autenticação, preços, formulários, configurações de conta e pagamentos.

Estes templates podem ser personalizados através de um editor integrado no Divjoy. Este editor permite ajustar estilos, adicionar novas páginas e arrastar novas secções a partir das componentes visuais disponibilizadas pela ferramenta.

Quando a base da aplicação de React estiver pronta, o Divjoy permite que o desenvolvedor faça o download ou a exportação do seu código base, garantindo que todo o sistema é responsivo e perfeitamente integrado com o conjunto de ferramentas previamente selecionadas pelo desenvolvedor.

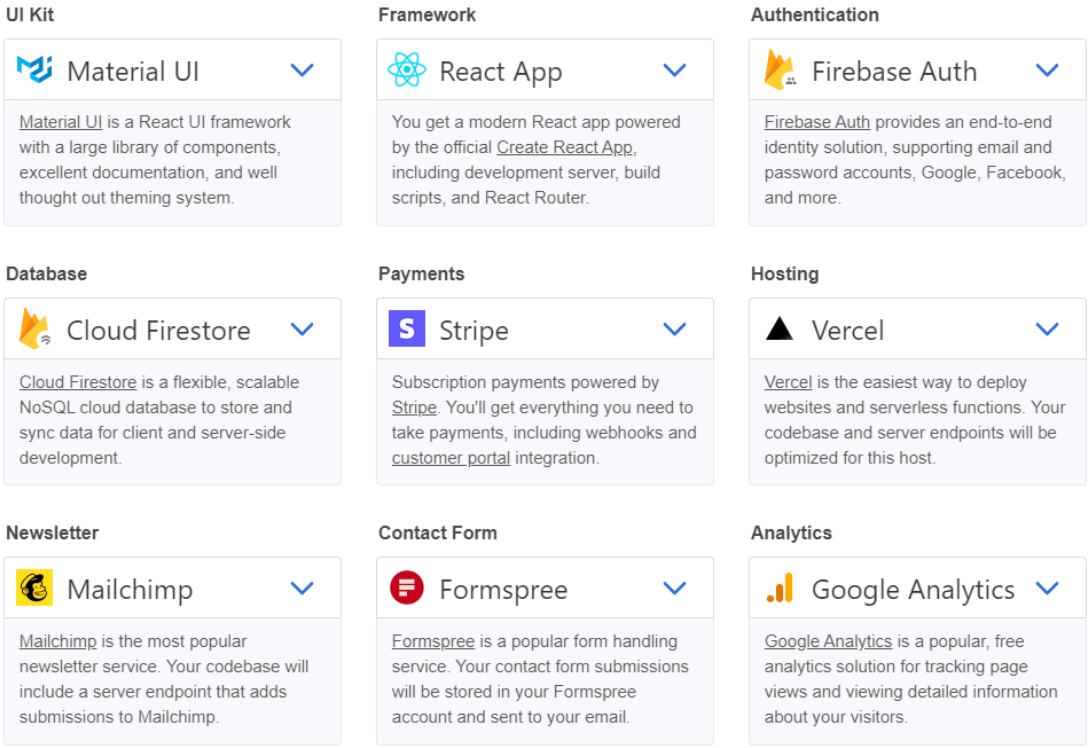


Figura 11 – Conjunto de ferramentas disponibilizadas pelo Divjoy[9]

[9] Retirado de: https://divjoy.com/

2.4.3.1 Sumário

A presente secção, tem o propósito de compreender melhor quais vantagens e desvantagens, que a utilização desta ferramenta pode trazer, tendo em consideração o problema a desenvolver. A Tabela 6 apresenta as vantagens e desvantagens da ferramenta Divjoy.

Tabela 6 – Vantagens e desvantagens do Divjoy

Vantagens	**Desvantagens**
• Disponibiliza um conjuntos de templates • Editor online que permite arrastar as componentes para o local pretendido	• É uma ferramenta Closed-Source • Não suporta tipos de componentes personalizados • Não suporta templates personalizados para as componentes • Não permite a criação de ações personalizadas

2.4.4 Comparação entre geradores para React

Com objetivo de analisar os geradores anteriormente descritos, nomeadamente o GRC, Plop e o Divjoy, elaborou-se uma tabela comparativa, Tabela 7, com o intuito de perceber quais as grandes diferenças entre essas ferramentas, tentando perceber se alguma delas seria uma boa base para a resolução do problema, ou seja, as funcionalidades descritas nesta secção são as consideradas mais importantes para a resolução do problema.

Tabela 7 – Comparação entre geradores para React

Geradores **Funcionalidades**	**GRC**	**Plop**	**Divjoy**
Ferramenta Open-Source	✓	✓	✗
Suporta diferentes tipos de componentes personalizados	✓	✓	✗
Suporta templates personalizadas para as componentes	✓	✓	✗
Permite definir a estrutura de como os ficheiros são gerados	✓	✓	?
Permite a criação de ações personalizadas	✗	✓	✗
Tem editor visual integrado	✗	✗	✓

✗- Não contém a funcionalidade ✓ - Contém a funcionalidade ? - Não há informação

Após uma breve análise comparativa da tabela acima descrita, pode-se verificar, que a ferramenta com mais potencial é o Plop, visto que, esta apenas não contém um editor visual, mas permite uma personalização muito completa do que deve ou não ser gerado com base em templates. No entanto, na Secção 3.4 encontra-se descrito um método de apoio à decisão que permite identificar qual a melhor alternativa (GRC, Plop ou Divjoy), para os critérios com mais relevância para a organização.

2.5 Trabalhos relacionados

A presente secção tem como objetivo apresentar alguns trabalhos científicos relacionados com o projeto a desenvolver. Os estudos apresentados permitem perceber algumas das técnicas utilizadas para o desenvolvimento de geradores de código. Para além disso, são analisados os resultados obtidos com a utilização desses geradores de código para perceber que impacto podem ter no processo de desenvolvimento.

2.5.1 Gerador de interfaces HTML

O sistema proposto por Bassil e Alwani [17] é um gerador automático de *UIs* de aplicações web. O objetivo do sistema desenvolvido é automatizar a personalização das interfaces gráficas de acordo com as regras de negócio, políticas e ambiente operacional com a mínima intervenção das equipas de desenvolvimento.

Normalmente as aplicações web são utilizadas por grupos de utilizadores diferentes, com diferentes direitos de autorização, privilégios de acesso etc. Deste modo é necessário criar várias versões de interface para a mesma aplicação. Posto isto, o processo de desenvolvimento dessas interfaces gráficas pode ser complexo, surgindo a necessidade da criação de um sistema capaz de autoconfigurar essas interfaces gráficas para aplicações web, com o mínimo de ação das equipas de desenvolvimento.

O sistema proposto é um gerador autónomo de *UIs* em HTML, com base em W3C XML Schema, e um ficheiro de estilo associado. Resumidamente, o processo consiste em analisar e interpretar um ficheiro XML, transformando-o num ficheiro HTML, que contém todas as restrições e componentes da *UI* (caixas de texto, listas, etc.), e num ficheiro de estilo, que tem como objetivo formatar e decorar a interface gráfica gerada. Para além disso, o sistema pode gerar funções de JavaScript que possam validar os dados inseridos na interface gráfica gerada.

O sistema desenvolvido é composto por três componentes, representadas na Figura 12:

- **Analisador léxico –** O objetivo desta componente é analisar o fluxo de dados recebidos do ficheiro XML e do ficheiro de estilo, produzindo, de acordo com os dados de entrada, um outro fluxo designado por *tokens;*

- **Analisador sintático –** É a componente responsável por reconhecer sintaticamente tudo o que se encontra descrito no XML de entrada;

- **Gerador de código –** É responsável por gerar a interface gráfica HTML esperada.

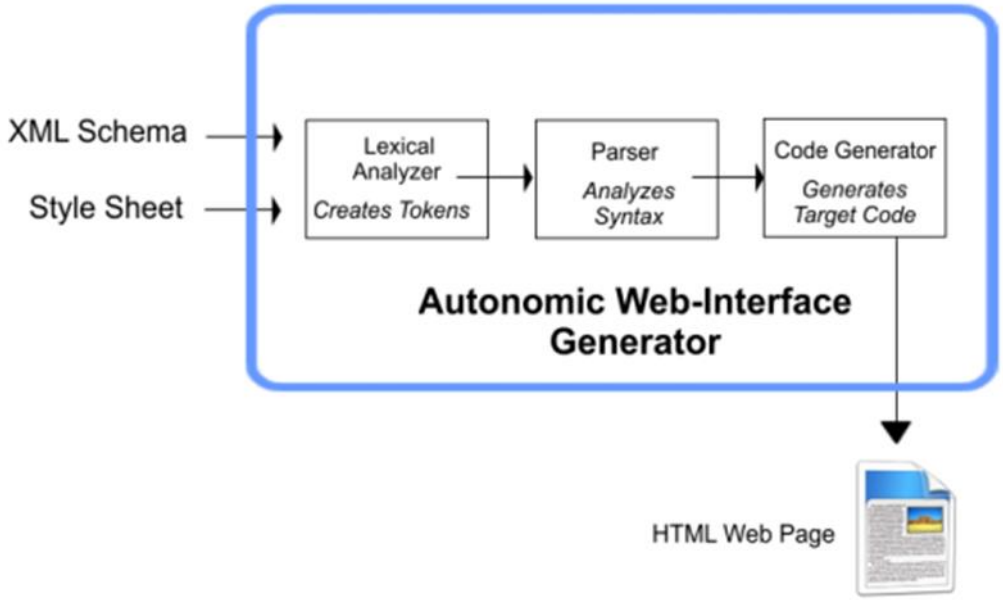


Figura 12 – Diferentes fases do gerador HTML proposto [17]

Com a utilização do sistema desenvolvido, foi possível diminuir os custos de desenvolvimento, uma vez que a introdução de código por parte das equipas de desenvolvimento é mais reduzido.

Desta maneira, é também possível tornar todo o processo de implementação de infraestruturas mais simples e mais rápido, contribuindo para que os custos de manutenção e desenvolvimento sejam também mais reduzidos.

Para além destes dois fatores, foi possível também verificar um aumento da produtividade.

2.5.2 Gerador de código para aplicações Web

Uyanik e Sahin [6] desenvolveram um gerador de código com base em *templates* para aplicações Web. Numa abordagem deste tipo, como já foi referido anteriormente, o gerador de código requer os dados de entrada, e com base nos *templates* previamente definidos produz o software esperado.

O sistema proposto por Uyanik e Sahin, tinha como objetivo ser integrado numa aplicação Web, nomeadamente num sistema Enterprise Resource Plannig (ERP) do Instituto Turco de Ciências de Gestão (TÜSSİDE)[10].

O gerador de código desenvolvido é capaz de criar novos módulos (processo de geração de código) e é também capaz de integrar esses novos módulos gerados no ERP do TÜSSİDE.

Com o objetivo de perceber de forma mais clara o funcionamento deste gerador, pode encontrar-se uma descrição passo a passo do fluxo de funcionamento deste gerador:

1. O programador cria a tabela na base de dados para o módulo que pretende criar;
2. O gerador de código cria automaticamente os procedimentos armazenados para as operações na nova tabela;
3. O programador seleciona o *template* HTML para a UI;
4. O gerador de código cria os *plugins* HTML com base na informação da tabela criada na base de dados;
5. O gerador de código cria os *plugins* JavaScript para os eventos dos *plugins* HTML;
6. O gerador cria esquemas JSON com base nos *templates* predefinidos;
7. O gerador cria uma API em ASP.NET para a comunicação do lado do servidor;
8. O gerador cria um ficheiro HTML com base no *template* previamente selecionado e de acordo com os *plugins* HTML e JavaScript gerados. O ficheiro gerado tem também em consideração o esquema JSON gerado;
9. O programador e o gerador de código integram o novo módulo no ERP.

Com o uso deste gerador, os autores conseguiram denotar claramente uma redução de tempo do processo de desenvolvimento. O tempo médio de desenvolvimento antes da implementação do gerador de código era cerca de 397,18 minutos, após a sua implementação este valor reduziu para uma média de 4,17 minutos. Deste modo pode verificar-se uma melhoria de cerca 98,95% no tempo médio de desenvolvimento.

Uma outra vantagem da utilização deste gerador, está relacionada com a contagem de bugs. O gerador desenvolvido é com base em templates, sendo estes templates preparados a partir de uma implementação de referência validada. Por este mesmo motivo, os autores não encontram qualquer tipo de bug ao usar o gerador de código desenvolvido. Durante a implementação

[10] https://tusside.tubitak.gov.tr/en

manual, algumas dezenas de bugs foram encontrados e corrigidos. Deste modo pode-se concluir que uma das vantagens da utilização de um gerador deste tipo é a redução de erros.

Uma desvantagem do gerador de código encontrada pelos autores é a sua inflexibilidade. Como o gerador tem por base templates, que precisam de estar previamente definidos, quando surge um novo requisito é necessário criar todas os templates manualmente e adicioná-los posteriormente ao gerador de código.

2.5.3 Framework de geração de código para plataformas *smart TV*

Nos últimos anos, as industrias de *smart* TVs tornam-se cada vez mais alvos para a indústria de *software*, uma vez que, o seu uso tem se tornado cada vez mais comum. Akbulut e Toprak [18], demonstram como desenvolveram uma framework de geração de código, para as principais plataformas de *smart* TV.

A estrutura proposta pelos autores tem por base projetos desenvolvidos em C#, que posteriormente são convertidos nas principais plataformas de *smart* TVs, nomeadamente Android TV, Firefox OS e Tizen OS. Para a construção desta framework é utilizada uma abordagem com base em *template* combinada com *Model Driven Architecture (*MDA*)*, o que torna este gerador extensível a novas plataformas de *smart* TV.

Primeiramente os projetos criados em C# são traduzidos em ficheiros PIM[11], que são ficheiros XML. Posteriormente, são injetados no sistema os *templates* associados. Como resultado da combinação dos ficheiros PIM e os *templates* associados, os ficheiros de saída correspondentes são gerados.

A arquitetura do sistema, representada na Figura 13, permite identificar duas camadas distintas:

- ***Application Modeler –*** Esta é a camada responsável por importar a aplicação C#, que se pretende converter numa plataforma para *smart* TV. É também nesta camada que as classes das aplicações em C# são analisadas e posteriormente transformadas em ficheiros PIM, num formato XML;

- ***Application Producer –*** Esta camada aceita os ficheiros PIM, gerados pelo *Application Modeler,* como entrada. Posteriormente estes ficheiros PIM, com um formato XML, são consultados e analisados, percorrendo todos os seus elementos XML. Através dos dados interpretados, o código é gerado de acordo com os templates específicos da plataforma para a qual se pretende gerar o código.

Através da arquitetura definida, com a implementação por camadas, foi possível garantir o baixo acoplamento, uma vez que, os PIMs são consumidos pelas diferentes plataformas de

[11] https://file.org/extension/pim

destino. Deste modo, é possível garantir, que a adição de novos módulos de geração é um processo simples.

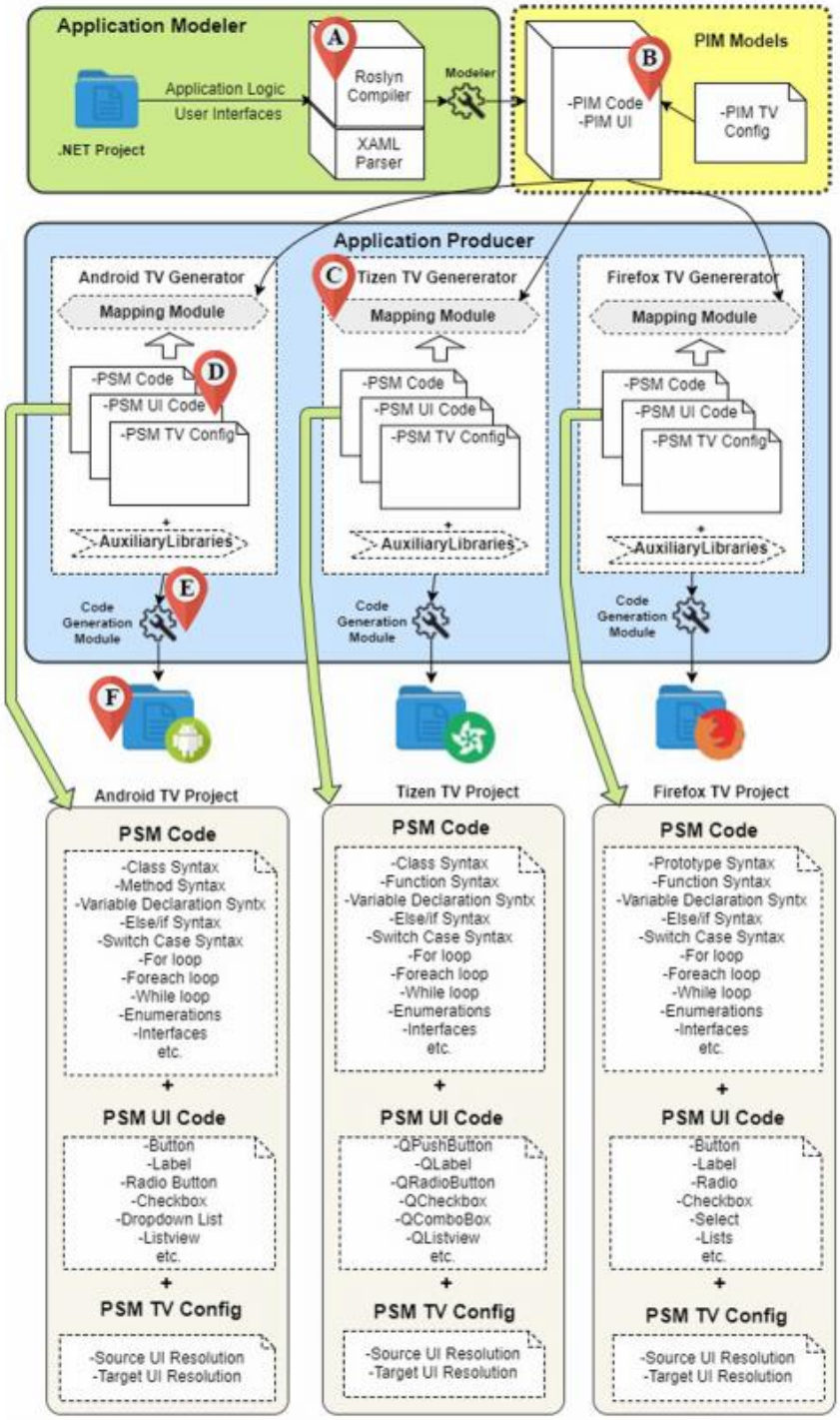


Figura 13 – Arquitetura da framework de geração de código proposta [18]

2.5.4 Comparação entre trabalhos realizados

Na presente secção apresenta-se uma breve comparação entre os trabalhos analisados relacionados com o projeto a desenvolver. Na Tabela 8, encontram-se descritos os critérios comparativos e a comparação entre os trabalhos relacionados.

Tabela 8 – Comparação entre trabalhos relacionados

Trabalho	Metodologia	Dados de Entrada	Funcionalidade
Gerador interfaces HTML	RBT	Ficheiro XML e um ficheiro de estilo	Gera uma página Web em HTML
Gerador de aplicações Web	TBCG	Tabela de base de dados	Gera um módulo de uma aplicação Web
Gerador para plataformas *Smart* TV	TBCG com MDA	Projeto em C#	Converte os projetos C# nas plataformas *Smart* TV

Após uma análise comparativa da tabela apresentada, pode-se concluir, que a técnica *Template-Based Code Generation* é muito utilizada no processo de desenvolvimento de geradores de código.

3 Análise de valor

No presente capítulo, é apresentada a análise de valor do projeto, com o objetivo de demonstrar o valor que acarreta para a organização, uma vez que, se trata de um projeto interno.

Nesta análise, foi utilizado o modelo *New Concept Development* (NCD)[19], para a identificação de oportunidade e a sua respetiva análise. Este modelo, representado na Figura 14, permite definir a fase inicial do processo de negócio e inovação. O NCD encontra-se dividido em três componentes:

- **Motor –** É a liderança, cultura e estratégia de negócios da organização que impulsiona os cinco elementos-chave que são controláveis pela organização[19];
- **Fatores de influência –** Correspondem aos diversos fatores que caracterizam a organização, os fatores externos à organização (por exemplo, legislação, clientes e concorrentes etc.) e a sua estratégia empresarial;
- **Elementos chave –** São os elementos controláveis pela organização. Existem cinco elementos chave, nomeadamente, identificação de oportunidade, análise de oportunidade, geração e enriquecimento de ideias, seleção de ideias e definição do conceito. No entanto, para este caso apenas se efetuou os dois primeiros elementos.

Para além da identificação de oportunidade e da sua análise, foi definida uma proposta de valor para o presente projeto. Foi também utilizado um método de apoio à decisão, que permitiu identificar, de entre os geradores de código para React, descritos no Capítulo 2, e com base em alguns critérios, qual a melhor opção para a resolução da solução.

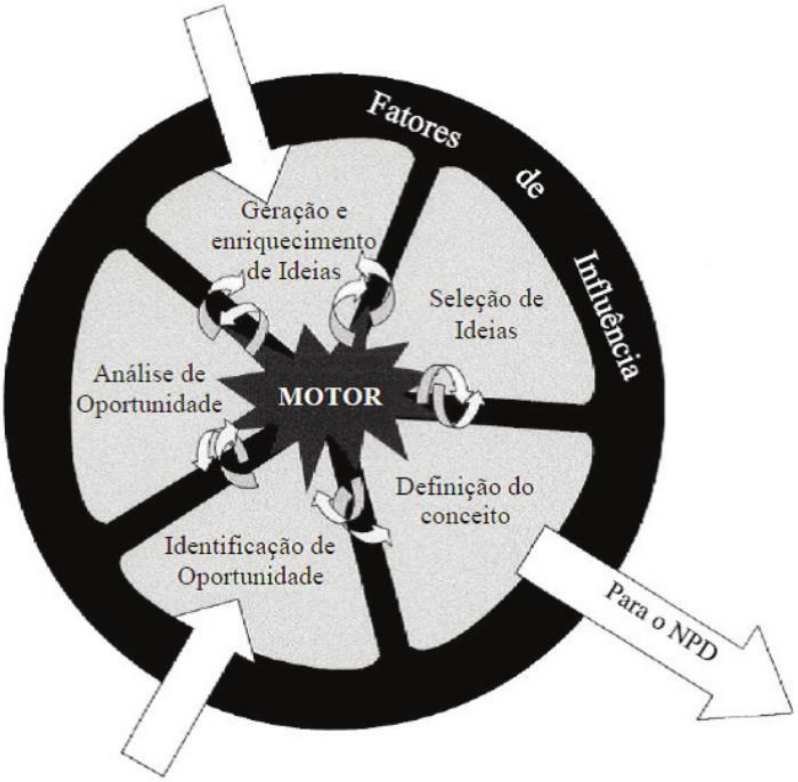


Figura 14 – Modelo NCD[12]

3.1 Identificação de oportunidade

O presente subcapítulo, permite identificar as oportunidades que a empresa pretende perseguir [19].

A Celfocus é uma empresa em expansão e cada vez mais há um número crescente de projetos a desenvolver. Nestes projetos verifica-se que as componentes de React utilizadas para o seu desenvolvimento são, muitas das vezes, reutilizadas, verificando-se apenas alterações na lógica de negócio.

Isto fez com que a organização sentisse a necessidade da criação de uma framework de geração de código capaz de gerar código-fonte de qualidade. O principal objetivo é a geração de React Journeys (conjunto de componentes React), mas deve também ser vista como algo mais genérico, ou seja, deve também ter em vista a integração de novos módulos de geração num futuro próximo (como por exemplo, geração de microsserviços).

Para além do crescente número de projetos, os requisitos do cliente são cada vez mais complexos, o que faz com que a lógica de negócio seja consequentemente mais complexa. Uma

[12] Retirado de: researchgate.net/figure/Figura-1-Modelo-New-Concept-Development-NCD_fig1_307712679

ferramenta de geração de código, gerando o código fonte com qualidade, permite que as equipas de desenvolvimento se foquem apenas nessa lógica de negócio mais complexa, garantindo desta maneira um aumento da produtividade, no que diz respeito ao processo de desenvolvimento.

3.2 Análise de oportunidade

Após a identificação da oportunidade é necessário avaliá-la, com o objetivo de comprovar se vale a pena ou não persegui-la. Existem enumeras estratégias para esta análise[19].

Para este projeto, em particular, foi utilizada a *Strategic framing*, que permite determinar como é que a oportunidade se encaixa no mercado da empresa, identificando os pontos fortes e fracos e as ameaças [19]. Para tal, foi definida uma análise SWOT[20], dividida em fatores internos (forças e fraquezas) e fatores externos (oportunidades e ameaças), ilustrada na Figura 15.

Como forças é necessário realçar que a criação de uma framework de geração de código, permite que o código gerado seja robusto e escalável. Para além disso, uma ferramenta deste tipo contribui para a reutilização de código, desta maneira, a framework desenvolvida espera ter um impacto significativo na produtividade das equipas de desenvolvimento, bem como, uma diminuição significativa do tempo gasto no processo de desenvolvimento de aplicações futuras. Como a redução de código produzido à mão, implica que o software desenvolvido seja menos suscetível a possíveis erros.

No que diz respeito às fraquezas, é importante relembrar, que o mundo da engenharia de software se encontra em constante evolução, e o surgimento de novos requisitos é constante. Por este mesmo motivo, a DSL terá de ser constantemente atualizada para conseguir cumprir todos esses novos requisitos. No entanto, esta não é a única fraqueza encontrada para o problema descrito anteriormente, com uma ferramenta de geração de código há sempre risco de o código gerado sobrescrever o código previamente escrito.

Como o projeto desenvolvido corresponde a um projeto interno de uma organização, consideram-se fatores externos os comportamentos organizacionais da mesma. Deste modo, podemos afirmar que as oportunidades deste projeto estão diretamente relacionadas com o crescimento da organização. Com o aumento do número de projetos a organização sente necessidade de aumentar a sua produtividade, tornando todo o seu processo de desenvolvimento mais eficiente. Também se deve ter em consideração, que os requisitos dos clientes são cada vez mais complexos e ambiciosos, contribuindo para uma lógica de negócio e software mais complexos. Por essa mesma razão, um software capaz de automatizar o processo de desenvolvimento, gerando o código fonte, nomeadamente React Journeys, permite que os desenvolvedores se foquem apenas em desenvolver essa mesma lógica. Por outro lado, existem algumas ameaças relativamente ao projeto a desenvolver, o surgimento de projetos prioritários pode levar à descontinuidade do mesmo. Para além disso, as frameworks de desenvolvimento

adotadas podem ser descontinuadas obrigando desta maneira a adoção de novas abordagens para o desenvolvimento deste projeto. Uma outra ameaça é que a organização pode optar, futuramente, pela adoção de novos paradigmas que o gerador não consegue cobrir.

Tendo em conta a análise efetuada pode-se afirmar que o maior valor, que uma *framework* deste tipo, traz para a organização é o aumento da sua produtividade. Para além disso, torna todos os processos de desenvolvimento mais eficientes, retirando desta maneira algumas responsabilidades para as equipas de desenvolvimento, o que permite que estas apenas se foquem na lógica de negócio.

Figura 15 – Análise SWOT

3.3 Proposta de Valor

Como se está a desenvolver um produto interno, para uso exclusivo da organização, através da proposta de valor, pretende-se demonstrar a correspondência entre as necessidades dos consumidores e os benefícios que o produto acarreta com a sua utilização. Como tal foi definida a seguinte proposta de valor:

"O projeto desenvolvido é uma framework responsável pela geração de componentes React, uma vez que os processos definidos pela organização contêm componentes comuns, para além disso tem em vista a integração de novos módulos de geração num futuro próximo, assegurando, desta maneira, a sua escalabilidade. Deste modo, esta ferramenta permite, às equipas de desenvolvimento, o reaproveitamento de código, bem como, um aumento significativo na produtividade da mesma, tornando o código gerado escalável, robusto e menos suscetível a erros."

Com o objetivo de perceber melhor, de que maneira o projeto a ser desenvolvido pode trazer ou não valor a todas as partes interessadas elaborou-se um modelo CANVAS para a proposta de valor, descrito na Figura 16.

Este modelo encontra-se dividido em seis partes, sendo elas[21]:

- **Gains –** É a secção onde se define que fatores fariam o cliente feliz, ou seja, que fatores facilitariam o seu trabalho;
- **Pains –** Esta secção, descreve que fatores incomodam os clientes;
- **Customer Jobs –** Permite identificar que trabalhos o cliente deseja obter;
- **Gain Creators –** É a secção onde é identificado de que maneira o produto a ser desenvolvido contribui para a felicidade do cliente;
- **Pain Relievers –** Esta secção permite identificar de que maneira se pode ajudar o cliente a aliviar o que o incomoda;
- **Products & Services –** Identifica quais os produtos e serviços se oferecem ao cliente para que eles realizem o trabalho esperado.

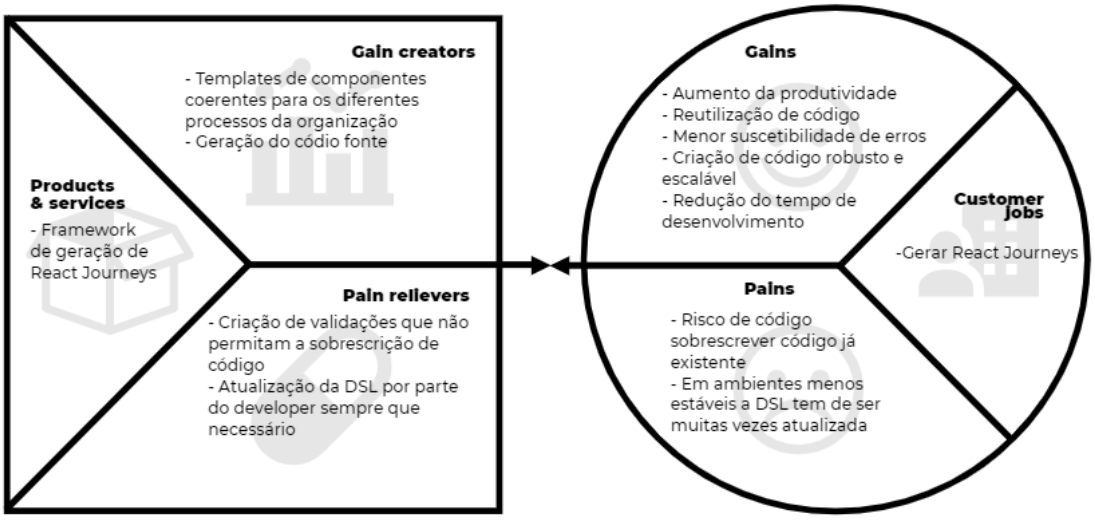


Figura 16 – CANVAS para a proposta de valor

3.4 Analytic Hierarchy Process

O Analytic Hierarchy Process (AHP)[22] é um método matemático, utilizado para organizar, analisar e apoiar decisões complexas que precisam de ser tomadas. Este é constituído por três partes:

- O objetivo final ou problema;
- Os critérios que irão ser usados para avaliar as diferentes opções;
- E as soluções possíveis, designadas por alternativas.

O principal foco desta tese foi implementar um gerador de código React, deste modo, o método AHP foi aplicado a esses geradores com o intuito de perceber qual a melhor ferramenta a adotar para a construção da solução. Com base nos geradores de código abordados anteriormente (GRC, Plop e Divjoy), foi utilizado o método AHP para a definição do grau de importância e para o auxílio na tomada de decisão de que tecnologia adotar. Para a análise em questão, foram escolhidos os três critérios mais relevantes para o estudo, sendo eles: (i) a tecnologia suporta diferentes tipos de componentes; (ii) a tecnologia suporta templates personalizados para as componentes; (iii) a tecnologia permite definir a estrutura dos ficheiros gerados.

Para a definição do nível de importância de cada critério definido foi utilizada a escala descrita na Figura 17.

Nivel de importância	Definição	Explicação
1	Igual importânica	As duas atividades contribuem igualmente para o objetivo
3	Fraca importância	A experiência e o julgamento favorecem levemente uma atividade em relação à outra
5	Forte importância	A experiência e o julgamento favorecem fortemente uma atividade em relação à outra
7	Muito forte importância	Uma atividade é muito fortemente favorecida em relação a outra
9	Importância absoluta	A evidência favorece uma atividade em relação a outra com o mais alto grau de certeza
2,4,6,8	Valores intermediários	Quando se procura uma condição de compromisso entre duas definições

Figura 17 – Valores para o nível de importância de cada critério

Para o cálculo da razão de consistência (RC) é utilizado um índice aleatório, IR, de acordo com o número de critérios definidos. Os valores de IR, para matrizes quadradas de ordem n, encontram-se descritos na Figura 18.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0.00	0.00	0.58	0.90	1.12	1.24	1.32	1.41	1.45	1.49	1.51	1.48	1.56	1.57	1.59

Figura 18 – Valores de IR para matrizes quadradas de ordem n

Numa primeira fase foi identificada a árvore hierárquica, descrita na Figura 19, foi identificado o problema, os critérios e as alternativas que foram julgadas com base nos critérios previamente definidos.

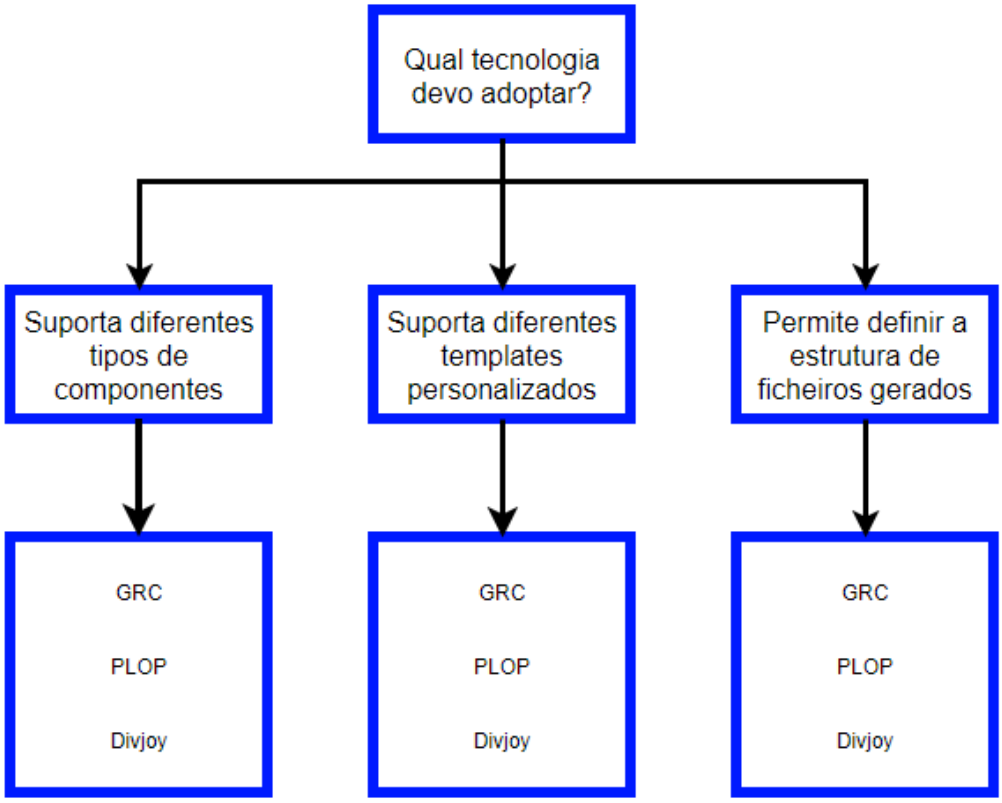


Figura 19 – Árvore hierárquica

Posteriormente foi definida a matriz par a par, Tabela 9, responsável por comparar os critérios atribuindo um valor de intensidade de importância mediante a escala descrita anteriormente.

Tabela 9 – Matriz de comparação par a par para os critérios

Critérios	Diferentes tipos de componentes	Templates Personalizados	Definição da estrutura de ficheiros
Diferentes tipos de componentes	1	1/3	4
Templates personalizados	3	1	5
Definição da estrutura de ficheiros	1/4	1/5	1
Total	4 1/4	1 1/2	10

Depois de definida a matriz de comparação par a par foi calculada a matriz normalizada, que resulta da divisão de cada elemento pelo somatório da respetiva coluna, a matriz normalizada da comparação dos critérios e o respetivo vetor de prioridades encontra-se descrito na Tabela 10.

Tabela 10 – Matriz normalizada para os critérios e respetivo vetor de prioridades

Critérios	Diferentes tipos de componentes	Templates personalizados	Definição da estrutura de ficheiros	Vetor de prioridades
Diferentes tipos de componentes	0,2353	0,2174	0,4000	0,2842
Templates personalizados	0,7059	0,6522	0,5000	0,6194
Definição da estrutura de ficheiros	0,0588	0,1304	0,1000	0,0964

De seguida foi realizado um teste de consistência para os critérios definidos anteriormente, em função das importâncias atribuídas.

- **1º Passo - Cálculo do vetor próprio**
  Foi calculado através da multiplicação da matriz de comparação par a par com o vetor de pesos calculado na normalização.

Matriz de comparação par a par				Pesos		
1	1/3	4	x	0,2842	=	7/8
3	1	5	x	0,6194	=	2
1/4	1/5	1	x	0,0964	=	2/7

Figura 20 – Cálculo do vetor próprio

- **2º Passo - Cálculo do vetor intermédio**
  Os valores obtidos no passo anterior são divididos, pelo respetivo valor de pesos normalizado**.**

$$\frac{0{,}875}{0{,}2842} = 3{,}0833$$

$$\frac{2}{0{,}6194} = 3{,}1547$$

$$\frac{0{,}2857}{0{,}0964} = 3{,}0221$$

- **3º Passo - Cálculo do valor próprio**
  Através dos valores obtidos no passo anterior:

$$\frac{3{,}0833 + 3{,}1547 + 3{,}0221}{3} = 3{,}0867$$

- **4º Passo - Cálculo do índice de consistência**
  O cálculo do índice de consistência é calculado pela seguinte formula:

$$IC = \frac{(\lambda \max - n)}{(\mathrm{n} - 1)}$$

  Ou seja,

$$IC = \frac{(3{,}0867 - 3)}{(3 - 1)} = 0{,}043339$$

- **5º Passo - Cálculo da razão de consistência**
  Para o cálculo da razão de consistência é utilizada a seguinte formula:

$$RC = \frac{IC}{\mathrm{IR}}$$

  O IR para três critérios é 0,58.

$$RC = \frac{0{,}043339}{0{,}58} = 0{,}074723$$

  No exemplo ilustrado RC é menor ou igual a 0,1, logo pode-se concluir que os valores de prioridades relativas utilizados foram consistentes.

Após a verificação de que os valores utilizados para a comparação entre critérios são consistentes, procedeu-se à elaboração das matrizes de comparação para cada critério (Diferentes tipos de componentes, templates personalizados e definição da estrutura de ficheiros), considerando cada uma das alternativas selecionadas (GRC, Plop e Divjoy).

A Tabela 11 demonstra a comparação paritária referente ao suporte de diferentes tipos de componentes.

Tabela 11 – Comparação paritária para diferentes tipos de componentes

Diferentes tipos de componentes	GRC	Plop	Divjoy
GRC	1	1/2	5
Plop	2	1	6
Divjoy	1/5	1/6	1
Total	3,2	1,6667	12

A Tabela 12 demonstra a comparação paritária referente ao suporte de templates personalizados.

Tabela 12 – Comparação paritária para templates personalizados

Templates personalizados	GRC	Plop	Divjoy
GRC	1	1/3	4
Plop	3	1	7
Divjoy	1/4	1/7	1
Total	4,25	1,4762	12

A Tabela 13 demonstra a comparação paritária referente à definição da estrutura de ficheiros gerada.

Tabela 13 – Comparação paritária para estrutura de ficheiros

Definição da estrutura de ficheiros	GRC	Plop	Divjoy
GRC	1	1/3	4
Plop	3	1	6
Divjoy	1/4	1/6	1
Total	4,25	1,5	11

Seguidamente foram calculadas as matrizes de comparação paritárias normalizadas para cada uma das alternativas

A Tabela 14 demonstra a comparação paritária normalizada referente ao suporte de diferentes tipos de componentes.

Tabela 14 – Matriz paritária para diferentes tipos de componentes normalizada

Diferentes tipos de componentes	GRC	Plop	Divjoy	Pesos
GRC	1/3	2/7	3/7	0,3431
Plop	5/8	0,6	0,5	0,5750
Divjoy	0,0625	0,1	0,0833	0,0819
Total				1

A Tabela 15 demonstra a comparação paritária normalizada referente ao suporte de templates personalizados

Tabela 15 – Matriz paritária para templates personalizados normalizada

Templates personalizados	GRC	Plop	Divjoy	Pesos
GRC	1/4	2/9	1/3	0,2648
Plop	0,7059	0,6774	0,5833	0,6555

Divjoy	0,0588	0,0968	0,0833	0,0796
Total				1

A Tabela 16 demonstra a comparação paritária normalizada referente à definição da estrutura de ficheiros gerada.

Tabela 16 – Matriz paritária para definição da estrutura de ficheiros normalizada

Definição da estrutura de ficheiros	GRC	Plop	Divjoy	Pesos
GRC	1/4	2/9	1/3	0,2737
Plop	5/7	2/3	1/2	0,6393
Divjoy	0,0588	1/9	0,0909	0,0869
Total				1

Seguidamente para a verificação da consistência dos dados foram calculadas as razões de consistência para cada um dos critérios, utilizando o mesmo raciocínio descrito anteriormente. A Tabela 17 apresenta a razão de consistência para cada um dos critérios.

Tabela 17 – Razões de consistência para os diferentes critérios

Critérios	Razão de consistência
Diferentes tipos de componentes	0,025131
Templates personalizados	0,028041
Definição da estrutura de ficheiros	0,046571

Após uma análise da tabela verificou-se que as razões de consistência calculadas para cada critério são menores 0,1, ou seja, pode-se afirmar que os valores utilizados são consistentes.

Posteriormente, foi contruída a matriz de prioridade com as prioridades globais para cada um dos critérios definidos. Esta matriz de prioridade encontra-se descrita na Tabela 18 .

Tabela 18 – Matriz com prioridades globais

Previsão	Diferentes tipos de componentes	Templates personalizados	Definição da estrutura de ficheiros
GRC	0,3431	0,2648	0,2737
Plop	0,575	0,6555	0,6393
Divjoy	0,0819	0,0796	0,0869

Por fim, foi calculada a prioridade composta para cada uma das alternativas, através da multiplicação da matriz com as prioridades globais, com os pesos calculados na normalização da comparação entre os critérios, obtendo-se como resultado a Tabela 19.

Tabela 19 – Prioridade composta para cada alternativa

GRC	0,287911
Plop	0,63106
Divjoy	0,080957

Após este cálculo pode verificar-se que a alternativa Plop é a mais indicada, em função dos critérios definidos e as suas respetivas importâncias.

4 Análise e desenho da solução

No presente capítulo apresenta-se a análise do problema descrito na secção 1.2. Numa primeira fase realiza-se uma breve análise do domínio do problema. Posteriormente, encontra-se uma especificação dos requisitos resultantes do problema, funcionais e não funcionais. Após a fase da análise do problema é possível definir o desenho da solução a desenvolver, apresentando diagramas UML que permitem compreender como o sistema será implementado.

4.1 Domínio do problema

O modelo de domínio é uma maneira de representar as entidades do mundo real e as relações existentes entre as mesmas [23]. Estas entidades são responsáveis pela representação do domínio do problema. O modelo de domínio resulta de uma compreensão dos requisitos a nível do sistema, a identificação das entidades e a representação das relações entre elas fornece uma base para a compreensão do problema.

O modelo representado na Figura 21, resulta da análise do problema e identifica as respetivas entidades de negócio e as suas relações.

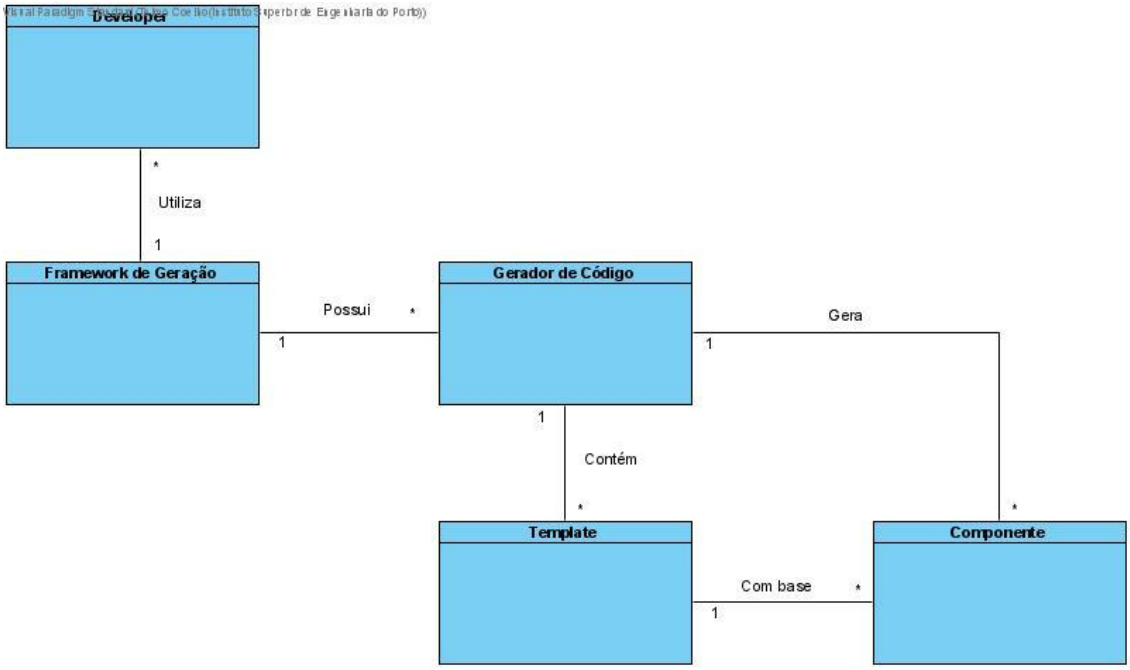


Figura 21 – Modelo de Domínio

Resultante da análise do problema é possível identificar que os utilizadores, do protótipo de *framework* de geração de código, são os diversos elementos das equipas de desenvolvimento.

O principal objetivo desta framework é a geração de React *Journeys*, que basicamente são um conjunto de componentes React. Para a geração das componentes React utilizam-se *templates,* que basicamente funcionam como um padrão para um tipo especifico de componente, ou seja, para cada tipo de componente existe um *template* associado.

No entanto, este protótipo de framework deve ser visto como algo mais genérico podendo adotar novos geradores futuramente.

4.2 Requisitos funcionais e não funcionais

Quando o problema apresentado é bastante complexo, a sua compreensão clara e correta torna-se bastante complicada, especialmente se o sistema for novo, como é o caso. Neste caso a identificação clara do que o sistema deve fazer, as suas funções e as suas restrições, tomam um papel fundamental no desenvolvimento da aplicação em questão.

Nesta secção apresentam-se detalhadamente todos os requisitos do sistema desenvolvido, funcionais e não funcionais.

4.2.1 Requisitos funcionais

Os requisitos funcionais do sistema representam funções que o sistema deve ser capaz de realizar [24]. Estes requisitos são responsáveis pela definição do comportamento do sistema,

isto é, representam o processo que o software efetua sobre entradas para gerar saídas. No presente documento, os requisitos funcionais do sistema foram representados através de casos de uso. Desta maneira, é possível perceber de que forma o utilizador interage com o sistema para atingir um determinado objetivo.

Uma representação gráfica, muito utilizada na identificação dos requisitos funcionais de um projeto é o diagrama de casos de uso [24]. Este diagrama corresponde a uma visão externa do sistema, representando graficamente os atores, os casos de uso e as respetivas relações entre os mesmos.

A Figura 22 representa o diagrama de casos de uso correspondente ao problema apresentado, sendo assim identificados os utilizadores e as suas respetivas ações.

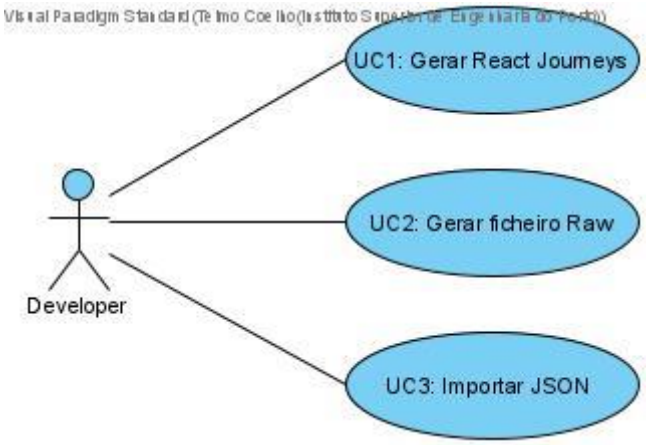


Figura 22 – Diagrama de Casos de Uso

Para os casos de uso identificados, segue-se uma breve descrição do mesmos, seguido do seu respetivo formato completo.

4.2.1.1 UC1: Gerar React Journeys

O presente caso de uso pode ser realizado por qualquer *developer* da organização. Este caso de uso permite aos *developers* especificarem um conjunto de componentes React, que pretendem gerar e as suas respetivas características, bem como relação entre essas componentes. Seguidamente o sistema é responsável por disponibilizar um conjunto de ficheiros de acordo com as especificações impostas, no início do processo, pelo utilizador.

Na Tabela 20 encontra-se uma especificação mais aprofundada do respetivo caso de uso através da apresentação do formato completo.

Tabela 20 – UC1: Gerar React Journeys

ARTEFACTO	DESCRIÇÃO
ATOR	*Developer.*
INTERESSE	Pretende gerar React Journeys de acordo com os dados de entrada fornecidos.
PRÉ-CONDIÇÕES	O sistema tem que ter as templates previamente configuradas.
PÓS-CONDIÇÕES	Os ficheiros gerados são disponibilizados ao developer; No final de todo o processo os ficheiros gerados são eliminados do sistema.
FLUXO BÁSICO	1. O *developer* inicia o processo de geração de uma React Journey; 2. O sistema solicita a inserção dos dados necessários; 3. O *developer* insere os dados de acordo com o que pretende gerar; 4. O sistema mostra resumo e solicita confirmação; 5. O *developer* confirma os dados inseridos; 6. O sistema retorna os ficheiros gerados.
FLUXO ALTERNATIVO	5a. O *developer* pretende alterar os dados inseridos; 1. O sistema apresenta os dados atuais e solicita alteração aos mesmos; 2. O *developer* realiza alterações aos dados inseridos; 3. O sistema solicita confirmação dos dados; 4. O *developer* confirma os dados inseridos.

4.2.1.2 UC2: Gerar ficheiro JSON

A presente funcionalidade pode ser realizada por qualquer *developer* da organização. Este caso de uso permite aos *developers*, após especificarem as componentes de uma determinada React *journey* e as suas respetivas propriedades, fazer o *download* de um ficheiro JSON com toda a informação às componentes descritas. Posteriormente esse ficheiro pode ser importado para o sistema desenvolvido.

Com o intuito de perceber de forma mais aprofundada a finalidade do presente caso de uso, na Tabela 21 encontra-se uma representação do formato completo do mesmo.

Tabela 21 – UC2: Gerar ficheiro JSON

ARTEFACTO	DESCRIÇÃO
ATOR	*Developer.*
INTERESSE	Pretende fazer o *download* de um ficheiro JSON com a informação relativa a uma React Journey especificada.
PRÉ-CONDIÇÕES	O sistema tem que ter as templates previamente configuradas.

ARTEFACTO	DESCRIÇÃO
PÓS-CONDIÇÕES	O ficheiro gerado é disponibilizado ao *developer;* No final de todo o processo o ficheiro gerado é eliminado do sistema.
FLUXO BÁSICO	1. O developer inicia o processo de geração de um ficheiro JSON; 2. O sistema solicita a inserção de dados; 3. O developer insere dados e solicita a geração do ficheiro JSON; 4. O sistema retorna o ficheiro JSON gerado.
FLUXO ALTERNATIVO	-

4.2.1.3 UC3: Importar JSON

O presente caso de uso permite ao *developer* fazer a importação de uma React Journey anteriormente desenvolvida ou até mesmo uma nova através de um JSON. Após esse ficheiro ser importado é possível verificar graficamente a disposição hierárquica das componentes disponibilizadas nesse JSON com as respetivas propriedades associadas.

Para uma melhor compreensão do presente caso de uso, na Tabela 22, encontra-se uma especificação mais aprofundada do mesmo, através da representação do seu formato completo.

Tabela 22 – UC3: Importar JSON

ARTEFACTO	DESCRIÇÃO
ATOR	*Developer.*
INTERESSE	Pretende importar para o sistema um JSON, para que possa visualizar graficamente as componentes lá descritas e editar a React Journey graficamente.
PRÉ-CONDIÇÕES	O sistema tem que ter as templates previamente configuradas.
PÓS-CONDIÇÕES	As componentes descritas no JSON importado devem estar presentes na interface gráfica.
FLUXO BÁSICO	1. O developer inicia o processo de importação de um JSON 2. O sistema solicita o JSON 3. O developer introduz o JSON pretendido 4. O sistema disponibiliza graficamente as componentes descritas no JSON importado
FLUXO ALTERNATIVO	3a. O *developer* solicita o cancelamento da operação • O caso de uso termina

4.2.2 Requisitos não funcionais

A identificação de requisitos não funcionais foi baseada no modelo FURPS+ [25]. Este modelo resulta da atualização do modelo FURPS [26], que classificava os requisitos em cinco categorias distintas, sendo elas:

- Funcionalidade (Functionality)
- Usabilidade (Usability)
- Confiabilidade (Reliability)
- Desempenho (Performence)
- Suportabilidade (Supportability)

Após a atualização do modelo FURPS, este passou a ser designado por FURPS+, passando a existir assim, mais uma categoria para a classificação dos requisitos não funcionais. O "+" permite identificar e especificar as restrições impostas ao sistema.

4.2.2.1 Funcionalidade

Esta categoria permite identificar funcionalidades que não se relacionam com os casos de uso. Relativamente ao projeto desenvolvido foi identificado o seguinte requisito de funcionalidade:

- O gerador de código deve obter a informação do que gerar através de um JSON.

4.2.2.2 Usabilidade

O parâmetro de usabilidade é responsável pela avaliação da interface, ou seja, todos os requisitos que dizem respeito à interação do utilizador com o sistema. Para o projeto apresentado foi identificado o seguinte requisito de usabilidade:

- Pretende-se que a aplicação desenvolvida possua uma interface simples, que seja facilmente utilizada pelos utilizadores da mesma.

4.2.2.3 Confiabilidade

A confiabilidade do sistema refere-se à integridade e conformidade do mesmo. Nesta categoria os requisitos a serem considerados são: frequência e gravidade de falhas, possibilidade de recuperação e previsibilidade. Para o projeto desenvolvido não foram identificados quaisquer tipo de requisitos não funcionais de confiabilidade.´

4.2.2.4 Desempenho

Esta categoria é responsável dos requisitos responsáveis pela avaliação do desempenho do software, nomeadamente: tempo de resposta, consumo de recursos e capacidade. Para o presente projeto não foram identificados requisitos não funcionais de desempenho.

4.2.2.5 Suportabilidade

O parâmetro de suportabilidade aborda e agrupa vários tipos de requisitos, tais como: requisitos de testabilidade, adaptabilidade, manutenibilidade, compatibilidade, escalabilidade entre outros. Para o problema descrito foram identificados os seguintes requisitos de suportabilidade:

- Deve ser assegurada a escalabilidade do sistema;
- O sistema deve permitir uma fácil integração de novos componentes em momentos futuros.

4.2.2.6 Outros (+)

A adição deste parâmetro permite a distribuição de novos requisitos não funcionais responsáveis por assegurar a qualidade de software. Este contém quatro categorias sendo estas: restrições de design, requisitos de implementação, requisitos de interface e requisitos físicos. Face ao problema apresentado, foram identificados os seguintes requisitos "+":

- A aplicação deve ser desenvolvida através de processos iterativos e incrementais;
- Aplicação de boas práticas de design, como por exemplo, adoção dos padrões SOLID e GRASP.

4.3 Desenho

O presente subcapítulo está dividido em duas partes, sendo estas, granularidade de sistema e granularidade de aplicação. O objetivo é perceber de que forma a aplicação está estruturada, apresentando assim, para cada uma das partes, as vistas lógicas, vistas de implementação, vistas de implantação e vistas de cenários das mesmas.

4.3.1 Granularidade do sistema

A análise com uma granularidade de sistema é uma análise mais superficial ao problema apresentado inicialmente onde são analisadas as várias vistas: lógica, implementação, implantação e cenários.

4.3.1.1 Vista Lógica

O principal objetivo do diagrama de componentes é ilustrar a relação entre as diversas componentes do sistema e as suas respetivas interfaces. Na Figura 23 encontra-se a representação do diagrama de componentes com granularidade de sistema para o problema apresentado. Para a granularidade de sistema destacam-se duas grandes componentes, que funcionam com aplicações destintas. Sendo que, uma das aplicações é responsável por toda a camada de apresentação (PresentationLayer) e outra é responsável por conter toda a lógica de geração de código (CodeGenerator).

A separação em duas aplicações distintas pode acrescentar algumas vantagens ao sistema, uma vez que, como o módulo de geração de código funciona como uma aplicação isolada e, que por sua vez, pode ser adotado em aplicações futuras da organização.

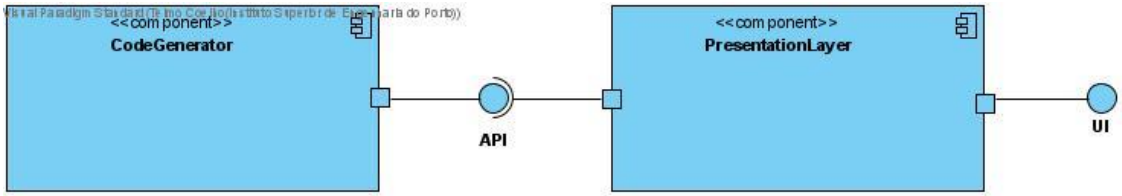


Figura 23 – Vista Lógica de Sistema

4.3.1.2 Vista de Implementação

Na Figura 24 encontra-se uma representação da estrutura do sistema implementado, face ao problema apresentado. O diagrama de packages divide o sistema em agrupamentos lógicos mostrando também as suas dependências. Para a granularidade de sistema destacam-se dois packages, um responsável pela camada de apresentação (PresentationLayer), ou seja toda a componente visual do sistema desenvolvido encontra-se neste package, e outro responsável por toda a lógica de negócio (CodeGenerator).

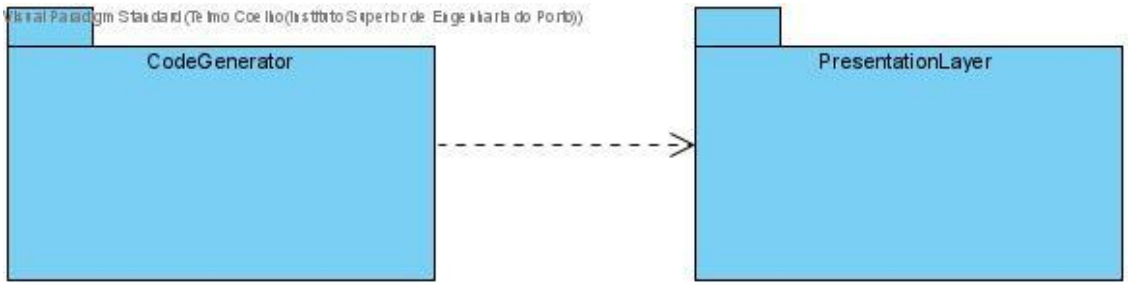


Figura 24 – Vista de Implementação de Sistema

4.3.1.3 Vista de Implantação

O diagrama de implantação da aplicação mapeia a arquitetura do hardware de acordo com as necessidades do software a ser implantado. Prevê-se que existam dois servidores distintos, um responsável pelo alojamento da camada de apresentação e outro responsável pelo alojamento do gerador de código.

A separação em duas componentes físicas diferentes, acrescenta algumas vantagens ao sistema. Esta separação permite que os servidores não compitam pelos mesmos recursos, assim sendo, isto implica que cargas altas num servidor não interfira no desempenho do outro. Para além disso, a separação em duas componentes físicas diferentes assegura a escalabilidade do sistema desenvolvido, o que faz com que a integração de módulos futuros seja mais fácil de implementar.

Na Figura 25 estão representadas as dependências entre as respetivas componentes do sistema. É possível também observar onde se alojam fisicamente os nós e como é assegurada a sua comunicação.

Figura 25 – Vista de Implantação

4.3.1.4 Vista de Cenários

A vista de cenários apresentada permite a compreensão do funcionamento das componentes anteriormente descritos.

A Figura 26 é uma representação da interação das componentes para gerar React Journeys.

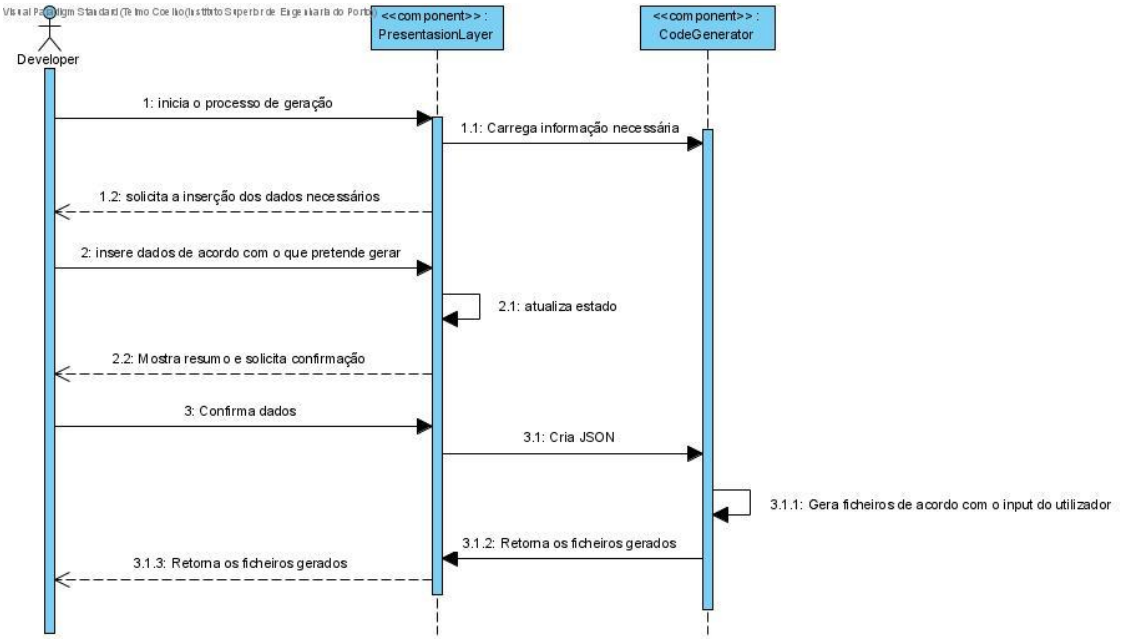


Figura 26 – Vista de Cenários de Sistema

4.3.2 Granularidade de Aplicação

A presente subsecção destina-se à representação, com um nível mais pormenorizado, das situações anteriores, onde são analisadas a vista lógica e as vistas de cenários para os diferentes casos de uso.

4.3.2.1 Vista lógica

A aplicação foi desenvolvida seguindo uma arquitetura em camadas. Deste modo é possível efetuar uma separação evidente das responsabilidades. Devido às restrições impostas, foi perspetivada uma arquitetura por camadas , sendo assim desenvolvida conforme a Figura 27. Deste modo, é possível a identificação das seguintes camadas:

PresentationLayer – Esta é a camada responsável pela interação com o utilizador, ou seja, diz respeito a toda a componente visual da aplicação. É responsável por recolher o pedido feito pelo utilizador, com a informação relativa ao que pretende gerar. Esta camada é constituída por duas componentes, a componente *App* que é responsável por mapear a arvore hierárquica de componentes a serem renderizadas e a componente *Components,* que basicamente permite dividir a interface gráfica em partes lógicas independentes e reutilizáveis.

Controller – Esta camada tem como objetivo assegurar a interação do utilizador com o sistema, ou seja, esta é a camada que funciona como ponto de ligação entre a camada de apresentação e o sistema desenvolvido. Deste modo podemos afirmar, que esta camada é responsável por determinar como o sistema deve responder de acordo com a ação despoletada pelo utilizador.

Services – Esta é a camada que contêm toda a lógica de negócio do sistema desenvolvido. É responsável por receber os ficheiros gerados e repassá-los para a camada *Controller*. Esta camada é também responsável pela validação dos dados de acordo com as regras de negócio definidas e também é responsável pela transformação dos dados recebidos num ficheiro JSON, para que este possa ser interpretados pelo gerador.

Generator – Esta camada é responsável por conter o domínio da aplicação. Contem toda a lógica de geração de código, presente na componente *Plopfile,* que faz uso das *Templates* que servem como *standards,* ou seja, para cada tipo de componente React que o sistema permite gerar deve existir uma *Template* associada. Deste modo também é assegurado que todas as componentes do mesmo tipo seguem o mesmo padrão. Os ficheiros são gerados de acordo com as restrições impostas inicialmente pelo utilizador.

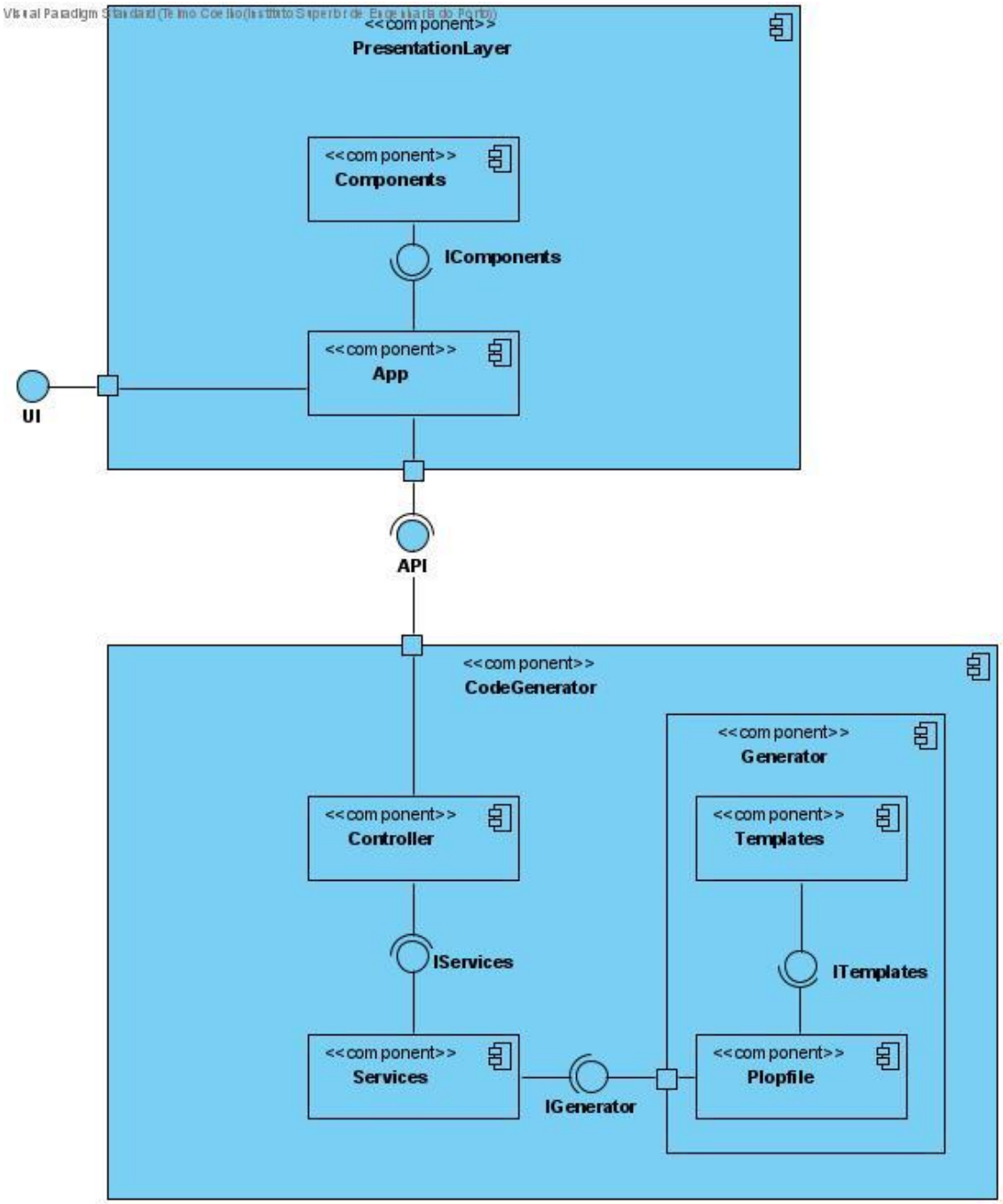


Figura 27 – Vista Lógica de Aplicação

4.3.2.2 Vista de Cenários

Na presente subsecção são apresentadas as vistas de cenários paras os casos de uso desenvolvidos. Em todos os diagramas apresentados nesta secção, a componente *App* é responsável pela apresentação dos dados. A ligação entre a camada de apresentação e o sistema desenvolvido é assegurada através do *Controller*, que define de que maneira o sistema deve responder a uma determinada interação com o utilizador. Por sua vez o Controller interage com os respetivos *Services,* onde se encontra toda a lógica de negócio. Quando necessário, a componente *Service* interage com a componente Plopfile, componente responsável por toda a lógica de geração das React Journeys, que faz uso da componente *Templates*, uma vez que são estas que servem como padrão para cada tipo de componente definido no sistema.

UC1: Gerar React Journey

Na figura Figura 28 encontra-se uma representação gráfica do diagrama de sequência para o caso de uso de geração de uma React Journey.

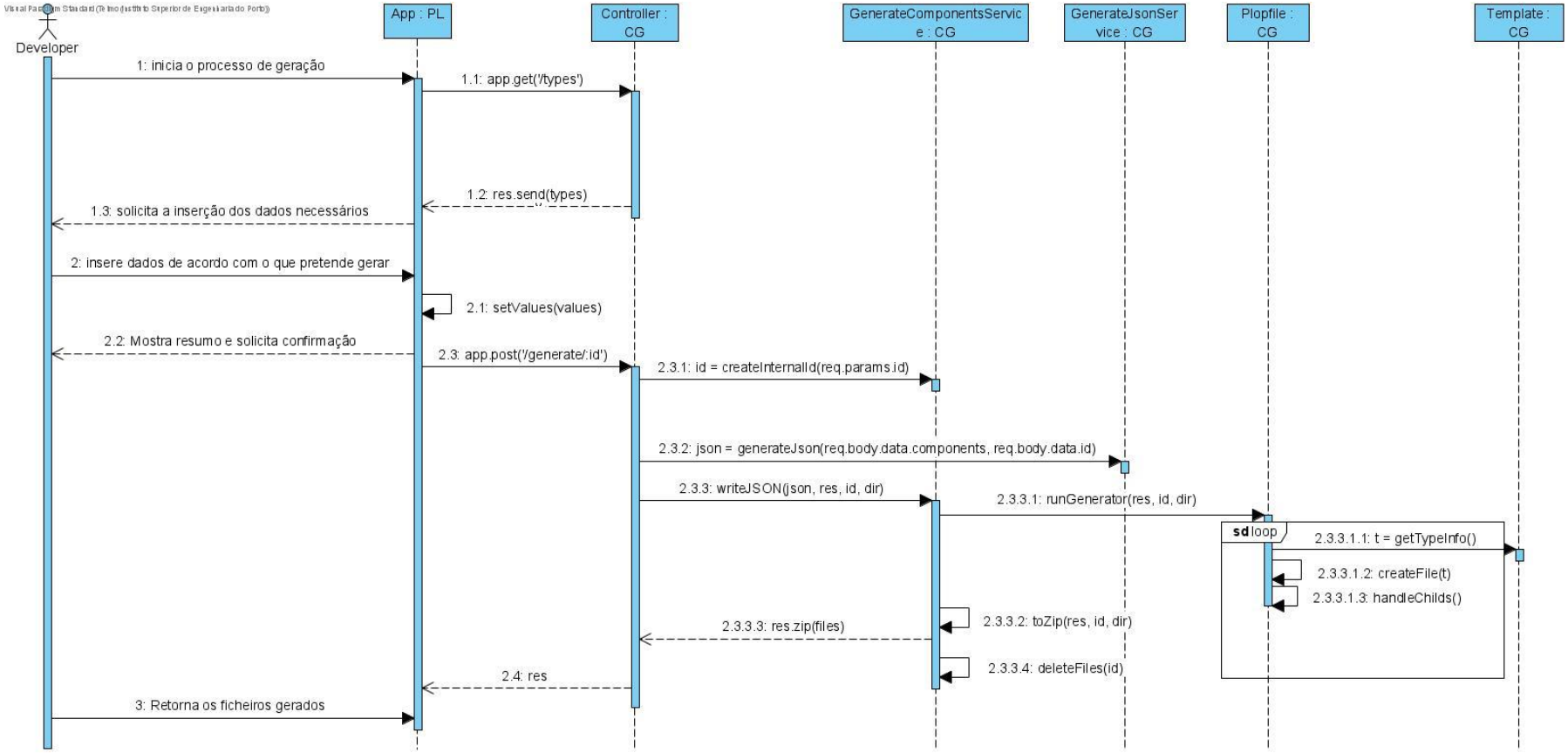


Figura 28 - Diagrama de Sequência UC1

UC2: Gerar Ficheiro JSON

A figura Figura 29 corresponde à representação do diagrama de sequência para o caso de uso de geração de um ficheiro JSON, com a informação relativa a uma React Journey.

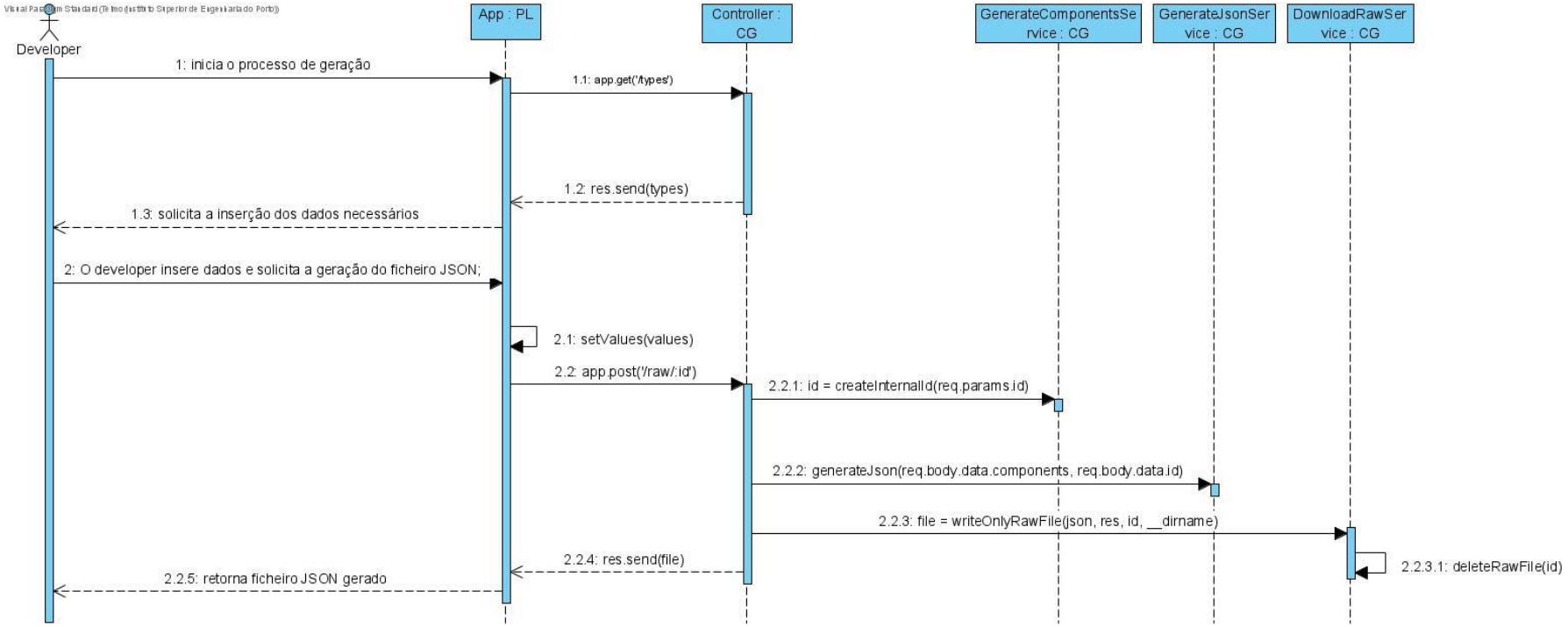


Figura 29 – Diagrama de Sequência UC2

UC3: Importar JSON

Na figura Figura 30 encontra-se uma representação gráfica do diagrama de sequência para a importação de um JSON. Este JSON deve conter a informação relativa a uma React Journey, para que esta, possa ser visualizada graficamente, através do sistema desenvolvido, e posteriormente editada.

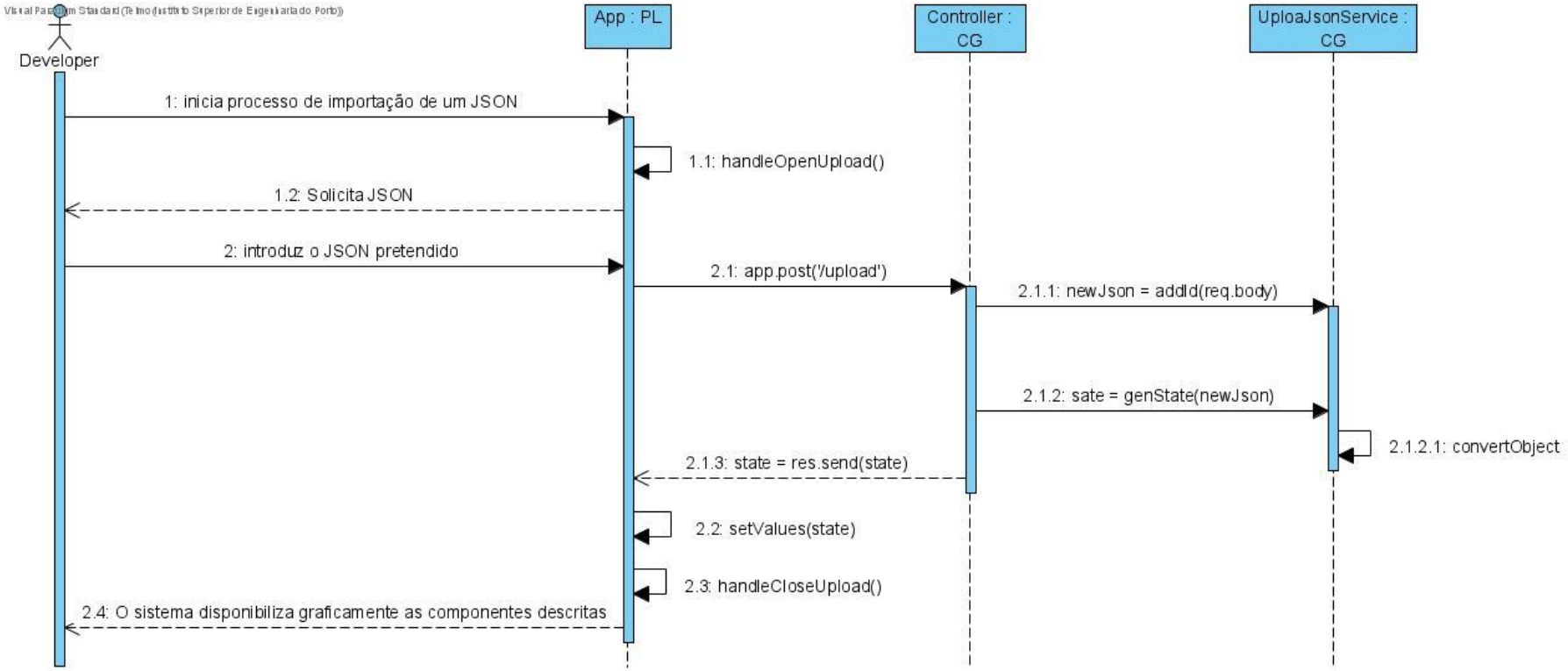


Figura 30 – Diagrama de Sequência UC3

5 Implementação da Solução

No presente capítulo encontra-se a descrição da implementação da solução desenvolvida. Para além disso é realizada, neste capítulo uma análise crítica ao trabalho desenvolvido.

5.1 Descrição da implementação

No presente subcapítulo encontra-se uma descrição detalhada da implementação da solução. Este subcapítulo encontra-se dividido em secções, descrevendo os principais aspetos da aplicação desenvolvida. Para facilitar o entendimento do trabalho realizado serão apresentados alguns excertos de código e imagens da aplicação desenvolvida.

É importante realçar que o sistema desenvolvido se encontra dividido em duas grandes aplicações. Uma responsável pela camada de apresentação, ou seja, toda a interação com o utilizador é realizada através desta aplicação e outra que contém toda a lógica de geração das React Journeys, ou seja, há uma separação entre Frontend e Backend.

Para a geração das React Journeys o sistema desenvolvido faz uso de um ficheiro JSON, onde é possível especificar a divisão hierárquica das componentes da respetiva React Journey, bem como, definir as propriedades dessas mesmas componentes. Uma vez que uma das funcionalidades requeridas é a importação de uma React Journey diretamente de um ficheiro JSON, é importante que a definição deste tipo de ficheiro seja facilmente compreendida pelos diversos utilizadores do sistema desenvolvido. Este ficheiro assume um papel relevante no desenvolvimento do protótipo da *framework* de geração de código, uma vez que muita da lógica de geração depende da definição deste mesmo ficheiro.

Na Figura 31 encontra-se um exemplo de ficheiro JSON que é interpretado pelo sistema desenvolvido.

```
{
    "alias": "ComponentePai",
    "compType": "wizard",
    "imported": false,
    "type": "",
    "library": "",
    "childs": [
        {
            "alias": "ComponenteFilho1",
            "compType": "step",
            "imported": false,
            "type": "",
            "library": "",
            "childs": [
                {
                    "alias": "Componente",
                    "compType": "",
                    "imported": true,
                    "type": "TextField",
                    "library": "material-ui",
                    "childs": [],
                    "props": [
                        {
                            "name": "prop4",
                            "value": "prop4Value"
                        }
                    ],
                }
            ],
            "props": [
                {
                    "name": "prop2",
                    "value": "prop2Value"
                }
            ],
        },
        {
            "alias": "ComponenteFilho2",
            "compType": "step",
            "imported": false,
            "type": "",
            "library": "",
            "childs": [],
            "props": [
                {
                    "name": "prop3",
                    "value": "Value"
                }
            ],
        }
    ],
    "props": [
        {
            "name": "prop1",
            "value": "prop1Value"
        }
    ],
```


Figura 31 – Exemplo de Ficheiro JSON

Após uma breve análise do exemplo de ficheiro JSON apresentado é possível perceber claramente a disposição hierárquica das componentes que compõem uma React Journey. Neste caso pode afirmar-se que esta React Journey é constituída por uma componente agregadora designada por "ComponentePai", esta componente agregadora, por sua vez, contem duas componentes designadas por "ComponenteFilho1" e "ComponenteFilho2". Por fim ainda podemos verificar que a "ComponenteFilho1" é constituída por uma outra componente designada "Componente".

Para além da disposição hierárquica, através da análise do exemplo de ficheiro JSON pode ainda visualizar-se todas as propriedades referentes a essas mesmas componentes. Com o objetivo de compreender melhor as propriedades apresentadas no ficheiro JSON, na Tabela 23 encontra-se uma breve descrição dessas mesmas propriedades.

Tabela 23 – Propriedades do ficheiro JSON

Propriedade	*Descrição*
alias	É responsável pela nomenclatura do ficheiro da componente correspondente.
compType	É responsável por indicar o tipo de componente.
imported	O sistema desenvolvido permite a importação de componentes de bibliotecas externas, nomeadamente MaterialUI[13]. Esta propriedade é um boolean que nos indica se essa componente é ou não importada de uma biblioteca externa.
type	Caso a componente seja importada esta propriedade indica o tipo de componente que vai ser importada.
library	Caso a componente seja importada esta propriedade indica de qual biblioteca esta componente vai ser importada
childs	Esta propriedade é um Array com a informação relativa aos filhos da respetiva componente
props	Esta propriedade é um Array com a informação relativa às propriedades das componentes. Para cada propriedade é indicado um nome e um valor.

5.1.1 PresentationLayer

A presente secção descreve as principais decisões tomadas na implementação e desenvolvimento da aplicação da camada de apresentação, bem como uma descrição mais aprofundada de alguns dos seus elementos. Esta componente foi desenvolvida com recurso a uma biblioteca de *JavaScript* designada por ReactJs[14], anteriormente analisada no subcapítulo 2.3.1.

[13] https://mui.com/pt/
[14] https://reactjs.org/

5.1.1.1 Componentes React

Como mencionado na subsecção 4.3.2.1 as *Components* (Componentes React), são um dos elementos constituintes da camada de apresentação.

As componentes React permitem dividir a interface gráfica em partes lógicas, tornando cada uma das componentes numa parte independente e reutilizável. Deste modo, é possível manipular cada uma das componentes como um bloco isolado, livre de quaisquer dependências externas.

As componentes React são funções de JavaScript que aceitam dados de entrada, esses dados funcionam como propriedades para a sua construção, e retornam elementos React que descrevem a estrutura da interface gráfica. Estas componentes são também responsáveis pela manipulação dos dados apresentados ao utilizador. Para além disso, são estas componentes que asseguram a interação do utilizador com o sistema. Na Figura 32 encontra-se um excerto de código de uma componente React utilizada no desenvolvimento do sistema.

```
const Import =({type, library, onChangeType, onChangeLibrary}) => {

    const classes = useStyles();

    return (
        <>
            <div className="imported-wrapper">
                <div>
                    <FormControl variant="outlined" className={classes.formControl}>
                        <TextField color="secondary" label="Component Type" id="imported-type" variant="outlined" value={type} onChange={onChangeType}/>
                    </FormControl>
                    <FormControl variant="outlined" className={classes.formControl}>
                        <InputLabel color="secondary" htmlFor="library-label">Library</InputLabel>
                        <Select native id="imported-library" color="secondary" options={libraries} label="Library" value={library} onChange={onChangeLibrary} inputProps={{
                            name: "Library",
                            id: "library-label"
                        }}>
                            {libraries.map(({value,id}) => <option key={id}>{value}</option> )}
                        </Select>
                    </FormControl>
                </div>
            </div>
        </>
    )
}

export default Import
```

Figura 32 – Exemplo de uma componente React

Após uma breve análise da figura pode verificar-se que a componente React, designada por "*Import*". Uma vez que, as componentes React funcionam como funções JavaScript, estas recebem dados de entrada (neste caso "*type*", *"library"*, *"onChangeType"* e *"onChangeLibrary"*) e retorna, elementos React escritos com uma sintaxe muito semelhante ao XML designada por JSX[15], já mencionado no secção 2.3.1.2.

5.1.1.2 Estado React

Um outro elemento constituinte da camada de apresentação é a componente *App*, que funciona também como uma componente React. Esta componente, para além de ser responsável pelo mapeamento da árvore hierárquica das componentes que devem ser

[15] https://reactjs.org/docs/introducing-jsx.html

apresentadas, possui também o estado React responsável por guardar todos os *inputs* do utilizador.

O estado React é um objeto JavaScript que começa com um valor padrão quando um componente é construído. No entanto, ao longo do tempo o estado React de uma componente vai sofrendo alterações, que resultam de eventos despoletados pelos utilizadores do sistema.

Cada componente pode ter o seu próprio estado React, mas no que diz respeito ao problema apresentado, era necessário que a componente agregadora ficasse responsável por guardar todos os *inputs* do utilizador com a informação relativa à React Journey que se pretende gerar. Com toda a informação contida num só objeto, o processo de envio dos dados ao backend tornou-se muito mais simples.

Numa primeira fase, decidiu-se que a estrutura do estado React tivesse uma disposição semelhante à do ficheiro JSON que é interpretado pelo gerador de código desenvolvido. No entanto, a cada alteração que o utilizador realizasse, era necessário iterar recursivamente os filhos da árvore de componentes, que se pretendem gerar, para que se pudesse alterar qualquer valor no estado React. Deste modo, o processo de registo de alterações ao estado das componentes tornava-se complexo.

Com o objetivo de tornar a alteração do estado React mais simples, decidiu-se que o objeto do estado React tivesse a estrutura descrita na Figura 33.

```
{
    "_ltl0alpq7":{
        "alias":"ComponentePai",
        "compType":"wizard",
        "imported":false,
        "type":"",
        "library":"",
        "childs":[
            "_xwbhqrlj8"
        ],
        "props":[
            {
                "name":"prop",
                "value":"prop1"
            }
        ],
        "parentId":""
    },
    "_xwbhqrlj8":{
        "alias":"ComponenteFilho",
        "compType":"step",
        "imported":false,
        "type":"",
        "library":"",
        "childs":[],
        "props":[
            {
                "name":"prop2",
                "value":"prop2"
            }
        ],
        "parentId":"_ltl0alpq7"
    }
}
```


Figura 33 – Estado React

Após uma análise da figura pode verificar-se que o estado React é um Array, no qual cada uma das posições corresponde à informação de uma das componentes que o utilizador pretende gerar. A *tag "*childs" é responsável por armazenar os identificadores das componentes filho de cada uma das componentes. Numa fase posterior do processo de geração o backend faz uso desta *tag* "childs" para a construção do JSON que é interpretado pelo gerador.

5.1.1.3 Decisões de interface

Um dos objetivos da aplicação desenvolvida é que esta possua uma interface simples e intuitiva. Para isso, é necessário que o utilizador consiga perceber corretamente a hierarquia da árvore de componentes, pertencentes à React Journey que pretende gerar. Para isso, cada vez que o utilizador adicione um filho a uma das componentes da React Journey, aparece um novo painel por baixo dessa mesma componente com uma ligeira indentação à direita, dando a entender que a componente em baixo pertence à de cima (Figura 34).

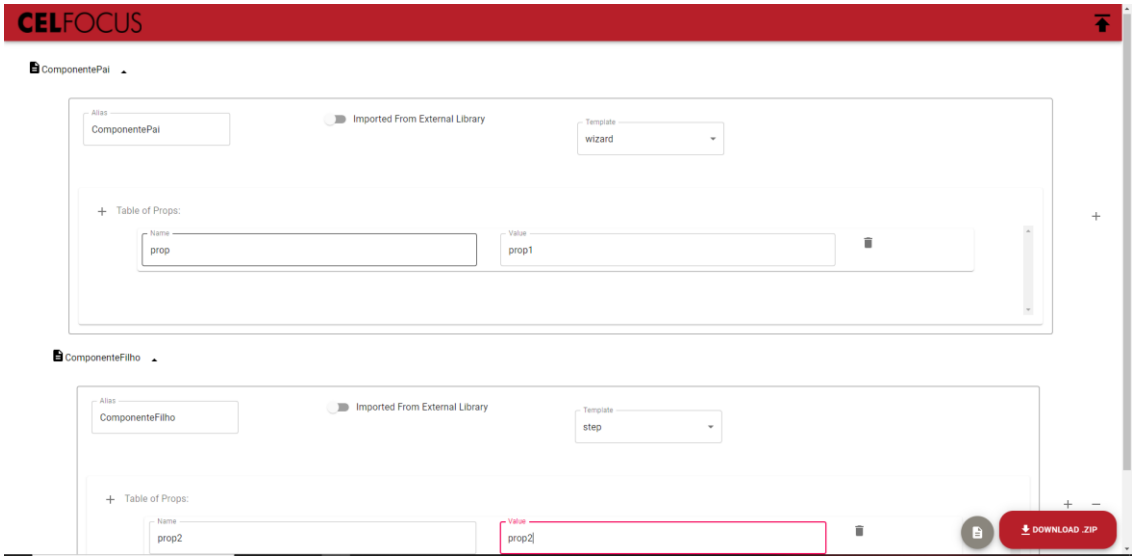


Figura 34 – Visualização da árvore hierárquica

Para além disso é permitido ao utilizador agrupar as diversas componentes da React Journey, tornando mais fácil a visualização da árvore hierárquica (Figura 35).

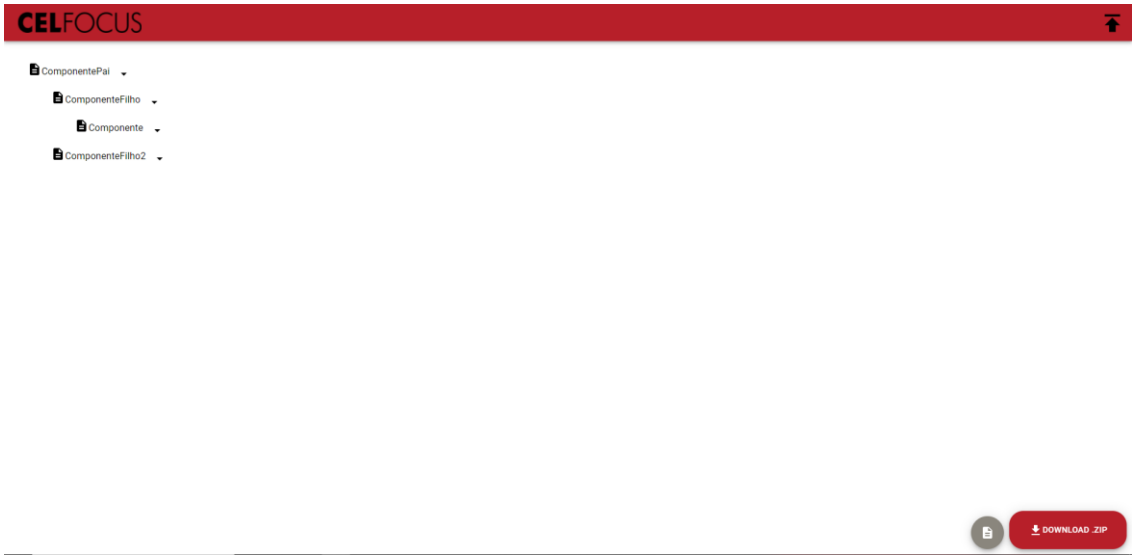


Figura 35 – Visualização da árvore hierárquica colapsada

5.1.2 Code Generator

A presente secção descreve as principais decisões tomadas na implementação e desenvolvimento da aplicação responsável por conter toda a lógica de geração, bem como uma

descrição mais aprofundada de alguns dos seus elementos. Esta componente foi desenvolvida em *NodeJs*[16].

5.1.2.1 Controller

O Controller funciona como uma camada intermediadora entre a camada de apresentação e o sistema desenvolvido. Esta componente é responsável por determinar como a aplicação deve responder mediante as ações do utilizador.

Para o desenvolvimento desta componente foi utilizada uma framework para aplicações web em *NodeJs*, designada por *Express*[17]. Esta framework disponibiliza um conjunto de recursos robustos, que podem ser utilizados na construção de aplicações *Web*, nomeadamente métodos *Hypertext Transfer Protocol* (HTTP). Através desta framework é possível contruir uma *Application Programming Interface* (API) de um modo simples e fácil.

Na Figura 36 pode verificar-se alguns métodos HTTP definidos, no sistema desenvolvido.

```
app.post('/generate/:id', async function(req, res) {
  let requestParams = req.params;
  if(!requestParams || !requestParams.id){
    console.error(`Bad request`);
    return;
  }
  let id = generateComponentsService.createInternalId(requestParams.id);

  let json = generateJsonService.generateJson(req.body.data.components, req.body.data.id);

  generateComponentsService.writeJSON(json, res, id, __dirname)

})

app.post('/upload', async function(req, res) {
  let stateDefault = {};
  const object1 = uploadJsonService.addId(req.body);
  const parentId = object1.id
  const firstId = '';
  const state = uploadJsonService.genState(object1, stateDefault, firstId);
  res.setHeader('Content-Type', 'application/json');
  res.send({
    "parentId": parentId,
    "state": state
  })
})
```

Figura 36 – Exemplo de métodos HTTP do Controller

[16] https://nodejs.org/en/
[17] https://expressjs.com/

5.1.2.2 Services

Os *Services* são responsáveis pela manipulação dos dados recebidos através do *Controller*, ou seja, esta é a componente que disponibiliza funcionalidades que são comuns a vários locais da aplicação. Para além disso é através desta componente que é realizada a comunicação com a componente de geração de código, sempre que necessário. Na Figura 37, encontra-se um exemplo de uma implementação de um serviço responsável pela transformação do estado React recebido da camada de apresentação num objeto JSON que pode ser interpretado pelo gerador de código.

```
/*This function recives the state from React and transforms it into a JSON
that can be used by generator */
module.exports = {
    generateJson: function (components, parentId) {
        const localValues = {...components};
        const parent = localValues[parentId];
        const json = generateJSONAux(parent, components);
        return json;
    }
};

function generateJSONAux (parent, components) {
    const localValues = {...components};
    for(var i = 0; i < parent.childs.length; ++i){
      var child = parent.childs[i];
      parent.childs[i] = localValues[child];
      generateJSONAux(parent.childs[i], components);
    }
    return parent;
};
```

Figura 37 – Exemplo de uma implementação de um serviço

5.1.2.3 Plopfile

A *Plopfile* é uma componente que tem por base *JavaScipt* e que faz uso de uma ferramenta designada por Plop[18], descrita previamente na secção 2.4.2. Esta é a componente que contem toda a lógica de geração da React Journey. Para o processo de geração, esta componente faz uso das *Templates* previamente definidas no sistema, criando um conjunto de componentes React de acordo com o *input* fornecido pelo utilizador inicialmente.

Esta componente recebe o caminho do ficheiro JSON (com uma estrutura igual à descrita no subcapítulo 5.1) gerado pela camada de *Services* e é responsável pela interpretação do mesmo. Para tal, esta componente percorre recursivamente a árvore hierárquica de componentes React

[18] https://plopjs.com/

que o utilizador pretende gerar, criando, para cada uma delas, um ficheiro com base na *tag* "*type"* descrita no ficheiro JSON. Sendo que, cada "type" tem uma *Template* associada.

Para além disso para cada uma das componentes React que o utilizador pretende gerar, esta componente é responsável por criar as variáveis que são necessárias substituir nas *Templates* de acordo com o *input* do utilizador.

Na Figura 38 encontra-se a configuração do Plop para o projeto desenvolvido.

```
module.exports = function (plop) {

    plop.setActionType('customAction', function (answers, config, plop) {
        const {path} = answers;
        const {id} = answers;
        const {compType, alias, props, childs} = require(path);
        const typeInfo = getTypeInfo(compType, alias, id);
        createFile(plop,typeInfo.dir, typeInfo.templateFile, typeInfo.path, props, childs, id, alias);
        handleChilds(plop, childs, id);
    });

    plop.setGenerator('generator', {
        prompts: [
            {
                type: 'input',
                name: 'path',
            },
            {
                type: 'input',
                name: 'id',
            }
        ],
        actions: [
            {
                type: 'customAction',
                speed: 'slow'
            }
        ],
    });
};
```

Figura 38 - Configuração do Plop

Após uma breve análise da figura apresentada pode verificar-se que o Plop recebe como parâmetro o caminho para o ficheiro JSON e um id, que funciona como um identificador único que é utilizado para gerar o código relativo a um utilizador num *package* específico. Deste modo, é possível evitar que o código gerado sobrescreva código já existente, para além disso é através deste id que o sistema sabe quais as componentes a retornar ao utilizador.

Ainda sobre a análise da figura, é possível identificar que o Plop faz uso de uma ação personalizada designada por "*customAction"* que é responsável pela interpretação da componente pai da React Journey descrita no ficheiro JSON recebido. Posteriormente, esta

ação percorre recursivamente os filhos das diversas componentes criando um ficheiro para cada uma delas.

5.1.2.4 Templates

Os *Templates* funcionam como ficheiros modelo, que têm a extensão *handelbars* (*hbs). Estes ficheiros pegam num conjunto de variáveis de *input* e transformam-nas numa saída de texto. Como o próprio nome da extensão sugere, os marcadores de posição de cada uma das variáveis aparecem no *Template* do seguinte modo: {{variável}}.

Os valores das variáveis são substituídos pelos valores fornecidos à medida que o script é chamado.

A Figura 39, ilustra um exemplo de um *Template* utilizado no desenvolvimento do sistema.

```
import React from 'react';
{{{Imports}}}

const {{Alias}} = props => {
  return (
    <div>
      <span>step</span>
      {{{Childrens}}}
    </div>
  );
};

export default {{Alias}};
```

Figura 39 – Exemplo de um *Template*

Após uma breve análise da figura apresentada pode verificar-se que existem três variáveis que necessitam de ser carregadas (*Alias, Imports e Childrens)*. Como se trata de um protótipo as templates definidas ainda são bastante simples. O objetivo ao longo do tempo é enriquecer o gerador com novas templates, mais robustas, que possam ser utilizadas na integração futura nos projetos da organização.

5.1.3 Conclusões

A separação de responsabilidades, obtida através da estrutura implementada, permite que, futuramente, o desenvolvimento e implementação de novas funcionalidades, bem como a correção de possíveis erros seja assegurada de forma mais fácil.

A aplicação desenvolvida, apesar de ser um protótipo, tem uma base sólida. No entanto, num futuro próximo, serão realizadas algumas alterações e serão adicionadas novas funcionalidades. A utilização de boas práticas de desenvolvimento de software, contribui para que haja uma

menor complexidade do projeto desenvolvido, tornando este processo, de alteração e adição de novas funcionalidades, bastante mais simples.

6 Experimentação e avaliação

No presente capítulo encontra-se descrito, para o problema e objetivos definidos, quais foram os indicadores, hipóteses, metodologia de avaliação e resultados que representaram a base para a avaliação da solução implementada.

6.1 Hipóteses e indicadores

Com o objetivo de validar se a solução apresentada poderá vir a ser ou não uma mais valia para a organização para a qual foi desenvolvida, é necessário definir os indicadores sobre os quais a solução vai ser avaliada. No caso abordado neste documento os indicadores definidos para a avaliação são:

- Usabilidade – com este indicador pretende-se definir o grau de facilidade que utilizador tem ao utilizar o sistema desenvolvido;
- Funcionalidade – através deste indicador pretende-se perceber se as funcionalidades do sistema desenvolvido vão de encontro aos objetivos definidos inicialmente e às expectativas dos utilizadores;
- Desempenho – este indicador permite perceber se o tempo de resposta do sistema desenvolvido é aceitável;
- Utilidade – através deste indicador é possível definir se o sistema desenvolvido acrescenta alguma vantagem ao quotidiano dos colaboradores da organização.

As hipóteses que se pretendem testar são: (i) perceber o grau de satisfação dos utilizadores face à interface do sistema desenvolvido e (ii) perceber se a solução desenvolvida pode ter um impacto positivo no auxílio das tarefas diárias dos colaboradores da organização.

6.2 Metodologia de avaliação

A avaliação da solução apresentada foi realizada com o recurso a um inquérito de satisfação e usabilidade (consultar Anexo A) , que se destina a alguns dos *developers* que fazem parte da organização para a qual o protótipo foi desenvolvido. A escolha desta metodologia deve-se ao facto de o projeto desenvolvido ser um projeto interno, tornando a opinião e o grau de satisfação de possíveis futuros utilizadores muito relevante no processo de avaliação da solução.

O questionário elaborado para a avaliação do protótipo desenvolvido encontra-se dividido pelos quatro indicadores, apresentados no subcapítulo 6.1, e é constituído por 17 perguntas. Na Tabela 24 encontram-se descritas as perguntas do questionário de avaliação.

Tabela 24 – Perguntas do questionário de avaliação

Indicador	Pergunta
Usabilidade	1 – A interface é simples?
	2 – O layout do sistema desenvolvido é consistente?
	3 – As cores utilizadas são adequadas?
	4 – A interface é atrativa?
	5 – A interface é intuitiva?
	6 – O utilizador consegue adaptar-se facilmente à interface desenvolvida?
	7 – O utilizador consegue familiarizar-se com as funcionalidades desenvolvidas?
	8 – A interface desenvolvida contribui para o uso continuo da plataforma desenvolvida?
Funcionalidade	9 – O utilizador consegue realizar a geração de uma React Journey?
	10 – A funcionalidade de geração de uma React Journey vai de encontro às suas necessidades?
	11 – O utilizador consegue fazer o download de um JSON relativo a uma React Journey?
	12 – A funcionalidade de download de um JSON vai de encontro às suas necessidades?
	13 – O utilizador consegue fazer a importação de um JSON relativo a uma React Journey?
	14 – A funcionalidade de importação de um JSON relativo a uma React Journey vai de encontro às suas necessidades?
Desempenho	15 – O desempenho do sistema desenvolvido é aceitável?
Utilidade	16 – O sistema desenvolvido poderá ser útil para a sua carreira profissional?
	17 – O sistema desenvolvido poderá ter um impacto positivo nas suas tarefas diárias?

Para a resposta a cada questão, foi utilizada uma escala de avaliação constituída por cinco níveis, também conhecida por escala de *Likert*[19], representada na Tabela 25.

Tabela 25 – Escala de classificação utilizada no inquérito

ESCALA	DESCRIÇÃO
1	Discordo totalmente
2	Discordo
3	Nem concordo, nem discordo
4	Concordo
5	Concordo totalmente

6.3 Resultados do Inquérito

A análise dos resultados obtidos toma um papel fundamental no que diz respeito à avaliação da solução desenvolvida. Os dados obtidos através do questionário de avaliação permitem perceber se a aplicação desenvolvida vai de encontro a alguns dos objetivos definidos inicialmente, para além disso, são responsáveis para validar as hipóteses definidas anteriormente.

Ao inquérito realizado foram obtidas 10 respostas e os resultados encontram-se representados na Tabela 26.

Tabela 26 – Resultados do questionário de avaliação

Pergunta	*Classificação*				
	1	*2*	*3*	*4*	*5*
1	0%	0%	0%	40%	60%
2	0%	0%	0%	40%	60%
3	0%	0%	10%	70%	20%
4	0%	0%	0%	60%	40%
5	0%	0%	0%	60%	40%
6	0%	0%	10%	50%	40%
7	0%	0%	10%	50%	40%
8	0%	0%	0%	60%	40%
9	0%	0%	0%	30%	70%
10	0%	0%	30%	20%	50%
11	0%	0%	0%	30%	70%

[19] https://mindminers.com/blog/entenda-o-que-e-escala-likert/

Pergunta	*Classificação*				
	1	*2*	*3*	*4*	*5*
12	0%	0%	30%	20%	50%
13	0%	0%	0%	30%	70%
14	0%	0%	30%	20%	50%
15	0%	0%	0%	70%	30%
16	0%	0%	20%	50%	30%
17	0%	0%	10%	30%	60%

Após uma breve análise dos resultados obtidos e associando as respostas aos indicadores definidos inicialmente é possível verificar que:

- **Usabilidade –** De um modo geral, os resultados referentes a este indicador são bastante positivos. Grande parte dos futuros utilizadores, que responderam ao questionário, consideram a interface simples e apelativa, para além disso consideram que a interface desenvolvida contribui para o uso contínuo do sistema (de acordo com as respostas da pergunta 1 à 8).
- **Funcionalidade –** No que diz respeito às respostas relativas às funcionalidades do sistema também foram muito positivas. Os possíveis utilizadores reconhecem que efetivamente as funcionalidades desenvolvidas vão de encontro às suas expectativas, uma vez que, as classificações às perguntas 9, 11 e 13 variam entre o nível 4 e 5. No entanto, nas perguntas onde se pretendia perceber se as funcionalidades desenvolvidas iam de encontro às necessidades dos inquiridos, apesar de grande parte das classificações estarem entre o nível 4 e 5, nas perguntas 10, 12 e 14, 30% dos inquiridos responderam com o nível 3. Talvez isto se verifique porque muito possivelmente alguns dos possíveis utilizadores ainda não trabalham com React. Contudo, pode concluir-se, que de um modo geral, as funcionalidades do sistema têm uma avaliação bastante positiva.
- **Desempenho –** Relativamente ao indicador de desempenho apenas foi colocada uma pergunta. Após uma análise das classificações à pergunta 15, podemos verificar que 70% dos inquiridos responderam com o nível 4 e os outros 30% com o nível 5. Deste modo, podemos concluir que os inquiridos consideram o desempenho do sistema bastante aceitável.
- **Utilidade –** Este indicador tem como objetivo perceber se efetivamente o sistema desenvolvido pode vir a ser útil na carreira profissional dos inquiridos, para além disso, pretende-se perceber se o sistema desenvolvido pode auxiliar as tarefas diárias dos colaboradores da organização. Nas perguntas 16 e 17, podemos verificar que a maior parte dos utilizadores classificaram a utilidade do sistema com o nível 4 e 5. Deste modo, pode-se afirmar que o sistema desenvolvido pode vir a ter um impacto positivo nas tarefas do quotidiano dos colaboradores da organização.

6.4 Análise de Resultados

Com base nas respostas obtidas através do questionário de avaliação, pode afirmar-se que de um modo geral os resultados obtidos foram bastante positivos. Apesar da amostra de possíveis utilizadores, que responderam ao questionário, ser bastante reduzida pode afirmar-se que estes apresentam um grau de satisfação elevado face ao protótipo apresentado.

Os resultados do questionário serviram também para validar as hipóteses testadas. Deste modo, é possível afirmar que os colaboradores da organização consideram que o sistema desenvolvido tem potencial para auxiliar algumas das suas tarefas diárias. Isto, deve-se ao facto das funcionalidades implementadas estarem em conformidade com os objetivos definidos inicialmente. Para além disso, os possíveis utilizadores consideram a interface desenvolvida apelativa, contribuindo deste modo para o uso continuo do protótipo desenvolvido.

Uma vez que a amostra de possíveis utilizadores, que responderam ao questionário de avaliação foi reduzida, seria interessante, numa fase mais avançada do projeto, se realizasse um novo questionário de avaliação com o objetivo de perceber se com uma amostra maior os resultados obtidos seriam diferentes.

7 Conclusão

O último capítulo presente na dissertação comporta o balanço do projeto implementado. Aqui são expostos os objetivos alcançados. Para além disso, são expostas as limitações do trabalho desenvolvido e possíveis trabalhos decorrentes a ideia central deste projeto. Por fim, é realizada um apreciação final do trabalho realizado.

7.1 Objetivos alcançados

Em primeiro lugar procedeu-se ao levantamento bibliográfico de alguns trabalhos relacionados com sistemas de geração de código. Elaborou-se, também, uma pesquisa tecnológica de ferramentas utilizadas para a geração de componentes React, onde foram listados os pontos fortes e fracos de maneira a que fosse possível escolher a mais indicada. Só depois da idealização de uma *framework* de geração de código, recorrendo ao uso de diagramas UML, é que se estava em condições de implementar a solução descrita neste documento.

Deste modo, foi possível atingir o principal objetivo do presente trabalho que seria desenhar e desenvolver um protótipo de uma *framework* de geração de *React Journeys,* onde a interação do utilizador fosse assegura através de uma interface gráfica. Para além disso, foi necessário assegurar que a interface desenvolvida fosse facilmente utilizada pelos utilizadores do sistema, para comprovar tal requisito foi utilizado um questionário de satisfação, onde foi possível concluir que a maior parte dos inquiridos se mostraram satisfeitos com a interface gráfica desenvolvida, considerando-a simples e intuitiva.

Neste caso de estudo, onde foi desenvolvido um questionário de satisfação aos possíveis futuros utilizadores do protótipo desenvolvido, foi possível avaliar o sistema desenvolvido segundo os indicadores de usabilidade, funcionalidade, desempenho e utilidade. Após a realização do questionário desenvolvido foi possível concluir que o feedback recebido relativamente aos indicadores, sobre os quais o sistema foi avaliado, foi bastante positivo.

Para sistematizar, Tabela 27 encontram-se descritas as relações entres os objetivos definidos inicialmente e os respetivos graus de cumprimento.

Tabela 27 – Relação entre objetivos definidos e respetivo grau de cumprimento

OBJETIVO	GRAU DE CUMPRIMENTO
ANÁLISE DE FERRAMENTAS DE GERAÇÃO DE CÓDIGO PARA COMPONETES REACT	Atingido
DESENHO E DESENVOLVIMENTO DE UM PRÓTOTIPO DE FRAMEWORK DE GERAÇÃO DE CÓDIGO PARA REACT JOURNEYS.	Atingido
DESENVOLVIMENTO DE UMA INTERFACE GRÁFICA SIMPLES E INTUITIVA	Atingido
REALIZAÇÃO DE UM CASO DE ESTUDO, PARA PERCEBER O GRAU DE SATISFAÇÃO DOS UTILIZADORES.	Atingido

7.2 Limitações e trabalho futuro

Este subcapítulo destina-se à especificação das limitações do trabalho desenvolvido, bem como à especificação de futuras implementações na aplicação desenvolvida.

A intenção da organização é utilizar o sistema desenvolvido na migração dos seus atuais projetos para ReactJs e na aceleração de criação de novos. Deste modo, é necessário que os templates utilizados na criação do protótipo sejam substituídos por novos mais complexos e que realmente correspondam às componentes que a organização pretende desenvolver. Para tal, é necessário que algum developer construa uma componente modelo, transformando essa componente num template, ou seja, é necessário definir as partes estáticas (que não sofrem alteração na geração) e as partes dinâmicas (que são substituídas no processo de geração).

Apesar de, não ser um dos objetivos para a conceção do protótipo de gerador de código seria interessante realizar a integração com o servidor *Lightweight Directory Access Protocol* (LDAP), restringindo deste modo o acesso à aplicação apenas a colaboradores da organização. Um outro melhoramento, que tornaria a o protótipo ainda mais interessante, era obter a lista de componentes, que podem ser importadas de uma livraria externa, de forma dinâmica.

Para além disso, para tornar o protótipo ainda mais completo, seria interessante o desenvolvimento de novos módulos de geração, nomeadamente de microsserviços, uma vez que se encontram muitas semelhanças arquiteturais nos microsserviços desenvolvidos pela organização.

7.3 Apreciação final

O facto do projeto se enquadrar num ambiente empresarial, onde o autor teve a oportunidade de realizar um estágio, permitiu ao autor adquirir novas competências que até hoje não havia explorado, o que permitiu o seu desenvolvimento a nível tecnológico. Para além do desafio tecnológico, o estágio desafiou o autor também a um nível pessoal, sendo este capaz de desenvolver algumas capacidades de espírito de aprendizagem e iniciativa.

A interação com profissionais na área de desenvolvimento de software e o seu acompanhamento, contribuíram também para o crescimento pessoal do autor, alargando e aprofundando os seus conhecimentos.

Uma vez que, o projeto foi concluído com sucesso, o autor teve a oportunidade de ingressar na sua primeira experiência profissional na Celfocus. Onde integrou uma das equipas de desenvolvimento, sendo também este um indicador do bom trabalho desenvolvido.

Referências

[1] “Spring Initializr,” *Spring Initializr*. https://start.spring.io (accessed Oct. 12, 2021).

[2] S. Jarzabek and H. D. Trung, “Flexible generators for software reuse and evolution: NIER Track,” in *2011 33rd International Conference on Software Engineering (ICSE)*, May 2011, pp. 920–923. doi: 10.1145/1985793.1985946.

[3] A. T. Imam, T. Rousan, and S. Aljawarneh, “An expert code generator using rule-based and frames knowledge representation techniques,” in *2014 5th International Conference on Information and Communication Systems (ICICS)*, Apr. 2014, pp. 1–6. doi: 10.1109/IACS.2014.6841951.

[4] D. Keppel, S. J. Eggers, and R. R. Henry, “A Case for Runtime Code Generation,” p. 10.

[5] E. Syriani, L. Luhunu, and H. Sahraoui, “Systematic Mapping Study of Template-based Code Generation,” *Comput. Lang. Syst. Struct.*, vol. 52, pp. 43–62, Jun. 2018, doi: 10.1016/j.cl.2017.11.003.

[6] T. Elec, C. Sci, B. Uyanik, and V. Şahin, “A template-based code generator for web applications,” *Turk. J. Electr. Eng. Comput. Sci.*, vol. 28, pp. 1747–1762, May 2020, doi: 10.3906/elk-1910-44.

[7] S. Jörges, “The State of the Art in Code Generation,” in *Construction and Evolution of Code Generators: A Model-Driven and Service-Oriented Approach*, S. Jörges, Ed. Berlin, Heidelberg: Springer, 2013, pp. 11–38. doi: 10.1007/978-3-642-36127-2_2.

[8] A. Pano, D. Graziotin, and P. Abrahamsson, “Factors and actors leading to the adoption of a JavaScript framework,” *Empir. Softw. Eng.*, vol. 23, no. 6, pp. 3503–3534, Dec. 2018, doi: 10.1007/s10664-018-9613-x.

[9] S. Aggarwal, “Modern Web-Development using ReactJS,” vol. 5, no. 1, p. 5, 2018.

[10] A. Fedosejev, *React*. Olton Birmingham: Packt Publishing Ltd, 2015.

[11] “Comparative analysis of angularjs and reactjs,” *Int. J. Latest Trends Eng. Technol.*, vol. 7, no. 4, 2016, doi: 10.21172/1.74.030.

[12] A. Broubakarian, *arminbro/generate-react-cli*. 2021. Accessed: Jan. 29, 2021. [Online]. Available: https://github.com/arminbro/generate-react-cli

[13] “Plop: Consistency Made Simple,” *Plop*. https://plopjs.com/documentation/ (accessed Jan. 29, 2021).

[14] “Automatically generate your own React components with plop.js,” *LogRocket Blog*, Apr. 30, 2019. https://blog.logrocket.com/automatically-generate-your-own-react-components-with-plop-js-2da3b39914f3/ (accessed Jan. 29, 2021).

[15] “GitHub - SBoudrias/Inquirer.js: A collection of common interactive command line user interfaces.” https://github.com/SBoudrias/Inquirer.js (accessed Jan. 29, 2021).

[16] “Divjoy.” https://divjoy.com (accessed Feb. 05, 2021).

[17] Y. Bassil, “Autonomic HTML Interface Generator for Web Applications,” *Int. J. Web Semantic Technol.*, vol. 3, no. 1, pp. 33–47, Jan. 2012, doi: 10.5121/ijwest.2012.3104.

[18] A. Akbulut and S. Toprak, “Code generator framework for smart TV platforms,” *IET Softw.*, vol. 13, no. 4, pp. 268–279, 2019, doi: https://doi.org/10.1049/iet-sen.2018.5157.

[19] P. Belliveau, A. Griffin, and S. Somermeyer, *The PDMA ToolBook 1 for New Product Development*. John Wiley & Sons, 2004.

[20] E. Gürel, “SWOT ANALYSIS: A THEORETICAL REVIEW,” *J. Int. Soc. Res.*, vol. 10, no. 51, pp. 994–1006, Aug. 2017, doi: 10.17719/jisr.2017.1832.

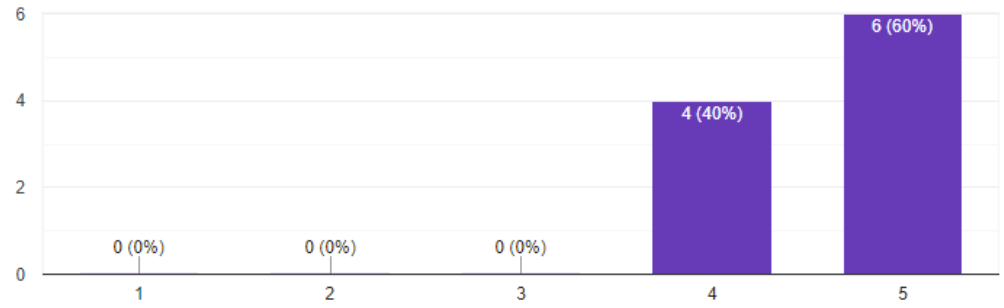
[21] C. Nielsen and J. Kyhnau, “Book review of Value Proposition Design,” *J. Bus. Models*, vol. 3, p. 81, Mar. 2015, doi: 10.5278/ojs.jbm.v3i1.1105.

[22] J. Wind, “Marketing Applications of the Analytic Hierarchy Process,” *Manag. Sci.*, pp. 641–658, Jul. 1980.
[23] B. Brahim, E. B. Omar, and G. Taoufiq, *Generating operations specification from domain class diagram using transition state diagram*.
[24] E. Hull, K. Jackson, and J. Dick, *Requirements engineering*, 2nd ed. London: Springer, 2005.
[25] R. E. Al-qutaish, *Quality Models in Software Engineering Literature: An Analytical and Comparative Study*.
[26] A. Rawashdeh and B. Matalkah, *© 2006 Science Publications A New Software Quality Model for Evaluating COTS Components*.
[27] O. Zimmermann, “Microservices tenets: Agile approach to service development and deployment,” *Comput. Sci. - Res. Dev.*, vol. 32, no. 3–4, pp. 301–310, Jul. 2017, doi: 10.1007/s00450-016-0337-0.
[28] N. Dragoni *et al.*, “Microservices: yesterday, today, and tomorrow,” *ArXiv160604036 Cs*, Apr. 2017, Accessed: Jan. 28, 2021. [Online]. Available: http://arxiv.org/abs/1606.04036
[29] “JHipster - Full Stack Platform for the Modern Developer!” https://www.jhipster.tech/ (accessed Feb. 24, 2021).
[30] “Maven – Introduction to Archetypes.” https://maven.apache.org/guides/introduction/introduction-to-archetypes.html (accessed Feb. 25, 2021).
[31] “Jeddict.” https://jeddict.github.io/index.html (accessed Feb. 25, 2021).

Anexo A – Questionário de Avaliação

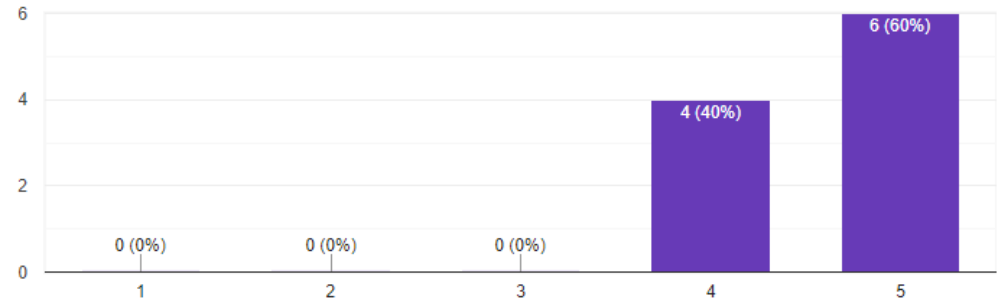
A interface é simples.

10 respostas



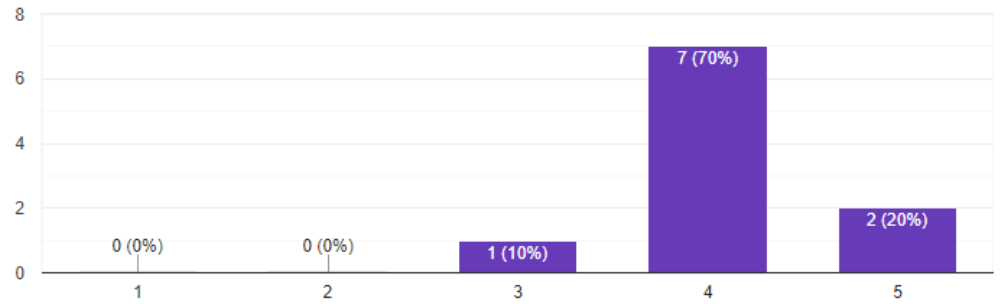
O layout do sistema desenvolvido é consistente.

10 respostas



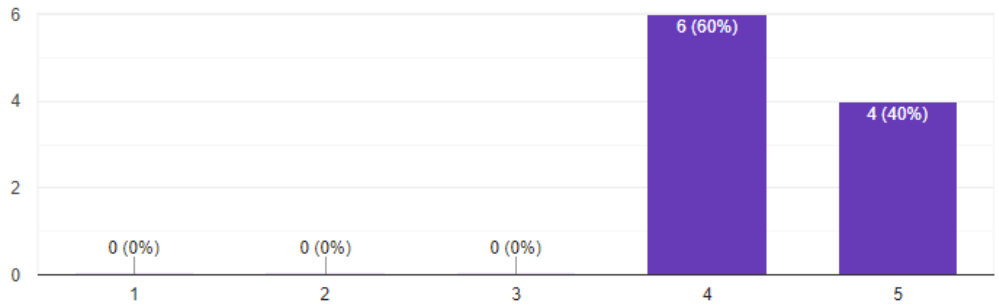
As cores utilizadas são adequadas.

10 respostas



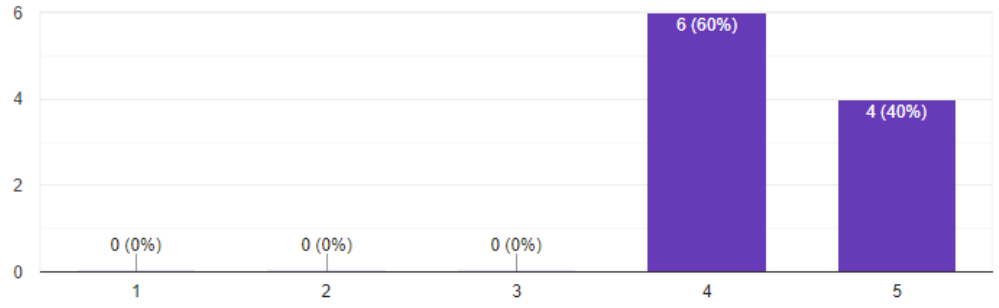
A interface é atrativa.

10 respostas



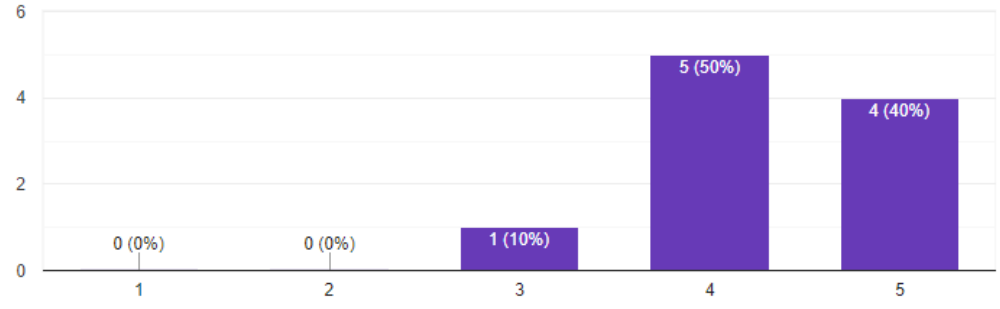
A interface é intuitiva.

10 respostas



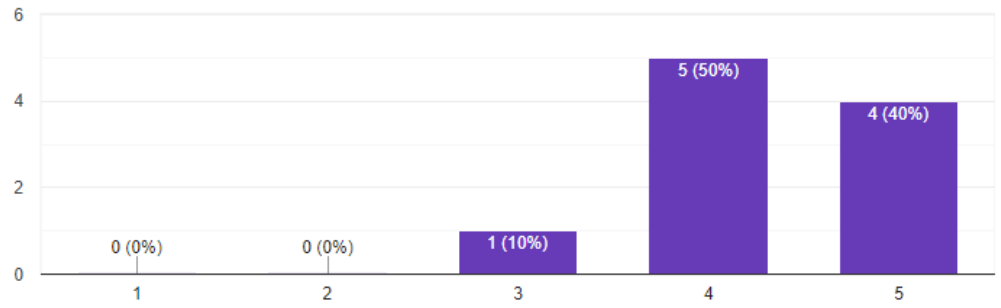
O utilizador consegue adaptar-se facilmente à interface desenvolvida.

10 respostas



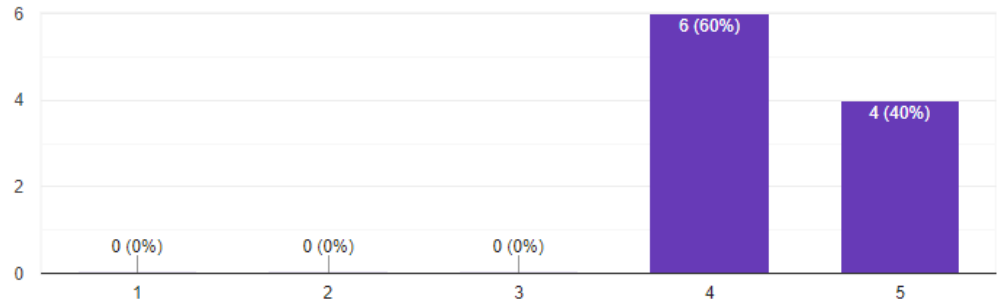
O utilizador consegue rapidamente familiarizar-se com as funcionalidades desenvolvidas.

10 respostas



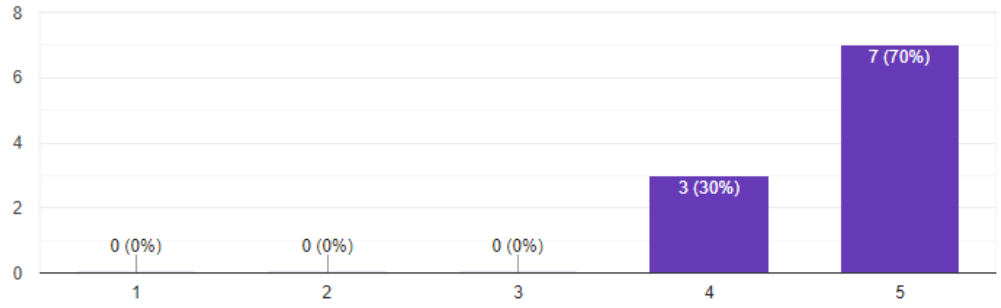
A interface desenvolvida contribui para o uso continuo da plataforma desenvolvida.

10 respostas



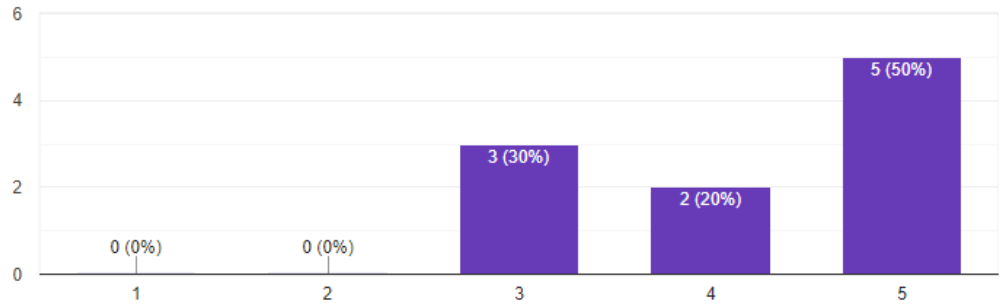
O utilizador consegue realizar a geração de uma react journey.

10 respostas



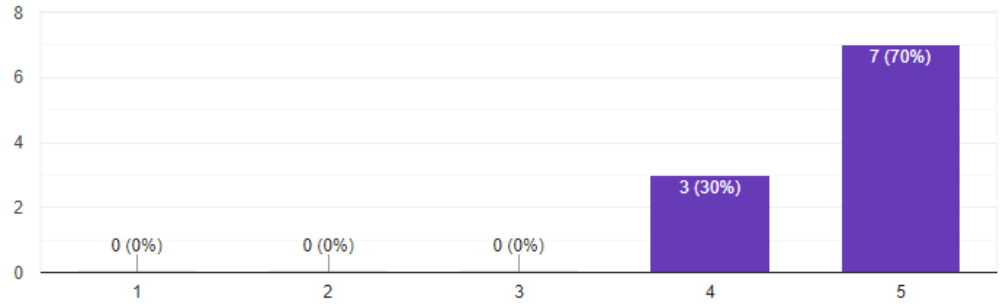
A funcionalidade de geração de uma react journey vai de encontro às minhas necessidades.

10 respostas



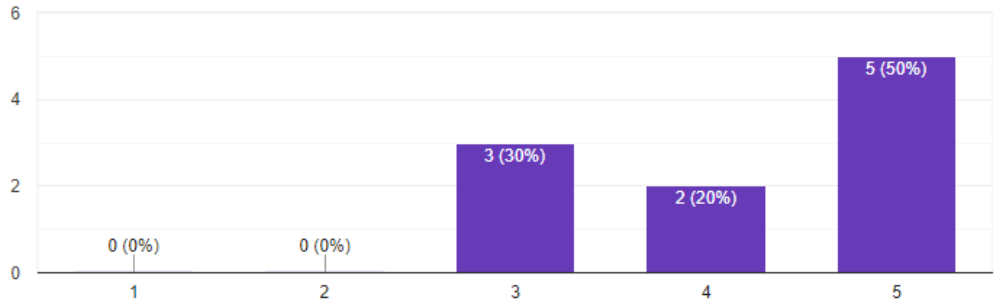
O utilizador consegue fazer o download de um JSON relativo a uma React Journey.

10 respostas



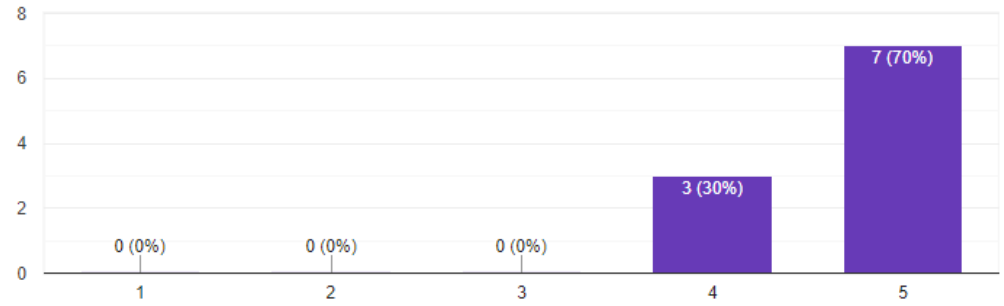
A funcionalidade de download de um JSON relativo a uma React Journey vai de encontro às minhas necessidades.

10 respostas



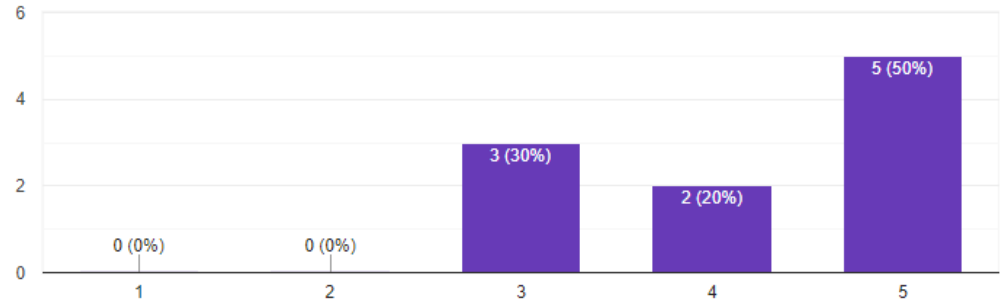
O utilizador consegue fazer a importação de um JSON relativo a uma React journey.

10 respostas



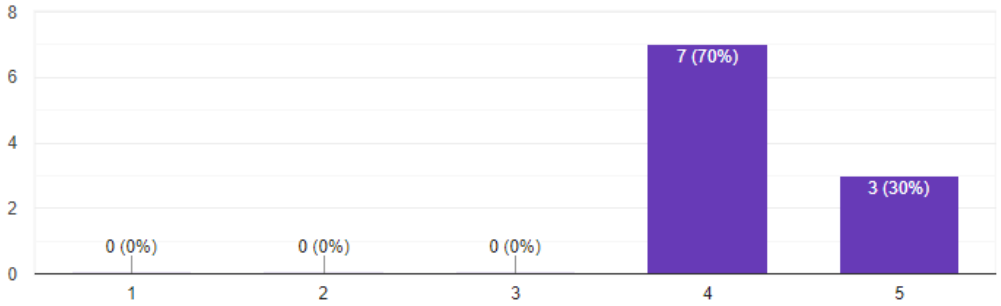
A funcionalidade de importação de um JSON relativo a uma React Journey vai de encontro às minhas necessidades.

10 respostas



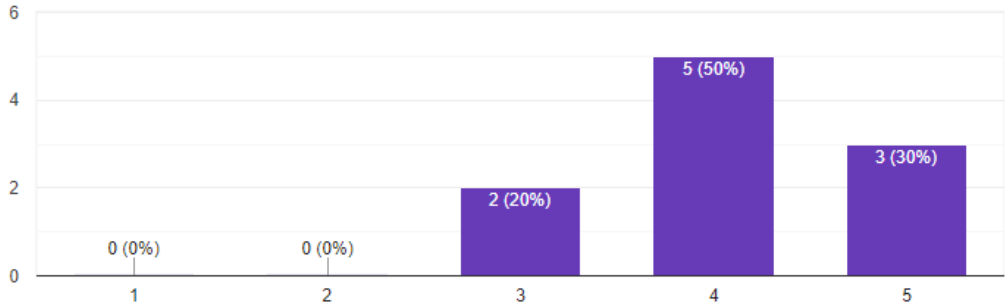
O desempenho do sistema desenvolvido é aceitável.

10 respostas



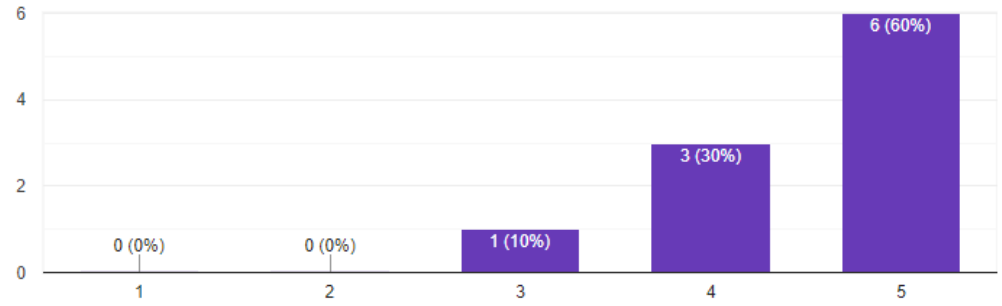
O sistema desenvolvido pode ser útil para a sua carreira profissional.

10 respostas



O sistema desenvolvido pode ter um impacto positivo nas minhas tarefas diárias.

10 respostas



Anexo B – Pesquisa realizada no âmbito de microsserviços

Microsserviços

No processo de desenvolvimento do *back-end* dos projetos da Celfocus, no que diz respeito à arquitetura, são geralmente utilizados microsserviços [27]. E do mesmo modo, como no ReactJs, a organização pretende acelerar o processo de desenvolvimento do seu *back-end,* recorrendo a um protótipo de uma *framework* de geração de código. Esta secção destina-se a apresentar as conclusões relativas ao estudo sobre microsserviços.

Os microsserviços são uma arquitetura de software e uma abordagem, cada vez mais adotada, no processo de desenvolvimento de software. O seu principal objetivo é desenvolver software altamente sustentável e escalável. Por esse mesmo motivo, esta abordagem foi construída sobre os princípios de Service-Orienred Architecture (SOA).

Os microsserviços são utilizados, para diminuir a complexidade dos sistemas desenvolvidos, decompondo grandes sistemas de software, num conjunto de serviços independentes. Cada um destes serviços independentes, executa os seus próprios processos. Esta segregação de grandes sistemas de software em serviços independentes realçam o baixo acoplamento e a alta coesão, contribuindo, desta maneira, significativamente para a modularidade de um sistema [28]. Para além da modularidade do sistema, esta abordagem, também tem um impacto significativo no que diz respeito a manutenibilidade e escalabilidade do sistema.

Neste tipo de abordagem, as funcionalidades relacionadas são combinadas num único recurso de negócio (Bounded Context). Este por sua vez, é implementado como um serviço isolado.

Resumidamente, as principais características dos microsserviços são [28]:

- Flexibilidade – O sistema desenvolvido é capaz de suportar todas as alterações necessárias.
- Modularidade – Com a adoção de microsserviços, em vez de ter um único componente responsável pela funcionalidade total, passa-se a ter componentes individuais que contribuem para o comportamento geral do sistema.
- Evolução – Como o sistema, que adota este tipo de abordagem, é mais fácil de manter e integrar novas funcionalidades.

A Figura 40**Error! Reference source not found.**, representa uma estrutura de microsserviços onde é possível perceber como é que este comunicam entre si e onde é possível também identificar as suas principais características, referidas anteriormente.

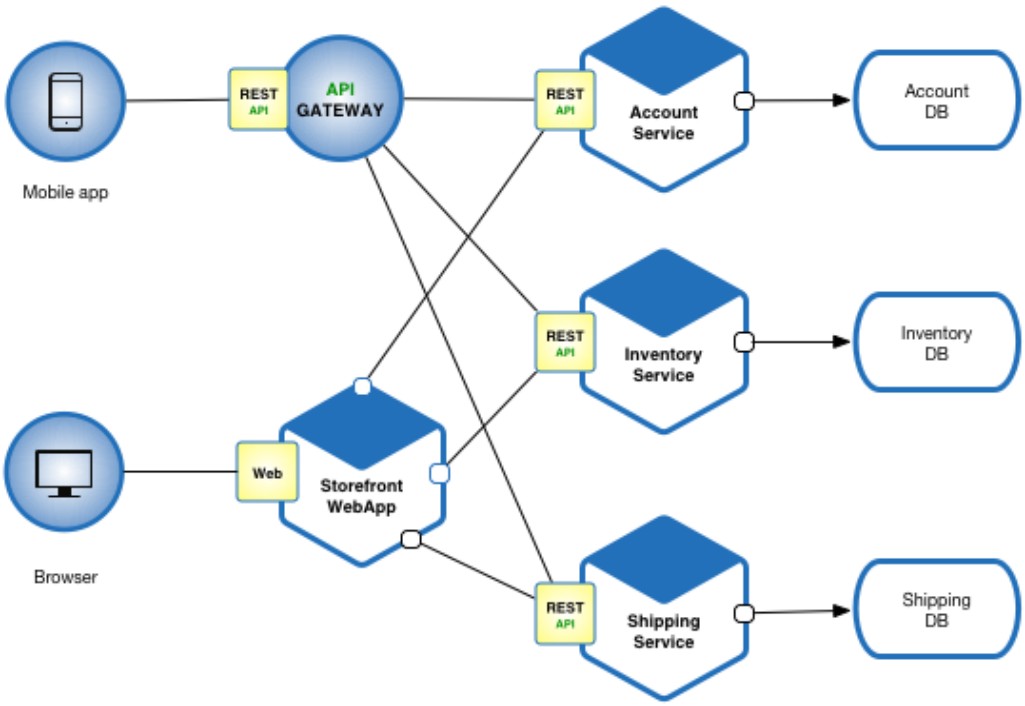


Figura 40 – Estrutura microsserviços[20]

Geradores para microsserviços

A presente secção reúne alguns geradores de código existentes no mercado, que são utilizados para geração de microsserviços. O critério escolhido para o estudo dos geradores de microsserviços é a variabilidade de técnicas utilizadas, com o intuito de perceber qual se adequa melhor ao problema a resolver.

As subsecções seguintes destinam-se, a descrever com algum detalhe de que maneira esses geradores funcionam, para se perceber se algum se enquadra no problema a desenvolver, contendo também uma análise crítica para cada um onde são descritas as principais características de cada uma das ferramentas. Para além disso, no final deste subcapítulo, encontra-se uma análise crítica entre os geradores encontrados.

JHipster

O JHipster [29] é uma ferramenta de desenvolvimento desenhada especificamente para gerar, desenvolver e implantar de uma maneira simples e rápida aplicações web com uma estrutura de microsserviços. Esta ferramenta suporta um conjunto de tecnologias capaz de criar uma base bastante sólida para uma aplicação web. No que diz respeito ao front-end esta ferramenta suporta Angular[21], React e Vue[22]. Para além disso, esta ferramenta suporta também o

[20] Retirado de: https://microservices.io/
[21] https://angular.io/
[22] https://vuejs.org/

desenvolvimento de aplicações móveis, nomeadamente em Ionic[23] e React Native[24]. No que diz respeito ao back-end, esta ferramenta suporta Spring Boot[25] (com Java ou Kotlin), Micronaut[26], Quarkus[27], Node.js e .NET[28]. Esta ferramenta também foi desenhada para implantar as aplicações web desenvolvidas, oferecendo suporte de algumas tecnologias utilizadas neste contexto, como por exemplo, Docker[29], Azure[30], Heroku[31], etc.

No contexto de microsserviços, o JHipster disponibiliza uma arquitetura, Figura 41, distribuída da seguinte maneira:

- Gateway – É uma camada gerada pelo JHipster responsável por manipular e controlar o tráfego da Web e contém um aplicativo React ou Angular. Podem existir diferentes *gateways*, se quiser adotar o padrão micro front-end, mas não é obrigatória a sua adoção;
- Traefik – É um proxy HTTP moderno responsável por equilibrar a carga, que pode funcionar em conjunto com o *gateway*;
- JHipster Registry – Esta é uma camada fornecida pelo JHipster responsável por registar todas as aplicações e conter a configuração dessas mesmas aplicações. Para além disso, disponibiliza painéis de monitorização sobre os tempos de execução das várias aplicações;
- Consul – Este componente pode ser utilizada como alternativa ao JHipster Registry e armazena objetos do tipo *key/value*;
- JHipster UAA – Contém o sistema de autorização e autenticação dos utilizadores e faz uso do protocolo OAuth2;
- Microservices - São as diferentes aplicações geradas pelo JHipster, que são responsáveis pela manipulação dos pedidos REST. Os microsserviços têm estados, podendo desta maneira várias instâncias de um mesmo microsserviço serem iniciadas em paralelo quando se está perante cargas mais pesadas.

[23] https://ionicframework.com/
[24] https://reactnative.dev/
[25] https://spring.io/projects/spring-boot
[26] https://micronaut.io/
[27] https://quarkus.io/
[28] https://dotnet.microsoft.com/
[29] https://www.docker.com/
[30] https://azure.microsoft.com/pt-pt/
[31] https://www.heroku.com/

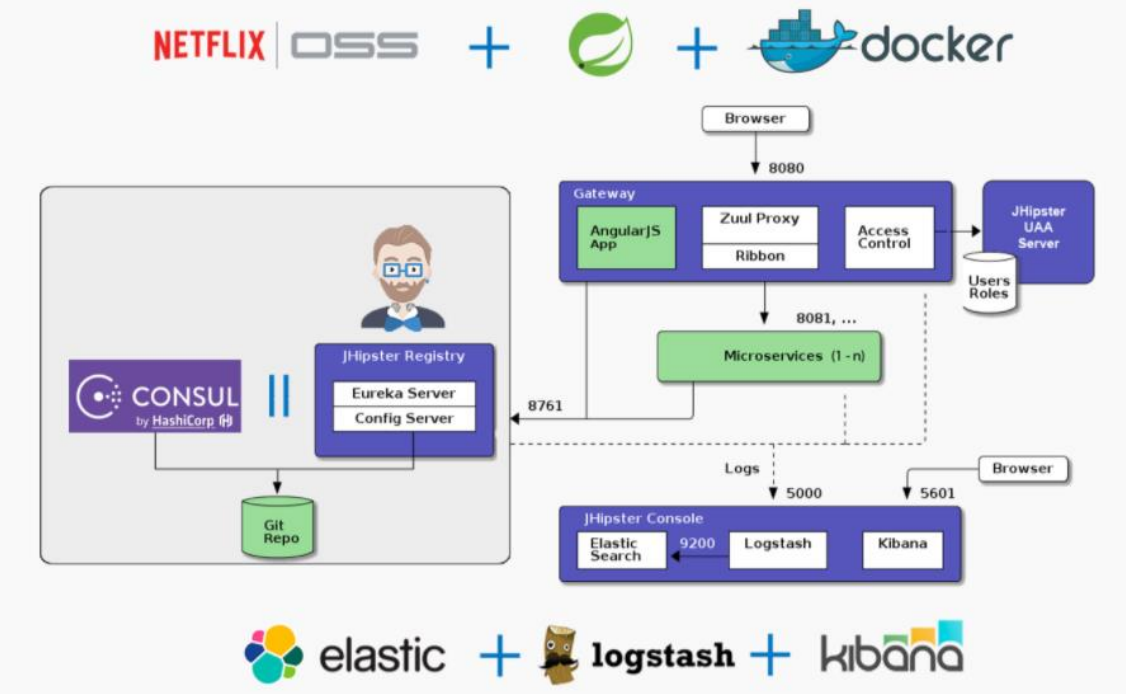


Figura 41 – Arquitetura de microsserviços do JHipster[32]

Sumário

A presente secção, tem o propósito de fazer um resumo geral da ferramenta JHipster. Deste modo, são identificadas quais as principais características desta ferramenta com o objetivo de perceber se esta ferramenta pode ser adotada no desenvolvimento da solução, tendo em conta o problema apresentado. A Tabela 28 faz o resumo dessas características.

Tabela 28 – Resumo das Caracteristicas do JHipster

Características
• Utilizado para desenvolver aplicações web *full-stack*; • Suporta elevado número de tecnologias para o desenvolvimento de uma aplicação web; • Não permite definir a estrutura dos microsserviços gerados;

Maven Archetype

O Maven Archetype [30] é um conjunto de ferramentas de modelação para um projeto Maven. Um Maven Archetype é uma abstração de um tipo de projeto que pode ser instanciado num

[32] Retirado de: https://www.jhipster.tech/microservices-architecture/

projeto Maven personalizado. Resumidamente, é um *template* de um projeto a partir do qual outros projetos são criados.

Um dos principais benefícios do uso de Archetypes é padronizar o desenvolvimento de projetos permitindo, desta maneira, que as equipas de desenvolvimento sigam facilmente as melhores práticas garantindo, ao mesmo tempo, um aumento de produtividade ao iniciar novos projetos.

A construção de um Archetype, é muito idêntica à criação de um projeto Maven. No entanto, é necessário conter dois recursos extra:

- src/main/resources/archetype-resources – basicamente consiste no template que vai servir como base para a criação dos novos projetos;
- src/main/resources/META-INF/maven/archetype-metadata.xml – este ficheiro é responsável por conter e descrever os meta-dados dos Archetypes.

O Maven Archetype permite também definir atributos que podem ser parametrizados, que serão substituídos durante a criação de um novo projeto, a partir de um Archetype previamente definido. Como se pode ver na Figura 42, o *groupId* e o *artifactId* são exemplos disso mesmo.

```
<project>
   <groupId>${groupId}</groupId>
   <artifactId>${artifactId}</artifactId>
   <version>${version}</version>
   <packaging>war</packaging>
   <dependencies>
      <dependency>
         <groupId>javax.ws.rs</groupId>
         <artifactId>javax.ws.rs-api</artifactId>
         <version>2.1</version>
         <scope>provided</scope>
      </dependency>
   </dependencies>
</project>
```

Figura 42 – Parametrização de atributos no Maven Archetype

Embora não seja uma ferramenta desenhada especificamente para o desenvolvimento ou geração de microsserviços, esta ferramenta pode ser aplicada neste contexto. O Maven Archetype permite definir uma estrutura padronizada para esses mesmos microsserviços.

Sumário

A presente secção, tem o propósito de fazer um resumo geral da ferramenta Maven Archetype. Deste modo, são identificadas quais as principais características desta ferramenta com o objetivo de perceber se esta ferramenta pode ser adotada no desenvolvimento da solução, tendo em conta o problema apresentado. A Tabela 29 faz o resumo dessas características.

Tabela 29 – Resumo das Características do Maven Archetype

Características
• Fornece ferramentas de modelação para um projeto Maven; • Geração com base em templates; • Permite definir a estrutura da aplicação gerada;

Jeddict

O Jeddict [31] é uma plataforma Open-Source de aplicativos Jakarta EE[33] com o objetivo de acelerar a produtividade dos programadores simplificando todo o processo de desenvolvimento, no que diz respeito à criação de modelos e ao relacionamento entre entidades, que em muitos dos casos são processos complexos. Para além disso, com o uso desta tecnologia, os programadores podem criar diagramas JPA[34], visualizar graficamente e alterar a base dados e, conseguem também, automatizar a geração de código em Java. O Jeddict oferece também recursos para que os programadores possam importar modelos diretamente de uma base de dados existente, e é capaz de gerar código complexo em SQL/DDL.

No contexto de microsserviços, o Jeddict disponibiliza uma arquitetura específica, que se encontra ilustrada na Figura 43.

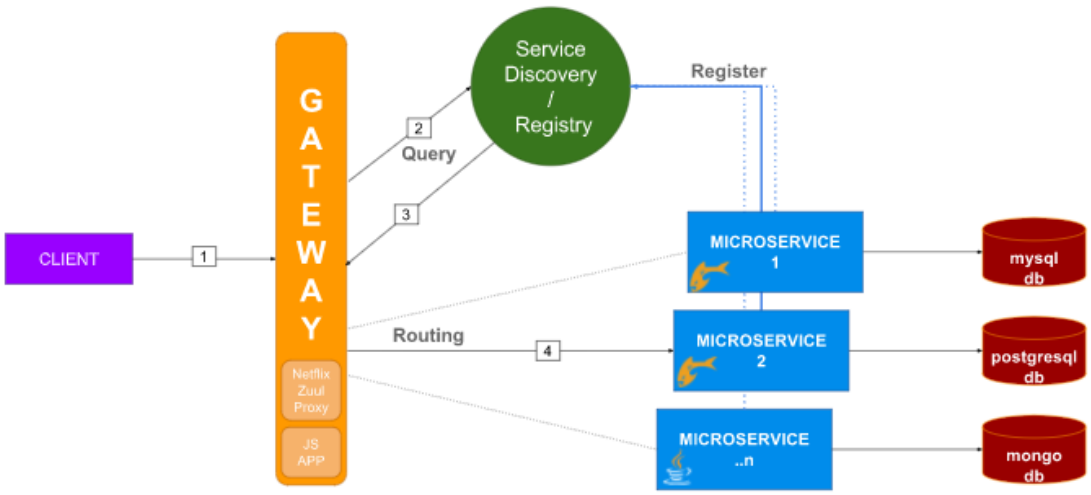


Figura 43 – Arquitetura de microsserviços do Jeddict[35]

Para a geração de microsserviços utilizando o Jeddict, o primeiro passo é criar diagramas JPA representativas do modelo específico de cada microsserviço que se pretende desenvolver.

[33] https://jakarta.ee/
[34] Java Persistence API
[35] Retirado de: https://jeddict.github.io/page.html?l=tutorial/MicroService

Posteriormente é necessário gerar o Gateway, que basicamente é utilizado para fornecer apenas um único ponto de entrada para a API desenvolvida, responsável por comunicar e manipular a comunicação com os vários microsserviços internos desenvolvidos. Este Gateway é responsável pelo roteamento dos pedidos. Todos os pedidos efetuados pelos utilizadores passam primeiro pelo API Gateway e este encaminha, de acordo com o pedido efetuado, para o microsserviço apropriado. Após o desenvolvimento dos diagramas JPA e do API Gateway, é necessário gerar os microsserviços de acordo com as classes JPA previamente definidas, garantindo que a comunicação com o front-end deve ser configurada através do API Gateway criado.

Sumário

A presente secção tem o propósito de fazer um resumo geral da ferramenta Jeddict. Deste modo, são identificadas quais as principais características desta ferramenta com o objetivo de perceber se esta ferramenta pode ser adotada no desenvolvimento da solução, tendo em conta o problema apresentado. A Tabela 30 faz o resumo dessas características.

Tabela 30 – Resumo das Características do Jeddict

Características
• Utilizado para desenvolver aplicações web full-stack; • Geração dos microsserviços em Java; • Geração dos microsserviços com base em modelos. • Não permite a definição da estrutura do microsserviço gerado.

Comparação entre geradores de microsserviços

Com objetivo de analisar os geradores anteriormente descritos, nomeadamente o JHipster, o Maven Archetype e o Jeddict, elaborou-se uma tabela comparativa, Tabela 31, com o intuito de perceber quais as grandes diferenças entre as mesma, tentando perceber se alguma delas seria uma boa base para a resolução do problema, de acordo com as suas principais características.

Tabela 31 – Comparação entre geradores de microsserviços

Geradores **Características**	**JHipster**	**Maven Archetype**	**Jeddict**
Desenhado para desenvolver aplicações full-satck	✓	✗	✓

Suporta diferentes tipos de tecnologias	✓	✓	✗
Geração dos microsserviços com base em templates	✗	✓	✗
Geração dos microsserviços com base em modelos	✗	✗	✓
Permite definir a estrutura dos microsserviços gerados	✗	✓	✗

✗- Não contém a funcionalidade ✓ - Contém a funcionalidade

No que diz respeito à geração de microsserviços, a organização pretende gerar microsserviços com base em Archetypes previamente definidos. Uma vez que, a única ferramenta que permite definir a estrutura dos microsserviços gerados é o Maven Archetype. A geração utilizando esta ferramenta é feita através de templates, tornando-se a melhor alternativa para a adoção no projeto.