IPS Instituto Politécnico de Setúbal
Escola Superior de Tecnologia de Setúbal

José Carlos Ferreira da Silva

RENOVAÇÃO DA PLATAFORMA DE CONTRATAÇÃO ELETRÓNICA - nextVISION

Mestrado em Engenharia de Software

Dissertação de Relatório de Estágio submetida como requisito parcial para obtenção do grau de **Mestre em Engenharia de Software**

Júri
Presidente (Doutor, Cláudio Miguel Garcia Loureiro dos Santos Sapateiro, ESTSetúbal/IPS)
Orientador (Engº, Filipe Alexandre da Silva Mariano, ESTSetúbal/IPS)
Coorientador (Doutor, Hugo Humberto Plácido da Silva, ESTSetúbal/IPS)
Arguente (Prof., João Pedro de Abreu Morais, ESTSetúbal/IPS)

Fevereiro de 2021

Agradecimentos

Em primeiro lugar, pretendo deixar os meus sinceros agradecimentos ao Instituto Politécnico de Setúbal e ao curso de Mestrado em Engenharia de Software, assim como a todos os professores, que ao longo do mestrado contribuíram para enriquecer o meu conhecimento.

Agradeço ao professor Cláudio Sapateiro, presidente do júri e coordenador de curso, que sempre se mostrou disponível para responder às minhas questões, encontrando as melhores soluções para os problemas que apareceram ao longo de todo o processo.

Fortemente agradeço ao professor Filipe Mariano, orientador de estágio, ao professor Hugo Silva, coorientador de estágio, e a Vera Moreira, supervisora de estágio, que cultivaram uma forte relação de comunicação, estando disponíveis para reuniões de acompanhamento e prontificados para responder a todo o tipo de questões. Tentaram sempre providenciar as melhores soluções para me integrar ao máximo neste estágio.

Não quero deixar de agradecer a todos os meus colegas, com quem tive o prazer de interagir durante o período de estágio, que se mostraram sempre disponíveis para eventuais esclarecimentos.

Gostaria de agradecer também ao IT – Instituto de Telecomunicações pelo apoio prestado durante a realização deste estágio.

Quero deixar um especial agradecimento à minha família, sobretudo à minha avó materna, à minha mãe e ao meu padrasto, que me apoiaram em todas as minhas decisões e me auxiliaram e continuam a auxiliar. Assim como dito no términos da licenciatura, sem eles nada disto seria possível, pois nunca deixaram de acreditar nas minhas capacidades, tentando encorajar-me em todos os momentos.

Por fim, e não menos importante, pretendo agradecer aos meus colegas de mestrado que, em todas as circunstâncias, se mostraram disponíveis para ajudar, prevalecendo um ambiente positivo entre todos.

De maneira geral, pode-se considerar que o estágio foi realizado com muito sucesso, e a razão está associada à contribuição de todas estas pessoas e a todo o esforço aplicado por mim na última etapa do mestrado acima mencionado.

Resumo

O seguinte relatório descreve o trabalho desenvolvido no estágio, que se encontra incluído no plano de estudos do Mestrado de Engenharia de Software e que é uma condição necessária para a conclusão do curso. Este estágio, com duração de nove meses, foi realizado na empresa PrimeIT, sendo apoiado e aceite pela Escola Superior de Tecnologia do Instituto Politécnico de Setúbal. Sendo a PrimeIT uma empresa de consultoria em tecnologia, que oferece serviços de *outsourcing*, o estagiário foi alocado como trabalhador externo na empresa Vortal, onde os nove meses de estágio foram passados a desenvolver uma plataforma de contratação eletrónica denominada nextVISION. Estas entidades (Instituto Politécnico de Setúbal, PrimeIT e Vortal) proporcionaram-me a possibilidade de desenvolver um trabalho prático de aplicação dos temas abordados durante a minha formação. Além de adquirir novas competências técnicas, o projeto em que fiquei integrado permitiu-me desenvolver competências ao nível do trabalho em equipa, interagindo e comunicando com outros colegas, existindo uma mútua troca de conhecimento, e permitindo ainda que pudesse tomar decisões autónomas e bem fundamentadas. Esta experiência foi fundamental para desenvolver hábitos de trabalho e sentido de responsabilidade profissional.

Palavras-chave: Vortal, nextVISION, nextWAY, contratação eletrónica, microserviços, PrimeIT.

Abstract

The following report describes the work developed in the internship, which is included in the study plans of the master degree in Software Engineering, and that is a necessary requirement for the completion of the course. This internship, lasting nine months, was held at the company PrimeIT, being supported and accepted by the Escola Superior de Tecnologia of the Instituto Politécnico de Setúbal. PrimeIT is a technology consulting company that offers outsourcing services. The intern was allocated as an outside worker at the company Vortal, where the nine months of internship were spent developing an electronic contracting platform called nextVISION. These entities (Instituto Politécnico de Setúbal, PrimeIT and Vortal) provided me with the possibility of developing a practical application of knowledge acquired during my training. Besides further developing my technical knowledge, the project I integrated enabled me to develop skills in terms of teamwork, interacting and communicating with other colleagues, and there was a mutual exchange of knowledge that allowed me to make autonomous and well-informed decisions. This experience was fundamental to develop work habits and a sense of professional responsibility.

Keywords: Vortal, nextVISION, nextWAY, electronic contracting, microservices, PrimeIT.

Índice

Lista de Figuras

Lista de Tabelas

Lista de Siglas e Acrónimos

API	*Application Programming Interface*
ASP	*Active Server Pages*
BD	Base de Dados
BE	*Back-End*
BIF	*Back-End-in-Front-End*
CD	*Continuous Delivery*
CI	*Continuous Integration*
CIL	*Common Intermediate Language*
CLI	*Common Language Infrastructure*
CLR	*Common Language Runtime*
CSS	*Cascading Style Sheets*
DEV	*Development*
DOM	*Document Object Model*
ES	ECMAScript
ESTS	Escola Superior de Tecnologia de Setúbal
FE	*Front-End*
GUI	*Graphical User Interface*
HTML	*HyperText Markup Language*
HTTP	*HyperText Transfer Protocol*
I18N	*Internationalization*
IDE	*Integrated Development Environment*
IPS	Instituto Politécnico de Setúbal
IT	*Information Technology*
JS	*JavaScript*
JSX	*JavaScript XML*
KPI	*Key Performance Indicator*

LESS	*Leaner Style Sheets*
MDA	*Model Driven Architecture*
MVC	*Model View Controller*
NPM	*Node Package Manager*
PM	*Project Manager*
PO	*Product Operations*
PR	*Pull Request*
QA	*Quality Assurance*
REST	*Representational State Transfer*
RM	*Release Management*
RSD	*Requirements Specification Document*
SASS	*Syntactically Awesome Style Sheets*
SD	*Solution Design*
SQL	*Structured Query Language*
TI	Tecnologias de Informação
TL	*Team Leader*
TS	*TypeScript*
UI	*User Interface*
UX	*User Experience*
XML	*Extensible Markup Language*
XSD	*XML Schema Definition*
XP	*eXtreme Programming*

Capítulo 1
Introdução

1.1. Motivação e Objetivos

Este relatório descreve o trabalho desenvolvido pelo estagiário na empresa Vortal, onde o mesmo foi alocado pela consultora PrimeIT. Este estágio foi realizado no âmbito da unidade curricular Projeto/Estágio e dita o fim do Mestrado em Engenharia de Software do Instituto Politécnico de Setúbal – Escola Superior de Tecnologia de Setúbal (ESTS). O estágio teve a duração de nove meses e decorreu entre 9 de Outubro de 2019 e 9 de Julho de 2020.

O processo de estágio foi coordenado por dois professores da ESTS, Filipe Mariano, orientador de estágio, e Hugo Silva, coorientador de estágio, e ainda por Vera Moreira, supervisora de estágio e gestora de projeto na PrimeIT.

A opção tomada pelo estagiário relativamente à entidade de acolhimento (PrimeIT) está relacionada com ambiente e transparência demonstrada, tendo sido transmitida uma boa sensação de confiança e segurança. Relativamente ao cliente (Vortal) a razão da escolha consiste no facto de o projeto possuir tecnologias recentes e do estagiário estar convicto de que podia ser uma mais valia em termos profissionais e pessoais. Este estágio teve como principais objetivos:

- Integrar uma equipa de desenvolvimento de um projeto em curso, que tem como principal missão a mudança de uma arquitetura monolítica existente num sistema legado para um novo paradigma de microserviços, contemplando a realização do *refactoring* necessário à adaptação da tecnologia anteriormente utilizada;
- Participar na fase de especificação e análise de requisitos impostos pelo cliente;
- Demonstrar *soft skills* e *know-how* ao nível de desenvolvimento de *Front-End*, contribuindo para a melhoria da usabilidade e experiência de utilização da plataforma;
- Desenvolver capacidades de abordagem pragmática aos desafios propostos, entregando soluções adequadas dentro de prazos reduzidos;
- Desenvolver métodos de comunicação com outras equipas, fomentando assim as relações interpessoais e a capacidade de trabalhar em equipa;
- Aprofundar o conhecimento teórico sobre temas relacionados com o projeto permitindo enriquecer o conhecimento tecnológico.

1.2. Contributos Principais

Relativamente aos contributos do estagiário para o projeto, nas diferentes entregas, podemos dividi-los em duas grandes fases.

1.2.1. Fase 1 – Desenvolvimento (Development Phase)

Sendo considerada como a fase mais importante do projeto, é neste momento que as funcionalidades são implementadas. Todas as implementações a realizar encontram-se documentadas em *Requirements Specification Documents* (RSD's), documentos onde são especificadas todas as funcionalidades pedidas pelos clientes. Os desenvolvimentos realizados pelo estagiário podem dividir-se em dois componentes diferentes:

Componentes Específicos: este tópico corresponde aos componentes que não se replicam nas diferentes seções do projeto.

Componentes Genéricos: diferindo do tópico acima, estes componentes foram desenvolvidos numa solução isolada, *VortalComponents*, de forma a dar suporte a todas as soluções do projeto Vortal. De forma a evitar a duplicação de código e considerando as boas práticas de programação, esta foi a solução encontrada para criar componentes agnósticos e isolados que pudessem cumprir todas as funcionalidades semelhantes/iguais. O desígnio e todo o modo de implementação do componente ficou ao encargo do estagiário, apenas lhe foi imposto que os requisitos do mesmo fossem cumpridos.

O ciclo de vida de um componente genérico é bastante simples, pois o componente é criado e testado na solução acima referida, posteriormente é publicado e assim torna-se disponível para ser instalado em qualquer solução do projeto Vortal.

1.2.2. Fase 2 – Certificação (Certification Phase)

É o momento em que o *software* é testado e que são realizadas as respetivas correções, para que seja entregue ao cliente com a qualidade desejada.

Sendo o momento posterior a cada fase de desenvolvimento, é importante que esta fase não seja desconsiderada pois como o erro humano é bastante comum e como todo o software desenvolvido passa por uma fase de testes, existe a necessidade de serem realizadas correções ao software, de forma a melhorar a sua usabilidade e performance, garantindo assim a qualidade suficiente exigida pelo cliente.

1.3. Estrutura de Trabalho

O presente documento encontra-se organizado de forma a permitir que o leitor

compreenda a evolução e progresso ao longo de todo o estágio. Este encontra-se organizado em sete capítulos, sendo eles:

Capítulo 1 – Introdução: neste capítulo são apresentadas as motivações e os objetivos do estágio, os contributos principais do estagiário e a respetiva estrutura do trabalho.

Capítulo 2 – Enquadramento: neste capítulo são apresentadas as duas entidades envolvidas no processo, a entidade de acolhimento e o cliente.

Capítulo 3 – Estado da Arte: neste capítulo são descritos os dois projetos abordados no relatório, o legado e o atual. É também feita uma abordagem sobre a renovação da *stack* tecnológica, a transição do sistema monolítico para o de microserviços e como foram geridos os riscos do projeto.

Capítulo 4 – Fundamentos Metodológicos: neste capítulo são descritos o planeamento, os temas referentes à metodologia e as ferramentas e tecnologias utilizadas.

Capítulo 5 – Descrição do Trabalho Realizado: neste capítulo é descrito todo o trabalho realizado ao longo do estágio, do qual podemos destacar a descrição da arquitetura proposta, os atores do projeto e os detalhes de implementação por *release*.

Capítulo 6 – Resultados: neste capítulo são focados o fluxo de certificação, o cumprimento dos KPI's e a discussão de resultados.

Capítulo 7 – Conclusão: este capítulo sumariza as conclusões tiradas ao longo do estágio.

Segue-se o **Glossário**, apresentado como uma lista alfabética de termos mencionados ao longo do documento.

Por fim é apresentada a **Bibliografia**, a qual contém uma lista das referências bibliográficas utilizadas como base de pesquisa para a realização do estágio.

Capítulo 2
Enquadramento

2.1. Entidade de Acolhimento

2.1.1. Empresa

A PrimeIT (logótipo na Figura 2.1) é uma empresa 100% portuguesa, com mais de 13 anos de experiência. Considerada uma entidade de referência e focada na prestação de serviços tecnológicos de IT, Telecomunicações, Energia, Infraestruturas e Marketing Digital, encontra-se presente nos diferentes continentes, tal como ilustrado na Figura 2.2. (PrimeIT, 2019)

Figura 2.1 – Logótipo da Empresa PrimeIT.

Figura 2.2 – Localização geográfica dos escritórios da PrimeIT.

Os números da PrimeIT (Tabela 2.1) explicam o porquê de ser considerada uma empresa de referência. (PrimeIT, n.d.)

Tabela 2.1 – Números da empresa PrimeIT.

2.1.2. Missão e Valores

Especializada em Consultoria, *Nearshore*, Gestão de Projeto e Projetos Chave-na-Mão, a empresa tem como principal objetivo potenciar o negócio de todos os seus parceiros, recorrendo à tecnologia mais avançada e aos melhores profissionais do setor. (PrimeIT, 2019)

Esta empresa assenta a sua ideologia sobre os seguintes valores: (PrimeIT, n.d.)

- **Visão:** todos os dias se desafiam e tomam decisões inovadoras; pensam, decidem e concretizam.
- **Compromisso:** são honestos na sua palavra e conscientes nas suas decisões, trabalhando com máxima transparência e responsabilidade.
- **Atitude:** a sua orientação para o sucesso é reflexo da sua atitude diferenciadora; uma postura que os distingue e motiva.
- **Paixão:** a paixão sentida pelo universo tecnológico e pela sua equipa, encoraja-os diariamente a superar os seus objetivos.
- **Competência:** trabalham com o melhor talento nacional e internacional, garantindo as soluções mais indicadas para cada situação.

2.1.3. Objetivos Definidos para o Estagiário

Tendo em consideração as competências do estagiário, foi prioritário definir quais as áreas e subáreas onde iria atuar. Com esse intuito, foi necessário definir qual o sector de atividade e qual o parceiro que iria acolher o estagiário, de forma a satisfazer as necessidades de ambos. As decisões por parte das entidades envolvidas, resultaram nas escolhas apresentadas na Figura 2.3.

Figura 2.3 – Escolhas realizadas pelo estagiário em conjunto com a consultora.

2.2. Cliente

2.2.1. Empresa

A Vortal (logótipo na Figura 2.4) é considerada líder em contratação eletrónica a nível mundial, conta com a vasta experiência de 20 anos e com um leque de clientes em todo o mundo, desde pequenas a grandes entidades públicas (municípios, cidades, governos regionais em diferentes países, organismos nacionais de contratação, ...), como também pequenas e grandes empresas privadas. (Vortal, n.d.)

Figura 2.4 – Logótipo da Empresa Vortal.

Criada com o objetivo de desenvolver e comercializar soluções de negócio inovadoras, opera em cinco mercados eletrónicos dirigidos a diferentes setores de atividade, entre os quais podemos destacar: (Vortal, n.d.)

- Setor da construção;
- Contratação pública;
- Setor industrial;
- Setor da energia e *utilities*;
- Material de escritório.

Possuem escritórios em seis países diferentes (Portugal, Espanha, Alemanha, Itália, Colômbia e México). Os números refletem a razão da Vortal ser líder mundial em contratação eletrónica (Tabela 2.2). (Vortal, n.d.)

Tabela 2.2 – Números da empresa Vortal.

2.2.2. Missão e Valores

Um dos principais objetivos da Vortal é ajudar empresas e entidades públicas a otimizar os seus processos de compra e venda com as soluções tecnológicas que oferecem (Tabelas 2.3 e 2.4). (Vortal, n.d.)

Tabela 2.3 – Soluções para Comprar Melhor.

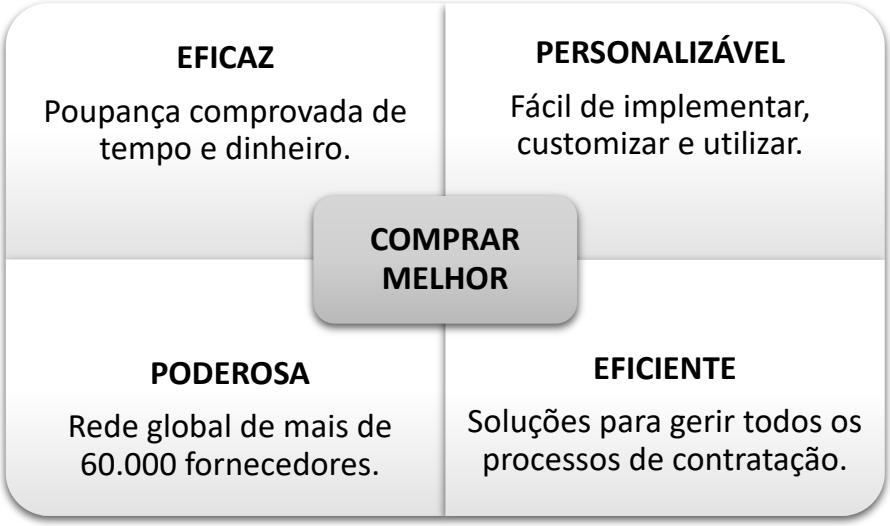
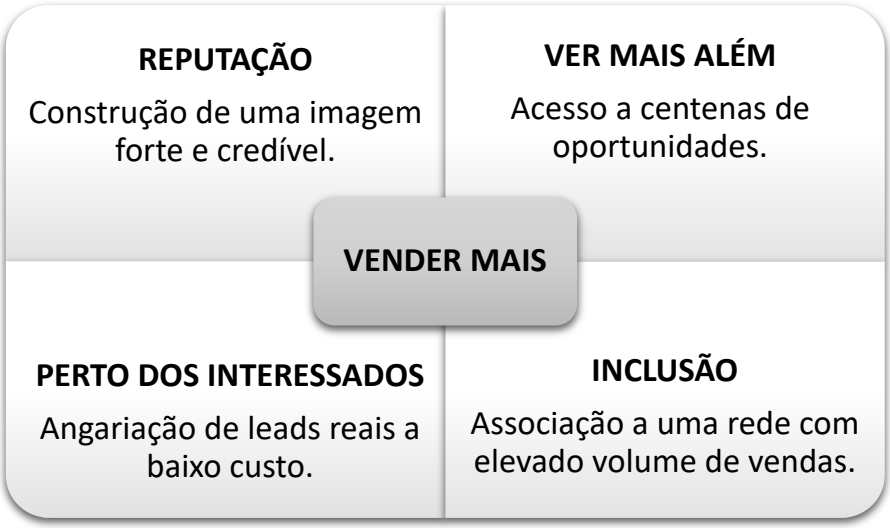


Tabela 2.4 – Soluções para Vender Mais.



Para atingir esses objetivos, a Vortal pretende ser reconhecida no mercado como uma empresa que se baseia nos seguintes valores: (Vortal, 2019)

- **Foco no cliente:** baseiam a sua atividade na qualidade e na criação de valor, atuam com agilidade e otimização de recursos, e têm sempre em conta a satisfação contínua do seu cliente.
- **Espírito de equipa:** trabalham como equipa, promovendo a participação e ajuda mútua, agindo sempre com responsabilidade e disciplina.
- **Criatividade e inovação:** promovem a partilha de ideias criativas para desenvolver novas soluções, tendo em conta o risco com bom senso e pragmatismo; assumem uma atitude inovadora antes de todos os desafios, avaliando e incorporando conscientemente todos os riscos.
- **Talento e simplicidade:** procuram de forma constante os melhores talentos, pois só assim conseguem alcançar a ambição; gerem eficientemente os seus talentos, capacitando-os para assumir e aceitar os desafios, estando em constante atualização do seu conhecimento e permanente melhoria.
- **Integridade:** agem com base em alguns princípios: dizem com franqueza aquilo que pensam, e fazem aquilo que dizem; abstêm-se de qualquer ação que os embarace e decidem com base no mérito.
- **Respeito pela diversidade:** quando estão em contacto com diferentes entidades culturais respeitam e adaptam-se às mesmas; enfrentam as diferenças de pensamento de uma forma natural, e tomam isso em consideração para uma melhoria contínua.

Capítulo 3
Estado da Arte

3.1. Sistema Legado – nextWAY

3.1.1. Introdução

O produto da empresa Vortal, denominado por nextWAY, consiste numa plataforma de contratação pública que serve entidades públicas nacionais, que de acordo com a legislação em vigor, estão obrigadas a utilizar a plataforma eletrónica para a realização de procedimentos públicos de aquisição. (Cabral, 2020)

3.1.2. Objetivo

De acordo com a lei em vigor, quando uma empresa pública efetua uma compra acima de um determinado montante, a compra tem de ser declarada numa plataforma *online*. A partir desta imposição surgiu a plataforma da Vortal. Para além de registar compras efetuadas pelas empresas públicas, a plataforma junta empresas compradoras (tanto públicas, como privadas) e empresas fornecedoras. Inicialmente a plataforma operava apenas no setor da construção, sendo que ao longo dos anos foi expandindo o negócio a vários outros setores. (Cabral, 2020)

Em termos técnicos, uma compra é designada por procedimento, sendo que um procedimento tem início quando uma empresa compradora publica um anúncio descrevendo a quantidade, o intervalo de preços e muitas outras opções do(s) artigo(s) que pretende adquirir. Os fornecedores, interessados em satisfazer as necessidades dos compradores, respondem ao anúncio submetendo uma determinada oferta; posteriormente a empresa compradora escolhe a oferta que mais lhe agrada, sendo este processo conhecido como adjudicação. O procedimento é finalizado quando as duas entidades assinam um "contrato final". (Cabral, 2020)

Graças à *cloud*, é uma plataforma acessível em qualquer parte do mundo, sendo utilizada em Portugal, Espanha, Itália, Alemanha, Colômbia, Honduras, México e Moçambique. Visto que é utilizada em diversos países, a nextWAY oferece um conjunto de diferentes configurações que permite a adaptação à legislação do país em questão. (Cabral, 2020)

As normas de contratação pública, definidas pelo código dos contratos públicos, exigem que as entidades públicas adotem procedimentos rigorosos, desde o ajuste direto ao concurso público internacional. Para executar estes tipos de procedimentos é necessário realizar um conjunto de ações, como a publicação do anúncio, o esclarecimento de dúvidas relativas ao procedimento, a receção de propostas e candidaturas, as notificações e comunicações de

admissão e exclusão de fornecedores, e por fim a escolha da entidade vencedora. (Cabral, 2020)

Desta forma, a função principal da plataforma é permitir que todas as fases e as demais ações exigidas por lei sejam realizadas, facilitando assim uma gestão integral do procedimento, garantido ao mesmo tempo toda a interação necessária entre os compradores e fornecedores. (Cabral, 2020)

3.1.3. Workflow de um Procedimento

Num procedimento os intervenientes são:

- Comprador(es) (*buyer*(*s*));
- Fornecedor(es) (*supplier*(*s*)).

O *worfklow* de um procedimento simples é ilustrado na Figura 3.1. (Cabral, 2020)

Figura 3.1 – *Workflow* de um Procedimento.

3.2. Sistema Atual – nextVISION

3.2.1. Introdução

Em 2019, a Vortal decide reformular totalmente o seu produto e as suas equipas de desenvolvimento. A empresa procurou avaliar o estado da plataforma até então usada, nextWAY, de forma a identificar limitações e listar as evoluções necessárias. O desafio era desenvolver um produto totalmente novo, integrando as ferramentas de *eSourcing* e *eProcurement* e utilizando novas tecnologias e arquiteturas previamente definidas. O produto nextVISION foi pensado e projetado para ser simples e intuitivo de utilizar, ter elevado desempenho, ser estável e resiliente, oferecendo ainda conteúdo inteligente e contextual, com recursos de inteligência artificial e análise preditiva. A principal intenção da Vortal era que o Ciclo de Vida de Desenvolvimento de Software proporcionasse à empresa a agilidade e o tempo de comercialização desejáveis, de forma cumprir os requisitos legais de negócio e as expectativas cada vez mais exigentes do cliente. (Cabral, 2020)

A nível tecnológico, a arquitetura da nextVISION é baseada nas últimas tendências e abordagens usadas nas organizações líderes em todos os setores, contando com uma *Tech Stack* completamente nova, combinada com a abordagem baseada em microserviços. Esta atualização tecnológica permite que a Vortal aborde não apenas as questões de *User Experience*/*User Interface* da plataforma legada, mas também melhore o desempenho, a resiliência, e beneficie de uma otimização de custos. (Cabral, 2020)

Em suma, a nextVISION é fruto de um produto com tecnologias modernas, reutilizando a lógica de negócio, as validações e o acesso aos dados, garantindo a transposição do "*core*" da solução legada para um novo mundo de desenvolvimento e um novo paradigma de implementação, mantendo a estabilidade e funcionalidades oferecidas pela nextWAY. (Cabral, 2020)

3.2.2. Objetivos

Devido ao crescimento do produto foram sendo identificadas limitações e dificuldades resultantes da utilização do mesmo. Dessa forma foram definidas e identificadas mudanças, que levaram à criação da plataforma nextVISION. Podemos assim, destacar os seguintes objetivos de mudança: (Cabral, 2019)

- **Estabilidade da plataforma:** a arquitetura definida, baseada em microserviços, permite fazer uso da camada atual de serviços que gere acessos e persiste os dados de todos os clientes. O desenvolvimento da nova plataforma permite transferir todo o código-fonte para o novo paradigma de microserviços, efetuando apenas o *refactoring* estritamente necessário para adaptação da tecnologia utilizada (ASP.NET Core 2.x/C#). O principal objetivo é transpor toda essa lógica, tanto no que diz respeito à panóplia de validações de negócio como dos serviços que asseguram num ponto centralizado todos os acessos à

base de dados, para o novo paradigma de microserviços. Assim, é garantida a estabilidade da solução e a coerência com a plataforma atual, visto que os alicerces são os mesmos. (Cabral, 2019)

- **Melhoria do UX/UI:** de forma a enriquecer a interação do utilizador com a plataforma, foi definido um *Front-End* baseado em HTML5/CSS3, totalmente responsivo, com suporte de ReactJS e comunicação REST. A utilização das tecnologias de HTML5/CSS3 potencia a compatibilidade com as ultimas versões dos browsers mais populares, a utilização de componentes visuais mais ricos e apelativos, e a integração das últimas *frameworks* de desenvolvimento de *Front-End* disponíveis, como ReactJS, entre outras. (Cabral, 2019)
- **Melhorias de performance e resiliência:** foi realizado um estudo intenso onde foram definidas as tendências e abordagens utilizadas pelas organizações líderes na indústria de IT. Após a mudança tecnológica, tal como descrito na seção 3.3. Renovação da *Tech Stack*, a Vortal consegue endereçar os problemas de UX/UI, como também de performance, resiliência, otimização de custos e manuseio do produto. (Cabral, 2019)
- **Rigor na especificação e desenvolvimento para melhoria da eficiência e qualidade:** devido ao aumento da autonomia e da flexibilidade na entrega do produto, valores assegurados pela restruturação da organização e pela adoção da *Tech Stack* e arquitetura propostas, foi considerado que a responsabilidade também aumentou relativamente às entregas com a agilidade pretendida, de forma a garantir a qualidade e segurança do produto Vortal. (Cabral, 2019)

3.3. Renovação da *Tech Stack*

3.3.1. Contexto nextWAY

A plataforma de desenvolvimento atual da Vortal, a nextWAY, nasceu em 2010 desenvolvida pela INDRA com o objetivo de proporcionar à equipa de desenvolvimento a possibilidade de criar um produto tecnologicamente atrativo, flexível, customizável e robusto, no menor espaço de tempo possível. (Cabral, 2019)

A alavanca que permitiu alcançar estes objetivos foi o MDA – "*Model Driven Architeture*", uma plataforma de desenvolvimento *low-code*, que permite definir modelos de negócio e de apresentação (XML e XSD), definindo no MDA a interpretação dos mesmos e a responsabilidade de produzir grande parte do código de .NET. O código gerado pelo MDA assenta no padrão ASP.NET MVC 1.1 e, através dos *Models* definidos pelo programador, o MDA gera toda a camada de *Controllers* e *Views* necessária à implementação dos requisitos propostos. A *tech stack* da nextWAY baseia-se nas tecnologias descritas na Tabela 3.1. (Cabral, 2019)

Tabela 3.1 – Tecnologias da Plataforma nextWAY.

Camada	Componentes
Apresentação	ASP.NET Views
Negócio	ASP.NET Controllers
Dados	ASP.NET Services e ASP.NET Models
Base de Dados	Microsoft SQL Server 2008 R2

Esta plataforma suportou a construção de um produto de envergadura considerável, permitindo o crescimento e estabilização da nextWAY. Como referido anteriormente, foram identificadas algumas limitações a nível de UX/UI, assim como dificuldades de agilidade durante o processo de entrega. O sistema monolítico de tamanho considerável acabou por se tornar pouco manuseável, com impactos negativos a nível performance e experiência do utilizador.

Para além destas limitações, com o desenvolvimento do produto suportado numa *framework* proprietária, acrescem algumas dificuldades. A utilização de uma plataforma desta natureza implica também uma curva de aprendizagem maior, o que impede a produtividade imediata de novos colaboradores, quando comparada com outro tipo de abordagens. (Cabral, 2019)

3.3.2. Contexto nextVISION

A equipa de IT foi desafiada a encontrar a forma mais direta e mais rápida de alterar a camada de UI, de modo a conseguir apresentar ao mercado e aos seus clientes uma plataforma com usabilidade e imagem renovadas, mais apelativas e indo ao encontro das expetativas dos mercados onde a Vortal opera. As tecnologias da plataforma desenhada no âmbito da atualização tecnológica definida estão descritas na Tabela 3.2. (Cabral, 2019)

Tabela 3.2 – Tecnologias da plataforma nextVISION.

Camada	Componentes
Front-End	HTML5/CSS3 + ReactJS
Comunicações	API Gateway
Serviços	ASP.NET Core 2.x Microservices
Base de Dados	Microsoft SQL Server 2016

As entidades mencionadas na Tabela 3.2, são explicadas de seguida: (Cabral, 2019)

Front-End: algumas das limitações identificadas na plataforma atual são relativas a UX/UI e em alguns pedidos que a mesma consegue satisfazer. Adicionalmente a tecnologia atual não está preparada para *multi-device* (não responsiva) e manuseia objetos e contextos de tamanho excessivo. Além do anteriormente indicado, o ReactJS promove a reutilização de componentes e é mais adaptável à componente móvel, sendo que se encontra entre as *frameworks* de desenvolvimento de *Front-End* mais populares, pois é considerada a mais estável e com maior tração. (Cabral, 2019)

Comunicações: a solução proposta sugere a utilização de um API Gateway como forma de exposição para o consumo dos microserviços (implementação das funcionalidades de negócio). O API Gateway é responsável pelo *routing* dos pedidos e gestão dos protocolos de comunicação entre os canais de *Front-End* e a implementação dos serviços de negócio. A tecnologia de comunicação deve também gerir a disponibilização de funcionalidades de negócio a vários clientes de diferentes naturezas, delegando no mesmo algumas tarefas relativas a segurança, gestão de *tokens*, *logging*, entre outros. (Cabral, 2019)

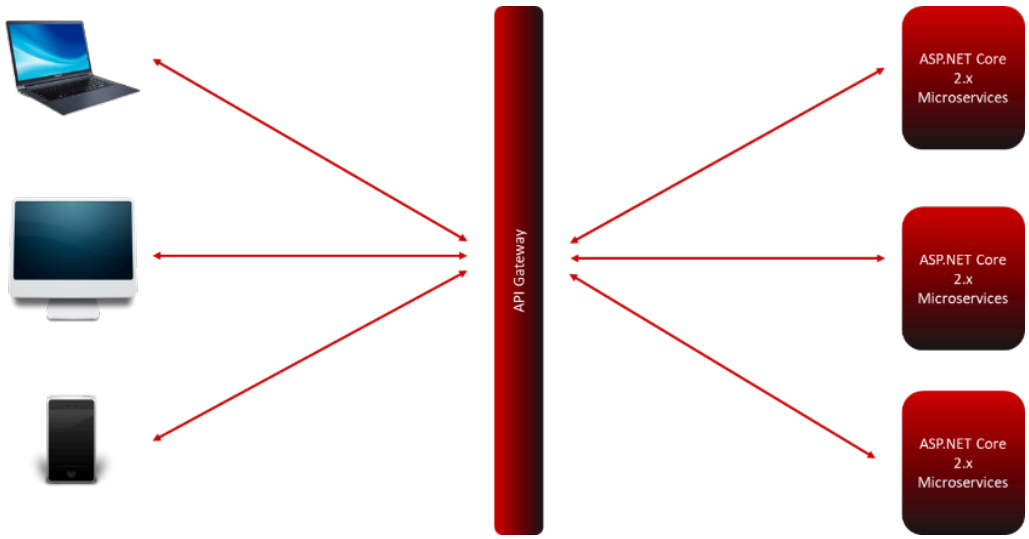


Figura 3.2 – Diagrama da camada de Comunicações.

Serviços: a recente abordagem de desenvolvimento de *software* baseada na arquitetura de microserviços tem como objetivo dar resposta a problemas e limitações cada vez mais frequentes em abordagens monolíticas que atingem alguma dimensão. Alguns exemplos destas limitações são a pouca agilidade na implementação, fraca escalabilidade das soluções, baixa disponibilidade de serviços e qualidade do produto. (Cabral, 2019)

Cada microserviço encapsula uma funcionalidade da lógica de negócio, e deverá ser tão independente e autónomo quanto possível, garantindo assim ciclos de vida independentes e proporcionado *deployments* ágeis e sem dependências. (Cabral, 2019)

A arquitetura de microserviços que suporta a renovação da *tech stack* da Vortal baseia-se em ASP.NET Core 2.x, que permite a execução em Windows, macOS e Linux, e a colocação dos mesmos em *containers* Docker. A escolha da linguagem de programação para

implementação recai naturalmente em C#, tirando partido do *know-how* existente na equipa de desenvolvimento existente na Vortal. Numa abordagem posterior, e dada a independência entre microserviços nada impede que determinadas funcionalidades sejam implementadas em tecnologias diferentes como NodeJS, Python ou Ruby, de modo a facilitar a expansão do ecossistema. (Cabral, 2019)

A implementação de microserviços faz uso da camada atual de serviços que gere os acessos e a persistência de dados. O objetivo é transpor toda essa lógica, tanto no que diz respeito à vasta panóplia de validações de negócio como dos serviços que asseguram num ponto centralizado todos os acessos às base de dados, para o novo paradigma de microserviços, garantindo assim a estabilidade necessária da solução e a coerência com a atual. (Cabral, 2019)

***Containers*:** estes ajudam a retirar o máximo partido dos recursos de hardware e software disponíveis. A virtualização clássica, através de máquinas virtuais implica a virtualização desde a camada mais baixa, enquanto que a virtualização assegurada por *containers* é mais otimizada a nível de recursos (Figura 3.3). Os *containers* são estruturas muito leves, pelo que tem uma gestão muito mais ágil que as tradicionais máquinas virtuais. (Cabral, 2019)

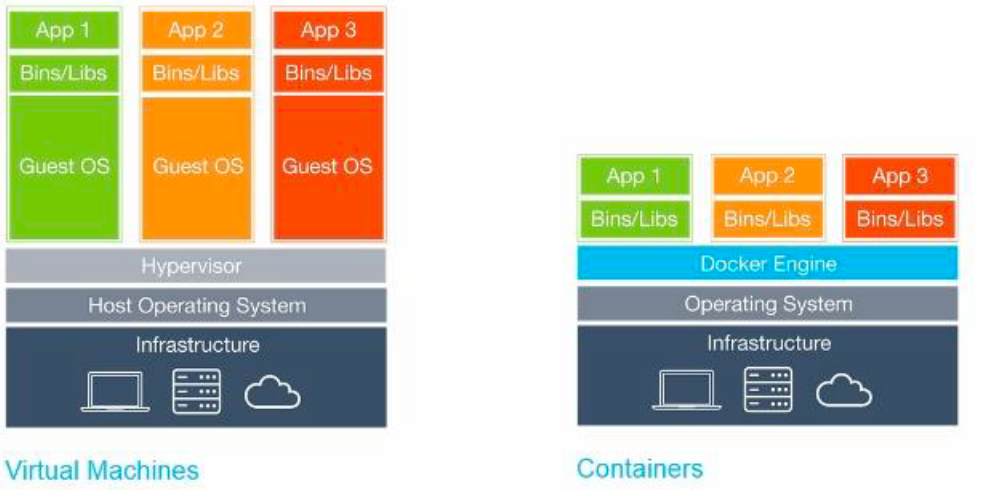


Figura 3.3 – Comparação entre máquinas virtuais e *containers*

No paradigma atual é quase impossível dissociar microserviços de *containers*, visto que é graças à junção de ambos que se garante a agilidade procurada por todas as organizações. (Cabral, 2019)

3.4. Transição do Monolítico para Microserviços

Muitas das empresas tecnológicas iniciaram a difícil jornada de migração de aplicações monolíticas para ecossistemas de microserviços. A recomendação mais consensual para uma

decomposição de sistemas monolíticos de grande envergadura em arquiteturas assentes em microserviços passa por iniciar esta árdua jornada por serviços mais simples, dando tempo e espaço às equipas para se familiarizarem com os conceitos, afinarem estratégias e conferir a possibilidade de experimentação de novas ferramentas e processos. (Cabral, 2019)

Além da implementação dos microserviços em si, existe uma fase importante de configuração dos mesmos, de ajustes dos mecanismos de *Continuous Integration/Continuous Delivery* (CI/CD) e de controlo e monitorização do novo ecossistema. Depois da dissociação das funcionalidades mais simples, deve-se partir para a desagregação do monolítico remanescente, procurando que cada passo desta migração represente uma melhoria atómica para o ecossistema que se vai formando, de forma gradual até que o sistema monolítico seja completamente abandonado. (Cabral, 2019)

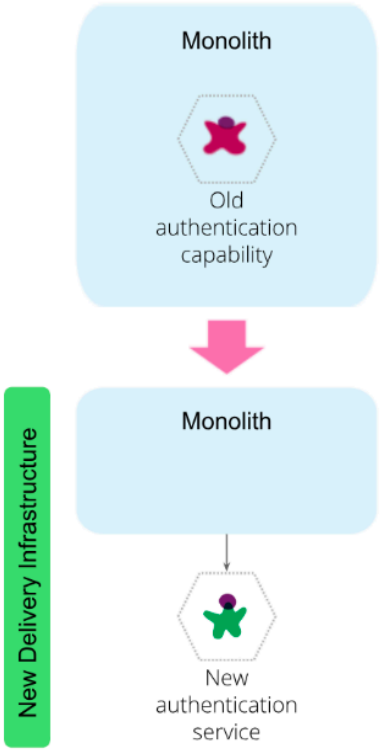


Figura 3.4 – Diagrama de Transição de Infraestrutura.

Para se conseguir um abandono faseado e controlado de cada uma das funcionalidades que serão portadas para a nova arquitetura, a estratégia de migração centra-se na utilização de um mecanismo que permite em tempo real fazer uma transição em cada funcionalidade da sua implementação legada para a atual, e vice-versa. Como referido anteriormente, o código-fonte que implementa a atual camada de serviços que gere os acessos, a persistência de dados e as validações de negócio que constituem o *core* da nextWAY foi portado para os microserviços criados, sempre com o *refactoring* necessário. (Cabral, 2019)

3.5. Gestão de Riscos

Um projeto com a envergadura e a complexidade do abrangido pelo presente trabalho, vem obrigatoriamente acompanhado de vários riscos, riscos esses que terão de ser devidamente analisados e mitigados. A equipa de IT analisou alguns riscos associados à execução deste projeto, tendo também identificado algumas ações mitigadoras para os mesmos: (Cabral, 2019)

- **Reação à mudança de paradigma e de *Tech Stack:*** a mudança que se planeou era uma mudança muito aguardada pela grande maioria da equipa e foi vista de forma muito positiva e altamente motivadora. As equipas/*squads* foram também preparadas com a formação específica para que pudessem desenvolver sob esta nova *Tech Stack*. (Cabral, 2019)
- **Portabilidade de código legado:** num projeto com datas e âmbitos ambiciosos, há por vezes a tentação de seguir pelo "caminho mais fácil". Neste caso essa tentação pode-se traduzir na tentativa de portar o código legado sem o *refactoring* adequado para a nova realidade. Na transposição do código-fonte da implementação atual para a nova *Tech Stack*, foi garantido o *refactoring* necessário para a correta adaptação à nova versão de ASP.NET Core 2.x/C#, efetuando os Testes de Regressão oportunos de forma a confirmar que não existia perda de funcionalidade. Desenvolvendo e divulgando *templates* de microserviços a utilizar, bem como *guidelines* a seguir no desenvolvimento, garantiu-se que o código-fonte seguia os padrões que se consideravam mais adequados. (Cabral, 2019)
- **Necessidade de *Rollback:*** a implementação de um mecanismo de configuração que garante que se consegue em tempo real e sem necessidade de modificação e *deploy* de *software*, reverter a entrega de qualquer funcionalidade nova/modificada. (Cabral, 2019)
- **Redefinição do Produto:** as funcionalidades que forem alvo de uma intervenção mais profunda ao nível de um redesenho completo de UI deverão ser analisadas, testadas em formato *mockup,* e detalhadas de forma atempada e segundo o plano a definir. (Cabral, 2019)
- **Tempo de Migração – Data Objetivo vs Âmbito Projeto:** tendo o projeto uma data objetivo a cumprir, o âmbito e o planeamento do mesmo devem ser alinhados com a mesma, e o progresso medido e monitorizado de forma regular para que se possam detetar possíveis desvios ao mesmo. (Cabral, 2019)

Capítulo 4
Fundamentos Metodológicos

4.1. Planeamento

O planeamento das 3 *releases* teve início no mês Fevereiro de 2019 e fim no mês de Novembro de 2020. O estágio teve início no dia 09 de Outubro de 2019 e terminou no dia 09 de Julho de 2020 (representado a laranja nas Tabelas 4.1 e 4.2). (Graça, 2019)

4.1.1. Planeamento da Release 1

Tabela 4.1 – Planeamento da *Release* 1.

	Meses											
	fev/19	**mar/19**	**abr/19**	**mai/19**	**jun/19**	**jul/19**	**ago/20**	**set/19**	**out/19**	**nov/19**	**dez/19**	**jan/20**
Release												
1.1												
1.2												
1.3												

4.1.2. Planeamento da Release 2 e da Release 3

Tabela 4.2 – Planeamento da *Release* 2 e da *Release* 3.

	Meses										
	jan/20	**fev/20**	**mar/20**	**abr/20**	**mai/20**	**jun/20**	**jul/20**	**ago/20**	**set/20**	**out/20**	**nov/20**
Release											
2											
3											

4.2. Metodologia

4.2.1. Spotify Engineering Model

No sentido de dar continuidade às medidas já implementadas pela organização – "*Vortal Agile Organization*" – que fomentam modos de trabalho colaborativos e ágeis, foi adaptado o "*Spotify Engineering Model*" às necessidades e realidades da empresa. (Bandeira, 2014)

4.2.1.1. Conceito

Este modelo consiste numa abordagem autónoma e orientada a pessoas, enaltecendo a importância da cultura e da rede. Como o nome indica, este modelo foi criado pela empresa Spotify, aumentando a inovação e produtividade da mesma, concentrando-se na autonomia, comunicação, responsabilidade e qualidade. O principal objetivo é concentrar a organização em torno do trabalho, pondo de parte certas práticas conhecidas, estruturando a organização e permitindo agilidade. Defende a autonomia de equipa, de modo a que cada uma delas (*squad*) selecione a sua estrutura (*scrum*, *kanban*, *scrumban*, ...). (Cruth, 2018)

4.2.1.2. Elementos Chave

O modelo Spotify centra a sua atividade na simplicidade, onde é definida a maneira como as pessoas e as equipas devem ser estruturadas.

- ***Squads*:** similar a uma equipa de *scrum*, são equipas multifuncionais e autónomas (6 a 12 pessoas) que se concentram numa área especifica. Cada *squad* tem uma missão, tem um *coach* ágil (C) e um *product owner* (P). As *squads* determinam qual a metodologia ágil a ser utilizada (ilustrado na Figura 4.1). (Mishra, 2020)

Figura 4.1 – Exemplificação de uma *squad*.

- ***Tribes*:** quando várias *squads* se coordenam na mesma área, formam uma tribo. As tribos são compostas por 40 a 150 pessoas, onde cada uma tem um líder responsável por coordenar as *squads* e incentivar a inovação e produtividade (ilustrado na Figura 4.2). (Mishra, 2020)

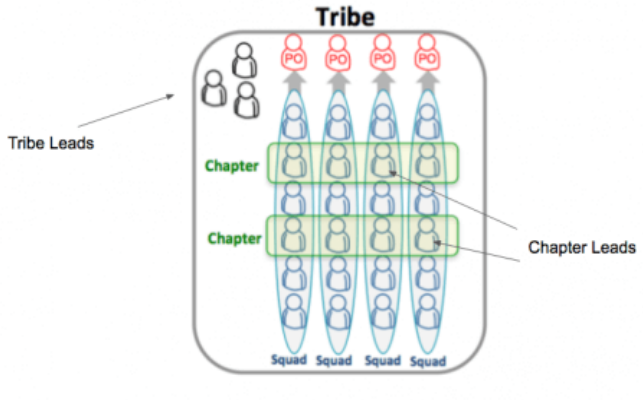


Figura 4.2 – Exemplificação de uma *tribe*.

- ***Chapter*:** é visto como um conjunto de pessoas que tem qualidades similares e trabalham na mesma área de competências. Um *chapter* pode ser de testes, de *Back-End*, de *Front-End* e de desenvolvimento web (ilustrado na Figura 4.3). (Mishra, 2020)

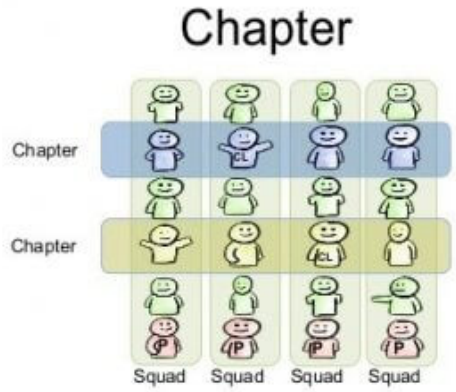


Figura 4.3 – Exemplificação de um *chapter*.

- ***Guild*:** como ilustrado na Figura 4.4, podemos considerar as *guilds* como grupos de interesse de uma determinada área, do qual cada um pode fazer parte, sem que exista um líder formal. Nestes grupos são partilhadas várias temáticas (conhecimento, ferramentas, código e práticas). (Mishra, 2020)

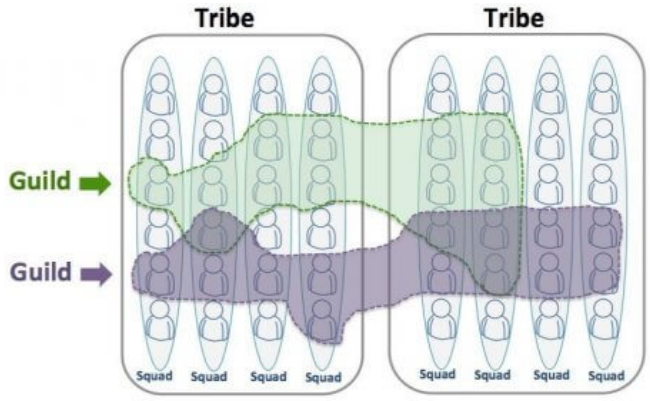


Figura 4.4 – Exemplificação de uma *guild*.

- ***Trio*:** é uma combinação entre o líder da *tribe*, líder do produto e líder de design. Cada tribo tem um trio de forma a garantir um alinhamento contínuo. (Mishra, 2020)

- ***Alliance*:** as alianças são uma combinação de tribos (normalmente de três ou mais) que trabalham com o objetivo de atingir algo maior (ilustrado na Figura 4.5). (Mishra, 2020)

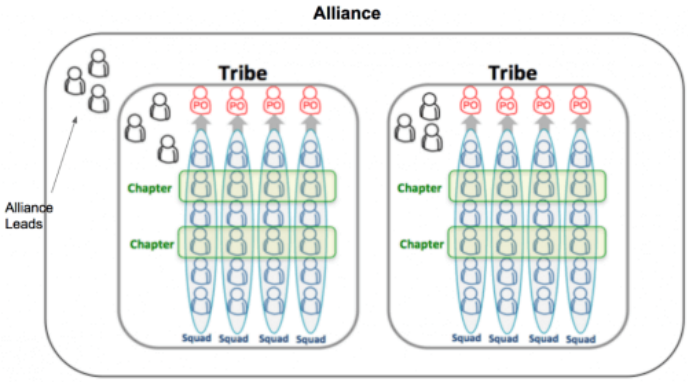


Figura 4.5 – Exemplificação de uma *alliance*.

4.2.1.3. Benefícios

- Assegura a melhoria continua;
- Permite um desenvolvimento interativo;
- Preserva a autonomia e confiança;
- Minimiza as dependências;
- Mantém a transparência com a comunidade;
- Origina *releases* pequenas e frequentes;
- Implementa a cultura de falhar rápido, aprender rápido e melhorar rápido.

(Cruth, 2018)

4.2.2. Agile

Por opção da equipa de coordenação, a empresa combina a prática de "*Agile*" com "DevOps", de modo a expandir o pensamento e desenvolvimento ágil com a entrega ágil. (Bandeira, 2014)

4.2.2.1. Conceito

Visto como um conjunto de metodologias de gestão de projeto e desenvolvimento de software baseado no desenvolvimento iterativo, tendo como principal objetivo ajudar as equipas a entregarem valor aos clientes com a máxima rapidez e a maior qualidade possível. Uma equipa ágil entrega o trabalho em pequenos incrementos. Requisitos, planos e resultados são avaliados continuamente para que as equipas tenham uma forma natural de responder às mudanças. Esta abordagem permite comunicação aberta, colaboração, adaptação e confiança entre os membros da equipa. Embora o gestor de projeto priorize o trabalho a ser entregue, a equipa assume a liderança na decisão de como o trabalho deverá ser feito, auto-organizando-se em torno das tarefas.

(Moran, 2014)

4.2.2.2. Benefícios

- **Clientes:** os clientes consolidam a ideia de que os fornecedores são mais responsivos às solicitações de desenvolvimento.
- **Fornecedores:** estes reduzem o desperdício concentrando o esforço de desenvolvimento em recursos de alto valor.
- **Equipas de Desenvolvimento:** permite aos membros das equipas de desenvolvimento, reduzindo o trabalho não produtivo, oferecendo-lhes tempo para realizarem o trabalho com valor.
- **Gestores de Produto:** estes são responsáveis por satisfazer os clientes, garantindo que o trabalho de desenvolvimento está alinhado com as necessidades apresentadas. Este tipo de metodologia oferece oportunidades frequentes para redefinir a prioridade de trabalho, de forma a garantir a entrega máxima de valor.
- **Gestores de Projeto:** através desta abordagem, os processos de planeamento e monitorização tornam-se mais fáceis e concretos, facilitando as tarefas dos gestores de projeto.

(Moran, 2014)

4.2.3. Aplicabilidade

De seguida são explicadas as funções de cada equipa, especificando com alguma minuciosidade a constituição hierárquica da equipa em que o estagiário esteve integrado e por fim é apresentado o fluxo de desenvolvimento.

4.2.3.1. Equipas do Projeto

Como representado na Tabela 5.1, o projeto possui várias equipas, onde cada uma tem uma função diferente.

Tabela 4.3 – Equipas do Projeto.

Equipa	Função
Arquitetura	Responsável pelo *design* da arquitetura da solução, definição de *templates* e *best pratices*, testes de *performance* e carga, *code review*, ...
Desenvolvimento (DEV)	Responsável pelos desenvolvimentos pedidos pelo cliente.
***Product Operations* (PO)**	Responsável pelas correções realizadas no ambiente de produção.
Base de Dados (BD)	Responsável pelas bases de dados de cada ambiente.
***Release Management* (RM)**	Responsável pelo lançamento de novas versões de software e pelas iniciativas de mudança.
***Quality Assurance* (QA)**	Responsável pelos testes de software, permitindo prevenir erros ou falhas antes da entrega do produto ao cliente.
***User Interface/User Experience* (UI/UX)**	Responsável pela construção dos ecrãs de interação entre os humanos e as máquinas.
***Solution Design* (SD)**	Responsável por definir e especificar os requisitos dos clientes em *user stories*, garantindo que as necessidades e expectativas do cliente são cumpridas.

4.2.3.2. Equipa de Desenvolvimento

Como é demonstrado na Figura 4.6, a equipa de desenvolvimento era constituída por um

Project Manager (PM), um *Team Leader* (TL), 17 programadores (11 programadores de *Front-End* e 9 programadores de *Back-End*).

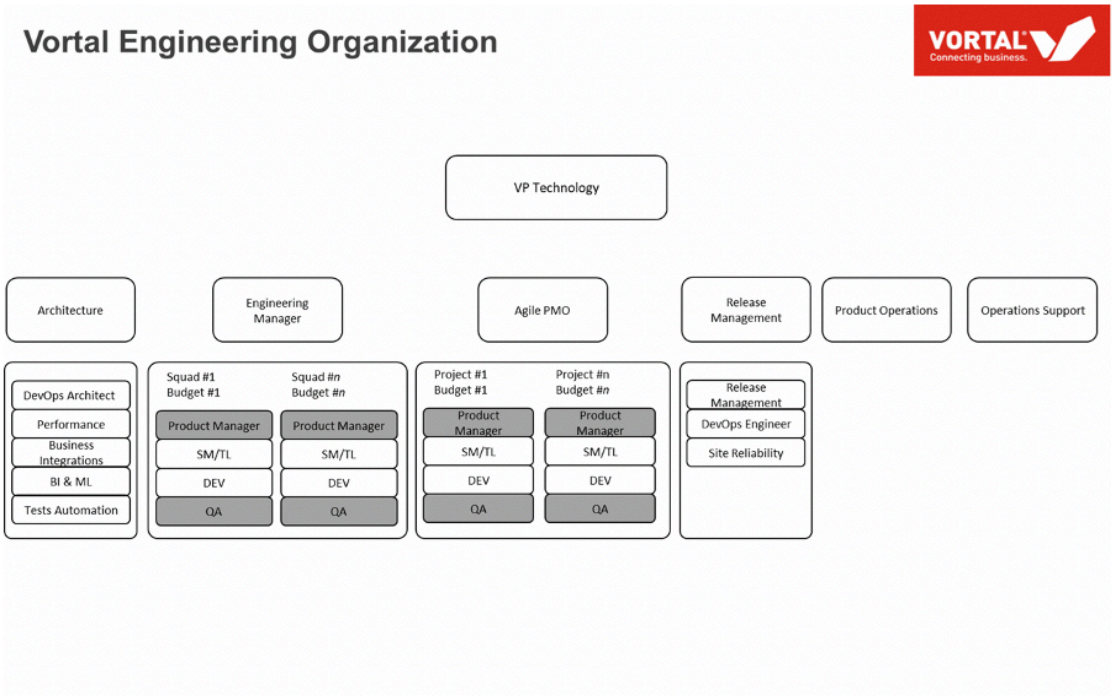


Figura 4.6 – Organização de Engenharia na Vortal.

Como já mencionado, para esta solução recorreu-se ao modelo Spotify e à metodologia *Agile*, visto que são efetuadas várias reuniões com o cliente o que implica um constante *feedback* e mudanças na especificação inicial. Com este método, existe a possibilidade de revisitar fases trabalhadas anteriormente e efetuar os ajustes necessários consoante os pedidos do cliente. (Fontana et al., 2014)

Além desta metodologia, pretende-se também seguir um modelo incremental, que consiste na entrega progressiva de módulos ao cliente, tornando mais fácil efetuar testes à plataforma (ilustrado na Figura 4.7). (Fontana et al., 2014)

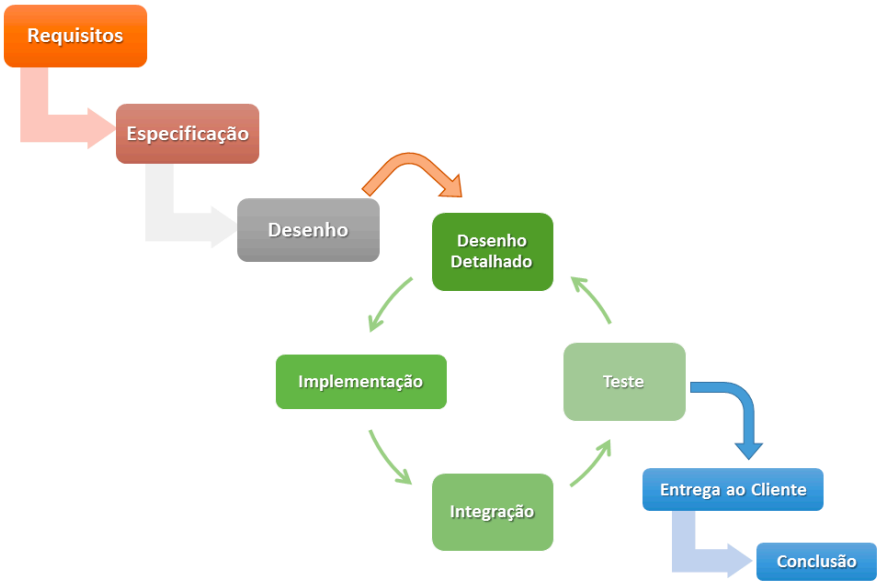


Figura 4.7 – Esquema ilustrativo da metodologia de desenvolvimento iterativa incremental.

No desenvolvimento deste projeto foram utilizadas as boas práticas do eXtreme Programming (XP), onde foram lançadas pequenas *releases* implementadas segundo as boas normas de programação.

Como explicado na Figura 4.8, a análise e especificação de requisitos foram elaborados não em função de cada entrega mas sim em documentos únicos relativos a cada módulo do projeto, permitindo assim ter uma visão global do sistema, garantindo a integridade do projeto. Nesta imagem é possível compreender o fluxo de desenvolvimento de cada módulo, onde são descritas as funções de cada equipa.

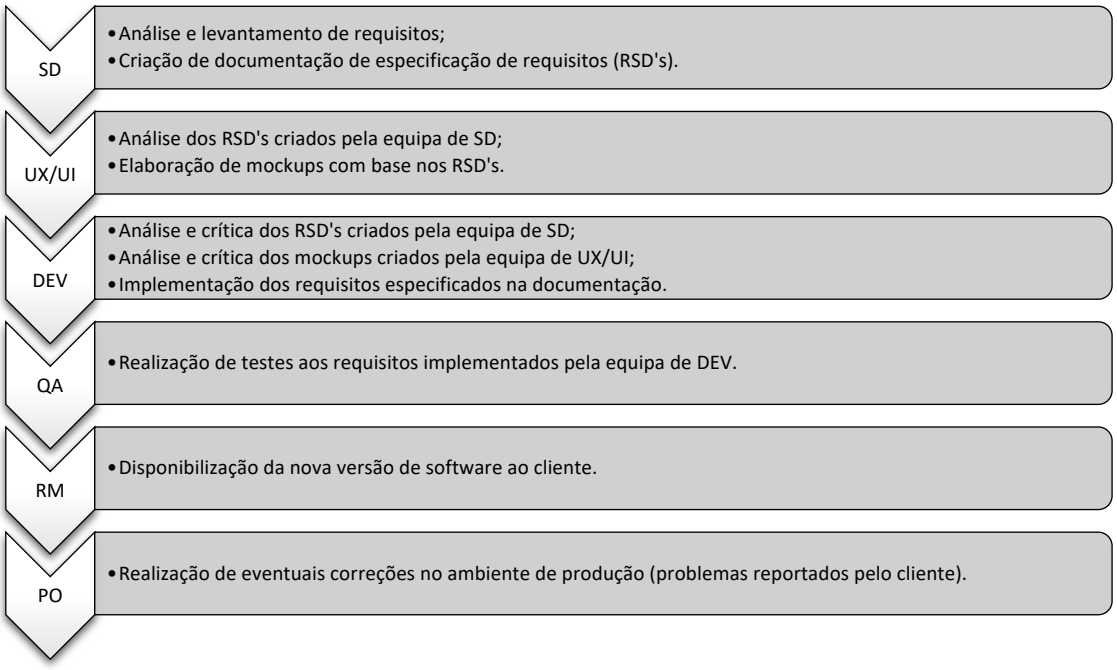


Figura 4.8 – Fluxo de Desenvolvimento.

É de referir que, por norma, existe a necessidade da equipa de DEV intervir após os testes realizados pela equipa de QA (*bugfixing*). Só após serem resolvidos os demais problemas reportados é que o software pode ser lançado.

4.3. Ferramentas e Tecnologias Utilizadas

4.3.1. Ambientes de Desenvolvimento

Tabela 4.4 – Ambientes de Desenvolvimento utilizados.

Back-End (BE)	*Front-End* (FE)	Microserviços	Documentação BE
 **Visual Studio 2019**	 **Visual Studio Code**	 **Docker Desktop**	 **Swagger**

4.3.1.1. Visual Studio 2019

É um ambiente de desenvolvimento integrado, pertencente à Microsoft, tendo como principal objetivo o desenvolvimento de software especialmente dedicado ao .NET Framework e às linguagens Visual Basic, C, C++, C# e F#. Visto como um produto de desenvolvimento também para a área web, usa a plataforma do ASP.NET, como *websites*, aplicações *web*, serviços *web* e aplicações móveis (Strauss, 2020). Relativamente ao projeto em que o estagiário foi integrado, este foi o ambiente escolhido para suportar todo o desenvolvimento de serviços que alimentam a parte de *Back-End* do projeto.

4.3.1.2. Visual Studio Code

Esta ferramenta suporta o desenvolvimento de projetos em inúmeras linguagens ou bibliotecas recentes, como JavaScript e React.js. O IDE oferece estabilidade aos programadores e a sua interação permite personalizá-lo com temas e extensões à medida de cada um (Visual Studio Code, 2015). Para este projeto foram usadas as linguagens JavaScript, Node.js, HTML, CSS, React.js, tendo sido escolhido para que de forma fácil e interativa fossem desenvolvidos todos os comportamentos visíveis do projeto em questão.

4.3.1.3. Docker Desktop

Docker é uma ferramenta projetada para facilitar a criação, implementação e *deploy* de aplicações usando *containers*. Os *containers* permitem ao programador empacotar a aplicação com todas as partes necessárias, tais como bibliotecas e outras dependências, e lançar tudo como um único pacote. Ao fazer isso, graças ao *container*, o programador pode ter certeza de que a aplicação será executada em qualquer outra máquina, independentemente das configurações personalizadas que a máquina possa ter e que possam diferir da máquina usada para escrever e testar o código. (Martin, 2011)

A ferramenta Docker foi escolhida para o projeto nextVISION, visto que este é um projeto baseado em microserviços e o Docker proporciona a execução de várias aplicações num único servidor, mantendo os componentes de cada aplicação em *containers* separados, evitando problemas com a gestão de dependências. Outra razão pela qual a ferramenta foi escolhida, foi pelo facto de oferecer a possibilidade de integrações contínuas e entregas contínuas, pois através de *containers* o código é facilmente *deployed* para produção, proporcionando às equipas de *Quality Assurance* (QA) a possibilidade de realização de testes rápidos, permitindo também aos clientes que acompanhem os desenvolvimentos realizados.

4.3.1.4. Swagger

Esta ferramenta permite que os programadores possam projetar, criar, documentar e consumir serviços da Web RESTful. As ferramentas associadas ao Swagger incluem suporte para documentação automatizada, geração de código e geração de casos de teste. (Surwase, 2016)

No projeto nextVISION esta foi uma ferramenta essencial para que o programador de *Front-End* pudesse testar todos os serviços desenvolvidos e até detetar problemas relacionados com o mesmo, sem a necessidade de criar qualquer ligação aos mesmos.

4.3.2. Ferramentas de Gestão

Tabela 4.5 – Ferramentas de Gestão utilizadas.

Práticas de DevOps	Gestão de Versões	Gestão de Reuniões
Azure DevOps	**GIT**	**Outlook Calendar**

4.3.2.1. Azure DevOps

Azure DevOps é uma ferramenta com soluções completas para as equipas conseguirem implementar práticas de DevOps em todas as fases do ciclo de vida das aplicações: planeamento, entrega e funcionamento e como tal também se poderia categorizar como uma ferramenta de gestão. Foi escolhida esta ferramenta pois permite que as equipas acrescentem valor continuado aos seus clientes, proporcionado os seguintes serviços: (Rossberg, 2019)

- ***Boards*:** é uma ferramenta de planeamento e de gestão de tarefas;
- ***Repos*:** repositórios GIT privados ilimitados com análises de código;
- ***Test Plans*:** para facilitar os testes manuais;
- ***Artifacts*:** repositório universal de *packages* (Maven, npm, NuGet e Python), integração de partilha de *packages* nos *pipelines* de CI/CD de forma fácil e dimensionável.

4.3.2.2. GIT

Para o controlo de versões dos projetos é utilizado o GIT. É um sistema de controlo de versões distribuído, grátis e *open source* mais utilizado em todo o mundo e é considerado como um *standard* para controlo de versões. GIT pode não ser só utilizado para projetos pequenos como pode ser utilizado em projetos mais complexos, mantendo a rapidez e eficiência. Uma das particularidades do GIT é a sua facilidade de aprendizagem e possui uma ótima performance. Ao ser distribuído, significa que a cópia local do código é um repositório de controlo de versão completo. Para os repositórios do projeto nextVISION foi utilizado o servidor Azure DevOps Git Server e o GIT GUI utilizado no desktop foi o *Source Tree*. A escolha do GIT apresenta os seguintes benefícios: (Git, 2018)

- **Desenvolvimento simultâneo**: Todos os programadores têm a sua cópia local e podem trabalhar normalmente nos seus ramos (*branches*). O GIT funciona até *offline* pelo que todas as operações são locais;
- **"Releases" rápidas**: Ao ter ramos repartidos com todos os programadores torna o desenvolvimento de um projeto como algo flexível e simultâneo. Existe um ramo principal e outros ramos que adicionam mais funcionalidades que ao estarem concluídos, serão combinados com o ramo principal;
- **Integração incluída**: Ao ser o controlo de versões mais popular internacionalmente, o GIT está integrado na maior parte das ferramentas e produtos;
- **Forte apoio da comunidade**: GIT é *open-source* e é considerado um *standard* para controlo de versões;
- **Benéfico para equipas de trabalho**: Ao utilizar GIT é possível incrementar a produtividade da equipa de trabalho, reforçar políticas, automatização de processos e melhorar a visibilidade e rastreamento de trabalho.

- **Pedidos *Pull***: Ao usar pedidos *pull* (*pull requests*) podem ser discutidas mudanças ao código antes do código ser integrado com o ramo principal.
- **Políticas de ramos**: Podem ser configurados os serviços de equipa do *Visual Studio* para reforçar processos e fluxos de trabalho consistentes para toda a equipa.

4.3.2.3. Outlook Calendar

Considerado como serviço de agenda e calendário on-line, disponível em interface web, este serviço permite agendar e controlar reuniões e compromissos empresariais, partilhando os agendamentos com outras pessoas e podendo ainda usufruir de várias funcionalidades agregadas.

4.3.3. Tecnologias Utilizadas

Tabela 4.6 – Tecnologias utilizadas.

Back-End (BE)	*Front-End* (FE)		*Icons*
C# (C Sharp)	JavaScript (JS)	TypeScript (TS)	Fontawesome
	React	Redux	
	SASS	Ant Design	

4.3.3.1. C# (C Sharp)

C# é uma linguagem de programação, multiparadigma, de tipagem forte, desenvolvida pela Microsoft como parte da plataforma .NET. A sua sintaxe orientada a objetos foi baseada no C++ mas inclui muitas influências de outras linguagens de programação, como Object

Pascal e, principalmente, Java. O código fonte é compilado para *Common Intermediate Language* (CIL) que é interpretado pela máquina virtual *Common Language Runtime* (CLR). C# é uma das linguagens projetadas para funcionar na *Common Language Infrastructure* (CLI) da plataforma .NET Framework. (Microsoft dotnet csharp, 2020)

A linguagem C# foi pensada para ser simples, moderna e orientada a objetos. A linguagem e suas implementações fornecem suporte para princípios de engenharia de software, tais como verificação de tipo forte, verificação de limites de *array*, deteção de tentativas de usar variáveis não inicializada. (Microsoft dotnet csharp, 2020)

4.3.3.2. JavaScript (JS)

Como linguagem principal, na vertente de *Front-End,* para o desenvolvimento do projeto nextVISION foi usada a linguagem JavaScript (em conjunto com a biblioteca React.js). JavaScript ou JS é uma linguagem dinâmica, baseada em protótipos e que suporta os paradigmas de programação orientada a objetos, imperativa e declarativa (incluindo programação funcional). Juntamente com HTML e CSS, o JavaScript é uma das três principais tecnologias da *web*. JavaScript permite páginas da Web interativas e, portanto, é uma parte essencial das aplicações da web. A grande maioria dos websites usa componentes implementados nesta linguagem, e todos os principais navegadores têm um interpretador de JavaScript dedicado para executar scripts implementados nesta linguagem. É atualmente a principal linguagem para programação *client-side* em navegadores *web*. É também bastante utilizada do lado do servidor através de ambientes como o Node.js. (Steyer & Steyer, 2014)

4.3.3.3. TypeScript (TS)

O TypeScript é um superconjunto da linguagem JavaScript que possui um único compilador de código *open source* e é desenvolvido principalmente por um único fornecedor: a Microsoft. O objetivo do TypeScript é ajudar a detetar erros através de um sistema de tipos e tornar o desenvolvimento do JavaScript mais eficiente. (Bierman et al., 2014)

O suporte dos tipos não faz parte do padrão ECMAScript e foi por essa razão que o TypeScript foi criado. O sistema de tipos do TypeScript é incrivelmente rico e inclui: interfaces, enumerações, tipos híbridos, genéricos, tipos de união / interseção, modificadores de acesso e muito mais. O sistema de tipos do TypeScript está a par da maioria das outras linguagens e, em alguns casos, é sem dúvida mais poderoso. (Bierman et al., 2014)

O uso do TypeScript também nos permite obter os benefícios do IntelliSense, ou seja, oferece sugestões de código. Em conclusão, TypeScript é um superconjunto de JavaScript, mas com um sistema de tipos. (Bierman et al., 2014)

4.3.3.4. React.js

O React (também denominado React.js ou ReactJS) é uma biblioteca JavaScript de código aberto com foco em criar interfaces de utilizador (*Front-End*) em aplicações web. Esta biblioteca de JavaScript pode ser usada para criar aplicações "*single-page*" pelo que aplicações complexas de React.js necessitam de bibliotecas adicionais para controlo de estados, *routing* e interação com alguma API. (Banks & Porcello, 2017)

O React é composto por entidades chamadas componentes. Os componentes podem ser renderizados para um elemento específico no DOM (*Document Object Model*) usando a biblioteca React DOM. Ao renderizar um componente, podem-se passar valores conhecidos como "props". Os componentes funcionais são declarados com uma função que retorna algum JSX (JavaScript XML). Componentes baseados em classes são declarados usando as classes ES6. São também conhecidos como componentes "com estado", porque o seu estado pode conter valores em todo o componente, podendo ser transmitido aos componentes filhos por meio de "props". (Banks & Porcello, 2017)

4.3.3.5. Redux.js

O Redux é uma ferramenta de gestão de estado, em particular, um *container* de estado previsível para aplicações JavaScript. O Redux funciona como um repositório que permite armazenar o estado dos diversos componentes utilizados, facilitando a manutenção e gestão dos diferentes estados, principalmente em situações que a mudança de um estado de um componente, requer que outros também transitem de estado. Ao realizar a gestão do estado ao nível do componente, dificulta a perceção dos estados dos outros componentes, algo que o Redux veio facilitar. (Banks & Porcello, 2017)

Se os dados de um componente residirem em apenas um componente, torna-se difícil de os partilhar entre componentes irmãos. Por exemplo, no React, para partilhar dados entre irmãos, um estado precisa residir no componente pai, mesmo que os dados não sejam necessários. Em seguida, esses dados são transmitidos como "props" aos elementos filhos, juntamente com um método para atualizar o estado. (Banks & Porcello, 2017)

Essa abordagem torna-se problemática quando o estado é partilhado entre componentes distantes na árvore de componentes. O estado terá que ser colocado no componente ancestral mais próximo e, em seguida, passado pela árvore filho até ser recebido pelo componente que precisa dele. Isso torna o estado difícil de manter e menos previsível. (Banks & Porcello, 2017)

O Redux aborda esse problema fornecendo um armazenamento central que mantém todo o estado da aplicação. (Banks & Porcello, 2017)

4.3.3.6. SASS (Syntactically Awesome Style Sheets)

O SASS é um pré-processador de CSS, que adiciona funcionalidades especiais como

variáveis, regras aninhadas (ou *nested*) ao CSS regular. O objetivo é tornar o processo de codificação mais simples e eficiente. Um pré-processador CSS é uma linguagem que estende o CSS, permitindo que o código seja escrito numa linguagem e depois é compilado no CSS. SASS é talvez o pré-processador mais popular do mundo. As funcionalidades principais do SASS incluem: (SASS, n.d.)

- **Variáveis:** permite o uso de variáveis que podem armazenar informações para serem usadas na folha de estilos. Por exemplo, pode-se armazenar um valor de cor numa variável na parte superior do arquivo e, em seguida, usar essa variável ao definir a cor dos seus elementos.
- ***Nesting*:** a ideia é aninhar seus seletores de CSS de forma a imitar sua hierarquia HTML.
- **Importar ficheiros:** é possível importar outros ficheiros SASS num único ficheiro.

4.3.3.7. Ant Design

O Ant Design é uma biblioteca de UI do React que consiste num conjunto de demonstradores e componentes de alta qualidade para a criação de interfaces de utilizador interativas e ricas. Os componentes React de alta qualidade são prontos a usar, escritos em TypeScript com tipos definidos completos. Embora a Material-UI continue a ser a biblioteca React UI mais popular, o Ant Design está atualmente em segundo lugar e está a crescer cada vez mais. (Ant Design, n.d.)

4.3.3.8. Font Awesome

Font Awesome é um conjunto de ferramentas de tipos de letra e ícones baseado em CSS e LESS. Os ícones são importantes para projetos em Web e não só, pois são uma maneira visual de ajudar a adicionar significado aos elementos. O Font Awesome permite uma adição fácil às páginas, economizando um tempo valioso porque assim não é necessário criar os ícones, mas sim apenas usá-los. (Awesome, n.d.)

É importante fornecer uma experiência de rapidez aos utilizadores quando abrem uma página. Todos os ícones estão incluídos num ficheiro de tipo de letra, portanto, é necessário apenas um pedido HTTP para carregar o Font Awesome.

Capítulo 5
Trabalho Realizado

5.1. Arquitetura Proposta

Como referido anteriormente, a nextVISION é uma plataforma complexa, de grande escala e com muitas funcionalidades. Esta utiliza uma arquitetura de microserviços, em que cada um utiliza o padrão de software *Model View Controller* (MVC). (Cabral, 2019)

A ligação entre o *Front-End* e o *Back-End* de um microserviço é feito através de *Back-End-In-Front-End* (BIF), ou seja, em vez do *Front-End* realizar chamadas diretas ao *Back-End*, existe um intermediário responsável por tal, o BIF. (Cabral, 2019)

Na Figura 5.1 encontra-se representada a arquitetura completa deste projeto. (Cabral, 2019)

Figura 5.1 – Arquitetura do Projeto.

Um microserviço pode-se dividir nas seguintes partes:

- **BackendApp:** onde se encontram os *controllers* e a lógica do microserviço.

- **ClientApp:** onde se encontram os componentes, as páginas de visualização, as folhas de estilo e o ficheiro de código gerado a partir do Back-End.

5.2. Detalhes de Implementação

Nesta secção serão descritos os atores intervenientes no processo, assim como os detalhes de implementação referentes a cada *release*, dividindo cada uma das *releases* pelos componentes implementados pelo estagiário.

5.2.1. Atores

Na Tabela 5.1 são descritos os atores participantes neste projeto.

Tabela 5.1 – Descrição dos atores.

Nome	Descrição
Comprador (*Buyer*)	Entidade responsável pela criação e publicação de um procedimento. Esta entidade pode também avaliar uma proposta e adjudicar determinados procedimentos.
Fornecedor (*Supplier*)	Entidade responsável pela criação e publicação de uma proposta.
Utilizador (*User*)	Entidade que faz uso da plataforma, tanto pode ser um comprador, como um fornecedor.

5.2.2. Release 1.2

Para esta entrega, foram entregues os módulos mencionados na tabela seguinte (Tabela 5.2).

Tabela 5.2 – Descrição dos módulos entregues na *release 1.2*.

Módulo	Descrição
Tenders Manager	Módulo responsável pela gestão de procedimentos, a este são associadas múltiplas funcionalidades.
Tenders List	Módulo responsável pela apresentação das propostas disponíveis.

5.2.2.1. Fase de Desenvolvimento

Visto que o estágio iniciou em Outubro de 2019, o estagiário não participou na Fase de Desenvolvimento da *Release 1.2.*

5.2.2.2. Fase de Certificação

Esta foi a fase de arranque do estágio, podendo considerar-se preponderante para a integração do estagiário no projeto e na equipa. Através da correção de problemas reportados relativamente às funcionalidades implementadas na Fase de Desenvolvimento, o estagiário obteve conhecimento da lógica de negócio e da plataforma nextVISION.

Ao longo de todo o estágio, o aluno tentou esquematizar mentalmente os problemas que lhe eram impostos. Através de uma metodologia própria "Problema/Solução" o estagiário tentava esquematizar uma solução rápida e eficaz consoante o problema proposto.

Na Fase de Certificação da *Release 1.2* foi imposto ao estagiário a criação de dois componentes genéricos, de forma a serem usados transversalmente pelas diferentes seções da plataforma (*ComponentTabs* e *DateTimePopover*).

É importante referir que os resultados destas implementações serão apresentados e explicados na secção "6.3 Discussão de Resultados".

5.2.2.2.1. ComponentTabs

Surgiu a necessidade de criar um componente que, consoante determinadas configurações, disponibilizasse diferentes tipos de *tabs* (*Default Tabs*, *Slide Tabs* e *Card Tabs*), contadores (fixos e variáveis) e que ainda permitisse adicionar novas *Tabs*. Os *mockups* disponibilizados ao estagiário encontram-se ilustrados na Figura 5.2, na Figura 5.3 e na Figura 5.4. (Afonso, 2019c)

Figura 5.2 – *Mockup* das *Default Tabs*.

Figura 5.3 – *Mockup* das *Slide Tabs*.

Figura 5.4 – *Mockup* das *Card Tabs*.

Como ilustrado na Figura 5.5, o estagiário utilizou o componente <Tabs />, pertencente à biblioteca Ant Design, sendo que cada <TabPane /> foi renderizado consoante as configurações definidas no *array* de objetos *tabs* passado por *props* aquando da chamada do componente.

```
render() {
  let tabs = this.props.tabs.map((tab, i) => {
    return (
      <TabPane
        key={tab.key}
        tab={
          <TitleTab
            onClickInfo={tab.onClickInfo}
            title={tab.title}
            variableCounter={tab.variableCounter}
            fixedCounter={tab.fixedCounter}
            hasIcon={tab.hasIcon}
          />
        }
        disabled={
          this.props.disabled === undefined ? false : this.props.disabled
        }
        closable={tab.closable}
      >
        {tab.component}
      </TabPane>
    );
  });

  return (
    <Tabs
      hideAdd={true}
      className={this.decideClassNames()}
      destroyInactiveTabPane={true}
      activeKey={this.props.activeKey}
      onChange={this.props.onChange}
      tabBarExtraContent={this.props.tabBarExtraContent}
      type={this.props.type || "line"}
      onEdit={this.props.onEdit}
    >
      {tabs}
    </Tabs>
```

Figura 5.5 – Renderização da interface gráfica do componente *ComponentTabs*.

Como é possível observar na Figura 5.6, de forma a facilitar o uso do componente foi definida uma tipagem própria para o objeto *tabs* (ITab).

```
interface ITab {
  key?: string;
  title: any;
  component?: any;
  closable?: boolean;
  hasIcon?: boolean;
  onClickInfo?(): void;
  variableCounter?: number;      José Carlos Ferreira da Silva, 3 months ago • Updated ComponentTabs component to receive two counter
  fixedCounter?: number;
}

José Carlos Ferreira da Silva, 3 months ago | 3 authors (pedroferreira and others)
interface IProps {
  tabs: ITab[];
```

Figura 5.6 – Definição da interface do objeto *tabs*.

Após a criação e publicação do componente, o mesmo ficou pronto a ser utilizado. A Figura 5.7 apresenta um exemplo de integração do componente criado.

```
<ComponentTabs
  tabs={tabs}
  disabled={false}
  type="editable-card"
  onEdit={this.onEdit}
  onChange={this.onChange}
/>
```

Figura 5.7 – Exemplo de integração do componente *ComponentTabs* (Tipo: *Card Tabs*).

Na Figura 5.8 encontra-se exemplificado o objeto *tabs* que anexa todas as configurações do *ComponentTabs*.

```
const tabs = [
  {
    key: "1",
    title: "WITH INTEREST",
    onClickInfo: () => this.onChangeVisibility(true),
    hasIcon: true,
    component: <Content text="WITH INTEREST" />,
    closable: false
  },
  {
    key: "2",
    title: "LEADS",
    onClickInfo: () => this.onChangeVisibility(true),
    hasIcon: true,
    component: <Content text="LEADS" />,
    closable: false
  },
  {
    key: "3",
    title: "PUBLIC",
    onClickInfo: () => this.onChangeVisibility(true),
    hasIcon: true,
    component: <Content text="PUBLIC" />,
    closable: false
  },
  {
    key: "4",
    title: "OPPORTUNITY",
    component: <Content text="OPPORTUNITY" />
  }
];
```

Figura 5.8 – Exemplo do objeto *tabs* com as configurações do *ComponentTabs*.

5.2.2.2.2. DateTimePopover

Esta funcionalidade exigiu a criação de um componente que disponibilizasse um

DateTimePicker que, quando selecionado, apresentasse paralelamente a seleção da data e a seleção do tempo (Figura 5.9). (Afonso, 2019d)

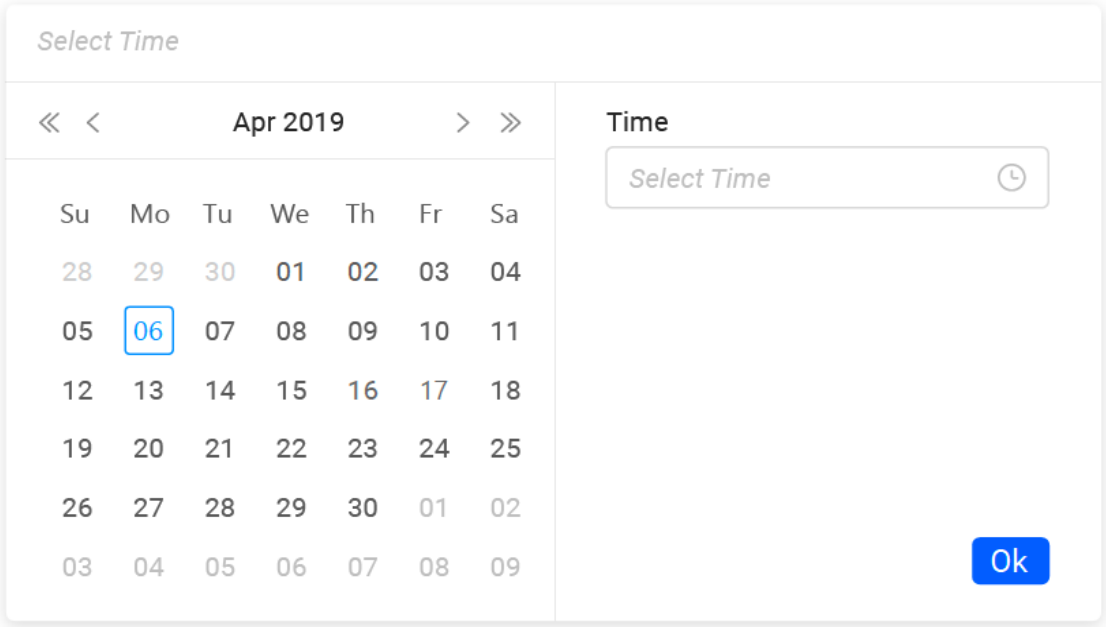


Figura 5.9 – *Mockup* do *DateTimePicker* quando selecionado.

Foi também pedido que quando o *DateTimePicker* não estivesse selecionado tivesse uma apresentação diferente, onde se pudesse digitar manualmente a data e o tempo (Figura 5.10 e Figura 5.11). (Afonso, 2019d)

Figura 5.10 – *Mockup* do *DateTimePicker* quando não está selecionado nem preenchido.

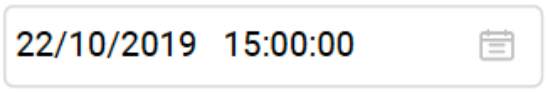


Figura 5.11 – *Mockup* do *DateTimePicker* quando não está selecionado, mas está preenchido.

Como ilustrado na Figura 5.12, o estagiário utilizou o componente <Popover />, pertencente ao Ant Design, de forma a renderizar o DateTimePicker sempre que este esteja visível. A visibilidade do Popover depende da seleção do campo <InputMask />, pertencente à

biblioteca “react-input-mask”, que permite definir um formato daquilo que é pretendido se escrever na caixa de texto.

```
<Popover
  getPopupContainer={(triggerNode: any) => triggerNode.parentNode}
  content={content}
  trigger="click"
  placement="bottomLeft"
  visible={this.state.visible}
  onVisibleChange={this.handleVisibleChange}
  autoAdjustOverflow={true}
  overlayClassName="calendar-container"
>
  <InputMask
    mask="99/99/9999 99:99:99"
    value={rawValue}
    onChange={e=>this.onChange(e)}
  />
</Popover>
```

Figura 5.12 – Renderização da interface gráfica do componente *DateTimePopover*.

O conteúdo do componente <Popover /> é a combinação entre os componentes <Calendar /> e <TimePicker />, ambos pertencentes à biblioteca Ant Design (Figura 5.13).

```
const content = (
  <>
    <div>
      <Calendar
        value={dateTimeValue}
        fullscreen={false}
        onSelect={this.onSelect}
        onPanelChange={this.onPanelChange}
        headerRender={({ onChange }) => {···
        }}
      />
    </div>

    <div className="time-content">
      <span className="ant-form-text">Time</span>
      <TimePicker value={dateTimeValue} onChange={this.onPanelChange} />      José Carlos Ferreira da Silva, a year ago • Merge b
      <Button type="primary" onClick={this.saveDateTime} className="btn-ok">
        OK
      </Button>
    </div>
  </>
);
```

Figura 5.13 – Conteúdo do componente <Popover />.

Na Figura 5.14 é exemplificado o simples uso do componente <DateTimePopover />.

```
<section className="component-box-inner">
  <DateTimePopover />
</section>
```

Figura 5.14 – Exemplo de integração do componente *DateTimePopover*.

5.2.3. Release 1.3

Para esta entrega, foi planeado entregar os módulos mencionados na tabela seguinte

(Tabela 5.3).

Tabela 5.3 – Descrição dos módulos entregues na *release 1.3*.

Módulo	Descrição
Teams Manager	Módulo responsável pela gestão e criação de equipas de propostas.
Company Registration	Módulo responsável pelo registo de empresas.

5.2.3.1. Fase de Desenvolvimento

Visto que durante a Fase de Certificação da *Release 1.2* o estagiário criou dois componentes genéricos com sucesso (*ComponentTabs* e *DateTimePopover*), foi lhe pedido a criação de mais quatro componentes de forma a serem usados transversalmente por diferentes secções do projeto (*FeedbackToast*, *ErrorBoundary*, *CultureUtils* e *PillLabel*). Os resultados destes componentes serão apresentados e explicados na secção "6.3 Discussão de Resultados".

5.2.3.1.1. CultureUtils

Visto que para bibliotecas externas (Ant Design, Moment, ...) as *keys* de tradução são diferentes (Figura 5.15), existiu a necessidade de se criar um componente que retorne a *key* correta para o contexto em que se pretende traduzir (Figura 5.16).

```
const localesList = {
  "ca-ES": { moment: "ca", antd: "ca_ES", decimalSeparator: "ca" },
  "ca-ESGov": { moment: "ca", antd: "ca_ES", decimalSeparator: "ca" },
  ca: { moment: "ca", antd: "ca_ES", decimalSeparator: "ca" },
  caGov: { moment: "ca", antd: "ca_ES", decimalSeparator: "ca" },
  cs: { moment: "cs", antd: "cs_CZ", decimalSeparator: "cs" },
  csGov: { moment: "cs", antd: "cs_CZ", decimalSeparator: "cs" },
  "de-AT": { moment: "de", antd: "de_DE", decimalSeparator: "de" },
  "de-ATGov": { moment: "de", antd: "de_DE", decimalSeparator: "de" },
  de: { moment: "de", antd: "de_DE", decimalSeparator: "de" },
  deGov: { moment: "de", antd: "de_DE", decimalSeparator: "de" },
  en: { moment: "en-gb", antd: "en_GB", decimalSeparator: "en-gb" },
  enCDE: { moment: "en-gb", antd: "en_GB", decimalSeparator: "en-gb" },
  enEConstroi: { moment: "en-gb", antd: "en_GB", decimalSeparator: "en-gb" },
  enGov: { moment: "en-gb", antd: "en_GB", decimalSeparator: "en-gb" },
  "es-CO": { moment: "es", antd: "es_ES", decimalSeparator: "es" },
  "es-MX": { moment: "es", antd: "es_ES", decimalSeparator: "es" },
  "es-PE": { moment: "es", antd: "es_ES", decimalSeparator: "es" },
  es: { moment: "es", antd: "es_ES", decimalSeparator: "es" },
  "es-ES": { moment: "es", antd: "es_ES", decimalSeparator: "es" },
  esGov: { moment: "es", antd: "es_ES", decimalSeparator: "es" },
  fr: { moment: "fr", antd: "fr_FR", decimalSeparator: "fr" },
  frCDE: { moment: "fr", antd: "fr_FR", decimalSeparator: "fr" },
  frGov: { moment: "fr", antd: "fr_FR", decimalSeparator: "fr" },
  "gl-ES": { moment: "gl", antd: "es_ES", decimalSeparator: "gl" },
  "gl-ESGov": { moment: "gl", antd: "es_ES", decimalSeparator: "gl" },
  "it-IT": { moment: "it", antd: "it_IT", decimalSeparator: "it" },
  "it-ITGov": { moment: "it", antd: "it_IT", decimalSeparator: "it" },
  pl: { moment: "pl", antd: "pl_PL", decimalSeparator: "pl" },
  "pl-PL": { moment: "pl", antd: "pl_PL", decimalSeparator: "pl" },
  pt: { moment: "pt", antd: "pt_PT", decimalSeparator: "pt" },
  "pt-PT": { moment: "pt", antd: "pt_PT", decimalSeparator: "pt" },
  ptEConstroi: { moment: "pt", antd: "pt_PT", decimalSeparator: "pt" },
  ptGov: { moment: "pt", antd: "pt_PT", decimalSeparator: "pt" },
  sl: { moment: "sl", antd: "sl_SI", decimalSeparator: "sl" },
  slGov: { moment: "sl", antd: "sl_SI", decimalSeparator: "sl" }
};
```

Figura 5.15 – Objeto mapeador da *key* em função do contexto.

```
export const getLocale = (locale: Locale) => {
  if (locale && localesList[locale.culture]) {
    return localesList[locale.culture][locale.context];
  } else {
    return localesList["en"][locale.context];       José Carlos Ferreira da Silva, a year ago • Refactored culture utils
  }
};
```

Figura 5.16 – Método que retorna a *key* do contexto desejado.

É importante referir que o objeto passado tem de ser do tipo *Locale* (interface ilustrada na Figura 5.17).

```
interface Locale {
  culture: string;
  context: "antd" | "moment" | "decimalSeparator";
}
```

Figura 5.17 – Definição da *interface Locale*.

Após o *import* da função getLocale do componente criado, a sua integração é bastante simples (Figura 5.18).

```
componentDidMount() {
  let cultureCode;

  if ((window as any) && (window as any).sessionInfo) {
    cultureCode = (window as any).sessionInfo.cultureCode;
  }
  let momentLocale = getLocale({
    culture: cultureCode,
    context: "moment"
  });

  import(`moment/locale/${momentLocale}`);
}
```

Figura 5.18 – Exemplo de integração da função getLocale do componente *CultureUtils*.

5.2.3.1.2. FeedbackToast

Esta funcionalidade é transversal a diferentes secções da plataforma, dessa forma surgiu a necessidade de criar um componente que disponibilizasse diferentes tipos de mensagens de alerta, com títulos e descrições variáveis. Os *mockups* disponibilizados encontram-se ilustrados na Figura 5.19, Figura 5.20, Figura 5.21 e Figura 5.22. (Afonso, 2019b)

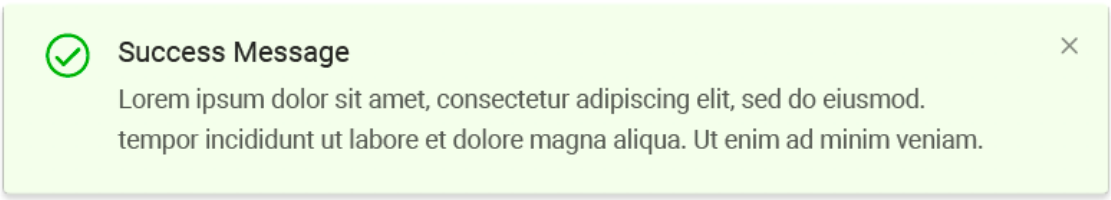


Figura 5.19 – *Mockup* da mensagem de sucesso.

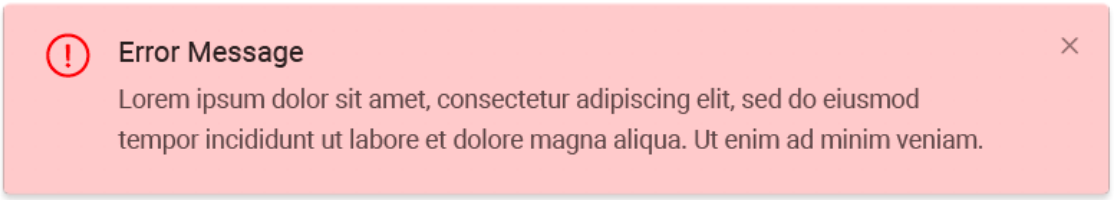


Figura 5.20 – *Mockup* da mensagem de erro.

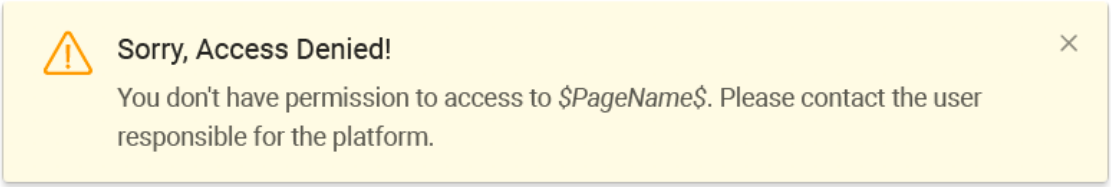


Figura 5.21 – *Mockup* da mensagem de aviso.

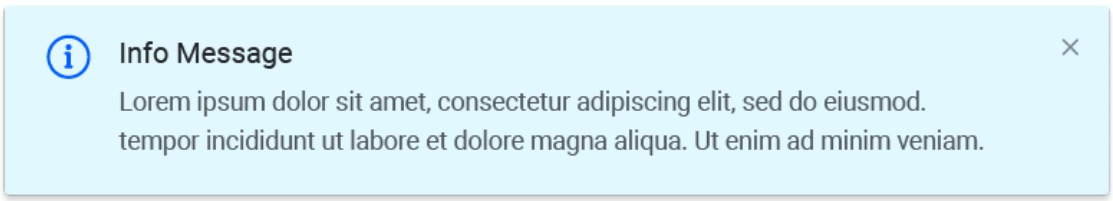


Figura 5.22 – *Mockup* da mensagem de informação.

Como ilustrado na Figura 5.23, o estagiário utilizou o objeto *notification*, pertencente à biblioteca Ant Design, tendo sido criado um método que retorna a mensagem de alerta consoante o tipo de *feedback* passado por *props* (*success*, *info*, *warning* e *error*).

Figura 5.23 – Retorno da mensagem de alerta, dependendo do tipo de *feedback* passado por *props*.

Como é possível observar na Figura 5.24, de forma a facilitar o uso do componente foi definida uma tipagem própria para o objeto passado por *props* (IFeedback).

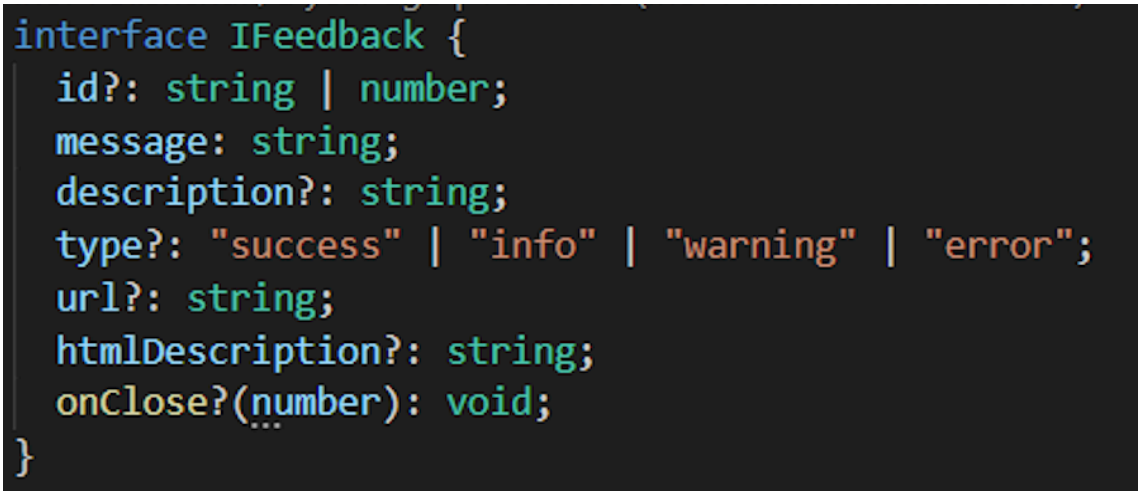


Figura 5.24 – Definição da interface IFeedback.

Na Figura 5.25 é exemplificado o uso do componente <FeedbackToast />, despoletado a partir de uma determinada ação.

```
<Button
  onClick={() =>
    FeedbackToast({
      message: "Success Message",
      description: "Success Description",
      type: "success"
    })
  }
>
  Press for success notification
</Button>
```

Figura 5.25 – Exemplo de integração do componente *FeedbackToast* (Tipo: *success*).

5.2.3.1.3. PillLabel

Esta funcionalidade é transversal a várias secções da plataforma, em que surgiu a necessidade de criar um componente que apresentasse *icons* (com ou sem texto) num formato "especial". Como ilustrado na Figura 5.26, este componente deveria disponibilizar dois modos de visualização (com e sem texto).

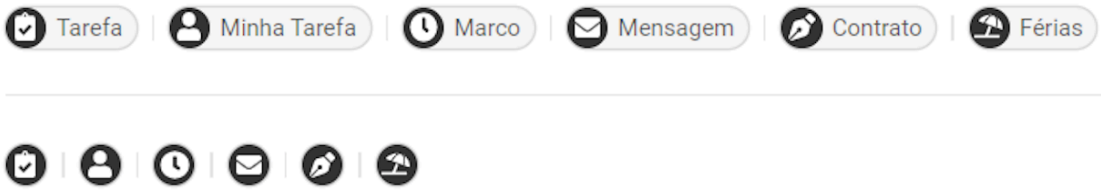


Figura 5.26 – *Mockup* dos dois modos de visualização do componente *Pill Label.*

O estagiário utilizou o componente <FontAwesomeIcon />, pertencente à biblioteca "fontawesome", em conjunto com a etiqueta que permite mostrar o texto correspondente ao *icon* apresentado (Figura 5.27).

```
render() {
  const { type, small } = this.props;
  const pill = this.Pills[type];

  return (
    <span className="pill-label">
      <span className={`pill-label--icon ${pill.class}`}>
        <FontAwesomeIcon icon={pill.icon} />
      </span>

      {!small && <span className="pill-label--text">{pill.text}</span>}
    </span>
  );
}
```

Figura 5.27 – Renderização da interface gráfica do componente *PillLabel*.

Como ilustrado na Figura 5.28, o tipo de PillLabel é restrito (*task*, *myTask*, *milestone*, *message*, *contract*, *holiday* e *amendment*) e existiu a necessidade de criar um *array* de objetos do tipo PillLabelProps (Figura 5.29).

```
interface PillLabelProps {
  text: string;
  icon: IconDefinition;
  class: string;
}
```

Figura 5.28 – Definição da interface PillLabelProps.

```
Pills: PillLabelProps[] = [
  { text: t("task"), icon: faClipboardCheck, class: "pill-label--Task" },
  { text: t("myTask"), icon: faUser, class: "pill-label--UserTask" },
  { text: t("milestone"), icon: faClock, class: "pill-label--Milestone" },
  { text: t("message"), icon: faEnvelope, class: "pill-label--Message" },
  { text: t("contract"), icon: faPenNib, class: "pill-label--Contract" },
  {
    text: t("holiday"),
    icon: faUmbrellaBeach,
    class: "pill-label--Holiday"
  },
  {
    text: t("amendment"),
    icon: faExclamationCircle,
    class: "pill-label--Amendment"
  }
];
```

Figura 5.29 – *Array* de objetos do tipo PillLabelProps.

5.2.3.1.4. ErrorBoundary

Esta funcionalidade é transversal a várias secções da plataforma, dessa forma existiu a necessidade de criar um componente visual que seja despoletado no ecrã do utilizador sempre que exista um erro impactante na plataforma. O *mockup* encontra-se ilustrado na Figura 5.30. (Afonso, 2019a)

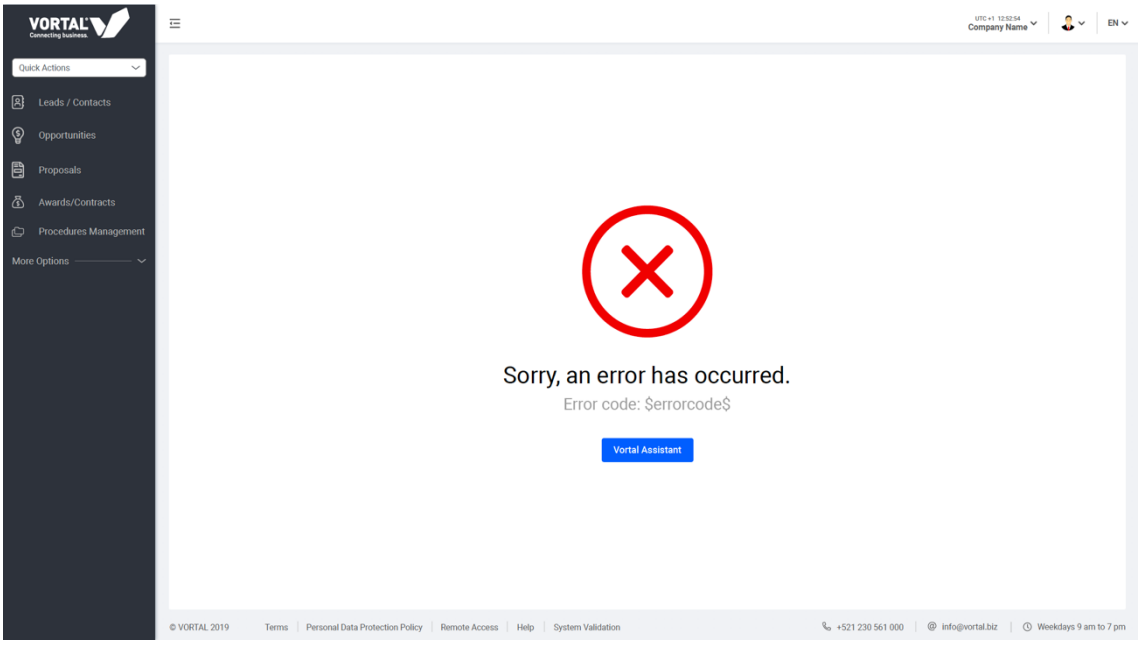


Figura 5.30 – *Mockup* da página de erro.

A Figura 5.31 ilustra que para este caso foi necessário criar um componente <ErrorBoundary /> que retorna um componente visual, assim que se depare com um erro na plataforma. Os erros são capturados através do método “componentDidCatch”.

```tsx
import * as React from "react";

José Carlos Ferreira da Silva, a year ago | 2 authors (pedroferreira and others)
interface Props {
  errorComponent: JSX.Element;
  changeErrorDescription(error, errorInfo): void;
}

José Carlos Ferreira da Silva, a year ago | 2 authors (pedroferreira and others)
interface State {
  hasError: boolean;
  error: any;
  info: any;
}

José Carlos Ferreira da Silva, a year ago | 2 authors (José Carlos Ferreira da Silva and others)
export default class ErrorBoundary extends React.Component<Props, State> {
  constructor(props) {
    super(props);
    this.state = { hasError: false, error: null, info: null };
  }

  componentDidCatch(error, info) {
    console.error(error);

    this.setState({
      hasError: true,
      error: error,
      info: info
    });
  }

  render() {
    const { hasError, error, info } = this.state;

    if (hasError) {
      this.props.changeErrorDescription(error.toString(), info.componentStack);

      return this.props.errorComponent;
    }

    return this.props.children;
  }
}
```

Figura 5.31 – Componente *ErrorBoundary*.

O componente visual retornado sempre que exista erro é o <VortalError />, que faz uso do componente <Result /> proveniente do Ant Design (Figura 5.32).

```tsx
import React from "react";
import { Result, Button } from "antd";
import { FontAwesomeIcon } from "@fortawesome/react-fontawesome";
import { faTimesCircle } from "@fortawesome/pro-light-svg-icons";
import "./styles.scss";

José Carlos Ferreira da Silva, a year ago | 1 author (José Carlos Ferreira da Silva)
interface IProps {
  pageTitle: string;
  pageSubtitle?: string;
  buttonTitle: string;
  buttonAction(): void;
}

You, a few seconds ago | 2 authors (José Carlos Ferreira da Silva and others)
class VortalError extends React.Component<IProps> {
  render() {
    const { pageTitle, pageSubtitle, buttonTitle, buttonAction } = this.props;

    return (
      <Result
        status="error"
        icon={
          <FontAwesomeIcon icon={faTimesCircle} className="vortalErrorIcon" />
        }
        title={pageTitle}
        subTitle={pageSubtitle}
        extra={[
          <Button type="primary" key="console" onClick={buttonAction}>
            {buttonTitle}
          </Button>
        ]}
      />
    );
  }
}

export default VortalError;
```

Figura 5.32 – Componente *VortalError*.

Para fazer uso destes dois componentes apenas é necessário garantir que o conteúdo que se pretende “validar” (*children*) é introduzido no componente e o componente de erro

também é passado ao componente <ErrorBoundary /> (Figura 5.33).

```
<ErrorBoundary
  errorComponent={
    <VortalError
      pageTitle={t("errorTitle")}
      buttonTitle={t("errorButtonTitle")}
      buttonAction={errorButtonAction}
    />
  }
  changeErrorDescription={(error, info) =>
    this.changeErrorDescription(error, info)
  }
>
  {children ? children : null}
</ErrorBoundary>
```

Figura 5.33 – Exemplo de Integração dos componentes *ErrorBoundary* e *VortalError*.

5.2.3.2. Fase de Certificação

Durante esta fase o estagiário resolveu alguns problemas (*bugs*) originados na Fase de Desenvolvimento da *Release 1.3* (Tabela 5.4).

Tabela 5.4 – Descrição dos bugs resolvidos durante a Fase de Certificação da *Release 1.3*.

Descrição do Problema	Módulo
O sistema não apresenta qualquer mensagem quando não existem dados.	*Teams Manager*
Não existe mensagem de sucesso quando a equipa é criada.	*Teams Manager*
A cor dos *steps* não é a correta.	*Teams Manager*
A mensagem de alerta é mostrada em posição incorreta.	*Transversal*
Na página de erro, o código de erro não aparece.	*Transversal*
A página de criação de empresas não é traduzida para português.	*Company Registration*

5.2.4. Release 2

Para esta entrega, foi planeado entregar os módulos mencionados na tabela seguinte (Tabela 5.5).

Tabela 5.5 – Descrição dos módulos entregues na *release 2*.

Módulo	Descrição
Buyer Dossier	Módulo responsável pela gestão de uma determinada proposta. Este módulo possui diversas funcionalidades, incluindo alguns dos módulos descritos abaixo (*Tender and Opportunity Owner*, *Timeline, Receipts WIdget*, *Events WIdget*, *Tasks WIdget*, *Ammendment and Invitation WIdget*, *Messages WIdget* e

	Vergabedokumentation WIdget).
Tender and Opportunity Owner	Módulo responsável pela atribuição de um proprietário a uma proposta/oportunidade.
Timeline	Módulo responsável pela gestão do cronograma disponível no dossier da proposta/oportunidade.
Receipts Widget	Módulo responsável pela gestão de recibos.
Events Widget	Módulo responsável pela gestão de eventos.
Tasks Widget	Módulo responsável pela gestão de tarefas.
Amendment and Invitation Widget	Módulo responsável pela gestão de aditamentos e convites.
Messages Widget	Módulo responsável pela gestão de mensagens.
Create and Publish Procedure	Módulo responsável pela criação e publicação de procedimentos.
Create Amendment	Módulo responsável pela criação de aditamentos.

5.2.4.1. Fase de Desenvolvimento

Na Fase de Desenvolvimento da *Release 2*, e visto que o estagiário tinha consolidado o conhecimento sobre o projeto e o modo de trabalho da equipa, foi pedido ao mesmo que implementasse não só componentes genéricos como componentes específicos do projeto (componentes únicos que foram utilizados de forma isolada).

Os componentes genéricos impostos ao estagiário foram o *ApprovalWorkflow*, o *ErrorValidator* e o *IntegrationMessage*. Tal como aconteceu em *releases* passadas, os resultados destes componentes serão apresentados e explicados na seção “6. Resultados”.

5.2.4.1.1. ApprovalWorkflow

Surgiu a intenção de implementar o fluxo de aprovação da criação de um procedimento em diferentes secções da plataforma, no intuito de criar um componente que cumprisse os seguintes requisitos:

- Assim que o *workflow* for acionado, aparecerá com os botões “Aceitar” e “Rejeitar”

disponíveis para os utilizadores que fazem parte do *workflow* (Figura 5.34);

- Quando o *workflow* for aceite, deve aparecer uma mensagem informativa a verde (Figura 5.35);
- Quando o *workflow* for rejeitado, deve aparecer uma mensagem informativa a vermelho;
- Deve estar disponível uma área de comentários (Figura 5.36) e de documentos (Figura 5.37) associados ao *workflow*;
- O utilizador deve ter disponível uma área de pesquisa, de forma a obter *steps* ou *tasks*;
- Deve estar disponível a possibilidade de reassinar (Figura 5.38), aprovar ou rejeitar cada tarefa.

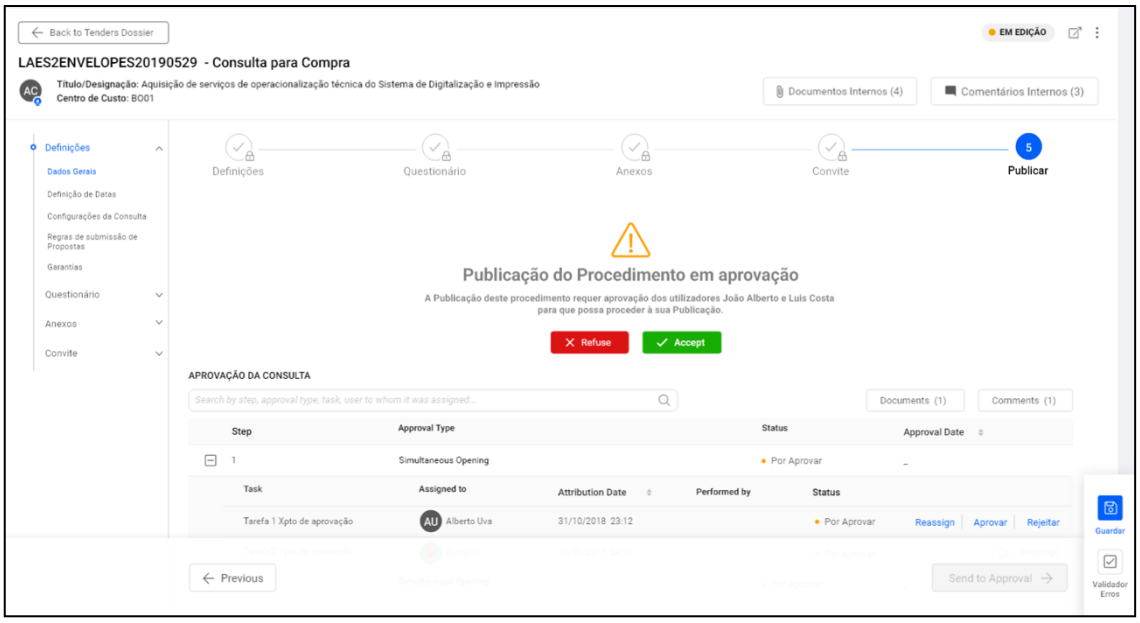


Figura 5.34 – *Mockup* do *Approval Workflow*.

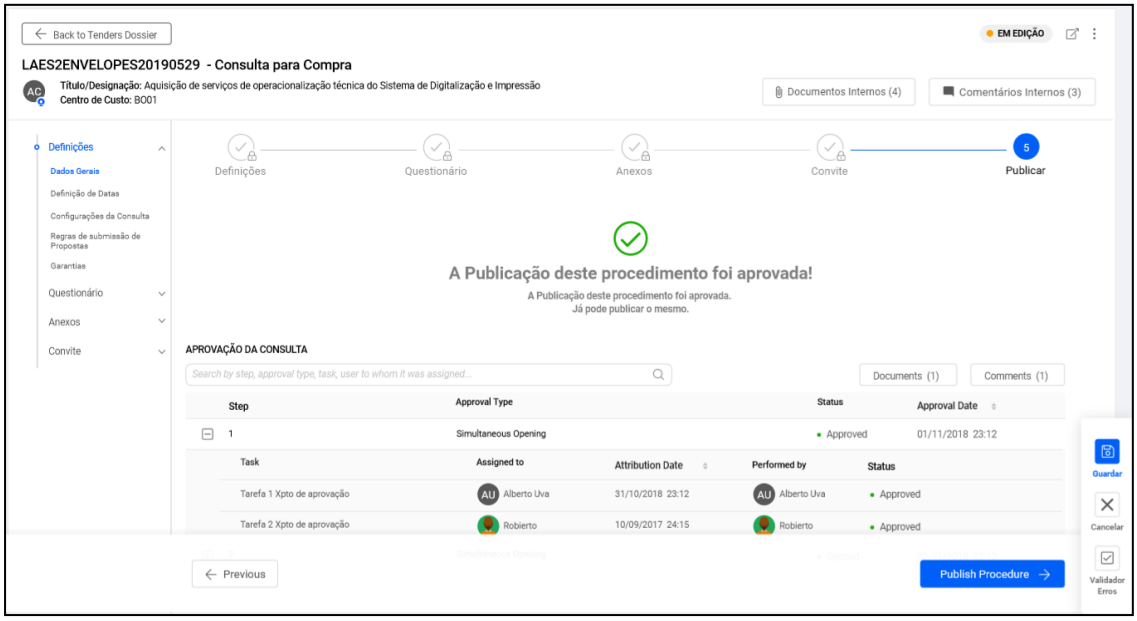


Figura 5.35 – *Mockup* do *Approval Workflow* (após aprovação).

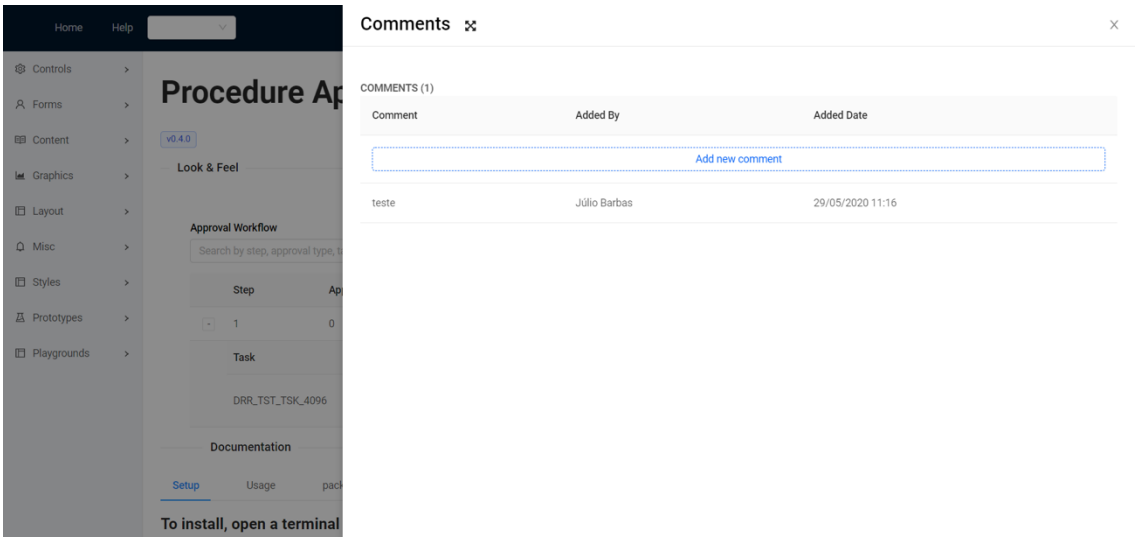


Figura 5.36 – *Mockup* da área de comentários do *Approval Workflow*.

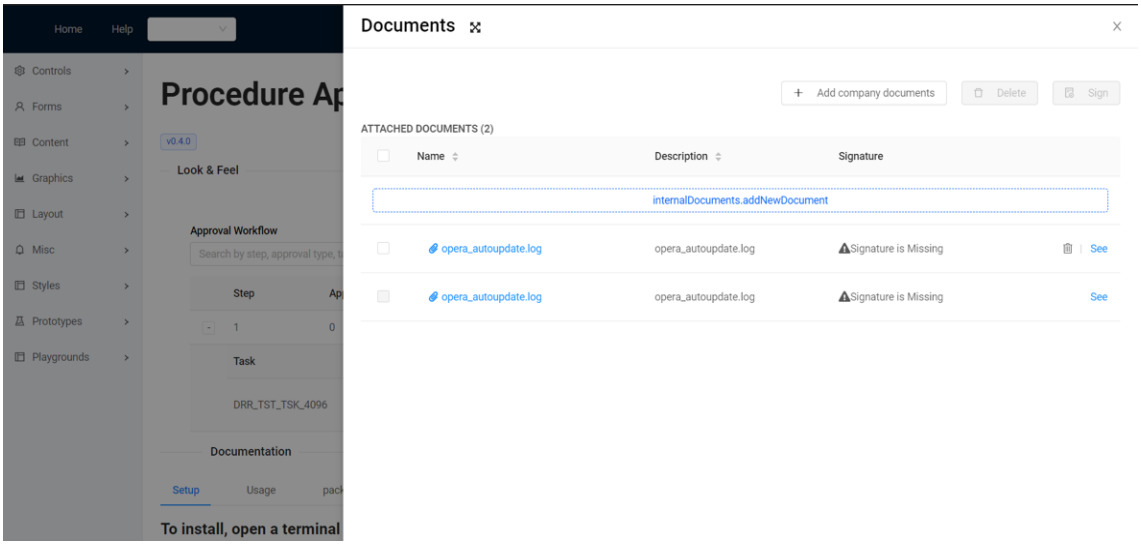


Figura 5.37 – *Mockup* da área de documentos do *Approval Workflow*.

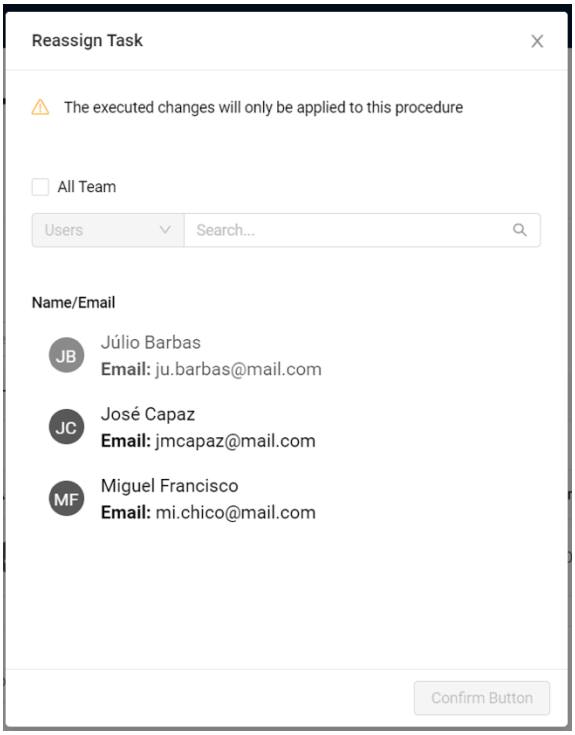


Figura 5.38 – *Mockup* do reassignar utilizadores no *Approval Workflow*.

Relativamente à implementação deste componente, visto ser um componente grande e complexo, o estagiário decidiu dividir o componente em dois subcomponentes (*ApprovalArea* e *ApprovalTable*).

Como ilustra a Figura 5.39, o primeiro subcomponente criado, designado por *ApprovalArea*, corresponde à secção que disponibiliza os botões “Aceitar” e “Rejeitar”. Caso o utilizador não faça parte do *workflow*, será mostrada uma mensagem de que o *workflow* se

encontra em aprovação. Esta área permite também alertar o utilizador que o fluxo já se encontra aprovado ou que foi rejeitado. Como mostra a Figura 5.40, para este subcomponente foram criados dois microcomponentes, *UserCanApprove* e o *UserCanNotApprove*, que como o nome indica será renderizado um dos dois componentes dependendo das permissões do utilizador sobre o *workflow*.

Figura 5.39 – *Mockup* do componente *Approval Area* (estado: para aprovação).

```
render() {
  if (this.props.checkIfUserCanApprove()) {
    return (
      <Spin spinning={this.props.approvalAreaLoading}>
        <UserCanApprove
          refuseProcedure={() => this.props.refuseProcedure()}
          acceptProcedure={() => this.props.acceptProcedure()}
          approvalState={this.props.approvalState}
          approvalMoment={this.props.approvalMoment}
        />
      </Spin>
    );
  } else {
    return (
      <Spin spinning={this.props.approvalAreaLoading}>
        <UserCanNotApprove
          approvalMoment={this.props.approvalMoment}
        />
      </Spin>
    );
  }
}
```

Figura 5.40 – Renderização da interface gráfica do componente *Approval Area*.

O segundo subcomponente criado, designado por *ApprovalTable*, corresponde à secção que disponibiliza as *tasks* associadas a cada *step*, com as respetivas ações (Figura 5.41). Neste componente também deverá ser feita a gestão de comentários e documentos.

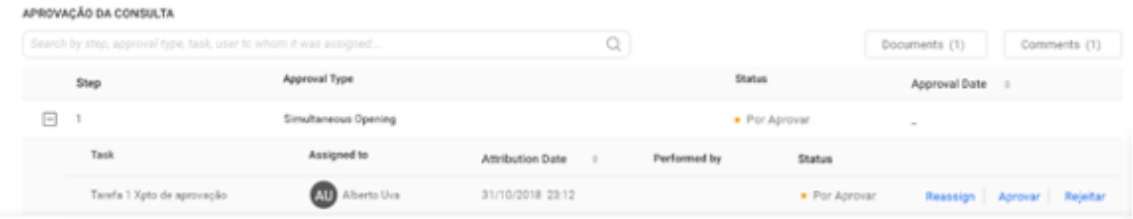


Figura 5.41 – *Mockup* do componente *Approval Table.*

Para disponibilizar as *tasks* associadas aos *steps* foi utilizado o componente <Grid />, componente criado pela equipa da Vortal, que permite organizar o conteúdo da página por *rows*, sendo que cada *row* contém um conjunto de colunas (Figura 5.42).

```
<Grid
  mode="form"
  rows={[
    {
      key: "row1",
      columns: [···
      ]
    },
    {
      key: "row2",
      columns: [···
      ]
    }
  ]}
/>
```

Figura 5.42 – Renderização da interface gráfica do componente *Approval Table* (disponibilização das *tasks*).

Relativamente às áreas de comentários e documentos, foram criados dois microcomponentes, <CommentsDrawer /> e <DocumentsDrawer />, que utilizam como base o componente <Drawer /> da biblioteca do Ant Design (Figura 5.43).

```
<DocumentsDrawer
  visibility={documentsDrawerVisibility}
  loading={documentsLoading}
  closeDocumentsDrawer={() => closeDocumentsDrawer()}
  changeDocumentsList={documents => changeDocumentsList(documents)}
  documents={documents}
  classifications={classifications}
  handleUpload={(file, id) => handleUpload(file, id)}
  deleteDocument={(id, rows) => deleteDocument(id, rows)}
  saveDocument={(ids, isFromCompanyDocuments: boolean) =>
    saveDocument(ids, isFromCompanyDocuments)
  }
  companyDocuments={companyDocuments}
  seeDocument={id => seeDocument(id)}
  getDocumentDetails={getDocumentDetails}
  onUploaded={(result: any, id: number) => onUploaded(result, id)}
  isAddButtonDisabled={isAddButtonDisabled}
  isDeleteButtonDisabled={this.props.isDeleteDocButtonDisabled}
  getSignatureModal={this.props.getSignatureModal}
/>
<CommentsDrawer
  visibility={commentsDrawerVisibility}
  closeCommentsDrawer={() => closeCommentsDrawer()}
  comments={comments}
  changeCommentToAdd={comment => changeCommentToAdd(comment)}
  changeCommentsList={comments => changeCommentsList(comments)}
  commentToAdd={commentToAdd}
  isAddCommentDisabled={this.props.isAddCommentDisabled}
/>
```

Figura 5.43 – Renderização da interface gráfica do componente *Approval Table* (comentários e documentos).

Por fim, para a implementação do *modal* de reatribuição de utilizadores foi utilizado o componente <VortalModal />, componente criado pela equipa da Vortal, que utiliza como base o componente <Modal /> da biblioteca Ant Design. Este faz também uso do componente <Form

/>, da biblioteca “react-final-form”, de forma a renderizar o formulário de seleção de utilizadores a reassinar (Figura 5.44).

```
<VortalModal
  header={t("approvalWorkflow.approvalTable.reassignModal.title")}
  visible={reassignModalVisibility}
  onCancel={() => this.onCloseModal()}
  footer={
    <Button
      type="primary"
      onClick={() => reassignTask()}
      disabled={entitySelected === ""}
    >
      {t("approvalWorkflow.approvalTable.reassignModal.confirmButton")}
    </Button>
  }
  content={
    <Form
      onSubmit={this.onSubmit} ···
      )}
    />
  }
/>
```

Figura 5.44 – Renderização da interface gráfica do componente *Approval Table* (reatribuição).

Para integrar este componente na aplicação, criou-se um componente de forma a agrupar os dois subcomponentes em apenas um (Figura 5.45 e Figura 5.46).

```
<ApprovalArea
  checkIfUserCanApprove={() => {
    if (this.state.workflowState !== WorkflowStateEnum.InProcess) {
      return true;
    } else if (this.state.approvalData !== null) {
      return (
        this.state.approvalData[0].activeUserTasksForUser.length > 0
      );
    } else {
      return false;
    }
  }}
  refuseProcedure={async () =>
    this.refuseAll(await this.getTasksToApproveOrReject())
  }
  acceptProcedure={async () =>
    this.acceptAll(await this.getTasksToApproveOrReject())
  }
  approvalState={this.state.workflowState}
  approvalMoment={this.props.approvalMoment}
  approvalAreaLoading={this.state.approvalAreaLoading}
/>
```

Figura 5.45 – Exemplo de Integração do subcomponente *ApprovalArea*.

```
<ApprovalTable
  approvalData={
    this.state.approvalData !== null ? this.state.approvalData : []
  }
  approvalTableLoading={this.state.approvalTableLoading}
  approvalColumns={this.approvalColumns}
  subApprovalColumns={this.subApprovalColumns}
  documentsAction={this.documentsAction}
  commentsAction={this.commentsAction}
  searchWorkflow={this.searchWorkflow}
  changeReassignModalVisibility={this.changeReassignModalVisibility}
  reassignTask={this.reassignTask}
  reassignModalVisibility={this.state.reassignModalVisibility}
  entityData={
    this.state.entityFilteredData?.length > 0 &&
    this.state.entitySearched != ""
      ? this.state.entityFilteredData
      : this.state.entityData
  }
  entitySelected={this.state.entitySelected}
  selectEntity={this.selectEntity}
  searchEntity={this.searchEntity}
  hideDocumentsAndComments={false}
  documentsDrawerVisibility={this.state.documentsDrawerVisibility}
  closeDocumentsDrawer={this.closeDocumentsDrawer}
  documents={this.state.documents}
  documentsLoading={this.state.documentsLoading}
  classifications={this.state.classifications}
  changeDocumentsList={this.changeDocumentsList}
  handleUpload={this.handleUpload}
  deleteDocument={this.deleteDocument}
  saveDocument={this.saveDocument}
  companyDocuments={this.state.companyDocuments}
  seeDocument={this.getDocumentDetail}
  commentsDrawerVisibility={this.state.commentsDrawerVisibility}
  closeComme this: this his.closeCommentsDrawer}
  comments={this.state.comments}
  changeCommentsList={this.changeCommentsList}
  isAddButtonDisabled={
    this.state.approvalData &&
    this.state.approvalData[0].activeUserTasksForUser.length === 0
  }
  isDeleteDocButtonDisabled={
    this.state.approvalData &&
    this.state.approvalData[0].activeUserTasksForUser.length === 0      José Carlos Ferreira da Silva, 6 months ago • Fixed reassign to pending start tasks (appr
```

Figura 5.46 – Exemplo de Integração do subcomponente *ApprovalTable*.

5.2.4.1.2. ErrorValidator

De maneira a validar os erros na criação de um procedimento, surgiu a necessidade de criar um componente que cumprisse os seguintes requisitos:

- O utilizador poderá validar os erros durante a criação de um procedimento clicando no botão “Validador de Erros” disponível do lado direito do ecrã;
- Todas as validações serão feitas assim que o utilizador aceder a nova etapa “Publicar” clicando na etapa ou clicando no botão “Avançar” de forma a ser redirecionado para a etapa “Publicar”;
- Ao aceder à etapa publicar se existir algum erro aparecerá o validador de erros e o utilizador não poderá publicar ou enviar o procedimento para aprovação sem corrigir os erros;
- Se não existirem erros e o utilizador clicar no botão, aparecerá uma mensagem em verde “Nenhum erro encontrado!”;
- As etapas que apresentarem erros serão apresentadas em vermelho e com um *icon*;
- As etapas que não tiverem erros serão mostradas em verde com um *icon*;
- Cada erro mostrado na ferramenta “Validador de Erros” funcionará como uma âncora,

portanto, ao clicar no erro, o utilizador deve ser redirecionado para o respetivo campo.

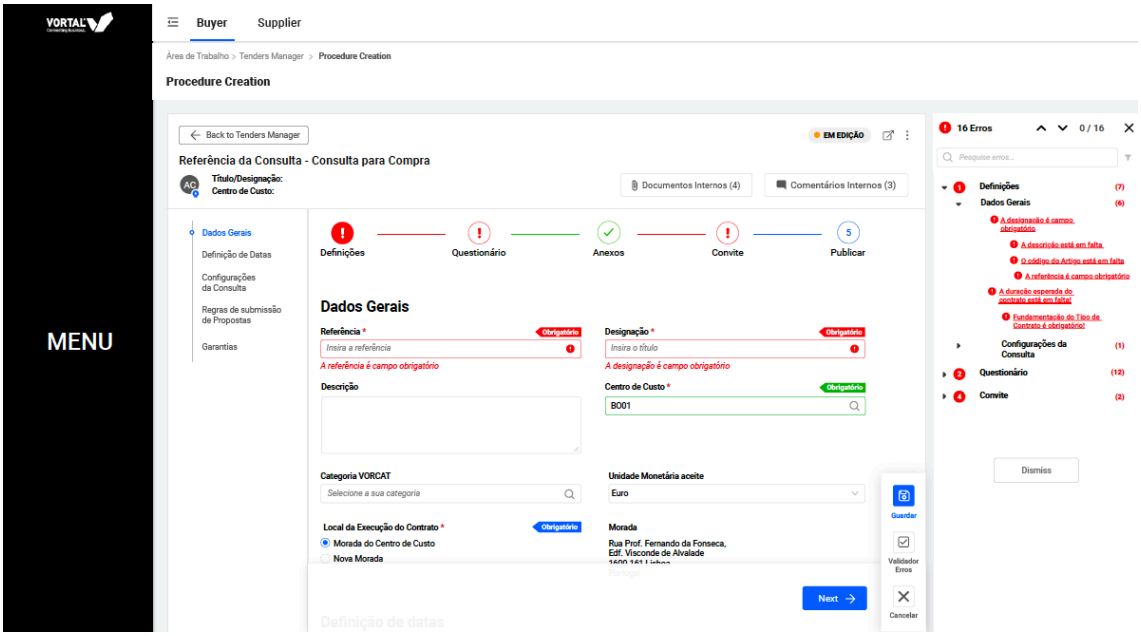


Figura 5.47 – *Mockup* do Validador de Erros (sem redireccionamento).

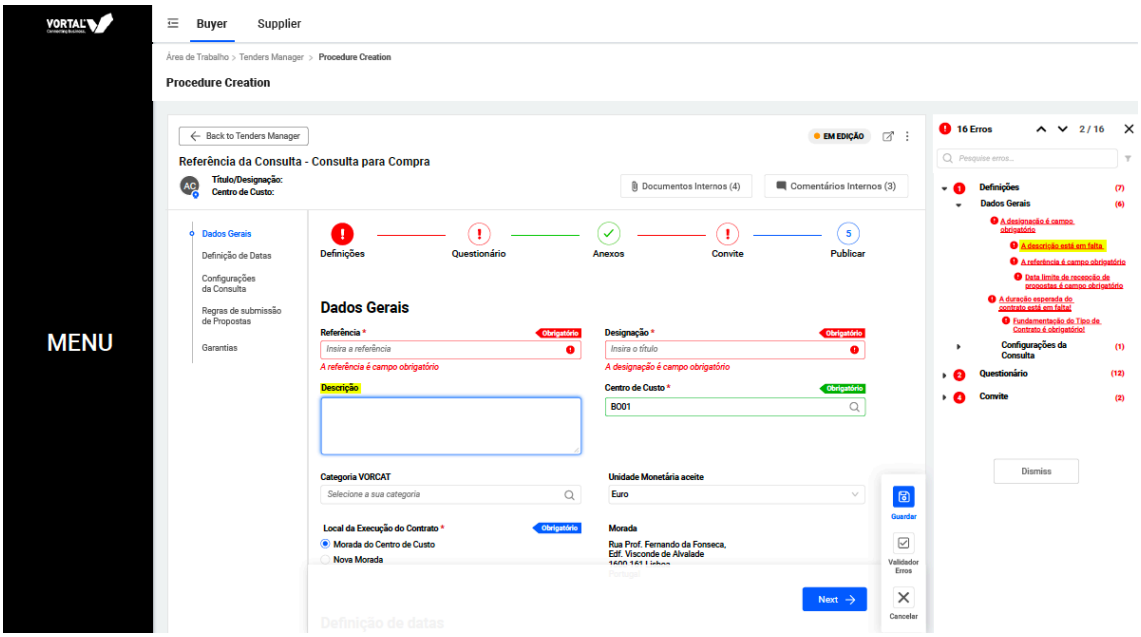


Figura 5.48 – *Mockup* do Validador de Erros (com redireccionamento).

As Figuras 5.47 e 5.48 ilustram que no validador de erros o utilizador terá as seguintes informações/ações:

- Número total de erros;

- Etapas em que os erros ocorreram (o utilizador pode estender ou encolher cada etapa);
- Número de erros em cada etapa;
- *Icon* de cruz: fecha o validador;
- *Icon* "*Dismiss*": fecha o validador;
- Setas de navegação: permite navegar entre os erros;
- Caixa de texto: permite pesquisar por um determinado erro.

Na Figura 5.49 é exemplificado o detalhe no Validador de Erros.

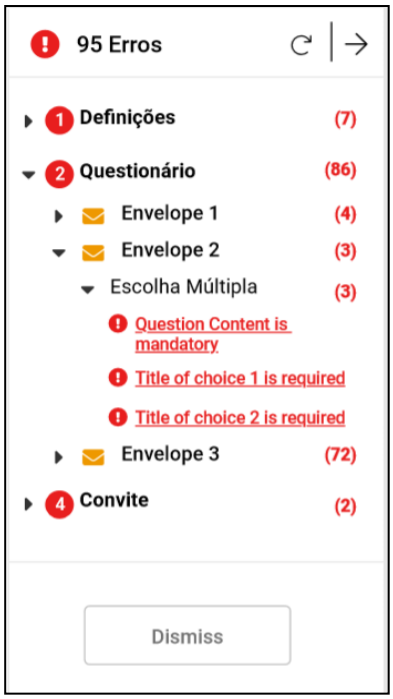


Figura 5.49 – *Mockup* de Detalhe do Validador de Erros.

Como ilustrado na Figura 5.50, o validador foi construído com base na biblioteca Ant Design, o componente <PageHeader /> para construir o cabeçalho do validador, utilizou o componente <Tree /> para a apresentação dos erros e utilizou ainda o componente <Button /> para o botão "*Dismiss*".

```
<div className="errorValidator">
  <PageHeader
    onBack={() => null}
    backIcon={<FontAwesomeIcon icon={faExclamationCircle} color="red" />}
    title={this.countTotalNumberOfErrors(this.props.data)}
    extra={
      <Button
        type="link"
        className="closeButton"
        onClick={() => this.props.onClose()}
      >
        <FontAwesomeIcon icon={faTimes} color="black" />
      </Button>
    }
    className="headerSection"
  />

  <Divider />

  <Tree
    showIcon
    selectable={false}
    onExpand={this.onExpand}
    expandedKeys={expandedKeys}
    autoExpandParent={autoExpandParent}
    className="treeSection"
  >
    {this.renderStep(this.props.data)}
  </Tree>

  <Divider />

  <Button
    hidden
    className="dismissSection"
    onClick={() => this.props.onDismiss()}
  >
    {t("errorValidator.dismissButton")}
  </Button>
</div>
```

Figura 5.50 – Renderização da interface gráfica do componente ErrorValidator.

Como é possível observar na Figura 5.51, de forma a facilitar o uso do componente foi definida uma tipagem própria para o objeto *data* (IStepValidation). O objeto *data* é composto pelos erros a serem renderizados no ecrã e contém o identificador do *step* e um *array* de *regions*. Este *array* é do tipo IRegionValidation, composto por objetos detentores de um identificador da region e por um *array* de erros do tipo IFieldStatusValidation. O último *array* mencionado é detentor de toda a informação associada a um erro.

```
interface IProps {
  data: IStepValidation[];
  onDismiss(): void;
  onClose(): void;
  goToParent(parentIdentifier: string, anchorUrl: string): void;
  getStepTranslation(stepIdentifier: string): string;
  getFieldStepTranslation(fieldSetIdentifier: string): string;
}

Bruno Vicente Machado, 7 months ago | 1 author (Bruno Vicente Machado)
export interface IStepValidation {
  stepIdentifier: string;
  regions: IRegionValidation[];
}

Bruno Vicente Machado, 7 months ago | 1 author (Bruno Vicente Machado)
export interface IRegionValidation {
  regionIdentifier: string;
  errors: IFieldStatusValidation[];
}

export enum EnumErrorType {
  Error = 1,
  Warning = 2
}

Bruno Vicente Machado, 7 months ago | 2 authors (Bruno Vicente Machado and others)
export interface IFieldStatusValidation {
  key: string;
  errorMessage?: string;
  isBackendError: boolean;
  errorType: EnumErrorType;
}

José Carlos Ferreira da Silva, 10 months ago | 1 author (José Carlos Ferreira da Silva)
interface IState {
  expandedKeys: any[];
  searchValue: string;
  autoExpandParent: boolean;
}
```

Figura 5.51 – Definição da interface do objeto *data*.

Para utilizar este componente é necessário que lhe seja passado um objeto com a informação de cada erro e todas as funções associadas a ações requeridas (Figura 5.52).

```
<ErrorValidator
  data={this.state.validationData}
  onDismiss={() => this.onDismiss()}
  onClose={() => this.onChangeSiderVisibility(false)}
  goToParent={this.changeCurrentStep}
  getStepTranslation={this.getStepTranslation}
  getFieldStepTranslation={this.getFieldStepTranslation}
/>
```

Figura 5.52 – Exemplo de Integração do componente *ErrorValidator*.

5.2.4.1.3. IntegrationMessage

Esta funcionalidade é transversal a várias secções da aplicação, surgindo assim a necessidade de criar um componente que disponibilizasse diferentes tipos de mensagens de integração da plataforma com os serviços externos. Esse componente devia incluir títulos, subtítulos, *icons* e botões (opcional). Os *mockups* disponibilizados encontram-se ilustrados na Figura 5.53, 5.54, 5.55 e 5.56.

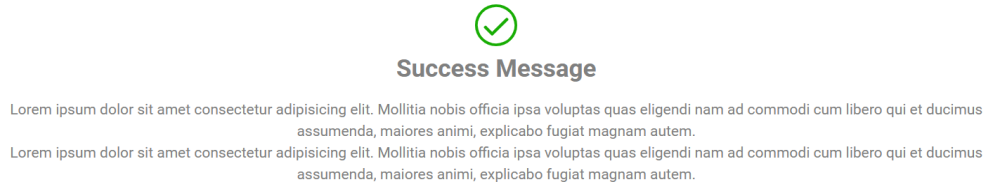


Figura 5.53 – *Mockup* da mensagem de sucesso.

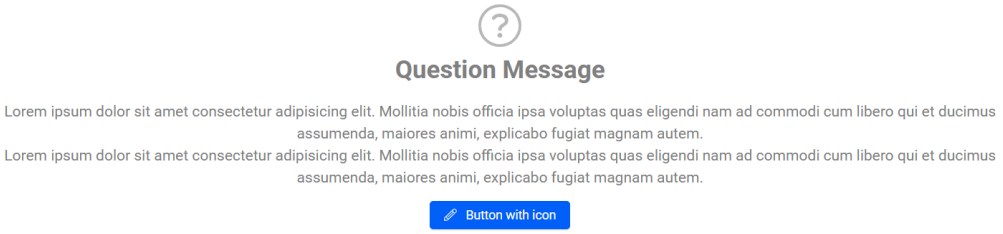


Figura 5.54 – *Mockup* da mensagem de questão.

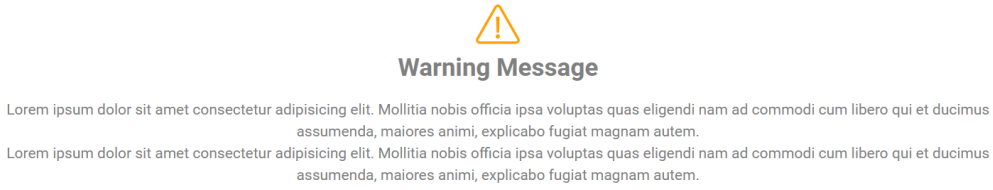


Figura 5.55 – *Mockup* da mensagem de aviso.

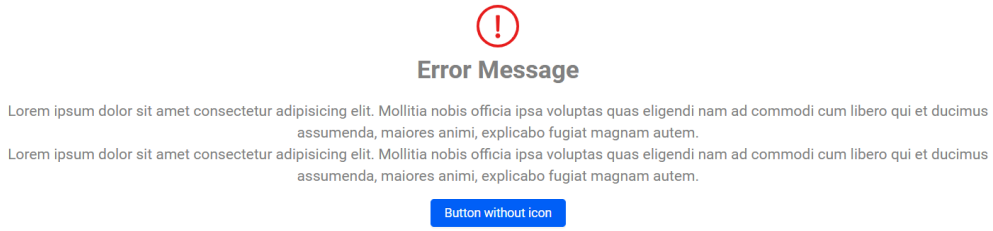


Figura 5.56 – *Mockup* da mensagem de erro.

Como ilustrado na Figura 5.57, o estagiário utilizou o componente <FontAwesomeIcon />, pertencente à biblioteca “fontawesome”, em conjunto com a componente <Button />, pertencente à biblioteca Ant Design.

```
renderSubtitles = (subtitles: string[]) => {
  return (
    subtitles &&
    subtitles.length > 0 &&
    subtitles.map(subtitle => {
      return <p>{subtitle}</p>;
    })
  );
};

renderButton = (button: ButtonProps) => {
  return (
    button && (
      <Button onClick={() => button.onClick()} type={"primary"}>
        {button.icon && (
          <FontAwesomeIcon
            icon={button.icon.logo}
            color={button.icon.color}
          />
        )}
        {button.label}
      </Button>
    )
  );
};

render() {
  const { icon, title, subtitles, button } = this.props;

  return (
    <div className="integrationMessage">
      <FontAwesomeIcon icon={icon.logo} color={icon.color} />
      <h3>{title}</h3>
      {this.renderSubtitles(subtitles)}
      {this.renderButton(button)}
    </div>
  );
}
```

Figura 5.57 – Renderização da interface gráfica do componente *IntegrationMessage*.

Como é possível observar na Figura 5.58, de forma a facilitar o uso do componente foi definida uma tipagem própria para o objeto *icon* (IconDefinition) e para o objeto *button* (ButtonProps).

```
interface IconProps {
  logo: IconDefinition;
  color: string;
}

José Carlos Ferreira da Silva, 8 months ago | 1 author (José Carlos Ferreira da Silva)
interface ButtonProps {
  onClick?(): void;
  icon?: IconProps;
  label: string;
}

José Carlos Ferreira da Silva, 8 months ago | 1 author (José Carlos Ferreira da Silva)
interface IProps {
  icon: IconProps;
  title: string;
  subtitles?: string[];
  button?: ButtonProps;
}
```

Figura 5.58 – Definição das interfaces IconDefinition e ButtonProps.

Na Figura 5.59 é exemplificado o uso do componente <IntegrationMessage />.

```
<IntegrationMessage
  icon={{ logo: faQuestionCircle, color: "#B8B8B8" }}
  title={"Question Message"}
  subtitles={this.subtitles}
  button={{
    onClick: () => this.onClick("Button clicked!"),
    icon: { logo: faPencilAlt, color: "#FFFFFF" },
    label: "Button with icon"
  }}
/>
```

Figura 5.59 – Exemplo de integração do componente *IntegrationMessage.*

Os componentes específicos impostos ao estagiário foram o *Invitation Step* e o *Publish Step*, ambos pertencentes ao módulo "*Create and Publish Procedure*".

5.2.4.1.4. Invitation Step

Durante o processo de criação de um procedimento, surgiu a necessidade de criar um ecrã de convite de empresas para o procedimento em questão. Dessa forma foram criados vários componentes de forma a cumprir os seguintes requisitos: (Costa, 2019)

- O sistema deve permitir ao utilizador pesquisar companhias convidadas (Figura 5.60);
- O sistema deve disponibilizar uma lista de empresas convidadas (Figura 5.60);
- O sistema deve permitir a sugestão automática de empresas a serem convidadas (Figura 5.60);
- O sistema deve permitir a adição de novas categorias (Figura 5.61);
- O sistema deve permitir ao utilizador pesquisar empresas a serem convidadas (Figura 5.62);
- O sistema deve disponibilizar uma lista de empresas a serem convidadas (Figura 5.62);
- O sistema deve permitir ao utilizador convidar empresas adicionando à lista de empresas convidadas (Figura 5.62);
- O sistema deve permitir criar companhias que não se encontram registadas na plataforma (Figura 5.63).

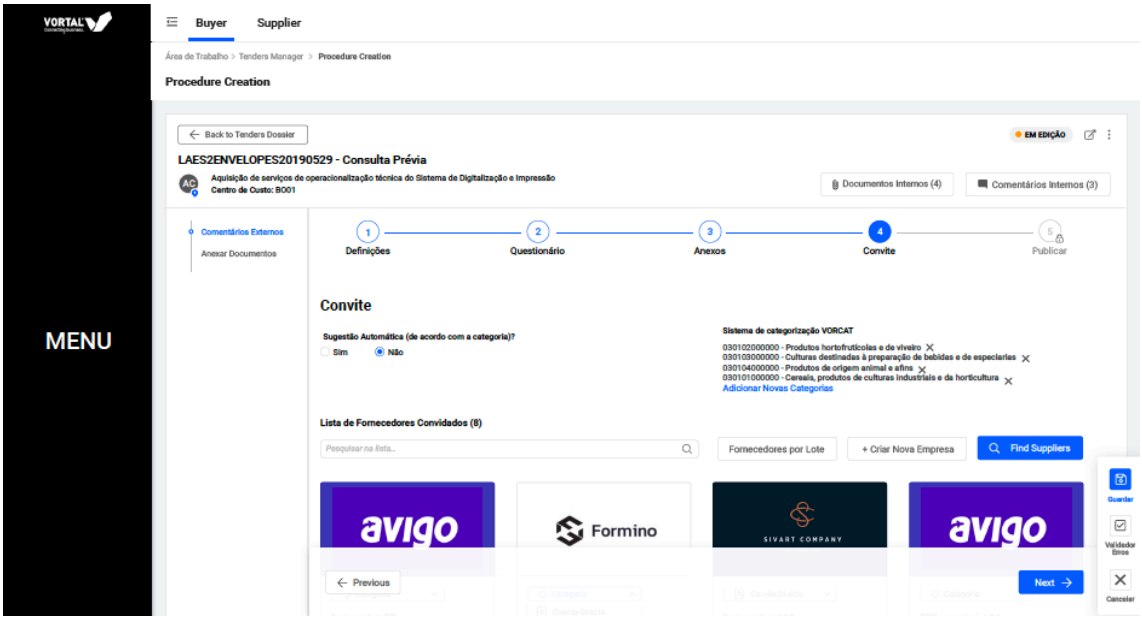


Figura 5.60 – *Mockup* do *Invitation Step.*

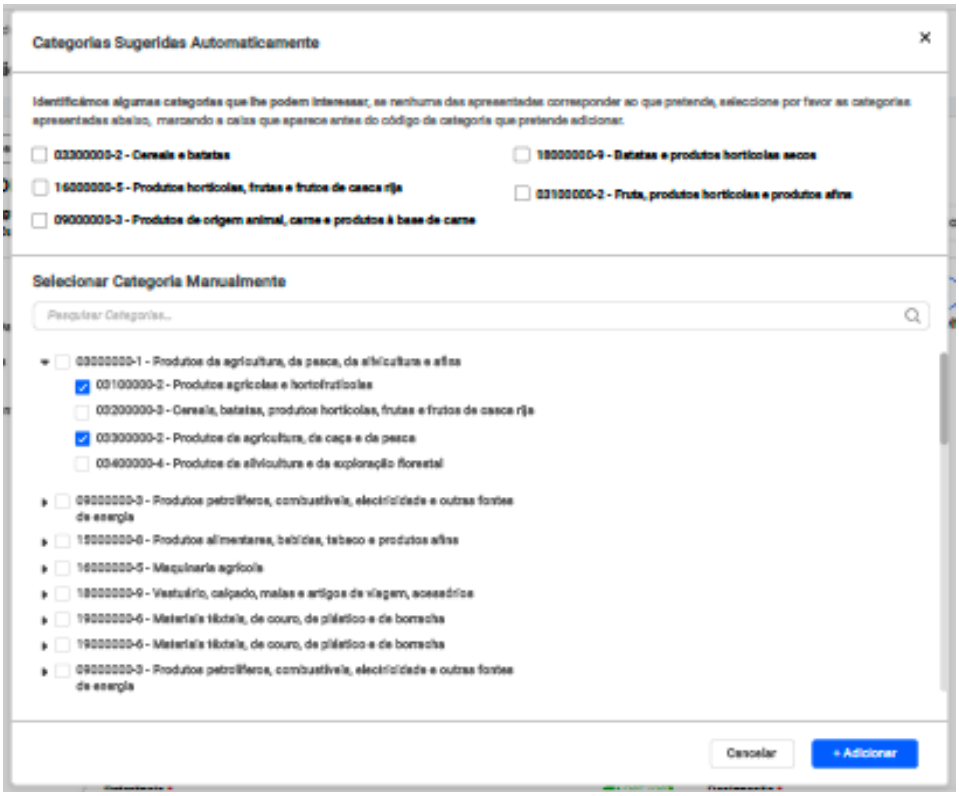


Figura 5.61 – *Mockup* de adição de categorias no *Invite Step*.

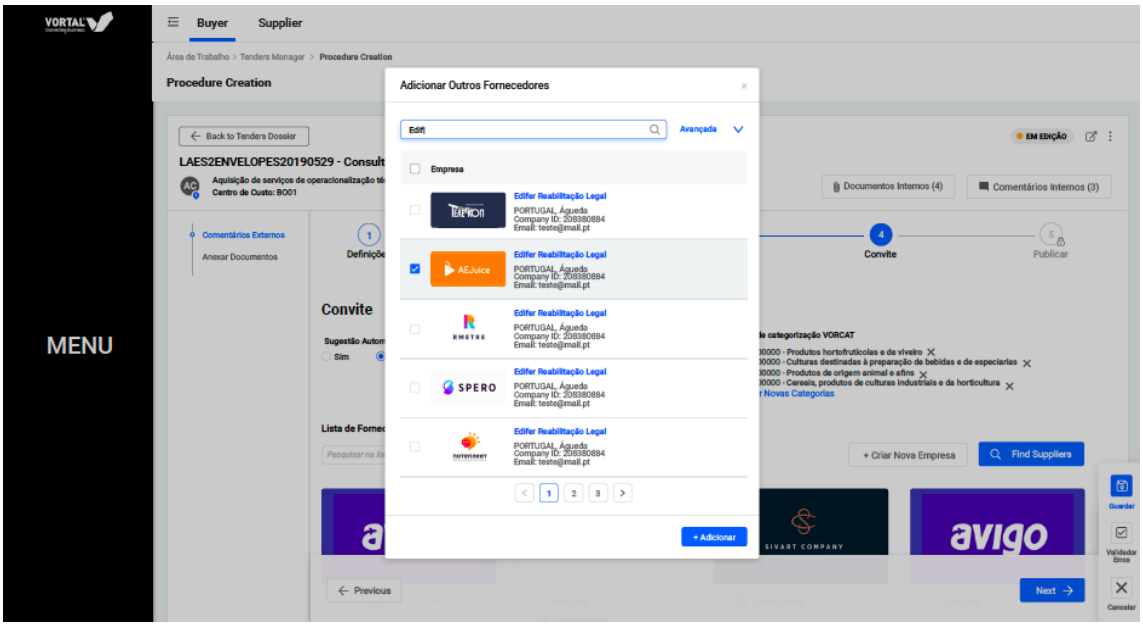


Figura 5.62 – *Mockup* de adição de companhias convidadas no *Invitation Step*.

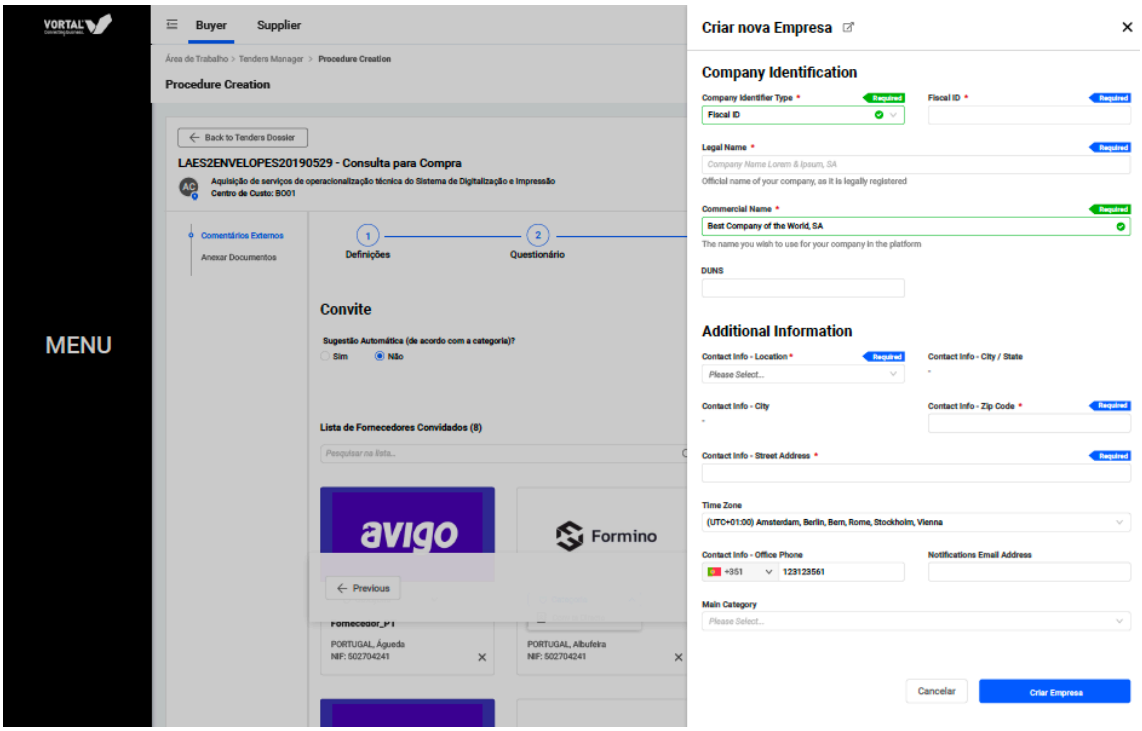


Figura 5.63 – *Mockup* de criação de uma nova empresa no *Invitation Step*.

De forma a disponibilizar as empresas convidadas no ecrã do *Invitation Step* foi novamente utilizado o componente <Grid />, componente criado pela equipa da Vortal, que permite organizar o conteúdo da página por *rows*, sendo que cada *row* contém um conjunto de colunas (Figura 5.64).

Figura 5.64 – Renderização da interface gráfica do componente *Invitation Step* (disponibilização das empresas convidadas).

Relativamente à área de adição de categorias, foi utilizado o componente <VortalModal />, componente criado pela equipa da Vortal, que utiliza como base o componente <Modal /> da biblioteca Ant Design e para o conteúdo da *modal* foi utilizado o componente <CategoryTree />, componente também criado pela equipa da Vortal, que utiliza como base o componente <Tree /> também da biblioteca Ant Design (Figura 5.65).

```
<VortalModal
  header={t("invitation.categoriesModal.modalHeader")}
  visible={this.state.modalCategoriesVisible}
  onOk={this.showModalCategories}
  onCancel={() => this.closeModalCategories(false)}
  width="55%"
  customClass="categoriesModal"
  footer={
    <Button
      key="back"
      onClick={() => this.closeModalCategories(true)}
    >
      {t("invitation.categoriesModal.modalBtn")}
    </Button>
  }
  content={
    <CategoryTree
      multiple={true}
      addSelectedCategory={this.addSelectedCategory}
      categoryData={this.state.categoryData}
      categoryViewMode={CategoryViewModeEnum.ForInterest}
      hasSuggestions={false}
      checkedCategories={this.props.manuallyIntroducedCategories}
    />
  }
  destroyOnClose={true}
/>
```

Figura 5.65 – Renderização da interface gráfica do componente *Invitation Step* (*modal* de adição de categorias).

Para implementação do *modal* de adição de empresas a convidar, foi utilizado o componente <SuppliersSelectionModal />, componente criado pela equipa da Vortal, que utiliza como base o componente <Modal /> da biblioteca Ant Design (Figura 5.66).

```
<SuppliersSelectionModal
  modalTitle={t("invitation.addSuppliersModal.title")}
  visible={visible}
  modalButton={t("invitation.addSuppliersModal.add")}
  columns={columns}
  supplierSearchList={supplierslist}
  searchPages={searchPages}
  onClose={this.closeModal}
  modalButtonAction={addData}
  isHide={isHide}
  isLoading={isLoading}
  selectedSuppliers={data.map(supplier => supplier.companyCode)}
  onChangePage={(pageNumber, pageSize) =>
    this.fetchSupplierslist({ pageNumber, pageSize })
  }
  defaultPageSize={pageSize}
  currentPage={pageNumber}
  totalRecords={totalRecords}
/>
```

Figura 5.66 – Renderização da interface gráfica do componente *Invitation Step* (adição de empresas a convidar).

Por fim, para a implementação da *drawer* de criação de empresas foi utilizado o componente <AddNewCompanyDrawer />, componente criado pela equipa da Vortal, que utiliza como base o componente <Drawer /> da biblioteca Ant Design (Figura 5.67).

```
<AddNewCompanyDrawer
  visible={modalAddNewCompanyVisible}
  onClose={() => {
    this.setState({ modalAddNewCompanyVisible: false });
  }}
/>
```

Figura 5.67 – Renderização da interface gráfica do componente *Invitation Step* (criação de empresa).

5.2.4.1.5. Publish Step

Como último passo da criação de um procedimento, surgiu a necessidade de criar um componente que permitisse realizar a publicação do procedimento criado. Os requisitos impostos foram os seguintes: (Costa, 2019)

- Validar erros automaticamente quando o utilizador prossegue para o *step* “Publicar”;
- Se existirem erros no processo de criação, o sistema deverá abrir automaticamente o Validador de Erros (explicado anteriormente nesta seção), deverá mudar o *icon* de cada *step* com erro para vermelho e o botão “Publicar” ou “Enviar para aprovação” não deve ficar disponível (Figura 5.68);
- Se não existirem erros no processo de criação, o sistema deverá mudar o *icon* de cada *step* para verde e o botão “Publicar” ou “Enviar para aprovação” deverá ficar disponível (Figura 5.69);

- Se não existirem erros mas se existir fluxo de aprovação, o sistema deverá disponibilizar o botão “Enviar para aprovação” que irá acionar o fluxo de aprovação (Figura 5.69);
- Se não existirem erros nem fluxo de aprovação para aprovar, o sistema deverá disponibilizar o botão “Publicar” (Figura 5.70).

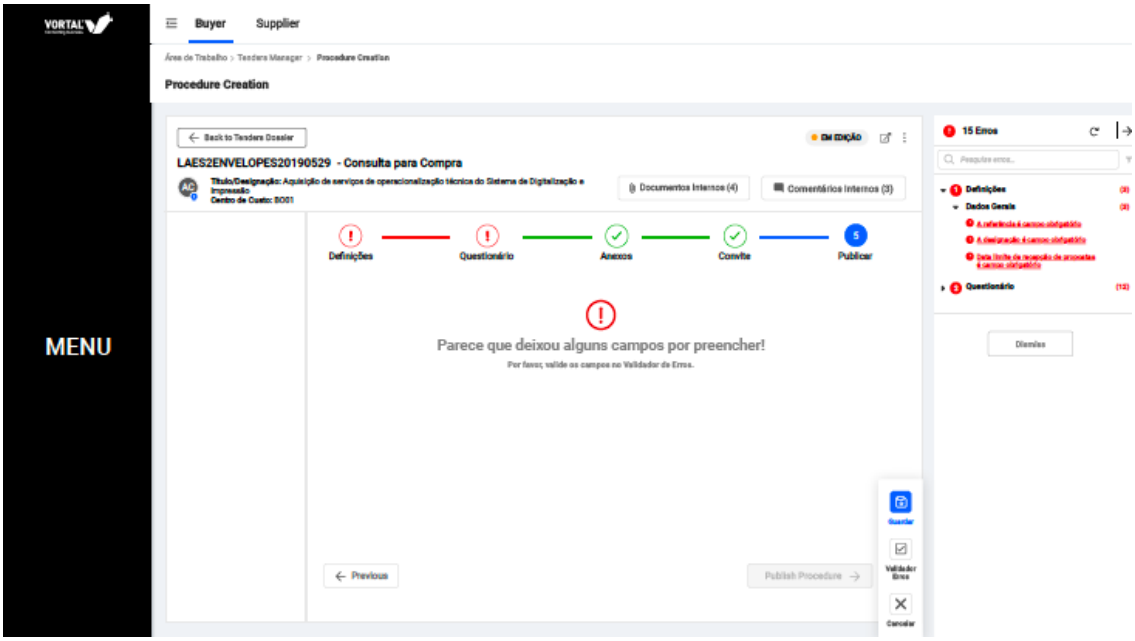


Figura 5.68 – *Mockup* do componente *Publish Step* (erros no processo de criação).

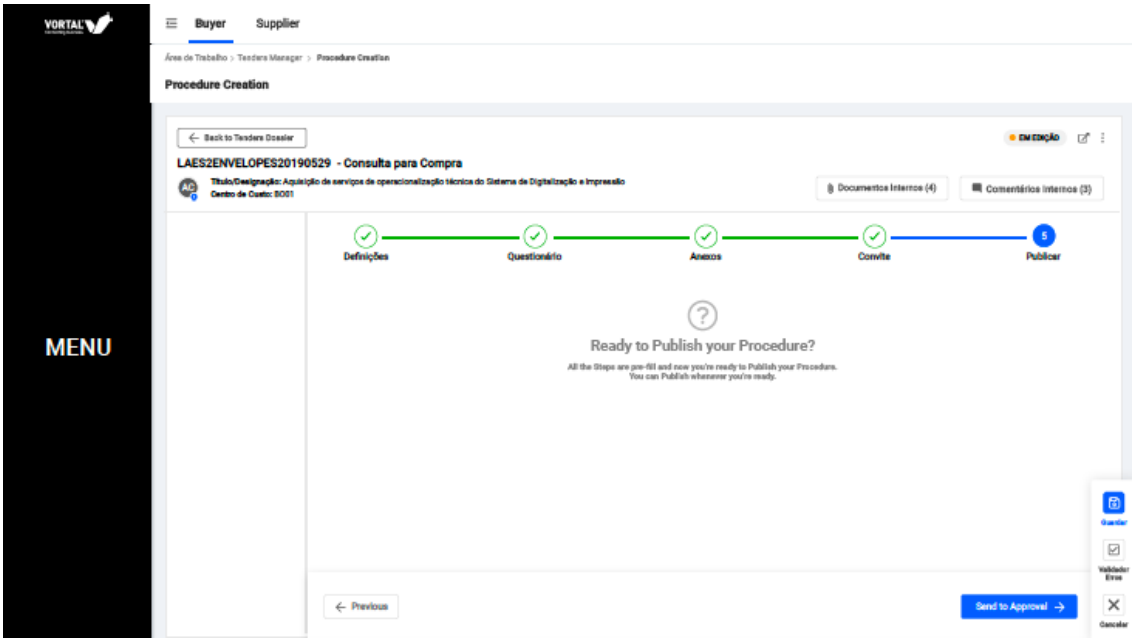


Figura 5.69 – *Mockup* do componente *Publish Step* (sem erros mas com fluxo de aprovação para aprovar).

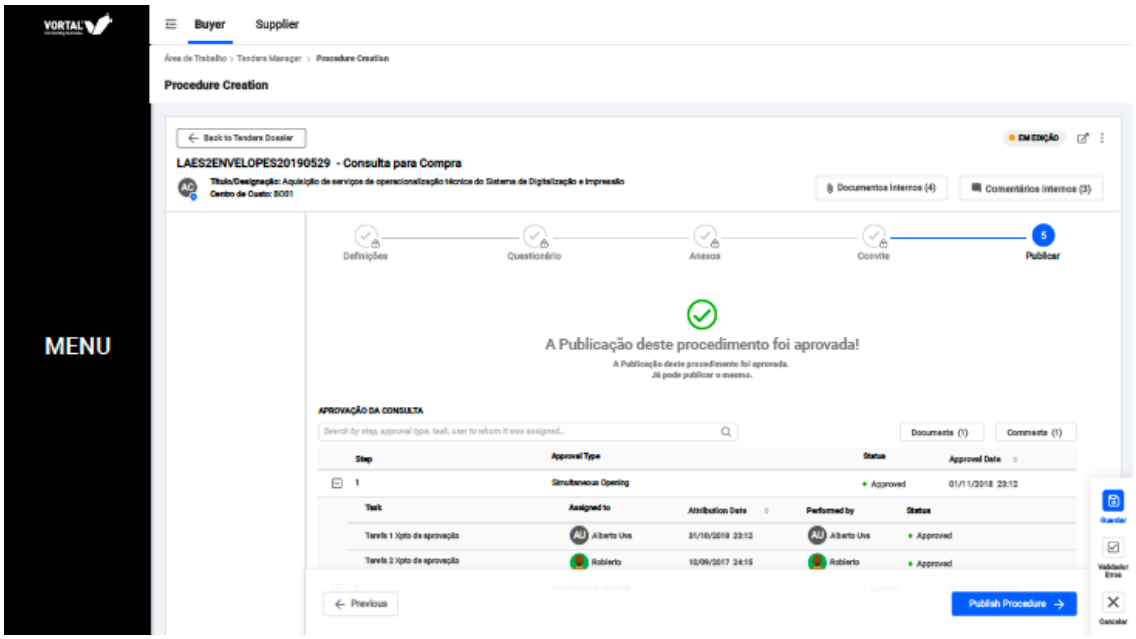


Figura 5.70 – *Mockup* do componente *Publish Step* (sem erros e pronto a publicar).

Como visto anteriormente, este é um componente que permite disponibilizar diferentes ecrãs, então o estagiário optou por criar um método mapeador do conteúdo a ser renderizado dependendo do estado da publicação. O estado da publicação é um enumerado *EnumPublishStatus* com diferentes valores (Figura 5.71).

```
let toRender = this.renderIsNotReady();
if (this.props.publishStatus) {
  switch (this.props.publishStatus) {
    case EnumPublishStatus.InvalidToPublish:
      toRender = this.renderInvalidToPublish();
      break;
    case EnumPublishStatus.ReadyToPublish:
      toRender = this.renderIsNotReady();
      break;
    case EnumPublishStatus.ReadyToApprovePublish:
      toRender = this.renderApprovalWorkflow();
      break;
    case EnumPublishStatus.ReadyToPublishWithApprove:
      toRender = this.renderReadyToApprovePublish();
      break;
    default:
      break;
  }
}
return <div>{toRender}</div>;
```

Figura 5.71 – Renderização da interface gráfica do componente *Publish*.

Como dito, o estado da publicação manipula o conteúdo a ser renderizado na página, que pode ser o seguinte:

- *InvalidToPublish*: disponibiliza a mensagem de integração (componente anteriormente explicado) que informa o utilizador que existem erros que impossibilitam a publicação do procedimento (Figura 5.72);
- *ReadyToPublish*: disponibiliza a mensagem de integração (componente anteriormente

explicado) que informa o utilizador que está pronto a publicar o procedimento (Figura 5.73);

- *ReadyToApprovePublish*: disponibiliza o fluxo de aprovação (componente anteriormente explicado) que permite ao utilizador aprovar ou rejeitar o procedimento (Figura 5.74);
- *ReadyToPublishWithApprove*: disponibiliza a mensagem de integração (componente anteriormente explicado) que informa o utilizador que está pronto a publicar o procedimento após aprovação do procedimento (Figura 5.75).

```
renderInvalidToPublish = () => {
  return (
    <IntegrationMessage
      key="invalidToPublish"
      icon={this.icons.invalidToPublish}
      title={this.titles.invalidToPublish}
      subtitles={this.subtitles.invalidToPublish}
    />
  );
```

Figura 5.72 – Renderização da mensagem de integração do tipo *InvalidToPublish*.

```
renderReadyToPublish = () => {
  return (
    <IntegrationMessage
      key="readyToPublish"
      icon={this.icons.readyToPublish}
      title={this.titles.readyToPublish}
      subtitles={this.subtitles.readyToPublish}
    />
  );
};
```

Figura 5.73 – Renderização da mensagem de integração do tipo *ReadyToPublish*.

```
renderApprovalWorkflow = () => {
  return <ApprovalWorkflow approvalMoment={ApprovalMoment.Publish} />;
};
```

Figura 5.74 – Renderização da mensagem de integração do tipo *ReadyToApprovePublish*.

```
renderReadyToApprovePublish = () => {
  return (
    <IntegrationMessage
      key="readyToApprovePublish"
      icon={this.icons.readyToPublish}
      title={this.titles.readyToApprovePublish}
      subtitles={this.subtitles.readyToApprovePublish}
    />
  );
};
```

Figura 5.75 – Renderização da mensagem de integração do tipo *ReadyToPublishWithApprove*.

5.2.4.2. Fase de Certificação

Durante esta fase o estagiário resolveu alguns problemas (*bugs*) originados na Fase de Desenvolvimento da *Release 2* (Tabela 5.6).

Tabela 5.6 – Descrição dos bugs resolvidos durante a Fase de Certificação da *Release 2*.

Descrição do Problema	Módulo
O *Invitation Step* não deveria mostrar o campo “Sugestão pela VORCAT”.	*Create and Publish Procedure*
O campo “Justificação para a Adjudicação Direta” não está a ser mostrado.	*Create and Publish Procedure*
O campo “Tipo de Contrato” exibe opções quando o campo “Vocabulário Principal CPV” está vazio.	*Create and Publish Procedure*

Capítulo 6
Resultados

6.1. Fluxo de Certificação

Antes de se iniciar a Fase de Certificação de cada *release*, a equipa de *Quality Assurance* cria os casos de teste para cada RSD (aprovados por DEV, SD e pelo cliente) e de seguida são criados os dados de retrocompatibilidade (caso exista essa necessidade).

Como ilustrado na Figura 6.1, quando se inicia a Fase de Certificação os testes são executados e são criados *tickets* no AzureDevOps com os problemas encontrados. Esses *tickets* passam por uma triagem, de forma a evitar temas duplicados ou inválidos. Os problemas reportados podem ser resolvidos pela equipa de DEV, pela equipa de POP (quando são problemas de traduções) ou podem já estar a ser resolvidos pela equipa de PO (apoio a produção). No fim da resolução de cada *ticket*, os testes são novamente executados, validando se o problema reportado se encontra resolvido, caso esta situação aconteça o *ticket* é fechado. (Monteiro, 2020)

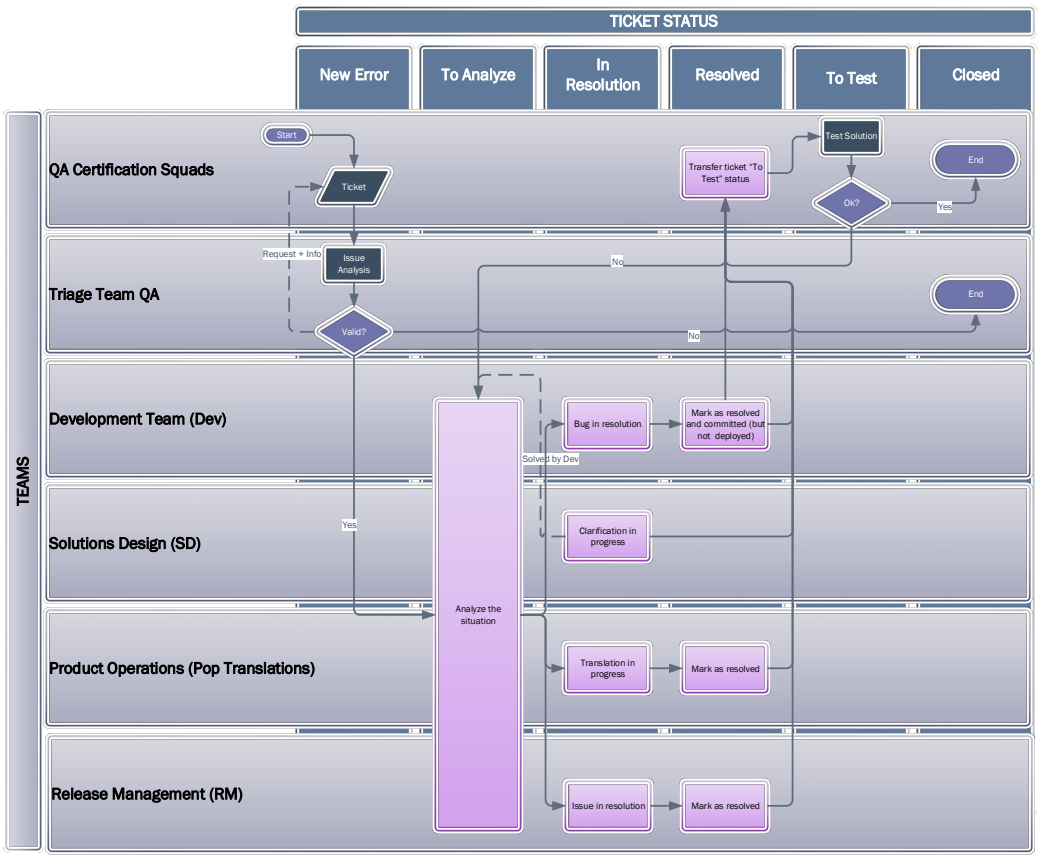


Figura 6.1 – Fluxo do processo de certificação.

6.2. Cumprimento de KPI's

Para cada *release* do projeto nextVISION foram definidos quais os KPI's (*Key Performance Indicators*) a serem cumpridos. É muito comum usar este termo em projetos de *software*, de forma a avaliar se determinadas ações estão a atender ou a superar as expetativas do cliente quanto ao compromisso inicialmente definido. Como ilustra a Figura 6.2, os KPI's definidos para este projeto estão relacionados com a percentagem máxima de tickets por resolver, dependendo do grau de criticidade (0% de *tickets blockers* e *criticals*, 20% de *tickets majors* e 80% de *tickets minors*). (Parmenter, 2010)

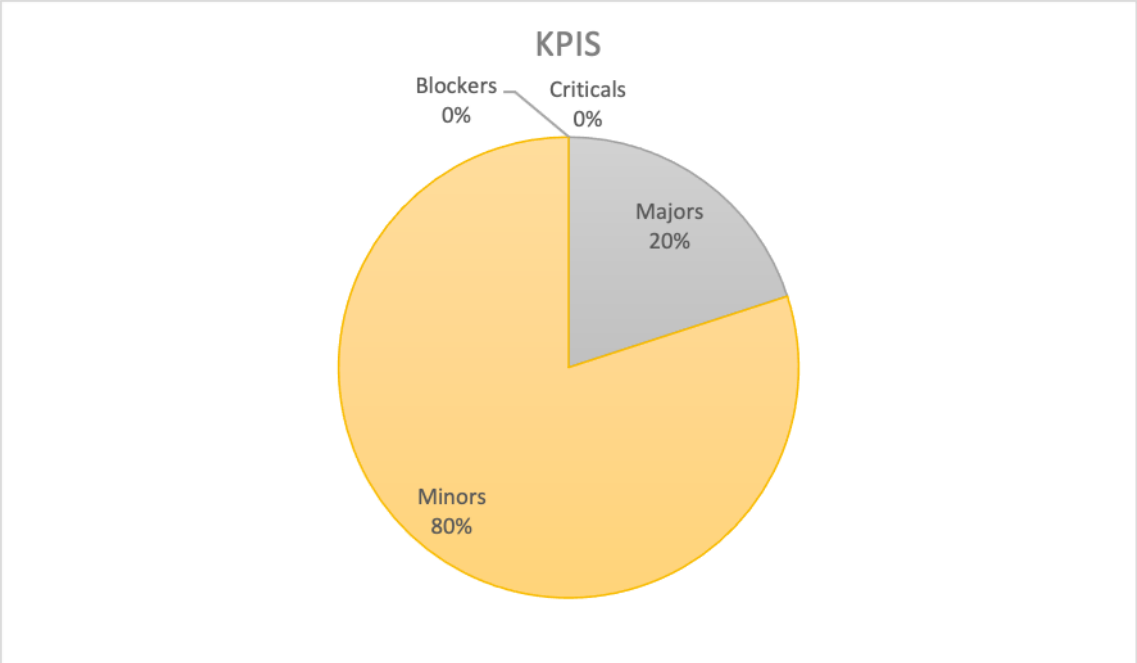


Figura 6.2 – Diagrama percentual de KPI's.

Após os KPI's estarem cumpridos e assim que existir uma versão estável e pronta para ser entregue em produção, são executados alguns testes de regressão, de maneira a garantir que não se perdeu nem se estragou nada com os novos desenvolvimentos. Após estes requisitos serem cumpridos o software está pronto para ser *deployed* e disponibilizado no ambiente de produção.

6.3. Discussão de Resultados

Os resultados serão explicados por *release* e através de uma comparação textual e ilustrativa dos componentes que foram implementados pelo estagiário (nextVISION) e dos componentes da plataforma legada (nextWAY).

6.3.1. Release 1.2

6.3.1.1. ComponentTabs

Este comportamento não existia na plataforma legada (nextWAY), mas para a plataforma atual (nextVISION) a equipa de UX/UI achou que seria uma mais valia, de forma a facilitar o utilizador no processo de mudança de ecrãs em contexto semelhante (Figura 6.3).

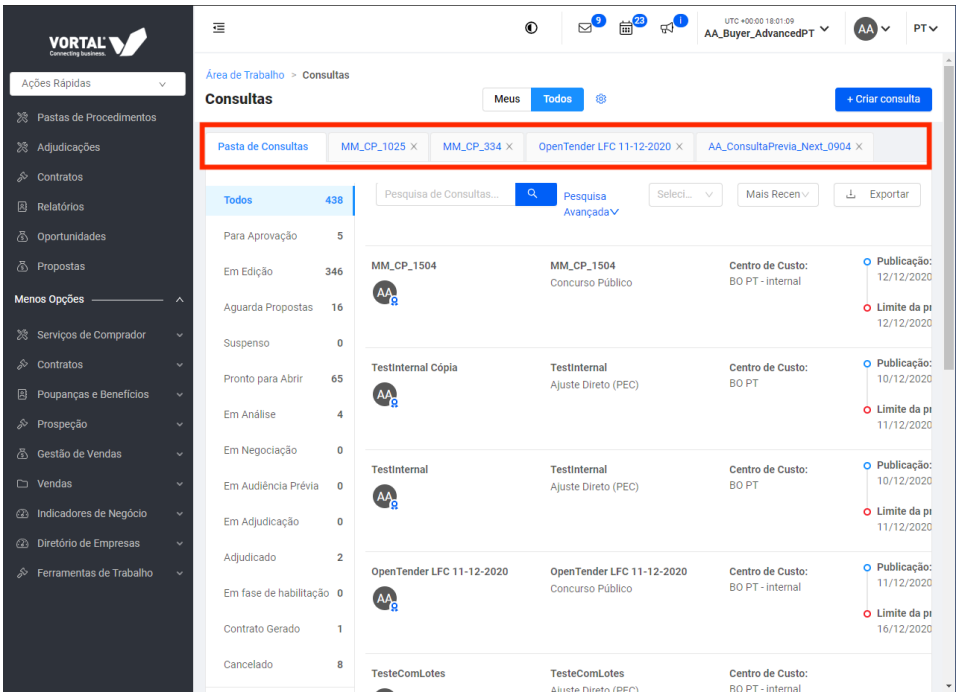


Figura 6.3 – *ComponentTabs* na plataforma nextVISION.

6.3.1.2. DateTimePopover

Como ilustrado na Figura 6.4, esta era uma funcionalidade que já existia na plataforma legada (nextWAY), mas que o seu comportamento era considerado um problema para o utilizador, pois este pretendia que no mesmo ecrã pudesse selecionar a data e o tempo (Figura 6.5).

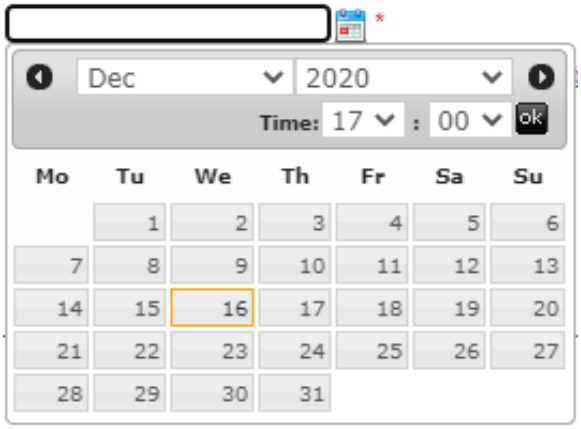


Figura 6.4 – *DateTimePopover* na plataforma nextWAY.

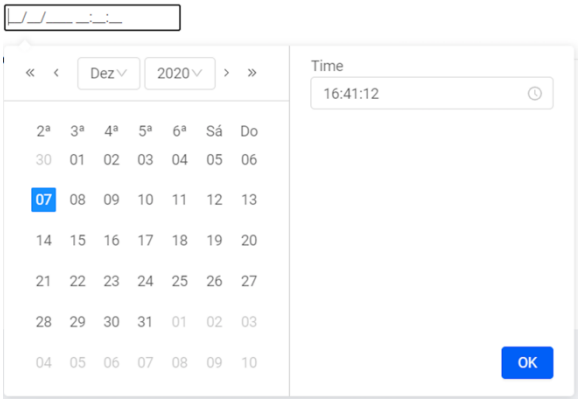


Figura 6.5 – *DateTimePopover* na plataforma nextVISION.

6.3.2. Release 1.3

6.3.2.1. CultureUtils

Este comportamento foi implementado de forma transversal para toda a plataforma, solucionando assim todos os casos em que existia a necessidade de tradução de bibliotecas externas. Como não é um comportamento graficamente visível, não é possível exemplificar o seu resultado através de ilustrações.

6.3.2.2. FeedbackToast

Como ilustra a Figura 6.6, esta funcionalidade já existia na plataforma legada, mas como é um comportamento fulcral para que o utilizador tome consciência das suas ações (exemplos: *save*, *update*, *delete*,...) foi considerado como prioridade, implementá-lo de forma transversal a toda a plataforma (Figura 6.7).

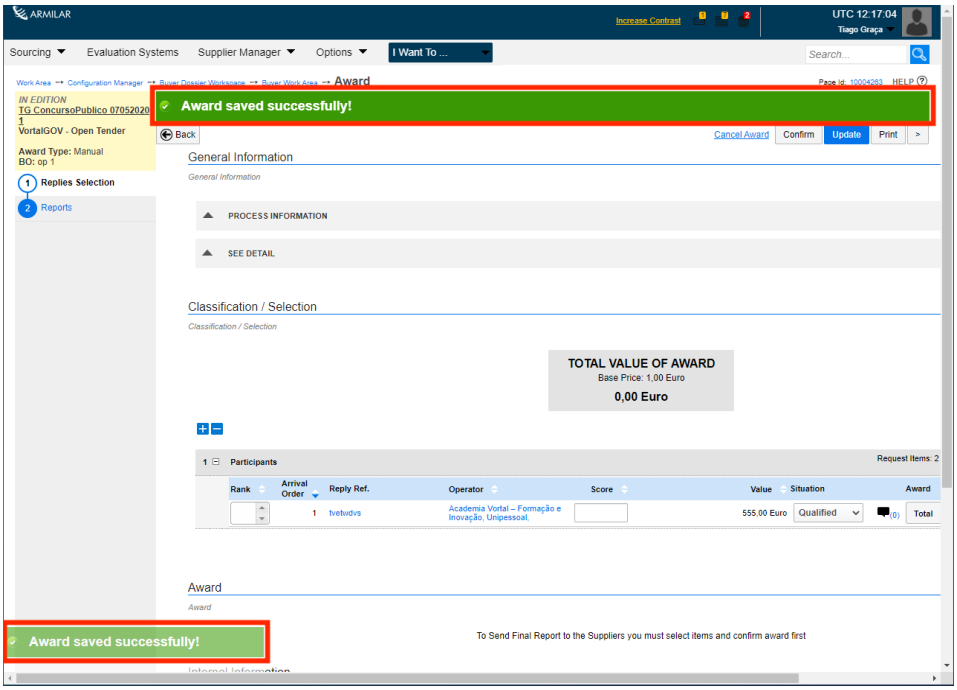


Figura 6.6 – *FeedbackToast* na plataforma nextWAY.

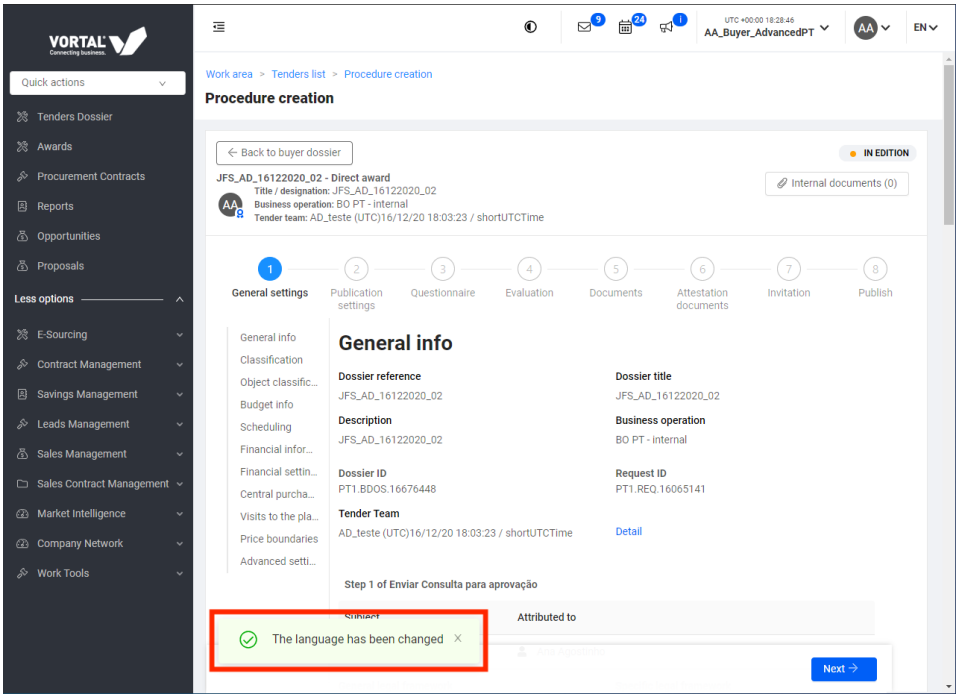


Figura 6.7 – *FeedbackToast* na plataforma nextVISION.

6.3.2.3. PillLabel

Esta funcionalidade não existia na plataforma legada, mas de forma a que o utilizador tivesse uma visualização graficamente apetecível do tipo de elemento a qual estava a interagir, surgiu a necessidade de ser implementado para a plataforma nextVISION (Figura 6.8).

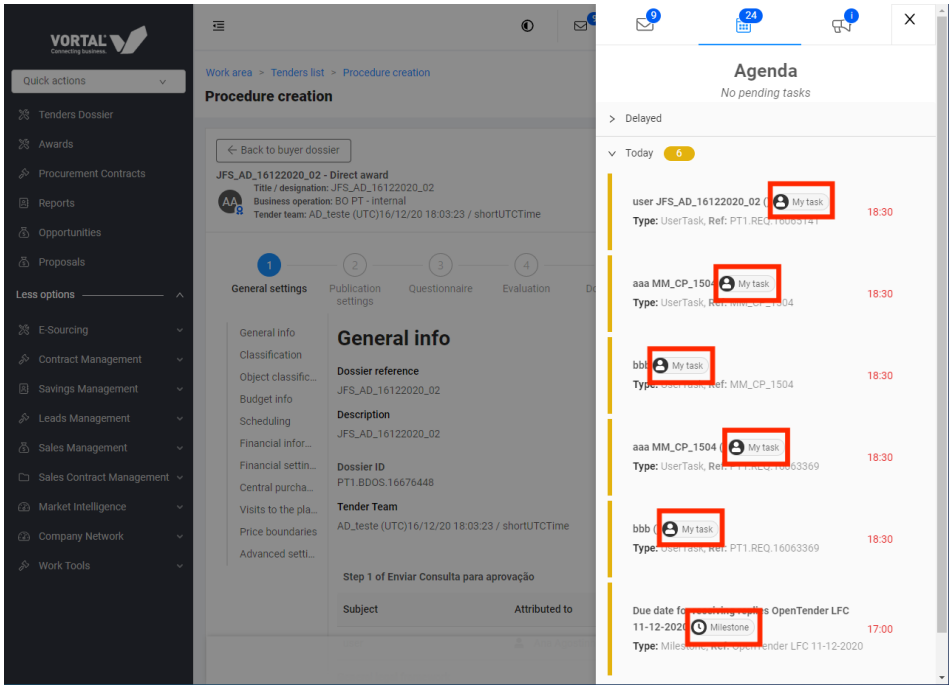


Figura 6.8 – *PillLabel* na plataforma nextVISION.

6.3.2.4. ErrorBoundary

Tal como existia para a plataforma nextWAY (Figura 6.9), era fulcral continuar a alertar o utilizador quando existisse um erro impactante na plataforma (Figura 6.10).

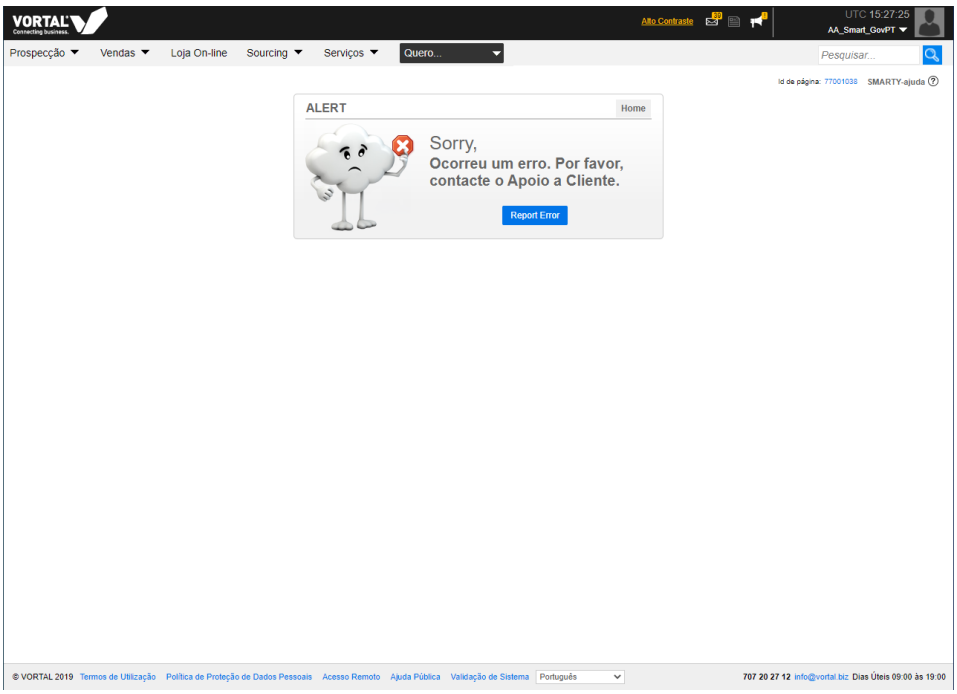


Figura 6.9 – *ErrorBoundary* na plataforma nextWAY.

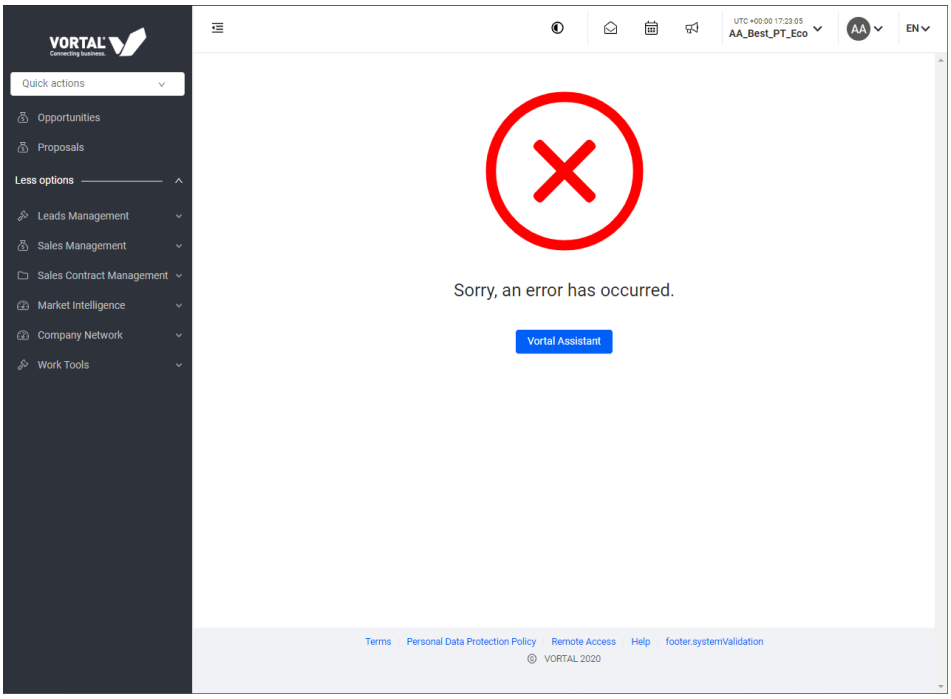


Figura 6.10 – *ErrorBoundary* na plataforma nextVISION.

6.3.3. Release 2

6.3.3.1. ApprovalWorkflow

Esta funcionalidade também existia na plataforma legada (Figura 6.11) e para a plataforma atual foi mantido o seu comportamento com a adição de novas funcionalidades (Figura 6.12).

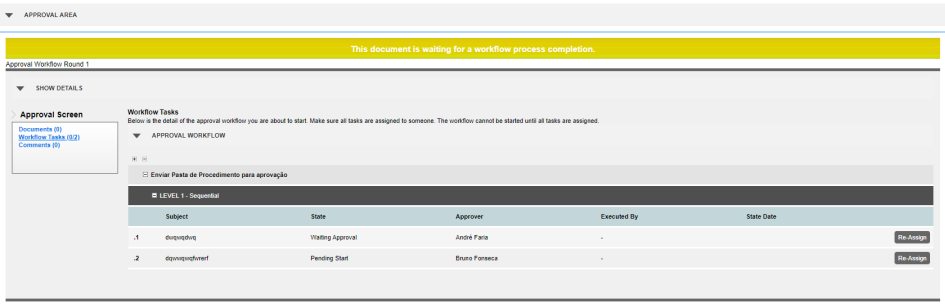


Figura 6.11 – *ApprovalWorkflow* na plataforma nextWAY.

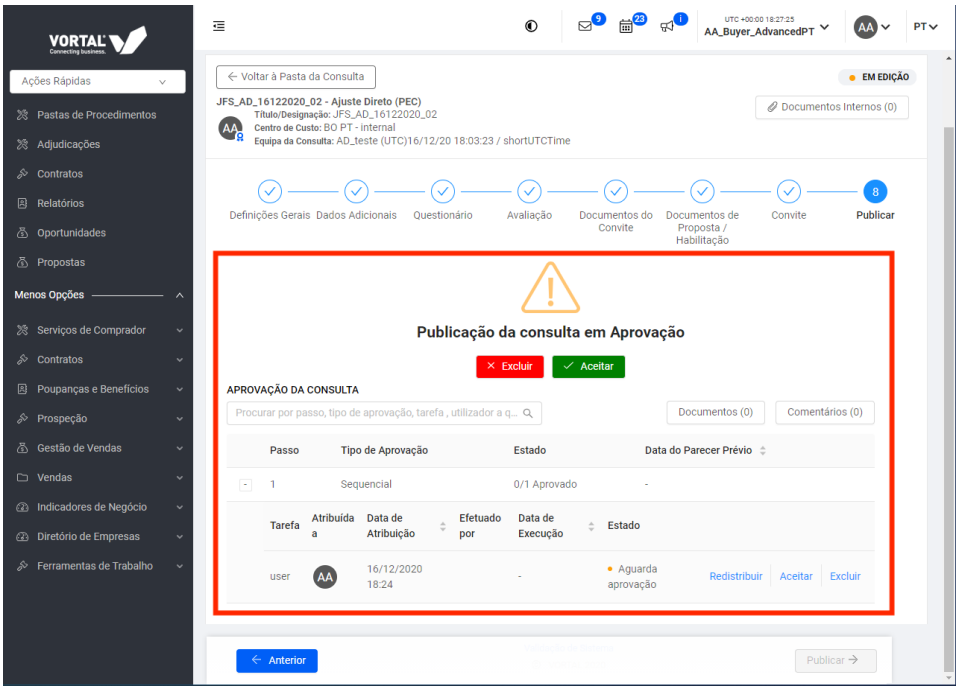


Figura 6.12 – *ApprovalWorkflow* na plataforma nextVISION.

6.3.3.2. ErrorValidator

Como ilustra a Figura 6.13, o comportamento de validação existia na nextWAY mas demasiado simples e pouco interativo para o utilizador. Na plataforma nextVISION, o validador de erros foi implementado com as funcionalidades descritas no tópico “5.2 Detalhes de Implementação” (Figura 6.14).

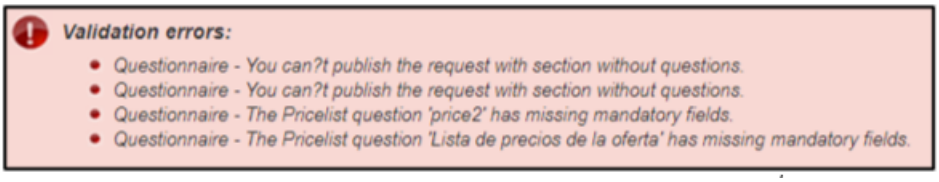


Figura 6.13 – *ErrorValidator* na plataforma nextWAY.

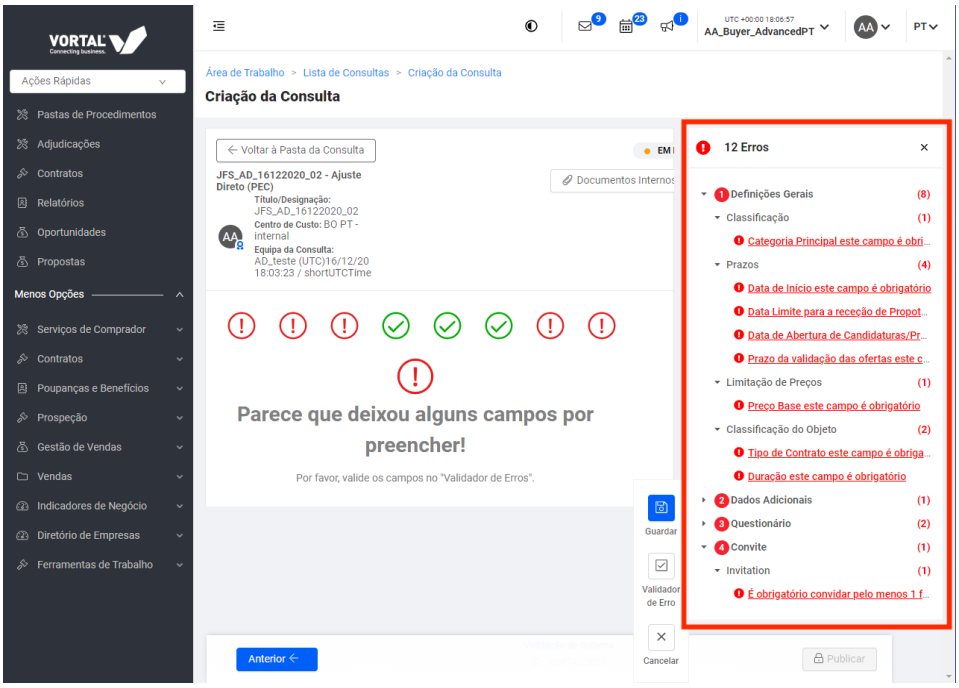


Figura 6.14 – *ErrorValidator* na plataforma nextVISION.

6.3.3.3. IntegrationMessage

Este comportamento não existia para a plataforma legada, mas o cliente achava preponderante que as mensagens de integração da plataforma com os serviços externos deviam ser mostradas no último *step* da criação do procedimento (Figura 6.15 e 6.16).

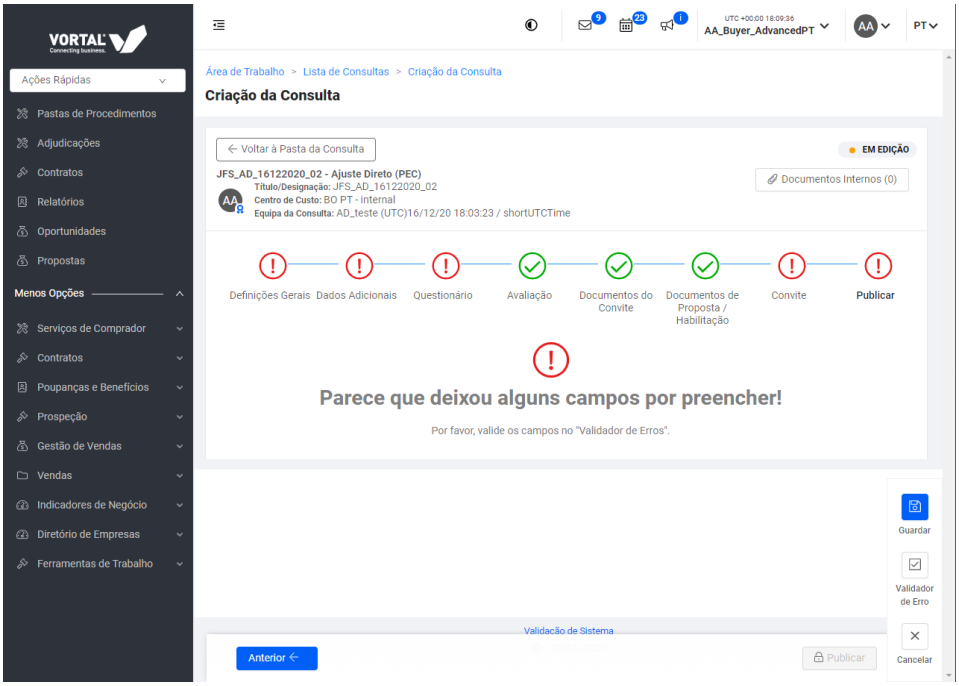


Figura 6.15 – *IntegrationMessage* na plataforma nextVISION (mensagem de erro).

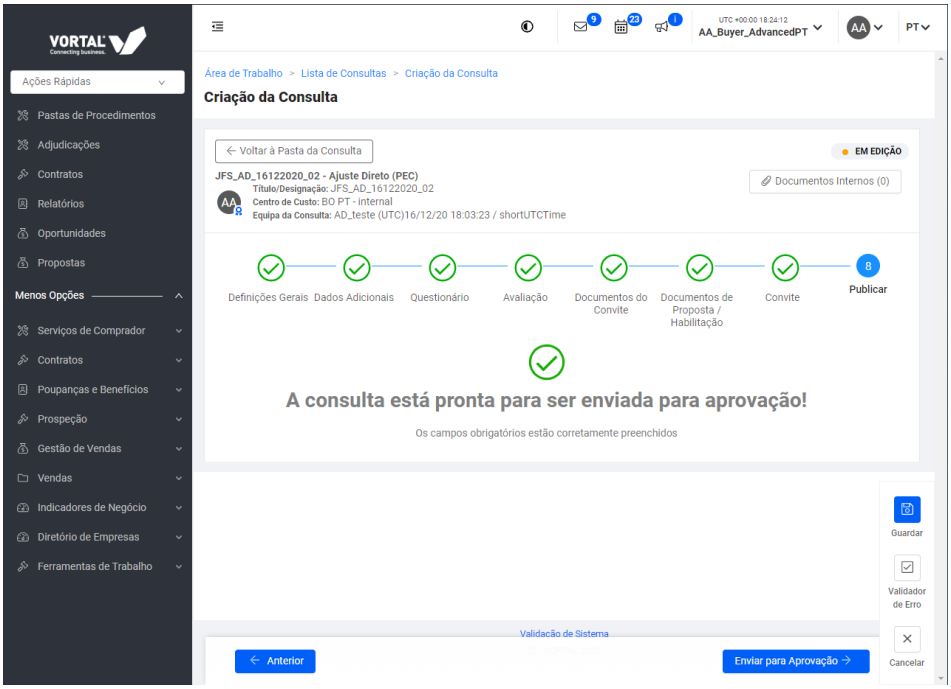


Figura 6.16 – *IntegrationMessage* na plataforma nextVISION (mensagem de sucesso).

6.3.3.4. Invitation Step

O *step* de convite de companhias era um comportamento que necessitava de ser replicado da plataforma legada (Figura 6.17) mas com uma apresentação diferente para o utilizador (Figura 6.18).

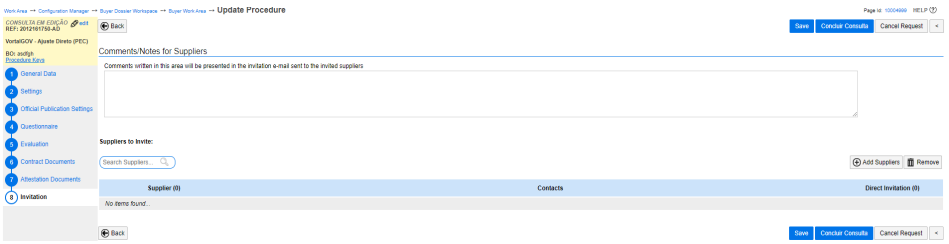


Figura 6.17 – *Invitation Step* na plataforma nextVISION (mensagem de sucesso).

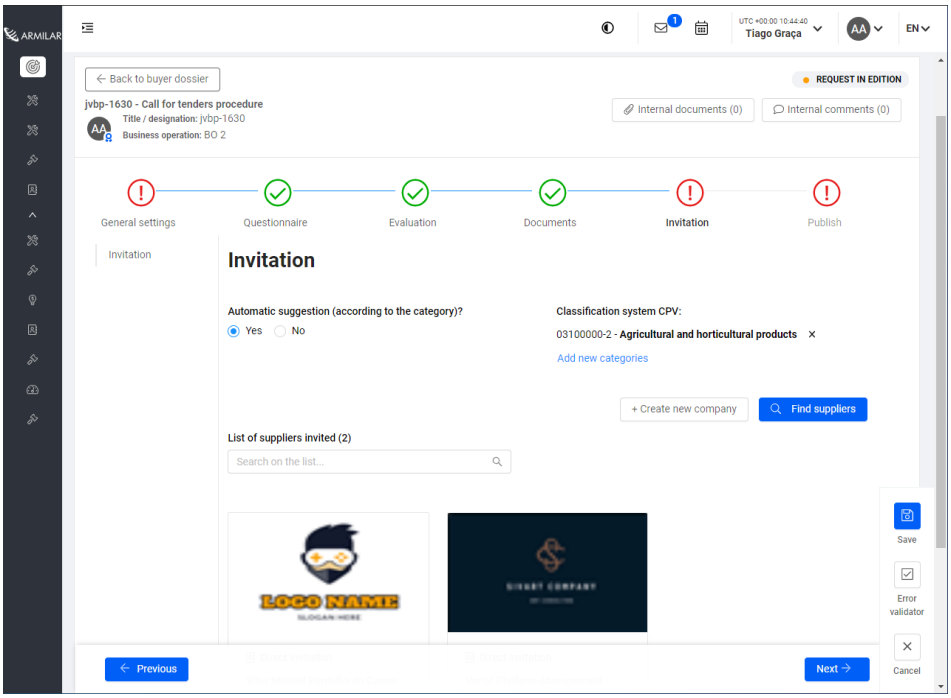


Figura 6.18 – *Invitation Step* na plataforma nextVISION (mensagem de sucesso).

6.3.3.5. Publish Step

Este *step* não existia na criação de procedimento da plataforma nextWAY, dessa forma surgiu a necessidade de ser implementado, de forma a que o utilizador seja informado da ação a realizar antes da publicação de um procedimento. Nas Figuras 6.19 e 6.20 são visíveis as apresentações dos comportamentos anteriormente implementados (*ApprovalWorkflow*, *ErrorValidator* e *IntegrationMessage).*

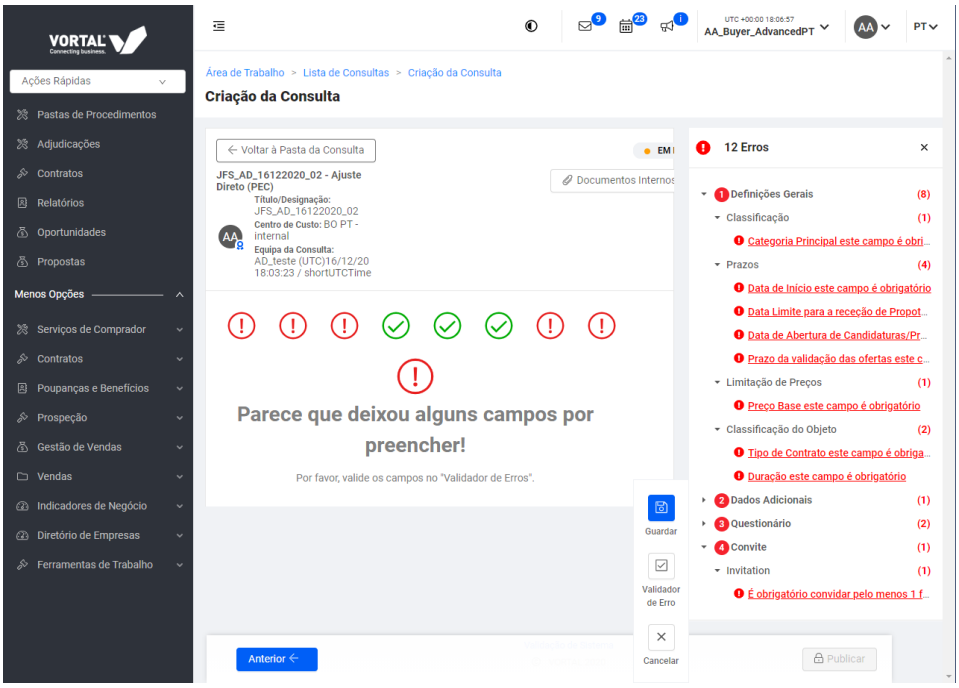


Figura 6.19 – *Publish Step* na plataforma nextVISION (*IntegrationMessage* e *ErrorValidator*).

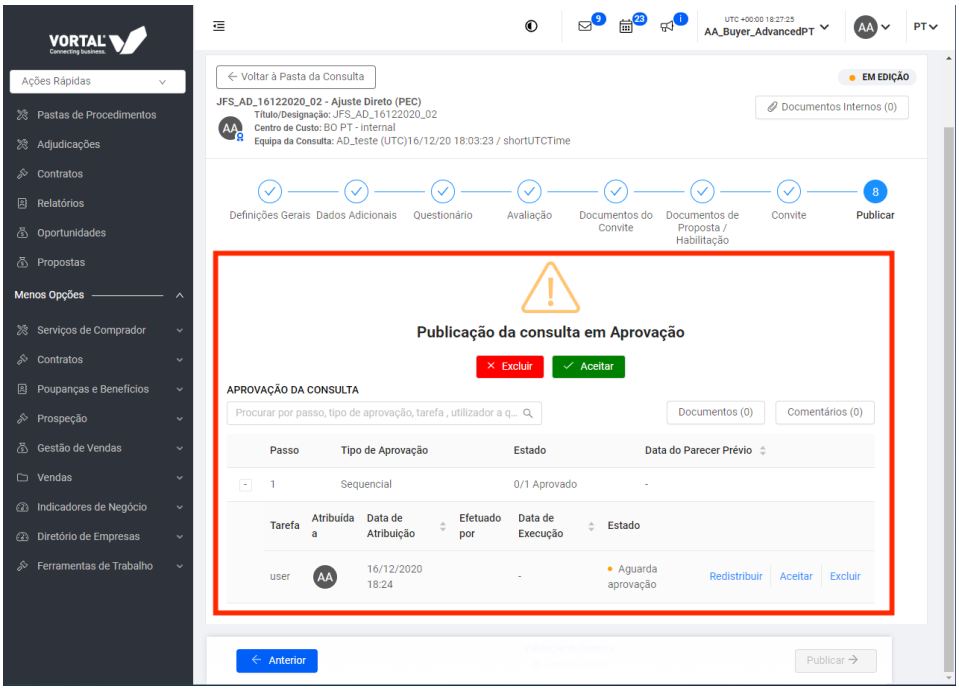


Figura 6.20 – *Publish Step* na plataforma nextVISION (*ApprovalWorkflow*).

É possível considerar que os resultados das tarefas desenvolvidas foram positivos. Os requisitos que foram exigidos em tempo de estágio, sendo eles prazos de entrega, reuniões semanais, contacto permanente com equipa de trabalho e clientes, foram realizados com total rigor e com os processos de trabalho mais adequados.

As tarefas e projetos foram abordados com o maior sentido de responsabilidade, sempre com o critério de desenvolver produtos de *software* de ótima qualidade, de acordo com os objetivos finais de cada um destes e com o pretendido pelo cliente final.

Capítulo 7
Conclusão

7.1. Resumo Geral do Trabalho

Nestes nove meses de estágio tive a excelente oportunidade de trabalhar na empresa Vortal, sobre a qual gostaria de destacar o bom ambiente de equipa, ambiente esse em que fui muito bem recebido, existindo sempre apoio e entreajuda entre todos os colegas.

Este estágio representou um grande desafio, tanto a nível pessoal como a nível profissional, onde muitas vezes existiu a necessidade de um esforço extra, procurando soluções alternativas, estudando certas tecnologias e bibliotecas. Este projeto envolveu e continua a envolver muito esforço, dedicação e ambição por parte de todos os elementos de equipa.

Relativamente às dificuldades sentidas, estas prenderam-se nas regras de negócio, que variam consoante o mercado a que se destinam. Todavia, as limitações de conhecimento foram ultrapassadas ao longo do tempo com o trabalho desenvolvido em equipa, que permitiu a troca de experiências relativamente à lógica de negócio com elementos seniores e experientes na empresa Vortal.

É importante frisar que os conhecimentos técnicos e práticos adquiridos no Mestrado de Engenharia de Software foram fundamentais no decorrer do estágio.

Em suma, concluo que numa profissão que se torna cada vez mais competitiva com o passar do tempo, existem duas competências essenciais que podem fazer toda a diferença: a especialização e a proatividade. Foi nesse sentido que este estágio decorreu. Obtenção e aprofundamento de *know-how* específico e bastante requisitado no mercado das IT e desenvolver a capacidade de iniciativa, rentabilizando o esforço e otimizando a produtividade, e assim transformar estas competências em mais-valias pessoais e profissionais.

7.2. Sugestões Futuras

Visto que a Vortal decidiu renovar a sua *stack* tecnológica, reutilizando as fundações da plataforma atual e garantindo uma transposição do *core* da solução atual para o novo paradigma de implementação, é imperativo afirmar que a estabilidade e que todas as funcionalidades serão mantidas, não prevendo a necessidade de realizar uma nova atualização tecnológica. Assim sendo, no futuro, espera-se melhorar alguns aspetos de usabilidade da plataforma, assim como implementar os módulos em falta, funcionalidades que atualmente se

encontram disponíveis na plataforma nextWAY.

É importante referir que devido ao bons resultados obtidos neste estágio e ao contributo reconhecido do estagiário no projeto, a empresa Vortal contratou o estagiário, deixando assim de ser um trabalhador contratado externamente à empresa PrimeIT.

Glossário

A

Adjudicação: processo em que os fornecedores, interessados em satisfazer as necessidades dos compradores, respondem a um anúncio submetendo uma determinada oferta. Posteriormente a empresa compradora escolhe a oferta que mais lhe agrada.

Arquitetura Monolítica: abordagem arquitetónica e organizacional no desenvolvimento de *software*, na qual o *software* é centrado numa única aplicação.

***Array*:** estrutura de dados que armazena uma coleção de elementos de tal forma que cada um dos elementos possa ser identificado por um índice ou por uma chave.

B

***Back-End*:** sistema responsável pelas regras de negócios, *webservices* e APIs de uma aplicação.

***Back-End-in-Front-End*:** nomenclatura da empresa Vortal para o sistema responsável pela ligação entre o *Back-End* e o *Front-End* da aplicação.

***Branch*:** representa uma linha independente de desenvolvimento.

***Bugfixing*:** fase de eliminação de erros de software.

C

Certificação: momento em que o *software* é testado e que são realizadas as respetivas correções.

***Cloud*:** computadores e servidores compartilhados e interligados via Internet, cujo os seus serviços podem ser fornecidos/alugados a clientes quer para processamento que para armazenamento de dados.

***Coach* Ágil:** indivíduos que ajudam as organizações a implementar a metodologia ágil.

Código-Fonte: conjunto de palavras ou símbolos escritos, contendo instruções numa das linguagens de programação existentes, de maneira lógica.

Consultoria: atividade profissional de diagnóstico e formulação de soluções acerca de um assunto ou especialidade. O profissional desta área é chamado de consultor.

***Container*:** unidade de *software* que empacota código e todas as dependências necessárias à execução da aplicação.

Common Intermediate Language: linguagem de programação de baixo nível do ambiente de programação da Microsoft.

Common Language Infrastructure: especificação aberta desenvolvida pela Microsoft que descreve o código executável.

Common Language Runtime: componente de máquina virtual da plataforma .NET que gere a execução de programas .NET.

Continuous Delivery: abordagem de engenharia de *software* na qual as funcionalidades do *software* são disponibilizadas continuamente através de *deployments* automáticos.

Continuous Integration: prática de automatização da integração de alterações de código de vários colaboradores num único projeto de *software*.

Contratação Eletrónica: contrato ou acordo realizado entre duas ou mais vontades, realizado por meio de um *software* e de aparelhos eletrónicos.

D

***Dashboard*:** painel de controlo.

***Deployment*:** atividades que permitem que o *software* esteja disponível para uso.

***Document Object Model*:** convenção multiplataforma e independente de linguagem de programação para representação e interação com objetos em documentos HTML, XHTML e XML.

E

Escalabilidade: característica desejável em todo o sistema que indica que o mesmo se encontra pronto para crescer.

***eSourcing*:** plataforma *online* que permite que as empresas realizem os seus processos de compras, utilizando as melhores práticas de negociação em cada categoria de negócio.

Estágio: período durante o qual uma pessoa ou um grupo exerce uma atividade temporária com vista à sua formação ou aperfeiçoamento profissional.

***eProcurement*:** sistema de negociação via internet que facilita a gestão de compras.

***eXtreme Programming*:** metodologia ágil de desenvolvimento de *software* que visa produzir *software* de alta qualidade.

F

***Framework*:** uma abstração que une código comum em vários projetos de *software*, com o fim de providenciar uma funcionalidade genérica.

***Front-End*:** interface de interação com o utilizador.

K

Key Performance Indicators: avaliação de determinadas ações, de forma a concluir se estão a atender ou a superar as expetativas do cliente quanto ao compromisso inicialmente definido

Know-How**:** conhecimento prático.

L

Leads**:** potencial comprador que demonstra interesse num produto ou serviço.

Logging**:** processo de registo de eventos que ocorrem num determinado *software*.

M

Máquina Virtual: *software* que executa programas como um computador real.

Microserviços: abordagem arquitetónica e organizacional no desenvolvimento de *software*, na qual o *software* é dividido em pequenos serviços independentes que se comunicam usando APIs bem definidas.

Mockup**:** modelo em escala ou em tamanho real de um protótipo, usado para implementação de *software*, demonstração, avaliação de *design*, promoção e outros fins.

Model Driven Architecture**:** metodologia de desenvolvimento de *software* onde o *core* de implementação é a modelação. A partir de um modelo abstrato do sistema é gerado um modelo mais concreto, através deste processo de refinamento dos modelos podemos gerar o código fonte a ser produzido.

Model View Controller**:** padrão de projeto de *software* focado na reutilização de código e na separação de conceitos em três camadas interconectadas, onde a apresentação dos dados e a interação dos utilizadores (*Front-End*) são separados dos métodos que interagem com a base de dados (*Back-End*).

Multi-Device**:** possibilidade de utilização em diversos aparelhos eletrónicos.

N

Nearshore**:** tipo de subcontratação ou terceirização de uma atividade com remunerações mais baixas que no próprio país, que se encontra relativamente perto na distância ou na zona horária.

O

Open-Source**:** ideologia de desenvolvimento que promove o acesso e uso do código implementado por qualquer pessoa, dependendo da licença optada.

P

Procedimento: nomenclatura usada para compras na área de contratação eletrónica.

***Product Owner*:** responsável pela gestão do produto.

***Project Manager*:** indivíduo responsável pela gestão de um projeto.

Proposta: processo de resposta a um determinado procedimento.

Props: técnica de partilha de código entre componentes de React.

***Pull Request*:** mecanismo de geração de notificação que sinaliza a conclusão do desenvolvimento de uma funcionalidade.

R

***Refactoring*:** processo de esclarecer e simplificar o *design* do código existente, sem alterar o seu comportamento.

***Release*:** lançamento de uma nova versão de *software* ou de um incremento de *software*.

Repos: repositórios GIT onde o código é armazenado.

Requisito: reflete a necessidade e expectativa das partes interessadas no projeto.

Responsivo: processo de resposta de forma adequada e rápida à situação.

***Rollback*:** processo de encerramento de uma transação.

***Routing*:** processo de reencaminhamento de pacotes que se basei no endereço de IP e na máscara de rede.

S

***Scrum*:** método de gestão de projetos grandes e complexos.

***Single Page*:** aplicação web que consiste em uma única página web com o objetivo de fornecer a experiência de utilizador semelhante a uma aplicação *desktop*.

Sistema Legado: sistema antigo que permanece em operação numa organização.

***Soft Skills*:** habilidades que destacam profissionais.

Software: considerado como um programa, rotina ou conjunto de instruções que controlam o funcionamento de um computador.

T

***Tech Stack*:** lista de serviços de tecnologia usados para construir e executar o *software*.

Testes de Regressão: técnica de teste de *software* que garante que não surgiram novos defeitos em componentes já analisados e testados.

Tipagem: conjunto de regras que atribuem uma propriedade chamada de tipo para as várias construções de um *software*.

U

***User Experience*:** disciplina responsável por projetar experiências de utilização encantadoras de forma a fidelizar e conquistar clientes.

***User Interface*:** espaço de interação entre os humanos e as máquinas.

***User Stories*:** especificação de uma ou mais funcionalidades de um *software*.

***Utilities*:** considera-se *utilities* como as áreas de produção, transporte, distribuição e comercialização de energia e água.

Bibliografia

Afonso, F. (2019a). *Documento de Especificação da "Error Page" - Versão 1*.

Afonso, F. (2019b). *Documento de Especificação das "Alert Messages" - Versão 2*.

Afonso, F. (2019c). *Documento de Especificação das "Tabs" - Versão 1*.

Afonso, F. (2019d). *Documento de Especificação do "Date Time Picker" - Versão 1*.

Ant Design. (n.d.). *Ant Design*. https://ant.design

Awesome, F. (n.d.). *Font Awesome*. https://fontawesome.com

Bandeira, R. (2014). *Vortal - A História do Agile*.

Banks, A., & Porcello, E. (2017). Learning React Functional Web Development with React and Redux. In *O'REILLY*.

Bierman, G., Abadi, M., & Torgersen, M. (2014). Understanding TypeScript. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. https://doi.org/10.1007/978-3-662-44202-9_11

Cabral, P. (2019). *Vortal - Do Monolítico para os MicroServiços - Versão 4*.

Cabral, P. (2020). *Visão Geral - nextVISION*.

Costa, L. (2019). *RSD do Módulo "Create and Publish Procedure" - Versão 203*.

Cruth, M. (2018). *Discover the Spotify Model*. https://www.atlassian.com/agile/agile-at-scale/spotify

Fontana, R. M., Reinehr, S., & Malucelli, A. (2014). Agile Processes in Software Engineering and Extreme Programming. In *Lecture Notes in Business Information Processing*. https://doi.org/10.1007/978-3-319-06862-6

Git. (2018). *Git - About Version Control*. Online.

Graça, T. (2019). *Planeamento - nextVISION*.

Martin, R. C. (2011). Using Docker. In *O'Reilly*.

Microsoft dotnet csharp. (2020). *C# Documents*. Https://Docs.Microsoft.Com/.

Mishra, A. (2020). *Spotify Scaling Agile Model*. https://www.pmtoday.co.uk/spotify-scaling-agile-model/

Monteiro, M. (2020). *Processo de Certificação - nextVISION*.

Moran, A. (2014). Agile software development. In *SpringerBriefs in Computer Science*. https://doi.org/10.1007/978-3-319-05008-9_1

Parmenter, D. (2010). Key Performance Indicators (KPI). In *Key Performance Indicators (KPI*

): Developing , Implementing , and Using Winning KPIs Second Edition.

PrimeIT. (n.d.). *Website da PrimeIT*. https://www.primeit.pt/sobre/

PrimeIT. (2019). *Apresentação da Empresa PrimeIT*.

Rossberg, J. (2019). Agile Project Management with Azure DevOps. In *Agile Project Management with Azure DevOps*. https://doi.org/10.1007/978-1-4842-4483-8

SASS. (n.d.). *SASS*. https://sass-lang.com

Steyer, R., & Steyer, R. (2014). JavaScript. In *JavaScript*. https://doi.org/10.3139/9783446439474.fm

Strauss, D. (2020). Getting Started with Visual Studio 2019. In *Getting Started with Visual Studio 2019*. https://doi.org/10.1007/978-1-4842-5449-3

Surwase, V. (2016). REST API Modeling Languages -A Developer's Perspective. *IJSTE - International Journal of Science Technology & Engineering*.

Visual Studio Code. (2015). *Documentation for Visual Studio Code*. Visual Studio Code Documentation.

Vortal. (n.d.). *Website da Vortal*.

Vortal. (2019). *Apresentação da Empresa Vortal*.