INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




Planificação e Desenvolvimento de Testes Aplicacionais

TOMÁS HENRIQUES DORDIO GODINHO
outubro de 2022

POLITÉCNICO
DO PORTO

Instituto Superior de Engenharia do Porto

Planificação e Desenvolvimento de Testes Aplicacionais

Tomás Henriques Dordio Godinho

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

Orientadora: Isabel Sampaio

Co-orientador: Alberto Sampaio

Júri:

Presidente:

Vogais:

Porto, Outubro 2022

Resumo

O ION é uma aplicação web e móvel para gestão de processos de operação e manutenção de ativos infraestruturais de sistemas de captação, tratamento, e distribuição de água e águas residuais. Nesta aplicação são utilizadas como *frameworks* de *frontend* e *backend* respetivamente Angular e Phalcon. Apesar de diversas vantagens desta aplicação, esta tem a desvantagem de que atualmente sempre que é disponibilizada uma nova versão a sua implementação depende de um processo manual de testes de controlo de qualidade, executado meramente por atores humanos, que é demorado e propenso a ocorrência de erros.

A automatização dos testes permite agilizar e tornar mais fiáveis os resultados. Consequentemente, este projeto teve então como objetivo o planeamento e desenvolvimento de um sistema de testes automáticos para o ION a ser utilizado pela DouroECI. Durante o desenvolver deste projeto, foi iniciado o desenvolvimento da aplicação sucessora do ION, o ION 2. Entre as duas versões houve mudança das *frameworks* de *frontend* e *backend*, passando a ser utilizadas o Vue 3 e o Laravel. Esta alteração implicou um período de adaptação às novas *frameworks*, bem como uma nova investigação e consequentemente a implementação de novas soluções.

De forma a alcançar os objetivos propostos foi feito um estudo do estado da arte na área da automação dos testes. Este estudo teve por objetivo identificar, comparar e escolher as metodologias, os tipos de testes e as ferramentas mais adequadas para as necessidades do projeto e da equipa de desenvolvimento. Este estudo permitiu selecionar para o ION a utilização das seguintes *frameworks* de teste: Codeception, Jest, Postman e Cypress. Para ION 2 foram selecionadas as *frameworks* PHPUnit, Vitest e Cypress.

Como método de avaliação do artefacto desenvolvido foi elaborado um questionário para avaliar o nível de satisfação da equipa de desenvolvimento e a gestão da mesma com a nova metodologia de testes e da sua documentação, em comparação com a abordagem existente anteriormente na empresa. Os resultados foram bastante positivos e mostraram um grande contraste de satisfação na equipa entre a metodologia anterior com a atual.

Este trabalho resultou num artefacto que facilita a realização de testes unitários *frontend* e *backend*, bem como de integração e *End to End*, de forma rápida e eficiente pela empresa DouroECI na sua nova aplicação, o ION 2.

Palavras-chave: Testes, Desenvolvimento de Software, Qualidade de Software

Abstract

ION is a web and mobile application for managing the operation and maintenance processes of infrastructure assets of water and wastewater collection, treatment, and distribution systems. In this application the frontend and backend frameworks used are Angular and Phalcon respectively. Despite several advantages of this application, it has the disadvantage that currently whenever a new version is released its implementation depends on a manual process of quality control tests, performed merely by human actors, which is time consuming and prone to the occurrence of errors.

Test automation makes it possible to speed up and make the results more reliable. Consequently, this project aimed at planning and developing an automatic testing system for the application ION to be used by DouroECI. During the progress of this project, the development of the successor application of ION, ION 2, was initiated. Between the two versions, there was a change in the frontend and backend frameworks, leading to the use of Vue 3 and Laravel respectively. This change implicated a period of adaptation to the new frameworks, as well as a new investigation and consequently the implementation of new solutions.

To achieve the proposed objectives, a study of the state of the art in test automation was carried out. This study aimed to identify, compare, and choose the methodologies, types of tests and the most appropriate frameworks for the needs of the project and the development team. This study allowed the selection of the following test frameworks for ION: Codeception, Jest, Postman, and Cypress. For ION 2 the frameworks PHPUnit, Vitest and Cypress were selected.

As a method of evaluating the developed artifact, a questionnaire was prepared to assess the level of satisfaction of the development team and management with the new testing methodology and its documentation, compared to the approach previously used in the company. The results were very positive and showed a great contrast of satisfaction in the team between the previous methodology and the new one.

This work resulted in an artifact that facilitates the performance of frontend and backend unit tests, as well as integration and End to End, quickly and efficiently by the DouroECI company in its new application, ION 2.

Keywords: Tests, Software Development, Software Quality

Agradecimentos

Em primeiro lugar quero agradecer à minha orientadora, Isabel Sampaio, e coorientador Alberto Sampaio, por me terem guiado ao longo dos meses contra todas as diversidades que surgiram durante a realização desta dissertação.

Queria também agradecer a toda a equipa da DouroECI que me acolheu e permitiu aprender muito. Em especial aos três membros que me acompanharam mais, o João Coelho, o Miguel Levi e o Carlos Costa.

Por fim queria também agradecer a minha família e amigos que me apoiaram ao longo de toda esta experiência.

Índice

Lista de Figuras

Lista de Tabelas

Acrónimos

Lista de Acrónimos

API	A*pplication Programming Interface*
ATLM	*Automated Test Lifecycle Methodology*
BDD	Behavior-Driven Development
CI/CD	Continuous integration/Continuous deployment
FAST	Function Analysis System Technique
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
ISEP	Instituto Superior de Engenharia do Porto
JSON	JavaScript Object Notation
MSW	Mock Service Worker
QEF	Quantitative Evaluation Framework
REST	Representational State Transfer

1 Introdução

Neste primeiro capítulo descreve-se, de forma sucinta, o problema principal que se pretende analisar e resolver juntamente com uma contextualização do projeto e as dificuldades encontradas. Em seguimento também são delineados os objetivos e planeamento de forma a resolver o problema descrito. Por fim é descrita a estrutura do documento para facilitar a leitura do mesmo.

1.1 Contexto

A DouroECI é uma empresa especializada na gestão da água oferecendo serviços nas áreas engenharia, consultoria e inovação[1]. A área de engenharia tem como foco projetos de engenharia hidráulica e hidrologia. A área de consultoria tem como foco o planeamento e gestão de projetos. Por fim, a área de inovação tem como foco o desenvolvimento de soluções tecnológicas inovadores, para promover a gestão e monitorização eficiente da água. Nesse âmbito a empresa desenvolveu e comercializa dois sistemas de informação, os produtos OBI e ION. O OBI é uma aplicação de integração de dados e Inteligência empresarial.

O ION, que vai ser destacado no capítulo 2, é uma aplicação web e móvel com uso intensivo de dados, para a gestão de processos complexos de manutenção e gestão de ativos infraestruturais na área das águas. O ION destina-se a ser usado pelos gestores e operadores de campo para planear e documentar os trabalhos realizados. É um software em evolução contínua e acelerada, muitas vezes com prazos exigentes para o cumprimento de requisitos especificados.

Durante o período de escrita desta dissertação, a empresa decidiu reduzir substancialmente o esforço dedicado ao desenvolvimento de novas funcionalidades na aplicação ION, a versão do software que se encontra em produção, a partir desta ponto denominada de ION 1. Esta decisão teve como objetivo principal desviar os recursos disponíveis para o desenvolvimento de uma nova versão do software, denominada de ION 2, com recurso a novas *frameworks* de *backend* e *frontend*, e código base completamente refeito de raiz. Esta decisão foi tomada principalmente por o ION 1 apresentar algumas lacunas importantes na estrutura do código

fonte e na performance de algumas funcionalidades, dificultando a manutenção e melhoria do produto, e também a integração de novos membros da equipa de desenvolvimento.

Em consequência, foi iniciado o desenvolvimento do uma nova aplicação com a mesma função e funcionalidades, o ION 2, de forma a corrigir os problemas existentes e facilitar o processo de manutenção a longo prazo.

Assim a planificação e desenvolvimento de todo o trabalho foi adaptado a estas alterações, tendo uma estratégia parcial para o ION 1 e uma completa para o ION 2.

1.2 Problema

A empresa iniciou o desenvolvimento das aplicações referidas em 2017, usando sempre processos manuais de testes, incluindo a realização de testes de aceitação na validação de cada funcionalidade desenvolvida, e na disponibilização de novas versões através da realização de testes de controlo de qualidade diretamente no UI. Este processo era considerado pela empresa como sendo demorado, complexo e falível. Para além disso, a cobertura dos testes era considerada insuficiente. Devido a tais limitações, a empresa decidiu complementar o processo manual de testes com um processo automático de testes, como componente integrante do ciclo de vida de desenvolvimento da aplicação, promovendo desta forma a adoção de boas práticas de desenvolvimento de software e a qualidade final do produto, e consequente redução de custos no seu desenvolvimento.

A implementação de testes automáticos nas diferentes fases de desenvolvimento ajudará a detetar problemas atempadamente, evitando que estes passem para produção. Outras vantagens em relação a testes manuais é melhorar a eficiência da equipa e a maior facilidade de replicar casos de teste [2].

1.3 Objetivos Propostos

De modo a resolver o problema exposto, este trabalho tem como objetivo principal planear e desenvolver um sistema de testes adequado e capaz de satisfazer as necessidades de uma plataforma como a ION. Este objetivo principal pode ser divido nos seguintes quatro objetivos:

1. Identificar os problemas e oportunidades de melhoria nos processos existentes do ION;
2. Estudar e especificar soluções, considerando o diagnóstico realizado, incluindo o estudo e escolha de tipos e técnicas de testes de software, casos de teste, fases/níveis de teste (unitários, integração e sistema), ferramentas de suporte aos testes - classificação e escolha de ferramentas a usar em cada contexto – e automatização de testes;
3. Garantir a documentação adequada de todos os processos de testes de software implementados;

4. Documentar os respetivos processos para futura manutenção e atualização dos testes se processará de modo eficiente e completo, em conformidade com a evolução do software;

1.4 Planeamento

Nesta secção vai ser descrito o planeamento de tarefas que foram desenvolvidas durante o projeto.

Durante os primeiros meses, isto é, até ao fim de fevereiro, foi feita a análise do estado da arte, a integração com a equipa e enquadramento com o projeto ION.

Durante o mês de março foram desenvolvidos e documentados os testes *End to End* para o ION 1. Em simultâneo começou o desenvolvimento experimental do ION 2 com as *frameworks* Angular e Laravel [3][4]. Após um mês de desenvolvimento no ION 2 foi decidido alterar o foco dos testes para o ION 2.

No fim de abril houve uma nova mudança estratégica da parte da empresa em substituir a *framework* de *frontend*, o Angular, pelo Vue 3 [5]. Como tal, foi feito um esforço inicial na aprendizagem da nova *framework* e no desenvolvimento da aplicação. Por esse motivo, o desenvolvimento dos testes aplicacionais só foi reiniciado em julho.

Os períodos de desenvolvimento e documentação de cada tipo de testes e todo o processo anterior pode ser observado na figura 1.

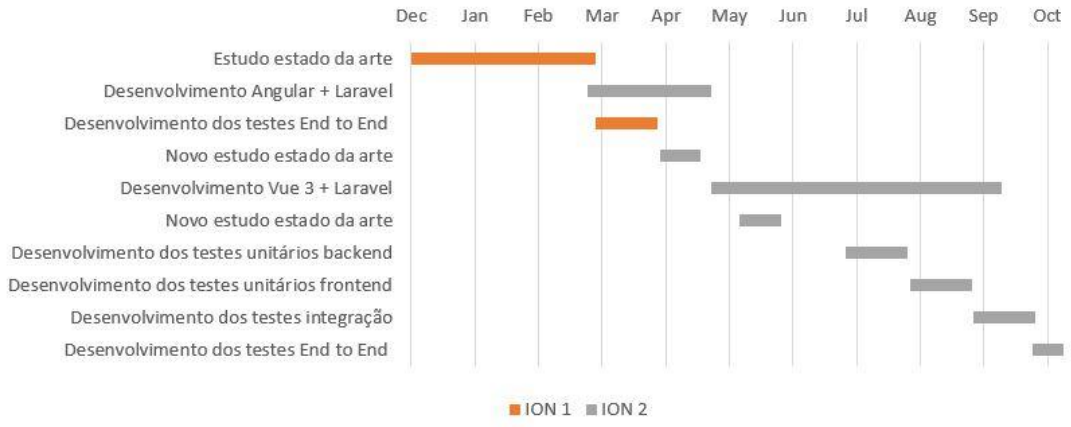


Figura 1 - Diagrama de Gantt

1.5 Estrutura do documento

Para além da introdução, este relatório inclui mais 8 capítulos. No capítulo 2 são descritas as duas aplicações ION 1 e ION 2, com foco na arquitetura, ferramentas utilizadas e quais as razões da transição. No capítulo 3 é descrito o estado da arte com foco nas metodologias utilizadas.

No capítulo 4 é feita uma análise de valor do projeto através de uma proposta de valor e de um diagrama FAST (*Function Analysis System Technique*). No capítulo 5 é feita a analise e desenho da solução com foco nas escolhas de tipos de testes e ferramentas. No capítulo 6 é feita uma contextualização do desenvolvimento do ION 2, do planeamento ao longo do processo e do estado atual da aplicação. No capítulo 7 é descrita a implementação da solução com um subcapítulo para cada tipo de testes a desenvolver. No capítulo 8 é descrita a abordagem e resultados do método de avaliação. Por fim no capítulo 9 são expostos os objetivos alcançados, não alcançados, dificuldades e uma apreciação final do projeto.

2 Análise das aplicações

Neste capítulo inicialmente é realizada uma análise sobre a aplicação ION 1 em relação à sua arquitetura e fluxo de trabalho. De seguida é descrita a metodologia de testes e os problemas existentes no ION 1. Esta descrição serve para contextualizar a realidade da empresa e a necessidade da transição para o ION 2.

De seguida é feita uma análise ao ION 2 em relação à arquitetura, ferramentas escolhidas e melhorias em comparação com os problemas presentes no ION 1.

2.1 ION 1

O ION é uma aplicação web e mobile que está dividida em 3 projetos distintos. Na figura 2 está representada a arquitetura da aplicação de num diagrama de componentes [6].

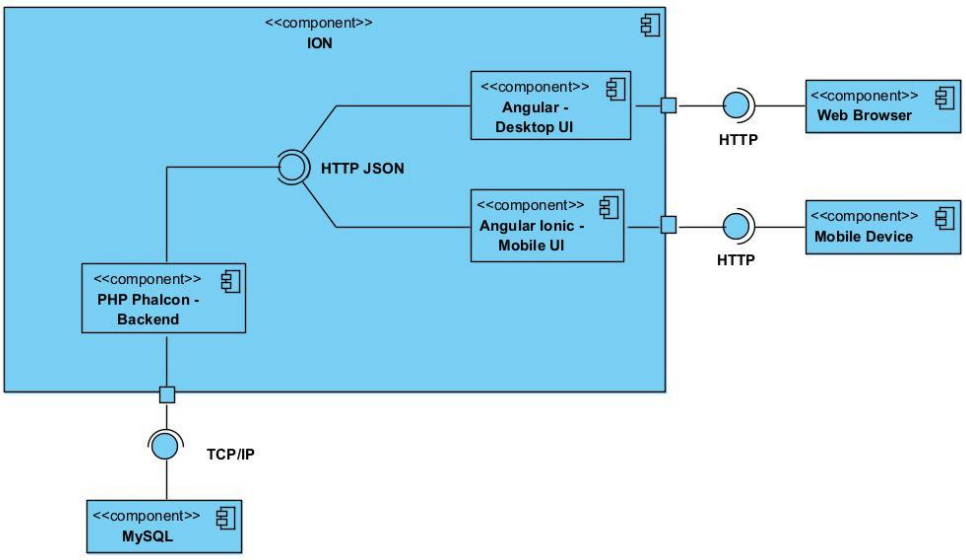


Figura 2 - Diagrama de componentes do ION 1

Como representado no diagrama, existem dois componentes de *frontend* ambos escritos em Angular. O primeiro é destinado ao uso num *web browser* enquanto o segundo é criado através da framework Ionic para utilização em dispositivos móveis [7]. O Ionic é uma *framework* de desenvolvimento de aplicações móveis híbridas. Uma das suas vantagens é permite desenvolver aplicações móveis utilizando tecnologias de desenvolvimento web como o Angular, o HTML e o CSS.

O terceiro projeto é o *backend* que está desenvolvido em PHP Phalcon e fornece uma REST *(Representational State Transfer)* API *(Application Programming Interface)* aos dois projetos de *frontend* através de uma ligação HTTP (*Hypertext Transfer Protocol*) e conteúdo em formato JSON (*JavaScript Object Notation*). O *backend* vai buscar informação a uma base de dados relacional em MySQL.

No ION os desenvolvimentos são geridos em sprints semanais, com *release* estável habitualmente num período de duas a quatro semanas. As instâncias dos clientes (em produção) são atualizadas em conformidade, via git, e podem incluir atualizações na base de dados.

2.2 Metodologia de testes no ION 1

Anteriormente nenhum dos projetos do ION tinha algum teste ou metodologia de testes automáticos. A única *framework* de testes presente no sistema era a Jasmine, que apenas existe nos projetos de *frontend* porque é criada por predefinição no Angular, porém a Jasmine nunca é utilizada [8].

Dado esta falta de testes automáticos, sempre que acaba um sprint de desenvolvimento, antes de a nova versão ser disponibilizada aos clientes, há sempre a necessidade de realizar um processo demorado e falível de testes manuais.

Consequentemente na DouroECI existe uma colaboradora especializada na qualidade do software, ou seja, uma das funções principais passa por testar e documentar todas as novas funcionalidades ou bugs encontrados. Para tal a colaboradora para além do contacto constante com a equipa de desenvolvimento e com os clientes, também necessita ter conhecimentos sobre a área de negócio e sobre a aplicação.

2.3 Problemas do ION 1

O desenvolvimento do ION 1 começou em 2017 através de *outsourcing*, ou seja, foi contratada uma empresa externa para o desenvolvimento da aplicação. Os problemas seguidamente descritos ficaram mais evidentes quando a DouroECI criou a sua própria equipa de desenvolvimento.

A complexidade da arquitetura implementada constitui um dos principais obstáculos. Nalgumas situações a sua implementação não seguiu as boas práticas de programação, e está pouco

documentada. Consequentemente dificulta a integração de novos membros na equipa de desenvolvimento, bem como o desenvolvimento de alterações significativas ao código existente.

Outro grande problema são as *frameworks* utilizadas. No caso do *backend*, o Phalcon é menos utilizado do que outras *frameworks* de PHP e consequentemente cria receio sobre um possível abandono da *framework*. No caso do *frontend* o Angular 7 utilizado está muitas versões atrasado da versão atual (14) e o processo de migração para conseguir aceder as melhorias e funcionalidades adicionadas seria complexo.

Por fim, o desempenho da aplicação, no lado do *frontend*, era inferior ao desejado pela empresa. Este baixo desempenho é causado pela combinação da *framework* utilizada e da arquitetura desenvolvida pela equipa externa. Estes problemas de desempenho não são muito significativos em produção, porém em ambiente de desenvolvimento dificultavam o trabalho do dia a dia da equipa de desenvolvimento.

De seguida segue uma listagem dos principais problemas:

1. Arquitetura complexa com grande interdependência entre módulos;
2. Nomes (variáveis/métodos/base de dados) pouco explicativos e não normalizados;
3. Falta de divisão de responsabilidades, ou seja, existem métodos com demasiadas linhas de código que deviam estar divididos em métodos mais curtos, compreensíveis e com responsabilidades mais concretas;
4. Fraca documentação;
5. Apoio limitado por parte da empresa externa;
6. Baixo desempenho em ambiente de desenvolvimento;
7. *Frameworks* desatualizadas;

2.4 ION 2

O ION 2 é uma aplicação web que está dividida em 2 projetos distintos. Na figura 3 está representada a arquitetura da aplicação num diagrama de componentes.

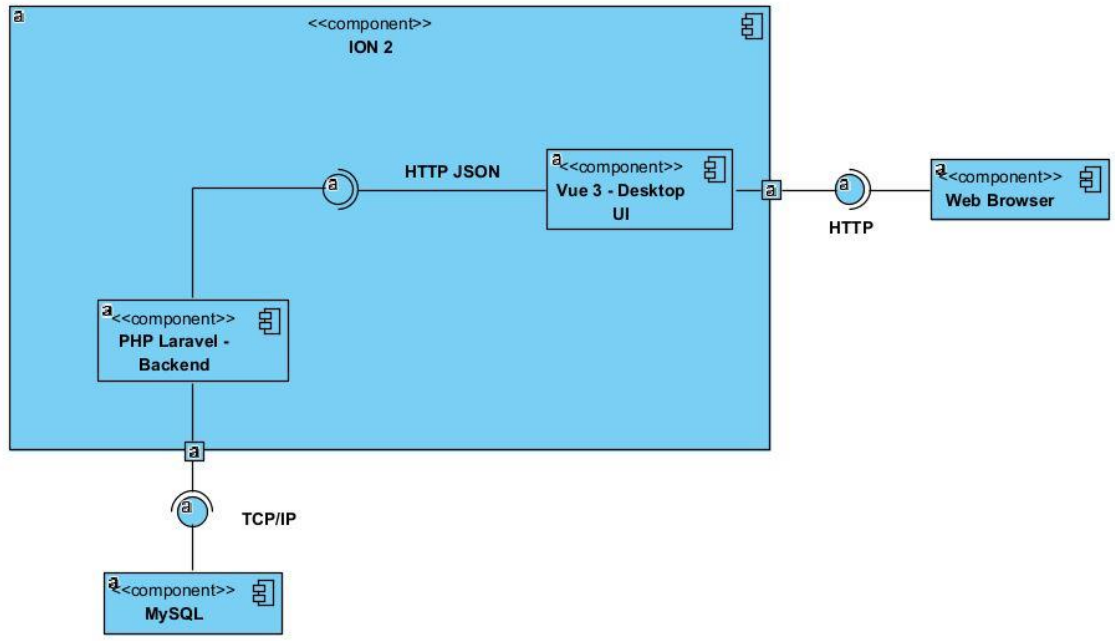


Figura 3 - Diagrama de componentes do ION 2

Como pode ser visualizado no diagrama, atualmente existe apenas um componente de *frontend*. Este componente foi desenvolvido em Vue 3 e é destinado apenas ao uso num *web browser*. A componente para dispositivos móveis é um projeto a começar brevemente.

O componente de *backend* está desenvolvido em PHP Laravel e, tal como o ION 1, fornece uma REST API ao componente de *frontend* através de uma ligação HTTP e conteúdo em formato JSON. O *backend* vai buscar informação a uma base de dados contruída em MySQL. Esta base de dados é alimentada a partir de um *script* de migração complexo de forma a normalizar a base de dados existente no ION 1 sem perder dados de cada cliente na transição.

2.5 Melhorias do ION 2

Em seguida são descritas as várias melhorias que o ION 2 trouxe à equipa de desenvolvimento para a resolução dos problemas descritos em 2.3.

Em relação ao primeiro problema identificado para o ION 1, a adoção de uma arquitetura de acordo com as boas práticas, reduziu significativamente problemas existentes no ION 1 como a interdependência dos módulos.

Em relação ao segundo problema, nomes dos identificadores, foram definidas e documentadas regras de nomenclatura para todos os nomes no código e na base de dados. Também existe um maior esforço por parte da equipa em criar nomes explicativos e coerentes em todo o código.

Em relação ao terceiro problema, falta de divisão de responsabilidades, a equipa fez um esforço em refletir sobre cada componente ou método criado com o objetivo de repartir caso a divisão de responsabilidades o justifique.

Em relação ao quarto problema, fraca documentação, foram adotadas as seguintes melhorias. Primeiro a equipa começou a utilizar uma plataforma para gestão da documentação, o Confluence, para documentar processos, procedimentos e outras informações técnicas de relevo. Segundo foi criada a norma de duas sessões semanais de revisão de código com a equipa de desenvolvimento. Terceiro foi incentivado a escrita de comentários explicativos no código. Por fim todos os requisitos e *commits* são descritivos e estão normalizados para facilitar consultas, com o apoio dos softwares Jira e Confluence.

Em relação ao quinto problema, baixo desempenho, no ION 2 a utilização da aplicação é significativamente mais rápida e cada compilação demora menos de um segundo, ou seja, são poupados perto de 20 a 50 segundos por compilação (dependendo da máquina).

Por fim, em relação ao último problema, *frameworks* desatualizadas, as duas *frameworks* selecionadas para o desenvolvimento do ION 2 têm grande popularidade e comunidades de contribuidores, com uma utilização massificada no mercado, e consequente expectativa de suporte e melhoria a longo prazo. São também com excelente documentação e recursos para a aprendizagem e suporte.

3 Estado da Arte

Neste capítulo é apresentada inicialmente a análise da metodologia adotada descrevendo cada um dos seis processos. De seguida é analisada a pirâmide dos testes de forma a posteriormente ser feita a escolha do tipo de testes a desenvolver. Por fim é descrita a metodologia SCRUM utilizada no desenvolvimento do ION 2.

3.1 Metodologia do Ciclo de Vida dos Testes Automáticos

A Metodologia do Ciclo de Vida dos Testes Automáticos (Automated Test Lifecycle Metodology, ou ATLM) é uma abordagem que permite organizar e executar testes de forma a aumentar a cobertura do código [2]. Esta metodologia (ver figura 4) é dividida em seis processos que se descrevem em seguida.

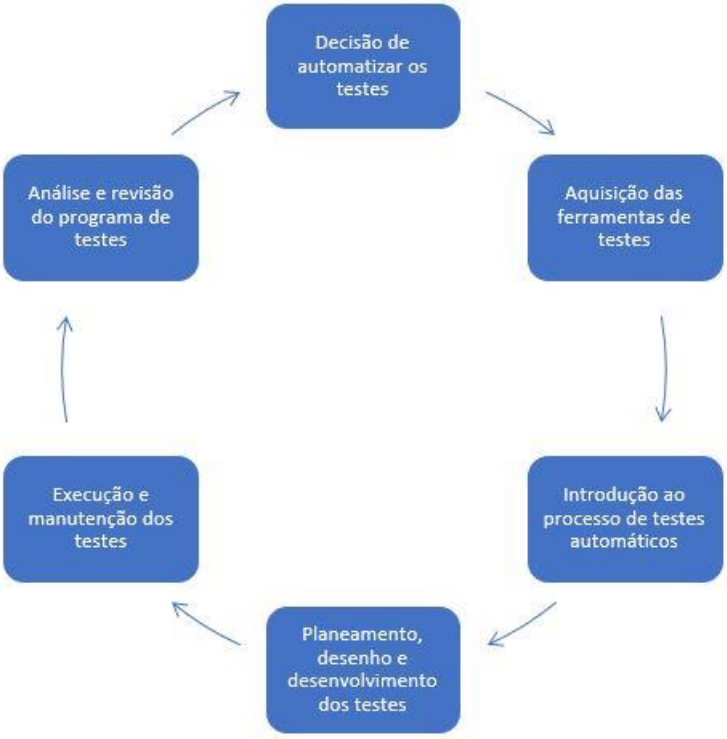


Figura 4 – ATLM

O primeiro processo consiste na decisão de iniciar o processo de criação de testes automáticos, este foca-se em melhorar a compreensão da equipa sobre a automação e em propor uma ferramenta de testes.

O segundo processo é a aquisição das ferramentas de testes. Para este passo deve ser feita a análise dos requisitos da empresa, dos sistemas da empresa e qualquer outro relevante para criar uma lista de critérios de avaliação. As ferramentas são então escolhidas a partir dos critérios definidos e, no caso de ferramentas pagas, o vendedor é contactado para a fornecer.

O terceiro processo é a introdução ao processo de testes automáticos. Este passo esta divido em dois outros passos, análise do processo de testes e a consideração da ferramenta de testes. O primeiro destes passos foca-se em planear uma estratégia global de testes adequada ao projeto. O segundo passo é destinado a analisar se as ferramentas de testes a integrar no projeto são apropriadas e trazem de facto alguma vantagem.

O quarto processo é o planeamento, desenho e desenvolvimento dos testes. O planeamento inclui identificar as normas de desenvolvimento de testes e os resultados dos passos anteriores. Para além disto o planeamento também identifica o hardware, software e infraestrutura necessária para suportar o ambiente de testes preliminar. A criação deste ambiente de testes também faz parte do planeamento. O desenho engloba definir o número de testes, a forma de abordar os testes e as condições de testes que devem ser utilizadas. No desenvolvimento é definido que os testes devem ser reutilizáveis, reproduzíveis e sustentáveis.

O quinto processo é a execução e manutenção dos testes. Todos os planos de testes devem ser executados de acordo com o planeamento previamente realizado e qualquer problema encontrado deve ser documentado.

O sexto e último processo é a análise e revisão do programa de testes desenvolvidos. Este passo deve ser realizado continuamente ao longo de todo o processo de forma a melhorar continuamente a solução e encontrar qualquer falha o mais cedo possível. Ao encontrar falhas o mais cedo possível são reduzidos os custos de resolução em fases posteriores.

3.2 Pirâmide de testes

O processo de escolha dos tipos de teste foi influenciado pelo modelo da pirâmide de testes (figura 5). Este modelo foi criado por Mike Cohn e é usado para mostrar como três tipos de testes se complementam [9]. Estes tipos de testes são os testes unitários, integração e de *End to End*.

Figura 5 - Pirâmide de testes

Começando pelo base da pirâmide estão os testes unitários. Estes tipos de testes são executados em código isolado sem qualquer integração e devido a este fator são muito mais rápidos de executar e precisos a encontrar problemas. Porém, sem as camadas superiores nunca seria possível garantir a funcionalidade das integrações entre os componentes [10].

No meio da pirâmide estão os testes de integração. Estes tipos de testes recaem principalmente sobre testar serviços, APIs ou módulos externos. Devido a não dependerem da UI estes testes são mais fáceis de desenvolver, manter e mais rápidos de executar. Por outro lado, quando existe um erro não é sempre fácil encontrar a origem sendo por isso necessário a base da pirâmide [10].

Por fim no topo da pirâmide estão os testes *End to End*. Este tipo de teste passa por todas as camadas da aplicação e mostram a interface tal como o utilizador final a vai ver. Este tipo de

testes oferece resultados rápidos em projetos sem nenhum teste, mas não se deve apenas fazer testes *End to End* pois estes são complexos de desenvolver, de manter e lentos a executar [10].

3.3 Scrum

O Scrum é uma metodologia ágil utilizada para a gestão e planeamento de projetos de software, tendo vários artefactos, funções e eventos incorporados [11].

O Scrum tem três artefactos:

- **Product Backlog:** O *Product Backlog* é uma lista ordenada de todas as tarefas necessárias realizar durante a duração do projeto para alcançar os requisitos pretendidos [11].

- **Sprint Backlog:** O *Sprint Backlog* é um conjunto de elementos do *Product Backlog* destinados ao planeamento de um Sprint [11].

- **Increment:** O *Increment* é o conjunto dos elementos do *Product Backlog* concluídos até à data [11].

A Scrum tem três funções:

- **Product Owner:** O *Product Owner* é a pessoa responsável por maximizar o valor do produto desenvolvido pela equipa de desenvolvimento. Como tal, o papel principal deste membro é a ordenação e descrição de todos os elementos do *Product Backlog* [11].

- **Development Team:** A *Development Team* é a equipa de profissionais responsável por realizar as tarefas definidas em cada *Sprint Backlog* de forma a criar o produto pretendido. A equipa deve ser independente na sua organização interna e não deve existir subequipas dentro desta [11].

- **Scrum Master:** O *Scrum Master* é a pessoa responsável por garantir que o *Product Backlog* está claro para toda a equipa, facilitar a marcação de eventos e por remover possíveis obstáculos da equipa. Esta ação tem por objetivos maximizar o valor da utilização da metodologia Scrum [11].

O Scrum tem cinco eventos:

- **Sprint:** O *Sprint* é um evento onde um conjunto de tarefas são realizadas de forma a criar um producto funcional e, em alguns casos, pronto a passar para produção. O período de cada *Sprint* deve ser constante durante todo o desenvolvimento e cada *Sprint* deve iniciar imediatamente após o anterior acabar. O *Sprint* contém e consiste nos outros eventos mencionados em baixo.

- **Sprint Plannning:** O *Sprint Planning* é um evento no início de cada *Sprint* onde o *Scrum Master* juntamente com a *Development Team* escolhem os elementos do *Product Backlog* a serem desenvolvidos no próximo *Sprint*.

- **Daily Scrum:** O *Daily Scrum* é um evento diário com um período de quinze minutos realizado pela *Development Team*. Este evento tem como objetivo principal a análise dos seguintes pontos: o progresso no dia anterior, o trabalho a ser realizado para o dia atual e os problemas encontrados pela equipa.

- **Sprint Review:** O *Sprint Review* é um evento onde se junta a *Development Team*, o *Scrum Master*, o *Product Owner* e outros interessados. Os objetivos principais desta reunião são:

    - Identificar os objetivos atingidos e não atingidos;
    - Analisar obstaculos encontrados e como foram ultrapassados;
    - Demonstrar o que foi feito ao longo do Sprint;
    - Projetar a data final do projeto a partir do Product Backlog (caso necessário);
    - Discussão dos próximos passos;

- **Sprint Retrospective:** O *Sprint Retrospective* é um evento realizado entre o *Sprint Review* e o *Sprint Planning*, onde a *Development Team* e o *Scrum Master* discutem se o Sprint anterior decorreu como projetado em relação a pessoas, relações, processos e ferramentas utilizadas. Devem também ser nomeados pontos a melhorar e elaborado um plano para fazer essas melhorias.

4 Análise de Valor

Foi realizada uma análise de valor do projeto começando por um *canvas* da proposta de valor. Esta proposta de valor tem por objetivo sintetizar a relação das necessidades do cliente e a forma como o projeto pretende criar valor face estas. De seguida foi desenvolvido um diagrama FAST de forma a representar graficamente as relações entre funções do projeto.

4.1 Proposta de Valor

Nesta secção é apresentada o *canvas* da proposta de valor do projeto de acordo com Alex Osterwalder et al., 2014 [12]. Uma representação deste modelo pode ser observada na figura 6:

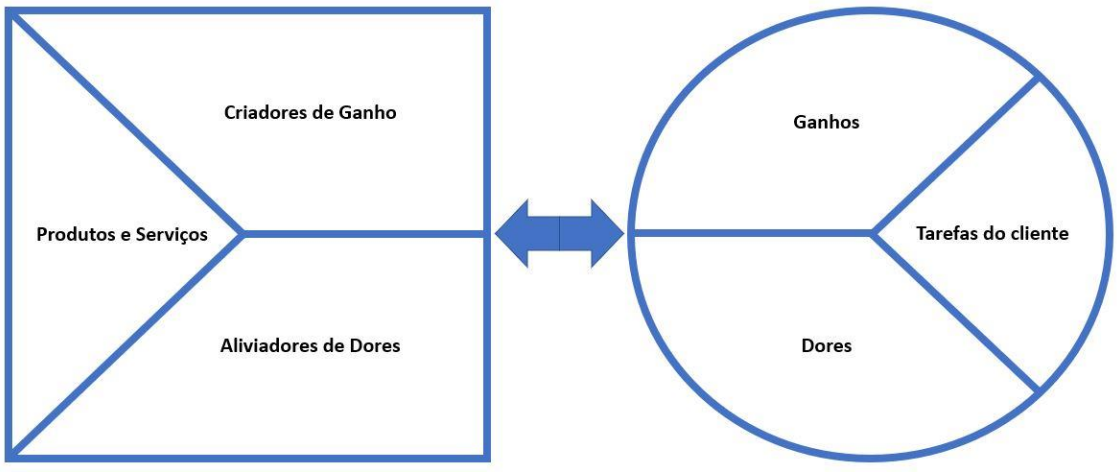


Figura 6 - Canvas da Proposta de Valor

Este *canvas* está dividido em dois, o perfil do cliente na direita e o mapa de valor na esquerda. O perfil do cliente deve descrever características do cliente do ponto de vista de quem está a fazer a proposta. Este lado é constituído por Tarefas do cliente, Dores e Ganhos. O mapa de

valor deve descrever como é que o nosso produto cria valor para o cliente. Este lado é constituído por Produtos e Serviços, Aliviadores de Dores e Criadores de Ganho.

Posto isto, o *canvas* da proposta de valor é o seguinte:

Tarefas do cliente: Os clientes do sistema são os desenvolvedores do ION e têm como objetivo testar a aplicação ION. Os clientes devem ser capazes de desenvolver e executar testes para a aplicação de forma organizada e eficiente.

Dores: Processo demorado e falível de testes manuais, pouca eficiência da equipa e não detetar problemas atempadamente.

Ganhos: Processo mais rápido e facilmente repetível de testes, maior eficiência da equipa, detetar problemas atempadamente e maior confiança no código desenvolvido.

Produtos e Serviços: Um sistema de testes automáticos para a aplicação ION.

Aliviadores de Dores: Torna menos prováveis erros passarem para produção e menos testes manuais a desenvolver.

Criadores de Ganho: Toda a nova metodologia de testes e funcionamento de cada ferramenta está documentada.

4.2 FAST

Nesta secção é apresentada o FAST (*Function Analysis System Technique*) do projeto de acordo com *Charles W. Bytheway*., 1965 [13].

O FAST é uma técnica para desenvolver uma representação gráfica das relações das funções de um projeto, produto ou processo. O FAST tem por objetivo apoiar o processo de reflexão sobre o problema de forma objetiva, conseguindo assim identificar o âmbito do projeto através das ligações entre funções. O FAST também é útil para verificar se a solução proposta atinge os objetivos propostos e para identificar funções desnecessárias, duplicadas ou em falta [14].

O diagrama FAST tem por base as perguntas como (“*How*”), porque (“*Why*”) e quando (“*When*”). A pergunta “Como” é a relação da esquerda para a direita das funções, ou seja, a função a direita de deve responder à pergunta “como posso alcançar esta função” da função à esquerda. Por outro lado, a pergunta “Porque” é a relação em sentido contrário e deve responder à pergunta “porque é que devo desenvolver esta função”. Por fim, a pergunta “Quando” está relacionada com o eixo vertical e corresponde a função que acontecem em conjunto ou por consequência da anterior.

Na seguinte figura 7 está representado o diagrama FAST da função principal deste projeto de automatizar os testes da aplicação.

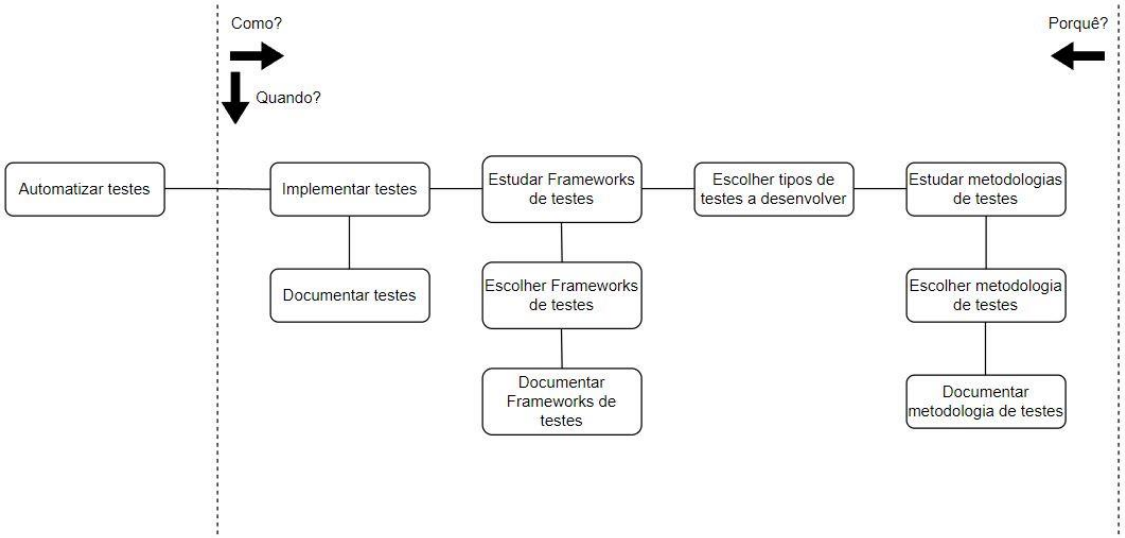


Figura 7 - Diagrama FAST

5 Conceção da proposta

Neste capítulo inicialmente é realizada a escolha dos tipos de teste a desenvolver e uma análise comparativa das ferramentas de testes automáticos para o ION 1. De seguida é feita uma nova análise tendo em consideração a mudança de plataforma para o ION 2. Por fim é feito um breve resumo da solução concebida.

5.1 Escolha dos tipos de teste a desenvolver

As seguintes escolhas foram feitas tendo em consideração a pirâmide de testes descrita no capítulo 3.2.

Para o ION 1, devido tratar-se de uma aplicação já em produção e sem testes automáticos, os tipos de testes a desenvolver a curto prazo para ganhos rápidos são do tipo *End to End*. Porém estava planeado estes serem complementados por testes unitários e de integração sendo o foco sempre que possível testar o nível mais baixo na pirâmide [10].

No caso do ION 2 devido a ser uma nova aplicação foi seguida a ordem ascendente da pirâmide, ou seja, sempre que possível foram desenvolvidos testes unitários no *frontend* e *backend*, testes de integração à REST API disponibilizada pelo *backend* para testar a integração desta com o *frontend* e, por fim, testes *End to End* para os processos mais críticos do sistema.

5.2 Análise Comparativa de Ferramentas de Automação do ION 1

Neste subcapítulo é feita a análise relativa ao ION 1 previamente ao início do desenvolvimento do ION 2. Nas próximas subsecções vão ser analisadas, comparadas e escolhidas as ferramentas a utilizar para cada tipo de teste definidos anteriormente.

5.2.1 Ferramentas de Testes unitários do Backend

As ferramentas a serem analisas nesta secção têm por objetivo permitir desenvolver os testes unitários do *backend* do ION 1. Todas estas ferramentas devem ser *Open Source* e ter uma boa documentação disponível.

Foram analisadas três *frameworks* para a codificação e execução destes testes:

- PHPUnit [15]
- Codeception [16]
- Atoum [17]

5.2.1.1 PHPUnit

O PHPUnit é uma *framework* de testes unitários em PHP com uma arquitetura xUnit. O PHPUnit foi das primeiras *frameworks* de testes unitários em PHP tendo sido criada há vinte anos, como tal é a *framework* das analisadas com mais documentação e informação relacionada.

5.2.1.2 Codeception

O Codeception é uma *framework* de testes unitários, integração e *End to End* em PHP. Devido ao Codeception utilizar o PHPUnit no seu *backend* é possível migrar qualquer teste de PHPUnit para Codeception diretamente.

5.2.1.3 Atoum

O Atoum é uma *framework* semelhante ao PHPUnit dedicada a realizar testes unitários em PHP. Alguns fatores diferenciadores do Atoum são a facilidade de leitura e realizar testes em paralelo.

5.2.2 Comparação e Escolha da Ferramenta de Testes unitários do Backend

Um dos fatores na escolha entre estas ferramentas foi a documentação e informação relacionada disponível. A comunidade do PHPUnit e do Codeception é maior do que o Atoum, como tal é mais difícil ter este tipo de suporte [18].

Quanto à forma de escrever testes o Codeception e o Atoum têm mais liberdade podendo utilizar BDD (*Behavior-Driven Development*) enquanto o PHPUnit não tem esta opção.

Quanto à execução dos testes todas as ferramentas permitem a execução através da linha de comandos e apenas a Atoum não tem uma integração com o *IDE* (*Integrated Development Environment*) utilizado na empresa (*Visual Studio Code*) [19][20]. Estes plugins melhoram o processo de criação de teste no dia a dia.

Quanto à paralelização do código apenas o Atoum tem esta funcionalidade por defeito. Não é impossível realizar testes em paralelo com o PHPUnit e o Codeception, porém é necessário utilizar outras ferramentas auxiliares como o Docker [21].

Por fim todas as ferramentas permitem executar testes unitários e com algumas limitações testes de integração. O objetivo atual destas ferramentas é apenas realizar testes unitários,

porém a maior variedade de tipos de testes disponível no Codeception, em específico os testes de API, podem ser uteis. Os testes da API vão ser elaborados com uma das ferramentas do ponto 3.5.5 que são mais especializadas neste tipo de teste, porém para executar testes exploratórios o Codeception é uma alternativa útil.

Os vários critérios utilizados para a escolha da ferramenta de testes unitários do *backend* estão organizados na seguinte tabela 1.

Tabela 1 - Ferramentas de testes unitários do Backend

	PHPUnit	**Codeception**	**Atoum**
Documentação	**Disponível**	**Disponível**	**Disponível**
Open Source	**Sim**	**Sim**	**Sim**
Pode utilizar BDD	**Não pode**	**Pode**	**Pode**
Execução	**Linha de comandos e IDE**	**Linha de comandos e IDE**	**Linha de comandos**
Testes em paralelo	**Não**	**Não**	**Sim**
Tipo de testes	**Unitários e integração**	**Unitários, integração e API**	**Unitários e integração**

Posto isto, a escolha final foi o Codeception. Esta escolha foi feita principalmente devido à maior extensão e flexibilidade da ferramenta, a facilidade de uso com o plugin no IDE e a documentação disponível.

O Atoum foi considerado principalmente pela utilização de testes em paralelo e a provável melhor desempenho consequente, no entanto ao menor número de utilizadores e consequente falta de informação causou o abandono da *framework*. Outro fator a ter em conta é o *backend* raramente ter novas funcionalidades e como tal não ser tão importante a desempenho deste tipo de testes.

5.2.3 Ferramentas de Testes unitários e integração do Frontend

As ferramentas a serem analisas nesta secção têm por objetivo permitir desenvolver os testes unitários do *frontend* e alguns testes pontuais de integração. Todas estas ferramentas são *Open Source* e as mais utilizadas nos últimos anos tendo por isso uma boa documentação e suporte disponível na internet.

Foram analisadas três *frameworks* para a codificação e execução destes testes:

- Jasmine [8]
- Mocha [22]
- Jest [23]

De seguida é apresentada uma breve descrição de cada uma das ferramentas:

5.2.3.1 Jasmine

Jasmine é a uma *framework* de testes em JavaScript que funciona bem com o executor de testes *Karma* e é recomendada na documentação do Angular [24]. Esta ferramenta vem por defeito no Angular e como tal, ao contrário das próximas *frameworks*, não necessita de instalação e requer pouca configuração.

5.2.3.2 Mocha

Mocha é uma *framework* de testes em JavaScript que corre no *Node.js* e no browser. Esta ferramenta distingue-se das outras por ser muito flexível e extensível devido aos vários plugins disponíveis.

5.2.3.3 Jest

Jest é uma *framework* de testes em JavaScript desenvolvida a partir do Jasmine com crescente popularidade nos últimos anos e utilizada por empresas como Meta, Spotify, Airbnb e Twitter [25][23]. É uma ferramenta rápida de instalar e configurar, além disso permite várias funcionalidades por defeito como testes paralelos, *mocks* e cobertura do código.

5.2.4 Comparação e Escolha da Ferramenta de Testes unitários e integração Frontend

Quanto à configuração inicial a Mocha é significativamente mais complexa devido ao número de extensões disponíveis que devem ser analisadas. Por outro lado, isto torna esta ferramenta mais extensível e flexível do que as outras.

Em termos de dependências a Jasmine é a única que não tem dependências externas [8]. Tanto o Jest como o Mocha têm dependências externas sendo o caso do Mocha agravado pela utilização de extensões.

Quanto à execução dos testes em paralelo, que aumenta a desempenho em máquinas com mais núcleos do processador, é possível no Jest e no Mocha, mas não no Jasmine.

Outro fator a ter em conta na comparação é uma provável migração do *frontend* de Angular para outra *framework* (como React). Esta futura migração torna a ligação do Jasmine ao Angular menos relevante a longo prazo.

Os vários critérios utilizados para a escolha da ferramenta de testes unitários e integração do *frontend* estão organizados na seguinte tabela 2.

Tabela 2 - Ferramentas de testes unitários e integração do Frontend

	Jasmine	**Mocha**	**Jest**
Documentação	**Disponível**	**Disponível**	**Disponível**
Open Source	**Sim**	**Sim**	**Sim**
Instalação e configuração	**Fácil**	**Media**	**Fácil**
Dependências	**Poucas**	**Muitas**	**Algumas**
Testes em paralelo	**Não**	**Sim**	**Sim**

Posto isto, a escolha final foi o Jest. Esta escolha foi feita principalmente devido à crescente comunidade, à fácil configuração e por permitir executar testes em paralelo.

5.2.5 Ferramentas de testes da API

As ferramentas a serem analisas nesta secção têm por objetivo permitir desenvolver os testes à API de *backend* da aplicação ION 1. Como tal devem ser capazes de enviar pedidos para o API e validar as respostas dos mesmos.

Foram analisadas 3 *frameworks* para a codificação e execução destes testes:

- Postman [26]
- Katalon [27]
- SoapUI [28]

De seguida é apresentada uma breve descrição de cada uma das ferramentas:

5.2.5.1 Postman

O Postman é uma ferramenta que permite testar, desenhar e documentar APIs. O Postman tem quatro planos entre 0€ e 99€ por mês. O plano grátis permite aceder as funcionalidades principais para testar uma API e os planos seguintes aumentam as ferramentas de colaboração, integrações e outras funcionalidades especificas [26].

5.2.5.2 Katalon

O Katalon é uma ferramenta que permite testar não só APIs mas também sites, aplicações móveis e aplicações de computador. O Katalon tem 3 planos entre 0€ e 1319€ por ano. O plano grátis permite aceder as funcionalidades principais para os testes mencionados e os planos pagos melhoram principalmente a escalabilidade e a integração com ferramentas de CI/CD (*Continuous integration/Continuous deployment*) [27].

5.2.5.3 SoapUI

A SoapUI é uma ferramenta que permite testar APIs com três planos entre 0€ e 4435€ por ano [29]. Tal como as ferramentas anteriores o plano grátis permite aceder as funcionalidades principais para testar uma API e o plano pago mais integrações ferramentas de gestão como o Jira, mais funcionalidades para teste e integração com ferramentas de CI/CD [28].

5.2.6 Comparação e Escolha da Ferramenta de testes da API

Os planos pagos foram abandonados devido à maioria das vantagens não serem relevantes perante as necessidades da DouroECI. Um exemplo destas vantagens não utilizadas seria a integração com sistemas CI/CD visto a ION 1 não utilizar esta prática. Outro exemplo é o das funcionalidades de colaboração, que apesar de interessantes não são prioritárias devido a se tratar de uma pequena equipa de desenvolvimento.

Apesar de a curto prazo não estar planeado utilizar nenhum dos planos pagos, é importante permitir a atualização do plano que for escolhido. Neste sentido o Postman tem planos intermédios muito mais acessíveis que a concorrência de 12€ e 29€ por mês (ou 144€ e 348€ por ano) comparado com no mínimo 839€ por ano para o Katalon e 755€ por ano para o SoapUI [26][27][29].

Quanto ao desenvolvimento dos testes no Postman são escritos scripts em JavaScript que é uma linguagem utilizada regularmente pela equipa de desenvolvimento da DouroECI. No Katalon os scripts são escritos com Java ou Groovy. No SoapUI os scripts são escritos em Groovy ou JavaScript.

Os vários critérios utilizados para a escolha da ferramenta de testes da API estão organizados na seguinte tabela 3.

Tabela 3 - Ferramentas de testes da API

	Postman	**Katalon**	**SoapUI**
Preço (por ano)	**0€ a 1188€**	**0€ a 1319€**	**0€ a 4435€**
Open Source	**Sim**	**Sim**	**Sim**
Linguagem	**JavaScript**	**Groovy ou Java**	**Groovy ou JavaScript**

Posto isto, a escolha final foi o Postman. Esta escolha foi feita tendo em conta a facilidade de uso, o elevado suporte disponível na internet e uma possível atualização de plano no futuro.

5.2.7 Ferramentas de testes de testes End to End

As ferramentas a serem analisas nesta secção têm por objetivo permitir desenvolver os testes *End to End* da aplicação ION 1. Como o *frontend* utiliza Angular para o site e Angular com Ionic para a aplicação móvel, é importante a ferramenta escolhida permitir ambos de modo a evitar a utilização de duas ferramentas distintas.

Foram analisadas 4 *frameworks* para a codificação e execução destes testes:

- Protractor [30]
- Cypress [31]
- Playwright [32]
- WebdriverIO [33]

De seguida é apresentada uma breve descrição de cada uma das ferramentas:

5.2.7.1 Protractor

O Protractor é focado em aplicações Angular e AngularJS, como tal é capaz de testar especificidades do Angular sem necessitar de configurações extras. Para além do Angular o

Protractor também é compatível com as principais *frameworks* em JavaScript como React e Vue. Por fim esta *framework* também é compatível com aplicações móveis *Ionic*.

5.2.7.2 Cypress

O Cypress é uma *framework* em rápido crescimento nos últimos anos com um dos pontos mais fortes sendo a facilidade de uso [34]. Tal como a *framework* anterior também é compatível com aplicações móveis *Ionic*.

5.2.7.3 Playwright

O Playwright é a *framework* mais recente das analisadas sendo a primeira versão publica de fevereiro de 2020 [35]. Embora se tratar de uma *framework* muito recente é desenvolvida por alguns dos maiores contribuidores do Puppeteer (outra *framework* de testes) e apoiada pela Microsoft [36]. A nível de comunidade é a mais pequena das *frameworks* analisadas, porém continua em rápido crescimento tendo duplicado o número de downloads semanais no último ano [34].

5.2.7.4 WebdriverIO

O WebdriverIO, em contraste da anterior, já é uma *framework* mais matura e com uma grande comunidade que permite mais suporte na internet. Outro fator distintivo desta *framework* é permitir testar aplicações web e móveis nativas.

5.2.8 Comparação e Escolha da Ferramenta de testes End to End

Todas as quatro ferramentas permitem testes em Ionic porém estão limitadas a testar num sistema simulado no browser sendo por isso impossível testar funcionalidades nativas como a câmara. A única exceção é o WebdriverIO que permite testar aplicações móveis nativas.

Outro fator a ter em consideração é a longevidade da ferramenta. No caso do Protractor foi anunciado que no fim de 2022 este projeto vai deixar de ser suportado e a própria equipa recomenda a migração para outras ferramentas como o Cypress, Playwright e o WebdriverIO [37].

Quanto ao suporte de browsers todas funcionam com Chrome, Microsoft Edge e Firefox. O Cypress apenas suporta estas enquanto as restantes ferramentas suportam Safari, Internet Explorer e até o Opera no caso do Playwright [30][38][32][33]. Apesar de ser importante ter suporte para estes browsers, principalmente para o Internet Explorer, não foi considerado prioritário. Acresce a isto o facto de ser um dos objetivos do Cypress aumentar o número de browsers suportado [38].

Os vários critérios utilizados para a escolha da ferramenta de testes de *End to End* estão organizados na seguinte tabela 4.

Tabela 4 - Ferramentas de testes de End to End

	Protractor	Cypress	Playwright	WebdriverIO
Documentação	**Sim**	**Sim**	**Sim**	**Sim**
Open Source	**Sim**	**Sim**	**Sim**	**Sim**
Fim suporte anunciado	**Sim**	**Não**	**Não**	**Não**
Principais browsers suportados	**Chrome, Microsoft Edge, Firefox, Safari e Internet Explorer**	**Chrome, Microsoft Edge e Firefox**	**Chrome, Microsoft Edge, Firefox, Safari, Internet Explorer e Opera**	**Chrome, Microsoft Edge, Firefox, Safari e Internet Explorer**
Suporta Ionic	**Sim**	**Sim**	**Sim**	**Sim**
Suporta aplicações móveis nativas	**Não**	**Não**	**Não**	**Sim**

Posto isto, a escolha final foi o Cypress. Esta escolha foi feita tendo em conta a facilidade de uso e o elevado suporte disponível na internet.

No caso do Protractor a razão principal de exclusão foi o fim do suporte no futuro próximo. Para o Playwright a razão principal foi o limitado suporte na internet devido a ser uma *framework* mais recente. Por fim o WebdriverIO tem algumas vantagens claras ao Cypress em relação ao suporte de browsers e aplicações móveis nativas, porém estas vantagens não são consideradas tão relevantes como a usabilidade e suporte na internet.

5.3 Análise de Ferramentas de Automação do ION 2

Neste capítulo é feita a análise relativa ao ION 2 tendo por base o trabalho desenvolvido no subcapítulo anterior, ou seja, cada uma das ferramentas escolhidas foram analisadas tendo em consideração novos fatores e novas ferramentas.

5.3.1 Ferramentas de Testes unitários do Backend

No *backend* a mudança de *framework* foi do Phalcon para o Laravel. Devido a ambas as *frameworks* utilizarem PHP todas as ferramentas analisadas continuam válidas e não foram acrescentadas novas ferramentas a analisar.

O Laravel tem suporte para PHPUnit incluído e consequentemente disponibiliza métodos auxiliares e comandos para facilitar o desenvolvimento de testes com esta ferramenta. Por defeito o PHPUnit no Laravel também disponibiliza funcionalidades úteis para testes da API a serem analisados em 3.4.3.

Outro fator importante é a escolha da ferramenta de testes a API, descrita no 3.4.3, pois simplifica a curva de aprendizagem da equipa de desenvolvimento se apenas utilizarem uma ferramenta para os dois tipos de testes.

Outro fator a ter em consideração é, ao contrário do Phalcon, o Laravel permitir a realização de testes com PHPUnit em paralelo diretamente pela linha de comandos.

Posto isto, a escolha final foi o PHPUnit. Esta escolha foi feita tendo em conta a nova flexibilidade da ferramenta em conjunto com o Laravel e a facilidade de realizar testes em paralelo.

5.3.2 Ferramentas de Testes unitários do Frontend

No *frontend* a mudança de *framework* foi do Angular para o Vue 3. Devido a esta mudança a compilação do código passou de Webpack para Vite [39][40]. Devido a estas mudanças foi acrescentada uma nova ferramenta na análise, o Vitest.

O Vitest é uma *framework* de testes em JavaScript nativa em Vite. O Vitest é compatível com o Jest, ou seja, qualquer teste em Jest é facilmente reutilizado com o Vitest. Por fim, tal com o Jest e o Mocha, o Vitest também permite executar testes em paralelo.

Quanto ao Jest, a escolha no ION 1, é apenas possível utilizar em conjunto com o Vite com o *vite-jest*. O *vite-jest* é um projeto com o objetivo de integrar o Jest com o Vite, porém ainda tem limitações e não tem *commits* há mais de dez meses [41].

No evento Porto Vue Meetup existiu uma apresentação relevante para a análise. Nesta apresentação foi descrito o trabalho e resultados de uma equipa de desenvolvimento na transição de um projeto desenvolvido com Webpack e Jest para Vite e Vitest. Um dos pontos iniciais desta apresentação foi o vite-jest que a equipa testou brevemente, considerando o projeto ainda incompleto e como tal abandonando esta ferramenta. De seguida foram descritos casos de testes que depois da transição davam avisos. Estes avisos eram sobre os testes estarem escritos de uma forma pouco rigorosa que, na maioria dos casos, podia causar falsos positivos. A equipa considerou estes avisos uma grande ajuda do Vitest para melhorar a qualidade dos testes previamente escritos em Jest. No fim da apresentação foram comparados os tempos de execução dos testes antes e depois da transição, de onde o Vite com Vitest tinha o menor tempo. Porém, estes resultados não são significativos para a análise pois a alteração principal foi do Webpack para Vite e não da *framework* de testes [42].

Por fim, uma desvantagem do Vitest em comparação com o Jest é a pior desempenho quando existem muitas dependências. Isto acontece porque o Jest agrupa o código dos testes enquanto o Vitest corre cada teste isoladamente [43].

Posto isto, a escolha final foi o Vitest. Esta escolha foi feita em grande parte devido às limitações do Jest em conjunto com o Vite e à semelhança na sintaxe do Vitest com o Jest.

5.3.3 Ferramentas de Testes da API

A escolha no ION 1 para testes da API foi o Postman. Esta escolha continua a ser uma opção válida, porém dois novos fatores foram considerados.

O primeiro fator foi a opinião da equipa de desenvolvimento sobre a quantidade de ferramentas a utilizar para a automação dos testes. Atendendo à metodologia de testes anterior foi considerado demasiado complexo aprender e utilizar quatro ferramentas distintas de testes. De forma a diminuir a quantidade de ferramentas utilizadas, sem perder uma camada da pirâmide, os testes da API devem ser realizados pela mesma ferramenta dos testes unitários do *backend*.

O segundo fator foi a mudança para o Laravel que, tal como foi mencionado em 3.4.1, tem um grande suporte para PHPUnit e disponibiliza funcionalidades úteis para testes da API. Algumas destas funcionalidades são facilidade de testar pedidos HTTP, comparar respostas JSON e métodos para inicializar ou recompor a base de dados durante a execução dos testes.

Posto isto, a escolha final foi o PHPUnit. Esta escolha foi feita tendo em conta a oportunidade de diminuir a quantidade de ferramentas utilizadas e a integração do PHPUnit com o Laravel.

5.3.4 Ferramentas de Testes End to End

Como o Cypress funciona independentemente das *frameworks*, não foi encontrada nenhuma razão para alterar a escolha ou adicionar outras ferramentas a analisar para este tipo de testes.

5.4 Resumo da solução

Inicialmente foi feita a escolha dos quatro tipos de testes a realizar tendo como ponto de partida o modelo da pirâmide de testes.

Após ser definido o tipo de testes a desenvolver foi feita a análise e escolha das ferramentas de testes para o ION 1 que na sua maioria não foram utilizadas.

Por fim, tendo em consideração novos fatores foi feita a análise e escolha das seguintes ferramentas de testes para o ION 2:

- Testes Unitários Backend: *PHPUnit*
- Testes Unitários Frontend: *Vitest*
- Testes da API: *PHPUnit*
- Testes End to End: *Cypress*

6 Desenvolvimento do ION 2

Neste capítulo inicialmente é descrito o contexto do desenvolvimento do ION 2. De seguida são descritas as cerimónias e artefactos utilizados pela equipa de desenvolvimento.

6.1 Contexto

Durante os primeiros meses de desenvolvimento do ION 2 houve um maior foco nas componentes de *frontend*. Tal como foi mencionado em 1.4 o desenvolvimento do ION 2 começou em março com um projeto em Laravel e Angular. Após dois meses de desenvolvimento, o componente em Angular foi abandonado e começou a ser desenvolvido um novo em Vue.

As primeiras semanas de trabalho em ambas as ferramentas, Angular e Vue, foram fases experimentais do projeto. Durante estas fases experimentais existiam *commits* sem *issue*, várias pessoas para o mesmo *issue* e situações de *pair programming, ou seja, várias pessoas para cada commit*. Como tal não é possível documentar detalhadamente estes períodos.

6.2 Planeamento do ION 2

Neste subcapítulo são exemplificados as funções, os artefactos e as cerimónias da equipa durante a realização do projeto.

Em relação às funções o *Product Owner* é um gestor com contacto constante com a equipa e alguns conhecimentos na área de programação. A *Development Team* é constituída por três profissionais, sendo o membro mais sénior o *Scrum Master* da equipa.

Os artefactos que a equipa criou foram:

- ***Product Backlog***:

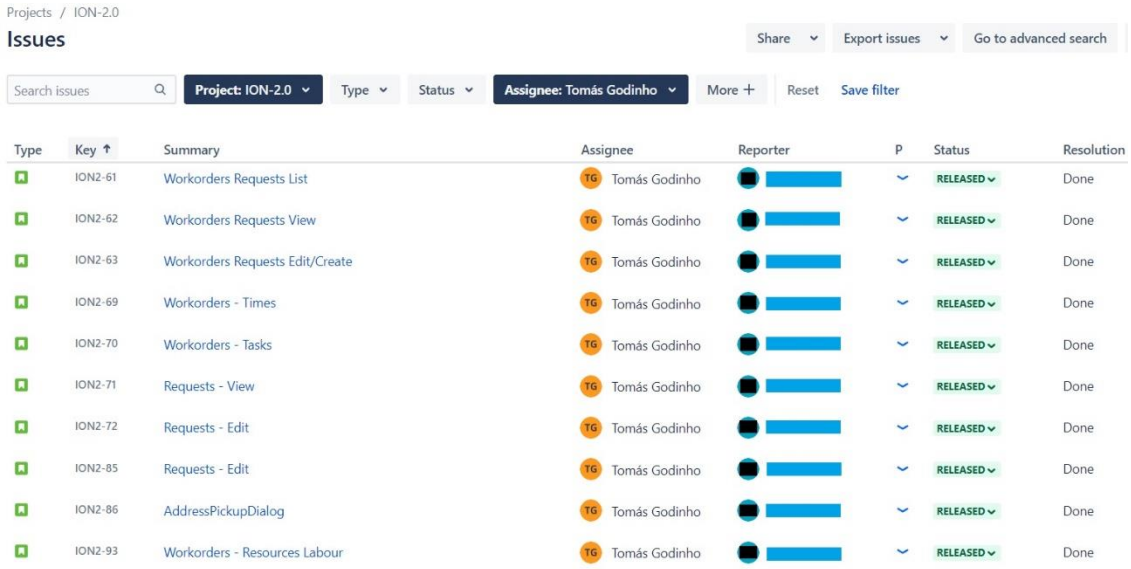


Figura 8 - Product Backlog

- ***Sprint Backlog***:

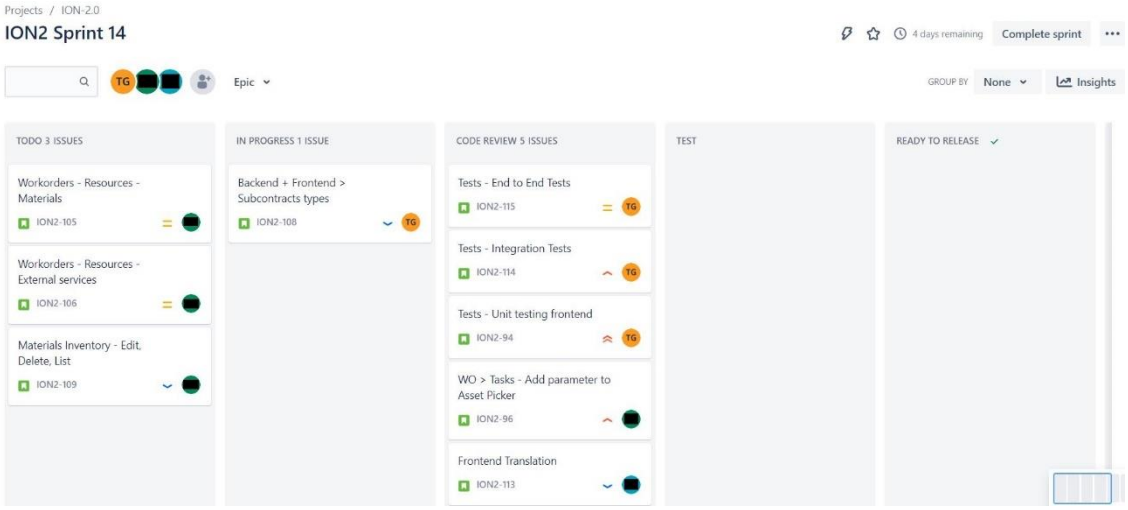


Figura 9 - Sprint Backlog

As cerimónias efetuadas pela equipa eram:

- *Sprints* semanais;
- O *Sprint Planning* na manhã do primeiro dia de cada *Sprint*, com duração máxima de uma hora;
- O *Daily Scrum* ao início do dia, com duração máxima de vinte minutos.

- O *Sprint Retrospective* era feito no início da manhã do primeiro dia do *Sprint*, com duração máxima de meia hora.
- Entre uma e duas reuniões de revisão de código por semana.

6.3 Estado atual do ION 2

O ION 2 continua em desenvolvimento sendo previsto passar a produção nos próximos seis a doze meses. Foram desenvolvidas a maioria das páginas do *frontend* e disponibilizados os controladores para cada entidade do *backend*. Alguns exemplos relevantes estão presentes no anexo A.

Por outro lado, faltam partes importantes como permissões, traduções e o componente de mapas. Para além destas falta também a componente móvel que ainda não começou a ser desenvolvida.

7 Implementação da solução

Este capítulo está dividido em quatro subcapítulos que analisam respetivamente os testes unitários do *backend*, os unitários do *frontend*, os de integração e os *End to End*. Cada um destes subcapítulos começa com os objetivos dos testes, seguidos da configuração e organização de ficheiros. De seguida são descritos os métodos de execução, a metodologia e por fim um teste exemplo. Toda a documentação deste capítulo está também disponível na página *wiki* interna da empresa como o exemplo do anexo B.

7.1 Testes unitários backend

7.1.1 Objetivos

O objetivo dos testes unitários no *backend* é testar todos os ficheiros auxiliares criados e, em casos pontuais, métodos dentro de controladores. Os ficheiros a testar devem mostrar uma cobertura de código superior a 80%.

As razões dos testes unitários estarem focados nos ficheiros auxiliares e não em todo o código do *backend* é a estrutura do mesmo.

Em primeiro lugar muitas dos ficheiros são nativos da *framework* Laravel e visto não terem código customizado não faz sentido testar estes (por exemplo as classes “Handler.php” e “ResponseBuilder.php”).

Para além dos ficheiros da *framework* e dos auxiliares a maioria dos outros ficheiros podem ser divididos em modelo ou controlador.

As classes modelo definem a estrutura da entidade, as relações com outras entidades e como construir as *queries* base (“list” e “listAll”). Por outro lado, os controladores definem as funcionalidades a ser expostas pela *backend*.

Tanto os métodos do modelo como os dos controladores têm pouca lógica necessária testar pois estes limitam-se a utilizar os métodos auxiliares criados e os métodos nativos do Laravel. Para além de terem pouco lógica para testar todos estes métodos estão também muito interligados à base de dados, ou seja, qualquer teste unitário teria de simular o comportamento da maioria do método.

Por fim as classes de modelo e controlador vão ser testadas a nível de integração no ponto 7.3 utilizando a mesma ferramenta, o PHPUnit.

7.1.2 Configuração

Para o desenvolvimento destes testes não foi necessário alterar a configuração inicial do Laravel ou instalar novas dependências.

7.1.3 Organização de ficheiros

Todos os testes unitários do *backend* devem ser colocados na pasta "tests/Unit" predefinida pelo Laravel. Todos os ficheiros de teste devem utilizar o nome do ficheiro a testar mais "Test" (por exemplo o teste do ficheiro "FilterQueryBuilder.php" fica "FilterQueryBuilderTest.php").

Caso futuramente a quantidade de ficheiros a testar aumente drasticamente pode fazer sentido subdividir em pastas de cada tópico, por exemplo auxiliar a *queries*.

7.1.4 Métodos de execução

O método de execução dos testes em PHPUnit é pela linha de comandos através do comando base "php artisan test", através do qual são executados todos os testes do PHPUnit (unitários e integração).

Dentro do comando base existem várias opções adicionais úteis, como por exemplo, "php artisan test --testsuite=Unit", "php artisan test --parallel" e "XDEBUG_MODE=coverage php artisan test --coverage --min=80".

A primeira opção executa apenas os testes da pasta "tests/Unit". A segunda opção executa todos os testes em paralelo. Por fim a terceira opção mostra a cobertura do código e a partir do "min" define um limite mínimo de cobertura.

7.1.5 Metodologia

Para o desenvolvimento dos testes unitários do *backend* podem ser utilizados métodos de preparação que executam antes ou depois dos testes. O exemplo mais utilizado é o "beforeEach" que executa um bloco de código antes da execução de cada teste individualmente. O excerto de código seguinte exemplifica como utilizar este método.

```
beforeEach(function () {
    parent::setup();
    //Setup code here
});
```

Para além deste método também existem outros que podem ser uteis como o “beforeAll”, o “afterEach” e o “afterAll”.

Para a criação dos testes é utilizado um formato semelhante ao seguinte código.

```
test('Teste Exemplo', function () {
    $expectedResult = 'expected';
    $result = method();
    expect($result)->toBe($expectedResult);
});
```

Um exemplo mais detalhado com valores reais está presente no próximo ponto 7.1.6.

7.1.6 Teste Exemplo

O seguinte teste é do método “applyFilters” da classe “FilterQueryBuilder” que ao receber uma *query* base e um conjunto de parâmetros constrói uma *query* com filtros e ordem definidos.

Antes do teste devem ser inicializadas duas variáveis no método de preparação “beforeEach” da figura 10.

```
beforeEach(function () {
    parent::setup();
    $this->faker = Faker\Factory::create();

    $this->workorders = Workorder::from('workorders AS w')
        ->select(···
        )
        ->leftjoin('workorders_priorities AS wp', 'wp.id',  '=', 'w.priority_id')
        ->leftjoin('workorders_schedules AS ws',  'ws.id',  '=', 'w.schedule_id')
        ->leftjoin('workorders_status AS wst',    'wst.id', '=', 'w.status_id')
        ->leftjoin('workorders AS sw',            'sw.id',  '=', 'ws.workorder_id')
        ->leftjoin('workorders AS wb',            'wb.id',  '=', 'w.born_from_workorder_id')
        ->leftjoin('users AS u',                  'u.id',   '=', 'w.responsible_user_id')
        ->leftjoin('users AS uc',                 'uc.id',  '=', 'w.created_user_id')
        ->leftjoin('symptoms AS s',               's.id',   '=', 'w.symptom_id')
        ->leftjoin('interventions AS i',          'i.id',   '=', 'w.intervention_id')
        ->leftjoin('assets AS a',                 'a.id',   '=', 'w.asset_id')
        ->leftjoin('addressbook_districts AS ad', 'ad.id',  '=', 'w.district_id')
        ->leftjoin('addressbook_counties AS ac',   'ac.id',  '=', 'w.county_id')
        ->leftjoin('addressbook_parishes AS ap',  'ap.id',  '=', 'w.parish_id')
        ->leftjoin('addressbook_streets AS as',   'as.id',  '=', 'w.street_id')
        ->leftjoin('requests AS r',               'r.id',   '=', 'w.request_id');

    $this->A_map_filters = [
        'code'     => 'w.code',
        'created_user_id' => 'w.created_user_id',
        'symptom_id' => 'w.symptom_id',
        'intervention_id' => 'w.intervention_id',
        'asset_id' => 'w.asset_id',
        'status_id' => 'w.status_id',
        'priority_id' => 'w.priority_id'
    ];
});
```

Figura 10 - Método de preparação beforeEach

A primeira variável inicializada no método da figura 10 é a “workorders” que guarda a *query* base para todos os testes do ficheiro. A segunda variável é a “A_map_filters” que guarda um mapa auxiliar utilizado para a conversão de nomes no método “applyFilters”.

Na seguinte figura 11 está o teste “Query with filters” que verifica se o método “applyFilters” aplica corretamente o filtro enviado por parâmetro.

```
test('Query with filters', function () {
    $createdQuery = FilterQueryBuilder::applyFilters($this->workorders, array('asset_id=100'), null, [], $this->A_map_filters);
    $queryWithParam = Str::replaceArray('?', $createdQuery->getBindings(), $createdQuery->toSql());
    $expectedResult = "select `w`.`id`, `w`.`code`, `wst`.`name` as `status`, `w`.`status_date`, `w`.`priority_id`, `wp`.`name`
    expect($queryWithParam)->toBe($expectedResult);
});
```

Figura 11 - Teste unitário no backend

Na primeira linha é executado o método a testar com as duas variáveis definidas no “beforeAll” e um filtro como parâmetros. Na segunda linha é guardado o texto da *query* criada na variável “queryWithParam”. Na terceira linha é definido o resultado esperado com o filtro definido. Por fim na última linha é comparado o valor esperado com o valor calculado pelo método.

7.2 Testes unitários frontend

7.2.1 Objetivos

O objetivo dos testes unitários no *frontend* é testar todos os ficheiros criados de forma a alcançar uma cobertura de código superior a 80%.

7.2.2 Configuração

Para o desenvolvimento destes testes foi necessário fazer algumas alterações à configuração inicial do Vue 3.

Primeiro foram adicionadas duas dependências ao ficheiro "package.json", o "happy-dom" e o "msw". O happy-dom é um dos ambientes de teste recomendados pelo Vitest para executar os testes num browser sem interface gráfica [44]. O MSW (Mock Service Worker) é uma API para intercetar pedidos tal como vai ser demostrado em 7.2.5 [45].

De seguida foi acrescentado o seguinte excerto de código ao ficheiro de configuração "vite.config.ts":

```
test: {
    environment: 'happy-dom',
    clearMocks: true
  },
```

A primeira opção "environment" tal como foi mencionado anteriormente define o browser onde os testes vão ser executados. A segunda opção "clearMocks" limpa o histórico dos *spies*, conceito explicado em 7.2.5, antes de cada teste.

7.2.3 Organização de ficheiros

Todos os testes unitários do *frontend* devem ser colocados na pasta “tests/unit”. Dentro desta pasta existe a mesma estrutura de pastas do *frontend*, ou seja, se pretender testar um ficheiro da pasta “modules/shared/components/dialogs” então este teste deve ser colocado na pasta “tests/unit/modules/shared/components/dialogs”.

Todos os ficheiros de teste devem utilizar o nome do ficheiro a testar mais “.test.ts” (por exemplo o teste do ficheiro “AddressPickerDialog.vue” fica “AddressPickerDialog.test.ts”).

7.2.4 Métodos de execução

Os métodos de execução dos testes em Vitest são pela linha de comandos através do comando base “npm run test” ou através de uma extensão do IDE.

Através do comando “npm run test” são executados todos os testes e é feito o relatório de cobertura dos ficheiros testados.

Para executar os testes diretamente no IDE utilizado pela empresa, o Visual Studio Code, é utilizada a extensão “Vitest” [46]. Esta extensão permite executar e fazer *debug* aos testes tal como pode ser observado na figura 12.

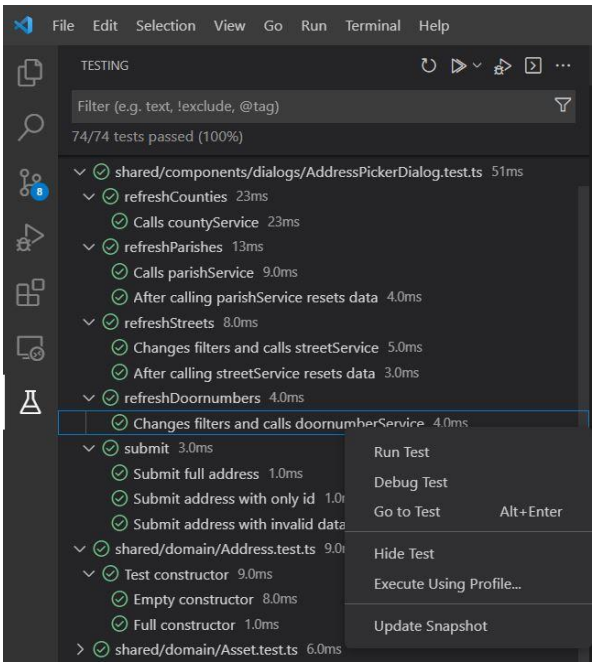


Figura 12 - Extensão Vitest

7.2.5 Metodologia

Para o desenvolvimento dos testes unitários do *frontend* tal como nos do *backend* podem ser utilizados métodos de preparação antes ou depois dos testes. Os métodos utilizados são o "beforeEach", o "beforeAll", o "afterEach" e o "afterAll".

Para a criação dos testes é utilizado um formato semelhante ao seguinte código.

```
describe("Name of Method", () => {
    it("Name of test 1", async () => {
        var expectedResult = 'expected'
        var result = method()
        expect(result).toEqual(expectedResult)
    });
    it("Name of test 2", async () => {
        //
    });
});
```

Um exemplo mais detalhado com valores reais está presente no próximo ponto 7.2.6.

Com a exceção de ficheiros auxiliares todos os ficheiros do *frontend* têm a extensão ".vue" e como tal é necessário criar uma instancia Vue para testar os métodos.

Para a criação de uma instância (ou "wrapper") é utilizado um formato semelhante ao seguinte código.

```
wrapper = shallowMount(ClassName, {
        propsData: {
            propName: propValue
        },
        components: { ExampleComponent }
    })
```

Com este código é criada uma instância da classe "ClassName" com as propriedades e componentes definidos. A opção "shallowMount" é utilizada pois cria uma instância da classe, mas simula as instâncias dos componentes filhos que não devem ser testados.

Devido a serem testes unitários não devem ser executados pedidos ao *backend* do ION 2. Para lidar com estes pedidos foram implementadas duas alternativas, o MSW e os espiões.

A alternativa mais utilizada é o MSW que, tal como foi mencionado em 7.2.2, é uma API para intercetar pedidos. Para a utilização desta API é necessário criar *handlers* e servidores. Os *handlers* definem a resposta pretendida para um certo pedido. Por outro lado, os servidores têm uma lista *handlers* e permitem criar um servidor que espera para intercetar pedidos da lista recebida.

Para a criação de *handlers* é utilizado um formato semelhante ao seguinte código.

```
const ExampleHandler = [
  rest.get('link', (req, res, ctx) => {
    return res(
      ctx.status(200),
      ctx.json({
        data: 'ExampleData'
      }),
    )
  }),
]
```

Para a criação e utilização de servidores é utilizado um formato semelhante ao seguinte código.

```
const server = setupServer(...ExampleHandler)
server.listen()
server.resetHandlers()
server.close()
```

Um exemplo mais detalhado com valores reais está presente no próximo ponto 7.2.6.

Os espiões são nativos do Vitest e ao contrário do MSW simulam métodos e não os pedidos diretamente. A razão de os espiões não serem a alternativa principal deve-se a apenas ser possível criar espiões depois de criar a instância de uma classe. Devido a esta limitação qualquer pedido que seja executado ao abrir a página não é simulado pelo espião.

Porém os espiões são mais eficientes do que o MSW e pode fazer sentido utilizar a simular métodos executados após a criação da instância ou em classes auxiliares.

Para a criação e utilização de espiões é utilizado um formato semelhante ao seguinte código.

```
serviceSpy = vi.spyOn(wrapper.vm.service,'method').mockResolvedValue(value)
expect(serviceSpy).toHaveBeenCalledOnce()
```

7.2.6 Teste Exemplo

O seguinte teste é do método “refreshStreets” da classe “AddressPickerDialog” que utiliza a propriedade “address” para atualizar “pagination.filters” e “streetsList”. No caso do “streetsList” o valor é preenchido através de um pedido ao *backend* que deve ser intercetado pelo MSW.

Antes do teste devem ser inicializadas várias variáveis no método de preparação “beforeAll” da figura 13.

```
beforeAll(async () => {
    server = setupServer(...CountyHandler, ...DoornumberHandler, ...ParishHandler, ...StreetHandler)
    server.listen()
    initialAddress = new Address({ system_id: 1, district_id: 13, district: 'Porto', county_id: 11, county: 'Penaf

    wrapper = shallowMount(AddressPickerDialog, {
        propsData: {
            address: initialAddress
        },
        components: { Dropdown, Button, Dialog }
    })

    responseCounty = [{ district: "Porto", district_id: 13, id: 11, name: "Penafiel", system: "Penafiel Verde" }]
    responseParish = [{ id: 57, name: 'SANTIAGO DE SUBARRIFANA' }, { id: 24, name: 'SÃO MAMEDE RECEZINHOS' }, { id
    responseStreet = [{ id: 3232, name: 'Parque António Soares Moreira\r' }, { id: 3174, name: 'Rua António Huet d
    responseDoornumber = [{ id: 1, name: '10' }, { id: 2, name: '2' }, { id: 3, name: '4' }]
})

afterEach(async () => {
    server.resetHandlers()
    wrapper.vm.pagination.filters = []
    wrapper.vm.countiesList = []
    wrapper.vm.parishesList = []
    wrapper.vm.streetsList = []
    wrapper.vm.doornumbersList = []
    initialAddress = new Address({ system_id: 1, district_id: 13, district: 'Porto', county_id: 11, county: 'Penaf
    wrapper.setProps({ address: initialAddress })
})

afterAll(async () => {
    server.close()
})
```

Figura 13 - Métodos beforeAll, afterEach e afterAll do frontend

No método “beforeAll” é inicialmente instanciado um servidor com quatro *handlers*. De seguida é criada uma morada base, a “initialAddress”, utilizada em vários testes. De seguida é instanciada a classe “AddressPickerDialog” com a estrutura mencionada anteriormente. Por fim são criadas quatro listas com as respostas simuladas do MSW para efetuar as comparações.

Na figura 13 existe também um método “afterEach” onde todas as variáveis alteradas ao longo dos testes voltam ao estado original. O último método da figura 13 é o “afterAll” que fecha o servidor criado no início do “beforeAll”.

Na seguinte figura 14 está o teste “Changes filters and calls streetService” do método “refreshStreets” que verifica se as duas variáveis são alteradas de acordo com o esperado após a execução do método.

```
98  describe("refreshStreets", () => {
99
100     it("Changes filters and calls streetService", async () => {
101         var newAddress = initialAddress
102         newAddress.street_id = 3232
103         wrapper.setProps({ address: newAddress })
104
105         wrapper.vm.refreshStreets()
106         await flushPromises()
107
108         expect(wrapper.vm.pagination.filters).toEqual(['street_id=3232'])
109
110         expect(wrapper.vm.streetsList).toEqual(responseStreet)
111     });
112
```

Figura 14 – Teste unitário do frontend

Nas primeiras três linhas é definida a propriedade “address” que é utilizada dentro do método “refreshStreets”. Na linha 105 é executado o método a testar e na linha seguinte é feito “await flushPromises” que espera pelo resultado do pedido antes de continuar o teste. Por fim são feitas duas comparações, se o filtro foi adicionado a “filters” e se a lista “streetsList” é igual à definida no “beforeAll”.

7.3 Testes integração

7.3.1 Objetivos

O objetivo dos testes de integração é testar todos os ficheiros do tipo controladores que são expostos pelo *backend*. Os ficheiros a testar devem mostrar uma cobertura de código superior a 80%.

Ao testar todos os controladores são também testadas indiretamente todas as classes de modelo, devendo então estas classes também mostrar uma cobertura de código superior a 80%.

7.3.2 Configuração

Tal como nos testes unitários do *backend* não foi necessário alterar a configuração inicial do Laravel do PHPUnit. Por outro lado, devido aos testes de integração interagirem com a base de dados foi decidido criar uma base de dados para estes testes.

Esta base de dados deve ser atualizada em paralelo com a base de dados de desenvolvimento de forma a realizar testes num ambiente o mais semelhante ao de produção possível.

Para a criação da base de dados é utilizado o mesmo processo interno da base de dados de desenvolvimento, mudando apenas o nome para "ion2_testing". O processo refere-se à

utilização de um script interno para a criação das tabelas e inserção dos dados pretendidos nestas.

Após a criação da base de dados foi necessário alterar o ambiente de testes de forma a utilizar uma base de dados diferente. Por defeito os testes são executados no ambiente de desenvolvimento definido através do ficheiro ".env", porém pode ser criado um novo ficheiro especifico para os testes. Para este efeito foi criado o ficheiro ".env.testing" com o mesmo conteúdo do ficheiro ".env" à exceção do parâmetro "DB_DATABASE" que passou a ser "ion2_testing".

7.3.3 Organização de ficheiros

Todos os testes de integração devem ser colocados na pasta “tests/Feature” predefinida pelo Laravel. Todos os ficheiros de teste devem utilizar o nome do ficheiro a testar mais “Test” (por exemplo o teste do ficheiro “EntityController.php” fica “EntityControllerTest.php”).

7.3.4 Métodos de execução

O método de execução dos testes em PHPUnit é pela linha de comandos através do comando base “php artisan test”, através do qual são executados todos os testes do PHPUnit (unitários e integração).

Dentro do comando base existem várias opções adicionais úteis, como por exemplo, “php artisan test --testsuite=Feature” e “XDEBUG_MODE=coverage php artisan test --coverage --min=80”.

A primeira opção executa apenas os testes da pasta “tests/Feature”. A segunda opção mostra a cobertura do código e a partir do “min” define um limite mínimo de cobertura.

7.3.5 Metodologia

Para o desenvolvimento dos testes de integração é utilizado o método de preparação “setUp” que executa um bloco de código antes da execução de cada teste, semelhante ao método “beforeEach” mencionado em 7.1.5. Neste método deve ser instanciado um utilizador para autenticar na API em cada pedido.

Para a criação dos testes é utilizado um formato semelhante ao seguinte código.

```
public function test_methodName(){
        $response = $this->actingAs($this->activeUser)->postJson('link', [
            "data" => "addRequestBodyHere",
        ]);
        $response->assertStatus(200)
            ->assertJsonCount(3, 'data.data')
            ->assertJsonPath('data.total', 10)
            ->assertJsonFragment(["name" => "TestValue_Entity_9"])}
```

No exemplo anterior existe não só a estrutura para fazer um pedido ao API, mas também os quatro tipos de comparações mais utilizados para a resposta a pedidos da API. O primeiro método compara o código de estado HTTP do pedido, o segundo compara o número de objetos num *array* JSON, o terceiro compara o conteúdo de um caminho JSON e o último confirma se um fragmento JSON existe na resposta.

Para a criação de novos registos na base de dados é utilizada uma funcionalidade do Laravel, as fábricas. As fábricas permitem adicionar registos a entidades definidas previamente através de um método definição. Estes métodos usam bibliotecas como o Faker para preencher os vários atributos. O Faker é uma biblioteca que permite gerar variáveis aleatórias dentro de um conjunto de parâmetros como por exemplo, um número aleatório com 9 dígitos [47].

Para a criação de uma fábrica com Faker é utilizado um formato semelhante ao seguinte código.

```
class ExampleFactory extends Factory
{
    public function definition()
    {
        return [
            'name' => $this->faker->company(),
            'address' => $this->faker->address(),
            'phone' => $this->faker->randomNumber(9, true),
            'is_subcontract' => $this->faker->boolean(),
        ];
    }
}
```

Por fim de forma a não preencher a base de dados de testes com registos desnecessários é colocada a linha “use DatabaseTransactions” no início de todos os testes de integração. Esta linha reverte todas as transações executadas na base de dados, ou seja, todos os elementos criados, editados ou apagados voltam ao estado anterior.

7.3.6 Teste Exemplo

O seguinte teste é do endereço “/api/entities/entities” que retorna uma lista de todas as “entities” que seguem as condições definidas.

Antes do teste devem ser inicializadas variáveis no método de preparação “setUp” da figura 15.

```
public function setUp(): void
{
    parent::setUp();
    $this->activeUser = User::factory()->create();

    $entity = Entity::factory()->create([
        "name" => "TestValue_Entity_1",
        "vat" => "123456789",
        "phone" => "987654321",
        "is_subcontract" => true,
        "address" => "TestValue_Address",
    ]);
    $this->entityID = $entity->id;
}
```

Figura 15 - Método de preparação setUp

No método “setUp” é inicialmente criado através de uma fábrica um utilizador para utilizar ao longo dos testes. De seguida é criada também através de uma fábrica uma “Entity” com valores fixos. Por fim é guardado o identificador do registo criado numa variável global.

Na seguinte figura 16 está o teste “test_listAll”.

```php
public function test_listAll()
{
    //Setup entities from TestValue_Entity_2 to TestValue_Entity_10
    Entity::factory()->count(9)
        ->sequence(fn ($sequence) => ['name' => 'TestValue_Entity_' . ($sequence->index + 2)])
        ->create();

    $ipp = '&ipp=3';
    $order = '&order[]=name:desc';
    $filters = '&filters[]=name~TestValue_Entity_';
    $response = $this->actingAs($this->activeUser)->get('/api/entities/entities?page=1' . $ipp . $order . $filters);

    $response->assertStatus(200)
        ->assertJsonPath('success', true)
        ->assertJsonCount(3, 'data.data')
        ->assertJsonPath('data.total', 10)
        ->assertJsonFragment(["name" => "TestValue_Entity_9"])
        ->assertJsonFragment(["name" => "TestValue_Entity_8"])
        ->assertJsonFragment(["name" => "TestValue_Entity_7"]);
}
```

Figura 16 - Exemplo Teste Integração

Nas primeiras três linhas é utilizada a fabrica da “Entity” sequencialmente para criar nove registos numerados de dois a dez com o início de nome igual ao criado no “setUp”.

De seguida são definidas três variáveis, “ipp”, “order” e “filters”. O “ipp” refere-se à quantidade de registos devolvidos (*Items Per Page*), o “order” refere-se à ordem da listagem e o “filters” refere-se ao filtro aplicado na listagem.

De seguida é feito o pedido utilizando as três variáveis criadas e o “activeUser” definido anteriormente.

Por fim é feito um conjunto de comparações da resposta recebida com a resposta esperada tendo em consideração os registos criados e os parâmetros utilizados.

7.4 Testes End to End

7.4.1 Objetivos

O objetivo dos testes *End to End* é testar apenas os processos considerados críticos ao funcionamento do ION.

Para o ION 1 foram criados testes para o início de sessão, as ordens de trabalho, as pavimentações e as requisições.

Para o ION 2 numa fase inicial os testes devem apenas incluir a página de início de sessão e o formulário de ordens de trabalho. Posteriormente à medida que o desenvolvimento do ION 2 evoluir devem ser analisados outros pontos críticos e criados testes conformemente.

7.4.2 Configuração

Para o desenvolvimento destes testes foi necessário fazer algumas alterações à configuração inicial do Cypress. Todas as configurações são definas no ficheiro "cypress.json" da figura 17.

Figura 17 - Ficheiro configuração Cypress no ION 1

Nas primeiras sete linhas do ficheiro são definidos os caminhos para as pastas mencionadas em 7.4.3. De seguida é definido o endereço *web* sobre o qual os testes são executados.

De seguida é definido o vídeo como falso para não gravar vídeos de forma a aumentar performance sendo facilmente alterado caso necessário.

De seguida é definido o "env", ou seja, as variáveis de ambiente que podem ser acedidas ao longo dos testes. Nestas variáveis estão definidos os links para o *backend*, os dados de dois utilizadores com permissões diferentes para logins e os tempos limite a utilizar. Os tempos limite são o tempo que o Cypress deve esperar pela aplicação responder ao pedido, que por predefinição é 4 segundos, porém este valor nem sempre é suficiente e como tal foram definidos três valores de 10, 20 e 40 segundos.

Por fim existe o “retries” que define quantas vezes os testes devem repetir em caso de erro, duas vezes em *run mode* (linha de comandos) e uma vez em *open mode* (interface gráfica). Por defeito está definido como zero repetições em ambas.

Caso seja pretendido correr os testes localmente basta copiar a pasta do Cypress juntamente com o ficheiro “package.json” e correr o comando “npm install” para acabar a configuração.

Caso seja pretendido correr os testes a partir do Docker com o resto do código são necessários os seguintes passos extra.

Primeiro deve ser criado um *container* junto do *container* desenvolvimento ao adicionar o seguinte código ao ficheiro “docker-compose.yml”:

```
cypress:
   build:
     dockerfile: ./cypress/cypress.Dockerfile
     context: ./
   depends_on:
     - debian-dev
   volumes:
     - ../ion/:/home/douro/ion/
   stdin_open: true # docker run -i
   tty: true # docker run -t
```

Para além deste código deve ser criado o ficheiro “cypress.Dockerfile” disponível no anexo C. De seguida para ser possível executar os testes pela interface gráfica foi utilizado o XLaunch [48]. Depois de instalado deve ser executado o comando "export DISPLAY=IP:0.0" substituindo "IP" pelo IP local a ser utilizado. Por fim deve ser ativada a definição “Disable access control” ao executar o XLaunch, deixando este pendente e correndo o comando mencionado no ponto 7.4.4.

Ambas as formas de executar os testes End to End, local e Docker, estão funcionais, porém existem vantagens em executar localmente. Executar o Cypress por Docker não só permite manter o código de desenvolvimento e testes juntos, mas também facilita a evolução para uma possível pipeline de produção.

Por outro lado, o processo de configuração e manutenção é significativamente mais complexo dentro do Docker, a DouroECI não está a planear utilizar pipelines no futuro próximo e a performance é significativamente mais lenta dentro do Docker. Esta diferença é especialmente visível em testes através da interface gráfica que é o modo mais utilizado durante a desenvolvimento dos testes.

7.4.3 Organização de ficheiros

Todos ficheiros relacionados com os testes *End to End* estão na pasta “cypress” predefinida pelo Cypress.

Dentro da pasta “cypress” existem as seguintes sub-pastas:

- **Download** - Guarda ficheiros que foram descarregados no decorrer dos testes;
- **Fixtures** - Ficheiros auxiliares aos testes como imagens ou ficheiros JSON com informação;
- **Integration** - Ficheiros de testes divididos em sub-pastas com os diferentes conceitos de negócio;
- **Plugins** – Contêm o ficheiro "index.ts" onde é possível criar e exportar plugins para modificar o cypress;
- **Screenshots** - Capturas de ecrã guardadas em caso de erro com sub-pastas para representar a localização do teste;
- **Support** - Ficheiros que permitem criar métodos auxiliares;
- **Videos** - Videos guardadas em caso de erro com sub-pastas para representar a localização do teste;

Todos os ficheiros de teste criados na pasta "cypress/integration" devem utilizar o nome da ação a testar em *camel case*, por exemplo "createBasicWorkorder.ts".

7.4.4 Métodos de execução

Existem dois métodos de execução dos testes *End to End*, pela linha de comandos ou através da interface gráfica do Cypress.

Para executar os testes pela linha de comandos é utilizado o comando "npm run cypress:run". Ao executar este comando o Cypress corre na consola com a opção *headless browser*. Neste modo o Cypress apenas mostra os resultados dos testes e em caso de falhas guarda capturas de ecrã (e opcionalmente grava um vídeo do erro).

Para executar os testes com a interface gráfica é utilizado o comando "npm run cypress:open". Ao executar este comando o Cypress abre a interface gráfica onde pode ser escolhido o browser e os testes a executar. Neste modo é possível ver o progresso dos testes no browser e utilizar certas funcionalidades descritas no 7.4.5.

Alguns exemplos dos dois métodos de execução estão presentes no anexo D.

7.4.5 Metodologia

Para o desenvolvimento dos testes *End to End* é utilizado o método de preparação "beforeEach". Neste método é normalmente realizado o início de sessão e caso necessário a criação de variáveis.

Para a criação dos testes é utilizado um formato semelhante ao seguinte código.

```
describe('Create Workorder Validations', () => {
    it('Create Workorder with invalid dates', () => {
        //add tests here
    })
})
```

Um exemplo mais detalhado com valores reais está presente no próximo ponto 7.4.6.

Para a criação de testes é uma boa pratica a utilização de identificadores únicos para testes. Isto é feito de forma a evitar a dependência dos testes em estilos CSS ou outros identificadores que podem ser alterados no futuro. Como tal deve ser criado um identificador “data-testid” nos elementos HTML que se pretende selecionar antes da realização de um teste.

Para a criação de métodos auxiliares, mencionados no ponto 7.4.3, devem ser utilizados os ficheiros da pasta “cypress/support”. No ficheiro “index.ts” da figura 18 devem ser declarados todos os métodos auxiliares.

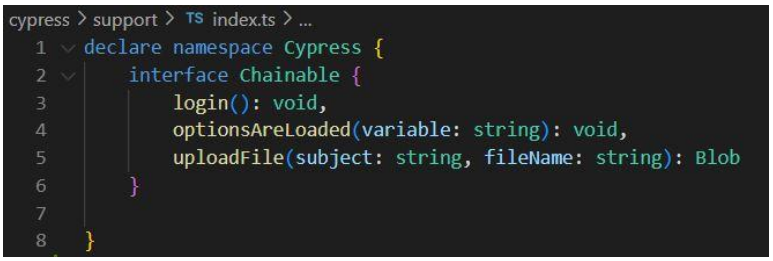


Figura 18 - Declarar métodos auxiliares Cypress do ION 1

Depois de serem declarados a implementação dos métodos auxiliares deve ser colocada no ficheiro “commands.ts” tal como na figura 19.

Figura 19 - Métodos auxiliares Cypress do ION 1

Na figura 19 podem ser observados os três métodos auxiliares criados para o desenvolvimento de testes no ION 1. O primeiro método inicia a sessão no site sem passar na página de início de sessão, de forma a remover essa dependência e aumentar a performance do teste. O segundo método espera até as opções numa *dropdown* estarem preenchidas pelo *backend*. Por fim o último método é utilizado para enviar um ficheiro de teste.

Por fim é importante explicar duas funcionalidades da interface gráfica do Cypress, viajar no tempo para estados passados e o Selector Playground [49].

A funcionalidade de "viajar no tempo" permite voltar para um estado passado do teste e visualizar como é que a página estava nesse momento. Por exemplo num teste da página de início de sessão é possível depois do teste acabar voltar para o estado antes de preencher os campos ou o estado diretamente antes de clicar no botão de iniciar sessão. Um exemplo disto está no anexo D.

O Selector Playground permite em qualquer altura encontrar um *selector* único para um elemento html da página ou o inverso de verificar a localização de um *selector* na página atual. Esta funcionalidade pode ser utilizada em conjunto com a funcionalidade de viajar no tempo para agilizar o processo de criação de testes.

7.4.6 Teste Exemplo

O seguinte teste é da pagina de inicio de sessão e pretende confirmar que o utilizador definido como "testUser" no ficheiro "cypress.json" consegue entrar no site a partir da pagina de inicio de sessão.

Na seguinte figura 20 está o teste "Login into home".

```
describe('Successfull Login', () => {
    it('Login into home', () => {
        var currentUser = Cypress.env('testUser')
        cy.visit('/')
        cy.get("[data-testid=username]").type(currentUser.username)
        cy.get("[data-testid=password]").type(`${currentUser.password}`)
        cy.get("[data-testid=Begin]").click()

        cy.url().should("include", "/home", () => {
            expect(localStorage.getItem("user-session-id")).to.exist
        })
        cy.get('[data-testid="+create"]', { timeout: Cypress.env('bigTimeout') })
        cy.url().should("include", "/dashboard")
    })
})
```

Figura 20 - Teste exemplo End to End

Inicialmente é guardado o “testUser” numa variável local “currentUser” e é visitado o endereço “/” de forma a chegar à página de início de sessão.

De seguida é utilizado o método “cy.get()” para encontrar um elemento a partir do “data-testid” definido para “username” e “password”, que são preenchidos com a informação do “currentUser”. De seguida é encontrado o elemento “Begin” (botão responsável por executar o início de sessão) e é feito um clique.

De seguida é verificado o endereço atual através do “cy.url().should” e se a variável de sessão do utilizador existe no armazenamento local. Por fim é verificado se a página é redirecionada para o “/dashboard” e se esta foi carregada com o elemento “+create”. Esta penúltima verificação tem um “bigTimeout” devido a ser uma transição muito lenta e ao elemento “+create” ser dos últimos elementos a ser apresentado pela aplicação.

Por fim na figura 21 existe um excerto do teste “Create Basic Workorder”.

```
import '../../support/commands'
describe('Create Basic Workorder', () => {
    beforeEach(function () {
        cy.login();
    })

    it('Create Basic Workorder', () => {
        cy.visit('/')
        cy.get('[data-testid="+create"]', { timeout: Cypress.env('mediumTimeout') }).click({ force: true })
        cy.get('[data-testid="headerNewWorkorder"]').click()

        cy.optionsAreLoaded('type')
        cy.get('[data-testid="type"] > .selec2insidecontainerAfter > .form-control').select(0, { force: true })
        cy.get('[data-testid="network_aa_corrective"]', { timeout: Cypress.env('mediumTimeout') }).click()
```

Figura 21 - Exemplo da utilização dos métodos auxiliares do Cypress

Na figura 21 é possível observar a utilização dos métodos auxiliares “login” antes da execução dos testes e “optionsAreLoaded” antes de tentar fazer um “cy.get()” a uma *dropdown*.

8 Experimentação e Avaliação

Neste capítulo inicialmente é descrita a abordagem de avaliação escolhida, o questionário. De seguida são expostos e analisados os resultados do questionário.

8.1 Abordagem

Como método de avaliação foi utilizado um questionário. O questionário foi desenvolvido em Google Forms. O Google Forms é uma ferramenta grátis de criação de questionários *online* disponibilizado pela Google.

O questionário foi escrito em português e tem um total de 26 perguntas, 24 de escolha múltipla obrigatórias e 2 abertas opcionais.

Estas perguntas estão divididas nas seguintes secções:

- **Dados Gerais**: Nesta secção é averiguada informação geral sobre o utilizador como a idade, as habilitações literárias e os anos de experiência profissional na área;
- **Testes Unitários**: Nesta secção é averiguado o nível de conhecimento do utilizador e o seu nível de satisfação sobre os testes Unitários desenvolvidos;
- **Testes Integração**: Nesta secção é averiguado o nível de conhecimento do utilizador e o seu nível de satisfação sobre os testes Integração desenvolvidos;
- **Testes End to End**: Nesta secção é averiguado o nível de conhecimento do utilizador e o seu nível de satisfação sobre os testes End to End desenvolvidos;
- **Avaliação geral**: Nesta secção é averiguado o nível de satisfação sobre a escolha do tipo de testes a desenvolver e sobre metodologia anteriormente utilizada. Também são disponibilizadas duas perguntas abertas, uma sobre a razão de não estar satisfeito com os tipos de testes escolhido (caso se verifique) e uma para comentários ou sugestões;

O questionário na sua integra está disponível no anexo E ou no link:
https://forms.gle/bSYJ77TfXqg82BmF6.

8.2 Participantes

O questionário foi respondido por três pessoas, dois membros da equipa de desenvolvimento e o gestor responsável pela equipa. Todos os membros conhecem bem os projetos ION, a metodologia de testes manuais utilizada anteriormente e a nova metodologia documentada neste documento.

8.3 Análise dos resultados

Os resultados obtidos na primeira secção, Dados Gerais, foram os seguintes:

Tabela 5 - Questionário da secção Dados Gerais

	Pessoa 1	**Pessoa 2**	**Pessoa 3**
Idade	45-54	25-34	45-54
Habilitações Literárias	Ensino Superior	Ensino Secundário	Ensino Superior
Quantos anos de experiência profissional é que tem na área de desenvolvimento de software?	8 ou mais	2 a 3	8 ou mais
Idade	45-54	25-34	45-54

A partir dos resultados da tabela 5 é possível observar que a maioria da equipa tem bastante experiência na área do desenvolvimento do software.

Os resultados obtidos das três secções de testes foram os seguintes:

Tabela 6- Questionário da secção Testes Unitários

	Pessoa 1	**Pessoa 2**	**Pessoa 3**
De 1 a 5, qual é o seu nível de conhecimento sobre testes unitários?	3	2	3
O quão satisfeito está com o processo de execução dos testes unitários no backend?	Muito satisfeito	Muito satisfeito	Muito satisfeito
O quão satisfeito está com a ferramenta de testes escolhida dos testes unitários no backend?	Muito satisfeito	Muito satisfeito	Muito satisfeito
O quão satisfeito está com a metodologia para o desenvolvimento de testes unitários no backend?	Muito satisfeito	Muito satisfeito	Muito satisfeito
O quão satisfeito está com a documentação auxiliar à criação de testes unitários no backend?	Muito satisfeito	Muito satisfeito	Muito satisfeito
O quão satisfeito está com o processo de execução dos testes unitários no frontend?	Muito satisfeito	Muito satisfeito	Muito satisfeito
O quão satisfeito está com a ferramenta de testes escolhida dos testes unitários no frontend?	Muito satisfeito	Muito satisfeito	Muito satisfeito
O quão satisfeito está com a metodologia para o desenvolvimento de testes unitários no frontend?	Muito satisfeito	Muito satisfeito	Muito satisfeito
O quão satisfeito está com a documentação auxiliar à criação de testes unitários no frontend?	Muito satisfeito	Muito satisfeito	Muito satisfeito

Tabela 7 – Questionário da secção Testes de Integração

	Pessoa 1	Pessoa 2	Pessoa 3
De 1 a 5, qual é o seu nível de conhecimento sobre testes de integração?	3	2	3
O quão satisfeito está com o processo de execução dos testes de integração?	Muito satisfeito	Muito satisfeito	Muito satisfeito
O quão satisfeito está com a ferramenta de testes escolhida dos testes de integração?	Muito satisfeito	Muito satisfeito	Muito satisfeito
O quão satisfeito está com a metodologia para o desenvolvimento de testes de integração?	Muito satisfeito	Muito satisfeito	Muito satisfeito
O quão satisfeito está com a documentação auxiliar à criação de testes de integração?	Muito satisfeito	Muito satisfeito	Muito satisfeito

Tabela 8 – Questionário da secção Testes End to End

	Pessoa 1	Pessoa 2	Pessoa 3
De 1 a 5, qual é o seu nível de conhecimento sobre testes End to End?	3	2	3
O quão satisfeito está com o processo de execução dos testes End to End?	Muito satisfeito	Muito satisfeito	Muito satisfeito
O quão satisfeito está com a ferramenta de testes escolhida dos testes End to End?	Muito satisfeito	Muito satisfeito	Muito satisfeito
O quão satisfeito está com a metodologia para o desenvolvimento de testes End to End?	Muito satisfeito	Muito satisfeito	Muito satisfeito
O quão satisfeito está com a documentação auxiliar à criação de testes End to End?	Muito satisfeito	Muito satisfeito	Muito satisfeito

A partir dos resultados das tabelas 6, 7 e 8 é possível observar o mais alto nível de satisfação da equipa de desenvolvimento em todos os aspetos analisados. Também é possível observar que o nível de conhecimentos da equipa sobre três tipos de testes estão ligeiramente abaixo da média.

Os resultados obtidos na última secção, Avaliação Geral, foram os seguintes:

Tabela 9 – Questionário da secção Avaliação Geral

	Pessoa 1	**Pessoa 2**	**Pessoa 3**
O quão satisfeito está com os três tipos de testes escolhidos?	Muito satisfeito	Muito satisfeito	Muito satisfeito
Caso não esteja satisfeito com algum dos três tipos de testes escolhidos, comente o porquê?	Sem resposta	Sem resposta	Sem resposta
O quão satisfeito estava com a metodologia de testes anterior?	Nem satisfeito nem insatisfeito	Muito insatisfeito	Muito insatisfeito
Deixe aqui os seus comentários ou sugestões.	Sem resposta	Sem resposta	Sem resposta

A partir destes resultados é possível observar que a equipa está satisfeita com os tipos de teste escolhidos e que nenhum membro da equipa estava satisfeito com a metodologia anteriormente utilizada.

9 Conclusão

Neste capítulo inicialmente são descritos quais os objetivos alcançados dos definidos na introdução. De seguida são analisados os objetivos não alcançados e as dificuldades encontradas. Por fim é feita uma apreciação final sobre o trabalho realizado.

9.1 Objetivos alcançados

Foram definidos quatro objetivos para este projeto referidos no ponto 1.3 que na sua totalidade foram alcançados.

Relativamente ao primeiro objetivo, foram devidamente identificados problemas com a solução previamente implementada na empresa e aplicadas as melhorias necessárias.

De acordo com o segundo objetivo proposto, ao longo dos diversos capítulos deste trabalho estão expostas as soluções encontradas para os diversos problemas, assim como o diagnostico necessário para cada uma dessas soluções e detalhes relativos à sua implementação.

O terceiro objetivo foi alcançado pois, tal como foi demonstrado no ponto 8.3, houve um grande nível de satisfação da equipa em relação à documentação disponibilizada neste documento e na *wiki* interna da empresa.

O quarto objetivo foi atingido uma vez que todos os processos críticos para a criação e manutenção dos testes foram documentados ao longo do capítulo 7.

9.2 Objetivos não alcançados e adversidades

Apesar de todos os objetivos terem sido alcançados, durante o decorrer do trabalho foram encontradas diversas dificuldades.

O principal obstáculo encontrado foram as duas mudanças de projeto a ser testado. Tal como foi mencionado no planeamento, inicialmente foi estudado e iniciado o desenvolvimento de testes no ION 1. Porém os testes foram redirecionados para um novo projeto em Laravel e Angular. Devido a esta mudança foi necessário voltar a estudar as ferramentas e iniciar lentamente o processo de testes à medida que o *software* evoluía.

A segunda mudança ocorreu passado dois meses, em que se alterou para um novo componente de *frontend*, o Vue 3, abandonando assim o Angular. Adicionalmente, como nenhum membro da equipa tinha experiência em Vue, acrescenta aos problemas descritos para o projeto do Angular também um período de adaptação à nova *framework*.

Apesar destas adversidades, como mencionado anteriormente todos os objetivos propostos foram atingidos e todas as adversidades ultrapassadas.

9.3 Apreciação final

Este projeto permitiu a construção de um artefacto a ser utilizado pela equipa de desenvolvimento da DouroECI para agilizar o processo de criação e execução dos testes unitários, integração e *End to End* para a aplicação ION 2. A partir deste artefacto a equipa pode fazer a mudança para testes automáticos com as vantagens de detetar atempada de erros e a garantir maior confiança no código desenvolvido.

Durante o projeto também foi possível o desenvolvimento da aplicação ION 2, que por sua vez proporciona à empresa diversas vantagens tais como melhor desempenho, melhor documentação e a utilização de *frameworks* que a equipa considera mais intuitivas

10 Referências

[1] "DouroECI." https://www.douroeci.com/ (accessed Aug. 28, 2022).

[2] D. Elfriede, R. Jeff, and P. John, *Automated Software Testing : Introduction, Management, and Performance.* Castleton, New York: Addison-Wesley Professional, 1999.

[3] Angular, "Angular." https://angular.io/ (accessed Sep. 13, 2022).

[4] Laravel, "Laravel." https://laravel.com/ (accessed Sep. 13, 2022).

[5] Vue, "Vue." https://vuejs.org/ (accessed Sep. 13, 2022).

[6] "C4 Model." https://c4model.com/ (accessed Feb. 19, 2022).

[7] ionic, "Ionic Framework." https://ionicframework.com/ (accessed Sep. 12, 2022).

[8] Jasmine, "Jasmine." https://jasmine.github.io/ (accessed Feb. 05, 2022).

[9] M. Cohn, *Succeeding with Agile: Software Development Using Scrum*. Boston, MA: Addison-Wesley, 2009.

[10] J. Rasmusson, *The Way of the Web Tester*. Raleigh, North Carolina: Pragmatic Bookshelf, 2016.

[11] K. Schwaber and J. Sutherland, *The Scrum Guide: The Definitive The Rules of the Game*. 2017.

[12] A. Osterwalder, Y. Pigneur, T. Papadakos, G. Bernarda, and A. Smith, *Value proposition design*. Hoboken, New Jersey: John Wiley & Sons, 2014.

[13] J. R. Wixson, "Function Analysis and Decomposition Using Function Analysis Systems Technique," 1999.

[14] Sundar, “Function Analysis and System Technique – FAST diagram.” https://extrudesign.com/function-analysis-and-system-technique-fast-diagram/ (accessed Feb. 19, 2022).

[15] PHPUnit, “PHPUnit.” https://phpunit.de/ (accessed Sep. 12, 2022).

[16] Codeception, “Codeception.” https://codeception.com/ (accessed Sep. 12, 2022).

[17] Atoum, “Atoum.” https://atoum.org/ (accessed Sep. 12, 2022).

[18] “Github Compare PHP Unit Tests.” https://www.githubcompare.com/atoum/atoum+codeception/codeception+sebastian bergmann/phpunit (accessed Feb. 20, 2022).

[19] Elon Mallin, “PHPUnit.” https://marketplace.visualstudio.com/items?itemName=emallin.phpunit (accessed Sep. 12, 2022).

[20] joelwmale, “VSCode Codeception.” https://marketplace.visualstudio.com/items?itemName=joelwmale.vscode-codeception (accessed Sep. 12, 2022).

[21] Codeception, “Codeception Parallel Execution.” https://codeception.com/docs/12-ParallelExecution (accessed Feb. 06, 2022).

[22] Mocha, “Mocha.” https://mochajs.org/ (accessed Sep. 12, 2022).

[23] Jest, “Jest.io.” https://jestjs.io/ (accessed Feb. 05, 2022).

[24] Angular, “Angular Testing.” https://angular.io/guide/testing (accessed Feb. 03, 2022).

[25] NPM Trends, “NPM Trends Unit Test Frameworks.” https://www.npmtrends.com/jasmine-vs-mocha-vs-jest (accessed Feb. 06, 2022).

[26] Postman, “Postman.” https://www.postman.com/ (accessed Sep. 12, 2022).

[27] Katalon, “Katalon.” https://katalon.com/ (accessed Sep. 12, 2022).

[28] SoapUI, “SoapUI Plans.” https://www.soapui.org/downloads/soapui/ (accessed Feb. 06, 2022).

[29] Smartbear, “SoapUI Pricing.” https://smartbear.com/store/readyapi-test/ (accessed Feb. 06, 2022).

[30] Protractor, “Protractor.” https://www.protractortest.org/#/ (accessed Sep. 12, 2022).

[31] Cypress, “Cypress.” https://www.cypress.io/ (accessed Sep. 12, 2022).

[32] Playwright, “Playwright.” https://playwright.dev/ (accessed Feb. 06, 2022).

[33] WebdriverIO, “WebdriverIO.” https://webdriver.io/ (accessed Sep. 12, 2022).

[34] NPM Trends, “NPM Trends End to End Frameworks.” https://www.npmtrends.com/cypress-vs-protractor-vs-webdriverio-vs-playwright (accessed Feb. 06, 2022).

[35] Playwright, “Playwright Releases.” https://github.com/microsoft/playwright/releases?page=6 (accessed Feb. 06, 2022).

[36] Puppeteer, “Github Puppeteer.” https://github.com/puppeteer/puppeteer (accessed Sep. 12, 2022).

[37] Keen Yee Liau, “Future of Angular E2E & Plans for Protractor.” https://github.com/angular/protractor/issues/5502 (accessed Feb. 06, 2022).

[38] Cypress, “Cypress Launching Browsers.” https://docs.cypress.io/guides/guides/launching-browsers#Browsers (accessed Feb. 06, 2022).

[39] Webpack, “Webpack.” https://webpack.js.org/ (accessed Sep. 18, 2022).

[40] Vite, “Vite.” https://vitejs.dev/ (accessed Sep. 18, 2022).

[41] sodatea, “vite-jest.” https://github.com/sodatea/vite-jest (accessed Sep. 18, 2022).

[42] Porto.Vue, “#11 Porto.Vue Meetup | In-Person Edition.” https://www.meetup.com/porto-vue/events/285859293/ (accessed Sep. 18, 2022).

[43] Joshua Bristow, “Vitest runs tests 3x slower than Jest with threads: true (default) setting.” https://github.com/vitest-dev/vitest/issues/579 (accessed Sep. 18, 2022).

[44] happy-dom, “happy-dom.” https://www.npmjs.com/package/happy-dom (accessed Oct. 12, 2022).

[45] Mock Service Worker, “Mock Service Worker.” https://mswjs.io/ (accessed Oct. 12, 2022).

[46] Zixuan Chen, “Vitest Extension.” https://marketplace.visualstudio.com/items?itemName=ZixuanChen.vitest-explorer (accessed Oct. 12, 2022).

[47] FakerPHP / Faker, “FakerPHP / Faker.” https://fakerphp.github.io/ (accessed Oct. 12, 2022).

[48] marha, “VcXsrv Windows X Server.” https://sourceforge.net/projects/vcxsrv/ (accessed Oct. 12, 2022).

[49] Cypress, “Selector Playground.” https://docs.cypress.io/guides/core-concepts/cypress-app#Selector-Playground (accessed Oct. 12, 2022).

[50] D. Coghlan and T. Brannick, *Doing Action Research In Your Own Organization*, Second. London: Sage, 2005.

[51] P. Escudeiro and J. Bidarra, “Quantitative Evaluation Framework (QEF),” *Conselho Editorial/Consejo Editorial*, p. 16, Feb. 2008.

Anexo A – ION 2

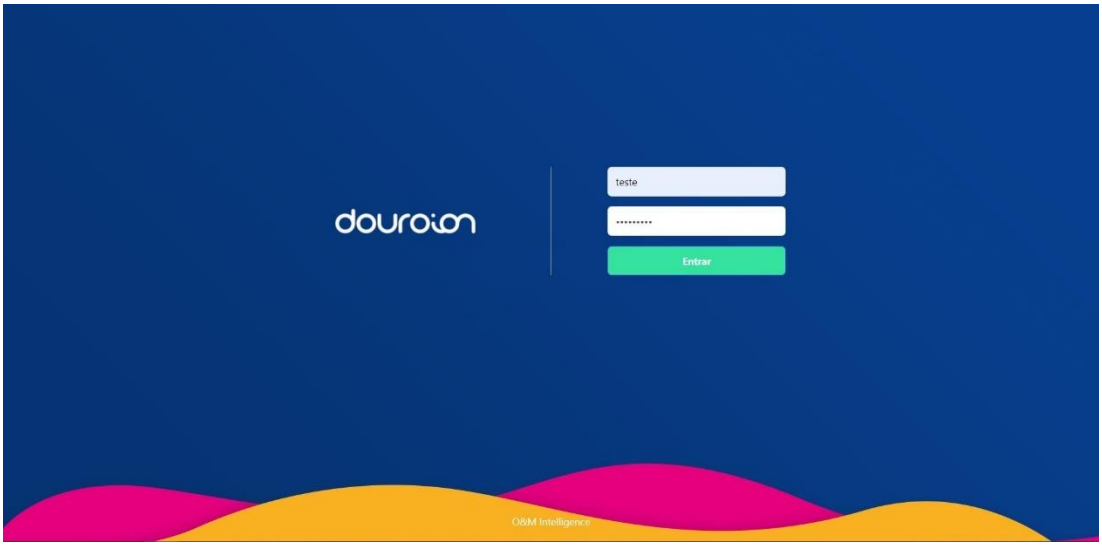


Figura 22 - Login do ION 2

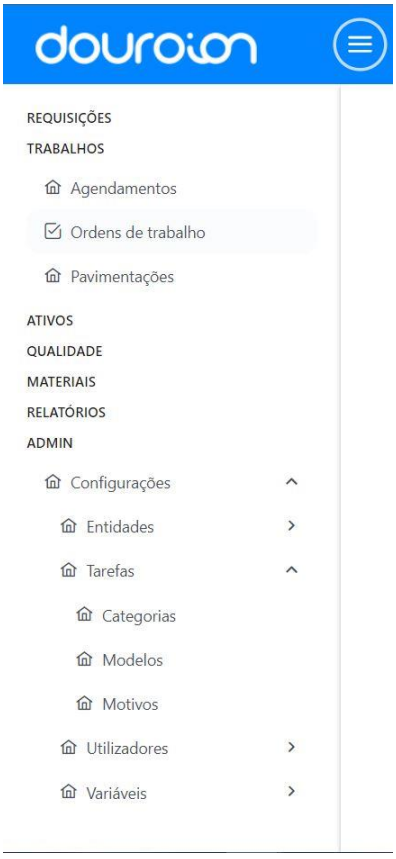


Figura 23 – Menu do ION 2

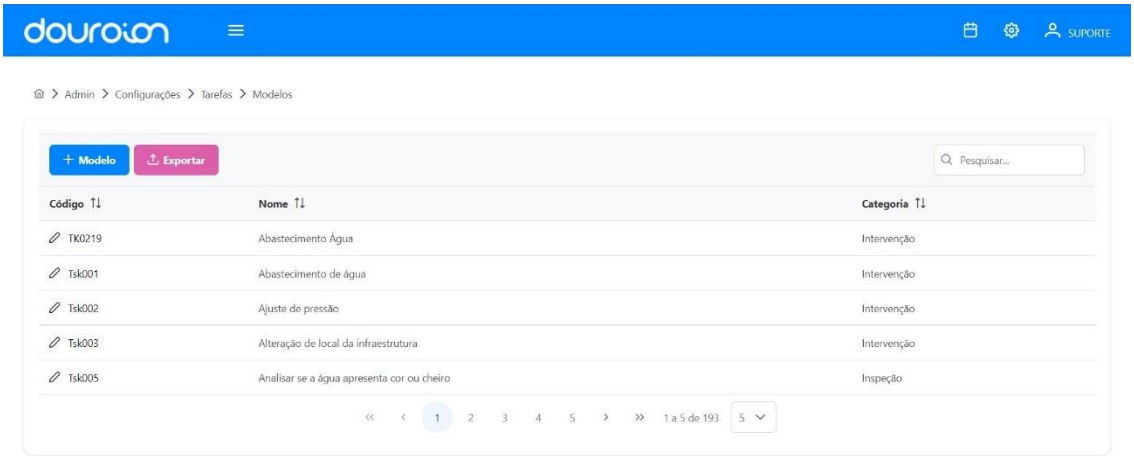


Figura 24 – Lista do ION 2

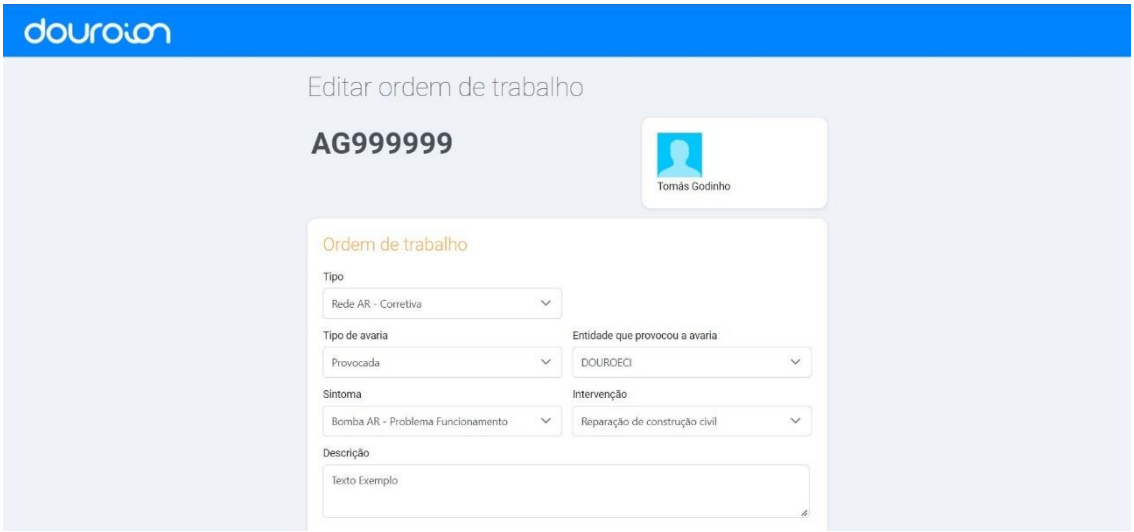


Figura 25 - Ordem de trabalho parcial 1 do ION 2

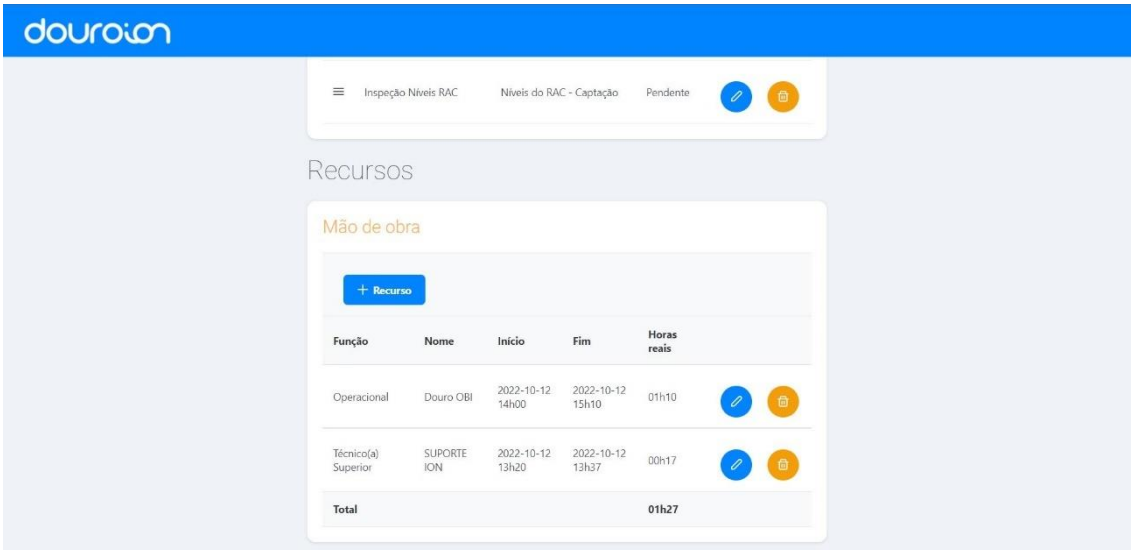


Figura 26 - Ordem de trabalho parcial 2 do ION 2

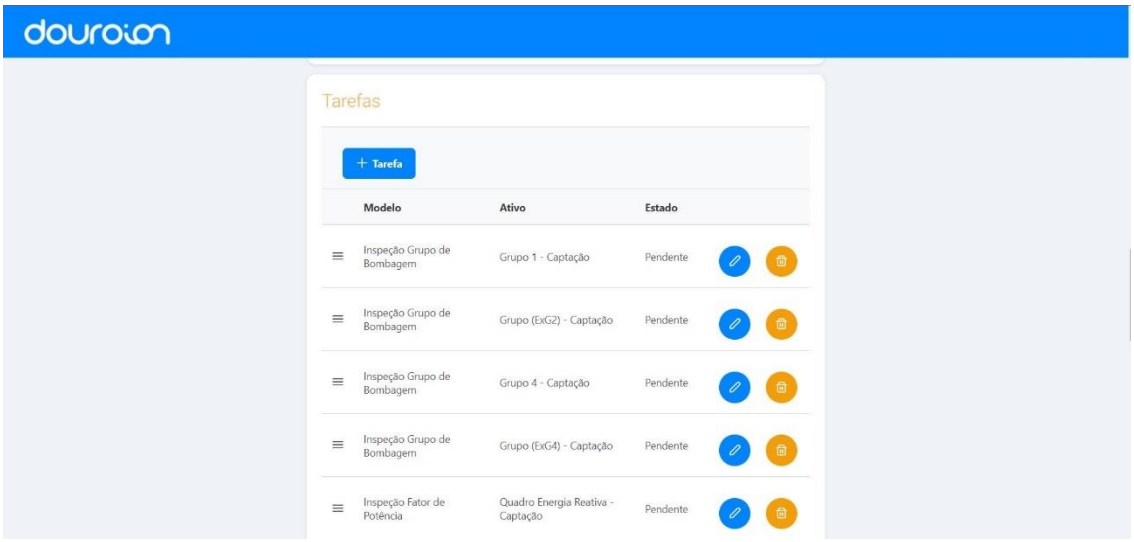


Figura 27 - Ordem de trabalho parcial 3 do ION 2

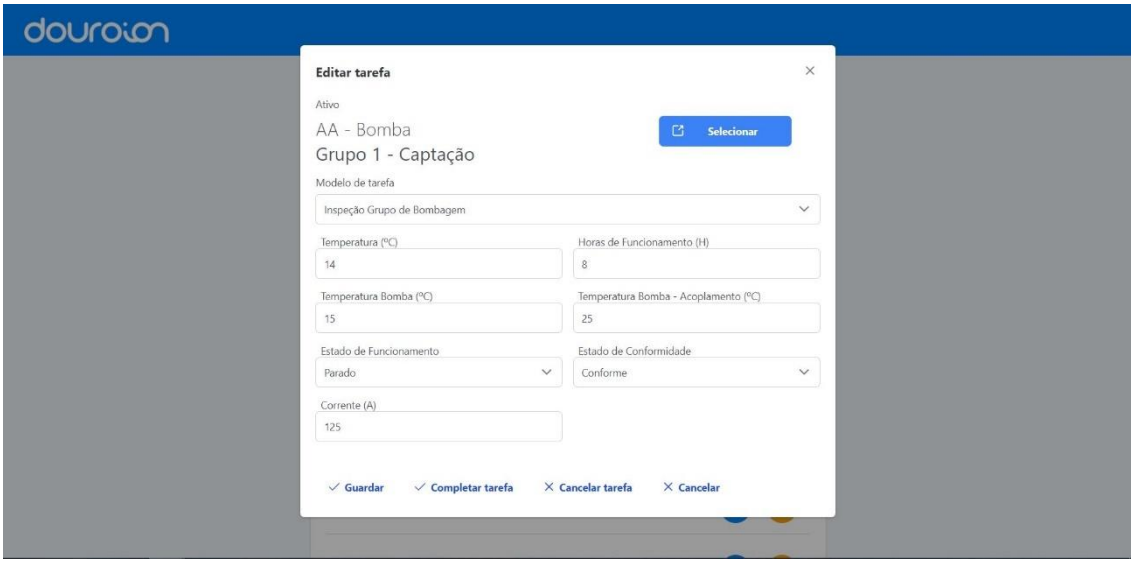


Figura 28 - Criar tarefa dentro de uma Ordem de trabalho no ION 2

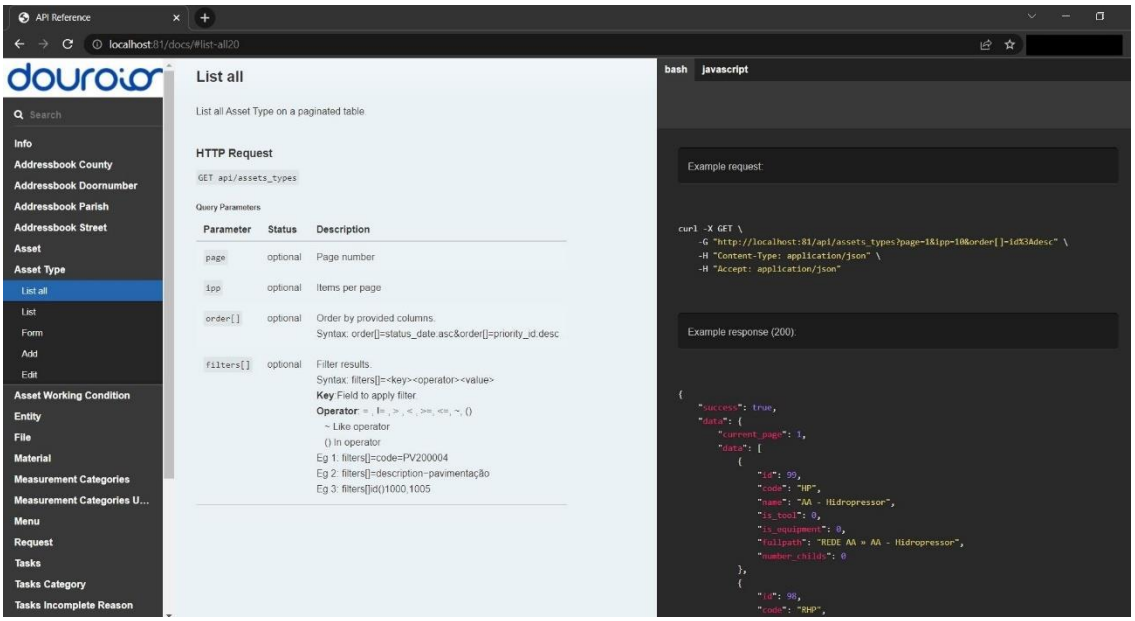


Figura 29 - Documentação da API do backend do ION 2

Anexo B – Exemplos da wiki interna

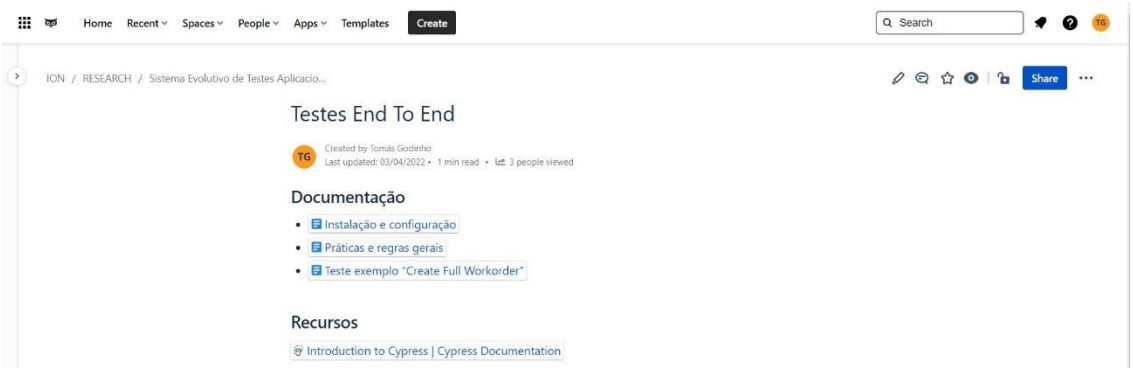


Figura 30 – Exemplo da organização de pastas

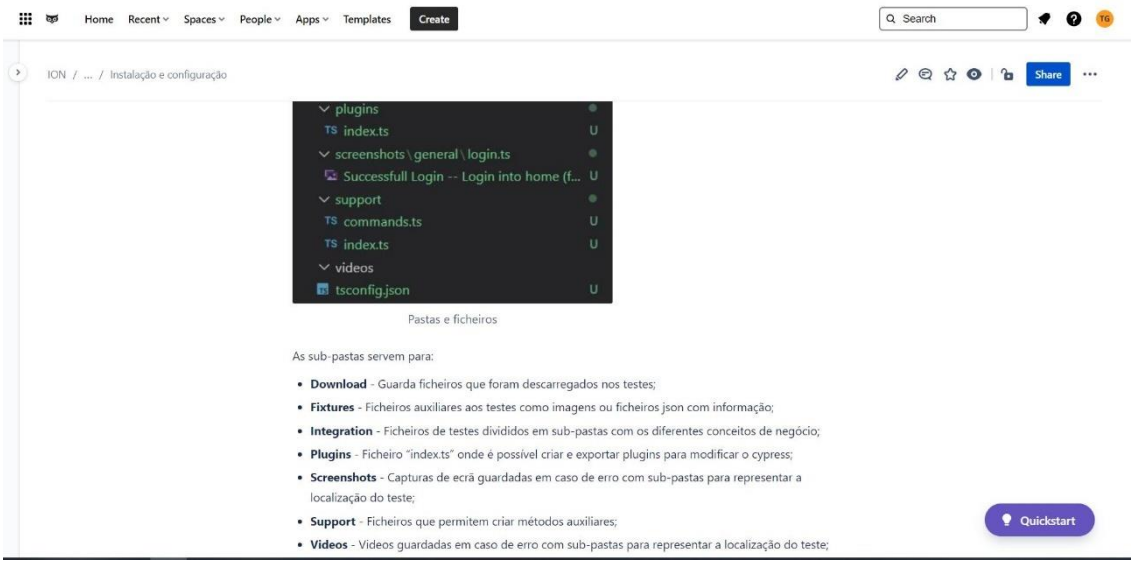


Figura 31 - Exemplo de uma página da wiki

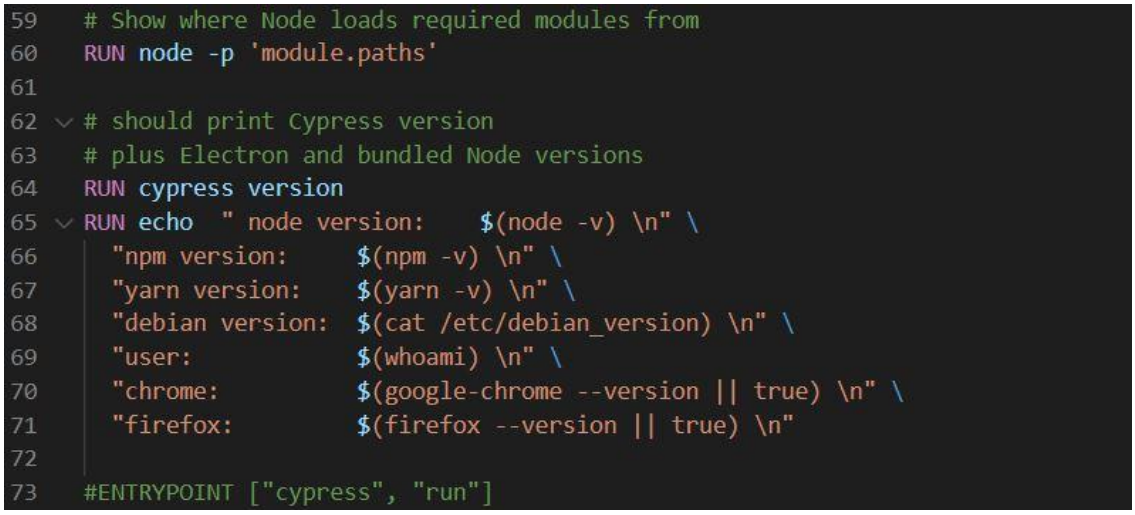
Anexo C – Dockerfile Cypress

```
FROM cypress/browsers:node16.14.0-slim-chrome99-ff97

WORKDIR /home/douro/ion/frontend_d
ENV PATH /app/node_modules/.bin:$PATH
COPY ./debian/node/package.json /home/douro/ion/frontend_d/

# Install all dependencies
RUN apt-get install libgtk2.0-0 libgtk-3-0 libgbm-dev libnotify-dev libgconf-2-4 libnss3 libxss1 libasound2 libxtst6 xauth xvfb

# Update the dependencies to get the latest and greatest (and safest!) packages.
RUN apt update && apt upgrade -y

# avoid too many progress messages
# https://github.com/cypress-io/cypress/issues/1243
ENV CI=1

# disable shared memory X11 affecting Cypress v4 and Chrome
# https://github.com/cypress-io/cypress-docker-images/issues/270
ENV QT_X11_NO_MITSHM=1
ENV _X11_NO_MITSHM=1
ENV _MITSHM=0

# should be root user
RUN echo "whoami: $(whoami)"
#RUN npm config -g set user $(whoami)
RUN npm config set user $(whoami)

# command "id" should print:
# uid=0(root) gid=0(root) groups=0(root)
# which means the current user is root
RUN id
```

Figura 32 - Cypress Dockerfile 1

```

  # point Cypress at the /root/cache no matter what user account is used
  # see https://on.cypress.io/caching
  ENV CYPRESS_CACHE_FOLDER=/root/.cache/Cypress
  RUN npm install -g --save-dev "typescript@4.2.3"
  RUN tsc -v

  RUN npm install -g "cypress@9.5.1"
  RUN cypress verify

  # Cypress cache and installed version
  # should be in the root user's home folder
  RUN cypress cache path
  RUN cypress cache list
  RUN cypress info
  RUN cypress version

  # give every user read access to the "/root" folder where the binary is cached
  # we really only need to worry about the top folder, fortunately
  RUN ls -la /root
  RUN chmod 755 /root

  # always grab the latest Yarn
  # otherwise the base image might have old versions
  # NPM does not need to be installed as it is already included with Node.
  RUN npm i -g yarn@latest

```

Figura 33 - Cypress Dockerfile 2



```
# Show where Node loads required modules from
RUN node -p 'module.paths'

# should print Cypress version
# plus Electron and bundled Node versions
RUN cypress version
RUN echo  " node version:    $(node -v) \n" \
  "npm version:     $(npm -v) \n" \
  "yarn version:    $(yarn -v) \n" \
  "debian version:  $(cat /etc/debian_version) \n" \
  "user:            $(whoami) \n" \
  "chrome:          $(google-chrome --version || true) \n" \
  "firefox:         $(firefox --version || true) \n"

#ENTRYPOINT ["cypress", "run"]
```

Figura 34 - Cypress Dockerfile 3

Anexo D – Execução dos testes Cypress

```
Running:  workorders/editWorkorder.ts                                    (7 of 7)


Edit Workorder
  √ Open Workorder View (27874ms)


1 passing (28s)


(Results)

┌─────────────┬──────────────────────────────┐
│ Tests:      │ 1                            │
│ Passing:    │ 1                            │
│ Failing:    │ 0                            │
│ Pending:    │ 0                            │
│ Skipped:    │ 0                            │
│ Screenshots:│ 0                            │
│ Video:      │ false                        │
│ Duration:   │ 27 seconds                   │
│ Spec Ran:   │ workorders/editWorkorder.ts  │
└─────────────┴──────────────────────────────┘


(Run Finished)

  Spec                                        Tests  Passing  Failing  Pending  Skipped
┌──────────────────────────────────────────────────────────────────────────────────────┐
│ √ general/login.ts                  00:52   1      1        -        -        -     │
├──────────────────────────────────────────────────────────────────────────────────────┤
│ √ pavements/createPavements.ts      00:08   1      1        -        -        -     │
├──────────────────────────────────────────────────────────────────────────────────────┤
│ √ requests/createWorkorderFromRequest 00:08 1      1        -        -        -     │
│   .ts                                                                                │
├──────────────────────────────────────────────────────────────────────────────────────┤
│ √ workorders/createBasicWorkorder.ts 00:33  1      1        -        -        -     │
├──────────────────────────────────────────────────────────────────────────────────────┤
│ √ workorders/createFullWorkorder.ts 01:57   1      1        -        -        -     │
├──────────────────────────────────────────────────────────────────────────────────────┤
│ √ workorders/createWorkordersValidati 00:16 2      2        -        -        -     │
│   ons.ts                                                                             │
├──────────────────────────────────────────────────────────────────────────────────────┤
│ √ workorders/editWorkorder.ts       00:27   1      1        -        -        -     │
├──────────────────────────────────────────────────────────────────────────────────────┤
│ √ All specs passed!                 04:24   8      8        -        -        -     │
└──────────────────────────────────────────────────────────────────────────────────────┘
```

Figura 35 - Cypress run mode

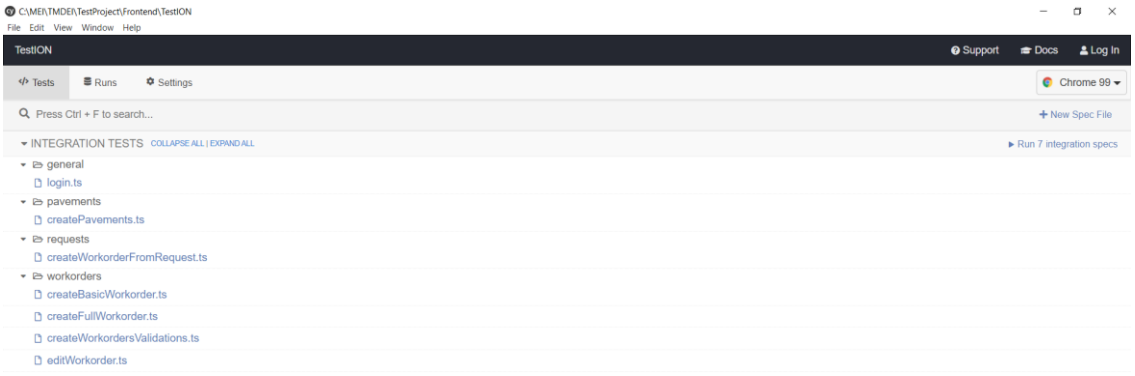


Figura 36 - Cypress open mode menu

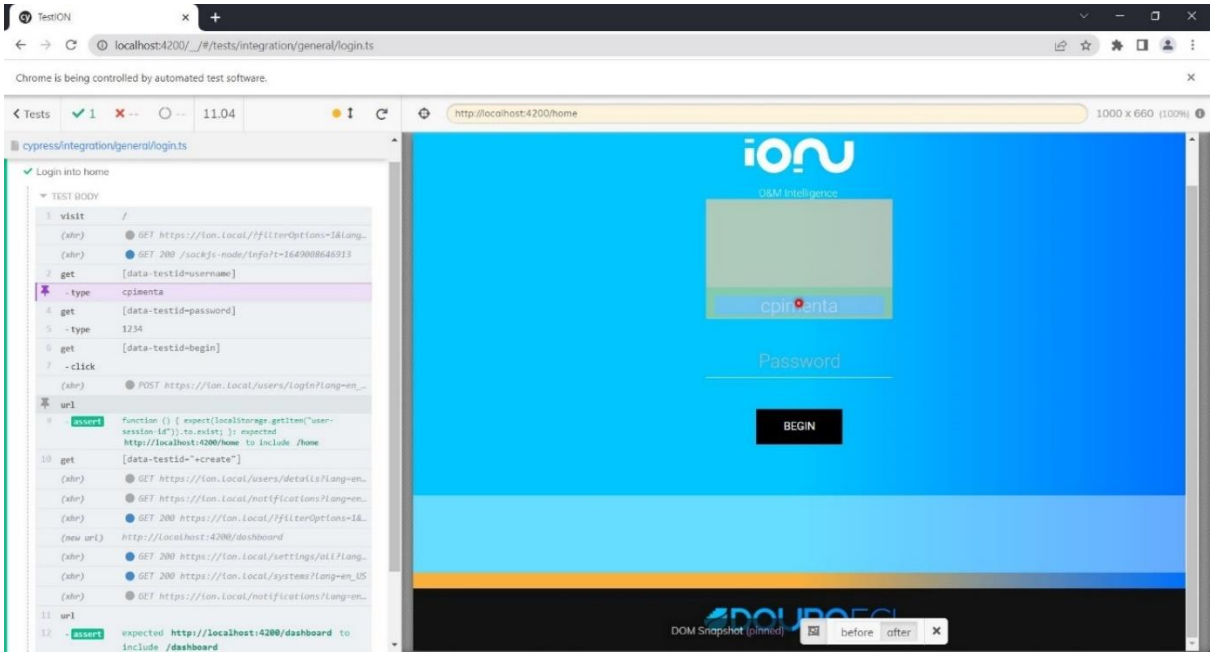


Figura 37 - Cypress open mode a “viajar no tempo”

Anexo E – Questionário

Tabela 10 - Questionário introdução

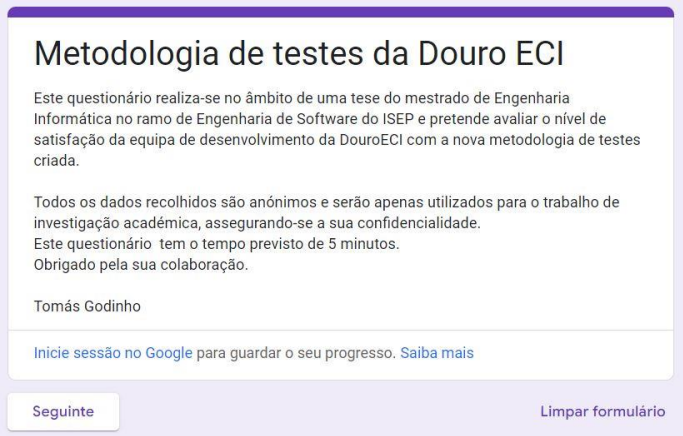


Tabela 11 - Questionário Dados Gerais

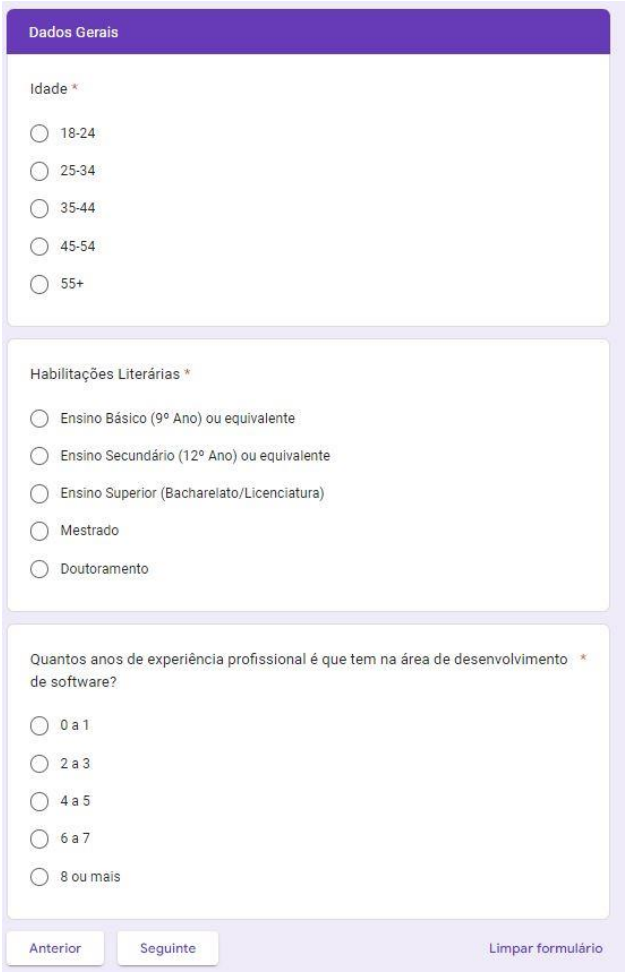
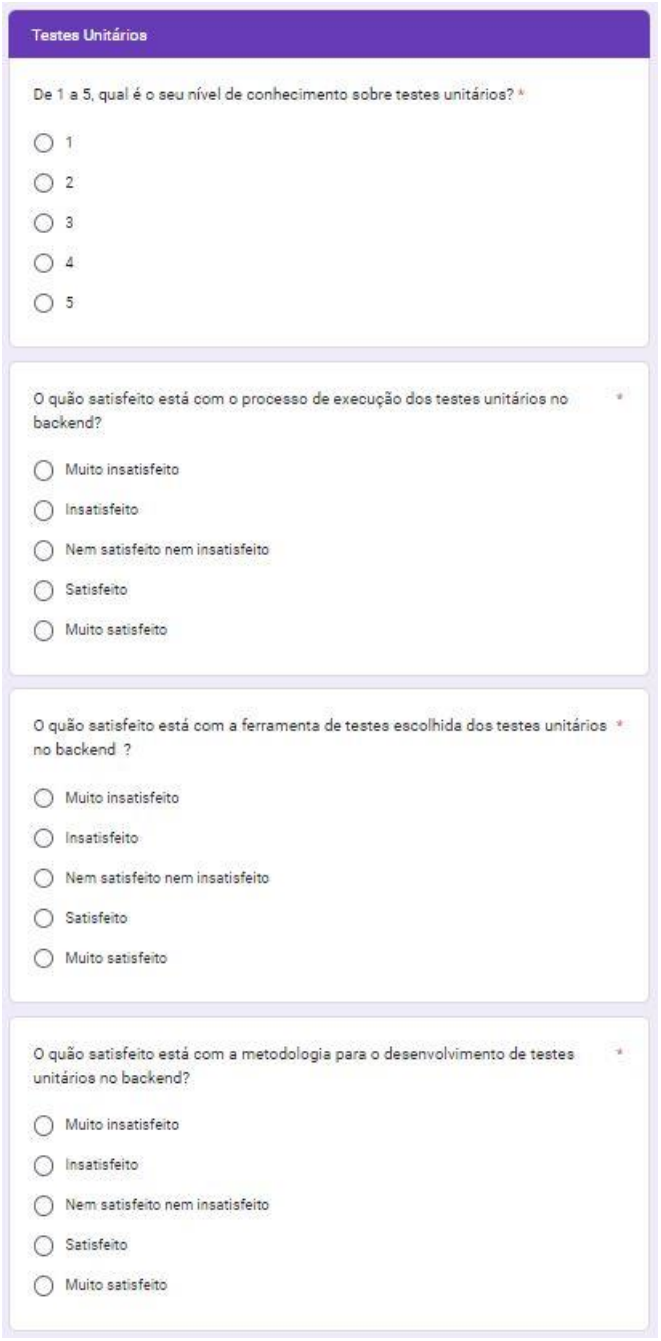


Tabela 12 - Questionário Testes Unitários 1



Testes Unitários

De 1 a 5, qual é o seu nível de conhecimento sobre testes unitários? *

- 1
- 2
- 3
- 4
- 5

O quão satisfeito está com o processo de execução dos testes unitários no backend? *

- Muito insatisfeito
- Insatisfeito
- Nem satisfeito nem insatisfeito
- Satisfeito
- Muito satisfeito

O quão satisfeito está com a ferramenta de testes escolhida dos testes unitários no backend ? *

- Muito insatisfeito
- Insatisfeito
- Nem satisfeito nem insatisfeito
- Satisfeito
- Muito satisfeito

O quão satisfeito está com a metodologia para o desenvolvimento de testes unitários no backend? *

- Muito insatisfeito
- Insatisfeito
- Nem satisfeito nem insatisfeito
- Satisfeito
- Muito satisfeito

Tabela 13 - Questionário Testes Unitários 2

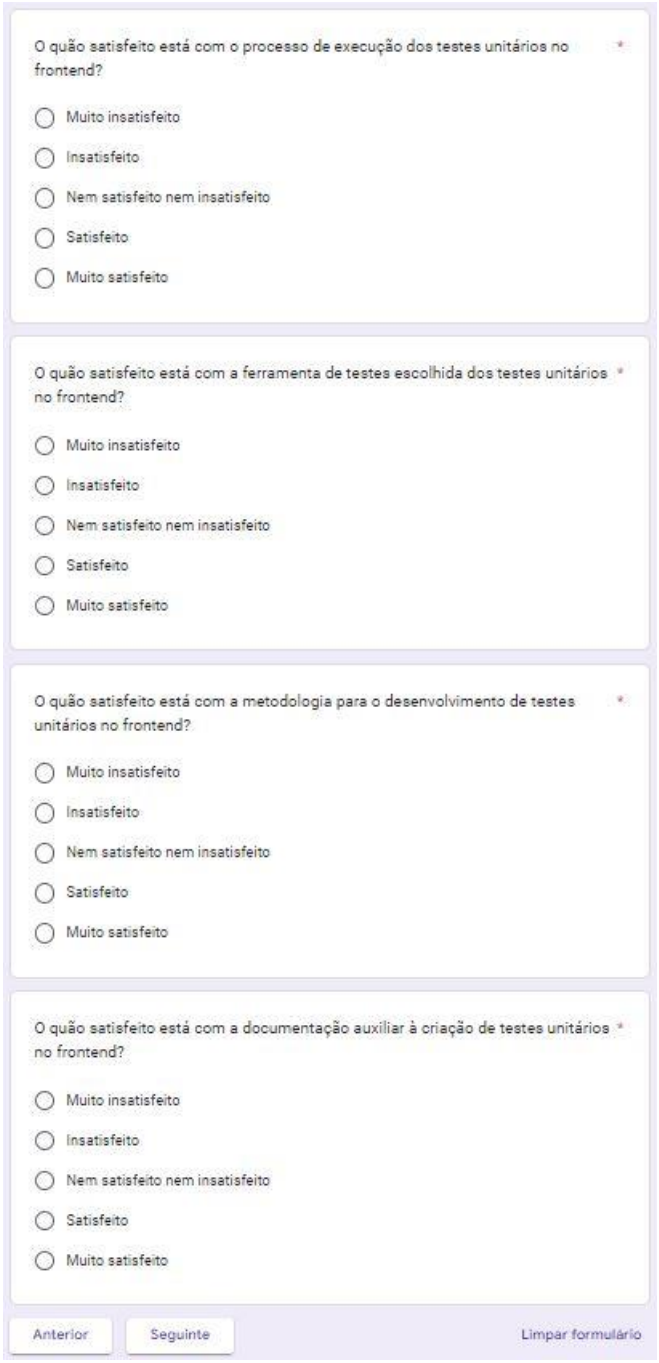


Tabela 14 - Questionário Testes Integração 1

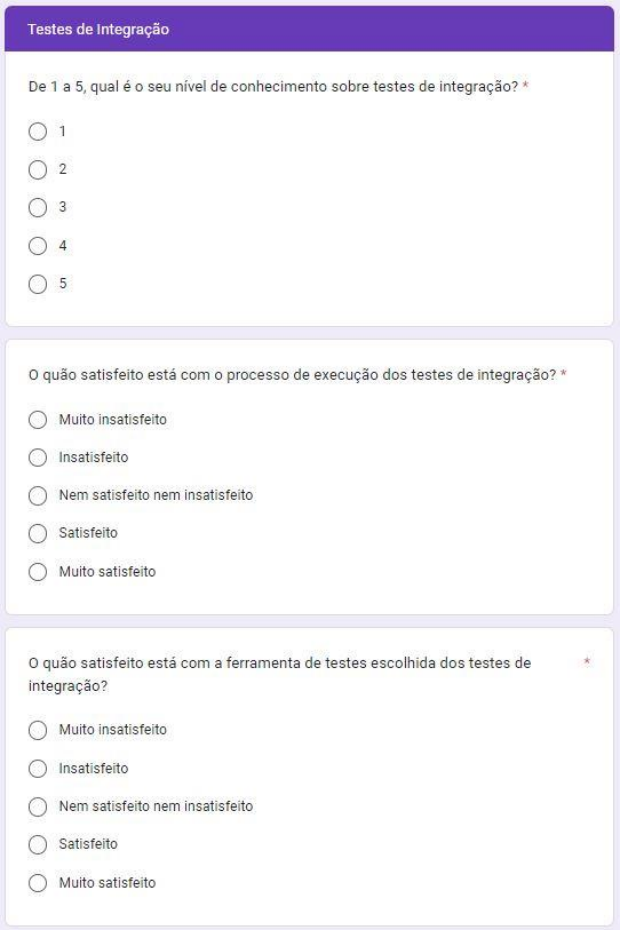


Tabela 15 - Questionário Testes Integração 2

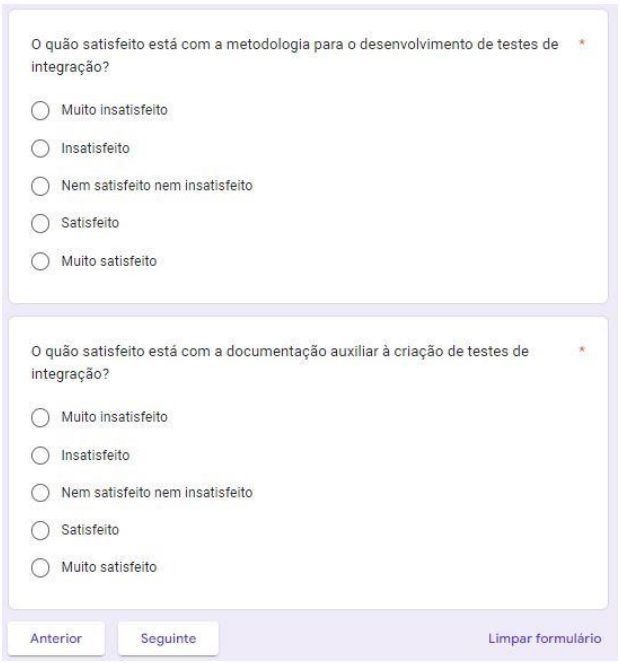


Tabela 16 - Questionário Testes End to End 1

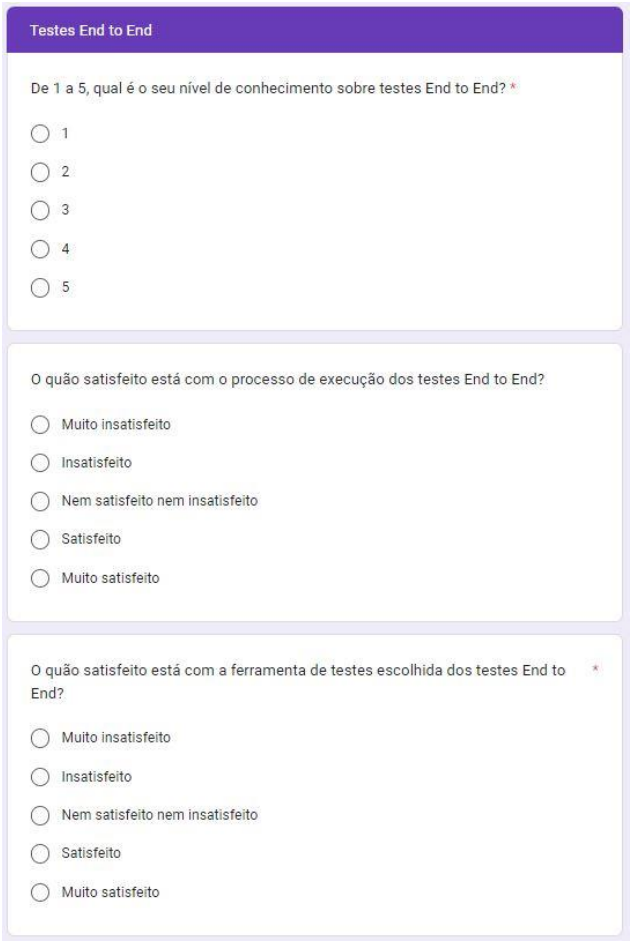


Tabela 17 - Questionário Testes End to End 2

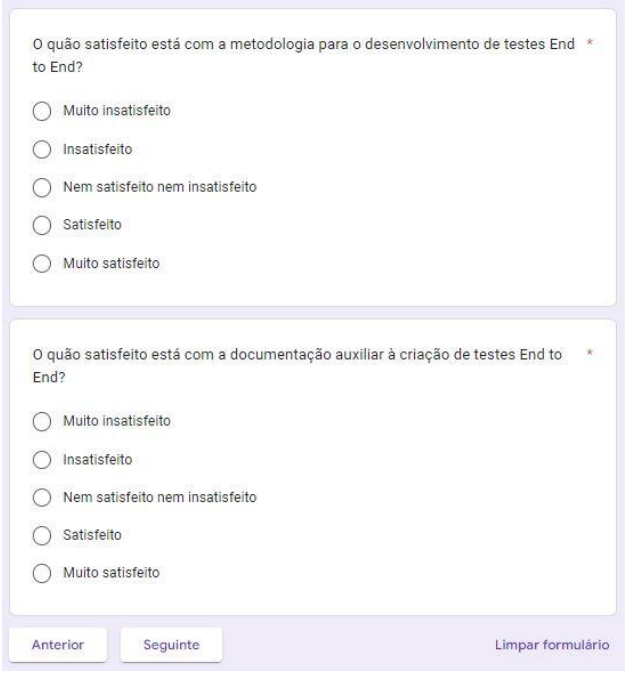


Tabela 18 - Questionário Avaliação Geral