INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




Plataforma Web Para Suporte a Cerimónias de Melhoria Contínua

LUÍS CARLOS GOMES QUEIROZ DOS SANTOS
Outubro de 2020

POLITÉCNICO
DO PORTO

Instituto Superior de Engenharia do Porto

Plataforma Web Para Suporte a Cerimónias de Melhoria Contínua

Luís Santos

Dissertação para obtenção do Grau de Mestre em Engenharia Informática, Área de Especialização em Sistemas Gráficos e Multimédia

Orientador: Doutor Filipe Pacheco

Porto, 14 de Outubro de 2020

Resumo

Com o avançar dos tempos o mundo tem se tornando cada vez mais tecnológico. Efetivamente de toda a indústria o sector das tecnologias é dos que mais cresce globalmente e isto leva também a uma maior procura às metodologias de trabalho. As empresas são cada vez mais dinâmicas e com foco nas pessoas.

Este projeto foi desenvolvido na *Mindera*, uma empresa no ramo de desenvolvimento de *software* que usa metodologias *Agile*, e que usa a tecnologia para desenvolver produtos de que se orgulha com pessoas que gosta.

As metodologias *Agile* têm em atenção o papel da pessoa no processo de desenvolvimento, e um exemplo destas metodologias é o *Scrum*.

O *Scrum* que é uma metodologia iterativa e incremental, tem como grande objetivo a entrega contínua de produtos com qualidade. Tem por sua base quatro cerimónias, a *Sprint Planning Meeting*, a *Daily Scrum*, a *Sprint Review Meeting* e a *Sprint Retrospective Meeting*.

Este trabalho irá incidir essencialmente sobre a *Sprint Retrospective Meeting* que é uma cerimónia de avaliação em que o grande objetivo é conseguir compreender como é que uma equipa pode ganhar mais talento, analisando pontos fracos e fortes e criando ações para a melhorar. Estas cerimónias têm impacto direto na qualidade e na produtividade das equipas apesar de por vezes serem menosprezadas.

Com o aumento do trabalho remoto a execução da *Sprint Retrospective Meeting* encontra novos desafios.

Este projeto visa desenvolver uma solução informática que permita aos elementos das equipas, que se encontram a trabalhar remotamente, participar ativamente e em tempo real numa sessão de *Sprint Retrospective*.

A solução desenvolvida visa promover a execução regular de sessões de *Sprint Retrospective* afim das equipas conseguirem aproveitar todos os benefícios que a mesma poderá trazer, e desta forma, comprovar que esta cerimónia poderá ser a mais importante no que toca ao desenvolvimento de talento para a empresa.

Palavras-chave: Agile, Scrum, Sprint Retrospective, WebApp, Auto-avaliação, Melhoria Continua

Abstract

With the passage of time the world has become increasingly technological. In fact, of all industry sectors, the technological one is the fastest growing globally and this leads to a greater demand for work methodologies. Companies are becoming increasingly dynamic and focused on people.

This project was developed at Mindera, a company in the field of software development that uses Agile methodologies alongside the people they like to proudly develop products.

Agile methodologies pay attention to the person's role in the development process, and an example of these methodologies is Scrum.

Scrum, which is an iterative and incremental methodology, has as its main objective the continuous delivery of quality products. It is based on four ceremonies, the Sprint Planning Meeting, the Daily Scrum, the Sprint Review Meeting and the Sprint Retrospective Meeting.

This work will focus mainly on the Sprint Retrospective Meeting, which is an evaluation ceremony in which the main objective is to understand how a team can gain more talent, analysing weaknesses and strengths and creating actions to improve it. These ceremonies have a direct impact on the quality and productivity of the teams despite being sometimes overlooked.

With the increase in remote work, running the Sprint Retrospective Meeting faces new challenges.

This project aims to develop a computer solution that allows team members, who are working remotely, to participate actively and in real time in a Sprint Retrospective session.

The developed solution aims to promote the regular execution of Sprint Retrospective sessions in order for the teams to be able to take advantage of all the benefits that it may bring, and in this way, prove that this ceremony may be the most important in terms of talent development for the company.

Agradecimentos

Quero agradecer a todos os que de forma direta ou indiretamente, tornaram este meu percurso académico mais fácil e aos que me apoiaram nos momentos mais difíceis.

Ao Sérgio Gomes e ao Rúben Amorim enquanto representantes da Mindera, por todo o apoio e atenção dada ao longo do desenvolvimento de todo o projeto.

Ao José Emídio Figueiredo pelos conhecimentos a mim transmitidos, à Manuela Dourado por toda a ajuda dada não só no desenvolvimento desta dissertação bem como em todo o percurso académico.

Ao Professor Nuno Silva pela oportunidade dada desde do primeiro dia no ISEP e ao Professor Filipe Pacheco pela disponibilidade, ajuda e mentoria dada ao longo do desenvolvimento da dissertação.

A toda a minha família e amigos!

Um grande e sentido, Obrigado!

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Acrónimos

AHP	Analytic Hierarchy Process.
CD	Continuous Delivery.
CSS	Cascading Style Sheets.
DOM	Document Object Model.
FURPS+	Funcionalidade, Usabilidade, Reliabilidade, Performance, Suportabilidade, Mais Outros Requisitos Não-Funcionais.
GPS	Global Positioning System.
HTTP	Hypertext Transfer Protocol.
MVC	Model-View-Controller.
NCD	New Concept Development.
PO	Product Owner.
SPA	Single Page Application.
UP	Unified Process.
XP	Extreme Programming.

Capítulo 1

Introdução

Este primeiro capítulo visa apresentar e descrever de forma sucinta o trabalho desenvolvido na criação de uma solução *online* para suporte à cerimónia de melhoria contínua de *Scrum*, bem como no mesmo serão abordados os vários temas. Poderemos também ver quais os grandes objetivos que se pretende atingir.

1.1 Contexto

No presente capítulo são apresentados os conceitos bases adjacentes ao projeto. Segue se posteriormente uma breve descrição do modelo de negócio da Mindera, empresa onde foi desenvolvido o projeto, bem como a relevância do desenvolvimento para a mesma.

1.1.1 Mindera

A Mindera é uma empresa de desenvolvimento de *software* fundada em Setembro de 2014, que conta nos dias de hoje com mais de 400 colaboradores espalhados por 6 escritórios e 3 continentes (Mindera 2019).

A Mindera para além de desenvolver *software* possuí outras pequenas empresas que atuam em diversas áreas. As soluções que desenvolvem destinam se aos mais variados sectores tais como, retalho, cadeias de restaurantes *fast-food*, produtos financeiros até ao jogo.

A Mindera defende que usa a tecnologia para criar produtos que se orgulha com pessoas que gosta (Mindera 2019). A cultura da empresa reflete-se na forma como é estruturada, e cada indivíduo é responsável pela organização do seu trabalho.

1.1.2 Metodologias

A Mindera adota a metodologia *Agile* como a sua forma típica de abordagem ao desenvolvimento de *software*. Assim sendo o trabalho é construído de forma iterativa e incremental, conhecidos por *Sprints*. Isto permite que as equipas consigam garantir a qualidade do *software* desenvolvido bem como lidar com a imprevisibilidade que possa acontecer. *Agile* é também promotor de avaliação contínua que guia o projeto ao longo do ciclo de desenvolvimento.

Juntamente com *Agile* são também aplicadas metodologias de *Scrum* ou *Kanban* que são escolhidas conforme o cliente e a especificidade do projeto a desenvolver.

A Mindera assenta também o seu desenvolvimento de software em *Continuous Delivery (CD)*, focando-se na automação e melhoramento do processo de entrega de *software*, utilizando desde testes automáticos, integração contínua e implantação contínua. Isto permite garantir a qualidade do desenvolvimento, facilitar todo o processo de lançamento de versões de *software* e assegurar que é implantado regularmente em ambientes de teste.

1.1.3 Agile

Agile é uma metodologia de desenvolvimento de *software* que se foca principalmente em fornecer valor incremental através de uma cadência de trabalho iterativa, também conhecida como *Sprints* (Agile Alliance 2019). *Agile* tem por base um conjunto de estruturas e práticas que se baseiam em valores e princípios que foram descritos no Manifesto a quando da sua criação. O Manifesto tem uma lista, de cerca de 12 regras, que são aconselhadas quando se adotam metodologias *Agile* no processo de desenvolvimento de *software* (M. Fowler 2001).

Um grande ponto diferenciador comparativamente com outras metodologias de desenvolvimento de *software* é o constante foco e preocupação com as pessoas que desenvolvem soluções e que trabalham em equipa. O desenvolvimento das soluções é um processo evolutivo e as mesmas dependem da colaboração entre as equipas multifuncionais, que têm um sentido de auto-organização adotando sempre um conjunto de boas práticas.

Agile tem um elevado foco pela cooperação e pela auto-organização onde cada um é responsável pelas suas decisões. Contudo isto não significa que as posições de gestão não existam, mas sim que as equipas de trabalho têm a capacidade de descobrirem e perceberem qual a melhor forma para abordarem os problemas. Isto faz com que as equipas não tenham que ter funções atribuídas tornando-as multidisciplinares, assegurando que têm todos os recursos necessários para desenvolverem os projetos a que são sujeitos.

Os gestores, mais conhecidos por *Product Owner (PO)*, têm a função de garantir que todas as equipas têm o talento necessário para se assegurar o desenvolvimento de soluções. São estes também os responsáveis pela criação de condições para que as equipas cheguem ao sucesso. Os gestores são observadores contudo têm o papel de intervir quando o funcionamento da equipa põe em causa o processo de desenvolvimento (Agile Alliance 2019).

Devido à sua natureza iterativa, *Agile* presta-se à avaliação contínua da direção de um projeto ao longo do ciclo de desenvolvimento. As reuniões ou "cerimónias" são uma parte importante do desenvolvimento ágil. Estas ajudam a disseminar informações oportunas, destacar objetivos e visões comuns e partilhar o progresso da equipa com todos os elementos (Agile Alliance 2019).

1.1.4 Scrum

Scrum é uma metodologia onde as pessoas conseguem gerir problemas complexos, contudo, ao mesmo tempo, permite que consigam assegurar a entrega de produtos de forma mais produtiva e criativa, com uma elevada quantidade de valor. Assume uma forma simples para a colaboração eficaz de equipas em projetos complexos. Segundo os criadores *Scrum* é leve, simples de entender e difícil de dominar (F. M. Fowler 2019).

Scrum é uma metodologia empírica, pois através da mesma as equipas conseguem estabelecer hipóteses de como é que elas acham que algo pode ser solucionado. Isto permite que as equipas lidem de uma forma mais controlada com a imprevisibilidade inerente ao desenvolvimento de *software* mantendo um fluxo de artefactos de *software* de alta qualidade.

Em projetos *Scrum*, há uma repetição sequencial de reuniões, a serem realizadas anteriormente, durante e após o ciclo de *Sprint*.

A metodologia *Scrum* sugere 4 cerimónias: a *Sprint Planning Meeting*, *Daily Scrum*, *Sprint Review Meeting* e a *Sprint Retrospective Meeting* (F. M. Fowler 2019).

Na *Sprint Planning Meeting* é planeado todo o trabalho que irá decorrer no *Sprint* e necessita que toda a equipa esteja presente (Scrum.org 2020). A *Daily Scrum* é um evento diário e cronometrado onde é feito um ponto de situação das próximas 24 horas (Ockerman 2019). A *Sprint Review Meeting* pretende analisar a lista de tarefas do projeto e assim perceber qual o valor incrementado ao produto em que a equipa está a trabalhar (Scrum.org 2019).

Na cerimónia de *Sprint Retrospective Meeting* a equipa deve reunir para refletir sobre o seu anterior *Sprint* para que consigam perceber como podem melhorar enquanto equipa. A *Retrospective* permite que a equipa se concentre no seu desempenho geral e identifique estratégias para melhoria contínua (SCrum.org 2010).

1.2 Problema

Com o crescente aumento de elementos de equipas localizados nos mais diversos pontos geográficos, com a possibilidade que as pessoas têm de trabalhar a partir de casa e juntando a situação pandémica que se vive atualmente a uma escala global, a dificuldade de se efetuar com sucesso todo o cerimonial de um *Sprint* tem aumentado cada vez mais (Jensen et al. 2020). A sessão de *Retrospective* sofre ainda de uma dificuldade acrescida que é o fato de muito elementos das equipas desvalorizarem a mesma apesar dos impactos positivos que pode ter no processo de desenvolvimento (*Scrum Sprint Retrospective Problems* s.d.).

Já existem algumas ferramentas no mercado para a elaboração da sessão de *Retrospective* contudo as mesmas apresentam algumas limitações ou até problemas levando a que as equipas tenham algumas dificuldades na elaboração desta cerimónia. A deslocalização das equipas torna uma tarefa que poderia ser feita fisicamente, com ajuda de notas escritas manualmente em *post-it*, impraticável. Assim sendo, surge a necessidade de analisar de que forma podemos satisfazer as necessidades que uma cerimónia de *Retrospective* exige recorrendo a um sistema informático bem como tentar atrair as equipas para a elaboração regular desta cerimónia.

1.3 Objetivos

O objetivo deste projeto é o desenvolvimento de uma solução baseada na *internet* que permita qualquer interveniente numa cerimónia de *Retrospective*, localizado em qualquer parte do globo, participar ativamente e remotamente em tempo real numa destas cerimónias, e desta forma fomentar as sessões de *Retrospective* para que as mesmas consigam ter impacto na constante melhoria do processo de desenvolvimento de software de uma equipa.

São identificados como sub-objetivos os seguintes pontos:

- Desenvolver um protótipo funcional de uma aplicação web que visa o apoio às cerimónias de *Retrospective*;
- Ter impacto na melhoria contínua no desenvolvimento de soluções de uma equipa de trabalho com o apoio da solução desenvolvida com este projeto;

- Incentivar à participação colaborativa de todos os elementos de uma equipa para a melhoria contínua da mesma.

1.4 Abordagem

A solução a desenvolver terá ao cuidado os dois grandes intervenientes:

- O gestor de projeto, o *PO*, é a pessoa que tem interesse na parte de gestão da equipa. O *PO* é aquele que com a cerimónia de *Retrospective* tem em vista conseguir perceber quais os pontos fortes da equipa para os poder potenciar, quais o pontos fracos para que os possa solucionar e que ações poderá desenvolver para continuar a potenciar e alimentar os pontos positivos e de que forma é que vai solucionar ou melhorar os pontos negativos;
- O programador, com a cerimónia de *Retrospective* conseguirá fazer um ponto de situação do projeto onde está envolvido bem como o estado da equipa de trabalho na qual está integrado. A sessão de *Retrospective* também poderá ser utilizada para fins mais individuais, onde cada indivíduo pode fazer uma autoavaliação do seu conhecimento e da sua prestação em relação à equipa, e desta forma também analisar a sua prestação enquanto profissional.

1.5 Planeamento do projeto

O planeamento do desenvolvimento do projeto visa numa fase inicial participação como observador em algumas sessões de *Retrospective* de forma a perceber quais a necessidades das equipas, as particularidades de funcionamento de cada uma e o funcionamento das cerimónias.

Numa fase seguinte inicia-se um processo de análise de requisitos funcionais e não funcionais. Este levantamento será efetuado em duas frentes, a do *PO* onde é analisado quais a suas necessidades enquanto gestor e organizador das cerimónias e de uma perspetiva dos participantes de forma ver as necessidades de cada utilizador.

Será adotado um processo desenvolvimento de *software* incremental e iterativo que visa construir uma solução sólida, com qualidade e que consegue lidar com a imprevisibilidade e constantes retornos de estado global do projeto. Para isto são criadas soluções de integração contínua automáticas com processos de testes automáticos e implantação regular que visa um crescimento controlado da solução. Para ajudar a alcançar o sucesso serão então usadas ferramentas de controlo de versões, nomeadamente o *GitHub*, ferramentas de integração contínua, *Pipelines* do *GitHub*, um sistema de gestão de tarefas, o *Trello*, e uma metodologia de trabalho apoiada em *Scrum*.

Todo o processo de desenvolvimento da solução é também acompanhado pelas partes interessadas de forma a conseguir um retorno de satisfação em relação à solução que se está a desenvolver.

Como se trata de um processo que assenta em metodologias iterativas e incrementais, permite então que o projeto esteja sobre um constante controlo de qualidade de desenvolvimento prático e de casos de uso a implementar, conseguindo-se assim uma solução que vai sofrendo metamorfoses conforme o seu crescimento em dimensão.

Assim sendo o projeto assenta essencialmente sobre 3 grandes fases.

- Inicialmente é efetuado um processo exaustivo de estudo e análise de soluções já existentes bem com testes de usabilidade aos mesmos. A finalidade é perceber quais os pontos positivos e negativos de cada solução e perceber quais são os pontos que afetam negativamente uma cerimónia de *Retrospective* levada a cabo por uma equipa da empresa, em suma perceber porque é que determinada solução não responde às necessidades da equipa. Com este estudo dá se início ao levantamento de requisitos funcionais e não funcionais com base na análise, só numa fase seguinte é que serão acrescentados os requisitos que as equipas têm para o desenvolvimento da solução. Por fim ainda nesta primeira fase é também levado a cabo um estudo tecnológico para se identificar quais as tecnologias que se enquadram para dar a melhor resposta ao desenvolvimento da solução.

- Numa segunda fase é efetuada uma análise detalhada dos requisitos já anteriormente levantados. Esta análise tem como finalidade criar uma estrutura de prioridades e relevo de cada requisito com a finalidade de se melhor estruturar o caminho que o desenvolvimento irá tomar.

  Após isso é então efetuado uma análise de *design* que visa perceber qual a melhor estrutura arquitetural que o projeto deverá tomar, para construir uma solução sólida e com qualidade. Esta análise tem como resultado um série de artefactos e diagramas, que visam sustentar as decisões tomadas bem como perceber se a solução está estruturada da forma mais adequada bem antes de se efetuar qualquer desenvolvimento prático, ou seja, antes de se começar a programar a solução.

  De seguida é então levado a cabo a implementação de todas as decisões arquiteturais seguindo as metodologias já decididas, considerando com rigor os requisitos levantados e analisados. A implementação é também controlada com a ajuda de testes automáticos que visam controlar a qualidade do *software*. Juntamente com os testes automáticos é também tido em conta uma constante relação com os destinatários para se perceber se as respostas às suas necessidades estão realmente a ser tomadas em conta e resolvidas.

- Por fim, é apresentada uma fase mais documental, onde todo este processo de desenvolvimento é relatado e sustentado. Também será criado, quando oportunamente, documentação relativa ao suporte da solução desenvolvida.

  Também nesta fase final será analisada e comparada todos os objetivos do projeto, para se verificar o sucesso ou insucesso de cada um. Será levantado um estado dos requisitos de forma a se perceber em que ponto de situação fica a solução face às necessidades dos utilizadores alvo.

  Serão ainda efetuados testes funcionais e de usabilidade que têm como finalidade perceber qual o *feedback* dos intervenientes na utilização da solução e que visam concluir se a solução efetivamente resolve o problema das equipas bem como estimar possíveis trabalhos futuros na solução.

Podemos ver na figura seguinte um diagrama de *Gantt* que ilustra os principais passos do planeamento do projeto.

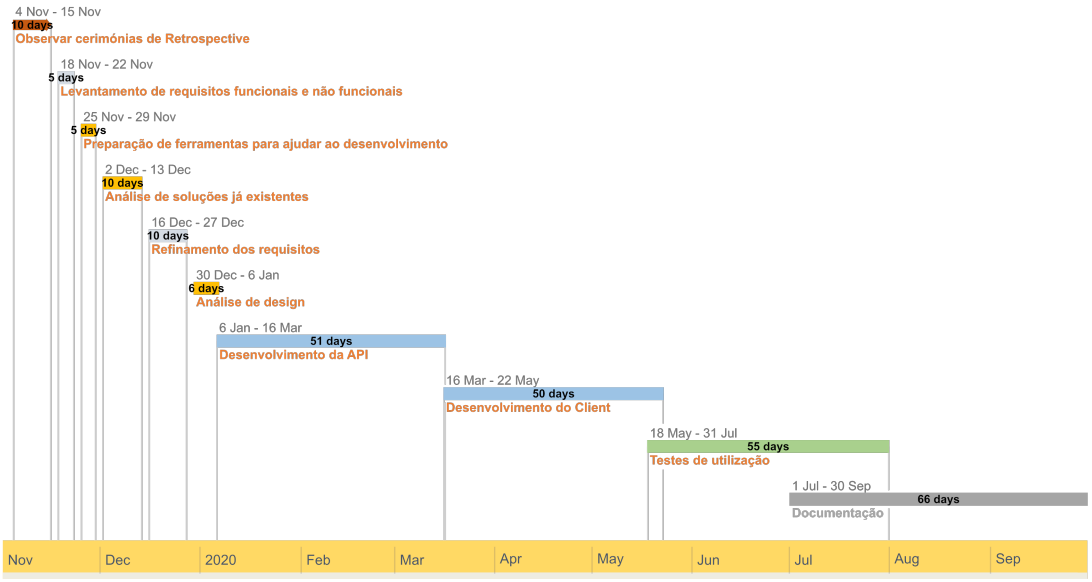


Figura 1.1: Diagrama de *Gantt* para o planeamento do projeto.

1.6 Estrutura do documento

No primeiro capítulo poderemos encontrar uma breve introdução que tem como objetivos contextualizar a empresa onde o projeto foi desenvolvido bem como a área do projeto. Neste capítulo podemos ver ainda qual o problema a ser trabalhado, quais os objetivos, qual a abordagem a adotar bem como qual o planeamento do projeto.

Num segundo capítulo temos um enquadramento teórico e tecnológico onde são apresentados os conceitos essenciais ao desenvolvimento do projeto para que o leitor possa enquadrar-se no tema. É ainda abordado o mercado existente no que se refere a soluções semelhantes.

No terceiro capítulo poderemos encontrar a proposta de valor que este projeto tem, bem como a oportunidade de negócio que o mesmo poderá ser.

O quarto capítulo destina-se ao estudo e analise das mais diversas opções de *design*. É neste capítulo que se analisa todas as possíveis abordagens ao problema bem como determinar qual a mais vantajosa e adequada à solução a ser desenvolvida.

Num quinto capítulo é relatado a forma como a solução irá ser submetida a avaliação, não só a nível de qualidade de *software* bem como testes funcionais e validação de objetivos.

1.7 Sumário

Conseguimos então com este capítulo perceber de forma generalista qual é o problema que existe e como é que o pretendemos resolver.

Em suma conseguimos verificar que existe a possibilidade de uma oportunidade de negócio que será analisada de forma mais profunda nos capítulos seguintes.

Capítulo 2

Estado de arte

No seguinte capítulo é abordado um estudo da arte em relativo ao tema do projeto, para tal, serão descritas e estudadas as metodologias de trabalho dinâmicas *Heavyweight* e *Agile*.

Posteriormente serão abordados os temas relativos ao desenvolvimento tecnológico das aplicações de forma nativa e o desenvolvimento de *Web Apps*.

Será efetuado um levantamento de estudo relativamente às tecnologias mais indicadas para desenvolver a solução, sendo analisado tecnologias para o desenvolvimento de *backend* e *frontend*.

Por fim será exposta uma analise detalhada de soluções já existentes que são utilizadas para ajudar na execução de cerimónias de *Retrospective*.

2.1 Enquadramento teórico

2.1.1 Metodologia

Metodologia é a arte de orientar a investigação da verdade, é um conjunto de regras ou normas utilizadas no ensino de uma ciência, é a parte da lógica que estuda os métodos (*metodologia | Definição ou significado de metodologia no Dicionário Infopédia da Língua Portuguesa* 2003).

Metodologia é a teorização sistémica de métodos a serem aplicados a um determinado estudo. A metodologia é benéfica por conseguir fornecer informações para planear, rever e controlar projetos. Geralmente têm por base quatro elementos: fornecimento de uma opinião do que precisa de ser resolvido, definição de técnicas sobre o que é necessário fazer e quando o fazer, encaminhar a gestão de qualidade do produto e por fim o provimento das ferramentas necessárias para o decorrer do projeto (Ishak 2005). Assim sendo uma metodologia não pretende desenvolver uma solução para um problema mas sim um método de trabalho.

A metodologia aplicada ao meio informático só teve aparecimento na década de 1960. Segundo Geoffrey Elliott o ciclo de vida de desenvolvimento de *software* pode ser considerado na sua essência uma metodologia, pois o mesmo tem como base o planeamento, criação, teste e implantação de um sistema de informação (Elliott 2004).

2.1.2 Ciclo de Vida do Desenvolvimento de Sistemas

Com vista ao desenvolvimento de soluções com qualidade o ciclo de vida do desenvolvimento de sistemas é constituído por uma série de fases que normalmente estão claramente definidas para serem usadas pelas equipas de trabalho (*Systems Development Life Cycle from FOLDOC* 2000).

Com o desenvolvimento das tecnologias acontece também o aumento consecutivo da complexidade dos processos de produção, para tal, e para ajudar a facilitar este processo foram sendo criadas metodologias que visam ajudar no ciclo do desenvolvimento. Algumas dessas metodologias são, Cascata, Espiral, Prototipagem Rápida, Desenvolvimento Iterativo e Incremental ou *Agile* (Berra s.d.).

Apesar do ciclo de vida do desenvolvimento não ser por si só uma metodologia, o mesmo assenta numa serie de fases que as próprias metodologias têm. Sendo então as seguintes (Usdoj 2006):

- Fase de Inicialização: a fase de inicialização começa quando se identifica a necessidade ou oportunidade de um determinado projeto. Nesta fase inicial deve se escolher o chamado *Project Manager* ou *Product Manager*. Todo este trabalho inicial deve ser documentado e só quando aprovado é que se dá inicio à fase seguinte.
- Desenvolvimento do Conceito do Sistema: após a validação da fase de inicialização é então dado inicio à avaliação das abordagens possíveis ao projeto. Nesta fase é analisado o âmbito do sistema do projeto e é onde são efetuadas as requisições e aprovações de financiamento que permitem avançar para a próxima fase.
- Fase de Planeamento: é aprofundado o conceito com vista a detalhar as necessidades para garantir que o produto desenvolvido cumpra os requisitos a que se propõe nos prazos e orçamentos definidos. É nesta fase que se definem as etapas do projeto, bem como os requisitos necessários para a elaboração do mesmo.
- Fase de Análise de Requisitos: neste momento é definido formalmente os requisitos em relação ao produto. Todos os requisitos são dissecados ao ponto de detalhe necessário para a próxima fase a do *design*. Todos os requisitos levantados têm que ser mesuráveis e testáveis.
- Fase de *Design*: as características físicas do produto são então desenhadas nesta fase. É especificado o ambiente de operação do sistema ou produto e os processos são distribuídos em função dos recursos. Todos os dados recolhidos, estudados e analisados têm que ser documentados nesta fase. Também nesta fase podem ser encontrados outros sub-sistemas necessários ao sistema principal.
- Fase de Desenvolvimento: todos os detalhes especificados anteriormente são postos em prática. Todo o *software* desenvolvido deverá ser testado (testes unitários e integração), e o processo de testes deverá ser repetitivo e sistemático;
- Fase de Integração e Teste: todos os diversos componentes do sistema são então integrados e testados sistematicamente. Nesta fase são integrados os testes com os utilizadores de forma a analisar o cumprimento das funcionalidades definidas anteriormente;
- Fase da Implementação: todo o sistema é instalado e posto operacional num ambiente de produção. Esta fase só é iniciada após a fase de testes ser aceite;

- Fase das Operações e Manutenção: o sistema está em operação e é monitorizado de forma contínua em relação à sua performance de acordo com o requisitos levantados e caso surjam modificações as mesmas vão sendo implementadas. O sistema é periodicamente submetido a revisões a fim de se estudar possíveis melhorias para o tornar mais eficiente e eficaz. O sistema pode sofrer metamorfoses conforme as necessidades da empresa. Quando aparecem novas modificações é necessário voltar ao ponto onde são definidas as etapas e os requisitos do projeto, isto acontece para que as modificações que ocorram sejam obrigadas a passar por todas as fases após o planeamento;
- Fase da Disposição: esta fase é responsável pela preservação de informações vitais sobre o sistema para caso surja a necessidade de reativação no futuro. É tido em especial cuidado a preservação dos dados do sistema para caso seja necessário uma migração para um outro sistema, ou para o caso do armazenamento ter de cumprir regulamentos e politicas que o exigem para futuras consultas.

2.1.3 Metodologias Aplicadas às Tecnologias

Do ponto de vista tecnológico surge então dois grandes grupos de metodologias: Metodologias *Heavyweight* e metodologias *Lightweight* (Awad 2005).

Metodologias Heavyweight

Metodologias *Heavyweight* como o nome indica são metodologias em que o processo de desenvolvimento é pesado e estas são consideradas a forma tradicional de desenvolvimento de *software*. Estas metodologias têm por base uma série de passos sequenciais tais como, definição de requisitos, construção da solução, teste e *deployment*. Existe um leque diverso de metodologias *Heavyweight*, contudo as três mais significantes são a Cascata (*Waterfall*), Modelo Espiral (*Spiral Model*) e Processo Unificado (*Unified Process*) (Awad 2005).

- *Waterfall*: Na década de 60 o método utilizado pelos programadores era *"code and fix"* (Awad 2005). Então tendo em conta todas as dificuldades que esta abordagem tinha, Winston Royce em 1970 apresentou a metodologia *Waterfall*.

  Esta metodologia promove a progressão estruturada por fases, em que cada fase consiste num conjunto definido de tarefas e artefactos que têm que ser executados antes de se iniciar a próxima fase.

  As fases podem ter nomenclaturas diferentes conforme os utilizadores contudo assentam sempre nos mesmos propósitos, a primeira fase é a que questiona o que é que o sistema terá que fazer, ou seja, é a fase do levantamento de requisitos, a segunda fase determina como é que a solução irá ser desenhada, a terceira é a fase do desenvolvimento, a quarta é a fase de testes e a ultima é a fase da implantação (Awad 2005).

  Na Figura 2.1 podemos então ver uma sistematização de como a metodologia *Waterfall* funciona.

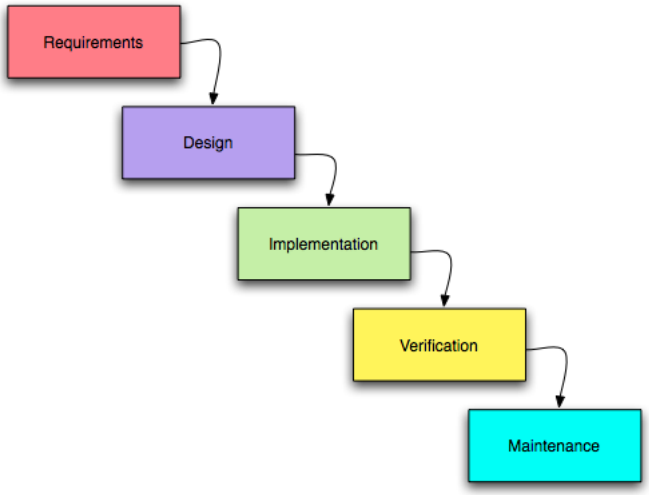


Figura 2.1: Waterfall Lifecycle (Hughey 2017).

- *Spiral Model*: A metodologia *Spiral* combina elementos de *design* e prototipagem em fases, com a finalidade de juntar as vantagens de conceitos *top-down* e *bottom-up*. O *Spiral Model* foi criado por Barry Boehm, onde este se baseou em experiências de refinação da metodologia *Waterfall* aplicada a grandes projetos (Awad 2005).

  Esta metodologia tem por base quatro fases: Criação de Objetivos onde são especificados os objetivos do projeto, Avaliação e Redução dos Riscos que visa a identificação e analise de riscos para o projeto, Desenvolvimento e Validação onde é escolhido um modelo para o desenvolvimento e por fim o Planeamento (Awad 2005).

  A Figura 2.2 é a representação gráfica de como a metodologia *Spiral* funciona.

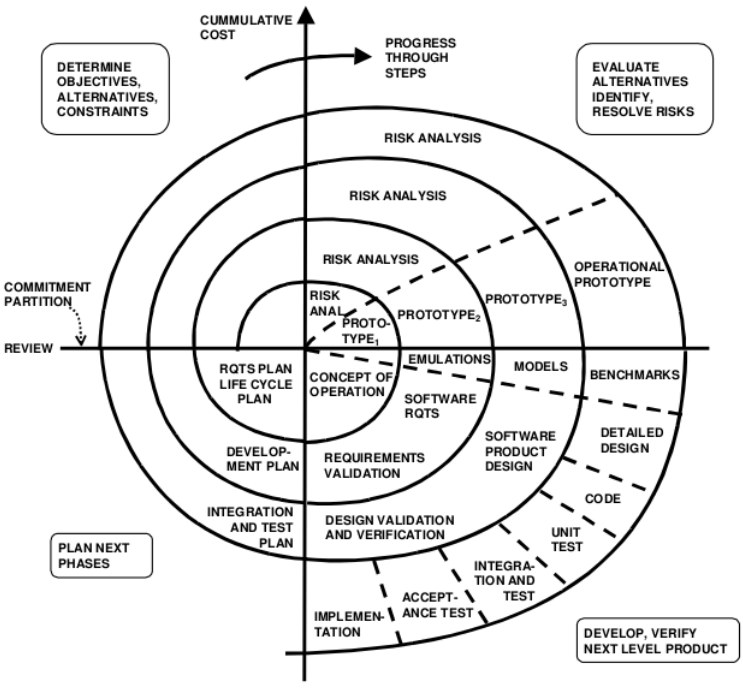


Figura 2.2: Spiral Model Lifecycle (Dmitry Gurendo 2015).

- *Unified Process (UP)*: Na metodologia *UP* todos os esforços são organizados num *workflow* e desenvolvidos de forma iterativa e incremental. A Figura 2.3 apresenta o ciclo de vida de um projeto que utiliza a metodologia *UP*.

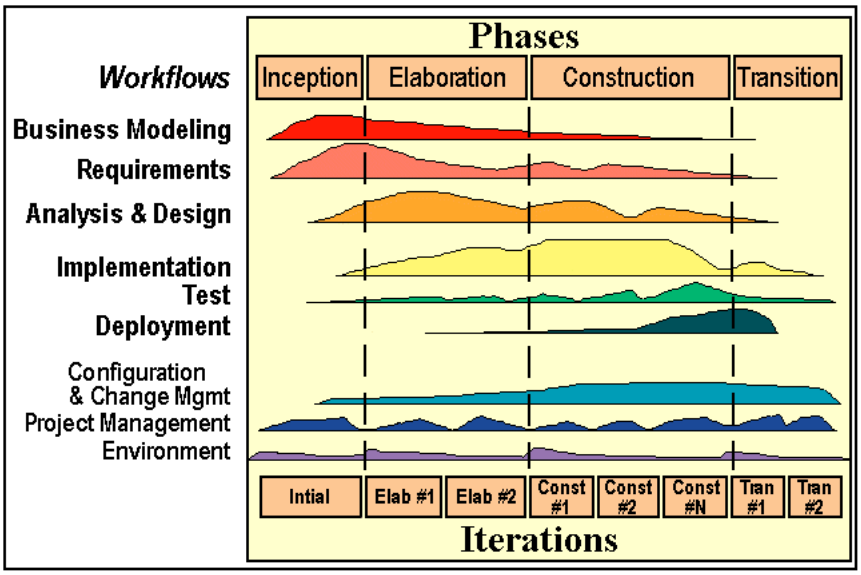


Figura 2.3: Unified Process Lifecycle (*Software Development Life Cycle - UML Tutorial for Beginners* 2013).

Então o ciclo de desenvolvimento tem por base 4 fases, Iniciação (*Inception* na figura) no final desta fase deverá obter-se um caso de negócio onde é avaliado a viabilidade do projeto e o âmbito do mesmo é definido, Elaboração (*Elaboration*) é aqui que são iniciados os esboços básicos da arquitetura do sistema é ainda aqui que deverá ser efetuada uma análise dos riscos que deverão ser tidos em conta, Construção (*Construction*) nesta fase é pressuposto que se encontre disponível uma versão *beta* do sistema que será utilizada para testes preliminares sobre situações reais, por fim temos a Transição (*Transition*) que é onde o sistema é introduzido aos utilizadores (Awad 2005).

Metodologias Lightweight

Como resposta aos desafios colocados pelas metodologias tradicionais nasceram as metodologias *Lightweight* denominadas de metodologias *Agile* (Javanmard e Alian 2015).

Agile utiliza um diversificado leque de táticas que visam superar as limitações das dinâmicas naturais do desenvolvimento de *software*. Esta metodologia encara os problemas adjacentes às metodologias tradicionais usando dois conceitos científicos que são, controlar os processos de forma empírica e o outro conceito pretende se que se desenvolva as equipas e os métodos tendo sempre em consideração que o desenvolvimento de sistemas é um processo complexo e adaptativo (Javanmard e Alian 2015).

Agile foi desenvolvido em cima de um manifesto elaborado pelos seus criadores que é o seguinte:

«**Manifesto para o Desenvolvimento Ágil de Software.**

Ao desenvolver e ao ajudar outros a desenvolver software,
temos vindo a descobrir melhores formas de o fazer.
Através deste processo começámos a valorizar:

Indivíduos e interacções mais do que processos e ferramentas

Software funcional mais do que documentação abrangente

Colaboração com o cliente mais do que negociação contratual

Responder à mudança mais do que seguir um plano

Ou seja, apesar de reconhecermos valor nos itens à direita,
valorizamos mais os itens à esquerda.» (Beck et al. 2001).

Um grande ponto diferenciador das metodologias *Agile* como se pode verificar no manifesto é o foco que as mesmas têm nas pessoas, ou seja são metodologias que se apoiam no desenvolvimento comunitário, colaborativo e promovem a auto-organização dos indivíduos intervenientes. *Agile* promove que cada interveniente de uma equipa de trabalho seja responsável pela sua própria organização, contudo, isto não significa que não existam cargos de gestão. Os gestores são supervisores das equipas de trabalho que as guiam quando as mesmas perdem o foco e são também responsáveis pela garantia de que as equipas têm os recursos necessários para a execução dos projetos (Agile Alliance 2019).

Agile é então caracterizado por ciclos curtos, com reflexões regulares, adaptação e integração contínua. O facto dos ciclos serem curtos permite que exista uma maior flexibilidade e agilidade quando existe necessidade de mudança de requisitos (Javanmard e Alian 2015).

Na Figura 2.4 podemos ver o mapa prático de *Agile*.

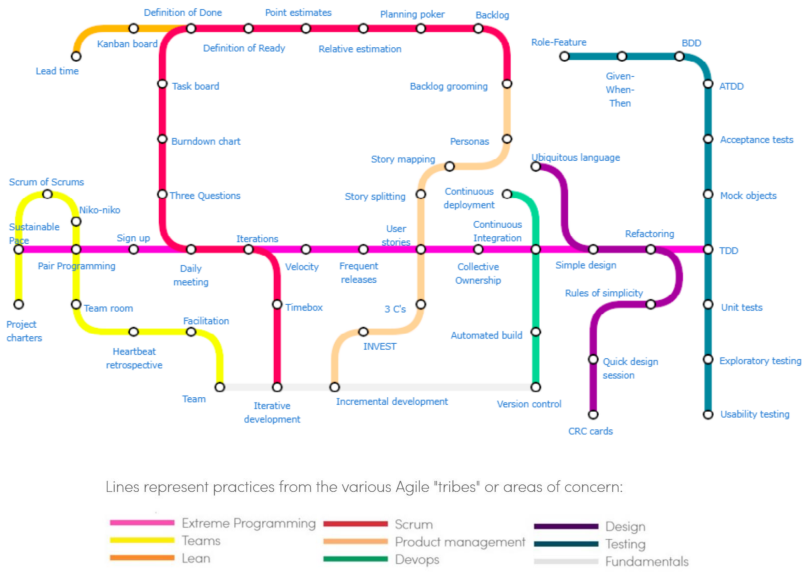


Figura 2.4: Mapa *Agile* (Scrum Alliance s.d.).

Algumas das metodologias que assentam em *Agile* são, *Extreme Programming*, *Kanban*, *Scrum*, *Feature-Driven Development* entre muitas outras. Então podemos ver em mais detalhe três das acima mencionadas:

1. *Extreme Programming (XP)*: *XP* é uma metodologia que se foca nas práticas para a construção de software e não na melhor forma de gerir o projeto como um todo. Esta metodologia adota um conjunto muito especifico de práticas de desenvolvimento de software, como *pair programming* (programação a pares), desenvolvimento guiado pelos testes e integração contínua (Javanmard e Alian 2015).

   A Figura 2.5 que se segue demonstra algumas das práticas que uma equipa deve adotar ao utilizar *Extreme Programming*.

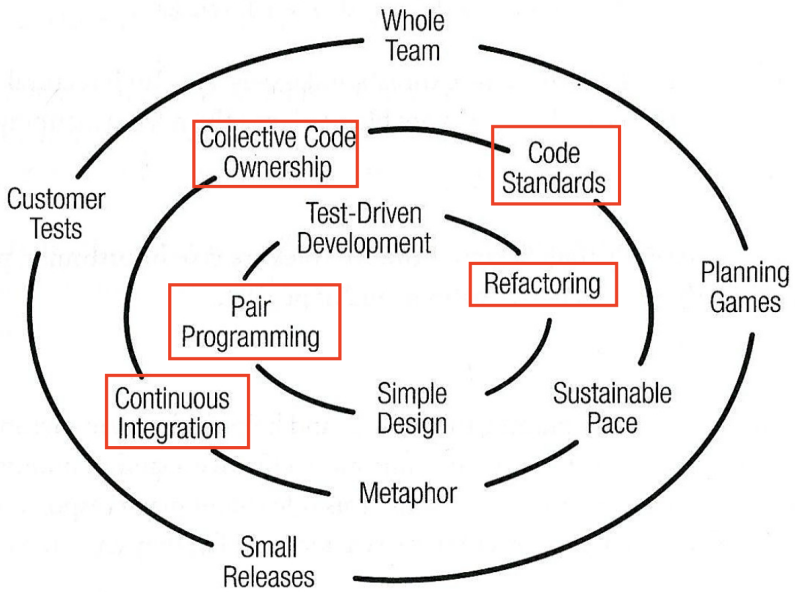


Figura 2.5: Práticas *Extreme Programming* (*Top 5 Extreme Programming (XP) Tools Every Team Should Use | Official Pythian® Blog* s.d.).

   Segundo Vinícius Manhães Teles (2014) *XP* tem por base quatro valores, *Feedback*, Comunicação, Simplicidade e Coragem. O *Feedback* está ligado ao constante retorno que o cliente envia que afeta diretamente todo o processo de desenvolvimento, a Comunicação é um ponto fulcral para o contacto direto com o cliente, a Simplicidade esta relacionada com a forma de comunicação com o cliente pois o nível de conhecimento tecnológico poderá não ser elevado e a Coragem está relacionado com a equipa, que tem que acreditar que utilizando metodologia *XP* conseguirá alcançar o sucesso no desenvolvimento de software (Teles 2014).

2. *Kanban*: *Kanban* tem a sua origem no *Toyota Production System* e o seu nome provem de uma palavra japonesa que significa *just-in-time*. Esta metodologia apoia o seu trabalho numa tabela ou quadro, conhecido por *Kanban board*, em que as colunas representam os fluxos da produção e na forma que o desenvolvimento vai evoluindo as informações na *board* vão sofrendo alterações (Sugimori et al. 1977).

   Kanban é uma metodologia que exige boa comunicação e transparência para que todos os elementos da equipa saibam exatamente o ponto de situação do desenvolvimento.

3. *Scrum*: *Scrum* nasceu na década de 1990 e é uma metodologia de desenvolvimento de *software* onde os problemas são abordados de forma adaptativa ao longo do tempo, é uma metodologia iterativa e incremental que segundo os criadores, *Scrum* é leve, simples de entender e difícil de dominar (Schwaber e Sutherland 2016).

   Esta metodologia faz uso de diversas técnicas e processos como facilitadores no processo de desenvolvimento. Estas técnicas visam tornar o desenvolvimento simples, claro e eficaz, focando na melhoria continua. É uma metodologia que tem foco nas pessoas e nas equipas de trabalho onde cada elemento tem um propósito e que todos são essenciais para o sucesso do desenvolvimento (Schwaber e Sutherland 2016).

   O *Scrum* tem por base muito do processo empírico que as metodologias *Agile* defendem, onde se acredita que o conhecimento é desenvolvido por via da experiência e que as decisões são tomadas em função do conhecimento.

   Scrum é iterativo e incremental o que faz com que consiga lidar de forma muito fácil com as imprevisibilidades do desenvolvimento, fazendo com que consiga entregar regulamente produtos com qualidade e com elevada quantidade de valor (F. M. Fowler 2019).

   A Figura 2.6 mostra como é a estrutura de trabalho quando se utiliza esta metodologia para o desenvolvimento de *software*.

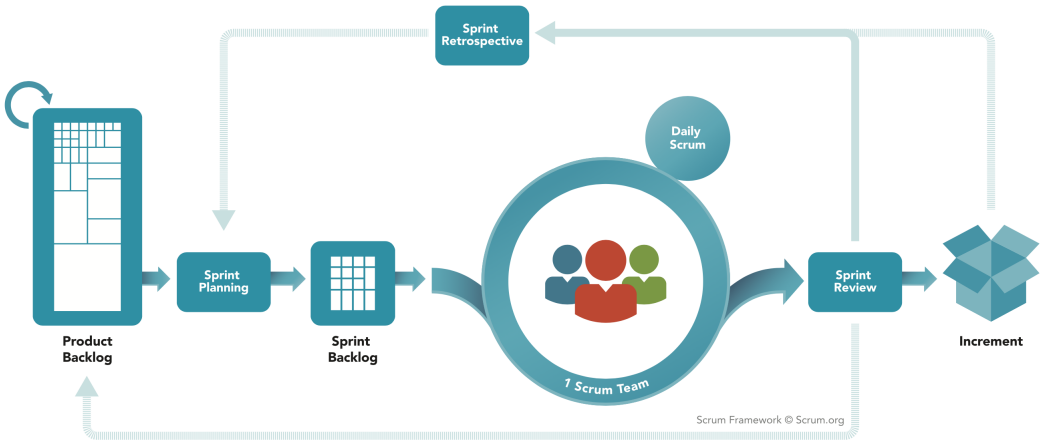


Figura 2.6: Estrutura da metodologia *Scrum* (F. M. Fowler 2019).

A metodologia *Scrum* tem por sua base uma serie de cerimónias que são repetidas regularmente ao longo do ciclo de vida do desenvolvimento, todas estas cerimónias têm funções muito especificas e visam ajudar a equipa a evoluir e não somente o projeto (F. M. Fowler 2019). As reuniões são agrupadas em *Sprints*, essencialmente uma janela temporal, que nunca deve durar mais de um mês e é onde a equipa tem mais foco e o produto realmente evolui. Assim sendo as cerimónias que devem acontecer ao decorrer de um *Sprint* são:

(a) *Sprint Planning Meeting*: É nesta cerimónia que se efetua o planeamento de um *Sprint*.

   A cerimónia de *Sprint Planning Meeting* deverá ser limitada ao máximo de 8 horas para um *Sprint* de um mês e nesta sessão deverão ser respondidas a perguntas

como, o que é que podemos desenvolver durante o próximo *Sprint* ou como é que vamos alcançar os objetivos planeados (Schwaber e Sutherland 2016).

(b) *Daily Scrum*: É uma reunião diária com a equipa de desenvolvimento que visa fazer um ponto de situação individual. Por norma tem uma duração de 15 minutos e todos os elementos da equipa têm que fazer o seu ponto de situação. Nesta cerimónia também é dito por cada elemento o que vai fazer nas próximas horas de trabalho.

(c) *Sprint Review Meeting*: A *Sprint Review Meeting* tem por função inspecionar o incremento que cada *Sprint* tem e ajustar o *Product Backlog* caso seja necessário. Esta sessão tem por função base não um ponto de situação mas sim a obtenção de *feedback* por todas as partes interessadas no produto (Schwaber e Sutherland 2016).

(d) *Sprint Retrospective Meeting*: Esta cerimonia é onde a equipa tem a oportunidade de se avaliar enquanto equipa mas também individualmente. O objetivo principal desta cerimónia é conseguir perceber quais os pontos mais fracos que a equipa apresenta para se poderem colmatar contudo, também são analisados os pontos positivos para que também se tomem ações que ajudem a continuar positivos (Schwaber e Sutherland 2016).

2.1.4 Lightweight VS Heavyweight

A Tabela 2.1 que se segue tem por fim mostrar de forma sucinta algumas das principais diferenças entre os dois grandes grupos de tipos de metodologias.

Tabela 2.1: As principais diferenças entre as metodologias *Lightweight* (*Agile*) e *Heavyweight* (Akbar et al. 2018).

	Lightweight	**Heavyweight**
Abordagem	Adaptativa	Preditiva
Medição do Sucesso	Valor do negócio	De acordo com o plano
Dimensão do Projeto	Pequeno	Grande
Perspetiva da Mudança	Mudança na adaptabilidade	Mudança na sustentabilidade
Cultura	Liderança e colaboração	Comando e controlo
Documentação	Pouca	Muita
Ênfase	Orientado para as pessoas	Orientado para os processos
Ciclos	Numerosos	Limitados
Domínio	Imprevisível e exploratório	Previsível
Planeamento Adiantado	Mínimo	Compreensivo
Retorno do Investimento	Início do projeto	Fim do projeto
Dimensão da equipa	Pequena e criativa	Grande

2.1.5 Adoção de Metodologias Agile

Segundo o estudo «13th annual State of Agile» (Version One 2019) levado a cabo pela *CollabNet VersionOne* analisou 1,319 respostas dadas entre Agosto e Desezmbro por empresas ligadas ao ramo tecnólogico e outras áeras, onde foi concluído nesse estudo que 97% das

organizações que responderam utilizam metodologias *Agile*. Podemos então ver de seguida alguns gráficos relativamente as metodologias mais usadas (Version One 2019).

Figura 2.7: Motivos pelos quais escolhem *Agile* (Version One 2019).

A Figura 2.7 mostra que as empresas estão menos preocupadas com o aumento da produtividade em comparação com o ano anterior onde o valor era de 55% contras os 51% do ano de 2019 podemos ainda verificar que um dos pontos fulcrais para as empresas que participaram no estudo é que as metodologias de trabalho melhorem a velocidade de entrega dos produtos. Um outro dado analisado é que no ano anterior ao do estudo a preocupação com a moral das equipas de trabalho pela parte das empresas era de 28% enquanto em 2019 este valor aumentou para os 34% (Version One 2019).

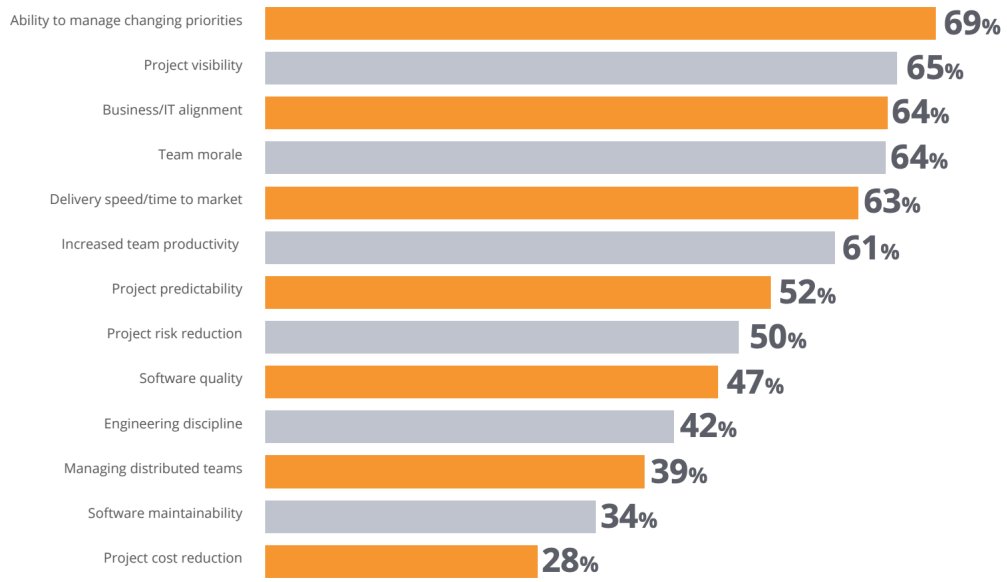


Figura 2.8: Benefícios da escolha de metodologias *Agile* (Version One 2019).

Na Figura 2.8 mostra essencialmente quais os benefícios que as empresas vêem ao adoptarem metodologias *Agile* para as suas equipas de trabalho. É claro que o ponto com mais destaque está relacionado com a capacidade que estas metodologias têm no que toca à gestão da imprevisibilidade que os projetos podem tomar. Um dado que também se verifica como na figura anterior é o da moral das equipas em que no ano anterior tinha uma expressão de 61% enquanto em 2019 este valor subiu para 64% (Version One 2019).

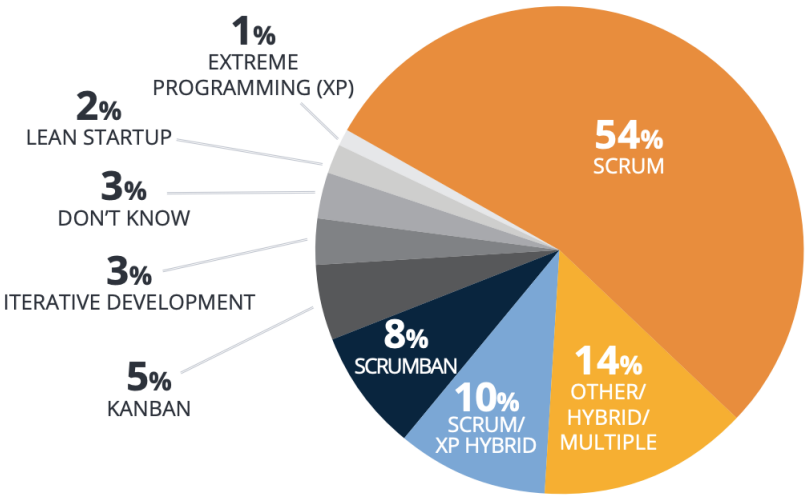


Figura 2.9: Metodologias *Agile* mais utilizadas (Version One 2019).

A Figura 2.9 mostra então quais são as metodologias mais utilizadas pelas empresas, que as metodologias *SCRUM* e *SCRUM/XP HYBRID* têm um total de 64% de adoção pelas empresas (Version One 2019).

2.1.6 Sprint Retrospective Meeting

Como descrito anteriormente a *Sprint Retrospective Meeting* é o ponto de retrospectiva de uma equipa. A *Retrospective* tem por finalidade a criação de um plano para melhorias a serem integradas no próximo *Sprint*.

Por norma a *Retrospective* ocorre após a *Sprint Review Meeting* e antes da *Sprint Planning Meeting*, e a mesma deverá ter uma duração de cerca de três horas para um *Sprint* de um mês. Durante um cerimónia de *Retrospective* a equipa deve ter em análise três grandes questões: O que é que correu bem? O que é que pode ser melhorado? Que ações podem ser feitas para melhorar no próximo *Sprint*?. Toda a cerimónia deve ser gerida pelo *Scrum Master* que é a pessoa responsável pela gestão da equipa de *Scrum*, e este tem por função base tornar a cerimónia o mais agradável possível aos intervenientes (SCrum.org 2010).

Durante a *Retrospective* a equipa planeia formas de aumentar a qualidade do produto melhorando o processo de trabalho. No final de uma sessão é pressuposto que a equipa tenha conseguido encontrar pontos que deve melhorar e ações a implementar no próximo *Sprint*, contudo não necessita de ser obrigatoriamente implementado no próximo, pois algumas das ações podem ser melhorias a implementar a longo prazo (SCrum.org 2010).

Os autores Maciej Wawryk e Yen Ying Ng concluíram que ao tentar implementar cerimónias de *Retrospective* baseadas no jogo conseguiram aumentar a popularidade das mesmas obtendo um aumento na participação por parte das equipas de trabalho (Wawryk e Ng 2019).

Segundo um estudo que analisou a performance de equipas de trabalho concluiu que as equipas que tinham uma performance mais baixa tinham também associado a má prática ou mesmo a inexistência de efetuarem regularmente sessões de *Retrospective*. O estudo conclui ainda que a elaboração recorrente e adequada de sessões de *Retrospective* consegue mudar o processo de trabalho das equipas levando consequentemente ao aumento de performance. Uma das principais diferenças que se encontrou entre equipas de alta e baixa performance e as suas sessões de *Retrospective* é a duração das mesmas onde as equipas com melhores performance tinham sessões mais longas. O estudo cita ainda colaboradores dizendo: «Without retros we would not be here, we changed the working practices a lot [based on retros], e.g., meeting a lot face to face. We would have stood still without them.» (Paasivaara et al. 2017).

O estudo *Software Metrics Classification for Agile Scrum Process* consegue ainda afirmar que o uso de *Retrospective* tem impacto direto no sucesso do projeto (Kurnia, Ferdiana e Wibirama 2018).

A Figura 2.10 mostra quais os passos principais que uma cerimónia de *Sprint Retrospective* deve ter por base.

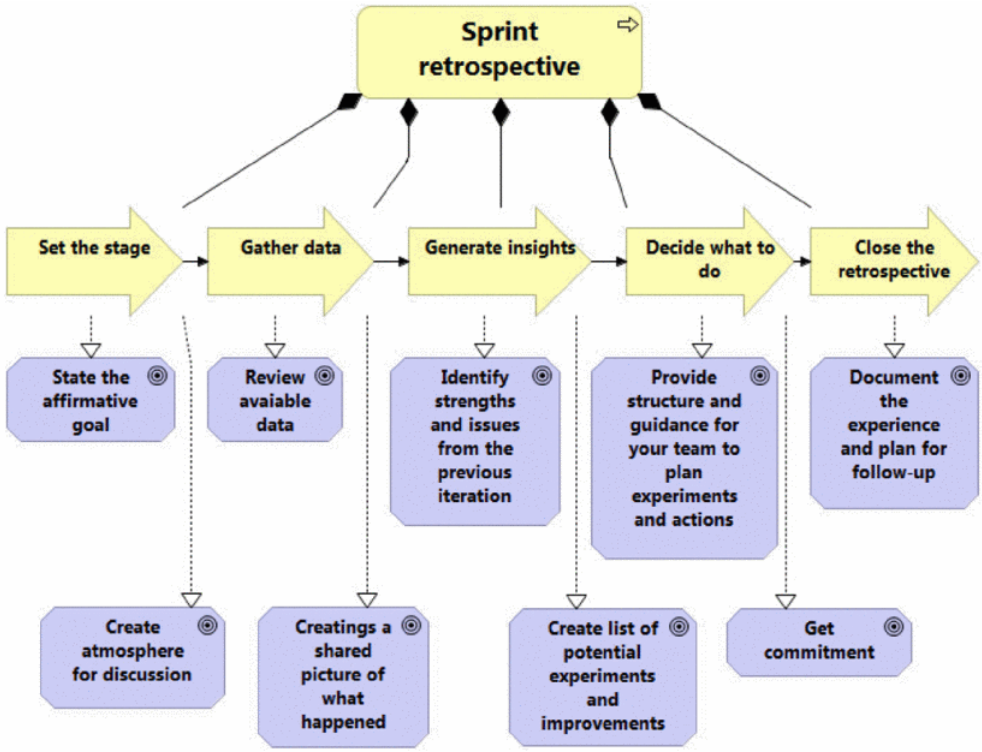


Figura 2.10: Passos de uma cerimónia de *Sprint Retrospective* (Werewka e Spiechowicz 2017).

2.2 Enquadramento tecnológico

2.2.1 Desenvolvimento Web

Web App

Uma *Web App* é uma aplicação desenvolvida para correr em servidores remotos através de um navegador de *Internet*, tal como o *Google Chrome* ou *Mozilla Firefox* ou ainda como o *Microsoft Edge*. As *Web Apps* podem ser usadas nos mais diversos sectores como serviços de *email* ou comercio *online*, por norma suportam múltiplos navegadores mas podem existir casos em que as mesmas só suportam um único. (Swain et al. 2016).

Como principais benefícios as *Web Apps* permitem assegurar que os utilizadores utilizam todos a mesma versão, não é necessário instalação na máquina do utilizador, a facilidade de acessibilidade pois podem ser usadas usadas em dispositivos móveis bem como em computadores e o facto de poderem ser acedidas através de vários navegadores de *Internet* (*What is Web Application (Web Apps) and its Benefits* s.d.).

Desenvolvimento: Web App ou Nativo

Aplicações nativas são aquelas que são desenvolvidas para funcionarem num sistema específico, sendo desenvolvidas especificamente para o dispositivo de destino (*What is Web Application (Web Apps) and its Benefits* s.d.).

As aplicações nativas fazem sentido quando o produto desenvolvido tem que tirar partido de características físicas e especificas do dispositivo para a qual foi desenvolvida, como por exemplo o uso de *Global Positioning System (GPS)* para equipamentos móveis.

Contudo as aplicações nativas exigem que o utilizador tenha um dispositivo ao qual a aplicação se destina, e que posteriormente o utilizador descarregue e instale a mesma. Do ponto de vista de desenvolvimento as aplicações nativas por norma exigem mais esforço por parte das equipas de desenvolvimento no caso de a aplicação ter de funcionar em mais do que uma plataforma e por norma o custo de desenvolvimento também é mais elevado (Buettner e Simmons 2011).

Na Tabela 2.2 podemos ver quais os principais pontos a ter em conta quando se está no momento de decisão da abordagem a ter entre *Web App* e Nativo.

Tabela 2.2: Pontos a serem considerados a quando a escolha da abordagem de desenvolvimento (Serrano, Hernantes e Gallardo 2013).

Considerações	**Nativo**	**Web App**
Esforço no suporte de plataformas e versões	Alto	Baixo
Acesso às funcionalidades totais do dispositivo	Total	Parcial
Experiência do utilizador (*User Experience - UX*)	Total	Média
Performance	Muito alta	Alta
Atualizações no cliente	Necessita	Não necessita
Facilidade de distribuição e publicação	Média	Alta
Ciclo de aprovação	Obrigatório	Não necessita
Monetizarão na loja das aplicações	Disponível	Não disponível

Web Design Responsivo

Uma disciplina chamada de arquitetura responsiva começou a questionar de que forma os espaços físicos poderiam responder à presença das pessoas. Com a combinação de robótica e materiais manipuláveis, os arquitetos com criações artísticas e estruturas flexíveis experienciaram obras que reagem à aproximação de pessoas. Sensores de movimento com sistemas de controlo de ambiente conseguem ajustar por exemplo temperatura ou a luminosidade de um espaço conforme a quantidade de pessoas num dado espaço (Marcotte 2010).

Em relação ao meio tecnológico também nasceram oportunidades para soluções responsivas.

Web Design Responsivo é um conceito que faz uso de *Cascading Style Sheets (CSS)*[1] e *Media Queries* [2] para determinarem a resolução do dispositivo usado e ajustar a apresentação do conteúdo da página *web* de acordo com essas dimensões. Ou seja o *Design* Responsivo faz com que os conteúdos de uma página *web* mudem as suas dimensões de acordo com as dimensões do destino de apresentação para serem apresentados da melhor forma possível, por exemplo, o *Design* Responsivo faz com que o conteúdo de uma pagina se ajuste quando o utilizador redimensiona a janela do navegador que está a utilizar (Jobe 2013).

Com a utilização do principio de *Web Design* Responsivo é possível otimizar a experiência de visualização para o utilizador (Marcotte 2010).

Na Figura 2.11 podemos ver um caso lado a lado de como o comportamento de uma página se transforma em função da dimensão do destino de apresentação.

Figura 2.11: Na imagem da esquerda temos a página apresentada num computador e no lado direito a mesma página mas apresentada num dispositivo móvel. Adaptado de (Editorial 2011).

A Figura 2.12 é de um estudo levado a cabo pela *Lanars* que mostra o crescimento de plataformas *Web* responsivas nos últimos anos (Mariia Lozhko 2019).

[1] *Cascading Style Sheets (CSS)*: linguagem de estilo que é usada para descrever a apresentação de paginas *web*.

[2] *Media Queries*: são regras de que o *CSS* faz uso para se ajustar aos diferentes tamanhos dos dispositivos.

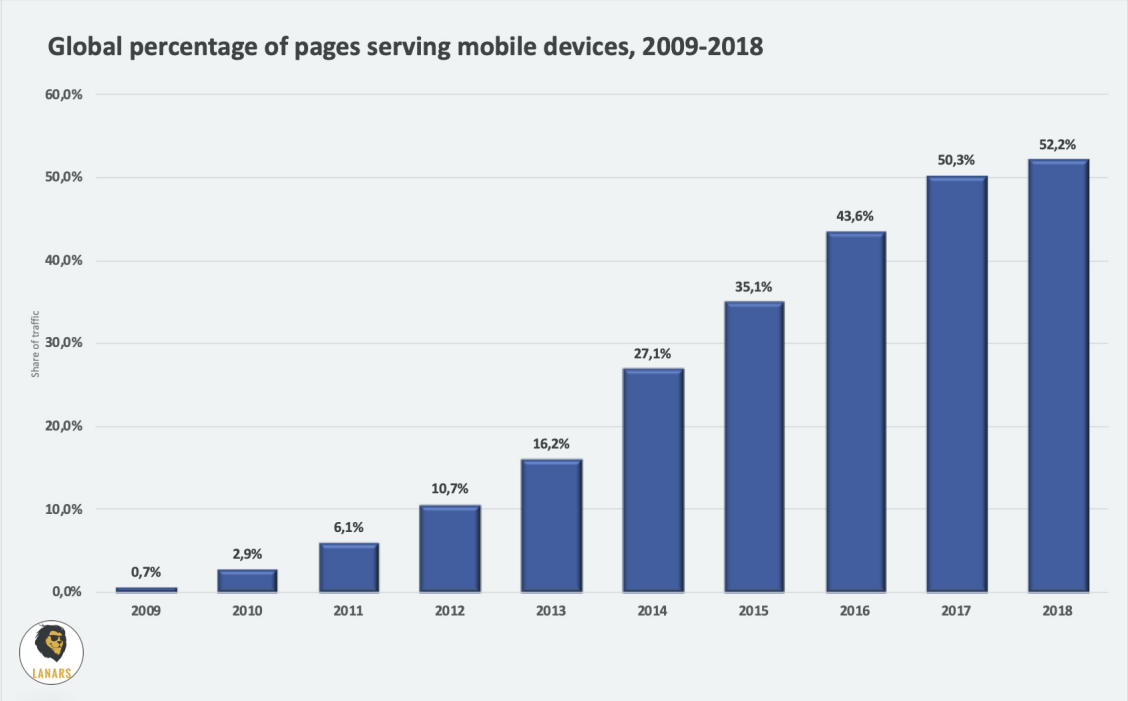


Figura 2.12: Crescimento das plataformas*Web* responsivas (Mariia Lozhko 2019).

2.2.2 Tecnologias de Backend

Java

Java é uma linguagem de programação interpretada orientada a objetos, de plataforma independente, portátil e robusta. Foi desenvolvida para ser utilizada como ambiente de distribuição da *Internet*. O código escrito em *Java* pode ser executando em qualquer *Java Virtual Machine* e é uma linguagem bastante flexível. É maioritariamente utilizada em aplicações móveis *Android* mas também é usada para cartões inteligentes, vídeo-jogos, sistemas integrados e robótica (Hartwich 2018).

Alguns pontos positivos de *Java* é o facto de ser uma linguagem sólida devido ao facto de já estar à mais de 20 anos no mercado, existe um vasto ecossistema de bibliotecas que a podem potencializar e um elevada capacidade computacional.

Do lado dos pontos negativos temos um processo de desenvolvimento lento, é necessário ter extremo cuidado na clareza e organização de escrita, elevada configuração de ferramentas e problemas com migração de código.

Node

Node.js é *JavaScrip* a correr em tempo real no motor de *JavaScript Chrome V8*. É uma linguagem acarinhada pelo seu elevado desempenho e escalabilidade. É uma tecnologia utilizada em aplicações *Web*, aplicações empresariais, soluções de analise de grandes quantidades de dados e sistemas integrados (Hartwich 2018).

Do ponto de vista de vantagens temos o facto da curva de aprendizagem ser gentil, principalmente para quem já tem conhecimentos de *JavaScript*, é de desenvolvimento rápido,

inclui ferramentas de reutilização de código e permite que o programador com uma única linguagem, *JavaScrip* consiga construir o servidor e o cliente de uma aplicação.

Relativamente a pontos negativos existe uma falha de ofertas de IDEs [3] nativos para o desenvolvimento em *Node.js* e como se trata de uma tecnologia recente contrariamente a *Java* a sua confiabilidade pode ser posta em causa.

Java VS Node

Na Tabela 2.3 compara as tecnologias ligadas ao desenvolvimento de *Backend*.

Tabela 2.3: Tabela comparativa entre *Java* e *Node.js*

	Java	**Node.js**
Alvo	Orientada a objetos	Tempo real
Desenvolvimento	Lento	Rápido
Curva de Aprendizagem	Elevada	Baixa
Oferta de IDEs	Elevada	Baixa
Principal uso	Aplicações *Andorid*	*Web*

2.2.3 Tecnologias de Frontend

Vue

Vue é uma *framework* criada para o desenvolvimento de aplicações *frontend* é escrita em *JavaScrip*, e nasceu devido à necessidade de organizar e simplificar o desenvolvimento *Web*. É caracterizada pela composição em componentes visuais e pela renderização declarativa. Tem como apoio bibliotecas que têm como principal função alimentar aplicações complexas.

Angular

Angular é pertencente à *Google* e foi adquirido para entrar no mercado do desenvolvimento de *Web Apps*. O seu nascimento surgiu da necessidade de facilitar a criação de *Web Apps* e garantir que as mesmas tinham comportamentos responsivos. Nasceu tendo por base *JavaScrip* contudo numa das actualizações foi trocado para *TypeScript*.

React

React é das tecnologias de desenvolvimento de *Web Apps* que tem mais popularidade. Foi criada e mantida pela *Facebook* e tem por base *JavaScrip*. Com o sucesso do *React* foi ainda criada uma versão especifica para o desenvolvimento de aplicações híbridas chamada de *React Native* que tem por grande função otimizar a usabilidade do utilizador em relação ao dispositivo onde executa a aplicação.

Na Figura 2.13 podemos ver o número de *downloads* nos últimos seis meses das três tecnologias. Na imagem é claramente notório que *React* está largamente mais popular do que as restantes tecnologias.

[3] Integrated Development Environments (IDE)

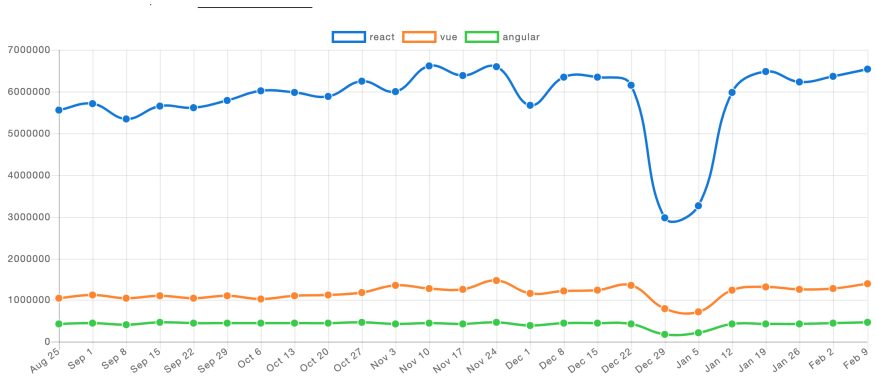


Figura 2.13: *Downloads* nos últimos 6 meses de *Vue.js*, *Angular* e *React* (Potter 2019).

Vue.js VS Angular VS React

A Tabela 2.4 tem por foco a comparação de tecnologias relativas ao desenvolvimento de *Frontend*.

Tabela 2.4: Tabela comparativa entre *Vue.js Angular* e *React*

	Vue.js	**Angular**	**React**
Tipo	*Framework*	*Framework*	*Framework*
Tipo de aplicações	Aplicações avançadas *SPA*	Aplicações nativas, híbridas ou *Web Apps*	Aplicações *SPA* ou moveis
Curva de aprendizagem	Pequena	Acentuada	Média
Modelo	Baseado em *Virtual DOM*	Baseado em arquitetura *MVC*	Baseado em *Virtual DOM*
Linguagem	*JavaScript*	*TypeScript*	*JavaScript*

Single Page Application (SPA)

Document Object Model (DOM)

Model-View-Controller (MVC)

A Figura 2.14 mostra um estudo levado a cabo pela *DevSkiller*, onde se verifica quais são as competências mais requisitadas para o ano 2020 do ponto de vista das empresas. Pode se verificar que as tecnologias que se baseiam em JavaScript são as mais procuradas (*Devskiller top IT skills report 2020: Demand and hiring trends* 2020).

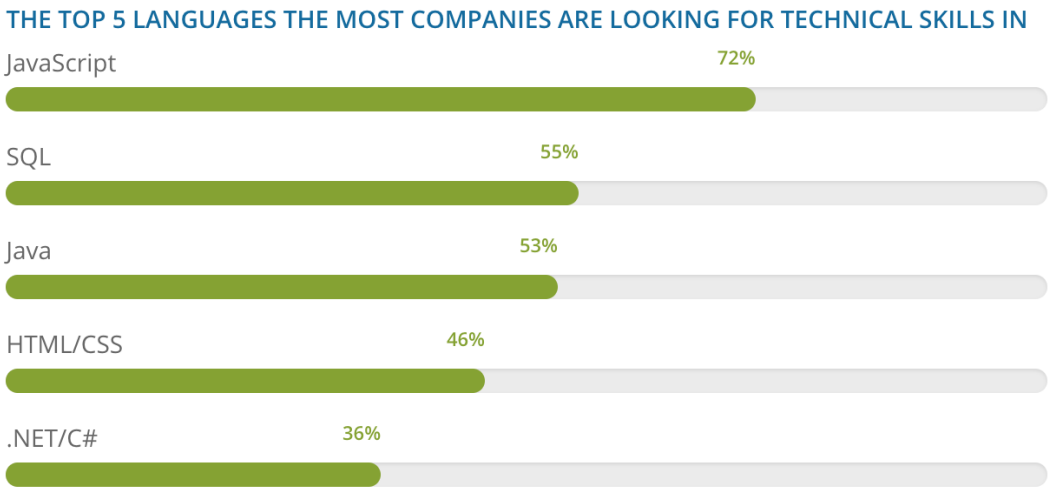


Figura 2.14: Tecnologias mais procuradas pelas empresas em 2020 (*Devskiller top IT skills report 2020: Demand and hiring trends* 2020).

Já na Figura 2.15 vemos dentro do *JavaScript* quais as tecnologias que têm mais relevo (*Devskiller top IT skills report 2020: Demand and hiring trends* 2020).

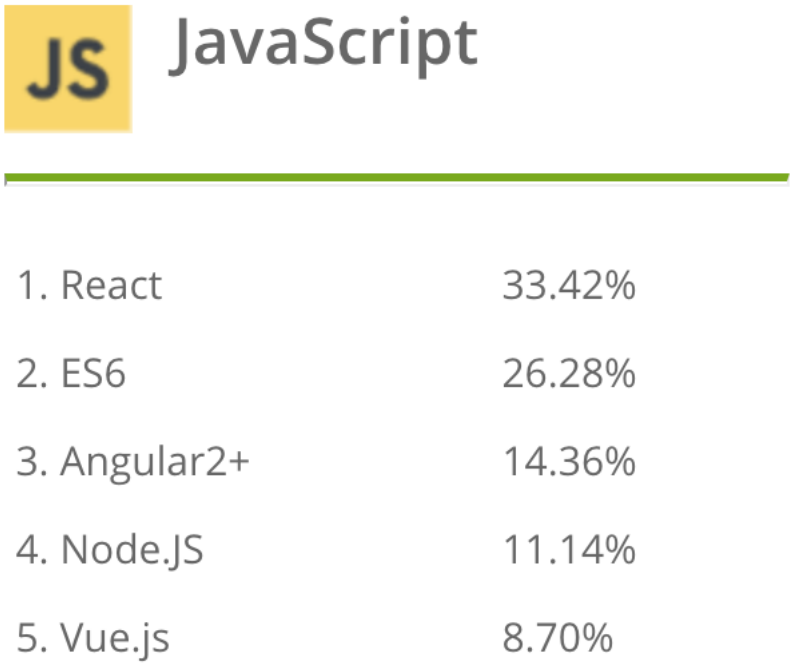


Figura 2.15: Tecnologias em *JavaScript* mais procuradas em 2020 (*Devskiller top IT skills report 2020: Demand and hiring trends* 2020).

2.2.4 Base de dados

MongoDB

MongoDB foi criado por Dwight Merriman e Eliot Horowitz após terem encontrado problemas de desenvolvimento e escalabilidade com as tradicionais base de dados relacionais. É uma base de dados *NoSQL* de acesso e de uso livre.

Bases de dados *NoSQL* são utilizadas como alternativas às base de dados relacionais e são úteis quando é necessário trabalhar com dados que estão distribuídos. *MongoDB* é uma ferramenta que consegue gerir informação orientada a documentos, armazena-los ou mesmo fornecer essas informações. Consegue suportar vários formatos de dados e em vez do tradicional uso de tabelas relacionais para o armazenamento de dados utiliza uma arquitetura de coleções e documentos que por norma utilizam um formato *JSON*.

Mongoose

De modo a facilitar o processo de implementação da base de dados *MongoDB* existem livrarias e uma delas é *Mongoose*.

Mongoose é uma biblioteca *Object Data Modeling (ODM* para *MongoDB* e *Node.js*, é quem gere o relacionamento entre os dados, validações dos esquemas de dados e é o tradutor entre o código escrito e a sua representação no *MongoDB*.

A Figura 2.16 é um diagrama de como este relacionamento acontece.

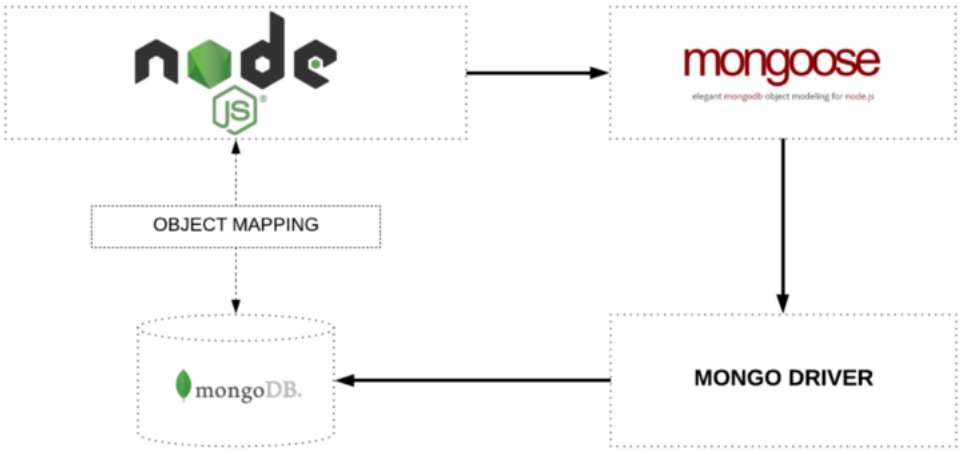


Figura 2.16: Relacionamento entre *MongoDB* e *Node.js* utilizando *Mongoose* (Nick Karnik 2018).

2.3 Análise de soluções existentes

No que toca a soluções de suporte às cerimónias de *Sprint Retrospective* a forma mais rudimentar é a utilização de *post-its*, contudo numa era tecnológica como é a de hoje já existem ofertas informáticas tais como, *Team O'Clock*, *Reetro*, *Retrium* ou então a *FunRetro* que é a aplicação que a equipa onde me insiro atualmente utiliza.

Post-its

Apesar de parecer a solução mais simples do ponto de vista técnico não é a solução que melhor se enquadra com as equipas de desenvolvimento dos dias de hoje.

Uma cerimónia apoiada em *post-its* não é uma cerimónia exequível. O grande motivo pela qual se torna impraticável é o aumento constante de elementos das equipas de trabalho que se encontram a trabalhar remotamente nos mais diversos pontos do planeta, assim sendo torna a utilização de cartões de papel pouco viável.

Com o aumento da proximidade dos clientes com as equipas de desenvolvimento estes também são parte interessada nos resultados obtidos de uma cerimónia de *Retrospective* pois o mesmo tem desejo que as equipas de desenvolvimento tenham cada vez mais sucesso e desenvolvam produtos com qualidade. Então a partilha de resultados de uma sessão apoiada em *post-its* torna esta tarefa ainda mais difícil ou até impossível.

Por fim como se trata de pequenos cartões de papel um problema da utilização de *post-its* o seu armazenamento para consultas futuras torna-se pouco prático, exigindo um alta capacidade de organização e espaço físico adequado para armazenamento. Mesmo existindo a possibilidade de se fotografar o placar, há a necessidade de se criar uma base de dados fotográfica que se torna pouco prático em questões de pesquisa e partilha de dados.

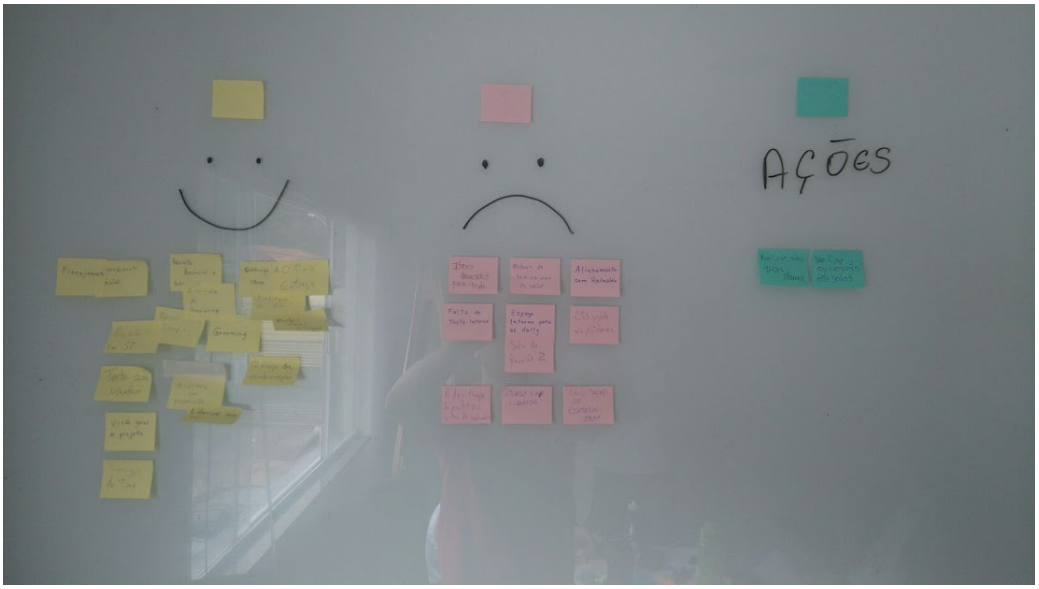


Figura 2.17: Exemplo de uma sessão de *Retrospective* utilizando *post-its* (Filho 2018).

Team O'Clock

Team O'Clock nasceu em 2016 e quando apareceu no mercado só tinha como função ajudar no suporte às *Daily Standup Meeting* através de aplicações de terceiros e só em Julho de 2018 é que começou a incorporar a possibilidade de se executar *Sprint Retrospective* (*Team O'clock for your meetings: The team* s.d.).

A aplicação disponibiliza um plano gratuito com limitações, só se podendo fazer uma sessão de *Retrospective* e tem um limite de 14 dias de histórico de atividades. Os planos pagos para uma empresa da dimensão da *Mindera* são sob consulta.

A aplicação tem um *design* moderno e permite a utilização de *templates* que permitem facilitar o inicio da sessão. Oferece também a possibilidade de integração com aplicações externas e ainda tens mais funcionalidades para ajudar as equipas para além das sessões de *Sprint Retrospective*.

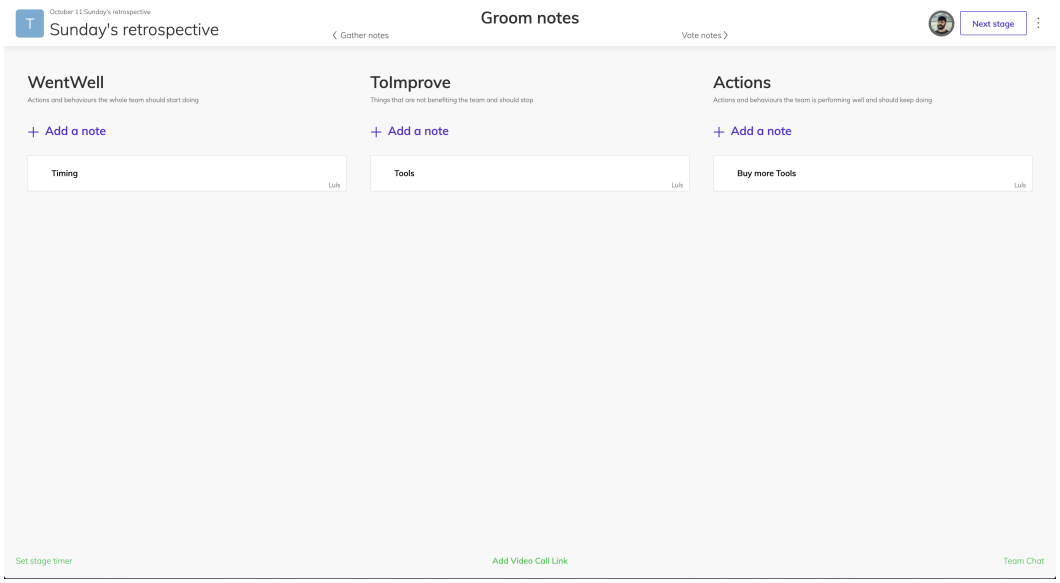


Figura 2.18: Ecrã do *Team O'Clock* no momento de escrita de cartões.

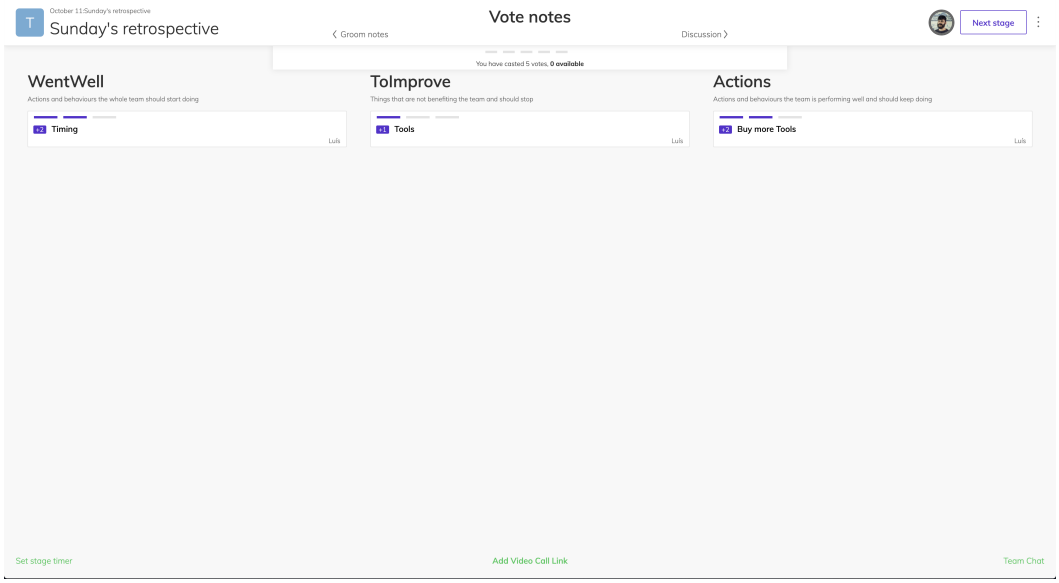


Figura 2.19: Ecrã do *Team O'Clock* no momento de votação.

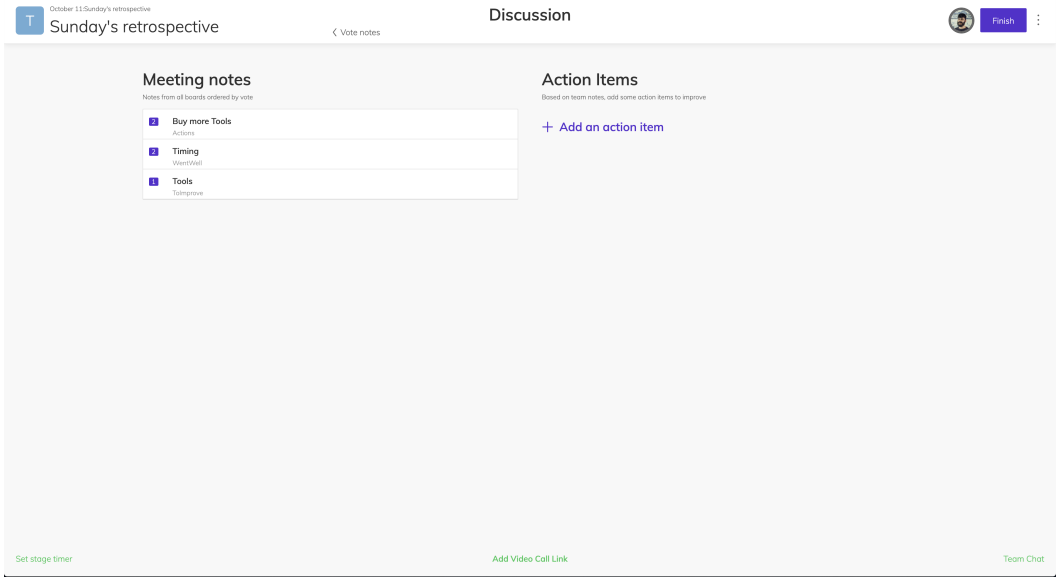


Figura 2.20: Ecrã do *Team O'Clock* no momento de escrita de ações.

As figuras anteriores foram retiradas do site oficial da *Team O'Clock*.

A *Team O'Clock* apresenta alguns problemas a nível de experiência com utilizador, não é uma tarefa simples a iniciação de uma sessão de *Retrospective*. Um ponto que também cria alguns problemas é o excesso de funcionalidades que desvia a atenção dos utilizadores do foco principal que é a sessão em que estão a participar. Tem ainda alguns problemas de interface com o utilizador que compromete a experiência de utilização.

Por fim apresenta algumas decisões de privacidade que se deve questionar pois qualquer elemento que participou numa sessão pode editar o bloco de ações mesmo após a sessão ter sido terminada, podendo até ser considerado uma falha de segurança.

Reetro

A *Reetro* é uma aplicação gratuita e que tem por objetivo conseguir chegar a todas as pessoas independente do seu nível de conhecimento (*About Reetro | Fun, Easy & 100% Free* s.d.).

Como dito anteriormente esta aplicação é gratuita e não oferece qualquer plano pago. A aplicação tem uma interface simples e faz uso de iconografia para simplificar algumas tarefas. A utilização destes mesmos ícones apesar de ser um facilitador em determinadas funcionalidades noutras é um elemento proporcionador de erro. A nível de experiência de utilização também tem algumas falhas sendo que o processo de iniciação de uma sessão não é de todo claro e o fluxo de uma sessão de *Retrospective* é confuso causando algumas dúvidas na sua utilização.

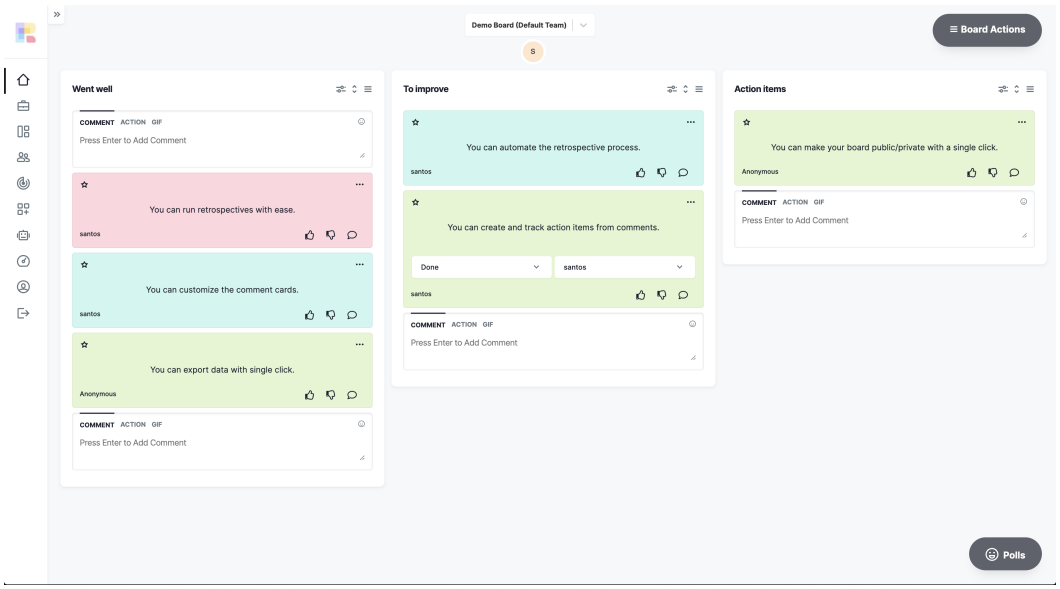


Figura 2.21: Exemplo de uma *Retrospective* no *Reetro* (Retirado do site oficial do *Reetro*).

Retrium

A *Retrium* foi criada em 2015 com o foco de satisfazer todas as necessidades da melhor forma possível de uma sessão de *Retrospective* (*Meet the Retrium Team: Retrium is a Retro Software Company | Retrium* s.d.).

Os planos que a aplicação oferece dividem-se em um grátis considerado como um teste por um período de 30 dias e após isso existem dois planos pagos, um para equipas pequenas e um plano para empresa que os preços são sob consulta.

A aplicação oferece um conjunto de *templates* que têm por objetivo facilitar o processo de *Retrospective*.

A interface da aplicação não oferece uma experiência de utilização boa e a experiência com o utilizador tem algumas falhas no fluxo de utilização. A aplicação tem também alguns problemas de performance onde por vezes não carrega todo o conteúdo ao mesmo tempo.

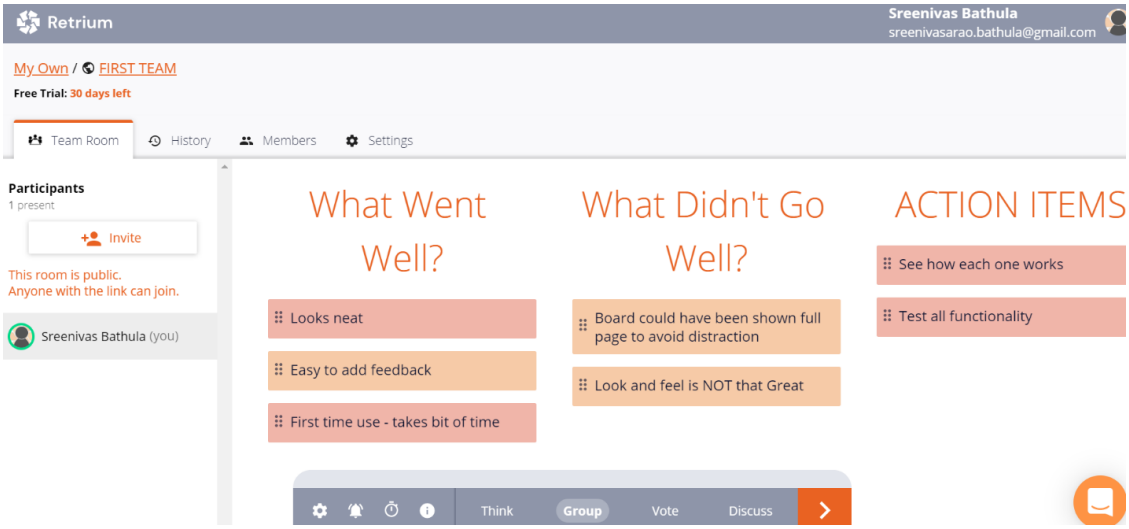


Figura 2.22: Exemplo de uma *Retrospective* no *Retrium* (Retirado do site https://agilefurther.com/retrium-online-retrospective-app-pricing-how-to-use-guide-review/ pois o site oficial encontrava-se em baixo).

FunRetro

A *FunRetro* foi criada em Glauber Ramos em 2015 e nasceu devido à necessidade do seu criador (*FunRetro | About us and how we started.* S.d.).

A aplicação apesar de ter uma conta gratuita a mesma tem uma serie de limitações, nomeadamente relacionadas com a capacidade de histórico e com os limites de sessões que se pode criar.

Os planos pagos que a mesma oferece têm limitações nos números de sessões que se pode criar, que para uma empresa com as dimensões significativas no número de colaboradores, ou seja apesar de oferecer planos pagos os mesmos têm limitações.

A *FunRetro* é uma aplicação relativamente simples de utilizar, tem por base caso o utilizador deseje uma base de dados de sessões já pré organizadas que o utilizador pode escolher. As secções têm cores que identificam qual o passo em que a equipa se encontra de forma a facilitar a associação visual.

A aplicação também apresenta várias falhas no que toca ao uso prático da mesma. Problemas sincronismo, funcionalidades mal desenvolvidas, problemas com persistência de dados enquanto se está a fazer uma sessão ou até mesmo funcionalidades essenciais que não existem e que as equipas sentem necessidade.

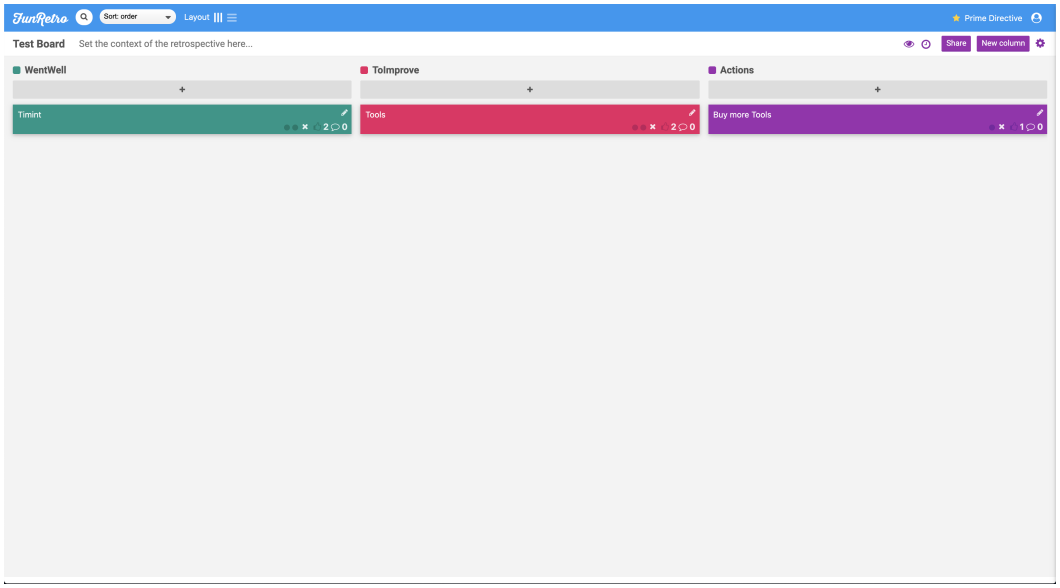


Figura 2.23: Exemplo de uma *Retrospective* no *FunRetro* (Retirado do site oficial do *FunRetro*).

2.4 Sumário

Com este capítulo é possível perceber que a *Sprint Retrospective Meeting* é uma cerimónia que tem um elevado foco nas pessoas e na sua melhoria contínua. Esta melhoria tem como principal consequência o aumento da qualidade dos produtos desenvolvidos pelas equipas de trabalho que realizam regularmente sessões de *Retrospective*.

Capítulo 3

Análise de valor

Neste capítulo será analisado a proposta de valor do projeto. Com a ajuda do modelo *New Concept Development* iremos analisar quais a necessidades, bem como as ideias que surgem até às ideias finais.

Será também efetuada uma análise de mercado de forma a perceber quais os impactos que a solução poderá ter numa equipa de trabalho.

Por fim será apresentada a solução do ponto de vista de negócio e para isso irá ser utilizado o modelo *Canvas* para enquadrarmos o projeto a desenvolver.

3.1 Modelo New Concept Development

New Concept Development (NCD) é um modelo de desenvolvimento relacional e não um processo linear.

NCD é um modelo que disponibiliza uma linguagem simples e uma visão holística. O modelo divide-se então em três grandes blocos, o centro também chamado como o motor, que é o que conta com a visão, estratégia e cultura, as cinco fatias que definem as atividades chave, e o anel exterior que representa os fatores ambientais externos (Koen s.d.).

É um modelo iterativo que tem como grande objetivo a facilitação e o esclarecimento.

A Figura 3.1 mostra o *NCD* e as suas áreas. Na imagem podemos também ver os cinco elementos de atividade do mesmo. A sua forma é circular e tem como propósito dar a entender que as ideias devem seguir um caminho fluído e interativo entre os elementos. As setas que apontam para o interior representam pontos iniciais e as que aponta para o exterior são os resultados que saem do modelo e entram no processo de desenvolvimento do produto.

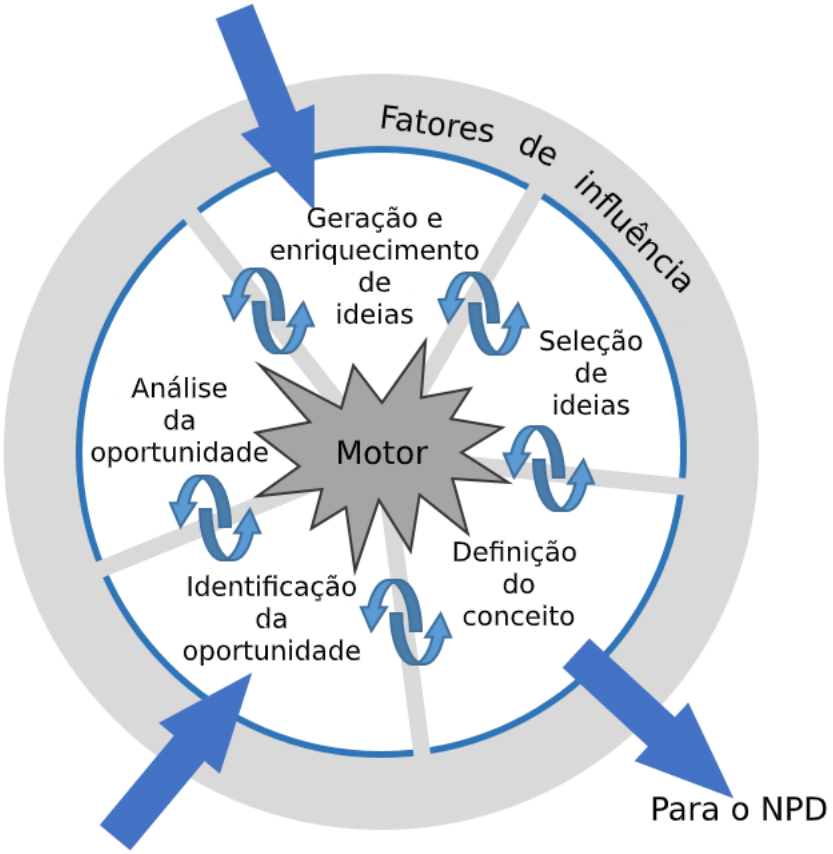


Figura 3.1: Ciclo do modelo *New Concept Development* (Koen s.d.).

3.1.1 Identificação da oportunidade

Segundo a classificação do *top* 100 das empresas que mais cresceram em 2019 pela *Fortune* temos sete empresas ligadas ao sector tecnológico na lista das dez primeiras (Fortune 2019).

A *CompTIA* pelas previsões que analisou estima que o mercado global da industria tecnológica chegue ao valor de 5.2 triliões de dólares, crescendo a uma taxa de 3,7% onde mais de 50% do mercado é dominado pelo sector do *software* (CompTIA 2019).

Como mencionado anteriormente, segundo o estudo «13th annual State of Agile» (Version One 2019) da *CollabNet VersionOne* conclui que 97% das organizações entrevistadas (1,319 respostas) utilizam metodologias *Agile*.

Com o aumento da utilização das emtodologias *Agile* verifica se o aumento das práticas de cerimónias como a *Sprint Retrospective Meeting* pois fazem parte das cerimónias de metodologias *Agile*, nomeadamente, *Scrum*.

Assim sendo, como referido pelo estudo *Software Metrics Classification for Agile Scrum Process* as sessões de *Retrospective* têm elevada influência no sucesso do projeto (Kurnia, Ferdiana e Wibirama 2018).

Com estes dados conseguirmos perceber que existe uma oportunidade de negócio referente às soluções para as elaborações de *Sprint Retrospective Meeting*.

3.1.2 Análise da oportunidade

Nos dias de hoje já existem diversas soluções no mercado com vista a facilitarem as sessões de *Sprint Retrospective Meeting* que poderiam responder às necessidades da *Mindera*.

Contudo, foram verificadas que estas mesmas soluções não conseguem responder a todas as necessidades das equipas bem como diversas delas apontam mesmo problemas estruturais.

As soluções existentes apresentam limitações técnicas, falhas de segurança ou até mesmo funcionalidades que não existem.

Surge a necessidade da criação de uma solução que responda diretamente às necessidades das equipas de trabalho com vista a promover a execução regular de sessões de *Retrospective* para que as mesmas consigam usufruir dos benefícios comprovados destas sessões.

3.1.3 Geração de ideias

Com as necessidades da *Mindera* surgem então algumas ideias que poderiam ser a solução para as mesmas.

As ideias são:

- Ideia 1: Tentar complementar as soluções que as equipas usam com outras ferramentas;
- Ideia 2: Desenvolver uma aplicação nativa que responde-se as necessidades das equipas;
- Ideia 3: Desenvolver uma *Web App* responsiva que responda às necessidades das equipas.

As ideias 2 e 3 sugerem então o desenvolvimento interno de uma solução protótipo que consiga responder a todas as necessidades das equipas de estudo, com a ambição de ampliar a todo o corpo da empresa, a grande diferença entre a ideia 2 e 3 é a metodologia de desenvolvimento em que uma apoia a criação de aplicações nativas para cada sistema operativo e a outra apoia a criação de uma *Web App* responsiva sendo assim multi-plataforma.

3.1.4 Seleção de ideias

Nesta fase todas as ideias serão estudadas ao mais pequeno pormenor para se analisar quais a vantagens e desvantagens a curto, médio e longo prazo, bem como perceber qual a que responde melhor as necessidades das equipas.

A ideia selecionada foi a de desenvolver um protótipo de uma *Web App* responsiva.

A solução desenvolvida terá por base as necessidades e requisitos que as equipas de desenvolvimento tenham.

Análise hierárquica

Métodos de decisão multicritério é uma estrutura para suporte à tomada de decisões complexas com múltiplos critérios que por vezes estão em conflito, e que, podem ter visões e pesos diferentes conforme quem decide (Saarikoski e Barton 2016).

O método de decisão escolhido para se analisar qual a melhor ideia para responder ao problema foi o método de análise hierárquica (*Analytic Hierarchy Process (AHP)*), criado

pelo professor Thoma L. Saaty em 1980. O *AHP* permite o uso de critérios qualitativos e quantitativos. A linha de orientação deste método é a divisão do problema em níveis hierárquicos para facilitar a sua compreensão e avaliação.

Com a necessidade de criar uma solução que responda às necessidades das equipas de trabalho na elaboração de *Sprint Retrospective Meetings* temos a seguinte análise multicritério.

De acordo com as ideias geradas foi criada a árvore hierárquica de decisão.

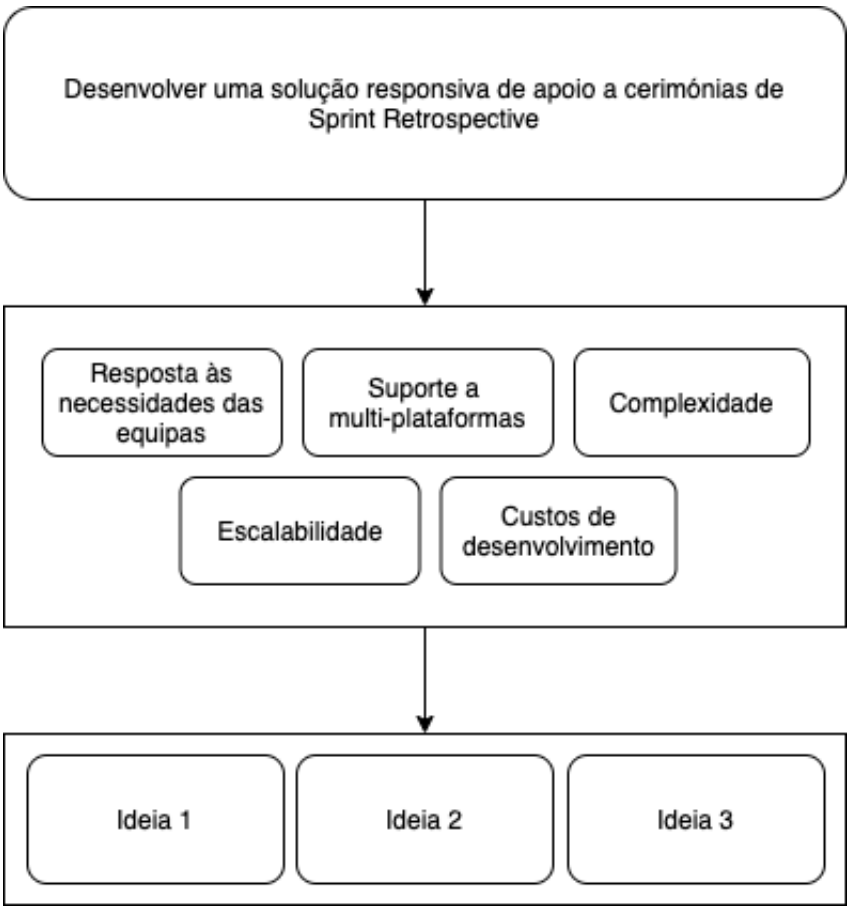


Figura 3.2: Árvore hierárquica de decisão

Como pode ser visto na Figura 3.2 a ideia a desenvolver tem que ser assente em cinco critérios de decisão:

- Resposta às necessidades das equipas;
- Suporte a multi-plataformas, a solução tem que ser responsiva;
- Complexidade da solução, se a solução é complexa ou simples no que toca ao desenvolvimento;
- Escalabilidade, pois poderá ser expandida a toda a empresa;
- Custos de desenvolvimento na relação tempo/recursos necessários.

De seguida encontra-se a tabela de avaliação *AHP*.

Tabela 3.1: Tabela de avaliação *AHP*

Crtiérios	**Resposta às necessidades das equipas**	**Suporte a multi-plataformas**	**Com-plexi-dade**	**Esca-labili-dade**	**Custos de desenvolvi-mento**
Resposta às necessidades das equipas	1	8	7	3	8
Suporte a multi-plataformas	1/8	1	7	6	8
Complexidade	1/7	1/7	1	2	5
Escalabilidade	1/3	1/6	1/2	1	5
Custos de desenvolvimento	1/8	1/8	1/5	1/5	1

A resposta as necessidades das equipas e o suporte a multi-plataformas são os dois critérios de maior relevância, contudo todos os outros têm também certo peso na decisão.

Tabela 3.2: Matriz normalizada do método *AHP*

Crtiérios	**Resposta às necessidades das equipas**	**Suporte a multi-plataformas**	**Com-plexi-dade**	**Esca-labili-dade**	**Custos de desenvolvi-mento**
Resposta às necessidades das equipas	0.5793	0.0724	0.0828	0.1931	0.0724
Suporte a multi-plataformas	0.8479	0.1060	0.0152	0.0177	0.0132
Complexidade	0.4459	0.4459	0.0637	0.0318	0.0127
Escalabilidade	0.2459	0.4918	0.1639	0.0820	0.0164
Custos de desenvolvimento	0.2963	0.2963	0.1852	0.1852	0.0370
Soma	1.0000	1.0000	1.0000	1.0000	1.0000

Tabela 3.3: Pesos dos critérios de avaliação

Crtiérios	**Pesos**
Resposta às necessidades das equipas	0.5720
Suporte a multi-plataformas	0.2480
Complexidade	0.0770
Escalabilidade	0.0770
Custos de desenvolvimento	0.0260

Com base na análise das tabelas 3.1, 3.2 e 3.3, e com base nos critérios definidos a ideia escolhida é a ideia 3.

3.1.5 Desenvolvimento do conceito

O projeto desenvolvido terá em especial cuidado não só os requisitos e necessidades de uma equipa de trabalho mas também todos os cuidados que uma cerimónia de *Retrospective* tem por natureza.

O protótipo terá em especial cuidado nas necessidades, requisitos e funcionalidades sugeridas pelas equipas de destino, pois a solução visa responder problemas que são comuns nas equipas e relatados pelas mesmas.

Apesar de o protótipo ser desenvolvido a pensar nas equipas com que o projeto trabalha de perto deverá também existir cuidados relativos a escalabilidade pois caso o protótipo consiga alcançar resultados positivos poderá ser aplicado a mais equipas na empresa.

O grande objetivo do projeto é conseguir incentivar as equipas a efetuarem sessões de *Retrospective* regulares com a finalidade de se conseguir uma melhoria constante nas equipas de trabalho.

3.1.6 Valor

O valor de um produto pode ser um elemento diferenciador, preço, mercado alvo ou até inovação, em suma, o valor de um produto pode ser praticamente tudo, desde que faça sentido para o modelo de negócio.

O valor também tem dois lados, por um lado temos o valor que um produto poderá ter para um consumidor, por outro lado temos o valor que esse mesmo produto tem para o lado da empresa que o disponibiliza no mercado.

De seguida, iremos analisar o valor que o projeto poderá ter.

Valor

A solução proposta a desenvolver está claramente relacionada com o facto das próprias cerimónias de *Retrospective* serem motivadoras de sucesso e melhoria continua das equipas.

Com estes valores conseguimos uma melhoria individual nos elementos das equipas, uma consequente satisfação pessoal que leva à felicidade de cada individuo que se reflete na sua produtividade.

Juntamente com o sucesso do individuo está o sucesso da organização pois está dotada de profissionais capazes e com qualidade para desenvolver produtos de qualidade para os seus clientes.

Valor percecionado

O valor percecionado que podemos ter numa solução que promove as sessões de *Retrospective* passa pelo que vários autores afirmam, em que existe uma correlação entre a quantidade de sessões efetuadas, a qualidade de cada sessão e o respetivo aumento de qualidade e performance das equipas de trabalho.

Este aumento de qualidade por parte das equipas vai-se refletir no trabalho das mesmas conseguindo fazer com que os clientes estejam mais satisfeitos com os produtos criados.

Valor para os Clientes

Como grande ponto de valor para os clientes, neste caso para os elementos das equipas a que a solução se destina, temos a ligação direta entre o desenvolvimento pessoal e individual que cada um pode ter com a criação de ações que resultam de uma sessão de *Retrospective*.

Da mesma forma que existe uma relação entre o desenvolvimento pessoal e uma sessão de *Retrospective* também existe uma relação global em que com as sessões promove-se a melhoria individual de cada elemento, consequentemente a qualidade da equipa eleva-se e por resultado temos equipas com mais qualidade que estão mais bem preparadas para desenvolver produtos onde os mesmo também serão de mais qualidade.

Levando ao outro ponto do cliente, que pode ser visto como o cliente final dos produtos que os elementos das equipas desenvolvem verifica se que são clientes mais satisfeitos. Isto porque se os elementos estão dotados e capacitados para desenvolver produtos com mais qualidade os clientes desses produtos poderão usufruir de resultados com maos qualidade consequentemente são clientes mais satisfeitos.

Benefícios e Sacrifícios

Do ponto de vista de benefícios do projeto podemos-nos focar em influência que as sessões de *Retrospective* conseguem ter na melhoria contínua do individuo. Esta melhoria continua se analisada mais profundamente não passa só pelo aumento da qualidade do profissional como também contribui para a sua felicidade pessoal o que está de certa forma ligada à produtividade do individuo.

Para o lado da empresa temos como grande benefício um corpo de profissionais competentes, satisfeitos, com qualidade e em constante evolução. A empresa consegue então estar dotado de profissionais com qualidade que se irá refletir positivamente nos produtos desenvolvidos e com isso possíveis benefícios para o cliente inclusive.

Para o consumidor do produto desenvolvido é possível distribuir produtos com mais qualidade fazendo com que estes se sintam satisfeitos com os produtos que compram ou usufruem.

Como sacrifício temos o fator de desenvolvimento da solução que requer tempo e recursos tecnológicos e humanos para o desenvolvimento da plataforma.

Juntamente com o desenvolvimento também requer tempo para juntamente com as equipas de desenvolvimento conseguir-se analisar quais são realmente as suas necessidades.

3.1.7 Proposta de Valor

A proposta de valor pode ser a chave de sucesso para qualquer produto. A sua construção deve ter apoio em três alicerces, o primeiro é a própria oferta e os seus atributos, a segunda é o consumidor os benefícios para o mesmo e o terceiro é o ponto diferenciador que o nosso produto oferece face aos concorrentes (*Desenvolva uma proposta de valor - O blog de Marketing e Vendas* 2018).

Como proposta de valor temos então:

Oferecer uma solução exclusiva que responda às necessidades diretas das equipas de desenvolvimento.

3.1.8 Modelo Canvas

O modelo Canvas é na sua essência uma ferramenta de ajuda ao planeamento estratégico que tem como finalidade desenvolver um modelo de negócio e uma visão do mesmo.

Na Figura 3.3 podemos ver o modelo *Canvas* desenvolvido para o projeto.

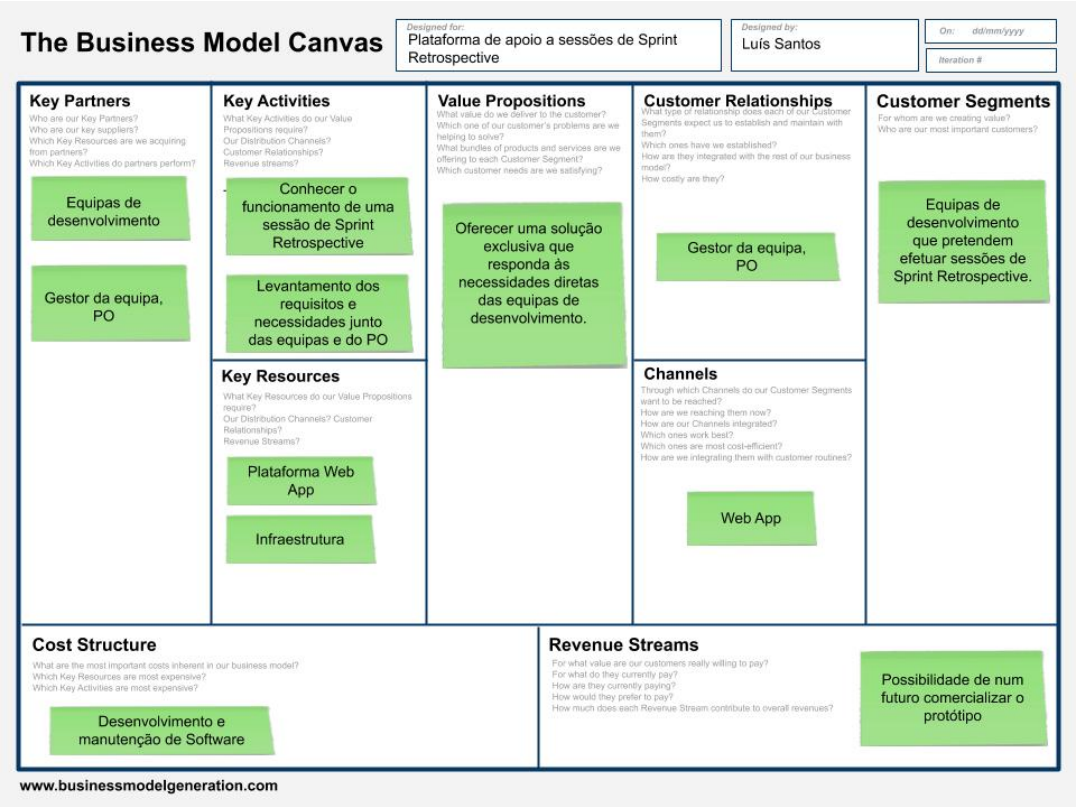


Figura 3.3: Modelo *Canvas* do projeto

Com o modelo *Canvas* do projeto conseguimos ter uma visão generalista do modelo de negócio conseguindo assim pontos de foco e referência de caminhos a seguir para o sucesso do projeto.

3.2 Sumário

Em suma neste capítulo é possível perceber que existe uma oportunidade de negócio. Apesar de existirem soluções no mercado não é sinónimo de que as mesmas conseguem responder às necessidades das equipas de trabalho da *Mindera*. Conseguimos também perceber qual o valor real que a solução desenvolvida terá para as equipas e e também o impacto que terá na empresa.

Assim sendo, como a *Mindera* afirma, que usa tecnologia para desenvolver produtos que se orgulha com pessoas que gosta, esta solução visa, como a proposta de valor afirma, satisfazer as necessidades das pessoas para que estas se tornem melhores profissionais e consecutivamente mais felizes.

Capítulo 4

Análise e Design

Este capítulo tem por função demonstrar as decisões arquiteturais para a solução bem como um levantamento de requisitos funcionais e não funcionais.

Foi efetuado o levantamento de requisitos com ajuda do modelo *FURPS+* (Funcionalidade, Usabilidade, Reliabilidade, Performance, Suportabilidade, Mais Outros Requisitos Não-Funcionais (FURPS+)) e foi também efetuada uma análise de padrões arquiteturais que se mais se enquadravam no projeto, tais como, *Model-view-controller*, *Layered* e *Microservices*, que visam facilitar to o processo de desenvolvimento

4.1 Análise

O levantamento de requisitos funcionais e não funcionais inicializaram-se por perceber quais os requisitos básicos exigidos pelo *Product Owner (PO)* e só posteriormente é que se deu início à segunda fase que pretendia uma análise através da observação. Esta segunda fase passou pela participação como observador em várias cerimónias de *Sprint Retrospective* que as equipas de trabalho fazem.

A observação junto das equipas teve uma duração de cerca de duas semanas, onde foi possível compreender quais a reais necessidades das equipas bem como quais os problemas que as equipas enfrentavam a quando a elaboração de uma *Retrospective*. Esta observação dos problemas foi um ponto importante para a definição dos requisitos, bem como, um elemento fulcral para se compreender de que forma a solução desenvolvida poderia responder a um dos objetivos do projeto, que passa por incentivar às equipas a tornar as cerimónias de *Retrospective* mais regulares nos projetos que estão envolvidos.

A participação como observador nos *Sprint Retrospective* foi muito importante, pois através da observação conseguiu se perceber como é que as equipas se organizavam e como é que era conduzida a cerimónia. Foi também tido em conta se existia alguma disparidade na forma como toda a sessão se desenrolava entre equipas diferentes, onde se percebeu, que as cerimónias decorriam de forma muito semelhante para todas as equipas observadas.

Alguns dos problemas encontrados neste período de observação foram os seguintes, dificuldades de utilização nas aplicações existentes, funcionalidades com limitações ou mesmo inexistentes, interfaces pouco intuitivas ou até problemas que levavam à perda de trabalho já realizado.

Assim sendo e tendo por base todos os problemas, as necessidades das equipas e os requisitos impostos pelo *PO* chegou se à lista de requisitos funcionais e não funcionais.

4.1.1 Requisitos funcionais

Os requisitos funcionais são todos aqueles que estão diretamente ligados às funcionalidades da própria solução. O diagrama de casos de uso apresentado na Figura 4.1 é uma demonstração gráfica dos requisitos funcionais.

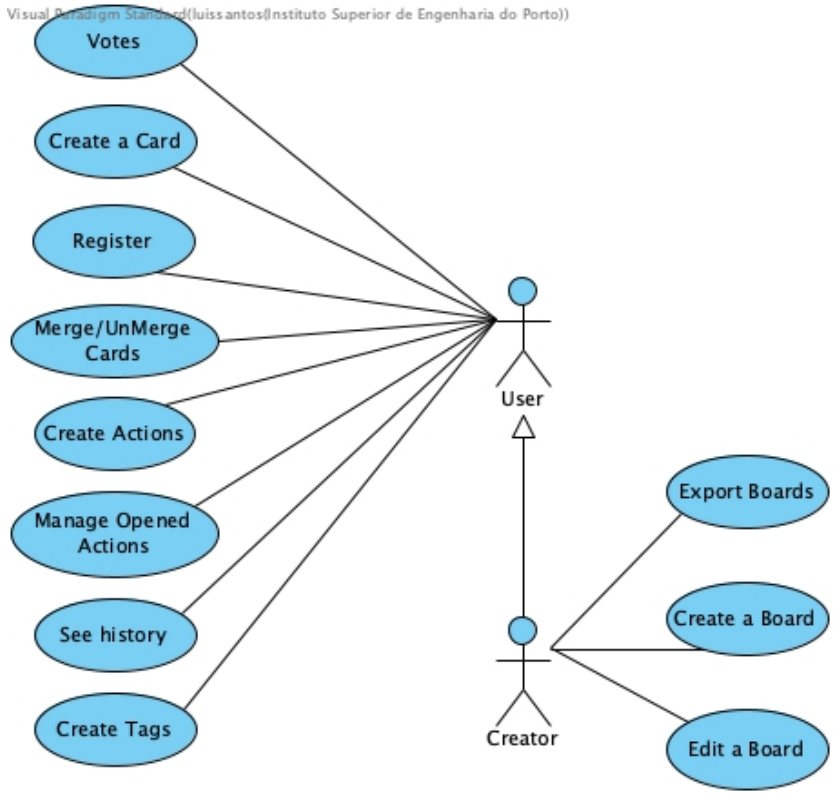


Figura 4.1: Diagrama de casos de uso.

Como já referido anteriormente a Figura 4.1 ilustra um diagrama de casos de uso, onde se verifica a existência de dois atores o *User* e o *Creator* que surge através do padrão *Generalization of an Actor*, ou seja, o *Creator* é um *User*, usufruindo de todas as capacidades do mesmo, contudo tem mais funcionalidades especificas que só ele tem acesso. O *Creator* é o elemento da equipa que por norma cria a sessão para a *Retrospective*, de notar que este elemento pode ou não ser o *PO* e posteriormente existe o outro ator que neste projeto é um grupo, pois é referente aos elementos das equipas que participam numa cerimónia de *Retrospective*.

O ator *Creator* tem como grande função criar a *Board* para se elaborar uma *Retrospective* e desta forma é o único a ter autorização para fazer a edição da mesma, por exemplo nome da sessão ou a equipa interveniente, e tem ainda uma funcionalidade extra disponível que é a exportação de uma *Board* no final de uma *Retrospective*, esta funcionalidade nasce das necessidades que o *PO* tem em partilhar os resultados de uma cerimónia com pessoas externas à equipa, nas restantes funcionalidades possui o mesmo grau de permissão como os restantes atores.

Em relação ao outro ator, o *User*, este possui todas as restantes funcionalidades para poder participar numa sessão de *Retrospective*, algumas dessas funcionalidades vão ser explicadas de seguida. O *Create a Card* é a representação da possibilidade que cada utilizador tem

de criar os cartões de comentários em relação ao *Sprint*, o *Votes* é a ação que cada utilizador dispõe para num determinado ponto da cerimónia votar em quais cartões acha mais importantes de todos os utilizadores, para que numa fase final sejam tomadas ações para os cartões com mais votos, levando à seguinte funcionalidade. *Create Actions* é a criação de cartões que representam as ações que se poderão tomar para potencializar o que correu bem no *Sprint* ou as ações a tomar para se tentar melhorar e corrigir o que correu mal, estas ações por norma tem um utilizador responsável pela execução da mesma, daí existir o *Manage Opened Actions* que é onde cada utilizador pode consultar que ações tem para elaborar bem como efetuar a gestão delas. Como funcionalidades extra que visam a responder as necessidades que as equipas demonstraram foi criada a funcionalidade *Merge/UnMerge Cards* que é uma funcionalidade utilizada para agregar ou desagregar cartões de uma mesma board que retratem o mesmo assunto, outra funcionalidade é o *Create Tags* que permite ao utilizador a criação de etiquetas que podem ser associadas aos cartões que relatem o mesmo tema, mas que não fazem sentido ser agregados, para desta forma o utilizador ao olhar para a sessão ter imediatamente uma referência gráfica que o ajuda a perceber o que ali se retrata.

Com os casos de uso acima referidos podemos chegar a um fluxo de utilização da aplicação. A Figura 4.2 demonstra qual o fluxo de uso que o protótipo a ser desenvolvido deverá ter.

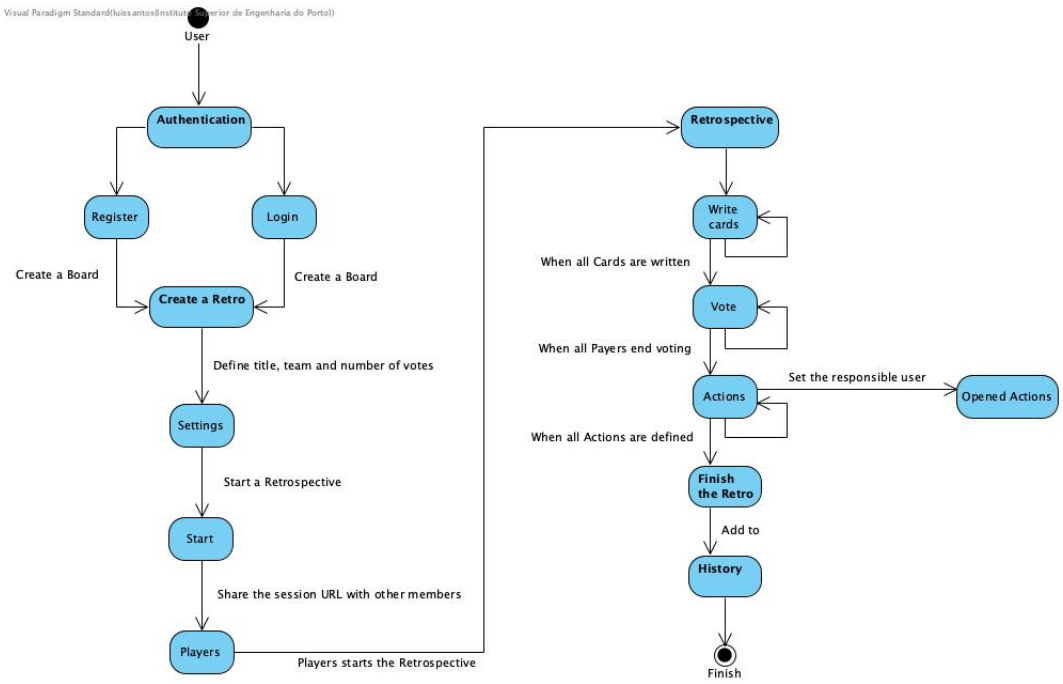


Figura 4.2: Diagrama de fluxo da utilização da aplicação.

Na Figura 4.2 temos uma breve descrição gráfica da forma como uma cerimónia de *Retrospective* é conduzida com a solução desenvolvida. O utilizador efetua a sua autenticação em que o sistema verifica se o utilizador já existe, caso sim faz *login* caso contrário é efetuado o registo do novo utilizador. Na criação de uma sessão o utilizador inicia o processo pela definição do nome da sessão e a identificação da equipa que irá fazer parte da sessão, e dá início à sessão, após a identificação da equipa é partilhado o *link* com o resto da equipa para os mesmos acedam à sessão. Passamos para a fase de criação de cartões em que cada utilizador cria e partilha com o resto da equipa, de seguida passa-se para a votação

dos respetivos cartões. Quando todos os utilizadores votarem inicia-se a fase de criação de ações e a respetiva atribuição de responsáveis pelas mesmas. Por fim quando se finaliza a cerimónia a mesma é adicionada ao histórico de cada utilizador.

4.1.2 Requisitos não funcionais

Os requisitos não funcionais serão então levantados com a ajuda do modelo *FURPS+*.

Na Tabela 4.1 abaixo pode se verificar quais os requisitos levantados e a sua respectiva categoria.

Tabela 4.1: Tabela FURPS+

FURPS+	**Requisito**
F	O registo e *login* da aplicação é feito através de utilização de autenticação via *google*
F	O *email* de registo e *login* só pode ser o da empresa (email@mindera.com)
U	O sistema deverá ser preventivo no que toca a erros de usabilidade que o utilizador possa ter
U	O sistema deverá ter uma interface responsiva
P	O sistema deve ter performance suficiente para o sistema funcionar de forma precisa e eficaz
S	O sistema deverá garantir qualidade de *software* através de testes
S	O sistema deverá ter em conta a possibilidade de escalabilidade no futuro
+	Deverão ser utilizadas tecnologias de *WebSockets*
+	A linguagem a adotar deverá ser baseada em *JavaScript*

4.2 Arquitectura de software

Esta secção tem por ambição conseguir perceber qual a melhor abordagem arquitetural para o projeto. Será feito um estudo da arquitetura a adotar e de algumas alternativas de forma a conseguirmos uma solução eficaz, sólida, que consiga lidar com escalabilidade e que se apoie em bons padrões de desenvolvimento.

4.2.1 Padrões Arquiteturais

Durante vários anos os criadores de *software* foram incentivados a desenvolver sistemas baseados exclusivamente nos requisitos técnicos. Arquitetura surgiu como fator fundamental do *design* de *software*, esta suporta a estruturação de um sistema de um sistema de grande dimensões. A vista arquitetural de um sistema é abstrata e concentra-se na interação e comportamento dos elementos (Bass, Clements e Kazman 2003).

De forma ajudar no processo arquitetural de uma solução temos uma série de padrões que ajudam no desenvolvimento de um sistema. Estes padrões têm diferentes características e devem ser adequados da melhor forma ao projeto a desenvolver. Alguns dos padrões são, *Model-view-controller*, *Layered* (camadas), *Client-server*, *Microkernel*, *Microservices* (micro serviços), entre outros.

Microservices pattern

Microservices distribui a aplicação em pequenos componentes em que cada um tem a sua própria responsabilidade e que pode ser desenvolvido de forma independente, onde a única dependência que existe é a ligação entre eles. Como é um padrão assente em comunicações entre componentes é necessário assegurar que a mensagem é compreendida pelos dois, isto exige alguma coordenação, principalmente entre as equipas que desenvolvem (Morlion 2018).

Na Figura 4.3 podemos ver um diagrama de como funciona o padrão *Microservices*.

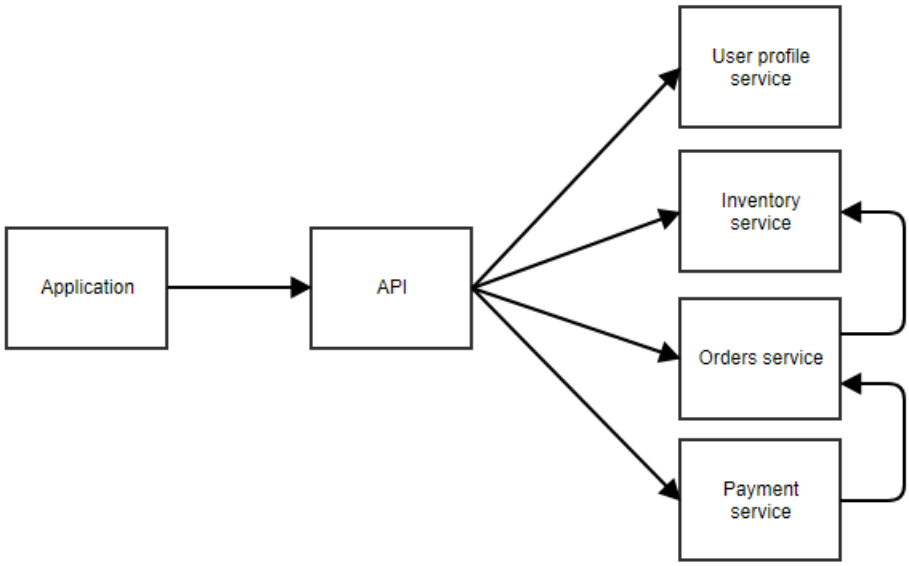


Figura 4.3: Diagrama do padrão *Microservices* (Morlion 2018).

O padrão *Microservice* tem como grandes vantagens a facilidade de manutenção pois os componentes são separados, a escalabilidade tende a ser fácil no que toca ao aumento de mais micro serviços e é fácil a reescrita de componentes porque por norma são pequenos e com baixo acoplamento.

Como desvantagem temos a dificuldade de planeamento, pois como é um desenvolvimento que envolve muitas ligações é necessário boas comunicações e coordenações entre as equipas o que requer um grande planeamento para que tudo seja claro. De notar que existem múltiplos componentes e que por isso a probabilidade de falhar é maior, pois existe mais pontos por onde podem surgir complicações.

Model-view-controller pattern

Model-view-controller também conhecido por *MVC* é um padrão que divide aplicações interativas em três partes, *Model* que contem todo o núcleo responsável pelas funcionalidades e dados, *View* que é responsável pela visualização das informações que chegam ao utilizador e por fim *Controller* que é a parte que está encarregue de gerir as ações efetuadas pelos utilizadores. Tudo isto é efetuado através da separação interna da representação da informação, da forma como a informação é apresentada ao utilizador. Isto dissocia os componentes e permite a reutilização de código de forma eficaz (Vijini Mallawaarachchi 2017).

A Figura 4.4 podemos ver um diagrama de como funciona o padrão *MVC*.

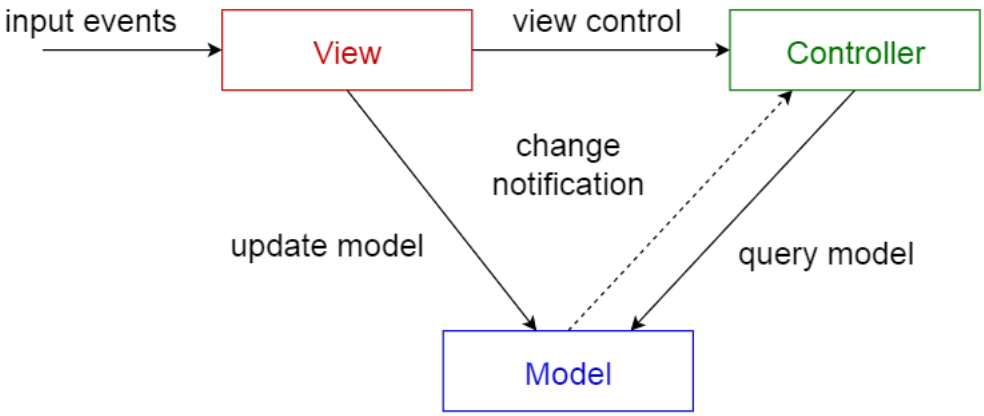


Figura 4.4: Funcionamento do padrão *MVC* (Vijini Mallawaarachchi 2017).

Layered pattern

O padrão *Layered* é provavelmente o padrão mais conhecido na arquitetura de *software*, inclusive muitos programadores provavelmente usam sem sequer saber o nome do padrão que estão a utilizar. A ideia é dividir a solução em *layers*, camadas, em que cada camada tem determinada responsabilidade e fornece um serviço a uma camada acima (Morlion 2018).

Não existe um número definido de camadas mas algumas das mais conhecidas são:

- *Presentation layer*
- *Application layer*
- *Business layer*
- *Persistence or data access layer*
- *Database layer*

A ideia é que o utilizador desencadeia um fluxo que compeça na *Presentation layer* através da execução de uma ação, esta comunica com a camada abaixo da mesma, e assim sucessivamente até à camada de armazenamento de dados.

A Figura 4.5 mostra o fluxo de utilização do padrão *Layered*.

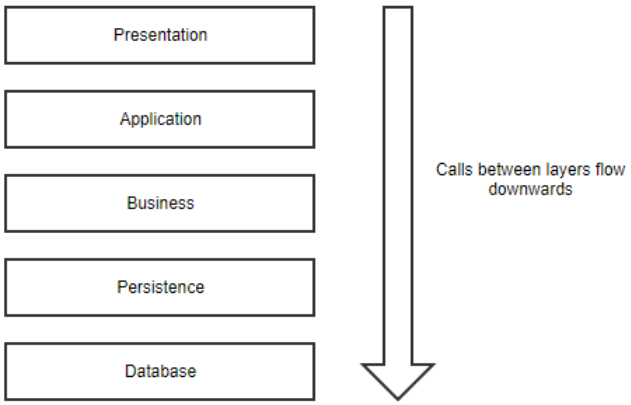


Figura 4.5: Fluxo de utilização do padrão *Layered* (Morlion 2018).

A *Presentation layer* é normalmente responsável pela apresentação gráfica da aplicação, bem como a interação com o utilizador. Por convenção esta camada não deverá conter lógica que não seja especifica da relação com a interação do utilizador

A *Business layer* é a camada que está ligada à lógica do negócio da solução.

Na *Application layer* é caracterizada pela abstração da ligação entre a camada de negócio e apresentação. Na teoria com esta camada conseguimos por exemplo alterar a tecnologia utilizada num dos lados sem que tenha influência no outro. Por outro lado esta camada tem também por função depositar lógica que não se encaixe nem na camada de apresentação nem de negócio.

Por fim temos a *Persistence layer* que contem a lógica responsável pela ligação com a base de dados. É nesta camada que está a manipulação da base de dados, como por exemplo detalhes de ligações.

Como vantagens o padrão *Layered* é conhecido por grande parte dos programadores, é fácil de desenvolver, organizar e testar uma aplicação. Contudo pode ter tendência a tornar a aplicação num monolítico, pois os programadores por questões de conveniência acabam muitas vezes a misturar as camadas.

4.2.2 Seleção

Após a análise dos mais diversos padrões chegou-se à conclusão que o padrão mais adequado à solução a desenvolver assenta em *Layered*. Podemos ver de seguida alguns diagramas que têm por função mostrar um previsão da solução a desenvolver de forma a agilizar todo o processo de desenvolvimento.

A Figura 4.6 mostra o diagrama de componentes da solução.

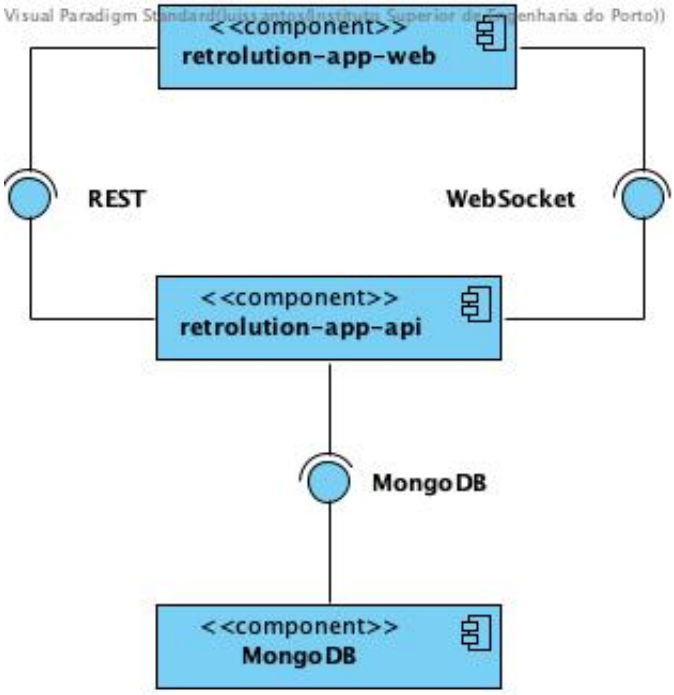


Figura 4.6: Diagrama de Componentes.

Verifica se que a aplicação tem por base três camadas. A primeira, é a camada responsável pela apresentação da solução ao utilizador, a camada intermédia é responsável pela lógica e pelas comunicações com a terceira camada que é a camada responsável pela base de dados. Entre a camada um e dois podemos ver que existem duas interfaces de comunicação, pois uma é responsável pela comunicação em tempo real (*WebSocket*) que a solução exige para que todos os utilizadores vejam em tempo real o que os outros estão a fazer, e depois temos uma outra interface usada para funcionalidades que não exigem ações em tempo real (*REST*) como por exemplo consulta de histórico ou registo dos utilizadores.

Na camada da base dados foi selecionada uma base de dados não relacional por ter uma utilização e implementação simples e eficaz para a solução.

Na Figura 4.7 temos um esboço do modelo de domínio da solução. A sua finalidade é ajudar na visualização da complexidade do sistema e mostrar uma vista generalista de comportamentos que o sistema deverá ter.

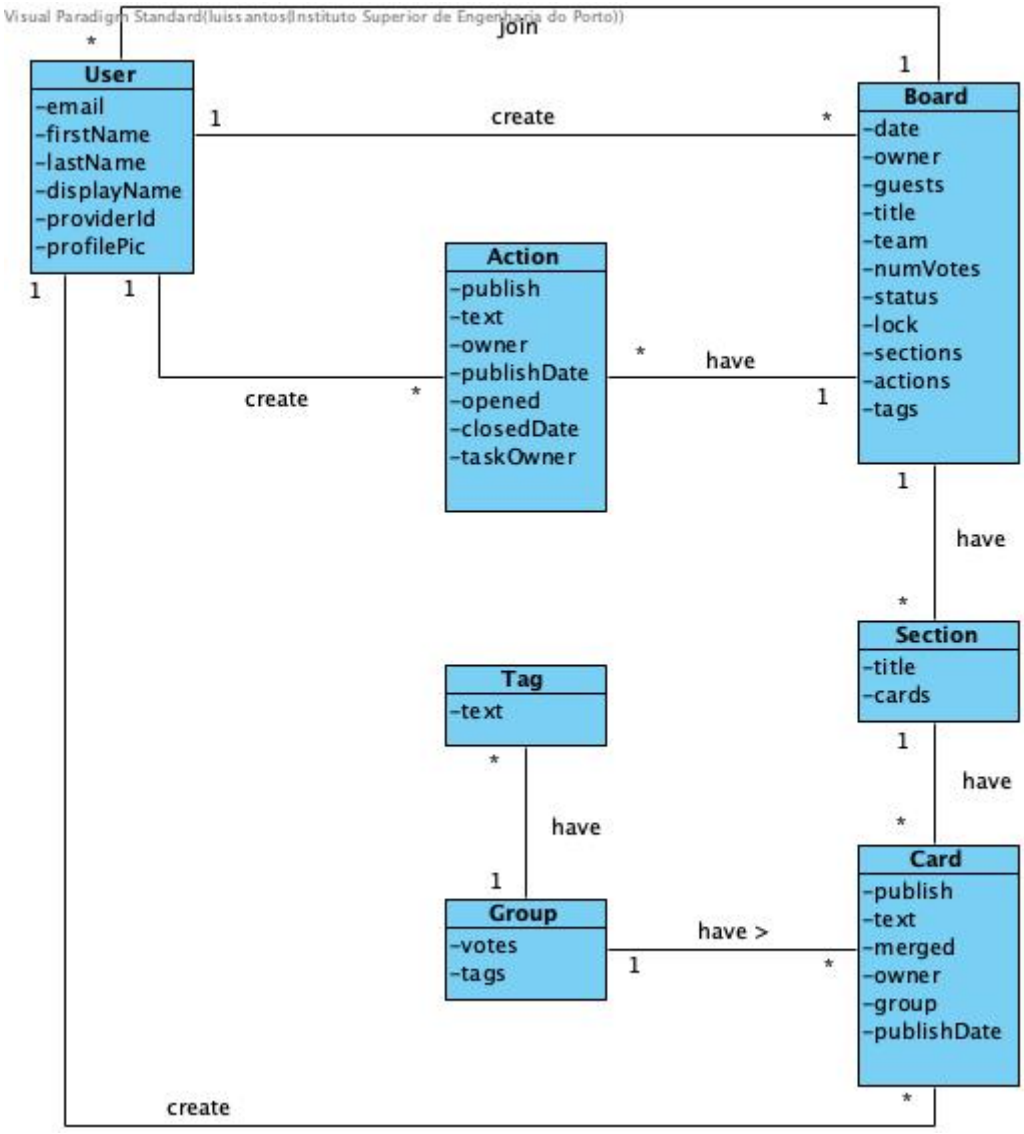


Figura 4.7: Modelo de domínio.

Segundo a Figura 4.7 temos então 7 elementos fundamentais para o funcionamento de todo o sistema. A classe *User* é a responsável pela gestão do dados do utilizador e este visa armazenar na base de dados o mesmo. De seguida temos a classe *Board* que pode de certa forma ser considerada o cérebro de todo este sistema, a mesma é a responsável pela reunião dos principais elementos contudo os mesmo não estão conectados à mesma. Ligado à *Board* existe a classe *Section* que é um elemento responsável pela ligação entre o componente *Card* à classe *Board*. A classe *Group* é também da mesma forma a responsável pela ligação da classe *Tag* à classe *Card*. Por fim temos ainda a classe *Action* que apesar de ser semelhante à classe *Card* contém uma responsabilidade diferente e desta forma torna-se numa classe independente.

Arquitetura do Servidor

No servidor após análise de varias opções chegou-se à conclusão que o melhor modelo arquitetural seria o *MVC*. Na Figura 4.8 temos então uma representação gráfica de como se irá representar o servidor.

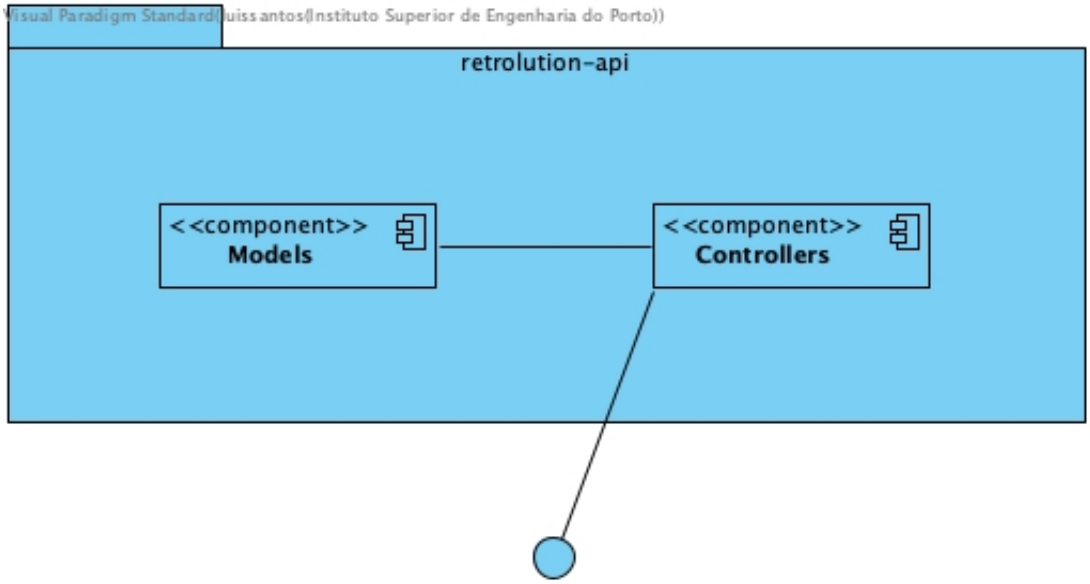


Figura 4.8: Diagrama arquitetural do servidor.

No diagrama arquitetural verifica se dois componentes, o componente *Models* que corresponde ao *Model* do padrão *MVC* onde estão reunidas todas as capacidades responsáveis pela gestão de funcionalidades e dados e o componente *Controllers* tem centralizado todas as ações responsáveis por controlar as ações pedidas à aplicação. Em relação ao *View* existe uma interface que o servidor disponibiliza que será consumida pela aplicação cliente que virá a ser a responsável pela apresentação dos dados ao utilizador.

Como uma possível alternativa poderíamos então ter uma aplicação assente no padrão *Gateway* que é um padrão baseado em *Microservices*. O diagrama representado na Figura 4.9 mostra então de como poderia ser esta opção onde temos uma camada superior, a *API Gateway* que é responsável pela criação de um único ponto de entrada nas restantes aplicações de forma a existir um controlo maior para se conseguir assegurar a segurança de utilização da mesma. Após esta camada responsável pelo aumento da segurança de todo

o sistema temos então as aplicações mais pequenas, em que cada uma é independente de todas as outras e que só tem responsabilidade por si mesma. Esta opção seria um solução ideal para um sistema de escala maior em que existe pouca dependência entre componentes, isto porque assim torna-se possível elevar a segurança e a fiabilidade de todo o sistema pois caso um serviço falhe a aplicação como um todo não falha ficando unicamente o serviço em falha indisponível. No entanto um sistema destes também obriga um grau de manutenção maior, de forma a garantir que cada elemento está em perfeito estado de funcionamento. Assim sendo para a solução a desenvolver este padrão arquitetural não é o mais indicado.

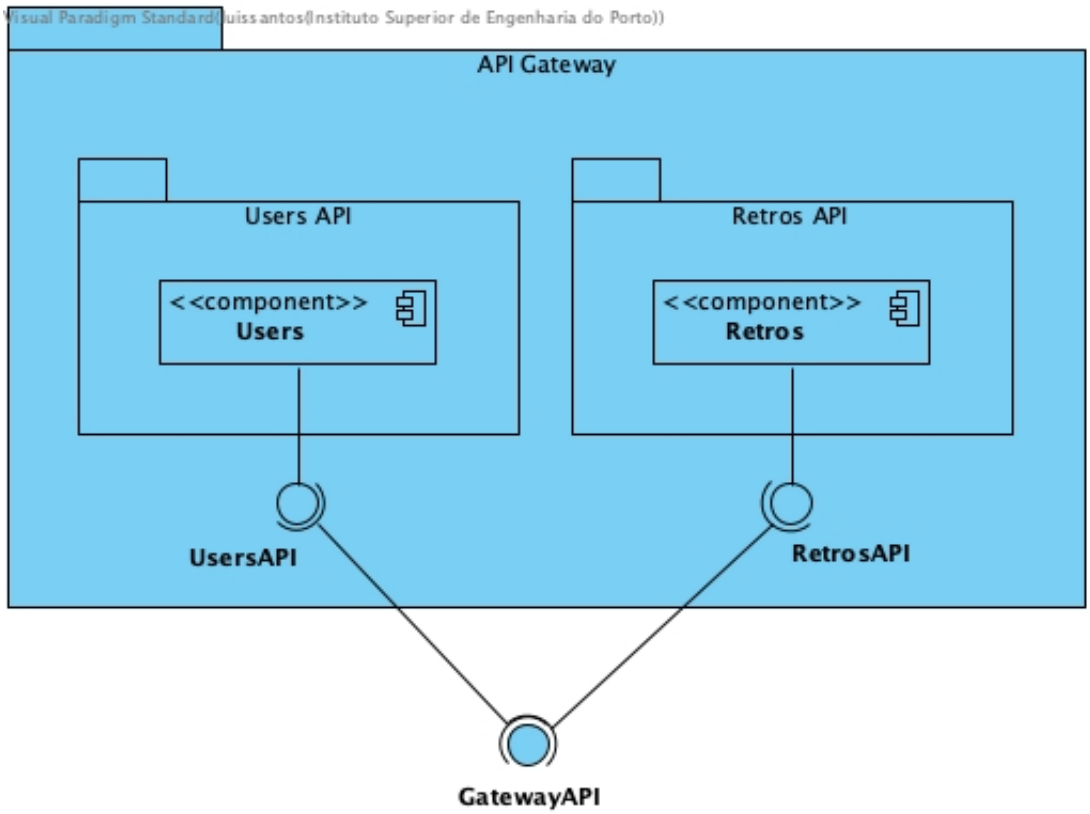


Figura 4.9: Diagrama arquitetural de uma alternativa para o servidor.

Arquitetura da Aplicação Cliente

Em relação à melhor solução arquitetural para a aplicação cliente temos então o uso do padrão *MVC*. Conforme a Figura 4.10 nos mostra temos dois componente importantes, o componente *Views* e o componente *Controllers*. O componente *Views* é o que tem a responsabilidade de mostrar ao utilizador os dados recebidos pelo componente *Controllers* que são os responsáveis por controlar as ações que o utilizador efetua.

Os *Controllers*, no caso especifico da solução a desenvolver, correspondem ao uso de *Redux* no desenvolvimento da aplicação cliente, que se trata de um gestor de estado de toda a aplicação. O *Redux* será então o responsável pelo controlo das ligações da aplicação cliente ao servidor em que a mesma fica com a papel de *Model* neste sistema.

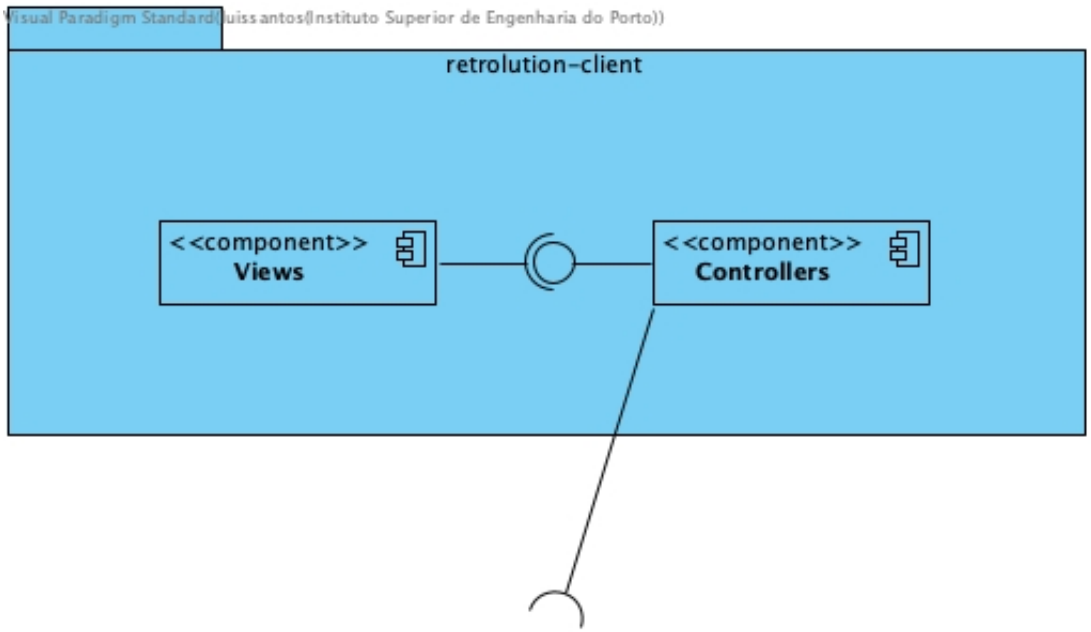


Figura 4.10: Diagrama arquitetural da aplicação cliente.

4.3 Sumário

Em síntese neste capítulo é possível perceber concretamente quais os requisitos funcionais e não funcionais da solução e com isso guiar todo o processo de desenvolvimento. Juntamente com a clarificação de quais os requisitos necessários é também possível verificar que foi levado a cabo uma análise e um estudo aproximado dos públicos alvos a que a solução se destina para que consiga responder com sucesso aos objetivos bem como às necessidades dos intervenientes.

É neste capitulo que se estuda qual a melhor forma de desenvolver a solução antes de se começar a programar a aplicação. Este estudo tem como grande objetivo perceber-se qual o melhor caminho a tomar para que o sistema funcione de forma coesa, fiável e com segurança adequada, para assim se construir uma solução sólida e com a possibilidade de crescer se surgir necessidade no futuro.

Como as teorias de padrões arquiteturais defendem, a adopção de padrões arquiteturais ajudam a prever o desenvolvimento para o mesmo se tornar mais ágil e eficaz, bem como, a desenvolver um sistema com um grau de fiabilidade superior. Um outro ponto importante que o estudo arquitetural ajuda é a forma como o desenvolvimento irá lidar com possíveis imprevistos, pois consegue se prever outros cenários e caminhos de forma a estar-se prevenido para os mesmos.

Foi possível concluir alguns pontos de partida essenciais ao desenvolvimento do sistema. Tais como, concluir que a aplicação como um todo terá uma arquitetura que se assemelha ao padrão *Layered*, contudo os elementos individuais da solução terão um padrão arquitetural diferente, que será o *MVC* assim sendo conseguimos então delinear um caminho para se iniciar o desenvolvimento prático da solução de forma a assegurar desde o início um controlo de qualidade da solução que se está a criar.

Capítulo 5

Implementação

Este capítulo tem como missão mostrar o processo de desenvolvimento e toda as decisões tomadas a quando a criação da solução criada. Todo o processo de desenvolvimento será então documentado e evidenciado e para tal foram selecionados os casos de uso mais relevantes.

Todo o processo de desenvolvimento adoptou padrões de desenvolvimento que visam assegurar a qualidade do *software* a ser criado, bem como conseguir garantir um crescimento iterativo e incremental de forma a ser possível analisar o que se está a criar para perceber se a solução está a conseguir responder aos objetivos a que se propõe.

5.1 Sistema de Controlo de Versões

Para que todo o desenvolvimento adotasse metodologias de desenvolvimento iterativas e incrementais surgir então a necessidade da utilização de um sistema de controlo de versões. Para tal foram analisadas as soluções disponíveis no mercado, onde a primeira escolha passou por ser o *Bitbucket*, no entanto ao longo do desenvolvimento o mesmo demonstrou-se insuficiente para as necessidades do projeto. Como segunda opção foi escolhido o *GitHub* onde passou a ser o local de armazenamento da aplicação bem como passou-se a utilizar o sistema de integração contínua que o mesmo oferece.

O sistema de integração contínua teve como grande função a automatização do processo de entrega contínua de *software*. Quando o mesmo era ativado, iniciava-se um processo de validação dos testes da aplicação, caso o mesmo falhasse todo o processo era interrompido. No caso de passar com sucesso era então desencadeada uma tarefa que colocava a aplicação hospedada num serviço de *cloud* e a mesma era colocada *online* para se utilizar.

Na Figura 5.1 podemos então ver o fluxo da *pipeline*[1] de desenvolvimento, o mesmo é criado quando se inicia um pedido de aprovação onde então é desencadeado o processo, inicia-se uma fase de teste de *software* e em paralelo uma análise ao código na procura de erros de escrita. No caso de falha o processo é cancelado e enviada uma notificação por email, caso passe com sucesso é então iniciado uma fase que inicia a hospedagem da aplicação num servidor de desenvolvimento para que a mesma possa ser testada. Por fim passa para a fase de revisão onde todo o código que foi para o repositório e que sofreu alterações deverá ser revisto por uma outra pessoa, no caso de aprovação é desencadeado o processo da Figura 5.2 se não é porque existem correcções a serem efetuadas e todo o processo volta se a repetir após as mesmas estarem concluídas.

[1] *Pipeline*: série de tarefas interligadas entre si, sistema de integração contínuo.

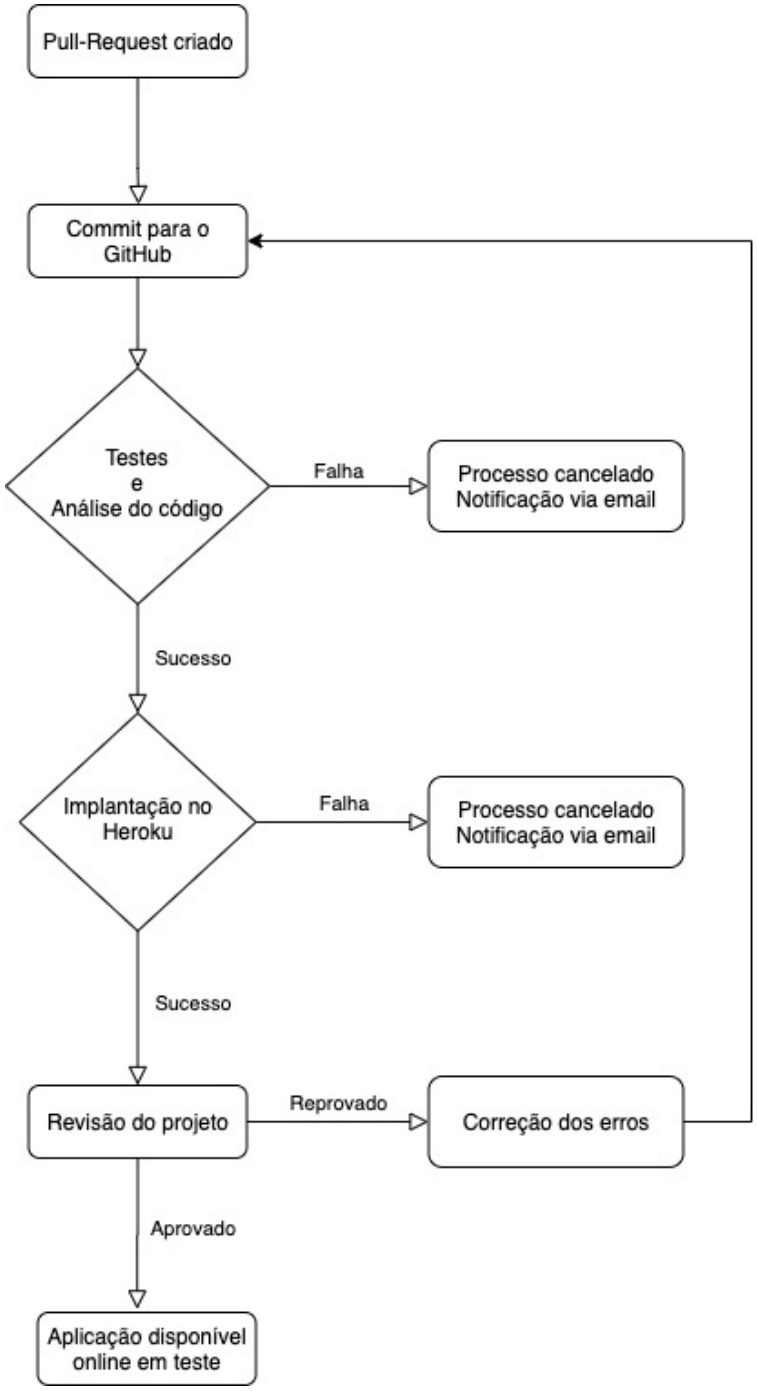


Figura 5.1: Diagrama do funcionamento da *pipeline* de desenvolvimento.

No caso da Figura 5.2 é ilustrado o diagrama de funcionamento da *pipeline* de produção, este processo só é iniciado após aprovação pelo revisor. A fase de testes e análise de código são as mesmas que na *pipeline* de desenvolvimento contudo no caso da hospedagem já é efetuada para um servidor de produção para o utilizador poder utilizar a aplicação desenvolvida.

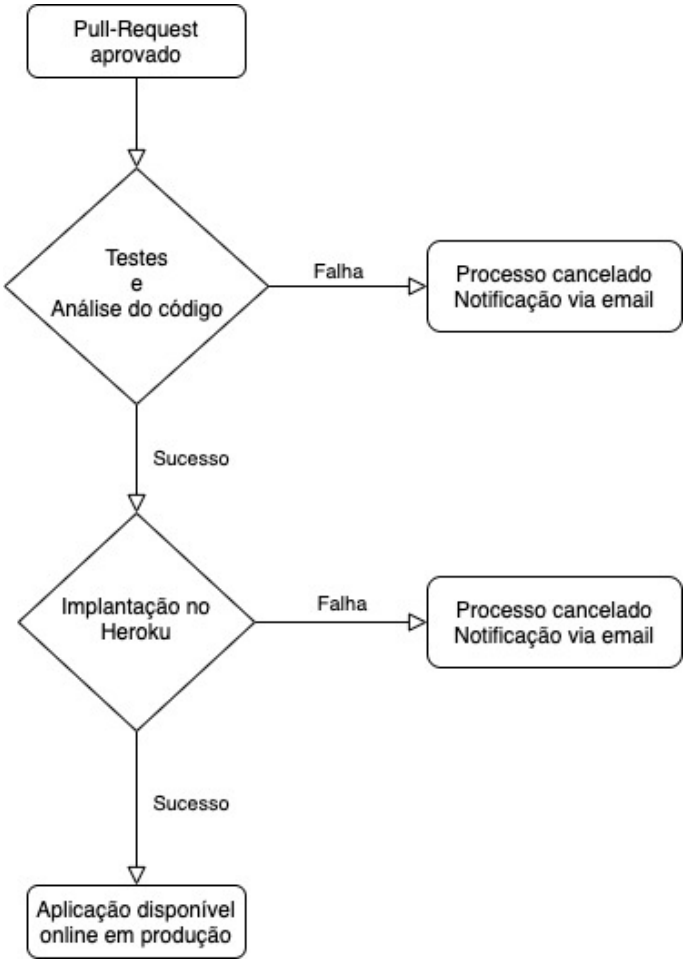


Figura 5.2: Diagrama do funcionamento da *pipeline* de produção.

5.2 Servidor

O servidor para além de ter que suportar todo o fluxo funcional associado às tarefas da realização da cerimónia de *Retrospective* também tem tem que garantir o uso de boas práticas, o controlo de qualidade do desenvolvimento bem como ter em conta a possibilidade de escalabilidade no futuro.

O servidor foi então desenvolvido em *Node*, esta escolha justifica-se pela sua facilidade de escrita e por existir uma enorme diversidade de bibliotecas complementares para ajudar a simplificar o desenvolvimento. O projeto foi então iniciado e está dividido em dois núcleos importantes, os *Models*, que são os esquemas de dados onde foi utilizada a tecnologia *mongoose* para facilitar a ligação com a base de dados como iremos ver mais à frente, e depois temos os *Routers* onde estão as funcionalidades que a aplicação necessitará.

As funcionalidades que estão nos *Routers* vão desde dos pedidos de dados, passando pela manipulação de objetos já existentes, criação de novos ou ainda quando necessário remover algum objeto da base de dados.

```
1 router.get('/', auth, async (req, res) => {
2     try {
3         const boards = await Board.find({
4             $or: [{ owner: req.user.id }, { guests: req.user.id }]
```

```
        })
            .sort({ createdAt: -1 })
            .populate('owner', 'displayName profilePic')
            .populate({
                path: 'guests',
                model: 'User',
                options: { sort: 'displayName' },
                select: 'displayName profilePic'
            })
            .populate({
                path: 'sections',
                populate: {
                    path: 'cards',
                    model: 'Card',
                    options: { sort: 'publish -publishDate' },
                    populate: {
                        path: 'owner',
                        model: 'User',
                        select: 'displayName profilePic'
                    }
                }
            })
            .populate({
                path: 'sections',
                populate: {
                    path: 'cards',
                    model: 'Card',
                    options: { sort: 'publish -publishDate' },
                    populate: {
                        path: 'group',
                        model: 'Group'
                    }
                }
            })
            .populate({
                path: 'actions',
                options: { sort: 'publish -publishDate' },
                populate: {
                    path: 'owner',
                    model: 'User',
                    select: 'displayName profilePic'
                }
            })
            .populate({
                path: 'actions',
                options: { sort: 'publish -publishDate' },
                populate: {
                    path: 'taskOwner',
                    model: 'User',
                    select: 'displayName profilePic'
                }
            });

        const response = boards.map(board => {
            return {
                ...board.toObject(),
                sections: board.sections.map(section => {
                    return {
                        ...section.toObject(),
```

```
64                          cards: section.cards.map(card => {
65                              return {
66                                  ...card.toObject(),
67                                  group: {
68                                      ...card.toObject().group,
69                                      votes: undefined,
70                                      totalVotes: card.group.votes.length,
71                                      userVotes: card.group.votes.filter(
72                                          item =>
73                                              item.toString() ===
74                                              req.user.id.toString()
75                                      ).length
76                                  }
77                              };
78                          })
79                      };
80                  })
81              };
82          });
83
84          res.send(response);
85      } catch (err) {
86          return res.status(500).send({ developerMessage: err.message });
87      }
88  });
```

Listing 5.1: Excerto de um *Router* do servidor.

No excerto do código anterior podemos verificar um exemplo de uma funcionalidade que é o pedido ao servidor para a obtenção de todas as *boards* existentes na base de dados. O utilizador que efetua o pedido é o *owner* (o criador da mesma) ou então que o mesmo seja um *guest* da *board* (*guest* é um utilizador que participou numa sessão de *Retrospective*). Os *populates* são funcionalidades associadas ao *mongoose* que servem para a resposta que vem do servidor vir com os dados que pertencem a outros modelos de dados populados. Como por exemplo o *owner* é um elemento da *board* que é criado num outro modelo de dados, nomeadamente o do *user*, e de forma a obtermos o *displayName* do mesmo é invocada a função *populate* em que a mesma vai ao modelo de *user* buscar essa informação. O *res.send(response)* é a função responsável por pegar na informação que vem da base de dados, que neste caso é o objeto *response* e enviar a mesma para o servidor.

Autenticação e Autorização

Um dos requisitos é que somente utilizadores com o *email* terminado *@mindera.com* pudessem criar conta e utilizar a aplicação. Desta forma o processo de registo e *login* de utilizadores foi desenvolvido utilizando um *middleware*[2] chamado de *Passport.js* em que o mesmo disponibiliza estratégias especificas para contas *Google* à qual o *email @mindera.com* pertence. Desta forma nas configurações das credenciais que o *Passport.js* necessita da própria *Google* basta configurar para que somente domínios internos do grupo *@mindera.com* possam entrar na aplicação, o mesmo pode ser visto na Figura 5.3.

[2] *Middleware*: Funções auxiliares que servem como intermediários entre outras funções.

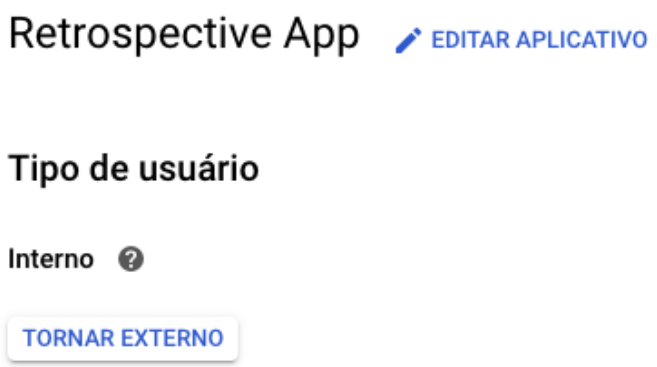


Figura 5.3: Configuração na plataforma da *Google*.

Já no desenvolvimento é somente necessário a criação de um ficheiro de configuração para que o projeto passe a utilizar o *Passport.js*. O excerto de código que se segue é o ficheiro de configuração do mesmo.

```
const passport = require('passport');
const GoogleStrategy = require('passport-google-oauth20');
const User = require('../models/user');

passport.serializeUser((user, done) => {
    //cookie
    done(null, user.id);
});

passport.deserializeUser((id, done) => {
    User.findById(id).then(user => {
        done(null, user);
    });
});
if (process.env.NODE_ENV !== 'test') {
    passport.use(
        new GoogleStrategy(
            {
                //options for the strategy
                clientID: process.env.GOOGLE_CLIENT_ID,
                clientSecret: process.env.GOOGLE_CLIENT_SECRET,
                callbackURL: process.env.CALLBACK_URL
            },
            (token, acessToken, refreshToken, profile, done) => {
                User.findOne({ providerId: profile.id }).then(
    currentUser => {
                    if (currentUser) {
                        done(null, currentUser);
                    } else {
                        new User({
                            email: profile._json.email,
                            firstName: profile.name.givenName,
                            lastName: profile.name.familyName,
                            displayName: profile.displayName,
                            providerId: profile.id,
                            profilePic: profile._json.picture
                        })
                            .save()
                            .then(newUser => {
```

```
                                    done(null, newUser);
                                });
                            }
                        });
                    }
                )
            );
        }
```

Listing 5.2: Ficheiro de configuração do *Passport.js*.

Podemos verificar no código acima ilustrado que a estratégia usada é a da *Google*. Esta função é chamada onde diz *new GoogleStrategy* e possui dois elementos, o primeiro é um objeto associado a configurações com a própria *Google* e o segundo é a função responsável por verificar se o utilizador que está a tentar fazer *login* já existe, caso sim então o mesmo é chamado à base de dados caso não exista é criado um utilizador novo.

Tempo Real

Um requisito fundamental para todo o sistema é que o mesmo tem que operar em tempo real, para tal foi utilizada a tecnologia *WebSocket*. Esta é uma tecnologia que permite a troca de dados *Hypertext Transfer Protocol (HTTP)* entre um servidor e um cliente numa ligação bidireccional fazendo com que sempre que haja novos dados os mesmos são difundidos em tempo real (Soewito et al. 2019).

Para o uso de comunicações em tempo real é necessário criar uma via de comunicação entre os dois meios, segue abaixo o que foi efetuado para se conseguir a comunicação em tempo real.

```
var io = require('socket.io')(server);
```

Listing 5.3: Iniciação *Socket.IO* no servidor.

A linha anterior de código é responsável por informar toda a aplicação que existe a possibilidade de existir comunicações via *WebSocket*, no entanto no projeto não foram utilizadas *WebSockets* nativas mas sim uma biblioteca *JavaScript*, a *Socket.IO* que utiliza *WebSockets* na sua base, contudo todo o processo de implementação e utilização é muito mais simplificado.

Após a aplicação estar informada é então que passamos ao uso especifico onde é aberta um porta de comunicação especifica para uma determinada tarefa.

```
res.io.emit(board._id, {
    type: 'updateBoard',
    payload: responseSocket
})
```

Listing 5.4: Exemplo de uma emissão de comunicação do lado do servidor.

No excerto anterior podemos verificar o seguinte, a função *emit* que é a responsável pela a abertura de uma porta para a emissão de informação, que depois a aplicação cliente estará à escuta para a processar. Dentro da própria função como primeiro elemento temos o nome associado a porta, no exemplo será o *id* da *board* de forma a garantir que cada porta tem um nome único entre *boards* e que somente os utilizadores que estiverem nessa mesma *board* é que recebem os dados correspondentes.

De seguida temos um objeto que é a informação a ser enviada para a aplicação cliente, neste caso o objeto tem duas propriedades, o *type* e o *payload*. O *type* serve para distinguir os tipos de dados que estão a ser transferidos e o *payload* refere se aos dados concretos que são transferidos. Resumidamente temos um tipo de dados que corresponde à atualização das informações referentes a uma *board* e a informação a ser enviada é a própria *board* atualizada.

Base de Dados

Relativamente a base de dados, a escolha recaiu sobre uma base de dados não relacional, justificada pela facilidade de implementação e por responder a todas as necessidades do sistema desenvolvido.

Para tal foi escolhido a plataforma *MongoDB* para a hospedagem da base de dados, assim como foi utilizado a tecnologia *mongoose* para a escrita das ligações à mesma. *Mongoose* é uma ferramenta de modelagem de objetos que facilita o desenvolvimento para *MongoDB* e assenta uma base de esquemas.

```
1  const mongoose = require('mongoose');
2  const Section = require('./section');
3
4  const cardSchema = new mongoose.Schema(
5      {
6          publish: {
7              type: Boolean,
8              default: false
9          },
10         text: {
11             type: String,
12             required: true
13         },
14         merge: {
15             type: Boolean,
16             default: false
17         },
18         owner: {
19             type: mongoose.Schema.Types.ObjectId,
20             ref: 'User',
21             required: true
22         },
23         group: {
24             type: mongoose.Schema.Types.ObjectId,
25             ref: 'Group'
26         },
27         publishDate: {
28             type: Date
29         }
30     },
31     { timestamps: true }
32 );
33
34 cardSchema.post('findOneAndDelete', async function(doc, next) {
35     const cardId = doc._id;
36     await Section.updateMany({}, { $pullAll: { cards: [cardId] } });
37     next();
38 });
39
```

```
40 const Card = mongoose.model('Card', cardSchema);
41
42 module.exports = Card;
```

Listing 5.5: Exemplo de um esquema para a base de dados.

O excerto de código anterior mostra um ficheiro completo de um modelo de dados que utiliza os esquemas do *mongoose* para efetuar a modelação do objeto para a base de dados.

A ligação entre o servidor e a base de dados é feita através de uma função de ligação que é responsável pela ligação à base de dados, o excerto de código que se segue é essa mesma função.

```
1  const mongoose = require('mongoose');
2
3  module.exports = function() {
4      if (process.env.NODE_ENV !== 'test') {
5          mongoose
6              .connect(process.env.MONGO_DB, {
7                  useUnifiedTopology: true,
8                  useNewUrlParser: true,
9                  useCreateIndex: true,
10                 useFindAndModify: false
11             })
12             .then(() => console.log('Connected to MongoDB...'))
13             .catch(err =>
14                 console.error('Could not connect to MongoDB...', err.
       stack)
15             );
16     }
17 };
```

Listing 5.6: Função que efetua a ligação entre o servidor e a base de dados.

Para a interação com a base de dados são utilizadas *queries* (linguagem de computador utilizada para consultas em bancos de dados) em função da manipulação que desejamos efetuar.

```
1 const board = await Board.findOne({
2     _id: req.params.id
3 });
```

Listing 5.7: Exemplo de uma função para interação com a base de dados.

No caso da função apresentada, temos a ação de pedir um determinado objeto em função do seu *id*. Existem funções para a obtenção de todos os objetos de uma dada categoria, para a atualização de um determinado objeto ou ainda funções para a eliminação de objetos da base de dados.

Testes

Para garantir a qualidade do *software* que estava em desenvolvimento foram então criados testes unitários utilizando *Jest* que é uma estrutura de testes para *JavaScript*. No excerto de código apresentado de seguida verifica se um exemplo de um teste unitário.

```
1 it('should obtain all boards successfully', async () => {
2     const response = await request
```

```
        .get('/boards')
        .set('x-auth-token', token);

    const expectedBoard = {
        owner: { _id: decoded.id, displayName: decoded.displayName },
        title: boards[1].title,
        team: boards[1].team,
        numVotes: boards[1].numVotes,
        status: boards[1].status,
        lock: boards[1].lock
    };

    expect(response.status).toBe(200);
    expect(response.body).toHaveLength(3);
    expect(response.body[1]).toMatchObject(expectedBoard);
    boardProps.forEach(prop => {
        expect(response.body[0]).toHaveProperty(prop);
    });
});
```

Listing 5.8: Exemplo de um teste unitário.

O teste apresentado refere se ao pedido de obtenção de todas as *boards* que existam na base de dados em que o utilizador que faz o pedido é o criador das mesmas ou então participou como convidado. O *response* é o pedido efetuado, o *expectedBoard* é o objeto que esperamos obter. Os *expect* são as validações que então queremos fazer, a primeira é expectável que se obtenha uma resposta com sucesso, a segunda validação é o número total de objetos que a resposta apresenta tem que ser 3, na terceira o objeto presente na posição 1 do *array* de resposta terá que corresponder com o objeto *expectedBoard*, por fim a última validação verifica se os objetos que vem na resposta contêm todos os campos que desejamos.

Os testes geram um relatório de cobertura que é apresentado na Figura 5.4. Verifica se no relatório apresentado que existe uma cobertura acima dos 80%.

All files

84.88% Statements 668/787 **82.91%** Branches 165/199 **66.98%** Functions 142/212 **85.44%** Lines 657/769

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

File	Statements		Branches		Functions		Lines	
models	100%	49/49	100%	0/0	100%	4/4	100%	49/49
routes	83.88%	619/738	82.91%	165/199	66.35%	138/208	84.44%	608/720

Figura 5.4: Cobertura geral dos testes.

Na Figura 5.5 encontra se o relatório de testes referente aos *Routers* em que temos uma cobertura acima dos 80%. Verifica se em alguns casos que a cobertura não é superior, isto acontece porque existem determinadas funcionalidades que são usadas especificamente em certos casos de uso de extrema dificuldade de reprodução o que leva a que não atinja valores mais altos.

All files routes

83.88% Statements 619/738 **82.91%** Branches 165/199 **66.35%** Functions 138/208 **84.44%** Lines 608/720

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

File	Statements		Branches		Functions		Lines	
boards.js	81.47%	453/556	81.46%	123/151	64.8%	116/179	82.19%	443/539
groups.js	88.79%	103/116	85.29%	29/34	68.18%	15/22	88.79%	103/116
openedActions.js	92.86%	26/28	87.5%	7/8	100%	3/3	92.59%	25/27
sections.js	100%	14/14	100%	2/2	100%	1/1	100%	14/14
users.js	95.83%	23/24	100%	4/4	100%	3/3	95.83%	23/24

Figura 5.5: Cobertura geral dos testes.

5.3 Cliente

No lado do utilizador temos o cliente que é a parte responsável pela interação com o utilizador. O desenvolvimento do cliente tem que ter em conta não só todos os cuidados no processo de desenvolvimento, como a adoção de boas práticas, bem como deverá ter em atenção todo o aspeto gráfico e o fluxo de interação com os utilizadores.

Estudo gráfico

Antes de se iniciar o processo de desenvolvimento prático do cliente foi levado a cabo um estudo gráfico de forma a chegar a uma solução gráfica para a aplicação. Este estudo teve em conta quais as gamas de cores que habitualmente a empresa utiliza de forma a ficar o mais integrado possível com a imagem da empresa, teve ainda atenção ao fluxo de funcionamento que mais se adequa às necessidades das equipas que vão utilizar a solução.

Como primeiro passo foi escolhido uma página inicial de forma a criar logo à partida algumas linhas condutoras para toda a solução.

Como estudos inicias surgiram varias versões, na Figura 5.6 e na Figura 5.7 temos dois estudos de *design*. Os mesmos foram sofrendo refinamentos até se obter uma interface gráfica agradável e intuitiva. Ao mesmo tempo foi efetuado um estudo de cores. Todas as cores utilizadas seguem o manual gráfico disponibilizado pela *Mindera* bem como os tipos de letra que são os habitualmente utilizados pela empresa.

Figura 5.6: Versão 1 para o *design* da página inicial.

Figura 5.7: Versão 2 para o *design* da página inicial.

Foi feito um levantamento de possíveis nomes para a aplicação onde surgiram varias ideias como *RetroMinder, RetroM, Rettrying* ou *RetroLift*, no entanto o nome escolhido foi *Retrolution* que é uma união entre a palavra *retrospective* e a palavra *evolution* que dão um certo simbolismo ao nome, pois a grande finalidade das cerimónias de *Retrospective* é que as equipas consigam evoluir preferencialmente com a ajuda da solução desenvolvida.

A Figura 5.8 é *design* final a que se chegou, foram escolhidas cores base claras pois é o normalmente associado a aplicações de produtividade, e para dar alguns relevos e pormenores foi escolhido o amarelo que a empresa usa como base para toda a sua imagem de marca.

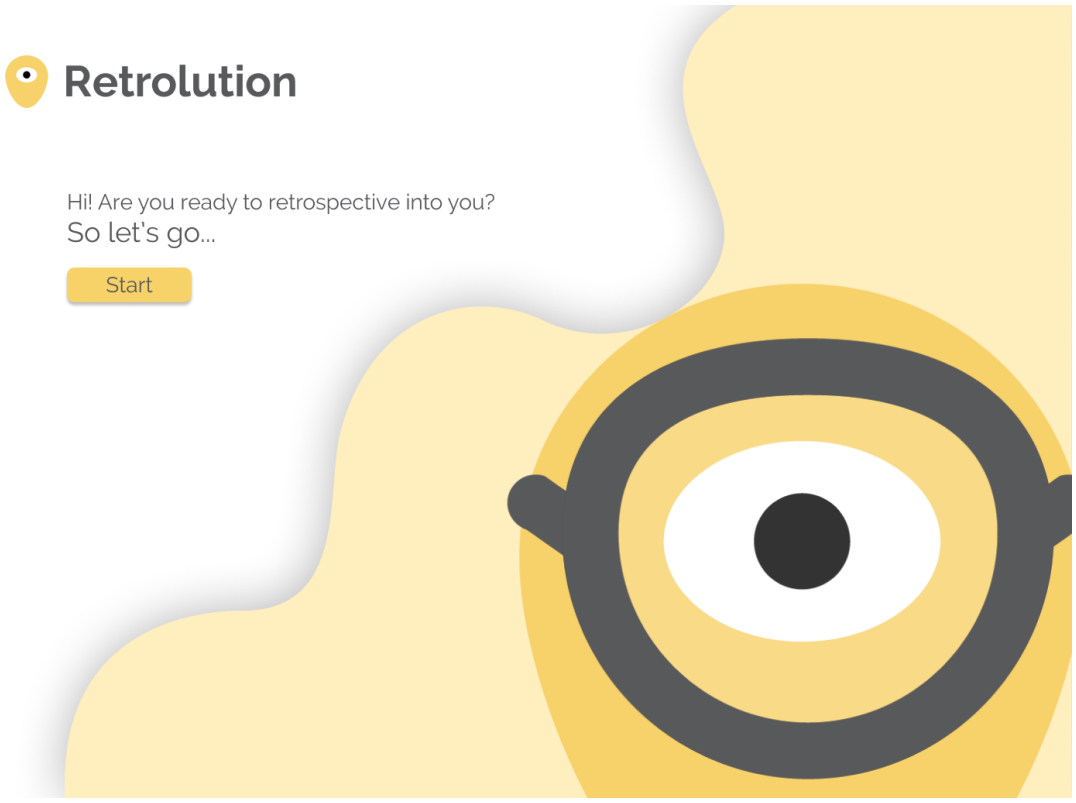


Figura 5.8: Versão final da página inicial.

Na Figura 5.9 temos um ecrã de exemplo da aplicação. O ecrã em questão é o de uma cerimónia de *Retrospective* e a mesma já se encontra na fase das *actions*. Verifica se a adoção de cores neutras e claras, tipo de letras sem ornamentos e são usados ícones quando possível de forma a tornar se mais intuitivo.

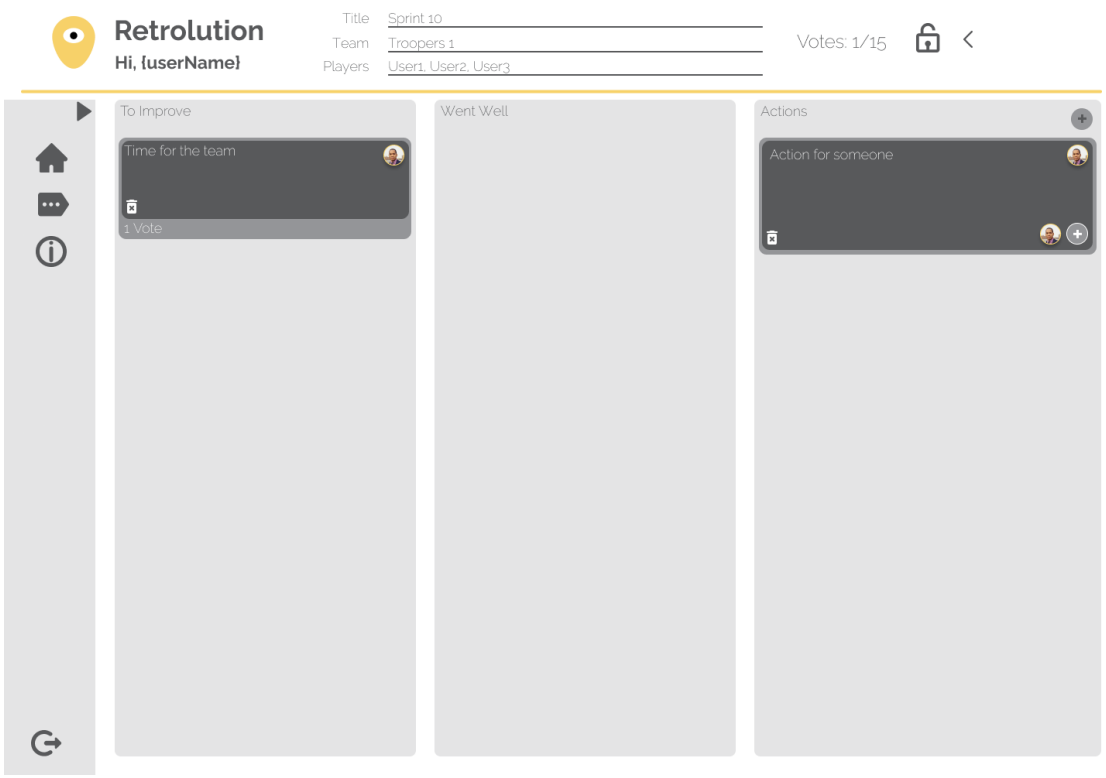


Figura 5.9: Exemplo de um ecrã da aplicação.

React

A linguagem adotada como já referenciado e justificado anteriormente foi o *React*. Assim sendo foi dado início ao desenvolvimento e para tal foi utilizado o projeto base que o próprio *React* disponibiliza, para tal, na linha de comandos foi corrido o comando *npm init react-app my-app* em que o *npm* é um gestor de pacotes de livrarias de *JavaScript* e o resto é a tarefa e o pacote a descarregar.

Dado isto é inicializado uma estruturação para que o projeto base se enquadre da melhor forma ao caminho que o desenvolvimento irá tomar. O projeto é estruturado em três grandes secções, *components, pages* e *state*. *Components* é onde serão construídos os componentes individuais e isolados que a aplicação irá fazer uso, nas *pages* são construídas as páginas que a aplicação irá ter. Como por exemplo a página principal ou a página de uma cerimónia e por fim temos *state* que é o local onde será gerido todo o estado da aplicação com a ajuda de *Redux*, que é uma biblioteca em *JavaScript* normalmente utilizada com aplicações *React* que gerem o estado da aplicação.

```
1  import React from 'react';
2  import { BrowserRouter as Router, Switch, Route } from 'react-router-dom
       ';
3  import { Global } from '@emotion/core';
4
5  import global from './components/themes/global';
6  import Home from './pages/home/Home';
7  import BasePage from './pages/basePage/BasePage';
8  import Board from './pages/board/Board';
9  import NotFound from './pages/404/NotFound';
10 import History from './components/historyAccount/History';
```

```
11 import OpenedActions from './components/openedActions/OpenedActions';
12
13 function App() {
14     return (
15         <>
16             <Global styles={global} />
17             <Router>
18                 <Switch>
19                     <Route exact path="/">
20                         <Home />
21                     </Route>
22                     <Route exact path="/home">
23                         <BasePage>
24                             <History />
25                         </BasePage>
26                     </Route>
27                     <Route exact path={['/boards', '/boards/:id']}>
28                         <BasePage>
29                             <Board />
30                         </BasePage>
31                     </Route>
32                     <Route exact path={'/openedactions'}>
33                         <BasePage>
34                             <OpenedActions />
35                         </BasePage>
36                     </Route>
37                     <Route path="*">
38                         <NotFound />
39                     </Route>
40                 </Switch>
41             </Router>
42         </>
43     );
44 }
45
46 export default App;
```

Listing 5.9: Ficheiro base de constução de toda a aplicação cliente.

Como exemplo temos o excerto de código anterior que é o ficheiro que de certa forma funciona como esqueleto de toda a aplicação cliente. É aqui que dizemos o que é que visualmente corresponde a cada rota da pesquisa. Por exemplo o elemento *Route* define uma rota em que é definido um caminho (*path*) que corresponde ao local onde o utilizador irá estar, */home* é quando o utilizador for ao site da aplicação */home* irá ser encaminhado para a então página principal após o *login* e o que iremos mostrar ao utilizador é então o seu histórico de *Retrospectives*, que é representado pelo componente *BasePage* mais o componente *History*. A propriedade *exact* define que o utilizador tem que estar exatamente naquele caminho e caso a mesa não seja defindа num exemplo de */home/menu* o utilizador não iria aceder ao *menu* mas sim à *home* porque é a que aparece primeiro na rota.

CSS-in-JS

Para estilar a aplicação foi utilizada uma técnica chamada de *CSS-in-JS*, que é uma técnica que utiliza *JavaScript* para resolver problemas complexos de *CSS*, quando o *JavaScript* analisa o componente o mesmo é convertido para *CSS*. De ter em conta que *CSS-in-JS* não é uma livraria mas sim uma abordagem que utiliza bibliotecas para o fazer e neste momento

existem 3 grandes bibliotecas, a *JSS*, a *Styled Components* e a escolhida para o uso neste trabalho, a *Emotion* (Isonen 2019).

Assim sendo para o desenvolvimento foi então escolhida a biblioteca *Emotion* e de seguida temos um excerto de código que mostra como é que se cria um componente deste género.

```
import styled from '@emotion/styled/macro';

import { white, darkGrey } from '../themes/colors';
import { mainFont } from '../themes/typography';

const Button = styled.button`
    color: ${darkGrey};
    font-family: ${mainFont};
    background: none;
    border: none;
    transition: color 0.1s ease-in;

    &:hover {
        color: ${white};
        cursor: pointer;
    }

    &:disabled {
        cursor: not-allowed;
    }
`;

export default Button;
```

Listing 5.10: Componente de estilo criado com *CSS-in-JS*.

No exemplo anterior temos então um componente que é um botão, este é um componente independente e genérico de forma a poder ser reutilizado nas mais diversas situações.

Tempo Real no Cliente

Como já foi enunciado anteriormente, foi então utilizado *Socket.IO* para se efetuarem comunicações em tempo real. Do lado do servidor já foi apresentado o seu funcionamento agora irá ser mostrado o que foi feito do lado da aplicação cliente.

Do lado da aplicação cliente para estar pronto a receber comunicações via *WebSockets* foi necessário efetuar um simples ficheiro de configuração.

```
import io from 'socket.io-client';

const socket = io(process.env.REACT_APP_IO_CONNECTION);

export default socket;
```

Listing 5.11: Ficheiro de configuração do *Socket.IO* na aplicação cliente.

No excerto de código anterior temos uma só configuração para fazer, a variável de ambiente *REACT APP IO CONNECTION* é referente ao servidor, e é esta mesma função que efetua a ligação entre a aplicação cliente e o servidor.

Já na implementação foi utilizado *hooks* que é uma funcionalidade disponível na versão mais recente do *react* e é uma ferramenta que surgiu pela necessidade de se facilitar determinadas

funcionalidades. Para se implementar os *WebSockets* foi utilizado a funcionalidade *Effect Hook* (*useEffect*) que tem como grande responsabilidade fazer com o que se encontra dentro desta função seja executado sempre que o componente for renderizado ou atualizado. A função do *Socket.IO* vai estar então à escuta do que poderá vir do lado do servidor. O servidor emite um evento e a aplicação cliente quando escutar esse mesmo evento efetuará a respectiva tarefa.

```
useEffect(() => {
        if (!id) {
            return;
        }

        socket.on(id, data => {
            switch (data.type) {
                case 'updateBoard':
                    dispatch(actions.updateBoardSocket(data.payload));
                    break;

                default:
                    break;
            }
        });
    }, [dispatch, id]);
```

Listing 5.12: Exemplo de uma função que está à escuta de comunicações do lado da aplicação cliente.

Explicado o excerto de código anterior temos então a função *useEffect* dos *hooks* que envolverá toda a funcionalidade referente aos *WebSockets*. Dentro temos a funcão que inicialmente verifica se existe *id* ou não, isto serve para verificar se o utilizador está efetivamente numa *board*. Após a verificação temos a função especifica para o *WebSocket* que é a função *socket.on*. Como primeiro parâmetro da função temos o nome pela qual deverá estar à escuta, ou seja, o servidor vai emitir um evento que tem como nome o *id* da *board* do lado da aplicação cliente quando escutar esse mesmo *id* vai então desencadear a sua tarefa. A tarefa é o segundo elemento da função, que se trata de um *switch/case* que verifica o *type* que o servidor emite e efetua a respetiva tarefa. Neste caso a aplicação cliente está à escuta e apanha uma ação do *type updateBoard* que é para atulizador os dados da board e quando escuta essa mesma ação despacha uma ação para o *Redux* que será explicado de seguida.

Redux

Para toda a gestão do estado da aplicação foi utilizado *Redux*, que é um gestor de estado para aplicações *JavaScrip*. A sua grande função é ajudar que a aplicação tenha um comportamento consistente. Apesar de parecer complexo a sua forma de funcionamento e implementação são relativamente simples (Redux 2019).

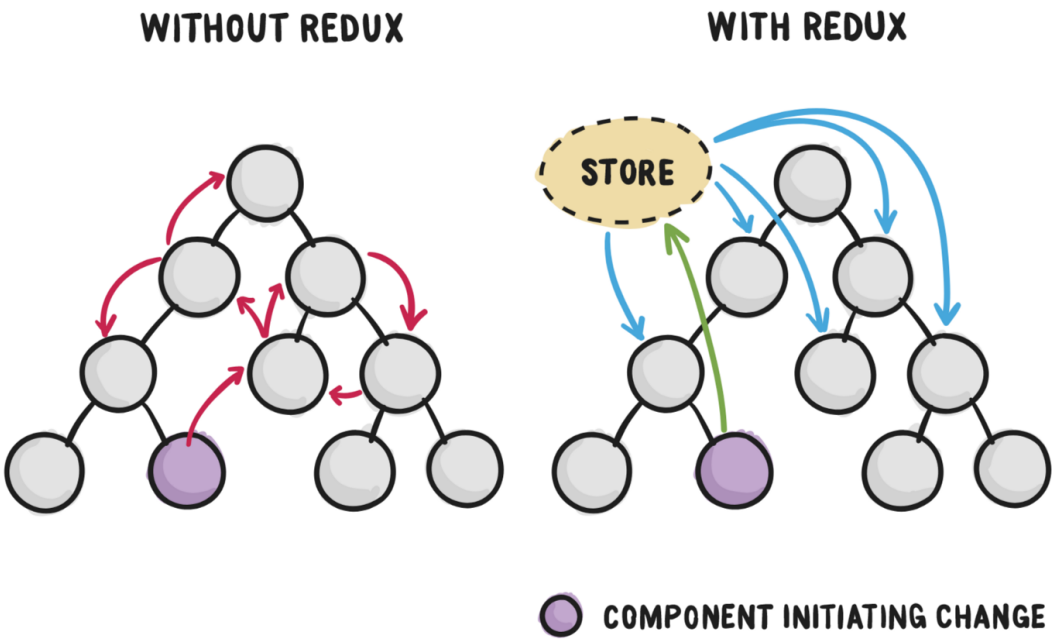


Figura 5.10: Diferença entre o uso de *Redux* e sem *Redux* (Maruta 2018).

A Figura 5.10 mostra qual a grande diferença entre a utilização de *Redux*, no lado esquerdo temos então que para navegarmos no estado da aplicação temos que percorrer toda a árvore tornado um processo mais demorado, complicado e com alta dependencia, do lado direito temos a proposta do *Redux* que passa pela utilização de uma *store* em que é guardado o estado global da aplicação e que pode ser acedido a partir de qualquer ponto da árvore.

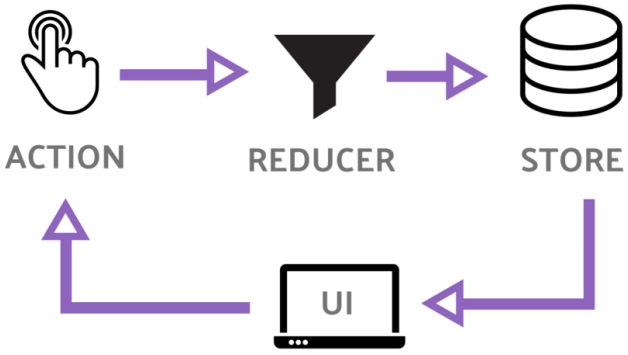


Figura 5.11: Fluxo do funcionamento do *Redux* (Fernandes 2018).

Na Figura 5.10 temos o fluxo de funcionamento do *Redux* e vemos que o mesmo é composto por três elementos, as *actions*, os *reducers* e a *store*. As *actions* são os fornecedores de dados que serão enviados para a *store* e as mesmas são lançadas por funções simples que ao serem executadas ativam os *reducers*. Os *reducers* são responsáveis pela receção dos dados e tratamento dos mesmos de forma a controlar o que é ou não enviado para a *store*. A *store* é o local onde todos os dados são armazenados de forma a ficarem disponíveis para toda a aplicação e a *store* é imutável, evoluindo apenas. Em resumo podemos dizer que as *actions* são "o quê?"os *reducers* são o "como?"e a *store* é o "onde?".

O *redux* assenta em três princípios, *Single source of truth* que significa que só existe uma verdade, ou seja, só existe uma *store*, *State is read-only* que quer dizer que o estado da aplicação é só para leitura e não escrita, sendo a única forma de alterar o estado através de ações que descrevem exatamente a ação, *Changes are made with pure functions* que significa que todas as alterações devem ser efetuadas por funções puras, ou seja, os *reducers* devem ser funções puras.

```
export const loadBoards = () => async dispatch => {
    dispatch({
        type: actionTypes.LOAD_BOARD_START
    });

    try {
        const result = await client.get('/boards');
        dispatch({
            type: actionTypes.LOAD_BOARD_SUCCESS,
            payload: result.data
        });
    } catch (error) {
        dispatch({ type: actionTypes.LOAD_BOARD_ERROR });
    }
};
```

Listing 5.13: Exemplo de uma *action* da aplicação.

No exemplo da *action* temos uma ação responsável por pedir ao servidor todas as *boards* existentes.

```
case actionTypes.LOAD_BOARD_SUCCESS:
    return {
        ...state,
        isLoading: false,
        result: action.payload
    };
```

Listing 5.14: Exemplo de um *reducer* da aplicação.

No *reducer* temos um *switch/case* que verifica o tipo da ação, neste caso, *LOAD BOARD SUCCESS* e trata os dados que recebe para os armazenar na *store*.

```
import { applyMiddleware, createStore, combineReducers } from 'redux';
import thunkMiddleware from 'redux-thunk';
import { composeWithDevTools } from 'redux-devtools-extension';
import { connectRouter, routerMiddleware } from 'connected-react-router'
    ;
import { createBrowserHistory } from 'history';

import accountReducer from './account';
import boardReducer from './board';
import openedActionsReducer from './openedActions';

const history = createBrowserHistory();

export default function configureStore(preloadedState) {
    const reducer = combineReducers({
        router: connectRouter(history),
        account: accountReducer,
        board: boardReducer,
        openedActions: openedActionsReducer
```

```
19      });
20
21      const middlewares = [routerMiddleware(history), thunkMiddleware];
22      const middlewareEnhancer = applyMiddleware(...middlewares);
23
24      const enhancers = [middlewareEnhancer];
25      const composedEnhancers = composeWithDevTools(...enhancers);
26
27      const store = createStore(reducer, preloadedState, composedEnhancers
        );
28
29      return store;
30  }
```

Listing 5.15: *Store* da aplicação.

Na criação da *store* temos a função *configureStore* que é a responsável por montar todo o estado da aplicação.

5.4 Sumário

Em forma de resumo foram explicados alguns dos pontos chave do desenvolvimento, bem como, demonstrado com exemplos práticos da solução desenvolvida. De referir que um ponto também importante e que tem impacto diretamente na utilização da aplicação é o estudo gráfico efetuado de forma a construir-se uma solução com qualidade e que responda de forma assertiva às necessidades dos seus utilizadores, neste caso, das equipas de trabalho.

Todo este desenvolvimento foi assente em boas práticas de desenvolvimento e adotando metodologias ágeis para que o cliente, neste caso o *Product Owner*, acompanhe de perto o desenvolvimento para que o crescimento da solução seja iterativo e incremental, dando assim a possibilidade de existir um constante processo de avaliação para que se possa ir integrando todas as alterações de forma contínua.

Todos estes cuidados são também fundamentais para que a sobrevivência da solução a longo prazo seja possível. Uma aplicação complexa em que não haja controlo no seu crescimento torna se em certo momento difícil de se efetuar a manutenção bem como a sua escalabilidade é posta automaticamente em causa.

Capítulo 6

Avaliação

Este capítulo tem por objetivo demonstrar como foi avaliada a solução desenvolvida. Esta avaliação tem por objetivos não só perceber se a solução desenvolvida cumpre todos os objetivos propostos como também perceber na realidade se a solução consegue colmatar todas as necessidades da equipa que a vai utilizar.

O processo de avaliação passa não só pela qualidade de *software* como acima de tudo visa perceber se a solução é eficaz na resolução do problema, para tal iremos utilizar algumas abordagens como inquéritos, para compreender a situação antes e depois da solução ser implementada a fim de se conseguir fazer uma analise e avaliação detalhada da situação e da solução.

6.1 Abordagem

Como metodologias de avaliação da solução iremos ter dois grandes campos, um do ponto de vista de desenvolvimento e outro do ponto de vista da solução em si.

Do ponto de vista do desenvolvimento toda a solução será submetida a testes unitários e de integração que visam garantir um desenvolvimento controlado pelas necessidades e requisitos do problema, bem como garantir que todo o *software* desenvolvido é de qualidade elevada.

A garantia de qualidade do desenvolvimento tem como grande objetivo o controlo de todo o código escrito para que o mesmo cumpra padrões de desenvolvimento a fim de garantir que a solução é sólida, segura e que a sua manutenção e escalabilidade não serão um problema no futuro.

Todo este processo de testes será controlado por sistemas de automatização, nomeadamente pelas *pipelines* do *GitHub*.

Do ponto de vista dos utilizadores serão feitos inquéritos que têm por finalidade recolher informações sobre a satisfação dos utilizadores face ao uso da solução desenvolvida.

6.2 Métricas

A solução criada está dividia em dois grandes elementos, o servidor, que é responsável por toda a lógica de funcionamento da aplicação e o outro elemento é a aplicação cliente que é a responsável pela comunicação com o utilizador. Devido a cada um dos módulos ter responsabilidades independentes no funcionamento da solução serão definidas métricas em função de cada um deles, e será também definido métricas para a avaliação como um todo.

Em relação ao servidor a aplicação foi submetida ao desenvolvimento de testes unitários que visão o controlo de qualidade de desenvolvimento de *software* e verificar se as funcionalidades funcionam como é espectável. Temos então a seguinte métrica:

- Testes unitários.

No lado da aplicação cliente a mesma será submetida a testes de utilização e após isso será efetuado um questionário para recolher informações obtendo a seguinte métrica:

- Satisfação geral do utilizador em relação à interface da aplicação.

Para avaliar a solução desenvolvida como um todo e para se compreender se os objetivos foram ou não alcançados será também efetuado um questionário que compara a situação antes e após a existência da solução desenvolvida, obtendo a seguinte métrica.

- Satisfação geral do utilizador em relação à aplicação.

6.3 Hipóteses

Baseando-se nas métricas anteriormente identificadas foram formuladas as respetivas hipóteses para cada uma das métricas.

- H0: **Testes unitários**
- H1: Cobertura superior a 75%
- H2: Cobertura inferior a 75%

Os valores atribuídos à percentagem em relação à cobertura dos testes unitários teve em conta o facto de se compreender que seja um valor aceitável salvaguardando que poderão existir situações particulares que afetem negativamente os resultados desejados.

- H0: **Satisfação geral do utilizador em relação à interface da aplicação**
- H1: Os resultados obtidos ter em um resultado igual ou superior a 4 (escala 0 a 5)
- H2: Os resultados obtidos ter em um resultado inferior a 4 (escala 0 a 5)

Em relação à aplicação cliente o foco é compreender se a solução desenvolvida tem ou não uma interface que satisfaça a maior parte dos utilizadores e para tal um valor igual ou superior a 4 é um número aceitável.

- H0: **Satisfação geral do utilizador em relação à aplicação**
- H1: Os resultados obtidos ter em um resultado igual ou superior a 4 (escala 0 a 5)
- H2: Os resultados obtidos ter em um resultado inferior a 4 (escala 0 a 5)

Em relação à satisfação geral da aplicação o valor considerável aceitável é igual ou superior a 4, no entanto esta hipótese deverá ter em conta os valores obtidos antes de se implementar a solução desenvolvida.

- H0: **A aplicação desenvolvida contribuiu para um aumento da elaboração de sessões de Sprint Retrospective**
- H1: A aplicação desenvolvida contribuiu positivamente
- H2: A aplicação desenvolvida não contribuiu positivamente

Esta ultima métrica tem como função responder à questão principal levantada pelo problema e a resposta poderá ser sim, não ou talvez.

6.4 Métodos

Os métodos para se proceder à avaliação em relação ao servidor, como já referido anteriormente, são os testes ao código da aplicação, testes unitários.

Em relação aos testes funcionais, os de satisfação do utilizador, terão duas ferramentas de apoio que visam ajudar a obter informações de forma a se elaborar uma avaliação sobre o produto desenvolvido e essa informação será então analisada de forma a se obter uma avaliação conclusiva.

6.4.1 Inquéritos

Os inquéritos têm em conta dois momentos temporais, uma parte das respostas são para se responder tendo em conta as soluções já existentes no mercado e a segunda parte as respostas já deverão ter em conta a solução desenvolvida.

A primeira fase de resposta deverá ter em conta antes da solução desenvolvida ser utilizada pelas equipas, esta primeira fase tem como objetivo criar um ponto de situação, ou seja, criar um ponto de referência para se conseguir fazer uma avaliação comparativa entre antes da solução e depois da solução desenvolvida ser utilizada.

A segunda fase é então após a solução desenvolvida ser implementada e utilizada pelas equipas, com esta fase pretende-se então fazer uma avaliação funcional da solução criada isto para se efectuarem conclusões sobre a assertividade da mesma, tem então como objetivo compreender e analisar se a nova aplicação soluciona ou não o problema levantado.

O modelo é destinado aos utilizadores, neste caso aos elementos das equipas que irão utilizar a aplicação desenvolvida como ferramenta de trabalho. Este inquérito tem por grande função fazer uma avaliação direta da aplicação, em que a grande resposta que se pretende obter com este inquérito é: O problema foi resolvido? Assim sendo este inquérito é efetuado antes, afim de conseguirmos um ponto de situação sem a solução existir e um após para ser possível efetuar análises e respetivas conclusões acerca da avaliação da mesma.

6.5 Análise de Resultados

De seguida temos uma análise mais detalhada aos resultados obtidos para cada uma das hipóteses levantadas.

Em relação aos questionários foram inquiridos 8 utilizadores referentes à equipa de trabalho com que a solução foi desenvolvida.

6.5.1 Testes unitários

Os testes unitários ao longo de todo o processo de desenvolvimento tiveram um cuidado contínuo. Os mesmos foram desenvolvidos pensando no controlo de qualidade da aplicação desenvolvida bem como garantir que todas as funcionalidades desejadas operavam da forma esperada.

Na Figura 6.1 está o relatório geral de cobertura de todo o servidor e é possível verificar que a mesma tem uma cobertura geral acima dos 80%.

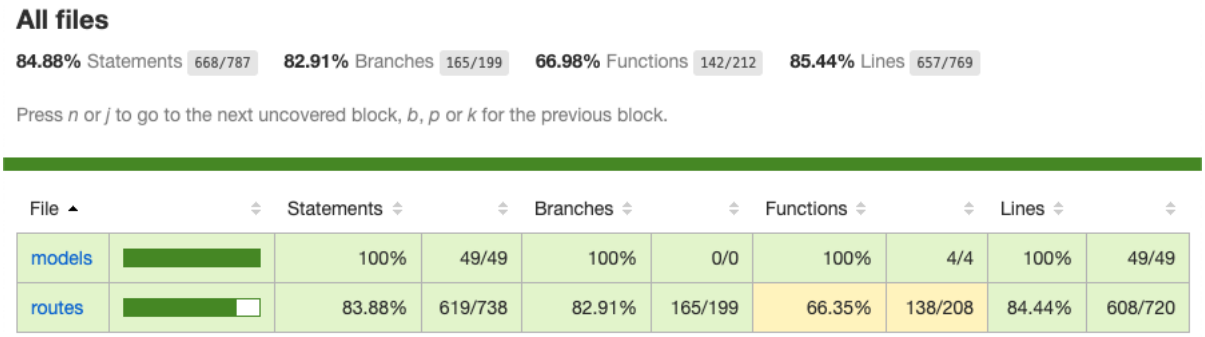


Figura 6.1: Cobertura geral dos testes.

Assim sendo e com o resultado obtido podemos aceitar a hipótese H1, que afirma uma cobertura superior a 75% em relação aos testes unitários.

6.5.2 Satisfação geral do utilizador em relação à interface da aplicação

Para a obtenção de avaliação sobre a satisfação do utilizador em relação à interface da aplicação foi efetuada a questão nos questionários realizados.

De forma a se perceber qual o fator mais relevante para o utilizador numa aplicação também se questionou sobre o que é que o utilizador dava mais importância.

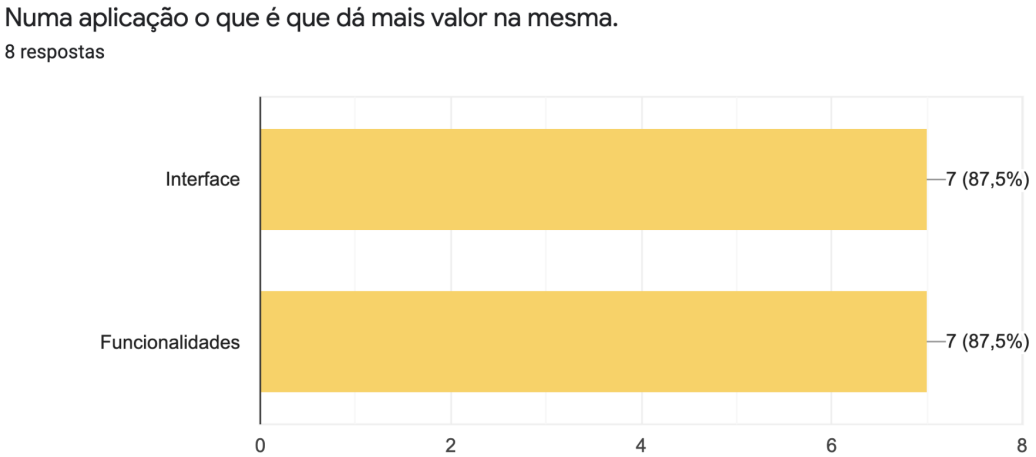


Figura 6.2: Numa aplicação o que é que dá mais valor?

Segundo a figura Figura 6.2, de um total de 8 resposta, podemos então evidenciar que 7 dos utilizadores têm no mesmo patamar de relevância a interface e as funcionalidades, querendo dizer que para o utilizador ter interesse numa aplicação deverá existir uma simbiose entre interface e funcionalidade.

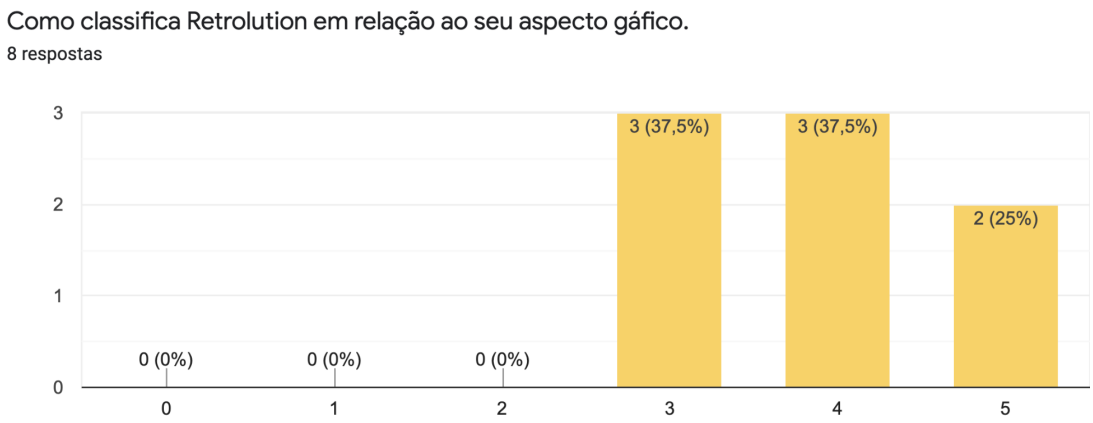


Figura 6.3: Como classifica *Retrolution* em relação à sua interface?

A Figura 6.2 mostra então que num total de 8 respostas 5 pessoas avaliaram com 4 ou mais a aplicação *Retrolution* em relação ao seu aspeto gráfico.

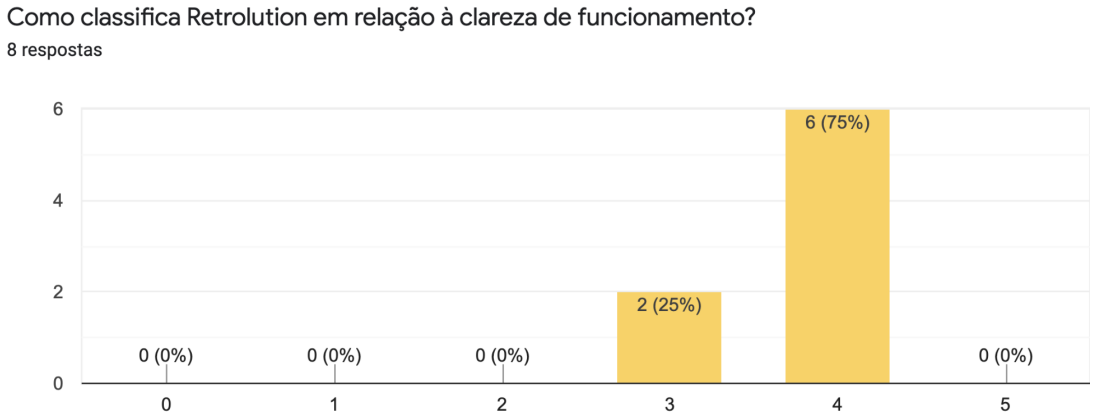


Figura 6.4: Como classifica *Retrolution* em relação ao seu funcionamento?

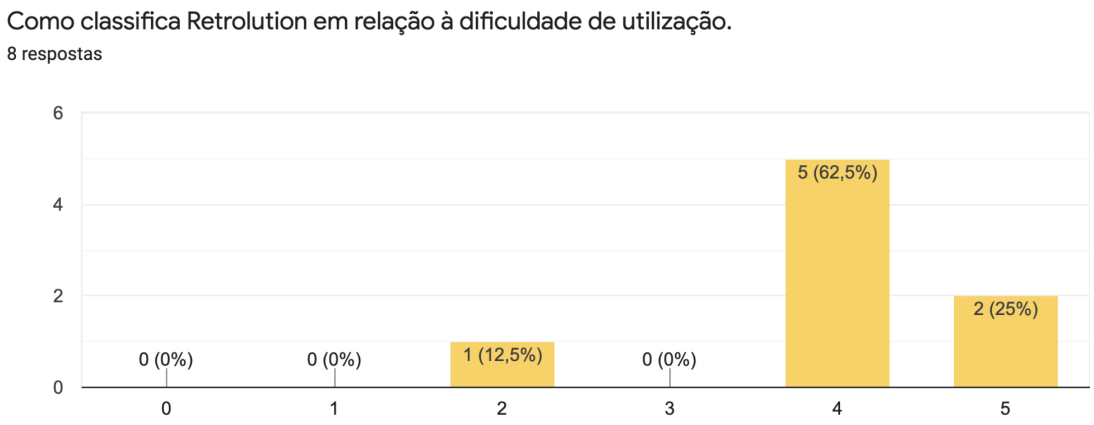


Figura 6.5: Como classifica *Retrolution* em relação à sua utilização?

As Figuras 6.4 e 6.5 mostram de forma geral que as pessoas inquiridas avaliam positivamente a aplicação em relação à sua clareza e dificuldade de funcionamento, obtendo maioritariamente respostas acima de 4.

Assim sendo podemos aceitar a hipótese H1 que diz que os resultados obtidos deverão ter em um resultado igual ou superior a 4.

6.5.3 Satisfação geral do utilizador em relação à aplicação

Em relação à satisfação geral da aplicação desenvolvida foi efetuada a questão aos utilizadores antes da implementação da mesma forma a se ter um ponto no tempo antes de existir a solução para que fosse possível compreender se realmente o *Retrolution* teria impacto ou não nas equipas.

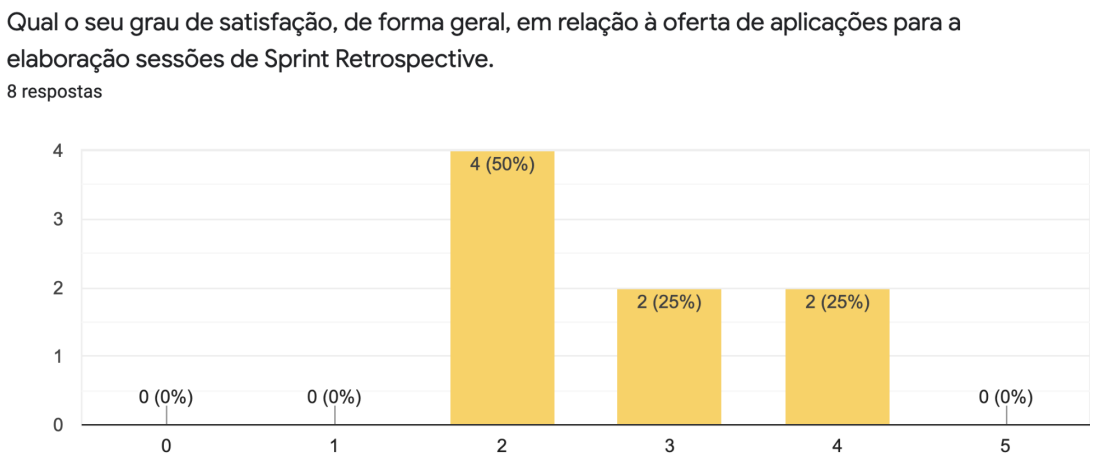


Figura 6.6: Grau de satisfação em relação às soluções já existentes.

A Figura 6.6 mostra que de forma geral os utilizadores inquiridos não estão satisfeitos com a soluções que existem no mercado com 6 e 8 a efetuarem uma votação inferior a 4.

Já na Figura 6.7 temos a mesma questão mas em relação à aplicação desenvolvida.

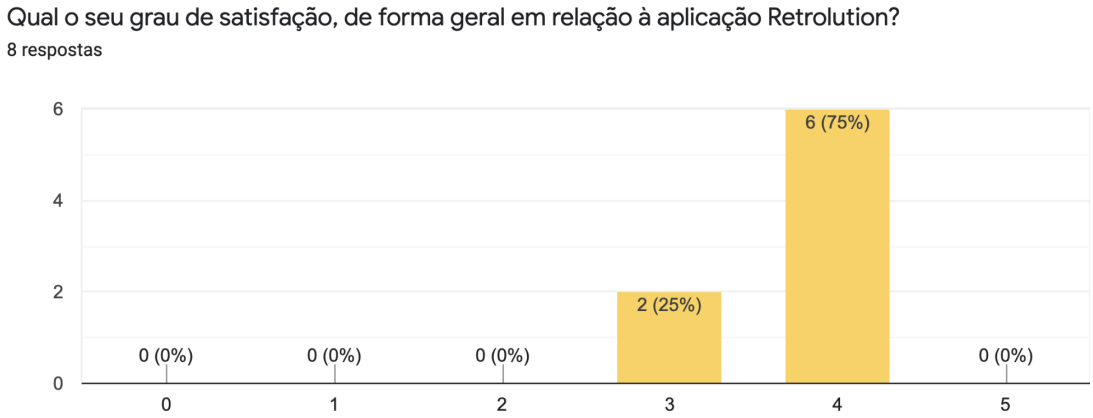


Figura 6.7: Grau de satisfação em relação ao *Retrolution*.

Com os resultados obtidos e verificados na Figura 6.7 podemos aceitar a hipótese H1 com 6 dos 8 inquiridos a votarem igual ou superior a 4, tal como afirma a hipótese.

6.5.4 A aplicação desenvolvida contribuiu para um aumento da elaboração de sessões de Sprint Retrospective

Um dos grandes objetivos com o desenvolvimento de uma solução personalizada para as equipas de trabalho, para além de responder às necessidades das mesmas, era que conseguisse impulsionar para que as as equipas realizassem mais frequentemente cerimónias de *Retrospective*.

Para tal foi levantada a questão se os utilizadores concordavam ou não que as soluções para a realização da cerimónias de *Retrospective* contribuem para a motivação das pessoas para a realização das sessões.

A Figura 6.8 mostra as respostas à questão levantada antes de existir a solução desenvolvida, ou seja, referente às soluções existentes.

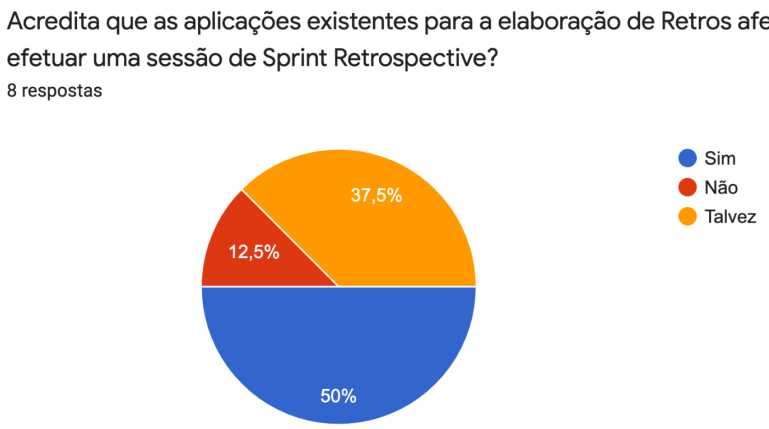


Figura 6.8: Motivação para a elaboração de *Retrospectives* em relação às soluções já existentes.

Podemos então ver que não existe uma resposta que afirma ou negue a questão levantada. Assim sendo é levantada a mesma questão em relação à aplicação criada, a *Retrolution*, e obtemos os resultados que se seguem.

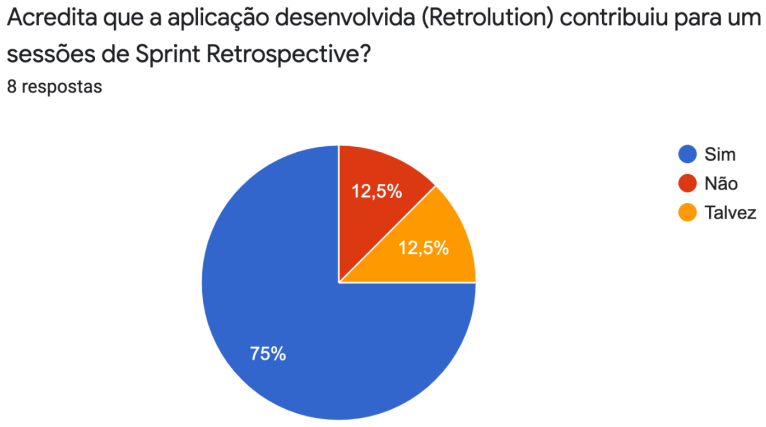


Figura 6.9: Motivação para a elaboração de *Retrospectives* em relação ao *Retrolution*.

Segundo a Figura 6.9 temos que 6 dos 8 inquiridos a afirmam que a solução desenvolvida contribui para a elaboração de cerimónias de *Sprint Retrospective* mais regulares, existindo ainda uma responda sem certezas e apenas um inquirido a negar esse impacto.

Com isto podemos então aceitar a hipótese H1 que afirma que a aplicação desenvolvida contribuiu positivamente para o elaboração mais regular de sessões de *Retrospective*.

6.6 Sumário

Concluímos que é necessário uma estratégia de avaliação afim de garantir não só padrões de qualidade de *software* bem como, e de certa forma mais importante, verificar se realmente a solução cumpre ou não com os seus objetivos e se realmente resolve ou não o problema das equipas de trabalho.

Assim sendo a melhor forma é mesmo com testes de usabilidade (inquéritos) obtendo *feedback* dos utilizadores antes e depois da solução estar implementada.

Com os resultados então obtidos conseguimos verificar que a solução criada responde em grande parte às necessidades da equipa de trabalho, podendo então afirmar-se que a mesma é uma solução viável para a elaboração de cerimónias de *Sprint Retrospective*.

Capítulo 7

Conclusão

Este capítulo tem por objetivo perceber até que ponto a solução desenvolvida respondeu às necessidades das equipas bem como verificar se todos os objetivos desta dissertação foram alcançados com sucesso.

A conclusão deste trabalho culmina numa apreciação global de todo o projeto desenvolvido, de forma a se verificar se os objetivos foram ou não compridos e que possíveis trabalhos futuros poderão existir.

7.1 Objetivos Alcançados

O grande objetivo deste trabalho era o desenvolvimento de uma solução que respondesse às necessidades das equipas de trabalho em relação à elaboração de cerimónias de *Sprint Retrospective*, assim sendo, com as provas acima dadas posso afirmar que o objetivo foi dado como comprido.

Paralelamente existem três objetivos que assentam não só no desenvolvimento do produto mas também focam se na criação de valor e no impacto que a solução criada deve ter nas equipas de trabalho, nascendo assim os seguintes objetivos:

- Desenvolver um protótipo funcional de uma aplicação web que visa o apoio às cerimónias de *Retrospective*;
- Ter impacto na melhoria contínua no desenvolvimento de soluções de uma equipa de trabalho com o apoio da solução desenvolvida com este projeto;
- Incentivar à participação colaborativa de todos os elementos de uma equipa para a melhoria contínua da mesma.

O objetivo referente ao desenvolvimento de um protótipo funcional foi atingido com sucesso com a criação da solução *Retrolution* que se encontra disponível *online* para o uso das equipas de trabalho. Todo o processo construtivo teve em conta os requisitos levantados, bem como, boas práticas de desenvolvimento de forma a se construir um produto com qualidade. Todo este processo decorreu de forma iterativa e incremental de forma a se integrar continuamente todas as melhorias e sugestões dadas pelas equipas de trabalho.

Os dois outros objetivos relacionados com o impacto das cerimónias de *Sprint Retrospective* nas equipas e motivação para as mesmas as efetuarem com sucesso também foram atingidos. As equipas nas respostas que deram aos inquéritos afirmam isto mesmo, bem como, o facto de uma equipa passar a utilizar unicamente a solução desenvolvida para a realização das suas sessões de *Retrospective*.

Em suma pode se verificar que os grandes objetivos deste trabalho foram atingidos com sucesso, obtendo-se uma solução viável para as equipas elaborarem as suas cerimónias de *Retrospective* e assim poderem usufruir das vantagens das mesmas como os mais diversos autores acima referenciados defendem.

7.2 Limitações

Como todos os processos de desenvolvimento há sempre limitações e dificuldades que surgem, e no desenvolvimento deste projeto ocorreram muitas dificuldades e limitações em todo o percurso.

Ao longo do projeto surgiram limitações técnicas que levaram com que determinados objetivos demorassem mais tempo a serem atingidos havendo a necessidade de se repensar toda a abordagem do mesmo.

A situação de pandemia que se viveu ao longo do ano foi certamente um fator limitativo para o desenvolvimento deste projeto.

7.3 Trabalhos Futuros

Apesar de todos os objetivos terem sido alcançados com sucesso, ficaram determinados requisitos por responder bem como foram encontrados pontos possíveis de melhorar.

Segue se uma lista de possíveis trabalhos futuros a se efetuar no projeto:

- Tornar a solução desenvolvida responsiva para os mais diversos dispositivos de utilização;
- Implementação de testes de interface gráfica;
- Implementação do sistema de *Tags*;
- Aumento do controlo de testes unitários de forma a se atingir o mais perto possível de 100% de cobertura;
- Desenvolvimento de novas funcionalidades que visem facilitar o trabalho das equipas;
- Incorporação de um sistema de vídeo conferência nativo na aplicação;
- Aumentar as opções disponíveis para a autenticação e registo na aplicação.

7.4 Apreciação Global

Todo este processo de desenvolvimento foi um caminho de crescimento e criação de valor não só para as equipas de trabalho com a construção de uma solução personalizada bem como do ponto de vista pessoal foi um constante crescimento, pois serviu para o aumento dos meus conhecimentos técnicos assim como os teóricos.

Sei que com este trabalho consegui compreender a dinâmica de uma equipa de trabalho com a observação que fiz e que entre equipas existem grandes diferenças nas maneiras de trabalhar ou de organizar um trabalho por exemplo.

Pessoalmente com este trabalho consegui colocar à prova o meu trabalho a solo, pois apesar de estar a trabalhar de perto com equipas da empresa todo o desenvolvimento levado a cabo foi feito unicamente por mim, exigindo uma capacidade de organização e autocrítica elevada.

Sobre o tema abordado fiquei a conhecer que existem várias abordagens ao que se refere a metodologias de desenvolvimento de *software* e apesar dos autores defenderem os seus pontos de vista um facto importante e comum entres os autores é a equipa sentir se bem com a forma como trabalha.

Em relação a *Sprint Retrospective* apesar de muitas vezes ser menosprezada é claro que a mesma consegue criar valor junto das equipas de desenvolvimento, valor esse que a longo prazo se transparece com profissionais mais capazes, talentosos e consequentemente a produzirem produtos com uma qualidade superior. Esta cerimónia também serve para uma equipa se tornar mais unida e consequentemente trabalhar de forma mais segura de si mesma e fluída.

A solução desenvolvida *Retrolution* mostra neste momento ser capaz de suportar uma cerimónia de *Sprint Retrospective* e que tem potencial para evoluir e tornar se num produto sólido e capaz de responder a todas as necessidades que as equipas têm.

Assim sendo considero que o desenvolvimento deste trabalho foi um factor de desenvolvimento pessoal e profissional e juntamente se tornou uma mais valia para as equipas de trabalho.

Bibliografia

About Reetro | Fun, Easy & 100% Free (s.d.). url: https://reetro.io/about-reetro.html.

Agile Alliance (2019). *What is Agile Software Development? | Agile Alliance*. url: https://www.agilealliance.org/agile101/.

Akbar, Muhammad Azeem et al. (fev. de 2018). «Statistical Analysis of the Effects of Heavyweight and Lightweight Methodologies on the Six-Pointed Star Model». Em: *IEEE Access* 6, pp. 8066–8079. issn: 21693536. doi: 10.1109/ACCESS.2018.2805702.

Awad, M A (2005). *A Comparison between Agile and Traditional Software Development Methodologies*. Rel. téc.

Bass, Len, Paul Clements e Rick Kazman (2003). *Software Architecture in Practice , Second Edition*, p. 560. isbn: 0321154959. url: https://books.google.pt/books?hl=pt-PT&lr=&id=mdiIu8Kk1WMC&oi=fnd&pg=PA1&dq=software+architecture&ots=UeR_R5c9PT&sig=L1sg298iexcB5-lBBu6TJ7_KKMM&redir_esc=y#v=onepage&q=software%20architecture&f=false%20https://books.google.com.co/books?id=-II73rBDXCYC&prints.

Beck, Kent et al. (2001). «Manifesto para o desenvolvimento ágil de software». Em: p. 1. url: http://agilemanifesto.org/iso/ptpt/manifesto.html%20https://www.manifestoagil.com.br/%0Ahttp://manifestoagil.com.br/.

Berra, Yogi (s.d.). *Software Development Life Cycle*.

Buettner, Kathleen e Anna M. Simmons (2011). «Mobile web and native apps: How one team found a happy medium». Em: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 6769 LNCS. PART 1. Springer, Berlin, Heidelberg, pp. 549–554. isbn: 9783642216749. doi: 10.1007/978-3-642-21675-6{_}63.

CompTIA (2019). «2019 IT (Information Technology) Industry Trends Analysis | CompTIA». Em: *CompTIA*. url: https://www.comptia.org/content/research/it-industry-trends-analysis.

Desenvolva uma proposta de valor - O blog de Marketing e Vendas (2018). url: https://www.marketing-vendas.pt/2018/10/29/desenvolva-uma-proposta-de-valor/.

Devskiller top IT skills report 2020: Demand and hiring trends (2020). url: https://devskiller.com/it-skills-report-2020/?utm_source=FreeCodeCamp&utm_medium=guest%20post&utm_campaign=BI%20report%202020.

Dmitry Gurendo (2015). *Spiral Model in Software Development Life Cycle (SDLC): Phases, Explanations, Methodology*. url: https://xbsoftware.com/blog/software-development-life-cycle-spiral-model/.

Editorial, Smashing (2011). *Responsive Web Design: What It Is And How To Use It – Smashing Magazine*. url: https://www.smashingmagazine.com/2011/01/guidelines-for-responsive-web-design/.

Elliott, Geoffrey (2004). *Global Business Information Technology: an integrated systems approach*. Addison-Wesley. isbn: 0321270126.

Fernandes, Rodrigo F. (2018). *Integrating semantic-ui modal with Redux – ITNEXT*. url: https://itnext.io/integrating-semantic-ui-modal-with-redux-4df36abb755c.

Filho, Miguel Marcondes (2018). *Pocket Sprint Retrospective*. url: https://share.atelie.software/pocket-sprint-retrospective-como-facilitar-uma-retrospectiva-enxuta-1e4895ccccab.

Fortune (2019). *100 Fastest-Growing Companies | Fortune*. url: https://fortune.com/100-fastest-growing-companies/%20https://fortune.com/100-fastest-growing-companies/2019/.

Fowler, Frederik M. (2019). «What Is Scrum?» Em: *Navigating Hybrid Scrum Environments*, pp. 3–8. doi: 10.1007/978-1-4842-4164-6{_}1. url: https://www.scrum.org/resources/what-is-scrum.

Fowler, Marin (2001). *Writing The Agile Manifesto*. url: https://martinfowler.com/articles/agileStory.html%20http://martinfowler.com/articles/agileStory.html.

FunRetro | About us and how we started. (S.d.). url: https://funretro.io/about.

Hartwich, Michał (2018). *Node.js vs Java Comparison - Differences in Performance, Development And Uses | Netguru Blog on Node.js*. url: https://www.netguru.com/blog/node.js-vs-java-comparison-differences-in-performance-development-and-uses.

Hughey, Douglas (2017). *The Traditional Waterfall Approach*. url: https://www.umsl.edu/~hugheyd/is6840/waterfall.html.

Ishak, Irny Suzila (2005). «DESIGNING A STRATEGIC INFORMATION SYSTEMS PLANNING METHODOLOGY FOR MALAYSIAN INSTITUTES OF HIGHER LEARNING (ISP-IPTA)». Em: *Issues in Information Systems* 6.1, pp. 325–331. issn: 1529-7314.

Isonen, Oleg (2019). *What actually is CSS-in-JS?. CSS-in-JS refers to a collection of... | by Oleg Isonen | DailyJS | Medium*. url: https://medium.com/dailyjs/what-is-actually-css-in-js-f2f529a2757.

Javanmard, Mahdi e Maryam Alian (2015). «Cumhuriyet Üniversitesi Fen Fakültesi Comparison between Agile and Traditional software development methodologies». Em: *Özel Sayı Science Journal (CSJ)* 36.3, p. 36. issn: 1300-1949. url: http://dergi.cumhuriyet.edu.tr/cumuscij%C2%A92015%20http://dergi.cumhuriyet.edu.tr/cumuscij.

Jensen, Nathan et al. (ago. de 2020). «Conspicuous monitoring and remote work». Em: *Journal of Economic Behavior and Organization* 176, pp. 489–511. issn: 01672681. doi: 10.1016/j.jebo.2020.05.010.

Jobe, William (out. de 2013). «Native Apps Vs. Mobile Web Apps». Em: *International Journal of Interactive Mobile Technologies (iJIM)* 7.4, p. 27. issn: 1865-7923. doi: 10.3991/ijim.v7i4.3226.

Koen, Peter (s.d.). *Front End Innovation - What is the New Concept Development (NCD) model?* url: http://frontendinnovation.com/fei/what-is-the-new-concept-development-ncd-model.

Kurnia, Reni, Ridi Ferdiana e Sunu Wibirama (nov. de 2018). «Software metrics classification for agile scrum process: A literature review». Em: *2018 International Seminar on Research of Information Technology and Intelligent Systems, ISRITI 2018*. Institute of Electrical e Electronics Engineers Inc., pp. 174–179. isbn: 9781538674222. doi: 10.1109/ISRITI.2018.8864244.

Marcotte, Ethan (2010). *A List Apart - Responsive Web Design*. url: http://alistapart.com/article/responsive-web-design.

Mariia Lozhko (2019). *15 Top web development trends in 2020*. url: https://lanars.com/blog/top-web-development-trends.

Maruta, Rafael (2018). *Iniciando com Redux em 9 passos – React Brasil – Medium*. url: https://medium.com/reactbrasil/iniciando-com-redux-c14ca7b7dcf.

Meet the Retrium Team: Retrium is a Retro Software Company | Retrium (s.d.). url: https://www.retrium.com/about.

metodologia | Definição ou significado de metodologia no Dicionário Infopédia da Língua Portuguesa (2003). url: https://www.infopedia.pt/dicionarios/lingua-portuguesa/metodologia.

Mindera (2019). *Mindera*. url: https://mindera.com/.

Morlion, Peter (2018). «Software Architecture: The 5 Patterns You Need to Know - DZone Microservices». Em: *DZone*, Microservices zone. url: https://dzone.com/articles/software-architecture-the-5-patterns-you-need-to-k.

Nick Karnik (2018). *Introduction to Mongoose for MongoDB*. url: https://www.freecodecamp.org/news/introduction-to-mongoose-for-mongodb-d2a7aa593c57/.

Ockerman, Stephanie (2019). *What is a Daily Scrum?* url: https://www.scrum.org/resources/what-is-a-daily-scrum.

Paasivaara, Maria et al. (jun. de 2017). «Do high and low performing student teams use scrum differently in capstone projects?» Em: *Proceedings - 2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering and Education Track, ICSE-SEET 2017*. Institute of Electrical e Electronics Engineers Inc., pp. 146–149. isbn: 9781538626719. doi: 10.1109/ICSE-SEET.2017.22.

Potter, John (2019). *react vs vue vs @angular/core | npm trends*. url: https://www.npmtrends.com/react-vs-vue-vs-angular.

Redux (2019). *Getting Started with Redux · Redux*. url: https://redux.js.org/introduction/getting-started.

Saarikoski, Heli e David Barton (2016). *Multi-Criteria Decision Analysis | Openness Project*. url: http://www.openness-project.eu/library/reference-book/sp-MCDA.

Schwaber, Ken e Jeff Sutherland (2016). *Guia do SCRUM - Um guia definitivo para o Scrum: As regras do jogo*. Vol. IV, p. 18. url: https://www.scrum.org/Portals/0/Documents/Scrum%20Guides/Scrum_Guide.pdf.

Scrum Alliance (s.d.). *Subway Map to Agile Practices | Agile Alliance*. url: https://www.agilealliance.org/agile101/subway-map-to-agile-practices/.

Scrum Sprint Retrospective Problems (s.d.). url: https://www.scrumexpert.com/knowledge/scrum-sprint-retrospective-problems/?fbclid=IwAR1lvVnyG9ZqCNQDHaPJJNmj1UIBc

SCrum.org (2010). *What is a Sprint Retrospective?* url: https://www.scrum.org/resources/what-is-a-sprint-retrospective.

Scrum.org (2019). *What is a Sprint Review?* url: https://www.scrum.org/resources/what-is-a-sprint-review.

– (2020). *What is Sprint Planning?* url: https://www.scrum.org/resources/what-is-sprint-planning.

Serrano, Nicolas, Josune Hernantes e Gorka Gallardo (2013). «Mobile web apps». Em: *IEEE Software* 30.5, pp. 22–27. issn: 07407459. doi: 10.1109/MS.2013.111.

Soewito, Benfano et al. (jan. de 2019). «Websocket to support real time smart home applications». Em: *Procedia Computer Science*. Vol. 157. Elsevier B.V., pp. 560–566. doi: 10.1016/j.procs.2019.09.014.

Software Development Life Cycle - UML Tutorial for Beginners (2013). url: https://www.startertutorials.com/uml/software-development-life-cycle.html.

Sugimori, Y. et al. (1977). «Toyota production system and kanban system materialization of just-in-time and respect-for-human system». Em: *International Journal of Production Research* 15.6, pp. 553–564. issn: 1366588X. doi: 10.1080/00207547708943149.

Swain, Nathan R. et al. (2016). «A new open source platform for lowering the barrier for environmental web app development». Em: *Environmental Modelling and Software* 85, pp. 11–26. issn: 13648152. doi: 10.1016/j.envsoft.2016.08.003.

Systems Development Life Cycle from FOLDOC (2000). url: http://foldoc.org/Systems+Development+Life+Cycle.

Team O'clock for your meetings: The team (s.d.). url: https://www.teamoclock.com/about-us.

Teles, Vinícius Manhães (2014). *Extreme Programming*. Ed. por Rubens Prates. 2ª ed. São Paulo: Novatec Editora Ltda. isbn: 978-85-7522-400-7.

Top 5 Extreme Programming (XP) Tools Every Team Should Use | Official Pythian® Blog (s.d.). url: https://blog.pythian.com/top-5-extreme-programming-xp-tools-every-team-should-use/.

Usdoj (2006). *DOJ Systems Development Life Cycle Guidance Chapter 1*. url: https://www.justice.gov/archive/jmd/irm/lifecycle/ch1.htm.

Version One (2019). «State Of Agile Survey». Em: *Annals of Physics* 54, p. 10. doi: 10.2777/17146. url: https://www.stateofagile.com/#ufh-i-521251909-13th-annual-state-of-agile-report/473508.

Vijini Mallawaarachchi (2017). «10 Common Software Architectural Patterns in a nutshell». Em: pp. 1–11. url: https://towardsdatascience.com/10-common-software-architectural-patterns-in-a-nutshell-a0b47a1e9013.

Wawryk, Maciej e Yen Ying Ng (set. de 2019). «Playing the sprint retrospective». Em: *Proceedings of the 2019 Federated Conference on Computer Science and Information Systems, FedCSIS 2019*. Institute of Electrical e Electronics Engineers Inc., pp. 871–874. isbn: 9788395541605. doi: 10.15439/2019F284.

Werewka, Jan e Anna Spiechowicz (nov. de 2017). «Enterprise architecture approach to SCRUM processes, sprint retrospective example». Em: *Proceedings of the 2017 Federated Conference on Computer Science and Information Systems, FedCSIS 2017*. Institute of Electrical e Electronics Engineers Inc., pp. 1221–1228. isbn: 9788394625375. doi: 10.15439/2017F96.

What is Web Application (Web Apps) and its Benefits (s.d.). url: https://searchsoftwarequality.techtarget.com/definition/Web-application-Web-app.

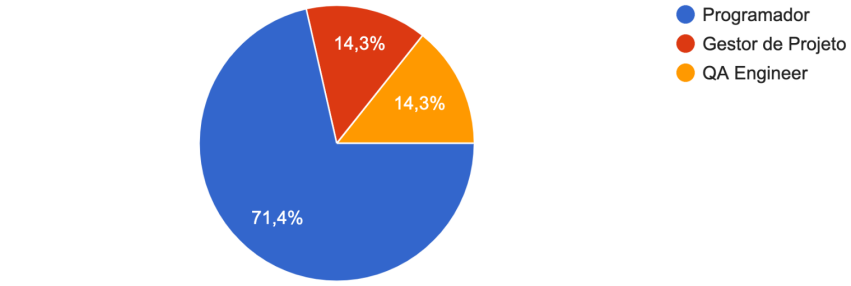
Apêndice A

Anexos

Questionários levados a cabo.

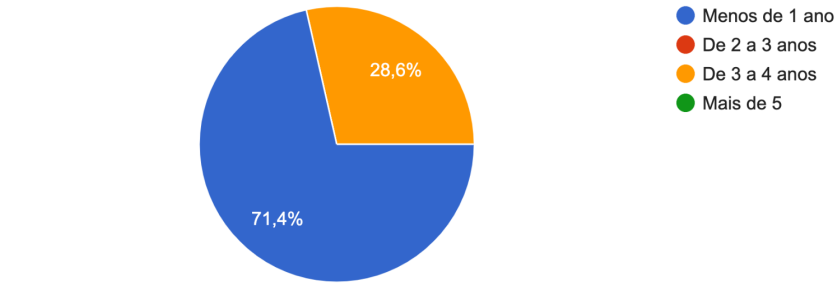
Qual o seu cargo

7 respostas



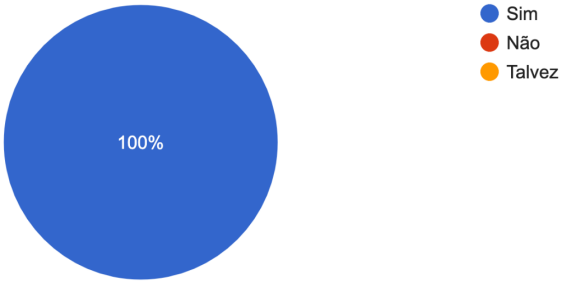
Há quanto tempo faz parte da Mindera?

7 respostas



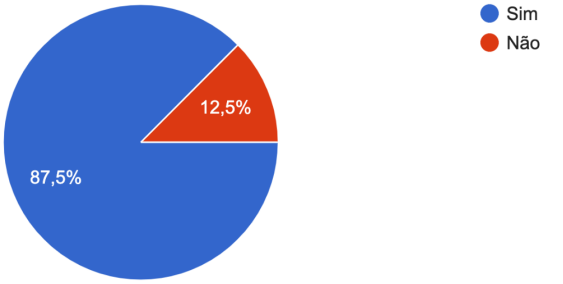
Sabe o que é uma sessão de Sprint Retrospective?

8 respostas



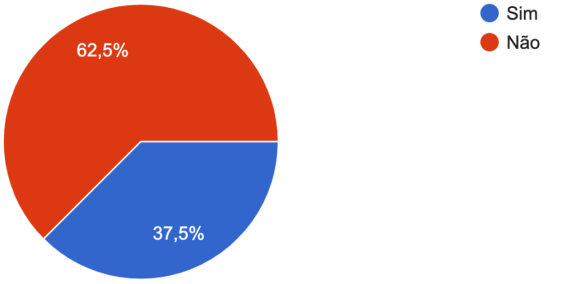
Costuma utilizar uma solução informática para auxiliar nas sessões de Sprint Retrospective?

8 respostas



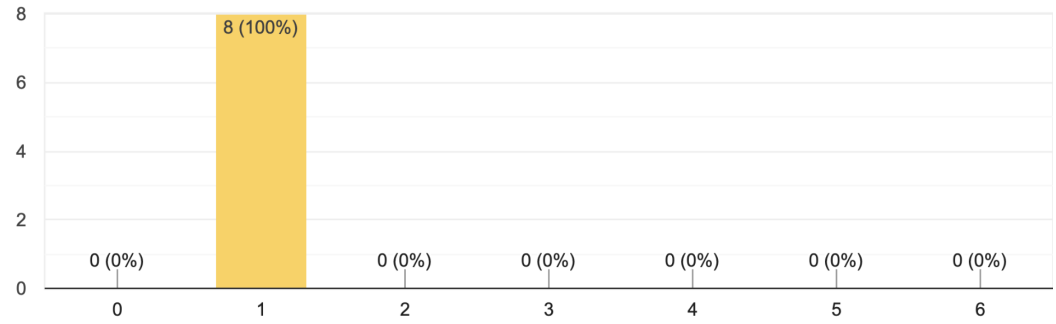
Sempre que termina um Sprint faz uma sessão de Sprint Retrospective?

8 respostas



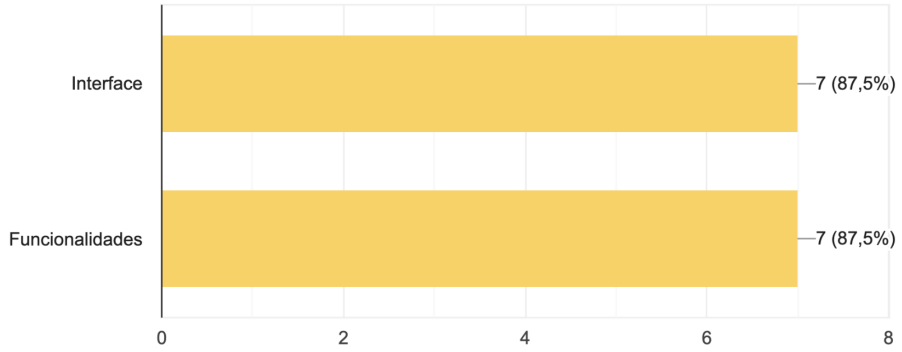
Quantas vezes participa numa sessão de Sprint Retrospective por semana?

8 respostas



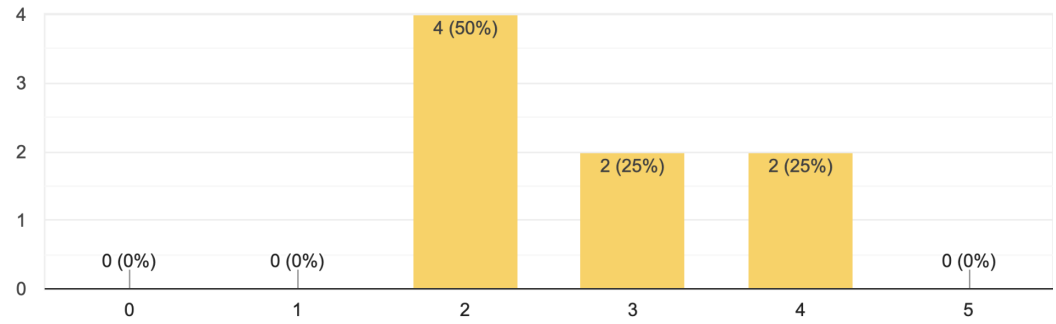
Numa aplicação o que é que dá mais valor na mesma.

8 respostas



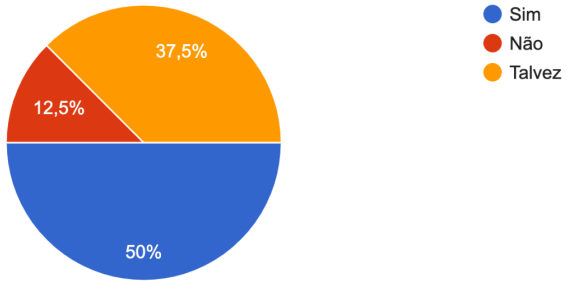
Qual o seu grau de satisfação, de forma geral, em relação à oferta de aplicações para a elaboração sessões de Sprint Retrospective.

8 respostas



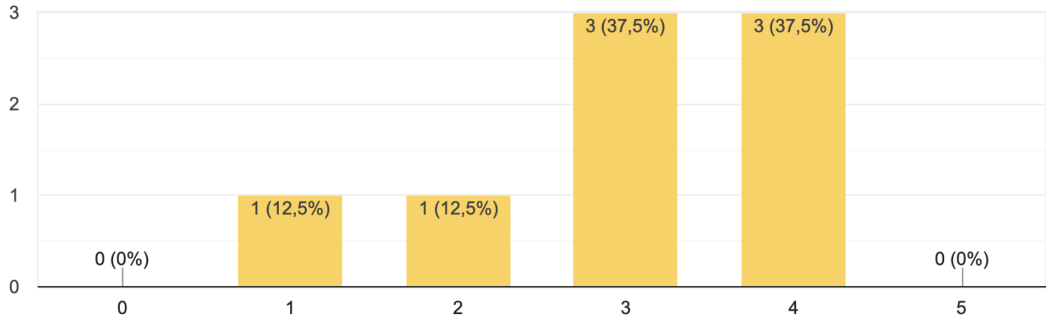
Acredita que as aplicações existentes para a elaboração de Retros afetam a motivação para efetuar uma sessão de Sprint Retrospective?

8 respostas



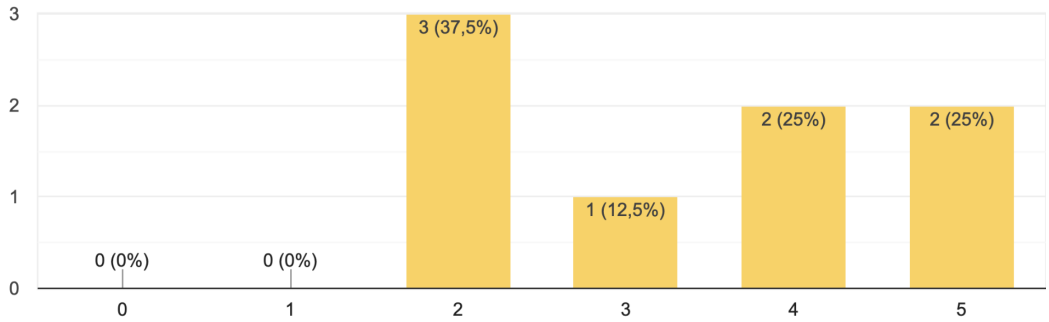
Em relação às aplicações já existentes como classifica as mesmas em relação à sua clareza de funcionamento.

8 respostas



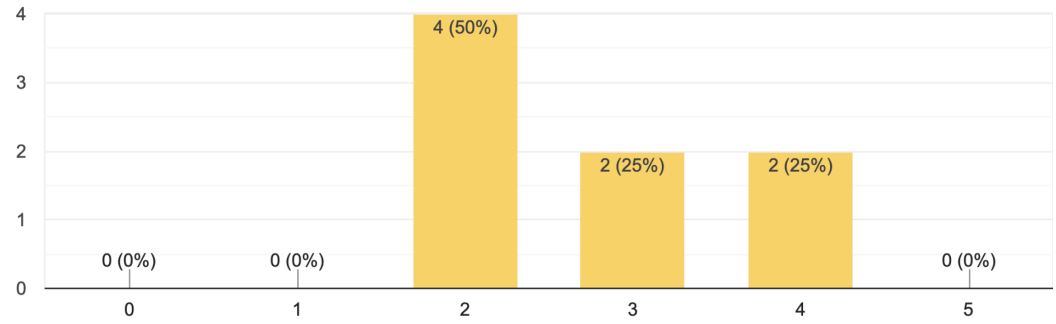
Como classifica as aplicações já existentes em relação à dificuldade de utilização.

8 respostas



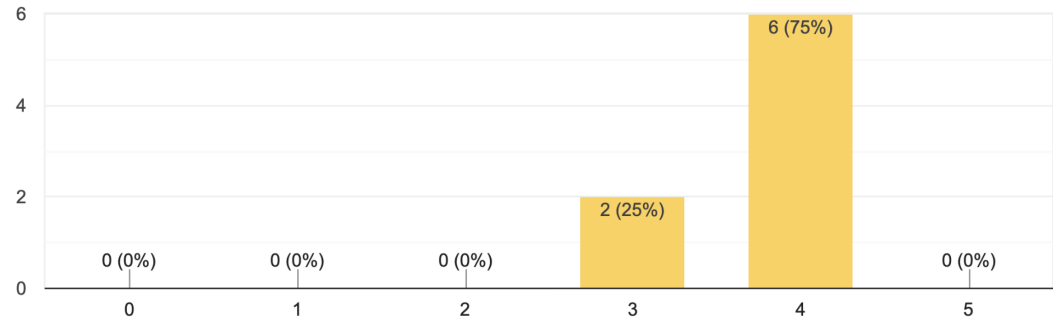
Como classifica as aplicações já existentes em relação ao seu desempenho.

8 respostas



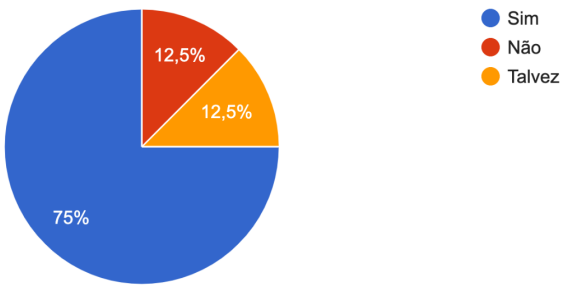
Qual o seu grau de satisfação, de forma geral em relação à aplicação Retrolution?

8 respostas



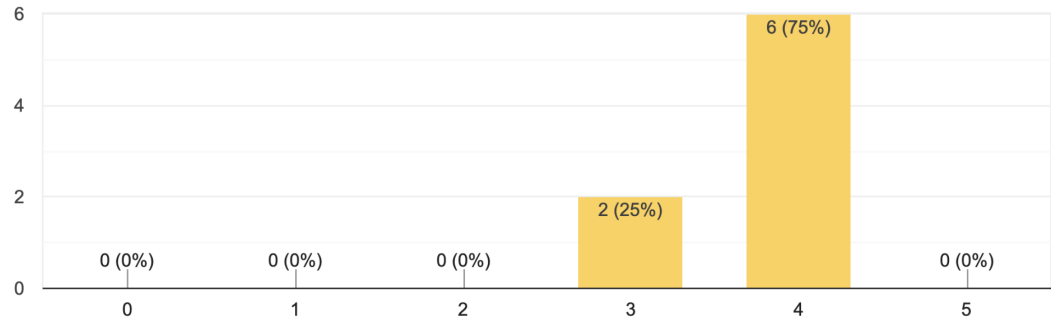
Acredita que a aplicação desenvolvida (Retrolution) contribuiu para um aumento da elaboração de sessões de Sprint Retrospective?

8 respostas



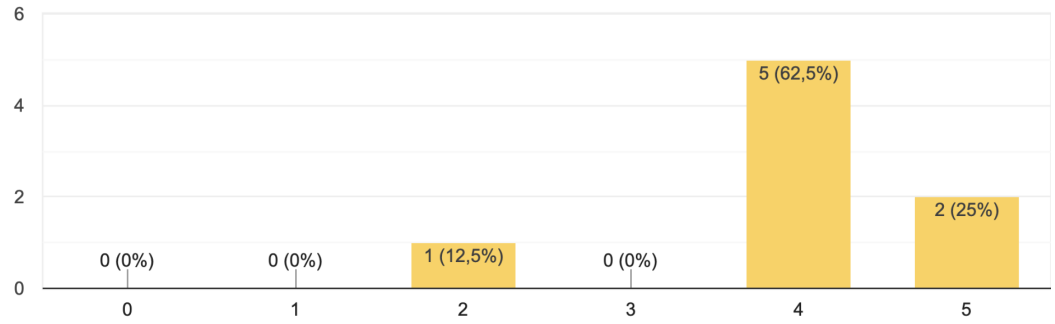
Como classifica Retrolution em relação à clareza de funcionamento?

8 respostas



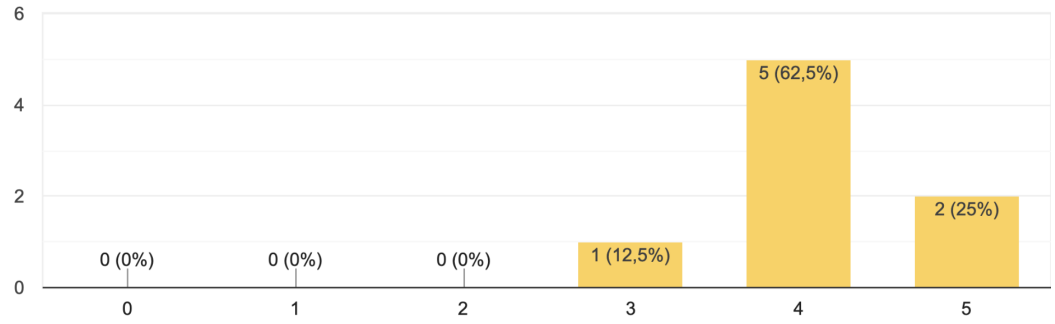
Como classifica Retrolution em relação à dificuldade de utilização.

8 respostas



Como classifica Retrolution em relação ao seu desempenho.

8 respostas



Como classifica Retrolution em relação ao seu aspecto gáfico.

8 respostas