INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




Plataforma Web para Disseminação de Informação de Suporte ao Desenvolvimento Ágil de Software

RÚBEN TIAGO NUNES AMORIM
Outubro de 2020

POLITÉCNICO
DO PORTO

Instituto Superior de Engenharia do Porto

Plataforma Web para Disseminação de Informação de Suporte ao Desenvolvimento Ágil de Software

Rúben Amorim

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

Orientador: António Jorge Santos Pereira

Júri:
Presidente:

Vogais:

Porto, 15 de Outubro de 2020

Dedicatória

Este trabalho é dedicado a todos que contribuíram para que este meu sonho se realizasse, cada um à sua maneira.

Resumo

Tradicionalmente, o processo de desenvolvimento de *software* era lento e não estava preparado para lidar com mudanças de requisitos do produto e com a agilidade inerente ao mercado. Para contornar este problema surgiram as metodologias ágeis, que se caraterizam pelo desenvolvimento iterativo e incremental, focando-se na melhoria contínua dos processos. No entanto, partilhar informações sobre artefatos, estar atento à visão geral do projeto e focar em objetivos específicos por um determinado período são desafios comuns para as equipas.

Assim sendo, a partilha de informações através de *dashboards* incentivam a comunicação e desempenham um papel fundamental nas equipas ágeis. Além disso, permitem que as equipas se mantenham atentas à visão geral de um projeto e se foquem em metas específicas.

Devido ao número de ferramentas usadas pelas equipas, por vezes é difícil visualizar a informação produzida por estas de modo a tirar partido das métricas fornecidas para monitorização e resolução de problemas no quotidiano.

Deste modo, o objetivo deste trabalho é o desenvolvimento de uma plataforma capaz de recolher e integrar informação de múltiplas fontes e apresentá-la de forma consistente e sistematizada através de *dashboards*. Além disso, foi estudada em que medida é que a utilização de *dashboards* contribui para agilizar o processo de desenvolvimento de *software*.

Com o propósito de validação, as equipas internas da organização onde o projeto foi inserido, como partes interessadas, conduziram a fase de testes e avaliação do projeto.

Palavras-chave: Desenvolvimento de Software, Metodologias Ágeis, Dashboards, Métricas, Melhoria Contínua

Abstract

Traditionally, the software development process was slow and wasn't prepared to deal with changes in product requirements and market agility. In this way, agile methodologies appeared, which are characterized by iterative and incremental development process and focus on continuous improvement. However, sharing information about artifacts, being aware of the project overview, and focusing on specific goals over a period of time are common challenges for teams.

Therefore, sharing information through dashboards encourages communication and plays a key role in agile teams. Also, they allow teams to remain attentive to a project's overall vision and focus on specific goals.

Due to the number of tools used by the teams, sometimes it's difficult to visualize the information produced by them to take advantage of the metrics provided for monitoring and troubleshooting.

In this way, the objective of this work is to develop a platform capable of collecting and integrating information from multiple sources and presenting it consistently and systematically through dashboards. In addition, it was studied how the use of dashboards contributes to the improvement of the software development process.

For validation purposes, the internal teams of the organization where the project was inserted, as stakeholders, conducted the testing and evaluation phase of the project.

Agradecimentos

Em primeiro lugar, quero agradecer ao Instituto Superior de Engenharia do Porto (ISEP) e aos seus docentes, pelo conhecimento transmitido no Mestrado em Engenharia Informática (MEI). Agradeço, em particular ao professor Jorge Santos pela prontidão e tempo despendido na orientação e acompanhamento do projeto, que contribuiu certamente para um trabalho de melhor qualidade.

À Mindera, agradeço pela oportunidade, pela experiência e pelas condições de trabalho proporcionadas, em especial ao Tiago Nunes pelos ensinamentos e por toda a disponibilidade manifestada.

Deixo também um agradecimento muito especial aos meus pais e à minha namorada pelo suporte, por me proporcionarem as condições ideais para traçar o meu percurso académico e tornarem os meus sonhos possíveis. Por fim, agradeço aos meus amigos e colegas que de uma forma ou de outra me ajudaram ao longo deste percurso.

Muito obrigado!

Conteúdo

Lista de Figuras **xiii**

Lista de Tabelas **xv**

Lista de Código **xvii**

Lista de Acrónimos **xix**

1 Introdução **1**
1.1 Contexto . . . 1
1.2 Problema . . . 2
1.3 Objetivos . . . 2
1.4 Planeamento do projeto . . . 3
1.5 Estrutura do documento . . . 4

2 Estado de arte **5**
2.1 Enquadramento teórico . . . 5
2.1.1 Metodologias Ágeis . . . 5
2.1.2 *Dashboards* . . . 7
2.2 Enquadramento tecnológico . . . 9
2.2.1 Node.js . . . 9
2.2.2 React . . . 10
2.2.3 GraphQL . . . 11
2.3 Análise de soluções existentes . . . 11
2.3.1 Mozaïk . . . 12
2.3.2 Geckoboard . . . 12
2.3.3 Smashing . . . 13
2.3.4 Tipboard . . . 14
2.3.5 Comparação das plataformas analisadas . . . 15

3 Análise de valor **19**
3.1 Identificação da oportunidade . . . 19
3.2 Análise da oportunidade . . . 19
3.3 Geração e enriquecimento de ideias . . . 20
3.4 Seleção de ideias . . . 20
3.4.1 Analytic Hierarchy Process (AHP) . . . 20
3.5 Definição do conceito . . . 22
3.6 Valor para o cliente . . . 23
3.7 Proposta de Valor . . . 23

4 Análise e Design **25**
4.1 Análise . . . 25

4.1.1 Requisitos funcionais 25
4.1.2 Requisitos não funcionais 26
4.1.3 Modelo de domínio 27
4.2 Arquitetura de software 28
4.2.1 Vista lógica 28
4.2.2 Vista de implantação 30
4.2.3 Alternativa arquitetural 31

5 Implementação **33**
5.1 *Frameworks* 33
5.2 Consultar *dashboard* 33
5.3 Adicionar *widget* ao dashboard 35
5.4 Criação de um *plugin* 36
5.5 Interface gráfica 39
5.5.1 Página inicial 39
5.5.2 Página de *dashboards* do utilizador 40
5.5.3 Página do *dashboard* 42
5.6 Testes 44
5.7 Auditorias 46

6 Avaliação **47**
6.1 Hipóteses 47
6.2 Metodologia 47
6.3 Análise de resultados 48
6.3.1 Utilização de *dashboards* no processo de desenvolvimento de *software* 48
6.3.2 Satisfação do utilizador relativamente à plataforma desenvolvida . . 51

7 Conclusão **55**
7.1 Objetivos alcançados 55
7.2 Limitações 55
7.3 Trabalho futuro 55
7.4 Apreciação final e pessoal 56

Bibliografia **59**

A Relatório de cobertura pelos testes **63**

B Relatórios da ferramenta Lighthouse **65**
B.1 Página inicial 65
B.2 Página de *dashboards* do utilizador 68
B.3 Página do *dashboard* 71

C Questionário **75**

D Questionário - Respostas **85**

Lista de Figuras

1.1 Quadro Kanban na ferramenta Trello 3

2.1 Exemplo de um espaço de trabalho informativo 7
2.2 Exemplo de um dashboard 9
2.3 Funcionamento do React Virtual DOM 10
2.4 Exemplo da User Interface (UI) da plataforma Mozaïk 12
2.5 Exemplo da UI da plataforma Geckoboard 13
2.6 Exemplo da UI da plataforma Smashing 14
2.7 Exemplo da UI da plataforma Tipboard 15

3.1 Árvore hierárquica de decisão 20

4.1 Diagrama de casos de uso 25
4.2 Modelo de domínio 28
4.3 Diagrama de componentes do sistema 29
4.4 Diagrama de componentes da aplicação servidora 29
4.5 Diagrama de componentes da aplicação cliente 30
4.6 Diagrama de implantação do sistema 31
4.7 Diagrama de componentes alternativo da aplicação servidora 31
4.8 Diagrama de componentes alternativo da aplicação cliente 32

5.1 Diagrama de sequência de consulta de um *dashboard* 35
5.2 Diagrama de sequência de adição de um *widget* ao *dashboard* 37
5.3 Exemplo do *widget* contador de *merge requests* 39
5.4 Exemplo de UI da página inicial 40
5.5 Exemplo de UI da página de *dashboards* do utilizador 40
5.6 Exemplo de UI da criação de um *dashboard* 41
5.7 Exemplo de UI do perfil do utilizador 41
5.8 Exemplo de UI da página do *dashboard* 42
5.9 Exemplo de UI da edição das configurações do *dashboard* 42
5.10 Exemplo de UI da configuração de um *widget* 43
5.11 Exemplo de UI da edição das configurações de um *widget* 43

6.1 Frequência relativa das funções dos inquiridos 48
6.2 Frequência relativa do uso de ferramentas de monitorização pessoal 49
6.3 Frequência relativa do uso de ferramentas de monitorização pelas equipas . 49
6.4 Frequência relativa das vantagens de utilização de *dashboards* 50
6.5 Frequência relativa das desvantagens de utilização de *dashboards* 50
6.6 Frequência absoluta da avaliação das funcionalidades da plataforma 51
6.7 Frequência absoluta da avaliação da usabilidade da plataforma 51
6.8 Frequência absoluta da avaliação do desempenho da plataforma 52
6.9 Frequência absoluta da avaliação global da plataforma 52

A.1 Relatório de cobertura pelos testes 63

Lista de Tabelas

2.1 Comparativo entre as soluções existentes 16

3.1 Comparação de alternativas e critérios 21
3.2 Pesos associados às alternativas de avaliação hierárquica 22
3.3 Benefícios e sacrifícios 23

4.1 Requisitos funcionais 26
4.2 Requisitos não funcionais segundo modelo FURPS+ 27

6.1 Escala de Likert usada para a avaliação 48
6.2 Valores médios de satisfação dos utilizadores 52

Lista de Código

2.1 Exemplo de uma schema de dados GraphQL 11
5.1 *Schema* do *dashboard* 33
5.2 *Callback* de validação das permissões de leitura do *dashboard* 34
5.3 GraphQL *query* do pedido dos detalhes de um *dashboard* 34
5.4 *Schema* do *widget* 36
5.5 GraphQL *mutation* do pedido de criação de um *widget* 36
5.6 Registo do *plugin* no contexto da aplicação 37
5.7 *Schema* do *plugin* 37
5.8 Pedido dos *merge requests* ao serviço do Gitlab 39
5.9 Exemplo de um teste de integração na aplicação servidora 44
5.10 Exemplo de um teste de integração na aplicação cliente 45

Lista de Acrónimos

AAA	Arrange Act Assert.
AHP	Analytic Hierarchy Process.
AM	Agile Modeling.
API	Application Programming Interface.
BI	Business Intelligence.
CD	Continuous Delivery.
CI	Continuous Integration.
CRUD	Create, Read, Update and Delete.
DOM	Document Object Model.
DRY	Don't Repeat Yourself.
DSL	Domain-Specific Language.
DTO	Data Transfer Object.
FFE	Fuzzy Front End.
HTTP	Hypertext Transfer Protocol.
HTTPS	Hypertext Transfer Protocol Secure.

IoC	Inversion of Control.
ISEP	Instituto Superior de Engenharia do Porto.
MEI	Mestrado em Engenharia Informática.
NCD	New Concept Development.
NPM	Node.js Package Manager.
PWA	Progressive Web App.
QA	Quality Assurance.
RC	Razão de Consistência.
REST	Representational State Transfer.
RGPD	Regulamento Geral de Proteção de Dados Pessoais.
SEO	Search Engine Optimization.
UI	User Interface.
UML	Unified Modeling Language.
XP	Extreme Programming.

Capítulo 1

Introdução

Neste capítulo, é feita uma contextualização do tema deste projeto, mencionado qual o problema, bem como os objetivos e o planeamento efetuado.

As equipas de desenvolvimento de *software* da Mindera utilizam diversas ferramentas para auxiliar os seus projetos. Apesar destas ferramentas operarem bem, existe uma dificuldade na leitura de toda a informação produzida por elas, de modo a facilitar a monitorização e a resolução dos problemas do quotidiano das equipas.

Assim sendo, surge a necessidade da criação de uma ferramenta que integre e mostre toda a informação produzida através de *dashboards* e também estudar de que forma esta ferramenta permite agilizar o processo de desenvolvimento de *software*.

Além disso, para este projeto foi adotada a metodologia de trabalho ágil designada Kanban.

1.1 Contexto

A Mindera é uma empresa de desenvolvimento de *software* fundada em Setembro de 2014 e atualmente conta com mais de quatro centenas de colaboradores envolvidos em projetos localizados nos vários países em que atua. Os produtos que desenvolve destinam-se principalmente às áreas de: *e-commerce*, *fast-food*, produtos financeiros e jogos de apostas [1].

No desenvolvimento de *software* são utilizadas metodologias ágeis que se caraterizam principalmente por fornecer valor incremental através de cadências de trabalho iterativas. Estas metodologias permitem que as equipas lidem com a imprevisibilidade inerente ao desenvolvimento de *software*, mantendo um fluxo constante de artefatos expectavelmente de alta qualidade [2].

Uma caraterística das metodologias ágeis de desenvolvimento de *software*, como o Scrum, é o aumento da compreensão e da interação da equipa devido a um ciclo de desenvolvimento de *software* reduzido e bastante iterativo. No entanto, partilhar informações sobre artefatos, estar atento à visão geral do projeto e focar em objetivos específicos por um período específico são desafios comuns para as equipas. Assim, espaços de trabalho informativos, ao incentivar a comunicação e partilha de informações úteis, desempenham um papel essencial para as equipas ágeis [3].

Os *dashboards* exibem vários elementos organizacionais importantes num formato consolidado usando várias ferramentas de visualização (gráficos, tabelas, pictogramas, entre outros) e ajudam as equipas a acompanhar o estado atual de um projeto, transmitindo informações de maneira não disruptiva. Além disso, são exibidas resumidamente uma série de métricas

críticas permitindo que a organização as use para a medição da produtividade e melhoria organizacional. O objetivo principal é fornecer um acompanhamento do projeto de forma acessível e de fácil leitura [4].

1.2 Problema

As equipas de desenvolvimento de *software* usam muitas ferramentas diferentes para rastrear os seus projetos, executar compilações, gerir o lançamento de novas versões, sistemas de controlo de versões, revisões de código, monitorização de sistemas, cobertura de código pelos testes, entre outras métricas de qualidade. Embora essas ferramentas funcionem bem, produzem resultados úteis e algumas podem ser integradas, é difícil visualizar a informação produzida por estas de modo a tirar partido das métricas produzidas para monitorização e resolução de problemas no quotidiano.

As integrações providenciadas pelas ferramentas utilizadas tendem a funcionar bem mas são limitadas relativamente às funcionalidades das plataformas. Isto é, quando uma plataforma seleciona os dados produzidos por outra plataforma para o seu consumo, tira partido apenas do que lhe é mais conveniente, descartando os restantes dados obtidos pela ferramenta, não facilitando a visualização da informação útil. Por exemplo, a plataforma de gestão de projetos integrada com o sistema de controlo de versões apenas usa dados do *commit* ou *branch* atribuído a uma determinada tarefa, descartando assim outras informações como o autor, estado da *pipeline*, entre outros elementos considerados relevantes para as equipas analisarem.

Deste modo, a informação produzida pelas ferramentas permite às equipas saber como os projetos estão a evoluir, acompanhar o processo de desenvolvimento, identificar causas de um eventual problema e possíveis melhorias.

1.3 Objetivos

O objetivo deste projeto é o desenvolvimento de uma ferramenta capaz de recolher e integrar informação de múltiplas fontes e apresentá-la de forma consistente e sistematizada.

Os *dashboards* permitem visualizar um conjunto de informações de uma maneira centralizada, ajudando no processo de monitorização e tomada de decisão. Além disso, a sua disposição em forma de painel permite a análise, comparação e o acompanhamento dos indicadores de forma direta e imediata [4].

Assim sendo, formulou-se uma hipótese para testar em que medida a utilização de *dashboards* integradores de informação contribuem para agilizar o processo de desenvolvimento de *software*.

Deste modo, há lugar ao desenvolvimento de uma plataforma para solucionar o problema descrito anteriormente, capaz de concentrar as métricas das ferramentas essenciais para as equipas num painel que se mantém atualizado e permite a visualização e acesso fácil da informação a todos os membros.

A plataforma a desenvolver permite que cada equipa crie, personalize e partilhe *dashboards* de acordo com suas necessidades com informações que considere relevantes, tais como: estados de *pipelines* de Continuous Integration (CI)/Continuous Delivery (CD), estados de revisão do código, estados da *sprint*, estados dos ambientes operacionais, entre outros.

O produto resultante deste projeto é totalmente direcionado para a organização, não havendo planos de comercialização do mesmo. Desta forma, o cliente do produto é a própria empresa, em que os principais *stakeholders* deste projeto são os seus colaboradores.

1.4 Planeamento do projeto

O projeto adotou uma metodologia de trabalho ágil denominada Kanban. Esta metodologia popular é implementada no desenvolvimento ágil de *software* e requer comunicação em tempo real da capacidade e total transparência do trabalho [5]. Os itens de trabalho são representados visualmente num quadro Kanban, permitindo que os membros da equipa vejam o estado de cada peça de trabalho a qualquer momento.

Na figura 1.1 é possível visualizar o quadro de trabalho utilizado com recurso à ferramenta Trello.

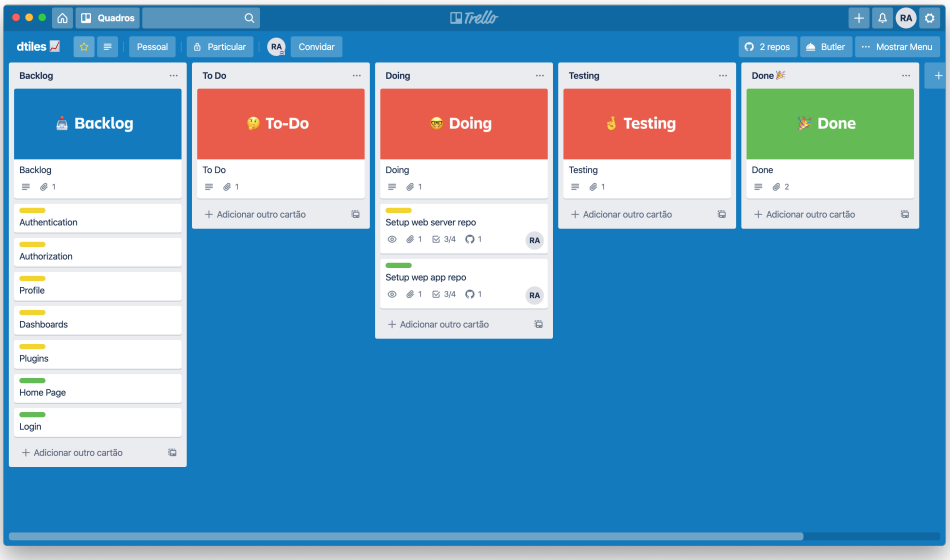


Figura 1.1: Quadro Kanban na ferramenta Trello

A metodologia Kanban oferece várias vantagens adicionais ao planeamento e processamento das tarefas para equipas de todas as dimensões, tais como [6]:

- A implementação possui alguns princípios e regras que facilitam o cumprimento;
- A aplicação no desenvolvimento de *software* significa a adoção de práticas e princípios no processo existente de melhoria;
- A visualização é a chave para identificar obstáculos e uma maneira de monitorizar o estado atual do processo de desenvolvimento;
- O uso da metodologia incentiva a equipa a ser cada vez melhor;
- Os resultados finais do desenvolvimento aumentam a confiança e a satisfação do cliente;
- Melhorias na qualidade do produto, despesas e tempo de entrega reduzidos.

Também foi utilizada a plataforma Github como sistema de controlo de versões Git adequado para o desenvolvimento do projeto. Para além disso, foram adotadas metodologias de CI/CD, através da ferramenta Github Actions, para facilitar as entregas e automatizar as diversas etapas do desenvolvimento de *software* garantindo a qualidade.

1.5 Estrutura do documento

Nesta secção será apresentada a estrutura desta tese, fornecendo uma visão geral do documento e dos aspetos abordados.

No primeiro capítulo é descrito o âmbito em que o projeto está envolvido no contexto da organização Mindera. Além disso, é descrito o problema, objetivos, abordagem e planeamento do projeto.

No segundo capítulo é realizado um enquadramento teórico e tecnológico, que permite uma melhor compreensão dos conceitos-chave e é também realizada uma análise do mercado referente às soluções existentes que resolvem problemas semelhantes.

No terceiro capítulo é descrita a análise de valor do projeto, com o intuito de demonstrar o seu valor de mercado para as partes interessadas.

O quarto capítulo é direcionado para o *design* da solução, apresentando os requisitos funcionais e não funcionais, assim como a proposta arquitetural a ser implementada recorrendo a diagramas em notação Unified Modeling Language (UML).

No quinto capítulo expõe-se o processo de implementação da solução proposta anteriormente, descrevendo as boas práticas de engenharia de *software* adotadas e os detalhes de implementação.

O sexto capítulo descreve a maneira como foram abordados e executados os testes à solução e apresenta os resultados dos inquéritos de satisfação realizados.

No último capítulo é feita uma síntese ao trabalho realizado onde se descreve os objetivos atingidos, conclusões, limitações encontradas e trabalho futuro.

Capítulo 2

Estado de arte

2.1 Enquadramento teórico

Neste capítulo são contextualizadas as metodologias de trabalho ágeis, a sua história e as suas caraterísticas. Além disso, são abordados os espaços de trabalho informativos, que pretendem aumentar a interação das equipas e a perceção do estado do trabalho através de ferramentas como os *dashboards*.

Para finalizar, são abordados os *dashboards* como ferramenta de monitorização, apontando os seus atributos e os tipos, para que são usados e que tipo de informação podem difundir.

2.1.1 Metodologias Ágeis

Com o passar dos anos, as empresas de *software* têm sofrido alterações relativamente à maneira como se organizam. Hoje em dia, as equipas são mais autónomas na definição dos seus objetivos e acompanhamento do seu progresso [7].

Tradicionalmente, para o desenvolvimento de *software*, estabelecer o *roadmap* de um projeto, executá-lo e controlar o seu desenvolvimento, era da responsabilidade do líder da equipa. As equipas apenas tinham de seguir o planeamento e pouco tempo lhes restava para fazer alterações [7].

No final do século XX, o desenvolvimento dos produtos de *software* era um processo lento, tornando-se um problema para as empresas [7]. As metodologias de trabalho existentes não conseguiam lidar com a alteração e a evolução dos requisitos do produto que estava em desenvolvimento e deste modo, não era possível acompanhar a flexibilidade do mercado. Surge assim um caminho para novas formas de trabalho, tal como as metodologias ágeis. As empresas viram uma possibilidade de mudança fundamental para o trabalho das suas equipas. O foco passou a ser a flexibilidade e a adaptabilidade, com equipas auto-organizadas e multifuncionais [7].

Anteriormente as equipas tinham de cumprir o *roadmap* definido pelo responsável, hoje em dia com as metodologias ágeis, são as próprias equipas que o definem e acompanham, desde a análise dos requisitos até ao lançamento do produto para o mercado. As equipas multifuncionais tiveram de desenvolver conhecimentos que no passado não eram considerados importantes [7].

O desenvolvimento ágil de *software* é uma alternativa iterativa à gestão sequencial tradicional, tal como os métodos em cascata [8]. Em oposição aos modelos tradicionais, as metodologias ágeis propõem ciclos de desenvolvimento curtos, com entregas bem definidas e foco na melhoria contínua dos processos. Com isso, passou a ser mais simples identificar

erros e falhas durante a execução do projeto e as pessoas envolvidas ganharam mais flexibilidade para efetuar alterações e evitar que determinados problemas afetassem o seu resultado final [9].

Apesar de muitas das metodologias serem anteriores a isso, em 2001 um grupo composto por 17 pessoas reuniu-se para debater sobre essas novas abordagens na gestão de projetos e criou o chamado Manifesto Ágil, que de certa forma, oficializa a existência das metodologias e estabelece princípios que as caraterizam [2].

No fundo, o desenvolvimento ágil segue um modelo incremental, que fomenta a colaboração entre a equipa, o planeamento contínuo, mas também a contínua evolução e aprendizagem. As metodologias ágeis devem respeitar o ciclo de desenvolvimento de *software* – planeamento, execução e entrega final – permitindo que o *software* seja desenvolvido por etapas, tornando mais fácil alterações ao planeamento [10].

A principal vantagem da utilização destas metodologias reside na constante entrega de valor ao cliente, uma vez que as entregas são incrementais, e no facto da entrega do produto de *software* ser mais rápida.

Existem inúmeras metodologias que seguem os princípios ágeis tais como: Scrum, Extreme Programming (XP), Agile Modeling (AM), Kanban, Lean Development, Crystal, entre outras [11]. Estudos recentes sobre o estado do desenvolvimento ágil indicam que pelo menos 72% dos inquiridos usam Scrum ou um híbrido que usa o mesmo como base [12].

Scrum é uma das muitas *frameworks* da metodologia ágil e é sem dúvida a mais utilizada. Carateriza-se pelos ciclos de desenvolvimento, definidas como *sprints*, e pela maximização do tempo de desenvolvimento de um produto de *software*. É tipicamente utilizado na gestão de projetos de desenvolvimento de produtos de *software*, mas também pode ser utilizado noutro contexto de negócio [13].

Uma caraterística das metodologias ágeis é o aumento da perceção do estado do trabalho e na interação da equipa devido a um ciclo de desenvolvimento de *software* reduzido e bastante iterativo. No entanto, partilhar informações sobre artefactos, manter-se atento à visão geral de um projeto e focar em metas específicas por um período específico são desafios comuns para equipas. Assim, os espaços de trabalho informativos, como exemplificado na figura 2.1, ao incentivar a comunicação e partilha de informações úteis, desempenham um papel essencial para as equipas ágeis [3].

Os espaços de trabalho informativos ajudam as equipas a acompanhar o estado atual de um projeto, transmitindo informações de maneira não disruptiva. O objetivo mais importante é fornecer um melhor acompanhamento do projeto de forma acessível e fácil de ler. A exibição contínua de conteúdo relevante nos espaços de trabalho é benéfica, pois fornece informações básicas de interesse comum. Além disso, as equipas de *software* usam os seus espaços de trabalho para comunicar internamente, o que por vezes implica uma compreensão mútua. A comunicação informal é um fator considerável nesse aspeto pois é crucial para a troca de informações nas equipas [14].

Dentro dos espaços de trabalho informativos das equipas ágeis, várias técnicas e ferramentas para a visualização de *software*, por exemplo, esboços e *dashboards*, são frequentemente aplicadas. A visualização de *software* é bastante relevante para as equipas ágeis, pois melhora a qualidade do mesmo aumentando a perceção sobre os artefactos de *software* [8].

[1] Obtido de: https://www.geckoboard.com/best-practice/tv-dashboards

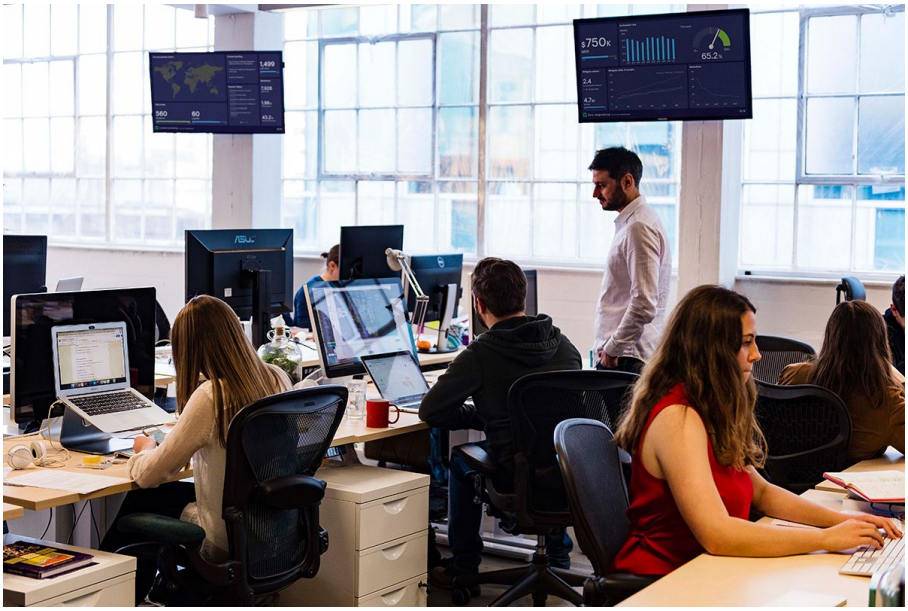


Figura 2.1: Exemplo de um espaço de trabalho informativo[1]

2.1.2 Dashboards

Os *dashboards* são ferramentas de comunicação e perceção cognitiva projetadas para ajudar as pessoas a identificar visualmente tendências, padrões e anomalias, raciocinar sobre o que vêem e ajudá-las a tomar decisões eficazes. O objetivo é transformar os dados brutos contidos nos repositórios de uma organização em informações de consumo [15].

Estas ferramentas podem ser usadas para várias finalidades e o seu *design*, tecnologia e âmbito diferem com base nos seguintes casos de uso [16]:

- Radiadores de informações - planeados para divulgar as informações de estados para grandes públicos, geralmente projetados em ecrãs e colocadas em locais centrais de projetos, equipas ou grupos;
- *Dashboards* de gestão - projetados para fornecer informações aos gerentes sobre o estado do projeto e os seus parâmetros subjacentes, com a possibilidade de detalhar os dados;
- *Dashboards* de Business Intelligence (BI) - pensados para apoiar os gerentes de produto no acesso, visualização e análise dos dados relacionados com o desenvolvimento de produtos e o mercado envolvente;
- *Dashboards* híbridos - combinam dois ou três dos cenários anteriores.

A necessidade de *dashboards* para melhoria do desempenho e melhoria organizacional é reconhecida pelas organizações [4]. A sua necessidade progride já que traz as informações de forma didática aos membros das equipas, garantindo que todos estão conscientes sobre o objetivo a ser atingido. Além disso, a sua adoção expõe as seguintes vantagens [17]:

- Colabora para o processo de decisão;
- Promove a integração das equipas de trabalho;
- Diminui os riscos que a empresa corre;

- Aumenta a visualização de oportunidades;
- Melhora a gestão de tempo dos funcionários;
- Ajuda a realocar recursos;
- Define e divulga a estratégia;
- Fomenta uma cultura com foco nos resultados.

Na engenharia de *software*, os *dashboards* são usados para fornecer informações e métricas sobre o produto em desenvolvimento, bem como exibir referências ou apoiar a análise do processo de desenvolvimento. Normalmente, são projetados com objetivos específicos em mente, e muitos desses objetivos relacionam-se direta ou implicitamente com algum aspeto da produtividade, incluindo a qualidade do produto, a velocidade do trabalho ou a satisfação das partes interessadas [15].

Um *dashboard* eficaz deve conter as seguintes informações [18]:

- Estado atual do progresso do projeto;
- Velocidade do trabalho realizado;
- Estado dos testes;
- Eficácia da estimativa do esforço.

Os *dashboards* exibem vários elementos organizacionais importantes num formato consolidado usando várias ferramentas de visualização, *widgets*, gráficos, tabelas e pictogramas. Permitem que as organizações visualizem os dados de forma estruturada, integrada e organizada. Os painéis resumem uma variedade de métricas de desempenho e permitem que as organizações visualizem as principais métricas críticas para medição de desempenho e melhoria organizacional [4].

A figura 2.2 mostra o exemplo de um *dashboard* com as caraterísticas descritas anteriormente. Neste caso, é apresentando um exemplo de monitorização de um sistema em que são salientados aspetos positivos e negativos, atribuindo cores relacionadas para melhor compreensão e atenção.

Geralmente, são usados televisores de dimensão considerável para exibir *dashboards* de modo a que as equipas possam ter uma visão rápida de como os *sprints* estão a progredir em projetos ágeis [15].

As equipas ágeis podem aproveitar os dados que são produzidos pelo processo de desenvolvimento de *software* para monitorização do mesmo. Algumas das métricas que as equipas ágeis costumam monitorizar são: velocidade, número de histórias prontas, pontos por membros de equipa, limites de trabalho em curso e tempo do ciclo. Embora estes sejam bons exemplos, não são restritos, dependendo do processo e do que a equipa gostaria de melhorar, podem criar outras métricas que ajudem o processo de melhoria contínua [19].

Além disso, a monitorização de métricas pode ajudar na produção de um produto de alta qualidade com uma infraestrutura de desenvolvimento e uma base de código saudáveis, tais como [20]:

- Estado da *pipeline* de CI;

[2]Obtido de: http://zomko.facach.org/it-dashboard

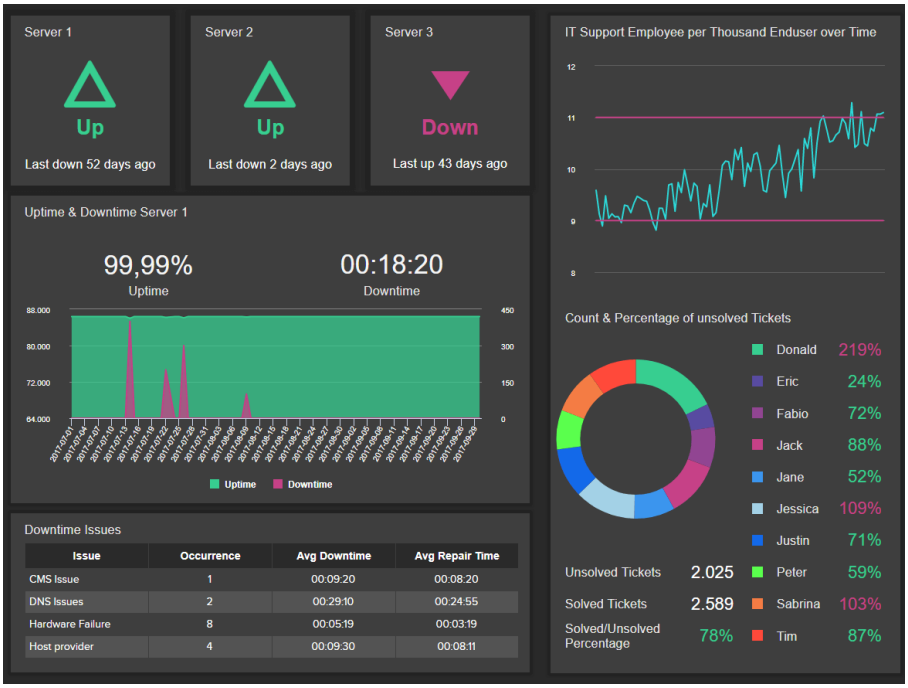


Figura 2.2: Exemplo de um dashboard[2]

- Cobertura e estados dos testes (automáticos, unitários, entre outros);
- Contagens de implantações em produção e os incidentes associados;
- Número de problemas encontrados pelas ferramentas de análise estática;
- Número de histórias de dívida técnica no *backlog*.

2.2 Enquadramento tecnológico

Neste capítulo são descritas as tecnologias usadas para o desenvolvimento da solução: Node.js, React e GraphQL. Deste modo, é apresentada uma breve descrição de cada tecnologia, bem como as suas caraterísticas para que se entendam alguns dos conceitos tratados nos próximos capítulos.

2.2.1 Node.js

Node.js é um ambiente de execução JavaScript assíncrono orientado a eventos, implementado com base no interpretador V8, interpretador de JavaScript em C++ *open source* da Google, utilizado no Chrome [21].

Apesar de ser uma tecnologia nova, as bibliotecas que estão na sua base são consideradas estáveis. No entanto, muitos módulos que se encontram no Node.js Package Manager (NPM) são de baixa qualidade e não estão devidamente documentados pois são desenvolvidas pela comunidade e não passam por qualquer processo de validação [22].

Devido ao uso da linguagem JavaScript, a sua curva de aprendizagem é considerada baixa, o que significa que programadores com experiência em *frontend* podem começar a programar

para o *backend* sem muitas dificuldades. O facto de ser leve e multiplataforma torna-o uma boa opção para arquiteturas de microsserviços [22].

Assim sendo, devido à sua natureza, Node.js é mais utilizado na realização dos seguintes tipos de projetos:

- Application Programming Interface (API);
- Aplicações *web* em tempo real como servidores para *chats* ou aplicações colaborativas entre múltiplos utilizadores;
- Jogos multijogador;
- Aplicações que requerem alta escalabilidade;
- Servidores de *streaming* de dados.

2.2.2 React

React é uma biblioteca JavaScript declarativa, eficiente e flexível para criar uma User Interface (UI) desenvolvida pelo Facebook e lançada em 2013 como uma ferramenta *open source* [23].

A maneira como o React funciona para criar interfaces é através do desdobramento de toda a estrutura da aplicação em pequenas peças reutilizáveis chamadas de componentes de UI.

Uma vantagem clara desta ferramenta é o modo como trabalha com o Document Object Model (DOM) e atualiza os componentes de acordo com seus estados. O que o React propõe é a criação do seu próprio DOM, mais eficiente e no qual os componentes atuam, o Virtual DOM. Assim sendo, sempre que um componente é renderizado, o React atualiza o Virtual DOM e depois compara-o com uma imagem do DOM feita antes da atualização. Deste modo, é possível identificar o que realmente mudou e atualizar somente os nós que mudaram de estado, providenciando um enorme ganho de *performance* neste aspeto [24].

A figura 2.3 mostra o funcionamento anteriormente descrito.

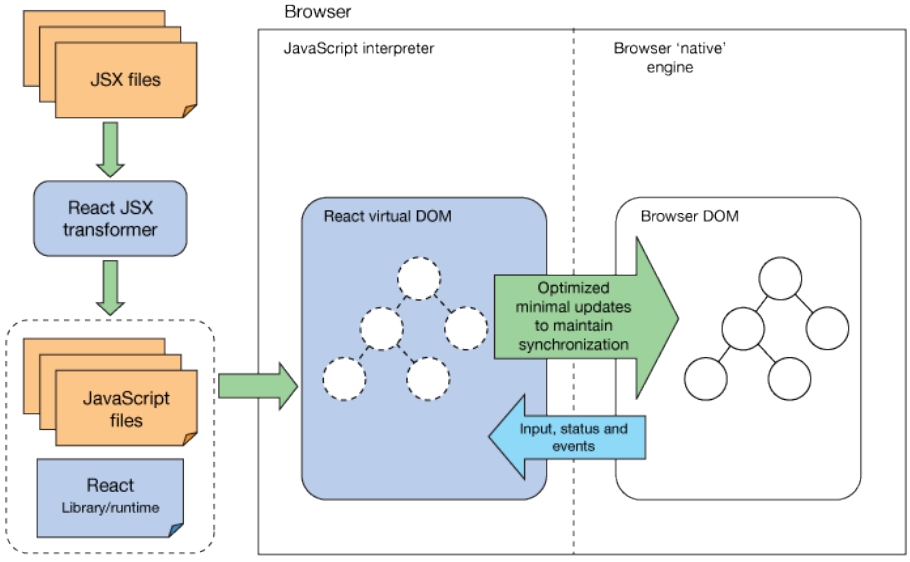


Figura 2.3: Funcionamento do React Virtual DOM[3]

2.2.3 GraphQL

GraphQL é uma especificação *open source* de uma linguagem de consulta para APIs criada pelo Facebook. A linguagem fornece uma descrição do modelo de dados, oferecendo aos clientes a capacidade de pedirem apenas os dados que precisam num só pedido, ao contrário de uma arquitetura Representational State Transfer (REST). É utilizada para consultar serviços que possuem recursos e dados definidos de acordo com o sistema de tipos que acompanha a linguagem, onde para cada recurso existe um tipo. Um conjunto de tipos formam um *schema*, como exemplificado no excerto de código 2.1 [25].

```
type Book {
    title: String
    author: Author
}

type Author {
    name: String
    books: [Book]
}
```

Código 2.1: Exemplo de uma schema de dados GraphQL

Com GraphQL é possível evitar o chamado *over-fetching* e o *under-fetching*, ou seja, impedir o consumo de dados que na realidade não são necessários e a necessidade de várias pedidos para se obter a informação necessária [26].

Por outro lado, a sua curva de aprendizagem é um pouco acentuada devido aos seus conceitos complexos. Além disso, a existência de apenas um recurso faz com que seja muito difícil limitar o acesso a determinadas entidades, assim como efetuar *cache* dos dados pois cada pedido pode ser diferente, mesmo que efetuado sobre a mesma entidade [27].

2.3 Análise de soluções existentes

Nesta secção é apresentada uma análise das soluções existentes no mercado no que diz respeito a plataformas que permitem a integração de informação através de *dashboards*. A análise baseou-se na recolha de informação de quatro plataformas de acordo com os seguintes critérios:

- Apenas plataformas gratuitas ou que permitam demonstração;
- Resultados de pesquisa mais populares nos motores de busca;
- Popularidade e recomendação pela comunidade ou em *blogs* [28];
- Diversidade de funcionalidades enunciadas.

Por fim, será feita uma descrição pormenorizada de diversos aspetos das diferentes plataformas e apresentada uma comparação das mesmas.

[3]Obtido de: https://www.ibm.com/developerworks/library/wa-react-intro

2.3.1 Mozaïk

Mozaïk é uma plataforma *open source* baseada em Node.js que permite a criação de *dashboards* usando configurações relativamente simples em JavaScript [29].

Conta com um basto leque de *widgets* criados para integrar com o Github, Travis, Google Analytics, Twitter, AWS e ElasticSearch, entre muitos outros. Para além disso, é possível a qualquer pessoa com conhecimento das tecnologias base criar as suas próprias integrações.

As suas funcionalidades são as seguintes:

- *Layout* responsivo - suporta dispositivos com ecrãs de diversas dimensões através do seu *layout* adaptável de maneira a ser visualizado numa televisão ou no telemóvel;
- Suporte de temas - permite a alteração do tema para um dos diversos incluídos ou para outro qualquer personalizado;
- Extensível por módulos - permite a integração de *widgets* desenvolvidos em separado em módulos Node.js;
- Posicionamento em grelha - fornece uma maneira simples de definir o *layout* do painel usando um sistema de grelha;
- Comunicação com o servidor otimizada - delega as chamadas à API das integrações para o servidor e envia os dados para os *widgets* através de *websockets*;
- Múltiplos *dashboards* - suporta a criação de vários *dashboards* e transita suavemente entre eles.

A figura 2.4 mostra o exemplo de um *dashboard* criado na plataforma Mozaik para monitorização de um repositório de controlo de versões.

Figura 2.4: Exemplo da UI da plataforma Mozaïk [29]

2.3.2 Geckoboard

Geckoboard é uma plataforma na *cloud* de *dashboards* para televisões, rápida e fácil de configurar. Foi criada para equipas de suporte ao cliente, vendas, marketing, produtos e *e-commerce*, com o objetivo de obter uma visão em tempo real das principais métricas e

do progresso das metas e assim otimizar o trabalho para atingir essas metas e reagir aos problemas mais rapidamente [30].

A plataforma pode ser configurada em minutos pois conecta-se diretamente às ferramentas, apenas é preciso escolher as métricas e a maneira como serão visualizadas sem necessidade de desenvolvimento. Permite a integração com Salesforce, Zendesk, Spreadsheets, Google Analytics, Hubspot, Intercom, Mailchimp e Mixpanel, entre muitas outras ferramentas.

Além disso, também é possível a partilha de *dashboards* através de *links* temporários, criar contas para os colegas de trabalho de maneira a que possam criar os seu próprios *dashboards*, assim como a visualização dos mesmos através de dispositivos com ecrãs de menores dimensões.

Na figura 2.5 é apresentado um exemplo de um *dashboard* criado para monitorização de uma aplicação *web*.

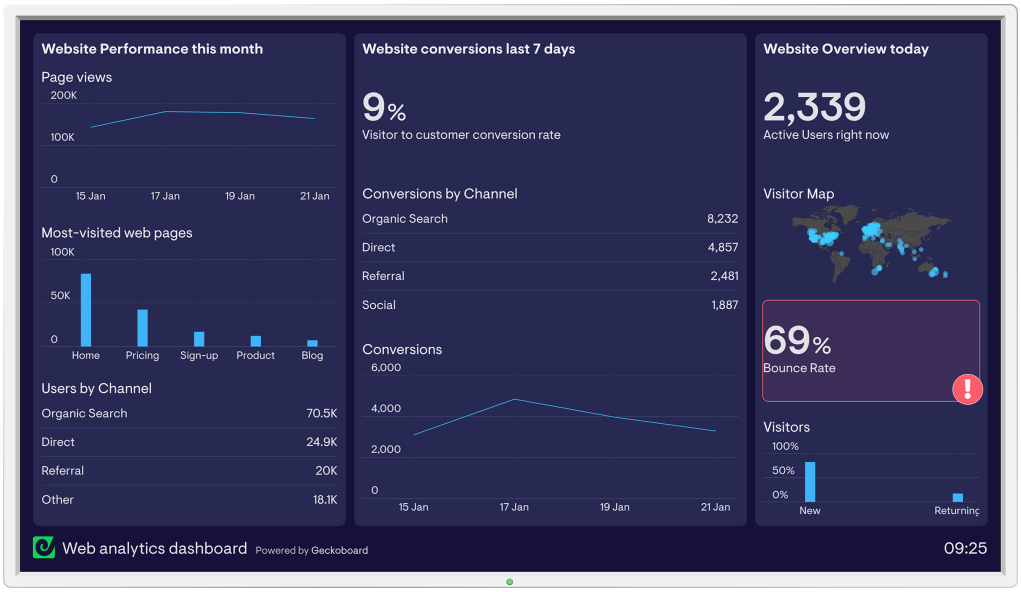


Figura 2.5: Exemplo da UI da plataforma Geckoboard [30]

2.3.3 Smashing

Smashing é uma plataforma *open source* baseada na *framework* Sinatra que teve origem num projeto chamado Dashing, atualmente descontinuado, criado pela empresa Shopify com o objetivo de criar *dashboards* personalizados para serem visualizados nas televisões dos seus escritórios [31].

Esta plataforma é totalmente personalizável e conta com uma enorme diversidade de integrações através de *widgets* criados pela comunidade. Além disso, de maneira a manter a informação segura, é possível adicionar autenticação para acesso aos *dashboards*.

As funcionalidades chave são as seguintes:

- Permite o uso de *widgets* desenvolvidos pela comunidade ou a criação própria com HTML, CSS e CoffeeScript;
- Os *widgets* aproveitam o poder das ligações de acordo como o princípio Don't Repeat Yourself (DRY), prevenindo conexões desnecessárias;

- Providencia uma API para receber os dados a mostrar ou permite fazer o pedido dos dados através de uma Domain-Specific Language (DSL) em Ruby;
- Interface que permite a organização dos *widgets* por arrasto;
- Alojamento na *cloud* através da plataforma Heroku de forma rápida.

A figura 2.6 mostra o exemplo de um *dashboard* na plataforma Smashing.

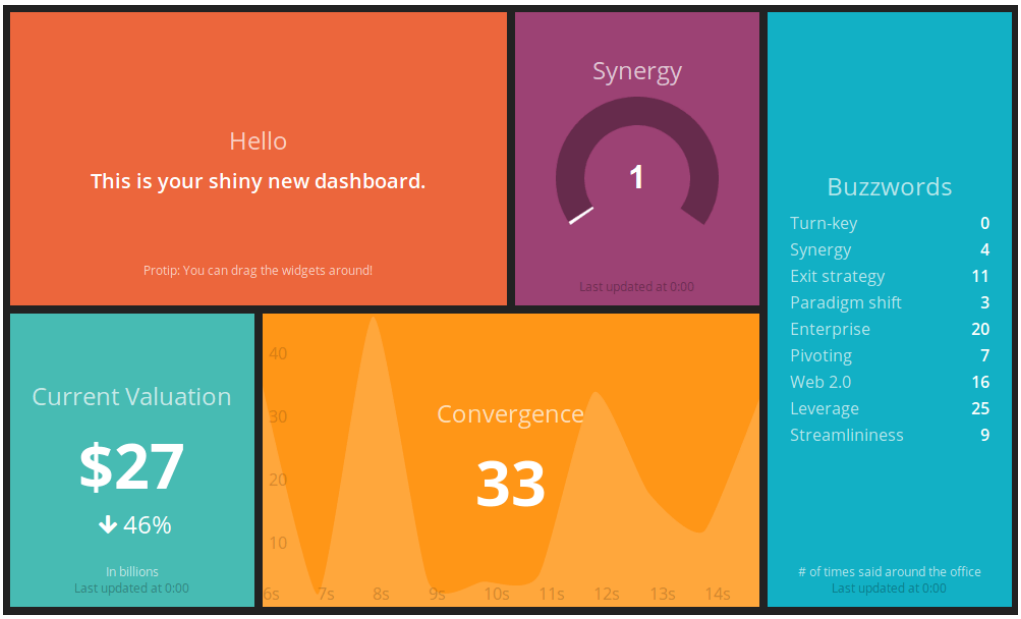


Figura 2.6: Exemplo da UI da plataforma Smashing [31]

2.3.4 Tipboard

Tipboard é uma plataforma para criação de *dashboards*, escrita em JavaScript e Python. Os seus *widgets* são completamente abstraídos das fontes de dados, o que fornece grande flexibilidade e um nível relativamente alto de possíveis personalizações. Devido ao seu objetivo, de exibir vários dados e estatísticas nos escritórios, é otimizado para ecrãs de maiores dimensões [32].

O projeto assenta nas seguintes premissas:

- Definir o *layout* do *dashboard* através de um sistema de grelha;
- Separação clara entre *widgets* e fontes de dados;
- Capacidade de criar *widgets* e *scripts* próprios através de fontes de dados (por exemplo, Jira, Bamboo, Confluence, entre outras);
- Alimentar os dados dos widgets através de uma API REST.

Na figura 2.7 é apresentado um exemplo de um *dashboard* criado na plataforma Tipboard para monitorização de um serviço *web*.

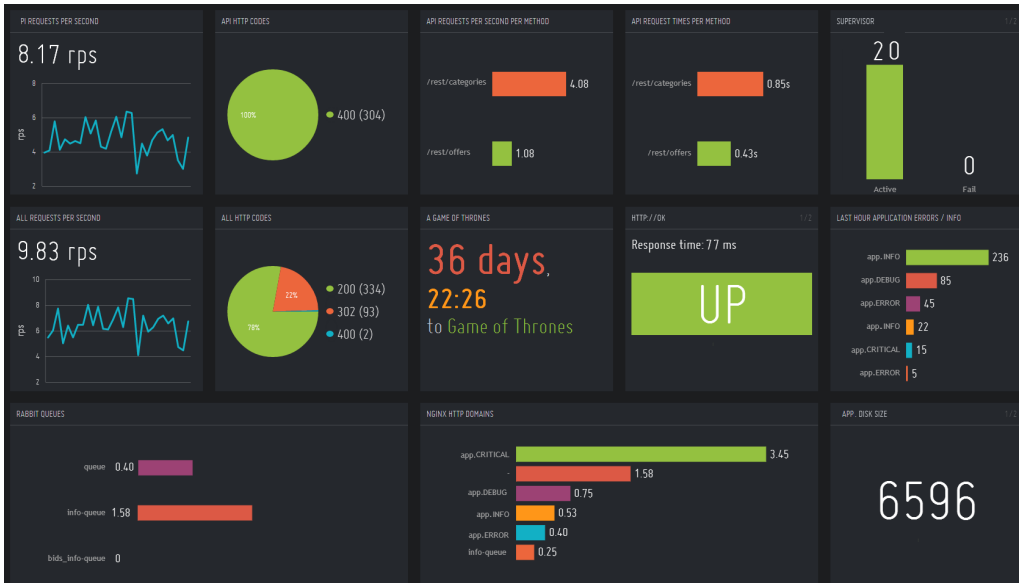


Figura 2.7: Exemplo da UI da plataforma Tipboard [32]

2.3.5 Comparação das plataformas analisadas

Com o objetivo de analisar detalhadamente e realizar a comparação, foi necessário instalar algumas das plataformas num servidor para se poder experimentar as mesmas, e assim descrever para cada uma delas, uma apresentação geral, funcionalidades disponibilizadas e pontos positivos ou inovadores.

A partir da análise foram levantados os seguintes critérios que diferenciam as plataformas entre si:

- Autenticação;
- Extensibilidade;
- Implantação;
- Licença;
- Partilha;
- Personalização;
- Vitalidade.

Na tabela 2.1 é possível visualizar o comparativo entre as várias plataformas que foram analisadas nas secções anteriores e é possível retirar as seguintes conclusões:

- Nenhuma das soluções cobre todos os critérios selecionados;
- Como critério de segurança, algumas das plataformas não possuem autenticação;
- As plataformas *open source* tiram partido da comunidade para a sua extensibilidade e, por isso, possuem mais integrações, e caso não exista, facilmente é criada;
- A maior parte das soluções necessitam de alojamento;

Tabela 2.1: Comparativo entre as soluções existentes

Plataforma	**Mozaïk**	**Geckoboard**	**Smashing**	**Tipboard**
Autenticação	Não suporta	Credenciais da plataforma na *cloud*	Basic auth, restrição de IP, Google ou Github	Não suporta
Extensibilidade	Uso de *widgets* criados pela comunidade e permite criação própria	Limitado às integrações existentes	Uso de *widgets* criados pela comunidade e permite criação própria	Uso de *widgets* criados pela comunidade e permite criação própria
Implantação	Necessita de alojamento	*Cloud*	Necessita de alojamento	Necessita de alojamento
Licença	*Open source*, MIT	Comercial	*Open source*, MIT	*Open source*, Apache 2.0
Partilha	Não suporta	Através de *links* temporários	Não suporta	Não suporta
Personalização	Possui 6 temas nativos e permite criação própria	Predefinido	Predefinido pelos *widgets*	Predefinido pelos *widgets*
Vitalidade	Última atividade no repositório a 28/05/2019	Constante atualização, última a 30/01/2020	Última atividade no repositório a 28/08/2019	Última atividade no repositório a 22/07/2019

- Apenas a plataforma na *cloud* suporta a partilha de *dashboards*, através de *links* temporários;
- As plataformas gratuitas têm um nível de personalização elevado;
- A maioria das soluções já não recebe atualizações, o que se supõem que, pelo lado positivo, possam estar estáveis ou então, pelo lado negativo, terem sido abandonadas.

A segurança das informações é um requisito indispensável nestas plataformas. Geckoboard e Smashing implementam ambos mecanismos de autenticação para limitar o acesso aos dados apenas a utilizadores autorizados, levando assim vantagem neste aspeto. No entanto, a funcionalidade de partilha através de *links* temporários pode ser considerado um ponto menos positivo devido a uma possível fuga de informação.

No que diz respeito à escalabilidade, a possibilidade de criar integrações é um fator muito positivo para as plataformas. Deste modo, é possível lidar com o aparecimento de novas ferramentas, assim como suportar a infinidade existente no mercado.

Relativamente à implantação, apenas a plataforma Geckoboard está alojada na *cloud*, sendo uma mais valia para evitar custos de servidores, manutenção e pessoas qualificadas para configuração dos mesmos. Por outro lado, esta também é a única solução paga, todas as restantes são gratuitas.

Além disso, no âmbito do problema descrito no capítulo 1.2, verifica-se que as ferramentas existentes no mercado não satisfazem a totalidade dos requisitos para a resolução do mesmo. Isto deve-se à existência de limitações a nível da escalabilidade das plataformas e da necessidade de partilha de *dashboards*, pois a única plataforma que suporta partilha tem integrações limitadas.

Em síntese, este processo de análise às plataformas revelou-se indispensável para o desenvolvimento e decisões futuras, pois foi possível identificar aspetos e comportamentos imprescindíveis quer a nível funcional, como de usabilidade.

Capítulo 3

Análise de valor

Neste capítulo é apresentada uma análise de valor do projeto com o intuito de provar o seu valor de mercado para a organização Mindera. A análise inclui os cinco elementos Fuzzy Front End (FFE) do modelo New Concept Development (NCD), diferentes perspetivas de valor e a proposta de valor.

Além disso, é demonstrado através do método Analytic Hierarchy Process (AHP) que o desenvolvimento da solução é mais indicado do que a adoção de uma alternativa existente no mercado.

3.1 Identificação da oportunidade

Este projeto tem o objetivo de auxiliar a Mindera numa melhor gestão e acompanhamento do processo de desenvolvimento de *software* através de uma ferramenta de produtividade para os colaboradores. Tal como já foi mencionado nas secções 1.1 e 1.2, as equipas utilizam diversas ferramentas que produzem inúmeros dados e é difícil agrupar essa informação, observar um quadro geral e tirar partida disso para a resolução de problemas do dia-a-dia.

Outra oportunidade passa pela análise da concorrência, descrita no capítulo 2.3, com vista a identificar lacunas e efetuar melhorias.

3.2 Análise da oportunidade

Deste modo, como analisado na secção 2.3, existem múltiplas plataformas, contudo, apresentam algumas lacunas e nenhuma preenche totalmente os requisitos impostos pela organização. A falta de funcionalidades e limitações das plataformas existentes no mercado, está na origem da oportunidade para a realização deste projeto. Além disso, algumas das plataformas são *open source* e encontram-se sem atividade recente, o que se supõem que o seu desenvolvimento possa ter sido abandonado por questões de financiamento, disponibilidade dos criadores, entre outros.

Depois da análise efetuada, concluiu-se que desenvolvimento de uma plataforma que agrega um pouco do melhor de cada opção concorrente, com eventuais melhorias, pode ser uma mais valia para a organização, tratando-se de uma oportunidade a explorar.

3.3 Geração e enriquecimento de ideias

A ideia do projeto proposto pela organização teve origem numa discussão entre uma equipa na adoção de uma ferramenta que permitisse concentrar as métricas produzidas pelas ferramentas usadas. Para isso, seria necessário possuir um painel de visualização e acesso fácil da informação aos membros da equipa e com isso ajudar o processo de desenvolvimento de *software*.

Assim sendo, ou seria utilizada uma ferramenta existente no mercado ou seria necessário o desenvolvimento interno de acordo com os requisitos enunciados. O desenvolvimento interno poderia ser efetuado no contexto de um estágio curricular, para minimizar os custos.

3.4 Seleção de ideias

De acordo com as ideias identificadas previamente, foi feita a análise das soluções existentes no mercado e não foi possível encontrar uma que satisfaz todos os critérios e funcionalidades requeridas. Além disso, através do método hierárquico AHP, prova-se que a alternativa de desenvolvimento interno é a mais indicada de acordo com os critérios selecionados.

3.4.1 Analytic Hierarchy Process (AHP)

Para selecionar uma das ideias identificadas anteriormente, foi usado o método AHP e construiu-se a árvore hierárquica de decisão de acordo com a figura 3.1.

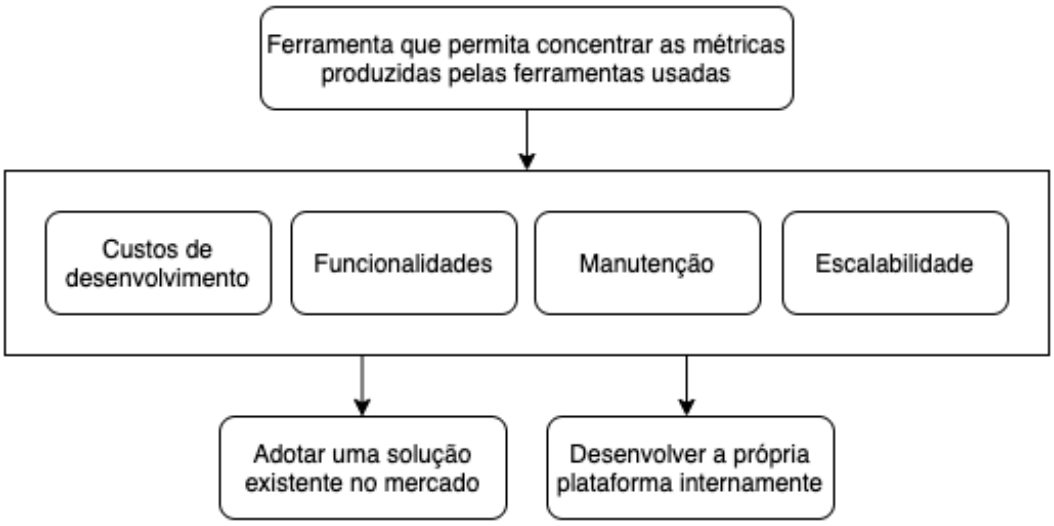


Figura 3.1: Árvore hierárquica de decisão

Os critérios para avaliar qual a melhor ideia para chegar à solução são:

- **Custos de desenvolvimento** - custos de tempo e dinheiro para desenvolver a solução;
- **Funcionalidades** - se a solução cumpre com os requisitos enunciados;
- **Manutenção** - facilidade em manter a aplicação usável;
- **Escalabilidade** - de que forma é possível adicionar novos recursos à solução.

Com recurso a estes critérios é possível avaliar qual é a melhor ideia para atingir o objetivo principal. A tabela 3.1 inclui a atribuição de pesos para cada critério segundo a escala de Satty.

Tabela 3.1: Comparação de alternativas e critérios

Critérios	**Custos de desenvolvimento**	**Funcionalidades**	**Manutenção**	**Escalabilidade**
Custos de desenvolvimento	1	1/4	2	1/3
Funcionalidades	4	1	3	2
Manutenção	1/2	1/3	1	1/3
Escalabilidade	3	1/2	3	1
Total	17/2	25/12	9	8/3

As funcionalidades e a escalabilidade são os dois critérios que mais importância tem na seleção da ideia, uma vez que afetam o caminho que a solução poderá tomar. A manutenção e os custos de desenvolvimento são também critérios que ajudam na seleção, no entanto, não se revelaram tão importantes.

A matriz seguinte, refere-se à matriz normalizada do método de avaliação AHP.

$$M = \begin{bmatrix} \frac{1}{17} & \frac{3}{15} & \frac{2}{9} & \frac{1}{8} \\ \frac{8}{17} & \frac{12}{25} & \frac{1}{3} & \frac{3}{4} \\ \frac{1}{17} & \frac{4}{25} & \frac{1}{9} & \frac{1}{8} \\ \frac{6}{17} & \frac{6}{25} & \frac{1}{3} & \frac{3}{8} \end{bmatrix}$$

Através da matriz anterior, obtém-se o vetor de prioridades que define a ordem de importância de cada critério. A partir dos resultados obtidos, o critério Funcionalidades aparece em primeiro lugar, seguido de Escalabilidade, Custos de desenvolvimento e Manutenção.

$$V = \begin{bmatrix} 0.146 \\ 0.508 \\ 0.114 \\ 0.325 \end{bmatrix}$$

De seguida, avalia-se a consistência das prioridades relativas:

$$\lambda_{max} = 4.134$$

$$IC = 0.045$$

$$RC = \frac{0.448}{0.90} = 0.049$$

Como o valor de Razão de Consistência (RC) < 0.1, podemos concluir que os valores das prioridades relativas estão consistentes.

$$M = \begin{bmatrix} \frac{2}{3} & \frac{1}{5} & \frac{2}{3} & \frac{1}{4} \\ \frac{1}{3} & \frac{4}{5} & \frac{1}{3} & \frac{3}{4} \end{bmatrix}$$

Após a construção da matriz anterior de comparação paritária para cada critério, considerando cada uma das alternativas selecionadas, obteve-se a prioridade composta para as alternativas, conforme a tabela 3.2.

Tabela 3.2: Pesos associados às alternativas de avaliação hierárquica

Alternativas	**Pesos**
Solução existente	0.356
Desenvolvimento da solução	0.737

Em suma, após análise à tabela 3.2, a alternativa de desenvolvimento da plataforma internamente aparece como a mais indicada para a resolução do problema, em função dos critérios definidos e das suas respectivas importâncias.

3.5 Definição do conceito

O objetivo deste projeto é o desenvolvimento de uma ferramenta capaz de recolher e integrar informação de múltiplas fontes e apresentá-la de forma consistente e sistematizada através de *dashboards*, permitindo uma melhor visualização do estado do trabalho e monitorização do desenvolvimento dos projetos na Mindera.

Por conseguinte, o produto resultante é totalmente direcionado para o uso interno da organização, não havendo planos de comercialização do mesmo.

3.6 Valor para o cliente

Para definir o valor de um produto ou serviço, primeiro é necessário identificar os clientes. A Mindera é a organização onde o projeto será desenvolvido e é a principal parte interessada. O objetivo é melhorar o processo de desenvolvimento de *software* para as equipas. Deste modo, é possível aumentar a confiança na qualidade dos seus produtos e serviços.

Benefícios	**Sacrifícios**
• Qualidade dos produtos e serviços; • Resolução de problemas; • Comunicação; • Personalização; • Monitorização; • Suporte; • Facilidade de utilização e adoção.	• Configuração; • Custos de instalação; • Custos de manutenção.

Tabela 3.3: Benefícios e sacrifícios

3.7 Proposta de Valor

A proposta de valor é uma afirmação com o objetivo de levar ao cliente uma ideia clara, concisa e transparente de como determinado negócio pode ser relevante para ele. Posto isto, foi definida a seguinte proposta de valor:

Uma ferramenta colaborativa com uma basta gama de integrações capaz de recolher informação e apresentá-la de forma consistente e sistematizada através de dashboards, permitindo economizar tempo na monitorização do desenvolvimento dos projetos nas organizações.

Capítulo 4

Análise e Design

4.1 Análise

Este capítulo tem como objetivo apresentar o processo no qual foram recolhidos e estruturados os requisitos, funcionais e não funcionais, inerentes ao projeto desenvolvido.

4.1.1 Requisitos funcionais

Os requisitos funcionais resultam do processo de elicitação dos requisitos, onde foram analisadas as necessidades das equipas de acordo com o problema levantado e os objetivos formulados (cf. secções 1.2 e 1.3 respetivamente), e descrevem as funcionalidades da plataforma.

A figura 4.1, apresenta o diagrama de casos de uso com uma visão dos atores e a sua relação com os casos de uso no sistema. Através do diagrama, é possível identificar dois atores que acedem às funcionalidades da plataforma, o utilizar não registado/autenticado, que apenas se pode autenticar e visualizar *dashboards* públicos, e o utilizador registado/autenticado que pode visualizar o seu perfil, consultar *dashboards* e fazer a gestão daqueles que lhe pertencem.

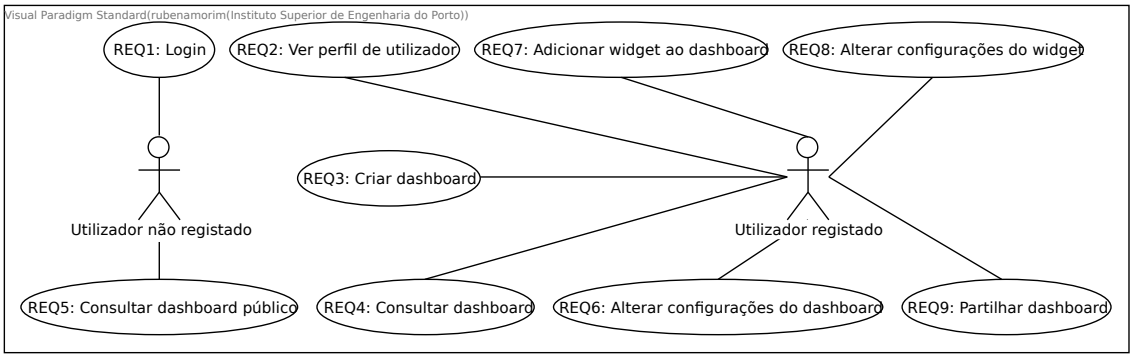


Figura 4.1: Diagrama de casos de uso

Na tabela 4.1, é apresentada uma descrição pormenorizada dos casos de uso levantados, onde são abordadas as condições para a realização do caso de uso e o seu fluxo de execução.

Requisito	Ator	Descrição
REQ1: Login	Utilizador não registado	Um utilizador pode autenticar-se na plataforma através da sua conta Google e para isso é necessário dar permissões de acesso às suas informações, caso seja a primeira vez ou as mesmas sejam revogadas.
REQ2: Ver perfil de utilizador	Utilizador registado	O utilizador pode verificar as suas informações pessoais acedendo ao seu perfil de utilizador.
REQ3: Criar dashboard	Utilizador registado	O utilizador pode efetuar a criação de um *dashboard*, definindo o nome, a sua visibilidade (público ou privado), tempo de transição entre páginas e o tempo de atualização.
REQ4: Consultar dashboard	Utilizador registado	O utilizador pode consultar os *dashboards* que tem acesso, ou seja, os que foram criados por si ou partilhados consigo.
REQ5: Consultar dashboard público	Utilizador não registado	O utilizador pode visualizar um *dashboard* que esteja marcado como público através de um *link* único.
REQ6: Alterar configurações do dashboard	Utilizador registado	O utilizador proprietário pode alterar as configurações de um *dashboard*. Deste modo, é possível alterar o nome, visibilidade e as definições das suas integrações.
REQ7: Adicionar widget ao dashboard	Utilizador registado	O utilizador proprietário pode configurar a integração de um *widget* num *dashboard*. Para isso, é necessário selecionar um dos *plugins* existentes, a sua variante de *widget*, referir a sua posição na grelha e as respetivas configurações do serviço.
REQ8: Alterar configurações do widget	Utilizador registado	O utilizador proprietário pode alterar as configurações de um *widget*. Deste modo, é possível alterar o seu posicionamento no *dashboard* e as suas configurações específicas.
REQ9: Partilhar dashboard	Utilizador registado	O utilizador proprietário de um *dashboard* pode partilhá-lo com outros utilizadores da plataforma. Para isso, é necessário especificar qual o utilizador, através do *email*.

Tabela 4.1: Requisitos funcionais

4.1.2 Requisitos não funcionais

Os requisitos não funcionais da plataforma estão classificados segundo o modelo FURPS+ e são apresentados na tabela 4.2.

Classificação	Requisito
Funcionalidade	• O sistema deve adotar sistemas básicos de segurança, como autenticação e autorização; • A autenticação deve fazer uso dos serviços da Google apenas; • As comunicações entre os serviços devem ser feitas usando o protocolo Hypertext Transfer Protocol Secure (HTTPS); • Os dados pessoais devem ser tratados tendo em consideração a legislação atual - Regulamento Geral de Proteção de Dados Pessoais (RGPD); • Seguir uma abordagem a *plugins* para as integrações.
Usabilidade	• A plataforma deve ter uma interface responsiva de modo a suportar o acesso através de dispositivos com ecrãs de diversas dimensões; • A plataforma deve ter uma interface acessível para pessoas incapacitadas - índice de acessibilidade superior a 90% através do Google Lighthouse.
Confiabilidade	• A plataforma deve ser resiliente para lidar com erros e tratá-los corretamente - testes com cobertura superior a 75%.
Desempenho	• O sistema deve ser rápido e eficiente, no acesso e funcionamento - índice de *performance* superior a 90% através do Google Lighthouse.
Suporte	• A linguagem da plataforma deve ser Inglês; • O código fonte deve ser fácil de manter, seguindo boas práticas de engenharia de *software* - índice de boas práticas superior a 90% através do Google Lighthouse; • Suporte dos *browsers* mais comuns: Chrome, Firefox, Edge e Safari.
+	• Tecnologias a adotar: Node.js, React e GraphQL.

Tabela 4.2: Requisitos não funcionais segundo modelo FURPS+

4.1.3 Modelo de domínio

O modelo de domínio resulta da análise de requisitos e captura os principais conceitos de negócio e relações entre si. Com o objetivo de compreender melhor o problema que se pretende solucionar, foi

elaborado o modelo de domínio apresentando na figura 4.2.

O *dashboard* é o conceito principal do domínio, tem um utilizador proprietário e pode ter vários utilizadores convidados, em que ambos são utilizadores da plataforma. Além disso, os *dashboards* contém ainda várias grelhas que são compostas por vários *widgets*. O *widget* no *dashboard* corresponde a um *plugin* que disponibiliza a configuração de vários *widgets* possivelmente integrados com um determinado serviço.

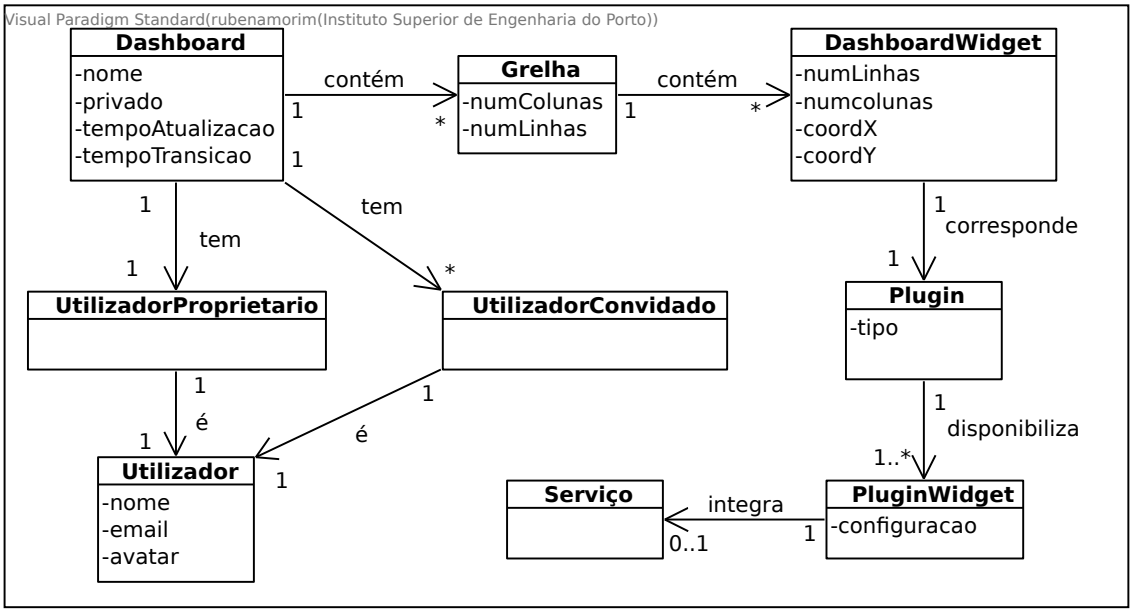


Figura 4.2: Modelo de domínio

4.2 Arquitetura de software

Após a análise da solução a desenvolver, foi realizado o estudo arquitetural. Nos capítulos seguintes são identificados todos os componentes do sistema e as suas conexões.

Por fim, são apresentadas e discutidas alternativas arquiteturais para a solução.

4.2.1 Vista lógica

A vista lógica tem o objetivo de fornecer uma base para a compreensão da estrutura e organização do *design* do sistema, ilustrar os principais componentes e as suas relações [33]. De seguida, encontram-se ilustradas as arquiteturas através do diagrama de componentes de alto nível, bem como a sua explicação.

A figura 4.3 define os seguintes componentes do sistema:

- **dtitles Web App** - aplicação cliente que corre num *web browser* e comunica com a aplicação servidora;
- **dtitles Web Service** - componente que contém a lógica de negócio da aplicação, é responsável por comunicar com a base de dados e disponibilizar os recursos;
- **dtitles DB** - componente responsável pela persistência dos dados.

Na figura 4.4 estão expostos os seguintes componentes da aplicação servidora:

- **Controllers** - responsável por tratar dos pedidos recebidos;

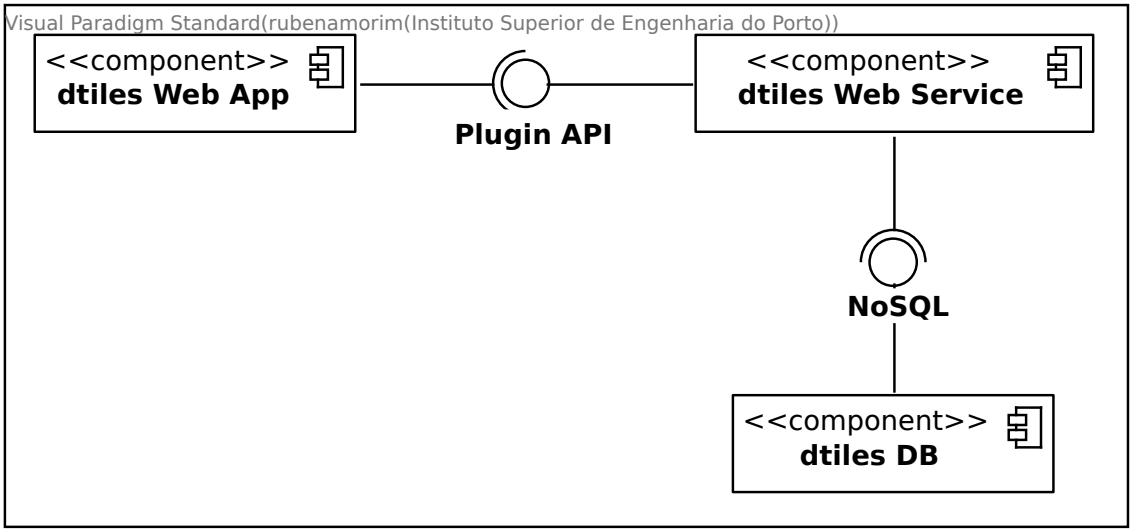


Figura 4.3: Diagrama de componentes do sistema

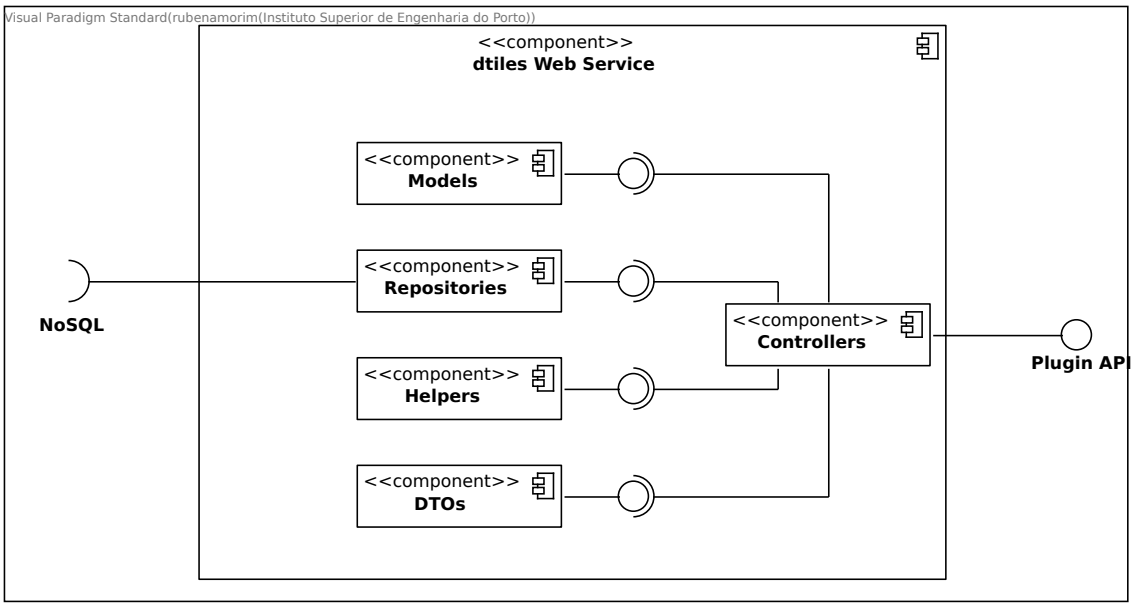


Figura 4.4: Diagrama de componentes da aplicação servidora

- **Repositories** - encarregue pela persistência dos dados e da interação com a base de dados, de acordo com o padrão Repository;
- **DTOs** - usado para transporte de dados entre diferentes componentes do sistema, de acordo com o padrão Data Transfer Object (DTO);
- **Models** - responsável por representar as classes de domínio e lógica de negócio;
- **Helpers** - conveniente por possuir classes utilitárias.

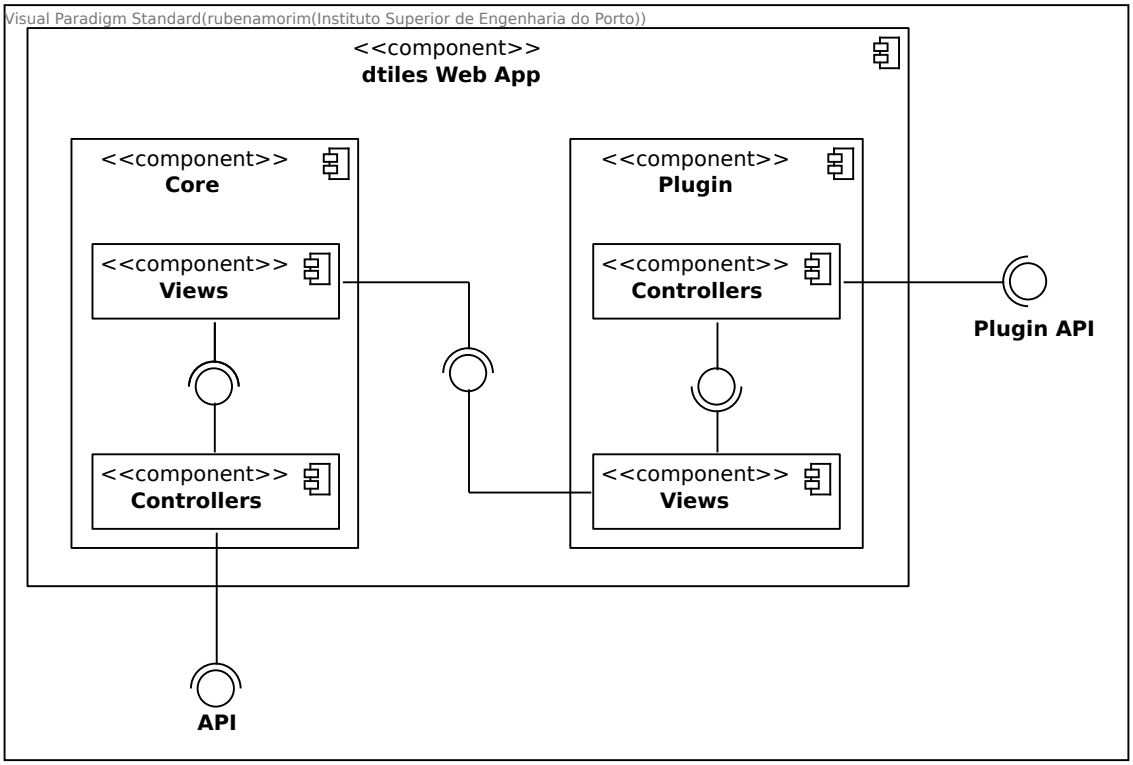


Figura 4.5: Diagrama de componentes da aplicação cliente

Conforme exibido na figura 4.5 a aplicação cliente segue o padrão Module, para manter componentes específicos independentes entre si, e é dividida nos seguintes componentes de alto nível:

- **Core** - responsável pela lógica base da aplicação cliente, interação com a aplicação servidora e apresentação dos dados;
- **Plugin** - representa uma das possíveis integrações, é responsável pela interação com o respetivo serviço e para isso contém a lógica associada, bem como a camada de apresentação dos dados.

Além disso, cada componente de alto nível descrito anteriormente, possui os seguintes componentes:

- **Views** - responsável pela interação do cliente final com o sistema, ou seja, a camada de apresentação;
- **Controllers** - encarregue de manipular os dados e responder a execuções originadas pela camada de apresentação.

4.2.2 Vista de implantação

A vista de implantação tem como objetivo entender as conexões físicas entre os nós, implantação e escalabilidade do sistema [33].

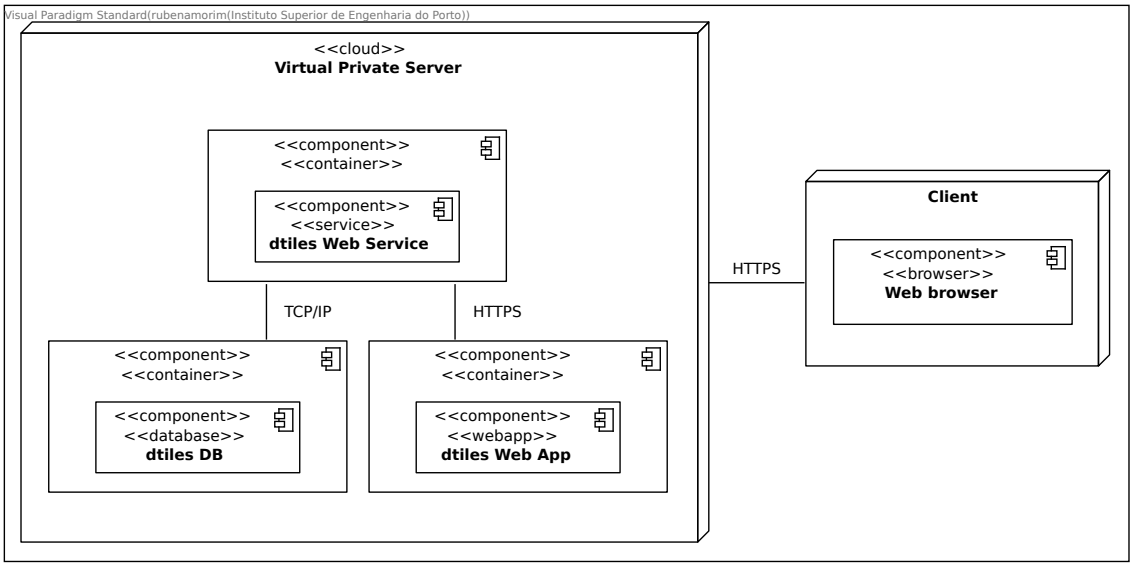


Figura 4.6: Diagrama de implantação do sistema

A figura 4.6 ilustra o diagrama de implantação do sistema e é possível observar que os componentes encontram-se dentro de contentores, a correr numa máquina virtual na *cloud*. Além disso, é apresentado o cliente que será o *web browser* e comunica com a aplicação servidora.

4.2.3 Alternativa arquitetural

A seguir, é apresentada uma alternativa arquitetural para as aplicações cliente e servidor, realçando aspetos relevantes sobre as mesmas.

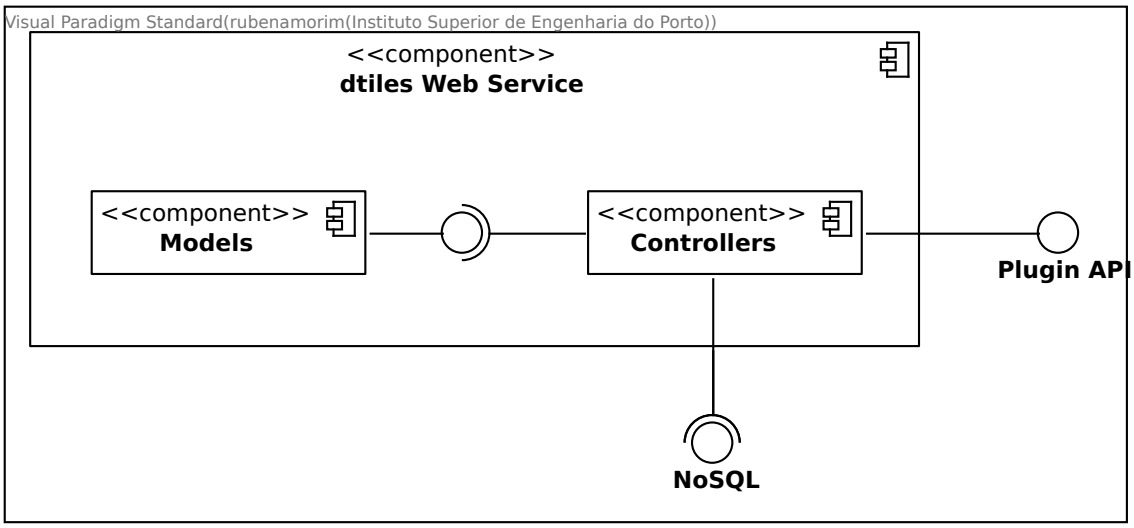


Figura 4.7: Diagrama de componentes alternativo da aplicação servidora

Quanto à aplicação servidora, como mostra a figura 4.7, a alternativa seria não adotar os padrões Repository e DTO. Deste modo, a comunicação com a base de dados seria feita nos *controllers*, o que iria centralizar alguma lógica distinta, e o transporte de dados entre diversos componentes do sistema teria limitações.

Por outro lado, numa perspetiva da aplicação cliente, uma alternativa seria manter a lógica da interface gráfica e controladores da aplicação nos mesmos componentes das integrações, como

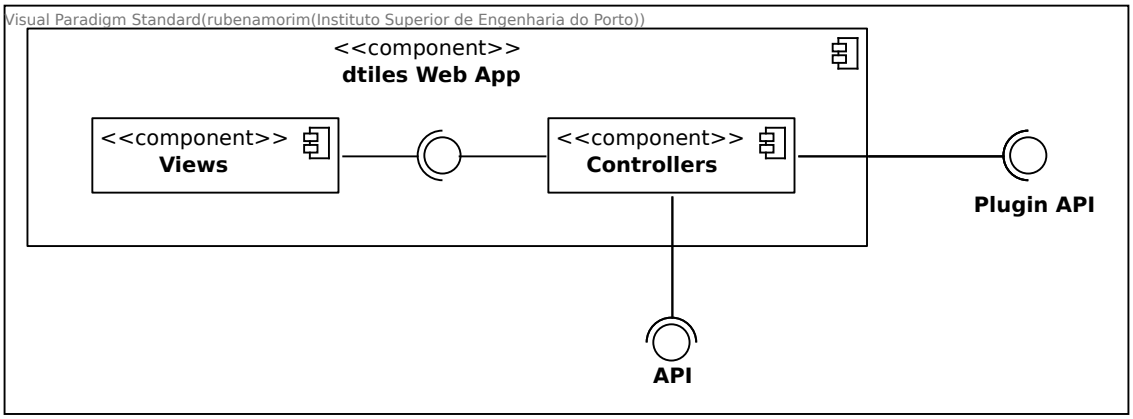


Figura 4.8: Diagrama de componentes alternativo da aplicação cliente

exibido na figura 4.8. Isto dificultaria a manutenção de código, não iria ao encontro da arquitetura de *plugins*, pois limitava a modularidade e escalabilidade da aplicação, e quebrava diversos padrões de engenharia de *software*. Além de que o desenvolvimento de módulos de integração com outros serviços não seria possível de ser feito de forma isolada e abstraída.

Posto isto, as alternativas arquiteturais apresentadas possuem diversos aspetos negativos face à arquitetura escolhida para o sistema.

Capítulo 5

Implementação

Neste capítulo é descrito o processo de implementação dos principais casos de uso da solução desenvolvida. Deste modo, nos subcapítulos que o compõem são apresentados os aspetos técnicos da consulta de um *dashboard*, público ou privado, assim como a adição de *widgets* ao mesmo.

Além disso, é exibida e descrita a UI da aplicação cliente, os testes unitários feitos ao sistema e o resultado das auditorias de qualidade realizadas com uma ferramenta da Google.

5.1 Frameworks

A implementação da aplicação servidora tem como base a *framework* KeystoneJS[1]que permite a geração de uma API GraphQL através de um *schema* declarado em JavaScript. A criação das operações Create, Read, Update and Delete (CRUD) é feita automaticamente com base no *schema* declarado, as quais podem ser modificadas ou substituídas, e facilmente se adicionam mecanismos de autorização e autenticação. Devido à sua extensibilidade, permite ainda servir na mesma instância do servidor Hypertext Transfer Protocol (HTTP) várias aplicações *web*.

Para a implementação da aplicação cliente foi usada a *framework* Next.js[2]que oferece uma melhor experiência de desenvolvimento com os recursos necessários para colocar uma aplicação otimizada em produção sem configurações adicionais.

Posto isto, no servidor para além da aplicação GraphQL para a API, foi configurada uma aplicação Next.js para servir a aplicação cliente. Deste modo, o servidor HTTP criado pela aplicação Node.js, disponibiliza no *endpoint* `/api` uma API GraphQL e as restantes rotas atuam como um *proxy* para a aplicação cliente.

5.2 Consultar dashboard

A consulta de um *dashboard* pode ser feita por qualquer utilizador registado com permissões de visualização, ou por qualquer utilizador, registado ou não, se o mesmo for público.

Posto isto, de acordo com excerto de código 5.1 que representa parte do *schema* declarado para o *dashboard*, é possível verificar a presença dos campos: `private`, `owner` e `guests`, que serviram de ponto de partida para verificar as permissões de acesso.

```
1 const fields = {
2     // ...
3     private: { type: Checkbox, defaultValue: true },
4     owner: { type: AuthedRelationship, ref: 'User.ownedDashboards' },
```

[1]https://www.keystonejs.com
[2]https://nextjs.org

```
    guests: {
        type: Relationship,
        ref: 'User.notOwnedDashboards',
        many: true,
    },
};
```

Código 5.1: *Schema* do *dashboard*

Deste modo, de acordo com o *schema* do *dashboard* e com recurso aos mecanismos de autorização do KeystoneJS[3], registou-se o *callback* apresentado no excerto de código 5.2 para verificação das permissões de leitura. A função declarada valida se o utilizador que fez o pedido tem autorização verificando se o *dashboard* é público ou se utilizador é proprietário do mesmo ou é um *guest*.

```
function getReadAccess({authentication: {item: user}, itemId}) {
    if (!user && itemId) {
        return { private: false };
    } else if (!user) {
        return false;
    }

    return {
        OR: [
            { owner: { id: user.id } },
            { guests_some: { id: user.id } },
        ],
    };
}
```

Código 5.2: *Callback* de validação das permissões de leitura do *dashboard*

Na aplicação cliente foi registada uma rota `/dashboard/[id]` para renderização do componente React associado à página *web*. Esta página é responsável por efetuar o pedido dos dados do *dashboard* ao servidor, através da *query* apresentada no excerto de código 5.3, e renderizar os respetivos *widgets* que o constituem (cf. secção 5.3).

```
query getDashboardById($id: ID!) {
    Dashboard(where: { id: $id }) {
        id
        name
        private
        isOwner
        automaticTransition
        transitionTime
        updateTime
        guests {
            id
            email
        }
        pages {
```

[3]https://www.keystonejs.com/api/access-control

```
15              id
16              widgets {
17                  id
18                  type
19                  x
20                  y
21                  rows
22                  columns
23                  data
24              }
25          }
26          _pagesMeta {
27              count
28          }
29      }
30  }
```

Código 5.3: GraphQL *query* do pedido dos detalhes de um *dashboard*

O diagrama de sequência da figura 5.1 apresenta o fluxo de consulta de um *dashboard*.

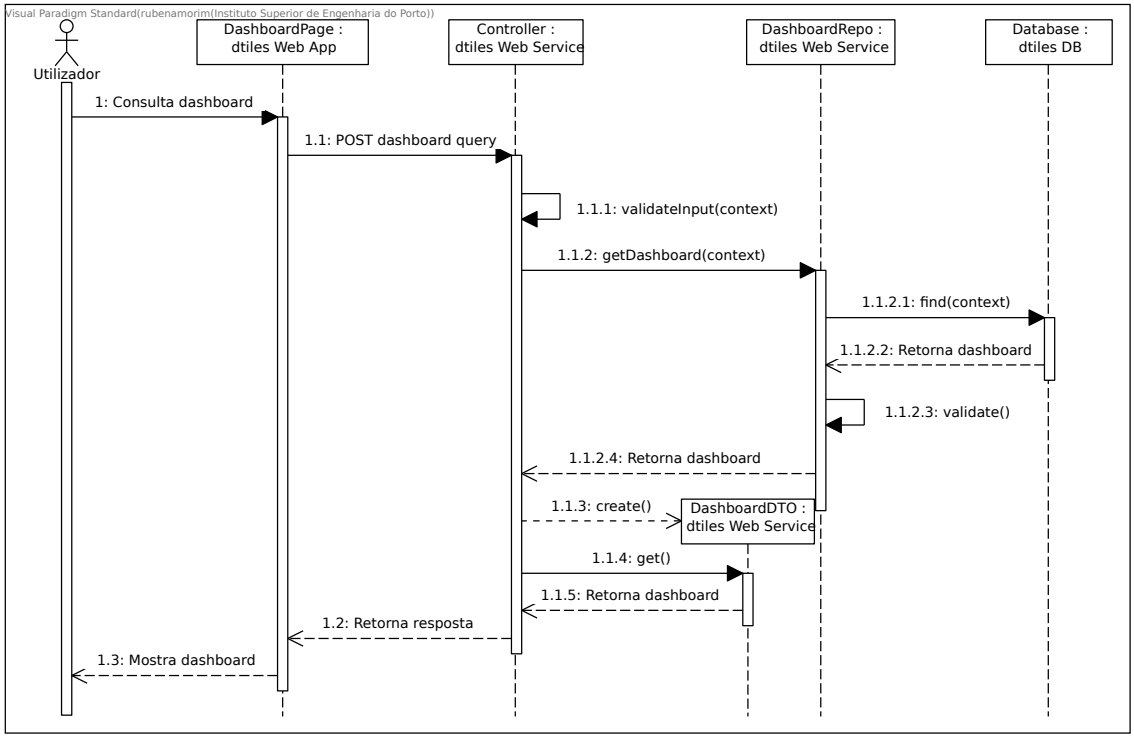


Figura 5.1: Diagrama de sequência de consulta de um *dashboard*

5.3 Adicionar widget ao dashboard

A adição de um *widget* ao *dashboard* pode ser feita pelo proprietário do mesmo, especificando qual o *plugin*, assim como os dados de configuração da integração, e o seu posicionamento na grelha da UI.

Deste modo, o excerto de código 5.4 mostra o *schema* declarado para representar um *widget* no *dashboard*. Este é composto por campos que identificam o tipo do *plugin*, a sua posição na grelha,

coordenadas e dimensão, e um outro campo livre que serve para o *plugin* armazenar os seus dados de configuração.

```
const fields = {
    type: { type: Text, isRequired: true },
    rows: { type: Integer, isRequired: true },
    columns: { type: Integer, isRequired: true },
    x: { type: Integer, isRequired: true },
    y: { type: Integer, isRequired: true },
    data: { type: Text },
    grid: {
        type: Relationship,
        ref: 'Grid.widgets',
        many: false,
    },
};
```

Código 5.4: *Schema* do *widget*

Na aplicação cliente foi criado um componente que renderiza um formulário e permite ao utilizador escolher o *plugin*, o *widget* e introduzir os seus dados de configuração. Este formulário, inicialmente mostra todos os *plugins* registados no contexto da aplicação e após a sua escolha, apresenta outro formulário totalmente dinâmico com os campos necessários à sua configuração. A necessidade de um formulário dinâmico surge devido a cada *plugin*/*widget* necessitar de configurações diferentes e que são desconhecidas do *core* da aplicação.

Assim sendo, aplicou-se o padrão Inversion of Control (IoC) para que cada *plugin* especifique o seu *schema* de configuração, passando a responsabilidade e controlo para o *plugin* e os componentes base da aplicação adaptarem-se, tornando assim a aplicação modular e facilitando a sua escalabilidade. Após o preenchimento do formulário, é feita a submissão dos dados com a *mutation* apresentada no excerto de código 5.5.

```
mutation createWidget($data: WidgetCreateInput) {
    createWidget(data: $data) {
        id
    }
}
```

Código 5.5: GraphQL *mutation* do pedido de criação de um *widget*

O diagrama de sequência da figura 5.2 mostra o fluxo de adição de um *widget* a um *dashboard*.

5.4 Criação de um plugin

Nesta secção é apresentado o processo de desenvolvimento de um *plugin* para a plataforma desenvolvida. Com o objetivo de mostrar e testar as funcionalidades, potencialidades da aplicação e validação arquitetural foi desenvolvido um *plugin* base para integração com a ferramenta de controlo de versões Gitlab[4]. Este *plugin* integra com a ferramenta e cria um *widget* contador de *merge requests* com os parâmetros suportados pelo serviço[5](estado, autor, projeto, entre outros).

[5]https://gitlab.com
[5]https://docs.gitlab.com/ee/api/merge_requests

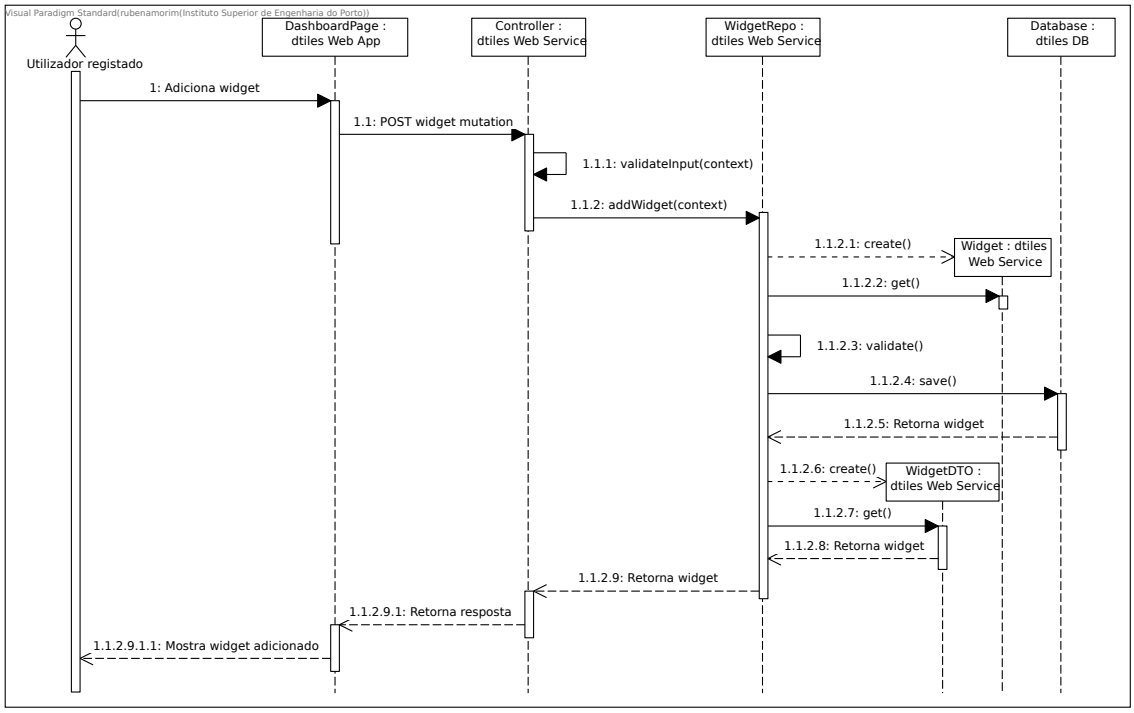


Figura 5.2: Diagrama de sequência de adição de um *widget* ao *dashboard*

A criação de um *plugin* é feita através da geração de um novo *package* na aplicação cliente de acordo com o padrão Module, para originar baixo acoplamento e manter as partes específicas do código independentes de outros componentes.

O excerto de código 5.6 exibe o registo do módulo do *plugin* no contexto da aplicação, através de um *engine* criado para o efeito, de maneira a que este fique disponível para ser adicionado aos *dashboards*.

```
pluginEngine.register({
    name: 'gitlab',
    label: 'Gitlab',
    description: 'Integrate Gitlab into your dashboard',
    schema,
    component: Gitlab,
});
```

Código 5.6: Registo do *plugin* no contexto da aplicação

Um *plugin* pode ter múltiplas variantes, que originam diferentes *widgets* com diferentes configurações associadas. Deste modo, foi criado o *schema* do *plugin* apresentado no excerto de código 5.7, com uma variante que corresponde ao *widget* contador de *merge requests* e com os parâmetros suportados pelo serviço que necessitam de ser preenchidos no formulário gerado (cf. secção 5.3).

```
const schema = {
    variants: [
        {
            id: 'mrs-counter',
            label: 'Merge Requests Counter',
```

```
            size: { columns: 2, rows: 3 },
            fields: [
                {
                    id: 'token',
                    label: 'Private Token',
                    type: 'text',
                    required: true,
                },
                {
                    id: 'domain',
                    label: 'Domain',
                    type: 'text',
                    placeholder: 'https://gitlab.com/',
                    required: true,
                },
                {
                    id: 'projectId',
                    label: 'Project ID',
                    type: 'text',
                },
                {
                    id: 'state',
                    label: 'State',
                    type: 'select',
                    options: [
                        { id: 'all', label: 'All' },
                        { id: 'opened', label: 'Opened' },
                        { id: 'closed', label: 'Closed' },
                        { id: 'merged', label: 'Merged' },
                    ],
                },
                {
                    id: 'targetBranch',
                    label: 'Target Branch',
                    type: 'text',
                    placeholder: 'master',
                },
                {
                    id: 'labels',
                    label: 'Labels',
                    type: 'text',
                },
                {
                    id: 'authorUsername',
                    label: 'Author Username',
                    type: 'text',
                },
            ],
        },
    ],
};
```

Código 5.7: *Schema* do *plugin*

Após a criação do *schema* foi criado o componente React correspondente ao que será apresentado no *dashboard*. Este componente recebe toda a configuração efetuada e tem a responsabilidade de fazer o pedido dos dados ao respetivo serviço, manipular a resposta e renderizar o *widget*. O excerto de código 5.8 apresenta o pedido que é feito à API REST do Gitlab para obter a contagem de *merge requests*.

```
const url = projectId
    ? `${domain}/api/v4/projects/${projectId}/merge_requests`
    : `${domain}/api/v4/merge_requests`;

const response = await axios.get(url, {
    headers: {
        'Private-Token': token,
    },
    params: {
        state,
        labels,
        author_username: authorUsername,
        target_branch: targetBranch,
    },
});
```

Código 5.8: Pedido dos *merge requests* ao serviço do Gitlab

Por fim, após o carregamento dos dados necessários e manipulação dos mesmos, é renderizado o *widget* com a UI similar ao exemplo da figura 5.3.

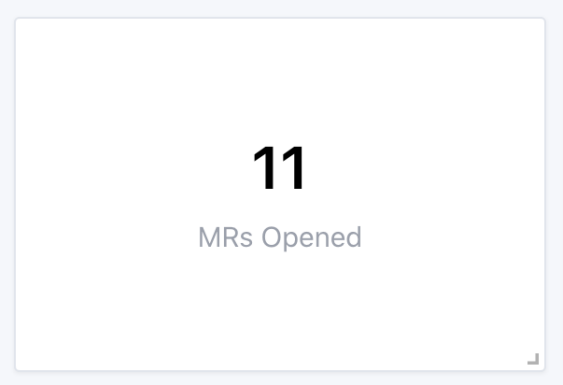


Figura 5.3: Exemplo do *widget* contador de *merge requests*

5.5 Interface gráfica

Nesta secção é apresentada e explicada a UI das três páginas desenvolvidas e que combinam os requisitos funcionais e não funcionais (cf. secções 4.1.1 e 4.1.2 respetivamente).

5.5.1 Página inicial

A página inicial é onde permite o utilizador efetuar o *login* e direciona-o para a sua página de *dashboards* (cf. secção 5.5.2). O *login* pode ser feito apenas usado o serviço da Google, onde é efetuado o processo de autenticação e após isso é feito o redirecionamento de volta para a plataforma. A figura 5.4 mostra a UI da página inicial.

Figura 5.4: Exemplo de UI da página inicial

5.5.2 Página de dashboards do utilizador

A página de *dashboards* do utilizador é a página inicial de um utilizador autenticado. Aqui é apresentada uma listagem de *dashboards* que o utilizador detém e outra dos quais tem permissões de visualização, e cada item desta lista dá acesso ao respetivo *dashboard* (cf. secção 5.5.3). A figura 5.5 exibe um exemplo da UI da página de listagem de *dashboards* do utilizador.

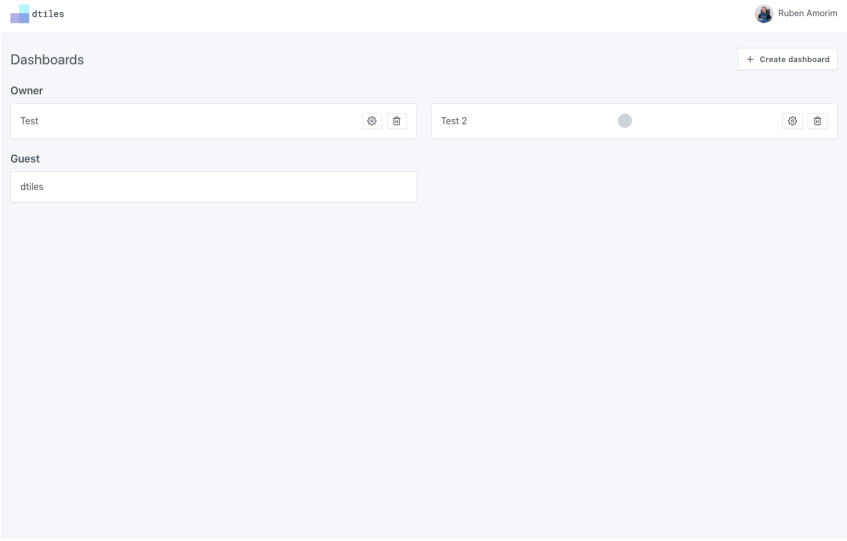


Figura 5.5: Exemplo de UI da página de *dashboards* do utilizador

Nesta página, também é possível efetuar a criação de um *dashboard* através do preenchimento e submissão do respetivo formulário, como apresentado na figura 5.6.

Além disso, no cabeçalho da página é possível visualizar o avatar do utilizador que ao ser clicado dá acesso às informações do utilizador autenticado. A figura 5.7 mostra um exemplo da UI do perfil do utilizador.

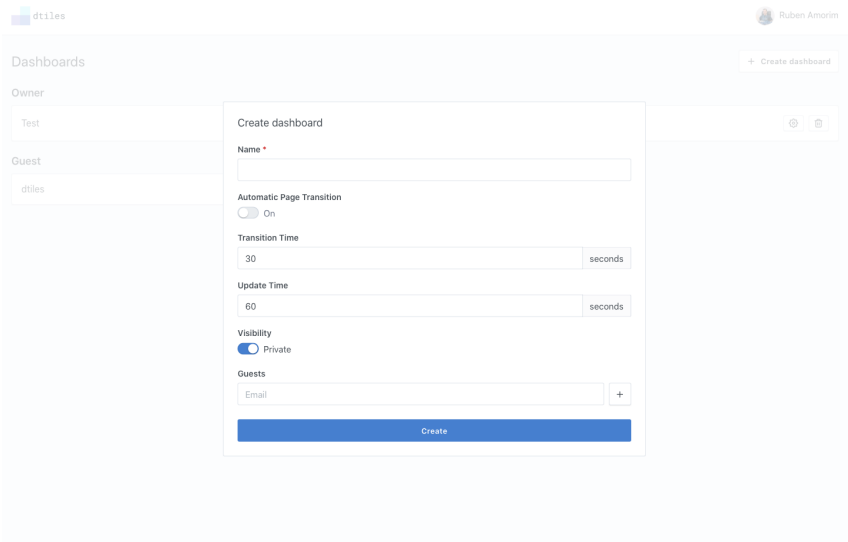


Figura 5.6: Exemplo de UI da criação de um *dashboard*

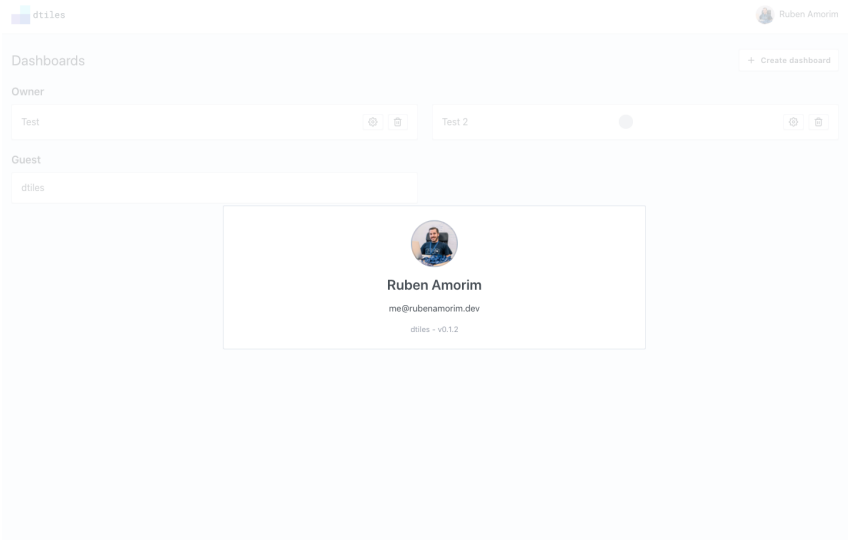


Figura 5.7: Exemplo de UI do perfil do utilizador

5.5.3 Página do dashboard

A página do *dashboard* é onde é possível visualizar os *widgets* configurados. Esta página possui um mecanismo de paginação com transição automática, quando configurado, e cada página contém uma grelha que é usada para dimensionar e posicionar os *widgets* de forma livre. A figura 5.8 apresenta um exemplo da UI da página do *dashboard*.

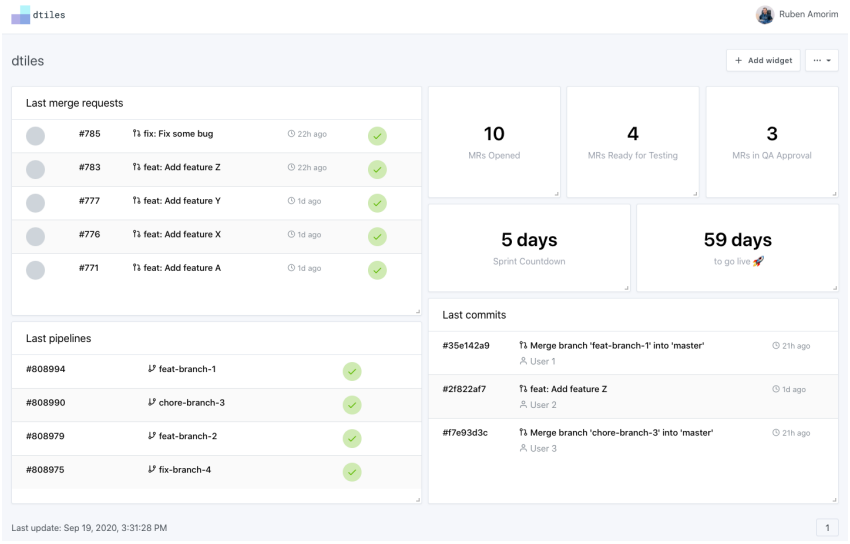


Figura 5.8: Exemplo de UI da página do *dashboard*

Nesta página também é permitido que os proprietários alterem todas as configurações associadas ao *dashboard* utilizando o mesmo formulário de criação, mas com os campos pré-preenchidos. A figura 5.9 mostra um exemplo de UI da edição das configurações do *dashboard*.

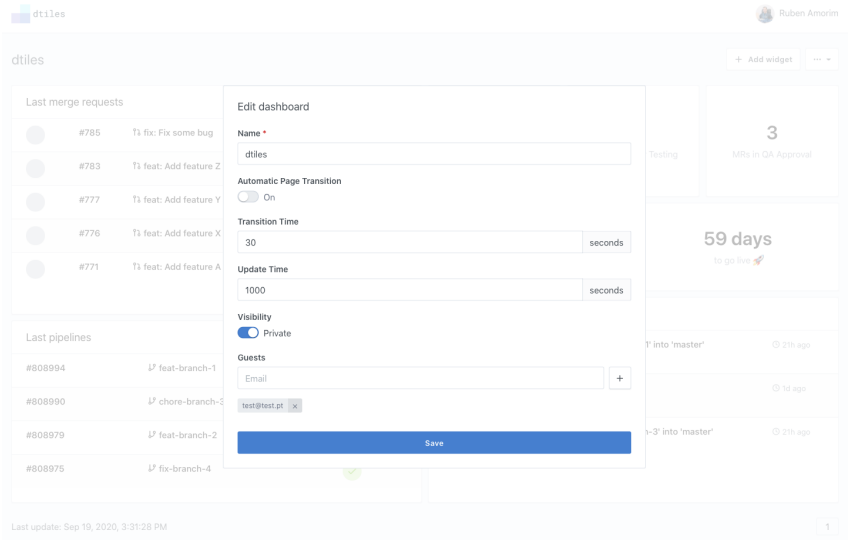


Figura 5.9: Exemplo de UI da edição das configurações do *dashboard*

Além disso, para serem adicionados *widgets* ao *dashboard*, surge um formulário que permite escolher um *plugin* e de seguida todos os campos relacionados com sua configuração. A figura 5.10 demonstra um exemplo de UI da adição de um *widget* ao *dashboard* e figura 5.11 da edição de um *widget*, com o formulário de configuração do respetivo *plugin*.

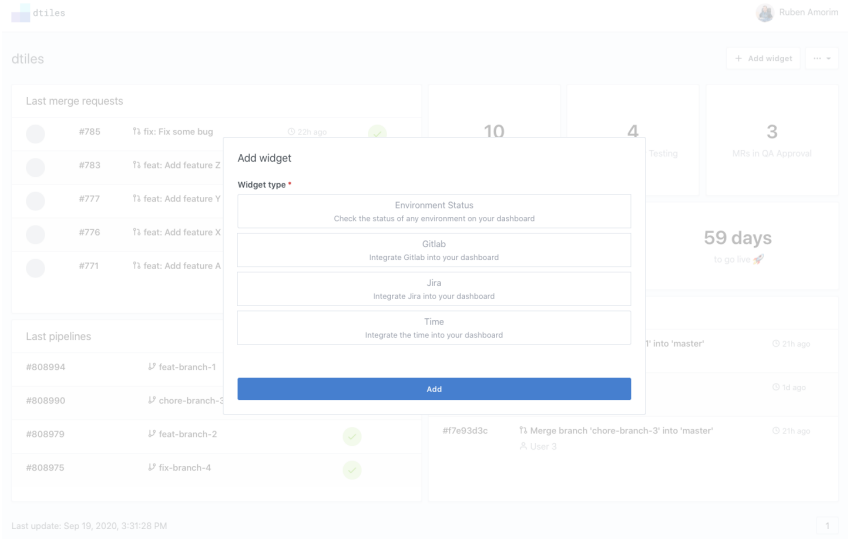


Figura 5.10: Exemplo de UI da configuração de um *widget*

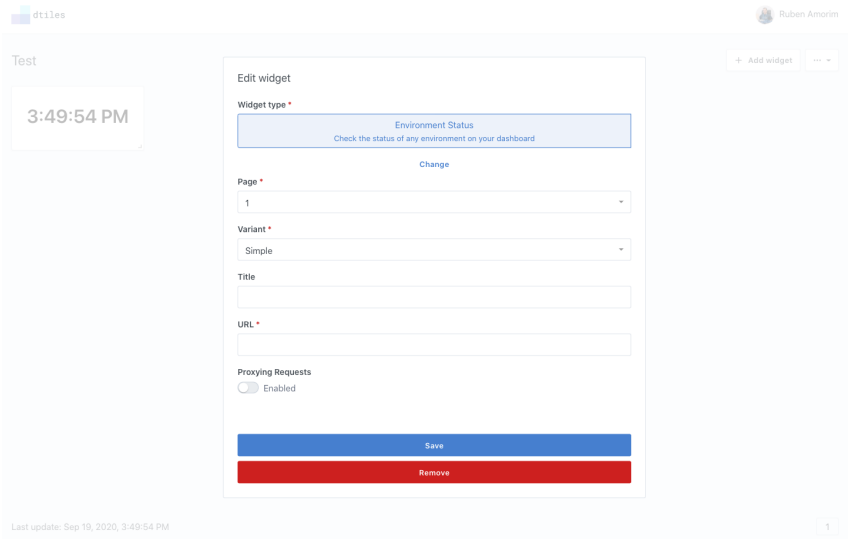


Figura 5.11: Exemplo de UI da edição das configurações de um *widget*

5.6 Testes

Neste projeto foram realizados testes unitários e de integração com recurso à *framework* de testes para JavaScript chamada Jest[6]em conjunto com React Testing Library[7]para testar componentes React e o Supertest[8]para auxiliar nos testes ao servidor HTTP. Além disso, na elaboração dos testes foi seguido o padrão Arrange Act Assert (AAA) para ajudar na organização e formatação dos mesmos.

O excerto de código 5.9 mostra o exemplo de um teste de integração realizado à aplicação servidora ao método de criação de um *dashboard*. Primeiro foi criada uma instância do servidor, depois é simulado um pedido ao recurso da API GraphQL com a respetiva *mutation* e no final espera-se uma resposta bem sucedida com os dados do *dashboard* criado.

```
1  it('should create a dashboard successfully', async (done) => {
2      const server = await keystone.before((adapterName) =>
3          setupServer({ ...configs, adapterName })
4      );
5
6      const query = `
7          mutation createDashboard($data: DashboardCreateInput) {
8              createDashboard(data: $data) {
9                  name
10                 private
11                 automaticTransition
12                 updateTime
13                 transitionTime
14             }
15         }
16     `;
17     const data = {
18         name: 'Dashboard 1',
19         private: true,
20         automaticTransition: true,
21         updateTime: 500,
22         transitionTime: 60,
23     };
24
25     request(server)
26         .post('/api')
27         .send({ query, variables: data })
28         .set('Accept', 'application/json')
29         .expect('Content-Type', /json/)
30         .expect(
31             200,
32             { data: { createDashboard: data }},
33             done
34         );
35 });
```

Código 5.9: Exemplo de um teste de integração na aplicação servidora

[8]https://jestjs.io
[8]https://testing-library.com
[8]https://github.com/visionmedia/supertest

Por outro lado, o excerto de código 5.10 mostra o exemplo de um teste de integração realizado na aplicação cliente ao componente que representa o formulário de criação do *dashboard*. Inicialmente fez-se o configuração dos *mocks* necessários, depois renderiza-se o componente de maneira a poder interagir com o mesmo, simulando o comportamento do utilizador, preenchendo os campos e no fim submete-se. Para a validação do mesmo, espera-se que no final tenha sido efetuada uma chamada ao servidor com os dados do formulário no *payload*.

```
    it('should create a dashboard', async () => {
        const newDashboard = {
            name: 'foo',
            transitionTime: 10,
            updateTime: 20,
            guests: ['user@test.com'],
            automaticTransition: true,
            private: false,
        };
        const mockedFn = jest.fn();

        mockedUseDashboards.mockImplementation(() => ({
            isLoading: false,
            actions: {
                create: mockedFn,
            },
        }));

        const { container, getByText, getByPlaceholderText } = render(
            <DashboardForm />
        );

        await waitFor(() => {
            expect(getByText('Create dashboard')).toBeInTheDocument();
        });

        userEvent.type(
            document.getElementsByName('name')[0],
            newDashboard.name
        );
        userEvent.click(getByText('On'));
        userEvent.clear(document.getElementsByName('transitionTime')[0]
    );
        userEvent.type(
            document.getElementsByName('transitionTime')[0],
            newDashboard.transitionTime.toString()
        );
        userEvent.clear(document.getElementsByName('updateTime')[0]);
        userEvent.type(
            document.getElementsByName('updateTime')[0],
            newDashboard.updateTime.toString()
        );
        userEvent.click(getByText('Private'));
        userEvent.type(getByPlaceholderText('Email'),
    newDashboard.guests[0]);
```

```
44          userEvent.click(container.getElementsByClassName('fe-plus')[0])
    ;
45          userEvent.click(getByText('Create'));
46
47          await waitFor(() => {
48              expect(mockedFn).toHaveBeenCalledWith(newDashboard);
49          });
50      });
```

Código 5.10: Exemplo de um teste de integração na aplicação cliente

Quanto à cobertura, o Instanbul[9]foi a ferramenta responsável pelo cálculo da percentagem de código que é abrangida pelos testes efetuados. No caso deste projeto, obtiveram-se valores elevados de cobertura (cf. anexo A), o que leva a que grande parte do código seja testado, prevenindo eventuais regressões e aumentando a qualidade do produto.

5.7 Auditorias

A medição do índice de desempenho, boas práticas, Search Engine Optimization (SEO) e acessibilidade da solução desenvolvida ficou a cargo da ferramenta Lighthouse[10]da Google. Para isso, foram realizadas auditorias às três páginas desenvolvidas na aplicação cliente e os resultados encontram-se no anexo B.

Na página inicial e na de listagem de *dashboards* obteve-se a pontuação máxima em todos os aspetos. Por outro lado, na página do *dashboard* foram obtidas pontuações próximas do máximo na *perfomance*, devido à quantidade de *widgets* presentes no ecrã afetarem o tempo de renderização dos mesmos no *browser*, e em acessibilidade, por causa do contraste de cores utilizado na UI de alguns *widgets*.

Assim sendo, na generalidade, a aplicação apresenta relatórios de auditorias com resultados excelentes, o que mostra elevada qualidade na aplicação desenvolvida.

[9]https://istanbul.js.org
[10]https://developers.google.com/web/tools/lighthouse

Capítulo 6

Avaliação

Neste capítulo é descrita a avaliação efetuada ao projeto realizado, analisando os questionários feitos aos colaboradores da organização e são apresentadas as conclusões em relação à qualidade da plataforma desenvolvida e à hipótese que se pretendia testar.

6.1 Hipóteses

A avaliação deste trabalho pretende validar a plataforma desenvolvida e através da utilização da mesma, testar em que medida a utilização de *dashboards* contribuem para agilizar o processo de desenvolvimento de *software*.

O sucesso de uma aplicação está dependente da aceitação do utilizador sendo imprescindível avaliar a satisfação do mesmo. Assim sendo, relativamente à solução desenvolvida, foi avaliada a satisfação do utilizador face à utilização geral da mesma. Deste modo, tendo em conta a escala de Likert, assume-se que o sistema é eficaz na dimensão em análise segundo as seguintes hipóteses formuladas:

H_0 : Inquéritos de satisfação não possuem um grau de satisfação igual ou superior a 3

H_1 : Inquéritos de satisfação possuem um grau de satisfação igual ou superior a 3

No que diz respeito à utilização de *dashboards* no contexto da organização, foi estudada a forma como estes permitem agilizar o processo de desenvolvimento de *software*, assim como as vantagens e desvantagens. Para isso, foram elaboradas as seguintes hipóteses a testar:

H_0 : O uso de *dashboards* não contribui para agilizar o processo de desenvolvimento de *software*

H_1 : O uso de *dashboards* contribui para agilizar o processo de desenvolvimento de *software*

6.2 Metodologia

Para testar e avaliar as hipóteses formuladas foram realizados questionários às equipas de desenvolvimento da organização utilizando a plataforma no seu quotidiano e providenciando *feedback* sobre a mesma.

O questionário elaborado (cf. anexo C) contém os seguintes grupos de questões:

1. Utilização de *dashboards* no processo de desenvolvimento de *software*;
2. Satisfação do utilizador relativamente à plataforma desenvolvida.

No primeiro grupo, pretende-se testar a segunda hipótese anteriormente formulada através de questões de resposta fechada, para perceber a adoção de *dashboards* pelas pessoas e equipas, assim como questões de resposta aberta para recolher informações sobre as possíveis vantagens e desvantagens.

Quanto ao segundo grupo, tem como objetivo testar a primeira hipótese formulada, através de questões de resposta fechada usando a escala de Likert, conforme a tabela 6.1. Estas questões avaliam a plataforma quanto às funcionalidades, usabilidade, desempenho e globalmente.

Tabela 6.1: Escala de Likert usada para a avaliação

Muito fraco	Fraco	Suficiente	Bom	Muito bom
1	2	3	4	5

6.3 Análise de resultados

De modo a testar as funcionalidades da plataforma, obter *feedback* e recolher informação sobre a solução desenvolvida, vários indivíduos usaram-na e no final preencheram um inquérito sobre o estudo em questão. Foi escolhida uma amostra por conveniência de 40 indivíduos, de uma equipa da organização, dos quais 11 providenciaram *feedback* (cf. anexo D).

Relativamente à amostra, 10 (90.9%) dos inquiridos apresentam funções de Programador na organização e 1 (9.1%) deles de Quality Assurance (QA), como é apresentado na figura 6.1.

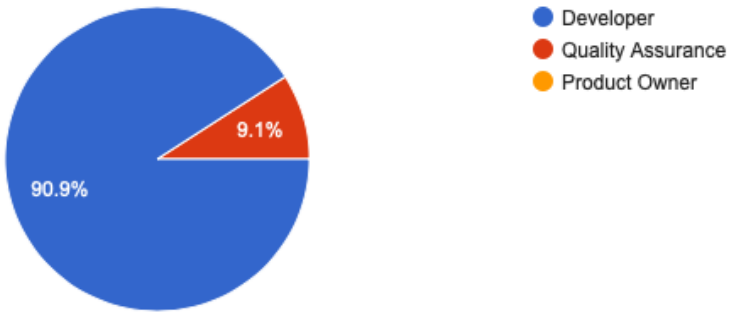


Figura 6.1: Frequência relativa das funções dos inquiridos

Além disso, apenas um dos participantes (9.1%) afirmou não trabalhar com metodologias ágeis.

6.3.1 Utilização de dashboards no processo de desenvolvimento de software

Para avaliar de que forma a adoção de *dashboards* contribuem para agilizar o processo de desenvolvimento de *software*, os inquiridos responderam a questões relacionadas com a adoção de *dashboards* para monitorização do trabalho, as suas vantagens e desvantagens.

Inicialmente, de acordo com a figura 6.2, nove (81.8%) pessoas responderam que utilizam *dashboards* para monitorização pessoal do trabalho, contra três (18.2%) que afirmou que não usam.

Por outro lado, na figura 6.3, em relação à monitorização por parte das equipas, dez (90.9%) dos inquiridos afirmou que a sua equipa utiliza *dashboards* como ferramenta de monitorização, enquanto apenas um (9.1%) inquirido aponta a não utilização.

Assim sendo, pode-se verificar uma taxa de adesão alta aos *dashboards* como ferramenta de monitorização, quer para utilização pessoal, quer pelas equipas de desenvolvimento de *software*. Alguns inquiridos especificaram o uso de diversas ferramentas para monitorização, tais como: Jira, Gilab,

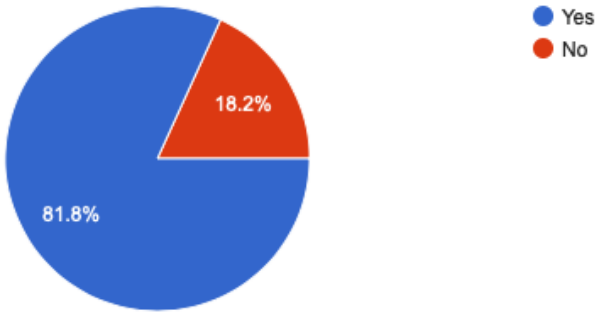


Figura 6.2: Frequência relativa do uso de ferramentas de monitorização pessoal

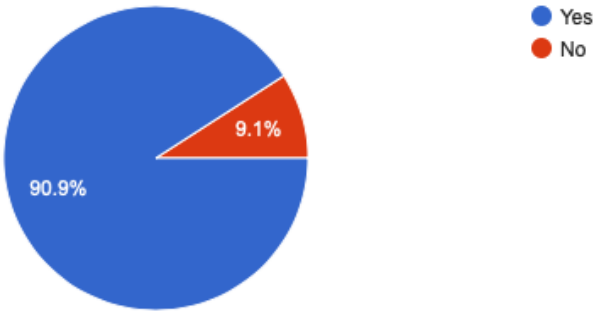


Figura 6.3: Frequência relativa do uso de ferramentas de monitorização pelas equipas

Jenkins, Github e Trello. Atualmente, com o uso da plataforma desenvolvida, estas ferramentas podem ser integradas e passar assim a ser usada apenas uma ferramenta de monitorização.

No que diz respeito à utilização de *dashboards* no processo de desenvolvimento de *software*, com base na figura 6.4, todos os inquiridos afirmaram que esta ferramenta de monitorização traz vantagens para o mesmo e apontaram as seguintes benefícios:

- Aumenta a consciencialização do processo;
- Melhora a organização da equipa;
- Melhora a comunicação da equipa;
- Acelera a resolução de problemas;
- Melhora o foco da equipa.

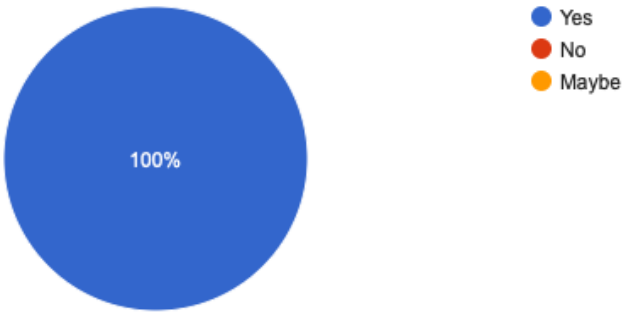


Figura 6.4: Frequência relativa das vantagens de utilização de *dashboards*

Em contrapartida, conforme a figura 6.5, apenas um (9.1%) dos participantes referiu que a adoção de *dashboards* traz desvantagens para o processo de processo de desenvolvimento de *software*. Desta forma, a única inconveniência apontada é o facto de piorar o foco da equipa.

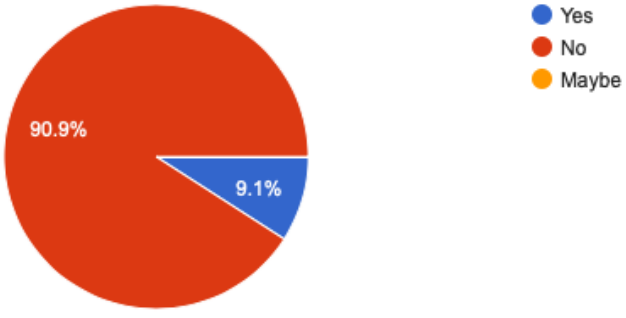


Figura 6.5: Frequência relativa das desvantagens de utilização de *dashboards*

Em suma, com base nos resultados obtidos anteriormente em que a amostra completa refere que é vantajoso a adoção de *dashboards* como ferramenta de monitorização e a hipótese formulada, pode-se concluir que o uso de *dashboards* contribui para agilizar o processo de desenvolvimento de *software*.

6.3.2 Satisfação do utilizador relativamente à plataforma desenvolvida

Após a avaliação do contexto em estudo, os participantes foram convidados a avaliar a plataforma desenvolvida em diversas dimensões, após a utilização da mesma. Os inquiridos responderam a um conjunto de questões, relacionados com a satisfação, com valores de 1 a 5, em que 1 significa muito fraco e 5 refere-se a muito mau.

Quanto às funcionalidades, como é apresentado na figura 6.6, cinco utilizadores classificaram com nota 4 e os restantes seis atribuíram nota 5. Assim sendo, pode-se concluir que as funcionalidades da solução desenvolvida são relevantes e têm um bom funcionamento.

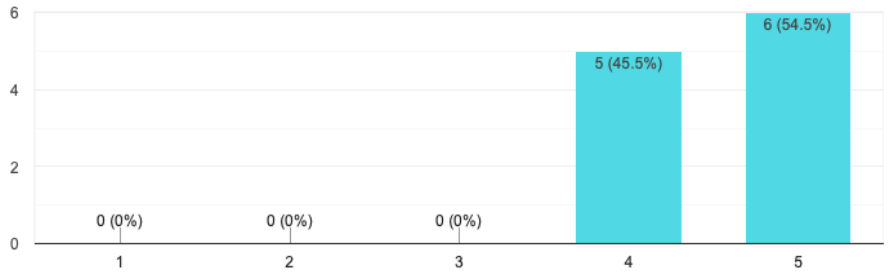


Figura 6.6: Frequência absoluta da avaliação das funcionalidades da plataforma

No que diz respeito à usabilidade, segundo a figura 6.7, um utilizador deu nota 3 e dos dez restantes, metade deu nota 4 e a outra metade deu nota 5. Deste modo, permite-nos deduzir que a plataforma apresenta uma interface simples e intuitiva para o utilizador.

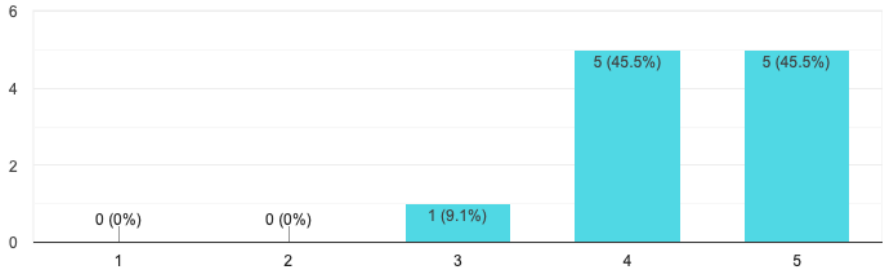


Figura 6.7: Frequência absoluta da avaliação da usabilidade da plataforma

O desempenho, como é mostrado na figura 6.8, foi classificado com nota 4 por três utilizadores e com nota 5 pelos restantes 8 utilizadores, permitindo assim inferir que a plataforma é rápida, consistente e exemplar.

De um modo geral, conforme a figura 6.9, cinco dos utilizadores classificaram a plataforma com nota 4 e os outros seis com nota 5. Posto isto, é possível concluir que a plataforma desenvolvida apresenta uma qualidade elevada de acordo com a classificação obtida, com notas altas e consistentes.

Finalmente, dois dos participantes providenciaram comentários sobre o trabalho desenvolvido:

- "Nice job!"
- "Nice work"

Para testar a hipótese desta avaliação, foram calculadas as médias para cada dimensão e os resultados são apresentados na tabela 6.2.

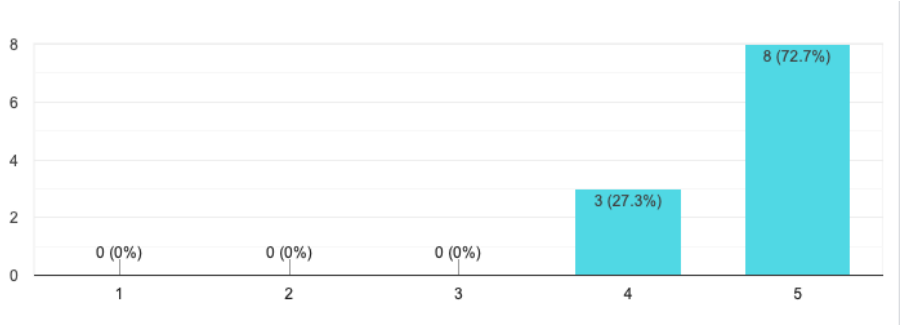


Figura 6.8: Frequência absoluta da avaliação do desempenho da plataforma

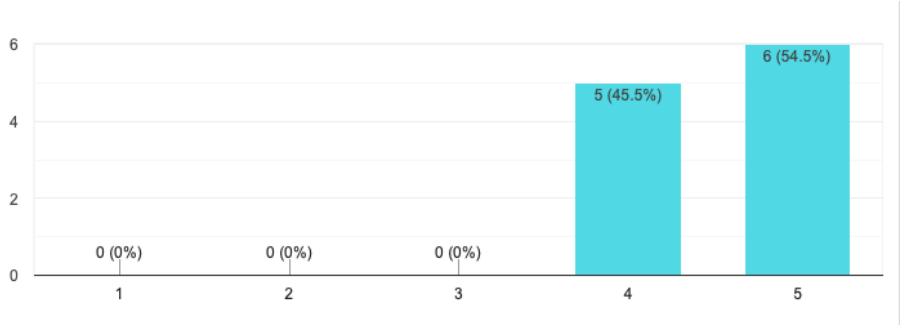


Figura 6.9: Frequência absoluta da avaliação global da plataforma

Tabela 6.2: Valores médios de satisfação dos utilizadores

Dimensão	**Média**
Funcionalidade	4.55
Usabilidade	4.36
Desempenho	4.73
Global	4.55
Total	4.55

Tendo em conta a hipótese formulada, a média das respostas ao inquérito é de 4.55, o que demonstra uma ótima satisfação do utilizador relativamente à plataforma desenvolvida.

Capítulo 7

Conclusão

Neste capítulo são apresentadas as conclusões relativas ao trabalho realizado, assim como uma descrição dos objetivos alcançados, as limitações do estudo efetuado e trabalho para o futuro, com algumas novas funcionalidades, visando a melhoria do produto.

Por último, é realizada uma apreciação final acerca do projeto de uma perspetiva pessoal e profissional.

7.1 Objetivos alcançados

O objetivo principal deste trabalho foi o desenvolvimento de uma ferramenta capaz de recolher e integrar informação de múltiplas fontes e apresentá-la de forma consistente e sistematizada. Esta ferramenta consiste numa plataforma que permite a criação, personalização e partilha de *dashboards*. Assim sendo, este objetivo considera-se cumprido com sucesso, uma vez que foram implementados todos os requisitos funcionais e não funcionais especificados tendo em conta boas práticas de engenharia de *software*.

O maior desafio foi a conceção de uma arquitetura suficiente modular para a integração de *plugins* desenvolvidos de forma isolada. Neste aspeto, a solução desenvolvida apresenta caraterísticas de modularidade, manutibilidade e escalabilidade exemplares.

Além disso, através da adoção da plataforma no quotidiano, foi estudado de que forma o uso de *dashboards* permite agilizar o processo de desenvolvimento de *software*, no contexto da organização Mindera. Os resultados obtidos permitem confirmar a hipótese formulada e ainda apontam os benefícios da adoção desta ferramenta de monitorização.

Em suma, todos os objetivos definidos para este projeto foram alcançados, assim como todas as tarefas foram realizadas.

7.2 Limitações

No contexto atual da pandemia do vírus COVID-19 foram impostas limitações às organizações para a adoção de teletrabalho. Uma vez que a plataforma desenvolvida seria para ser utilizada pelas equipas nos seus espaços de trabalho, em televisores para difundir a informação, tal não foi possível e condicionou assim o número de utilizadores, limitando os testes e estudo realizado.

Deste modo, apesar de existirem *dashboards* partilhados, os utilizadores da plataforma foram os colaboradores com iniciativa própria que a usaram como ferramenta de monitorização nos seus locais de trabalho.

7.3 Trabalho futuro

Um projeto dificilmente pode ser considerado como acabado ou completo pois existem sempre melhorias que podem ser feitas ou novas funcionalidades adicionadas.

Continuamente irão ser desenvolvidas integrações para a plataforma de acordo com as necessidades das equipas. Num futuro próximo, de acordo com o *roadmap*, serão desenvolvidos os seguintes *plugins*:

- Gitlab;
- Jira;
- Slack;
- Jenkins;
- Github;
- Iframe;
- Relógio;
- Estado de um ambiente/servidor.

Por outro lado, para tornar a plataforma num produto mais robusto, poderiam ser efetuadas as seguintes melhorias:

- Autenticação - adoção de mais alternativas de autenticação tais como: *email/password*, Facebook, Apple, Github, entre outras;
- Notificações - suporte de alertas personalizados tendo em conta aquilo que está a ser monitorizado, através dos seguintes canais: *email*, tecnologia *push* e *webhooks*;
- Progressive Web App (PWA)[1]- tornar a aplicação web existente em uma PWA para melhorar a experiência de utilização;
- Documentação - criação de documentação específica para programadores, explicando a API da plataforma e um guia de como criar *plugins* para a mesma;
- *Templates* - possibilidade de tornar um *dashboard* como *template* para outros utilizadores poderem usá-lo como base na criação de outros;
- Temas - utilização de múltiplos temas de personalização na plataforma, por exemplo, o *dark mode*;
- *Marketplace* - criação de uma loja para *plugins*, *templates* e temas de maneira a facilitar a instalação e adoção dos mesmos nos *dashboards*.

7.4 Apreciação final e pessoal

No final do projeto, e fazendo uma retrospetiva ao percurso realizado, este projeto ajudou-me a consolidar conhecimentos adquiridos anteriormente, bem como aprender novas linguagens e ferramentas de programação. Deste modo, posso afirmar que este trabalho teve um balanço muito positivo no meu crescimento pessoal e profissional.

No que se refere à solução desenvolvida, à documentação produzida e ao planeamento seguido, foi guiado por boas práticas do desenvolvimento de *software*, o que proporciona qualidade à solução alcançada. Vários padrões de desenho de *software* foram utilizados na realização do projeto, alguns implementados na totalidade pelo código da aplicação, outros abstraídos pelas *frameworks* e linguagens de programação utilizadas.

[1] https://web.dev/what-are-pwas/

Quanto à minha postura como estudante do Mestrado em Engenharia Informática (MEI) do Instituto Superior de Engenharia do Porto (ISEP) perante a organização Mindera, penso que dignifiquei positivamente a instituição de ensino, tendo cumprido com tudo aquilo a que me propus contratualmente e voluntariamente.

Em suma, considero-me satisfeito com este projeto pois teve um contributo positivo para mim, a nível pessoal e profissional, e para a organização na qual foi inserido.

Bibliografia

[1] *Mindera*. 2020. url: https://mindera.com/ (acedido em 08/11/2019).

[2] Martin Fowler e Jim Highsmith. *The Agile Manifesto*. Rel. téc. 2001. url: www.martinfowler.com/articles/newMethodology.html.

[3] Jan Schwarzer et al. «Ambient surfaces: Interactive displays in the informative workspace of co-located scrum teams». Em: *ACM International Conference Proceeding Series*. Association for Computing Machinery, 2016. isbn: 9781450347631. doi: 10.1145/2971485.2971493.

[4] Devender Maheshwari e Marijn Janssen. «Dashboards for supporting organizational development: Principles for the design and development of public sector performance dashboards». Em: *ACM International Conference Proceeding Series*. Association for Computing Machinery, 2014, pp. 178–185. isbn: 9781605586113. doi: 10.1145/2691195.2691224.

[5] Dan Radigan. *Kanban - A brief introduction*. 2020. url: https://www.atlassian.com/agile/kanban (acedido em 08/11/2019).

[6] Nevenka Kirovska e Saso Koceski. «Usage of Kanban methodology at software development teams». Em: *Journal of Applied Economics and Business* 4.3 (2015), pp. 25–34. url: http://www.aebjournal.org/articles/0303/030302.pdf.

[7] Wilhelm Meding. «Effective monitoring of progress of agile software development teams in modern software companies - an industrial case study». Em: *ACM International Conference Proceeding Series*. Vol. Part F1319. Association for Computing Machinery, 2017, pp. 23–32. isbn: 9781450348539. doi: 10.1145/3143434.3143449. url: https://doi.org/10.1145/3143434.3143449.

[8] Julia Paredes, Craig Anslow e Frank Maurer. «Information Visualization for Agile Software Development». Em: *2014 Second IEEE Working Conference on Software Visualization*. IEEE, 2014, pp. 157–166. isbn: 978-1-4799-6150-4. doi: 10.1109/VISSOFT.2014.32. url: http://ieeexplore.ieee.org/document/6980227/.

[9] A Cockburn e J Highsmith. «Agile software development, the people factor». Em: *Computer* 34.11 (2001), pp. 131–133. issn: 1558-0814. doi: 10.1109/2.963450.

[10] Frauke Paetsch, Armin Eberlein e Frank Maurer. «Requirements engineering and agile software development». Em: *WET ICE 2003. Proceedings. Twelfth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, 2003.* IEEE Comput. Soc, 2003, pp. 308–313. isbn: 0-7695-1963-6. doi: 10.1109/ENABL.2003.1231428. url: http://ieeexplore.ieee.org/document/1231428/.

[11] Torgeir Dingsøyr et al. *A decade of agile methodologies: Towards explaining agile software development*. 2012. doi: 10.1016/j.jss.2012.02.033.

[12] VersionOne. *13th State of Agile Report*. Rel. téc. 2019. url: https://www.stateofagile.com/%7B%5C#%7Dufh-i-521251909-13th-annual-state-of-agile-report/473508.

[13] Torgeir Dingsøyr e Casper Lassenius. «Emerging themes in agile software development: Introduction to the special section on continuous value delivery». Em: *Information and Software Technology* 77 (2016), pp. 56–60. issn: 09505849. doi: 10.1016/j.infsof.2016.04.018. url: https://linkinghub.elsevier.com/retrieve/pii/S0950584916300829.

[14] Olivier Liechti, Jacques Pasquier e Rodney Reis. «Beyond dashboards: On the many facets of metrics and feedback in agile organizations». Em: *Proceedings - 2017 IEEE/ACM 10th International Workshop on Cooperative and Human Aspects of Software Engineering, CHASE 2017*. Institute of Electrical e Electronics Engineers Inc., 2017, pp. 16–22. isbn: 9781538640395. doi: 10.1109/CHASE.2017.5. url: https://doi.org/10.1109/CHASE.2017.5.

[15] Margaret-Anne Storey e Christoph Treude. «Software Engineering Dashboards: Types, Risks, and Future». Em: *Rethinking Productivity in Software Engineering*. Apress, 2019, pp. 179–190. doi: 10.1007/978-1-4842-4221-6_16. url: https://doi.org/10.1007/978-1-4842-4221-6%7B%5C_%7D16.

[16] Miroslaw Staron. *Dashboard development guide - How to build sustainable and useful dashboards to support software development and maintenance*. Rel. téc. 2015. url: https://gupea.ub.gu.se/bitstream/2077/41120/1/gupea%7B%5C_%7D2077%7B%5C_%7D41120%7B%5C_%7D1.pdf.

[17] Christoph Treude e Margaret-Anne Storey. «Awareness 2.0: Staying aware of projects, developers and tasks using dashboards and feeds». Em: *Proceedings - International Conference on Software Engineering*. Vol. 1. 2010, pp. 365–374. isbn: 9781605587196. doi: 10.1145/1806799.1806854. url: https://ieeexplore.ieee.org/abstract/document/6062104.

[18] Vladimir Ivanov et al. «Design and validation of precooked developer dashboards». Em: *ESEC/FSE 2018 - Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. Association for Computing Machinery, Inc, 2018, pp. 821–826. isbn: 9781450355735. doi: 10.1145/3236024.3275530. url: https://doi.org/10.1145/3236024.3275530.

[19] Ali Pourshahid. *Learn how dashboards can help agile software development teams*. 2016. url: https://www.klipfolio.com/blog/dashboards-agile-software-development (acedido em 22/12/2019).

[20] Ali Pourshahid. *Key metrics for agile development teams*. 2015. url: https://www.klipfolio.com/blog/key-metrics-agile-development-teams (acedido em 16/02/2020).

[21] Priyesh Patel. *What exactly is Node.js?* 2018. url: https://www.freecodecamp.org/news/what-exactly-is-node-js-ae36e97449f5/ (acedido em 11/02/2020).

[22] AltexSoft. *Pros and Cons of Node.js Web App Development | AltexSoft*. 2019. url: https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-node-js-web-app-development/ (acedido em 11/02/2020).

[23] *React*. 2020. url: https://reactjs.org/ (acedido em 05/02/2020).

[24] Ari Lerner. *Fullstack React: What is React?* 2018. url: https://www.newline.co/fullstack-react/30-days-of-react/day-1/ (acedido em 12/02/2020).

[25] *GraphQL*. 2020. url: https://graphql.org/ (acedido em 05/02/2020).

[26] Olaf Hartig e Jorge Pérez. «Semantics and Complexity of GraphQL». Em: (2018), p. 10. doi: 10.1145/3178876.3186014. url: https://doi.org/10.1145/3178876.3186014.

[27] Matheus Seabra, Marcos Felipe Nazário e Gustavo Pinto. «REST or GraphQL? A Performance Comparative Study». Em: (2019), pp. 123–132. doi: 10.1145/3357141.3357149. url: https://doi.org/10.1145/3357141.3357149.

[28] Ben Gregory. *Six Open Source Dashboards to Organize Your Data*. 2016. url: https://www.astronomer.io/blog/six-open-source-dashboards/ (acedido em 08/11/2019).

[29] *Mozaïk*. 2019. url: http://mozaik.rocks/ (acedido em 06/02/2020).

[30] *Geckoboard*. 2020. url: https://www.geckoboard.com/ (acedido em 06/02/2020).

[31] *Smashing*. 2019. url: https://smashing.github.io/ (acedido em 06/02/2020).
[32] *Tipboard*. 2019. url: https://allegro.tech/tipboard/ (acedido em 06/02/2020).
[33] P.B. Kruchten. «The 4+1 View Model of architecture». Em: *IEEE Software* 12.6 (1995), pp. 42–50. issn: 07407459. doi: 10.1109/52.469759. url: http://ieeexplore.ieee.org/document/469759/.

Anexo A

Relatório de cobertura pelos testes

All files

96.5% Statements `3635/3767` **75.66%** Branches `1169/1545` **76.59%** Functions `265/346` **96.5%** Lines `3635/3767`

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

File	Statements		Branches		Functions		Lines	
app-web/client	100%	9/9	100%	3/3	100%	1/1	100%	9/9
app-web/components/base-page	100%	80/80	87.5%	21/24	85.71%	6/7	100%	80/80
app-web/components/dashboard	89.29%	100/112	56.36%	31/55	61.54%	8/13	89.29%	100/112
app-web/components/dashboard-form	95.85%	231/241	83.72%	108/129	85.71%	18/21	95.85%	231/241
app-web/components/dashboard-list	100%	146/146	82.42%	75/91	88.89%	16/18	100%	146/146
app-web/components/page-loader	100%	14/14	75%	3/4	100%	2/2	100%	14/14
app-web/components/pagination	100%	55/55	70%	7/10	100%	2/2	100%	55/55
app-web/components/profile	100%	36/36	62.5%	5/8	100%	2/2	100%	36/36
app-web/components/sign-in	100%	27/27	75%	3/4	100%	2/2	100%	27/27
app-web/components/widget	98.14%	158/161	80%	32/40	73.33%	11/15	98.14%	158/161
app-web/components/widget-form	91.99%	425/462	76.62%	154/201	71.43%	20/28	91.99%	425/462
app-web/hocs	100%	71/71	80.43%	37/46	75%	12/16	100%	71/71
app-web/hooks	100%	527/527	68.2%	163/239	79.37%	50/63	100%	527/527
app-web/pages	98.98%	97/98	80%	24/30	84.62%	11/13	98.98%	97/98
app-web/pages/dashboard	84.74%	211/249	79.66%	47/59	37.5%	6/16	84.74%	211/249
plugin-env-status/src	100%	94/94	78.16%	68/87	87.5%	14/16	100%	94/94
plugin-gitlab/src	99.1%	330/333	77.78%	7/9	75%	3/4	99.1%	330/333
plugin-gitlab/src/components/commits	100%	147/147	74.12%	63/85	81.25%	13/16	100%	147/147
plugin-gitlab/src/components/merge-requests	99.13%	229/231	76.79%	86/112	81.25%	13/16	99.13%	229/231
plugin-gitlab/src/components/merge-requests-counter	100%	93/93	72.5%	58/80	81.25%	13/16	100%	93/93
plugin-gitlab/src/components/pipelines	100%	147/147	73.49%	61/83	81.25%	13/16	100%	147/147
plugin-jira/src	100%	82/82	88.89%	8/9	75%	3/4	100%	82/82
plugin-jira/src/components/sprint-countdown	100%	101/101	72.62%	61/84	81.25%	13/16	100%	101/101
plugin-time/src	100%	72/72	90%	9/10	75%	3/4	100%	72/72
plugin-time/src/components/clock	94.74%	36/38	73.33%	11/15	66.67%	4/6	94.74%	36/38
plugin-time/src/components/countdown	96.72%	59/61	80%	16/20	66.67%	4/6	96.72%	59/61
server/src	72.5%	58/80	100%	8/8	28.57%	2/7	72.5%	58/80

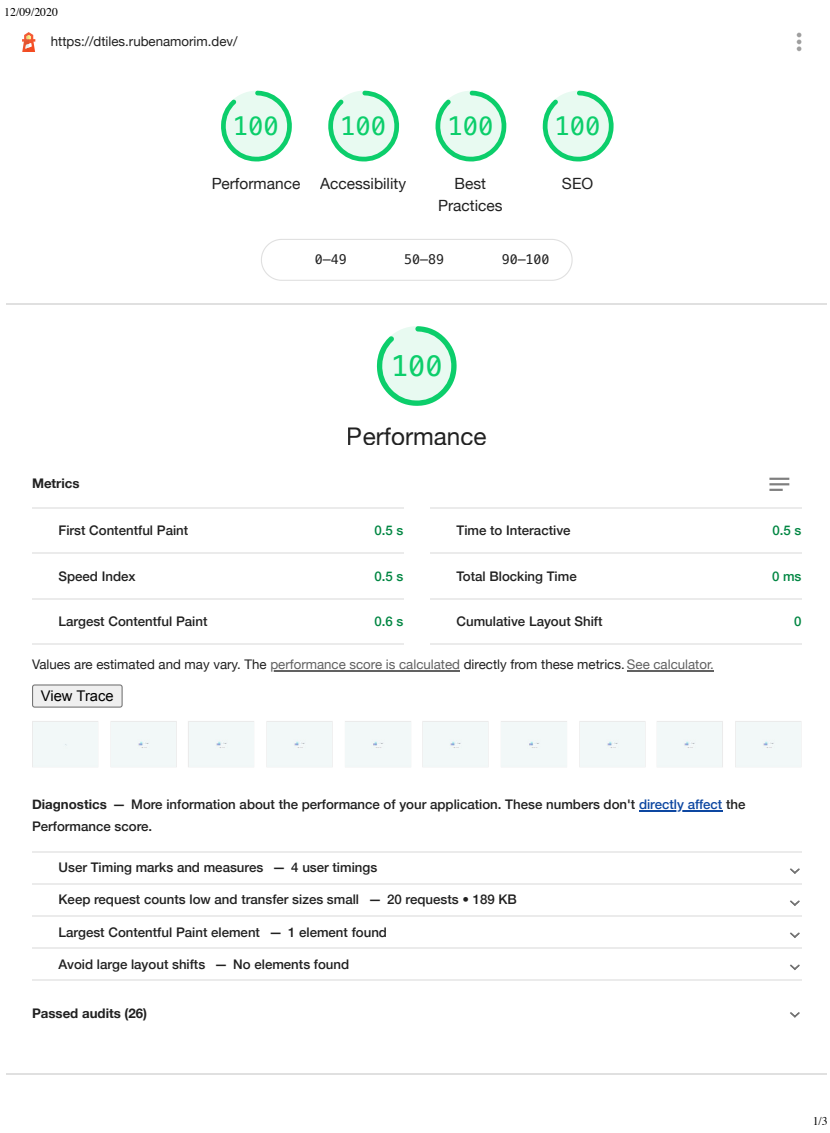
Code coverage generated by istanbul at Fri Oct 02 2020 22:23:49 GMT+0000 (Greenwich Mean Time)

Figura A.1: Relatório de cobertura pelos testes

Anexo B

Relatórios da ferramenta Lighthouse

B.1 Página inicial



Accessibility

These checks highlight opportunities to improve the accessibility of your web app. Only a subset of accessibility issues can be automatically detected so manual testing is also encouraged.

Additional items to manually check (10) — These items address areas which an automated testing tool cannot cover. Learn more in our guide on conducting an accessibility review.

Passed audits (8)

Not applicable (33)

Best Practices

Passed audits (14)

SEO

These checks ensure that your page is optimized for search engine results ranking. There are additional factors Lighthouse does not check that may affect your search ranking. Learn more.

Additional items to manually check (1) — Run these additional validators on your site to check additional SEO best practices.

Passed audits (8)

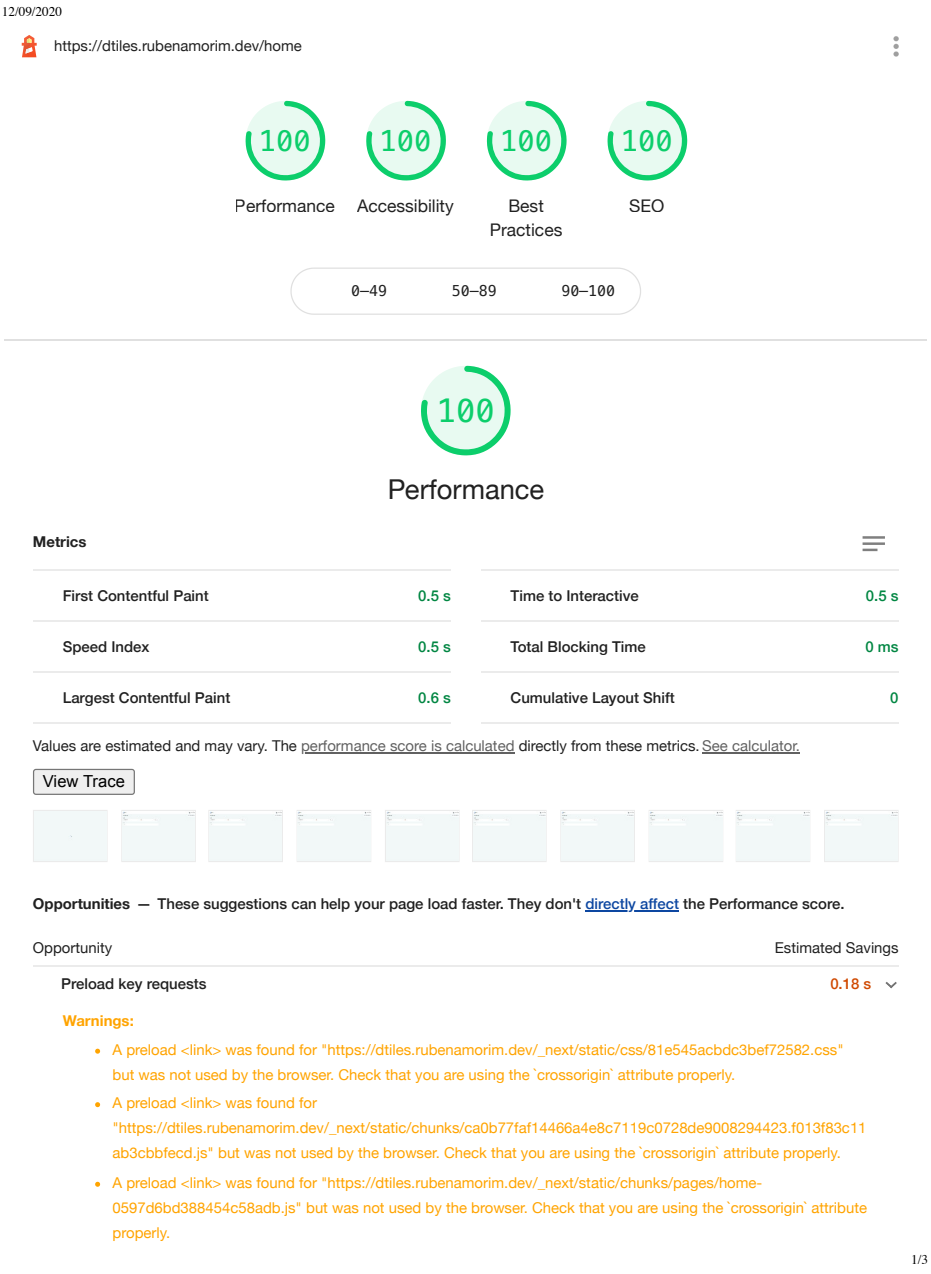
Not applicable (5)

Runtime Settings

URL	https://dtiles.rubenamorim.dev/
Fetch Time	Sep 12, 2020, 11:37 PM GMT+1
Device	Emulated Desktop
Network throttling	40 ms TCP RTT, 10,240 Kbps throughput (Simulated)
CPU throttling	1x slowdown (Simulated)
Channel	devtools
User agent (host)	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.102 Safari/537.36
User agent (network)	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3963.0 Safari/537.36 Chrome-Lighthouse
CPU/Memory Power	1650

Generated by **Lighthouse** 6.0.0 | File an issue

B.2 Página de dashboards do utilizador



12/09/2020

https://dtiles.rubenamorim.dev/home

100 Performance

100 Accessibility

100 Best Practices

100 SEO

0–49 50–89 90–100

100

Performance

Metrics

First Contentful Paint	0.5 s	Time to Interactive	0.5 s
Speed Index	0.5 s	Total Blocking Time	0 ms
Largest Contentful Paint	0.6 s	Cumulative Layout Shift	0

Values are estimated and may vary. The performance score is calculated directly from these metrics. See calculator.

View Trace

Opportunities — These suggestions can help your page load faster. They don't directly affect the Performance score.

Opportunity	Estimated Savings
Preload key requests	0.18 s

Warnings:

- A preload <link> was found for "https://dtiles.rubenamorim.dev/_next/static/css/81e545acbdc3bef72582.css" but was not used by the browser. Check that you are using the `crossorigin` attribute properly.
- A preload <link> was found for "https://dtiles.rubenamorim.dev/_next/static/chunks/ca0b77faf14466a4e8c7119c0728de9008294423.f013f83c11ab3cbbfecd.js" but was not used by the browser. Check that you are using the `crossorigin` attribute properly.
- A preload <link> was found for "https://dtiles.rubenamorim.dev/_next/static/chunks/pages/home-0597d6bd388454c58adb.js" but was not used by the browser. Check that you are using the `crossorigin` attribute properly.

1/3

Diagnostics — More information about the performance of your application. These numbers don't directly affect the Performance score.

- Serve static assets with an efficient cache policy — 3 resources found
- User Timing marks and measures — 4 user timings
- Keep request counts low and transfer sizes small — 30 requests • 415 KB
- Largest Contentful Paint element — 1 element found
- Avoid large layout shifts — 1 element found

Passed audits (24)

Accessibility

These checks highlight opportunities to improve the accessibility of your web app. Only a subset of accessibility issues can be automatically detected so manual testing is also encouraged.

Additional items to manually check (10) — These items address areas which an automated testing tool cannot cover. Learn more in our guide on conducting an accessibility review.

Passed audits (13)

Not applicable (28)

Best Practices

Passed audits (14)

These checks ensure that your page is optimized for search engine results ranking. There are additional factors Lighthouse does not check that may affect your search ranking. Learn more.

Additional items to manually check (1) — **Run these additional validators on your site to check additional SEO best practices.**

Passed audits (8)

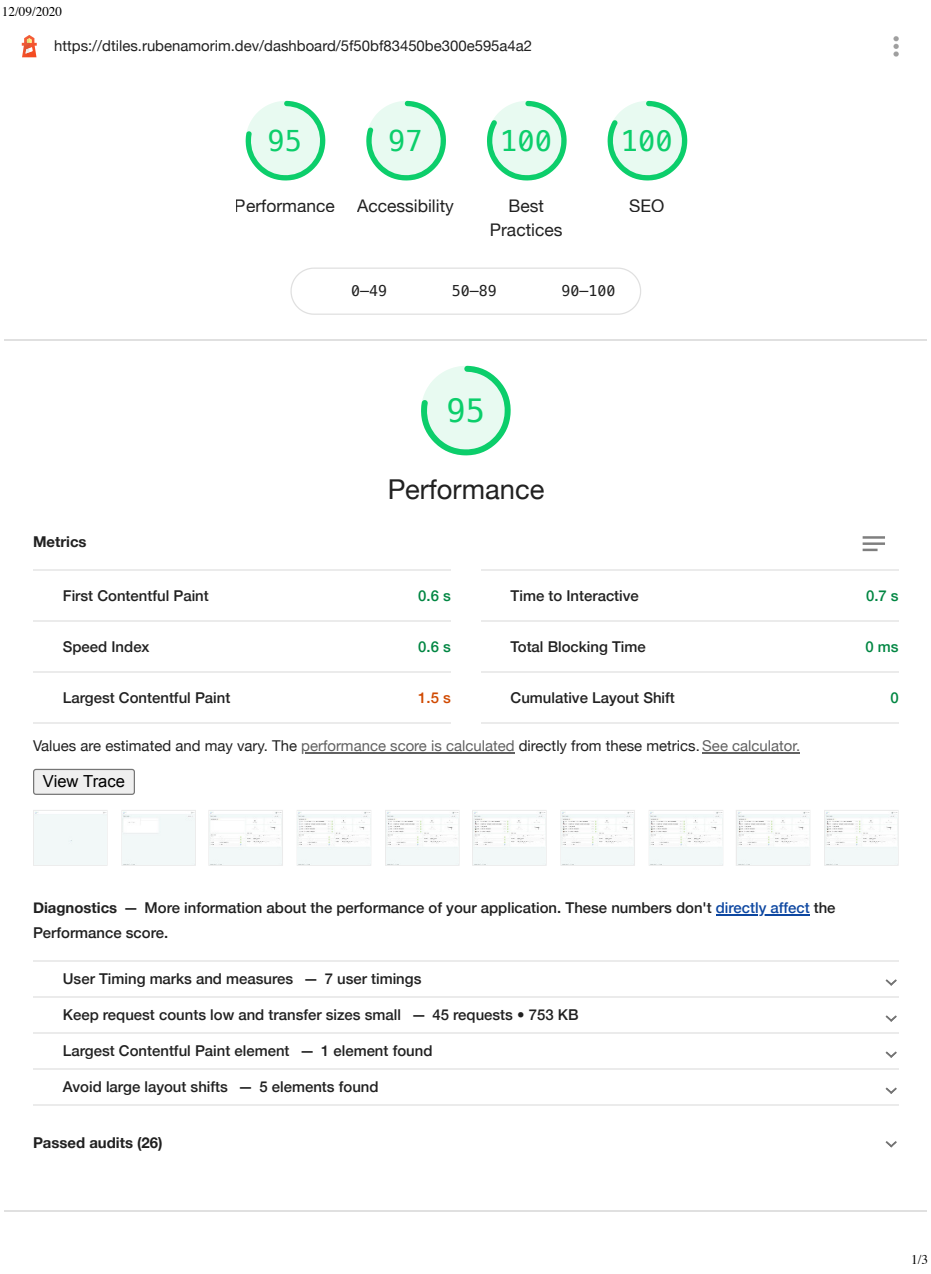
Not applicable (5)

Runtime Settings

URL	https://dtiles.rubenamorim.dev/home
Fetch Time	Sep 12, 2020, 11:39 PM GMT+1
Device	Emulated Desktop
Network throttling	40 ms TCP RTT, 10,240 Kbps throughput (Simulated)
CPU throttling	1x slowdown (Simulated)
Channel	devtools
User agent (host)	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.102 Safari/537.36
User agent (network)	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3963.0 Safari/537.36 Chrome-Lighthouse
CPU/Memory Power	1652

Generated by **Lighthouse** 6.0.0 | File an issue

B.3 Página do dashboard



Accessibility

These checks highlight opportunities to improve the accessibility of your web app. Only a subset of accessibility issues can be automatically detected so manual testing is also encouraged.

Contrast — These are opportunities to improve the legibility of your content.

▲ Background and foreground colors do not have a sufficient contrast ratio.

Additional items to manually check (10) — These items address areas which an automated testing tool cannot cover. Learn more in our guide on conducting an accessibility review.

Passed audits (14)

Not applicable (26)

Best Practices

Passed audits (14)

SEO

These checks ensure that your page is optimized for search engine results ranking. There are additional factors Lighthouse does not check that may affect your search ranking. Learn more.

Additional items to manually check (1) — Run these additional validators on your site to check additional SEO best practices.

Passed audits (8)

Not applicable (5)

Runtime Settings

URL	https://dtiles.rubenamorim.dev/dashboard/5f50bf83450be300e595a4a2
Fetch Time	Sep 12, 2020, 11:41 PM GMT+1
Device	Emulated Desktop
Network throttling	40 ms TCP RTT, 10,240 Kbps throughput (Simulated)
CPU throttling	1x slowdown (Simulated)
Channel	devtools
User agent (host)	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.102 Safari/537.36
User agent (network)	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3963.0 Safari/537.36 Chrome-Lighthouse
CPU/Memory Power	1636

Generated by **Lighthouse** 6.0.0 | File an issue

Anexo C

Questionário

dtiles - Questionnaire

A study of the contribution of dashboards in the software development process is being created in the scope of a master's thesis in Software Engineering for the Scholl of Engineering of Porto Polytechnic Institute(ISEP).

dtiles is the platform created for the context where you can easily create and share some dashboards with your team members.

It's expected that you take a maximum of 5 minutes to complete this questionnaire.

Thank you for your time. Please share this questionnaire with your team members, it's really important for the study 🙏

🌍 http://dtiles.rubenamorim.dev/

Note: This is completely anonymous and the purpose is academic only.

Next

Page 1 of 4

Never submit passwords through Google Forms.

dtiles - Questionnaire

* Required

Introduction

This section aims to gather some information about you.

What's your role? *

- ◯ Developer
- ◯ Quality Assurance
- ◯ Product Owner
- ◯ Other:

Do you work with agile methodologies? *

- ◯ Yes
- ◯ No

Back Next

dtiles - Questionnaire

* Required

Context

This section aims to gather some information about the context under study.

Do you use any tool to monitor the software development process? *

- ◯ Yes
- ◯ No

Does your team use any tools to monitor the software development process?

- ◯ Yes
- ◯ No

If you chose "Yes" in either of the two previous questions, what tools do you use?

Your answer

Do you think the use of dashboards brings any advantage to the software development process? *

- ◯ Yes
- ◯ No
- ◯ Maybe

If you chose "Yes" in the previous question, please specify the advantage(s).

- ☐ Improves team communication
- ☐ Increases awareness of the process
- ☐ Speeds up troubleshooting
- ☐ Improves team organization
- ☐ Improves team focus
- ☐ Other:

Do you think the use of dashboards brings any disadvantage to the software development process? *

- ◯ Yes
- ◯ No
- ◯ Maybe

If you chose "Yes" in the previous question, please specify the disadvantage(s).

- [ ] Worsens team communication
- [ ] Decreases awareness of the process
- [ ] Delay troubleshooting
- [ ] Worsens team organization
- [ ] Worsens team focus
- [ ] Other:

Back Next

dtiles - Questionnaire

* Required

Platform

This section aims to gather some evaluation about the developed platform.

How would you rate the platform in terms of functionality? *

	1	2	3	4	5	
Very Bad	○	○	○	○	○	Very Good

How would you rate the platform in terms of usability? *

	1	2	3	4	5	
Very Bad	○	○	○	○	○	Very Good

How would you rate the platform in terms of performance? *

	1	2	3	4	5	
Very Bad	○	○	○	○	○	Very Good

How would you rate the platform globally? *

	1	2	3	4	5	
Very Bad	○	○	○	○	○	Very Good

Comments and Suggestions

Your answer

Back Submit Page 4 of 4

Anexo D

Questionário - Respostas

dtiles - Questionnaire

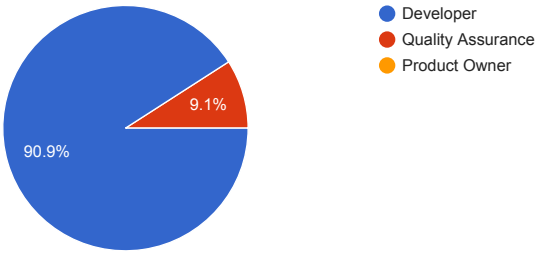
11 responses

Publish analytics

Introduction

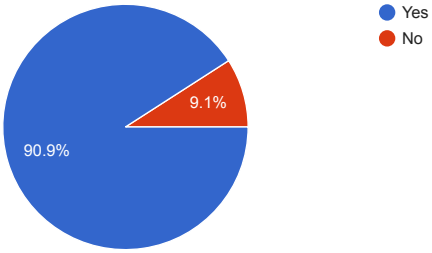
What's your role?

11 responses



Do you work with agile methodologies?

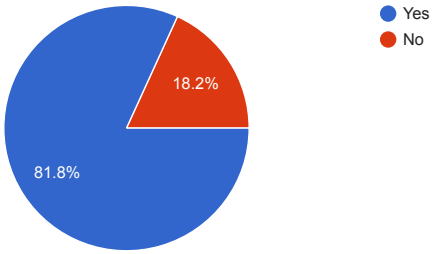
11 responses



Context

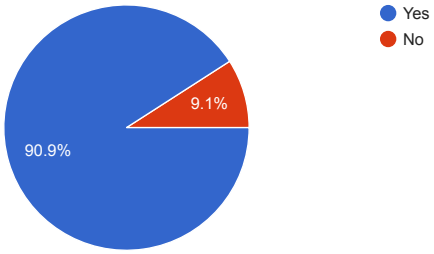
Do you use any tool to monitor the software development process?

11 responses



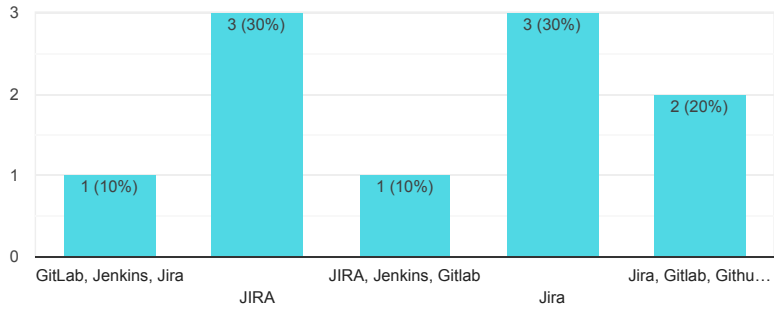
Does your team use any tools to monitor the software development process?

11 responses



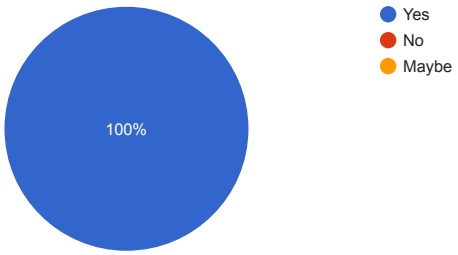
If you chose "Yes" in either of the two previous questions, what tools do you use?

10 responses



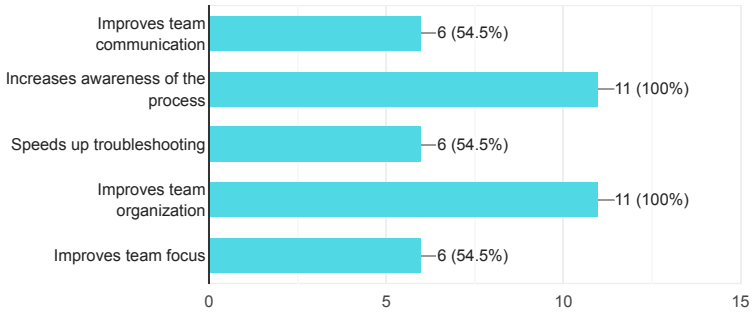
Do you think the use of dashboards brings any advantage to the software development process?

11 responses



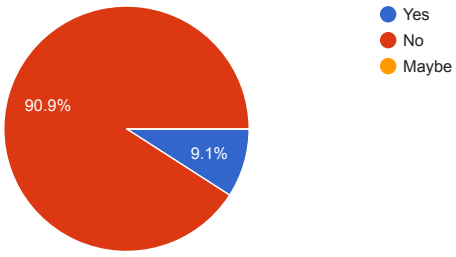
If you chose "Yes" in the previous question, please specify the advantage(s).

11 responses



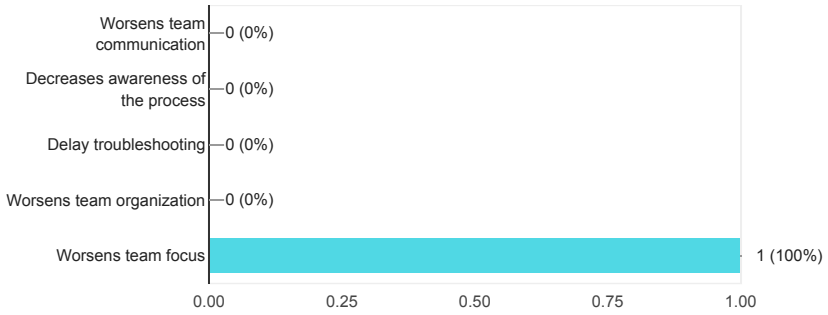
Do you think the use of dashboards brings any disadvantage to the software development process?

11 responses



If you chose "Yes" in the previous question, please specify the disadvantage(s).

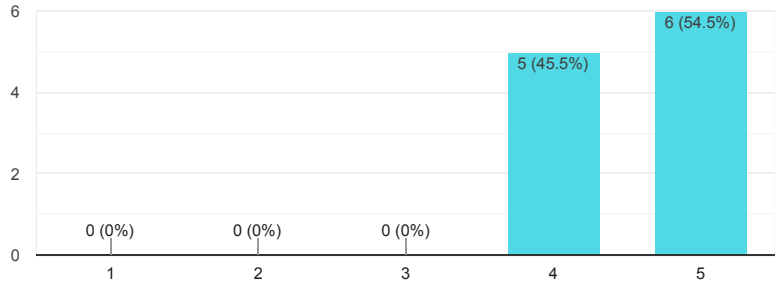
1 response



Platform

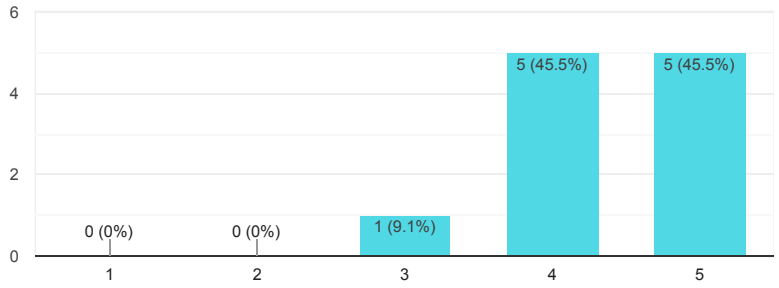
How would you rate the platform in terms of functionality?

11 responses



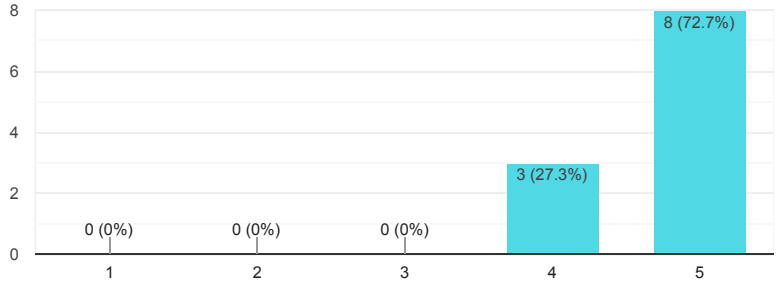
How would you rate the platform in terms of usability?

11 responses



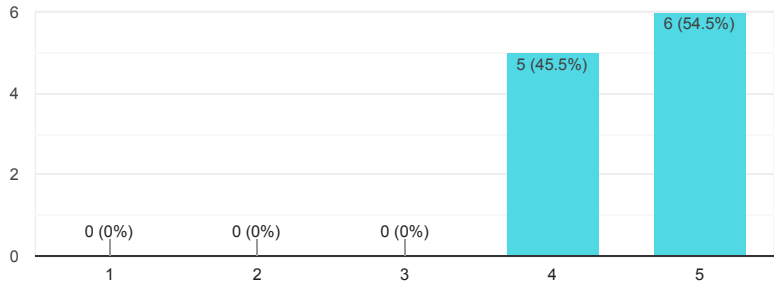
How would you rate the platform in terms of performance?

11 responses



How would you rate the platform globally?

11 responses



Comments and Suggestions

2 responses

Nice job!

Nice work