



Dissertação/Estágio
Mestrado em Engenharia Informática
Engenharia de Software

Project Management Web App

Pedro de Sousa Alves Janeiro

Orientadores
Prof. Tiago Baptista
Sara João

Departamento de Engenharia Informática
Faculdade de Ciências e Tecnologia
Universidade de Coimbra

2016/2017



Dissertação/Estágio
Mestrado em Engenharia Informática
Engenharia de Software

Project Management Web App

Pedro de Sousa Alves Janeiro

Orientadores

Prof. Tiago Baptista

Sara João

Júri Arguente

Prof. Jorge Miguel Sá Silva

Júri Vogal

Prof. Fernando José Barros

Departamento de Engenharia Informática
Faculdade de Ciências e Tecnologia
Universidade de Coimbra

2016/2017

Este documento foi redigido ao abrigo do novo acordo ortográfico.

Resumo

Num mundo cada vez mais tecnológico, é também cada vez maior a necessidade de ferramentas que auxiliem à gestão de projetos e organização de equipas de desenvolvimento de *software*. A complexidade dos projetos de *software* tende a aumentar com o aumento da quantidade de recursos disponíveis, o que apenas dificulta este trabalho de gestão.

O objetivo deste trabalho é, portanto, desenvolver uma ferramenta que auxilie no processo de gestão de projetos de *software*, desde a distribuição e calendarização de tarefas por entre os vários membros, à análise do trabalho desenvolvido pela equipa e recolha do *feedback* dos vários membros da equipa no fim de realizarem cada tarefa.

O projeto foi iniciado com a análise do estado da arte, onde foi estudado o próprio conceito de gestão de projetos, vários produtos e plataformas que também têm como objetivo auxiliar à gestão de projetos, e vários modelos de desenvolvimento e tipos de teste que podem ser aplicados ao projeto e ao produto desenvolvido.

Passando à descrição do sistema, a mesma foi feita com base na apresentação dos requisitos levantados para a plataforma, tanto funcionais como não funcionais. É ainda apresentada a interface pensada para a plataforma, usando para o efeito vários *wireframes* e um diagrama de navegação na plataforma. Por último, é apresentada a arquitetura da plataforma usando um diagrama Entidade Relacionamento, um diagrama de *containers* e um diagrama de componentes.

Para o planeamento do trabalho a realizar, é escolhido o modelo de desenvolvimento a aplicar, e é feita uma calendarização do segundo semestre de trabalho. São ainda apresentadas várias escolhas feitas a nível de diferentes tecnologias e ferramentas a utilizar.

Ao explicar o desenvolvimento da plataforma, foram tidas em conta as várias decisões tomadas durante a implementação dos vários componentes da plataforma. Foi também descrito o ambiente de desenvolvimento utilizado, bem como algumas ferramentas usada para *Continuous Integration* e *deployment* da plataforma.

Por último, foi descrito o plano de testes usado para garantir que os vários requisitos da plataforma são cumpridos.

Keywords: Gestão de Projetos, Engenharia de *Software*

Abstract

In a World increasingly technological, the need for tools that aid in the process of project management and organization of software development teams is also raising. The complexity of software projects tends to grow with the increase of available resources, which only complicates this management process.

The purpose of this project is, therefore, to develop a tool that aids in the process of software project management, from the distribution of tasks among members, to the analysis of the work developed by the team and collection of feedback from the various team members upon completing each task.

The project started with the analysis of the state of the art, where the concept of project management itself was studied, along with some products and platforms with the same purpose of aiding project management, and various software development models and types of tests that can be applied to the project and the resulting product.

Moving on to the description of the system, it was based on the presentation of the many requirements collected for the platform, either functional or non functional. The interface expected for the platform is also presented, with the use of several wireframes and of a navigation diagram for the platform. Lastly, the architecture of the system is described, using an Entity Relationship diagram, a containers diagram and a components diagram.

In the moment of planning the work to be done, a development model is chosen, and a scheduling for the second semester is presented. Some choices regarding different technologies and tools to use during development are also made.

Explaining the development of the platform, follows the listing of several decisions made during implementation of its several components. The development environment is also described, as well as some tools used for Continuous Integration and deployment of the platform.

In the end, the test plan used is presented, used to make sure the many requirements for the platform are met.

Keywords: *Project Management, Software Engineering*

Índice

Resumo	v
Abstract	vii
Capítulo 1: Introdução	1
1.1 A HYP	1
1.2 Contexto	1
1.3 Motivação	2
1.4 Objetivos	2
1.5 Descrição do relatório	3
Capítulo 2: Estado da Arte	5
2.1 Gestão de Projetos	5
2.2 Outras plataformas	7
2.2.1 Redmine	8
2.2.2 Microsoft Project	9
2.2.3 Trello	10
2.2.4 JIRA <i>Software</i>	12
2.2.5 Slack	12
2.2.6 Yammer	14
2.2.7 Toggl	15

2.2.8	TrackingTime	17
2.2.9	Lattice Check-ins	18
2.2.10	Podio	19
2.2.11	Comparação	20
2.2.12	Discussão e conclusões	22
2.3	Modelos de desenvolvimento de <i>Software</i>	23
2.3.1	<i>Waterfall</i>	23
2.3.1.1	Variações	25
2.3.2	<i>Agile</i>	25
2.3.3	Espiral	27
2.4	Tipos de testes	30
2.4.1	Testes unitários	30
2.4.1.1	Testes <i>Black Box</i>	31
2.4.1.2	Testes <i>White Box</i>	31
2.4.1.3	Testes <i>Gray box</i>	31
2.4.2	Testes de integração	31
2.4.2.1	Integração <i>Big Bang</i>	32
2.4.2.2	Integração <i>Bottom Up</i>	32
2.4.2.3	Integração <i>Top Down</i>	33
2.4.3	Testes de sistema	33
2.4.3.1	Testes de usabilidade	33
2.4.3.2	Testes de escalabilidade	34
2.4.3.3	Testes de <i>stress</i>	34
2.5	Conclusão	34
Capítulo 3: Descrição do Sistema		37

3.1	Requisitos	37
3.1.1	Requisitos funcionais	37
3.1.1.1	Informação	38
3.1.1.2	Conta	38
3.1.1.3	Equipa	39
3.1.1.4	Projeto	40
3.1.1.5	Tarefa	43
3.1.1.6	<i>Time entry</i>	45
3.1.1.7	<i>Check-in</i>	47
3.1.1.8	Questionário	48
3.1.1.9	Notificação	49
3.1.1.10	Relatório	52
3.1.2	Requisitos não funcionais	54
3.1.2.1	Usabilidade	54
3.1.2.2	Variabilidade	54
3.1.2.3	Segurança	55
3.1.2.4	Desempenho	55
3.1.2.5	Portabilidade	55
3.1.2.6	Compatibilidade	55
3.1.3	Restrições	56
3.2	Arquitetura do sistema	57
3.2.1	Diagrama Entidade Relacionamento	57
3.2.2	Diagrama de <i>Containers</i>	59
3.2.3	Diagrama de Componentes	60
3.2.4	Diagrama de Navegação	61

3.3	Conclusão	62
Capítulo 4: Planeamento		63
4.1	Modelo de desenvolvimento	63
4.2	Calendarização	65
4.2.1	Primeiro Semestre	65
4.2.2	Segundo Semestre	65
4.3	Escolhas tecnológicas	67
4.3.1	<i>Git</i>	67
4.3.2	<i>Ruby on Rails</i>	67
4.3.3	<i>PostgreSQL</i>	68
4.4	Ferramentas escolhidas	69
4.4.1	Gitlab	69
4.4.2	<i>ShareLaTeX</i>	70
4.4.3	<i>Trello</i>	70
4.4.4	<i>Heroku</i>	71
4.5	Conclusão	71
Capítulo 5: Desenvolvimento		73
5.1	Gestão de utilizadores e equipas	73
5.2	Gestão de projetos	76
5.3	Gestão de tarefas e <i>time entries</i>	78
5.4	Gestão de notificações	80
5.5	Gestão de <i>check-ins</i> e questionários	82
5.6	Gestão de relatórios	83
5.7	Ambiente de desenvolvimento	84

5.8	<i>Continuous Integration</i>	87
5.8.1	Configuração do projeto no <i>Gitlab</i>	87
5.8.2	<i>Brakeman</i>	87
5.8.3	<i>rubycritic</i>	87
5.8.4	<i>RuboCop</i>	87
5.8.5	<i>rails_best_practices</i>	88
5.9	Conclusão	90
Capítulo 6: Testes		91
6.1	Testes funcionais	91
6.1.1	Utilizadores e equipas	91
6.1.2	Projetos	92
6.1.3	Tarefas e <i>time entries</i>	92
6.1.4	Notificações	93
6.1.5	<i>Check-ins</i> e questionários	93
6.1.6	Relatórios	93
6.2	Conclusão	94
Capítulo 7: Conclusão		95
7.1	Trabalho futuro	95
7.2	Testes não funcionais	95
7.3	<i>Deployment</i>	95
7.4	Manutenção	96
7.5	Considerações finais	97
Bibliografia		99

Apêndice A: *Wireframes* 101

Lista de Figuras

2.1	Interface da aplicação Redmine	9
2.2	Interface do Microsoft Project, mostrando um diagrama de Gantt gerado pela aplicação	10
2.3	Interface do Trello	12
2.4	Interface do JIRA Software	13
2.5	Interface do Slack	14
2.6	Interface do Yammer	15
2.7	Interface de edição de entradas de tempo do Toggl	16
2.8	Interface de geração de relatórios do Toggl	16
2.9	Interface da aplicação TrackingTime	17
2.10	Interface do Lattice Check-ins	18
2.11	Interface da aplicação Podio, com várias aplicações associadas	20
3.1	Diagrama ER da base de dados	57
3.2	Diagrama de <i>Containers</i>	59
3.3	Diagrama de componentes	60
3.4	Diagrama de Navegação	61
4.1	Diagrama de Gantt com calendarização do segundo semestre	66
4.2	Configuração da ferramenta de <i>Continuous Integration</i>	69
5.1	<i>Screenshot</i> - Registo na aplicação	74

5.2	<i>Screenshot</i> - Login na aplicação	75
5.3	<i>Screenshot</i> - Edição de perfil	75
5.4	<i>Screenshot</i> - Equipas	75
5.5	<i>Screenshot</i> - Projetos	77
5.6	<i>Screenshot</i> - Editar projeto	77
5.7	<i>Screenshot</i> - Tarefas	79
5.8	<i>Screenshot</i> - Criar tarefa	79
5.9	<i>Screenshot</i> - <i>Time entries</i>	79
5.10	<i>Screenshot</i> - Notificações	81
5.11	<i>Screenshot</i> - Questionário	82
5.12	<i>Screenshot</i> - Relatório	83
5.13	Configuração usada para o <i>container</i> de <i>docker</i> com o servidor de <i>Ruby on Rails</i>	85
5.14	Configuração usada para o sistema do <i>docker-compose</i> . É usado o <i>container</i> gerado com a configuração da figura 5.13	86
5.15	Relatório gerado pelo <i>rubycritic</i>	88
5.16	Relatório gerado pelo <i>RuboCop</i>	88
5.17	Resultado da execução de <i>rails_best_practices</i>	89
A.1	<i>Wireframe</i> - Login	101
A.2	<i>Wireframe</i> - Check-in	102
A.3	<i>Wireframe</i> - Check-out	102
A.4	<i>Wireframe</i> - Lista de projetos	103
A.5	<i>Wireframe</i> - Informação de um projeto	103
A.6	<i>Wireframe</i> - Lista de tarefas	104
A.7	<i>Wireframe</i> - Tarefa	104
A.8	<i>Wireframe</i> - Lista de <i>time entries</i>	105

A.9 <i>Wireframe</i> - Geração de relatórios	105
A.10 <i>Wireframe</i> - Notificação	106

Lista de Tabelas

2.1	Comparação de plataformas usadas em gestão de projetos	21
2.2	Utilização das plataformas nas várias etapas de Gestão de Projetos	21
2.3	Vantagens e desvantagens do modelo <i>Waterfall</i>	24
2.4	Vantagens e desvantagens do modelo <i>Agile</i>	27
2.5	Vantagens e desvantagens do modelo em Espiral	29
3.1	Duração prevista para a realização de cada tarefa na plataforma	54
3.2	Tipos de participação de um utilizador num projeto	56
4.1	Modelo de <i>branches</i> do repositório <i>Git</i>	67
5.1	Métodos e <i>routes</i> criados para listar tarefas segundo várias regras diferentes	78
5.2	Estado de cada componente da plataforma no fim do estágio	90

Glossário

A

API *Application programming interface.* 8, 13, 14, 16, 17, 20

B

BD Base de Dados. 62

C

CEO *Chief executive officer.* 18

CI *Continuous Integration.* v, xv, 3, 69, 87, 97

E

ER Entidade Relacionamento. v, 57, 62, 76

ES Engenharia de *Software.* vi

F

FAQ *Frequently Asked Questions.* 38

G

GP Gestão de Projetos. vi, xix, 5, 20, 21, 27, 97

I

IPN Instituto Pedro Nunes. 1

M

MVC *Model-View-Controller.* 68

P

PaaS *Platform as a Service.* 71

PDF *Portable Document Format.* 53

PM *Project Management.* vii

PMI *Project Management Institute.* 5

PNG *Portable Network Graphics*. 53

R

RoR *Ruby on Rails*. xvi, 59, 85, 87, 88

S

SE *Software Engineering*. vii

SGBD *Sistema de Gestão de Bases de Dados*. 56, 67, 68, 71, 84

SRS *Software Requirements Specification*. 23, 26

T

TCP *Transmission Control Protocol*. 80

U

UI *User Interaction*. 1

US *User Story*. 38–53

UX *User Experience*. 1

V

VCS *Version Control System*. 69

X

XSS *Cross-Site Scripting*. 87

Capítulo 1

Introdução

Apesar da necessidade de ferramentas que auxiliem à gestão de projetos de *software* existir desde cedo, o contínuo crescimento da prática do desenvolvimento de *software* levou a que esta necessidade crescesse também. Existem várias ferramentas no mercado com este mesmo propósito, mas todas com limitações ou com âmbitos demasiado restritos. Dessa forma, surgiu a necessidade de levar a cabo este projeto e desenvolver este novo produto.

1.1 A HYP

A HYP¹ é uma empresa de desenvolvimento de *software* de Coimbra, sediada no Instituto Pedro Nunes². Fundada em janeiro de 2015, a HYP tem como principais objetivos resolver problemas e ajudar os clientes a alcançar o sucesso.

Conta para isso com uma equipa com experiência nas áreas de *Design*, desenvolvimento *Web* e *Mobile* e *UI/UX*. Conta com vários produtos no seu portfolio, desenvolvidos para clientes nacionais e internacionais, que não hesitam em dar o seu *feedback* positivo.

1.2 Contexto

Como empresa de desenvolvimento de *software* que é, também a equipa da HYP sente esta necessidade de uma ferramenta que auxilie à gestão dos seus projetos. Apesar de já usarem várias ferramentas para o propósito, algumas das quais analisadas no capítulo 2 deste documento, consideram que não existe uma ferramenta que agregue todas as funcionalidades necessárias de forma satisfatória, mantendo uma interface eficaz e simples de usar.

Desta necessidade, surge a ideia que deu origem a este projeto, de criar uma nova

¹Para mais informações consultar <http://hypsoftware.com/>

²Para mais informações consultar <https://www.ipn.pt/>

ferramenta de gestão de projetos, proposta como dissertação de mestrado pela HYP, na pessoa da Sara João.

1.3 Motivação

Para suprir as necessidades de qualquer equipa de desenvolvimento de *software*, será produzida uma aplicação web que possa ser acedida a partir de qualquer lugar e em qualquer dispositivo, que ofereça às equipas um leque de funcionalidades que auxilie à gestão dos seus projetos sem esforço e de forma centralizada.

1.4 Objetivos

O principal objetivo deste projeto é o desenvolvimento de uma aplicação de gestão de projetos de *software* para a HYP.

Esta aplicação deve auxiliar não só à calendarização e atribuição de várias tarefas, mas também à recolha de *feedback* dos membros da equipa de desenvolvimento por parte do gestor de projetos, e análise de várias métricas referentes ao estado do projeto.

Espera-se que os utilizadores da plataforma possam realizar as seguintes tarefas:

- Criar equipas de desenvolvimento, compostas pelos próprios utilizadores da plataforma.
- Criar projetos de *software*, que podem ou não estar associados a uma equipa de desenvolvimento (pode tratar-se de um projeto pessoal).
- Criar tarefas a realizar para um projeto, que serão atribuídas aos utilizadores que estão a trabalhar nesse projeto.
- Criar *time entries*, de forma a contabilizar o tempo gasto a trabalhar em cada tarefa.
- Preencher um pequeno inquérito no fim de cada sessão de trabalho, de forma a dar *feedback* ao gestor do projeto.
- Gerar relatórios que permitam avaliar o progresso de um projeto, em termos do número de tarefas finalizadas pela equipa.

A aplicação final será colocada *online* e disponibilizada para o público em geral, de forma a poder ser utilizada por qualquer equipa de desenvolvimento de *software*.

1.5 Descrição do relatório

O presente documento encontra-se atualmente dividido nos capítulos do Estado da Arte (capítulo 2), Descrição do Sistema (capítulo 3) e Planeamento (capítulo 4).

No capítulo do estado da arte aborda-se o conceito de Gestão de Projetos, sendo de seguida analisadas várias plataformas e ferramentas existentes atualmente no mercado e que auxiliem nesta tarefa. São ainda estudados vários modelos de desenvolvimento de *software* e vários tipos de teste a que a aplicação pode ser sujeita.

Passando ao capítulo da descrição do sistema, são apresentados os vários requisitos levantados para o sistema. É apresentada a interface pensada para a plataforma, com o uso de vários *wireframes* e de um diagrama de navegação na mesma. Por último, é apresentada a arquitetura da plataforma, usando um diagrama Entidade Relacionamento, um diagrama de *containers* e um diagrama de componentes.

No capítulo do planeamento, é apresentada a calendarização prevista para o segundo semestre de desenvolvimento do projeto, bem como o modelo de desenvolvimento a usar e algumas decisões tomadas relativamente a tecnologias e ferramentas a usar.

Para o capítulo do desenvolvimento são listadas várias decisões tomadas durante a implementação dos vários componentes da plataforma, bem como algumas ferramentas utilizadas para *Continuous Integration* e *deployment* do produto desenvolvido.

No capítulo de testes, é apresentado o plano de testes usado para garantir que os vários requisitos da plataforma são cumpridos.

Capítulo 2

Estado da Arte

No capítulo do estado da arte, será inicialmente considerado o que é esperado atualmente de um bom processo de gestão de projetos (especificamente, projetos de desenvolvimento de *software*), estudando algumas plataformas atualmente existentes que pretendem auxiliar neste processo, e comparando-as.

Relativamente ao artefacto que irá ser desenvolvido como produto deste estágio, serão inicialmente estudadas as várias opções atualmente utilizadas em termos de modelos de desenvolvimento de *software*, e vários tipos de teste que poderão ser levados a cabo para determinar a qualidade do produto.

2.1 Gestão de Projetos

Gestão de Projetos é definida pelo *Project Management Institute* (PMI)¹ como "*a aplicação do conhecimento, habilidades, ferramentas e técnicas às atividades de um projeto para cumprir os seus requisitos*"[1].

Já Harold R Kerzner² define Gestão de Projetos como "*o planeamento, organização, supervisão e controlo dos recursos de uma empresa para atingir um objetivo a curto prazo que foi estabelecido para completar metas e objetivos específicos. Além disso, a gestão de projetos utiliza uma abordagem sistemática, com elementos especializados atribuídos a projetos específicos.*"[2]

Uma boa gestão é fundamental para o sucesso de qualquer projeto de *software*. Segundo Kerzner, implica a aplicação de vários processos de gestão, agrupados em cinco etapas fundamentais[2]:

1. Iniciação do projeto

- Selecionar um projeto dados os recursos disponíveis

¹Para mais informações consultar <https://www.pmi.org/>

²Para mais informações consultar <https://www.bw.edu/academics/bios/kerzner-harold>

- Reconhecer os benefícios do projeto
- Preparar os documentos para aprovação do projeto
- Nomear o gestor de projeto

2. Planeamento do projeto

- Definir os requisitos
- Definir a qualidade e quantidade de trabalho
- Definir os recursos necessários
- Calendarizar atividades
- Analisar os vários riscos

3. Execução do projeto

- Escolher os membros da equipa
- Supervisionar e gerir o trabalho
- Trabalhar com os membros da equipa e ajudá-los a melhorar

4. Monitorização e controlo do projeto

- Acompanhar o progresso
- Comparar os resultados obtidos com os esperados
- Analisar variações e impactos
- Fazer ajustes

5. Encerramento do projeto

- Verificar se todo o trabalho foi concluído
- Encerrar o projeto contratualmente
- Encerrar o projeto financeiramente
- Encerrar o projeto administrativamente

Dada a quantidade de ferramentas disponíveis para as fases de iniciação e planeamento do projeto, e a especificidade inerente à fase de encerramento do mesmo, o foco deste relatório encontra-se nas fases de execução, monitorização e controlo do projeto.

2.2 Outras plataformas

Para o planeamento deste projeto, foram estudadas as seguintes plataformas (agrupadas por objetivo da plataforma):

Gestão de projetos

1. Redmine³
2. Microsoft Project⁴

Gestão de *issues*

3. Trello⁵
4. JIRA Software⁶

Comunicação

5. Slack⁷
6. Yammer⁸

Time tracking

7. Toggl⁹
8. TrackingTime¹⁰

Recolha de *feedback*

9. Lattice Check-ins¹¹

Multi-objetivos

³Para mais informações consultar <http://www.redmine.org/>

⁴Para mais informações consultar <https://products.office.com/pt-PT/project>

⁵Para mais informações consultar <https://trello.com/>

⁶Para mais informações consultar <https://www.atlassian.com/software/jira>

⁷Para mais informações consultar <https://slack.com/>

⁸Para mais informações consultar <https://www.yammer.com/>

⁹Para mais informações consultar <https://toggl.com/>

¹⁰Para mais informações consultar <https://trackingtime.co/>

¹¹Para mais informações consultar <https://latticehq.com/check-ins/>

10. Podio¹²

Após uma breve análise, é possível perceber que as aplicações listadas não servem todas os mesmos propósitos, havendo várias *scopes* presentes. Podemos identificar ferramentas específicas de comunicação (Slack e Yammer) e gestão de tempo (Toggl e TrackingTime), bem como ferramentas mais abrangentes que cobrem várias áreas da gestão de projetos (JIRA Software e Redmine).

No entanto, e apesar de se pretender produzir, no contexto deste estágio curricular, uma ferramenta que possibilite o apoio à execução, monitorização e controlo do projeto, é importante considerar também ferramentas que apenas cubram áreas mais específicas, pois poderão ter funcionalidades semelhantes, como um todo, a funcionalidades integrantes do artefacto.

2.2.1 Redmine

O Redmine apresenta-se como uma ferramenta utilizável não só para gestão de projetos mas também para controlo de *issues*[3]. No momento da redação deste relatório, encontra-se no topo do *ranking* de aplicações *open-source* de gestão de projetos[4].

Para cada projeto, é possível criar *wikis*¹³ e fóruns, é possível gerir o tempo gasto por cada utilizador (à semelhança do Toggl e do TrackingTime), e é possível gerir o acesso dos utilizadores às várias funcionalidades através de um sistema de *roles* dentro da equipa, com vários níveis de permissão.

O Redmine tem ainda associado um calendário, e permite a criação de diagramas de Gantt, para melhor gerir as *deadlines* associadas a um projeto.

Uma diferença do Redmine relativamente a outras aplicações aqui apresentadas é a integração com vários sistemas de controlo de versões, como o Git, com possibilidade de navegação no sistema de ficheiros associado e comparação de ficheiros. Além disso, o *website* do Redmine apenas contém um projeto de demonstração (o próprio Redmine), sendo necessário instalar o Redmine localmente para o usar com projetos pessoais.

Após instalar e configurar o Redmine (trata-se de uma aplicação em Ruby on Rails, com flexibilidade quanto à base de dados utilizada), é possível criar projetos, com utilizadores e tarefas associados, bem como criar *issues* relativas a eventuais *bugs* no código, por exemplo. As *issues* criadas podem ter vários níveis de prioridade, de forma a melhorar a sua organização e planear a sua resolução.

O Redmine tem uma API RESTFUL associada¹⁴, com *endpoints* para controlo de projetos, utilizadores, *time entries*, *issues* e páginas de *wiki*, entre outros.

¹²Para mais informações consultar <https://podio.com/>

¹³Para mais informações consultar <https://en.wikipedia.org/wiki/Wiki>

¹⁴Para mais informações consultar http://www.redmine.org/projects/redmine/wiki/Rest_api

The screenshot shows the Redmine web application interface. At the top, there is a navigation bar with links for Home, My page, Projects, Administration, and Help. The user is logged in as 'admin' and can access their account or sign out. The main header displays 'My project' and a search bar. Below this is a secondary navigation bar with tabs for Overview, Activity, Roadmap, Issues, News, Documents, Wiki, Files, Repository, and Settings. The 'Issues' section is active, showing a list of issues with columns for #, Tracker, Status, Priority, Subject, Assigned to, and Updated. A context menu is open over issue #79, offering actions like Edit, Status, Priority, Assigned to, Copy, Move, and Delete. The 'Priority' submenu is expanded, showing options: Immediate, Urgent, High, Normal, and Low (which is selected). On the right side, there are sections for 'New issue' (with a Tracker dropdown), 'Issues' (with links for View all issues, Summary, and Change log), and 'Custom queries' (with links for Assigned to me, Due this week, and Late features).

#	Tracker	Status	Priority	Subject	Assigned to	Updated
127	Bug	New	Normal	Ticket with attachments		12/22/2007 12:11 PM
116	Bug	New	Low	Keep playing audio when rw/ff and preserve pitch.	John Smith	12/17/2007 09:56 PM
88	Feature	Assigned	Low	HTTP Challenge-MD5 authentication		12/22/2007 04:33 PM
83	Feature	Assigned	Low	Export the parameters of an input	John Smith	12/17/2007 09:56 PM
82	Feature	New	Low	Formatted text rendering support	Dave Loper	12/17/2007 06:58 PM
81	Feature	New	Normal	DVTS support		12/17/2007 06:58 PM
79	Feature	New	Low	QuickTime capturing		12/17/2007 06:58 PM
78	Feature	New	Low	Full H323 videoconferencing		12/17/2007 06:58 PM
77	Feature	Assigned	Low	Closed captions / Teletext support		12/17/2007 06:58 PM
74	Feature	New	Low	Progressive download playing		12/17/2007 06:58 PM
73	Feature	New	Low	Dshow tuning support		12/17/2007 06:58 PM
72	Feature	New	Low	V4L tuning support		12/17/2007 06:58 PM
70	Feature	New	Low	Electric Program Guide		12/17/2007 06:58 PM
69	Bug	New	Low	SDL vout cleaning		12/17/2007 06:58 PM
65	Feature	New	Low	Protocol rollover support		12/17/2007 06:58 PM
64	Feature	New	Normal	Improve ZLM functionality		12/22/2007 04:33 PM
63	Feature	New	Low	Gstreamer and Helix integration		12/17/2007 06:58 PM
62	Feature	New	Low	Gnutella servlet		12/17/2007 06:58 PM
59	Feature	New	Low	Finalization of Pocket PC port		12/17/2007 06:58 PM
58	Bug	Assigned	Low	Re-write of the AppleScript bindings		12/22/2007 04:33 PM
57	Feature	New	Low	MacOS X SVCD support	Dave Loper	12/17/2007 06:58 PM
51	Bug	New	Low	Better Mozilla plugin control		12/17/2007 06:58 PM

Figura 2.1: Interface da aplicação Redmine

2.2.2 Microsoft Project

O Microsoft Project[5] foi lançado pela Microsoft em 1984, com o objetivo de ajudar gestores de projeto a desenvolver planos de ação, atribuir recursos, acompanhar progresso, gerir orçamentos, e analisar cargas de trabalho.

De forma a calcular o orçamento associado a um projeto, o Project usa os valores associados aos recursos atribuídos a cada tarefa e a estimativa de tempo de conclusão desta, e gera uma estimativa que pode ser aplicada a todo o projeto e usada em relatórios gerados pela aplicação.

No Project, membros da equipa, equipamento e materiais são diferentes tipos de recurso, que podem ser utilizados independentemente em vários projetos diferentes, e têm um calendário associado, de forma a melhor determinar quando cada um está disponível. Dado que a aplicação consegue determinar quando cada recurso está ocupado, é possível determinar caminhos críticos na calendarização, gerando para o efeito diagramas de Gantt que são disponibilizados ao gestor do projeto.

O Project oferece várias funcionalidades relativamente à geração de relatórios, sendo esta uma área em que a Microsoft sempre foi evoluindo nas várias versões da ferramenta.

Ao contrário das outras ferramentas estudadas até agora, o Microsoft Project apenas está disponível sobre a forma de uma aplicação *desktop*, para sistemas Windows. Existem duas versões, *Standard* e *Professional*, tendo esta última mais funcionalidades disponíveis no campo da colaboração entre membros da equipa.

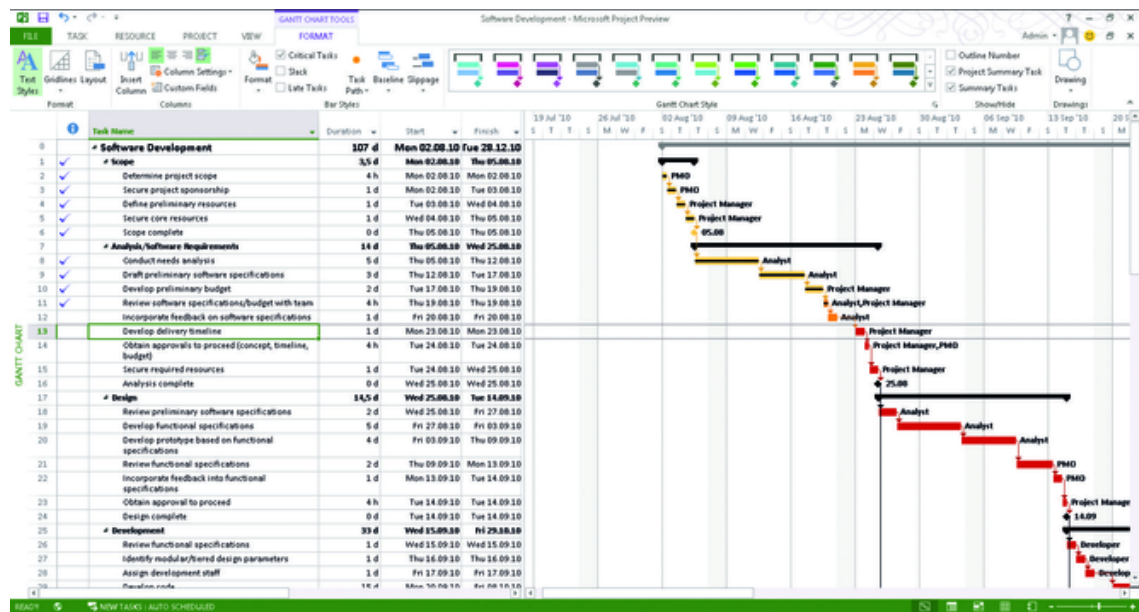


Figura 2.2: Interface do Microsoft Project, mostrando um diagrama de Gantt gerado pela aplicação

2.2.3 Trello

O Trello apresenta-se como uma ferramenta de gestão de projetos *web-based*.

Usa o modelo de Kanban¹⁵ para gestão de projetos: diferentes projetos são representados por diferentes quadros (*boards*), cada quadro com várias listas de tarefas. Nestas listas, cada tarefa é identificada por um cartão (*card*), que pode circular entre várias listas de forma a simular o percurso de uma funcionalidade da ideia à implementação. Se for necessário aumentar o nível de especificidade apresentado, cada cartão pode ter uma *checklist* associada, com vários objetivos menores que a equipa considere que não necessitem de um cartão próprio. Cada cartão no Trello pode ter um ou mais membros da equipa associados, que receberão alertas sempre que for feita alguma alteração ao seu conteúdo.

Como exemplo, uma organização possível de listas de tarefas no Trello envolve:

1. *Backlog* - funcionalidades a implementar; um *developer* que pretenda começar a trabalhar deve escolher um cartão desta lista
2. *In Progress* - após selecionar um *card* da lista *Backlog*, o *developer* deve movimentá-lo para esta lista, indicando que já alguém se encontra a implementar esta funcionalidade. Esta lista permite ter uma ideia do que cada elemento da equipa está a fazer num dado momento
3. *Code Review* - quando a funcionalidade estiver implementada, o cartão deve ser movido para esta lista, que deve ser consultada periodicamente. O código associado às funcionalidades nesta lista deve ser revisto, para garantir que segue as *guidelines* da empresa, por exemplo

¹⁵Para mais informações consultar <https://www.atlassian.com/agile/kanban>

4. *Quality Assurance* - se o código tiver sido aceite na etapa anterior, deve ser testado e aprovado pelo cliente. Idealmente, se houver alguma falha associada ao código, deve ser criado um novo cartão de forma a não alterar muito o *flow* usado ao mover um cartão de volta para uma lista anterior
5. *Completed/Deployed* - após ser revisto e testado, o código implementado entretanto deve ser disponibilizado, mantendo-se nesta lista para criar um histórico das funcionalidades implementadas

No entanto, o Trello oferece liberdade a cada equipa de criar as listas de tarefas que achar necessárias, sendo por isso bastante flexível e adaptável a equipas com vários tamanhos e metodologias de trabalho diferentes. De facto, a equipa do Trello mantém na sua plataforma uma página com várias sugestões de *layouts*, de acordo com os objetivos da equipa¹⁶.

É possível personalizar cada cartão no Trello com etiquetas coloridas, que podem estar associadas a significados diferentes. Por exemplo, é possível associar uma cor a funcionalidades a implementar, uma segunda cor a testes a realizar, e ainda uma cor apenas para trabalho de configuração do servidor.

O Trello tem ainda funcionalidades para adição de comentários e anexos em cada cartão, permitindo a discussão sobre o conteúdo do mesmo sem ser necessária a migração para outra plataforma. Para uma melhor gestão de prazos, cada cartão pode ter uma *due date*, sendo que qualquer membro da equipa associado ao cartão será notificado quando a mesma for atingida.

Para além da aplicação *web*, o Trello tem também aplicação oficial para iPhone/iPad e Android. Existe ainda uma aplicação não oficial para Windows.

Dadas estas características, o Trello apresenta-se como uma ferramenta útil para gestão de *issues* e até calendarização. No entanto, não permite gestão de recursos ou orçamentos, nem *time tracking*.

Aquando da redação deste relatório, e segundo o próprio website do Trello [6], são várias as empresas internacionais da área da tecnologia que usam a plataforma, como por exemplo a Google, Adobe, Pixar, Kickstarter e Paypal.

Para além de todas estas características, o Trello tem também uma API pública, que permite o controlo das *boards* e dos seus constituintes através de aplicações externas (mediante autenticação válida). Isto torna mais fácil estabelecer um paralelismo entre os cartões do Trello e cartões de uma plataforma independente, para implementar por exemplo uma aplicação de gestão de issues, que utilize simultaneamente a API do Trello para controlo dos cartões e a API de um repositório de Git como o Github¹⁷ para gerir as *branches* associadas aos mesmos.

¹⁶Para mais informações consultar <https://trello.com/inspiration>

¹⁷Para mais informações consultar <https://developer.github.com/v3/>

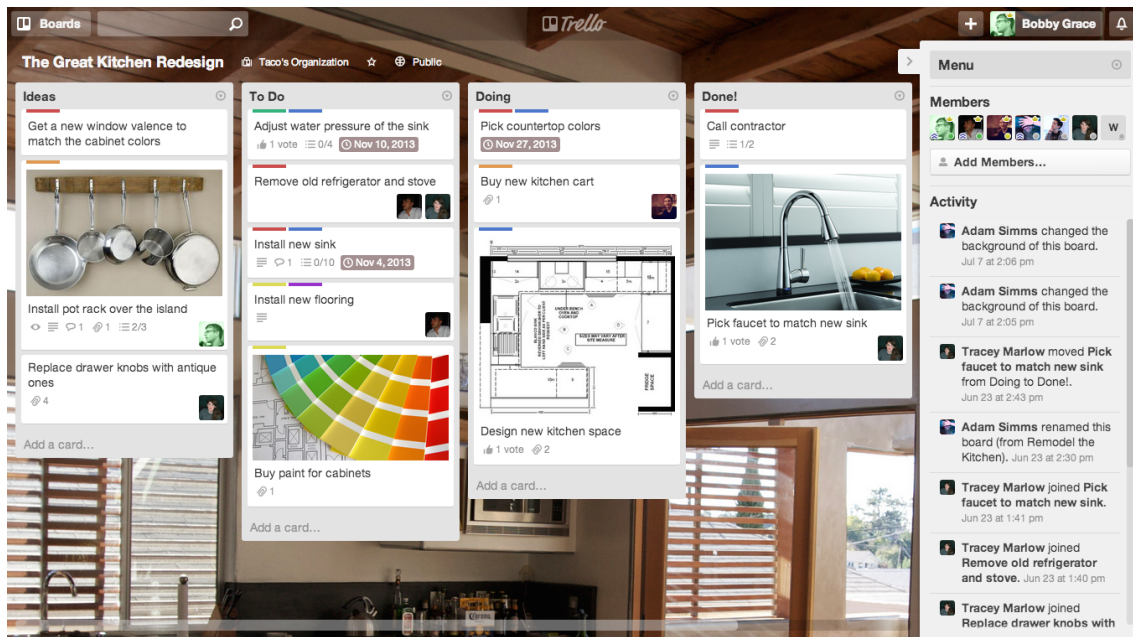


Figura 2.3: Interface do Trello

2.2.4 JIRA Software

O JIRA apresenta-se como uma ferramenta de controlo de *issues*, à semelhança do Trello[7].

Ao contrário do Trello, o JIRA não é limitado ao modelo Kanban, dando de facto prioridade a um modelo *Agile* para o desenvolvimento de *software*. O JIRA inclui integração com o Git, o que auxilia no controlo das *issues*, por poder associá-las diretamente a secções do código produzido.

Limitando-se à aplicação *web*, o JIRA tem no entanto duas possibilidades para o *hosting* da aplicação, entre os seus servidores e servidores pessoais, oferecendo um período de *trial* na segunda opção para as equipas poderem testar livremente o *software* antes de adquirir uma licença.

2.2.5 Slack

O Slack é uma ferramenta de comunicação para equipas[8]. Slack é acrónimo para "*Searchable Log of All Conversation and Knowledge*"[9].

No Slack é possível criar canais (*channels*), abertos a toda a equipa ou privados e apenas visíveis para alguns membros, onde os membros podem comunicar livremente entre si. Os canais podem ser criados para qualquer propósito (um projeto, uma equipa, uma *feature*), e qualquer membro de um canal pode convidar novos membros para o mesmo.

O Slack oferece também um sistema de mensagens diretas, em que cada utilizador pode falar diretamente com outro utilizador ou com um grupo mais restrito de utilizadores sobre temas mais específicos ou que não justifiquem a criação de um canal.

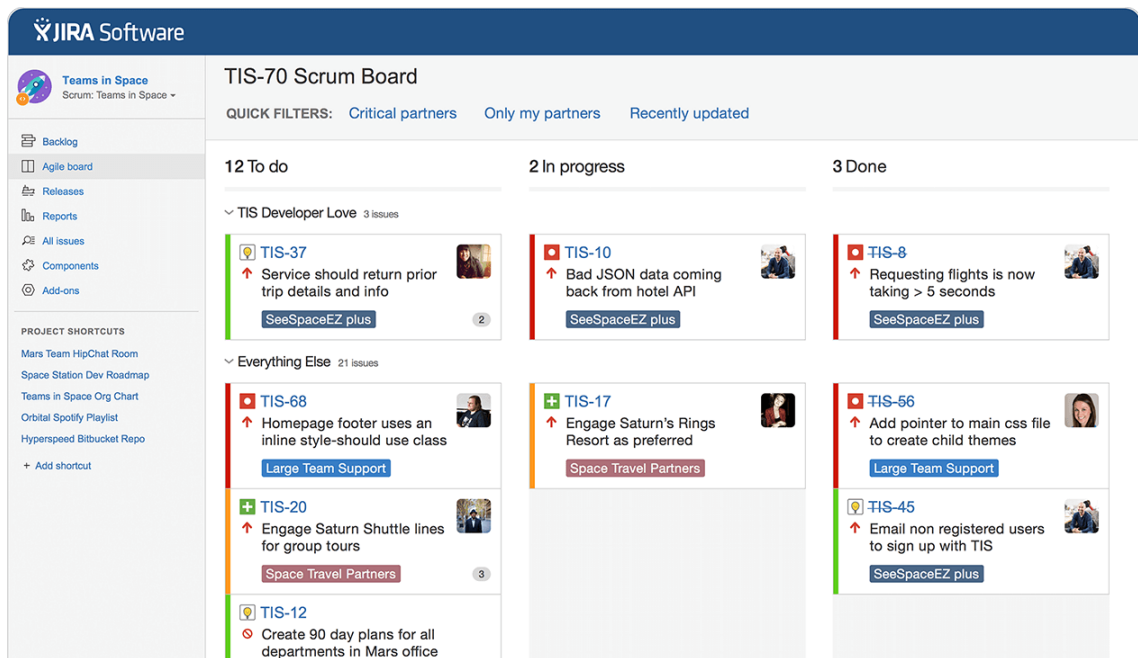


Figura 2.4: Interface do JIRA Software

Mais recentemente, a equipa por trás do Slack também acrescentou uma funcionalidade para realização de chamadas (de áudio ou vídeo) dentro de um canal ou em mensagens diretas. Torna-se assim desnecessário migrar para outra ferramenta para realizar chamadas entre membros da equipa.

O Slack permite a troca de ficheiros (imagens, PDFs, documentos, folhas de cálculo) entre utilizadores e em canais, com a possibilidade de acrescentar comentários aos mesmos, ou até mesmo atribuir-lhes uma estrela (de forma a ser mais fácil encontrá-los futuramente). É também possível usar serviços externos de partilha de ficheiros (como Google Drive¹⁸ ou Dropbox¹⁹): enviando o *link* para o ficheiro como mensagem, o ficheiro fica automaticamente sincronizado e pesquisável dentro da plataforma.

Uma grande vantagem do Slack é a sua biblioteca de integrações²⁰: existem integrações com uma quantidade cada vez maior de ferramentas externas, como ferramentas de desenvolvimento e produtividade, ferramentas de gestão de ficheiros e simples *bots*. Além disso, é possível criar *custom integrations*: o utilizador define que comandos a integração pode receber, e dado um *endpoint* público, são feitos pedidos a uma API externa, que irá processar os dados e devolver a resposta num formato que permita ao Slack apresentá-lo ao utilizador da maneira mais correta.

Outra vantagem do Slack (e uma das razões da sua existência) é o seu sistema de pesquisa. É possível pesquisar qualquer conteúdo que tenha sido partilhado no Slack, entre mensagens, notificações e ficheiros, com vários filtros relativos ao autor, canal usado, e data de envio, e mesmo dentro de ficheiros, dado que o Slack indexa o conteúdo de todos os ficheiros partilhados.

¹⁸Para mais informações consultar <https://drive.google.com>

¹⁹Para mais informações consultar dropbox.com

²⁰Para mais informações consultar <https://slack.com/apps>

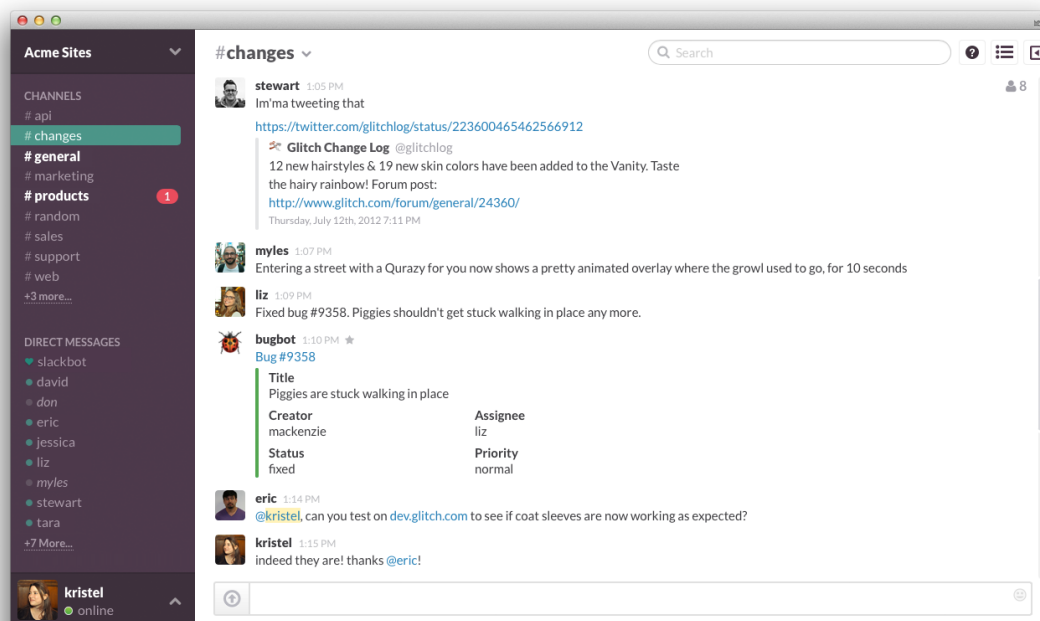


Figura 2.5: Interface do Slack

O Slack oferece, para além da interface *web*, aplicações *desktop* para os três sistemas operativos principais (Windows, Mac e Linux) e aplicações móveis para iOS, Android e Windows Phone²¹.

O Slack tem uma API própria, com várias funcionalidades desde a criação de *bots* à autenticação com a aplicação.

2.2.6 Yammer

O Yammer apresenta-se como uma rede social empresarial para comunicações privadas dentro de organizações[10].

Inicialmente, o Yammer foi criado como um sistema de comunicação interna para a equipa do website *geni.com*, sendo lançado como produto independente em 2008 e adquirido pela Microsoft em 2012.

O Yammer funciona sobre um sistema *freemium*, com possibilidade de *trial* gratuito durante um mês.

O Yammer é mais comunmente usado como serviço de mensagens instantâneas, com possibilidade de criação de grupos dentro da equipa, tal como o Slack.

²¹Para mais informações consultar <https://slack.com/downloads>

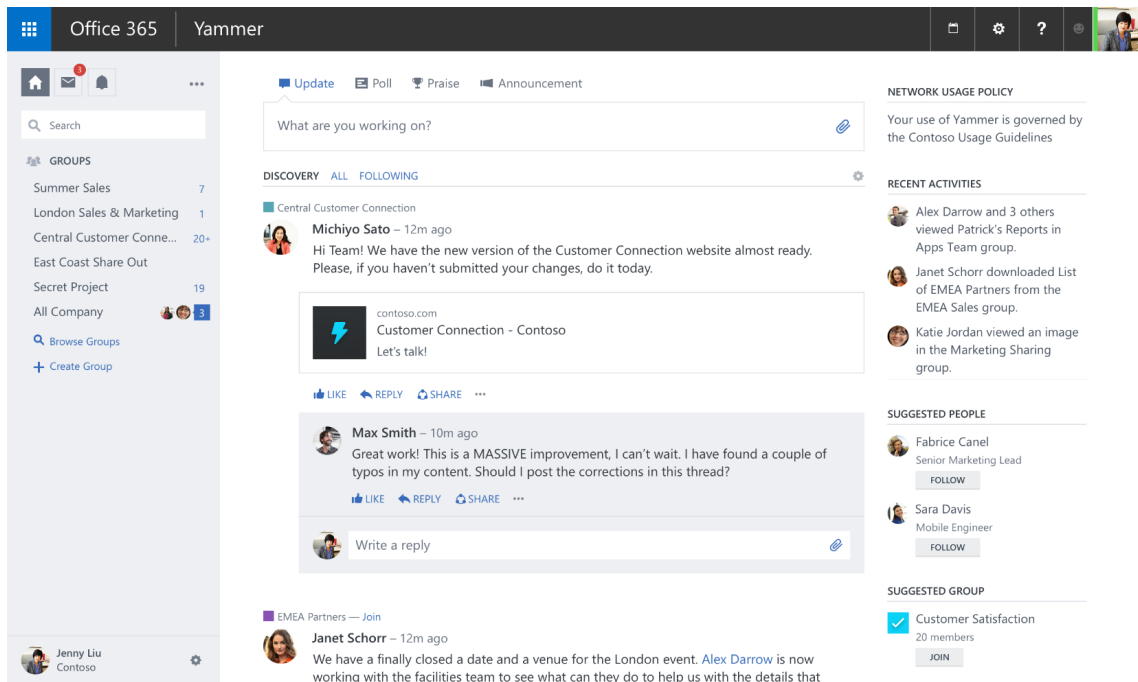


Figura 2.6: Interface do Yammer

2.2.7 Toggl

O Toggl é uma ferramenta de *time tracking*, isto é, permite identificar o tempo que cada utilizador demora a executar uma tarefa, em tempo real.

O Toggl permite executar esta medição com base em projetos e tarefas, apresentando também funcionalidades que permitem a criação de relatórios periódicos relativos a um utilizador, um projeto ou uma equipa.

Para além de permitir esta medição do tempo gasto numa tarefa em tempo real, que pode ser controlada através de uma das várias aplicações disponíveis (*web-based*, versão desktop para Windows, Mac e Linux, e versão móvel para iOS, Android, Symbian e BlackBerry OS), o Toggl também permite a inserção de intervalos de tempo manualmente, podendo ser associados a uma tarefa e projeto normalmente.

Cada entrada de tempo, adicionada manual ou dinamicamente, pode ter *tags* associadas, que podem ser usadas como filtro. Por exemplo, é possível criar uma *tag* para desenvolvimento e outra para escrita de documentação, e usá-las em vários projetos diferentes. Assim, ao obter estatísticas da aplicação, é possível obtê-las relativamente a cada projeto ou relativamente a cada tipo de tarefa.

Todas estas funcionalidades estão disponíveis na versão gratuita do Toggl[11], que permite equipas de até 5 utilizadores. Na versão paga, o Toggl também permite associar a cada tarefa o valor a pagar, de forma a justificar gastos perante o cliente, ou para uma simples estimativa pessoal.

O toggl é ideal para equipas pequenas ou trabalhadores individuais, dado o limite de utilizadores numa equipa na versão grátis. Para equipas maiores, é necessário o uso de um

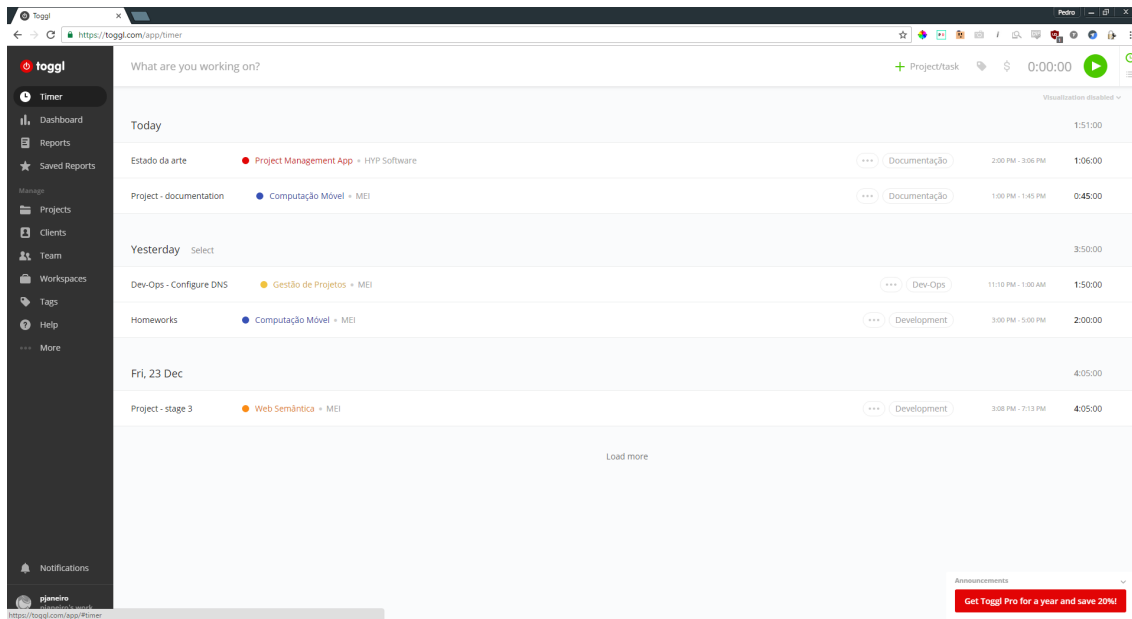


Figura 2.7: Interface de edição de entradas de tempo do Toggl

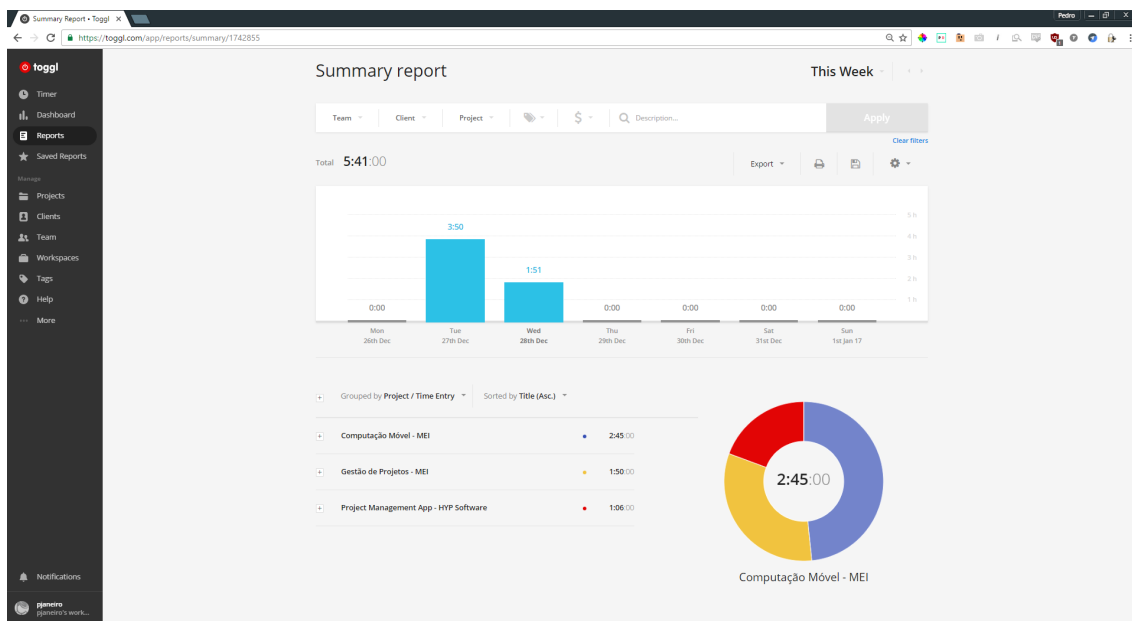


Figura 2.8: Interface de geração de relatórios do Toggl

plano pago).

O Toggl também tem uma API oficial[12], que permite executar qualquer tarefa disponível na aplicação através de uma fonte externa. Estas tarefas incluem autenticação, criação de entradas de tempo, edição de *tags* e geração de relatórios (incluindo a vista geral da *dashboard* de um projeto).

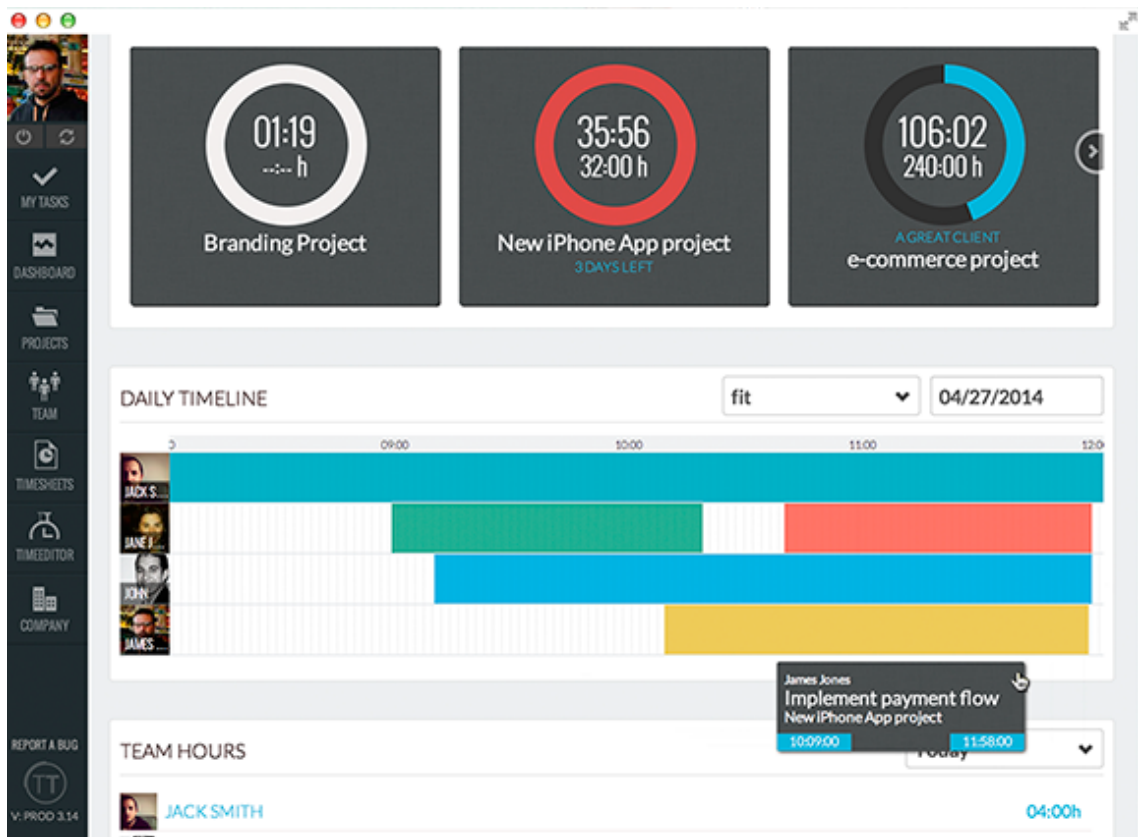


Figura 2.9: Interface da aplicação TrackingTime

2.2.8 TrackingTime

À semelhança do Toggl, o TrackingTime é uma ferramenta de *time tracking*[13]. O foco do TrackingTime é a gestão fácil e eficaz de tarefas de uma equipa ou projeto; pretende ajudar uma equipa a manter-se organizada, enquanto mantém um registo em tempo real das horas de trabalho efetivamente usadas[14].

Dado que se destina a equipas, e com um foco na colaboração, é possível ao mesmo tempo gerir clientes, projetos e tarefas pessoais, tendo noção daquilo que cada colega de equipa está a fazer em cada momento, com atualizações em tempo real, adição de comentários e notificações do sistema.

Tal como o Toggl, o TrackingTime permite analisar todos os dados e gerar relatórios automaticamente. Também tem aplicação em várias plataformas: *web-based*, versão desktop para Mac, aplicação para Chrome, e aplicação móvel para Android e iOS. Tem ainda um *bot* disponível para o Slack²².

O TrackingTime segue um modelo *freemium*, com a maioria das funcionalidades disponíveis para uso por todos, e com a versão paga a oferecer funcionalidades extra em termos de análise dos dados recolhidos e relatórios gerados.

Ao contrário do Toggl, o TrackingTime não tem uma API pública disponível.

²²Para mais informações consultar <https://slack.com/apps/A1YC1J8TH-trackingtime>

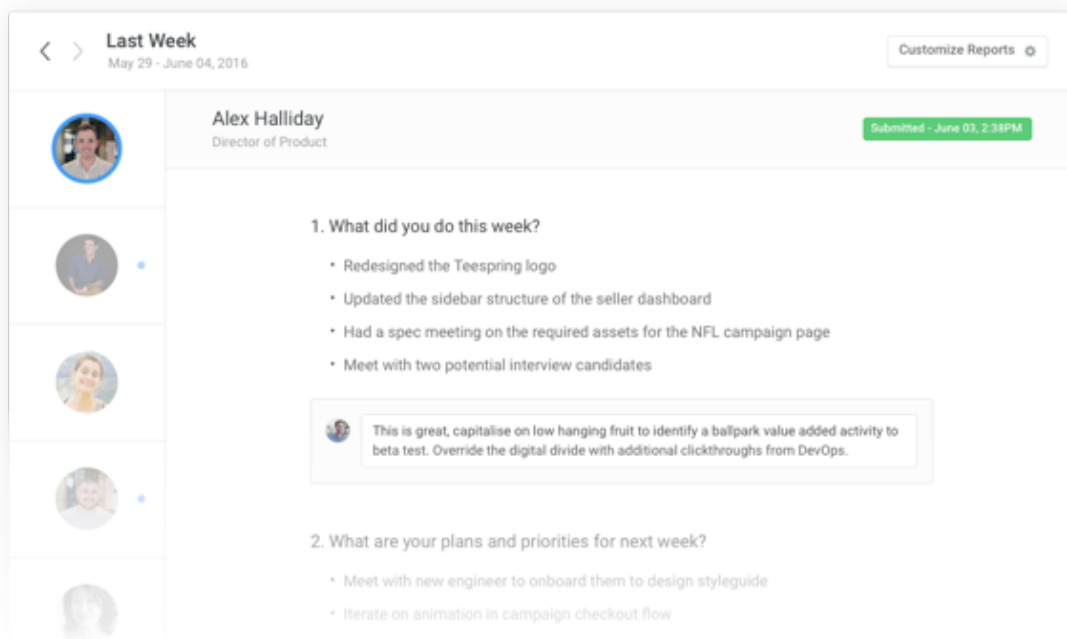


Figura 2.10: Interface do Lattice Check-ins

2.2.9 Lattice Check-ins

Lattice Check-ins é uma ferramenta criada pela Lattice²³ para ajudar gestores de projetos a comunicar com as suas equipas[15]. Em vez de se focar na gestão de recursos através da calendarização de tarefas associadas a um projeto, o Lattice Check-ins foca-se no contacto direto com os membros da equipa, para recolher a sua opinião relativamente ao trabalho desenvolvido e por desenvolver. Segundo *Jack Altman*, CEO da Lattice²⁴, os *check-ins* são uma sessão semanal de perguntas e respostas entre o gestor e os membros da equipa[16].

O utilizador deve responder, semanalmente, a algumas questões sobre como a semana correu, planos para a semana seguinte, e desafios ultrapassados ou vitórias atingidas. De seguida, é enviado um e-mail ao gestor de projeto, que deverá responder com palavras de encorajamento, reprimendas, ou com pedidos de mais informação.

O Check-ins permite a escolha de diferentes periodicidades de atualização, não sendo obrigatório realizar os inquéritos semanalmente. Além disso, o gestor de projeto também pode definir quais as questões a realizar aos elementos da equipa, de forma a adaptar o inquérito às necessidades da equipa em cada momento.

²³Para mais informações consultar <https://latticehq.com/about/>

²⁴Para mais informações consultar <https://www.linkedin.com/in/jackealtman>

2.2.10 Podio

O Podio apresenta-se como uma ferramenta que pode ser usada para virtualmente qualquer objetivo[17], incluindo gestão de projetos[18]. Para atingir este fim, o Podio funciona como um sistema de *templates*, que podem ser adaptados às necessidades de cada um.

Para a gestão de projetos, o Podio pretende minimizar o número de reuniões, telefonemas e e-mails necessários, ajudando ao mesmo tempo a estruturar melhor o trabalho realizado[18]. Para isso, sugere a criação de três aplicações (segundo os *templates* referidos anteriormente):

1. Projetos - esta aplicação pode ser usada para ver o estado atual de cada projeto em que o utilizador está envolvido, bem como outros projetos dentro da equipa
2. Entregas - esta aplicação contém as várias tarefas inerentes a cada projeto, bem como informação sobre o estado dos *deliverables* a elas associados
3. Reuniões - nesta aplicação, é possível gerir as reuniões do utilizador, e associá-las a calendários externos. Estas reuniões podem também ser associadas a projetos ou tarefas individuais

De forma a atingir os objetivos pretendidos com o Podio, o utilizador deve definir o fim com que pretende usar a aplicação (neste caso, gestão de projetos). De seguida, a sua conta será criada com o *template* respetivo, que pode ser alterado mais tarde (para gestão de vendas, por exemplo). Será redirecionado para uma área de trabalho com as três secções referidas anteriormente (Projetos, Entregas, Reuniões). Estes modelos podem ser alterados a qualquer momento, adicionando ou removendo campos (acrescentando um campo do tipo "ficheiro" para o logótipo do projeto, por exemplo).

O utilizador deve começar por criar um Projeto, que na versão *default* tem associados um título, uma *deadline*, utilizadores responsáveis, estado ("Iniciado", "Em progresso", "Terminado") e possíveis ficheiros e *tags*.

De seguida, pode criar uma ou mais Entregas, com um título, responsável, descrição, estado ("Não começado", "Em progresso", "Completo"), progresso (sobre a forma de uma barra de progresso), *deadline*, projeto associado (criado no passo anterior), e possíveis ficheiros e *tags*.

Noutra vista, é possível criar Reuniões, dando-lhes um nome, uma hora, lista de utilizadores convocados, uma ordem de trabalhos, localização e duração, projetos e entregas associados (dos passos anteriores), e possíveis ficheiros e *tags*.

Por último, na vista Atividade, é possível ver o estado geral dos vários Projetos, Entregas e Reuniões, com vista de calendário relativa às *deadlines* e com possibilidade de acrescentar mais vistas (por exemplo, uma vista com os contactos de todos os membros da equipa).

Na sua versão gratuita, o Podio permite equipas de até 5 elementos. Na versão paga, o número de elementos da equipa aumenta, com a adição de várias funcionalidades em

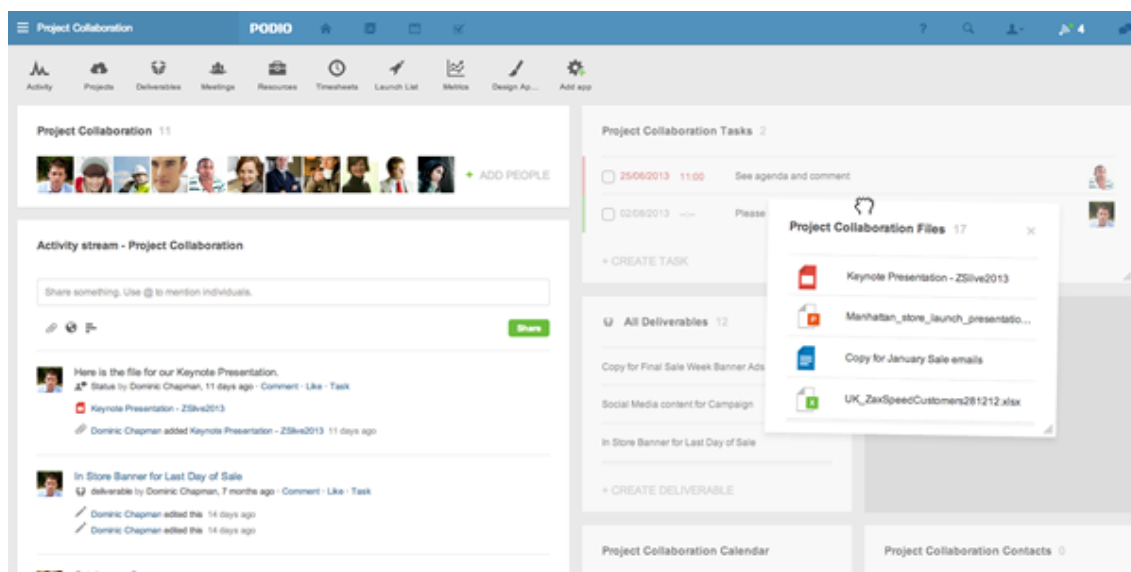


Figura 2.11: Interface da aplicação Podio, com várias aplicações associadas

termos de geração de relatórios, sincronização de dados pessoais, e automação de algumas tarefas de gestão.

O Podio está disponível sob a forma de aplicação *web* e aplicação móvel para iOS e Android.

Existem várias extensões disponíveis para o Podio, como por exemplo uma extensão para a geração de diagramas Gantt²⁵ ou para automação das funcionalidades da aplicação²⁶.

Todas as funcionalidades presentes na interface do Podio estão também disponíveis na sua API pública. De facto, a equipa diz que todo o website foi feito usando a API, de forma a mostrar como esta pode ser utilizada.

2.2.11 Comparação

Na tabela 2.1 é feita a comparação entre as várias plataformas estudadas nesta secção. Dado que as plataformas têm várias diferenças entre si, é difícil estabelecer uma melhor comparação entre todas. No entanto, é possível identificar as plataformas pagas e as plataformas *open source*. De notar que, para algumas das plataformas, existem alternativas *open source*, mas estas alternativas são usadas por um número muito inferior de utilizadores.

Na tabela 2.2 é possível ver em que etapas da Gestão de Projetos as várias plataformas poderão ser usadas, de acordo com o estudo efetuado na secção 2.1. Enquanto que as ferramentas de comunicação (Slack, Yammer) e de Gestão de Projetos (Redmine, Microsoft Project e Podio) poderão ser usadas virtualmente em qualquer etapa da Gestão de Projetos, para comunicação entre os membros da equipa e registo das decisões tomadas, ferramentas de gestão de tempo (Toggl, Tracking Time), gestão de *issues* (Trello, JIRA) e recolha de

²⁵Para mais informações consultar <https://smartgantt.com/>

²⁶Para mais informações consultar <http://www.globiflow.com/>

feedback só farão sentido nas etapas de Execução e Monitorização e controlo, para controlo do trabalho desenvolvido e a desenvolver em cada momento.

Plataforma	Funcionalidade	Pago	Open Source
Redmine	Gestão de Projetos	Não	Sim
Microsoft Project	Gestão de Projetos	Não	Não
Trello	Gestão de <i>issues</i>	Não	Não
JIRA	Gestão de <i>issues</i>	Não	Não
Slack	Comunicação	Não	Não
Yammer	Comunicação	Não	Não
Toggl	<i>Time tracking</i>	A partir de 6 <i>users</i>	Não
TrackingTime	<i>Time tracking</i>	A partir de 4 <i>users</i>	Não
Lattice Check-ins	Recolha de <i>feedback</i>	Não	Não
Podio	Gestão de Projetos	Não	Não

Tabela 2.1: Comparação de plataformas usadas em gestão de projetos

Plataforma	Etapa 1	Etapa 2	Etapa 3	Etapa 4	Etapa 5
Redmine	X	X	X	X	X
Microsoft Project	X	X	X	X	X
Trello			X	X	
JIRA			X	X	
Slack	X	X	X	X	X
Yammer	X	X	X	X	X
Toggl			X	X	
TrackingTime			X	X	
Lattice Check-ins			X	X	
Podio	X	X	X	X	X

Tabela 2.2: Utilização das plataformas nas várias etapas de Gestão de Projetos

2.2.12 Discussão e conclusões

Nesta secção, foram analisadas várias ferramentas que podem ser usadas, individualmente ou em conjunto, para auxiliar na tarefa de gestão de projetos.

Enquanto que algumas aplicações se focam em tarefas não específicas de projetos de *software*, como ferramentas de comunicação (Slack e Yammer) e *time tracking* (Toggl e TrackingTime), outras podem ser utilizadas para atingir um maior número de objetivos no campo da gestão de projetos de *software*.

2.3 Modelos de desenvolvimento de *Software*

No estudo de modelos de desenvolvimento de *software*, são considerados os seguintes modelos:

1. *Waterfall*
2. *Agile*
3. Espiral

2.3.1 *Waterfall*

A primeira descrição formal do modelo de desenvolvimento *Waterfall* surgiu em 1970, por Winston W. Royce[19], num artigo em que o mesmo é apresentado como um modelo com várias falhas e problemas de utilização.

O modelo *Waterfall*, tendo sido o primeiro modelo de desenvolvimento a ser introduzido, é bastante simples de aprender e utilizar. Trata-se de um modelo sequencial, ou seja, cada fase deve ser completada antes de começar a fase seguinte. No modelo original de Royce, estas fases são:

1. Levantamento de requisitos de *sistema e software*
2. Análise
3. *Design*
4. Implementação
5. Teste
6. Manutenção

Na primeira fase, levantamento de requisitos, são recolhidos todos os requisitos funcionais e não funcionais possíveis para o sistema a ser desenvolvido, e recolhidos num artefacto como o SRS (*Software Requirements Specification*).

Passando à segunda fase, é feita a análise do sistema como um todo e dos seus vários componentes. No final desta fase, é expectável a produção de vários artefactos, entre modelos de negócios e modelos de dados.

Na fase de *design* é feito todo o planeamento da solução de *software* a desenvolver. São analisadas as especificações criadas nas primeiras fases, e é definida a arquitetura global do sistema. Nesta fase podem ainda ser detetados requisitos de *hardware* e sistema que não tenham sido considerados anteriormente.

Dado o *design* definido para o *software*, passa-se ao desenvolvimento do mesmo. Os vários componentes de *software* são desenvolvidos individualmente, procedendo-se posteriormente à sua integração para produzir uma unidade coesa.

A fase de testes destina-se a detetar todos os erros que possam resultar das fases anteriores, e corrigi-los. Primeiramente são testados os vários componentes individualmente, e após a sua integração o sistema é testado como um todo.

Por último, o sistema desenvolvido é disponibilizado ao cliente. Resta a deteção e resolução de problemas inerentes ao ambiente de produção utilizado e à utilização do produto pelo cliente. Para resolver estes problemas, são frequentemente disponibilizados *patches* com as correções a efetuar, sendo também comum disponibilizar novas versões do produto para melhorar a sua qualidade.

A deteção de problemas em fases iniciais de um projeto de *software* traz vantagens relativamente à sua deteção em fases mais tardias. De facto, corrigir um problema detetado na especificações de requisitos requer menos recursos à equipa do que a correção do mesmo problema detetado apenas na implementação[20]. Dado que no modelo *Waterfall* cerca de 30% do tempo gasto num projeto se destina às fases de levantamento de requisitos e sua análise, a probabilidade de detetar erros nestas fases aumenta, tornando-se numa vantagem do modelo.

Outra vantagem do modelo *Waterfall* é a sua aposta na documentação. Dado que todas as decisões relativas à arquitetura do sistema estão bem documentadas, a entrada de novos membros na equipa torna-se fácil, pois terão acesso a toda a informação necessária através da leitura destes documentos.

No entanto, o modelo *Waterfall* também é alvo de críticas. Por exemplo, o cliente pode pretender acrescentar requisitos ao projeto, o que leva a que todas as fases do modelo tenham que ser revisitadas, aumentando o custo do projeto. Além disso, podem existir limitações nas tecnologias usadas que não são aparentes na fase de *design*, e a sua deteção deveria levar à reformulação da arquitetura do sistema a qualquer momento, o que não acontece no modelo *Waterfall*.

O modelo *Waterfall* é particularmente eficiente em projetos em que os requisitos existentes estão bem definidos e documentados e as tecnologias a usar estão bem interiorizadas pela equipa. No entanto, não é um modelo facilmente aplicável a projetos de maior escala, devido à sua rigidez e à inexistência de fases de revisão.

Vantagens	Desvantagens
Simple e fácil de aprender e usar	Não aconselhado para projetos maiores ou contínuos
Fases bem definidas e completadas uma de cada vez	Não aconselhado para projetos em que os requisitos podem mudar facilmente
Fácil de gerir, devido à rigidez do modelo e sequência das fases	Não é produzido <i>software</i> funcional até fases tardias de desenvolvimento

Tabela 2.3: Vantagens e desvantagens do modelo *Waterfall*

2.3.1.1 Variações

Existem variações do modelo *Waterfall* que removem algumas das restrições apresentadas, permitindo o regresso a uma fase anterior do modelo quando necessário, ou a repetição de algumas das fases de forma cíclica, como no modelo de *Staged Delivery*[21].

2.3.2 Agile

O modelo *Agile* começou a ser desenvolvido na década de noventa como resposta ao descontentamento de alguns programadores à rigidez dos modelos de desenvolvimento então usados, maioritariamente baseados no modelo *Waterfall*. De uma forma geral, eram criticadas a forma como era feito o levantamento de requisitos, de forma exaustiva no início do projeto e a forte componente de documentação esperada do processo. Neste sentido, começaram a ser aplicados modelos de desenvolvimento considerados mais "leves".

Em 2001, um grupo de engenheiros de *software* reuniu-se em Snowbird para discutir estes modelos emergentes e chegar a uma especificação de um modelo mais geral que englobe todas as visões neste ponto potencialmente divergentes. Desta forma, surge o *Manifesto for Agile Software Development*[22], que refere os principais valores e princípios do desenvolvimento *Agile*:

- Valores

1. *Indivíduos e interações mais do que processos e ferramentas*
2. *Software funcional mais do que documentação abrangente*
3. *Colaboração com o cliente mais do que negociação contratual*
4. *Responder à mudança mais do que seguir um plano*

- Princípios

1. *A nossa maior prioridade é, desde as primeiras etapas do projecto, satisfazer o cliente através da entrega rápida e contínua de software com valor.*
2. *Aceitar alterações de requisitos, mesmo numa fase tardia do ciclo de desenvolvimento. Os processos ágeis potenciam a mudança em benefício da vantagem competitiva do cliente.*
3. *Fornecer frequentemente software funcional. Os períodos de entrega devem ser de poucas semanas a poucos meses, dando preferência a períodos mais curtos.*
4. *O cliente e a equipa de desenvolvimento devem trabalhar juntos, diariamente, durante o decorrer do projecto.*
5. *Desenvolver projectos com base em indivíduos motivados, dando-lhes o ambiente e o apoio de que necessitam, confiando que irão cumprir os objectivos.*
6. *O método mais eficiente e eficaz de passar informação para e dentro de uma equipa de desenvolvimento é através de conversa pessoal e directa.*
7. *A principal medida de progresso é a entrega de software funcional.*

8. *Os processos ágeis promovem o desenvolvimento sustentável. Os promotores, a equipa e os utilizadores deverão ser capazes de manter, indefinidamente, um ritmo constante.*
9. *A atenção permanente à excelência técnica e um bom desenho da solução aumentam a agilidade.*
10. *Simplicidade – a arte de maximizar a quantidade de trabalho que não é feito – é essencial.*
11. *As melhores arquitecturas, requisitos e desenhos surgem de equipas auto-organizadas.*
12. *A equipa reflecte regularmente sobre o modo de se tornar mais eficaz, fazendo os ajustes e adaptações necessárias.*

O modelo *Agile* baseia-se na ideia de todos os projetos terem que ser avaliados de forma diferente e de os métodos terem que ser adaptados de forma a melhor respeitar as necessidades do projeto. Em *Agile*, os projetos são divididos em várias etapas, cada uma com um diferente objetivo em termos de funcionalidades a apresentar. É adotada uma abordagem iterativa sobre o desenvolvimento e o *software* produzido é entregue após cada uma destas metas. As várias entregas têm um número crescente de funcionalidades presentes, culminando no produto final com todas as funcionalidades implementadas.

Enquanto no modelo *Waterfall* existem 6 fases que devem ser completadas individualmente e pela ordem estipulada para gerar o produto de software, em modelos *Agile* estas fases são adaptadas a cada uma das metas individuais. Desta forma, sendo definidas as várias metas do projeto, para cada uma são cumpridas as seguintes fases (este modelo pode ser facilmente adaptado às necessidades do projeto e da equipa):

1. Planeamento
2. Levantamento e análise de requisitos
3. *Design*
4. Implementação
5. Teste

Na primeira fase, é feito o planeamento relativo à meta em mãos, nomeadamente a escolha das funcionalidades a implementar, calendarização e divisão de tarefas.

Passando à segunda fase, são recolhidos os requisitos funcionais e não funcionais associados às funcionalidades selecionadas para a meta. Enquanto que no modelo *Waterfall*, realizando-se apenas uma vez o levantamento de requisitos do projeto, se espera que no final da mesma seja produzido um artefacto como o SRS (*Software Requirements Specification*), no modelo *Agile* é comum este documento ser também redigido de forma iterativa, sendo melhorado em cada meta com a informação dos novos requisitos.

Na fase de *design* são analisadas as alterações que a equipa se propôs a fazer na meta e estudada a melhor forma de adaptar o sistema para as introduzir.

Espera-se no modelo *Agile* que a fase de desenvolvimento seja bastante mais rápida que no modelo *Waterfall*, uma vez que se prevê a entrega de uma nova versão do *software* ao fim de um período relativamente curto, entre poucas semanas e poucos meses. Para isso, aposta-se na simplicidade do código produzido e na constante análise do comportamento da equipa de forma a estudar possíveis alterações na sua organização.

Vantagens	Desvantagens
Abordagem realista do desenvolvimento de <i>software</i>	Não adequado a projetos muito complexos
Funcionalidades podem ser desenvolvidas bastante rapidamente	As funcionalidades a ser apresentadas e a qualidade do código são fortemente influenciadas pelo sistema rígido de entregas
Bom modelo para projetos cuja <i>scope</i> pode mudar frequentemente	Depende fortemente de um bom planeamento e de uma boa Gestão de Projetos
Apresenta resultados rapidamente	Depende muito do trabalho individual, devido à documentação mínima
Facilmente implementado	

Tabela 2.4: Vantagens e desvantagens do modelo *Agile*

2.3.3 Espiral

O modelo em Espiral pretende combinar o conceito de desenvolvimento iterativo com o controlo e a sistematização do modelo *Waterfall*, e dando ênfase à análise de riscos.

No modelo em Espiral são consideradas quatro fases, que se repetem em cada ciclo do processo. Essas fases são:

1. Definição de objetivos
2. Análise de alternativas e identificação de riscos
3. Desenvolvimento e verificação
4. Planeamento

Na primeira fase, tal como no modelo *Agile*, são identificados os objetivos da equipa de desenvolvimento para a iteração atual.

Na fase de análise de riscos, estes são primeiramente identificados, sendo de seguida estudadas alternativas ao processo de desenvolvimento ou mesmo a componentes do produto de forma a mitigar estes riscos.

A terceira fase do modelo em Espiral deve ser adaptada aos objetivos da iteração em que a equipa se encontra. Se numa iteração no início do projeto é expectável o levantamento e validação de requisitos, uma iteração posterior deverá levar à definição da arquitetura do produto e eventualmente será a fase do modelo em que o código é implementado e testado.

A última fase do modelo em Espiral é usada para o planeamento das etapas seguintes, seja o planeamento do levantamento de requisitos, de desenvolvimento e de testes.

Para garantir alguma uniformidade na aplicação deste modelo, dada a liberdade que é dada às equipas na sua definição, Boehm definiu[23] as 6 Invariantes do Modelo em Espiral, que consiste em 6 características que todas as iterações to modelo devem ter. Estas invariâncias são:

1. Determinação concorrente de artefactos O desenvolvimento deve ser planeado de forma a que os artefactos planeados sejam produzidos de forma sequencial e lógica. Por exemplo, os requisitos devem ser levantados antes de se dar início à fase de desenvolvimento.
2. Cada ciclo deve passar pelas etapas de objetivos, riscos/alternativas, implementação e revisão/planeamento
3. O nível de esforço deve ser determinado pelas considerações inerentes à análise dos riscos detetados
4. O grau de detalhe deve ser definido pelas considerações inerentes à análise dos riscos detetados
5. Cumprimento de três metas-chave no desenvolvimento do projeto
 - (a) Definição dos objetivos
Cumprida quando a resposta à questão *"Foi definida uma abordagem técnica e de gestão que permita satisfazer os objetivos de todos?"* for positiva
 - (b) Definição da arquitetura
Cumprida quando a resposta à questão *"Foi definida uma abordagem que permita satisfazer os objetivos de todos, e todos os riscos significativos foram eliminados ou mitigados?"* for positiva
 - (c) Capacidade operacional
Cumprida quando a resposta à questão *"Existe uma preparação suficiente do software, utilizadores e operadores para satisfazer os objetivos de todos lançando o sistema?"* for positiva

Sempre que a resposta a uma destas questões for negativa, a equipa pode optar por abandonar o projeto ou utilizar mais um ciclo do modelo para tentar obter uma resposta positiva

6. Ênfase nas atividades e artefactos de sistema e ciclo de vida

O modelo em Espiral é muito usado na indústria do *software* visto que se adequa ao processo de desenvolvimento natural de qualquer produto, ao mesmo tempo que envolve risco mínimo para o cliente. É normalmente usado em projetos com maiores limitações de *budget* e que portanto dependem muito da avaliação de riscos, em projetos de risco mais elevado, ou em projetos em que o cliente apresenta uma grande incerteza em relação aos seus requisitos.

Vantagens	Desvantagens
Alterações aos requisitos são facilmente realizadas	Implica uma gestão mais complexa do processo
Permite a implementação de vários níveis de protótipos	Pode ser difícil estimar o fim do projeto
O levantamento de requisitos pode ser mais preciso	Não adequado a projetos pequenos ou de baixo risco
O desenvolvimento pode ser dividido em módulos, com os módulos com maior risco associado a ser desenvolvidos mais cedo, ajudando à gestão de riscos	O elevado número de passos intermédios pode produzir documentação excessiva

Tabela 2.5: Vantagens e desvantagens do modelo em Espiral

2.4 Tipos de testes

Teste de *software* é um processo, ou conjunto de processos, definidos para garantir que um produto de *software* faz aquilo que foi criado para fazer, e que não faz nada que não seja pretendido[24].

Nesta secção, serão estudados alguns tipos de teste a que habitualmente se sujeitam os produtos de *software*, agrupados pelo seu nível de especificidade: primeiro, serão estudados testes unitários, que são aplicados a cada módulo independente do artefacto; de seguida, são estudados testes de integração, que testam a forma como estes módulos comunicam entre si; por último, são estudados testes de sistema, que pretendem testar o sistema como um todo, e que definem se alguns requisitos não-funcionais são cumpridos, como a *performance* e a escalabilidade.

2.4.1 Testes unitários

Testes unitários são testes em que módulos individuais do sistema são testados para identificar problemas ao mais baixo nível. Não requerem o conhecimento de outros módulos nem das interações entre estes, possibilitando que sejam preparados de forma independente pelos responsáveis por cada módulo.

Entre as vantagens dos testes unitários, podem-se referir:

- Reduzem o custo dos testes, visto que os defeitos são detetados em fases iniciais do desenvolvimento.
- Facilitam a reestruturação do código.
- Não têm que ser reformulados se se adicionar ou remover um módulo do projeto.
- Podem ser executados a qualquer momento pelo programador.

Idealmente, os testes unitários são executados ao fim de cada alteração no código. O programador obtém a versão mais recente do código, faz as alterações planeadas, executa os testes unitários, corrige os defeitos detetados e executa os testes novamente, até nenhum defeito ser identificado. De seguida, o código pode seguir para a fase seguinte do ciclo de vida do modelo de desenvolvimento.

Os testes unitários podem ser subdivididos em:

1. Testes *Black Box*
2. Testes *White Box*
3. Testes *Gray Box*

De seguida, é descrito cada um destes tipos de testes unitários.

2.4.1.1 Testes *Black Box*

Nos testes *Black Box*, é testada a interface do utilizador, através da inserção de vários *inputs* e análise do *output* produzido.

A ideia por trás dos testes *Black Box* envolve a noção de, literalmente, uma caixa preta, em que é impossível observar o seu interior. Esta noção é extrapolada para os testes *Black Box*, em que se parte do princípio de que a pessoa a testar não tem forma de saber como uma função funciona, sabendo apenas qual é o seu domínio e qual é o valor de retorno esperado.

Dado que nos testes *Black Box* o responsável pelos testes precisa de saber *o que é suposto um módulo fazer*, mas não *como o faz*, não precisa de ter conhecimentos de programação, apenas do funcionamento do produto.

2.4.1.2 Testes *White Box*

Nos testes *White Box*, ao contrário dos testes *Black Box*, é analisada a estrutura interna dos módulos, e não a sua funcionalidade. O responsável pelos testes analisa o código produzido, e estuda todos os caminhos que os fluxos de informação e de controlo podem seguir, isto é, todas as variações possíveis do comportamento do módulo.

De forma a analisar corretamente as várias variações do fluxo dos módulos, espera-se que o responsável pelos testes tenha não só um bom conhecimento do projeto, mas também conhecimentos de programação em geral. São, por isso, testes mais complexos de preparar do que testes *Black Box*, contribuindo no entanto para o conhecimento interno do projeto por parte da equipa e para a deteção de ramificações do código que nunca são utilizadas.

2.4.1.3 Testes *Gray box*

Nos testes *Gray Box* é usado um intermédio entre testes *Black Box* e testes *White Box*: o responsável pelos testes tem acesso à informação sobre os requisitos do módulo, e à documentação referente ao *design* da aplicação; no entanto, não tem acesso ao código original do módulo, pelo que não poderá realizar testes *White Box*.

Os testes *Gray Box* trazem benefícios tanto de testes *White Box* como de testes *Black Box*, possibilitando a formulação de testes mais complexos do que *Black Box*, mas não requerendo que os testes sejam realizados por um programador com conhecimentos profundos sobre o sistema como é exigido pelos testes *White Box*.

2.4.2 Testes de integração

Após a realização dos testes unitários, é possível passar para a realização de testes de integração. Nos testes de integração, o responsável deve verificar se se encontram respeitados

os requisitos funcionais, de desempenho e de confiabilidade entre módulos que se previam nos requisitos do projeto. Nesta fase, deixam-se de lado os testes individuais e os vários módulos são testados como um grupo coeso.

Nos testes de integração, os vários módulos são utilizados como num modelo *Black Box*, visto que não se pretende estudar a sua estrutura interna, mas apenas analisar o seu comportamento dado o resultado da execução de outros módulos com os quais interage.

Existem várias estratégias de testes de integração, entre as quais:

1. Integração *Big Bang*
2. Integração *Bottom Up*
3. Integração *Top Down*

De seguida, é descrito cada um destes tipos de testes de integração.

2.4.2.1 Integração *Big Bang*

Em testes de integração usando uma estratégia *Big Bang*, todos os módulos são agrupados de uma só vez, resultando no sistema completo.

Usando esta estratégia, é difícil isolar os erros encontrados, pois não se dá grande atenção às interfaces entre módulos individuais. Além disso, é muito difícil cobrir todos os cenários possíveis da integração, e é bastante provável que erros críticos não sejam detetados e possam surgir em produção. Por outro lado, é uma estratégia que permite poupar bastante tempo nos testes de integração, pelo que poderá ser uma estratégia interessante para projetos mais pequenos, com menor número de módulos ou requisitos de integração menos rígidos.

2.4.2.2 Integração *Bottom Up*

Na estratégia *Bottom Up*, primeiro são testados os módulos de mais baixo nível, e de seguida são testados os módulos que dependem destes, até todos serem testados. Desta forma, apesar de os módulos mais complexos serem testados em último lugar, é possível determinar mais facilmente os seus erros de interface, pois são excluídos entretanto possíveis erros de ligação entre módulos mais básicos.

Para ser possível realizar testes de integração com esta estratégia, é necessário que módulos com o mesmo nível de complexidade estejam prontos ao mesmo tempo, o que nem sempre é possível. No entanto, este método ajuda a determinar o nível de complexidade do *software* desenvolvido, e permite limitar a quantidade de casos de teste necessárias em níveis superiores, por darem origem a cenários já testados em níveis mais baixos.

2.4.2.3 Integração *Top Down*

O objetivo da estratégia *Top Down* é simular o comportamento de módulos de baixo nível que ainda não estejam integrados. Para isso, são usados *stubs*, que irão simular a produção do resultado esperado por um módulo dado um conjunto de dados de entrada. Para além de estes *stubs* poderem substituir módulos do próprio sistema, também podem ser usados para simular integração com serviços externos.

Uma vantagem dos testes de integração *Top Down* é que permitem que os vários módulos sejam desenvolvidos de forma independente, em vez de requerer que módulos do mesmo nível estejam prontos ao mesmo tempo, como na abordagem *Bottom Up*. Por outro lado, esta metodologia pressupõe que os módulos mais complexos sejam integrados antes dos módulos de mais baixo nível, o que pode nem sempre ser possível.

2.4.3 Testes de sistema

Os testes de sistema têm como objetivo testar um sistema completo e já integrado, de forma a avaliar se verifica os requisitos previamente definidos. Os testes de sistema têm como alvo não apenas o *design* do sistema, mas também o seu comportamento e inclusivamente a forma como os utilizadores interagem com ele.

Trata-se de um tipo de teste no molde dos testes *Black Box*, em que não é necessário um conhecimento do funcionamento interno do sistema. Por isso, é comum serem realizados por uma equipa independente da equipa de desenvolvimento, de modo a que a qualidade do sistema seja medida sem qualquer tipo de predisposição.

Alguns tipos de testes normalmente realizados no contexto de testes de sistema incluem:

1. Testes de usabilidade
2. Testes de escalabilidade
3. Testes de *stress*

De seguida, é descrito cada um destes tipos de testes de sistema.

2.4.3.1 Testes de usabilidade

Testes de usabilidade são testes não funcionais cujo objetivo é determinar a facilidade com que o utilizador final usa o sistema. Este atributo é difícil de avaliar de forma objetivo, mas alguns parâmetros que normalmente são usados com esse propósito são:

- Nível de experiência necessário para usar o sistema.
- Tempo necessário para realizar determinadas operações.

- Número de operações atômicas necessárias para realizar uma operação mais complexa.
- Grau de satisfação do sujeito do teste após utilização do sistema.

2.4.3.2 Testes de escalabilidade

Os testes de escalabilidade têm como objetivo estudar a capacidade de resposta do sistema perante um incremento no número de pedidos por utilizador, número de utilizadores a usar o sistema em paralelo, ou tamanho de uma base de dados (caso se aplique).

Alguns parâmetros medidos são:

- Tempo de resposta.
- Quantidade de dados transmitidos.
- Número de pedidos ou repostas por unidade de tempo.

2.4.3.3 Testes de *stress*

Para realizar testes de *stress*, o sistema é sujeito a condições extremas, de forma a garantir que as consegue suportar da melhor maneira. Dado que se trata de uma situação que potencialmente irá ocorrer em produção, é fundamental perceber a forma como o sistema responde e recupera, por exemplo verificando se há perda de dados ou se é dada informação suficiente ao utilizador em caso de falha em vez de informação de *debug* que potencialmente não irá saber utilizar.

Alguns cenários a que normalmente se sujeita o sistema incluem:

- Monitorizar o sistema quando se atinge o número máximo de utilizadores ligados em simultâneo.
- Monitorizar o sistema quando vários utilizadores realizam a mesma operação crítica ao mesmo tempo, ou acedem ao mesmo recurso de sistema.
- Monitorizar o sistema depois de propositadamente desativar um servidor ou base de dados (conforme aplicável).

2.5 Conclusão

No capítulo do estado da arte, foram apresentados vários estudos realizados na fase inicial do projeto em mãos, úteis para a tomada de decisões nas fases de planeamento, descrição do sistema, implementação e testes.

Inicialmente, foi estudado o próprio conceito de gestão de projetos, com a enumeração de várias etapas consideradas essenciais para a atividade.

De seguida, foram estudadas plataformas já existentes no mercado para auxiliar ao processo de gestão de projetos de *software*, com vários âmbitos diferentes, de forma a contemplar o maior leque possível de funcionalidades. Analisando a tabela 2.2, é possível ver que a maior parte das ferramentas disponíveis tende a suportar preferencialmente as etapas de Execução e Monitorização e controlo de projeto, sendo difícil aplicar às outras etapas uma vez que se tratam de etapas de planeamento e conclusão. A plataforma a implementar deverá ter também foco nestas duas etapas, tentando ao mesmo tempo ser uma ferramenta útil para recolha de métricas úteis no encerramento de projetos.

Foi feito um estudo relativo a possíveis modelos de desenvolvimento de *software* a aplicar a este projeto, considerando vários fatores como a complexidade de implementação e possibilidade de reanalisar os requisitos após o seu levantamento inicial. Na secção 4.1 será tomada uma decisão de acordo com os dados recolhidos. Além disso, a plataforma deverá suportar o desenvolvimento de projetos nos vários modelos de desenvolvimento, promovendo a comunicação entre os membros da equipa e o acompanhamento do estado de desenvolvimento do projeto, independentemente do modelo utilizado.

Por último, foram estudados vários tipos de teste possíveis de aplicar ao produto, em várias etapas do seu desenvolvimento. Para os testes unitários, será dada prioridade a testes *white box*, dado o conhecimento existente sobre a constituição interna e fluxo de dados dos vários componentes do sistema; nos testes de integração, será usada uma abordagem *bottom up*, integrando módulos entre si até todo o sistema ser considerado; para testes de sistema, serão necessários testes de usabilidade, escalabilidade e *stress*, que foram apresentados.

Capítulo 3

Descrição do Sistema

Neste capítulo é descrito o sistema que será produzido neste projeto, descrevendo vários fatores importantes para o seu desenvolvimento.

Será feita inicialmente uma análise dos requisitos do produto, tanto requisitos funcionais como requisitos não funcionais. Ainda nesta secção, serão revistas as restrições de sistema a ter em conta.

Por último, será apresentado um diagrama de entidade relacionamento, que descreve a base de dados do sistema.

3.1 Requisitos

Para iniciar o levantamento de requisitos, é necessário analisar os requisitos funcionais, requisitos não funcionais, e restrições do sistema. Estes requisitos constituem a base do projeto, definindo o que os *stakeholders* do sistema pretendem dele, e serão tidos em conta em todas as etapas consequentes do projeto.

3.1.1 Requisitos funcionais

Para a apresentação dos requisitos funcionais, serão enumeradas várias *user stories* recolhidas para o efeito. Para cada *user story* é apresentado um *id*, através do qual esta será identificada posteriormente, uma pequena descrição, possíveis dependências de outras *user stories*, e a sua prioridade. Quando aplicável, algumas *user stories* têm também uma referência para um dos *wireframes* presentes na secção A.

Para identificar prioridades, é usado um sistema com três níveis diferentes de prioridade. As *user stories* com prioridade *Must have* são as *user stories* com prioridade mais elevada, devendo ser cumpridas obrigatoriamente no produto final; as *user stories* com prioridade *Should have* têm prioridade intermédia, sendo cumpridas mediante a conclusão das *user*

stories com prioridade *Must have*; por último, as *user stories* com prioridade *Could have* são *user stories* que trazem algo mais à aplicação, não identificando um requisito que os *stakeholders* considerem obrigatório, mas que melhoraria a experiência do utilizador final ao utilizar o produto.

3.1.1.1 Informação

US 1. Enquanto utilizador da plataforma, devo poder visitar uma *home page*

- Descrição: Uma página com o nome da plataforma e opções de *login*/registo
- Dependências: Nenhuma
- Prioridade: *Must have*
- *Wireframe*: Não tem

US 2. Enquanto utilizador da plataforma, devo poder visitar uma página com *FAQ*

- Descrição: Uma página com um conjunto de dúvidas frequentes e suas respostas
- Dependências: Nenhuma
- Prioridade: *Must have*
- *Wireframe*: Não tem

3.1.1.2 Conta

US 3. Enquanto utilizador não registado, devo poder registar-me usando o meu *e-mail* e uma *password*

- Descrição: Uma página de registo com um formulário para o meu *e-mail* e a *password*
- Dependências: US1
- Prioridade: *Must have*
- *Wireframe*: Figura A.1

US 4. Enquanto utilizador não autenticado, devo poder fazer *login* usando o meu *e-mail* e *password*

- Descrição: Uma página de *login* com um formulário para o meu *e-mail* e a *password*

- Dependências: US3
- Prioridade: *Must have*
- *Wireframe*: Figura A.1

US 5. Enquanto utilizador autenticado, devo poder visualizar a minha informação de conta

- Descrição: Página de perfil com o meu nome, *e-mail*, equipas e foto de perfil
- Dependências: US4
- Prioridade: *Must have*
- *Wireframe*: Não tem

US 6. Enquanto utilizador autenticado, devo poder atualizar a minha informação de conta

- Descrição: Uma página com um formulário para o meu nome, foto de perfil e *password*
- Dependências: US4
- Prioridade: *Must have*
- *Wireframe*: Não tem

US 7. Enquanto utilizador autenticado, devo poder visitar uma página inicial com informação geral dos meus projetos

- Descrição: Uma página com informação sobre os projetos do utilizador
- Dependências: US4, US12
- Prioridade: *Must have*
- *Wireframe*: Não tem

3.1.1.3 Equipa

US 8. Enquanto utilizador autenticado, devo poder criar uma equipa

- Descrição: Uma página com um formulário para inserir o nome da equipa e membros iniciais
- Dependências: US4

- Prioridade: *Must have*
- *Wireframe*: Não tem

US 9. Enquanto utilizador autenticado, devo poder listar todas as equipas a que pertença

- Descrição: Uma página com uma lista de todas as equipas a que pertença
- Dependências: US8
- Prioridade: *Must have*
- *Wireframe*: Não tem

US 10. Enquanto utilizador autenticado, devo poder atualizar a informação de uma equipa a que pertença

- Descrição: Uma página com um formulário para o nome de uma equipa ou seus membros
- Dependências: US8
- Prioridade: *Must have*
- *Wireframe*: Não tem

US 11. Enquanto utilizador autenticado, devo poder abandonar uma equipa a que pertença

- Descrição: Um botão em cada elemento da lista de equipas a que pertença
- Dependências: US8
- Prioridade: *Must have*
- *Wireframe*: Não tem

3.1.1.4 Projeto

US 12. Enquanto utilizador autenticado, devo poder criar um projeto, tornando-me seu *master*

- Descrição: Uma página com um formulário para o nome, descrição, equipa, cliente e *budget* de um projeto
- Dependências: US4

- Prioridade: *Must have*
- *Wireframe*: Não tem

US 13. Enquanto utilizador autenticado, devo poder listar todos os projetos em que trabalho

- Descrição: Uma página com uma lista de todos os projetos em que trabalho
- Dependências: US12
- Prioridade: *Must have*
- *Wireframe*: Figura A.4

US 14. Enquanto utilizador autenticado, devo poder listar todos os projetos de uma equipa a que pertença

- Descrição: Uma página com uma lista de todos os projetos de uma equipa a que pertença
- Dependências: US8, US12
- Prioridade: *Must have*
- *Wireframe*: Figura A.4

US 15. Enquanto utilizador autenticado, devo conseguir visualizar informação detalhada de um projeto em que trabalho

- Descrição: Uma página com informação detalhada de um projeto
- Dependências: US12
- Prioridade: *Must have*
- *Wireframe*: Figura A.5

US 16. Enquanto utilizador autenticado, devo poder atualizar a informação de um projeto, desde que seja seu *master*

- Descrição: Uma página com um formulário para o nome, descrição, equipa, cliente e *budget* de um projeto
- Dependências: US12
- Prioridade: *Must have*

- *Wireframe*: Não tem

US 17. Enquanto utilizador autenticado, devo poder apagar qualquer projeto, desde que seja seu *master*

- Descrição: Um botão em cada elemento da lista de projetos em que trabalho
- Dependências: US13
- Prioridade: *Must have*
- *Wireframe*: Não tem

US 18. Enquanto utilizador autenticado, devo poder listar todos os membros de um projeto em que trabalhe

- Descrição: Uma lista com os membros do projeto
- Dependências: US13
- Prioridade: *Must have*
- *Wireframe*: Não tem

US 19. Enquanto utilizador autenticado, devo poder adicionar membros a um projeto, desde que seja seu *master*

- Descrição: Uma página com um formulário para adicionar membros a um projeto através do seu nome, com um campo para o tipo de participação (*guest*, *reporter*, *developer* ou *master*)
- Dependências: US13
- Prioridade: *Must have*
- *Wireframe*: Não tem

US 20. Enquanto utilizador autenticado, devo poder remover membros de um projeto, desde que seja seu *master*

- Descrição: Um botão em cada elemento da lista de membros de um projeto
- Dependências: US13
- Prioridade: *Must have*
- *Wireframe*: Não tem

US 21. Enquanto utilizador autenticado, devo poder alterar o tipo de participação de um utilizador num projeto (*guest, reporter, developer, master*), desde que seja seu *master*

- Descrição: Um menu de *dropdown* em cada elemento da lista de membros de um projeto
- Dependências: US18
- Prioridade: *Should have*
- *Wireframe*: Não tem

US 22. Enquanto utilizador autenticado, devo poder determinar a prioridade de um projeto em que trabalho, usando um sistema de 3 níveis

- Descrição: 3 botões em cada elemento da lista de projetos em que trabalho
- Dependências: US13
- Prioridade: *Should have*
- *Wireframe*: Não tem

3.1.1.5 Tarefa

US 23. Enquanto utilizador autenticado, devo poder criar uma nova tarefa, para qualquer projeto, desde que seja seu *master* ou *developer*

- Descrição: Uma página com um formulário para o nome, descrição, data de início e data de fim da tarefa, bem como o projeto a que pertence e utilizadores destacados
- Dependências: US12
- Prioridade: *Must have*
- *Wireframe*: Não tem

US 24. Enquanto utilizador autenticado, devo poder listar todas as tarefas de todos os projetos em que trabalho

- Descrição: Uma página com uma lista de todas as tarefas de todos os projetos em que trabalho
- Dependências: US23
- Prioridade: *Must have*

- *Wireframe*: Não tem

US 25. Enquanto utilizador autenticado, devo poder listar todas as tarefas de um único projeto em que trabalho

- Descrição: Uma página com uma lista de todas as tarefas de um único projeto em que trabalho
- Dependências: US23
- Prioridade: *Must have*
- *Wireframe*: Figura A.6

US 26. Enquanto utilizador autenticado, devo poder visualizar informação detalhada sobre uma tarefa de um projeto em que trabalho

- Descrição: Uma página com informação detalhada sobre uma tarefa
- Dependências: US23
- Prioridade: *Must have*
- *Wireframe*: Figura A.7

US 27. Enquanto utilizador autenticado, devo poder atualizar a informação de qualquer tarefa para a qual estou destacado, desde que seja *developer* ou *master* do respetivo projeto

- Descrição: Uma página com um formulário para o nome, descrição, data de início e data de fim da tarefa, bem como o projeto a que pertence e utilizadores destacados
- Dependências: US23
- Prioridade: *Must have*
- *Wireframe*: Não tem

US 28. Enquanto utilizador autenticado, devo poder apagar qualquer tarefa para a qual estou destacado, desde que seja *developer* ou *master* do respetivo projeto

- Descrição: Um botão em cada elemento da lista de tarefas para as quais estou destacado
- Dependências: US23
- Prioridade: *Must have*
- *Wireframe*: Não tem

US 29. Enquanto utilizador autenticado, devo poder destacar-me para qualquer tarefa de um projeto em que trabalho

- Descrição: Um botão em cada elemento de uma lista de tarefas em que posso trabalhar, desde que não esteja já destacado para essa tarefa
- Dependências: US23
- Prioridade: *Must have*
- *Wireframe*: Não tem

US 30. Enquanto utilizador autenticado, devo poder abandonar uma tarefa para a qual estou destacado

- Descrição: Um botão em cada elemento de uma lista de tarefas em que posso trabalhar, desde que esteja destacado para essa tarefa
- Dependências: US23
- Prioridade: *Must have*
- *Wireframe*: Não tem

US 31. Enquanto utilizador autenticado, devo poder marcar uma tarefa como completa, desde que esteja destacado para essa tarefa

- Descrição: Um botão na página da tarefa
- Dependências: US23
- Prioridade: *Must have*
- *Wireframe*: Não tem

3.1.1.6 *Time entry*

US 32. Enquanto utilizador autenticado, devo poder começar uma nova *time entry*

- Descrição: Uma página com um formulário para a identificação do projeto e tarefa
- Dependências: US23
- Prioridade: *Must have*
- *Wireframe*: Não tem

US 33. Enquanto utilizador autenticado, devo poder começar uma nova *time entry*, para uma dada tarefa para a qual esteja destacado

- Descrição: Um botão em cada elemento de uma lista de tarefas para as quais esteja destacado
- Dependências: US23
- Prioridade: *Must have*
- *Wireframe*: Figura A.7

US 34. Enquanto utilizador autenticado, devo poder criar uma nova *time entry*, para para uma dada tarefa para a qual esteja destacado, definindo manualmente o período da *time entry*

- Descrição: Uma página com um formulário para a identificação do projeto e tarefa, bem como os seus momentos de início e fim
- Dependências: US23
- Prioridade: *Must have*
- *Wireframe*: Não tem

US 35. Enquanto utilizador autenticado, devo poder parar uma *time entry* em execução

- Descrição: Um botão em cada elemento de uma lista de tarefas para as quais esteja destacado, se tiver uma *time entry* em execução para essa tarefa
- Dependências: US32, US33, US34
- Prioridade: *Must have*
- *Wireframe*: Figura A.7

US 36. Enquanto utilizador autenticado, devo poder listar todas as minhas *time entries*

- Descrição: Uma página com uma lista com todas as minhas *time entries*
- Dependências: US32, US33, US34
- Prioridade: *Must have*
- *Wireframe*: Figura A.8

US 37. Enquanto utilizador autenticado, devo poder editar uma *time entry* minha, alterando os seus momentos de início e de fim

- Descrição: Uma página com um formulário para os momentos de início e fim da *time entry*
- Dependências: US32, US33, US34
- Prioridade: *Should have*
- *Wireframe*: Não tem

US 38. Enquanto utilizador autenticado, devo poder apagar uma *time entry* minha

- Descrição: Um botão em cada elemento da lista das minhas *time entries*
- Dependências: US32, US33, US34
- Prioridade: *Should have*
- *Wireframe*: Não tem

3.1.1.7 *Check-in*

US 39. Enquanto utilizador autenticado, devo poder fazer *check-in*, assinalando o início de uma sessão de trabalho

- Descrição: Botão de *check-in* nas várias páginas da plataforma
- Dependências: US4
- Prioridade: *Should have*
- *Wireframe*: Figura A.2

US 40. Enquanto utilizador autenticado, devo poder fazer *check-out*, assinalando o fim de uma sessão de trabalho

- Descrição: Botão de *check-out* nas várias páginas da plataforma
- Dependências: US39
- Prioridade: *Should have*
- *Wireframe*: Figura A.3

3.1.1.8 Questionário

US 41. Enquanto utilizador autenticado, devo poder criar um novo questionário para um projeto, de forma a recolher *feedback* dos seus membros, desde que seja *master* desse projeto

- Descrição: Uma página com um formulário para as questões do questionário
- Dependências: US23
- Prioridade: *Should have*
- *Wireframe*: Não tem

US 42. Enquanto utilizador autenticado, devo poder alterar o questionário de um projeto, desde que seja *master* desse projeto

- Descrição: Uma página com um formulário para as questões do questionário
- Dependências: US41
- Prioridade: *Should have*
- *Wireframe*: Não tem

US 43. Enquanto utilizador autenticado, fazendo *check-out*, devo poder responder aos questionários dos projetos em que trabalhei durante essa sessão de trabalho

- Descrição: Uma página com o questionário de cada projeto
- Dependências: US41
- Prioridade: *Should have*
- *Wireframe*: Figura A.3

US 44. Enquanto utilizador autenticado, devo poder listar as respostas ao questionário de um projeto, desde que seja *master* desse projeto

- Descrição: Uma lista com todas as respostas de um questionário
- Dependências: US43
- Prioridade: *Should have*
- *Wireframe*: Não tem

3.1.1.9 Notificação

US 45. Enquanto utilizador autenticado, devo poder listar as minhas notificações

- Descrição: Uma lista com todas as minhas notificações
- Dependências: US4
- Prioridade: *Should have*
- *Wireframe*: Não tem

US 46. Enquanto utilizador autenticado, devo receber uma notificação quando uma equipa a que pertença é apagada

- Descrição: Caixa de texto com o conteúdo da notificação
- Dependências: US8
- Prioridade: *Should have*
- *Wireframe*: Figure A.10

US 47. Enquanto utilizador autenticado, devo receber uma notificação quando o nome de uma equipa a que pertença é alterado

- Descrição: Caixa de texto com o conteúdo da notificação
- Dependências: US8
- Prioridade: *Should have*
- *Wireframe*: Figure A.10

US 48. Enquanto utilizador autenticado, devo receber uma notificação quando sou adicionado a uma equipa

- Descrição: Caixa de texto com o conteúdo da notificação
- Dependências: US8
- Prioridade: *Should have*
- *Wireframe*: Figure A.10

US 49. Enquanto utilizador autenticado, devo receber uma notificação quando sou removido de uma equipa

- Descrição: Caixa de texto com o conteúdo da notificação
- Dependências: US8
- Prioridade: *Should have*
- *Wireframe*: Figure A.10

US 50. Enquanto utilizador autenticado, devo receber uma notificação quando um projeto em que estou a trabalhar é apagado

- Descrição: Caixa de texto com o conteúdo da notificação
- Dependências: US12
- Prioridade: *Should have*
- *Wireframe*: Figure A.10

US 51. Enquanto utilizador autenticado, devo receber uma notificação quando o nome de um projeto em que estou a trabalhar é alterado

- Descrição: Caixa de texto com o conteúdo da notificação
- Dependências: US12
- Prioridade: *Should have*
- *Wireframe*: Figure A.10

US 52. Enquanto utilizador autenticado, devo receber uma notificação quando sou adicionado a um projeto

- Descrição: Caixa de texto com o conteúdo da notificação
- Dependências: US12
- Prioridade: *Should have*
- *Wireframe*: Figure A.10

US 53. Enquanto utilizador autenticado, devo receber uma notificação quando sou removido de um projeto

- Descrição: Caixa de texto com o conteúdo da notificação
- Dependências: US12
- Prioridade: *Should have*

- *Wireframe*: Figure A.10

US 54. Enquanto utilizador autenticado, devo receber uma notificação quando o meu tipo de participação num projeto é alterado

- Descrição: Caixa de texto com o conteúdo da notificação
- Dependências: US12
- Prioridade: *Should have*
- *Wireframe*: Figure A.10

US 55. Enquanto utilizador autenticado, devo receber uma notificação quando uma tarefa para a qual estou destacado é apagada

- Descrição: Caixa de texto com o conteúdo da notificação
- Dependências: US23
- Prioridade: *Should have*
- *Wireframe*: Figure A.10

US 56. Enquanto utilizador autenticado, devo receber uma notificação quando o nome ou descrição de uma tarefa para a qual estou destacado é alterado

- Descrição: Caixa de texto com o conteúdo da notificação
- Dependências: US23
- Prioridade: *Should have*
- *Wireframe*: Figure A.10

US 57. Enquanto utilizador autenticado, devo receber uma notificação quando sou destacado para uma tarefa

- Descrição: Caixa de texto com o conteúdo da notificação
- Dependências: US23
- Prioridade: *Should have*
- *Wireframe*: Figure A.10

US 58. Enquanto utilizador autenticado, devo receber uma notificação quando sou removido de uma tarefa

- Descrição: Caixa de texto com o conteúdo da notificação
- Dependências: US23
- Prioridade: *Should have*
- *Wireframe*: Figure A.10

US 59. Enquanto utilizador autenticado, devo receber uma notificação quando uma tarefa para a qual estou destacado é assinalada como completa

- Descrição: Caixa de texto com o conteúdo da notificação
- Dependências: US31
- Prioridade: *Should have*
- *Wireframe*: Figure A.10

3.1.1.10 Relatório

US 60. Enquanto utilizador autenticado, devo poder produzir relatórios de qualquer projeto em que esteja a trabalhar, desde que tenha permissão para tal (*reporter*, *developer* ou *master*)

- Descrição: Página com formulário com campos para projeto, equipa e utilizador, sendo que um ou mais destes campos podem ser usados para filtrar os dados a usar. O formulário também terá campos para o intervalo de tempo a que o relatório se refere
- Dependências: US23
- Prioridade: *Must have*
- *Wireframe*: Figure A.9

US 61. Enquanto utilizador autenticado, devo poder imprimir um relatório produzido pela plataforma

- Descrição: Botão para imprimir relatório
- Dependências: US60
- Prioridade: *Should have*
- *Wireframe*: Não tem

US 62. Enquanto utilizador autenticado, devo poder exportar um relatório produzido pela plataforma num formato comum (PNG ou PDF)

- Descrição: Menu *dropdown* na página do relatório
- Dependências: US60
- Prioridade: *Should have*
- *Wireframe*: Não tem

3.1.2 Requisitos não funcionais

Existem vários requisitos não funcionais que a aplicação deve respeitar, como:

3.1.2.1 Usabilidade

É fundamental que a plataforma ofereça uma interface de fácil utilização, de forma a que o utilizador complete as suas tarefas no mínimo tempo possível. Na tabela 3.1, para cada tarefa fundamental que o utilizador pode realizar, é definido o limite máximo de tempo em que se prevê que o utilizador realize essa tarefa.

Atividade	Duração
Registo	30 segundos
<i>Log in</i>	30 segundos
Criação de equipa	1 minuto
Edição de equipa	1 minuto
Eliminação de equipa	5 segundos
Criação de projeto	2 minutos
Edição de projeto	2 minutos
Eliminação de projeto	5 segundos
Criação de tarefa	1 minuto
Edição de tarefa	1 minuto
Eliminação de tarefa	5 segundos
Início de <i>time entry</i>	5 segundos
Fim de <i>time entry</i>	5 segundos
Criação de questionário	5 minutos
Preenchimento de questionário	3 minutos

Tabela 3.1: Duração prevista para a realização de cada tarefa na plataforma

3.1.2.2 Variabilidade

O número de plataformas disponíveis para a gestão de projetos tende a aumentar. Com cada nova plataforma, surgem várias novas funcionalidades, que facilitam o trabalho dos gestores de projeto e *developers*.

Sendo um projeto desenvolvimento no âmbito de um estágio de Mestrado, a quantidade de funcionalidades implementadas é relativamente reduzido. Por isso, a plataforma deve ser desenvolvida de forma a que novas funcionalidades possam ser facilmente adicionadas, sem a necessidade de *refactoring* intensivo.

3.1.2.3 Segurança

A plataforma contém informações pessoais de todos os utilizadores (nomeadamente dados de autenticação). Esta informação deve estar disponível apenas para o utilizador respetivo, e só o utilizador respetivo deverá poder alterá-la.

A informação relativa a equipas e projetos só deve ser disponibilizada aos utilizadores com autorização para tal, e só membros dessas equipas e projetos deverão ter os meios de dar ou retirar essa autorização a outros utilizadores da plataforma.

3.1.2.4 Desempenho

A cada dia que passa, espera-se que aplicações *web* respondam mais depressa a cada pedido. Esta plataforma não deverá ser exceção, e como tal é esperado que cada pedido obtenha resposta ao fim de poucos segundos. Todo e qualquer atraso verificado nas respostas deverá dever-se apenas a restrições da máquina onde a plataforma estiver alojada, e não à plataforma em si.

3.1.2.5 Portabilidade

A plataforma deve ser acessível de qualquer dispositivo com ligação à internet e *web browser*.

3.1.2.6 Compatibilidade

A plataforma deve ser acessível de qualquer um dos principais *web browsers* no mercado.

3.1.3 Restrições

- **Restrições de negócio** Deverá existir apenas um tipo de utilizador na plataforma. No entanto, ao associar um utilizador a um projeto, esta participação poderá ser de um de quatro tipos, cada um com diferentes níveis de permissões. Na tabela 3.2, cada tipo de participação tem as permissões do tipo anterior, para além das permissões destacadas.

Tipo de participação	Permissões
<i>Guest</i>	Apenas pode visualizar informação geral sobre o projeto (nome, descrição, membros)
<i>Reporter</i>	Pode gerar relatórios sobre o progresso do projeto, em termos de tarefas realizadas
<i>Developer</i>	Pode criar, alterar e eliminar tarefas e <i>time entries</i> no projeto
<i>Master</i>	Pode alterar as informações do projeto e alterar a lista de membros e suas participações

Tabela 3.2: Tipos de participação de um utilizador num projeto

- **Restrições técnicas**
 - A aplicação deverá ser desenvolvida usando a *framework Ruby on Rails*.
 - O Sistema de Gestão de Bases de Dados utilizado deverá ser *PostgreSQL*.

3.2 Arquitetura do sistema

3.2.1 Diagrama Entidade Relacionamento

No diagrama seguinte é possível analisar o diagrama ER que define a arquitetura da base de dados a construir para a aplicação.

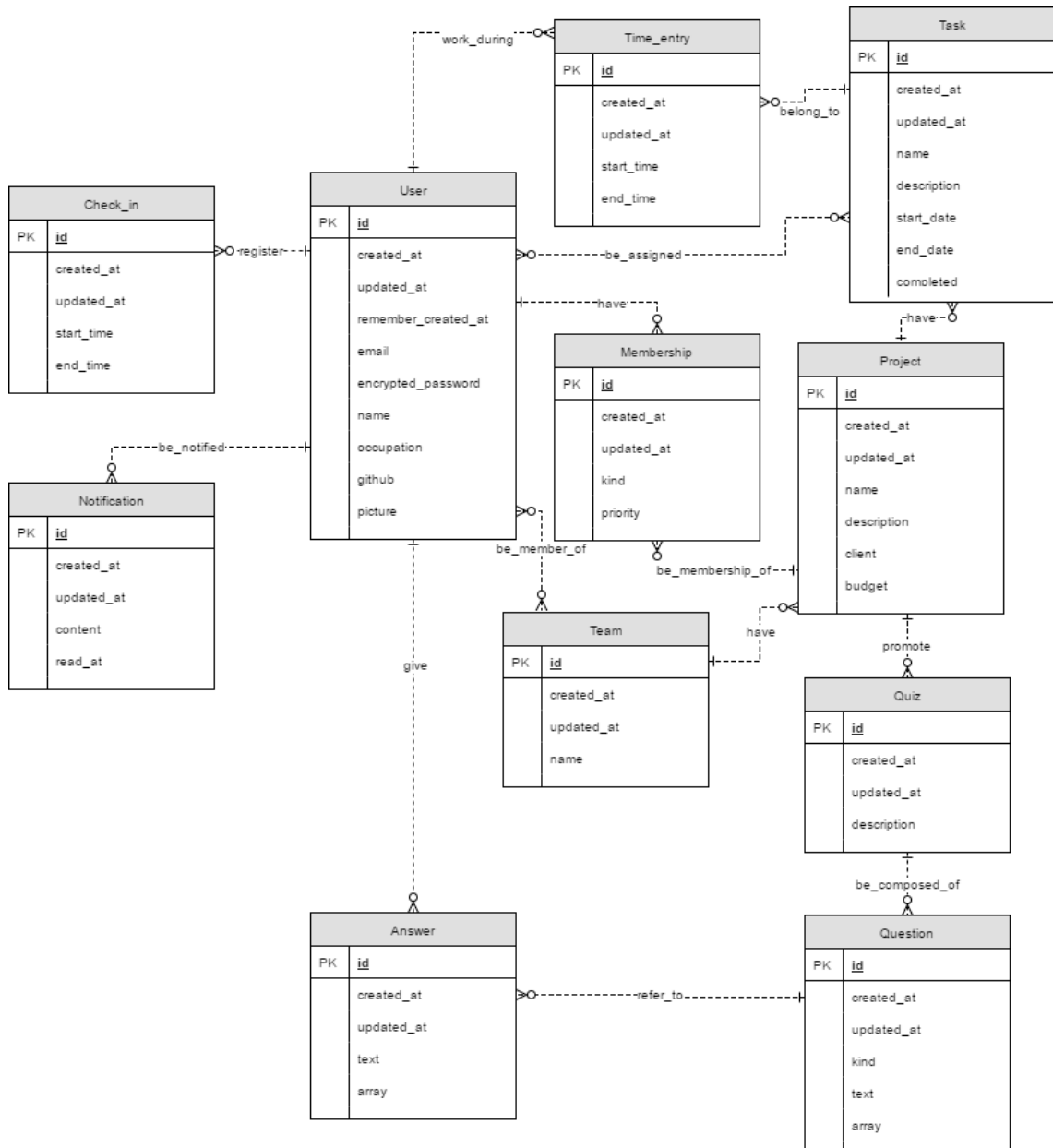


Figura 3.1: Diagrama ER da base de dados

- *User*
Utilizador da plataforma.
- *Team*
Equipa de utilizador, de que estes podem ser membros.

- *Project*
Um dos vários projetos que é possível gerir na aplicação, com membros associados e com tarefas a realizar.
- *Membership*
Indica que um user está associado a um projeto, e define o seu cargo no projeto (utilizador normal ou administrador, por exemplo).
- *Task*
Uma das tarefas a realizar num dado projeto, com uma data prevista para início e conclusão.
- *Time_entry*
Registo temporal. Define um intervalo de tempo em que um utilizador esteve a trabalhar numa dada tarefa.
- *Notification*
Notificação do sistema, que é apresentada ao utilizador sempre que necessário.
- *Check_in*
Check_in, define quando um utilizador iniciou e terminou uma sessão de trabalho.
- *Quiz*
Questionário a apresentar aos membros de um projeto sempre que fizerem *check-out* da aplicação.
- *Question*
Uma das várias questões do questionário. Tem um campo *kind* para indicar se se espera uma resposta curta, de parágrafo, ou com respostas pré-definidas, o texto a apresentar ao utilizador e, caso aplicável, um *array* com as várias respostas pré-definidas.
- *Answer*
Resposta dada por um utilizador a uma pergunta de um questionário.

3.2.2 Diagrama de *Containers*

Na figura 3.2 é possível analisar o diagrama de *containers* da plataforma a desenvolver. Como se pode verificar, trata-se de uma aplicação bastante trivial de *Ruby on Rails*, com um servidor *web* a correr a aplicação e ligação a uma base de dados relacional que armazenará os dados dos utilizadores e dos vários componentes da aplicação.

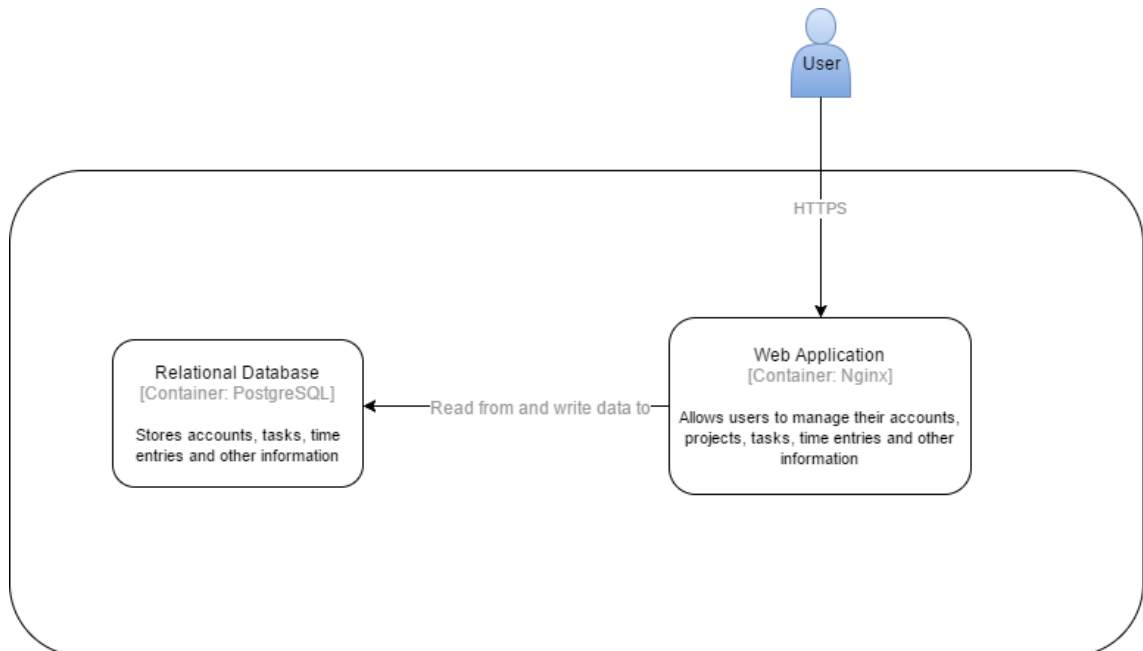


Figura 3.2: Diagrama de *Containers*

3.2.3 Diagrama de Componentes

O diagrama da figura 3.3 relaciona os vários componentes da aplicação entre si e de acordo com os *containers* definidos anteriormente. Os vários componentes foram escolhidos listando os vários modelos existentes na plataforma e identificado os que estão mais intrinsecamente ligados de forma a gerar uma componente modular da aplicação.

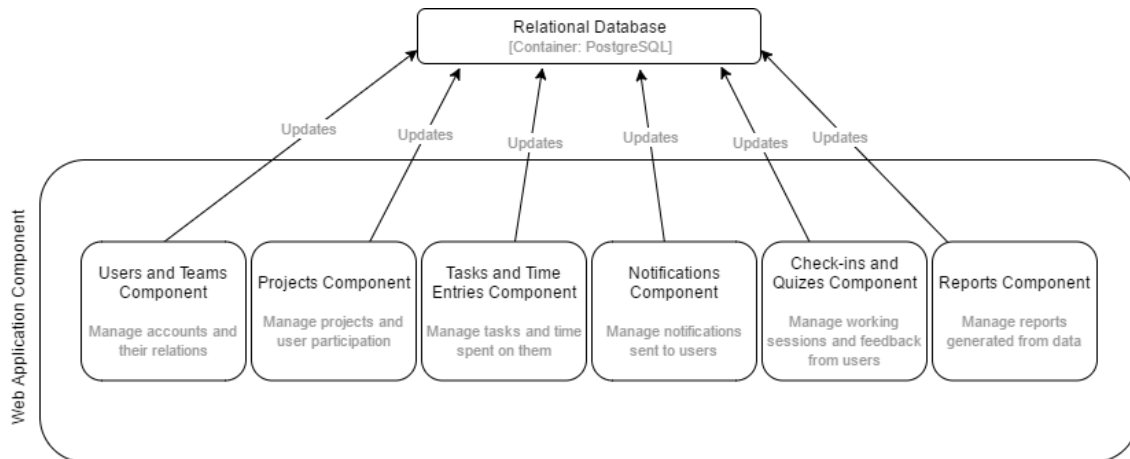


Figura 3.3: Diagrama de componentes

3.2.4 Diagrama de Navegação

De seguida, é apresentado um diagrama de navegação da plataforma, que explica como será o *flow* do utilizador dentro da mesma, desde a página inicial até aos seus vários projetos (no caso de ser um utilizador autenticado).

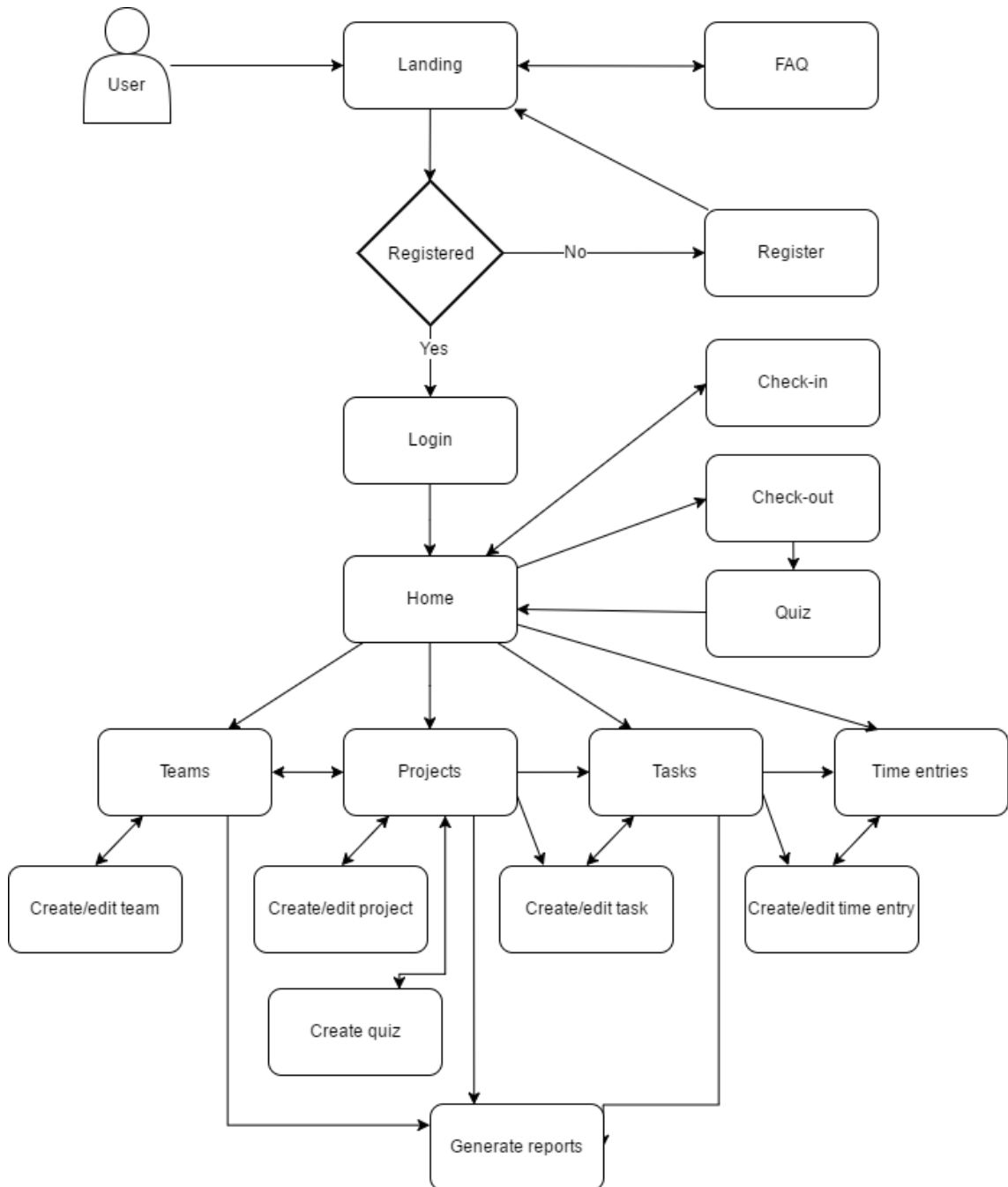


Figura 3.4: Diagrama de Navegação

3.3 Conclusão

No capítulo da descrição do sistema foram apresentados os vários requisitos levantados para o sistema a desenvolver, desde requisitos funcionais a restrições do sistema. Este levantamento é necessário para o desenvolvimento de um diagrama de navegação para a aplicação e para definição da sua arquitetura, que ainda não estão contemplados neste documento.

Ainda neste capítulo foi estudada arquitetura da aplicação a implementar, com o auxílio de um Diagrama Entidade Relacionamento a aplicar na criação da Base de Dados da aplicação, com a descrição das várias entidades para uma mais fácil compreensão, um Diagrama de *Containers* e um Diagrama de Componentes.

Capítulo 4

Planeamento

Neste capítulo serão apresentadas todas as decisões tomadas em relação ao planeamento do desenvolvimento deste projeto.

Inicialmente, será apresentada a escolha feita relativamente ao modelo de desenvolvimento a utilizar, de entre os vários apresentados no capítulo 2 deste documento, bem como uma breve explicação das razões que levaram à sua escolha.

Por último, serão enumeradas as várias escolhas tecnológicas que foram feitas antes de dar início ao desenvolvimento.

4.1 Modelo de desenvolvimento

No capítulo 2.3, foram analisados os modelos *Waterfall*, *Agile* e em Espiral. Para cada um, após a sua descrição, foi feita uma análise dos tipos de projeto a que melhor se adequa, bem como um estudo das suas vantagens e desvantagens.

O modelo *Waterfall* tem como grande vantagem a facilidade de aprendizagem e de aplicação. No entanto, não é um modelo de desenvolvimento adequado a projetos em que os requisitos possam ter que ser alterados, e apenas produz *software* funcional em fases tardias de desenvolvimento, o que dificulta a análise do estado do projeto em momentos intermédios.

Por outro lado, o modelo *Agile* permite a alteração dos requisitos a qualquer momento, e a apresentação de *software* inicial em etapas mais iniciais do desenvolvimento. Também é facilmente implementado, apesar de requerer uma melhor gestão durante todo o processo. A qualidade do código produzido pode ficar aquém das expectativas dos *stakeholders*, devido à pressão das metas definidas, e é produzida pouca documentação, o que também não é desejado num projeto de dissertação de mestrado.

Por último, o modelo em Espiral, sendo o que implica uma maior complexidade em termos de gestão, permite alterações aos requisitos do projeto a qualquer momento, e uma

vez que produz mais documentação do que o modelo *Agile* leva a que no final o levantamento de requisitos seja muito mais preciso. Aliado ao elevado fator de documentação, o modelo em Espiral também permite o desenvolvimento por módulos, levando à produção de vários protótipos intermédios, com diferentes níveis de complexidade.

Dadas estas considerações, o modelo de desenvolvimento de *software* a utilizar no desenvolvimento deste projeto será o modelo em Espiral. Este modelo permite:

- alteração dos requisitos do projeto após o seu levantamento inicial.
- divisão do desenvolvimento em módulos.
- implementação e estudo de vários protótipos, com diferentes níveis de complexidade.
- produção de uma grande quantidade de documentação.

O modelo *Waterfall* foi excluído pela rigidez do processo e por levar a que a produção de *software* funcional demore muito tempo. O modelo *Agile* foi excluído pela reduzida quantidade de documentação expectável, e pelo risco de o código gerado não apresentar a qualidade esperada pelos *stakeholders*.

4.2 Calendarização

4.2.1 Primeiro Semestre

Durante o primeiro semestre, começaram por ser definidos, em conjunto com a equipa da HYP, o âmbito e objetivos da aplicação a desenvolver, com uma definição das várias tarefas básicas que um utilizador da mesma deve poder desenvolver.

Estando definido o foco da aplicação, foi feito o estudo do estado da arte. Foram estudadas plataformas existentes no mercado com fins semelhantes aos esperados para a plataforma a desenvolver, ou com funcionalidades que pudessem ser incluídas na mesma e que não tivessem sido ainda consideradas. De seguida, foram estudados os vários tipos de teste a implementar e possíveis modelos de desenvolvimento a seguir.

Foi feita uma descrição mais pormenorizada do sistema, através do levantamento de requisitos e da geração de um diagrama Entidade Relacionamento.

4.2.2 Segundo Semestre

No diagrama de Gantt da figura 4.1 é possível identificar as tarefas planeadas para o segundo semestre, relativas à implementação da plataforma em mãos. Trata-se de uma calendarização de alto nível, pois em cada iteração do modelo em Espiral são definidos de forma mais concreta os objetivos dessa iteração, de acordo com os objetivos concluídos na iteração anterior.

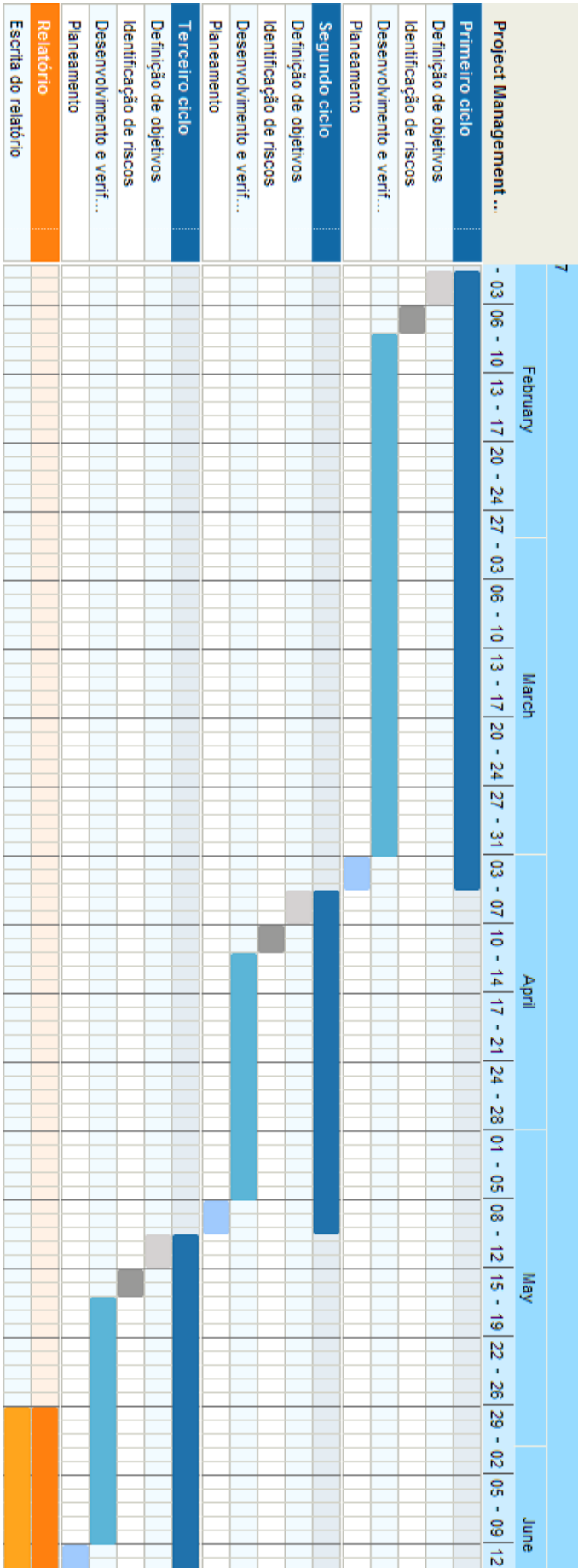


Figura 4.1: Diagrama de Gantt com calendarização do segundo semestre

4.3 Escolhas tecnológicas

Nesta secção, serão enumeradas várias escolhas feitas relativamente à realização deste projeto, nomeadamente quanto a linguagem de programação/*frameworks* a aplicar e Sistema de Gestão de Bases de Dados.

4.3.1 *Git*

O *Git*¹ foi a tecnologia escolhida para controlo de versões. Permite que vários utilizadores trabalhem em paralelo no mesmo projeto, com um repositório central para sincronização.

Ao longo deste projeto, foi usada no repositório a estrutura de *branches* presente na tabela 4.1.

<i>Branch</i>	Descrição
<i>master</i>	A <i>branch master</i> é a <i>branch</i> principal do projeto, usada apenas para versões estáveis do mesmo, ou seja, atualizada a cada nova <i>release</i> da plataforma.
<i>development</i>	<i>Branch</i> atualizada quando uma nova funcionalidade é implementada com sucesso. A <i>branch master</i> é atualizada com o conteúdo da <i>branch development</i> só e apenas se o código desta se encontrar devidamente testado e aprovado.
<i>design</i>	<i>Branch</i> usada para a estilização das várias páginas da plataforma. Esta <i>branch</i> deve ser atualizada com o conteúdo da <i>branch development</i> sempre que uma nova funcionalidade é implementada e as suas páginas estão preparadas para ser estilizadas.
<i>styleguide</i>	<i>Branch</i> de apoio, contém <i>templates</i> e <i>snippets</i> de vários elementos da interface, para fácil implementação na <i>branch design</i> .

Tabela 4.1: Modelo de *branches* do repositório *Git*

Para cada funcionalidade a implementar, promove-se a criação de uma nova *branch*, derivada da *branch development*, cujo nome deve explicitar a funcionalidade a implementar e deve seguir o modelo `<add_[feature_name]>`. Por exemplo, para implementar o sistema de utilizadores e autenticação, a *branch* poderá chamar-se `add_user_model`.

4.3.2 *Ruby on Rails*

Por restrição do cliente, a linguagem de desenvolvimento da aplicação será *Ruby*, nomeadamente através da *framework Ruby on Rails*².

¹Para mais informações, consultar <https://git-scm.com/>

²Para mais informações, consultar <http://rubyonrails.org/>

Ruby é uma linguagem de programação interpretada multi-paradigma. Inicialmente desenvolvida porque o seu autor, Yukihiro Matsumoto, sentia que não existia no mercado uma linguagem de programação interpretada verdadeiramente orientada a objetos[25], a sua primeira versão já incluía várias das características das versões mais recentes, como a orientação a objetos, classes com herança, tratamento de exceções e *garbage collection*.

Ruby on Rails é uma *framework* de desenvolvimento de aplicações *web* escrita em *Ruby*. *Rails* está desenhada como uma *framework* MVC; numa *framework* MVC, existem três componentes principais:

1. *Model* - o modelo é a componente central da aplicação. É independente da interface do utilizador, e lida diretamente com os dados armazenados pela aplicação e regras para a sua utilização.
2. *View* - a vista consiste numa qualquer representação da informação dos modelos, seja texto corrido, uma tabela ou um diagrama.
3. *Controller* - o controlador aceita o *input* dado pelo utilizador e transforma-o em comandos a aplicar aos modelos ou às vistas.

De forma a aplicar esta metodologia, em projetos de *Rails* é feita a seguinte divisão de ficheiros:

- *app*
 - *assets* - ficheiros auxiliares, como *Cascading Style Sheets* e *Javascript*.
 - *controllers* - os vários controladores da aplicação, como definidos anteriormente.
 - *models* - modelos; em *Rails* é usado o padrão *active record*.
 - *views* - vistas; podem ser de vários tipos, sendo o mais comum para uma aplicação web ficheiros *HTML*.
- *config* - ficheiros de configuração, entre definições de língua, *scripts* a definir como deve ser corrida a aplicação, e ficheiros de configuração da ligação à base de dados.
- *db* - ficheiros que definem toda a estrutura a aplicar à base de dados, e opcionalmente uma *seed*, ou seja, conteúdo inicial a inserir na base de dados.
- *lib* - bibliotecas usadas pela aplicação.
- *test* - ficheiros com o código usado nos testes à aplicação.

4.3.3 *PostgreSQL*

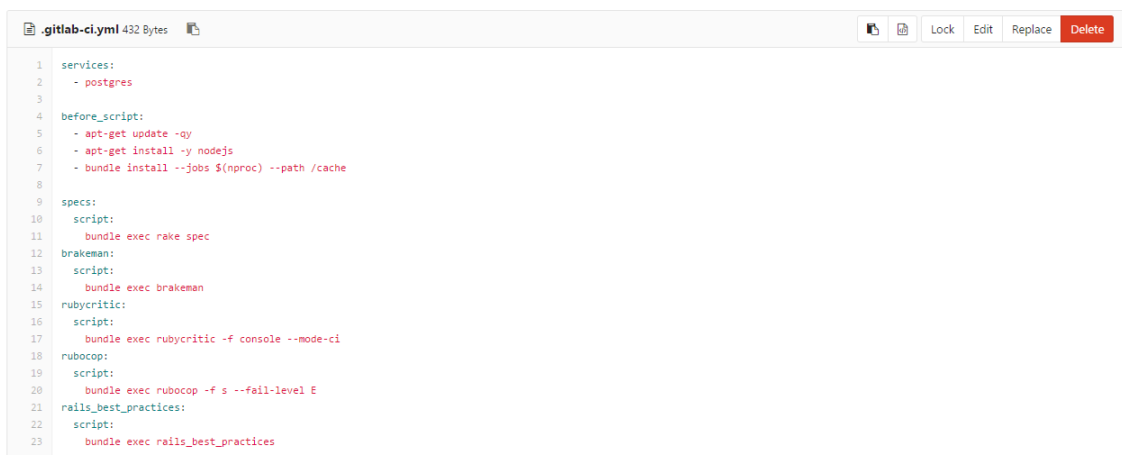
Também por restrição do cliente, o Sistema de Gestão de Bases de Dados utilizado é o PostgreSQL. O PostgreSQL é um sistema de gestão de bases de dados objeto-relacional, que se comporta fundamentalmente como um sistema de gestão de bases de dados relacional (baseado em tabelas de dados e na sua relação), mas com suporte para classes de objetos e herança entre os mesmos.

4.4 Ferramentas escolhidas

Nesta secção, serão enumeradas várias ferramentas escolhidas para o desenvolvimento deste processo, como servidor de *Version Control System* e ferramenta de gestão de *issues*.

4.4.1 Gitlab

O servidor usado para VCS é o *Gitlab*³, servidor usado pelo cliente e que oferece várias funcionalidades para além do controlo de versões, como a integração de um serviço de *Continuous Integration* (um exemplo de configuração encontra-se na figura 4.2), que passa o código por um conjunto de testes sempre que é feito um *commit*.



```
.gitlab-ci.yml 432 Bytes
1  services:
2    - postgres
3
4  before_script:
5    - apt-get update -qy
6    - apt-get install -y nodejs
7    - bundle install --jobs $(nproc) --path /cache
8
9  specs:
10   script:
11     - bundle exec rake spec
12  brakeman:
13   script:
14     - bundle exec brakeman
15  rubycritic:
16   script:
17     - bundle exec rubycritic -f console --mode-ci
18  rubocop:
19   script:
20     - bundle exec rubocop -f s --fail-level E
21  rails_best_practices:
22   script:
23     - bundle exec rails_best_practices
```

Figura 4.2: Configuração da ferramenta de *Continuous Integration*

Como se pode ver na figura 4.2, o sistema de CI do *Gitlab* é usado não só para correr os vários testes implementados com o uso de *specs*⁴, mas também para automatizar várias tarefas acessórias:

- *brakeman*⁵ - biblioteca de *Ruby on Rails* usada para detetar vulnerabilidades em aplicações desenvolvidas com a *framework*.
- *rubycritic*⁶ - biblioteca usada para analisar código de *Ruby on Rails* e identificar *code smells*, que podem dar origem a erros, gerando um relatório de qualidade do projeto.
- *RuboCop*⁷ - biblioteca usada para analisar código de *Ruby on Rails* e determinar se as várias *guidelines* da linguagem são cumpridas.
- *rails_best_practices*⁸ - alerta para a existência de código redundante ou não utilizado.

³Para mais informações, consultar <https://gitlab.com/>

⁴Para mais informações, consultar <http://rspec.info/>

⁵Para mais informações, consultar <https://github.com/presidentbeef/brakeman>

⁶Para mais informações, consultar <https://github.com/whitesmith/rubycritic>

⁷Para mais informações, consultar <https://github.com/bbatsov/rubocop>

⁸Para mais informações, consultar https://github.com/flyerhzm/rails_best_practices

4.4.2 *ShareLaTeX*

Para a escrita deste documento e de toda a documentação associada ao projeto, convencionou-se a utilização de *Latex*, por permitir a escrita de documentos com aspeto profissional através da utilização de comandos simples. De forma a permitir o acesso ao documento a partir de qualquer máquina, é utilizado o *ShareLaTeX*⁹.

Internamente, o projeto encontra-se dividido em três pastas principais:

1. *bibliography* - esta pasta contém apenas um ficheiro, com as várias referências usadas ao longo do documento, no formato *BibTeX*¹⁰.
2. *document* - esta é a pasta principal do documento. No seu interior, contém três sub-pastas:
 - (a) *constants* - nesta pasta existe um único ficheiro, com valores utilizados repetidamente ao longo do documento, como o nome do autor ou título do projeto.
 - (b) *figures* - nesta pasta são armazenadas as várias imagens presentes no documento, em sub-pastas correspondentes aos vários capítulos.
 - (c) *structure* - nesta pasta encontra-se o texto do documento. A pasta contém uma sub-pasta para cada capítulo do documento, e nestas sub-pastas existe um ficheiro para cada secção do capítulo.
3. *template* - nesta pasta encontra-se um único ficheiro, com a definição de várias estruturas usadas no documento, como a capa e página de rosto.

Todos os ficheiros listados são agrupados através do ficheiro *main.tex*, que se encontra na raiz do projeto.

4.4.3 *Trello*

Tratando-se o próprio projeto em mãos de um projeto de *software*, existe um conjunto de tarefas a realizar, e haverá a necessidade de gerir a sua concretização. Apesar de apenas estar um *developer* associado ao projeto, cada funcionalidade da aplicação deverá passar por várias etapas, e é necessário manter um registo deste fluxo.

Com o objetivo de auxiliar à gestão de tarefas, foi escolhido o *Trello*, ferramenta estudada na secção 2.2.3 deste documento. Será também implementada a distribuição de listas proposta nessa secção, com as listas *Backlog*, *In Progress*, *Code Review*, *Quality Assurance* e *Completed/Deployed*. Além disso, serão usadas as labels da ferramenta para identificar as diferentes prioridades das funcionalidades.

Uma vez que se trata de um projeto desenvolvido por um aluno de Mestrado em Engenharia Informática em conjunto com uma aluna de Mestrado em Design e Multimédia, o *Trello* também facilita a comunicação entre os dois, em termos de gestão de tarefas realizadas e a realizar.

⁹Para mais informações, consultar <https://www.sharelatex.com/>

¹⁰Para mais informações, consultar <http://www.bibtex.org/>

4.4.4 Heroku

Para *deployment* da plataforma, será usado o *Heroku*. Trata-se de uma *Platform as a Service* que permite o *deployment* de aplicações em várias linguagens de programação diferentes, entre elas *Ruby*. Usando *Heroku*, todo o processo de *deployment* é facilitado, uma vez que apenas é necessário criar um *Procfile*, que consiste num ficheiro de configuração que indica quais os processos necessários para a aplicação funcionar.

Na sua modalidade gratuita, o *Heroku* apenas permite a execução de um processo. No entanto, como o Sistema de Gestão de Bases de Dados é fornecido como um *addon* gratuito, apenas é necessária a execução do servidor de *Ruby on Rails* (no caso da plataforma em mãos, *Puma*).

Usando a suite de ferramentas oferecidas pela equipa do *Heroku*, para lançar uma nova versão de uma aplicação basta, com *Git*, fazer *push* para um repositório próprio. A aplicação é compilada automaticamente de acordo com um *buildpack* escolhido no *website* e são iniciados os processos definidos no *Procfile*.

Existem algumas considerações a ter ao usar a versão gratuita de *Heroku*, uma vez que se trata de uma máquina virtual partilhada por várias aplicações. Apesar de não se esperarem problemas a nível de segurança (devido aos vários testes realizados à plataforma), podem existir limitações a nível da *performance* das aplicações. Este fator deve ser tido em conta ao realizar testes de *performance* à aplicação.

4.5 Conclusão

Neste capítulo foram apresentadas algumas decisões tomadas relativamente à forma como este projeto será levado a cabo.

Inicialmente, foi escolhido o modelo de desenvolvimento a aplicar, concluindo-se que a melhor opção seria o modelo em Espiral, devido às suas vantagens relativamente à possibilidade de reavaliar os requisitos após o seu levantamento inicial e à possibilidade de desenvolvimento de protótipos com vários níveis de complexidade antes de o produto final estar concluído.

Foram também apresentadas algumas escolhas tecnológicas levadas a cabo, maioritariamente por restrição do cliente, como a linguagem de programação, *framework* para desenvolvimento *web* e Sistema de Gestão de Bases de Dados a utilizar.

Capítulo 5

Desenvolvimento

No capítulo do desenvolvimento é analisado o processo de desenvolvimento dos vários componentes da plataforma, com principal foco em:

- componente de gestão de utilizadores e equipas
- componente de projetos
- componente de gestão de tarefas e *time entries*
- componente de notificações
- componente de *check-ins* e questionários
- componente de geração de relatórios

5.1 Gestão de utilizadores e equipas

O componente de gestão de utilizadores e equipas engloba as funcionalidades necessárias para a criação e manutenção de contas/perfis, acesso ao sistema e gestão de equipas de desenvolvimento. As funcionalidades de gestão de utilizadores e contas foram desenvolvidas primeiro, e depois foram implementadas as funcionalidades de gestão de equipas por cima das mesmas.

Para a gestão de contas de utilizador foi usada a biblioteca de *Ruby on Rails devise*¹.

Em primeiro lugar, gerou-se o modelo *default* de utilizador usado na biblioteca, com os atributos *e-mail* e *encrypted password*, ao qual foram acrescentados os campos nome e foto de perfil. Usando *devise*, as *passwords* de utilizador nunca são guardadas em *plain text*, o que seria uma enorme falha de segurança.

Ao registar-se, o utilizador deve usar um nome único (na verdade, o nome funciona como um comum *username*, e não como o nome próprio do utilizador). O *e-mail* usado

¹Para mais informações, consultar <https://github.com/plataformatec/devise>

também deve ser válido e único, e a *password* deve ter um comprimento entre 6 e 128 caracteres. Se um utilizador introduzir valores inválidos em algum dos atributos, será exibida uma mensagem de erro e o pedido de registo deve ser repetido. Se o utilizador assim o desejar, os seus dados de *login* podem ser armazenados pelo *web browser*, de forma a facilitar o processo em futuros acessos ao sistema.

Para a edição do perfil de utilizador, deverá ser filtrada a informação a mostrar e informação que pode ser alterada pelo utilizador. Por exemplo, a *password* não pode ser mostrada ao utilizador, mesmo que encriptada, e o *e-mail* não pode ser alterado após o registo, visto que é o identificador único desse utilizador.

Existindo um modelo válido de utilizador, que identifique unicamente cada utilizador do sistema, e havendo um mecanismo de início e manutenção de sessão, é fácil em todos os restantes componentes da plataforma limitar o acesso à informação a utilizadores com sessão iniciada ou com alguma ligação direta à informação requerida. Por exemplo, a página de perfil apenas será mostrada se o utilizador tiver sessão iniciada, e cada utilizador apenas poderá aceder à sua página de perfil.

Relativamente à gestão de equipas, foi criado um modelo simples de *Ruby on Rails*, que pode ser ligado ao modelo de utilizador anteriormente criado para relacionar utilizadores entre si e permitir o acesso a informação a vários utilizadores de uma só vez, em vez de essa permissão ter que ser dada individualmente.

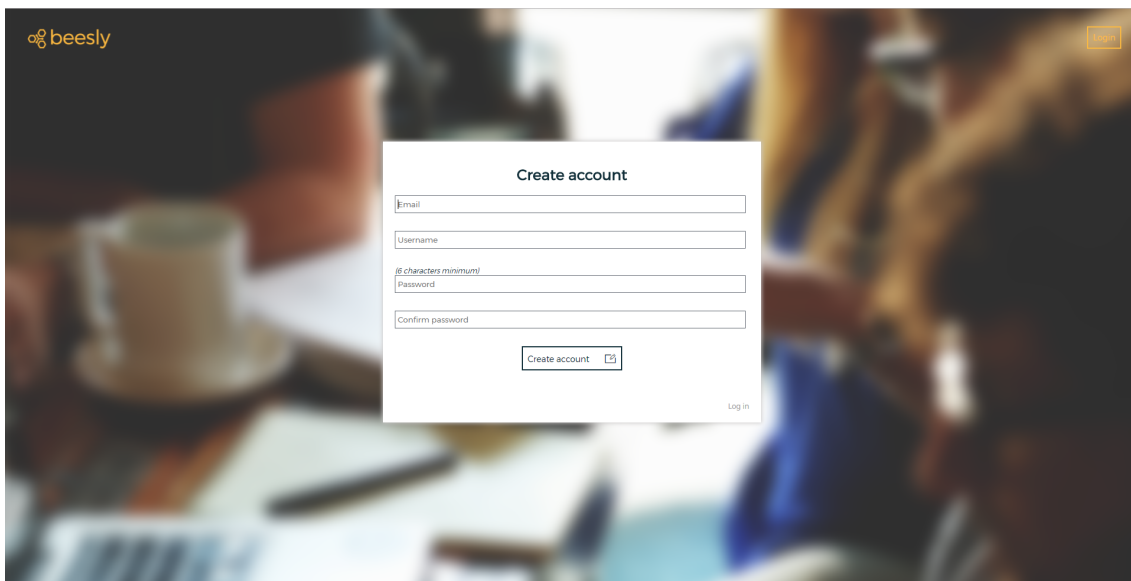


Figura 5.1: Screenshot - Registo na aplicação

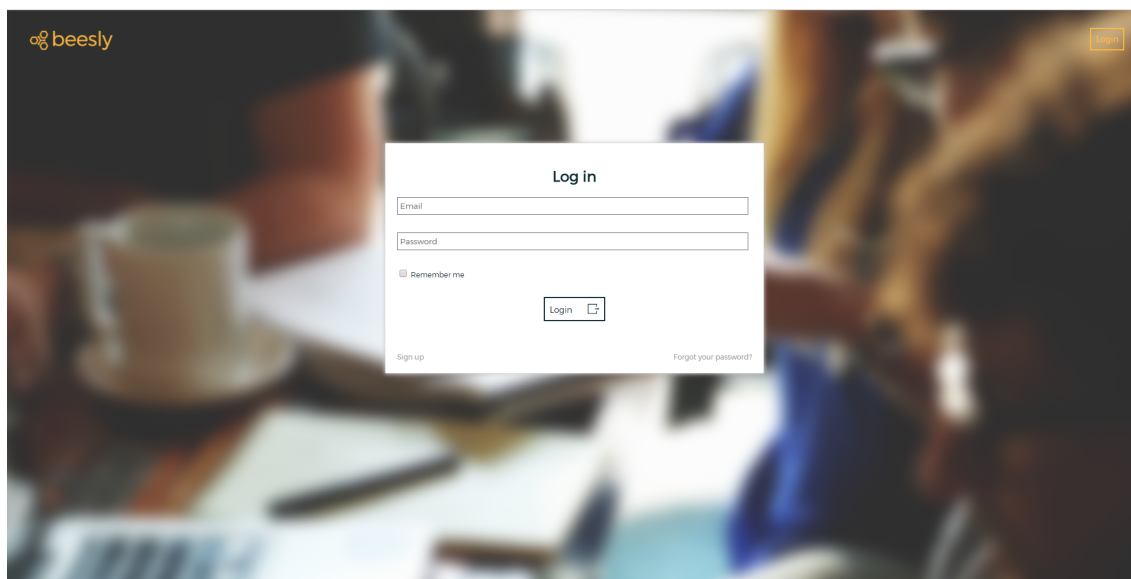


Figura 5.2: Screenshot - Login na aplicação



Figura 5.3: Screenshot - Edição de perfil

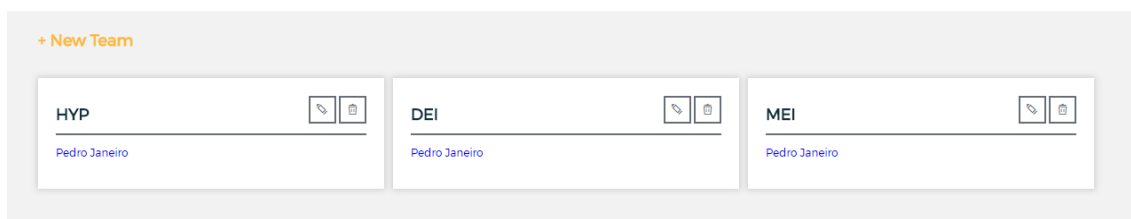


Figura 5.4: Screenshot - Equipas

5.2 Gestão de projetos

O componente de gestão de projetos pode ser considerado o componente mais importante da plataforma. Todos os outros componentes estão de alguma forma relacionados com o componente de gestão de projetos, nomeadamente:

- Componente de gestão de utilizadores e equipas - os utilizadores trabalham em projetos, e gerem os mesmos; de facto, todas as operações aplicadas aos projetos requerem a existência de utilizadores com permissão para as realizar.
- Componente de gestão de tarefas e *time entries* - as tarefas pertencem aos projetos, e são diretamente dependentes dos mesmos (não é possível criar tarefas que não pertençam a um projeto).
- Componente de notificações - muitas das operações realizadas sobre os projetos e seus componentes levam ao envio de notificações aos utilizadores envolvidos.
- Componente de *check-ins* e questionários - os questionários são criados para projetos específicos, isto é, os *masters* de cada projeto criam os questionários para obter *feedback* da equipa sobre esse projeto específico, e sobre os avanços verificados no mesmo.
- Componente de geração de relatórios - os relatórios são gerados relativamente a projetos, utilizadores que trabalhem nos projetos, e tarefas dos projetos.

Para a criação do modelo de projeto, é recolhida a informação básica do mesmo (nome e descrição), bem como informação auxiliar que poderá ser útil com a introdução de novas funcionalidades (cliente e *budget*).

Apesar de o modelo de projeto ser relativamente simples, é também introduzido o modelo de participação no projeto (*Membership* do diagrama ER da figura 3.1). Este modelo irá ligar utilizadores e projetos na plataforma, e define os vários tipos de acesso que um utilizador tem ao projeto. Dependendo do tipo de acesso do utilizador, mudam as operações que o utilizador pode executar nesse projeto, nas suas tarefas, e na geração de relatórios sobre o projeto. A participação permite também ao utilizador definir a prioridade que dá ao projeto, entre três valores possíveis, de forma a ordenar os mesmos quando os está a listar.

De forma a evitar o uso de tipos de participação inválidos, é usada uma *enum*, o que quer dizer que o campo em questão apenas pode ter um de 4 valores disponibilizados, como se de um novo tipo de dados se tratasse.

Uma vez que vários elementos da plataforma dependem diretamente do modelo de projeto, foi importante assegurar que, quando um projeto é eliminado, todos os elementos dele dependentes são também eliminados. Portanto, ao eliminar um projeto, são eliminadas as suas tarefas e respetivas *time entries*, as suas participações de utilizadores, e os seus questionários (e respetivas respostas).

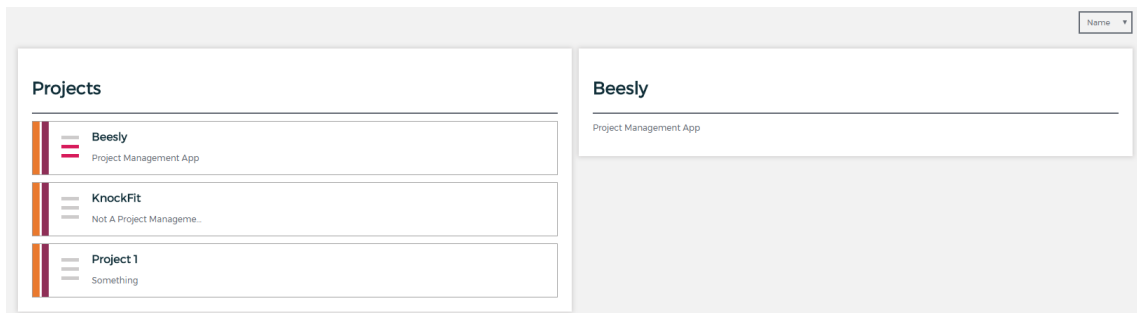


Figura 5.5: Screenshot - Projetos



Figura 5.6: Screenshot - Editar projeto

5.3 Gestão de tarefas e *time entries*

O componente de gestão de tarefas e *time entries* é um dos principais focos desta plataforma, e de várias outras plataformas apresentadas na secção 2.2. Uma tarefa comporta-se como um dos vários blocos que compõem um projeto de *software* e como tal devem estar explicitamente ligadas a esse projeto: não é possível criar uma tarefa sem um projeto ao qual ela pertença; não também possível criar *time entries* sem uma tarefa à qual esta pertença, é também impossível criar *time entries* sem um projeto ao qual esta pertença (através da tarefa).

O modelo da tarefa tem campos para nome e descrição, de forma a ser facilmente identificada pelo utilizador, data prevista para início e fim da tarefa, e data de conclusão da mesma (esta informação é útil para a geração de relatório e para melhor controlo do progresso de um projeto).

Uma vez que o utilizador deve ser capaz de listar as tarefas segundo várias regras diferentes, foram criados métodos extra no Controlador das tarefas, e foram geradas as *routes* da tabela 5.1, cada uma com a sua finalidade:

Método	Route	Finalidade
<i>tasks#index</i>	<i>/tasks</i>	Lista todas as tarefas de projetos em que o utilizador trabalha
<i>tasks#subscribed</i>	<i>/tasks/subscribed</i>	Lista todas as tarefas para as quais o utilizador está destacado
<i>tasks#to_start</i>	<i>/tasks/to_start</i>	Lista todas as tarefas para as quais o utilizador está destacado cuja data de início se esteja a aproximar ("próximas tarefas")
<i>tasks#completed</i>	<i>/tasks/completed</i>	Lista todas as tarefas que o utilizador já completou
<i>project#tasks</i>	<i>/projects/:id/tasks</i>	Lista todas as tarefas de um projeto

Tabela 5.1: Métodos e *routes* criados para listar tarefas segundo várias regras diferentes

O modelo de *time entry* é mais simples, apenas com os campos que representam os momentos de início e fim da mesma. No entanto, a utilização de *time entries* implica alguma sincronização, caso se pretenda que o utilizador saiba, a cada momento, há quanto tempo a *time entry* foi criada, sem ter que terminar a mesma. O início e fim de uma *time entry* são feitos recorrendo a pedidos assíncronos, com *Javascript* e *AJAX*, pelo que a *time entry* só é efetivamente iniciada quando é recebida uma confirmação do servidor, e é usada a *timestamp* devolvida pelo servidor. A partir daí, periodicamente, o cliente deve atualizar o tempo de execução indicado pela *time entry*, calculando a diferença entre o momento atual e o momento em que esta foi iniciada. Sempre que é carregada uma página em que esteja presente uma *time entry* ativa, é necessário iniciar automaticamente este procedimento.

Uma limitação detetada foi, devido à utilização apenas de *timestamp* de início e fim, a impossibilidade de pausar uma *time entry* e retomar a mesma. Um comportamento

semelhante pode ser obtido terminando a *time entry* e iniciando uma nova *time entry* para a mesma tarefa mais tarde, mas certamente que a possibilidade de pausar deverá ser tida em conta no futuro.

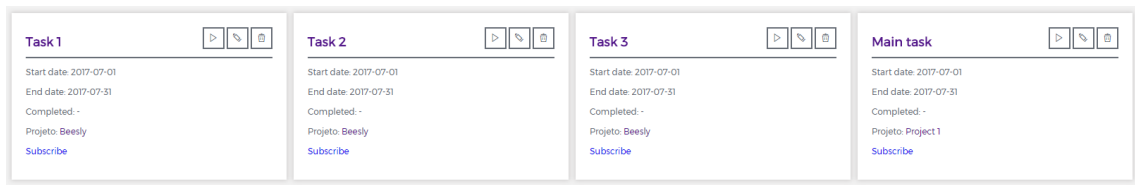


Figura 5.7: Screenshot - Tarefas

The screenshot shows a 'Create new task' form with the following fields and controls:

- Task name:** A text input field.
- Task description:** A large text area.
- Start date:** A date input field with the placeholder 'DD/MM/YYYY'.
- End date:** A date input field with the placeholder 'DD/MM/YYYY'.
- Knock...:** A dropdown menu.
- New task:** A checkbox that is currently checked.

Figura 5.8: Screenshot - Criar tarefa

The screenshot shows a table titled 'Time Entries' with the following data:

Task	Start time	End time	Duration
New task	2017-07-11 14:00:00 UTC	2017-07-11 18:45:00 UTC	04:45:00
First task	2017-07-11 10:05:00 UTC	2017-07-11 10:55:00 UTC	00:50:00
New task	2017-07-10 14:30:00 UTC	2017-07-10 18:30:00 UTC	04:00:00
First task	2017-07-10 10:00:00 UTC	2017-07-10 11:45:00 UTC	01:45:00
New task	2017-07-09 14:00:00 UTC	2017-07-09 18:30:00 UTC	04:30:00
New task	2017-07-08 14:00:00 UTC	2017-07-08 18:30:00 UTC	04:30:00
New task	2017-07-08 10:00:00 UTC	2017-07-08 12:00:00 UTC	02:00:00

Figura 5.9: Screenshot - Time entries

5.4 Gestão de notificações

As notificações são enviadas aos utilizadores sempre que este deve ser alertado da realização de uma ação por parte de outro utilizador (alteração de um projeto ou eliminação de uma tarefa, por exemplo). Idealmente, o utilizador recebe a notificação no momento em que esta é enviada, sem a necessidade de atualizar o *web browser*. Para atingir esse objetivo, existem duas opções:

- *Polling*
- *Websockets*

Polling consiste em enviar pedidos ao servidor, de forma cíclica, de forma a determinar a existência ou ausência de novas notificações (neste caso em concreto). Implementar *polling* manualmente implica criar um ciclo que faz um novo pedido ao servidor a cada iteração. Apesar de este pedido poder ser assíncrono, e não influenciar o funcionamento do resto da aplicação, pode aumentar a quantidade de recursos gastos pelo cliente, e com um elevado número de clientes a verificar se existem notificações rapidamente se atingia o limite de largura de banda do servidor.

Usando *websockets*, por outro lado, é criado um canal de comunicação bidirecional sobre uma única ligação TCP. Desta forma, o servidor pode enviar dados para o cliente sem a necessidade deste realizar pedidos, diminuindo o *overhead* da operação.

O método escolhido, portanto, é o uso de *websockets*, através da biblioteca *actioncable*². Esta biblioteca permite a implementação de *websockets* sem a necessidade de servidores auxiliares. Desta forma, é possível manter o *deployment* numa instância gratuita de *Heroku*[26].

O modelo da notificação, para além do conteúdo da mesma, tem um campo com a *timestamp* em que esta foi lida, que é definido no momento em que o utilizador clica na notificação na interface. Esse campo é usado para determinar se uma notificação foi lida ou não.

²Para mais informações, consultar http://edgeguides.rubyonrails.org/action_cable_overview.html

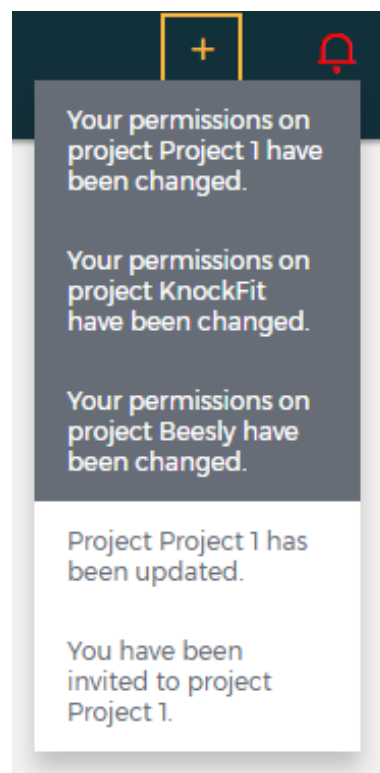


Figura 5.10: *Screenshot* - Notificações

5.5 Gestão de *check-ins* e questionários

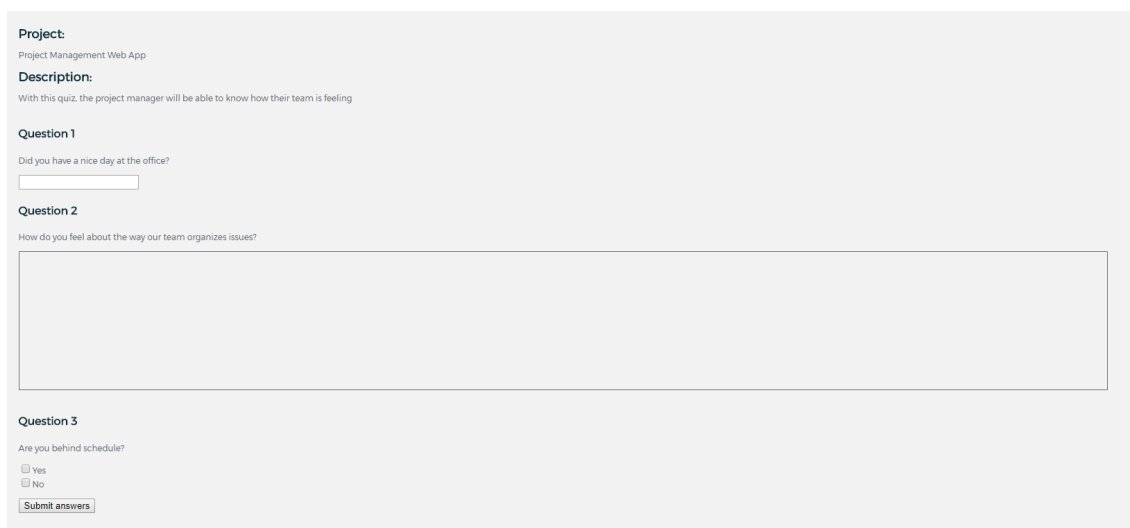
Os *check-ins* e questionários são componentes relativamente simples da plataforma, mas que permitem a recolha de *feedback* por parte do gestor de um projeto de, por exemplo, o grau de satisfação dos trabalhadores desse projeto.

O modelo de *check-in* é em tudo semelhante ao modelo de *time entry*. De facto, um *check-in* pode ser considerado uma *time entry*, que não depende de uma tarefa mas apenas do utilizador. Define uma sessão de trabalho, em que o utilizador pode trabalhar em várias tarefas, de vários projetos.

Ao fazer *check-out*, são analisadas as *timestamps* de início e de fim do *check-in*, e determinam-se as tarefas em que o utilizador trabalhou durante essa sessão de trabalho. A partir daí, é possível perceber em que projetos trabalhou, e são invocados os questionários existentes para esses projetos.

Os questionários são criados pelos *masters* de cada projeto, e a criação de um novo questionário implica a eliminação de qualquer questionário previamente existente para esse projeto (apenas é usado o questionário mais recente). O modelo de questionário apenas tem um campo para a descrição do mesmo, um pequeno texto introdutório que o *master* pretenda transmitir aos trabalhadores, possivelmente indicando o objetivo do questionário.

Dentro de um questionário, podem existir várias perguntas, que terão um tipo associado (texto, parágrafo, escolha múltipla), o texto da pergunta em si, e um campo que pode incluir todas as respostas possíveis para uma pergunta de escolha múltipla, sob a forma de um *array*. Da mesma maneira, uma resposta pode conter texto, caso se trate de uma pergunta de texto ou de parágrafo, ou um *array* com as opções escolhidas, no caso de uma pergunta de escolha múltipla.



Project:
Project Management Web App

Description:
With this quiz, the project manager will be able to know how their team is feeling

Question 1
Did you have a nice day at the office?

Question 2
How do you feel about the way our team organizes issues?

Question 3
Are you behind schedule?
 Yes
 No

Figura 5.11: Screenshot - Questionário

5.6 Gestão de relatórios

Um relatório consiste, no contexto do projeto, de um gráfico que permita visualizar a quantidade de tarefas realizadas ao longo do tempo para um projeto, uma equipa, um utilizador, ou vários destes fatores. Relembrando os tipos de participação num projeto existentes, os relatórios podem ser gerados por *reporters*, *developers* e *masters*, sendo os *guests* os únicos utilizadores a não ter este privilégio.

Para a geração dos relatórios é usada a biblioteca *Highcharts*³. Tratando-se, a este ponto, de um projeto realizado em âmbito escolar, esta biblioteca pode ser usada sem qualquer custo, sendo no entanto necessário considerar a utilização de uma alternativa no futuro, ou o pagamento de uma licença.

Para gerar um relatório, o utilizador apenas deve inserir os filtros a aplicar (em termos de projetos, equipas e utilizadores), e é devolvido pelo servidor um gráfico, que pode ser exportado para conveniência do utilizador.

Não existe um modelo para os relatórios, uma vez que se trata de uma análise realizada a outros elementos da plataforma já existentes.

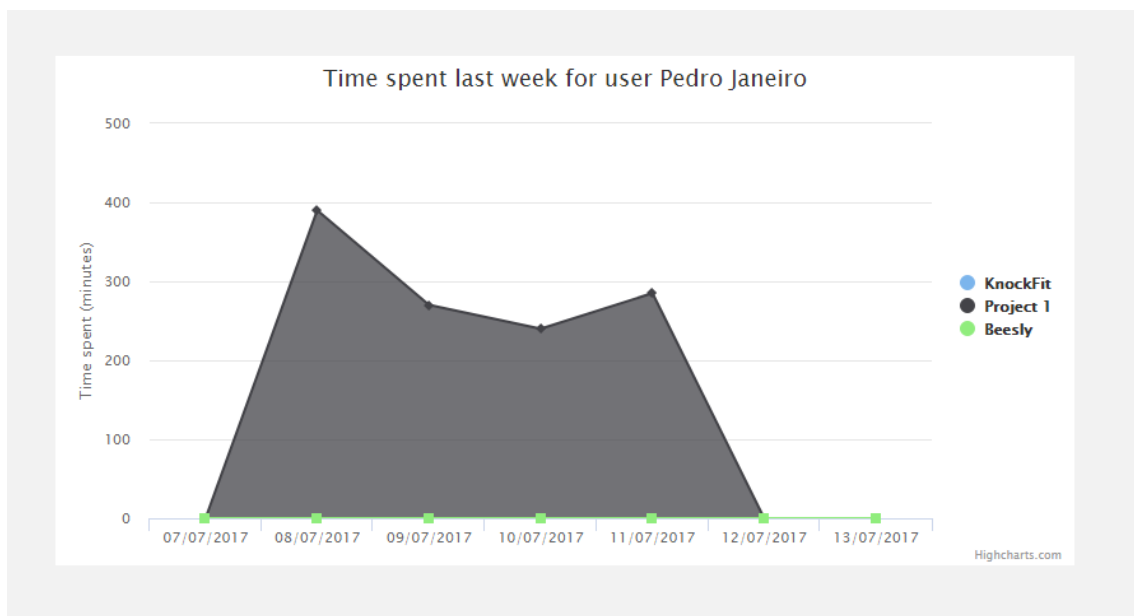


Figura 5.12: Screenshot - Relatório

³Para mais informações, consultar <https://www.highcharts.com/demo>

5.7 Ambiente de desenvolvimento

Para o desenvolvimento da aplicação, foram usados *docker*⁴ e *docker-compose*.

Docker é uma plataforma baseada no uso de *containers*: ao contrário de uma máquina virtual, um *container* não implica a virtualização de um sistema operativo diferente para cada projeto a desenvolver; em vez disso, todos os projetos usam o mesmo sistema operativo base, e é usado um *container* que contém as bibliotecas e definições necessárias para o *software* funcionar. Assim, novos sistemas são muito mais leves do que usando máquinas virtuais, e garante-se uma maior uniformidade em máquinas diferentes.

Docker-compose é uma ferramenta usada para associar vários *containers* de *docker* de forma a correr aplicações *multi-container*. O caso de uso mais comum (e o utilizado neste projeto) é quando um projeto requer um *container* que contenha um servidor de uma qualquer linguagem e um *container* com um Sistema de Gestão de Bases de Dados. No entanto, sistemas mais complexos podem ser criados com *docker* e *docker-compose*, por exemplo para simular várias redes de computadores para fazer testes de segurança.

Nas figuras 5.13 e 5.14 é possível ver a configuração usada para o sistema de *docker* usado para o desenvolvimento deste projeto.

⁴Para mais informações, consultar <https://www.docker.com/>

```
1 FROM ruby:2.4
2
3 ENV HOME /home/user
4 RUN useradd --create-home --home-dir $HOME user \
5     && chown -R user:user $HOME
6 RUN chown -R user:user $GEM_HOME
7
8 RUN apt-get update -qq && apt-get install -y build-essential libpq-dev nodejs
9
10 ENV CODE /code
11
12 USER user
13
14 WORKDIR $CODE
15
16 VOLUME $CODE
17
18 EXPOSE 3000
19 CMD ["rails","server","Puma","-b","0.0.0.0"]
20 |
```

Figura 5.13: Configuração usada para o *container* de *docker* com o servidor de *Ruby on Rails*

```
1 |version: '2'
2 |services:
3 |  db:
4 |    image: postgres
5 |    environment:
6 |      - POSTGRES_USER=userdb
7 |      - POSTGRES_PASSWORD=passworddb
8 |      - POSTGRES_DB=management_app_development
9 |      - DATABASE_HOST=db
10 |  web:
11 |    build: .
12 |    command: bundle exec rails server Puma -p 3000 -b '0.0.0.0'
13 |    volumes:
14 |      - ./code
15 |      - gems:/usr/local/bundle
16 |    ports:
17 |      - "3000:3000"
18 |    depends_on:
19 |      - db
20 |    environment:
21 |      - DATABASE_URL=postgres://userdb:passworddb@db/management_app_development
22 |  volumes:
23 |    gems:
24 |
```

Figura 5.14: Configuração usada para o sistema do *docker-compose*. É usado o *container* gerado com a configuração da figura 5.13

5.8 *Continuous Integration*

Como referido na secção 4.4.1, foi usada a funcionalidade de *Continuous Integration* do *Gitlab* para automatizar algumas tarefas com cada *commit* efetuado. Foram selecionadas bibliotecas de *Ruby on Rails* que permitem recolher o máximo de informação sobre a qualidade do código produzido e potenciais falhas de segurança. Antes disso, foi necessário configurar o projeto no *Gitlab* para usar a funcionalidade de CI.

5.8.1 Configuração do projeto no *Gitlab*

Para a funcionalidade de CI, o *Gitlab* permite a utilização de uma máquina pública ou de uma máquina privada. Usando uma máquina pública, qualquer projeto pode usar CI sem necessidade de configuração acessória. No entanto, a máquina é partilhada por vários projetos alojados na plataforma, e os resultados dos testes são visíveis por qualquer utilizador, independentemente de estar associado ao projeto ou não. Por isso, optou-se por não se usar uma máquina pública, usando-se antes uma máquina privada da HYP. Essa máquina teve que ser configurada, de forma a usar um ambiente de *docker* apropriado para correr a plataforma e os comandos necessários.

5.8.2 *Brakeman*

O *Brakeman* é uma biblioteca usada para detetar possíveis falhas de segurança numa aplicação em *Ruby on Rails*. Testa sistemas de autenticação de utilizadores, ataques *Cross-Site Scripting*, acesso a ficheiros, entre outros, num total de 63 verificações de segurança. Ao longo do desenvolvimento da plataforma, o número de falhas encontradas manteve-se a zero. No entanto, a biblioteca em si tem algumas limitações no *parsing* de ficheiros com caracteres não-ASCII, o que provocava a deteção de erros nesses casos.

5.8.3 *rubycritic*

rubycritic é uma ferramenta que utiliza várias outras ferramentas já existentes para análise estática de código em *Ruby on Rails* e agrupa os vários resultados num relatório interativo e intuitivo. É possível detetar vários *code smells* que podem dar origem a erros, como código duplicado ou métodos demasiado extensos. O *rubycritic* também analisa ficheiros gerados pelo próprio *Ruby on Rails*, cujos *code smells* não podem ser corrigidos de forma tão trivial. Na figura 5.15 pode-se ver um exemplo do relatório gerado pelo *rubycritic*.

5.8.4 *RuboCop*

Tal como o *rubycritic*, o *RuboCop* analisa o código produzido estaticamente. No entanto, em vez de detetar código duplicado, deteta quando alguma regra de boa prática de *Ruby on Rails* não é seguida como, por exemplo, linhas de código muito extensas ou indentação

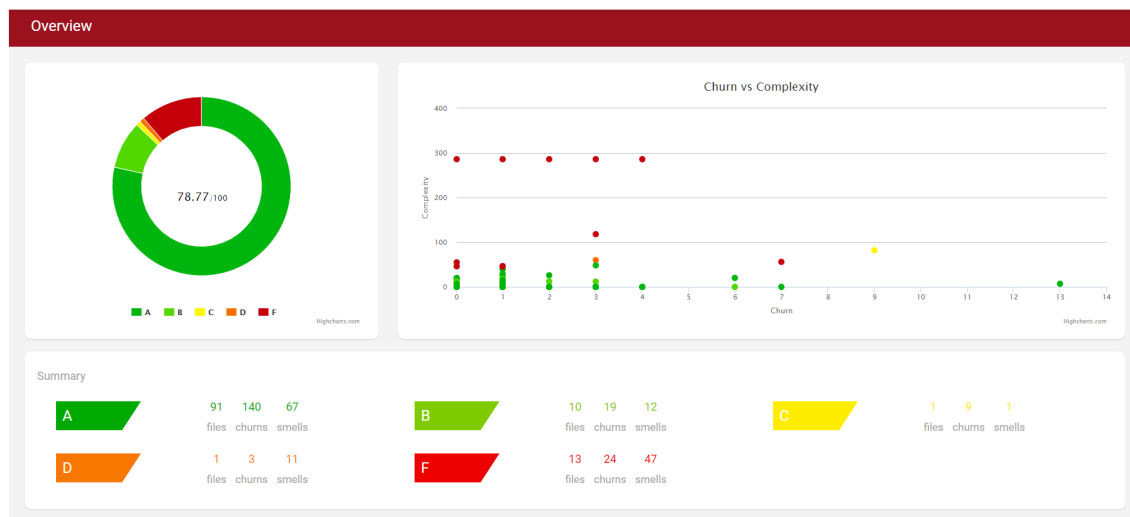


Figura 5.15: Relatório gerado pelo *rubycritic*



Figura 5.16: Relatório gerado pelo *RuboCop*

inconsistente. O *RuboCop* também pode gerar um relatório com os vários erros detetados, como se pode ver na figura 5.16.

5.8.5 rails_best_practices

rails_best_practices é mais uma ferramenta de análise estática de código de *Ruby on Rails*, que também tenta assegurar que se cumpram regras de boa prática da *framework*. No caso específico de *rails_best_practices*, é detetado código redundante ou inutilizado, como a presença de ficheiros em branco (maioritariamente *helpers* gerados automaticamente). Também sugere a movimentação de algum código, normalmente entre *controllers* e *helpers*. Na figura 5.17 é possível ver um exemplo do resultado da execução de *rails_best_practices*.


```
Source Code: |=====|
/code/app/views/projects/show.html.erb:34 - law of demeter
/code/app/views/tasks/index.html.erb:24 - law of demeter
/code/app/views/projects/index.html.erb:5 - move code into helper (array_count >= 3)
/code/app/views/styleguide/widgets/_dropdown.html.erb:3 - move code into helper (array_count >= 3)
/code/app/helpers/general_pages_helper.rb:1 - remove empty helpers
/code/app/helpers/projects_helper.rb:1 - remove empty helpers
/code/app/helpers/tasks_helper.rb:1 - remove empty helpers
/code/app/helpers/teams_helper.rb:1 - remove empty helpers
/code/app/helpers/time_entries_helper.rb:1 - remove empty helpers
/code/app/helpers/users_helper.rb:1 - remove empty helpers
/code/app/models/task.rb:5 - remove unused methods (Task#pending)
/code/app/views/styleguide/index.html.erb:22 - simplify render in views
/code/app/views/tasks/index.html.erb:21 - use query attribute (task.start_date?)
/code/app/views/tasks/index.html.erb:22 - use query attribute (task.end_date?)
/code/app/views/tasks/index.html.erb:23 - use query attribute (task.completed?)

Please go to http://rails-bestpractices.com to see more useful Rails Best Practices.
Found 15 warnings.
```

Figura 5.17: Resultado da execução de *rails_best_practices*

5.9 Conclusão

Neste capítulo, foram analisadas as várias questões analisadas durante o desenvolvimento da aplicação. Na tabela 5.2 é possível verificar o estado de cada componente da plataforma no momento do fim do estágio.

Componente	Estado
Utilizadores e equipas	Todas as <i>user stories</i> se encontram implementadas
Equipa	Todas as <i>user stories</i> se encontram implementadas
Projetos	Todas as <i>user stories</i> se encontram implementadas, exceto a <i>user story</i> 21
Tarefas e <i>Time entries</i>	Todas as <i>user stories</i> se encontram implementadas, exceto as <i>user stories</i> 31, 34 e 37
<i>Check-ins</i> e questionários	Encontram-se implementadas as <i>user stories</i> 39, 40 e 43
<i>Notificações</i>	Todas as <i>user stories</i> se encontram implementadas
Relatórios	Apenas é possível a criação de um tipo de relatório, não indo ao encontro do resultado esperado pela <i>user story</i> 60; estão implementadas as <i>user stories</i> 61 e 62

Tabela 5.2: Estado de cada componente da plataforma no fim do estágio

Capítulo 6

Testes

Neste capítulo são analisados os vários testes analisados e efetuados à plataforma. Como se trata de uma plataforma *web*, a maioria dos testes prende-se com a segurança da mesma, ao testar o acesso dos utilizadores a informação que não têm permissão para ver. São também feitos testes para garantir que as várias operações produzem as alterações desejadas na base de dados. Na seguinte lista são listados os vários testes, separados por componente do projeto.

6.1 Testes funcionais

6.1.1 Utilizadores e equipas

1. Um utilizador consegue registar-se com dados válidos
2. Um utilizador não consegue registar-se usando um e-mail ou nome já existente
3. Um utilizador consegue fazer *login* usando dados de uma conta existente
4. Um utilizador não consegue fazer *login* usando dados inválidos
5. Um utilizador consegue aceder ao seu perfil
6. Um utilizador consegue editar o seu perfil utilizando dados válidos
7. Um utilizador não consegue editar o seu perfil se usar dados inválidos
8. Um utilizador deve conseguir criar uma equipa usando um nome válido
9. Um utilizador não deve conseguir aceder a equipas a que não pertença
10. Um utilizador deve conseguir adicionar membros a uma equipa a que pertença
11. Um utilizador não deve conseguir adicionar membros a uma equipa a que não pertença

6.1.2 Projetos

1. Um utilizador deve conseguir criar um projeto
2. Um utilizador deve conseguir listar os seus projetos
3. Um utilizador deve conseguir listar projetos de uma equipa sua
4. Um utilizador não deve conseguir aceder a projetos a que não pertença, ou que não pertençam a uma equipa sua
5. Um utilizador que seja *master* de um projeto deve conseguir alterar a informação do mesmo, se a informação for válida
6. Um utilizador que seja *master* de um projeto não deve conseguir alterar a informação do mesmo, se a informação for inválida
7. Um utilizador que não seja *master* de um projeto não deve conseguir alterar a informação do mesmo
8. Um utilizador que seja *master* de um projeto deve conseguir alterar o tipo de participação de elementos do projeto
9. Um utilizador que não seja *master* de um projeto não deve conseguir alterar o tipo de participação de elementos do projeto

6.1.3 Tarefas e *time entries*

1. Um utilizador deve conseguir criar uma tarefa
2. Um utilizador deve conseguir listar as suas tarefas e as tarefas dos seus projetos
3. Um utilizador não deve conseguir aceder a tarefas de projetos em que não participe
4. Um utilizador que seja *master* ou *developer* de um projeto deve conseguir alterar a informação de uma tarefa desse projeto, se a informação for válida
5. Um utilizador que seja *master* ou *developer* de um projeto não deve conseguir alterar a informação de uma tarefa desse projeto, se a informação for inválida
6. Um utilizador que não seja *master* ou *developer* de um projeto não deve conseguir alterar a informação de tarefas desse projeto
7. Um utilizador deve conseguir iniciar uma *time entry*
8. Um utilizador deve conseguir parar uma *time entry* sua
9. Um utilizador deve conseguir listar as suas *time entries*
10. Um utilizador não deve conseguir aceder a *time entries* que não sejam suas

6.1.4 Notificações

1. Um utilizador deve conseguir listar as suas notificações
2. Um utilizador não deve conseguir aceder a notificações que não lhe sejam dirigidas
3. O número de notificações de um utilizador deve aumentar quando certos eventos acontecem na plataforma

6.1.5 *Check-ins* e questionários

1. Um utilizador deve conseguir fazer *check-in*, desde que não tenha nenhum *check-in* em aberto
2. Um utilizador não deve conseguir fazer *check-in* se tiver um outro *check-in* iniciado
3. Um utilizador deve conseguir fazer *check-out* se tiver um *check-in* iniciado
4. Um utilizador não deve conseguir fazer *check-out* se não tiver um *check-in* iniciado
5. Ao fazer *check-out* um utilizador deve ser redirecionado para um questionário, se tiver trabalhado em pelo menos um projeto com questionário associado
6. Um utilizador deve conseguir submeter a sua resposta a um questionário, se a informação for válida
7. Um utilizador não deve conseguir submeter a sua resposta a um questionário, se a informação for inválida
8. Um utilizador que seja *master* de um projeto deve conseguir alterar o questionário ativo desse projeto, se a informação for válida
9. Um utilizador que seja *master* de um projeto não deve conseguir alterar o questionário ativo desse projeto, se a informação for inválida
10. Um utilizador que não seja *master* de um projeto não deve conseguir aceder ao questionário desse projeto

6.1.6 Relatórios

1. Um utilizador deve conseguir gerar um relatório associado a um projeto, se for *reporter*, *developer* ou *master* desse projeto
2. Um utilizador não deve conseguir gerar um relatório associado a um projeto, se for *guest* desse projeto ou se não participar no mesmo
3. Ao usar filtros para a geração do relatório, a informação usada para a geração do relatório deverá conter apenas dados que respeitem os filtros usados

6.2 Conclusão

No contexto deste projeto de estágio, não foram idealizados testes não funcionais, como testes de usabilidade e de *performance*. Estes testes serão necessários para assegurar a melhor experiência possível por parte dos utilizadores da plataforma. Deverão ser planeados e efetuados testes de *performance*, de forma a garantir que o servidor onde a plataforma se encontra alojada tem as capacidades para responder aos pedidos de um número elevado de utilizadores sem perdas de *performance* que influenciem a navegação na aplicação. Testes de usabilidade permitem analisar a forma como os vários componentes da plataforma se encontram associados de forma intuitiva, e determinar se a navegação pela plataforma está bem estruturada.

Capítulo 7

Conclusão

Neste capítulo será feita, primeiramente, uma análise do trabalho a efetuar no futuro, e de seguida algumas considerações finais sobre o trabalho desenvolvido.

7.1 Trabalho futuro

Apesar do trabalho desenvolvido no contexto deste estágio, são várias as funcionalidades que podem ser implementadas na plataforma. Deve ser estudada a integração com plataformas externas, de forma a permitir uma integração com outros serviços e a migração entre os mesmos. Além disso, todo o código produzido entretanto deverá ser analisado, de forma a aferir a necessidade de *refactoring* para facilitar o crescimento da plataforma e a integração de novos módulos.

7.2 Testes não funcionais

Apesar de ser fácil verificar o cumprimento dos requisitos funcionais da plataforma, os requisitos não funcionais requerem métodos mais complexos de análise. Nesse sentido, será necessário realizar testes de usabilidade à plataforma, que assegurem que a navegação na mesma é fácil e que as várias tarefas podem ser realizadas sem problemas, e testes de *performance* que verifiquem não só se os pedidos feitos ao servidor recebem uma resposta num período de tempo aceitável, mas também que o servidor consegue suportar as necessidades de um número maior de clientes.

7.3 *Deployment*

Em paralelo com os testes de *performance* referidos anteriormente, deve considerar-se a possibilidade de migrar a plataforma para um servidor dedicado, se o cliente optar pela

distribuição do produto. Além disso, deverão ser introduzidas ferramentas que facilitem a análise do comportamento do servidor e dos utilizadores da plataforma.

As imagens de perfil dos utilizadores são armazenadas no próprio servidor de desenvolvimento. Com o crescimento da plataforma e do número de utilizadores da mesma, pode ser útil considerar a migração das mesmas para um serviço externo especializado no armazenamento de ficheiros.

7.4 Manutenção

A plataforma deverá ser mantida pela equipa de desenvolvimento durante as primeiras fases de implementação, de forma a garantir uma rápida resposta em caso de falha.

7.5 Considerações finais

Ao longo do desenvolvimento deste projeto, foram percorridas as várias etapas de um processo de desenvolvimento de *software*.

A análise de plataformas concorrentes permitiu estudar diferentes funcionalidades e diferentes abordagens à problemática da Gestão de Projetos, permitindo definir melhor o rumo a seguir no desenvolvimento da aplicação. O estudo de modelos de desenvolvimento e tipos de testes também foi uma ajuda importante mais tarde para o planeamento do desenvolvimento.

Para a descrição da plataforma a desenvolver, foram tidas em conta as conclusões retiradas na etapa anterior, criando-se uma lista das várias funcionalidades a implementar e das várias ações que o utilizador poderia levar a cabo na aplicação. O levantamento dos requisitos não funcionais foi levado a cabo considerando o comportamento esperado da plataforma para uma melhor experiência por parte dos utilizadores. Para a análise dos vários requisitos foi também tida em conta a ideia original do cliente e as funcionalidades que este desejava para a plataforma.

O planeamento foi certamente uma componente complexa, maioritariamente no primeiro semestre. A calendarização nem sempre foi respeitada, levando a atrasos no planeamento e desenvolvimento da aplicação. Para o futuro desenvolvimento desta plataforma, deverá estudar-se o modelo de desenvolvimento que se deve adotar, verificando se o atual modelo em espiral continua a ser o indicado ou se se deve adaptar, dependendo da equipa de desenvolvimento.

Durante o desenvolvimento da plataforma, a utilização de uma plataforma de desenvolvimento como o *docker* foi importante, de forma a garantir o isolamento dos vários componentes e a facilitar a integração com serviços como a *Continuous Integration* do *Gitlab*. Além disso, com a entrada de novos elementos na equipa de desenvolvimento, a sua integração será mais fácil e serão recriadas ao pormenor as condições de desenvolvimento iniciais.

Bibliografia

- [1] Project Management Institute Staff. *A Guide to the Project Management Body of Knowledge (PMBOK® Guide)*. Project Management Institute, 5th edition, 2013.
- [2] Harold R. Kerzner. *Project Management : A Systems Approach to Planning, Scheduling, and Controlling*. John Wiley & Sons, 11th edition, 2013.
- [3] Jean-Philippe Lang. Overview - redmine, December 2016.
- [4] Project Management Zone. Popularity ranking of project management systems, December 2016.
- [5] Microsoft. Project management software | microsoft project, December 2016.
- [6] Trello Inc. Trello, December 2016.
- [7] Atlassian. Jira software - issue & project tracking for software teams | atlassian, December 2016.
- [8] Slack. Slack, December 2016.
- [9] Business Insider. Slack, the red hot \$3.8 billion startup, has a hidden meaning behind its name, December 2016.
- [10] Microsoft. Yammer: Work smarter, work together, December 2016.
- [11] GetApp. Toggl pricing, features, reviews & comparison of alternatives | getapp, December 2016.
- [12] Toggl OÜ. Toggl api documentation, December 2016.
- [13] Tracking Time LLC. Trackingtime - real time, collaboration and organization, December 2016.
- [14] GetApp. Trackingtime pricing, features, reviews & comparison of alternatives | getapp, December 2016.
- [15] Lattice. Lattice check-ins | become a better manager, December 2016.
- [16] Jack Altman. Lattice check-ins: A better way to stay aligned, December 2016.
- [17] Citrix. Software de gerenciamento de projetos online | podio, December 2016.
- [18] Citrix. Software de gerenciamento de projetos online | podio, December 2016.

- [19] Winston W Royce. Managing the development of large software systems. In *proceedings of IEEE WESCON*, volume 26, pages 328–338. Los Angeles, 1970.
- [20] Steve McConnell. *Rapid development: taming wild software schedules*. Pearson Education, 1996.
- [21] Chandra Vennapoosa. Staged delivery model, 2013.
- [22] Agile Alliance. Manifesto for agile software development, 2001, 2001.
- [23] Barry Boehm and Wilfred J Hansen. Spiral development: Experience, principles, and refinements. Technical report, DTIC Document, 2000.
- [24] Glenford J Myers, Corey Sandler, and Tom Badgett. *The art of software testing*. John Wiley & Sons, 2011.
- [25] S Maeda, D Thomas, and A Hunt. The ruby language faq, 2002.
- [26] Sophie Debenedetto. Real-time rails: Implementing websockets in rails 5 with action cable, 2016.

Apêndice A

Wireframes

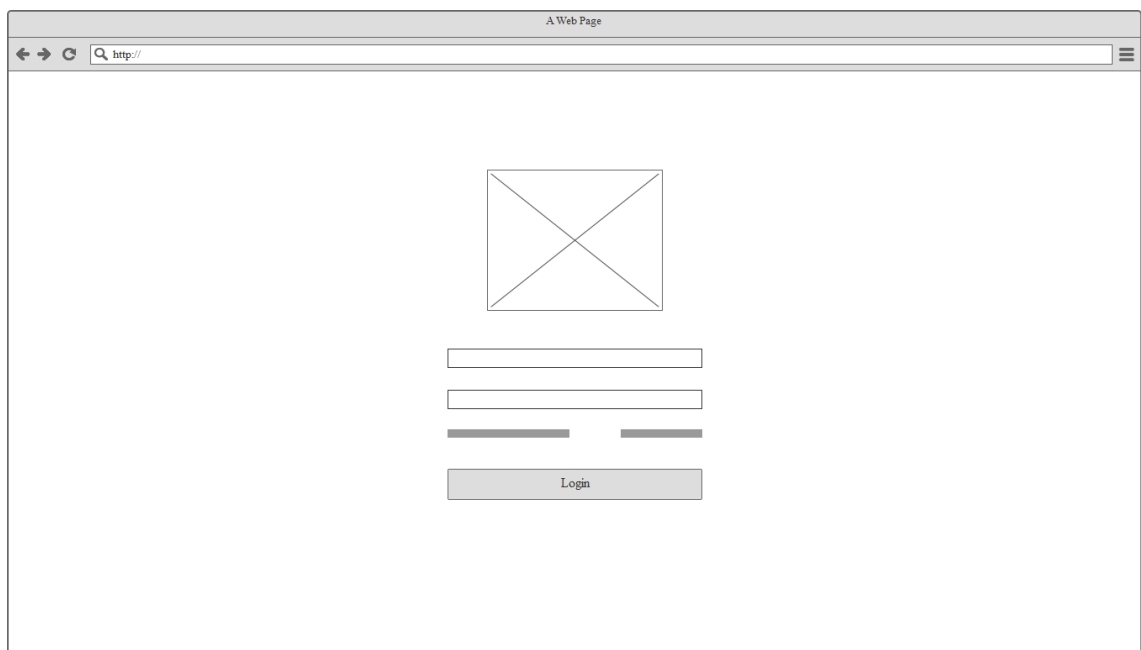


Figura A.1: *Wireframe - Login*

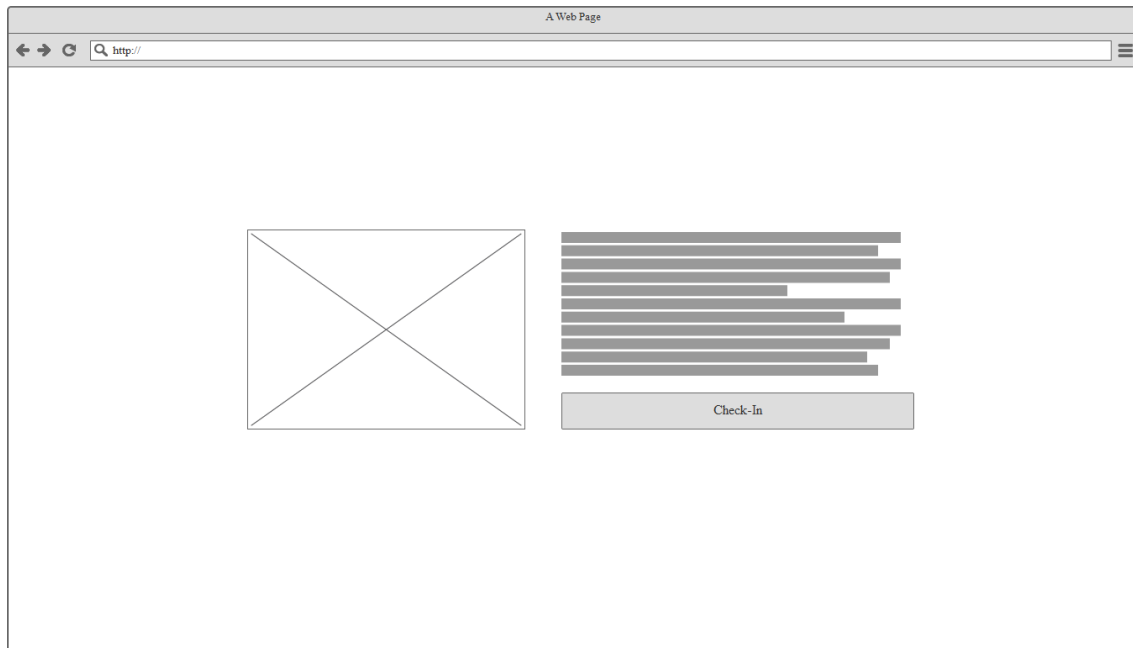


Figura A.2: *Wireframe - Check-in*

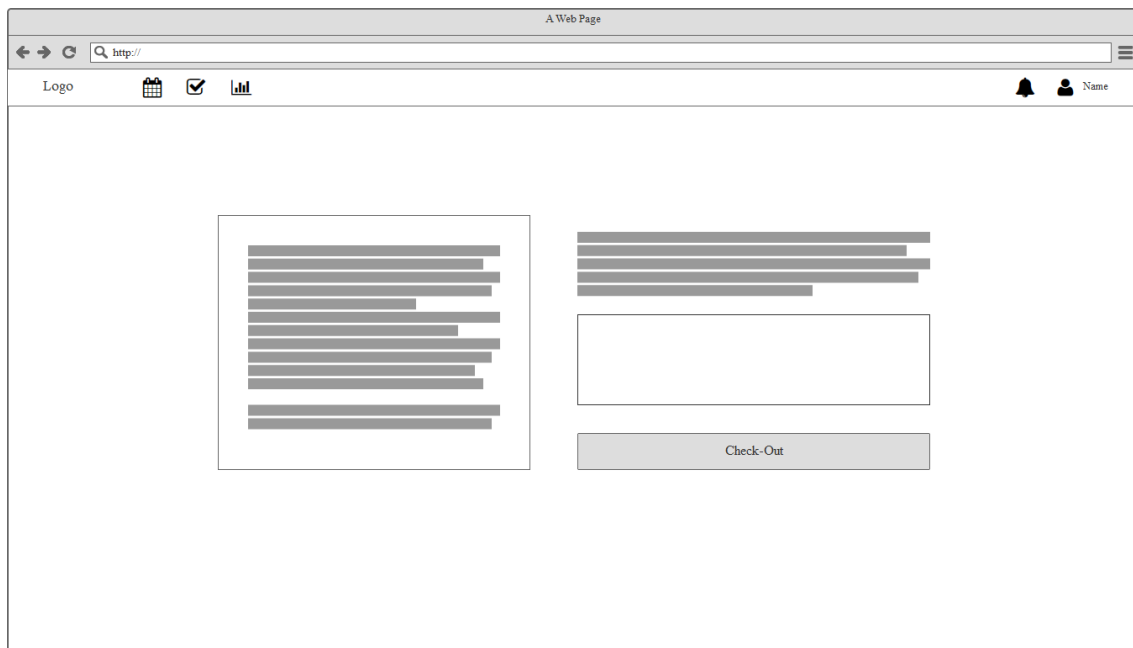


Figura A.3: *Wireframe - Check-out*

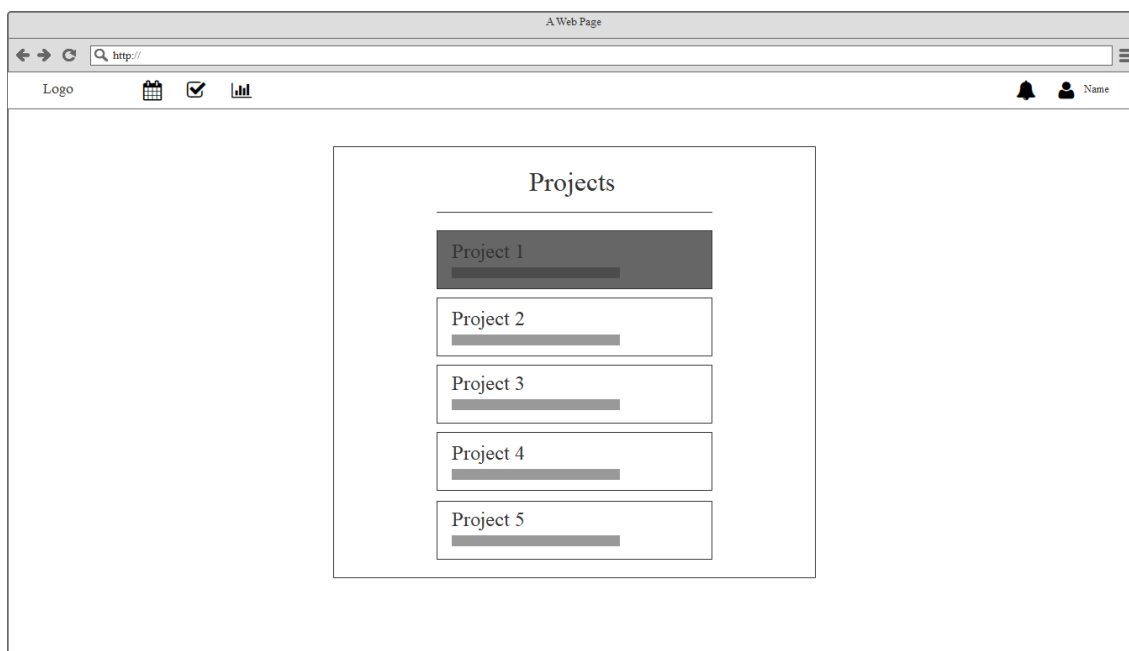


Figura A.4: *Wireframe* - Lista de projetos

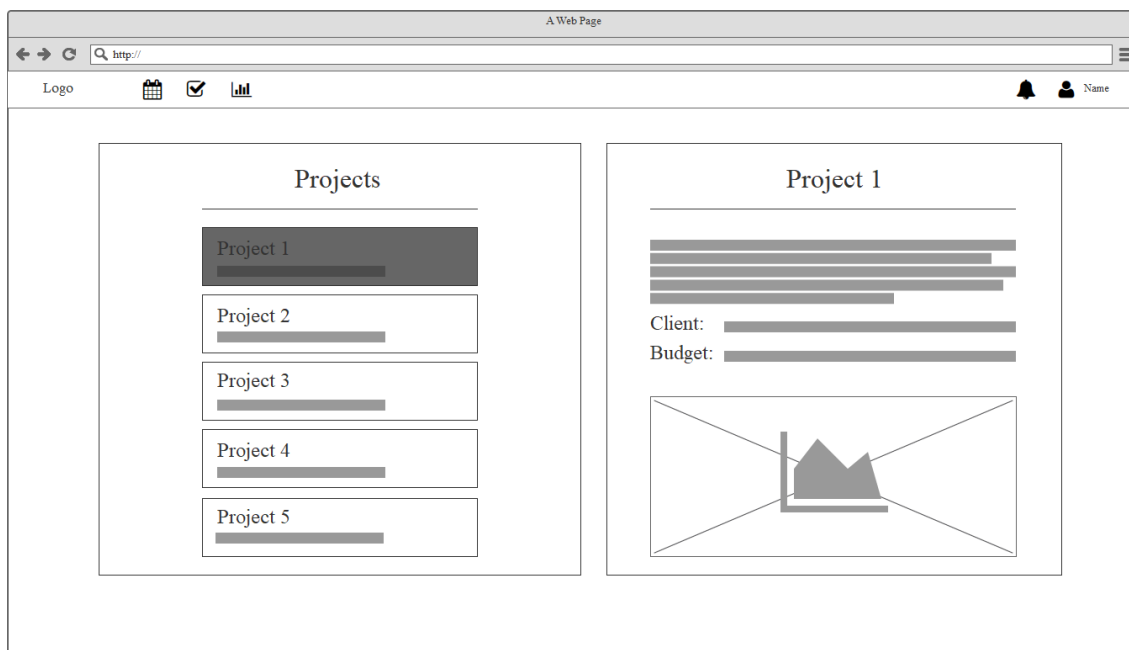


Figura A.5: *Wireframe* - Informação de um projeto

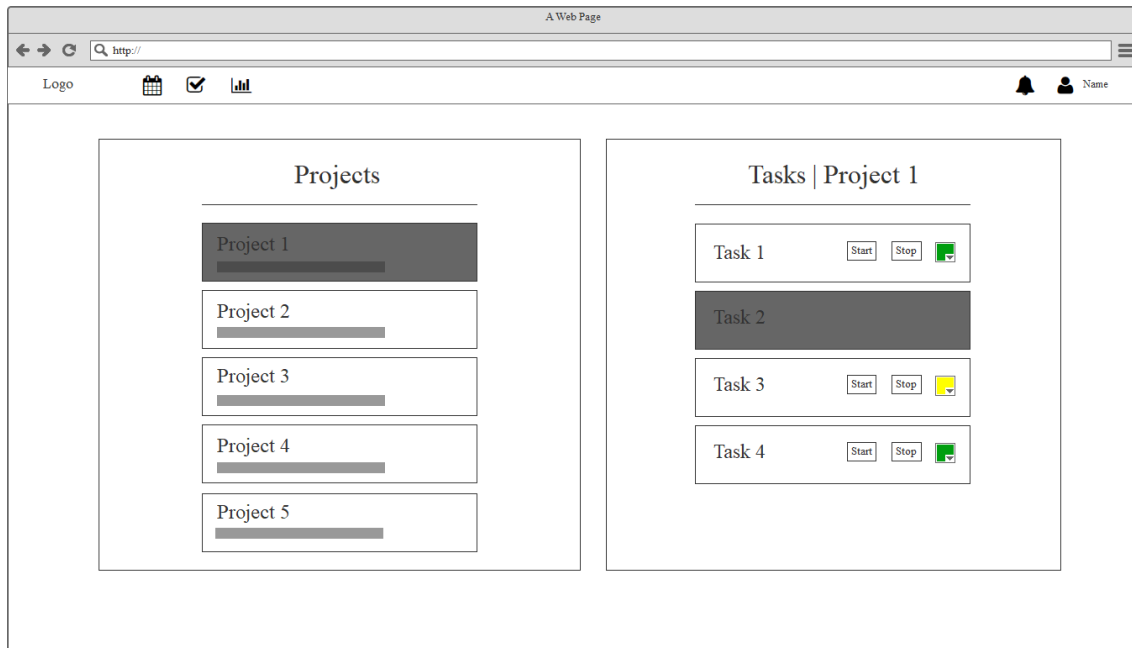


Figura A.6: Wireframe - Lista de tarefas

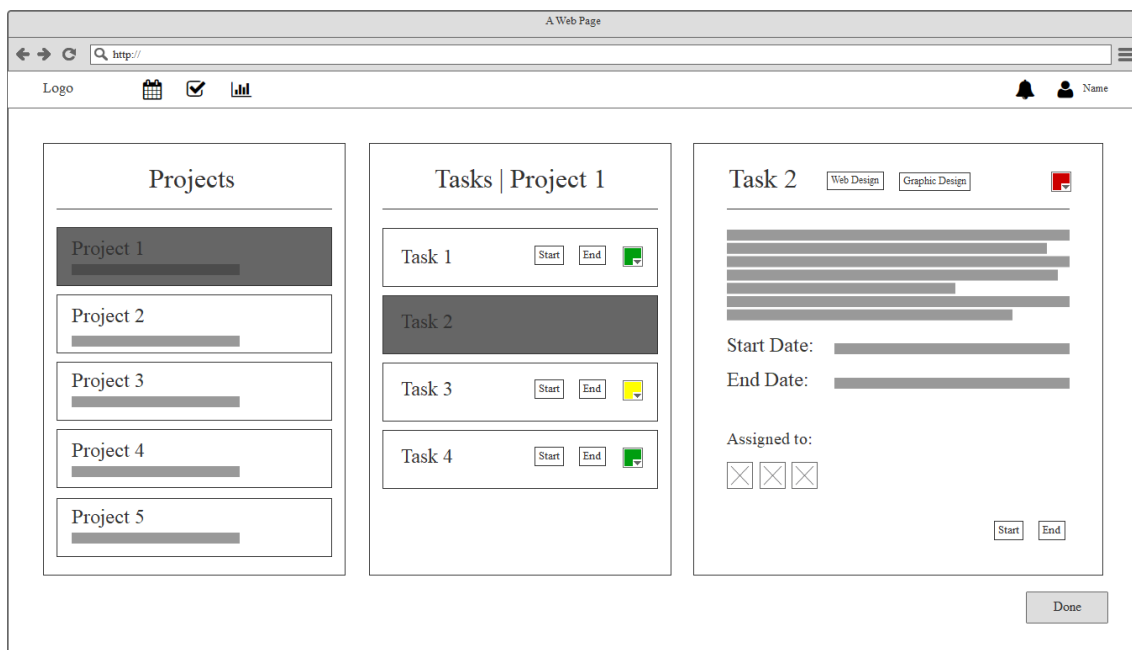


Figura A.7: Wireframe - Tarefa



Figura A.8: *Wireframe* - Lista de *time entries*

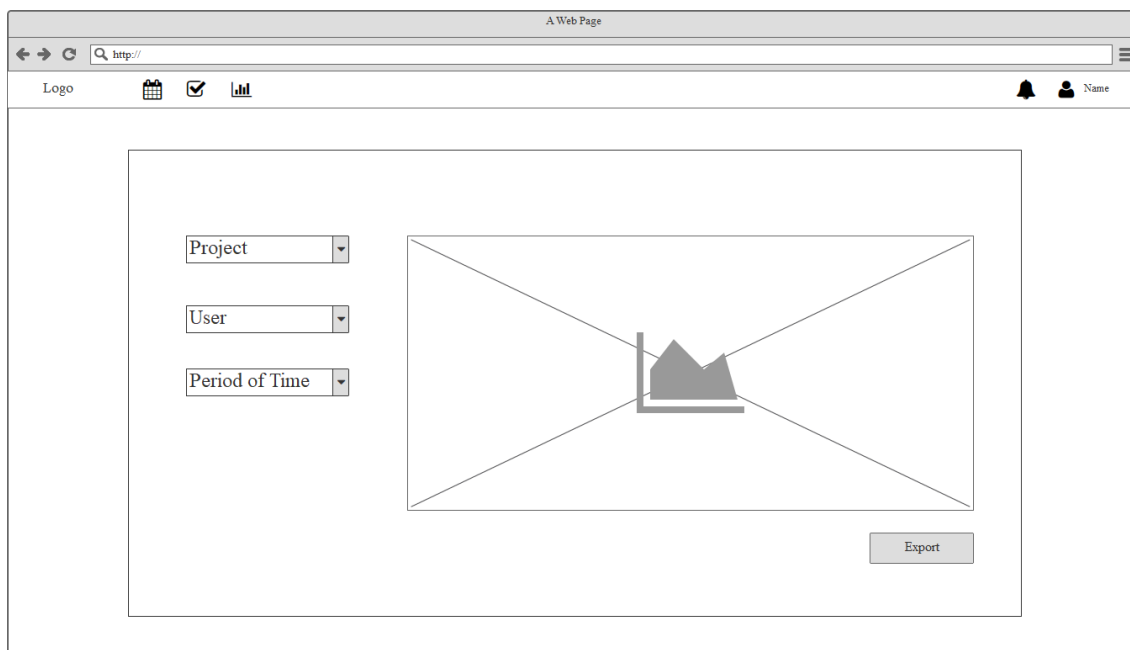


Figura A.9: *Wireframe* - Geração de relatórios

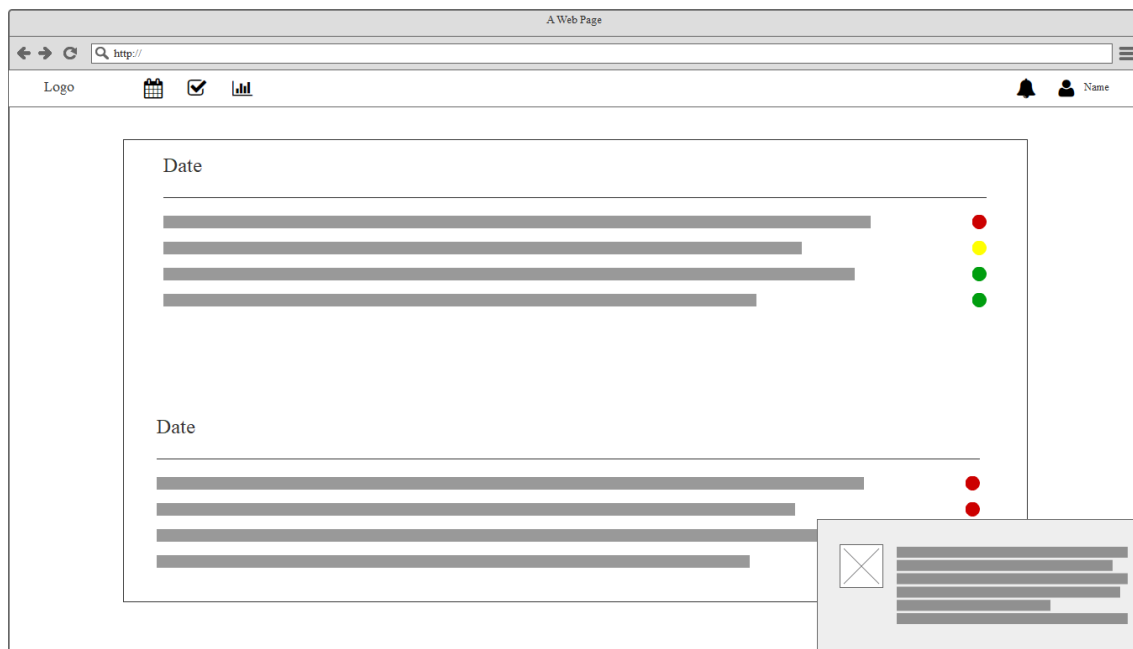


Figura A.10: *Wireframe* - Notificação