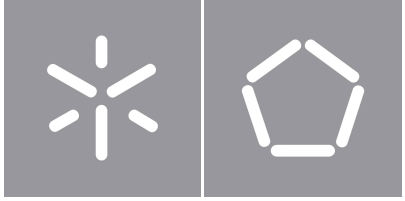
Universidade do Minho
Escola de Engenharia

João Pedro Coimbra Martins de Sá

Integração de uma aplicação de reporting para testes de software no Confluence Cloud

outubro 2023



Universidade do Minho
Escola de Engenharia

João Pedro Coimbra Martins de Sá

Integração de uma aplicação de reporting para testes de software no Confluence Cloud

Dissertação de Mestrado
Mestrado em Engenharia Informática

Trabalho efetuado sob a orientação de
João Miguel Lobo Fernandes

outubro 2023

Direitos de Autor e Condições de Utilização do Trabalho por Terceiros

Este é um trabalho académico que pode ser utilizado por terceiros desde que respeitadas as regras e boas práticas internacionalmente aceites, no que concerne aos direitos de autor e direitos conexos.

Assim, o presente trabalho pode ser utilizado nos termos previstos na licença abaixo indicada.

Caso o utilizador necessite de permissão para poder fazer um uso do trabalho em condições não previstas no licenciamento indicado, deverá contactar o autor, através do RepositóriUM da Universidade do Minho.

Licença concedida aos utilizadores deste trabalho:

CC BY

https://creativecommons.org/licenses/by/4.0/

Agradecimentos

Em primeiro lugar, gostaria de agradecer à empresa Sngular PT pela oportunidade dada. Em especial, ao meu supervisor Rui Rodrigues por me acolher de braços abertos e pela sua disponibilidade e confiança demonstrada desde o primeiro momento pois, sem ele, todo este trabalho não seria possível.

Expresso a minha gratidão ao Professor João Miguel Lobo Fernandes, pela sua orientação na elaboração desta tese de mestrado e por todo o apoio e esclarecimentos fornecidos.

Quero deixar uma menção especial à minha família e amigos que acompanharam de perto todo o meu percurso durante os últimos 5 anos, encorajando-me e apoiando-me a cada passo dado.

Declaração de Integridade

Declaro ter atuado com integridade na elaboração do presente trabalho académico e confirmo que não recorri à prática de plágio nem a qualquer forma de utilização indevida ou falsificação de informações ou resultados em nenhuma das etapas conducente à sua elaboração.

Mais declaro que conheço e que respeitei o Código de Conduta Ética da Universidade do Minho.

Universidade do Minho, Braga, outubro 2023

João Pedro Coimbra Martins de Sá

Resumo

A crescente evolução tecnológica verificada nas últimas décadas e a necessidade de criação e implementação de novas soluções de *software*, levam a que o papel dos testes de *software* assuma, cada vez mais, uma maior relevância no respetivo ambiente de desenvolvimento. Boas ferramentas de gestão e monitorização de testes, assim como, de geração de relatórios de estados e resultados associados aos mesmos testes, melhoram o processo de desenvolvimento tornando-o melhor e mais completo a todos os níveis.

O principal objetivo da presente dissertação de mestrado é criar uma nova aplicação de geração de relatórios em páginas *Confluence*, utilizando como fonte de dados, os testes gerados a partir do *Xray*. O *Xray* é uma aplicação de gestão e monitorização de testes de *software* em instâncias *JIRA*. O desenvolvimento da aplicação implica o estudo das diferentes estratégias de desenvolvimento de *software* disponíveis no ecossistema *Atlassian*.

A aplicação *Xray* foi analisada em detalhe, com especial foco nos relatórios de testes de *software* disponibilizados, o que permitiu a formulação de uma proposta de solução para o problema em questão. De seguida, foram analisadas diferentes *frameworks* de desenvolvimento de *software* para produtos *Atlassian*, culminando na escolha da ferramenta *Forge* face a todas as vantagens e desvantagens a ela associadas.

De igual modo, é apresentado todo o processo de desenvolvimento da aplicação, assim como as estratégias adotadas para a correta implementação de cada um dos relatórios propostos.

Palavras-chave Confluence, Desenvolvimento de Software, Documentação, Forge, Macro, Relatório, Testes de Software, Xray

Abstract

The growing technological evolution of recent decades and the need to create and implement new software solutions mean that the role of software testing is becoming increasingly important in the development environment, in the development environment. Good tools for managing and monitoring tests, as well as for generating status reports and results associated with the same improve the development process, make this process better and more complete at all levels.

The main objective of this master's thesis is to create a new application for generating reports on *Confluence* pages, using the tests generated from *Xray* as a data source. *Xray* is an application for managing and monitoring software tests on *JIRA* instances. The development of the application involves studying the different software development strategies available in the *Atlassian* ecosystem.

The *Xray* application was analyzed in detail, with a special focus on the reports provided, which allowed the formulation of a proposed solution to the problem in question. Different software development frameworks for *Atlassian* products were then analyzed, culminating in the choice of the Forge tool in view of all the advantages and disadvantages associated with it.

The entire application development process is presented, as well as the strategies adopted for the correct implementation of each of the proposed reports.

Keywords Confluence, Documentation, Forge, Macro, Report, Software Development, Software Testing, Xray

Conteúdo

Lista de Figuras

Lista de Tabelas

Acrónimos

API Application Programming Interface.

BDUF Big Design Up Front.

CA Critério de Aceitação.

CD Continuous Delivery.

CI Continuous Integration.

CVDS Ciclo de Vida do Desenvolvimento de Software.

CVTS Ciclo de Vida do Teste de Software.

ES Engenharia de Software.

FaaS Function as a Service.

JSON JavaScript Object Notation.

OTAN Organização do Tratado do Atlântico Norte.

REST Representational State Transfer.

SaaS Software as a Service.

SUT System Under Test.

TI Tecnologias de Informação.

V&V Verificação e Validação de Software.

Parte I

Capítulo 1

Introdução

Neste capítulo, faz-se uma introdução ao tema principal desta dissertação de mestrado, apresenta-se o enquadramento e a motivação para a sua realização e indicam-se os objetivos alcançados. É também apresentada uma síntese da estrutura do documento.

1.1 Enquadramento e Motivação

A presente dissertação de mestrado insere-se no ciclo de estudos do curso de Mestrado em Engenharia Informática da Universidade do Minho e o tema foi proposto pela *Sngular PT*. A *Sngular PT* é uma empresa de consultoria, presente no setor da tecnologia, especializada no desenvolvimento de soluções de *software* de valor e qualidade que são utilizadas por inúmeras empresas em todo o mundo.

O *Xray*, um dos principais produtos do qual a *Sngular PT* é responsável pelo desenvolvimento de vários componentes, é uma aplicação incorporada no *JIRA* que é utilizada para gestão, execução e monitorização de testes de *software*, criação de relatórios de estados e que integra várias ferramentas de automação de testes de *software*.

Por sua vez, o *JIRA* é uma ferramentas desenvolvida pela *Atlassian* para gestão de projetos e respetivas tarefas de modo a agilizar todo o processo de conceção de um produto. Para além do *JIRA*, a *Atlassian* disponibiliza outras ferramentas, como o Confluence, para gestão e partilha de conteúdo, ou o *Bitbucket* para gestão de código e controlo de versões.

Face ao grande crescimento observado no mercado de testes de *software* e à procura de novas e melhores soluções de *reporting*, a *Sngular PR* pretende desenvolver uma aplicação a integrar em instâncias *Confluence*. Esta aplicação deve disponibilizar, aos utilizadores da aplicação *Xray*, a integração de alguns dos seus relatórios em páginas do *Confluence Cloud*, algo que, apesar da grande variedade de funcionalidades disponibilizadas e das inúmeras integrações com outras ferramentas, neste momento não é suportado pelo *Xray*.

1.2 Objetivos

Como enunciado no subcapítulo anterior e visando que o respetivo objetivo seja alcançado com sucesso, o projeto desenvolvido na presente dissertação de mestrado divide-se em duas partes.

Na primeira parte, de cariz teórico e investigativo, são definidos os seguintes objetivos:

O1: Estudo dos testes de *software* e o seu enquadramento no processo de desenvolvimento de *software*;

O2: Análise da aplicação *Xray* e das suas principais funcionalidades, nomeadamente a geração de relatórios de estados dos testes de *software*.

Para a segunda fase, de cariz prático, os objetivos são:

O3: Estudo das *frameworks* de desenvolvimento disponíveis para a criação de aplicações no ambiente *Atlassian*, culminando com a escolha da *framework* que melhor satisfaz as necessidades da aplicação;

O4: Levantamento e análise dos requisitos da aplicação seguido da conceção e implementação da respetiva aplicação.

Por fim, é realizada a comparação da aplicação e respetivos relatórios desenvolvidos com os originais, assim como é feita a retrospetiva e análise crítica a todo o trabalho realizado ao longo da concretização do projeto.

1.3 Estrutura do Documento

O presente documento encontra-se organizado da seguinte forma:

- **Capítulo 1 - Introdução**: É realizada a introdução ao problema a abordar durante a realização da presente dissertação, assim como, os objetivos esperados;
- **Capítulo 2 - Estado da Arte**: Contém o levantamento do estado de arte associado aos testes de software e a documentação produzida durante a realização dos mesmo, bem como, o estudo do ecossistema *Atlassian* e da aplicação *Xray*;
- **Capítulo 3 - O Problema e os seus Desafios**: Contém a descrição detalhada de todo o problema e a idealização da proposta de solução para a sua resolução;

- **Capítulo 4 - Organização e Estrutura do Projeto**: É realizada a apresentação da equipa, assim como, as ferramentas utilizadas para a gestão e controlo do projeto;

- **Capítulo 5 - Requisitos**: É descrito todo o processo de levantamento e análise dos requisitos do problema e enunciados os diferentes cenários de utilização da aplicação;

- **Capítulo 6 - Desenvolvimento da Aplicação**: São enunciados e descritos todos os detalhes associados ao processo de desenvolvimento, assim como, as decisões tomadas pela equipa de desenvolvimento de forma a garantir a implementação dos requisitos da aplicação;

- **Capítulo 7 - Conclusões e Trabalho Futuro** É realizada a retrospetiva de todo o trabalho realizado e os propósitos a abordar num trabalho futuro.

Capítulo 2

Estado da arte

Neste capítulo é abordado o tema dos testes de *software*, explorando-se qual o seu enquadramento e relevância no processo de desenvolvimento de *software*, assim como, qual o ciclo de vida geral de um teste, quais as diferentes variantes que o mesmo pode assumir e qual a importância da escrita de documentação no processo de testes. São também analisadas a aplicação *Xray* e a empresa *Atlassian* com a finalidade de compreender a forma como operam, as funcionalidades que disponibilizam, os objetivos para as quais são utilizadas e as vantagens que oferecem em relação aos seus concorrentes diretos no mercado.

2.1 Testes de Software

Nesta secção são abordados os testes de *software*, o seu enquadramento no desenvolvimento de *software*, os diferentes tipos e qual o ciclo de vida de um teste, terminando com a análise da documentação produzida aquando e após a realização dos testes de *software*.

2.1.1 Enquadramento no Desenvolvimento de Software

> *"All software you write will be tested—if not by you and your team, then by the eventual users—so you might as well plan on testing it thoroughly."*
>
> Andy Hunt[1]

No ano de 2018 celebrou-se o 50º aniversário da **Engenharia de Software** (**ES**) após a realização da primeira conferência exclusivamente dedicada ao tema, organizada pela **OTAN** em Garmisch, na Alemanha, impulsionando um movimento de estudo, pesquisa e desenvolvimento na área fazendo com que a **ES** seja aquilo que conhecemos hoje [Erdogmus et al., 2018]. Segundo Sommerville [2016] a **ES** tornou-se indispensável para o bom funcionamento do governo e da sociedade, assim como, de instituições

[1] Hunt, A., Thomas, D.: The Pragmatic programmer: from journeyman to master. Addison-Wesley, Boston etc. (2000)

e negócios, quer a nível nacional quer a nível internacional, sendo a área responsável por todos os aspetos relacionados com a criação e desenvolvimento de *software*.

Ao longo do curto período de existência da **ES**, foram propostas e adotadas diferentes abordagens para encarar o processo de desenvolvimento de *software*, também conhecidas como metodologias, esquematizando e organizando as diferentes etapas do **Ciclo de Vida do Desenvolvimento de Software** (**CVDS**). Deste ciclo resulta uma estratégia bem definida e formalizada, em que cada etapa possui um objetivo claro e preciso daquilo que é pretendido na mesma, que pode ser adotada pela indústria para a criação de um produto [Scacchi, 2002]. As metodologias de desenvolvimento de *software* podem ser dividas em dois grupos principais: as metodologias tradicionais ou clássicas e as metodologias ágeis. Segundo Mccauley [2001] não existe nenhuma metodologia que seja universalmente considerada a melhor para qualquer tipo de projeto tornando-se necessário a investigação e escolha daquela que melhor satisfaz as características do mesmo.

As metodologias tradicionais visam uma abordagem preditiva ao problema, baseando-se numa sequência de etapas que deverá ser rigorosamente respeitada e empregue para que todos os requisitos do projeto sejam satisfeitos. Estas metodologias tendem a privilegiar uma abordagem **Big Design Up Front** (**BDUF**), ou seja, como fase inicial de qualquer projeto é feito o levantamento e definição de requisitos, podendo apenas ser seguida pela fase da implementação da respetiva solução quando toda a documentação associada à primeira fase estiver concluída. Assim sendo, estas metodologias são conhecidas por serem extremamente burocráticas e massacrantes, podendo levar à perda de tempo em todo o processo de desenvolvimento de *software* [Islam e Ferworn, 2020]. A propagação da frustração entre os programadores na indústria das **Tecnologias de Informação** (**TI**) ao adotar metodologias clássicas levou a que, nos últimos anos, a popularidade das metodologias ágeis tenha vindo cada vez mais a aumentar. As metodologias ágeis, como o próprio nome sugere, tornam todo o processo de desenvolvimento mais eficiente, colocando todas as burocracias do processo em segundo plano e visando principalmente o fator humano, tanto dos programadores como dos clientes. A promoção da comunicação entre os diferentes elementos da equipa, clientes e stakeholders e o desenvolvimento iterativo e incremental do produto, com o objetivo de minimizar riscos e solucionar possíveis problemas mais rapidamente, são apenas algumas das características citadas por Islam e Ferworn [2020] que reduzem significativamente o tempo gasto na criação de um produto quando adotada uma metodologia ágil.

Independentemente da metodologia seguida, o desenvolvimento de *software* pode ser dividido em quatro fases, sendo que a sequência e a forma como se encontram organizadas ao longo do **CVDS** já é influído pela metodologia em questão [Sommerville, 2016]:

- **Especificação:** Fase de compreensão, definição e especificação dos requisitos do sistema, assim como, das restrições associadas à implementação. A especificação de *software* é considerada como uma das etapas mais críticas no processo de desenvolvimento de *software* uma vez que qualquer erro cometido nesta fase originará consequentemente erros em etapas futuras;

- **Conceção e Implementação:** Nesta fase são tomadas todas as decisões relativas à conceção de *software* (arquitetura do sistema, esquema de base de dados, *design* da interface, entre outros) e implementados todos os requisitos levantados na fase anterior;

- **Teste de Software:** Etapa também conhecida como **Verificação e Validação de Software** (**V&V**) na qual se utilizam os testes de *software* como principal ferramenta para verificar e validar que o produto desenvolvido está em conformidade com as expectativas do cliente e utilizador do sistema. Existem várias técnicas abordadas na validação de um produto de *software*, porém é nos testes e no processo de *debugging* onde é gasto a maior percentagem do tempo nesta fase;

- **Evolução:** Dificilmente o lançamento no mercado de um produto de *software* significa o fim do seu ciclo de vida. Um produto de *software* requer monitorização e manutenção para garantir a sua disponibilidade e necessita de constantes atualizações para que acompanhe os novos requisitos dos seus clientes ou utilizadores e as tendências do mercado. A partir do momento que é tomada a decisão de atualizar o produto, é iniciado um novo ciclo abordando-se novamente todas as etapas até aqui enunciadas.

Segundo Broy [2018], observando a rápida evolução que se verificou na **ES** e no processo de desenvolvimento de *software* nos últimos 50 anos, é impossível prever com algum nível de detalhe aquilo que é expectável para o seu futuro. No entanto, e tendo em conta algumas estatísticas, torna-se difícil considerar que os testes de *software* serão um fator a abandonar no processo de desenvolvimento num futuro próximo. Conforme Mens e Demeyer [2008], 50% do tempo gasto no desenvolvimento de um novo produto é alocado para a realização de testes, sendo 40 a 80% do orçamento gasto para o mesmo fim. Por outro lado, Sommerville [2016] têm uma visão mais conservadora quanto aos gastos, considerando que estes assumem valores a rondar os 40% do orçamento disponível mas reconhecendo que, em determinados casos, o valor disponibilizado poderá ser maior. Analisando estes números, verifica-se o peso e a importância daquilo que o processo de testes representa no desenvolvimento de *software*.

2.1.2 Definição, Tipos e Estratégias de Testes de Software

> *"Program testing can be a very effective way to show the presence of bugs, but is hopelessly inadequate for showing their absence."*
>
> Edsger Dijkstra[2]

Um teste de *software* representa um processo, ou um conjunto de processos, de forma a garantir que o código produzido realiza todas as funcionalidades pretendidas e, ao mesmo tempo, não possui nenhum comportamento inesperado indo ao encontro daquilo que são os requisitos e expectativas do cliente de forma assegurando a qualidade do produto [Myers et al., 2011].

Os testes de *software* são principalmente utilizados na fase de **Verificação e Validação de Software (V&V)**, ou seja, para confirmar que se está a construir o produto correto para o problema em questão (Validação) e que se está a construir o produto da forma correta (Verificação) [Sommerville, 2016]. De tal modo, é necessário a existência de diferentes tipos de testes de *software* de forma a abranger as diferentes funcionalidades do produto, destacando-se os seguintes na Tabela 1 [Institute, 2014].

Para a realização de testes, existem duas técnicas que podem ser aplicadas: realização de testes manuais, que corresponde à forma mais antiga e morosa de realizar testes de *software*, ou a realização de testes automáticos. Na realização de testes manuais, todo o processo está fortemente dependente do *tester* e, para tal, é fulcral que este seja capaz de compreender quais os requisitos do produto e quais os resultados e comportamentos esperados, de modo a que consiga criar adequadamente os planos de teste, verificar e analisar os resultados obtidos e criar os respetivos relatórios [Dustin, 2002]. Apesar de em alguns casos ser considerado o método mais confiável, a realização de testes manuais é um processo bastante demorado, uma vez que todos os processos que compõem um teste de *software* são realizados única e especificamente pelo *tester*. Por outro lado e tirando proveito daquilo que são as propriedades de um teste automático, a realização de testes de *software* automáticos permite ao *tester* utilizar inúmeras ferramentas auxiliares de teste e economizar bastante tempo na realização do processo, essencialmente quando existem certos componentes que necessitam de ser testados recorrentemente ao longo de todo o desenvolvimento do produto de *software*. Outra das principais vantagens da automação de testes de *software* é a mitigação de erros humanos associados à execução manual de um teste [Dustin et al., 2009].

[2] Dijkstra, E. W. 1972. "The Humble Programmer." Comm. ACM 15 (10): 859–66. doi:10.1145/ 355604.361591

Tipo	Descrição
Funcional	Verificam-se que as funcionalidades e comportamento do sistema estão de acordo com os requisitos do cliente.
Usabilidade	É verificada a facilidade de uso do produto a partir de três fatores: O quão conveniente é o *software* para o utilizador? o utilizador será capaz de utilizar o produto? O utilizador será capaz de aprender a utilizar o produto?
Interface	É testada a interação do utilizador com a interface da aplicação assim como é avaliada o seu nível de atração e que todos os resultados são os esperados.
Configuração e Portabilidade	Neste tipo de testes é analisado o comportamento do produto em diferentes plataformas e com diferentes especificações e configurações de forma a garantir a correta execução de todas as funcionalidades em várias circunstâncias possíveis.
Segurança, Autenticação e Autorização	São procuradas falhas que possam levar ao comprometimento de informações cruciais, quer da aplicação quer dos seus utilizadores.
Integridade de Dados	Nos testes de integridade é verificado que todos os dados armazenados durante a utilização do produto são precisos, corretos e seguros, assim como, os acessos aos mesmos.
Tolerância a Falhas	É analisado o comportamento da aplicação aquando a ocorrência de alguma falha e como o *software* é capaz de recuperar ao seu estado original após estes eventos.
Carga e Desempenho	Nos testes de carga o sistema é submetido a níveis de carga bastantes elevados é analisado o comportamento do mesmo sob stress.
Regressão	Os testes de regressão são realizados após a correção de uma falha de forma a garantir que as novas alterações ao código não geraram novas falhas noutra parte do sistema.

Tabela 1: Tipos de testes de *software*

No entanto e apesar de na generalidade os testes automáticos serem a primeira escolha na estratégia a adotar, é importante salientar que nem todos os testes podem ser realizados de forma automática. É essencial que a equipa de desenvolvimento e qualidade avalie qual a estratégia de testes mais adequada a seguir, tendo em conta as vantagens e desvantagens de cada, uma vez que, não existe uma técnica infalível e capaz de detetar todos os tipos de erros. Segundo Kaner et al. [2002], uma boa estratégia de teste deve ser:

- **Específica ao produto:** As técnicas utilizadas para a definição da estratégia devem ser as corretas e adequadas ao produto em questão, sendo preferível a adoção de estratégias específicas ao

produto ao invés de estratégias genéricas;

- **Orientada ao risco:** A estratégia definida deverá focar-se nos componentes principais do produto e nos quais a existência de falhas é mais crítica;
- **Diversificada:** Nem todos os problemas podem ser solucionados do mesmo método e com as mesmas técnicas logo é essencial que a estratégia adotada inclua várias técnicas de forma a identificar e corrigir o maior número de falhas possíveis;
- **Exequível:** Após definida a estratégia esta deverá ser possível de concretizar tendo em conta as capacidades, ferramentas e tempo útil disponível do *tester*.

2.1.3 Ciclo de Vida

Os testes de *software*, assim com o próprio processo de desenvolvimento em que se inserem, possuem um ciclo de vida com um conjunto de etapas e ações bem definidas, de forma a uniformizar o processo e melhorar a qualidade do produto desenvolvido, conhecido como o **Ciclo de Vida do Teste de Software** (**CVTS**). Segundo Sommerville [2016] o modelo tradicional de testes agrega as seguintes etapas:

1. **Construção dos Casos de Teste:** são elaborados diferentes cenários (casos de teste) tendo em conta os requisitos do sistema e o comportamento e resultados expectáveis;
2. **Preparação dos Dados de Teste:** são preparados e gerados todos os *inputs* e informação necessária para a execução dos casos de teste;
3. **Execução do Teste:** o teste é executado automaticamente, através da execução de *scripts* de teste ou da utilização de ferramentas de auxílio, ou de forma manual por um *tester*;
4. **Análise e Comparação de Resultados:** os resultados provenientes da execução do teste são analisados e comparados com os resultados expectáveis para os respetivos casos de teste, resultando deste processo a criação de um relatório com toda a informação relevante como, por exemplo, a descrição do teste executado e os seus resultados bem como os defeitos encontrados e possíveis propostas de resolução para os mesmos.

Na sequência de passos apresentada anteriormente não se encontra explicita a fase de análise de requisitos e criação do plano de testes, enunciando todos os passos para a realização de um teste e a simulação de utilização por parte de um utilizador, mas compreende-se que esta etapa é o ponto de partida do **CVTS** [Institute, 2014], apresentada na Figura 1 a sequência de todas as etapas enunciadas.

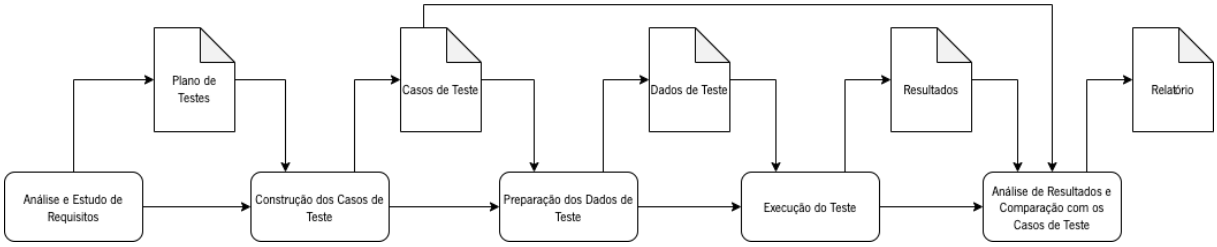


Figura 1: Ciclo de Vida de um Teste de Software[3]

2.1.4 Documentação e Entregáveis dos Testes de Software

No **ES**, a documentação é essencial na comunicação de informação à audiência do projeto em que se insere, quer sejam eles utilizadores especializados em tecnologia ou não, transparecendo conhecimentos sobre o sistema a desenvolver e ajudando também a manter a qualidade do *software* em altos níveis, a planificação e gestão de tarefas e a transferência de elementos da equipa entre projetos [Kipyegen e Korir, 2013].

Ao longo de todo o **CVTS** são produzidos vários tipos de documentos, podendo estes ser divididos em dois grandes grupos [Kit, 1995]:

- **Planificação/Especificação:** Todos os documentos produzidos, usualmente antes da execução do teste, detalhando o requisito a testar e todos os passos necessários para a sua realização bem como os critérios de aceitação associados ao mesmo;
- **Reporting***:* Documentos produzidos após a execução do teste, contendo todas as informações relevantes da execução e os respetivos resultados.

Segundo Aimar [1998], existem três tipos de de entregáveis distintos que são essenciais no processo de testes: plano de testes, casos de testes e *reports*.

Plano de Testes

O Plano de Testes detalha a estratégia elegida para a realização de um teste, contendo todos os requisitos necessários para a correta realização do teste e análise dos respetivos resultados, o tipo do teste, assim como, todas as métricas definidas para o processo [Institute, 2014]. As principais métricas passam pela definição do critério de saída de um teste, ou seja, o critério ou valor que quando alcançado determina

[3] Adaptado da Figura 8.3 do livro Sommerville [2016]

o fim da execução do teste ou, em caso contrário, o seu retorno a um estado estável [Ammann e Offutt, 2016] e pela estimação do esforço necessário para a realização do mesmo, principalmente impactado pelo nível de complexidade do teste em causa.

Casos de Testes

Os Casos de Testes são documentos com uma estrutura bem definida que representam passo a passo tudo aquilo que deve ser realizado para levar o teste a cabo, contendo a ação para a simulação de utilização, os dados de *input* e os resultados expectáveis da aplicação perante as ações e dados introduzidos no sistema para uma única característica[Ammann e Offutt, 2016]. Muitas das vezes, surge confusão na distinção entre casos e cenários de teste sendo que estes últimos, devido à sua maior complexidade, representam um conjunto de ações simulando a utilização do sistema num ambiente real para um determinado requisito podendo incluir na sua especificação vários casos de teste.

Reports

Os *Reports*, ou relatórios, contém os resultados observados perante a execução dos casos de teste e a análise em mais detalhe dos mesmos. Dependendo do resultado obtido ou do estado em que se encontra um teste, existem diferentes tipos de relatórios que podem ser gerados [Institute, 2014]:

- **Relatório de Estado dos Testes:** Estes relatórios são criados ainda antes da execução dos primeiros testes e atualizados, em regra geral, diariamente de forma a que todos os elementos da equipa e *stakeholders* consigam obter informação sobre o estado de desenvolvimento do sistema;
- **Relatório de Incidentes:** Um incidente representa o sintoma associado a uma falha do sistema que por sua vez sinaliza ao utilizador a ocorrência de um erro [Jorgensen, 2013]. Os relatórios de incidentes são criados para reportar uma determinada falha à equipa de desenvolvimento, devendo conter uma comparação entre o resultado obtido e aquele que era esperado e, se possível, informações sobre o motivo dessa falha. Entre outros detalhes, os relatórios de incidentes devem conter a identificação do caso de teste, o requisito em questão e a descrição da falha observada;
- **Relatório Final:** Relatório criado assim que o teste e todas as atividades a ele relacionadas são concluídas, a entregar ao cliente e *stakeholders* do projeto. No relatório final são detalhados todos os casos de teste concluídos com detalhe e enunciados aqueles em que existiram falhas.

2.2 Ecossistema Atlassian

Nesta secção são abordados os produtos *JIRA* e *Confluence* da empresa *Atlassian*[4] e quais as vantagens de utilização dos mesmos num ambiente de desenvolvimento de *software*.

2.2.1 JIRA

O *JIRA*, criado em 2002, foi o primeiro produto lançado pela *Atlassian* e é utilizado, em inúmeras empresas espalhadas pelo mundo, para que gestores de projetos e programadores acompanhem o estado dos projetos em que se encontram envolvidos e monitorizem as diferentes tarefas a eles associados.

Para que melhor seja compreendido, é necessário ter em mente os seguintes conceitos utilizados no ambiente do *JIRA* [Packt, 2015]:

- **Projeto:** No *JIRA* um projeto é um conjunto de *issues*, visível por todos os utilizadores da instância onde este é criado, utilizado para coordenar e monitorizar o desenvolvimento de um produto. Cada projeto é identificado pela sua *Project Key*, possui um tipo e um gestor de tarefas denominado como *Project Lead* [Defining a Project, 2022];
- **Categorias de Projetos:** Permite agregar projetos por categorias, facilitando a gestão dos mesmos quando o número de projetos existentes numa instância *JIRA* é consideravelmente elevado;
- **Issue:** Uma *issue* no *JIRA* representa "algo que necessita de ser feito", podendo assumir diferentes representações consoante as necessidades num projeto. No âmbito do desenvolvimento de *software* uma *issue* assume principalmente os tipos: tarefa, sub-tarefa, *story*, *bug* e *epic*;
- **Componentes:** São uma secção dos projetos que permitem a agregação de *issues* num projeto.
- **Versões:** É possível criar diferentes versões num projeto para melhor organização no desenvolvimento do produto e gestão das *releases*.

Uma das principais vantagens do ecossistema *Atlassian*, e que se engloba aos seus produtos como é o caso do *JIRA*, é a possibilidade de instalação de *plugins* tanto da própria *Atlassian* como de empresas ou programadores independentes, conhecidos como *Atlassian Vendors*. Estes *plugins* permitindo acrescentar várias funcionalidades às ferramentas da *Atlassian*, tornando os seus produtos ainda mais completos e customizáveis às diferentes necessidades de cada equipa.

[4] A *Atlassian* é uma empresa de *software* Australiana, fundada em 2002, que têm como principal alvo o mercado de desenvolvimento de *software* oferecendo a todos os diferentes intervenientes produtos que os auxiliam em todas as fases do processo.

Por outro lado, é possível a instalação, configuração e utilização do *JIRA* em duas plataformas distintas: *JIRA Server*, primeira opção disponibilizada no mercado e que atualmente ainda possui o maior número de utilizadores: *JIRA Cloud* (**SaaS**), opção mais recente e que surge com a par das necessidades do mercado e da aposta no desenvolvimento em *cloud*. De seguida, na Tabela 2 são apresentadas as principais vantagens e motivos de utilização para cada opção [Rodrigues].

Server	**Cloud**
Controlo do código fonte	Alta escabilidade e disponibilidade
Maior flexibilidade de customização	Não é necessária infraestrutura própria
Maior número de *plugins* disponíveis	Sem requisitos de hardware adicionais, e consequentemente menor custos
Controlo sobre atualizações	Departamento de IT não realiza nenhum trabalho adicional (instalação e configuração)
Sem restrições de administração	Não é necessário realizar tarefas de manutenção
Acesso total a bases de dados	A curto prazo, os sistemas podem ser usados mais rapidamente pela equipa

Tabela 2: Comparação entre *JIRA Server* e *JIRA Cloud*

Para atender às altas necessidades de escalabilidade, desempenho, disponibilidade e requisitos críticos de negócio de algumas empresas, não suportadas pela versão *Server*, a *Atlassian* disponibiliza também a opção *JIRA Data Center*. Esta grande diferença entre versões do *JIRA* deve-se à arquitetura distribuída do *JIRA Data Center*, permitindo o balanceamento e distribuição da carga entre os diferentes servidores e apresentando uma maior redundância a falhas, ao invés do que acontece na versão *JIRA Server* uma vez que a arquitetura utilizada baseia-se num único servidor para disponibilização do serviço.

2.2.2 Confluence

O *Confluence*, um produto da *Atlassian* criado em 2004, é uma ferramenta de colaboração utilizada para a criação de documentação no desenvolvimento de um produto, estruturando e organizando todo o processo e permitindo a colaboração e partilha de ideias entre todos os elementos da equipa para que estes obtenham toda a informação que necessitam para desempenhar as suas funções. De igual modo ao *JIRA*, o *Confluence* encontra-se disponível nas versões *Cloud*, *Server*, e *Data Center*.

Para melhor entender o que é o *Confluence* e como este pode ser utilizado, é necessário compreender o que são as *pages* e os *spaces*. Estas ferramentas, abordadas em mais detalhe de seguida, podem ser utilizadas para a criação e partilha informações entre diferentes tipos de utilizadores.

Pages

As *pages*, ou páginas, são documentos criados no *Confluence*, que podem ser editados e visualizados em tempo real por qualquer pessoa com acesso à instância e com as permissões necessárias. Estes documentos podem ser criados a partir de *templates* disponíveis para uma rápida criação de vários tipos de documentos, ou criados de raiz sempre que as entidades responsáveis pela sua criação considerarem que é a melhor opção e a cada lançamento ou publicação é gerada uma nova versão do documento e guardada no histórico da página, permitindo a reversão para versões anteriores sempre que exista algum erro ou surja a necessidade para tal[Confluence basics, 2022].

Para edição, além de todas as funcionalidades genéricas e comuns a um editor de texto, são disponibilizados blocos de conteúdo dinâmico denominados como *macros*. Estes blocos podem originados pela própria *Atlassian*, facilitando a integração com outros produtos do ecossistema e simplificando a visualização de propriedades desses mesmos produtos numa página do *Confluence*, ou de origem externa, acrescentando outras funcionalidades às páginas e/ou permitindo a disponibilização de conteúdo externo ao ecossistema.

Spaces

Os *spaces*, ou espaços, assim como o próprio nome o indica, são locais de trabalho no *Confluence* onde podem ser criadas e consultadas *pages* com informação relativa a um determinado projeto ou produto, ou seja, páginas que mantém algum elo de relação entre si. Por omissão, no *Confluence* existem vários espaços tipo [Use spaces to organize your work, 2022]: *Team Spaces* utilizados por equipas e onde diferentes utilizadores com os mesmos interesses podem partilhar e consultar informações mais facilmente; *Project Spaces* onde é agrupada toda a informação relevante e associada a um projeto; *Personal Space* onde um utilizador guarda informações pessoais ou de interesse próprio.

2.3 Xray

Nesta secção é feito o estudo da aplicação *Xray*, bem como, das suas características, das funcionalidades que oferece e das vantagens da sua utilização perante os principais competidores no mercado dos testes de *software*.

2.3.1 O que é o Xray?

> *"This high-quality software supports agile development processes very well, integrates smoothly into Jira and proves to be the missing part for testing in Jira in any aspect."*
>
> Michael Merwald, Test Manager - BMW [5]

O *Xray Test Management*, criado pela *Xpand IT* [6] e disponível no *marketplace* da *Atlassian* desde o início de 2014, é uma ferramenta de testes para o *JIRA* que permite a gestão e monitorização de testes de *software* ao longo de todo o **CVTS**, com funcionalidades específicas para planificação, construção, execução e *reporting* de testes.

De forma a abranger todo o processo de testes, o *Xray* cria alguns novos tipos de *JIRA issues* para as seguintes fases [Test Process, 2022]:

- **Planificação:** *Test Plan issues* representam o plano de testes a ser executado para uma determinada versão do *software*.
- **Construção:** *Test issues* descrevem passo a passo o teste a executar, podendo ainda conter pré-condições à sua realização formalizadas em *Precondition issues*. Para melhor gestão dos testes podem ser criadas *Test Set issues* agrupando e organizando testes de forma lógica, sendo que, não existe nenhum impedimento para que um teste pertença a vários grupos de testes.
- **Execução:** Para a execução controlada de um ou mais testes podem ser criadas *Test Execution issues*, associando a cada teste o seu resultado e estado de execução. Uma vez que um determinado teste pode possuir diferentes estados conforme o ambiente ou versão do produto em que é executado, estas *issues* permitem armazenar os diferentes resultados para as diferentes condições de execução.
- **Reporting:** Para consulta dos resultados dos testes executados podem ser analisadas as próprias *Test Execution issues* ou utilizadas as ferramentas disponibilizadas pelo *Xray* para geração de vários tipos de relatórios. Também são disponibilizados vários *gadgets* que podem ser utilizados para a complementação de *dashboards* de forma a acompanhar os diferentes estados de todas as etapas do processo de testes em tempo real.

[5] Retirado de https://www.getxray.app/customers

[6] Empresa de consultoria e desenvolvimento de *software* portuguesa, criada no final do ano 2003, com vários produtos e serviços de sucesso em diferentes ramos de IT como *Big Data, User Experience* e *Business Intelligence & Analytics*.

Além dos *Test Sets* enunciados anteriormente, o *Xray* também permite o agrupamento de testes em *Test Repositorys* [Test repository, 2022]. Cada projeto configurado para utilização do *Xray* possui o seu próprio repositório que, por sua vez, possui várias pastas onde testes do mesmo contexto ou com algum tipo de dependência e/ou parentesco possam ser armazenados de forma a facilitar a consulta dos mesmos. Cada teste apenas pode estar associado a uma única pasta por repositório. Os *Test Repositorys* também disponibilizam métodos de consulta para todos os *Test Sets* do respetivo projeto que representam.

Outro dos conceitos de possível abordagem aquando a utilização do *Xray* para a realização de testes são os *Test Environments* [Working with test environments, 2022] permitindo a associação de testes ao respetivo **System Under Test** (**SUT**), ou seja, ao respetivo sistema ou ambiente no qual o teste está a ser executado. Este sistema pode representar e assumir diferentes significados como o estado de desenvolvimento do produto, um dispositivo ou sistema operativo e, a nível de desenvolvimento *web*, o *browser* no qual foi realizado o teste. Desta forma não é necessária a duplicação de testes quando estes necessitam de ser testados em mais do que um ambiente, sendo que, para cada um dos ambientes o resultado de execução do teste pode diferenciar.

Na etapa de *Reporting*, a mesma *issue* ou teste pode apresentar resultados diferentes dependendo da respetiva análise estratégica definida para a geração do relatório[Coverage analysis, 2022]. No *Xray* existem três dimensões de análise perante uma *issue*:

- **Latest**
- **Version**
- **Test Plan**

Para cada dimensão enunciada acima, existem outros dois critérios que podem influenciar o estado calculado:

- **Test Environment**
- **Final statuses precedence over non-final ones**

Estes conceitos e os respetivos efeitos são aprofundados nos próximos capítulos do documento.

2.3.2 Concorrência de Mercado

> *"The top three benefits of using Xray are better collaboration, higher transparency, and lower licensing cost."*
>
> Sanny Mok, Aashish Sharma [7]

No momento de escrita do presente documento[8], o *Xray* encontra-se perto de alcançar as 28 000 instalações[9] sendo uma das principais aplicações para gestão e monitorização de testes presentes no *marketplace* da *Atlassian*, no entanto, existem várias outras alternativas:

- **Zephyr Scale** (28.1 mil instalações)
- **AIO Tests** (3.1 mil instalações)
- **Requirements & Test Management for Jira** (2.7 mil instalações)
- **QMetry** (2.3 mil instalações)

A *Zephyr* é um conjunto de três aplicações, a *Zephyr Squad* para equipas e projetos de menor escala, a *Zephyr Scale* para equipas e projetos mais complexos oferecendo também um maior número de ferramentas e soluções ao nível dos testes de *software* e a *Zephyr Enterprise*, versão *standalone* para a gestão de testes utilizada por empresas ou companhias que possuem processos com necessidades de alta disponibilidade, escalabilidade e privacidade de forma a manter a sua estrutura.

Uma vez que a *Zephyr* e o *Xray* são as principais aplicações para gestão de testes presentes no mercado e com uma proposta de valor e funcionalidades bastante semelhantes, é compreensível a comparação e análise das vantagens e desvantagens de cada uma das ferramentas. Quanto ao processo de criação e organização de testes, o *Xray* permite a criação e reutilização de pré-condições e o agrupamento de testes em *Test Sets* enquanto que a *Zephyr* não oferece tais funcionalidades. Quanto ao nível de **Continuous Integration** (**CI**) com outras ferramentas de testes o *Xray* disponibiliza o maior número integrações [Xray vs Zephyr, 2022]. Por sua vez, a nível de custos a *Zephyr* é a única aplicação a fornecer uma alternativa grátis uma vez que a *Zephyr Squad* não possui qualquer custo associado para instâncias com até 10 utilizadores mas sendo todos os preços para as restantes condições de utilização superiores quando comparados com aqueles efetuados pelo *Xray*.

[7] Sanny Mok and Aashish Sharma. The Total Economic Impact™ Of Xray. Forrester Consulting, 2022

[8] Novembro de 2022 a Outubro de 2023

[9] Uma vez que uma instalação refere-se a uma instalação numa instância *JIRA* e que cada instância pode suportar até 20.000 utilizadores, a própria *Xpand It* estima que a sua aplicação alcançou até ao momento mais de 4.5 milhões de utilizadores.

Capítulo 3

O Problema e os seus Desafios

Neste capítulo é abordado o âmbito do problema a tratar durante a realização da presente dissertação de mestrado, assim como uma primeira proposta de solução, analisando e comparando as diferentes ferramentas disponibilizadas pela *Atlassian* para o desenvolvimento de aplicações, e os respetivos resultados esperados a alcançar.

3.1 Descrição

Esta secção descreve o problema e justifica a necessidade de implementação do produto idealizado.

Atualmente, o *Xray* apenas possui integração com o *Confluence* nas versões *server*, permitindo a reutilização e integração dos *JIRA Gadgets*, criados pelo próprio *Xray*, em páginas *Confluence*. Os *JIRA Gadgets*, ou *Dashboard Gadgets*, são componentes nativos do *JIRA* ou criados por aplicações externas que permitem a exibição de informações pertinentes ao seu contexto, sendo que cada *gadget* consiste num painel que pode ser adicionado e configurado num *dashboard*, sendo este um conjunto de diferentes *gadgets*. Nas versões *Server* e *Data Center*, um *JIRA Gadget* pode ser reutilizado em instâncias *Confluence* na forma de uma *macro*.

No entanto e tendo em conta que desde setembro de 2022, a *Atlassian* descontinuou a possibilidade de utilizar estes *gadgets* em instâncias de *Confluence Cloud* [Use gadgets to add dynamic content, 2022] e que, a partir de fevereiro de 2024, deixará de disponibilizar suporte para as versões *server* dos seus produtos [Moving from server to cloud for developers, 2021]. Dada esta descontinuação, existe a necessidade de adaptar ou criar novas soluções para esta nova realidade. Uma dessas novas soluções passa pela migração para o desenvolvimento em *cloud*, sendo que a integração do *Xray* com o *Confluence* e a possibilidade de gerar relatórios com dados provenientes do *JIRA* não é exceção.

Deste modo, a aplicação a desenvolver deverá disponibilizar uma *Confluence Macro* que permita a um administrador do produto em ambas as instâncias (*JIRA* e *Confluence*) a geração de quatro tipos de

relatórios originários do *Xray* em páginas do *Confluence*:

- **Overall Coverage**
- **Tests List**
- **Traceability**
- **Test Runs**

No momento de configuração da *macro*, o utilizador poderá escolher qual o tipo de relatório a gerar bem como a respetiva análise estratégica a abordar.

3.2 Análise Estratégica e outros Filtros

Nesta secção, é apresentada a análise estratégica utilizada pelo *Xray* para a geração dos relatórios e do cálculo dos respetivos estados, assim como, alguns dos filtros também disponibilizados, com especial ênfase naqueles que se encontram disponíveis nos diferentes tipos de relatórios a implementar.

3.2.1 Análise Estratégica

Nas funcionalidades disponibilizadas pelo *Xray*, existe a possibilidade de configurar diferentes tipos de *issues* como *testable/coverable*, ou seja, é possível a sua associação com diferentes testes de forma a validar o seu **Critério de Aceitação** (**CA**), definido pelo resultado de cada uma das execuções dos testes a ela associada segundo a análise estratégica escolhida.

O utilizador tem acesso à configuração da análise estratégica em diferentes ecrãs do *Xray*, nomeadamente no momento de consulta de uma *coverable issue* e na grande maioria dos relatórios, podendo ser definida pela escolha de uma de três dimensões disponíveis: *Latest*, *Version* e *Test Plan*. Para além e independentemente de qual a dimensão selecionada, é possível também definir outros dois critérios adicionais, sendo eles o *Test Environment* e o *Final Statuses Precedence* que, da mesma forma, têm impacto no cálculo do resultado do critério de aceitação.

Latest

No âmbito *Latest*, apenas a última execução de cada teste, ou seja a mais recente, é considerada para o cálculo do resultado do **CA** das *issues* a analisar. Por omissão, a dimensão *Latest* é a selecionada quando o ecrã de uma *coverable issue* ou relatório é aberto.

Version

No âmbito *Version*, apenas as execuções de cada teste criadas para a versão pretendida são consideradas no processo de cálculo. Para associar uma versão a uma execução é possível realizá-lo no momento de criação de uma *Test Execution* a partir do ecrã do teste ou através da edição do campo *Fix Version/s* no ecrã da *issue* de execução.

Test Plan

Na dimensão *Test Plan*, apenas as execuções e respetivos testes associados ao *Test Plan* pretendido são considerados no momento de cálculo, sendo que, que caso o teste não se encontre associado é apresentada a informação de tal acontecimento.

Test Environment

O critério *Test Environment* permite complementar as dimensões acima anunciadas, filtrando apenas as execuções aos testes realizadas num determinado ambiente. Por omissão e, caso não seja alterado pelo utilizador, é selecionada a opção *All Environments* e são considerados todos os ambientes.

Final Statuses Precedence

No *Xray*, diferentes estados de um teste, etapa de teste ou execução podem possuir diferentes prioridades, nomeadamente identificadas pelo atributo *Final*, ou seja, um estado final apresenta prioridade em relação aos estados não finais.

Deste modo, quando selecionado o critério *Final Statuses Precedence* apenas execuções com estados finais são consideradas para o cálculo do estado do **CA**, ignorando as execuções intermédias, ou seja, execuções com estados não finais.

3.2.2 Outros Filtros

Além das diferentes dimensões acima referidas para a definição da análise estratégica a adotar é possível, através da utilização de diferentes filtros, a integração destes dois domínios e complementação dos resultados obtidos, sendo que, apenas aqueles que satisfazem todas as condições são considerados na construção do relatório e consequentemente apresentados ao utilizador.

É possível a definição de diferentes filtros para cada entidade ou tipo de *issue*, assim como, a reutilização de filtros *JQL* criados na instância *JIRA* onde o *Xray* se encontra instalado.

Saved Filter

O campo *Saved Filter* permite a seleção de um filtro *JQL*, permitindo a conjugação de vários filtros numa única *query* para a respetiva filtragem das *coverable issues*. De salientar que, quando selecionado um filtro *JQL* qualquer um dos restantes filtros são ignorados e apenas a *query JQL* é considerada no processo de filtragem.

Test Execution

Quando selecionado algum dos seguintes filtros relativos às execuções de testes, apenas são apresentados resultados de execuções:

- *Project:* que pertençam ao projeto selecionado;
- *Fix Version/s:* associadas à versão ou versões selecionadas;
- *Test Environment:* executadas nos ambientes de teste escolhidos;
- *Revision:* cujo o atributo *Revision* contenha o valor inserido;
- *Test Plan:* associadas ao plano de testes pretendido;
- *Contains:* cuja a chave ou sumário da *issue* de execução do teste contenha o valor inserido.

Test

Quando selecionado algum dos seguintes filtros, apenas são apresentados resultados de testes:

- *Component:* associados aos componentes selecionados;
- *Priority:* cuja prioridade da *issue* é equivalente à selecionada;
- *Contains:* cuja a chave ou sumário da *issue* do teste contenha o valor inserido.

Test Runs

Quando selecionado algum dos seguintes filtros, apenas são apresentados resultados de *Test Runs*:

- *Assignee:* designadas para o utilizador selecionado;
- *Executed By:* executada pelo utilizador selecionado;
- *Status:* cujo estado da *Test Run* é equivalente ao selecionado.

3.2.3 Cálculo de Estados para Coverable Issues

No *Xray*, testes e *coverable issues* podem ser representados pelo seu estado, no entanto, as categorias de estados de cada uma destas entidades diferem. O estado de um teste é caracterizado pelo estado da respetiva execução, enquanto que, o estado de uma *coverable issue* é representado pelo mapeamento e conjunção dos estados de cada um dos testes a ela associada.

Um teste pode assumir, a cada instante, um dos seguintes estados:

- **PASS**: o teste foi executado sem a presença de defeitos;
- **FAIL**: o teste foi executado e foram detetados defeitos;
- **ABORTED**: o teste foi abortado;
- **TODO**: o teste ainda não foi executado;
- **EXECUTED**: o teste encontra-se em execução.

Os primeiros três tipos de estados representam estados finais, ou seja, a execução do teste foi concluída independentemente do resultado final. Por outro lado, os últimos dois tipos representam estados não finais.

Apesar dos tipos de estados possíveis para uma *coverable issue* diferirem dos estados possíveis de um teste, existe um mapeamento entre eles, como se encontra demonstrado na Tabela 3.

Estado (Teste)	**É Final?**	**Estado (Coverable Issue)**
PASS	Sim	OK
FAIL	Sim	NOK
TODO	Não	NOTRUN
ABORTED	Sim	NOTRUN
EXECUTING	Não	NOTRUN

Tabela 3: Mapeamento entre estados de testes e estados de *coverable issues*[1]

No caso de uma *coverable issue* não se encontrar associado com nenhum teste, o seu estado será do tipo *UNKNOWN*, não se encontrando representado na tabela acima apresentada.

[1] Retirado de https://docs.getxray.app/display/XRAYCLOUD/Understanding+the+calculation+of+coverage+status+and+the+status+of+Tests

3.3 Relatórios a Implementar

Nesta secção são descritos e aprofundados em detalhe os diferentes relatórios gerados pelo *Xray*, disponibilizados na *macro Confluence* desenvolvida, após a análise do respetivo funcionamento e comportamento do *Xray*, assim como, da documentação disponibilizada para os mesmos [Reports & Analysis, 2022].

3.3.1 Tests List

O relatório *Tests List* (exemplo visível na Figura 2) consiste numa tabela com todos os testes associados ao projeto ou projetos em análise e os seus respetivos estados para a análise estratégica definida. Para além do estado do teste e do seu tipo, atributos do *Xray*, é possível consultar qualquer um dos outros valores associados aos *custom fields* do teste em questão, na forma de colunas. Um *custom field* representa um campo configurável no *JIRA* que permite a personalização de determinados valores associados a uma *issue*, de modo a atender às diferentes necessidades e especificações dos projetos e utilizadores.

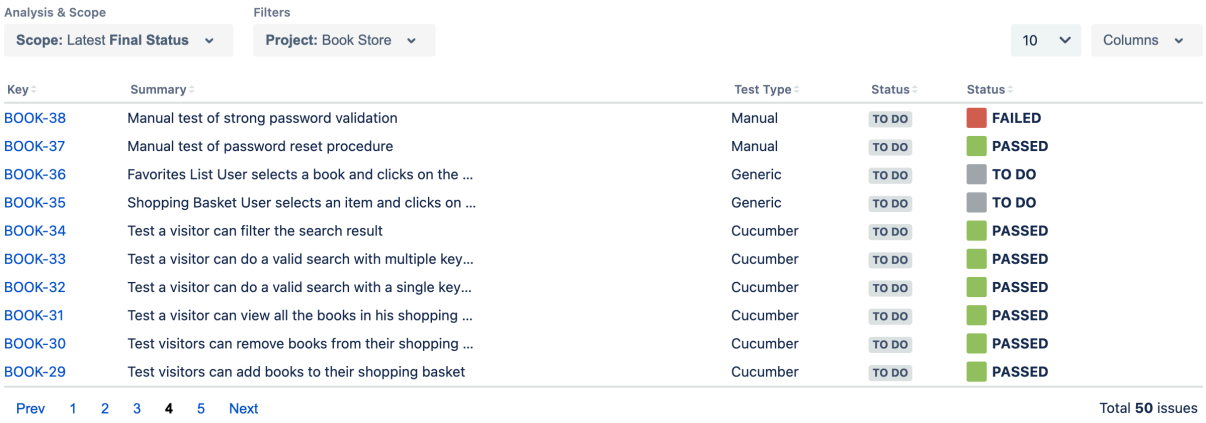


Figura 2: Exemplo de um relatório do tipo *Tests List*

O utilizador, no momento de consulta, pode alterar o número de resultados apresentados (10, 50 ou 100), assim como, quais as colunas apresentadas e proceder à reordenação da lista de testes por qualquer uma das mesmas. Dependendo do tipo do atributo exibido, existem vários tipos de interação disponíveis ao utilizador como a redirecionamento para outras páginas, por exemplo o redirecionamento para a página do teste quando o utilizador clica no campo *Key*, ou o retrato de informações adicionais na forma de balões, por exemplo a exibição de um balão com a descrição do estado quando o utilizador posiciona o cursor sobre o campo *Status*.

3.3.2 Test Runs List

O relatório *Test Runs List* disponibiliza uma tabela para listagem de todas as *Test Runs* (exemplo visível na Figura 3) cujas propriedades respeitem os filtros definidos no momento de configuração do relatório. Por se tratar de uma entidade inteiramente gerida pelo *Xray* e não existir nenhum tipo de *issue* para a identificar, uma *Test Run*, não é possível definir uma análise estratégica.

Test				Test Run											Linked Defects	
Key	Summary	Components	Priority	Test Execution	Test Environments	Test Plan	Test Type	TestRun Status	Executed By	Test Execution Fix Version/s	Revision	Started At	Finished At	Elapsed Time	Open	Closed
TP-2	TC001: Test Create a Homepage			TP-5 View Details	-	-	Manual	EXECUTING	João Sá		-	14/Nov/22 03:29 PM	-	17 secs	0	0
TP-12	Add items to Shopping Basket			TP-17 View Details	TE1	TP-11	Manual	PASSED	João Sá		-	15/Nov/22 12:06 PM	15/Nov/22 12:08 PM	2 mins	0	0
TP-14	Buy items from the shopping basket			TP-17 View Details	TE1	TP-11	Manual	FAILED	João Sá		-	15/Nov/22 12:10 PM	15/Nov/22 12:10 PM	0 millisec	0	0
TP-10	Test 2 for user story WEB-1	Visits		TP-21 View Details	-	-	Manual	PASSED	João Sá		-	23/Nov/22 06:14 PM	23/Nov/22 06:14 PM	3 secs	0	0
TP-9	Test 1 for user story WEB-1	Visits		TP-21 View Details	-	-	Manual	PASSED	João Sá		-	23/Nov/22 06:15 PM	23/Nov/22 06:15 PM	0 millisec	0	0
TP-23	Test NOT RUN 1			TP-24 View Details	-	-	Manual	TO DO			-	-	-	-	0	0

Figura 3: Exemplo de um relatório do tipo *Test Runs List*

A tabela gerada encontra-se dividia em três secções:

- **Test**: apresenta todas as propriedades do teste para a qual a *Test Run* foi criada;
- **Test Runs**: apresenta todas as informações associadas à *Test Run* em causa e respetiva *Test Execution*;
- **Linked Defects**: apresenta a totalidade de defeitos encontrados durante a execução do teste, divididos pelo estado de resolução da respetiva *issue* criada para identificação do defeito.

Uma vez que não existe qualquer restrição quanto ao número de execuções realizadas para um determinado teste, é possível que o mesmo teste se encontre presente em diferentes linhas da tabela, ou seja, uma entrada para cada execução do mesmo.

3.3.3 Overall Coverage

O relatório *Overall Coverage* (exemplo visível na Figura 4) apresenta os estados de todas as *issues* do projeto configuradas como *coverable issues*, na forma de um gráfico de barras horizontal e normalizado. Consoante a análise estratégica definida, os resultados apresentados permitem a análise e consulta do estado dos requisitos para o respetivo momento e/ou contexto.

Esta análise estratégica pode ser parametrizada para que apenas sejam apresentados os resultados mais recentes, os resultados de uma determinada versão do produto ou aqueles que estão associados a um determinado plano de testes. Todas estas escolhas podem ser combinadas com a escolha de um determinado ambiente de testes e apenas as execuções associadas a ele são consideradas para a geração do relatório. De forma a separar os resultados para uma melhor compreensão, estes podem ser agrupados por inúmeros campos como, por exemplo, a prioridade ou resolução das *issues*.

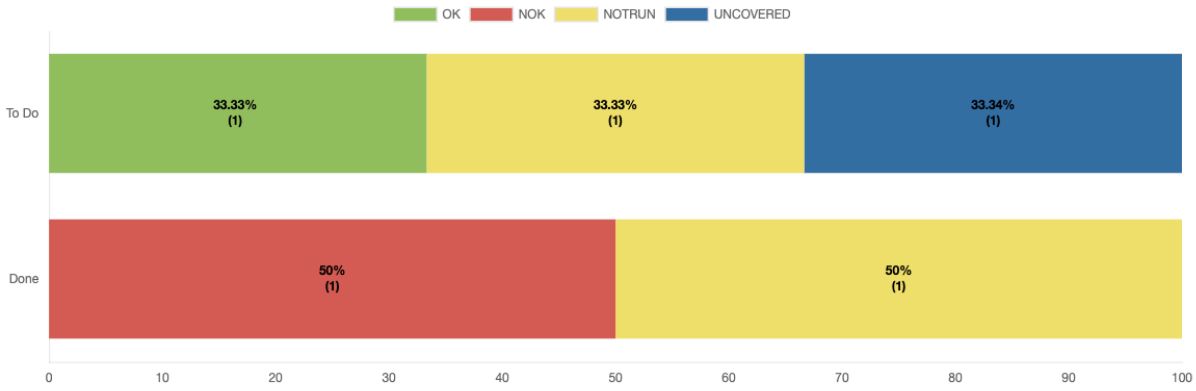


Figura 4: Exemplo de um relatório do tipo *Overall Coverage*

É importante salientar que o relatório descrito também apresenta algum nível de detalhe e profundidade quanto à interatividade com o utilizador. Sempre que o cursor é posicinado sobre uma secção de uma das barras é apresentado um *popup* com informação relativa ao grupo, estado e valor (em percentagem) da respetiva secção e, quando clicado, é apresentada uma tabela com as *issues* e informação detalhada para cada uma, como por exemplo, o seu grau de completividade.

3.3.4 Traceability

O relatório *Traceability* (exemplo visível na Figura 5) é o relatório com maior nível de complexidade disponibilizado pelo *Xray* e, consequentemente, aquele com maior nível de informação associada a um teste de *software*.

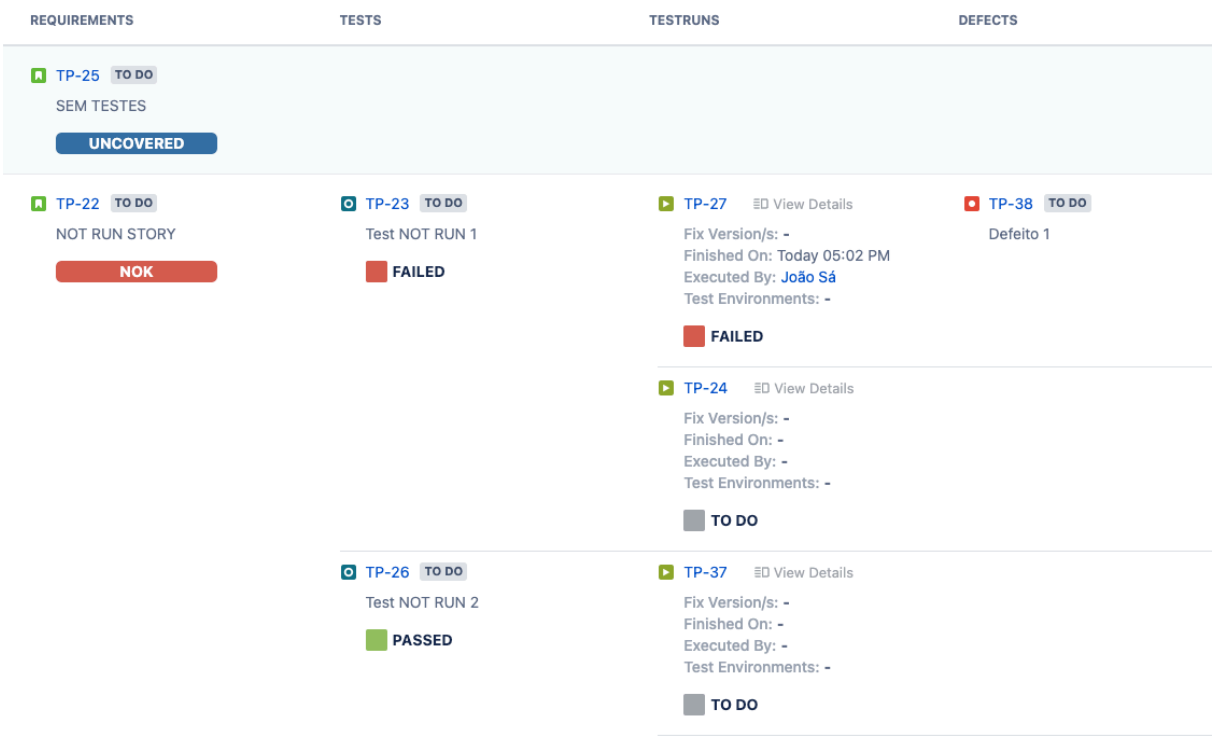


Figura 5: Exemplo de um relatório do tipo *Traceability*

Este relatório permite a consulta dos testes de *software* associados a 3 tipos de *issues* (*Epic*, *Story* e *Sub-task*) e de todo o seu **CVTS**, ou seja, de todas as execuções de testes realizadas e dos respetivos estados e defeitos se encontrado algum.

Assim como nos restantes relatórios é possível definir uma análise estratégica e, por omissão e motivos de *performance*, apenas são mostrados até um máximo de três execuções por teste podendo, no entanto, caso o utilizador pretenda, poderão ser apresentadas a totalidade das mesmas.

3.4 Proposta de Solução e Resultados Esperados

Nesta secção é descrita a proposta de solução, tendo em conta o estudo das diferentes ferramentas presentes no ambiente de desenvolvimento *Atlassian*, e são enunciados alguns pontos de reflexão e avaliação ao produto desenvolvido.

3.4.1 Forge vs Connect

Para o desenvolvimento de aplicações para produtos do seu ecossistema, a *Atlassian* fornece, aos programadores interessados pela criação de tais aplicações, duas *frameworks* distintas: *Atlassian Connect* e

Forge. Na Tabela 4 é apresentada uma breve comparação entre estas duas tecnologias.

	Forge	**Connect**
Hosting	Gerido pela *Atlassian*	Gerido pelo *Atlassian Vendor*
UI	Módulos disponibilizados pela *Atlassian* (*custom UI* e *UI kit*)	Módulos disponibilizados pela *Atlassian*
APIs	*Managed* **REST** e *JavaScript APIs*	*REST* e *JavaScript APIs*
Linguagens da aplicação	*Serverless Node.js*	Qualquer linguagem (*Atlassian* oferece suporte para *Node.js* e *Java*)
Autorização e Autenticação	Gerido pela *Atlassian*	Equipa de desenvolvimento possui total controlo sobre autorização e autenticação da aplicação
Distribuição	Uso próprio ou comercial	Uso próprio ou comercial
Compatibilidade	*JIRA, Confluence*	*JIRA, Confluence* e *Bitbucket*

Tabela 4: Comparação entre as *frameworks Forge* e *Connect* para desenvolvimento em *cloud*.[2]

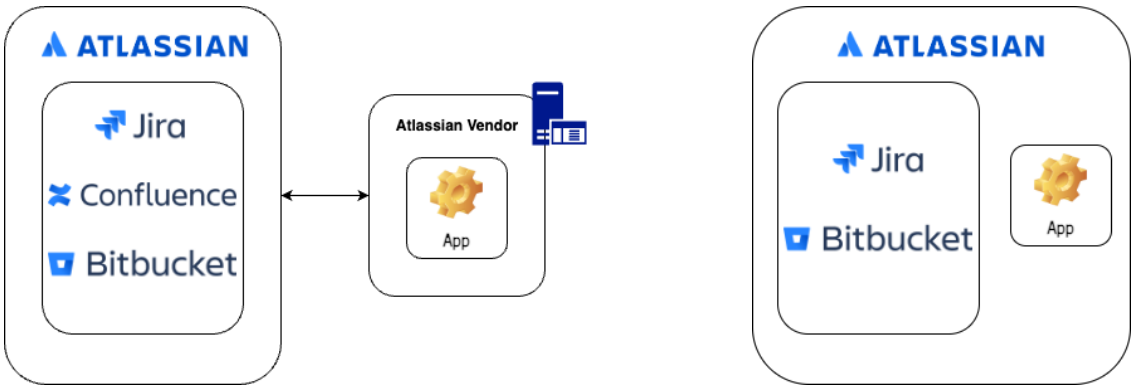
A primeira, *Atlassian Connect*, é a forma mais antiga de desenvolver aplicações para o ecossistema *Atlassian* apresentando uma arquitetura *loosely coupled*, ou seja, o produto alvo para o qual a aplicação é desenvolvida e a própria aplicação são hospedados e executados em ambientes e infraestruturas independentes entre si, comunicando, neste caso, através de dois métodos:

- **App descriptor:** ficheiro **JSON** que descreve toda a informação da aplicação, ou seja, informações sobre hospedagem da aplicação e respetivas permissões, restrições e módulos a ela associada ao produto *Atlassian* na qual é instalada [Connect app descriptor, 2022];
- **REST APIs:** todos os produtos *Atlassian* possuem a própria *REST API* com a qual a aplicação pode interagir de forma a obter informações ou executar ações sobre o produto na qual se encontra instalada.

Desta forma, a equipa de desenvolvimento do respetivo *Atlassian Vendor* têm liberdade total de escolha na forma como irá desenvolver e hospedar a aplicação, assim como, do modo como os dados são tratados e armazenados. No entanto, esta liberdade também traz consigo algumas desvantagens, nomeadamente, ao nível de segurança, confiança dos utilizadores e disponibilidade [Muschol, 2021]. Uma vez que a aplicação se encontra desacoplada do ambiente *Atlassian* mas, por sua vez, continua a ter acesso aos dados do ambiente e dos respetivos utilizadores é responsabilidade dos programadores assegurar que não existe nenhum comprometimento no acesso aos mesmos. Em alguns casos, devido a este

[2] Adaptado de https://developer.atlassian.com/developer-guide/cloud-development-platform-overview/

desacoplamento, poderão existir situações em que não é garantida a equivalência de disponibilidade da aplicação e do ambiente para a qual foi concebida, inviabilizando a utilização da aplicação por parte dos utilizadores.



(a) Connect (b) Forge

Figura 6: Comparação entre as arquiteturas alto nível das *frameworks Connect* e *Forge*[3]

Com o objetivo de solucionar tais desvantagens, a *Atlassian* lançou no final do ano de 2019 uma nova ferramenta - *Forge* - para o desenvolvimento de aplicações. Por ser uma **Function as a Service** (**FaaS**), a adoção de *Forge* permite com que os programadores apenas se concentrem na construção do produto idealizado, deixando preocupações relacionadas à disponibilidade, hospedagem e escalonamento do serviço ou de autenticação e autorização dos utilizadores fora da equação, uma vez que, todas estas questões são de responsabilidade da *Atlassian*, fornecedora do serviço [Build with forge, 2023]. Na Figura 6 é possível observar a diferença entre as arquiteturas de ambas as *frameworks* mencionadas até ao momento. Tal como acontece com a utilização de *Atlassian Connect* e a respetiva criação do *app descriptor*, em *Forge*, é necessária a criação de um ficheiro - *Manifest* [Manifest, 2023] - que descreve todas as informações da aplicação, desde os módulos que são integrados na ou nas instâncias e as permissões necessárias para o correto funcionamento da mesma, assim como, qualquer outra informação complementar.

Resumindo, a utilização de *Forge* permite aos programadores criarem aplicações para produtos *Atlassian* sem a necessidade de construção de toda a infraestrutura de suporte para a manutenção das respetivas aplicações. No entanto, por ser ainda um serviço em desenvolvimento, existem algumas restrições nas possíveis *features* a implementar quando comparado com a *framework Atlassian Connect*.

Tendo em conta as desvantagens e vantagens de cada *framework* apresentada, como também as

[3] Logótipos retirados de https://atlassian.design/resources/logo-library/

atuais recomendações da *Atlassian* decidiu-se que, para a implementação da aplicação, será adotado *Forge* como a *framework* de desenvolvimento.

3.4.2 Integração com o Xray, JIRA e Confluence

Por ser uma aplicação para o *JIRA*, é necessário estabelecer uma estratégia para a extração de todos os dados necessários à implementação dos relatórios gerados pelo *Xray* nas páginas do *Confluence*, nomeadamente, informações sobre projetos, *issues*, versões e ambientes de testes da instância *JIRA* na qual o *Xray* se encontra instalado. Para que tal interação aconteça será utilizada a *REST API* [Rest api v2, 2023].

Para permitir a integração de diferentes aplicações consigo, o *Xray* disponibiliza aos programadores duas formas de acesso aos seus dados na forma de *APIs*: *REST API* e *GraphQL API*. Todos os pedidos realizados para ambas as *APIs* devem ser autorizados tendo em conta as credenciais de um utilizador associado a uma instância *JIRA*, na qual o *Xray* se encontra instalado. Estas credenciais podem ser encontradas na respetiva área de administração da aplicação.

Uma vez que, apenas a *REST API* permite a geração do *token* de autorização, esta será a primeira *API* a ser utilizada. De seguida, por ser mais completa, quando comparada com a *REST API*, no que diz respeito à obtenção de inúmeras informações, relacionadas com os testes de software e respetivo elementos criados pelo *Xray* assim como as suas configurações, será utilizada a *GraphQL API*. A partir dela são obtidos todos os dados necessários para a geração dos relatórios. Em *GraphQL* um utilizador solicita exatamente os dados e informações de que necessita, sem excessos ou escassez, o que é uma das desvantagens de utilização de uma *REST API* para a obtenção de dados.

3.4.3 Métricas para Avaliação da Solução

Finalizado o processo de desenvolvimento, com o objetivo de avaliar o produto desenvolvido e que todos os objetivos propostos foram alcançados com êxito, pretende-se responder às seguintes questões:

1. *A solução idealizada foi alvo de alterações? Em caso positivo, quais foram e qual o impacto de cada uma?*
2. *Todos os relatórios e respetivos requisitos foram implementados?*
3. *Os relatórios permitem a configuração de diferentes análises estratégicas ou filtros sendo o resultado obtido o esperado?*

4. *Os relatórios implementados são semelhantes aos originais?*

5. *A aplicação encontra-se pronta para a sua disponibilização no marketplace da Atlassian?*

Parte II

Capítulo 4

Organização e Estrutura do Projeto

No presente capítulo é apresentada a equipa de desenvolvimento encarregue da implementação da solução idealizada e são descritos os processos utilizados pela mesma para a gestão do projeto. Também são enunciadas as ferramentas e tecnologias utilizadas para a implementação da solução e mencionados os motivos que levaram à adoção das respetivas ferramentas.

4.1 Metodologia e Planeamento de Trabalho

Nesta secção, é feita em primeiro lugar, a apresentação da equipa de desenvolvimento assim como as rotinas de trabalho utilizadas pela mesma durante a realização da dissertação de mestrado. De seguida é mencionado o processo utilizado para a gestão do projeto, contemplando todas as vertentes do mesmo, e apresentada a estrutura final da solução implementada.

4.1.1 Equipa

Além do autor da dissertação de mestrado e único membro da equipa de desenvolvimento, a equipa é ainda composta por um supervisor e dois engenheiros de *software*. O autor reporta ao supervisor todos os avanços do projeto e os motivos para a tomada de determinadas decisões, podendo estas, em caso de discórdia do supervisor, serem rejeitadas ou alteradas para que o resultado final do produto a implementar não seja comprometido. Para além destas funções, o supervisor também estava encarregue do processo de testes às funcionalidades desenvolvidas pelo programador, efetuando testes manuais para a validação e verificação dos critérios de aceitação definidos.

Os restantes engenheiros de *software*, são responsáveis pela revisão de todo o código produzido, podendo sugerir alternativas e, se necessário, correções à estratégia abordada pelo autor. Graças à vasta experiência já obtida pelos presentes engenheiros no que toca ao desenvolvimento de aplicações para o ecossistema *Atlassian*, estes aconselharam e guiaram o programador nas boas práticas a seguir para que

a aplicação produzida fosse ao encontro aos padrões de qualidade da *Atlassian*.

Uma vez por semana, a equipa reunia-se numa videoconferência onde o autor expunha todo o trabalho produzido durante a semana e eram esclarecidas quaisquer dúvidas ou questões ao trabalho realizado ou por realizar. Para além desta reunião, foi estabelecido um canal de *chat* onde a equipa podia, de uma forma rápida e informal, comunicar entre si.

4.1.2 Gestão do Projeto

Para a gestão de todo o processo de trabalho e tarefas realizadas durante a realização da presente dissertação de mestrado, foi criado um projeto no *JIRA*, tendo por base a metodologia *Kanban* onde as tarefas a desenvolver são representados num quadro com diferentes colunas, correspondentes ao estado de desenvolvimento e progresso da tarefa em questão. No projeto em questão, foram definidos os seguintes estados possíveis:

- **To Do**: a tarefa ainda não foi iniciada;
- **In Progress**: a tarefa encontra-se em desenvolvimento e ainda não foi concluída;
- **Code Review**: a tarefa foi concluída mas o código produzido ainda não foi revisto e aceite;
- **Waiting for Deploy**: a tarefa foi concluída mas o ambiente de testes ainda não possui as novas funcionalidades produzidas;
- **Waiting for Testing**: as funcionalidades produzidas já se encontram refletidas no ambiente de testes mas o processo de teste às mesmas ainda não foi iniciado;
- **Testing**: as funcionalidades produzidas encontram-se a ser testadas pelo supervisor;
- **Done**: as funcionalidades foram testadas e aprovadas terminando o ciclo da tarefa.

Para a gestão de todo o código produzido e controlo de versões, foi criado um repositório no *Bitbucket*. Desta forma é possível gerir todo o histórico de alterações ao código, bem como, criar e gerir diferentes *branches* seguindo um modelo semelhante ao expresso na Figura 7.

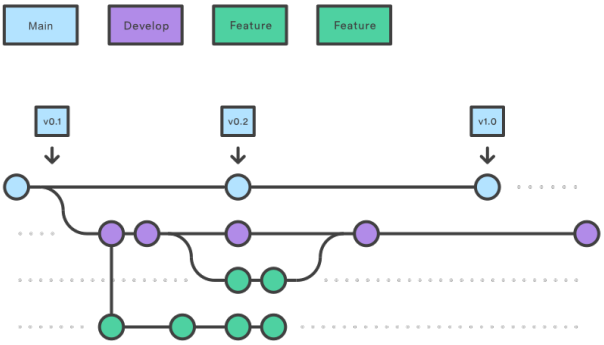


Figura 7: Bitbucket *workflow*[1]

Apesar de não ser estritamente necessário neste caso, uma vez que a equipa de desenvolvimento apenas é constituída por um único elemento que apenas trabalha numa tarefa a cada instante, são seguidas as boas práticas consideradas no mundo do desenvolvimento de *software* e é criado um novo *branch* para cada tarefa a desenvolver. Por sua vez, graças à implementação de um serviço de **Continuous Integration** (**CI**) e **Continuous Delivery** (**CD**) no *Bitbucket Cloud*, conhecido como *Bitbucket Pipelines*, sempre que o código produzido para uma determinada tarefa é aceite pelos revisores e acoplado com o código disponível no *branch* de desenvolvimento, um novo processo automático é criado e iniciado de forma a compilar e realizar o *deployment* do respetivo código numa instância *JIRA* que será utilizada para a realização dos testes de *software*.

[1] Imagem retirada de https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow

4.1.3 Estrutura do Projeto

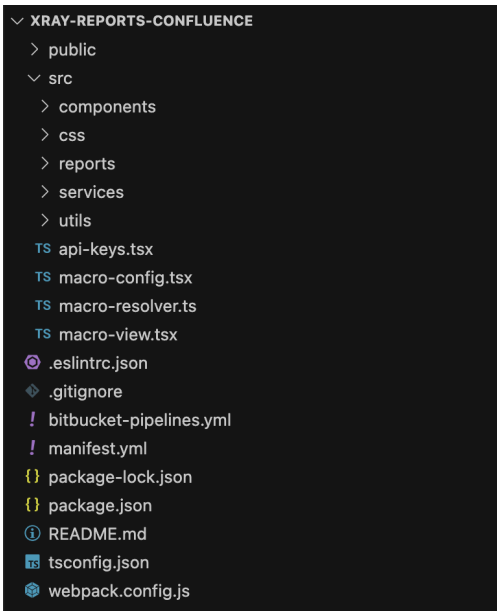


Figura 8: Estrutura do Projeto

A Figura 8 ilustra a estrutura resultante para todo o código produzido ao longo do desenvolvimento da aplicação, dos quais se destacam os seguintes ficheiros e diretorias:

- **public/**: diretoria com o código compilado e fornecido ao cliente;
- **src/api-keys.tsx**: ficheiro definido como ponto de partida para a construção da página de administração;
- **src/macro-config.tsx**: ficheiro definido como ponto de partida para a construção do menu de configuração da Confluence *macro*;
- **src/macro-view.tsx**: ficheiro definido como ponto de partida para a construção do conteúdo da *macro* e respetivo relatório;
- **src/macro-resolver.ts**: ficheiro com o código de cada um dos *resolvers* utilizados para a obtenção de dados na geração dos relatórios;
- **src/components/**: diretoria com componentes utilizados por diferentes módulos da aplicação;
- **src/css/**: diretoria com ficheiros de estilo a aplicar na apresentação;
- **src/reports/**: diretoria com ficheiros de código para cada um dos relatórios implementados;

- **src/services/**: diretoria com ficheiros de código que controlam todos os pedidos feitos às diferentes **API**s utilizadas;
- **src/utils/**: diretoria com ficheiros utilitários para controlo de variados aspetos da aplicação;
- **manifest.yml**: Ficheiro com a descrição da aplicação *Forge* desenvolvida e respetivos módulos e permissões;
- **bitbucket-pipelines.yml**: ficheiro com a descrição do processo de **CI**/**CD**
- **package.json**: Ficheiro com toda a informação da aplicação e respetivas dependências.

4.2 Ferramentas e Tecnologias

Nesta secção são enunciadas e descritas as tecnologias adotadas para o desenvolvimento da aplicação. Uma vez que é utilizado *Forge* como a *framework* de desenvolvimento de aplicações para o ecossistema Atlassian o espetro de tecnologias disponíveis é altamente influenciado por tal decisão.

4.2.1 Linguagem de Programação

TypeScript é uma linguagem de programação *strongly typed*, ou seja, os diferentes tipos de dados e valores devem obrigatoriamente ser declarados e as interações entre diferentes tipos são restritas, permitindo a captura de erros em momentos mais antecipados quando comparados a uma linguagem *weakly typed*.

No momento de execução, o código *TypeScript* é convertido para *JavaScript* sendo executado em qualquer ambiente capaz de o interpretar. Uma vez que as aplicações *Forge* são executadas num ambiente *Node.js*, a adoção de *TypeScript* como principal linguagem de programação torna-se evidente tendo em conta as características mencionadas.

4.2.2 Atlassian Design System

De forma a estandardizar as diferentes interfaces gráficas de cada uma das aplicações desenvolvidas para os seus produtos, a *Atlassian* criou o *Atlassian Design System* onde todos os padrões de *design* e comportamento a seguir são descritos para que a integração de novos componentes seja o mais transparente possível.

Para que estes padrões sejam alcançados na aplicação a desenvolver, com um grau de sucesso satisfatório, serão utilizados componentes das seguintes bibliotecas criadas pela *Atlassian*: *Atlaskit* e *UI Kit*. Estas bibliotecas disponibilizam inúmeros componentes em conformidade com o *Atlassian Design*

System, sendo o *Atlaskit* a biblioteca com maior detalhe e profundidade de componentes. Por outro lado, o *UI Kit* disponibiliza componentes para utilização aquando a implementação de aplicações *Forge* e, determinados módulos *Forge* e as respetivas configurações podem restringir exclusivamente a utilização de componentes do *UI Kit*.

4.2.3 React

React é uma *framework JavaScript* de desenvolvimento de interfaces gráficas, criada pelo Facebook em 2011. Em *React* são utilizados componentes, com estados e lógicas próprias, para a construção das respetivas páginas e baseia-se no conceito de *Virtual DOM* para a atualização das mesmas, onde é mantida em memória uma representação da verdadeira *DOM* e, sempre que é efetuada uma alteração é criada uma nova *Virtual DOM* e comparada com a anterior e apenas as diferenças resultantes são refletidas na *DOM* real.

Por ser uma *framework* compatível com *TypeScript* e com as bibliotecas de componentes utilizadas para a construção da aplicação, *React* torna-se a escolha predileta.

4.2.4 Construção de Relatórios

Tendo em conta, as tecnologias descritas até ao momento e as características dos gráficos utilizados para a representação de alguns relatórios gerados pelo *Xray*, maioritariamente representados por gráficos de barras, é necessário que a biblioteca escolhida para a construção de gráficos seja compatível com *TypeScript* e *React* e permita a configuração e customização de gráficos com diferentes níveis de interação.

Desta forma, foi escolhida a biblioteca *react-chartjs-2* que permite a criação de componentes *React* reutilizáveis que, fazendo uso da biblioteca *Chart.js*, permitem a geração de diferentes tipos de gráficos: barras, linhas, radar, área, dispersão, entre outros. A extensa documentação e exemplos fornecidos pela biblioteca *react-chartjs-2* também tornam a sua utilização viável e sustentada, quando comparada a outras bibliotecas para o mesmo fim.

Capítulo 5

Requisitos

Neste capítulo, são definidos os diferentes tipos de requisitos da aplicação, funcionais e não funcionais, e o processo para elaboração dos mesmos, assim como, enunciados os diferentes casos de utilização da aplicação por parte dos seus atores.

5.1 Contexto

Nesta secção são apresentados os diferentes atores da aplicação e os diferentes cenários de utilização para cada um deles.

No contexto da aplicação a desenvolver, existem por base dois atores:

- **Administrador**: o administrador corresponde à pessoa com permissões de administração do sistema na instância Confluence na qual a aplicação é instalada.
- **Utilizador**: o utilizador corresponde a qualquer pessoa com permissões de visualização e/ou edição de documentos na instância Confluence.

De forma a facilitar a identificação dos requisitos, foram previamente definidos diferentes cenários de utilização da aplicação, também designados como *User Stories* e os quais se encontram explícitos na Tabela 5.

Id	Sumário
US-1	Como administrador, tenho um ecrã onde posso configurar uma *API Key*
US-2	Como utilizador, posso escolher diferentes opções de configuração para um relatório
US-3	Como utilizador, posso criar um relatório do tipo *Test Runs List*
US-4	Como utilizador, posso criar um relatório do tipo *Overall Coverage*
US-5	Como utilizador, posso criar um relatório do tipo *Traceability*
US-6	Como utilizador, posso criar um relatório do tipo *Tests List*

Tabela 5: Lista de user stories

Através da Figura 9 é possível identificar as diferentes funcionalidades a que o utilizador tem acesso. Todas estas funcionalidades encontram-se associadas com a criação e, consequentemente, a configuração em páginas do *Confluence* dos relatórios que, por sua vez, necessitam de dados provenientes de diferentes fontes: *Xray API* e *JIRA API*. Estes dados são obtidos em tempo real de forma a que sejam apresentados relatórios com a informação mais recente, tanto ao nível do *JIRA* como do *Xray*.

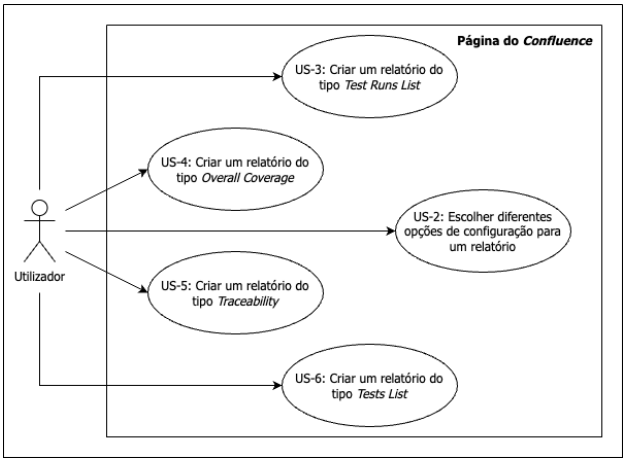


Figura 9: Diagrama de casos de uso para o ator *Utilizador*

O administrador tem acesso a diferentes cenários no processo de configuração de credenciais [Figura 10], nomeadamente a visualização, remoção, criação ou edição das mesmas. De tal modo é necessário a integração com a *Storage API*, para o armazenamento e consulta dos dados, e a *Xray API* para a validação da nova credencial armazenada no sistema.

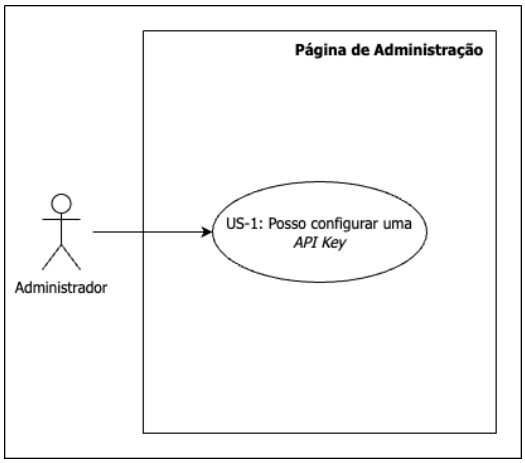


Figura 10: Diagrama de casos de uso para o ator *Administrador*

5.2 Requisitos Funcionais

Tal como referenciado na Secção 2.1, o processo de levantamento de requisitos enquadra-se nas primeiras fases do ciclo de desenvolvimento de *software* e a sua importância para o sucesso do mesmo é igualmente considerada na construção da aplicação. Os requisitos funcionais descrevem características ou funcionalidades que deverão ser disponibilizadas aos utilizadores de um sistema e, de modo oposto, não devem descrever qualquer aspeto relacionado com a implementação dos mesmos [Fernandes and Machado, 2016].

Conforme enunciado anteriormente, previamente à identificação dos requisitos funcionais foram criadas *User Stories* que posteriormente foram refinadas e detalhadas em requisitos. Este processo teve por base a análise da descrição de cada uma das *User Stories* e a sua discussão em várias reuniões com a equipa de forma a garantir que todos os aspetos a considerar no desenvolvimento da aplicação seriam corretamente identificados.

O resultado final do levantamento de requisitos pode ser consultado na Tabela 6, na qual os requisitos são identificados, particularizados e associados às respetivas *User Stories*.

Id	Requisito	User Stories
RF-1	O administrador pode consultar a configuração guardada	US-1
RF-2	O administrador pode adicionar uma nova configuração	US-1
RF-3	O administrador pode editar a configuração guardada no sistema	US-1
RF-4	O administrador pode remover a configuração do sistema	US-1
RF-5	O utilizador pode escolher qual o tipo de relatório a gerar (*Tests List, Test Runs List, Overall Coverage, Traceability*)	US-2, US-3, US-4, US-5, US-6
RF-6	O utilizador pode escolher um projeto	US-2, US-3, US-4, US-5, US-6
RF-7	O utilizador pode escolher um filtro	US-2, US-3, US-4, US-5, US-6
RF-8	O utilizador pode escolher o tipo de análise estratégica (Latest, Version, Test Plan)	US-2, US-4, US-5, US-6
RF-9	O utilizador pode escolher a versão alvo para análise	US-2, US-4, US-5, US-6
RF-10	O utilizador pode escolher o *Test Plan* alvo para análise	US-2, US-4, US-5, US-6
RF-11	O utilizador pode escolher o ambiente de teste alvo para análise	US-2, US-4, US-5, US-6
RF-12	O utilizador pode escolher a opção *Final Statuses Precedence*	US-2, US-4, US-5, US-6
RF-13	O utilizador pode escolher quais as colunas exibidas no relatório no momento de criação	US-2, US-3, US-6
RF-14	O utilizador pode escolher quais as colunas exibidas no relatório no momento de visualização	US-3, US-6
RF-15	O utilizador pode filtrar as execuções de teste por *Test Plan*	US-2, US-3
RF-16	O utilizador pode filtrar as execuções de teste por versão	US-2, US-3
RF-17	O utilizador pode filtrar as execuções de teste por ambiente	US-2, US-3
RF-18	O utilizador pode filtrar os testes por componente	US-2, US-3
RF-19	O utilizador pode filtrar os testes por prioridade	US-2, US-3
RF-20	O utilizador pode filtrar as *test runs* por *assignee*	US-2, US-3
RF-21	O utilizador pode filtrar as *test runs* por *executed by*	US-2, US-3
RF-22	O utilizador pode filtrar as *test runs* por estado	US-2, US-3
RF-23	O utilizador pode agrupar *coverable issues*	US-2, US-4
RF-24	O utilizador pode alterar o modo de apresentação do relatório (*Flat, Hierarchical*)	US-2, US-4
RF-25	O utilizador pode exibir/esconder a coluna *Test Runs*	US-2, US-5
RF-26	O utilizador pode filtrar as *coverable issues* pelo seu estado	US-5

Tabela 6: Lista de requisitos

5.3 Requisitos Não Funcionais

Os requisitos não funcionais correspondem ao conjunto de especificações ou restrições do sistema, sem que a essência do mesmo e as suas funcionalidades sejam diretamente relacionadas [Fernandes and Machado, 2016]. Dependendo do autor considerado, existem várias propostas para a classificação dos mesmos, como a proposta de Robertson e Robertson [2016], na qual são definidos oito tipos de requisitos não funcionais, ou a de Sommerville [2016] na qual os mesmos são divididos em três categorias.

No contexto do presente problema, ambas propostas foram consideradas sem qualquer tipo de exclusividade e foram levantados requisitos não funcionais para Segurança, Resiliência e Aparência do sistema a construir.

5.3.1 Segurança

RNF-1 As credenciais configuradas na página de Administração devem ser armazenadas de forma encriptada.

RNF-2 As credenciais configuradas na página de Administração devem unicamente ser consultadas por utilizadores da instância *Confluence* na qual a aplicação se encontra instalada.

5.3.2 Resiliência

RNF-3 As *Confluence macros* criadas para a geração de relatórios devem ser independentes entre si.

RNF-4 Um erro gerado numa das *Confluence macros* não deve afetar o comportamento das restantes.

5.3.3 Aparência

RNF-5 A interface gráfica da aplicação deve seguir os padrões de *design* da *Atlassian*.

RNF-6 Os relatórios devem manter-se semelhantes aos originais.

Capítulo 6

Desenvolvimento da Aplicação

No presente capítulo, são fundamentadas as decisões tomadas pela equipa de desenvolvimento de forma a apresentar resposta e contornar os obstáculos levantados pela proposta de solução idealizada, assim como explicado todo o processo de geração das diferentes páginas e relatórios da aplicação.

6.1 Decisões Técnicas

Ao longo desta secção, são descritas em detalhe as várias decisões técnicas tomadas durante o processo de desenvolvimento, sendo que, o motivo pelo qual foi necessário para algumas das escolhas deveu-se a limitações e restrições do *Forge*, *framework* adotada para o desenvolvimento da aplicação.

6.1.1 Armazenamento

Para que seja possível comunicar, tanto com a *REST API* como com a *GRAPHQL API* do *Xray*, com a finalidade de obter dados relacionados com os testes de *software* por ele monitorizados é necessário que os pedidos sejam autenticados utilizando as credenciais de um utilizador do *Xray*. Estas credenciais deverão ser utilizadas para a obtenção de um *token* de autenticação, com um prazo de validade de 24 horas, que poderá ser utilizado em pedidos consequentes.

Tendo em conta as necessidades da aplicação a desenvolver, a solução adotada para o armazenamento das respetivas credenciais na aplicação desenvolvida foi a *Storage API*[Storage API, 2023] fornecida pelo *Forge*. A *Storage API* possui vários métodos que permitem o armazenamento e consulta de dados no formato **JSON** por parte das aplicações *Forge*, encriptação e associação dos respetivos dados à instância na qual a aplicação é instalada e impedimento de partilha dos mesmos com outras aplicações ou diferentes instalações. Para a resolução de conflitos de escrita é utilizada a estratégia *last write wins* [1] e é de igual modo gerido e de responsabilidade da *Atlassian*. Apesar de existirem certas cotas e limites associados

[1] O resultado final a considerar de duas ou mais operações concorrentes é o resultado da operação efetuada em último lugar.

à utilização da *API* em questão, impedindo a sua utilização em situações que seja necessário lidar com vastos volumes de dados, neste caso não existe qualquer constrangimento na utilização da mesma, uma vez que, por cada instalação da aplicação apenas são armazenadas as credenciais de um único utilizador.

Desta forma, o objeto guardado por cada instância para que todo o processo de autenticação e autorização no acesso aos dados dos testes de *software* armazenados pelo *Xray* seja possível, possui o seguinte formato:

- **id**: identificador público (*client_id*[2]) do utilizador a personificar que, preferencialmente, deverá ser criado no momento de instalação da aplicação para identificação da mesma;
- **secret**: identificador secreto (*client_secret*[2]) do mesmo utilizador acima descrito;
- **key**: *token* gerado, por parte do *Xray*, para autenticação e autorização do utilizador, com um prazo de utilização de 24 horas;
- **lastUpdate**: horário da última atualização do token, sendo que sempre que a diferença entre este horário e o do momento em que é realizado um pedido for superior ao período de validade do token o mesmo é reatualizado.

A utilização de um sistema de base de dados convencional também poderia ser considerada, no entanto, no contexto da aplicação a desenvolver iria ser criada uma camada de complexidade adicional ao processo desenvolvimento e sem ganhos de *performance* esperados. Caso fosse esta a estratégia adotada, a aplicação teria de estabelecer uma ligação e comunicar com um servidor externo ao ambiente *Atlassian* que, por sua vez, iria extrair da base de dados criada para o efeito e, posteriormente, retornar esses mesmos dados à aplicação. Consequentemente, todos os problemas associados à construção de um servidor, como a hospedagem do mesmo ou a autorização/autenticação de pedidos recebidos, e de um sistema de base de dados, como a encriptação dos dados ou a resolução de conflitos de escrita, seriam de total responsabilidade da equipa de desenvolvimento.

Quando comparadas as vantagens e desvantagens de ambas as estratégias disponíveis, no contexto da presente dissertação, conclui-se que a adoção e construção de um sistema de base de dados não se torna viável quando comparado à utilização da *Storage API*.

[2] Tanto o valor de *client_id* como o valor de *client_secret* podem ser obtidos na página de administração *API Keys* na instância *JIRA* em que o *Xray* se encontra instalado.

6.1.2 Limitação de Acessos a APIs

O *Xray*, como já têm sido referenciado ao longo do documento, disponibiliza o acesso aos seus dados através de duas *APIs*, a *REST API* e, com maior relevância no contexto do problema a solucionar, a *GraphQL API*.

Segundo a própria documentação da *GraphQL API* do *Xray* [GraphQL API, 2023]. são utilizados vários tipos de limites nos pedidos realizados, de forma a proteger os servidores do *Xray* contra pedidos excessivos e/ou abusivos:

- o número de elementos retornados em cada pedido deverá estar entre 1 e 100, identificado pelo argumento *limit*;
- o número total de itens retornados e que compõem cada um dos elementos não pode exceder os 10000 itens;
- cada pedido não pode utilizar mais de 25 *resolvers*[3].

Assim sendo, e, caso não seja possível obter a informação desejada num único pedido, é necessário a criação e gestão de múltiplos pedidos para o mesmo efeito. Para tal, e recorrendo às funcionalidades disponíveis em *JavaScript*, irão ser utilizadas *promises* para a criação e resolução de cada um dos pedidos realizados e que, caso os pedidos não possuam nenhum tipo de dependência entre si, também permitem a paralelização dos mesmos, gerando um ganho substantivo no tempo de resposta necessário para a obtenção dos dados. Uma *promise*, representa um valor associado à conclusão de um evento assíncrono, ou seja, um valor que no momento da sua requisição poderá não se encontrar disponível mas que, num determinado momento futuro, apresentará um valor real associado à conclusão ou falha da operação a ele associado.

No entanto, cada uma das *APIs* do *Xray* também limita o número de ligações que cada utilizador pode estabelecer por minuto, não podendo serem realizados mais de cem pedidos neste intervalo de tempo. Em casos que o volume de dados seja bastante significativo e a decomposição dos mesmos em pedidos ultrapasse o limite imposto pelo *Xray* surgem várias implicações no acesso à informação e nas funcionalidades disponíveis ao utilizador em cada um dos relatórios. Nestas situações o número de pedidos gerados pela aplicação é limitado às restrições do *Xray*, um aviso é retornado ao utilizador e, dependendo do tipo de relatório, algumas funcionalidades são desabilitadas.

[3] Em *GraphQL* um *resolver* representa uma função responsável por mapear campos do pedido em funções responsáveis pela obtenção desses mesmos valores requisitados.

Assim como acontece no *Xray*, a própria *REST API* do *JIRA* possui vários limites de acesso aos seus dados, nomeadamente, na quantidade de resultados retornados por pedido. Desta forma foi implementado um serviço de paginação para que nem os servidores do *JIRA* nem o tempo de resposta dos pedidos seja comprometido face ao grande volume de dados.

Dependendo das necessidades a satisfazer, e mesmo recorrendo à paralelização de múltiplos pedidos, a limitação no número de objetos retornados por pedido pode pressupor um *bottleneck* [4] significativo no acesso aos dados. De tal forma, é disponibilizado, na mesma *API*, o método *Evaluate Jira Expression* que, através da análise de uma linguagem especifica ao contexto do ambiente *JIRA*, pode retornar até um máximo de 1000 resultados por pedido, resultando num ganho de até 10 vezes quando comparado aos restantes métodos. No entanto, a variabilidade de dados disponíveis neste método é limitada às variáveis de contexto suportadas pela linguagem e bastante menor à variabilidade de dados e profundidade dos mesmo que é possível obter nos restantes métodos, o que restringe o seu uso a casos bastante específicos como, por exemplo, a obtenção de todos os *issue ids* de um determinado projeto.

6.1.3 Configuração da Confluence Macro

Para a implementação do menu de configuração, no momento de desenvolvimento da presente dissertação, em Forge os programadores encontram-se limitados à utilização de um único componente - *MacroConfig* [MacroConfig, 2023] - da biblioteca *UI Kit* e, ele mesmo, apresenta limitações quanto ao tipo possível dos seus componentes filho que apenas poderão ser campos de imputação de valores da própria *UI Kit*.

Este componente permite a criação de um menu lateral [Figura 11], visível no lado direito da página de edição, com todas as opções de configuração possíveis. Em consequência do comportamento das *Confluence macros* no momento de edição de uma página, sempre que uma das opções de configuração é alvo de uma alteração, a *macro* é recarregada e é possível pré-visualizar o resultado esperado na própria página de edição para a combinação de configuração mais recente, no entanto, não é possível interagir com a mesma.

A restrição de utilização do componente *MacroConfig* e respetivos componentes filhos, assim como a ausência de um método de submissão de alterações à configuração acarreta algumas dificuldades em transparecer ao utilizador determinadas informações sobre o processo de configuração, nomeadamente a presença de uma configuração incompleta ou incorreta. Deste modo, todas as informações relevantes às escolhas feitas pelo utilizador, são apresentadas no momento de pré-visualização, tirando partido desta

[4] Em termos de Engenharia de Software, um *bottleneck* representa uma limitação num sistema ou aplicação originada por um único componente.

funcionalidade.

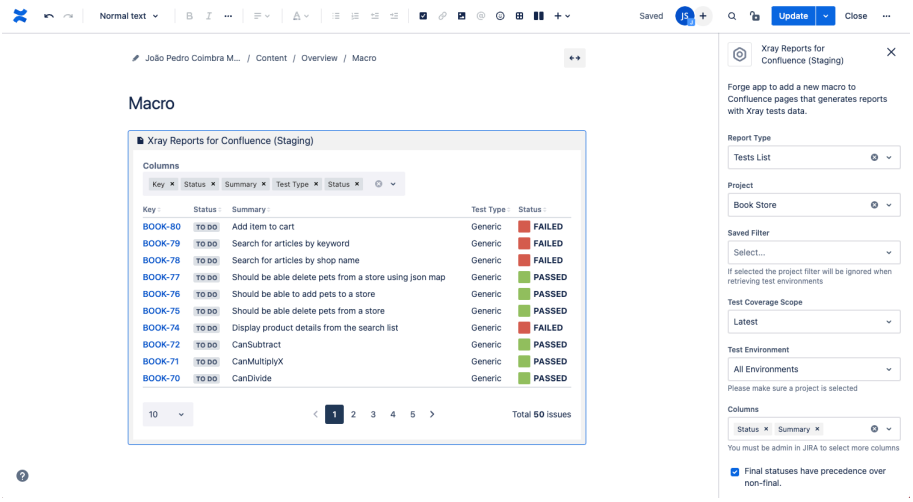


Figura 11: Exemplo do processo de configuração de uma *macro*

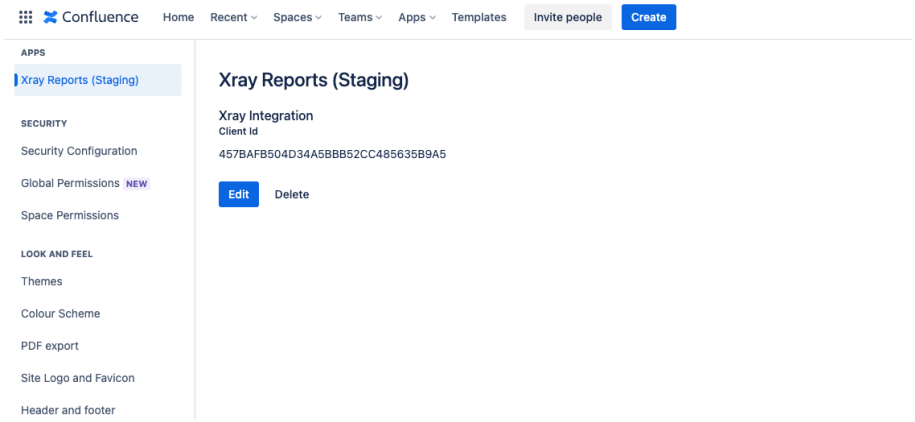
6.2 Implementação da Página de Administração

Nesta secção é abordado todo o processo de implementação da página de administração, assim como, todas as ferramentas utilizadas e decisões técnicas.

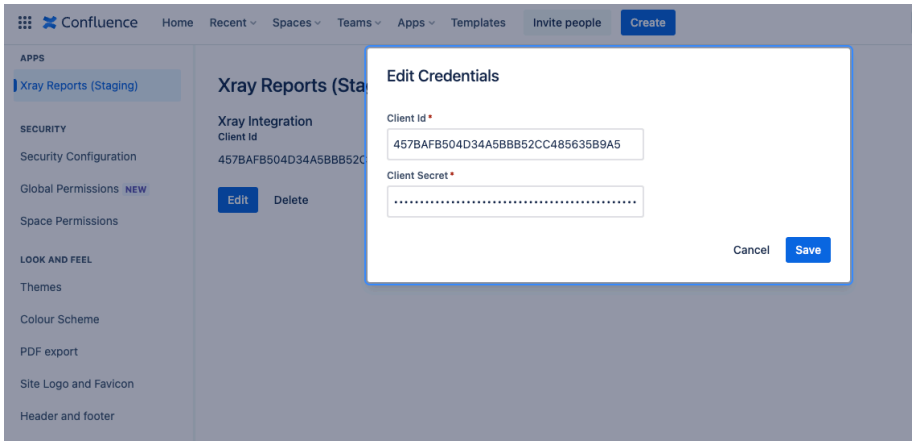
A página de administração (ou configuração), acessível através do menu de gestão de aplicações da instância *Confluence* onde se encontra instalada, permite a um administrador da instância armazenar as credenciais obtidas na secção *API Keys* do *Xray*. Tais credenciais são posteriormente utilizadas para a gestão do *token* de autenticação, necessário para que a aplicação possa comunicar e consultar dados, através das diferentes *APIs*, providenciadas pelo *Xray*. Por este motivo, a implementação desta *feature* torna-se fulcral para o desbloqueio do restante processo de desenvolvimento da aplicação, nomeadamente da implementação dos relatórios que necessitam de dados provenientes do *Xray* para que os resultados apresentados sejam precisos e consistentes quando comparados com os originais.

Para a implementação da página em questão foi utilizado o módulo *Global Settings*[Confluence Global Settings, 2023] juntamente com vários componentes do *UI kit*, nomeadamente componentes que permitam a construção de formulários e diálogos e cujo resultado final é visível na Figura 12. Na página de configuração o administrador tem a possibilidade de consultar as credencias já configuradas e, no caso de ausência, adicionar uma nova configuração. Posteriormente, esta configuração poderá ser atualizada ou removida pelo mesmo. As credenciais introduzidas pelo administrador são armazenadas recorrendo à *Storage API* e sempre que é criada ou atualizada uma configuração, o ficheiro **JSON** que contém todas

as informações necessárias é, respetivamente, gerado ou atualizado mediante a utilização dos diferentes métodos fornecidos pela *API* para o armazenamento de dados encriptados. No caso de remoção da configuração o documento é destruído.



(a) Ecrã Principal



(b) Ecrã de configuração da *API Key*

Figura 12: Implementação da página de administração

6.3 Implementação dos Relatórios

Na presente secção é descrito todo o processo de implementação de cada um dos relatórios e respetivos menus de configuração, sendo detalhado todo o processo de obtenção de dados necessários à construção dos relatórios e elucidadas todas as decisões tomadas durante o processo de desenvolvimento.

6.3.1 Tests List

Configuração

Qualquer um dos relatórios implementados necessita de ser previamente configurado para que o resultado final seja o mais adequado às necessidades e pretensões do utilizador. Por ser o primeiro implementado, a construção do menu de configuração do relatório *Tests List* foi realizada visando o desenvolvimento de componentes genéricos. Desta forma estes componentes podem ser reutilizados por diferentes relatórios e respetivas configurações com possibilidades semelhantes, nomeadamente, toda a lógica de definição da análise estratégica ou escolha dos projetos ou filtros alvo.

Na presente configuração, o utilizador pode selecionar qual o projeto ou filtro pretendido para visualização dos respetivos testes, alterar a análise estratégica e respetivas propriedades e editar quais as colunas apresentadas sempre que o relatório é carregado pela primeira vez. É importante notar que a seleção do projeto e filtro é exclusiva e caso ambos sejam selecionadas o valor final considerado é o do filtro.

Dependendo da dimensão alvo da análise estratégica, tanto os valores para as possíveis opções de escolha da *Fix Version* ou *Test Plan* são filtrados tendo em conta o projeto selecionado ou os projetos associados ao filtro escolhido.

Macro

Para a implementação do relatório *Tests List*, e construção da respetiva coluna, são necessários dados relativos aos testes, tanto das próprias *issues* como dos estados e tipos dos mesmos, assim como informação de cada um dos campos (*custom fields*) do *JIRA* de forma a construir de forma adequada cada uma das colunas em conformidade com cada um dos tipos de valores possíveis. Estes dados são carregados em *real-time* em diferentes ocasiões, recorrendo ao *resolver*[5] *GET_TESTS_RESOLVER* [Figura 13], criado para tal efeito: numa primeira fase, sempre que a *macro* é carregada; em fases posteriores de interação com a tabela, ou seja, sempre que é alterada a página ou o número de entradas nela exibidas, assim como, a alteração da sua ordenação.

[5] Na *framework Forge*, um *resolver* permite a definição de uma função responsável pelo tratamento de pedidos realizados por módulos de *Custom UI* dos quais, por sua vez, não é possível realizar determinado tipo de funcionalidades, como por exemplo, realizar pedidos externos à aplicação.

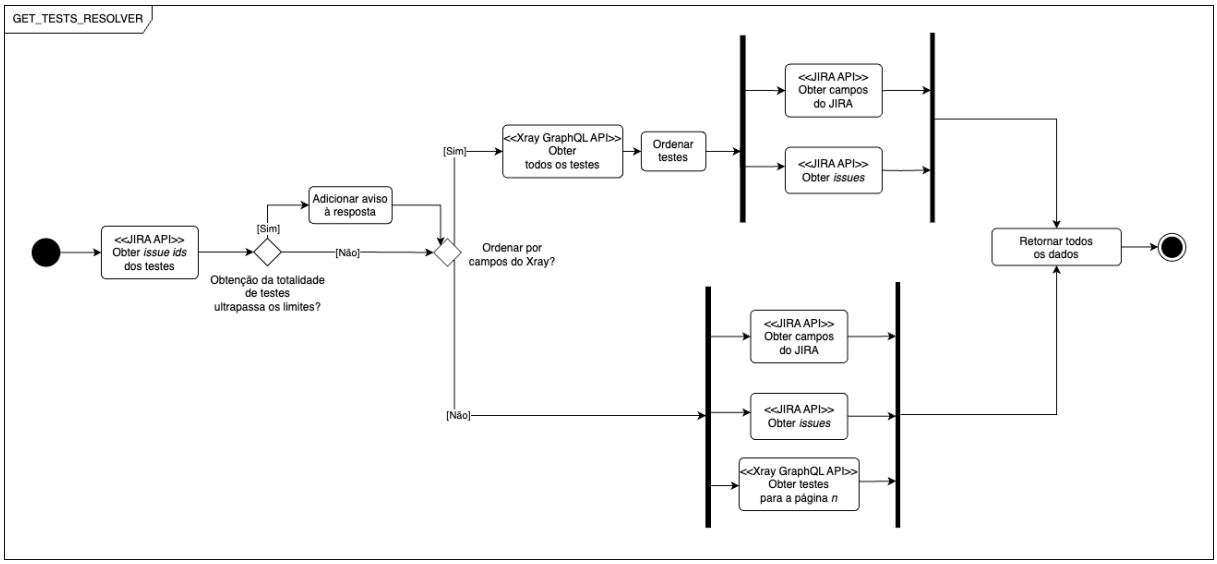


Figura 13: Fluxograma do processo de obtenção de dados para o relatório *Tests List*

O primeiro passo, para a obtenção da totalidade dos dados, passa pela obtenção de todos os *issue ids* dos testes, alvos da configuração escolhida, ou seja, os testes associados ao projeto ou *saved filter* selecionado. Para este passo, é utilizado o *endpoint JIRA Evaluate Expression* de forma a tirar partido do limite de dados que é possível obter por pedido, superior aos restantes *endpoints* disponíveis. Caso o número total de *issues* que respeitem a configuração ultrapasse as 5000 unidades algumas das funcionalidades do relatório são afetadas, nomeadamente o impedimento de ordenação da tabela por campos provenientes de dados do *Xray*, e devidamente sinalizadas ao utilizador.

O passo seguinte é dependente do tipo de ordenação selecionado. Caso a ordenação deva ser feita por campos do *Xray*, ou seja pelo tipo ou estado do teste, é necessário proceder à obtenção da totalidade dos testes e posteriormente à sua ordenação uma vez que não é possível obter os testes ordenados pelos respetivos campos recorrendo à *API* do *Xray*. Apenas de seguida é que são obtidas, em paralelo, todas as informações relativas a dados do *JIRA*, ou seja, as especificações de cada uma das *issues* e respetivos *custom fields*. Por sua vez, a *API* do *JIRA* permite a obtenção de *issues* de forma ordenada, podendo o processo de obtenção dos dados ser também paralelizado, representado um ganho no tempo de resposta total. Por fim todos os dados obtidos nos passos anteriormente descritos são retornados.

Para a construção da tabela final [Figura 14], de forma a manter a harmonia com o estilo utilizado nos produtos do sistema *Atlassian* e a similaridade com o próprio relatório gerado por parte do *Xray*, foi utilizado o componente para geração de tabelas presente na biblioteca *Atlaskit*. Sempre que o utilizador edita a combinação das colunas apresentadas no relatório a tabela é atualizada para que esta concordância seja perpetuada.

Columns

Key × Summary × Test Type × Status × Status ×

Key	Summary	Test Type	Status	Status
BOOK-38	Manual test of strong password validation	Manual	TO DO	FAILED
BOOK-37	Manual test of password reset procedure	Manual	TO DO	PASSED
BOOK-36	Favorites List User selects a book and clicks on the star	Generic	TO DO	TO DO
BOOK-35	Shopping Basket User selects an item and clicks on "express checkout"	Generic	TO DO	TO DO
BOOK-34	Test a visitor can filter the search result	Cucumber	TO DO	PASSED
BOOK-33	Test a visitor can do a valid search with multiple keywords	Cucumber	TO DO	PASSED
BOOK-32	Test a visitor can do a valid search with a single keyword	Cucumber	TO DO	PASSED
BOOK-31	Test a visitor can view all the books in his shopping basket	Cucumber	TO DO	PASSED
BOOK-30	Test visitors can remove books from their shopping basket	Cucumber	TO DO	PASSED
BOOK-29	Test visitors can add books to their shopping basket	Cucumber	TO DO	PASSED

10 ‹ 1 2 3 4 5 › Total **50** issues

Figura 14: Exemplo de implementação do relatório *Tests List*

6.3.2 Test Runs List

Configuração

Dado que a configuração do relatório *Test Runs List* permite a definição de diferentes filtros relativos a testes, *test runs* e execuções de teste e não permite a escolha de uma análise estratégica, é necessário a implementação dos filtros definidos na Secção 3.2. Alguns dos valores destes filtros são também influenciados pelo projeto ou filtro. Dado que os componentes para a escolha do projeto e filtro já se encontram implementados é tirado partido da sua reutilização para a conclusão do menu de configuração.

Macro

Para a geração do relatório *Test Runs List*, é necessário popular três grupos de colunas com dados de diferentes categorias, nomeadamente, testes, *Test Runs*, execuções de testes e defeitos. Estes dados são carregados pela aplicação, sempre que o relatório é renderizado ou o utilizador solicita a visualização de mais entradas na tabela, através da invocação do *resolver GET_TEST_RUNS_RESOLVER* [Figura 15].

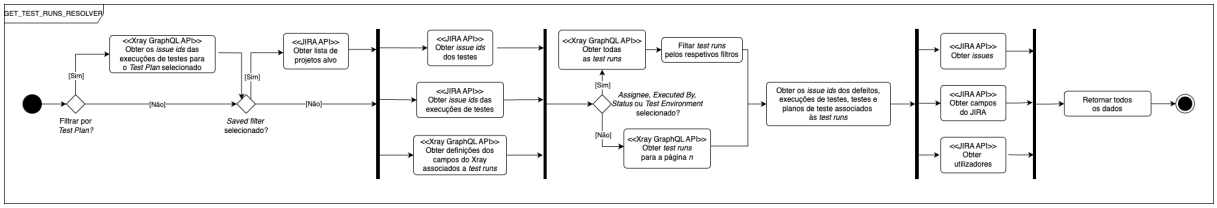


Figura 15: Fluxograma do processo de obtenção de dados para o relatório *Test Runs List*

Caso o relatório se encontre configurado com o filtro *Test Plan*, a primeira etapa passa pelo escrutínio

de quais são as execuções de teste associadas ao plano de testes pretendido. De seguida, em paralelo, são obtidos os *issue ids* dos testes e execuções de teste associadas ao projeto ou filtro selecionado, assim como, a configuração de *Test Runs* que cada destes projetos utiliza, para que seja possível conhecer quais os atributos adicionais que cada *Test Run* possui. Caso algum dos filtros para *Test Runs* ou ambiente de testes seja selecionado, então é necessário obter cada uma das *Test Runs* para que possam ser obtidos os dados para a página requisitada, depois de aplicados os respetivos filtros. Caso contrário esses mesmos dados podem ser diretamente obtidos usando a *API* do *Xray*, tirando partido do suporte de paginação fornecido pela mesma.

Depois de obtidas quais as *Test Runs* a retornar, são recolhidos todos os *issue ids* de testes, execuções de testes e defeitos a ela associados para que posteriormente sejam obtidos todos os dados de cada uma das *issues* mencionadas, enquanto que, em simultâneo, são obtidas informações relativas aos seus tipos de atributos e dos utilizadores da instância *JIRA* em causa.

Para a construção da tabela [Figura 16], ao contrário daquilo que foi feito na construção do relatório *Tests List*, não foi utilizado o componente *Dynamic Table* da biblioteca *Atlaskit*, uma vez que, a sua utilização implicaria mudanças no aspeto da tabela e que não é o pretendido na resolução do presente problema. O botão *SHOW MORE* é apresentado na parte inferior do relatório até que não existam mais resultados por apresentar na tabela e sempre que é clicado são carregadas, até um máximo de dez, novas entradas a acrescentar à tabela.

Figura 16: Exemplo de implementação do relatório *Test Runs List*

6.3.3 Test Coverage

Configuração

Assim como o *Tests List*, o relatório *Overall Coverage* permite a definição de uma análise estratégica, simplificando a construção do seu menu de configuração uma vez que ambos os componentes para implementação de tal funcionalidade e escolha do projeto ou filtro são reutilizados no atual relatório. Para além destes campos, também é possível configurar dois filtros adicionais:

- **Group By**: permite a definição de um campo pelo qual as *coverable issues* serão agrupadas por linhas correspondendo à propriedade em comum entre si.
- **View**: no *JIRA* é possível definir e criar vários tipos de relações entre *issues* sendo uma delas a relação pai-filho. Neste caso é possível escolher entre duas opções diferentes nas quais diferentes graus de abstração são considerados: *Flat* - ambos os níveis são considerados, ou seja, todas as *coverable issues* são apresentadas no gráfico independentemente do seu nível; *Hierarchical* - apenas dados de *issues* de nível superior (pai) serão apresentados no gráfico, no entanto, dados de *issues* de nível inferior ainda poderão ser consultados se visualizada a tabela de informação adicional.

Macro

Para a construção do relatório *Test Coverage* e, consequentemente, do gráfico de barras e tabela a ele associado, são necessárias informações sobre as *coverable issues* e respetivos estados de acordo com a configuração do relatório. Esta informação é unicamente obtida quando o relatório é carregado na página e o processo correspondente à obtenção desta mesma informação pode ser contemplado na Figura 17.

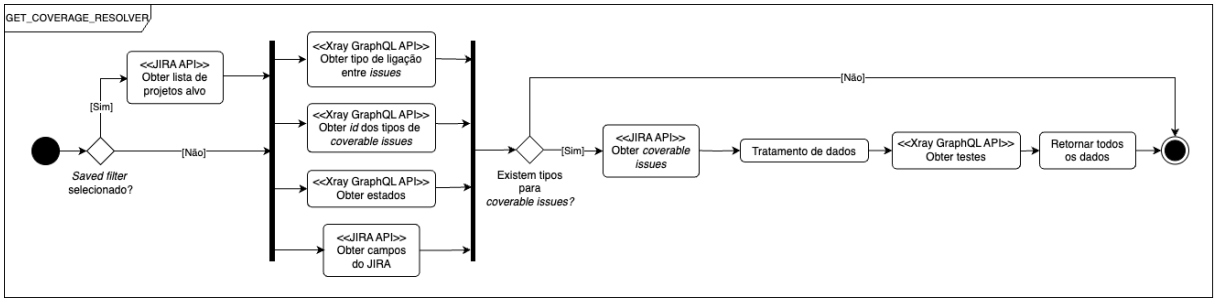


Figura 17: Fluxograma do processo de obtenção de dados para o relatório *Test Coverage*

O primeiro passo do *GET_COVERAGE_RESOLVER* centra-se na obtenção do projeto ou projetos alvo

da configuração selecionada. De seguida, são executados quatro passos em paralelo, independentes entre si:

- obtenção do tipo de ligação pai-filho entre *issues*;
- obtenção dos *ids* dos tipos de *coverable issues* configuradas nos projetos alvo;
- obtenção de informação relativa aos tipos de estados dos testes, para que possa ser feito o correto mapeamento de cada um dos estados das *coverable issues*;
- obtenção de dados das propriedades possíveis de uma *issue* para que o campo de configuração *Group By* possa ser corretamente calculado.

Caso existam tipos de *coverable issues* previamente configuradas para os projetos selecionados prossegue-se à obtenção das mesmas. De seguida, estes dados são alvo de um processamento com o objetivo de agrupar as *issues* segundo o seu grau de parentesco e no qual todos os *issue ids* dos testes a elas associadas são identificados. No passo seguinte, são obtidos os respetivos estados que permitem o cálculo do estado final de cada uma das *coverable issues* para a análise estratégica em causa.

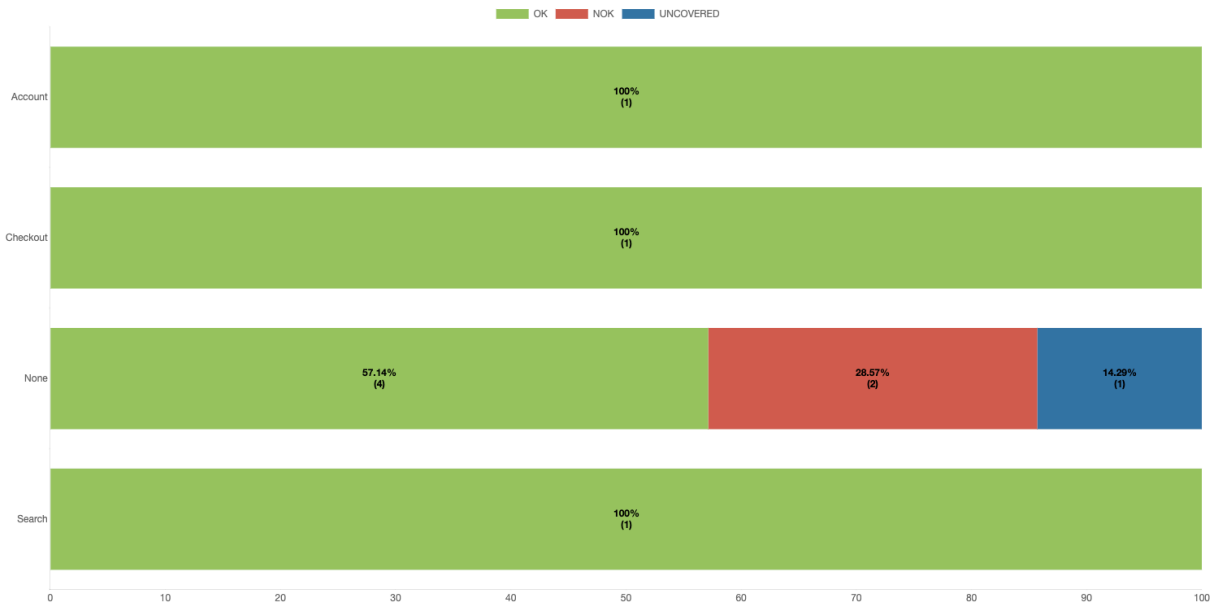


Figura 18: Exemplo de implementação do gráfico de barras para o relatório Overall Coverage

Para a construção do gráfico de barras [Figura 18], elemento primário do relatório *Overall Coverage*, foi utilizado a biblioteca *react-chartjs-2* por todos os motivos já referenciados na Secção 4.2. Cada barra presente no gráfico corresponde a um valor possível das *coverable issues* para o campo de configuração

Group By, sendo que como esse campo pode assumir múltiplos valores então a mesma *coverable issue* pode estar presente em mais do que uma barra do gráfico. Por sua vez cada barra encontra-se dividida em diferentes secções, correspondentes aos diferentes estados das *coverable issues*. Dado que uma *coverable issue* apenas pode assumir um único estado para uma determinada análise estratégica não é possível que a mesma *issue* se encontre presente em diferentes secções na mesma barra ou em barras distintas com estados distintos entre si.

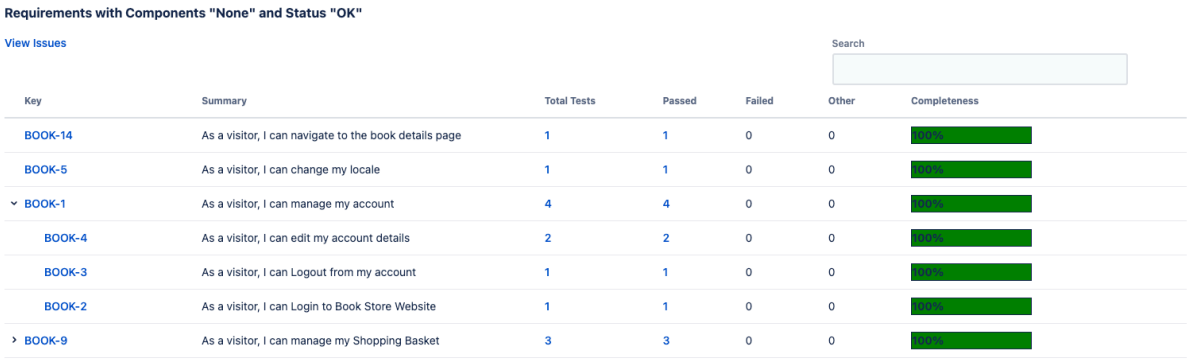


Figura 19: Exemplo de implementação da tabela de informação adicional no relatório *Overall Coverage*

Sempre que é efetuado um clique numa das secções de uma das barras do gráfico [Figura 19], uma tabela com informações adicionais sobre as *coverable issues* associadas à secção em questão é apresentada na parte inferior do relatório. Para além de identificar a *issue* pela apresentação da *issue key* e sumário também são apresentados a totalidade dos testes de *software* efetuados, divididos em diferentes colunas segundo o seu estado. Por fim, por cada linha também é apresentada uma barra de progresso com a percentagem do número de testes que respeitaram os critérios de validação estabelecidos.

Na construção da tabela com informações adicionais das *coverable issues* do relatório em causa foi utilizado o componente *Table Tree* da biblioteca *Atlaskit*. À semelhança do componente *Dynamic Table*, este componente permite a criação de tabelas com a particularidade que determinadas linhas podem ser expandidas ou colapsadas de forma a revelar novas linhas, apresentando ou escondendo informações correlacionadas com a linha principal. Esta funcionalidade facilita a perceção do utilizador quanto ao grau de parentesco de diferentes *issues*, uma vez que, uma linha principal que possa ser expandida representa uma *issue* pai e as *issues* propagadas representam as *issues* filhas.

6.3.4 Traceability

Configuração

Na configuração do relatório *Traceability*, para além da escolha do projeto ou filtro alvo assim como da análise estratégica pretendida, cujos componentes são reutilizados a partir dos processos de implementação anteriores, é possível definir se a coluna *Test Runs* se encontra visível. No entanto, o estado de visibilidade da coluna pode ser alterado no momento de visualização do relatório, para a satisfação das necessidades do utilizador e manter a consistência com o relatório original.

Macro

Para a implementação do relatório *Traceability*, assim como no relatório *Test Coverage*, são necessárias informações sobre as *coverable issues* para o projeto ou filtro alvo e respetivos estados, somando-se a estes as execuções de testes, *Test Runs* e defeitos associados. Seguindo a lógica dos restantes relatórios, estes dados são carregados sempre que a *macro* é também carregada, sendo, neste caso em concreto, invocado o *resolver GET_TRACEABILITY_RESOLVER* [Figura 20].

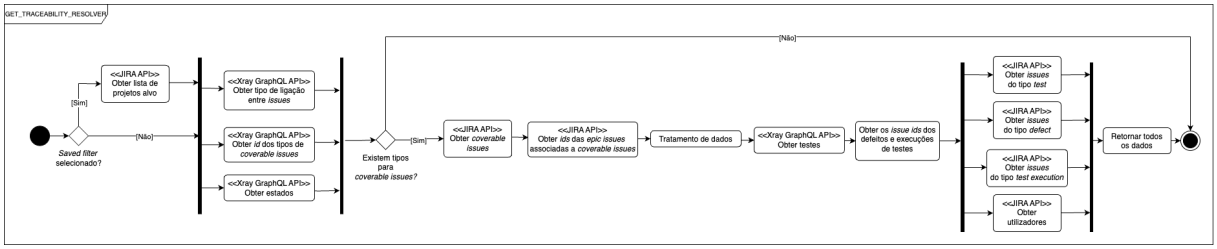


Figura 20: Fluxograma do processo de obtenção de dados para o relatório *Traceability*

Na primeira fase de obtenção de dados, até à obtenção das respetivas *coverable issues*, o processo é maioritariamente semelhante ao efetuado no relatório *Test Coverage* diferenciando-se apenas na obtenção dos campos do *JIRA* que para neste relatório não são necessários. De seguida, são identificados todos os *issue ids* das *epics* em causa para o relatório a gerar. O passo seguinte, relativo ao processamento dos dados é semelhante ao explicado no relatório anterior em que é realizado o agrupamento de *issues* com o grau de parentesco pai-filho e a obtenção dos *issue ids* de todos os testes a elas associados também são identificados. Os testes em causa são posteriormente obtidos através da *Xray API* e, a partir dos mesmos, são extraídos os *issue ids* dos defeitos e execuções de testes a eles associados para que possam ser obtidas as *issues* dos referentes tipos, em paralelo, com a obtenção dos utilizadores a elas afiliadas.

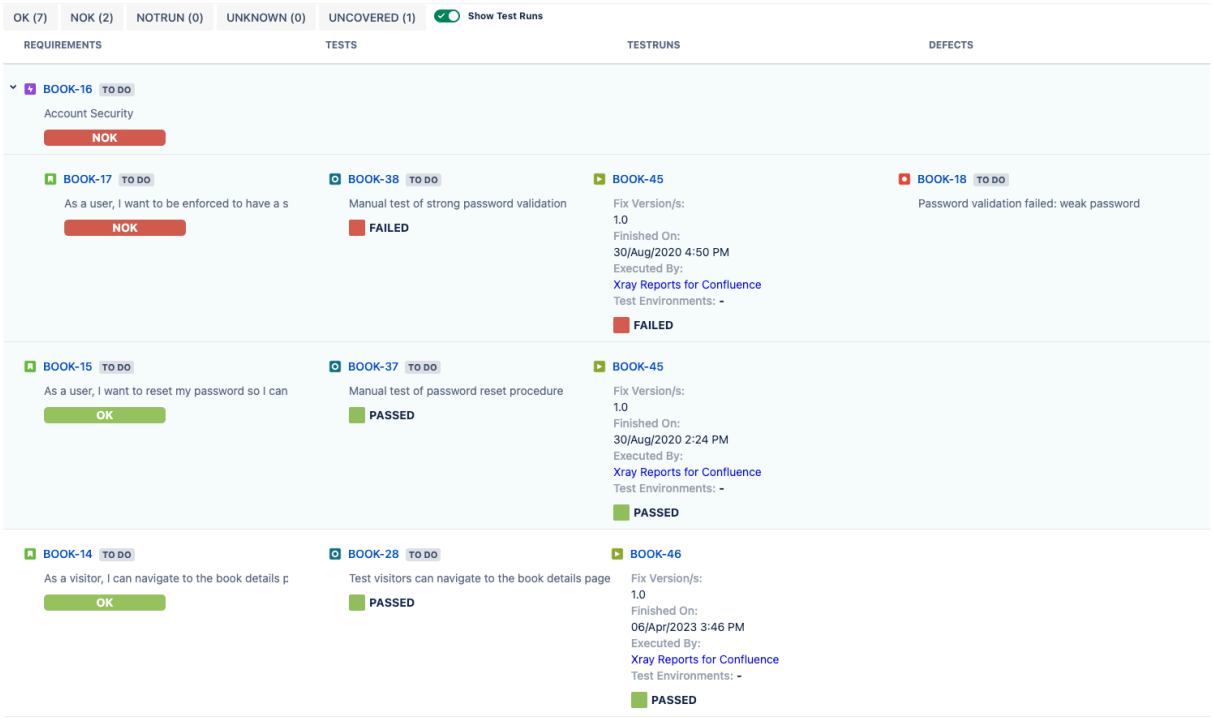


Figura 21: Exemplo de implementação do relatório *Traceability*

Para a construção da tabela referente ao presente relatório, visível na Figura 21, foi novamente utilizado o componente *Dynamic Table* para que seja possível a criação de linhas expansíveis, de forma a evidenciar quais as *coverable issues* com um grau de parentesco entre si. Cada linha da tabela encontra-se dividida em quatro colunas:

- **Requirements**: identifica a *coverable issue* e o respetivo estado para a análise estratégica definida;
- **Tests**: apresenta todos os testes associados à *coverable issue* e os respetivos estados;
- **Test Runs**: apresenta todas *Test Runs* associadas ao teste e *coverable issue* em questão e outras informações, como o estado e a *Test Execution* associada;
- **Defects**: identifica os defeitos encontrados durante a execução de um teste para a respetiva *coverable issue*.

No topo da tabela, para além da opção que permite esconder ou mostrar a coluna de *test runs*, existem cinco botões relativos a cada um dos estados possíveis de uma *coverable issue* que permitem filtrar as entradas da tabela pelos respetivos estados selecionados. Na ausência de qualquer filtro selecionado são apresentadas a totalidade de entradas carregadas até ao instante em questão.

Capítulo 7

Conclusões e Trabalho Futuro

Neste capítulo é realizada a retrospetiva e conclusão a todo o trabalho realizado, assim como, enunciados todos os aspetos e desafios enfrentados e as aprendizagens obtidas.

Por fim, é enunciado o trabalho idealizado e que poderá vir a ser implementado em fases futuras de forma a expandir as funcionalidades da aplicação desenvolvida, com o intuito de fortalecer e agregar valor à mesma.

7.1 Conclusões

Visando o estudo realizado aos de testes de *software*, na primeira parte do documento, é possível verificar a sua elevada importância no processo de desenvolvimento de um produto de *software* e o seu impacto no resultado final do produto, sendo a principal fase onde são encontrados e corrigidos diversos erros e garantida a disponibilização de todas as funcionalidades propostas.

Na segunda parte da presente dissertação de mestrado, foram exploradas e analisadas as diferentes possibilidades fornecidas pela *Atlassian* para desenvolvimento e integração de novas aplicações nos produtos do seu ecossistema. O estudo das diferentes estratégias de desenvolvimento permitiu escolher a ferramenta *Forge* como *framework* de desenvolvimento para a aplicação em questão.

Considerando as métricas para a avaliação da solução implementada definidas na Secção 3.4, pode-se deduzir que o trabalho desenvolvido cumpriu, no essencial, com os objetivos inicialmente traçados:

1. A proposta de solução idealizada, previamente à fase de desenvolvimento, apresentou uma base sólida e fundamentada tendo em conta os conhecimentos obtidos resultantes dos objetivos O1, O2 e O3 definidos na Secção 1.2. Apenas foram necessárias definir certas estratégias de forma a contornar as limitações apresentadas pela *framework Forge* e das *APIs* fornecidas do *Xray* e das quais não apresentam restrições impeditivas de utilização;

2. Cada um dos quatro tipos de relatórios inicialmente propostos, assim como cada um dos requisitos funcionais associados foram implementados permitindo a integração destes tipos de relatórios em páginas do *Confluence*, não apresentando perdas de funcionalidades quando comparados aos originais, sendo o objetivo O4 alcançado;

3. No momento de configuração da *macro*, é possível a definição de diferentes análises estratégicas para três dos tipos de relatórios implementados (*Tests List*, *Overall Coverage*, *Traceability*) e, assim como acontece no funcionamento do *Xray*, apenas é possível a definição de diferentes filtros no momento de configuração do relatório *Test Runs List*;

4. Todos os relatórios implementados apresentam um elevado grau de semelhança quando comparados aos originais, sendo a sua curva de aprendizagem bastante baixa para utilizadores com experiência de utilização do Xray, sendo os requisitos não funcionais de aparência devidamente implementados, definidos no objetivo O4;

5. A aplicação encontra-se numa versão estável para o seu lançamento no mercado através da sua disponibilização no *marketplace* da *Atlassian*.

Apesar de alcançado o objetivo proposto O3, a adoção da *framework Forge* trouxe consigo desafios à implementação da proposta idealizada, uma vez que, a pouca variedade de funcionalidades disponibilizadas e os limites de utilização associados levam a que a equipa de desenvolvimento formule estratégias para o contorno destas limitações sem que as funcionalidades esperadas sejam afetadas e, apesar de não ser o caso refletido na presente dissertação de mestrado, poderão levar ao seu abandono, optando por outro tipo de *frameworks* já consolidadas no desenvolvimento de aplicações para o ecossistema *Atlassian*, como é o caso da *Atlassian Connect*. No entanto, as vantagens atualmente fornecidas pelo *Forge*, nos âmbitos de segurança e *hosting* da aplicação, e a constante aposta, por parte da *Atlassian*, no desenvolvimento da *framework* poderão levar a que, num futuro próximo, as equipas de desenvolvimento optem cada vez mais pela adoção de *Forge* tornando-o no principal método de desenvolvimento de aplicações para o ecossistema *Atlassian*.

7.2 Perspetiva de trabalho futuro

O documento produzido retrata todo o processo de desenvolvimento da aplicação para geração de relatórios do *Xray* em páginas do *Confluence*, desde a definição dos requisitos e funcionalidades da mesma até à sua implementação final, considerando todas as decisões técnicas tomadas durante o processo.

Uma vez que a aplicação apresenta uma versão estável para a sua disponibilização no *marketplace* da *Atlassian*, a principal tarefa a realizar no futuro trata-se do lançamento oficial da aplicação no mercado. Com o lançamento da aplicação, é necessário monitorar o seu comportamento face à sua utilização e, se necessário, proceder à correção de determinados aspetos.

Outro dos aspetos a considerar em etapas futuras passa pela integração de novos tipos de relatórios, de forma, a completar o espetro oferecido pelo *Xray* e acompanhar o desenvolvimento da *framework* Forge, que se encontra em constante evolução, para que estas novas funcionalidades estejam de acordo com os padrões e recomendações da *Atlassian*.

Bibliografia

Reports & analysis. Disponível em https://docs.getxray.app/pages/viewpage.action?pageId=31623136 (14/10/2023).

Test process. Disponível em https://docs.getxray.app/display/XRAYCLOUD/Test+Process (14/10/2023).

Moving from server to cloud for developers. Disponível em https://developer.atlassian.com/developer-guide/moving-from-server-to-cloud-for-developers (14/10/2023), 2021.

Confluence basics. Disponível em https://www.atlassian.com/software/confluence/guides/get-started/ (14/10/2023), 2022a.

Connect app descriptor. Disponível em https://developer.atlassian.com/cloud/jira/platform/connect-app-descriptor/ (14/10/2023), 2022b.

Coverage analysis. Disponível em https://docs.getxray.app/display/XRAYCLOUD/Coverage+Analysis (14/10/2023), 2022.

Use gadgets to add dynamic content. Disponível em https://support.atlassian.com/confluence-cloud/docs/use-gadgets-to-add-dynamic-content/ (14/10/2023), 2022a.

Defining a project. Disponível em https://confluence.atlassian.com/adminjiraserver/defining-a-project-938847066.html (14/10/2023), 2022b.

Use spaces to organize your work. Disponível em https://support.atlassian.com/confluence-cloud/docs/use-spaces-to-organize-your-work (14/10/2023), 2022.

Working with test environments. Disponível em https://docs.getxray.app/display/XRAYCLOUD/Working+with+Test+Environments (14/10/2023), 2022a.

Test repository. Disponível em https://docs.getxray.app/display/XRAYCLOUD/Test+Repository (14/10/2023), 2022b.

Xray vs zephyr. Disponível em https://www.getxray.app/xray-vs-zephyr (14/10/2023), 2022.

Confluence global settings. Disponível em https://developer.atlassian.com/platform/forge/manifest-reference/modules/confluence-global-settings/ (14/10/2023), 2023.

Build with forge. Disponível em https://developer.atlassian.com/platform/forge/ (14/10/2023), 2023.

Ui kit. Disponível em https://docs.getxray.app/display/XRAYCLOUD/GraphQL+API (14/10/2023), 2023.

Rest api v2. Disponível em https://developer.atlassian.com/cloud/jira/platform/rest/v2/ (14/10/2023), 2023.

Macroconfig. Disponível em https://developer.atlassian.com/platform/forge/ui-kit-components/form/#macroconfig (14/10/2023), 2023.

Ui kit. Disponível em https://developer.atlassian.com/platform/forge/manifest/ (14/10/2023), 2023.

Storage api. Disponível em https://developer.atlassian.com/platform/forge/runtime-reference/storage-api/ (14/10/2023), 2023.

A Aimar. Introduction to software documentation. 1998. doi: 10.5170/CERN-1998-008.135.

Paul Ammann and Jeff Offutt. *Introduction to Software Testing*. Cambridge University Press, Cambridge, England, 2 edition, 2016.

Manfred Broy. Yesterday, today, and tomorrow: 50 years of software engineering. *IEEE Software*, 35(05): 38–43, 2018. ISSN 1937-4194. doi: 10.1109/MS.2018.290111138.

Elfriede Dustin. *Effective Software Testing*. Addison Wesley, Boston, MA, 2002.

Elfriede Dustin, Thom Garrett, and Bernie Gauf. *Implementing Automated Software Testing*. Addison-Wesley Educational, Boston, MA, 2009.

Hakan Erdogmus, Nenad Medvidović, and Frances Paulisch. *50 Years of Software Engineering*, volume 35, pages 20–24. 2018. doi: 10.1109/MS.2018.3571240.

João M. Fernandes and Ricardo J. Machado. *Requirements in Engineering Projects*. Springer International Publishing, 2016. doi: 10.1007/978-3-319-18597-2.

Internation Software Test Institute. *Software Testing Revealed*. 2nd edition, 2014.

A.K.M Islam and Alex Ferworn. A comparison between agile and traditional software development methodologies. *Global Journal of Computer Science and Technology*, pages 7–42, 2020. doi: 10.34257/GJCSTCVOL20IS2PG7.

Paul C Jorgensen. *Software Testing*. Auerbach, Philadelphia, PA, 4 edition, 2013.

Cem Kaner, James Bach, and Brett Pettichord. *Lessons Learned in Software Testing*. John Wiley & Sons, 2002.

N.J. Kipyegen and W.P.K. Korir. Importance of software documentation. *International Journal of computer science issues.*, 2013.

Edward Kit. *Software Testing in the Real World*. Addison Wesley, 1995.

Renée Mccauley. Agile development methods poised to upset status quo. *SIGCSE Bulletin*, 33:14–15, 2001. doi: 10.1145/572139.572150.

Tom Mens and Serge Demeyer, editors. *Software Evolution*. Springer, 2008. doi: 10.1007/978-3-540-76440-3.

Matt Muschol. Forge vs connect - understanding jira cloud app development. Disponível em https://www.swiftix-software.com/post/forge-vs-connect-understanding-jira-cloud-app-development (14/10/2023), 2021.

Glenford J Myers, Corey Sandler, and Tom Badgett. *The Art of Software Testing*. John Wiley & Sons, 3 edition, 2011.

Packt. Jira - an overview. Disponível em https://hub.packtpub.com/jira-overview/ (14/10/2023), 2015.

Suzanne Robertson and James C Robertson. *Mastering the Requirements Process*. Addison-Wesley Educational, 2 edition, 2006.

André M. P. Rodrigues. Xray for JIRA Cloud. Dissertação de Mestrado, Técnico Lisboa, 2018.

Walt Scacchi. *Process Models in Software Engineering*. 01 2002. doi: 10.1002/0471028959.sof250.

Ian Sommerville. *Software Engineering*. Pearson, 10th edition, 2016.