INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES

Front-end para Capgemini Engineering Test Automation Framework

LUÍS MANUEL VASCONCELOS BATEIRA
Setembro de 2023

POLITÉCNICO
DO PORTO

Politécnico do Porto
Instituto Superior de Engenharia do Porto

Front-end for Capgemini Engineering Test Automation Framework

Luís Manuel Vasconcelos Bateira

Mestrado em Engenharia Electrotécnica e de Computadores
Área de Especialização em Sistemas e Planeamento Industrial

Departamento de Engenharia Eletrotécnica
Instituto Superior de Engenharia do Porto

Setembro, 2023

Esta dissertação satisfaz, parcialmente, os requisitos que constam da Ficha de Unidade Curricular de Tese/Dissertação, do 2º ano, do Mestrado em Engenharia Electrotécnica e de Computadores, Área de Especialização em Sistemas e Planeamento Industrial.

Candidato: Luís Manuel Vasconcelos Bateira, Nº 1170789,
1170789@isep.ipp.pt

Orientação Científica: Veríssimo Manuel Brandão Lima Santos,
vms@isep.ipp.pt

Empresa: Capgemini Engineering

Orientador: João Oliveira, joao.lemosoliveira@capgemini.com

Departamento de Engenharia Eletrotécnica
Instituto Superior de Engenharia do Porto
Rua Dr. António Bernardino de Almeida, 431, 4200–072 Porto

Setembro, 2023

Aos meus pais, irmã, avós, família e amigos

Agradecimentos

Este trabalho é o culminar do trajeto percorrido ao longo dos anos no Instituto Superior de Engenharia do Porto.

Gostaria de começar por agradecer à minha família, em especial aos meus pais e à minha irmã que sempre me apoiaram no meu percurso académico, e que me deram a possibilidade que não lhes foi dada.

Aos meus amigos, nomeadamente o João, Miguel, Ana, Nuno, André, Costa, Mesquita, Rui, Faria, Amadeu, Pedro, Maria, Francisco, Luana, Rocha, Tiago, Gil e ao Carlos, pelos bons e maus momentos ao longo de todo este tempo, que me permitiram crescer tanto a nível pessoal como a nível académico.

Agradecer também de forma especial à Capgemini Engineering, particularmente aos meus supervisores ao longo deste projeto: João Oliveira, Raquel Ribeiro e Ivo Santos. A sua orientação e apoio foram fundamentais em cada etapa deste trabalho. Um reconhecimento igualmente merecido a Filipe Silva e João Azevedo, cujas contribuições têm sido inestimáveis para o meu crescimento profissional.

Ao orientador do projeto, o Professor Veríssimo Santos, pelas sugestões e auxílio prestados na realização do presente relatório. E por fim, mas não menos importante, ao ISEP e ao seu corpo docente, pelas condições concedidas, ao longo destes anos.

Resumo

Testes automatizados têm emergido como um componente vital na engenharia de software, permitindo a validação rápida e eficiente de soluções em ambientes complexos e em constante evolução. O presente projeto detalha o desenvolvimento de um *front-end* intuitivo para a *Embedded Test Automation Framework* (ETAF). O foco principal foi otimizar os processos de teste ao utilizar técnicas avançadas, garantindo robustez e escalabilidade ao sistema.

A automação de características fundamentais do ETAF, como a deteção de testes e *tags*, atualização de servidores e configurações YAML, tornam a solução altamente eficaz. A integração de um *feedback* em tempo real otimizou ainda mais o processo, permitindo resultados instantâneos após a execução dos testes. Adicionalmente, a inclusão de documentação e ficheiros de report no front-end ofereceu uma visão abrangente e acessível a todos os envolvidos no processo de teste.

A colaboração com a Capgemini alavancou a ferramenta no ambiente empresarial, abrindo portas para futuras integrações e oportunidades. O *design user-friendly* da ETAF promete atração de mais clientes para testes automatizados.

Um *script* PowerShell complementar enriqueceu o *front-end*, automatizando processos e garantindo testes consistentes.

Há desafios a serem superados. A aplicação continua em desenvolvimento e requer ajustes. Funcionalidades adicionais estão em consideração.

O projeto é um avanço na otimização de testes automatizados. À medida que a tecnologia avança, o ETAF deverá evoluir para garantir a entrega de software de qualidade.

Palavras-Chave: Testes Automatizados, ETAF, *front-end*, software, *Framework*, Interface Gráfica de Usuário (GUI), Python, Otimização.

Abstract

Automated tests have emerged as a vital component in software engineering, allowing for the quick and efficient validation of solutions in complex and constantly evolving environments. The present project delineates the development of an intuitive front-end for the ETAF (Embedded Test Automation Framework). The primary emphasis was placed on optimizing test processes by employing advanced techniques, thus ensuring the system's robustness and scalability.

The automation of key features, such as the detection of tests and tags, server updates, and YAML configurations, stood out, making the solution highly efficacious. The integration of real-time feedback further optimized the process, allowing for instantaneous results following test execution. Additionally, the incorporation of documentation and report files into the front-end provided a comprehensive and readily accessible view for all stakeholders involved in the testing process.

Collaboration with Capgemini elevated the tool in the business environment, paving the way for future integrations and opportunities. The user-friendly design of ETAF promises to attract more clients for automated testing.

A supplementary PowerShell script enriched the front-end, automating tasks and ensuring consistent testing.

There are challenges to be overcome. The application remains under development and necessitates adjustments. Additional functionalities are under consideration.

The project represents a stride forward in the optimization of automated testing. As technology progresses, ETAF will need to adapt to ensure the delivery of high-quality software.

Keywords: Automated Testing, software Development, ETAF, front-end, Framework, Graphical User Interface (GUI), Python, Optimization.

Índice

Lista de Figuras

Lista de Acrónimos

AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
CAN	Controller Area Network
CI	Continuous Integration
CI/CD	Continuous Integration/Continuous Deployment
CSS	Cascading Style Sheets
ECU	Electronic Control Unit
ER&D	Engineering Research & Development
ETAF	*Embedded Test Automation Framework*
EV&V	Embedded Verification and Validation
GUI	Graphical User Interface
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
ISO 9001	International Organization for Standardization 9001
JSON	JavaScript Object Notation
LIN	Local Interconnect Network
MD	Markdown
PNG	Portable Network Graphics
R&D	Research & Development
URL	Uniform Resource Locator
VCS	Version Control System
XHTML	Extensible HyperText Markup Language

XML	Extensible Markup Language
XP	Extreme Programming
YAML	YAML Ain't Markup Language

Capítulo 1

Introdução

A validação e verificação são pilares essenciais no mundo da engenharia de software. Estes conceitos representam a garantia de que um sistema atende às necessidades e especificações definidas pelos seus utilizadores e *stakeholders*. Além disso, eles confirmam que o software foi construído corretamente, ao aderir ao *design* e aos requisitos especificados [1]. A ausência destes processos pode ter consequências graves, particularmente em aplicações críticas onde falhas podem resultar em perdas financeiras significativas, danos à reputação de uma empresa, ou até mesmo perda de vidas.

Particularizando, há o setor da aviação. Aqui, falhas de software em sistemas de controlo de voo podem ser catastróficas. Em situações menos críticas, como aplicações bancárias, falhas podem resultar em transações incorretas ou exposição de dados sensíveis de clientes. Ambos os cenários ilustram a extrema necessidade de processos rigorosos de validação e verificação.

No entanto, à medida que o software se torna mais complexo e as equipas de desenvolvimento procuram ciclos de lançamento mais rápidos, realizar validação e verificação manual torna-se cada vez mais difícil, dispendioso e propício a erros. É aqui que entra a importância da automatização de testes. Através desta, é possível acelerar estes processos, garantindo simultaneamente que a qualidade e segurança do software não sejam comprometidos.

A automatização de testes tem crescido na área da engenharia de software. Muitas organizações têm investido em testes automatizados para a prevenção de defeitos e aumentar a eficácia dos testes durante o desenvolvimento de software. Reconhecendo esta tendência, empresas e instituições de pesquisa têm dado ênfase ao estudo e implementação de técnicas de testes automatizados, dada as claras mais-valias que estas trazem.

Neste contexto, o presente projeto, encomendado pela Capgemini Engineering, procura abordar e, sobretudo, facilitar a implementação e gestão da automatização de testes. Ao simplificar este processo, o objetivo é proporcionar aos desenvolvedores uma ferramenta que lhes permita concentrar-se mais intensamente no desenvolvimento de testes robustos e menos na infraestrutura e gestão desses testes. Com isso, espera-se acelerar a adição de novas funcionalidades e melhorar a frequência e qualidade das entregas. Esta otimização não beneficia apenas o desenvolvedor, ao tornar o seu trabalho mais eficiente, mas também o utilizador final, que passa a usufruir de um software mais fiável e atualizado com maior frequência. Assim, ao facilitar a prática da automatização de testes, o projeto coloca o foco onde realmente importa: na criação de testes eficazes e na entrega contínua de valor.

1.1 Contextualização

O ciclo de vida de desenvolvimento de software inclui diferentes fases, as mesmas podem ser distinguidas como: a análise de requisitos, arquitetura, *design*, implementação, testagem, a entrega de software e por fim, a manutenção [2].

A testagem de software é uma atividade crucial ao longo do ciclo de vida de desenvolvimento do mesmo. O objetivo é validar que as alterações no código aplicadas ao produto não comprometem a qualidade do produto. Em termos de metodologia *agile*, entregar atividades de teste de software é desafiador devido a mudanças frequentes no código, velocidade de entrega e exigências do mercado [3]. Para atender a esses desafios, a automatização de testes desempenha um papel importante em manter o ritmo com as necessidades de desenvolvimento. Executar testes rapidamente e fornecer *feedback* imediato é um elemento-chave para que as atividades de entrega de automatização de testes sejam bem-sucedidas.

Ao utilizar ferramentas de automação de testes, é possível mapear e parametrizar os resultados dos testes anteriormente manuais. Dessa forma, eles podem ser executados e validados sem a intervenção direta de um testador humano.

Os benefícios da implementação desta automatização refletem em práticas *agile* otimizadas, como trabalho em equipa, distribuição adequada de tarefas, utilização de ferramentas apropriadas e gestão eficaz do conhecimento.

Neste panorama, a automação dos testes de interface do utilizador emerge como uma significativa melhoria no processo de desenvolvimento de software. Esta abordagem permite a execução de testes de regressão e a reutilização dos testes já criados, consolidando-se como uma solução eficiente. Através dela, é possível simular interações do utilizador de forma automatizada, validar fluxos de trabalho complexos e identificar potenciais problemas, seja em termos de usabilidade ou de compatibilidade.

1.1.1 Estudo do Problema

Enquanto os testes automatizados representam um avanço crítico na engenharia de software, a abordagem e a eficiência com que esses testes são conduzidos podem ser drasticamente afetados pela interface através da qual os desenvolvedores e testadores interagem. Em muitas ferramentas de teste, percebeu-se uma lacuna significativa: a ausência de uma interface amigável ao utilizador.

Historicamente, muitas soluções de testes automatizados eram acompanhadas por interfaces que, apesar de funcionais, não eram projetadas com a experiência do utilizador em mente, como é o caso do ETAF. Isso significava que, embora a lógica subjacente fosse poderosa e eficaz, a barreira de entrada para novos utilizadores ou mesmo para profissionais experientes, mas não familiarizados com a ferramenta específica, era alta.

A ausência de um *front-end* intuitivo e *user-friendly* pode resultar em vários desafios. Primeiramente, um tempo significativo pode ser gasto apenas tentando entender como utilizar a ferramenta efetivamente, ao invés de focar na criação e execução de testes. Isso leva a uma redução na produtividade e eficácia dos testes. Em segundo lugar, uma interface complicada pode resultar em erros inadvertidos, nos quais os testes podem ser mal configurados ou mal interpretados devido à complexidade da ferramenta.

Tendo em vista esses desafios, identificou-se a necessidade de desenvolver um *front-end* que poderia servir como uma ponte entre a lógica poderosa dos testes automatizados e a interação humana, tornando o processo mais intuitivo, acessível e eficaz. Acredita-se que, ao enfrentar este problema, não só a eficácia dos testes pode ser melhorada, mas também a adoção de práticas de testes automatizados pode ser

acelerada, pois os utilizadores sentir-se-iam mais confiantes e habilitados ao usar ferramentas que são construídas a pensar neles.

Em resumo, este projeto, ao reconhecer a complexidade que muitas ferramentas de teste possuem, procura entregar uma solução que, além de eficiente em termos técnicos, seja projetada com a experiência do utilizador em mente. E, ao fazer isso, pretende-se não só melhorar a eficiência dos testes, mas também democratizar a prática de testes automatizados, tornando-a mais acessível a uma gama mais ampla de profissionais.

1.2 Apresentação da Empresa

A Capgemini é uma multinacional francesa fundada em 1967 que está entre os maiores fornecedores de serviços de consultoria, tecnologia e *outsourcing* do mundo [4]. O Grupo Capgemini conta com mais de 300 mil profissionais em 44 países, e apresentou uma receita global de 22.000 Milhões de euros em 2022.

Figura 1.1: Logotipo da Capgemini.

De seguida estão representados os gráficos com os valores percentuais da empresa por região, negócio e por setor, respetivamente:

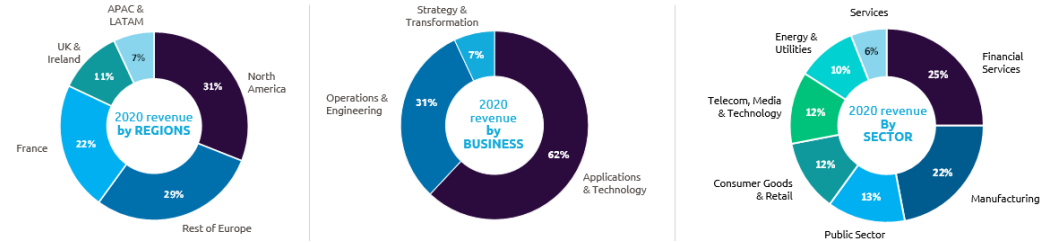


Figura 1.2: Dados relativos à Capgemini.

Em Portugal conta com quatro centros físicos, situados no Porto, Évora, Fundão e Lisboa, conta com mais de dois mil e duzentos engenheiros, mais de cinquenta clientes espalhados pelo mundo, e com mais de cem projetos em andamento [5].

A Capgemini Engineering é a central de engenharia e Research & Development (R&D) do Grupo Capgemini, esta central é orientada para ajudar clientes a desenvolver produtos e serviços inteligentes, operações e serviços em escala para alcançar uma indústria mais inteligente. Com o conhecimento da indústria, tecnologias de vanguarda em digital e software, além do *mindset agile*, faz com que esta central do Grupo seja o líder global de Engineering Research & Development (ER&D). Os principais clientes desta multinacional estão demonstrados na Figura 1.3:

Figura 1.3: Principais parceiros da Capgemini.

A Capgemini é dotada de diversos certificados de referência, dos quais se destaca a norma International Organization for Standardization 9001 (ISO 9001), sistemas de gestão mais utilizada mundialmente, sendo a referência internacional para a Certificação de Sistemas de Gestão da Qualidade.

O presente projeto enquadra-se na equipa Embedded Verification and Validation (EV&V). Esta equipa, tem como missão reduzir o risco de falha de software através da utilização de técnicas de teste mais eficientes na conceção do caso de teste. A visão de que a equipa de teste e quem testa são o ponto de partida para a qualidade do software, aumentando a exposição aos defeitos e prestando apoio nas atividades de depuração. E com valor de fornecer relatórios de testes claros e fiáveis e as

Figura 1.4: Certificados da Capgemini.

necessárias provas de falhas, e assegurar os métodos de teste mais eficientes através da utilização de testes automáticos sempre que possível.

1.3 Objetivos

O cerne deste projeto é o desenvolvimento de um *front-end* integrado, capaz de automatizar tanto a execução quanto a configuração dos testes. Este interface deverá fornecer todas as informações relevantes ao utilizador de forma intuitiva e eficaz.

Os objetivos delineados para este projeto são esquematizados em tarefas específicas:

- **Criação da Interface**:
  - Desenvolvimento do *front-end* via *microframework* Flask;
  - Integração com tecnologias como HTML e JavaScript;
  - Estabelecer comunicação entre o *front-end* e o *back-end*.
- **Operações relativas aos Servidores**:
  - Modificar o estado dos servidores;
  - Iniciar servidores;
  - Desativar servidores.
- **Gestão e Execução de *Test Cases***:
  - Distribuir os ficheiros robot pelos diretórios apropriados;
  - Ler e identificar *tags* de cada ficheiro robot;

  - Exibir *tags* apenas dos ficheiros que foram selecionados;
  - Permitir a filtragem dos testes a serem executados.

- **Apresentação de *logs* na interface**;
- **Visualização de resultados (ficheiros *report* na interface**;
- **Acesso à documentação de 'libraries' e 'resources' via interface**;
- **Automatização das configurações dos servidores, consoante o repositório**;
- **Documentação interativa dos *testcases***;
- **Guia de Utilização**.

1.4 Plano de Trabalho

A dissertação foi desenvolvido ao longo de trinta e três semanas, com o planeamento de execução de tarefas demonstrados na figura1.5. Encontram-se as datas de início e término de cada uma das tarefas realizadas no estágio na empresa, que conduziram no final à realização da dissertação de tese de mestrado.

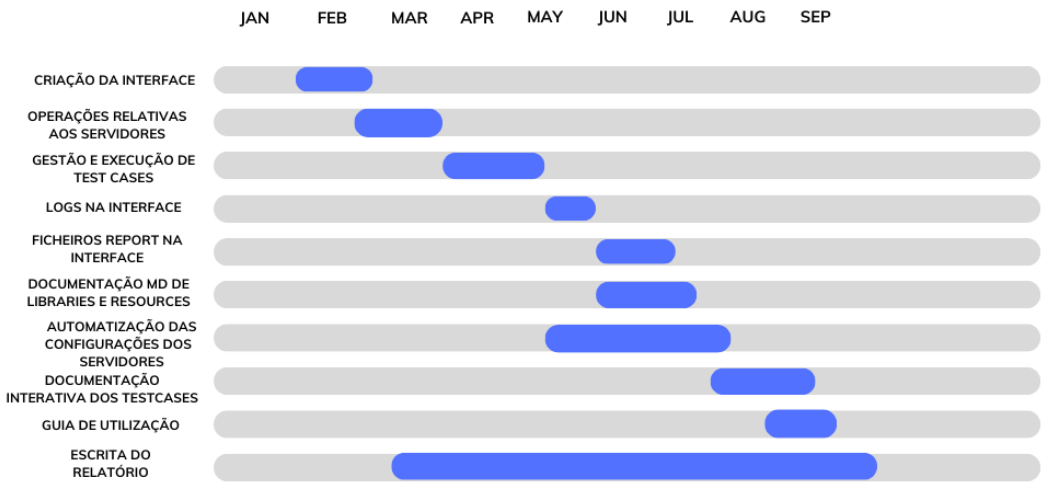


Figura 1.5: Calendarização de tarefas.

1.5 Organização da Dissertação

O vigente documento é composto por cinco capítulos distintos. No primeiro capítulo, a Introdução, é feita uma contextualização do tema da dissertação, os objetivos e resultados esperados da mesma.

No segundo capítulo, é feito a revisão de literatura, que consiste no estudo aprofundado de conceitos e tecnologias envolventes da automatização de testes. São apresentadas diferentes perspetivas, vantagens e desvantagens, e possíveis alternativas.

Posteriormente, no capítulo três, é explicado o desenvolvimento do projeto em si. São enunciados os requisitos para o funcionamento do mesmo, a arquitetura do sistema, e o trabalho desenvolvido.

No capítulo quatro são demonstrados os resultados. É, respetivamente, feita uma análise crítica aos mesmos, e enunciados possíveis aspetos que possam ser melhorados.

Por fim, no quinto capítulo são reunidas as principais conclusões. É avaliado se o projeto desenvolvido está de acordo com o pretendido pela empresa, e são perspetivados futuros desenvolvimentos, de modo a otimizar e dar continuidade ao projeto.

Capítulo 2

Revisão da literatura

O presente capítulo apresenta o conceito teórico, o meio envolvente do tema da dissertação, além de tecnologias e linguagens de programação que se enquadram com o mesmo. São explicados os respetivos conceitos, dados, vantagens e alternativas das diferentes modalidades.

2.1 Ciclo de Vida do software

O ciclo de vida do software é um processo que abrange todas as etapas do desenvolvimento de um software, desde a sua conceção até a sua implementação, manutenção e até à sua eventual retirada. Cada fase do ciclo de vida desempenha um papel fundamental na criação de software com qualidade e funcional.

Figura 2.1: Ciclo de Vida do software [6]

De um alto nível é possível classificar, da seguinte forma, as diferentes fases do ciclo de software [2] [7]:

- Conceção (*Inception*): Nesta fase, a ideia do software é concebida. São definidos os objetivos gerais, o âmbito do projeto, os requisitos iniciais e a viabilidade do projeto. É uma fase de planeamento e tomada de decisões estratégicas.
- Definição (*Requirements Gathering and Analysis*): Na fase de definição, são recolhidos e analisados os requisitos do software de forma detalhada. Isso envolve a compreensão das necessidades dos utilizadores e a tradução dessas necessidades em requisitos técnicos claros, objetivos e precisos.
- Design (*Design*): O *design* do software é elaborado nesta fase. São definidas as arquiteturas de software, a estrutura do sistema e as interfaces entre os diferentes módulos. O *design* pode ser tanto de alto nível, abordando a estrutura global do software, como de baixo nível, em que são detalhados os componentes individuais.
- Implementação (*Coding*): O desenvolvimento de código do software acontece nesta fase. Os programadores traduzem o *design* em código-fonte executável. É uma fase de construção ativa do software.
- Testes (*Testing*): Nesta fase, o software é testado para garantir que ele atenda aos requisitos definidos e funcione conforme o esperado. Diferentes tipos de testes, como testes unitários, testes de integração e testes de aceitação, são realizados para identificar defeitos e erros.

- Implantação (*Deployment*): Após os testes serem bem-sucedidos, o software é implantado num ambiente de produção. Isto pode envolver a configuração de servidores, a instalação do software em dispositivos dos utilizadores ou a disponibilização em ambientes *cloud*.
- Manutenção (*Maintenance*): O software é mantido e atualizado para dar resposta a problemas identificados após a implantação e para incorporar novos recursos ou melhorias. Como correção de *bugs*, otimizações e adaptações a novos requisitos.
- Retirada (*Retirement*): No final do ciclo de vida, o software pode ser retirado de uso. Tal pode ocorrer quando o software se torna obsoleto, não atende mais às necessidades dos utilizadores ou é substituído por uma versão mais recente.

O ciclo de vida do software é um processo contínuo cuja finalidade é criar, manter e melhorar sistemas do mesmo ao longo do tempo, garantindo que estes atendam às necessidades dos utilizadores e permaneçam atualizados e funcionais.

2.2 Metodologias *Agile* no contexto de desenvolvimento de software

No campo do desenvolvimento de software, as metodologias *agile* surgiram como uma resposta inovadora à crescente demanda por adaptabilidade e entrega contínua de valor. Não são meramente técnicas ou procedimentos, trata-se de uma filosofia que coloca o cliente e a adaptabilidade como centro do processo de desenvolvimento [8].

Agile caracteriza-se por ciclos curtos de desenvolvimento, frequentemente designados como *sprints* ou iterações. Estes *sprints*, que normalmente têm uma duração de duas a quatro semanas, permitem a produção constante de versões do software que são potencialmente disponibilizáveis. A intenção é assegurar que, ao invés de aguardar meses ou mesmo anos por um produto final, os clientes e *stakeholders* obtenham valor de forma contínua e tenham a possibilidade de fornecer feedback regularmente [9].

Esta entrega contínua é complementada por uma forte ênfase na colaboração. No desenvolvimento ágil, a dinâmica de equipa é crucial. A equipa não só opera de forma coesa, mas também interage frequentemente com os *stakeholders* e clientes, garantindo que as suas necessidades e opiniões sejam integradas no produto em evolução. Esta interação constante também assegura que a equipa esteja sempre

preparada para se adaptar a mudanças, sejam elas novas exigências do mercado, opiniões dos clientes ou desafios tecnológicos emergentes.

Dentro das diversas metodologias *agile*, algumas destacam-se pela sua notoriedade e eficácia. O *scrum*, por exemplo, é reconhecido pelo seu foco em *sprints* e pelas suas reuniões regulares de planeamento e revisão. O Kanban, por sua vez, foca-se na visualização do fluxo de trabalho, permitindo que as equipas observem o progresso e identifiquem possíveis constrangimentos. Extreme Programming (XP) prioriza práticas de engenharia robustas, enquanto o *Lean* software *Development* encontra inspiração nos princípios da produção *lean*, priorizando a redução de desperdícios e o foco na qualidade desde a fase inicial [10].

A adoção do *agile* apresenta numerosos benefícios. A flexibilidade intrínseca a estas metodologias garante que as equipas possam responder prontamente, fornecendo soluções que correspondam às necessidades atuais do mercado ou do cliente. Esta entrega contínua de valor, aliada à capacidade de obter feedback regular, estabelece uma dinâmica em que tanto os desenvolvedores quanto os clientes se encontram mais em sintonia e satisfeitos com os resultados. No entanto, o *agile* não está isento de desafios. A transição para uma mentalidade *agile* pode requerer uma significativa mudança cultural nas organizações. Além disso, devido à natureza adaptativa do *agile*, pode ser um desafio fornecer estimativas rigorosas em termos de prazos e recursos.

Em resumo, as metodologias *agile* introduziram uma abordagem revolucionária ao desenvolvimento de software, priorizando a entrega contínua, a colaboração e a adaptabilidade. Embora apresente os seus próprios desafios, quando corretamente adotado, o *agile* pode resultar numa entrega mais célere, numa maior satisfação do cliente e em produtos de superior qualidade.

2.3 V-Model no Desenvolvimento de Software

O V-Model, ou Modelo V, é uma extensão do modelo de ciclo de vida tradicional de software. É assim chamado devido à sua forma característica, que se assemelha à letra "V", onde a parte esquerda representa as fases de especificação ou definição e a parte direita denota as fases de validação ou testagem [11].

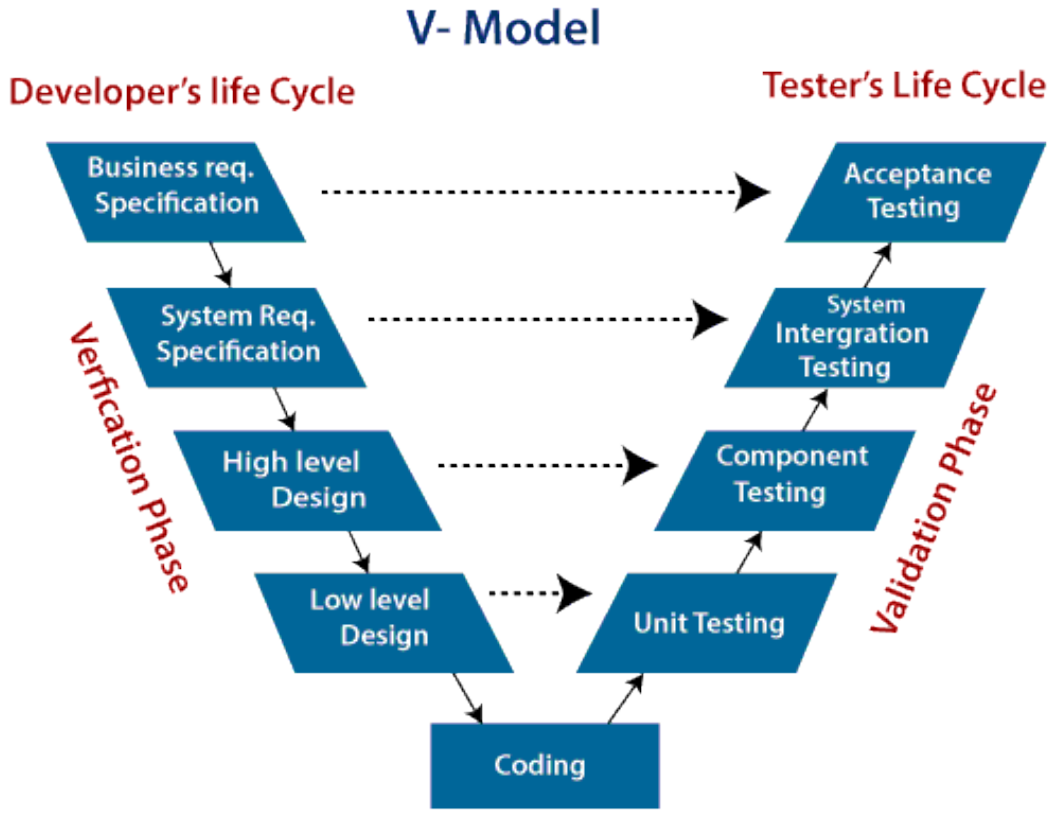


Figura 2.2: Representação do V-Model [12]

A principal ideia subjacente ao V-Model é que as fases de desenvolvimento (especificação e desenho) são correspondidas, de forma paralela, por fases de testagem. Assim, para cada nível de detalhe no desenvolvimento, existe um correspondente nível de testagem [13].

O V-Model pode ser interpretado com base em três eixos principais:

- Eixo horizontal: Representa o progresso temporal do projeto, desde a conceção até a finalização.
- Eixo vertical esquerdo: Denota as fases de especificação e desenho do software. Começa com a definição de requisitos, seguida pela arquitetura de alto nível e, por fim, pelos detalhes técnicos.
- Eixo vertical direito: Representa as fases de validação e verificação. A verificação do software começa no nível de unidade, seguida pela integração e, por fim, a aceitação.

O que torna o V-Model tão valioso é a sua ênfase numa abordagem sistemática e disciplinada para a validação e verificação. Esta estrutura garante que cada etapa do

desenvolvimento seja cuidadosamente testada, minimizando assim a probabilidade de defeitos não detetados.

Algumas vantagens do V-Model incluem:

- **Clareza**: Devido à sua natureza visual, o V-Model proporciona uma representação clara e sistemática do processo de desenvolvimento e testagem.
- **Rigor**: O V-Model enfatiza a necessidade de testes rigorosos e estruturados em cada etapa do desenvolvimento.
- **Deteção precoce de defeitos**: Ao alinhar as fases de desenvolvimento e testagem, o V-Model ajuda na identificação e correção de defeitos em estágios iniciais.

No desenvolvimento *front-end*, o V-Model é empregue para garantir uma abordagem estruturada e eficaz à criação e testagem de interfaces do utilizador. Ao iniciar um projeto de *design* de interface, os requisitos do utilizador são claramente definidos e documentados. Conforme se avança para a fase de *design*, criam-se protótipos e maquetes para visualizar a solução proposta no software Figma. Quando a fase de implementação começa, o código é escrito e depois testado em paralelo com as etapas correspondentes do lado esquerdo do V, garantindo que cada nível de detalhe seja validado e verificado. Por exemplo, uma funcionalidade como um formulário de inscrição pode ser testada quanto à sua funcionalidade (testes de unidade), integração com outros componentes (testes de integração) e, finalmente, a experiência do utilizador final (testes de aceitação). O V-Model ajuda a garantir que o produto final esteja alinhado com as expectativas iniciais e atenda aos padrões de qualidade desejados.

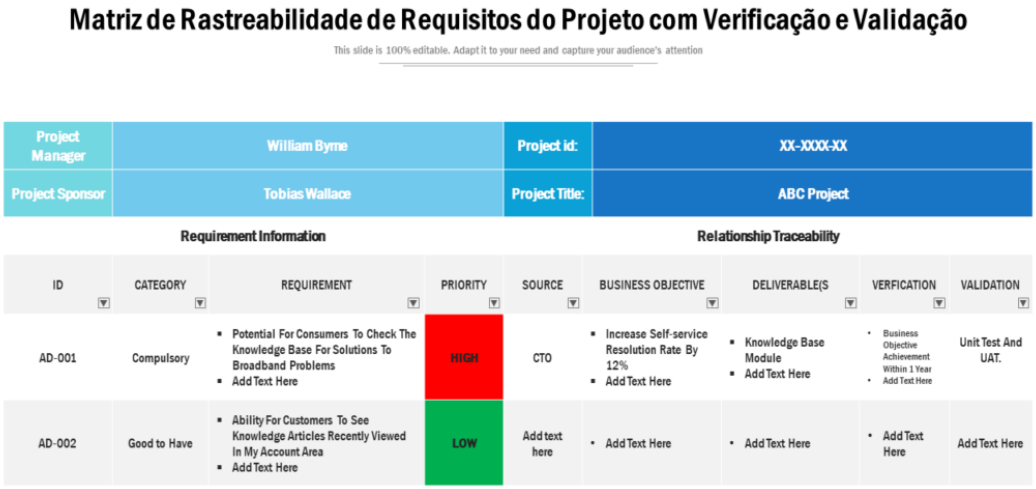
Em suma, o V-Model destaca-se como uma metodologia que entrelaça de forma meticulosa o desenvolvimento e a testagem de software. No cenário de desenvolvimento *front-end*, onde a experiência do utilizador é fundamental, esta abordagem assegura que as interfaces sejam não apenas funcionais, mas também intuitivas e alinhadas com as expectativas iniciais.

2.4 Testagem de software

A testagem de software é, inquestionavelmente, a ferramenta primordial para assegurar e controlar a qualidade num ambiente de desenvolvimento de software. Ela é uma componente indispensável do ciclo de vida do software. Esta etapa não apenas valida a conformidade do software com os requisitos estipulados pelos clientes, mas

também identifica discrepâncias entre o comportamento real e o comportamento desejado do produto final [14].

Uma ferramenta fundamental neste contexto é a matriz de rastreabilidade. Ela permite associar requisitos a testes específicos, assegurando que todos os requisitos são devidamente testados e validados. Através desta matriz, é possível rastrear a origem de um problema ou defeito até ao requisito original, simplificando o processo de diagnóstico e resolução.



Matriz de Rastreabilidade de Requisitos do Projeto com Verificação e Validação

This slide is 100% editable. Adapt it to your need and capture your audience's attention

Project Manager	William Byrne	Project id:	XX-XXXX-XX
Project Sponsor	Tobias Wallace	Project Title:	ABC Project

Requirement Information					Relationship Traceability			
ID	CATEGORY	REQUIREMENT	PRIORITY	SOURCE	BUSINESS OBJECTIVE	DELIVERABLE(S	VERFICATION	VALIDATION
AD-001	Compulsory	▪ Potential For Consumers To Check The Knowledge Base For Solutions To Broadband Problems ▪ Add Text Here	HIGH	CTO	▪ Increase Self-service Resolution Rate By 12% ▪ Add Text Here	▪ Knowledge Base Module ▪ Add Text Here	▪ Business Objective Achievement Within 1 Year ▪ Add Text Here	Unit Test And UAT.
AD-002	Good to Have	▪ Ability For Customers To See Knowledge Articles Recently Viewed In My Account Area ▪ Add Text Here	LOW	Add text here	▪ Add Text Here	▪ Add Text Here	▪ Add Text Here	Add Text Here

Figura 2.3: Exemplo de Matriz de Rastreabilidade de Requisitos de um Projeto [15]

O custo associado à testagem de software pode ser significativo, oscilando entre 30% e 80% do orçamento total de desenvolvimento. Grande parte do ciclo de lançamento de um software é consagrado à testagem. Para garantir uma testagem de qualidade e validada por padrões reconhecidos, muitas organizações optam por obter certificações. O ISTQB (International software Testing Qualifications Board) é um dos organismos líderes em certificação de testagem de software e oferece um padrão global para qualificação de profissionais na área. Ter uma equipa certificada pelo ISTQB não só eleva o padrão de qualidade como também fornece um marco de confiança para os *stakeholders*.

No que concerne à garantia de qualidade, os *quality gates* desempenham um papel crucial. Estes são essencialmente critérios que um software deve cumprir antes de avançar para a próxima fase do ciclo de vida. Os *quality gates* asseguram que os padrões de qualidade são mantidos em cada etapa, minimizando a probabilidade de falhas no produto final. Cada gate possui critérios específicos, muitas vezes referidos

como "test quality"ou "critérios gates", que determinam se o software está apto a prosseguir ou se necessita de revisões adicionais.

O ciclo de testagem de software é composto por várias fases. Tudo começa com a análise meticulosa dos requisitos para delineação dos cenários de teste. Posteriormente, são criados os casos de teste que abordam distintas funcionalidades e contextos de uso. Depois da execução destes casos, os resultados obtidos são confrontados com os resultados antecipados.

Dentro da testagem de software, existem várias técnicas e abordagens que podem ser empregadas. Testes de unidade, integração, sistema e aceitação são apenas alguns dos tipos de testes que garantem uma cobertura abrangente durante o processo de testagem (Abordados detalhadamente no subcapítulo 2.5).

A testagem pode ser dividida em dois tipos principais: manual e automatizada. Enquanto a testagem manual depende da interação humana, a automatização procura otimizar os processos e economizar tempo e recursos.

2.5 Automatização de Testes

Os testes automatizados são uma técnica utilizada para verificar a qualidade e funcionalidade de software. Estes são executados através de *scripts* que simbolizam a interação de um utilizador com o software. Permite a verificação repetitiva de funcionalidades, o que torna o processo de teste mais eficiente e menos propenso a erros humanos [16].

Antes de automatizar, é necessário desenvolver um caso de uso e uma estratégia de automação para auxiliar nas decisões sobre quais partes do âmbito do teste devem ser automatizadas e em que grau.

Um teste automatizado geralmente começa com a identificação dos casos de teste para o software em causa. Estes casos de teste são então documentados em *scripts* e realizados por ferramentas de teste automatizadas. Os resultados são então comparados com o previsto e, se houver uma discrepância, é realizada uma análise para identificar a causa raiz do problema.

A automatização de testes é um componente crucial no desenvolvimento *agile*, em que é utilizado para um rápido feedback e permite a realização de testes por parte de todos os responsáveis que entregam o código. Os testes automatizados podem ser executados repetidamente com resultados comparativamente mais baixos custos e a uma velocidade mais rápida.

Há diferentes abordagens entre a automatização de testes de software, nas quais se destacam a *User Interface Test*, visto ser a desenvolvida no âmbito desta dissertação.

Os testes *User Interface* e as interfaces *front-end* de uma aplicação são mostradas e funcionam conforme os requisitos, em todos os browsers e plataformas de suporte. As virtual machines são por norma utilizadas em paralelo para testar diversas combinações a grande velocidade para otimizar todo o processo.

2.5.1 Tipos de Testes Automatizados

Os testes automatizados diferenciam-se em vários tipos, cada um com foco em objetivos específicos. São exemplos os testes unitários, de integração, de regressão, de aceitação, desempenho e de interface gráfica. Esta diversidade de tipos de testes automatizados permite abordar diferentes perspetivas do software e garantir a sua qualidade ao longo do ciclo de desenvolvimento [17]. Em 2018, segundo a Katalon, os tipos de testes que aplicam automatização têm especial foco nos testes funcionais e de regressão. Como é possível observar na Figura 2.4.

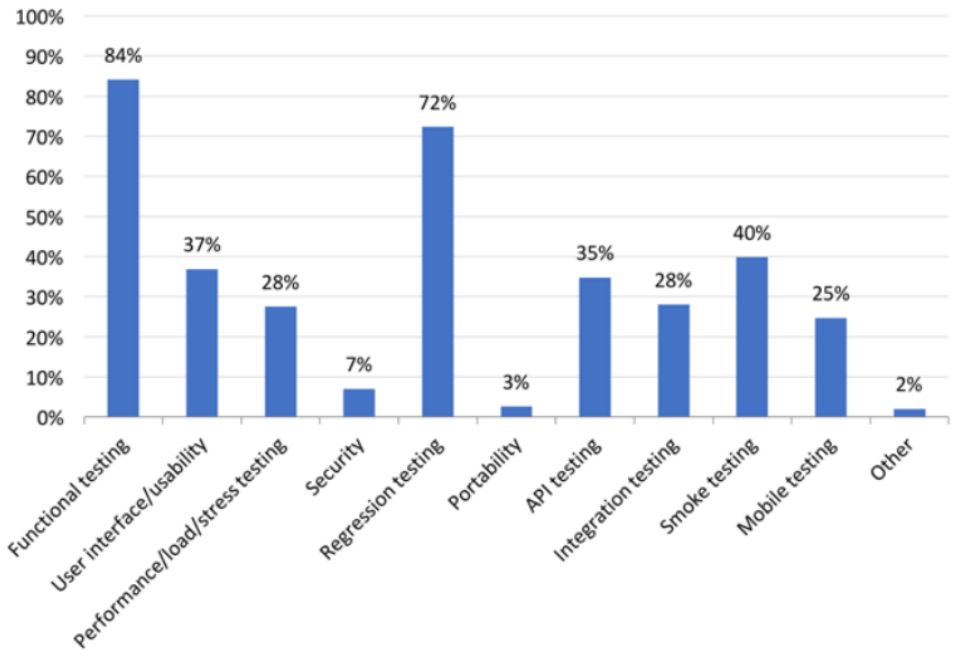


Figura 2.4: Distribuição de tipos de testes automatizados.

- **Testes Unitários**

  Os testes unitários orientados a objetos são programas que testam classes. Cada caso de teste consiste numa sequência fixa de invocações de métodos

com argumentos fixos, em que é explorado um aspeto específico do comportamento da classe em teste. Os testes unitários estão-se a tornar um componente importante no desenvolvimento de software, permitindo mudanças contínuas e controladas no código. Ao contrário dos testes tradicionais, são os desenvolvedores, e não os *testers*, que escrevem os testes para todas as partes das classes que estão a ser desenvolvidos. No entanto, a criação manual de testes é demorada, o que faz com que os conjuntos de testes unitários cubram apenas alguns aspectos das classes [18].

Com a crescente importância dos testes unitários, muitas empresas oferecem ferramentas, *frameworks* e serviços direcionados para esse tipo de teste. Essas ferramentas vão desde *frameworks* especializados, como o JUnit, até ferramentas de geração automática de testes, como o Jtest da Parasoft. No entanto, muitas dessas ferramentas, apesar de gerarem testes, não garantem a cobertura completa do código, especialmente em termos de cobertura de ramificações e caminhos internos dos métodos das classes em teste.

Uma prática padrão durante os testes unitários envolve um ciclo iterativo de ações que garante que o código em desenvolvimento atenda às expectativas e esteja livre de defeitos. A começar por uma revisão minuciosa do código escrito. Esta etapa garante que se entenda o comportamento esperado e permite a identificação preliminar de potenciais problemas ou áreas de melhoria. De seguida, quaisquer alterações necessárias são implementadas para otimizar ou corrigir o código. Uma vez feitas essas mudanças, os testes são executados, e os resultados esperados são comparados aos resultados reais. Discrepâncias indicam que existem *bugs* ou inconsistências no código. Por isso, a etapa seguinte é identificar e corrigir esses *bugs*. Para garantir que as correções sejam eficazes, é essencial reexecutar os testes e validar que os problemas identificados foram de facto resolvidos. Este ciclo é repetido até que o código atenda aos padrões de qualidade desejados e passe em todos os testes unitários. Este processo não só garante a robustez do código, mas também aumenta a confiança no seu desempenho quando integrado em sistemas maiores. O processo é assim demonstrado na Figura 2.5: *bugs*

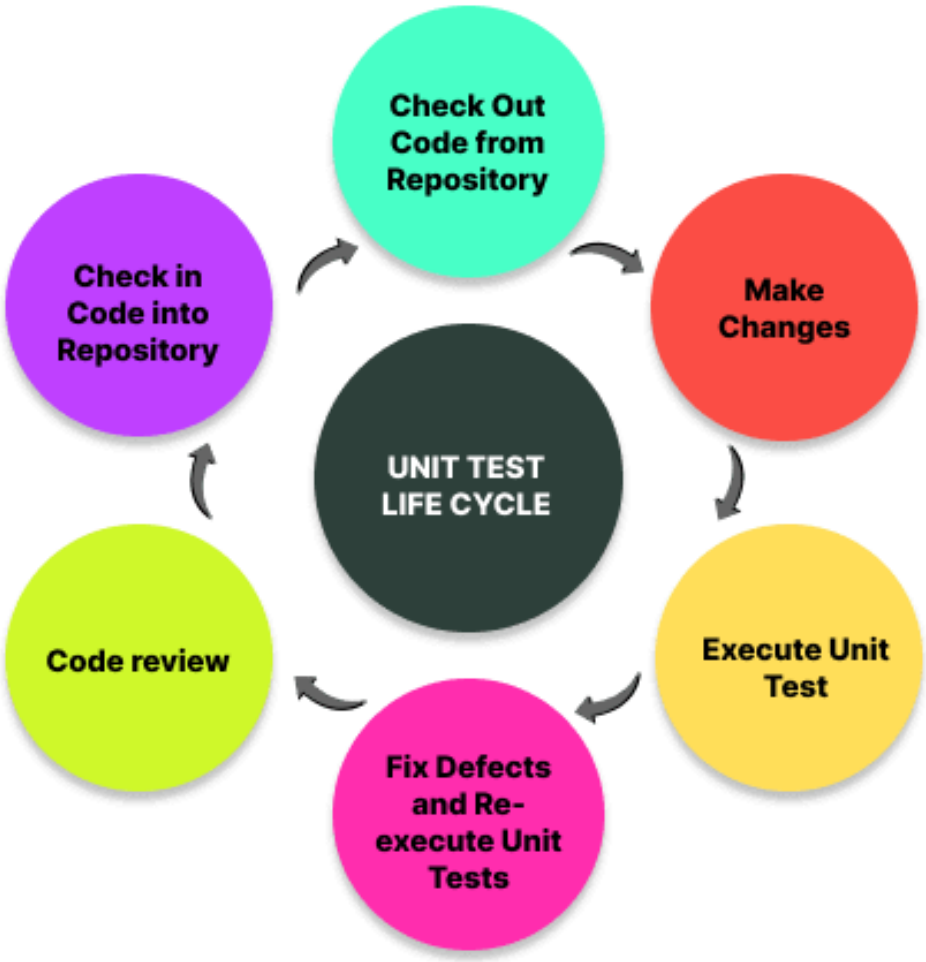


Figura 2.5: Ciclo de vida de testes unitários [19].

- **Testes de Integragção**

  Os testes de integração são uma prática essencial no desenvolvimento de software de modo a assegurar que diferentes módulos ou componentes do sistema funcionem de forma correta juntos. Esses testes têm como finalidade verificar a integração entre os elementos do software, identificar problemas de comunicação e garantir que o software funcione como um todo.

  Existem diversas estratégias para realizar os testes de integração, e uma abordagem comum é a realização de testes incrementais, onde os módulos são integrados gradualmente para verificar o seu funcionamento em conjunto.

Essa abordagem permite detectar problemas de compatibilidade mais cedo no processo de desenvolvimento, facilitando a sua correção.

Os testes de integração podem ser realizados manualmente ou de forma automatizada, dependendo da complexidade do sistema e dos recursos disponíveis. Ferramentas de automatização de testes são frequentemente utilizadas para agilizar o processo e garantir a cobertura abrangente da integração entre os componentes.

Além disso, é importante salientar que os testes de integração devem ser realizados após a conclusão dos testes unitários, que verificam o funcionamento individual de cada componente isoladamente. Os testes de integração visam validar o comportamento do sistema como um todo, identificando possíveis falhas de interação entre os componentes.

Em suma, os testes de integração são importantes para garantir a qualidade do software, ao evitar problemas de compatibilidade e assegurando que todos os componentes trabalhem em harmonia para atender aos requisitos do sistema.

- **Testes de Regressão**

Testes de regressão são executados para garantir que alterações recentes no código não introduziram regressões, ou seja, não afetaram negativamente funcionalidades previamente testadas.[20]

Uma estratégia de testes de regressão propõe a reexecução de todos os casos de teste que ainda pertencem ao domínio de input da nova versão. Contudo, devido ao alto consumo de recursos dessa estratégia de reexecução completa, têm sido feitos esforços para reduzir o seu custo [21].

As técnicas de reexecução seletiva visam reduzir o custo dos testes de regressão, ao testar apenas partes selecionadas do software. Tradicionalmente, essas técnicas têm se concentrado em dois problemas: no problema de seleção de casos de teste para reexecução, por exemplo, selecionando um subconjunto dos casos de teste existentes, e no problema de identificação de cobertura, ou seja, identificar partes do software que requerem testes adicionais.

Existem duas abordagens principais para a reexecução seletiva dos testes de regressão: a abordagem segura e a abordagem de minimização. A abordagem segura seleciona todos os casos de teste existentes que exercem qualquer elemento do programa que possa ser afetado por uma determinada mudança no código. Por outro lado, a abordagem de minimização procura selecionar o menor conjunto de casos de teste necessário para testar os elementos do programa que foram afetados por as respetivas alterações [21].

Em geral, as estratégias de teste de regressão devem ser escolhidas com base nas características específicas do projeto de desenvolvimento de software. Casos em que as mudanças são frequentes e o código sofre muitas atualizações pode ser benéfico o uso de mais de técnicas de reexecução seletiva para economizar tempo e recursos. Por outro lado, em projetos mais estáveis, a reexecução completa pode ser mais adequada para garantir a integridade do software. A seleção da abordagem apropriada dependerá do contexto do projeto, do tamanho e complexidade do software e dos recursos disponíveis para o teste.

- **Testes de Aceitação**

Os testes de aceitação em software desempenham um papel crucial para garantir que o software atenda aos requisitos e expectativas dos utilizadores. Ao contrário dos testes de regressão que visam validar mudanças, os testes de aceitação concentram-se em verificar se o software atende aos critérios de aceitação definidos pelo cliente ou *stakeholders* [17].

Existem várias estratégias para realizar testes de aceitação de software. Uma abordagem comum é a criação de cenários de teste que representam casos de uso realistas. Esses cenários são elaborados com base nos requisitos funcionais e não funcionais do sistema e são executados para validar o comportamento e a funcionalidade do software em cenários reais de uso.

Outra técnica para testes de aceitação é a automação de testes de interface do utilizador. Ferramentas como o Selenium permitem simular interações do utilizador com o software, simplificando a validação de fluxos de trabalho e funcionalidades complexas.

Além disso, os testes de aceitação podem envolver testes de desempenho, segurança e usabilidade para garantir que o software atenda aos padrões e requisitos de qualidade definidos.

Assim como nos testes de regressão, a reexecução seletiva também é relevante nos testes de aceitação. A depender das mudanças no software, é importante identificar e selecionar os casos de teste relevantes para validar as novas funcionalidades e áreas impactadas.

A escolha da estratégia de teste de aceitação deve ter em conta a natureza do projeto, os requisitos do cliente, a complexidade do software e os recursos disponíveis. Projetos com requisitos em constante alteração podem se beneficiar de abordagens mais flexíveis e automatizadas, enquanto projetos estáveis podem priorizar a reexecução completa para garantir a qualidade global do sistema.

Em resumo, os testes de aceitação são essenciais para a validação final do software e garantir que o produto entregue atenda às expectativas dos utilizadores e aos requisitos do cliente. A seleção da abordagem correta e a reutilização seletiva de casos de teste são fundamentais para tornar o processo de teste mais eficiente e eficaz.

- **Testes de Desempenho**

Os testes de desempenho são uma etapa crucial no processo de garantia da qualidade do software, especialmente quando se trata de aplicações que visam lidar com cargas significativas de utilizadores ou grandes volumes de dados. Esses testes procuram avaliar o desempenho do sistema em condições de uso intensivo e identificar possíveis gargalos e problemas de escalabilidade [22].

Existem várias estratégias para realizar testes de desempenho. Uma abordagem comum é a realização de testes de carga, onde o software é submetido a uma carga simulada, representando a atividade real do utilizador, para avaliar como o sistema se comporta sob pressão. Isso pode ser feito por meio de ferramentas que geram solicitações automatizadas e simulam o comportamento dos utilizadores num ambiente controlado [23].

Os testes de desempenho também podem envolver a medição de tempos de resposta, utilização de recursos do sistema, taxa de transferência de dados e outros indicadores-chave de desempenho. Isso permite que os desenvolvedores identifiquem gargalos e otimizem o código ou infraestrutura conforme necessário.

Além disso, os testes de desempenho podem ser realizados em diferentes fases do ciclo de vida do software, desde o desenvolvimento até a fase de produção. Isso permite que problemas de desempenho sejam identificados e corrigidos o mais cedo possível, reduzindo riscos e custos associados a problemas de desempenho em produção.

Ao conduzir testes de desempenho, é essencial estabelecer critérios de aceitação claros com base em requisitos de desempenho definidos. Isso ajudará a determinar se o software atende aos padrões de desempenho desejados e garantir que ele seja escalável e capaz de suportar um número crescente de utilizadores sem degradação significativa no desempenho.

Em suma, os testes de desempenho são fundamentais para garantir que o software funcione de maneira eficiente e confiável sob condições reais de uso. A identificação e correção precoce de problemas de desempenho são essenciais para fornecer uma experiência positiva aos utilizadores e garantir o sucesso do software no mercado. A utilização de ferramentas apropriadas e a definição de critérios de aceitação bem definidos contribuem para o sucesso dos testes de desempenho.

- **Testes de interface do utilizador**

  Os testes de interface gráfica são uma etapa essencial no processo de garantia da qualidade do software, especialmente no que diz respeito a aplicações que possuem interações diretas com os utilizadores. Esses testes têm como objetivo verificar se a interface do software atende aos requisitos de usabilidade, *design* e experiência do utilizador [24].

  Uma das estratégias comuns para realizar testes de interface gráfica é a automação de testes de Graphical User Interface (GUI). Ferramentas populares, como o Selenium, permitem simular interações do utilizador com a interface gráfica, verificando se os elementos da interface respondem adequadamente e se as funcionalidades são executadas corretamente [25].

  Os testes de interface gráfica também podem envolver verificações visuais, onde se compara a aparência atual da interface com imagens de referência para garantir a consistência visual em diferentes plataformas e resoluções.

  Além disso, é comum a utilização de testes de acessibilidade para verificar se a interface é facilmente utilizável, cumprindo normas e diretrizes de acessibilidade.

Esses testes de interface gráfica podem ser realizados em diferentes fases do ciclo de vida do software, desde o desenvolvimento até a fase de produção, para garantir que a interface atenda aos padrões de qualidade estabelecidos e proporcione uma experiência positiva ao utilizador.

Vantagens dos testes de interface gráfica incluem:

- Identificação precoce de problemas de usabilidade e *design* que poderiam impactar negativamente a experiência do utilizador.
- Automação de testes para acelerar o processo de validação de diferentes cenários de interação do utilizador.
- Garantia de conformidade com padrões e diretrizes de *design* e acessibilidade.
- Deteção de problemas de compatibilidade com diferentes dispositivos e resoluções de ecrã.

No entanto, também existem desvantagens, como a dificuldade em abranger todos os cenários possíveis de interação do utilizador e a necessidade de atualização constante dos testes para acompanhar as mudanças na interface e funcionalidades do software.

Casos de uso para testes de interface gráfica incluem:

- Testes de navegação: Verificam se os botões, menus e links direcionam o utilizador para as páginas corretas.
- Testes de entrada de dados: Testes de entrada de dados garantem que os campos de entrada de dados funcionem adequadamente e aceitem dados válidos e inválidos.
- Testes de resposta a eventos: Testes de resposta a eventos verificam como a interface responde a ações do utilizador, como cliques, toques ou arrastar e soltar.
- Testes de *layout* e *design*: Testes de *layout* e *design* asseguram que os elementos gráficos estão dispostos corretamente e possuem uma aparência consistente em diferentes dispositivos.

2.5.2 Processo

O processo de automatização de testes inicia-se com uma fase de avaliação e planeamento. Nesta fase, torna-se primordial identificar os casos de teste que são candidatos ideais para a automatização. A regra geral é optar por testes que se

repetem com frequência ou que exigem um grau elevado de precisão. Paralelamente, a seleção da ferramenta de automatização é conduzida com base nas especificidades do projeto e no ambiente técnico em que se insere [26].

Segue-se o desenvolvimento dos *scripts* de teste. Para tal, é imperativo definir e configurar o ambiente de teste, garantindo a instalação de todos os software necessários e as configurações de hardware pertinentes ao cenário de teste. Com o ambiente pronto, avança-se para a elaboração dos *scripts* de teste, utilizando a ferramenta previamente selecionada, para que estas ações automatizem os passos delineados nos casos de teste.

A execução dos testes automatizados é o passo subsequente. Estes podem ser programados para decorrer em momentos específicos, tornando-se particularmente úteis para testes de regressão ou aqueles que necessitam ser realizados com regularidade. Durante esta fase, a monitorização do progresso é essencial, de modo a identificar potenciais falhas ou anomalias.

Posteriormente, entra-se na fase de análise e elaboração de relatórios. Os resultados, após serem coletados, são comparados com as expectativas pré-definidas, permitindo determinar a aprovação ou reprovação do software em teste. Esta análise é meticulosamente documentada, gerando relatórios detalhados que evidenciam os resultados obtidos, as falhas detetadas, métricas relevantes e, claro, recomendações dirigidas à equipa de desenvolvimento.

Por último, mas não menos importante, é a fase de manutenção. O software, por natureza, está em constante evolução, o que exige que os *scripts* de teste sejam revistos e atualizados periodicamente. Assim, recomenda-se uma avaliação recorrente da suite de testes automatizados, garantindo que esta mantém a sua relevância e eficácia à luz das alterações no software e das necessidades empresariais.

2.5.3 Riscos e Desvantagens

A automatização dos testes requer de um investimento inicial de tempo e possivelmente de dinheiro para configurar e implementar as ferramentas de testes automatizados [27].

Muitas organizações excedem os seus orçamentos de desenvolvimento de software em geral, e, nesse sentido, não é surpreendente que o custo seja considerado um fator limitante para a automação [28].

Perceber que a automação precisa seguir um ciclo de vida empresarial razoável, em vez de obter os benefícios imediatamente, é uma condição importante para a automação bem-sucedida.

As características do produto e dos recursos em desenvolvimento são relevantes para a possibilidade de obter um retorno aceitável do investimento. Em produtos ou projetos pequenos e de curta duração, pode-se discutir se é possível obter um ROI aceitável antes do término do desenvolvimento.

A falta de tempo, pessoal e financiamento também pode levar à escolha do trabalho manual em vez da automação. Projetos sob pressão de tempo tendem a priorizar metas de curto prazo e negligenciar atividades importantes para o sucesso a longo prazo.

Ao considerar a automação, há relatos de escolha do trabalho manual em vez da automação, se o tempo for curto ou caso não haja recursos suficientes disponíveis para a automação. Isso permite que os *testers* testem imediatamente, em vez de estabelecer a infraestrutura e os *scripts* de teste. O resultado provavelmente será a acumulação de dívidas técnicas que causarão problemas posteriormente.

2.6 Front-end como sistema de testagem automatizada

O componente da aplicação que interage diretamente com o utilizador é chamado de *front-end*. Ele consiste em todos os elementos gráficos da interface do utilizador (GUI), páginas *web* e outros elementos de exibição que o utilizador pode ver e interagir. O *front-end* é geralmente construído recorrendo a tecnologias como HyperText Markup Language (HTML), Cascading Style Sheets (CSS) e JavaScript. Ele é responsável por lidar com a apresentação visual, interação do utilizador e colheita de dados. O objetivo principal do *front-end* é garantir que a aplicação seja *user-friendly* e envolvente para o utilizador, proporcionando a melhor experiência possível [29].

Conforme mencionado no subcapítulo 2.5.1, os testes automatizados abrangem uma variedade de tipos, com foco em diferentes aspetos do software. E é aqui que o *front-end* torna-se especialmente relevante. Quando pensamos nos testes de interface gráfica ou de aceitação, por exemplo, o *front-end* é o primeiro ponto de contato. Ele serve como um portal visual para a execução desses testes automatizados, permitindo aos testadores verificar se o software se comporta conforme o esperado em termos de interação do utilizador e exibição visual.

No mundo dos testes automatizados, o *front-end* não é apenas o rosto da aplicação; ele é a peça central na garantia de que o software proporciona uma experiência de

utilizador otimizada. Para testes de aceitação, o *front-end* desempenha um papel crucial ao representar os cenários do mundo real em que o utilizador interage com a aplicação. Mediante ferramentas automatizadas, os testadores podem simular interações do utilizador com o *front-end*, certificando-se que a aplicação se comporta conforme o esperado. Da mesma forma, para os testes de interface gráfica, o *front-end* é examinado detalhadamente para assegurar que todos os elementos visuais apareçam e funcionem conforme o *design* e as especificações.

No contexto da automação de testes, um *front-end* de automatização é uma interface que permite aos utilizadores criar, executar e gerir testes de automação de forma fácil e intuitiva. As principais vantagens da utilização de um *front-end* incluem facilidade de uso, aumento da produtividade e visibilidade aprimorada. Este tipo de *front-end* garante que o utilizador não precisa ter um conhecimento técnico profundo, permitindo a criação e execução de testes de forma mais rápida. Além disso, os resultados são apresentados de forma clara e organizada, facilitando a análise e identificação de eventuais problemas.

A visibilidade aprimorada que o *front-end* oferece é crucial. Os resultados dos testes são apresentados de forma gráfica e compreensível, permitindo que desenvolvedores e testadores identifiquem rapidamente falhas ou anomalias no software. Isso agiliza o processo de correção e aprimoramento contínuo do software. No entanto, é vital considerar a dependência de uma ferramenta específica para o *front-end* de automatização de testes. Se a ferramenta escolhida se tornar obsoleta ou não atender mais aos requisitos do projeto, poderá ser necessário fazer uma mudança significativa no processo de automação. Além disso, algumas ferramentas de *front-end* podem ter custos elevados, o que deve ser considerado em projetos com restrições orçamentárias.

2.7 Linguagens de Programação e Tecnologias Utilizadas

Nesta secção são abordadas as linguagens de programação bem como as tecnologias que se enquadram no desenvolvimento do sistema dimensionado do projeto. Com esta pesquisa procura-se adquirir um conhecimento maior de várias alternativas às tecnologias, conceitos ou linguagens utilizadas nas várias soluções já existentes.

2.7.1 Python

Python é uma linguagem de programação de uso geral interpretada, orientada a objetos e de alto nível. De acordo com o TIOBE Index, o Python está atualmente no segundo lugar na lista das linguagens de programação mais populares no mundo,

com uma taxa de crescimento contínuo. É amplamente utilizada em muitos campos, como automatização, desenvolvimento *web*, *data science* e inteligência artificial [30].

Esta linguagem de programação centra-se em tornar o código legível para o programador, por ser interpretativa, faz com que as instruções possam ser executadas direta e livremente, sem que previamente tenha de ser compilado o código em máquinas de baixo nível. Por ser uma linguagem de alto nível, não depende de detalhes do computador ou do microprocessador. Este software não lida com registos, endereços de memória e *call stack*, mas sim com variáveis, objetos e *arrays*.

O processo de desenvolvimento com Python é fácil e intuitivo, graças à sua sintaxe *clean* e direta, bem como à presença de uma ampla gama de bibliotecas e módulos. Python oferece também uma ampla comunidade de utilizadores, que fornece suporte, soluções de problemas e uma vasta documentação.

No contexto de automatização de testes, python é uma das linguagens de programação mais utilizadas. Isto deve-se, em parte, à presença de bibliotecas como o Robot Framework, que oferecem uma ampla gama de recursos e ferramentas para automatização de testes em aplicações Python.

2.7.2 Robot Framework

Robot Framework é um framework *open-source* de testes automatizados que permite escrever testes de forma clara, objetiva e concisa. É baseado em Python e é utilizado para realizar testes de aceitação, testes funcionais, testes de integração e testes de performance. Tem uma fácil adaptabilidade a inúmeras linguagens de programação, como Python ou Java, o que faz com que seja uma ferramenta completa e abrangente [31].

O processo de utilização do Robot Framework consiste em escrever *scripts* de teste em linguagem natural, utilizando uma estrutura de *keywords*, e depois executá-los de modo a verificar se as funcionalidades da aplicação estão a funcionar corretamente [32]. O Robot Framework permite também a criação de testes personalizados com base nas necessidades do projeto em causa, bem como a integração com outras ferramentas de teste, como Selenium e PyTest.

As *keywords* representam ações específicas que podem ser realizadas durante o teste, como clicar num botão ou verificar o conteúdo de uma página da *web*. Os testes escritos com Robot Framework são fáceis de entender e manter, tornando-os uma opção atraente para equipas que desejam aumentar a eficiência da sua automação de testes. Trata-se de uma tecnologia altamente personalizável, permitindo que as

equipas criem as suas próprias *keywords* para se adaptarem às suas necessidades específicas.

Como principal desvantagem do Robot Framework apresenta a sua maior lentidão de processamento do que outras ferramentas de teste, especialmente em casos de testes mais complexos.

RobotRemoteServer

O RobotRemoteServer é uma biblioteca, que consiste num servidor remoto do Robot Framework. Em vez de executar os casos de teste do Robot Framework no mesmo sistema, é possível usar o RobotRemoteServer para executá-los num ambiente remoto. O processo do RobotRemoteServer envolve a configuração do servidor, a inicialização e a conexão com o cliente de teste. Uma vez estabelecida a conexão, os casos de teste podem ser executados no servidor remoto como se estivessem a ser executados localmente. Desta forma, a execução de testes é feita em paralelo e a distribuição é simplificada.

2.7.3 HTML

O HTML é uma linguagem de marcação, amplamente utilizada na criação de páginas *web* [33]. A linguagem permite aos desenvolvedores criarem documentos que incluem texto, imagens, links, tabelas, formulários, áudio e vídeo, entre outros tipos de conteúdo. O HTML define a estrutura e o conteúdo de uma página da *web*, mas não é responsável do *design* da mesma.

As aplicações do HTML são diversas e incluem desde sites pessoais até grandes portais de notícias, lojas online e aplicações *web*. Além disso, o HTML também é utilizado para criar documentos estáticos, como folhetos e folhas de informações, que não requerem interação com o utilizador. Alguns dos exemplos mais conhecidos de sites criados com HTML incluem Google, Facebook, Amazon e Wikipedia.

O HTML tem como vantagem a facilidade de utilização e a disponibilidade de ferramentas de desenvolvimento de código gratuitas, como o bloco de notas, bem como a ampla disponibilidade de recursos online, como é o caso de tutoriais e fóruns de auxílio. Além disso, o HTML é amplamente compatível com a maioria dos navegadores da *web*, o que permite que os desenvolvedores criem aplicativos da *web* que possam ser acedidos por utilizadores em todo o mundo.

Como desvantagem, a linguagem de marcação tem a falta de recursos avançados de *design*, o que implica que os desenvolvedores *web* precisem de utilizar outras tecnologias, como o CSS, para melhorar a aparência das suas páginas da *web*. E

ainda, o HTML pode ser vulnerável a ataques de segurança se não for implementado de forma correta.

Como alternativas ao HTML, existem outras linguagens de marcação, como são o caso do Extensible Markup Language (XML) e do Extensible HyperText Markup Language (XHTML), que oferecem recursos adicionais aos desenvolvedores. O XML é uma linguagem de marcação que permite aos desenvolvedores criarem os seus próprios tipos de marcas, enquanto o XHTML é uma versão mais rigorosa e estruturada do HTML. Além disso, o CSS é uma tecnologia complementar que permite aos desenvolvedores controlarem a aparência das suas páginas da *web* sem afetar.

2.7.4 CSS

O CSS (*Cascading Style Sheets*) é uma linguagem de folha de estilo utilizada para a separação a apresentação da estrutura e do conteúdo de uma página *web* [33]. Em vez de especificar como a página deve ser apresentada no código HTML, o CSS permite que os desenvolvedores definirem estilos para serem aplicados a vários elementos da página. Esses estilos incluem, por exemplo, fontes, cores, margens, *padding*, bordas e posicionamento.

A capacidade de criar estilos de modelos de página que podem ser aplicados a múltiplos documentos graças ao CSS, tornam a manutenção de uma aparência unificada em todo o *website* mais fácil e mais eficaz [34]. Além disso, o CSS também permite aos programadores criar estilos únicos e personalizados que são exclusivos para cada página, conforme a necessidade.

A capacidade do CSS em separar a apresentação da estrutura e do conteúdo da página torna mais fácil para os programadores manterem e atualizarem o site. Para além de que, o CSS é altamente compatível com a maioria dos navegadores online, o que faz com que as páginas *web* criadas com CSS sejam acessíveis aos utilizadores em todo o mundo. O uso do CSS também pode levar a páginas *web* mais rápidas, já que os estilos podem ser carregados uma vez e aplicados a várias páginas, em vez de serem personalizados separadamente para cada página.

Contudo, também há algumas desvantagens ao usar CSS. Uma delas é a curva de aprendizagem. Os diferentes níveis de suporte entre navegadores podem tornar difícil para os desenvolvedores projetarem estilos que funcionem consistentemente em todos os navegadores.

Como alternativas ao CSS, existem outras tecnologias de estilo, como o Sass e o Less, que oferecem recursos adicionais aos desenvolvedores. O Sass é uma linguagem

de pré-processamento CSS que permite aos desenvolvedores escrever CSS de forma mais avançada e eficiente, enquanto o Less é uma linguagem de pré-processamento que permite aos programadores escrever CSS de forma mais organizada e eficiente. Além disso, o JavaScript é uma tecnologia complementar que permite aos desenvolvedores adicionar interatividade e efeitos visuais às páginas *web*. No entanto, o JavaScript não é uma linguagem de estilo, por isso, para controlar a aparência da página, os programadores ainda precisam recorrer ao CSS, como é o caso do presente projeto.

2.7.5 JavaScript

JavaScript é uma linguagem de *script*, frequentemente utilizada para fornecer recursos interativos e efeitos visuais às páginas *web* [35]. É uma linguagem de programação completa que permite aos desenvolvedores criar aplicações *web* dinâmicas, além de páginas *web* estáticas. Por ser uma linguagem de programação dinâmica, o tipo de dado usado em JavaScript pode mudar durante a execução do programa.

Para a utilização de JavaScript, é necessário adicionar o código JavaScript a uma página HTML. Uma vez adicionado, o código é então executado pelo navegador *web* do utilizador quando a página é carregada. Isto permite que o JavaScript adicione recursos interativos e efeitos visuais a uma página sem precisar de gerá-la novamente. É possível escrever JavaScript diretamente numa página HTML ou carregá-lo como um arquivo externo.

O processo de programação em JavaScript é simplificado por possuir uma sintaxe intuitiva e considerada de fácil aprendizagem [36]. Ao ter uma grande diversidade de bibliotecas e *frameworks* disponíveis, faz com que o precesso de desenvolvimento de aplicações seja ainda mais simples.

Como benefícios, o recurso a JavaScript permite a criação de aplicações online interativas e dinâmicas, o que é fundamental para uma experiência positiva do utilizador. Além disso, é uma linguagem de programação de propósito geral, o que significa que pode ser usada para desenvolver aplicações que não são baseadas na *web*, como aplicações em computador e em dispositivos móveis. A sua ampla disponibilidade, já que é suportado por quase todos os navegadores *web*, além de ser de código aberto. Outra vantagem é a sua capacidade de se integrar com outras tecnologias *web*, como HTML, CSS e Application Programming Interface (API)'s, o que torna o desenvolvimento de aplicativos *web* mais fácil e rápido.

Uma desvantagem do JavaScript é que é vulnerável em termos de segurança. *scripts* JavaScript maliciosos podem ser facilmente inseridos em páginas online, o

que pode prejudicar o utilizador final. Além disso, o desempenho de aplicações JavaScript pode ser restrito em dispositivos com recursos limitados, e assim a sua performance ser afetada.

Como alternativas, pode-se considerar outras linguagens de programação como Python, Ruby ou PHP. No entanto, estas linguagens não oferecem as mesmas vantagens e recursos que o JavaScript oferece para desenvolvimento de aplicações *web* interativas e dinâmicas.

2.7.6 Flask

Flask é uma *framework web* escrita em Python, que se trata de uma biblioteca de software livre do lado do servidor que facilita e agiliza a criação de aplicações *web*, para o arranque das mesmas [37].

O desenvolvimento de aplicações que utilizam Flask iniciam com a definição da estrutura fundamental da aplicação, que inclui a configuração do servidor, a definição de rotas e a criação de funções de retorno de chamada para lidar com solicitações HTTP [38]. O programador pode então adicionar recursos, como autenticação, arquivamento de dados e integração com outras bibliotecas.

Flexibilidade e facilidade de utilização são as principais vantagens do Flask, permite aos programadores escolher as bibliotecas e recursos que desejam utilizar, em vez de ficarem remetidos a uma grande quantidade de tecnologias pré-definidas. Apesar disso, o Flask tem uma escala limitada para programas de tamanho pequeno e médio, pode dificultar a sua manutenção e escalar à medida que o programa cresce de tamanho, pelo que não é a melhor opção para aplicações de grande escala que precisam de muitos recursos.

Como alternativas ao Flask existem o Django, Ruby on Rails e Express.js. Estes *frameworks* são mais completos e oferecem recursos mais avançados, no entanto, também são mais complexos e com uma curva de aprendizagem consideravelmente maior [38].

2.7.7 YAML Ain't Markup Language (YAML)

YAML, que significa "YAML Ain't Markup Language", é uma linguagem de serialização legível por humanos, frequentemente usada para escrever configurações ou dados que necessitam de ser compreendidos e eventualmente modificados por humanos. A simplicidade e flexibilidade do YAML tornaram-no uma escolha popular para arquivos de configuração em ferramentas de desenvolvimento e operações [39].

Distingue-se pela sua simplicidade, e ao contrário de outras linguagens de serialização como XML ou JavaScript Object Notation (JSON), o YAML não requer delimitadores como chavetas ou *tags*. Em vez disso, usa-se indentação para representar a estrutura e dois-pontos para pares chave-valor. Esta natureza minimalista não só facilita a leitura, mas também a escrita, tornando-a uma ferramenta favorita para configurações que podem necessitar de ajustes manuais.

No contexto do projeto atual, arquivos YAML são utilizados extensivamente para definir a configuração dos servidores e suas especificidades. Por exemplo, o arquivo servers_CI_FoD_station1.yaml define vários servidores, como *androidDevices* ou a *webcam*. Cada entrada no arquivo YAML fornece detalhes específicos, como argumentos necessários para iniciar o servidor, o caminho para o arquivo binário, o método de inicialização e se o servidor deve ser iniciado automaticamente. Este arquivo de configuração serve como um único ponto de referência para todas as configurações dos servidores, permitindo uma gestão centralizada e fácil modificação conforme necessário.

Além da sua utilização direta no projeto, muitas ferramentas modernas, como Docker Compose, Kubernetes e Ansible, dependem de arquivos YAML para suas configurações, demonstrando a ampla aplicabilidade e relevância desta linguagem de serialização no mundo do desenvolvimento e operações.

2.7.8 PowerShell Scripting

PowerShell é uma linguagem de *script* e uma interface de linha de comando baseada em tarefas, desenvolvida pela Microsoft [40]. Embora o PowerShell em si seja uma ferramenta poderosa e extensível, o denominado"PowerShell scripting", foca na criação de *scripts* ou pequenos programas escritos na linguagem de *script* do PowerShell.

A capacidade de criar *scripts* com o PowerShell oferece aos desenvolvedores uma ferramenta flexível para automatizar processos, gerir sistemas e realizar operações de maneira programada. A sua sintaxe intuitiva e o acesso direto a uma ampla gama de *comandlets* (comandos especializados do PowerShell) tornam mais simples a realização de tarefas que, de outra forma, seriam complexas ou trabalhosas [41].

No contexto deste projeto, o PowerShell *script* foi utilizado para auxiliar na escrita e gestão de determinadas operações. Esta escolha foi motivada pela facilidade de integração do PowerShell com outras tecnologias e plataformas, bem como pela sua capacidade de executar tarefas de forma rápida e eficiente.

Um aspeto notável do PowerShell *script* é sua capacidade de interagir diretamente com objetos .NET, permitindo que os *scripts* façam uso de classes e métodos

disponíveis na *framework* .NET. Além disso, sua interoperabilidade com outras linguagens e sistemas o torna uma opção viável para ambientes mistos, onde diferentes tecnologias e plataformas precisam trabalhar em conjunto.

Em resumo, o uso de *scripts* PowerShell no projeto não apenas facilitou certos aspetos do desenvolvimento, mas também reforçou a abordagem modular e automatizada adotada na solução.

2.7.9 Markdown

Markdown é uma linguagem de marcação leve criada por John Gruber, em parceria com Aaron Swartz, em 2004. Desde o seu lançamento, tem sido amplamente adotada por escritores, programadores e profissionais em áreas diversas, devido à sua simplicidade e legibilidade [42].

Markdown foi projetado para ser facilmente convertido em HTML. Ele usa caracteres de pontuação e símbolos frequentemente encontrados em textos simples, como asteriscos, cerquilha (ou "hash") e traços, para denotar diferentes níveis de cabeçalho, ênfase, listas e outras formatações.

Dentre as principais características do Markdown, destacam-se:

- **Legibilidade**: O código Markdown é facilmente legível e editável, mesmo sem conversão para HTML.
- **Simplicidade**: A sintaxe de Markdown é intencionalmente minimalista, requerendo um período curto de aprendizado.
- **Flexibilidade**: Permite a incorporação de HTML, caso funcionalidades mais avançadas sejam necessárias.

Com a ascensão de plataformas como o GitHub, GitLab, entre outras, Markdown tornou-se uma ferramenta padrão para a criação de READMEs, documentação de projetos e até mesmo blogs. O motivo de sua popularidade é a capacidade de escrever conteúdo formatado de forma rápida e fácil, sem a necessidade de se preocupar com a complexidade do HTML.

Além disso, várias ferramentas de conversão foram criadas para transformar Markdown em vários formatos, como PDF, Word e, claro, HTML. Estas ferramentas ampliam ainda mais a utilidade do Markdown, tornando-o uma opção versátil para escrita digital.

No contexto moderno de desenvolvimento e documentação, Markdown destaca-se como uma linguagem de marcação de escolha para muitos. A sua simplicidade

e eficácia no que diz respeito à criação de conteúdo formatado de forma limpa e legível garantem que continue a ser uma ferramenta valiosa para profissionais em várias áreas.

No que diz respeito ao projeto, a capacidade do Markdown de converter texto simples em HTML bem formatado tornou-se inestimável. Em particular, utilizamos Markdown para criar e manter a documentação da aplicação, garantindo que informações essenciais sobre o sistema estejam facilmente acessíveis e legíveis tanto para desenvolvedores quanto para utilizadores finais.

A aplicação *web*, que é intensamente centrada em fornecer informações claras e concisas aos seus utilizadores, beneficia da simplicidade do Markdown. Por exemplo, a página de documentação, que converte ficheiros '.md' diretamente em HTML, demonstra a capacidade do Markdown de criar conteúdos visuais ricos a partir de uma sintaxe simples e direta. A integração dessa funcionalidade não só melhorou a eficiência na criação de conteúdo, mas também aprimorou a experiência geral do utilizador ao interagir com a plataforma.

2.7.10 Git

Git é um sistema de controlo de versões (VCS) distribuído, gratuito e open source, concebido para lidar com projetos independentemente do tamanho dos mesmos, com rapidez e eficiência [43]. É amplamente utilizado por equipas de desenvolvimento de software para garantir a integridade e a colaboração no código.

Um sistema Version Control System (VCS) monitora o histórico de alterações à medida que os utilizadores e equipas colaboram em conjunto nos respetivos projetos. Os programadores, ao procederem a alterações no projeto, têm a possibilidade de recuperar qualquer versão anterior do projeto. Num sistema de versão de controlo distribuído, cada *developer* tem uma cópia completa do projeto e do seu histórico, sendo que o git não precisa de uma conexão constante com um repositório central.

Um repositório abrange toda a coleção de arquivos e pastas associados a um projeto, com o respetivo histórico de revisão de cada arquivo. O histórico de arquivos aparece instantaneamente através dos commits. Estes podem ser organizados em várias linhas de desenvolvimento denominadas *branches*. Por meio de plataformas como GitLab ou GitHub, o Git através de repositórios públicos oferece mais oportunidades para a transparência e colaboração de projetos, e assim ajudam equipas a cooperar para criar o melhor produto final possível.

O GitLab é uma plataforma de desenvolvimento de software que fornece ferramentas de controlo de versão, gestão de projetos, construção de pipelines de Continuous Integration/Continuous Deployment (CI/CD) e testes de automação integrados [44]. É baseado em Git e fornece uma interface *user-friendly* amigável para gerir projetos.

Git e GitLab podem ser integrados com testes de automatização para garantir a qualidade do código. Os pipelines de CI/CD podem ser configurados para executar automaticamente os testes de automatização sempre que houver uma alteração no código. Isto permite que os desenvolvedores detetem rapidamente problemas de qualidade e os corrijam antes de serem implementados em ambiente de produção.

Neste projeto, a utilização do Git, e em particular da plataforma GitLab, revelou-se fundamental para uma gestão eficiente e organizada. Dada a natureza dinâmica e multifacetada do desenvolvimento, era imperativo trabalhar em várias atualizações e componentes em simultâneo. O GitLab, com a sua abordagem baseada em *branches*, ofereceu a capacidade de segmentar e isolar distintamente cada aspeto do trabalho. Assim, enquanto os testcases eram aprimorados numa *branch* específica, outra *branch* era dedicada exclusivamente à documentação, e uma terceira focava nas *tags*. Esta organização permitiu maximizar a eficiência ao prevenir conflitos indesejados e fornecer uma visão clara do progresso de cada componente individualmente. A integração contínua no GitLab também foi valiosa: a cada push ou merge request, os testes eram executados, assegurando que as atualizações não introduzissem erros ou problemas não previstos. Em resumo, o Git e o GitLab desempenharam um papel importante, proporcionando uma estrutura bem organizada e uma metodologia de trabalho fluida, sendo determinantes para o êxito do projeto.

2.7.11 Jira

O Jira é um software que serve de ferramenta de suporte, onde é possível fazer o monitoramento de tarefas, acompanhamento de projetos e gestão de defeitos, garantindo a monitorização e controlo de todas as atividades num único lugar [45]. Um projeto Jira é uma sequência de pontos, em que equipas utilizam um, para, por exemplo, coordenar o desenvolvimento de um produto, acompanhar um projeto ou gerir um *helpdesk*. Um projeto Jira também pode ser configurado e personalizado consoante as suas necessidades pelas equipas.

O processo de utilização do Jira começa com a criação de uma estrutura de projeto, incluindo a definição de tarefas, histórico do utilizador, *bugs* e problemas. Os membros da equipa podem então criar e atribuir tarefas, adicionar comentários e acompanhar o progresso através de gráficos, quadros e relatórios personalizados.

Além disso, o Jira permite a integração com outras ferramentas de desenvolvimento, como o Git, para garantir a continuidade da equipa e a eficiência do processo.

Para ajudar a equipa a acompanhar os resultados de teste e garantir a qualidade do programa, o Jira pode ser integrado com ferramentas de teste automatizados, como o Selenium. A integração permite às equipas acompanhar facilmente quaisquer problemas que possam afetar o projeto e ver os resultados dos testes.

No âmbito deste projeto, o Jira desempenhou um papel crucial na organização e monitorização das atividades. Através dele, foram mapeadas e categorizadas todas as tarefas, estabelecendo prazos, estimando durações e definindo prioridades. Esta ferramenta possibilitou uma visão clara e estruturada do que era desenvolvido, permitindo identificar de imediato os pontos em progresso, aqueles que aguardavam ação e os já concluídos. Além disso, cada tarefa era acompanhada de comentários relevantes, que ofereciam direcionamentos, *feedbacks* ou clarificações necessárias. Estes apontamentos foram essenciais para garantir que o desenvolvimento fosse alinhado com as expectativas e necessidades identificadas. Em resumo, o Jira não só assegurou a organização e o rastreamento eficaz das atividades, como também proporcionou uma comunicação fluída e transparente, fundamental para o sucesso e pontualidade das entregas do projeto.

2.8 ETAF

O ETAF é uma *framework* totalmente desenvolvida pela Capgemini Portugal com o propósito de simplificar e otimizar a automação de testes, reunindo ferramentas numa única plataforma. A ferramenta procura fornecer uma solução eficiente para a criação e execução de testes, maximizando tanto a qualidade quanto a produtividade. A criação do ETAF foi motivada pela identificação da necessidade de desenvolver um *framework* para testes automatizados que pudesse integrar, de forma simplificada, um conjunto de ferramentas que até então eram muito difíceis de automatizar em conjunto.

A *framework* permite a integração com várias tecnologias e plataformas para atender às necessidades específicas de cada projeto. Com o ETAF, é possível melhorar a eficiência dos testes, simplificando a automação de testes e permitindo a execução de testes em cenários mais complexos.

O ETAF é desenvolvido inteiramente com software *open-source*, promovendo a colaboração, além de trazer benefícios como personalização, integração com outras ferramentas e redução de custos com licenças de software. O uso de software de código *open-source* no desenvolvimento do ETAF também permite um maior controlo

sobre o *framework*, permitindo que seja personalizado conforme necessário. Esta *framework* utiliza tecnologias modernas e populares no mercado. A plataforma é desenvolvida em Python, que permite uma programação ágil e intuitiva, com muitas bibliotecas disponíveis, o que facilita o desenvolvimento de testes automatizados. O Python ajuda a tornar o *framework* capaz de testar uma variedade de aplicações, como sistemas embebidos, aplicações *desktop*, *web* e móveis.

Esta ferramenta é capaz de se integrar facilmente à integração contínua (CI), ao utilizar ferramentas como o Jenkins e plataformas como o GitLab. Isso permite a execução automatizada de testes, agiliza a deteção de problemas de qualidade e simplifica a gestão dos testes via pipelines Continuous Integration (CI).

O ETAF é orientado para uso com o Robot Framework, o que permite a criação de testes automatizados numa linguagem simples e fácil de entender. O Robot Framework é altamente extensível e suporta vários tipos de testes, como testes unitários, testes de integração, testes de sistema e outros.

O ETAF utiliza os resultados gerados pelo Robot Framework nos formatos HTML e XML para fornecer relatórios detalhados sobre o *status* dos testes executados. Esses relatórios incluem informações como casos de teste aprovados e falhados e permitem a análise de métricas como tempo de execução e cobertura de código.

No contexto dos sistemas *automotive* e embebidos, o ETAF oferece capacidades distintas que são especialmente vantajosas. Estes sistemas, muitas vezes, têm requisitos rigorosos de desempenho, segurança e confiabilidade, dado o seu uso em ambientes críticos e em aplicações onde falhas podem ter consequências graves. Dessa forma, a necessidade de testes precisos, abrangentes e repetíveis é crucial. A flexibilidade do ETAF, aliada à sua capacidade de integração com múltiplas ferramentas e plataformas, facilita a automação de testes em cenários complexos típicos de sistemas embebidos. Por exemplo, a possibilidade de simular e testar interações em tempo real, validar interfaces de hardware e software e certificar-se de que os sistemas respondem adequadamente sob condições variadas. A integração facilitada com o Robot Framework permite que esses testes sejam escritos de forma clara e compreensível, assegurando que os requisitos específicos do domínio *automotive* e embebido sejam devidamente atendidos. Assim, o ETAF emerge como uma solução robusta e confiável para garantir que sistemas *automotive* e embebidos operem com o mais alto padrão de qualidade e confiabilidade.

Em suma, o sistema é composto por várias bibliotecas desenvolvidas em Python. Essas bibliotecas são as interfaces que possibilitam a automatização de certas aplicações, por exemplo, uma biblioteca pode disponibilizar a automação de uma aplicação

desktop exclusiva para o projeto onde o ETAF está a ser utilizado. Essas bibliotecas são disponibilizadas por meio de servidores num modelo cliente-servidor, geridos pelo ETAF, que atua como um orquestrador, conectando automaticamente os servidores e inicializando os testes implementados no Robot Framework. O funcionamento pode ser visto na Figura 2.6.

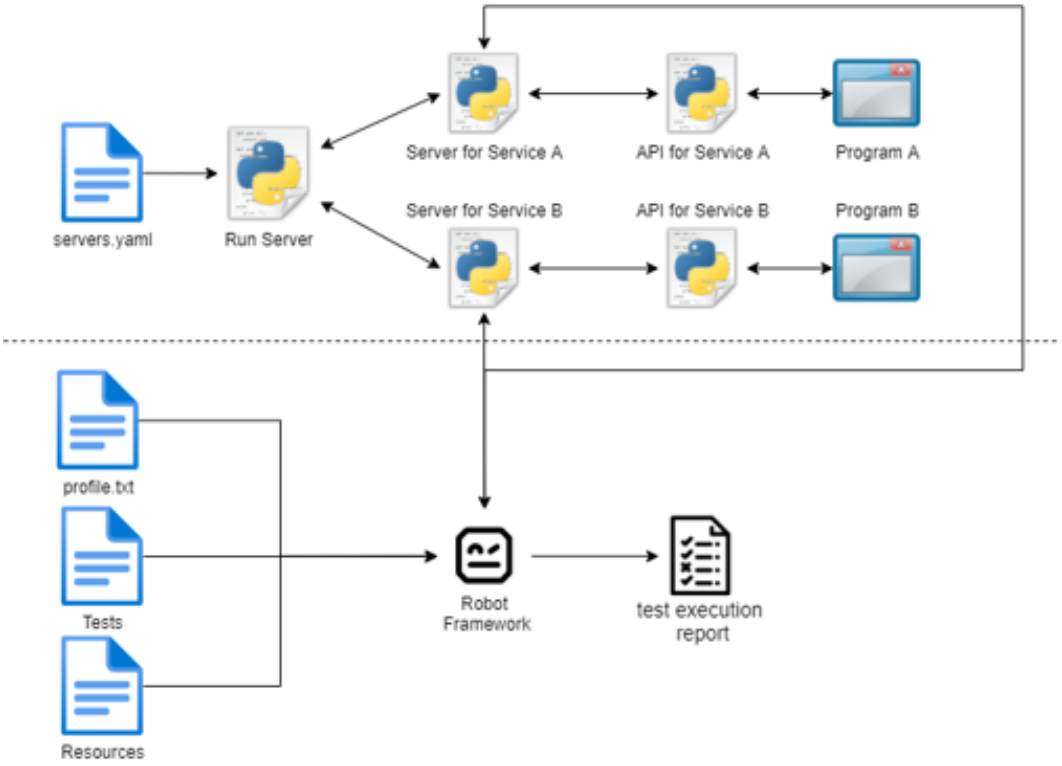


Figura 2.6: Estrutura base do ETAF.

2.9 Alternativas existentes ao ETAF

À medida que os sistemas embarcados se tornam cada vez mais complexos e integrados em diversos setores industriais, especialmente no automóvel, a necessidade de ferramentas robustas e eficientes para teste e validação tornou-se premente. A integridade e confiabilidade desses sistemas são cruciais não apenas para a funcionalidade do produto final, mas também para a segurança e satisfação do utilizador [46]. Dada esta importância, várias soluções foram desenvolvidas ao longo dos anos, procurando abordar os desafios inerentes à automação e gestão de testes em sistemas embarcados. Neste contexto, além do ETAF, ferramentas como o ECU-Test e CANoe surgiram como referências no mercado, oferecendo um conjunto de funcionalidades que atendem às exigências específicas desta área.

2.9.1 ECU-Test

Desenvolvido pela TraceTronic, o ECU-Test é uma ferramenta especializada para a automação e gestão de testes para ECUs [47]. Esta ferramenta tem ganho destaque, nomeadamente no setor automóvel, devido à sua abordagem inovadora e flexibilidade.

Para utilizar o ECU-Test, inicia-se o processo configurando os ficheiros tbc (*test bench configuration*) e tcf (*test configuration file*). O ficheiro tbc é essencial para definir as configurações de APIs que serão utilizadas, enquanto o ficheiro tcf estabelece parâmetros específicos, como as Electronic Control Unit (ECU) em foco e outras configurações pertinentes. Uma vez configurados estes ficheiros, os utilizadores podem proceder à criação de um projeto. Neste contexto, são estabelecidos os parâmetros de teste e integram-se modelos e simulações conforme a necessidade. A ferramenta ECU-Test destaca-se pela sua vasta gama de bibliotecas e *templates*, que auxiliam significativamente na configuração e definição dos cenários de teste. Uma vez que os cenários estão estabelecidos, os testes podem ser iniciados. Durante esta etapa, o ECU-Test monitoriza de forma contínua e em tempo real a ECU, capturando e registando todos os dados e atividades. Após a conclusão dos testes, a ferramenta apresenta os resultados de forma clara e organizada, simplificando a fase de análise e interpretação dos dados.

O ECU-Test apresenta como principais características:

- Testes Multinível: O ECU-Test permite a execução de testes em todos os níveis de integração, desde testes unitários até testes de sistema, garantindo uma cobertura abrangente.
- Integração com Ferramentas Externas: A capacidade de se integrar com várias ferramentas de desenvolvimento e teste, como MATLAB/Simulink e ASCET, permite uma abordagem abrangente e eficiente para a automatização de testes.
- Simulação Avançada: Permite a criação de cenários de teste complexos, simulando diferentes situações que um ECU pode enfrentar no mundo real.
- Regressão e Teste de Carga: A ferramenta é projetada para identificar rapidamente regressões em atualizações de software e para testar a resistência e durabilidade do software sob condições extremas.

A ferramenta apresenta diversas vantagens, tais como:

- Interface Intuitiva: A interface gráfica do utilizador é projetada para ser clara e fácil de usar, permitindo que os engenheiros configurem e executem testes com mínimo esforço.

- Configuração Flexível: A ampla capacidade de configuração permite adaptar-se a diferentes cenários e necessidades, tornando a ferramenta versátil para diversos projetos.
- Integração Contínua: A ferramenta pode ser facilmente integrada em pipelines de CI/CD, facilitando a automatização de testes em ambientes de desenvolvimento *agile*.
- Reutilização de Testes: A capacidade de reutilizar cenários de teste e configurações em diferentes projetos reduz significativamente o esforço de testes e garante consistência.

Contudo, existem diversas desvantagens na escolha deste software:

- Investimento Inicial: Pode ser necessário um investimento inicial considerável, não apenas financeiramente, mas também em termos de tempo, formação e adaptação ao ambiente de trabalho.
- Limitações de Personalização: Por ser uma ferramenta proprietária, não oferece a mesma liberdade de personalização que ferramentas *open-source*.

2.9.2 CANoe

O CANoe, desenvolvido pela Vector Informatik, é uma ferramenta abrangente projetada para o desenvolvimento, teste e análise de sistemas de comunicação em redes automóveis [48].

Para iniciar o trabalho com o CANoe, os utilizadores devem configurar a rede, definindo todos os intervenientes e os seus respetivos comportamentos. Através da interface, pode-se observar o tráfego de rede em tempo real, aplicar cargas, simular falhas e analisar o comportamento dos dispositivos. Uma característica distintiva do CANoe é a capacidade de simular a ausência de determinados dispositivos, permitindo testar o comportamento da rede em situações adversas. O ambiente de scripting permite a automação deste processo, proporcionando repetibilidade e robustez nos testes.

As principais características do CANoe são [49]:

- Análise Multimodal: Além de suportar protocolos padrão como Controller Area Network (CAN) e Local Interconnect Network (LIN), a ferramenta também suporta análises em protocolos mais recentes como FlexRay e Ethernet, tornando-se ideal para sistemas modernos.

- Simulação Realista: A capacidade de simular ECUs e o tráfego de rede permite testes em condições muito próximas às reais, sem necessidade de hardware físico.
- *Scripting* Avançado: O ambiente de *script* incorporado permite a criação de cenários de teste personalizados, automação e análises específicas.

A ferramenta apresenta, um leque considerável de vantagens, por exemplo:

- Interface Abrangente: O CANoe é conhecido pela sua interface rica em recursos que apresenta informações detalhadas de forma objetiva e organizada.
- Suporte a Protocolos Múltiplos: A sua capacidade de trabalhar com uma variedade de protocolos de comunicação torna o CANoe versátil.
- Diagnóstico Avançado: As capacidades de diagnóstico e deteção de falhas do CANoe são inigualáveis, permitindo identificar problemas rapidamente.
- Atualizações Regulares: A Vector Informatik oferece atualizações frequentes para a ferramenta, garantindo que ela esteja sempre alinhada com as últimas tendências e tecnologias da indústria.

O software desenvolvido pela Vector Informatik, tem como principais desvantagens:

- Curva de Aprendizagem: Devido à sua riqueza de recursos e capacidades, pode haver uma curva de aprendizagem íngreme para novos utilizadores.
- Custo e Licenciamento: Por ser uma ferramenta proprietária pode torná-la cara, e o modelo de licenciamento pode não ser ideal para todos os projetos, mesmo no contexto automóvel.

2.9.3 Comparação entre ETAF, ECU-Test e CANoe

Ao considerarmos o ETAF, ECU-Test e CANoe, é evidente que cada ferramenta foi projetada para servir propósitos ligeiramente diferentes, mas todas se concentram na garantia da qualidade dos sistemas embarcados e na eficiência da automação de testes.

O ETAF, sendo uma *framework open-source* e desenvolvida em Python, apresenta uma flexibilidade e adaptabilidade superiores. O seu foco na integração de diversas ferramentas sob uma única plataforma é uma das suas principais forças, além de conseguir editar livremente configurações dos servidores de APIs ao contrário das outras duas opções.

Já o ECU-Test é especificamente projetado para testes de sistemas eletrónicos automóveis, oferecendo uma profundidade e precisão consideráveis neste domínio. A ferramenta combina eficiência com facilidade de uso, permitindo a execução de testes abrangentes sem um esforço considerável.

Por outro lado, o CANoe ao mesmo tempo que é uma ferramenta poderosa para testes, destaca-se ainda mais pelas suas capacidades de análise e diagnóstico de redes automóveis e sistemas de comunicação. É uma solução de referência no seu nicho e é amplamente utilizado pela indústria.

Em suma, a escolha da ferramenta mais adequada dependerá dos requisitos específicos do projeto, do ambiente em que será utilizada e das preferências e competências da equipa de desenvolvimento e teste.

Capítulo 3

Trabalho desenvolvido

Nesta secção é demonstrado todo o processamento realizado no projeto. São descritos os requisitos, a arquitetura do sistema, o trabalho realizado e o funcionamento de todo o projeto.

3.1 Requisitos

Para que seja possível a utilização do trabalho desenvolvido, são utilizados diferentes recursos, nomeadamente a nível de software. São necessários alguns requisitos, nomeadamente é necessário ter instalado as seguintes tecnologias:

- **Git e Sistema de Controlo de Versões:** Uma conta num sistema de controlo de versões é fundamental. Neste projeto, foi utilizado o GitLab, por ser o utilizado pela empresa, mas outros sistemas como GitHub ou Bitbucket poderiam ser alternativas. Estas ferramentas são cruciais para rastrear mudanças, colaborar com outros desenvolvedores e manter um histórico das versões do código.
- **Navegador *web*:** Para aceder à interface da aplicação *web* gerada pelo Flask, é indispensável ter um navegador *web* moderno e atualizado. Recomendam-se browsers como o Google Chrome, Mozilla Firefox ou o Safari. Estes navegadores garantem que todas as funcionalidades da aplicação sejam apresentadas corretamente e que o desempenho seja otimizado.

- **Python:** O núcleo da aplicação é escrito em Python, uma linguagem de programação versátil e amplamente adotada em desenvolvimento *web*, entre outras áreas. A versão do Python utilizada durante o desenvolvimento foi a 3.9, mas versões subsequentes deverão ser compatíveis. No entanto, é sempre recomendado testar a aplicação em ambientes diferentes. Além do interpretador Python, são necessários os seguintes pacotes:
  - click 8.1.3;
  - colorama 0.4.6;
  - Flask 2.2.2: Essencial para criar e executar a aplicação *web*, responsável por servir a página e lidar com pedidos do cliente;
  - importlib-metadata 6.0.0;
  - itsdangerous 2.1.2;
  - Jinja2 3.1.2: Utilizado pelo Flask para renderizar templates HTML e integrar variáveis Python no *front-end*.
  - Markdown 3.4.4: Permite que ficheiros .md sejam convertidos e visualizados diretamente como HTML. Isto é especialmente útil para a página de documentação;
  - MarkupSafe 2.1.1;
  - numpy 1.25.2;
  - opencv-python 4.7.0.68;
  - pip 22.3.1;
  - psutil 5.9.5;
  - PyYAML 6.0: fundamental para ler e escrever ficheiros .yaml, que são usados para configurações, como a definição dos servidores;
  - robotframework 6.1.1: ferramenta utilizada para os ficheiros de testes automatizados;
  - robotremoteserver 1.1.1;
  - ruamel.yaml 0.17.21;
  - ruamel.yaml.clib 0.2.7;
  - setuptools 65.5.0;
  - typing extensions 4.4.0;
  - Werkzeug 2.2.2: biblioteca WSGI que fornece funcionalidades para o Flask;
  - zipp 3.11.0.

- **Ambiente Virtual Python (opcional):** Para garantir que a aplicação rode num ambiente isolado e que não ocorram conflitos entre versões de pacotes, pode-se optar pela utilização de um ambiente virtual Python, como o denominado 'venv' ou 'virtualenv'. Este passo, apesar de não ser obrigatório, é altamente recomendado, especialmente em fases de desenvolvimento. Portanto foi assim utilizado ao longo do projeto.

3.2 Arquitetura do Sistema

A arquitetura do projeto foi delineada com vista a assegurar uma organização clara e modular. Esta estruturação não só facilita o entendimento do sistema como também permite uma escalabilidade e manutenção eficientes.

3.2.1 Visão Geral da Arquitetura

A Figura 3.1 oferece uma visão geral da arquitetura do sistema, ilustrando as principais interações entre os componentes do ETAF.

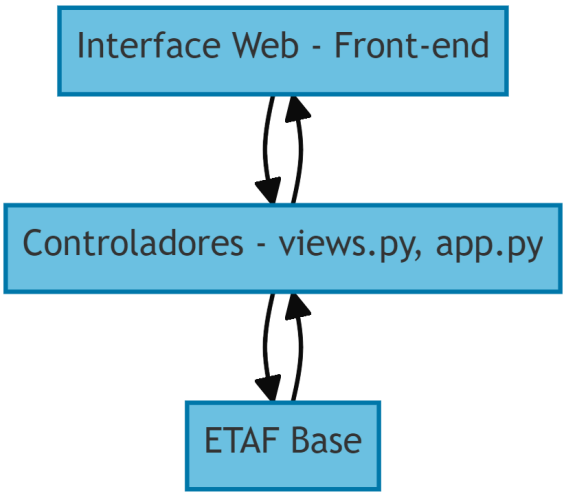


Figura 3.1: Visão Geral da Arquitetura do Sistema.

O diagrama destaca três componentes fundamentais:

- **Interface *Web* - *front-end***: Este é o ponto de contato inicial do utilizador com o sistema, onde as interações ocorrem num ambiente gráfico.
- **Controladores**: Atuam como intermediários, processando as solicitações do *front-end* e interagindo com a base de dados.
- **ETAF Base**: É a camada de persistência onde os dados são armazenados e recuperados.

Na Figura 3.2, é apresentada uma descrição detalhada da estrutura do repositório, de ressalvar que o diretório 'front-end' contém os ficheiros responsáveis pela interface *web*, além dos controladores:

Figura 3.2: Estrutura base do ETAF.

front-end: É o diretório responsável por albergar todos os arquivos que possibilitam a configuração dos parâmetros do ETAF diretamente da interface do utilizador. Esta pasta é essencial para que se possa adaptar o sistema às necessidades variáveis sem mergulhar no código base.

libraries: Este diretório contém as APIs necessárias para o funcionamento do sistema. Cada API está acompanhada dos seus respetivos ficheiros "...api.py"e "...server.py", que possibilitam a sua integração e comunicação com os demais componentes do sistema.

logs: Todas as execuções no sistema geram registos que são essenciais para a depuração, monitorização e rastreio de potenciais problemas. Estes registos são automaticamente guardados neste diretório após cada execução.

profiles: Contém ficheiros de configuração que especificam diferentes perfis de execução. Estes ficheiros definem variáveis, caminhos, *tags* e outras opções essenciais para a execução dos testes. É o módulo do Robot Framework que interpreta e processa estas configurações durante a execução dos testes.

resources: Aloja ficheiros robot, que são usados como recursos em vários test cases. Eles servem como uma camada de abstração, evitando repetições e tornando o código de teste mais modular e legível.

results: Após a realização dos testes, os resultados são gerados em formato HTML e XML. Esta pasta organiza e armazena estes ficheiros, oferecendo uma visão detalhada do desempenho e dos resultados dos testes.

scripts: Este diretório contém *scripts*, nomeadamente em PowerShell, que são utilizados para automatizar a execução dos testes. Embora os parâmetros estejam

previamente definidos, eles podem ser alterados através do *front-end* para adaptar-se às necessidades variáveis.

servers: Aqui estão guardados os ficheiros .yaml que contêm configurações para as diferentes bibliotecas, como a decisão sobre se um servidor particular deve ser iniciado ou não. Estas configurações podem também ser ajustadas através do *front-end*.

tests: O diretório 'tests' é o coração da lógica de teste. Ele contém os ficheiros robot que definem os testes a serem realizados, garantindo que o sistema funciona conforme esperado.

run_tests.py: É o ficheiro executável, geralmente invocado pelo *script* PowerShell, responsável por orquestrar e iniciar a execução dos testes, com os devidos parâmetros definidos pelo mesmo. No fluxograma da Figura 3.3 é descrito o seu funcionamento

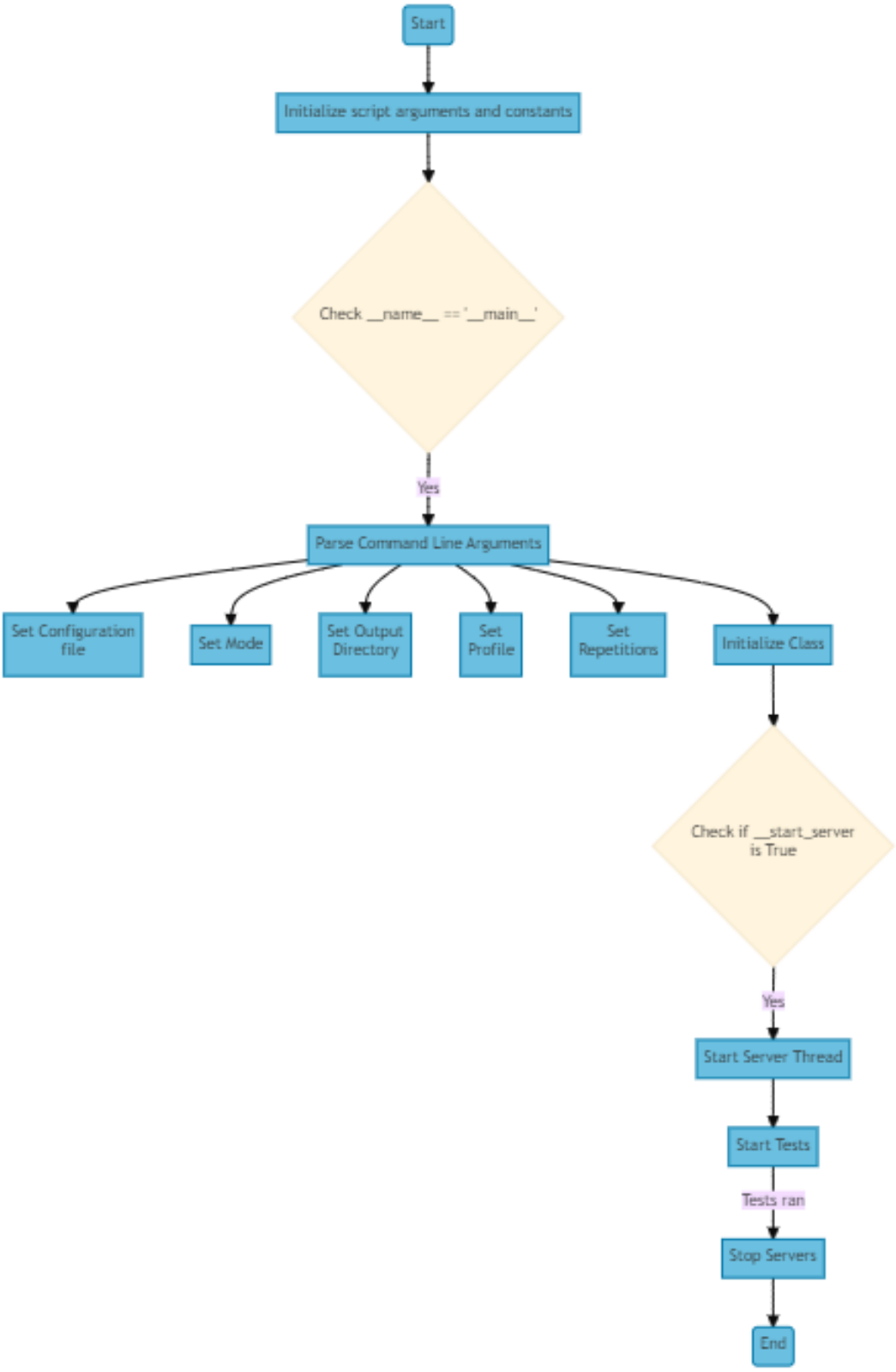


Figura 3.3: Fluxograma do run_tests.py

3.2.2 Estrutura do diretório *front-end*

O diretório *front-end* desempenha um papel crucial no projeto ao permitir a configuração interativa do sistema. Detalhadamente os componentes mais relevantes:

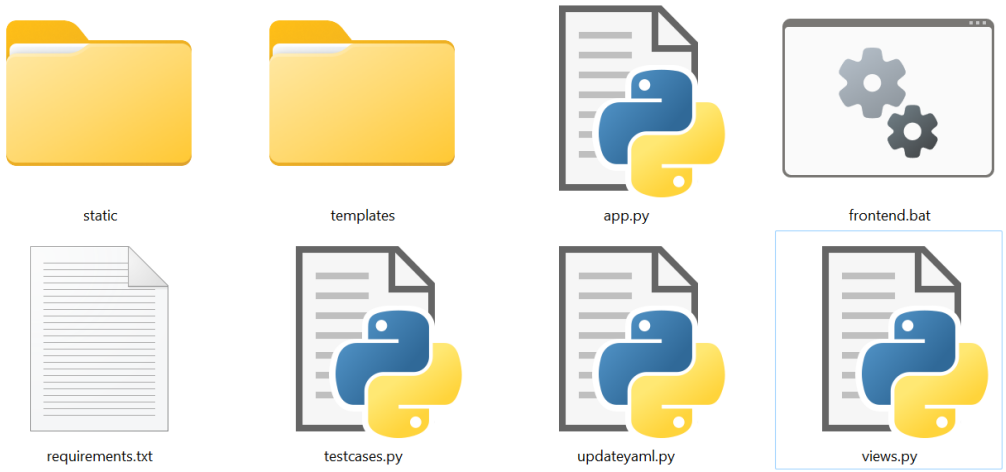


Figura 3.4: Arquitectura e fluxo de interacção no *front-end*.

Pasta *static*: Esta pasta é essencial na arquitetura do *front-end*, sobretudo quando se utiliza o Flask. Ela contém todos os ficheiros estáticos necessários para o correto funcionamento visual e interativo da aplicação *web*. Dentro dela, podemos encontrar:

- **Ficheiros CSS:** Estes são responsáveis pela estilização da aplicação, definindo aspetos como cores, tipografia, espaçamento e outros elementos visuais.
- **Ficheiros Portable Network Graphics (PNG):** Estas imagens contêm recursos gráficos utilizado na página para melhorar a experiência do utilizador ou para ilustrar algum ponto específico.

Pasta *templates*: Esta pasta armazena todos os ficheiros HTML que formam as páginas da aplicação *web*. Estes ficheiros HTML:

- Representam a estrutura e o conteúdo das páginas da aplicação.
- Contém elementos e funções de JavaScript que adicionam interatividade à página, como animações, validações de formulário e chamadas Asynchronous JavaScript and XML (AJAX).

O Flask utiliza esta pasta para renderizar as páginas em resposta a pedidos dos utilizadores.

app.py: Este é o ponto de entrada principal da aplicação *web*. Utiliza o *microframework* Flask para criar e executar a aplicação. Algumas funcionalidades incluem:

- **Blueprints:** views é registado como um blueprint, permitindo modular a estrutura do projeto.
- **Rota `/update1`:** Aceita dados POST e reencaminha-os para a função updateyaml no blueprint views, dependendo dos dados recebidos no formulário.
- **Execução:** Ao iniciar a aplicação, ela é servida no endereço http://127.0.0.1:8000 na porta 8000.

front-end.bat: Este é um ficheiro batch que serve para simplificar a execução do `app.py`. Ao dar um duplo clique neste ficheiro, o *script* `app.py` é automaticamente executado e abre o navegador *web* com o endereço da página. Isto facilita a inicialização da aplicação *web*, especialmente para utilizadores que não estão familiarizados com a linha de comandos.

requirements.txt: Este ficheiro lista todos os módulos Python necessários para correr o *front-end*. Se alguém quiser configurar o projeto numa nova máquina, basta executar pip install -r requirements.txt para instalar todas as dependências necessárias. Este é um passo crucial para garantir a portabilidade do projeto e para assegurar que todos os utilizadores ou developers têm as versões corretas das bibliotecas necessárias.

testcases.py: Este módulo é encarregue de extrair e catalogar os *test cases* e as respetivas *tags* dos ficheiros .robot no diretório 'tests'. As principais funcionalidades incluem:

- Extração de *tags* e documentação de cada teste.
- Categorização dos testes com base na localização e nome do ficheiro.
- Geração de uma representação JSON das informações dos testes.

updateyaml.py: Este *script* desempenha um papel fundamental na geração dinâmica de ficheiros de configuração .yaml com base na estrutura e nos argumentos identificados nas bibliotecas. Ele automaticamente:

- Explora o diretório libraries à procura de ficheiros do tipo "*_server.py".
- Extrai argumentos e configurações de cada servidor.
- Gera o ficheiro .yaml no diretório servers, que pode ser usado para configurar os servidores do projeto.

views.py: Este módulo integra a camada de apresentação e interação do projeto. A sua principal função é definir as rotas, lógica e renderização das páginas da aplicação *web*. As funcionalidades mais relevantes incluem:

- **Registo do Blueprint:** O módulo começa por registar um blueprint denominado "views", que será usado para adicionar rotas e funcionalidades à aplicação Flask.
- **Rotas Básicas:** Inclui rotas para a página inicial (`"/"` e `"/home"`), assim como para operações de atualização (`"/updateyaml"`), estado (`"/status"`), execução (`"/run"`), paragem (`"/stop"`), atualização e visualização de resultados (`"/results"`).
- **Atualização YAML:** A rota `"/updateyaml"` é utilizada para processar alterações submetidas pela interface do utilizador e atualizar o ficheiro YAML.
- **Execução e Monitorização de *scripts*:** As rotas `"/run"` e `"/stop"` são responsáveis pelo arranque e interrupção de *scripts*. O estado dos *scripts* é monitorizado através da variável global `scripts_running`.
- **Transmissão de *logs*:** A funcionalidade `stream_logs` é usada para monitorizar e exibir *logs* em tempo real, permitindo que o utilizador acompanhe o progresso das operações.
- **Documentação:** A rota `"/documentation"` permite ao utilizador aceder a documentações armazenadas em ficheiros .md, tornando-as acessíveis através da interface do utilizador.
- **Funções Auxiliares:** Diversas funções auxiliares, tais como `getCostumersServices`, `servicesStatusYaml` e `processYamlFile`, contribuem para estruturar e modular o código, permitindo operações específicas, como leitura e atualização de ficheiros YAML, identificação de serviços e a sua correta apresentação na interface do utilizador.

O módulo `views.py` é fundamental para a interface do utilizador, uma vez que gere as interações do utilizador com o backend, incluindo a atualização de configurações, execução e monitorização de *scripts*, visualização de resultados e acesso à documentação.

3.2.3 Arquitetura do *front-end* na perspetiva do utilizador

Baseado no diagrama apresentado na Figura 3.5, o fluxo de interação e as funcionalidades do *front-end* podem ser descritas detalhadamente para ilustrar o seu funcionamento.

Ao iniciar a **Execução da instância do *front-end***, o sistema automaticamente **Abre uma janela no navegador predefinido do sistema, no caso o Chrome**. Esta janela apresenta ao utilizador quatro seções principais, a saber:

- **Libraries Section:** Esta secção permite ao utilizador **Escolher Servidores**, após o que se torna possível **Modificar o ficheiro YAML de servidores**.
- **Test Section:** Aqui, o utilizador pode fazer uma **Escolha dos ficheiros de teste**, depois selecionar as **Tags** referentes aos ficheiros de teste que foram selecionados, e finalmente iniciar a execução dos testes ao pressionar o **Botão Run Tests**. Os servidores escolhidos na **Libraries Section** arrancam. Quando este botão é ativado, ocorre uma **Execução no Backend** correspondente a execução dos ficheiros de teste. Ao mesmo tempo, os **Logs são atualizados em tempo real na *logs* Section**. A execução pode seguir dois caminhos: se o utilizador clicar no botão "Stop", o processo é **Abortado**. Caso contrário, o processo segue até o **Término após execução dos testes**.
- **Logs Section:** Esta secção é atualizada em tempo real conforme mencionado anteriormente, proporcionando feedback ao utilizador sobre a execução dos testes. Além disso, quando arrancado a instância do *front-end*, os últimos *logs* são mostrados na página *main* nesta secção.
- **Buttons Section:** Na realidade, um conjunto de botões para várias ações, entre as quais:
  - **Results:** Aqui, o utilizador pode **Visualizar *Reports* HTML** dos testes.
  - **Documentation:** Permite ao utilizador **Consultar a Documentação do Repositório**.
  - **Update YAML:** Esta opção permite ao utilizador **Atualizar o ficheiro YAML** com novas configurações ou modificações.

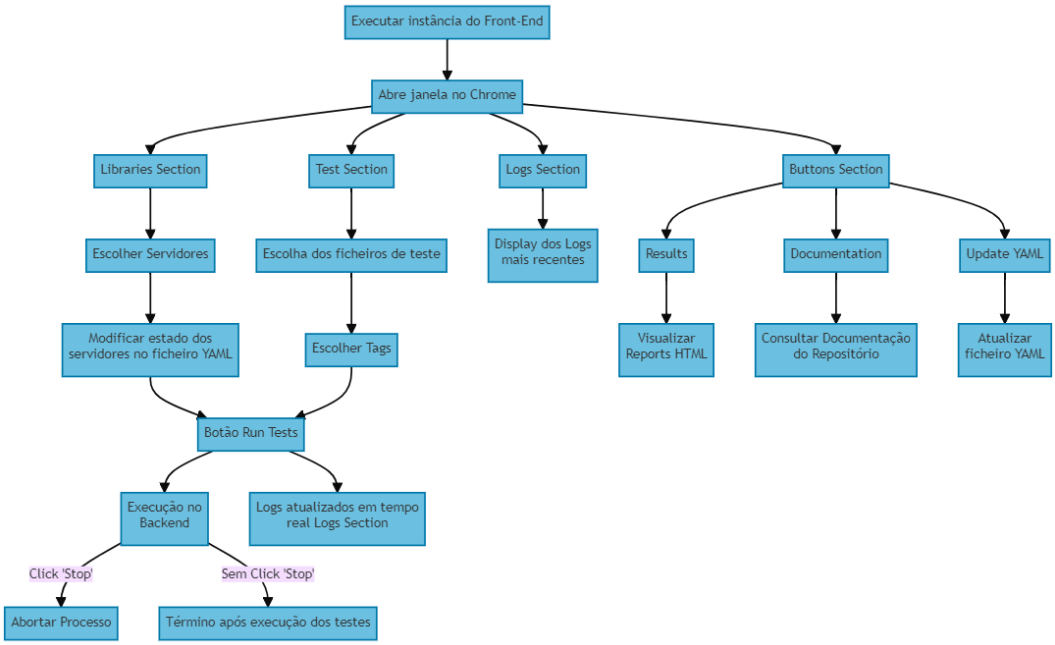


Figura 3.5: Arquitetura do *front-end*.

Toda a interação no *front-end* é projetada para ser intuitiva e amigável ao utilizador, de forma que mesmo aqueles que não possuem familiaridade com o backend ou a estrutura do código possam facilmente configurar e executar os testes conforme necessário.

3.3 Desenvolvimento

Nesta subsecção, descreve-se o trabalho realizado ao longo do desenvolvimento do projeto. O relato é estruturado por fases, alinhando-se ao planeamento previamente estabelecido. Para cada fase, delineia-se o seu propósito, contribuição e relevância para o projeto como um todo.

3.3.1 Implementação do *script* PowerShell

A automatização e a facilidade de integração são cruciais quando se trata de desenvolvimento de software. No âmbito deste projeto, foi implementado um *script* em PowerShell com o objetivo principal de automatizar e simplificar algumas das tarefas associadas à configuração e execução dos testes.

O *script* inicia-se com uma secção denominada "Configuring Station and Test-Data", onde são definidos vários parâmetros essenciais para a execução dos testes. Estes parâmetros incluem, por exemplo, o nome da estação, a pasta principal, os

servidores, entre outros. Esta estrutura parametrizada permite uma personalização e flexibilidade notáveis, tornando o *script* adaptável a diferentes cenários de teste.

Das funcionalidades do *script*, destacam-se:

- Definição da Data de Execução do Teste: Esta funcionalidade garante que os resultados dos testes sejam armazenados de forma organizada, categorizando-os pela data e hora de execução.
- Configuração de Diretórios: O *script* automatiza a criação e organização de pastas para armazenar *logs* e resultados dos testes. Esta organização é crucial para garantir que os dados sejam facilmente acessíveis e interpretáveis.
- Preparação para a Execução de Teste: Antes de iniciar os testes, o *script* prepara o ambiente, garantindo que as pastas de *logs* antigos sejam removidas e que uma nova seja criada, pronta para armazenar os novos *logs*.
- Execução do Teste com o RobotFramework: O *script* chama o *framework* de teste, passando todos os parâmetros necessários, desde o perfil de teste até as *tags* e variáveis.
- Cópia de *logs* para a Pasta de Resultados: Após a execução, os *logs* são automaticamente transferidos para a pasta de resultados, assegurando que sejam mantidos em local seguro e de fácil acesso.

No contexto do *front-end* da ferramenta ETAF, este *script* PowerShell é vital. Permite que os desenvolvedores, sem conhecimentos profundos de PowerShell, possam, de forma simples e rápida, configurar e executar testes. Além disso, ao garantir que os *logs* e resultados sejam armazenados de forma estruturada, a análise dos dados torna-se mais eficiente e intuitiva, contribuindo assim para a otimização contínua do software.

3.3.2 Inicialização do Servidor Flask

O ponto de partida de qualquer aplicação Flask é a sua inicialização. No contexto deste projeto, a instância do servidor Flask é criada com recurso à classe Flask proveniente do módulo flask [50]. Esta instância atua como o núcleo da aplicação, proporcionando as funcionalidades necessárias para gerir as solicitações HyperText Transfer Protocol (HTTP) e responder adequadamente.

```
app = Flask(name)
```

O argumento ___name___ é passado para a classe Flask para determinar o caminho raiz da aplicação. Isso é crucial, especialmente quando a aplicação tem de lidar com recursos estáticos, como folhas de estilo e imagens.

Após a inicialização, são registados diferentes módulos e blueprints. Neste projeto, o blueprint views foi registado para associar as funções de visualização definidas em views.py à instância principal da aplicação:

```
app.register_blueprint(views, url_prefix="/")
```

Esta linha de código garante que todas as rotas e funções definidas no módulo views sejam reconhecidas pela aplicação Flask. O parâmetro url_prefix determina um prefixo comum para todas as rotas definidas no blueprint. Neste caso, um prefixo raíz "/"foi usado, indicando que não há prefixo adicional.

Por fim, a aplicação Flask é configurada para ser executada no modo de depuração e escutar na porta 8000. O modo de depuração permite que os desenvolvedores recebam feedback em tempo real sobre erros e alterações na aplicação, sem a necessidade de reiniciar manualmente o servidor:

```
if name == 'main':
app.run(debug=True, port=8000)
```

Nesta etapa, a aplicação Flask está pronta e aguarda por solicitações HTTP para responder de acordo com as rotas e funções definidas.

3.3.3 Redirecionamento de Rotas

A capacidade de direcionar solicitações HTTP para funções específicas é uma das mais valias do Flask. Cada função de visualização atua como um ponto de resposta para uma rota específica, e a resposta é gerada com base nos dados e lógica incorporados na função.

Neste projeto, foi definida uma rota específica denominada /update1, que é direcionada utilizando o método POST:

```
@app.post('/update1')
def update1():
...
```

A rota /update1 tem a responsabilidade de recolher determinados dados do formulário enviado através de uma solicitação HTTP POST, processar esses dados e, em seguida, redirecionar o *user* para outra rota.

O redirecionamento é feito utilizando as funções redirect e url_for. No contexto deste código:

```
return redirect(url_for("views.updateyaml",front-end1=front-end1))
```

A função redirect instrui o navegador a navegar para um novo Uniform Resource Locator (URL), enquanto a função url_for gera esse URL com base no nome da função de visualização fornecida (neste caso, views.updateyaml) e em quaisquer argumentos adicionais.

Existem várias formas de manipular e responder a solicitações em *frameworks web*. No Flask, o uso de redirect e url_for é uma abordagem padrão por várias razões [50]:

- Separação de Responsabilidades: Mantém as funções de visualização independentes e focadas em tarefas específicas. Uma função coleta e processa os dados; outra exibe a resposta ou o resultado. Esta separação permite uma manutenção mais fácil e um código mais limpo.
- Flexibilidade: O Flask não impõe uma forma específica de definir ou manipular rotas. Isso oferece aos desenvolvedores a liberdade de estruturar as suas aplicações como acharem melhor, adaptando-se às necessidades específicas do projeto.
- Evita URL Hardcoded: A utilização da função url_for evita ter URLs hardcoded no código. Se a URL associada a uma função de visualização mudar no futuro, não será necessário procurar e atualizar manualmente cada ocorrência dessa URL no código.

Como alternativa ao redirecionamento, pode-se enviar uma resposta diretamente na mesma função. Contudo, isso pode resultar em funções extensas e complicadas, tornando o código mais intrincado e de difícil manutenção.

3.3.4 Arranque e encerramento das bibliotecas

Na gestão de uma aplicação contemporânea, é imperativo garantir a eficiência, segurança e robustez através de um manuseamento apropriado das bibliotecas. Este segmento enfatiza a abordagem adotada pelo projeto durante as fases cruciais de inicialização (arranque) e conclusão (encerramento) destas entidades.

O arranque de uma biblioteca implica, frequentemente, mais do que uma simples importação do módulo. Em muitos casos, pode envolver:

- Alocação de Recursos: Alguns módulos podem necessitar de recursos do sistema, como memória ou ligações de rede, para funcionar corretamente.
- Configuração: Definição de parâmetros específicos que orientam o comportamento da biblioteca.

- Validações: Verificações iniciais que garantem que as dependências e as configurações estão corretamente estabelecidas.

Neste projeto, as bibliotecas são iniciadas através de um processo meticuloso que assegura que todos os recursos e configurações necessários estão em vigor antes de serem utilizados. Assim, o encerramento de bibliotecas é tão crucial quanto o arranque, o encerramento adequado de uma biblioteca garante que:

- Recursos são Libertos: Evitam-se assim potenciais fugas de memória ou outras ineficiências.
- Dados são Armazenados: Algumas bibliotecas podem ter operações pendentes ou dados em cache que necessitam de ser persistidos antes do encerramento.
- Operações são Terminadas de Forma Fiável: Assegurando que não ocorram erros ou interrupções abruptas que possam corromper dados ou desestabilizar o sistema.

O projeto adota uma abordagem sistemática para encerrar as bibliotecas, garantindo que cada uma delas é terminada de maneira ordenada e segura.

A gestão adequada do ciclo de vida das bibliotecas não é isenta de desafios. Aqui estão alguns problemas comuns enfrentados e as soluções implementadas neste projeto:

- Dependências Cruzadas: Algumas bibliotecas podem depender de outras para funcionar corretamente. Assegurar uma ordem de inicialização e encerramento que respeite estas dependências foi essencial.
- Exceções Inesperadas: As bibliotecas podem, por vezes, lançar exceções durante o arranque ou encerramento. Estas situações foram tratadas com mecanismos de captura e manipulação de exceções, proporcionando feedback útil e garantindo a estabilidade da aplicação.
- Performance: A inicialização de múltiplas bibliotecas pode ser demorada. Foi dada especial atenção à otimização deste processo, garantindo um arranque rápido e eficiente da aplicação.

O arranque e encerramento das bibliotecas não são apenas etapas técnicas do ciclo de vida da aplicação, são momentos críticos que, quando geridos corretamente, potenciam a eficiência, segurança e robustez da solução. No contexto da dissertação, foi dada especial atenção a estes aspetos, assegurando um funcionamento suave e confiável em todos os momentos.

3.3.5 Atualizar configuração dos servidores (ficheiros YAML)

Os ficheiros em formato YAML (Ain't Markup Language) têm sido amplamente adotados em diversas aplicações devido à sua legibilidade e simplicidade. Na configuração de servidores, por exemplo, esses ficheiros são utilizados para definir e armazenar detalhes essenciais sobre os servidores, tornando-se assim uma ferramenta crucial no processo de configuração.

YAML, sendo um superconjunto de JSON, oferece uma representação mais humanamente legível de dados [51]. No contexto deste projeto, os ficheiros YAML são empregues para manter as configurações dos servidores. Estes ficheiros são essenciais porque proporcionam uma maneira estruturada e padronizada de definir as características e comportamentos dos servidores, assegurando assim uma inicialização e operação consistentes.

Atualizar o ficheiro de configuração é fundamental para refletir as mudanças e evoluções no sistema. Este processo implica a leitura e análise dos *scripts* dos servidores, a extração das informações necessárias e a subsequente escrita dessas informações no ficheiro YAML correspondente.

O *script* updateyaml.py apresentado realiza precisamente essa tarefa. Através da função extract_info_from_server_file, extrai-se detalhes específicos dos *scripts* dos servidores. Posteriormente, a função generate_servers_yaml cria ou atualiza o ficheiro YAML com as informações coletadas. Esta automação assegura que as configurações se mantêm atualizadas, reduzindo a possibilidade de erros humanos e garantindo que os servidores operem segundo as definições mais recentes.

A sequência do processo executado pelo *script* pode ser visualizada no fluxograma da Figura 3.6:

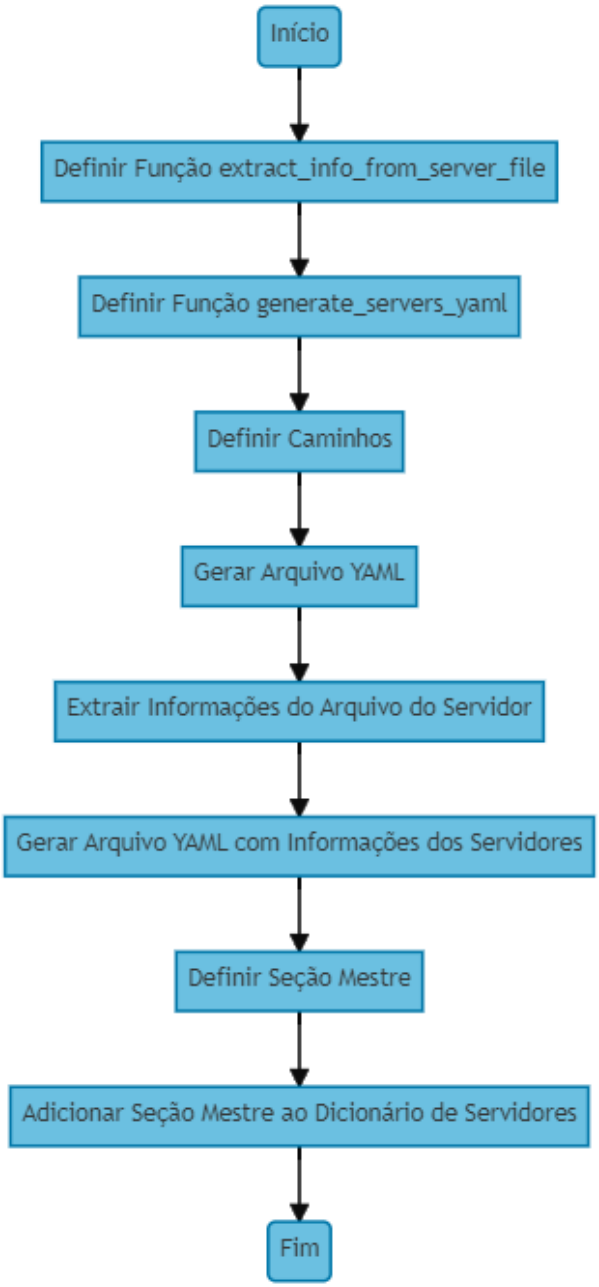


Figura 3.6: Fluxograma referente .

O fluxograma inicia-se com o ponto de partida do programa. A partir daí, definem-se duas funções principais: a extract _info_from_server_file e a `generate_servers_yaml`. A primeira função é responsável por ler um arquivo de servidor específico e extrair informações cruciais das API's, como bin, method, startup e arguments. Por outro lado, a função generate_servers_yaml foca-se na criação de um arquivo YAML contendo detalhes de todos os servidores.

Após a definição das funções, o programa estabelece o diretório atual e determina o nome do arquivo de saída através da etapa "Definir Caminhos". Logo em seguida,

a função para a criação do arquivo YAML é ativada, dando início à etapa de "Gerar Arquivo YAML".

A subsequente etapa de "Extrair Informações"aproveita a função `generate_servers_yaml` para processar os arquivos dos servidores. Durante esse processo, os detalhes são extraídos com a ajuda da função extract_info_from_server_file. Uma vez que todas as informações necessárias são coletadas, a etapa "Elaboração do YAML"entra em ação, gerando e salvando o arquivo YAML correspondente.

Além disso, o fluxograma destaca uma etapa denominada "Seção Mestre", onde informações específicas são definidas manualmente. Por fim, durante a etapa de "Integração da Seção Mestre", essa seção mestre é cuidadosamente integrada ao dicionário de servidores, garantindo que ela seja adequadamente inserida no arquivo YAML final.

Durante o desenvolvimento e implementação do *script* updateyaml.py, vários desafios surgiram, exigindo soluções específicas e adaptativas. O primeiro grande desafio foi a necessidade de automatizar a extração de informações dos *scripts* dos servidores. A variedade e estrutura desses *scripts* demandaram uma abordagem que permitisse extrair, de forma autónoma, os argumentos e as suas configurações padrão. Para superar este obstáculo, foi necessário analisar diretamente o código dos servidores. Utilizando operações de string, conseguimos identificar e processar as informações essenciais, resultando numa extração de dados precisa e eficaz.

Contudo, extrair informações era apenas parte do desafio. Uma vez extraídas, era fundamental que essas informações fossem representadas num formato claro e legível, especialmente quando se tratava de atualizar ou gerar ficheiros YAML. Estes ficheiros, devido à sua natureza, são apreciados pela legibilidade [52]. Portanto, recorreu-se à biblioteca yaml em Python. Esta escolha garantiu que os dados fossem não apenas estruturados corretamente, mas também apresentados de uma maneira que qualquer pessoa com um entendimento básico de YAML pudesse compreender facilmente.

O último, mas não menos importante, desafio era manter a consistência nas atualizações. Com uma diversidade tão grande de servidores, cada um com as suas configurações distintas, era vital que o *script* conseguisse abordar cada servidor como uma entidade única, enquanto, ao mesmo tempo, mantinha um padrão consistente para todos. E foi precisamente o que fizemos, concebendo o *script* para individualizar o tratamento para cada servidor, garantindo, assim, que todas as atualizações fossem uniformes e consistentes.

Ao refletir sobre este processo, fica claro que a combinação de uma análise detalhada, ferramentas adequadas e uma abordagem adaptativa foi crucial para superar os desafios encontrados, resultando numa solução robusta e eficiente.

3.3.6 Gerir e Executar Testes no *front-end*

Para garantir a qualidade do software no lado do cliente, a gestão e execução de testes de *front-end* são cruciais. Neste projeto, foi utilizada a ferramenta RobotFramework, uma estrutura de automação de testes genérica que fornece capacidades para teste de aceitação, teste de aceitação de aplicações de *web* e automação de processos robóticos.

O RobotFramework foi escolhido devido à sua extensibilidade e capacidade de integração com várias outras ferramentas, além de ser amplamente utilizada no ramo da empresa em que se insere este projeto. A *framework* desempenha o papel fundamental de executar os testes escritos num formato legível por humanos e apresentar os resultados de forma formatada e fácil de entender.

O processo inicia-se com o código contido no arquivo testcases.py, responsável por percorrer todos os arquivos com extensão .robot no diretório especificado. Cada arquivo é *scaneado* na procura de detalhes cruciais de cada teste, incluindo o nome do teste e as *tags* associadas. Estes detalhes são organizados e compilados numa estrutura de dicionário que não só categoriza os testes pelo seu nome, mas também com base no diretório e nome do arquivo onde estão contidos.

A representação visual dos testes acontece no arquivo etaf.html. Nesse ambiente, a informação previamente extraída é exibida numa estrutura que remete a uma árvore, facilitando o processo de seleção por parte dos utilizadores. Estes, por sua vez, podem optar por selecionar todos os testes de uma categoria específica ou realizar seleções individuais. Ao escolher um teste, a(s) sua(s) *tags* associadas são imediatamente destacadas numa secção própria.

A etapa final deste processo é a execução dos testes selecionados. Ao decidir quais testes executar, e eventualmente as respetivas *tags*, os utilizadores têm à sua disposição o botão 'Run'. Ao clicar no mesmo, a rota /run no arquivo views.py é acionada. Este segmento de código é responsável por dar início à execução dos testes por um *script* PowerShell. À medida que os testes são processados, tanto o progresso quanto qualquer saída gerada são claramente exibidos no *front-end*, mantendo o utilizador informado a cada passo.

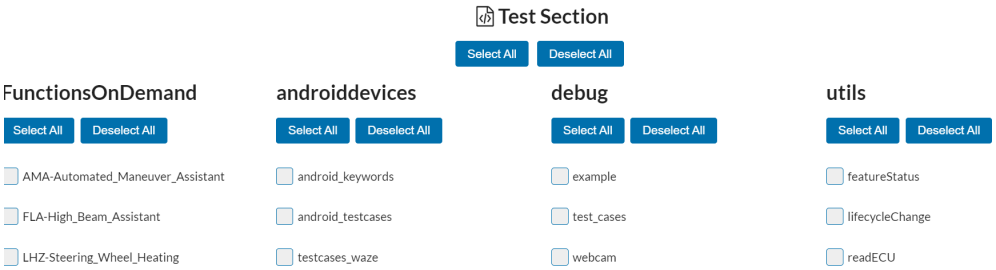


Figura 3.7: Secção de Testcases.

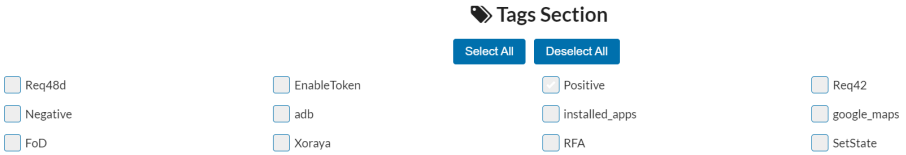


Figura 3.8: Secção de *tags* após a seleção de Testcases.

A automação de testes acelera o ciclo de desenvolvimento, reduzindo a necessidade de testes manuais repetitivos e melhorando a confiabilidade dos resultados. Melhora a eficiência do processo de desenvolvimento, mas também garante a entrega de um software de alta qualidade para o utilizador final.

Uma característica distintiva deste projeto é a maneira como os testes são apresentados e organizados no *front-end*. Ao selecionar testes, o sistema exibe automaticamente as *tags* associadas, permitindo uma filtragem ainda mais refinada. Esta funcionalidade é particularmente útil em projetos grandes, onde pode haver centenas de testes com várias *tags*.

Além disso, o código faz uso extensivo da biblioteca Flask para criar uma aplicação *web* e da biblioteca RobotFramework para a gestão de testes. O *script* run_tests_powershell.ps1 permite a execução dos testes através de uma chamada do PowerShell, proporcionando uma integração mais profunda com o sistema operacional.

A capacidade de gerir e executar testes de *front-end* é crucial para garantir a qualidade do software. A combinação do Robot Framework com uma aplicação *web* interativa oferece uma solução robusta para testes automatizados. As características técnicas avançadas, como a seleção dinâmica de *tags*, destacam a complexidade e o esforço dedicado a este projeto.

3.3.7 Logs integrados no *front-end*

A integração dos *logs* diretamente na interface *front-end* é uma prática fundamental para facilitar o monitoramento e a manutenção do sistema em tempo real. Aqui, abordaremos como esses *logs* são gerados, capturados e, em seguida, integrados no *front-end*.

Os *logs* são gerados por várias rotas e funções dentro do código do servidor. Cada evento, erro ou atividade significativa é registado nos *logs*, permitindo uma análise e diagnóstico eficientes em caso de problemas.

A função stream_logs foi implementada para servir *logs* em tempo real para a interface *front-end*. Esta função utiliza a técnica "Server-Sent Events"(EventSource) para enviar *logs* continuamente ao *front-end*. O arquivo de *log* é lido e, sempre que há uma nova linha de *log*, ela é enviada como um evento para o *front-end*.

Figura 3.9: Secção de *logs*.

A integração desses *logs* na interface é feita mediante uma área designada para exibição dos mesmos. No código HTML fornecido, a secção designada para exibição dos *logs* é um div com o ID *log-display*. Este *container* possui um estilo específico para garantir uma exibição limpa e organizada dos *logs*, incluindo uma barra de rolagem quando o conteúdo excede a altura definida. O JavaScript associado a essa secção é responsável por estabelecer uma conexão com a fonte do evento (neste caso, a rota /stream_logs), ouvir novos eventos (novas linhas de *log*) e, em seguida, adicionar essas linhas à área de exibição. Para evitar o excesso de mensagens de *log*, o código limita a exibição às últimas quinze mensagens, removendo as mais antigas conforme novas mensagens chegam.

Integrar *logs* ao *front-end* possui várias vantagens:

- Monitoramento em Tempo Real: Permite que os administradores ou desenvolvedores monitorem a atividade e o status do sistema em tempo real, sem a necessidade de acessar servidores ou arquivos de *log* diretamente;
- Diagnóstico Rápido: Qualquer problema ou erro pode ser rapidamente identificado e diagnosticado;
- Facilidade de Uso: Integrar *logs* diretamente no *front-end* torna a experiência do utilizador (especialmente o administrador ou desenvolvedor) mais fluida e integrada.

3.3.8 Documentação de bibliotecas e recursos

A documentação é o pilar do desenvolvimento de software. Atua como um guia para os desenvolvedores, elucidando as nuances dos códigos, bibliotecas e recursos disponíveis. No contexto atual, com a crescente complexidade dos projetos de software, a necessidade de uma documentação robusta e facilmente acessível tornou-se mais premente do que nunca.

A primeira e mais evidente razão para documentar software centra-se na ideia de clareza e consistência. Quando um projeto é complexo, é crucial ter uma única fonte de verdade. Este ponto central de referência ajuda a evitar ambiguidades. Imagine ter um projeto com várias versões do mesmo software, e cada versão tem nuances subtis que diferem das outras. Sem uma documentação adequada, a confusão seria inevitável.

Adicionalmente, a documentação desempenha um papel crucial ao facilitar a integração de novos membros na equipa. Imagine ser um novo membro e ter de navegar por um mar de código sem um mapa ou guia. Isso não só prolongaria o período de integração, como também poderia conduzir a erros por equívocos. Com uma documentação bem organizada, esse novo membro pode compreender as principais áreas do projeto de forma autónoma, sem necessidade de perguntar frequentemente aos colegas.

A terceira grande vantagem da documentação é que ela promove a manutenção do software. Conforme a evolução do mesmo, a documentação atua como um registo histórico das decisões tomadas, tornando mais fácil para as equipas entenderem por que certas escolhas foram feitas e como podem ser adaptadas no futuro.

Ao considerar a integração da documentação no *front-end*, o código que apresentou oferece uma visão prática de como isso pode ser concretizado. O processo inicia-se com a procura de ficheiros. Embora pareça banal, esta etapa é essencial para

assegurar que toda a documentação seja considerada. Depois, temos a conversão para HTML. O fascinante aqui é a transformação de um formato tão legível como o Markdown para o versátil HTML. Por fim, a apresentação no *front-end* oferece uma interface interativa, usando um acordeão para maximizar o espaço, mantendo a informação facilmente acessível.

A utilização de ficheiros Markdown (MD) para documentação oferece várias vantagens. Em primeiro lugar, destaca-se a simplicidade do Markdown, que é um formato fácil tanto de escrever quanto de ler. Esta simplicidade elimina a complexidade inerente ao HTML, proporcionando uma experiência muito mais amigável para os autores. Em segundo lugar, temos a versatilidade dos arquivos MD, que são notavelmente leves e portáveis. Esta característica permite que eles sejam versionados, compartilhados e até mesmo convertidos em outros formatos com grande facilidade. Por fim, a integração dinâmica é outro ponto forte. Esta abordagem facilita a adição, atualização ou remoção de documentação. Basta modificar os ficheiros MD nas pastas pertinentes, sem que haja necessidade de interferir no código principal.

Ao analisar a estrutura e *design* do template Documentation.html, torna-se evidente o empenho em tornar a informação acessível e atraente. A organização hierárquica distribui a informação de uma forma lógica, permitindo uma pesquisa rápida. O componente acordeão é uma escolha de *design* astuta que economiza espaço e proporciona uma experiência fluida ao utilizador. Além disso, a estilização personalizada assegura que a documentação não é só informativa, mas também visualmente apelativa, contribuindo para uma experiência de utilizador enriquecida.

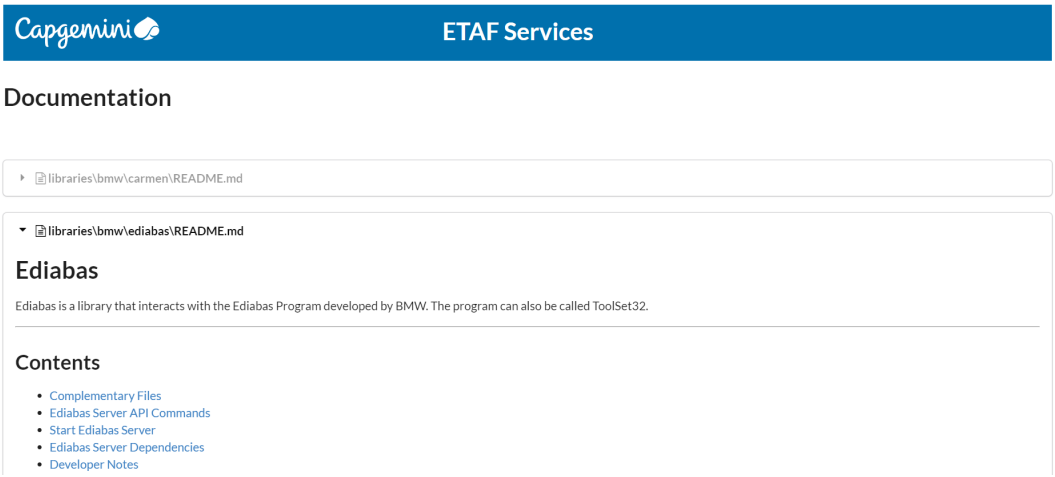


Figura 3.10: Página de Documentação do Repositório no *front-end*.

Dado que a solução proposta é baseada em arquivos MD e templates HTML, há ampla margem para expansão. Funcionalidades adicionais, como uma barra de pesquisa, *tags* para categorização, ou mesmo integração com sistemas de comentários, podem ser facilmente incorporadas para enriquecer ainda mais a documentação no *front-end*.

3.3.9 Resultados das execuções dos testes no *front-end*

O **RobotFramework** é uma ferramenta de automação genérica utilizada, entre outras coisas, para testes de aceitação. Através da sua execução, são gerados ficheiros '.html' que retratam detalhadamente os resultados dos testes realizados. Estes ficheiros contêm informações acerca das especificações testadas, se passaram ou falharam, e outros metadados relevantes.

A representação visual destes resultados é de crucial importância por diversas razões:

- **Acessibilidade:** Facilita o acesso e a interpretação dos dados por parte do utilizador, tornando a informação mais compreensível do que se estivesse em formato bruto.
- **Organização:** Dada a potencial grande quantidade de testes realizados, é fundamental ter uma estrutura organizada, categorizada por pastas e ficheiros, para facilitar a localização de informações específicas.
- **Identificação Rápida:** Uma representação visual bem-estruturada permite uma identificação rápida de testes que falharam, otimizando o processo de debug.

Integração e Apresentação dos Ficheiros

O código em 'results.html' apresenta uma página *web* onde os resultados são exibidos numa estrutura de acordeão, categorizados por pastas. Cada pasta, quando selecionada, expande para mostrar os ficheiros de resultados específicos contidos nela.

O backend, implementado em Flask e definido em 'view.py', tem a responsabilidade de:

- Ler e categorizar os ficheiros '.html' na pasta 'results'.
- Codificar os caminhos dos ficheiros para garantir a segurança no acesso através da *web*.

- Renderizar o template 'results.html', fornecendo os dados necessários para sua correta visualização.

Ao lidar com a abertura de ficheiros através de uma aplicação *web*, a segurança é uma preocupação primordial. O método utilizado para codificar e posteriormente decodificar o caminho do ficheiro serve para garantir que não sejam abertos ficheiros não intencionais ou maliciosos.

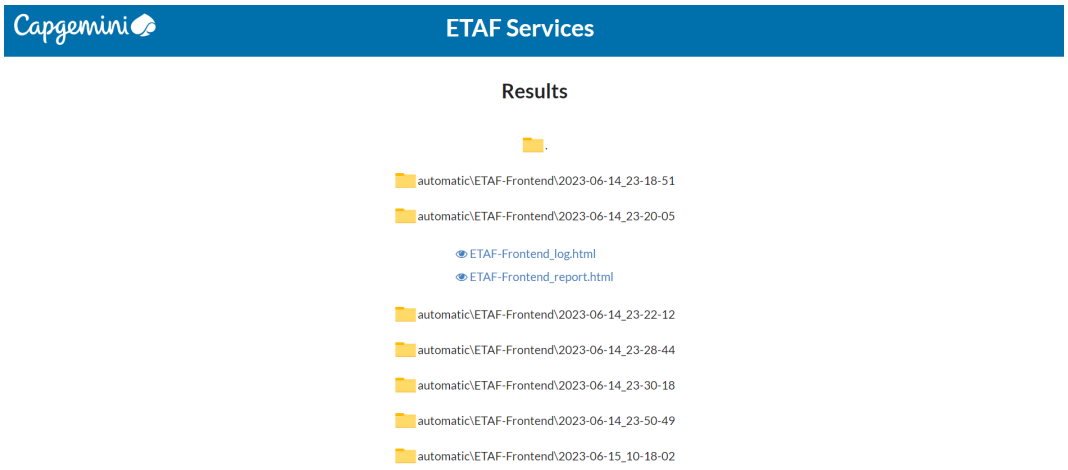


Figura 3.11: Página de resultados gerados por builds no *front-end*.

Os resultados são organizados alfabeticamente, tanto em termos de nome da pasta como de nome do ficheiro. Isto facilita a localização e a acessibilidade, tornando a experiência do utilizador mais fluida e intuitiva.

A estrutura de código apresentada é modular e de fácil manutenção. Futuras melhorias, como integrar outras ferramentas de teste, podem ser facilmente implementadas dada a clara separação entre a lógica de backend e a representação *front-end*.

3.3.10 Documentação dos Testcases de Forma Interativa

Documentar testes de software é uma tarefa fundamental para qualquer equipa que procura garantir a qualidade do código e a consistência do comportamento do software. Uma boa documentação permite que testadores e desenvolvedores entendam os objetivos dos testes, os passos a seguir e os resultados esperados. Além disso, a documentação é útil para manutenção e evolução do código, bem como para a integração de novos membros na equipa.

O principal objetivo da documentação interativa é tornar a experiência do utilizador mais fluida e intuitiva. Ao contrário da documentação estática, a documentação

interativa permite aos utilizadores selecionar, filtrar e interagir com os casos de teste de forma visual. Os principais benefícios incluem:

- **Facilidade de Uso:** Com a possibilidade de selecionar e deselecionar testes, o utilizador pode rapidamente identificar e executar um conjunto específico de testes.
- **Visualização Clara:** A estrutura hierárquica facilita a localização de testes específicos, agrupados por pastas e *tags*.
- **Informações Detalhadas:** Tooltips fornecem informações detalhadas sobre cada teste, incluindo a documentação associada e as *tags* relacionadas.
- **Flexibilidade:** A capacidade de filtrar testes por *tags* permite que os utilizadores encontrem testes relevantes para cenários específicos ou áreas de foco.

A documentação foi implementada usando a linguagem de template Jinja, que permite a inserção dinâmica de conteúdo em páginas HTML. Com esta abordagem, podemos gerar listas e estruturas interativas baseadas em dados provenientes de *scripts* Python.

Os principais componentes técnicos incluem:

- **HTML e CSS:** Utilizados para criar a estrutura visual e estilização da página.
- **Jinja Templates:** Permite a integração dinâmica dos dados de testes dentro da página HTML.
- **JavaScript:** Usado para fornecer interatividade, como a seleção de testes e a geração de tooltips.
- **Python:** O *script* Python testcases.py analisa os arquivos de teste .robot para extrair informações relevantes, como nome do teste, documentação e *tags*.

Durante a implementação, enfrentaram-se vários desafios:

- **Extração de Dados:** Os arquivos de teste .robot não são projetados para serem analisados facilmente. Recorreu-se a implementação lógica adicional para lidar com *tags*, documentações e hierarquias de teste.
- **Performance:** Com um grande número de testes, a página pode se tornar lenta. Otimizações foram necessárias para garantir uma experiência suave ao utilizador.
- **Usabilidade:** Garantir que os utilizadores possam navegar e interagir com a página de forma intuitiva foi essencial, o que levou a várias iterações de *design*.

A documentação interativa dos testcases é uma ferramenta poderosa que combina a rigorosidade da documentação tradicional com a flexibilidade e usabilidade das interfaces modernas. Permite uma visualização rápida, interação intuitiva e proporciona uma experiência aprimorada para os utilizadores. Mesmo com desafios, o resultado provou ser um valioso ativo para as futuras equipas de desenvolvimento e de teste.

Capítulo 4

Resultados e Discussão

O presente capítulo destina-se a realizar uma análise crítica do sistema. Assim, são explorados os resultados obtidos através da execução do sistema implementado, e de testes feitos ao mesmo, bem como uma análise ao seu desempenho. Por fim, são abordadas possíveis adições que poderiam ser implementadas no projeto desenvolvido.

4.1 Análise dos resultados obtidos

Para uma melhor compreensão dos resultados e da interação do *user* com a aplicação, é crucial entender o fluxo de funcionamento do *front-end*. Este processo inicia-se através da execução do ficheiro *app.py*.

Foi criado um pequeno ficheiro bat para facilitar o desenvolvimento da aplicação do *front-end*. O mesmo tem um comando para um ficheiro powershell e para abrir uma janela no navegador chrome. Ao iniciar a aplicação através do comando `powershell -file front-end.ps1`, somos apresentados a uma série de mensagens no terminal, as quais ilustram os passos subsequentes do fluxo de funcionamento.

Como observado na Figura 4.1, após a inicialização, a aplicação é servida pelo servidor Flask, um *microframework* de desenvolvimento *web*. O modo de depuração (Debug mode) é ativado, proporcionando uma ferramenta essencial para os desenvolvedores identificarem e corrigirem possíveis problemas em tempo real.

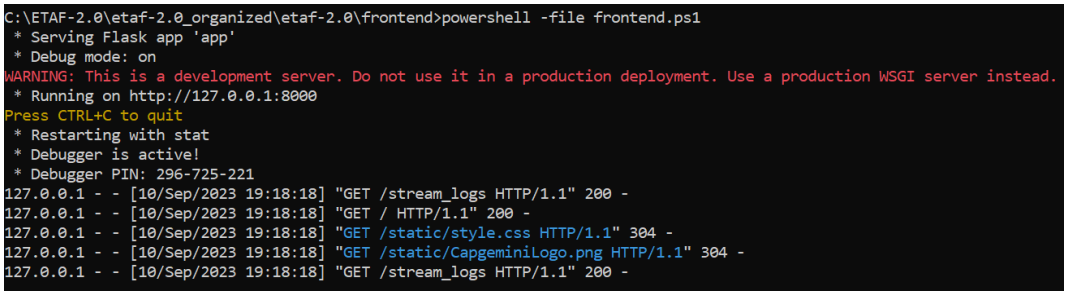


Figura 4.1: Arranque da Aplicação no terminal

É importante salientar o aviso sobre este ser um servidor de desenvolvimento, o que implica que não é, ainda, adequado para ambientes de produção, sendo recomendado o uso de um servidor WSGI para tais contextos.

A aplicação fica acessível através do endereço `http://127.0.0.1:8000`, e os utilizadores podem aceder à mesma usando um navegador *web*. Em caso de necessidade, os desenvolvedores têm a opção de terminar a execução da aplicação, pressionando as teclas CTRL+C.

Finalmente, informações sobre o estado do servidor e a atividade do depurador (*debugger*) são fornecidas. O PIN do depurador é uma camada adicional de segurança, garantindo que apenas pessoas autorizadas tenham acesso às ferramentas de depuração.

Com a aplicação em funcionamento e o depurador ativo, a aplicação está pronta para receber e processar as requisições dos utilizadores, retornando as respostas adequadas e garantindo uma interação fluida e eficaz com a interface de utilizador.

Ao aceder ao *front-end* através do navegador, é apresentada uma interface gráfica claramente estruturada, observável na Figura 4.2.

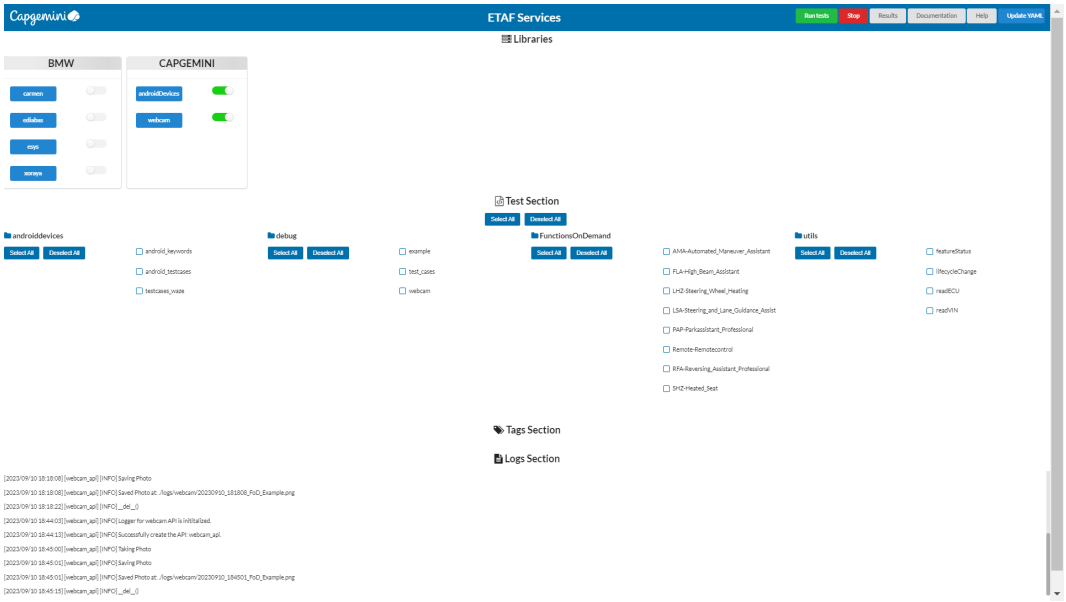


Figura 4.2: Display do *front-end* em grande escala

No topo da página, encontra-se o título *ETAF Services* seguido pelo logotipo da entidade parceira, *Capgemini*. Este logotipo não só serve para contextualizar a aplicação, mas também para reforçar a colaboração entre as entidades envolvidas no projeto. Abaixo do logotipo, a GUI é subdividida em várias secções, cada uma responsável por uma funcionalidade específica:

- *Libraries*: Nesta secção são demonstrados os servidores organizados pelo nome dos diretórios em que se encontram na pasta **libraries** do repositório do *front-end*. Os mesmos têm uma **toggle checkbox**, se ficar à direita (ou verde), vai comunicar com o ficheiro YAML, transmitindo a informação que o respetivo servidor deve ser ativo, para a execução que se queira correr.

- **Test Section**: Aqui, os utilizadores podem selecionar diversos testes a ser realizados. A estrutura é composta por categorias como *androiddevices*, *debug* e *utils*, que são os respetivos diretórios presentes no diretório **tests** do ETAF, para uma melhor organização. Cada uma contendo os respetivos testes que podem ser selecionados ou desselecionados.

- **Tags Section**: Nesta secção são demonstradas as *tags* dos ficheiros **robot** selecionados na **Test Section**. Tal como é o caso para esta última, nesta secção existe igualmente a possibilidade de selecionar as *tags*, uma a uma, todas de uma vez ou desselecionar.

- **Logs Section**: Esta secção, parte essencial da interface, exibe *logs* em tempo real, relacionados à execução e ao estado de execuções da aplicação. Estes *logs* fornecem informações cruciais sobre o funcionamento da aplicação, permitindo

aos utilizadores e aos desenvolvedores monitorizar e identificar problemas, bem como confirmar ações bem-sucedidas. Por exemplo, é possível identificar que certas API's foram inicializadas com sucesso, enquanto outras apresentaram erros. Mal a aplicação seja inicializada, esta função é ativada. Sempre que decorra uma execução por parte do *front-end* a mesma é também chamada, para ir mostrando os *logs* atualizados. Contém as últimas quinze linhas do ficheiro all_logs, e uma *scroll bar* em que se receber alguma atualização, faz o *scroll* automático para o mais recente.

Para além dos pontos já mencionados, é importante frisar os botões do canto superior direito. Os mesmos são fulcrais ao sistema. O 'Run tests' e o 'stop', são responsáveis pela execução de testes. Isto é, após selecionados os servidores, testes e respetivas *tags* na página principal do *front-end*, o botão verde deve então ser clicado para que a execução suceda. Caso, em algum momento durante a execução, o desenvolvedor queira parar a execução, basta clicar no botão 'stop', e todo o processo a decorrer (terminais abertos relativos à execução) é encerrado imediatamente.

No que diz respeito às restantes opções da interface, temos os três botões, o **Results**, **Documentation** e **Help**. O ETAF, aquando de uma execução, guarda os ficheiros gerados num diretório com o respetivo **timestamp**. O botão de **Result** direciona para uma página secundária, onde são demonstrados os ficheiros HTML do diretório 'results' do ETAF, organizados por diretório, tal como demonstra a Figura 3.11.

Temos também o botão 'Documentation', no qual são apresentados todos os ficheiros MD presentes nos diretórios 'libraries' e 'resources' do ETAF. Por norma, os ficheiros de documentação em 'libraries', são referentes aos servidores, enquanto os de 'resources', são referentes a documentação dos ficheiros 'robot'. O resultado da página está demonstrado na Figura 3.10.

O botão 'Help', é um botão ainda não desenvolvido, mas que num futuro próximo vai conter documentação das melhores práticas de utilização do *front-end* por parte de desenvolvedores, com o intuito de que estes consigam retirar o máximo proveito possível desta ferramenta. Algumas destas práticas englobariam, por exemplo, de que forma devem ser estruturados os testes robot ou os servidores, para que a conversão para o ficheiro YAML possa ser feita da forma mais eficaz.

Por fim, temos o botão 'Update YAML'. De forma simplista, a função prende-se com a necessidade de leitura de ficheiros dos servidores que estão escritos em python e retirar a informação necessária, a fim de a escrever no ficheiro YAML. Esta funcionalidade é também ativa, caso o ficheiro YAML esteja vazio no arranque da aplicação.

4.1.1 Resultados da Implementação do *script* PowerShell

O *script* em PowerShell implementado demonstrou ser uma ferramenta eficaz no processo de configuração e execução dos testes automatizados. Abaixo estão resumidos os principais resultados alcançados com o uso deste *script*:

Parâmetros Configuráveis: O *script* permitiu a configuração de diversos parâmetros essenciais para a execução dos testes. Estes parâmetros, tais como o nome da estação ($station_name), a pasta principal ($mainfolder), entre outros, ofereceram uma grande flexibilidade, adaptando-se a diferentes cenários de teste. De notar que parâmetros como os *suites* (ficheiros de teste) e as *tags* são especificamente definidos através do *front-end*, permitindo assim uma personalização do conjunto de testes a serem executados e das *tags* associadas a eles.

Automatização do Processo: Desde a configuração de diretórios até a execução de testes com o RobotFramework, o *script* assegurou que todo o processo fosse realizado de forma automatizada. Os diretórios de *logs* são geridas automaticamente, os resultados dos testes são categorizados pela data e hora de execução, e os *logs* são transferidos para uma pasta de resultados, facilitando a posterior análise.

Saída de Execução: Durante a execução do *script*, é apresentado no terminal um resumo dos parâmetros e configurações aplicados. A Figura 4.3 apresenta uma captura de tela desta saída, destacando a estrutura organizada e clara das informações retornadas.

Figura 4.3: Saída de execução do *script* PowerShell no cmd.

Esta saída não apenas confirma que o *script* foi executado com sucesso, mas também fornece uma visão clara dos parâmetros utilizados e das ações realizadas. Esta

visibilidade é essencial para os desenvolvedores, permitindo uma rápida validação e eventual depuração.

4.1.2 Demonstração de uma execução

A fim de demonstrar o funcionamento prático do *front-end*, é apresentada uma execução passível de ser realizada. Neste exemplo, são selecionados os servidores 'androidDevices' e 'webcam' (Figura 4.4):

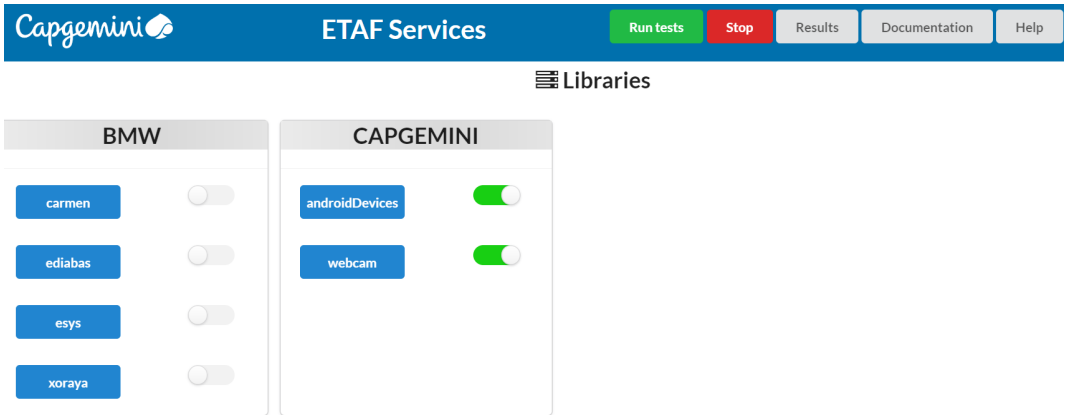


Figura 4.4: Parâmetros selecionados nos servidores atribuídos no *front-end*

Esta informação é passada para o ficheiro YAML, responsável pelas ações dos servidores. Além do arranque dos servidores já referidos, o servidor arranca o *master* que funciona como orquestrador, coordenando a execução e comunicação entre as várias livrarias. O resultado que o *front-end* passa para o ficheiro YAML é demonstrado na Figura 4.5, referente ao terminal da execução:

```
Configuration file: C:\ETAF-2.0\etaf-2.0_organized\etaf-2.0\servers\servers_CI_FoD_station1.yaml
params: {
    "androidDevices": {
        "arguments": {
            "ip": "{IP_MACHINE}",
            "logger": "../logger/configuration.json",
            "port": "20013"
        },
        "bin": "C:\\ETAF-2.0\\etaf-2.0_organized\\etaf-2.0\\frontend\\..\\libraries\\capgemini\\androidDevices\\androidDevices_server.py",
        "method": "python",
        "startup": "True"
    },
    "master": {
        "bin": "./run_server3.0",
        "ip": "{IP_MACHINE}",
        "logger": "./libraries/capgemini/logger/configuration.json",
        "port": 20000,
        "startup": "True"
    },
    "webcam": {
        "arguments": {
            "camera-type": "0",
            "ip": "{IP_MACHINE}",
            "logger": "./libraries/capgemini/logger/configuration.json",
            "port": "20012"
        },
        "bin": "C:\\ETAF-2.0\\etaf-2.0_organized\\etaf-2.0\\frontend\\..\\libraries\\capgemini\\webcam\\webcam_server.py",
        "method": "python",
        "startup": "True"
    }
}
```

Figura 4.5: Parâmetros dos servidores atribuídos no *front-end* para o ficheiro YAML

Seguindo com a *demo*, na secção de testes é selecionado o ficheiro 'example', que contém várias *tags*, como demonstrado na Figura 4.6. Dessas mesmas *tags* é selecionada a *webcam*. Quer isto dizer que o *front-end* vai passar a informação desses parâmetros para o ficheiro powershell *script*.

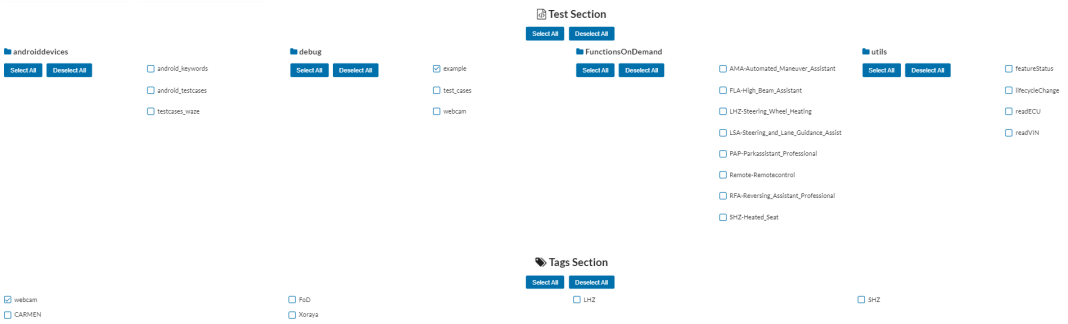


Figura 4.6: Secção de *Tags* do *testcase webcam*

Desta forma, após o clique no botão 'run tests' do *front-end*, a informação é passada para o *backend*. Na imagem 4.7 são demonstrados todos os argumentos do ficheiro powershell, que os passa, posteriormente, para o 'run_tests.py' para a fase da execução. Neles, podemos ver que os 'suites' (testes) e 'tags' são os definidos no *front-end*.

```
args:   {
    "conf": "./servers/servers_CI_FoD_station1.yaml",
    "mode": "single",
    "output_dir": "./results/automatic/ETAF-Frontend/2023-09-10_18-43-52/",
    "profile": "./profiles/CI_FoD_Station1_testCases.txt",
    "repetitions": 1,
    "serv": "all",
    "tags": "webcam",
    "default_path": "None",
    "test": "None",
    "report_name": "ETAF-Frontend",
    "start_server": true,
    "suites": [
        "example"
    ],
    "variables": [
        "None"
    ]
}
```

Figura 4.7: Parâmetros do *script* Powershell

A partir desse momento, o processo fica responsável pelo *core* da ferramenta ETAF, que vai executando o processo em segundo plano. Aqui, é visualizado o processamento dos testes, reportados pelos ficheiros 'robot'.

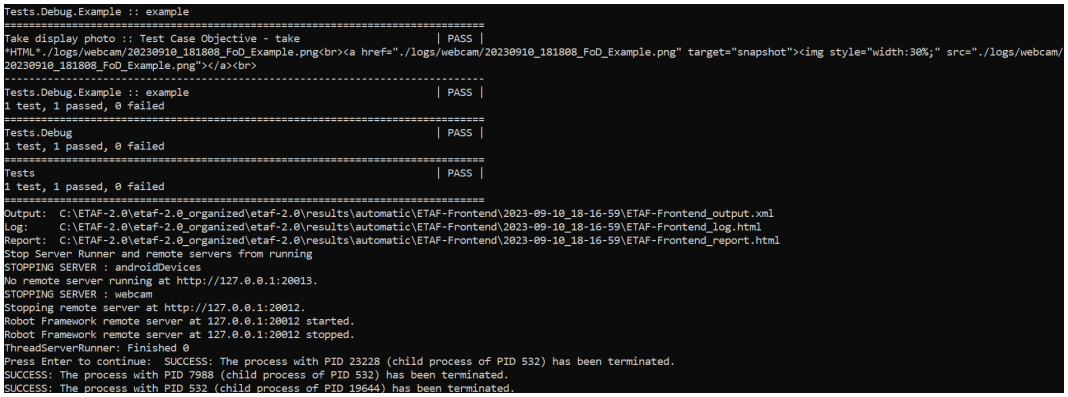


Figura 4.8: Execução no terminal do teste

Os resultados gerados através da execução ficam disponíveis na página 'Results', assim que os ficheiros HTML sejam gerados. Desta forma, o desenvolvedor tem conhecimento do conteúdo dos ficheiros de *report*, sem que seja necessário aceder localmente, tal como demonstra a Figura 4.9.

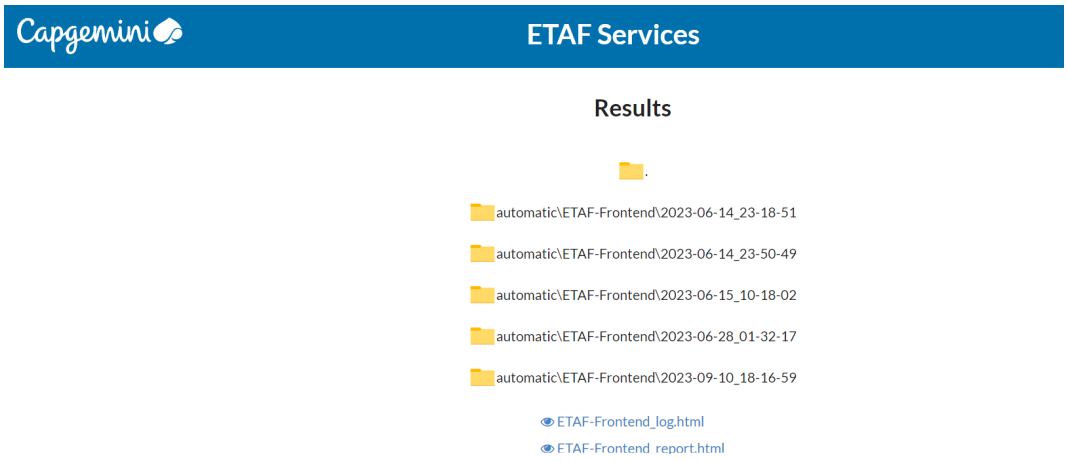


Figura 4.9: Página de *reports*, com o diretório criado e os respetivos ficheiros de *report* criados

4.1.3 Demonstração da automatização do *front-end* nas suas diferentes secções

A presente secção do relatório concentra-se na demonstração prática das diversas funcionalidades que o *front-end* da ferramenta proporciona. Por meio de exemplos concretos, ilustra-se como a ferramenta facilita e otimiza a gestão de testes, servidores e documentação.

Atualização Automatizada do YAML

A ferramenta oferece a capacidade de atualizar automaticamente o ficheiro YAML. Esta funcionalidade é evidenciada pelo botão 'Update YAML', que, ao ser acionado,

reflete imediatamente as alterações mais recentes feitas pelo utilizador no ficheiro correspondente. No exemplo abordado, integrou-se uma API já desenvolvida para o ETAF, o docker, no diretório 'libraries'. Após carregar no botão 'Update YAML', o excerto de código resultante da conversão do arquivo docker_server.py é transladado para o ficheiro YAML, tal como é demonstrado na Figura 4.10:

```yaml
docker:
  arguments:
    executable: C:/Program Files/Docker/Docker/resources/bin
    ip: '{IP_MACHINE}'
    logger: ./libraries/altran/logger/configuration.json
    port: '20020
  bin: C:\ETAF-2.0\etaf-2.0\etaf-2.0\frontend\..\libraries\ISEP\docker\docker_server.py
  method: python
  startup: 'False'
```

Figura 4.10: Código convertido para o ficheiro `servers_CI_FoD_station1.yaml` através do botão 'Update YAML'

Gestão Automatizada dos servidores

A gestão dos servidores é uma característica essencial da ferramenta, permitindo que os utilizadores adicionem ou modifiquem servidores conforme necessário. Esta secção destaca duas funcionalidades centrais:

- **Reflexo na Secção de Bibliotecas:** Ao adicionar servidores, a ferramenta atualiza automaticamente estas adições na secção de bibliotecas, tornando mais intuitiva a gestão e visualização dos servidores disponíveis. O exemplo aqui dado, vem da sequência da "Atualização Automatizada do YAML"com a API docker. Após atualizar a página, as mudanças são refletidas no *front-end*, como demonstra a Figura 4.11.

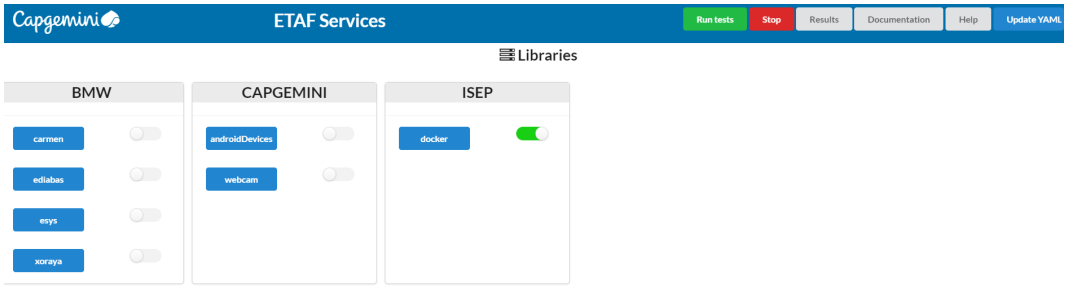


Figura 4.11: Demonstração da atualização das 'Libaries' no *front-end*

- **Atualização do estado do servidor:** Com um simples clique na *toogle checkbox* referente ao servidor docker, o estado dos servidores são atualizados no ficheiro YAML, garantindo que no momento da execução apenas sejam iniciados os servidores pretendidos. Assim, face à imagem 4.10, a Figura 4.12 altera o estado do servidor no respetivo ficheiro YAML:

```
docker:
  arguments:
    executable: C:/Program Files/Docker/Docker/resources/bin
    ip: '{IP_MACHINE}'
    logger: ./libraries/altran/logger/configuration.json
    port: '20020
  bin: C:\ETAF-2.0\etaf-2.0\etaf-2.0\frontend\..\libraries\ISEP\docker\docker_server.py
  method: python
  startup: 'True'
```

Figura 4.12: Estado do servidor da API docker, agora ativado através da *toogle checkbox* no *front-end*

Deteção de Ficheiros de Testes Robot

A ferramenta é dotada de uma capacidade robusta de deteção:

- **Identificação Automática:** A ferramenta é capaz de identificar automaticamente os ficheiros de testes de robot, bem como os diretórios onde se encontram, tornando a gestão e execução de testes mais eficiente.
- **Influência das *tags*:** Uma demonstração particularmente útil é a capacidade da ferramenta em alterar a visibilidade dos testes com base nas *tags*. Ao remover certas *tags*, os testes associados a elas deixam de estar visíveis, oferecendo uma gestão flexível dos testes.

No teste demonstrado, foi adicionado um novo diretório contendo ficheiros robot na pasta 'tests' com o nome 'ISEP'. Por outro lado, o ficheiro de teste 'featureStatus' foi removido do diretório 'utils', e a *tag* 'VIN' foi excluída do teste 'readVIN'. Como ilustrado nas figuras 4.13 e 4.14, o *front-end* deteta todas as alterações mencionadas:

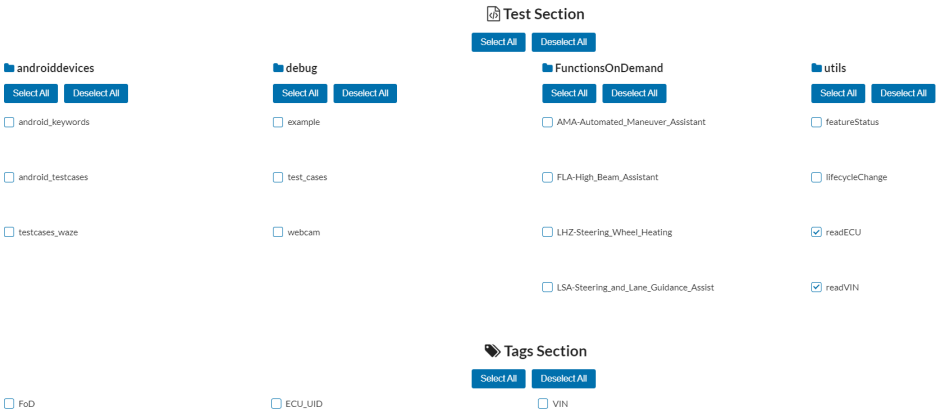


Figura 4.13: *Test Section* e *Tags Section* antes da modificação do repositório

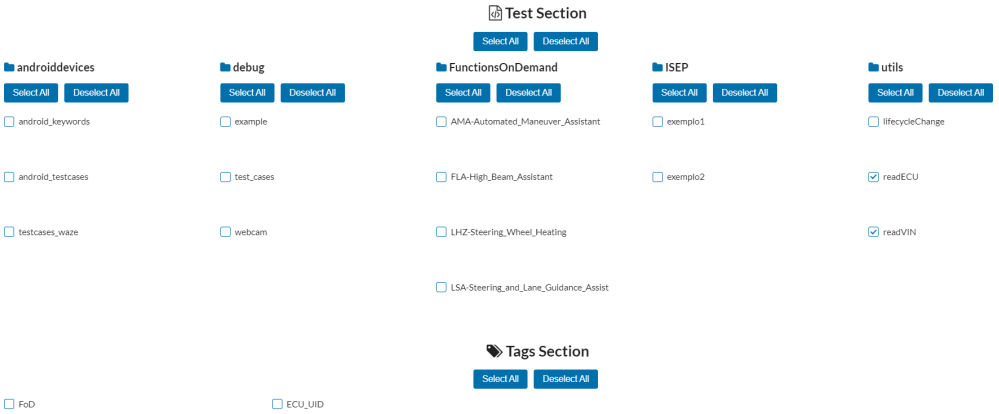


Figura 4.14: *Test Section* e *Tags Section* antes da modificação do repositório

Atualização da Documentação

Sempre que um ficheiro MD é alterado ou adicionado, a ferramenta atualiza automaticamente os resultados da documentação. Isto garante que os utilizadores estejam sempre a par das informações mais recentes. Para demonstrar esta funcionalidade, foi criado um ficheiro de teste, conforme exibido na Figura 4.15:

Figura 4.15: Página da Documentação

Documentação interativa dos *testcases*

Um dos principais destaques da documentação interativa é a capacidade de fornecer informações detalhadas e rápidas ao passar o cursor sobre um teste específico, sem a necessidade de abrir novas páginas ou janelas. A Figura 4.16 demonstra esta funcionalidade: ao posicionar o cursor sobre o teste "readVIN", uma *tooltip* é exibida, apresentando a descrição detalhada do teste, as suas *tags* e a documentação associada.

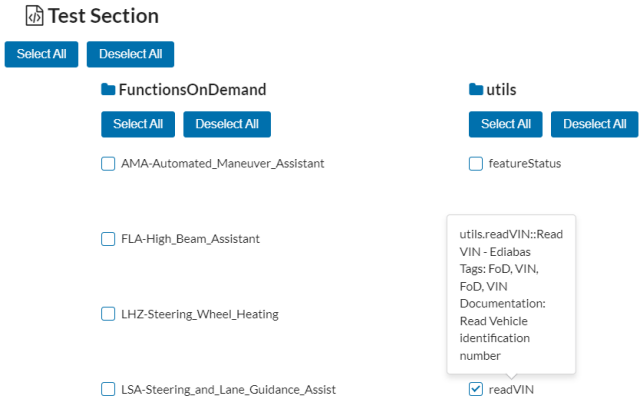


Figura 4.16: Tooltip exibida ao passar o cursor sobre o teste "read-VIN"

Esta funcionalidade não só proporciona uma forma rápida e visual de acessar informações sobre cada teste, mas também ajuda a reduzir o esforço necessário para entender e localizar detalhes específicos. Num ambiente de desenvolvimento dinâmico, onde a agilidade e eficiência são essenciais, esta ferramenta de documentação torna-se uma aliada para desenvolvedores e testadores.

4.2 Impacto e a importância dos resultados

O desenvolvimento de uma aplicação envolve diversas fases, desde a sua conceção até à sua utilização final. Contudo, o sucesso de tal empreitada não é apenas medido pela sua funcionalidade ou eficácia técnica, mas também pela sua capacidade de atender às necessidades do utilizador e adaptar-se às exigências de um mercado em constante evolução. Neste contexto, a análise dos resultados é de fundamental importância, pois proporciona uma compreensão clara do impacto e relevância do que foi desenvolvido.

4.2.1 Impacto Tecnológico

O *front-end* desenvolvido representa uma significativa inovação na forma como os utilizadores interagem com a ferramenta ETAF. Por meio de uma interface gráfica intuitiva, é agora possível gerir e monitorizar as execuções dos testes com precisão, tornando o processo que anteriormente poderia exigir um conhecimento técnico mais aprofundado, em algo mais simplificado.

A incorporação de funcionalidades como a visualização dos *logs* em tempo real, seleção direta dos servidores e a organização clara dos testes, oferece uma melhor experiência ao utilizador. Estas características não só aumentam a produtividade,

como minimizam erros que possam surgir devido a mal-entendidos ou informações perdidas.

4.2.2 Relevância para os Desenvolvedores

Para os desenvolvedores, o *front-end* apresentado é uma ferramenta poderosa. O processo de depuração é facilitado pelo modo de depuração em tempo real, enquanto a estruturação clara das bibliotecas, testes e *tags* permite uma organização lógica e eficiente do trabalho. Além disso, o sistema de atualização do YAML garante que as informações sejam sempre atualizadas, reduzindo as possibilidades de erros humanos.

A possibilidade de visualizar resultados diretamente na interface elimina a necessidade de procurar relatórios em diretórios locais, tornando o processo mais eficiente e centralizado.

A ferramenta oferece a capacidade distintiva de visualizar resultados de testes em tempo real. Este recurso permite que os desenvolvedores recebam feedback instantâneo sobre o desempenho e possíveis falhas do software. Como resultado, é possível uma intervenção rápida: corrigir erros, aprimorar o código e assegurar que o software esteja de acordo com os padrões de qualidade estabelecidos.

4.2.3 Importância Estratégica

Num ambiente empresarial, a eficiência e a precisão são cruciais. Com a integração do logotipo da entidade parceira, Capgemini, a aplicação não só reforça a colaboração entre as partes envolvidas, mas também destaca o compromisso em fornecer soluções de alta qualidade.

Em termos estratégicos, a implementação deste *front-end* pode ser vista como um diferencial competitivo. Empresas que procuram soluções robustas e intuitivas para gestão de testes encontrarão no ETAF uma ferramenta que atende a essas exigências, potencialmente atraindo mais parcerias e clientes.

4.2.4 Benefícios para a Entidade Parceira - Capgemini

A colaboração com a Capgemini, evidenciada pela presença do logotipo e pela integração da ferramenta, destaca a importância da aplicação no contexto empresarial. Ao otimizar o processo de teste e ao garantir uma comunicação eficaz entre as equipas, a aplicação pode resultar em economização de tempo, recursos e, consequentemente, custos para a empresa.

A presença de uma interface amigável e funcional pode tornar a ferramenta ETAF mais atraente para outros clientes ou projetos dentro da Capgemini, promovendo a adoção em larga escala e solidificando a sua posição como uma solução preferencial para testes automatizados.

4.2.5 Limitações e Desafios

Apesar dos muitos benefícios e avanços apresentados, é importante também reconhecer as limitações e desafios enfrentados. Por exemplo, a aplicação ainda está como um servidor de desenvolvimento, o que implica que não está pronta para ambientes de produção em larga escala. A segurança, a escalabilidade e a capacidade de resposta em situações de alta demanda são questões que devem ser abordadas antes da sua implementação em ambientes de produção.

Além disso, a aplicação ainda possui funcionalidades em desenvolvimento, como o botão "Help", que ainda não foi implementado. Isto indica que o projeto, apesar de avançado, continua em fase de evolução e requer mais desenvolvimento e otimização.

Mesmo tendo em conta as limitações e desafios mencionados anteriormente, considera-se que os resultados obtidos com o desenvolvimento deste *front-end* vão além da simples execução de testes. Representam um avanço na forma como as ferramentas de teste são percebidas e utilizadas, destacando a importância de uma boa interface de utilizador e funcionalidades robustas. O impacto deste desenvolvimento será sentido não só pelos utilizadores diretos, mas também a nível estratégico, reforçando a posição da ferramenta ETAF no mercado e potenciando futuras colaborações e inovações.

Capítulo 5

Conclusões

O desenvolvimento da ferramenta ETAF representou um avanço significativo na otimização de testes automatizados, fornecendo uma interface intuitiva e funcionalidades que apoiam os desenvolvedores nos seus esforços para garantir a qualidade do software. Neste documento, foi revisto as fases-chave do projeto, desde o trabalho desenvolvido até aos resultados obtidos. A colaboração com a Capgemini serviu não apenas como validação da aplicação, mas também como uma porta de entrada para a adoção em maior escala no contexto empresarial.

Em retrospetiva, o processo de desenvolvimento do *front-end* integrado da ferramenta ETAF, utilizando tecnologias como Python, Flask, HTML e JavaScript, foi uma jornada desafiadora e enriquecedora. Ao analisar os objetivos estabelecidos no início do projeto, fica evidente que a maioria deles foi alcançada com sucesso. Foram implementadas funcionalidades cruciais para a operação de servidores, gestão e execução de *test cases*, apresentação de *logs*, visualização de resultados e automatização das configurações dos servidores, entre outros. No entanto, a questão da "Guia de Utilização", mesmo sendo parte dos objetivos, pode parecer menos relevante quando comparada com as demais inovações técnicas do projeto. No entanto, é essencial ressaltar que, em projetos de software, a documentação tem uma relevância crucial para garantir a compreensão, manutenção e expansão futura do sistema. O desenvolvimento foi caracterizado por um ciclo iterativo de planeamento, implementação e validação, com a colaboração da Capgemini desempenhando um papel fundamental no aperfeiçoamento e na aceitação da ferramenta no mundo empresarial.

A parceria com a Capgemini não só validou a ferramenta num cenário real de negócios, mas também abriu possibilidades para sua adoção noutros projetos da empresa. A economia de tempo e recursos traduz-se numa vantagem competitiva para empresas que procuram eficiência e qualidade no seu software.

O desenvolvimento de software é, por natureza, um processo evolutivo. À medida que a tecnologia avança e as necessidades dos utilizadores mudam, as ferramentas devem adaptar-se. A ETAF está bem posicionada para ser uma solução de referência em testes automatizados, mas requer um compromisso contínuo com a inovação e atualização.

5.1 Trabalho Futuro

O projeto vai continuar a avançar pós-o término da dissertação, e, portanto, existem diversas áreas de foco já identificadas para melhorias futuras e expansão da ferramenta ETAF:

Melhoramento da Versão: É essencial garantir que a ferramenta esteja sempre atualizada, incorporando as mais recentes tecnologias e atendendo às necessidades emergentes dos utilizadores. O feedback contínuo dos utilizadores pode guiar esse processo, assegurando que as atualizações sejam relevantes e eficazes.

Documentação Interativa dos *Testcases*: Visa-se aprimorar a documentação dos *testcases*, tornando-a mais visualmente apelativa e interativa. Pode-se criar uma interface que não apenas apresenta a informação de forma estruturada, mas também incorpora elementos visuais, tornando a compreensão ainda mais fácil e intuitiva.

Release do *Front-end*: Com recurso ao Nuitka, pretende-se fazer o *release* do *front-end* para poder ser facilmente implementado em ambientes de produção. Este processo garantirá uma integração mais suave e otimizada da ferramenta em diferentes sistemas e infraestruturas.

Guia de Utilização: Pretende-se criar um guia abrangente de utilização, que auxilie os utilizadores na adoção e maximização da ferramenta ETAF. Este guia incluirá documentação em formato MD, detalhando todos os passos e funcionalidades da ferramenta. Além disso, serão desenvolvidos vídeos explicativos, proporcionando uma experiência de aprendizagem multimodal e facilitando a compreensão de processos.

Otimização das Opções de Utilizador: Com a crescente complexidade da ferramenta e as suas opções, torna-se crucial aprimorar e otimizar as opções disponíveis para desenvolvedores. Um foco particular seria na comunicação com o *script* 'run_tests.py', permitindo uma execução mais flexível e intuitiva dos testes. As opções de 'repetitions' (número de repetições de execução) e 'mode' são especialmente importantes, dado que controlam a frequência e o modo como os testes são executados. Esta melhoria proporcionará aos *testers* maior controlo sobre a execução dos testes, permitindo, por exemplo, que um conjunto específico de testes seja repetido várias vezes em diferentes modos.

Automatização mais avançada: Uma vez que a ferramenta ETAF centra-se na execução de testes, seria benéfico explorar níveis mais avançados de automatização. Isso pode incluir integração com ferramentas de CI/CD, permitindo que os testes sejam executados automaticamente a cada *commit* ou *push* num repositório.

Referências

[1] M. Rodriguez, M. Piattini, and C. Ebert, "Software Verification and Validation Technologies and Tools," *IEEE Software*, vol. 36, 2019. [Citado na página 1]

[2] R. S. Pressman, *Software Quality Engineering: A Practitioner's Approach*, vol. 9781118592. 2014. [Citado nas páginas 2 e 10]

[3] L. Crispin and J. Gregory, *Agile Testing: A Practical Guide for Testers and Agile Teams*. Addison-Wesley Professional, 2009. [Citado na página 2]

[4] C. Engineering, "Welcome, on board!," 2022. [Citado na página 4]

[5] C. Engineering, "Capgemini engineering," 2022. [Citado na página 5]

[6] J. Pinheiro, "Software development life cycle (sdlc) phases," 2018. [Citado nas páginas ix e 10]

[7] I. Sommerville, *Software Engineering (9th ed.; Boston, Ed.). Massachusetts: Pearson Education.* 2011. [Citado na página 10]

[8] M. Cohn, *Agile Estimating and Planning*. Prentice Hall Professional, 2005. [Citado na página 11]

[9] L. Khong, L. Yu Beng, T. Yip, and T. Soofun, "Software development life cycle agile vs traditional approaches," Feb. 2012. [Citado na página 11]

[10] J. Rasmusson, *The Agile Samurai: How Agile Masters Deliver Great Software*. Pragmatic Bookshelf, 2010. [Citado na página 12]

[11] A. Tierno, M. M. Santos, B. A. Arruda, and J. N. H. da Rosa, "Open issues for the automotive software testing," in *2016 12th IEEE International Conference on Industry Applications (INDUSCON)*, pp. 1–8, IEEE, 2016. [Citado na página 12]

[12] "V-model (software engineering) - javatpoint." [Citado nas páginas ix e 13]

[13] S. Mathur and S. Malik, "Advancements in the V-Model," *International Journal of Computer Applications*, vol. 1, no. 12, pp. 30–35, 2010. [Citado na página 13]

[14] J. Rubin and D. Chisnell, *Handbook of Usability Testing: How to Plan, Design, and Conduct Effective Tests*. Hoboken, NJ: John Wiley & Sons, 2 ed., 2012. [Citado na página 15]

[15] D. Dhaka, "Requisitos rastreabilidade matriz modelos ppt," 2022. [Citado nas páginas ix e 15]

[16] M. A. Umar and Z. Chen, "A study of automated software testing: Automation tools and frameworks," *International Journal of Computer Science Engineering (IJCSE)*, vol. 8, 2019. [Citado na página 16]

[17] A. Spillner, T. Linz, and H. Schaefer, *Software Testing Foundations(4th Edition)*. 2014. [Citado nas páginas 17 e 21]

[18] R. Sablatnig, W. Kropatsch, and A. Hanbury, *Lecture Notes in Computer Science: Preface*, vol. 3663. 2005. [Citado na página 18]

[19] LambdaTest, "Unit testing tutorial: A comprehensive guide with examples and best practices." Next-Generation Mobile Apps and Cross Browser Testing Cloud. [Citado nas páginas ix e 19]

[20] N. Alshahwan, "Automated Session Data Repair for Web Application Regression Testing," pp. 298–307, 2008. [Citado na página 20]

[21] A. M. Memon, "Automatically Repairing Event Sequence-Based GUI Test Suites for Regression Testing," vol. 18, no. 2, 2008. [Citado nas páginas 20 e 21]

[22] G. Legramante, M. Bernardino, E. M. Rodrigues, and F. Basso, "Systematic Literature Review on Web Performance Testing," pp. 285–295, 2021. [Citado na página 22]

[23] L. Tan, W. Shang, and T. Xie, "Performance testing and optimization of big data software: A systematic literature review," *Journal of Systems and Software*, vol. 120, pp. 28–49, 2016. [Citado na página 22]

[24] G. J. Myers, B. Morgan, and C. Sandler, *The Art of Software Testing*. John Wiley & Sons, 2015. [Citado na página 23]

[25] D. Burns, *Selenium 2 Testing Tools: Beginner's Guide*. Packt Publishing Ltd, 2012. [Citado na página 23]

[26] E. Dustin, J. Rashka, and J. Paul, *Automated Software Testing: Introduction, Management, and Performance*. Addison-Wesley Professional, 2009. [Citado na página 25]

[27] M. Fewster and D. Graham, *Automated Software Testing: Adding Value and Delivering Results*. Addison-Wesley Professional, 2001. [Citado na página 25]

[28] D. M. Rafi, K. R. K. Moses, K. Petersen, and M. V. Mäntylä, "Benefits and limitations of automated software testing: Systematic literature review and practitioner survey," in *2012 7th International Workshop on Automation of Software Test (AST)*, IEEE, 2012. [Citado na página 25]

[29] J. Humble and D. Farley, *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Addison-Wesley Professional, 2010. [Citado na página 26]

[30] "Tiobe index for march 2023," 2023. [Citado na página 28]

[31] "Robot framework," 2023. [Citado na página 28]

[32] *Robot Framework User Guide*, 2018. Accessed on September 20, 2018. [Citado na página 28]

[33] J. Duckett, *HTML and CSS: Design and Build Websites*. John Wiley & Sons, 2011. [Citado nas páginas 29 e 30]

[34] E. A. Meyer and E. Weyl, *CSS: The Definitive Guide*. O'Reilly Media, Inc., 2012. [Citado na página 30]

[35] D. Flanagan, *JavaScript: The Definitive Guide: Activate Your Web Pages*. O'Reilly Media, Inc., 2011. [Citado na página 31]

[36] D. Crockford, *JavaScript: The Good Parts*. O'Reilly Media, Inc., 2008. [Citado na página 31]

[37] M. Grinberg, *Flask Web Development: Developing Web Applications with Python*. O'Reilly Media, Inc., 2018. [Citado na página 32]

[38] A. Ronacher, "Introducing flask," *Python Software Foundation*, 2010. [Citado na página 32]

[39] O. B.-K. Clark Evans, Ingy döt Net, "Yaml ain't markup language (yaml) version 1.2." `https://yaml.org/spec/1.2/spec.html`, 2009. [Citado na página 32]

[40] Microsoft, "Windows powershell," 2021. [Citado na página 33]

[41] H. Deshev, *Pro Windows PowerShell*, vol. 3. 2018. [Citado na página 33]

[42] J. Gruber, "Markdown," 2004. Acessado em: [data de acesso]. [Citado na página 34]

[43] "Sobre o git," 2022. [Citado na página 35]

[44] "Gitlab documentation," 2022. [Citado na página 36]

[45] "Jira software documentation," 2023. [Citado na página 36]

[46] D. Ganea, R. Bogdan, V. Ancusa, and M. Popa, "A case study of automated testing implementation in the automotive industry," 2013. [Citado na página 39]

[47] Tracetronic, "ECU-Test Overview," 2022. [Citado na página 40]

[48] P. E. Lanigan, P. Narasimhan, and T. E. Fuhrman, "Experiences with a CANoe-based fault injection framework for AUTOSAR," *Proceedings of the International Conference on Dependable Systems and Networks*, pp. 569–574, 2010. [Citado na página 41]

[49] Vector, "Product Information CANoe," no. 14998, pp. 1919–1920, 2023. [Citado na página 41]

[50] A. Ronacher, "Flask documentation." `https://flask.palletsprojects.com/`, 2021. [Citado nas páginas 56 e 58]

[51] O. B.-K. Clark Evans, Ingy döt Net, *YAML Ain't Markup Language (YAML™) Version 1.2*. 2009. [Citado na página 60]

[52] K. Simonov, "Pyyaml documentation." `https://pyyaml.org/wiki/PyYAMLDocumentation`, 2021. [Citado na página 62]