UNIVERSIDADE EDUARDO MONDLANE

FACULDADE DE ENGENHARIA

LICENCIATURA EM ENGENHARIA INFORMÁTICA

Proposta de Automação de Testes Funcionais no Sistema de Recrutamento Headhunting

Caso de estudo: **Techsolutions, Lda**

Autor:

Manhiça, Lourenço Nelson

Supervisor:

Engº. Cristialino Maculuve

Supervisor da Instituição:

Hélio Herculiano Mocumbi

Maputo, Junho 2025

UNIVERSIDADE EDUARDO MONDLANE

FACULDADE DE ENGENHARIA

LICENCIATURA EM ENGENHARIA INFORMÁTICA

Proposta de Automação de Testes Funcionais no Sistema de Recrutamento Headhunting

Caso de estudo: **Techsolutions, Lda**

Autor:

Manhiça, Lourenço Nelson

Supervisor:

Engº. Cristialino Maculuve

Supervisor da Instituição:

Hélio Herculiano Mocumbi

Maputo, Junho 2025

UNIVERSIDADE
EDUARDO
MONDLANE

UNIVERSIDADE EDUARDO MONDLANE

FACULDADE DE ENGENHARIA

DEPARTAMENTO DE ENGENHARIA ELECTROTÉCNICA

TERMO DE ENTREGA DE RELATÓRIO DO ESTÁGIO PROFISSIONAL

Declaro que o estudante **Lourenço Nelson Manhiça** entregou no dia __ / 06 / 2025 as 2 cópias do relatório do seu Estágio Profissional com a referência: _____ intitulado: Proposta de Automação de Testes Funcionais no Sistema de Recrutamento Headhunting (Caso de Estudo: Techsolutions, Lda).

Maputo, __ de Junho de 2025

O chefe da Secretaria

_____

UNIVERSIDADE EDUARDO MONDLANE

FACULDADE DE ENGENHARIA

DEPARTAMENTO DE ENGENHARIA ELECTROTÉCNICA

DECLARÇÃO DE HONRA

Declaro sob compromisso de honra que o presente trabalho é resultado da minha investigação e que foi concebido para ser submetido apenas para a obtenção do grau de Licenciatura em Engenharia Informática na Faculdade de Engenharia da Universidade Eduardo Mondlane.

Maputo, __ de Junho de 2025

O Autor

_____

(Lourenço Nelson Manhiça)

Dedicatória

Aos meus pais Nelson Manhice e Anita Vilanculos

Aos demais familiares e amigos

Agradecimentos

Em primeiro lugar, expresso a minha profunda gratidão a Deus, pela dádiva da vida, pela sabedoria, pela força e pela resiliência que me concedeu ao longo deste percurso académico, sustentando-me tanto nos momentos de superação como nas fases de maior adversidade.

Agradeço profundamente à minha família, em especial aos meus pais, Nelson Manhice e Anita Vilanculos, pelo amor incondicional, pelo encorajamento constante e pela confiança que sempre depositaram em mim, mesmo quando me vi rodeado de incertezas.

Ao meu supervisor, Eng.º Cristialino Maculuve, manifesto o meu sincero agradecimento pela orientação, disponibilidade e compreensão demonstradas ao longo de todas as fases deste trabalho.

Aos meus colegas e amigos, agradeço pelas partilhas, pelo companheirismo e pelo apoio contínuo que marcaram a minha caminhada académica.

Aos docentes, expresso a minha gratidão pelo conhecimento transmitido, pela exigência académica e pelo contributo fundamental para a minha formação técnica e pessoal.

Por fim, agradeço à empresa Techsolutions pela oportunidade de realização do estágio profissional, que possibilitou a aplicação prática dos conhecimentos adquiridos e a concretização deste trabalho.

Epígrafe

"Falhei mais de 9000 lançamentos ao longo da minha carreira. Perdi quase 300 jogos. Em 26 ocasiões, confiaram-me o lançamento decisivo... e falhei. Fracassei vezes sem conta na minha vida. E é precisamente por isso que tenho sucesso."

— Michael Jordan

Resumo

O presente trabalho descreve o desenvolvimento de uma solução de automação de testes funcionais aplicada à plataforma de recrutamento Headhunting, desenvolvida pela empresa Techsolutions, Lda, no âmbito de um estágio profissional. A necessidade de implementar esta solução surgiu face aos constrangimentos identificados no processo manual de validação do sistema, nomeadamente a morosidade, a repetitividade, a susceptibilidade a erros humanos e a dificuldade em garantir escalabilidade e cobertura abrangente dos testes.

Para responder a este desafio, procedeu-se à adopção do Playwright, um framework moderno de automação de testes, reconhecido pela sua arquitectura optimizada, suporte a múltiplos navegadores e elevada fiabilidade. A solução desenvolvida substitui parte dos testes manuais por scripts de automação, capazes de simular interacções reais dos utilizadores com a aplicação, de forma rápida, consistente e reproduzível.

O trabalho inclui uma revisão teórica sobre qualidade de software, tipos e níveis de testes, bem como uma análise comparativa entre as ferramentas Selenium, Cypress e Playwright, que fundamentou a selecção da ferramenta adoptada. A implementação da solução baseou-se na definição de cenários de teste utilizando a notação Gherkin e no desenvolvimento de scripts em JavaScript, através do Playwright.

Os resultados obtidos demonstram melhorias significativas na produtividade, na fiabilidade dos testes e na cobertura dos fluxos críticos do sistema. A automação revelou-se, assim, uma estratégia eficaz para optimizar o processo de validação, garantir a qualidade do produto final e responder de forma mais eficiente às exigências operacionais e aos desafios próprios de ambientes de desenvolvimento ágeis e dinâmicos.

Palavras chaves: Automação de Testes, Testes Funcionais, Playwright, Qualidade de Software, Gherkin.

Abstract

This paper describes the development of a functional test automation solution applied to the Headhunting recruitment platform, developed by Techsolutions, Lda, as part of a professional internship. The need to implement this solution arose from constraints identified in the manual validation process, including slowness, repetitiveness, susceptibility to human error, and difficulties in ensuring scalability and comprehensive test coverage.

To address this challenge, the Playwright framework was adopted, a modern automation tool recognised for its optimised architecture, multi-browser support, and high reliability. The implemented solution replaced part of the manual testing with automated scripts capable of simulating real user interactions with the system in a fast, consistent, and repeatable manner.

This study includes a theoretical review of software quality, testing types and levels, as well as a comparative analysis of the tools Selenium, Cypress, and Playwright, which supported the selection of the adopted framework. The implementation was based on the definition of test scenarios using the Gherkin notation and the development of scripts in JavaScript, leveraging the Playwright platform.

The results demonstrate significant improvements in productivity, test reliability, and coverage of the system's critical workflows. Therefore, automation proved to be an effective strategy for optimising the validation process, ensuring product quality, and responding more efficiently to operational demands and the challenges of dynamic and agile development environments.

Keywords: Test Automation, Functional Testing, Playwright, Software Quality, Gherkin Syntax.

Índice

Lista de Figuras

Lista de Tabelas

Lista de abreviaturas e acrónimos

Sigla/Acrónimo	Significado
ABNT	Associação Brasileira de Normas Técnicas
API	Application Programming Interface (Interface de Programação de Aplicações)
CI/CD	Continuous Integration / Continuous Delivery (Integração/Entrega Contínua)
CCTV	Closed-Circuit Television (Circuito Fechado de Televisão)
CV	Curriculum Vitae
E2E	End-to-End (De ponta a ponta)
IDE	Integrated Development Environment (Ambiente de Desenvolvimento Integrado)
IEEE	Institute of Electrical and Electronics Engineers
ISO	International Organization for Standardization
JS	JavaScript
LDA	Limitada (forma jurídica de empresa)

NBR	Norma Brasileira
RC	Remote Control (usado no contexto do Selenium RC)
RF	Requisito Funcional
SAL	Nome da empresa cliente da plataforma Headhunting
SA	Sociedade Anónima
TS	TypeScript
UEM	Universidade Eduardo Mondlane
UI/UX	User Interface / User Experience
VS Code	Visual Studio Code
W3C	World Wide Web Consortium

Glossário de Termos

Automação de Testes	Processo de uso de ferramentas e scripts para executar testes de software automaticamente, reduzindo o esforço manual e aumentando a eficiência.
Testes Funcionais	Testes que verificam se as funcionalidades do sistema operam conforme os requisitos definidos, sem considerar a estrutura interna do código.
Testes de Regressão	Testes que garantem que modificações no sistema (correções ou novas funcionalidades) não causaram falhas em funcionalidades já existentes.
Playwright	Framework de automação de testes de aplicações web, desenvolvido pela Microsoft, com suporte a múltiplos navegadores e linguagens.
Gherkin	Linguagem de formatação usada para escrever cenários de testes de forma legível para humanos, utilizada em testes baseados em comportamento (BDD).
Script de Teste	Conjunto de instruções automatizadas criadas para verificar funcionalidades específicas de um software.

Testes Manuais	Procedimentos de verificação realizados por humanos, sem uso de ferramentas de automação, geralmente repetitivos e sujeitos a erros.
Teste de Unidade	Verificação isolada de pequenas partes do código (ex.: funções ou métodos) para garantir seu correto funcionamento.
Teste de Integração	Testes que avaliam a interação entre diferentes módulos ou componentes do sistema.
Teste de Sistema	Verificação completa do sistema como um todo, validando se o comportamento geral atende aos requisitos definidos.
Teste de Aceitação	Testes finais realizados pelo cliente para validar se o sistema está pronto para ser usado em ambiente real.
Teste de Usabilidade	Avaliação da facilidade com que um utilizador interage com o sistema, focando na experiência do utilizador.
Framework	Estrutura reutilizável de código usada como base para o desenvolvimento ou teste de aplicações.
Interface do Utilizador (UI)	Parte visual do sistema com a qual o utilizador interage, composta por botões, menus, formulários, etc.

Requisito Funcional (RF)	Descrição de funcionalidades que o sistema deve realizar, como login, cadastro ou envio de mensagens.
Notação Gherkin	Forma padronizada de descrever comportamentos do sistema em linguagem natural estruturada (ex.: “Dado que... Quando... Então...”).
Test Runner	Componente responsável por executar os scripts de teste e apresentar os resultados da execução.
WebSocket	Protocolo de comunicação que permite troca de dados em tempo real entre cliente e servidor em conexões persistentes.
Headless Mode	Modo de execução de testes em que o navegador não é exibido graficamente, usado para agilizar os testes em ambientes automatizados.
CI/CD	Práticas de Integração e Entrega Contínuas que automatizam o processo de integração de código e distribuição de software.

1. Capitulo I – Introdução

1.1. Contextualização

Testar um sistema informático consiste em verificar se o mesmo se comporta de acordo com os requisitos previamente definidos. Trata-se de uma actividade essencial para garantir que a aplicação funcione correctamente em diversos cenários e condições de utilização. Com o avanço das tecnologias e a crescente digitalização dos processos, os sistemas de informação tornaram-se cada vez mais centrais para o funcionamento das organizações, tanto públicas como privadas, servindo de suporte às operações críticas e à tomada de decisões estratégicas.

Esta crescente dependência de sistemas digitais impõe exigências cada vez mais elevadas em termos de qualidade, estabilidade e desempenho. Tal realidade torna indispensável a adopção de práticas eficazes de verificação e validação. Neste contexto, os testes de software assumem um papel fundamental, pois garantem que os sistemas satisfaçam tanto os requisitos técnicos como as expectativas dos utilizadores.

Os testes podem ser classificados segundo diferentes critérios, sendo uma das categorias mais relevantes os testes funcionais. Segundo Leloudas (2023), este tipo de teste visa validar se o software está a funcionar conforme o esperado e em conformidade com os requisitos definidos. Os testes funcionais concentram-se, portanto, na avaliação do comportamento do sistema face às suas especificações funcionais.

Um método amplamente utilizado para testar aplicações web é o teste de ponta a ponta (end-to-end, ou E2E). Segundo a Microsoft, este tipo de teste verifica o fluxo funcional e de dados de uma aplicação, abrangendo os diversos subsistemas envolvidos numa operação completa, desde o início até ao fim. O seu principal objectivo é simular o comportamento real do utilizador, assegurando que as funcionalidades operam correctamente quando integradas num fluxo completo de utilização.

Tradicionalmente, estes testes são realizados de forma manual, com os testadores a percorrerem, passo a passo, os fluxos da aplicação. No entanto, a execução manual apresenta diversas limitações, especialmente em projectos de média e grande dimensão.

É neste contexto que se insere o presente trabalho, desenvolvido no âmbito do estágio profissional realizado na empresa Techsolutions, responsável pela criação da plataforma Headhunting, um sistema de recrutamento actualmente em fase final de desenvolvimento. Durante o estágio, verificou-se que os testes da plataforma estavam a ser executados de forma manual, originando diversos constrangimentos práticos.

Face a esta realidade, propõe-se a implementação de uma solução de automação de testes funcionais, com recurso a ferramentas de automação, tendo como foco a automação de fluxos críticos da plataforma. Esta abordagem visa não apenas reduzir o esforço associado à execução manual, mas também garantir testes mais rápidos, fiáveis, reproduzíveis e com maior cobertura, contribuindo significativamente para a qualidade do producto final e a eficiência da equipa de desenvolvimento.

1.2. Motivação

Segundo Gil (2017), as razões que motivam a realização de uma pesquisa podem ser agrupadas em dois grandes domínios, razões de ordem intelectual, associadas ao desejo de conhecer pelo próprio acto de conhecer, e razões de ordem prática, orientadas para a procura de soluções que permitam realizar algo de forma mais eficiente ou eficaz.

Neste trabalho, a motivação insere-se predominantemente no domínio das razões práticas. A proposta de desenvolvimento de uma solução de automação de testes funcionais utilizando um framework de automação surge da constatação, durante o estágio profissional na empresa Techsolutions, das dificuldades enfrentadas no processo de validação manual da plataforma de recrutamento Headhunting.

Neste contexto, a automação de testes surge como uma mais-valia para a empresa, permitindo reduzir drasticamente o esforço manual. A adopção desta solução permitirá ainda acelerar os ciclos de desenvolvimento, reduzir os custos operacionais e mitigar o

risco de erros que poderiam impactar negativamente a experiência dos utilizadores finais da plataforma.

Para além de responder directamente às necessidades identificadas na validação do sistema Headhunting, a solução desenvolvida poderá ser reaproveitada futuramente noutros projectos da empresa, contribuindo para a melhoria contínua dos processos internos, para o aumento da produtividade da equipa e para a entrega de soluções tecnológicas com padrões mais elevados de qualidade e fiabilidade.

1.3. Definição do Problema

A validação de sistemas informáticos constitui uma etapa crítica no ciclo de desenvolvimento de software, sendo essencial para assegurar que as funcionalidades implementadas correspondem aos requisitos previamente definidos e se comportam de forma correcta em diferentes cenários de utilização.

No contexto actual da Techsolutions, a validação funcional da plataforma de recrutamento Headhunting tem sido realizada manualmente, recorrendo a processos repetitivos e dependentes de intervenção humana. Esta abordagem, embora comum em fases iniciais de desenvolvimento, apresenta várias limitações que comprometem a eficiência, a fiabilidade e a escalabilidade do processo de testes.

O processo manual consiste na execução recorrente dos fluxos da aplicação, como preenchimento de formulários, submissão de dados e navegação entre funcionalidades. Esta realidade tem originado constrangimentos operacionais significativos, nomeadamente a morosidade na execução dos testes, o elevado risco de erros humanos, dificuldades na cobertura adequada de cenários, entre outros.

A inexistência de um mecanismo automatizado para validação das funcionalidades tem tornado o processo de testes pouco sustentável, tanto em termos de tempo como de recursos. A dependência de execução manual não só impacta os prazos de entrega, como também compromete a qualidade do producto, uma vez que algumas falhas podem não ser detectadas atempadamente.

Estes factores tornam evidente a necessidade de uma solução mais robusta, que permita à equipa realizar a validação funcional da plataforma de forma mais eficiente, consistente e escalável.

1.4. Objetivos

1.4.1. Objetivo Geral

- Propor uma solução de automação de testes funcionais para o sistema de recrutamento Headhunting

1.4.2. Objetivos Específicos

- Descrever os benefícios proporcionados pela automação de testes funcionais;
- Selecionar a ferramenta de automação mais adequada para solução;
- Mapear os requisitos funcionais do sistema sob a forma de funcionalidades e cenários de teste, utilizando a notação Gherkin;
- Desenvolver scripts de automação de testes para as funcionalidades e cenários mapeados.

1.5. Metodologia

Secundo Fonseca (2002, como citado em Gerhardt & Silveira, 2009), methodos significa organização, e logos, estudo sistemático, pesquisa, investigação, ou seja, metodologia é o estudo da organização, dos caminhos a serem percorridos, para se realizar uma pesquisa ou um estudo, ou para se fazer ciência. Etimologicamente, significa o estudo dos caminhos, dos instrumentos utilizados para fazer uma pesquisa científica.

Nesta secção, apresenta-se a abordagem metodológica adotada para a realização do presente trabalho, descrevendo-se as etapas seguidas com vista à obtenção de respostas e soluções para o problema identificado.

1.5.1. Metodologia de Pesquisa

A pesquisa é definida como um procedimento racional e sistemático que tem como objectivo proporcionar respostas aos problemas que são propostos. A pesquisa desenvolve-se ao longo de um processo que envolve inúmeras fases, desde a adequada formulação do problema até a satisfatória apresentação dos resultados. (Gil, 2017)

1.5.1.1. Classificação da Metodologia

Segundo Gerhardt e Silveira (2009), a pesquisa pode ser classificada de acordo com diversos critérios, nomeadamente: quanto à abordagem, quanto à natureza, quanto aos objectivos e quanto aos procedimentos.

De acordo com esses critérios esse trabalho classifica-se em:

- **Quanto a Abordagem**

Qualitativo: Segundo Gerhardt e Silveira (2009), a pesquisa qualitativa não se preocupa com representatividade numérica, mas, sim, com o aprofundamento da compreensão de um grupo social, de uma organização. No presente trabalho, adoptou-se uma abordagem qualitativa, pois pretende-se compreender e actuar sobre um processo específico no contexto do estágio profissional.

- **Quanto a Natureza**

Pesquisa Aplicada: Este trabalho baseou-se numa pesquisa aplicada, que, segundo Gil (2017), são pesquisas voltadas à aquisição de conhecimentos com vistas à aplicação numa situação específica.

- **Quanto aos objectivos**

O trabalho tem objectivos exploratórios e descritivos. Segundo Gil (2017), as pesquisas exploratórias têm como propósito proporcionar maior familiaridade com o problema, com vistas a torná-lo mais explícito ou a construir hipóteses.

As pesquisas descritivas têm como objetivo a descrição das características de determinada população ou fenômeno. São em grande número as pesquisas que podem

ser classificadas como descritivas e a maioria das que são realizadas com objetivos profissionais provavelmente se enquadra nestas categorias. (Gil, 2017)

- **Quanto aos procedimentos**

Para alcançar os objectivos traçados no presente trabalho, foram definidos os seguintes procedimentos:

- **Pesquisa bibliográfica:** a pesquisa foi realizada com base em materiais já publicados, como livros, artigos científicos e outros trabalhos relacionados com o tema em estudo.
- **Estudo de caso:** o escopo do trabalho foi delimitado com base no contexto da empresa Techsolutions, que será utilizada como caso de estudo.
- **Levantamento e priorização de requisitos do sistema:** nesta etapa, foram realizadas sessões presenciais com os desenvolvedores da Techsolutions, com o objectivo de levantar os requisitos funcionais do sistema em estudo e identificar os mais críticos.
- **Implementação:** após a definição dos requisitos, foi realizado o desenvolvimento de scripts de automação utilizando um framework adequado para esse fim.
- **Ferramenta tecnológica:** para a implementação da solução proposta, foi utilizado o Playwright, um framework de automação de testes desenvolvido pela Microsoft. A linguagem de programação utilizada será o JavaScript, e o ambiente de desenvolvimento será o Visual Studio Code.

  Foram também utilizados os navegadores Chromium, WebKit (Safari) e Firefox para realizar os testes do sistema.
- **Consultas aos supervisores:** durante o desenvolvimento do trabalho, os supervisores foram consultados periodicamente para fins de orientação, esclarecimento de dúvidas, apresentação de progresso e outras questões pertinentes.

1.6. Estrutura do Trabalho

O presente trabalho é composto por seis capítulos, devidamente enumerados e duas secções não enumeradas referentes a bibliografia e aos anexos. A seguir é apresentada a descrição de cada uma das partes constituintes do trabalho:

- **Capítulo I - Introdução**

Neste capítulo são apresentados aspectos introdutórios do trabalho, a contextualização, definição do problema, motivação, definição dos objectivos e a metodologia utilizada na realização do trabalho.

- **Capítulo II - Revisão de Literatura**

Neste capítulo são apresentadas em sequência lógica voltada ao tema, descrições teóricas referentes a todos aspectos considerados importantes para o desenrolar do tema.

- **Capítulo III - Caso de Estudo**

Neste capítulo é apresentado o caso de estudo, Techsolutions, onde foi possível descrever a empresa, o sistema Headhunting em estudo e a sua situação actual, constrangimentos enfrentados.

- **Capítulo IV - Desenvolvimento da Solução**

Neste capítulo após a apresentação clara e precisa do problema, e de revisão de literaturas relacionadas a este, faz-se a descrição da solução proposta e apresenta-se detalhadamente as fases para o desenvolvimento da solução.

- **Capítulo V - Apresentação dos Resultados**

Neste capítulo é feita a apresentação dos resultados da solução proposta.

- **Capítulo VI - Discussão dos Resultados**

Neste capítulo são encontradas as conclusões e recomendações, o tema é encerado, são apresentadas recomendações para pesquisas futuras e as formas através pelas quais o presente trabalho pode ser útil**.**

- **Referencias bibliográficas**

Nesta secção são mostradas todas fontes que permitiram a elaboração do projecto bem como do relatório;

- **Apêndice**

Nesta secção são apresentados os apêndices que complementam e aprofundam o conteúdo desenvolvido ao longo deste trabalho

2. Capitulo II – Revisão da Literatura

2.1. A Influência da Tecnologia na Sociedade e nas Organizações

A presença da tecnologia tornou-se indispensável em praticamente todas as dimensões da vida contemporânea, abrangendo desde as actividades quotidianas até aos processos internos das organizações. Dispositivos como computadores e telemóveis constituem actualmente instrumentos fundamentais para o acesso a serviços, comunicação e gestão de actividades diversas.

No âmbito organizacional, esta evolução reflecte-se numa transformação digital intensa, motivada pela necessidade de reforçar a competitividade num mercado progressivamente mais exigente. As organizações, pressionadas por um mercado cada vez mais exigente, têm vindo a investir em soluções informatizadas com o objectivo de optimizar os seus processos, reduzir custos e tomar decisões de forma mais eficiente e eficaz.

Conforme observa Bartié (2002), este fenómeno traduziu-se numa dependência crescente da tecnologia por parte das organizações, cujas operações internas são, cada vez mais, conduzidas e orientadas por sistemas informatizados. Tal tendência, embora estratégica, implica igualmente um acréscimo na complexidade dos sistemas, o que, por sua vez, eleva o risco de falhas. Neste contexto, torna-se imperioso adoptar práticas orientadas para a qualidade no desenvolvimento de software, sendo os testes eficazes uma das estratégias fundamentais para garantir o correcto funcionamento e a fiabilidade dos sistemas.

2.2. Garantia da Qualidade de Software

A Garantia da Qualidade de Software é uma área da engenharia de software que tem como principal objectivo assegurar que o producto desenvolvido cumpre os requisitos estabelecidos e apresenta um desempenho fiável. Trata-se de um conjunto estruturado de práticas que visam acompanhar e avaliar, de forma contínua, todas as fases do ciclo de vida do software, desde a concepção até à entrega final.

Segundo Santos e Oliveira (2017), esta disciplina engloba actividades tanto técnicas como de gestão, sendo fundamental para garantir que os processos de desenvolvimento estão em conformidade com os padrões definidos. Entre essas actividades incluem-se a aplicação de métricas, o uso de ferramentas de análise e especificação, a realização de revisões técnicas formais, a execução de testes em diferentes fases e o controlo de alterações e documentação. Estas práticas permitem detectar falhas de forma precoce e asseguram que o producto final se alinha com os objectivos estabelecidos.

Dentro deste conjunto de actividades, os testes de software assumem um papel central. Segundo Rocha (2023), os testes fazem parte integrante da Garantia da Qualidade, contribuindo directamente para a verificação da conformidade dos processos e productos com os critérios de qualidade definidos.

A realização de testes não é uma actividade isolada, mas uma componente essencial de uma abordagem mais abrangente que visa garantir a fiabilidade, a robustez e a adequação funcional das soluções desenvolvidas.

2.2.1. Qualidade de software

A qualidade de software é um tema amplamente discutido na literactura técnica. Embora existam diferentes interpretações, a maioria dos autores converge para a ideia de que qualidade significa conformidade com requisitos técnicos e expectativas dos utilizadores.

Segundo a IEEE Computer Society, qualidade pode ser entendida como o grau em que um sistema, componente ou processo satisfaz os requisitos especificados ou as necessidades dos utilizadores. Esta definição demonstra que a qualidade não depende apenas dos aspectos técnicos, mas também da experiência do utilizador final.

A norma ABNT NBR ISO 9000:2005 reforça esta ideia, ao definir qualidade como o grau em que um conjunto de características cumpre os requisitos estabelecidos.

Portanto, no desenvolvimento de software, garantir a qualidade implica assegurar que o sistema corresponde às especificações técnicas e responde de forma adequada às

necessidades do seu público-alvo. Para isso, é essencial adoptar práticas que permitam detectar e corrigir falhas ao longo do processo.

Uma das abordagens mais eficazes para esse fim é a realização de testes de software, que funcionam como mecanismos de verificação e controlo de qualidade, contribuindo directamente para a entrega de soluções mais fiáveis, robustas e alinhadas com os objectivos do projecto.

2.3. Testes de software

O conceito de teste de software pode assumir diferentes interpretações, consoante a perspectiva dos profissionais envolvidos no desenvolvimento. Segundo Bartié (2002), embora as equipas de desenvolvimento realizem testes nos seus projectos, nem sempre estas actividades seguem uma metodologia adequada ou estruturada, o que limita a sua eficácia na detecção de erros. Esta limitação está muitas vezes relacionada com a forma como os testes são compreendidos e integrados no processo de desenvolvimento.

O autor distingue duas abordagens principais na forma como os testes de software são tradicionalmente compreendidos. A primeira, mais comum nas equipas de desenvolvimento, reduz os testes a uma simples verificação superficial do funcionamento do sistema, centrando-se apenas na ausência de erros visíveis.

A segunda abordagem, considerada mais adequada por Bartié (2002), defende que os testes devem assumir uma postura activa na procura de falhas, funcionando como um processo técnico rigoroso, planeado e sistemático.

2.3.1. Definição Comum dos Testes

Em muitos contextos de desenvolvimento, os testes de software são entendidos de forma simplificada como um processo destinado apenas a confirmar que o sistema está a funcionar correctamente e sem erros visíveis. Esta abordagem, segundo Bartié (2002), reflecte uma visão excessivamente positiva da actividade de testar, na qual o principal objectivo é provar que tudo decorre como planeado.

Esta interpretação leva muitas equipas a encarar os testes como uma etapa final, quase protocolar, do processo de desenvolvimento. O foco centra-se na validação de comportamentos esperados, desvalorizando a necessidade de procurar activamente situações que possam revelar falhas, inconsistências ou limitações do sistema. Esta postura limita significativamente o potencial dos testes como ferramenta de melhoria da qualidade.

Figura 1 - Simplificação das definições dos testes

Fonte: Reproduzido de Garantia de Qualidade, por A. Bartié, 2002, p.20.

Bartié (2002) observa que é mais comum idealizar cenários positivos e testar apenas os casos em que se espera que o sistema funcione, do que investir na identificação de situações desfavoráveis que possam comprometer a fiabilidade do software. Quando os testes são encarados apenas como confirmação do funcionamento, perde-se a oportunidade de questionar o sistema em profundidade. Este comportamento compromete a objectividade da avaliação, especialmente quando a equipa de testes está demasiado envolvida com o desenvolvimento do projecto.

A verdadeira eficácia dos testes depende, assim, da forma como são concebidos. Se o objectivo for apenas comprovar o funcionamento, o esforço será limitado e direccionado para os casos de sucesso. Mas, se a finalidade for detectar falhas, é necessário ir além dos cenários previsíveis e explorar contextos que desafiem os limites do sistema.

2.3.2. Definição Adequada

A abordagem mais adequada à prática de testes de software exige uma mudança significativa na forma como esta actividade é tradicionalmente compreendida. Em vez de tratar os testes como uma simples verificação do funcionamento esperado, é necessário assumir uma postura mais crítica e investigativa, focada na identificação activa de falhas. Bartié (2002) defende que entender os testes como um meio de provar que algo não está a funcionar correctamente representa um avanço real no campo da qualidade de software.

Esta mudança de perspectiva implica uma reformulação dos objectivos da actividade de testar. Em vez de limitar-se a confirmar se tudo decorre conforme o previsto, o processo deve explorar cenários que desafiem o comportamento do sistema, incluindo situações-limite, excepções e contextos menos prováveis. Ao adoptar esta abordagem, os testes deixam de ser uma tarefa de rotina e assumem um papel estratégico no desenvolvimento, funcionando como ferramenta de validação, diagnóstico e prevenção de defeitos.

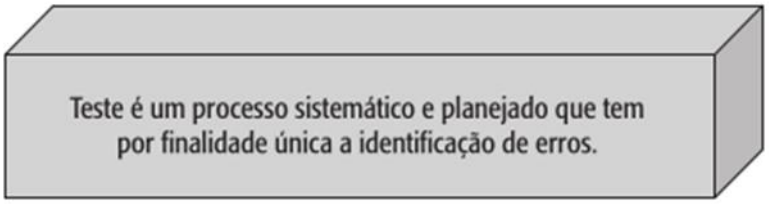

Figura 2 - Definição ampliada de testes

Fonte: Reproduzido de Garantia de Qualidade, por A. Bartié, 2002, p.22.

Rocha (2023) reforça esta visão ao afirmar que testar software é, essencialmente, simular a sua utilização com o propósito de encontrar defeitos. O autor sublinha que os testes não servem para garantir a inexistência de erros, mas apenas para detectar a sua presença. A tentativa de provar que um sistema está isento de falhas é ilusória e pode levar a expectativas irrealistas, especialmente quando se ignora a limitação natural dos testes em termos de cobertura e escopo.

A compreensão destes limites é fundamental para o planeamento eficaz dos testes. Em vez de confiar exclusivamente em cenários favoráveis, as equipas devem conceber estratégias que incluam casos negativos e situações de risco. Este esforço adicional contribui directamente para o aumento da fiabilidade e robustez do produto final, aproximando-o dos requisitos técnicos e das necessidades dos utilizadores.

Ao reconhecer o carácter investigativo e imperfeito do processo de teste, promove-se uma cultura de melhoria contínua, na qual a detecção de erros deixa de ser vista como fracasso e passa a ser encarada como uma oportunidade essencial para elevar a qualidade do software.

2.3.3. Tipos de Testes de Softwares

No processo de desenvolvimento de software, os testes podem assumir diferentes formas, consoante a finalidade e o momento em que são aplicados. Essa diversidade reflecte a complexidade crescente dos sistemas e a necessidade de garantir a sua fiabilidade em múltiplas dimensões. Rocha (2023) propõe uma classificação abrangente que organiza os testes em quatro grandes categorias, de acordo com a sua natureza e o objectivo a que se destinam: testes funcionais, não funcionais, estruturais e testes relacionados a mudanças.

Cada uma destas categorias está associada a técnicas específicas, que devem ser seleccionadas de acordo com os requisitos do sistema e os riscos identificados. Os testes funcionais, por exemplo, centram-se na validação das funcionalidades especificadas, enquanto os testes não funcionais avaliam aspectos como o desempenho, a segurança e a usabilidade. Os testes estruturais, por sua vez, analisam a lógica interna do código, e os testes relacionados a mudanças verificam se alterações introduzidas não provocaram efeitos inesperados noutras partes do sistema.

Esta classificação permite abordar o sistema sob diferentes ângulos, aumentando a probabilidade de identificar defeitos que poderiam passar despercebidos num único tipo de verificação.

Rocha (2023) destaca que estes testes não devem ser encarados como actividades isoladas ou restritas a um momento específico do desenvolvimento, mas como práticas integradas ao longo de todo o ciclo de vida do software. Quando bem planeados, e aplicados, estes testes tornam-se fundamentais para antecipar falhas, garantir a estabilidade do sistema e reduzir custos associados à correcção de erros em fases avançadas.

2.3.3.1. Teste Funcional

Os testes funcionais têm como objectivo verificar se o sistema cumpre correctamente as funcionalidades previstas nos requisitos, avaliando o seu comportamento a partir da perspectiva do utilizador. De acordo com Rocha (2023), este tipo de teste adopta uma abordagem de caixa-preta, na qual se analisam apenas as entradas e saídas do sistema, sem qualquer consideração sobre a sua estrutura interna ou lógica de implementação.

Nesta abordagem, a atenção centra-se exclusivamente nas respostas produzidas pelo sistema em função de determinados dados de entrada. O teste é bem-sucedido se, perante um conjunto de condições previamente definidas, o sistema apresentar os resultados esperados, conforme especificado na documentação de requisitos. Esta estratégia permite avaliar a conformidade funcional da aplicação com os seus objectivos de negócio, independentemente de como as funcionalidades são implementadas no código.

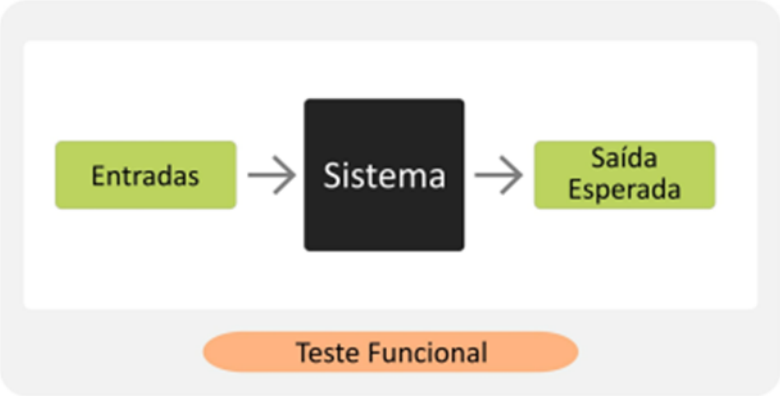

Figura 3 - Modelo representando a execução de um teste funcional

Fonte: Reproduzido de Simplificando Teste de Software, por A. C. Rocha, 2023.

Os testes funcionais simulam o uso real do sistema, tal como seria realizado por um utilizador final. Por essa razão, não exigem o acesso ao código-fonte, sendo orientados por casos de uso e critérios de aceitação. Esta característica torna-os particularmente úteis na validação de interfaces e fluxos de interacção.

2.3.3.2. Teste não funcional

Os testes não funcionais têm como objectivo avaliar atributos do sistema que não se relacionam directamente com a execução de funcionalidades específicas, mas que influenciam a sua qualidade global em contextos reais de utilização. De acordo com Rocha (2023), estes testes permitem verificar se o sistema se comporta de forma adequada em diferentes condições, respeitando critérios como desempenho, segurança, usabilidade, disponibilidade e capacidade de resposta.

Ao contrário dos testes funcionais, que validam "o que" o sistema faz, os testes não funcionais concentram-se em "como" o sistema se comporta, por exemplo, a rapidez com que responde, o grau de segurança que oferece, ou a facilidade com que pode ser utilizado por um novo utilizador.

Figura 4 - Modelo representando testes não funcionais

Fonte: Reproduzido de Simplificando Teste de Software, por A. C. Rocha, 2023.

Entre os atributos mais comuns avaliados por este tipo de testes, destacam-se:

- Desempenho: avalia a resposta do sistema sob diferentes níveis de carga, incluindo o número de utilizadores simultâneos e o volume de dados processados. Este teste ajuda a identificar gargalos, tempos de resposta inadequados e limitações de escalabilidade.
- Segurança: procura identificar vulnerabilidades que possam comprometer a integridade, confidencialidade ou disponibilidade da informação. Inclui a avaliação de processos de autenticação, protecção de dados sensíveis e resistência a acessos não autorizados.
- Usabilidade: verifica se o sistema pode ser utilizado de forma intuitiva por um utilizador comum, sem necessidade de consulta a documentação técnica. Frequentemente, os utilizadores são convidados a realizar tarefas simples, de forma autónoma, para avaliar a facilidade de navegação e compreensão da interface.
- Disponibilidade e fiabilidade: avalia se o sistema mantém um funcionamento estável e contínuo, mesmo sob condições adversas ou durante operações de manutenção.

2.3.3.3. Teste Estrutural

O teste estrutural, também conhecido como teste de caixa-branca, consiste na verificação da lógica interna do sistema, exigindo, por isso, o acesso directo ao código-fonte da aplicação. Segundo Rocha (2023), este tipo de teste tem como principal objectivo identificar defeitos introduzidos durante a programação, assegurando que os componentes internos do sistema funcionam de forma correcta, mesmo após alterações ou adição de novas funcionalidades.

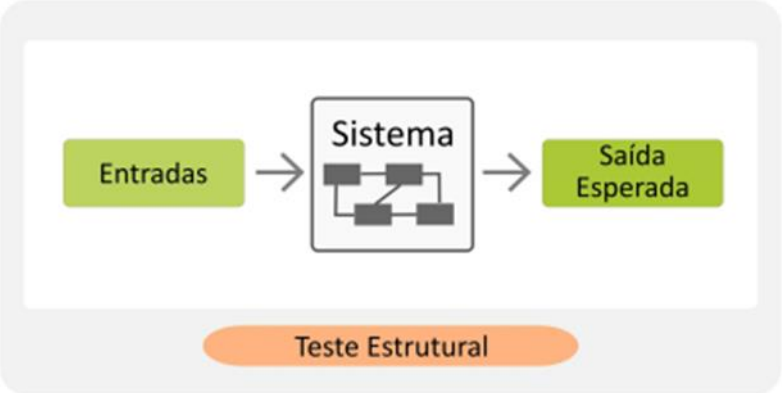

Figura 5 - Modelo representando a execução de um teste estrutural

Fonte: Reproduzido de Simplificando Teste de Software, por A. C. Rocha, 2023.

Ao contrário dos testes funcionais, que avaliam o comportamento do sistema com base nas suas entradas e saídas, os testes estruturais concentram-se na análise detalhada da estrutura lógica do código. Este tipo de teste permite validar desde funções isoladas até componentes integrados, garantindo que o sistema mantém um comportamento consistente à medida que evolui.

Habitualmente, os testes estruturais são realizados pelos próprios programadores, uma vez que exigem conhecimento aprofundado sobre a arquitectura interna do software. No entanto, membros da equipa de testes podem igualmente contribuir para o seu aperfeiçoamento, propondo casos adicionais ou validando cenários que permitam aumentar a cobertura do teste e reduzir o risco de falhas em partes críticas do sistema.

2.3.3.4. Teste Relacionado a Mudanças

Os testes relacionados a mudanças são aplicados quando o sistema sofre modificações, seja pela correcção de defeitos, pela introdução de novas funcionalidades ou por ajustes evolutivos. De acordo com Rocha (2023), o principal objectivo deste tipo de teste é garantir que as alterações efectuadas não introduziram novos problemas nem afectaram negativamente componentes que anteriormente funcionavam de forma correcta.

Figura 6 - Modelo representando execução de testes relacionados à mudança

Fonte: Reproduzido de Simplificando Teste de Software, por A. C. Rocha, 2023.

Este tipo de teste insere-se no contexto da manutenção evolutiva e correctiva do software e é fundamental para assegurar a estabilidade contínua da aplicação ao longo do seu ciclo de vida. A sua função é detectar possíveis efeitos colaterais inesperados resultantes das alterações no código.

Entre as abordagens utilizadas nestes testes, destacam-se o teste de confirmação, que verifica se a correcção de um defeito foi efectivamente bem-sucedida, e o teste de regressão, cuja finalidade é confirmar que funcionalidades previamente testadas continuam a comportar-se correctamente após a realização de modificações.

Rocha (2023) refere que os testes relacionados a mudanças podem combinar técnicas funcionais e estruturais, sendo por vezes designados como testes de caixa-cinza.

2.3.4. Níveis de Testes

Durante o ciclo de vida do desenvolvimento de software, a aplicação de testes ocorre em diferentes momentos e com diferentes objectivos, organizando-se em níveis distintos de verificação e validação. Esta estrutura por níveis permite abordar o sistema em diferentes graus de granularidade, desde o código individual até ao comportamento do sistema como um todo.

O modelo em V é frequentemente utilizado para representar esta organização, estabelecendo uma correspondência directa entre as fases do desenvolvimento e os respectivos níveis de teste. Segundo Rocha (2023), no lado esquerdo do modelo situam-se as etapas de desenvolvimento, nomeadamente o levantamento de requisitos, a especificação, o projecto e a codificação, enquanto no lado direito se encontram os níveis de teste, teste de unidade, teste de integração, teste de sistema e teste de aceitação.

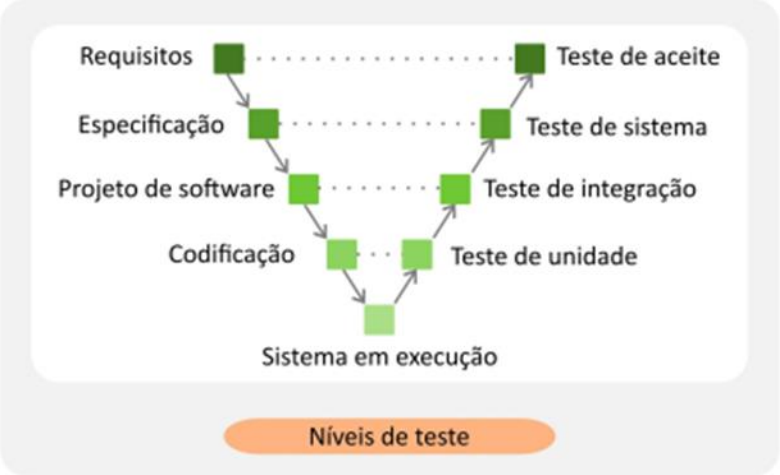

Figura 7 - Modelo V que descreve os níveis de teste

Fonte: Reproduzido de Simplificando Teste de Software, por A. C. Rocha, 2023.

Cada nível de teste está associado a uma fase específica do desenvolvimento, tendo como finalidade validar os artefactos produzidos nessa fase e garantir que o software evolui com qualidade. O teste de unidade, por exemplo, avalia componentes individuais, o teste de integração verifica a interacção entre módulos, o teste de sistema avalia o comportamento do sistema como um todo, e o teste de aceitação confirma que o producto final cumpre os requisitos definidos pelo cliente.

Rocha (2023) sublinha que cada nível de teste é indispensável e cumpre uma função própria, não sendo substituível pelos demais. A combinação estruturada destes níveis assegura que defeitos possam ser detectados em diferentes estágios do processo, aumentando a fiabilidade do producto final.

2.3.4.1. Nível de Teste de Unidade

O teste de unidade, também denominado teste unitário, é realizado durante a fase de codificação e tem como principal objectivo verificar se cada componente individual do sistema funciona correctamente de forma isolada. Este nível de teste foca-se na validação de unidades mínimas de código, antes da sua integração com outros módulos.

Segundo Rocha (2023), os testes de unidade concentram-se em elementos como funções, métodos, classes e módulos, sendo cada um avaliado separadamente no seu ambiente de desenvolvimento. Por esse motivo, este tipo de teste é geralmente executado pelo próprio programador, uma vez que exige conhecimento detalhado sobre a estrutura interna do código.

Os objectos de teste podem incluir o próprio código-fonte, as classes que representam entidades do sistema e os módulos que agrupam funcionalidades com um propósito comum. Ao testar estas unidades individualmente, é possível identificar erros de forma precoce, o que reduz significativamente os custos de correcção e contribui para a fiabilidade global do sistema.

Entre os defeitos mais frequentes detectados neste nível de teste encontram-se:

- Erros de implementação das regras de negócio, que comprometem o comportamento esperado da aplicação;
- Inconsistência de dados, resultante de falhas na comunicação entre componentes;
- Problemas de lógica, relacionados com desvios no fluxo de controlo ou cálculos incorrectos;
- Ausência de tratamento de excepções, o que pode provocar a interrupção inesperada da execução.

A adopção sistemática dos testes de unidade proporciona uma base sólida para os níveis seguintes de validação, permitindo detectar problemas logo nas fases iniciais e assegurando maior estabilidade e confiança no processo de desenvolvimento.

2.3.4.2. Nível de Teste de Integração

O teste de integração constitui uma etapa fundamental no processo de validação do software, tendo como principal finalidade verificar se os diferentes componentes do sistema funcionam de forma correcta quando interligados. Segundo Rocha (2023), este tipo de teste complementa o teste de unidade, ao permitir avaliar a interacção entre módulos distintos e detectar eventuais falhas que só ocorrem durante a comunicação entre partes do sistema.

Ao contrário do teste de unidade, centrado no comportamento isolado de cada componente, o teste de integração analisa como esses componentes colaboram entre si, assegurando que a comunicação entre as interfaces internas (como classes e módulos) e as interfaces externas (como APIs ou serviços Web) decorre sem erros ou interferências.

Este nível de teste pode ser realizado tanto por programadores como por testadores, consoante o contexto do projecto. Em ambientes de desenvolvimento, os próprios programadores podem utilizar ferramentas semelhantes às usadas nos testes unitários.

Entre os problemas mais comuns identificados nesta fase encontram-se:

- Falhas de lógica resultantes da má interpretação dos dados entre módulos;
- Inconsistências na transferência de informação;
- Interrupções causadas por excepções não tratadas; e
- Problemas na comunicação entre o sistema em teste e serviços externos.

Estas falhas, por vezes, passam despercebidas nos testes de unidade e só se manifestam quando os componentes são combinados num cenário de execução real.

2.3.4.3. Nível de Teste de Sistema

O teste de sistema representa um dos últimos níveis de verificação antes da entrega do software ao utilizador final. Tem como objectivo validar se o sistema, já completamente integrado, cumpre correctamente os requisitos funcionais e comportamentais definidos

na fase de especificação. Segundo Rocha (2023), este tipo de teste permite detectar falhas que comprometem o funcionamento global da aplicação, incluindo erros na execução das funcionalidades e incumprimentos das regras de negócio.

Nesta fase, o sistema é testado como um todo, tal como será utilizado em produção. A avaliação é feita sem acesso ao código-fonte, o que permite que este tipo de teste seja realizado por profissionais da equipa de testes ou até por utilizadores finais, em ambientes controlados criados especificamente para esse fim. Estes ambientes procuram simular, com realismo, as condições operacionais reais, sem colocar em risco a estabilidade do sistema em produção.

Durante a execução dos testes de sistema, podem ser identificados diversos tipos de defeitos. Entre os mais comuns encontram-se:

- Erros de execução, que originam excepções e podem interromper o funcionamento do sistema;
- Falhas nas regras de negócio, em que o sistema não responde de forma adequada às condições previamente definidas nos requisitos;
- Inconsistências nos dados, como informações apresentadas de forma incorrecta, omissões ou corrompimento de dados;
- Problemas específicos de ambiente, como erros que apenas ocorrem em determinados navegadores, dispositivos móveis ou versões de sistemas operativos.

A realização deste nível de teste é essencial para assegurar que o software satisfaz os requisitos globais e oferece uma experiência de utilização estável e coerente.

2.3.4.4. Nível de Teste de Aceitação

O teste de aceitação constitui a fase final do processo de validação do software, sendo habitualmente conduzido pelo cliente ou por utilizadores representativos, com o objectivo de verificar se o produto desenvolvido satisfaz os requisitos funcionais acordados. De acordo com Rocha (2023), esta etapa é determinante para confirmar se o sistema está pronto para ser entregue e colocado em ambiente de produção.

Neste nível, o sistema é testado num ambiente de homologação, configurado para simular o contexto real de utilização, mas sem recorrer a dados confidenciais ou de produção. A ideia é permitir ao cliente interagir com a aplicação de forma segura e avaliar se as funcionalidades implementadas correspondem efectivamente às suas expectativas. Esta avaliação pode ser feita com ou sem o apoio de documentação formal, dado que, muitas vezes, o cliente é capaz de reconhecer directamente se os requisitos foram cumpridos.

O teste de aceitação pode assumir diferentes formas, dependendo do momento e do público envolvido. Quando realizado por um grupo restrito de utilizadores em ambiente de pré-produção, é designado teste alfa. Já quando ocorre em ambiente real, com utilizadores finais seleccionados, toma o nome de teste beta.

- Durante esta fase, podem ser detectadas várias categorias de defeitos, tais como:
- Erros de execução, que resultam na interrupção inesperada do sistema;
- Desvios nas regras de negócio, quando as funcionalidades não respondem como especificado;
- Problemas de apresentação ou integridade dos dados, como omissões, duplicações ou formatação incorrecta;
- Dificuldades de usabilidade, quando o utilizador encontra obstáculos ao realizar acções simples;
- Falhas não funcionais, relacionadas com aspectos como segurança, desempenho, disponibilidade ou compatibilidade com dispositivos e navegadores.

O teste de aceitação é, portanto, uma etapa crítica para a validação do produto final. Além de permitir a detecção de erros remanescentes, assegura que o sistema está em conformidade com os objectivos do cliente e que está apto para ser disponibilizado em ambiente de produção com confiança.

2.4. Testes Manuais

Antes da introdução das ferramentas de automação, o teste de softwares era conduzido, quase exclusivamente, de forma manual. Neste modelo tradicional, os testadores

executavam cada passo dos casos de teste de forma interactiva, com o objectivo de detectar falhas e verificar a conformidade funcional do sistema. Embora este processo permitisse uma análise detalhada do comportamento da aplicação, exigia mais tempo, maior alocação de recursos e estava sujeito a erros humanos. Segundo Pathak (2025), apesar destas limitações, o teste manual continua a desempenhar um papel relevante, sobretudo em contextos onde a observação directa da experiência do utilizador é essencial.

O teste manual envolve um conjunto estruturado de actividades que visam garantir que o software satisfaz os requisitos funcionais e técnicos definidos. As fases mais comuns deste processo incluem:

- **Análise dos requisitos**: interpretação e compreensão aprofundada dos objectivos do projecto, das funcionalidades esperadas e das necessidades dos utilizadores, servindo de base para a elaboração da estratégia de teste;
- **Planeamento**: definição dos objectivos, escopo, recursos e prazos para a execução dos testes, bem como das abordagens e técnicas que serão utilizadas;
- **Concepção dos casos de teste**: elaboração de cenários detalhados que descrevem os passos a seguir, os dados de entrada, os resultados esperados e as condições necessárias para a sua execução;
- **Preparação do ambiente de teste**: configuração de todos os elementos necessários, como hardware, software, rede e dados simulados, garantindo que o ambiente reproduz as condições reais de utilização;
- **Execução dos testes**: realização manual dos passos definidos nos casos de teste, interagindo directamente com a aplicação, de forma a observar comportamentos e resultados;
- **Registo de defeitos**: documentação rigorosa de quaisquer anomalias encontradas, incluindo informações que permitam a sua reprodução, como capturas de ecrã, mensagens de erro ou registos de sistema;
- **Validação das correcções**: após a resolução dos defeitos por parte da equipa de desenvolvimento, os testadores repetem os testes afectados para confirmar que os problemas foram efectivamente resolvidos;

- **Testes de regressão**: execução adicional de testes para assegurar que as alterações recentes não introduziram novos erros nem afectaram negativamente funcionalidades já validadas.

2.4.1. Benefícios de Testes Manuais

Apesar dos avanços na automação, os testes manuais continuam a oferecer vantagens significativas, sobretudo em contextos que exigem sensibilidade humana, criatividade e adaptabilidade. Segundo Pathak (2025), uma das principais mais-valias desta abordagem reside na flexibilidade com que pode ser aplicada em diferentes fases do ciclo de desenvolvimento.

Entre os benefícios mais relevantes, destacam-se:

- **Exploração flexível e criativa**: o teste manual permite ao testador explorar livremente diferentes áreas da aplicação, seguindo a sua intuição e experiência para detectar comportamentos inesperados ou erros que escapam aos roteiros formais;
- **Identificação precoce de falhas**: profissionais experientes conseguem, frequentemente, identificar problemas de forma antecipada, mesmo antes da formalização de casos de teste automatizados, contribuindo para a qualidade desde os primeiros ciclos;
- **Avaliação da experiência do utilizador**: como envolve interacção directa com o sistema, o teste manual é particularmente eficaz para avaliar a usabilidade, a fluidez da interface e a facilidade de navegação, aspectos difíceis de medir automaticamente;
- **Adaptação rápida a alterações**: ao contrário dos testes automatizados, que exigem actualização constante do código de teste, os testes manuais podem ser ajustados quase de imediato a mudanças nos requisitos ou em funcionalidades em evolução;
- **Cobertura de cenários complexos ou não padronizados**: situações que envolvem lógica variável, decisões humanas ou fluxos dinâmicos são muitas vezes melhor tratadas através de uma abordagem manual.

2.4.2. Limitações dos Testes Manuais

Esta modalidade também apresenta desafios importantes que limitam a sua eficiência em certos contextos. De acordo com Pathak (2025), os principais entraves associados ao teste manual incluem:

- **Elevado consumo de tempo**: a execução repetida de testes, especialmente em grandes volumes ou em projectos com ciclos curtos, pode tornar o processo moroso e pouco escalável;
- **Susceptibilidade a erros humanos**: a natureza manual do processo expõe-no a lapsos, omissões e inconsistências, o que pode comprometer a fiabilidade dos resultados;
- **Dependência de recursos especializados**: a eficácia do teste depende directamente das competências dos testadores, exigindo investimento em formação, acompanhamento e remuneração adequada;
- **Dificuldade de escalabilidade**: em ambientes com integração contínua ou entregas frequentes, o teste manual mostra-se menos adequado devido à sua limitação para acompanhar volumes elevados e execuções automatizadas.

Em suma, embora apresente limitações operacionais, o teste manual continua a ser uma abordagem indispensável em muitos contextos, especialmente na validação da experiência do utilizador, na detecção exploratória de falhas e em projectos onde a flexibilidade e a análise crítica humana são fundamentais.

2.5. Automação de testes

O crescimento das metodologias ágeis no desenvolvimento de software alterou profundamente a forma como os testes são conduzidos. Com ciclos de entrega cada vez mais curtos e requisitos em constante evolução, tornou-se essencial garantir a qualidade do software de forma rápida, repetível e eficaz. Neste contexto, a automação de testes consolidou-se como uma resposta estratégica às exigências impostas pela agilidade e pela complexidade técnica dos sistemas actuais.

Segundo Pathak (2025), a adopção da automação foi impulsionada pela necessidade de acompanhar a dinâmica acelerada das entregas ágeis, nas quais os testes manuais demonstraram limitações significativas, desde o tempo de execução até à vulnerabilidade a erros humanos. A automação permite executar testes de forma contínua, em ciclos curtos, assegurando não só maior velocidade, como também maior consistência na validação de funcionalidades.

A automação de testes viabiliza certos tipos de verificação que seriam impraticáveis ou extremamente morosos quando realizados manualmente, como os testes de regressão recorrente, testes de carga e testes de desempenho. Estes são particularmente relevantes em ambientes onde o sistema está em constante evolução e exige validações frequentes sem comprometer a produtividade da equipa.

Outro factor determinante na disseminação da automação de testes prende-se com a crescente complexidade dos productos de software modernos. Estes são frequentemente desenvolvidos para múltiplas plataformas e ambientes, recorrendo a diferentes linguagens de programação, frameworks e arquitecturas. Nesse sentido, a automação surge como uma solução robusta, permitindo a reutilização de scripts e a aplicação de testes consistentes em diferentes versões e configurações do sistema (Pathak, 2025).

Para Leloudas (2023), a automação tornou-se cada vez mais valorizada por permitir uma redução significativa do tempo e dos custos associados aos testes, ao mesmo tempo que melhora a precisão dos resultados. Esta visão é corroborada por Ammann e Offutt (2017), que salientam que a automação dos testes não só reduz o esforço manual e os custos envolvidos, como também elimina a subjectividade e facilita a repetição de testes com maior fiabilidade. Como referem, "a automação dos testes não apenas reduz o custo, mas também diminui os erros humanos e facilita os testes de regressão, permitindo que sejam executados repetidamente com o apertar de um botão."

Portanto, a automação de testes transcende o simples uso de ferramentas técnicas, representa uma mudança estrutural na forma como se garante a qualidade do software. Trata-se de uma prática alinhada com a eficiência, a escalabilidade e a robustez exigidas

pelo desenvolvimento de sistemas contemporâneos, onde a agilidade e a qualidade precisam coexistir em equilíbrio.

2.5.1. Definição de Automação

A automação de testes pode ser compreendida como o recurso a ferramentas e tecnologias que permitem executar, gerir e avaliar processos de teste de software de forma automatizada. Segundo Ammann e Offutt (2017), esta prática envolve a utilização de software específico para controlar todas as etapas do teste, desde a definição das pré-condições até à execução, comparação dos resultados obtidos com os esperados e geração de relatórios. Esta visão demonstra que a automação não se restringe à execução automática de scripts, mas abrange todo o ciclo de controlo e análise do teste.

Leloudas (2023) reforça esta abordagem ao caracterizar a automação de testes como o uso de ferramentas especializadas para conduzir a execução dos testes e verificar, de forma sistemática, se os resultados obtidos correspondem aos resultados esperados. A ênfase é colocada na precisão do processo e na eficiência das ferramentas utilizadas, o que permite reduzir significativamente o esforço manual e os erros associados.

Estas definições convergem no reconhecimento de que a automação de testes representa uma evolução nas práticas de garantia da qualidade, proporcionando maior fiabilidade, rastreabilidade e escalabilidade no processo de desenvolvimento. Ao automatizar não apenas a execução, mas também o controlo e a análise dos testes, esta prática torna-se um componente essencial na gestão moderna da qualidade de software, particularmente em ambientes onde a rapidez, a consistência e a repetibilidade são cruciais.

2.5.2. Benefícios da automação

A automação de testes representa uma abordagem estratégica no desenvolvimento de software contemporâneo, permitindo melhorar significativamente a eficiência, a cobertura e a fiabilidade dos testes, especialmente em contextos caracterizados por alta frequência de entregas e complexidade técnica. Segundo Pathak (2025), a utilização de scripts de automação não apenas acelera o processo de verificação, como também

proporciona ganhos relevantes em termos de repetibilidade, precisão e consistência dos resultados.

- **Repetibilidade:** Um dos principais benefícios da automação reside na capacidade de repetição. Esta torna-se particularmente valiosa nos testes de regressão, onde é necessário garantir que novas alterações não afectam funcionalidades previamente validadas. Por exemplo, após a correcção de um defeito identificado, o mesmo script pode ser reutilizado para confirmar a resolução, sem necessidade de intervenção manual.
- **Detecção precoce de defeitos:** Outro aspecto determinante é a detecção precoce de defeitos, viabilizada pela integração dos testes automatizados em pipelines de integração e entrega contínua (CI/CD). Esta prática permite que os testes sejam executados automaticamente após cada alteração no código, sinalizando rapidamente problemas que, se não corrigidos de imediato, poderiam comprometer etapas subsequentes do desenvolvimento.
- **Feedback mais rápido:** A automação também oferece feedbacks mais rápido para os programadores, que passam a dispor de resultados quase imediatos após a submissão do código. Isto contribui para uma resolução mais ágil dos problemas e para uma cultura de melhoria contínua. Em ambientes ágeis, esta capacidade de resposta rápida é fundamental para manter a frequência das entregas.
- **Redução de custos:** Embora a implementação inicial da automação exija investimento em ferramentas, configuração de ambientes e desenvolvimento de scripts, os custos tendem a reduzir-se a médio e longo prazo. Com a reutilização dos testes em várias versões e iterações, o esforço manual é substancialmente minimizado, o que permite redireccionar os recursos humanos para tarefas mais analíticas ou criativas.
- **Cobertura abrangente de testes**: A automação permite simular cenários complexos, demorados ou pouco viáveis de executar manualmente, aumentando a confiança na robustez do sistema. Por exemplo, em aplicações críticas como sistemas de gestão hospitalar, é possível testar sistematicamente funções como

o agendamento de consultas, a inserção de dados clínicos e a emissão de receitas electrónicas.

- **Testes orientados por dados (data-driven testing):** A possibilidade de realizar testes orientados por dados (data-driven testing) é também uma vantagem significativa. Um único script pode ser executado com diferentes conjuntos de dados, permitindo testar uma vasta gama de cenários de forma sistemática. Em plataformas educativas, por exemplo, esta técnica permite validar a criação de contas com múltiplas combinações de dados válidos e inválidos, assegurando a fiabilidade do processo de registo.
- **Testes em múltiplas plataformas e navegadores:** A automação facilita a verificação multiplataforma, garantindo que a aplicação funciona correctamente em diferentes dispositivos, sistemas operativos e navegadores. Isto é especialmente importante no desenvolvimento de aplicações Web responsivas, onde a experiência do utilizador deve manter-se consistente independentemente do contexto de acesso.

Em suma, a automação de testes constitui um pilar essencial da engenharia de software moderna, não apenas pela redução de custos e de esforço manual, mas também pela sua capacidade de garantir qualidade, escalabilidade e confiança num cenário de desenvolvimento contínuo e competitivo.

2.5.3. Desafios e considerações na automação de testes

Embora a automação de testes apresente inúmeros benefícios em termos de eficiência, precisão e escalabilidade, a sua adopção requer uma análise criteriosa de diversos factores. Como observa Leloudas (2023), o custo inicial associado à implementação de ferramentas de automação pode ser elevado, e a manutenção dos scripts de automação tende a exigir esforço contínuo ao longo do ciclo de vida do projecto.

- **Análise de custo-benefício:** Antes de adoptar esta abordagem, torna-se indispensável realizar uma análise de custo-benefício. É essencial que o investimento em infra-estrutura, ferramentas e formação da equipa seja proporcional aos ganhos esperados em termos de redução de esforço manual,

rapidez na entrega e melhoria na qualidade do software. Em projectos de curta duração ou com baixa complexidade, por exemplo, a automação pode não se justificar plenamente.

- **Viabilidade técnica:** Um aspecto determinante é a viabilidade técnica da solução escolhida. A ferramenta de automação seleccionada deve ser compatível com a pilha tecnológica do sistema em desenvolvimento, bem como com as especificidades da aplicação a testar, como o tipo de interface, linguagem utilizada e ambientes de execução. Caso contrário, o esforço necessário para adaptar ou personalizar a ferramenta pode inviabilizar a sua utilização eficiente.
- **Capacitação da equipa**: A capacitação da equipa constitui um pilar fundamental para o sucesso da automação. Os profissionais envolvidos devem possuir competências técnicas adequadas para conceber, executar e manter os testes automatizados, assegurando que estes acompanhem a evolução contínua da aplicação. A ausência de conhecimentos apropriados pode levar à criação de scripts frágeis, difíceis de manter ou ineficazes na detecção de defeitos reais.
- **Definição do escopo da automação:** A definição do escopo da automação é igualmente crítica. Nem todos os testes são candidatos ideais à automação. Deve-se priorizar os testes repetitivos, de regressão, críticos para o negócio ou com elevado custo de execução manual. Já os testes exploratórios, de usabilidade ou que envolvem julgamento humano tendem a manter-se mais eficazes quando realizados manualmente.

A consideração destes elementos não visa desencorajar o uso da automação, mas sim promover uma adopção estratégica e informada, que maximize os seus benefícios e reduza os riscos associados. Quando bem planeada e sustentada, a automação de testes pode tornar-se um activo valioso no processo de desenvolvimento, contribuindo para produtos de software mais fiáveis, robustos e entregues com maior agilidade.

2.6. Ferramentas de Automação

As ferramentas de automação de testes consistem em soluções tecnológicas concebidas para executar, de forma sistemática e autónoma, os casos de teste previamente

definidos, reduzindo a necessidade de intervenção humana directa nos processos de verificação e validação do software. Segundo Santo e Brito (2025), estas ferramentas desempenham um papel central na promoção da eficiência e da qualidade nos ambientes de desenvolvimento modernos, ao permitir a execução repetida e fiável de testes com rapidez e precisão.

Estas ferramentas permitem ainda a sua integração em pipelines de desenvolvimento contínuo, contribuindo para uma verificação constante e actualizada da qualidade do software. Este nível de integração é particularmente relevante em contextos ágeis, nos quais a velocidade de entrega e a fiabilidade do sistema são requisitos fundamentais.

2.6.1. Automação de navegador

A automação de navegador consiste na utilização de ferramentas específicas que permitem executar, de forma programada e autónoma, acções típicas de um utilizador num ambiente web. Estas acções incluem, por exemplo, a navegação entre páginas, o preenchimento e submissão de formulários, a interacção com elementos da interface gráfica e a simulação de comportamentos do utilizador final.

Segundo García, et al. (2024), este tipo de automação é particularmente relevante no contexto dos testes automatizados de aplicações Web, sendo amplamente aplicado em sistemas executados na nuvem, plataformas interactivas e arquitecturas compostas por microsserviços. De acordo com os autores, a automação de navegador é geralmente implementada de forma programática, permitindo não apenas simular interacções complexas com a interface, mas também verificar o comportamento do sistema em diferentes cenários de uso.

Para além da sua aplicação em testes de software, esta técnica pode igualmente ser utilizada em actividades como web scraping ou a automação de tarefas repetitivas, demonstrando a sua versatilidade no ecossistema de aplicações baseadas na Web.

2.6.2. Ferramentas em estudo

Nos últimos anos, tem-se verificado um aumento significativo na quantidade de ferramentas dedicadas à automação de navegadores. Este crescimento acompanha a evolução das tecnologias Web e a crescente procura por soluções que garantam eficiência e precisão nos testes. Embora ainda não exista uma base científica consolidada que descreva em profundidade o panorama actual destas ferramentas, diversos recursos informais, como repositórios colaborativos e publicações técnicas, têm-se revelado úteis na organização e actualização do conhecimento nesta área (García et al., 2024).

Essa diversidade de soluções disponíveis permite aos profissionais seleccionar a ferramenta que melhor se adequa às necessidades específicas dos seus projectos, considerando factores como compatibilidade tecnológica, facilidade de integração e suporte da comunidade. No entanto, face a tantas opções, torna-se necessário definir critérios claros de selecção.

Neste contexto, o presente trabalho opta por centrar-se na análise de três ferramentas largamente adoptadas em ambientes de desenvolvimento modernos: Selenium, Cypress e Playwright. Estas ferramentas distinguem-se pela sua estabilidade, versatilidade e forte presença em projectos que requerem automação de testes em aplicações Web. A escolha recaiu sobre estas por serem reconhecidas pela comunidade técnica e por oferecerem um conjunto robusto de funcionalidades que respondem bem às exigências dos contextos actuais de desenvolvimento ágil e entrega contínua.

2.6.2.1. Selenium

O Selenium é um framework de código aberto amplamente utilizado na automação de testes de aplicações Web. Concebido para simular interacções de utilizadores com navegadores, permite verificar a funcionalidade, a consistência e a compatibilidade de websites em diferentes ambientes e plataformas (BrowserStack, s.d.). Esta ferramenta destaca-se pela sua flexibilidade e pelo forte apoio da comunidade, sendo uma das soluções mais adoptadas no mercado para testes funcionais e de regressão.

Uma das suas principais características é a possibilidade de escrever casos de teste em diversas linguagens de programação, como Java, Python, C#, Ruby, PHP, Perl, Scala e Groovy (GeeksforGeeks, 2025). Esta versatilidade torna o Selenium uma opção acessível para equipas com diferentes perfis técnicos. Ela também oferece uma linguagem de domínio específica (DSL) orientada à criação de scripts de teste, o que facilita a leitura e a manutenção do código de teste.

O Selenium é compatível com os principais sistemas operativos, incluindo Windows, Linux, Solaris e macOS, e suporta também ambientes móveis, como Android, iOS e Windows Mobile. Do ponto de vista da compatibilidade com navegadores, cobre uma ampla gama, incluindo Google Chrome, Mozilla Firefox, Safari e Internet Explorer.

Graças à sua arquitectura modular, o Selenium pode ser integrado com outras ferramentas de teste e plataformas de integração contínua, potenciando a automação em pipelines DevOps. A sua utilização estende-se tanto a testes manuais convertidos em scripts como a cenários mais avançados, onde se requerem testes em paralelo e em múltiplos navegadores.

Selenium Suite

O Selenium não é uma ferramenta única, mas sim um conjunto de componentes que integram a chamada Selenium Suite. Esta suite é composta por quatro elementos principais, cada um com um papel específico na automação de testes de aplicações Web, Selenium IDE, Selenium RC, Selenium WebDriver e Selenium Grid (GeeksforGeeks, 2025).

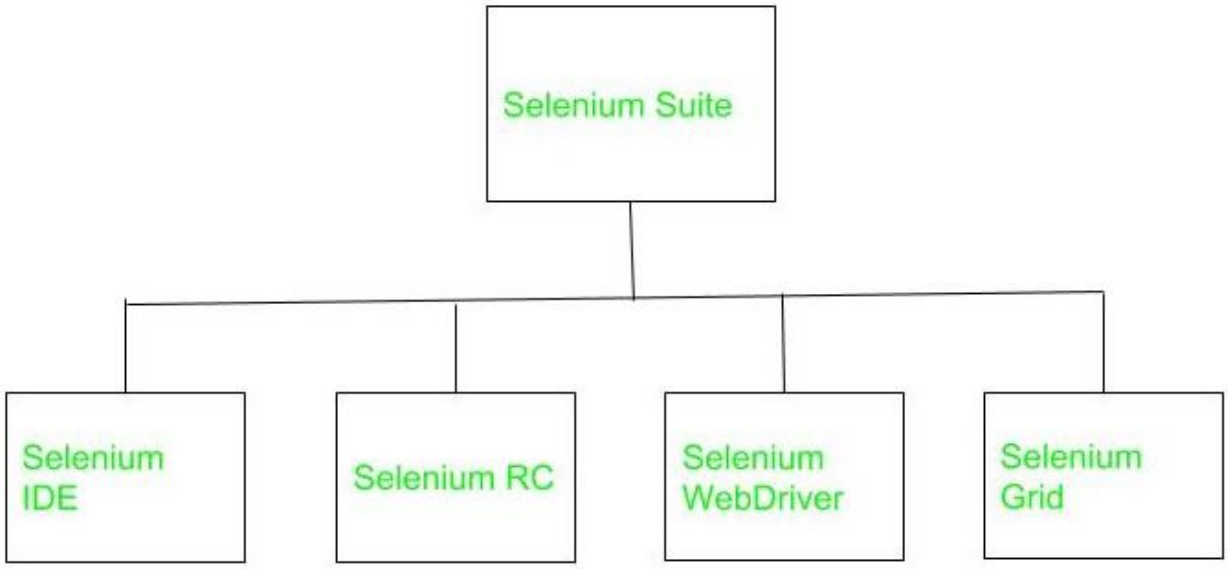

Figura 8 - Ferramentas do Selenium Suite

Fonte: Adaptado de Selenium automation tool – Software engineering, por GeeksforGeeks, 2025

- **Selenium IDE:** O Selenium IDE é uma extensão disponível para os navegadores Google Chrome e Mozilla Firefox. A sua principal funcionalidade consiste na gravação das acções realizadas pelo utilizador, como cliques, selecções e preenchimento de formulários, permitindo posteriormente a sua reprodução automática sob a forma de testes. Os scripts gerados podem ser exportados para diversas linguagens de programação, tais como C#, Java, Python, Ruby ou Selenese, a linguagem nativa do Selenium. É especialmente útil para a criação rápida de testes, validação de funcionalidades e identificação de erros durante a execução interactiva.

- **Selenium RC:** O Selenium Remote Control (RC) foi uma das primeiras soluções desenvolvidas para a automação de testes Web através da simulação de interacções do utilizador em diferentes navegadores e sistemas operativos. Permitia a execução de testes escritos em várias linguagens e a automação remota dos navegadores. No entanto, com o tempo, surgiram limitações significativas relacionadas com a sua arquitectura baseada em proxies, o que impactava o desempenho, a compatibilidade e a manutenção dos testes. Devido

a essas restrições, o Selenium RC foi descontinuado e substituído por uma solução mais robusta: o Selenium WebDriver.

- **Selenium WebDriver:** O Selenium WebDriver foi concebido para ultrapassar as limitações do RC, oferecendo uma abordagem moderna e directa à automação. Ao invés de recorrer a um proxy intermédio, o WebDriver comunica directamente com os navegadores através de métodos nativos, proporcionando maior velocidade, estabilidade e precisão na execução dos testes. Este componente pode ser utilizado em conjunto com o Selenium IDE e o Selenium Grid, o que permite a criação de suites de testes mais escaláveis e eficientes.

- **Selenium Grid:** O Selenium Grid é um servidor inteligente que permite a execução paralela de testes em várias máquinas, navegadores e sistemas operativos. A sua arquitectura baseia-se num hub central que distribui comandos de teste, geralmente em formato JSON, para nós (nodes) registados na rede. Esta abordagem permite uma poupança substancial de tempo e recursos, tornando-se especialmente vantajosa em ambientes de testes contínuos ou com múltiplas configurações de dispositivos.

Selenium WebDriver

O Selenium WebDriver é um dos componentes mais avançados da Selenium suite, concebido para a automação eficiente de testes em aplicações Web. Ao contrário do seu antecessor, o Selenium RC, o WebDriver interage directamente com os elementos do navegador, recorrendo a comandos nativos. Esta abordagem confere-lhe maior velocidade, estabilidade e precisão na execução dos testes (BrowserStack, 2025).

Trata-se de uma ferramenta multiplataforma, compatível com os principais navegadores e sistemas operativos. Também permite o desenvolvimento de testes em várias linguagens de programação, tornando-o acessível a diferentes perfis de desenvolvimento.

Entre as principais aplicações do Selenium WebDriver destacam-se:

- **Testes funcionais**: automatiza acções típicas de um utilizador, como clicar em botões, preencher formulários ou navegar entre páginas, validando os comportamentos esperados da aplicação.
- **Testes entre navegadores**: verifica se a aplicação mantém o seu comportamento e aparência consistentes em diferentes navegadores e versões.
- **Testes multiplataforma**: possibilita a execução de testes em ambientes diversos, assegurando o funcionamento adequado em diferentes sistemas operativos.
- **Testes de regressão:** garante que funcionalidades previamente validadas continuam operacionais após alterações no código.
- **Execução paralela de testes**: através da integração com o Selenium Grid, permite realizar testes simultâneos em diferentes ambientes, reduzindo significativamente o tempo total de verificação.
- **Integração com pipelines CI/CD**: integra-se facilmente com pipelines de integração e entrega contínuas, assegurando a execução automática dos testes sempre que ocorrem alterações no código-fonte.
- **Verificação de interface (UI)**: permite testar visualmente a interface da aplicação, assegurando que o layout e os elementos gráficos se mantêm consistentes.
- **Testes de ponta a ponta (end-to-end)**: simula fluxos completos de utilização, validando a experiência do utilizador do início ao fim da navegação.
- **Testes em cenários complexos**: é capaz de lidar com situações como pop-ups, alertas, conteúdos dinâmicos, iframes e outros elementos não triviais.
- **Testes de desempenho básicos**: embora não seja a sua principal vocação, pode ser utilizado para simular múltiplas interacções de utilizadores e analisar o comportamento do sistema sob carga ligeira.

Estas características fazem do WebDriver uma ferramenta extremamente versátil e poderosa para equipas que pretendem garantir a fiabilidade, compatibilidade e robustez das suas aplicações Web.

Arquitectura

A arquitectura do Selenium 4 WebDriver é composta por um conjunto de componentes que trabalham de forma coordenada para garantir uma comunicação eficaz entre os testes automatizados e os navegadores Web. Esta arquitectura segue o padrão definido pelo protocolo W3C WebDriver, proporcionando maior interoperabilidade, estabilidade e previsibilidade na execução dos testes (BrowserStack, s.d.).

De forma simplificada, o processo inicia-se com o código escrito pelo utilizador numa das linguagens suportadas, que é convertido em requisições HTTP. Estas são enviadas para um driver de navegador, o qual interpreta os comandos recebidos e interage directamente com o navegador real. Por fim, os resultados da execução são devolvidos ao script de teste, permitindo a validação automática do comportamento da aplicação.

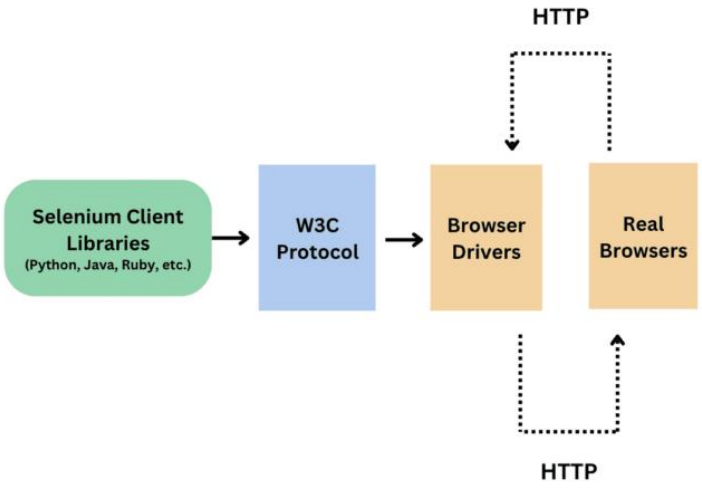

Figura 9 - Arquitrtura do Selenium

Fonte: Adaptado de Selenium testing, por BrowserStack. (s.d.)

Os principais componentes desta arquitectura são:

Bibliotecas Cliente do Selenium (Language Bindings)

O Selenium disponibiliza bibliotecas específicas para várias linguagens de programação, como Java, Python, C#, JavaScript e Ruby. Estas bibliotecas funcionam como interfaces que permitem que o programador escreva os seus testes na linguagem da sua preferência. Por exemplo, para projectos desenvolvidos em Java, é comum utilizar a biblioteca selenium-java. Estas bindings traduzem os comandos do programador para o formato compreendido pelos drivers dos n davegadores.

Protocolo W3C WebDriver

Desde a versão 3.8, o Selenium passou a adoptar oficialmente o protocolo WebDriver do W3C, substituindo o anterior JSON Wire Protocol. Este protocolo padroniza a comunicação entre o cliente (script de teste) e o servidor (navegador), definindo uma API comum que todos os navegadores e drivers devem seguir. A adopção do W3C WebDriver representa um avanço significativo em termos de consistência e compatibilidade entre diferentes ambientes.

Drivers dos Navegadores

Os drivers actuam como uma ponte entre os scripts de teste e os navegadores reais. São executáveis específicos para cada navegador e servem como intermediários na execução dos testes. Alguns exemplos incluem:

- chromedriver.exe para Google Chrome
- geckodriver.exe para Mozilla Firefox
- msedgedriver.exe para Microsoft Edge
- safaridriver para o navegador Safari

Estes drivers interpretam os comandos enviados pelo Selenium e executam-nos directamente no navegador.

Navegadores

São os navegadores reais, como Chrome, Firefox, Edge ou Safari, onde os testes são efectivamente executados. A interacção directa com os navegadores permite reproduzir, de forma fiel, o comportamento do utilizador final, possibilitando uma validação mais realista da interface e das funcionalidades da aplicação Web.

2.6.2.2. Cypress

O Cypress é um framework de testes end-to-end concebido para facilitar a escrita de testes automatizados em aplicações web modernas. Oferece uma solução completa, permitindo verificar tanto as interacções do utilizador com a interface como os processos que ocorrem no backend.

Desenvolvido sobre a linguagem JavaScript, o Cypress destaca-se por ser especialmente acessível a programadores que já trabalham com tecnologias web. A sua API é directa e a sintaxe apresenta-se simples e intuitiva, o que torna a criação de testes mais rápida e fluida, mesmo em cenários complexos.

Ao integrar, numa única ferramenta, funcionalidades para escrita, execução e depuração de testes, o Cypress permite uma abordagem mais interactiva e produtiva ao controlo de qualidade em ambientes de desenvolvimento ágil. (GeeksForGeeks, 2025)

O Cypress tem vindo a afirmar-se como uma das ferramentas de automação de testes mais adoptadas nos ambientes de desenvolvimento actuais. Esta popularidade deve-se, em grande parte, a um conjunto de funcionalidades que oferecem praticidade, desempenho e controlo na validação de aplicações web. Entre os principais factores que justificam a sua ampla aceitação, destacam-se:

- **Execução em múltiplos navegadores**: O Cypress suporta testes em diversos navegadores modernos, o que permite validar o comportamento da aplicação em diferentes ambientes, assegurando a consistência da experiência do utilizador.

- **Configuração simples e rápida**: A instalação do Cypress é directa, sem necessidade de bibliotecas externas ou configurações complexas, o que facilita a sua adopção em projectos novos ou já em curso.
- **Integração com ferramentas de Integração Contínua**: A ferramenta integra-se facilmente com soluções populares de integração contínua, como o Jenkins, bem como com plataformas de testes na nuvem, como o BrowserStack, permitindo a execução automatizada dos testes em pipelines de desenvolvimento.
- **Gestão automática de esperas**: O Cypress inclui um mecanismo nativo que aguarda automaticamente pela presença dos elementos na página antes de realizar interacções ou verificações. Desta forma, elimina-se a necessidade de utilizar comandos como wait ou sleep.
- **Recarregamento em tempo real**: Sempre que há alterações nos ficheiros de teste ou no código da aplicação, o executor de testes é actualizado automaticamente, agilizando o processo de desenvolvimento e depuração.
- **"Viagem no tempo" (Time Travel)**: Durante a execução, é possível visualizar capturas de ecrã correspondentes a cada comando executado, permitindo uma análise cronológica e precisa do comportamento da aplicação.
- **Detecção de testes instáveis (Flake Detection)**: A ferramenta permite identificar testes intermitentes, ou "flaky tests", que apresentam comportamentos inconsistentes, contribuindo assim para a estabilidade do processo de validação.
- **Executor de testes interactivo**: A interface gráfica do Cypress apresenta, em tempo real, o estado da aplicação, os comandos executados e os resultados das verificações, permitindo acompanhar visualmente cada etapa do teste.
- **Capacidades avançadas de depuração**: Com o suporte de ferramentas integradas e compatibilidade com as DevTools do navegador, o Cypress permite inspecionar o estado da aplicação, facilitando o diagnóstico de erros durante a execução dos testes.
- **Controlo sobre requisições de rede (XHR)**: A ferramenta permite interceptar, simular ou modificar respostas de chamadas HTTP, o que possibilita testar cenários específicos sem recorrer a servidores externos ou dados reais.

Arquitectura

O Cypress adopta uma arquitectura baseada no modelo cliente-servidor, em que a componente cliente é executada directamente no navegador. Esta estrutura proporciona um controlo detalhado e eficiente sobre os testes, permitindo uma integração profunda com a aplicação em teste.

Um dos principais elementos desta arquitectura é o Test Runner (Executor de Testes), que se destaca por oferecer uma interface acessível e intuitiva para a escrita, gestão, execução e depuração de testes. Esta ferramenta disponibiliza uma experiência fluida para os programadores, permitindo visualizar em tempo real os resultados dos testes, inspecionar cada etapa individualmente e identificar eventuais problemas com precisão e agilidade.

Com funcionalidades como a execução imediata de testes aquando da alteração do código, bem como relatórios de erro claros e detalhados, o Test Runner contribui significativamente para o aumento da produtividade da equipa de desenvolvimento e para a confiança nos resultados obtidos durante a fase de validação.

Adicionalmente, o Cypress pode ser ampliado através de plugins, que oferecem comandos personalizados e integrações com outras ferramentas ou plataformas. Estes plugins permitem adaptar o ambiente de testes às necessidades específicas de cada projecto, acrescentando flexibilidade e escalabilidade ao processo de automatização. Deste modo, o Cypress pode ser facilmente ajustado a diferentes contextos de desenvolvimento, proporcionando uma solução robusta e personalizável para a garantia da qualidade de aplicações web. (GeekforGeek, 2025)

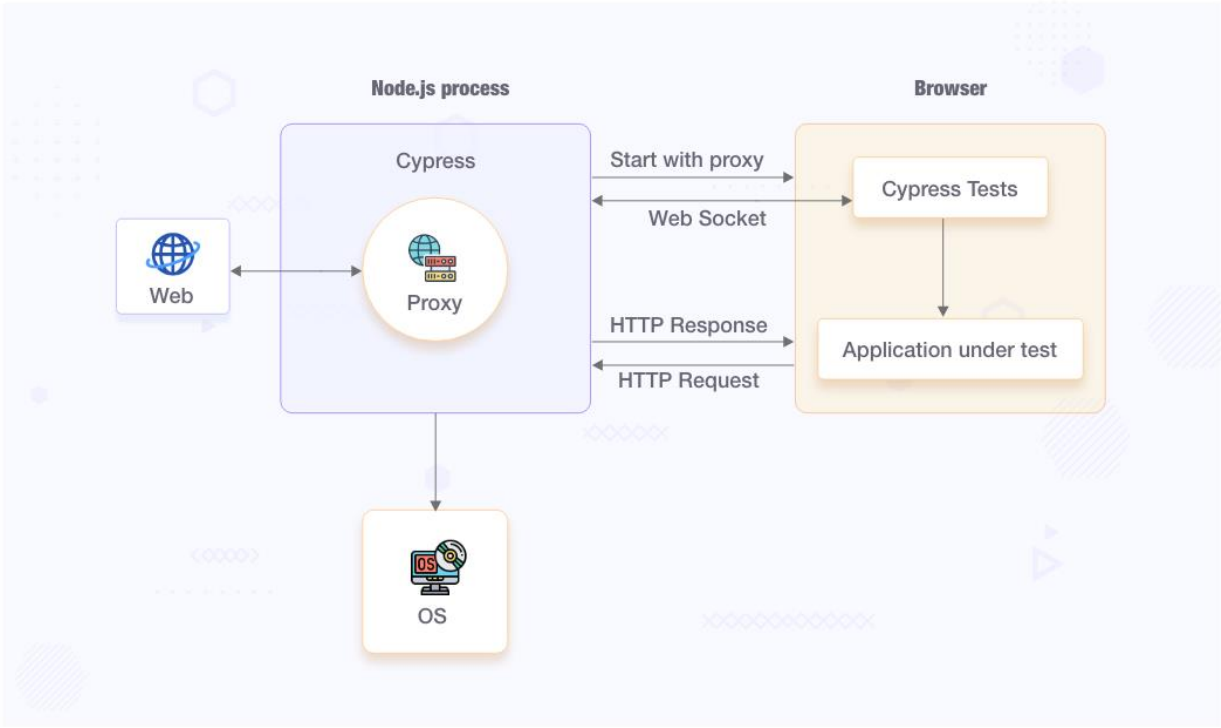

Figura 10 - Funcionamento da Execução de um Testeno Cypress

Fonte: Adaptado de Cypress automation tutorial, por BrowserStack. (s.d.)

Componentes do Cypress

O Cypress é constituído por diversos componentes que, em conjunto, viabilizam um processo de automação de testes robusto e adaptável às necessidades de diferentes projectos:

- **Test Runner**: Constitui a interface interactiva onde se escrevem, organizam, executam e depuram os ensaios automatizados. Permite observar, em tempo real, a execução dos testes e analisar os seus resultados com elevado grau de detalhe.
- **Cypress Dashboard**: Serviço baseado na nuvem que oferece funcionalidades adicionais, como o registo detalhado dos ensaios realizados, a possibilidade de paralelização da execução e a análise de métricas relevantes para o acompanhamento da qualidade do software.
- **Plugins**: Componentes complementares que estendem as funcionalidades do Cypress, permitindo a adição de comandos personalizados e integrações com

outras ferramentas ou plataformas, o que facilita a adaptação da solução a contextos específicos de desenvolvimento e testes

Funcionamento da execução de testes no Cypress

O processo de execução de testes no Cypress segue uma sequência estruturada de interacções entre os seus componentes, permitindo uma verificação precisa do comportamento da aplicação:

- Quando o utilizador inicia a interacção com a aplicação em teste, os scripts de ensaio elaborados no Cypress enviam comandos para o Cypress Runner.
- O Runner comunica com um servidor proxy, que, por sua vez, reencaminha os pedidos para o servidor da aplicação.
- O servidor da aplicação processa os pedidos recebidos e responde conforme o esperado.
- Durante a execução, o Cypress regista capturas de ecrã e vídeos automáticos, documentando visualmente o comportamento da aplicação ao longo do ensaio.
- Por fim, os profissionais responsáveis pela validação analisam os resultados dos ensaios, verificando se o sistema responde de acordo com os requisitos e se o comportamento observado corresponde ao funcionamento correcto da aplicação.

2.6.2.3. Playwright

O Playwright é uma plataforma de automação de código aberto, desenvolvida pela Microsoft, que permite efectuar testes end-to-end em aplicações web. Lançado em 2020, este framework rapidamente se destacou no mercado por oferecer uma abordagem moderna e eficiente para a automação de navegadores, suportando o Chromium (Google Chrome), o Firefox e o WebKit (Safari). Uma das principais vantagens do Playwright reside na sua capacidade de ultrapassar várias limitações de ferramentas mais tradicionais, como o Selenium, proporcionando testes mais rápidos, fiáveis e fáceis de gerir (Pathak, 2025).

A ferramenta também suporta múltiplas linguagens de programação, nomeadamente Java, JavaScript, TypeScript, Python e C#, o que facilita a sua adopção por equipas com diferentes perfis técnicos. O Playwright permite executar tarefas como navegação entre páginas, interacção com elementos da interface, manipulação de dados e captura de imagens de ecrã, tanto em modo visível (headed) como em modo headless (sem interface gráfica), adaptando-se às diversas necessidades dos ambientes de desenvolvimento e de ensaio (Pathak, 2025).

Arquitectura

A arquitectura do Playwright assenta num modelo cliente-servidor, sustentado por uma comunicação baseada em WebSocket, o que garante uma transmissão contínua de dados, mais rápida e eficiente do que o tradicional protocolo HTTP (TestingMavens, s.d.).

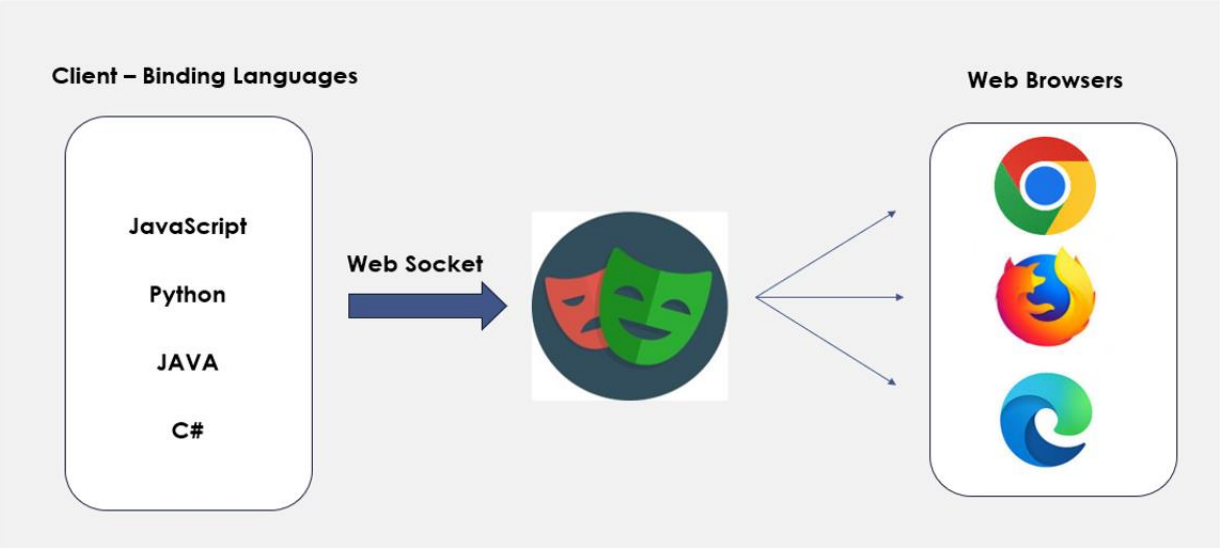

Figura 11 - Arquitetura do Playwright

Fonte: adaptado de *Architecture Breakdown: Selenium, Cypress and more por TestingMavens*

Os principais componentes da sua arquitectura são:

- **Cliente (Language Binding)**: Representa a camada onde os scripts de teste são escritos. O Playwright oferece bindings para várias linguagens, como JavaScript,

TypeScript, Python, Java e C#, que comunicam directamente com os navegadores através do servidor Playwright.

- **WebSocket Connection**: Em vez de enviar requisições individuais como no modelo HTTP, o Playwright estabelece uma ligação persistente através de WebSocket, permitindo o envio contínuo de comandos e respostas durante toda a execução do teste. Este modelo reduz drasticamente o tempo de comunicação e melhora a performance dos testes.
- **Browser Contexts**: Um dos diferenciais do Playwright é o conceito de contextos de navegador. Trata-se de instâncias isoladas dentro de um mesmo navegador, cada uma com o seu próprio armazenamento, cookies e cache. Isto permite a execução paralela de testes sem interferência entre sessões, o que é extremamente vantajoso para simular múltiplos utilizadores ou ambientes.
- **Navegadores**: São os browsers reais (Chromium, Firefox e WebKit) onde os testes são executados. O Playwright interage directamente com os motores destes navegadores, garantindo elevada fiabilidade e precisão na automação.

Funcionamento

O funcionamento do Playwright pode ser descrito em quatro etapas principais:

- **Escrita dos testes**: O programador desenvolve os scripts utilizando uma das linguagens suportadas, definindo as acções e verificações a serem realizadas.
- **Estabelecimento da conexão**: Assim que os testes são iniciados, o cliente estabelece uma ligação via WebSocket com o servidor Playwright, mantendo a conexão activa durante todo o processo.
- **Execução nos contextos do navegador**: Cada cenário de teste é executado dentro de um contexto isolado, garantindo que os dados (como sessões, cookies e armazenamento local) não se misturam entre testes.
- **Colecta de resultados e encerramento**: Durante a execução, são capturadas imagens, vídeos e logs, que auxiliam na análise dos resultados. Após a conclusão, a ligação é encerrada e os contextos são destruídos, assegurando um ambiente limpo para futuras execuções.

Funcionalidades do Playwright

O Playwright apresenta um conjunto robusto de funcionalidades que o tornam uma das ferramentas mais completas e versáteis da actualidade:

- **Testes multiplataforma e multi-navegador**: Suporte nativo aos navegadores Chromium, Firefox e WebKit, em sistemas operativos como Windows, Linux e macOS.
- **Modo Headless e Headed**: Permite executar testes sem interface gráfica (ideal para pipelines CI/CD) ou com interface visível (útil para depuração).
- **Isolamento com Browser Contexts**: Criação de múltiplos contextos dentro de uma única instância do navegador, permitindo testes paralelos e independentes.
- **Execução paralela**: Acelera significativamente o tempo de execução, distribuindo os testes entre diferentes processos ou máquinas.
- **Esperas automáticas e gestão de eventos**: A arquitectura orientada a eventos assegura que as acções só são executadas quando os elementos estão prontos, eliminando a necessidade de comandos manuais como wait ou sleep.
- **Intercepção de rede**: Permite interceptar, modificar ou simular respostas de APIs, tornando possível testar cenários offline, falhas de rede ou dados fictícios.
- **Emulação de dispositivos**: Oferece a capacidade de simular diferentes dispositivos, tamanhos de ecrã e condições de rede, facilitando testes responsivos.
- **Gravação de vídeo e screenshots**: Captura automática de imagens e vídeos durante os testes, facilitando a análise de erros e documentação.
- **Integração com frameworks populares**: Integra-se facilmente com Jest, Mocha, pytest, e outros frameworks de teste, além de ferramentas de integração contínua como Jenkins, GitHub Actions e GitLab CI.
- **Comunidade activa e manutenção contínua**: Sendo um projecto desenvolvido pela Microsoft e com ampla adopção na comunidade, o Playwright recebe actualizações regulares, novos recursos e suporte constante.

2.6.3. Comparação das ferramentas apresentadas

Para efectuar uma análise comparativa entre as ferramentas de automação de testes Selenium, Cypress e Playwright, foram considerados critérios técnicos que possibilitam avaliar as suas funcionalidades, vantagens e limitações em diferentes contextos de utilização. Esta análise é essencial para fundamentar a escolha da ferramenta mais adequada, tendo em conta os objectivos específicos de cada projecto.

Segundo Pathak (2025), os critérios adoptados para esta comparação incluem:

- **Facilidade de uso**: Mede o quão simples é começar a utilizar a ferramenta, analisando a clareza da API, a legibilidade dos comandos e a necessidade ou não de configurações adicionais.
- **Linguagens suportadas**: Avalia a variedade de linguagens de programação compatíveis, o que oferece maior flexibilidade para equipas multidisciplinares.
- **Instalação e configuração**: Verifica se o processo de instalação é simples e directo, ou se exige configurações mais complexas, como instalação manual de drivers.
- **Velocidade de execução**: Analisa o tempo necessário para executar os testes, um aspecto crítico em pipelines de integração contínua (CI/CD).
- **Suporte a múltiplos navegadores**: Avalia a capacidade da ferramenta para executar testes em diferentes navegadores e motores (Chrome, Firefox, Safari, Edge, entre outros).
- **Execução paralela**: Verifica se é possível executar vários testes simultaneamente, optimizando o tempo total de execução.
- **Modo headless**: Refere-se à possibilidade de executar testes sem interface gráfica, uma funcionalidade crucial para ambientes de automação e servidores.
- **Funcionalidades avançadas**: Considera recursos como simulação de dispositivos, interceptação de rede, captura de vídeo, screenshots, manipulação de cookies e dados de sessão.

- **Curva de aprendizagem**: Mede o grau de dificuldade para dominar a ferramenta, desde a sua configuração inicial até ao desenvolvimento de testes mais complexos.
- **Depuração (debugging)**: Avalia os mecanismos que auxiliam na identificação e resolução de falhas, como logs detalhados, visualização interactiva dos testes e ferramentas de inspecção.
- **Estabilidade e fiabilidade**: Verifica a consistência dos testes, especialmente no que diz respeito à gestão de tempos, sincronização e redução de falhas intermitentes (testes flaky).

Critério	Selenium	Playwright	Cypress	Vencedor
Facilidade de uso	API mais extensa, curva mais acentuada	API moderna, clara e intuitiva	API muito simples e intuitiva	**P, C**
Linguagens suportadas	Java, Python, C#, Ruby, PHP, Perl, JS, Scala, Groovy	Java, JS/TS, Python, .NET	JS/TS	**S**
Instalação e configuração	Processo mais complexo, exige drivers externos	Setup simples, com navegadores embutidos	Instalação extremamente fácil, com interface gráfica	**C**
Velocidade de execução	Pode ser mais lenta, dependendo do driver e contexto	Muito rápida, arquitectura optimizada	Boa, mas inferior ao Playwright em grandes cargas	**P**
Suporte a múltiplos browsers	Chrome, Firefox, Edge, Safari (via WebDriver)	Suporte nativo: Chromium, Firefox, WebKit	Chrome, Edge, Firefox, WebKit	**P**

Execução paralela	Possível, mas com configuração manual	Suporte nativo	Exige configuração adicional e plugins	**P**
Modo Headless	Suporte, mas requer configuração adicional	Suporte nativo e consistente	Simples e funcional	**P**
Recursos avançados	Depende de ferramentas externas e plugins	Embutidas: interceptação de rede, simulação, vídeos, etc.	Boas funcionalidades, principalmente na interface	**P**
Depuração (debugging)	Limitado, depende de ferramentas externas	Debugging robusto, logs claros, screenshots, vídeos	Test Runner interactivo com logs detalhados	**C**
Curva de aprendizagem	Mais íngreme, exige mais tempo	Rápida, graças à API moderna e documentação acessível	Muito rápida, com foco na simplicidade	**P, C**

Estabilidade e confiabilidade	Pode enfrentar problemas de sincronização e timing	Muito estável: esperas automáticas, gestão de eventos	Alta integração com a aplicação, boa fiabilidade	**P**

Tabela 1 - Comparação de Selenium, Playwright e Cypress

Fonte: Tabela de autoria do autor

A partir da análise dos critérios técnicos apresentados, fundamentada na literactura de Pathak (2025), observa-se que todas as ferramentas possuem pontos fortes e limitações que as tornam adequadas a diferentes contextos.

O Selenium continua a ser uma solução sólida, especialmente quando há necessidade de suportar múltiplas linguagens de programação e quando se trabalha com projectos legados ou ambientes altamente heterogéneos.

O Cypress oferece uma abordagem muito amigável e prática, com uma curva de aprendizagem suave, ideal para equipas que trabalham no ecossistema JavaScript/TypeScript e que procuram rapidez na adopção e simplicidade na depuração.

Por outro lado, o Playwright destaca-se como uma solução moderna, robusta e altamente eficiente. Apresenta uma API intuitiva, suporta execução paralela nativa, permite testes em múltiplos navegadores (incluindo WebKit) e oferece, de forma integrada, funcionalidades avançadas que nas outras ferramentas exigem dependências externas.

Adicionalmente, a sua arquitectura baseada em WebSocket, com gestão eficiente de múltiplos contextos de navegador, garante uma execução mais rápida, estável e fiável, reduzindo significativamente problemas típicos como flakiness, sincronização e dependência de waits manuais.

Face a estas considerações, a escolha do Playwright para este projecto justifica-se pela sua superioridade em termos de velocidade, estabilidade, funcionalidades integradas, facilidade de manutenção e adaptabilidade a ambientes modernos de desenvolvimento e integração contínua. Trata-se, portanto, da ferramenta que melhor responde às exigências actuais de automação de testes web, tanto no plano técnico como operacional.

3. Capitulo III: Caso de Estudo

3.1. Techsolutions

A Techsolutions, Lda é uma empresa moçambicana que actua no sector das tecnologias de informação, oferecendo uma ampla gama de serviços que incluem o desenvolvimento de websites, aplicações móveis, sistemas empresariais, design gráfico, gestão de redes sociais e outras soluções digitais adaptadas às necessidades do mercado.

Fundada por uma equipa jovem, dinâmica e altamente qualificada, a empresa tem como missão fornecer soluções tecnológicas inovadoras, alinhadas às exigências dos seus clientes, tanto no contexto local como no global. O seu principal compromisso reside em entregar serviços de elevada qualidade, que contribuam efectivamente para a transformação digital das organizações, independentemente da sua dimensão ou área de actuação (Techsolutions, 2025).

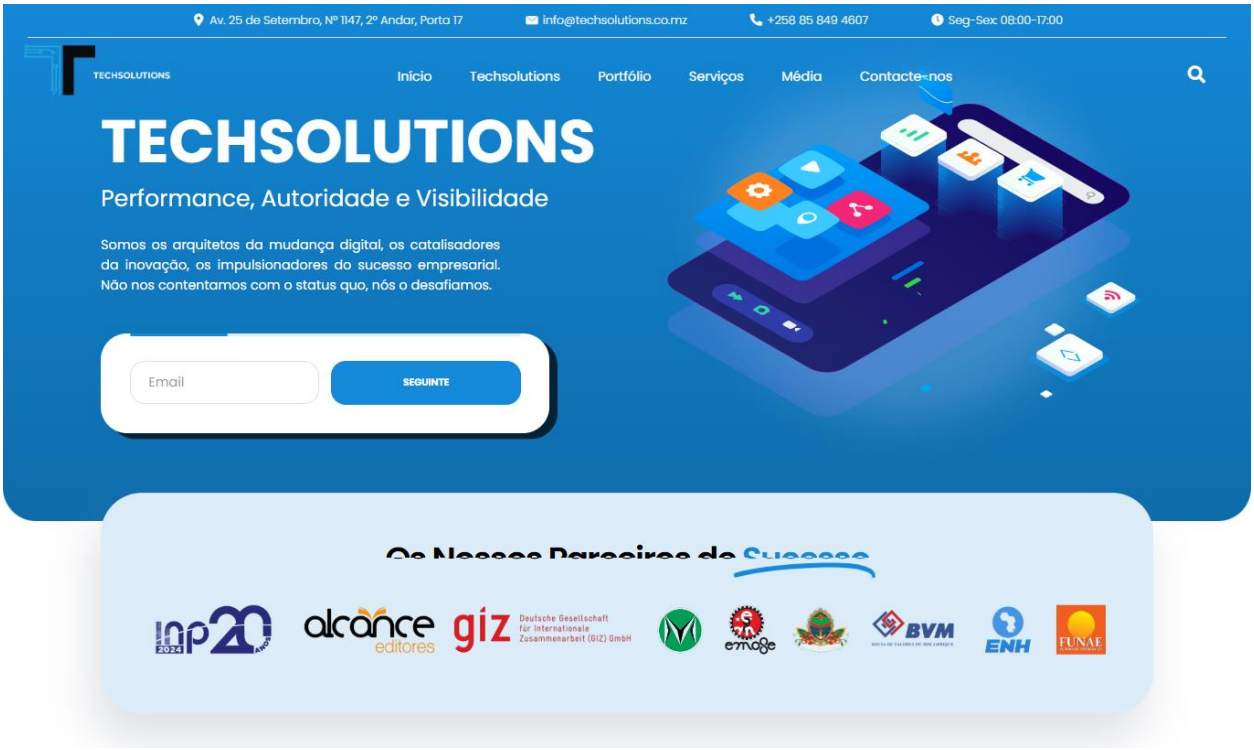

Figura 12 - Website da Techsolutions, LDA

Fonte: Imagem de autoria do autor

3.1.1. Breve Histórial

A Techsolutions, Lda é uma empresa que tem vindo a construir uma trajectória de crescimento sólido e contínuo no sector das tecnologias de informação em Moçambique, destacando-se pela sua capacidade de adaptação e inovação.

Fundada em Fevereiro de 2020 e oficialmente registada no dia 16 de Novembro do mesmo ano, a empresa iniciou as suas actividades com foco na prestação de serviços digitais, tendo rapidamente conquistado espaço no mercado nacional.

O ano de 2021 marcou um ponto de viragem na história da Techsolutions, com a conquista do seu primeiro concurso público de grande dimensão. Este projecto consistiu no desenvolvimento do website da Alcance Editores, representando não apenas um marco significativo para a empresa, mas também o início da sua actuação em projectos tecnológicos de maior complexidade e exigência.

Em 2022, a empresa consolidou a sua presença no mercado, expandindo a sua actuação para clientes de grande porte. Esta fase foi crucial para fortalecer a sua reputação no sector tecnológico, através da entrega de soluções robustas e alinhadas às necessidades dos clientes.

O ano de 2023 revelou-se particularmente estratégico, marcado pela diversificação dos serviços prestados e pelo estabelecimento de parcerias com instituições públicas e entidades governamentais. Destaca-se, neste período, a adjudicação, no dia 27 de Dezembro, de um concurso para o fornecimento de equipamentos informáticos, bem como a execução de um projecto de grande escala na Clínica Marcelino dos Santos. Este projecto envolveu a instalação de um sistema avançado de videovigilância (CCTV), composto por mais de 50 câmaras, e a implementação de um sistema de controlo de acessos com recurso a tecnologias biométricas, como impressão digital e reconhecimento facial.

Em 2024, a Techsolutions consolidou definitivamente a sua posição no mercado moçambicano, através do desenvolvimento de mais de 15 websites para organizações de grande dimensão. Neste ano, a empresa registou um crescimento superior a 100%, acumulando uma carteira com mais de 30 clientes dos mais diversos sectores, posicionando-se como uma referência nacional em soluções tecnológicas inovadoras e sustentáveis.

3.1.2. Missão, Visão e Valores

A organização pauta a sua actuação por princípios que reflectem o seu compromisso com a excelência, a inovação e a responsabilidade social. Estes princípios orientam não só o desenvolvimento de soluções tecnológicas, como também a forma como se relaciona com clientes, parceiros e colaboradores.

Missão

Impulsionar o desenvolvimento tecnológico a nível nacional e internacional, através da criação e implementação de soluções práticas e inovadoras que contribuam positivamente para a sociedade.

Visão

Ser reconhecida como uma empresa de excelência na prestação de serviços de Tecnologias de Informação, destacando-se pela criação de soluções úteis e pelo desenvolvimento de produtos de impacto global.

Valores

A cultura organizacional da Techsolutions assenta nos seguintes valores fundamentais:

- Inovação – Compromisso constante com a criação de soluções criativas e eficazes, que acompanhem a evolução tecnológica.
- Paixão – Dedicação plena a cada projecto, com entusiasmo e motivação no exercício das actividades profissionais.

- Integridade – Actuação ética e transparente em todas as áreas da empresa, promovendo a confiança mútua com parceiros e clientes.
- Desenvolvimento – Valorização do crescimento contínuo, tanto a nível técnico como humano, promovendo o progresso individual e colectivo.
- Inclusão – Promoção de um ambiente acessível, diverso e acolhedor, onde todas as pessoas têm oportunidade de participar e contribuir.

3.1.3. Serviços Prestados Pela Empresa

A empresa posiciona-se como uma referência no sector das Tecnologias de Informação, destacando-se pela oferta de soluções tecnológicas inovadoras, adaptadas tanto à realidade moçambicana como ao contexto global. Com uma abordagem centrada na personalização, na eficiência e na qualidade, presta serviços que abrangem desde o desenvolvimento de soluções digitais até à implementação de infra-estruturas e sistemas de segurança electrónica.

O portfólio de serviços é diversificado e visa apoiar organizações públicas, privadas e particulares, proporcionando ferramentas tecnológicas que impulsionam a sua transformação digital e melhoram os seus processos operacionais.

Entre os principais serviços prestados destacam-se:

- **Criação de Websites**: Desenvolvimento de websites profissionais, responsivos e optimizados para todos os dispositivos, com foco na experiência do utilizador e na identidade visual dos clientes.
- **Assistência Informática e Gestão de Redes**: Fornecimento, instalação e manutenção de equipamentos informáticos, redes de dados e acesso à internet, bem como suporte técnico especializado.
- **Gestão de Redes Sociais e Produção Audiovisual**: Gestão de presença digital, produção de conteúdos, fotografia, vídeo e materiais gráficos para redes sociais e campanhas digitais.

- **Desenvolvimento de Softwares Personalizados**: Criação de soluções de software à medida, alinhadas às necessidades específicas de cada cliente e projecto, com foco na eficiência e na escalabilidade.
- **Sistemas de Segurança Electrónica (CCTV)**: Implementação de sistemas de videovigilância de alta qualidade, com integração de tecnologias avançadas, como biometria (impressão digital e reconhecimento facial) e controlo de acessos.
- **Fornecimento de Equipamento Informático**: Comercialização de equipamentos e acessórios informáticos com garantia, incluindo serviços de acompanhamento e suporte para assegurar a melhor utilização dos recursos.

3.1.3.1. Criação de Websites

A criação de websites é o serviço que sustenta a base do crescimento e da reputação da empresa no mercado. Desde a sua fundação, este serviço tem sido central na estratégia da organização, que se destaca pela entrega de soluções digitais com elevados padrões de qualidade e inovação.

Cada projeto é executado com atenção ao detalhe, desde a fase de levantamento de requisitos até à entrega final, assegurando que o produto final reflita a identidade da instituição ou empresa cliente. Esta dedicação à excelência tem permitido à Techsolutions firmar parcerias com organizações de referência em Moçambique, que confiaram à empresa a responsabilidade de representá-las digitalmente.

A confiança de instituições de grande relevância demonstra o reconhecimento da qualidade dos serviços prestados. Entre os principais clientes encontram-se:

- EMOSE, SA;
- Conselho Municipal de Maputo;
- Eletrotec;
- Instituto Nacional de Petróleo (INP);
- FUNAE, FP;
- Entre outras instituições públicas e privadas de destaque.

3.1.3.2. Desenvolvimento de Softwares Personalizados

O desenvolvimento de softwares personalizados constitui uma das principais apostas estratégicas da empresa, reflectindo o seu compromisso com a inovação e a transformação digital. Este serviço vai muito além da simples implementação de sistemas, representa um compromisso com a criação de soluções tecnológicas totalmente alinhadas às necessidades de cada cliente e projecto, respeitando as suas especificidades, o seu contexto operacional, os seus objectivos e os seus desafios de crescimento.

Num cenário onde a transformação digital é uma prioridade tanto para instituições públicas como privadas, a empresa tem se destacado pelo desenvolvimento de plataformas robustas, dinâmicas e escaláveis, concebidas para optimizar processos, automatizar tarefas e reforçar a eficiência operacional dos seus clientes.

Este investimento tem-se refletido em projectos de grande relevância a nível nacional, entre os quais se destacam:

- **Portal de Licenciamento de Transporte e Logística:** Desenvolvido para o Ministério dos Transportes e Comunicações, este portal permite a digitalização integral dos processos de licenciamento de operadores de transporte em Moçambique, promovendo maior agilidade, transparência e eficiência.

- **Portal de Recrutamento da Eletrotec**: Plataforma concebida para gerir e simplificar o processo de recrutamento da empresa, facilitando a captação, triagem e selecção de candidatos de forma automatizada e eficaz.

- **Plataforma Headhunting da SAL Group**: Uma solução inovadora orientada para a gestão de recrutamento especializado, actualmente em fase de testes, que permitirá automatizar e optimizar os processos de identificação e selecção de talentos altamente qualificados.

Através destes e de outros projectos em curso, a organização tem consolidado a sua reputação como um parceiro tecnológico de referência, contribuindo activamente para a transformação digital no contexto nacional, através de soluções de software de elevado valor, desenhadas à medida das necessidades dos seus clientes.

3.2. Sistema de Recrutamento Headhunting

A plataforma de recrutamento Headhunting é uma solução digital desenvolvida pela Techsolutions para a empresa SAL Group, com o objectivo de transformar e modernizar os processos de recrutamento em Moçambique. Este projecto surge da aposta na criação de ferramentas tecnológicas personalizadas, orientadas para resolver desafios concretos enfrentados pelas organizações nacionais no contexto da gestão de recursos humanos.

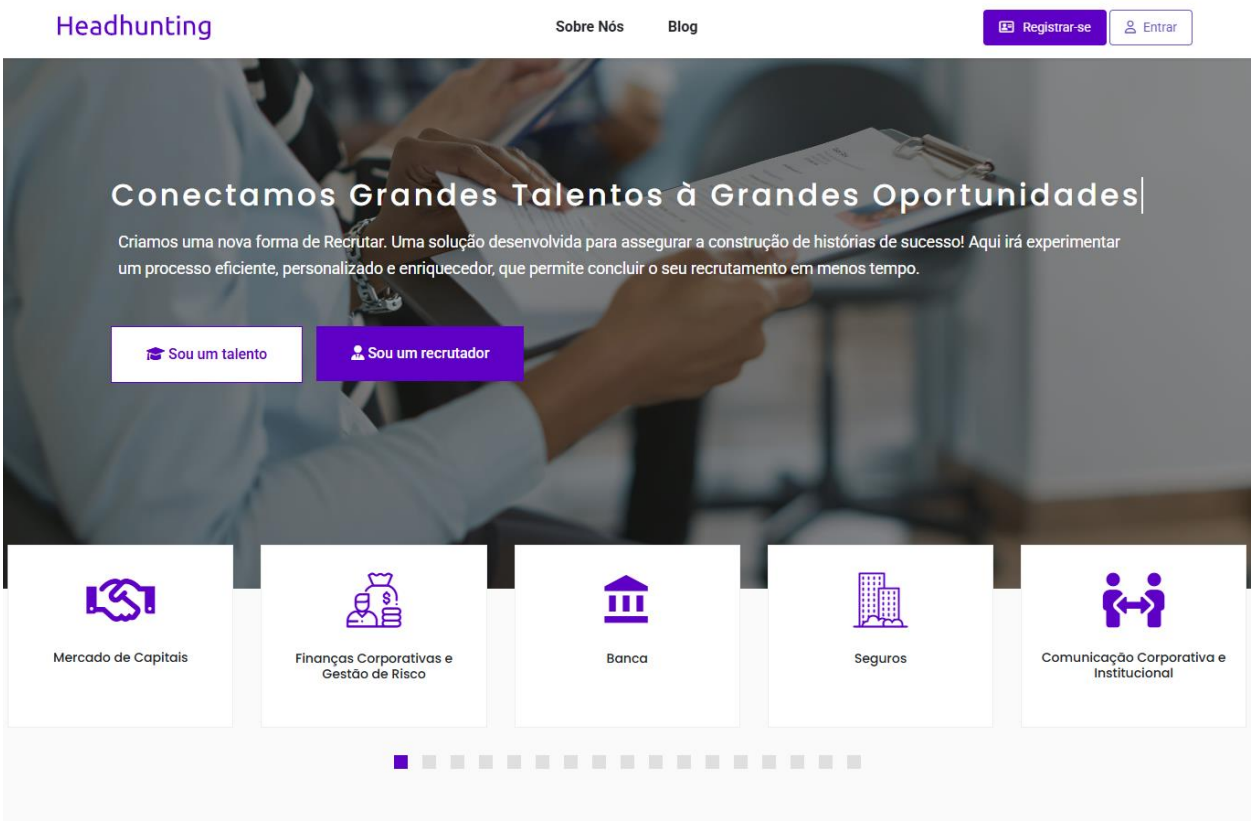

Figura 13 - Plataforma de Recrutamento Headhunting

Fonte: Imagem de autoria do autor

Trata-se de uma plataforma inteligente, concebida para automatizar e optimizar todas as etapas do recrutamento e selecção de candidatos. Ao contrário dos métodos tradicionais, geralmente caracterizados por processos longos, dispendiosos e sujeitos a elevada rotatividade, esta solução visa não só reduzir o tempo e os custos associados ao recrutamento, como também aumentar significativamente a eficácia na correspondência entre vagas e perfis profissionais.

A Headhunting incorpora tecnologias avançadas, incluindo algoritmos de inteligência artificial, que permitem analisar e cruzar dados de forma precisa, assegurando uma selecção mais eficiente e alinhada não apenas com as competências técnicas, mas também com o grau de afinidade e potencial de engajamento dos candidatos em relação às empresas contratantes.

Portanto, a plataforma Headhunting posiciona-se como uma resposta tecnológica robusta aos desafios do mercado de trabalho em Moçambique, contribuindo para a digitalização e modernização dos processos de recrutamento no país.

3.3. Descrição da Situação Actual

Actualmente, a plataforma Headhunting encontra-se na fase final do seu desenvolvimento. Embora já esteja funcional e operacional, o sistema está submetido a um ciclo rigoroso de testes internos, com o propósito de assegurar que todas as funcionalidades correspondem às especificações definidas e que a experiência do utilizador seja estável, intuitiva e segura.

Esta etapa é crucial para validar o correcto funcionamento da pltaforma, identificando eventuais falhas, inconsistências ou melhorias necessárias antes da sua disponibilização oficial no mercado. O processo de validação é realizado de forma colaborativa, envolvendo duas equipas distintas, por um lado, a equipa técnica da Techsolutions, responsável pela concepção e desenvolvimento da plataforma, por outro, a equipa da SAL Group, que actua enquanto cliente e futuro utilizador directo do sistema.

Esta colaboração entre as duas partes permite combinar diferentes perspectivas, a perspectiva técnica, centrada na robustez, desempenho e segurança da aplicação, e a

perspectiva operacional, focada na usabilidade, adequação dos fluxos de trabalho e alinhamento com os processos de negócio da empresa contratante.

Ao longo desta fase, o sistema está sujeito a testes contínuos, com o objectivo de garantir que todos os requisitos funcionais e não funcionais sejam cumpridos, assegurando, assim, uma entrega final de elevada qualidade e fiabilidade.

3.3.1. Processo de Teste

O processo de teste da plataforma Headhunting é actualmente realizado de forma manual, envolvendo a simulação de interacções reais no sistema, tal como seria feito por um utilizador final. Este procedimento é conduzido tanto pela equipa técnica responsável pelo desenvolvimento da plataforma como pela equipa da SAL Group, cliente que encomendou o sistema.

A abordagem consiste na execução de cenários de uso previamente definidos, percorrendo as funcionalidades da aplicação com o objectivo de verificar se estas se comportam de acordo com os requisitos estabelecidos. Durante os testes, são simuladas acções comuns, como registos de utilizadores, candidaturas a vagas, submissão de formulários, pesquisa de perfis e validação de respostas do sistema.

Este método de validação visa assegurar que todas as funcionalidades estão operacionais, que não existem erros na navegação, nos fluxos de trabalho ou no processamento dos dados, e que a experiência proporcionada aos utilizadores é coerente e satisfatória.

O processo é conduzido directamente na interface do utilizador, explorando os diferentes componentes e fluxos. Cada funcionalidade é validada isoladamente, assim como em combinação com outras partes do sistema, de modo a garantir a integridade dos processos e das interacções entre módulos.

3.3.2. Constrangimentos do processo de testes manuais

Apesar de ser uma etapa indispensável no desenvolvimento da plataforma, a realização dos testes de forma exclusivamente manual apresenta diversos constrangimentos que

comprometem a sua eficácia e eficiência. Entre os principais desafios identificados, destacam-se:

- **Morosidade no processo**: A validação de cada funcionalidade exige múltiplas execuções de testes, o que consome tempo considerável, especialmente à medida que o sistema cresce.
- **Repetição exaustiva de tarefas**: Testes recorrentes sobre os mesmos fluxos, como o preenchimento de formulários ou submissão de candidaturas, tornam-se cansativos e susceptíveis a erros por fadiga.
- **Propensão a erros humanos**: A monotonia e o desgaste inerentes ao processo manual aumentam a probabilidade de falhas, como esquecimentos, cliques incorretos ou omissão de verificações cruciais.
- **Cobertura limitada de cenários**: Nem sempre é possível testar, de forma manual, todos os casos possíveis, incluindo situações menos prováveis, mas críticas, que podem impactar negativamente os utilizadores.
- **Inconsistência nos resultados**: Diferentes testadores podem executar os testes de formas diversas, seguindo sequências distintas ou aplicando interpretações subjectivas, o que compromete a padronização e dificulta a comparação entre execuções.
- **Aumento dos custos e atrasos nos prazos**: A dependência de recursos humanos para a execução dos testes implica custos mais elevados e pode atrasar entregas, sobretudo em sistemas que passam por actualizações constantes.
- **Dificuldade no rastreio e na documentação dos testes**: A ausência de ferramentas automatizadas torna mais difícil o registo sistemático dos testes realizados, comprometendo a rastreabilidade, a replicação e a análise posterior dos resultados.
- **Baixa produtividade das equipas**: O tempo consumido por tarefas repetitivas reduz significativamente o foco em actividades de maior valor, como testes exploratórios, análises de qualidade e melhoria contínua dos processos

4. Capitulo IV - Desenvolvimento da solução Proposta

4.1. Descrição da solução

Face aos desafios identificados no processo actual de testes manuais da plataforma Headhunting, propõe-se a adopção de uma solução baseada na automação de testes, recorrendo à ferramenta Playwright. Esta abordagem surge como uma resposta estratégica, orientada para optimizar o processo de validação do sistema, reduzir custos operacionais e assegurar maior fiabilidade, consistência e escalabilidade nos testes.

O principal objectivo desta solução consiste em substituir gradualmente os testes manuais por testes automatizados de ponta a ponta, permitindo validar, de forma mais rápida e precisa, os fluxos críticos da plataforma. Entre estes fluxos destacam-se funcionalidades como o registo de candidatos, preenchimento de formulários, candidatura a vagas, entre outras.

Através do desenvolvimento de um conjunto de scripts de teste, será possível simular interacções reais dos utilizadores, abrangendo diferentes cenários, ambientes e condições de utilização. Este processo garantirá que o sistema se comporta correctamente, não apenas nas situações mais comuns, mas também em cenários excepcionais.

A implementação desta solução permitirá ganhos significativos, nomeadamente:

- Redução significativa do tempo de execução dos testes, permitindo ciclos de validação mais curtos;
- Diminuição da carga de trabalho manual, libertando os testadores para actividades mais analíticas e de maior valor agregado;
- Maior fiabilidade e consistência, eliminando a ocorrência de erros humanos frequentes nos testes manuais;
- Detecção antecipada de falhas e regressões, assegurando que eventuais problemas sejam identificados e resolvidos antes de chegar ao ambiente de produção;

- Reutilização dos scripts de teste, facilitando a manutenção e a execução recorrente dos testes em futuras actualizações do sistema;
- Execução dos testes em diferentes navegadores e dispositivos, garantindo que a experiência dos utilizadores é consistente, independentemente do ambiente utilizado;
- Aumento da cobertura de testes, possibilitando validar um maior número de cenários e fluxos do sistema;
- Contribuição directa para a estabilidade, segurança e fiabilidade da plataforma, assegurando uma melhor experiência aos utilizadores finais.

As partes interessadas envolvidas nesta solução incluem:

- Equipa de desenvolvimento (Techsolutions) – responsável pela implementação da solução de automação, bem como pela criação, manutenção e actualização dos scripts de teste.
- Equipa funcional (SAL Group) – encarregada de validar se as funcionalidades testadas respondem efectivamente aos requisitos operacionais e às necessidades do negócio.
- Utilizadores finais (empresas e candidatos) – beneficiários directos da maior estabilidade, fiabilidade e qualidade proporcionadas por uma plataforma devidamente testada e validada.

Deste modo, a proposta de implementação da automação de testes, suportada pelo Playwright, surge como uma solução robusta, sustentável e alinhada com as boas práticas actuais no desenvolvimento de software. Esta abordagem contribuirá significativamente para o sucesso e a longevidade da plataforma Headhunting, assegurando a sua evolução de forma segura e eficiente.

4.2. Requisitos Funcionais

Os requisitos funcionais correspondem às funcionalidades que o sistema deve oferecer para satisfazer as necessidades dos utilizadores e garantir o seu correcto funcionamento.

Estes requisitos descrevem, de forma clara e objectiva, os comportamentos esperados da aplicação em situações específicas de uso.

Para efeitos deste trabalho, foi fornecida uma lista contendo os requisitos funcionais prioritários da aplicação, os quais serviram de base para a elaboração e implementação dos testes. Cada um dos testes desenvolvidos tem como objectivo validar o correcto cumprimento desses requisitos, assegurando que a aplicação responde conforme o esperado em diversos cenários.

A seguir, será apresentada a tabela de requisitos funcionais, acompanhada das respectivas descrições e identificações, que servirão de referência para as fases subsequentes deste trabalho.

ID	Requisito	Cenário de Teste	Prioridade
RF01	Criação da Conta/Registo	Registo de novo usuário como Recrutador e Talento	Essencial
RF02	Iniciar Sessão	Login com credenciais válidas	Essencial
RF03	Selecção de Pacote (Pagamento)	Selecionar e pagar pacote	Importante
RF04	Criar Proposta	Criar proposta para candidato	Importante
RF05	Visualizar e Filtrar Candidatos	Filtro por localização e qualificações	Importante
RF06	Selecionar Candidatos	Selecionar candidatos conforme pacote	Importante

RF07	Iniciar Conversa	Iniciar nova Conversa	Importante
RF08	Enviar Proposta	Enviar proposta existente	Importante
RF09	Agendar Meeting	Agendar reunião por Zoom ou Google Meet	Importante
RF10	Publicidade	Criar publicação patrocinada	Importante
RF11	Envio/Recepção de Mensagens	Enviar mensagem e receber resposta	Importante
RF12	Métodos de Pagamento	Realizar pagamento com carteira móvel	Importante
RF13	Criação/Actualização de CV	Criar/editar CV	Importante
RF14	Receber Proposta	Candidato recebe nova proposta	Importante
RF15	Avaliar Proposta	Aceitar ou rejeitar proposta	Importante
RF16	Enviar Mensagens Personalizadas	Sistema sugere mensagens	Importante
RF17	Estatísticas	Acesso às estatísticas	Importante
RF18	Terminar Sessão	Logout do sistema	Importante

RF19	Gerir Contas	Criar, editar, desativar e remover contas	Importante
RF20	Gerir Pacotes	Criar e editar pacotes disponíveis	Importante
RF21	Ver Estatísticas Gerais	Administrador visualiza estatísticas	Importante

Tabela 2 - Requisitos Funcionais do sistema headhunting

Fonte: Tabela de autoria do autor

4.3. Implementação

Nesta secção, apresenta-se o processo de implementação da solução proposta, que recorre à utilização do framework Playwright para a automação de testes na plataforma de recrutamento Headhunting. O desenvolvimento da solução encontra-se estruturado em três componentes principais, arquitectura da solução, cenário de teste e desenvolvimento dos scripts de teste.

Numa primeira fase, descreve-se a arquitectura da solução, com o objectivo de clarificar o seu funcionamento interno e os principais elementos que a compõem. Esta arquitectura define a comunicação entre os diferentes componentes envolvidos.

De seguida, é apresentado o cenário de teste, que serve de base à demonstração prática da solução proposta. Este cenário descreve, de forma sequencial, os passos necessários para validar uma funcionalidade específica da plataforma, abrangendo o seu fluxo completo (end-to-end).

Por fim, procede-se à configuração do ambiente necessário para a implementação dos testes. Esta etapa inclui:

- A instalação do Visual Studio Code, que será utilizado como editor de código para o desenvolvimento dos scripts;
- A instalação do Playwright, que constitui o motor responsável pela execução dos testes automatizados;
- A configuração dos navegadores necessários, com especial enfoque no Google Chrome, que será utilizado para a simulação das interacções com a plataforma.

Com o ambiente preparado, o cenário de teste definido em linguagem natural é traduzido para notação Gherkin, facilitando a sua posterior conversão em código JavaScript. Este código, desenvolvido com base nas funcionalidades do Playwright, permite a execução automatizada dos testes, simulando de forma precisa as interacções dos utilizadores com a plataforma Headhunting.

4.3.1. Arquitetura da Solução Proposta

Dado que o presente trabalho se centra na implementação de automação de testes utilizando a ferramenta Playwright, a arquitectura da solução proposta resume-se na arquitectura técnica do Playwright. Esta arquitectura define os principais componentes, protocolos de comunicação e mecanismos de controlo que permitem a execução eficiente e robusta dos testes automatizados.

A compreensão detalhada da arquitetura do Playwright é, portanto, fundamental para a concepção e desenvolvimento da solução, uma vez que a ferramenta é a base técnica que suporta toda a lógica de automação implementada no presente trabalho.

O Playwright foi concebido para proporcionar uma abordagem moderna e eficiente à automação de testes em aplicações web. A sua arquitectura baseia-se na utilização de WebSockets para comunicação entre o cliente (onde os testes são escritos) e o servidor (o browser propriamente dito), o que permite uma execução mais rápida e contínua, em comparação com ferramentas que utilizam o protocolo HTTP, como o Selenium.

Esta arquitectura foi concebida para proporcionar uma comunicação eficiente e fluida entre os scripts de teste e os navegadores. Uma das suas características principais é o facto de incluir os binários dos navegadores directamente com a sua biblioteca,

permitindo um controlo mais directo sobre o comportamento do navegador sem necessidade de servidores externos.

O Playwright estabelece uma única ligação ao navegador e envia os comandos necessários por essa ligação. Esta abordagem reduz a sobrecarga de comunicação, possibilitando uma execução mais rápida e eficiente dos testes. Em vez de enviar cada interacção como um pedido HTTP separado, o Playwright é capaz de agrupar múltiplas acções numa única transmissão, optimizando assim o desempenho.

A comunicação entre o script de teste (cliente) e o servidor do Playwright é realizada através de um protocolo baseado em WebSocket. Esta ligação permanece activa durante toda a execução do teste, permitindo uma troca contínua de mensagens. Quando um teste é iniciado, o script utiliza esta ligação WebSocket para transmitir comandos ao servidor do Playwright. Estes comandos, que orientam o servidor na realização de diversas tarefas, como visitar um URL, interagir com elementos na interface ou capturar imagens do ecrã, são frequentemente transmitidos no formato JSON.

Esta arquitectura modular e directa é um dos factores que contribuem para a eficácia do Playwright, permitindo testes robustos, rápidos e com menor complexidade na comunicação entre os componentes envolvidos.

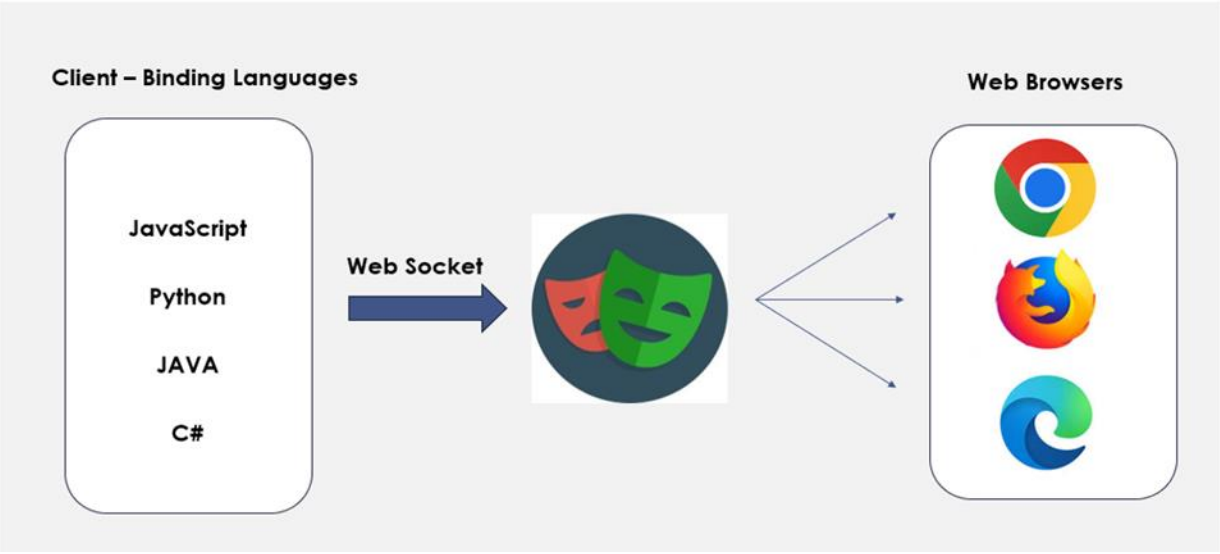

Figura 14 - Arquitetura do Playwright

Fonte: adaptado de *Architecture Breakdown: Selenium, Cypress and more por TestingMavens*.

Na Figura 14, pode observar-se como funciona a arquitectura do Playwright. A arquitectura foi dividida em vários componentes-chave, os quais são explicados de seguida (Pathak, 2025):

- **Código do Cliente:** é neste componente que se escrevem os scripts de teste utilizando a API do Playwright. O código do lado do cliente contém os comandos que definem as interacções com o navegador, tais como navegar entre páginas, clicar em elementos, preencher formulários, entre outros.
- **Ligação WebSocket:** o Playwright opera com base num modelo cliente-servidor, em que o código do lado do cliente comunica com o componente do lado do servidor através de uma ligação WebSocket. Esta ligação é estabelecida no início do teste e permanece activa durante toda a sua execução.
- **Servidor Playwright:** o servidor do Playwright é responsável por receber os comandos enviados pelo cliente, executá-los numa instância do navegador (Chromium, Firefox ou WebKit) e devolver as respostas. É o servidor que gere as instâncias do navegador e orquestra o seu comportamento com base nos comandos recebidos.
- **Formato JSON:** os comandos escritos no código de teste são traduzidos para o formato JSON (JavaScript Object Notation) antes de serem transmitidos ao servidor Playwright. O formato JSON é um meio leve de intercâmbio de dados, fácil de ler e escrever tanto por humanos como por máquinas.
- **Processo de Execução:** quando um teste é iniciado, o código do lado do cliente é executado. Os comandos definidos são serializados em formato JSON e transmitidos através da ligação WebSocket para o servidor Playwright. O servidor interpreta estes comandos e executa as acções correspondentes na instância do navegador.
- **Transmissão de Respostas:** após a execução de cada comando, o servidor Playwright devolve uma resposta em formato JSON através da ligação

WebSocket. Esta resposta inclui informações sobre o sucesso ou falha do comando, bem como quaisquer dados ou resultados relevantes.

- **Processo Iterativo:** a execução do teste decorre de forma iterativa, com o cliente a enviar comandos, o servidor a executá-los, e as respostas a serem reenviadas ao cliente, até que todos os comandos do script de teste tenham sido processados.

4.3.2. Cenário de Teste

Um Cenário de Teste é uma descrição de alto nível de uma funcionalidade ou recurso a ser testado, delineando as condições sob as quais o teste será realizado. Ele se concentra no que testar em vez de como testar, servindo como um guia para a criação de casos de teste e garantindo uma cobertura abrangente da aplicação. (Browserstack, s.d)

Nesta secção, apresenta-se o cenário que será utilizado para demonstrar o funcionamento da solução proposta. Para esse efeito, será testado o Requisito Funcional RF01 – Criação de Conta, o qual consiste na validação do processo de registo de um novo utilizador com perfil de Candidato na plataforma.

O cenário proposto tem como objectivo simular, de forma automatizada, os passos percorridos por um utilizador ao criar uma conta na aplicação. A descrição deste processo será inicialmente apresentada utilizando a notação Gherkin, permitindo uma representação estruturada e compreensível do comportamento esperado. Adicionalmente, será utilizado um diagrama de sequência com vista a ilustrar a interacção entre o utilizador e os diferentes componentes do sistema durante a execução do cenário.

Notação Gherkin

Gherkin é uma linguagem simples e estruturada usada para escrever cenários de teste de uma forma que seja fácil de entender para partes interessadas técnicas e não técnicas. (Browserstack, s.d)

A Tabela apresenta, de forma resumida, os passos necessários para a criação de uma conta com o perfil de Candidato. A descrição detalhada do cenário, bem como os dados que serão utilizados para testar o sistema, encontra-se disponível no Apêndice.

Funcionalidade: Registo de Utilizador
Como utilizador visitante Quero poder registar-me como Recrutador ou Candidato Para aceder às funcionalidades da plataforma
Cenário: Registo como Candidato
Dado que estou na página inicial da plataforma Quando clico na opção "Registrar-se" E selecciono o perfil "Candidato" E preencho o formulário de dados pessoais E avanço para a secção de resumo profissional E preencho o resumo profissional E adiciono uma formação académica E adiciono uma experiência profissional E adiciono uma habilidade E adiciono uma certificação E adiciono uma referência profissional

E adiciono um idioma E avanço para a secção de credenciais E preencho o formulário de credenciais E aceito os termos e condições E submeto o formulário de registo Então devo ver uma mensagem de confirmação de conta criada

Tabela 3 - Registo de utilizador na notação Gherkin

Fonte: Imagem de autoria do autor

Diagrama de sequencia

De acordo com a IBM (s.d.), um diagrama de sequência é um diagrama UML que ilustra a sequência de mensagens trocadas entre objectos no contexto de uma interacção. Este tipo de diagrama é composto por um conjunto de objectos representados através de linhas de vida, bem como pelas mensagens que são transmitidas entre eles ao longo da execução da interacção.

O Apêndice, apresenta o diagrama de sequência que descreve o processo automatizado de criação de conta com o perfil de Candidato, utilizando o framework Playwright. Este diagrama ilustra a interacção entre os principais actores e componentes envolvidos no fluxo de execução do teste automatizado, desde a abertura da aplicação até à confirmação da criação bem-sucedida da conta.

O processo tem início com o utilizador (visitante), que desencadeia a execução do teste de registo. Este pedido é encaminhado para o sistema de testes automatizados, que, através do Playwright, interage com a aplicação web simulando as acções de um utilizador real. O teste inicia ao abrir a página inicial, navegar até à funcionalidade de registo e seleccionar o tipo de utilizador "Candidato".

De seguida, o script automatizado procede ao preenchimento sequencial dos dados solicitados, divididos por secções temáticas:

- Dados pessoais: inclui nome, data de nascimento, género, contacto, nacionalidade, e localização geográfica.
- Currículo: contempla o resumo profissional, seguido das secções de formação académica, experiência profissional, competências técnicas, certificações, referências e domínio de idiomas.
- Credenciais de acesso: inserção do endereço de correio electrónico, definição de palavra-passe e aceitação dos termos de utilização da plataforma.

Após o preenchimento de todas as secções, a aplicação web envia os dados submetidos para o servidor backend, o qual trata o pedido e comunica com a base de dados, onde as informações são armazenadas. Uma vez concluído o processo de registo, o servidor envia uma confirmação de sucesso à aplicação, que por sua vez devolve uma mensagem de sucesso ao sistema de testes automatizados, indicando que a conta foi criada com êxito.

Este diagrama permite visualizar, de forma clara e estruturada, os passos envolvidos na automatização do teste de criação de conta, bem como a interacção entre os componentes do sistema.

4.3.3. Desenvolvimento de Scripts de Automação

Para proceder à automação dos testes com o framework Playwright, tornou-se necessário configurar um ambiente de desenvolvimento adequado que permitisse a escrita, execução e validação dos scripts de teste. Esta configuração incluiu a instalação de ferramentas essenciais e a preparação do ambiente de execução.

Instalação do Visual Studio Code

O primeiro passo consistiu na instalação do Visual Studio Code (VS Code), um editor de código-fonte leve, gratuito e amplamente utilizado para o desenvolvimento de aplicações

e testes. O VS Code foi seleccionado pela sua compatibilidade com JavaScript/TypeScript, integração com sistemas de controlo de versões e suporte a extensões, incluindo aquelas relacionadas com automatização de testes.

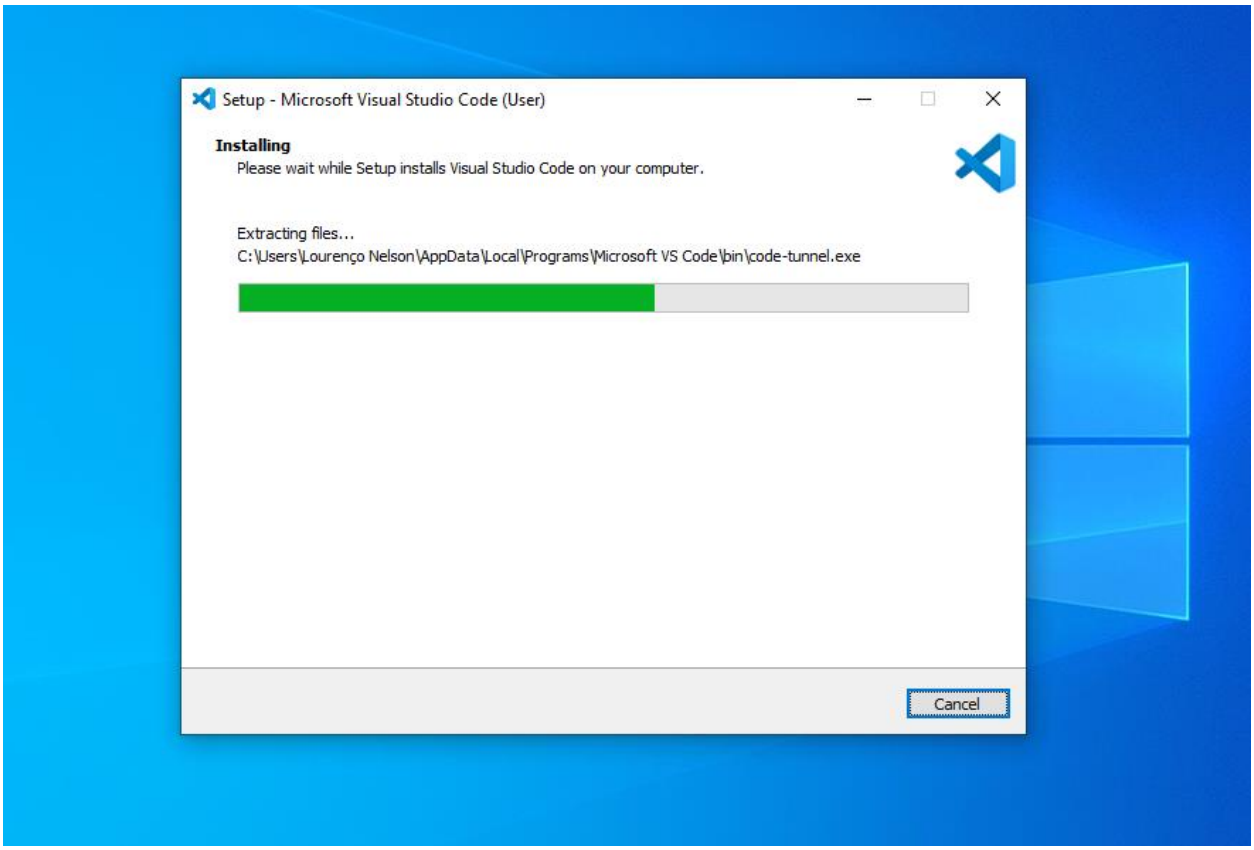

Figura 15 - Instalação do Visual Studio code

Fonte: Imagem de autoria do autor

Após a instalação, foram adicionadas extensões úteis, como o suporte a JavaScript e a extensão oficial do Playwright, para facilitar a escrita e execução dos testes.

Instalação do Framework Playwright

Com o editor de código devidamente instalado, procedeu-se à instalação do framework Playwright. A instalação foi realizada através do Node Package Manager (npm), requerendo, assim, a instalação prévia do Node.js.

A instalação do Playwright foi efectuada com o comando ¨*npm init playwright@latest*¨, executado no terminal do VS Code, dentro da directoria do projecto:

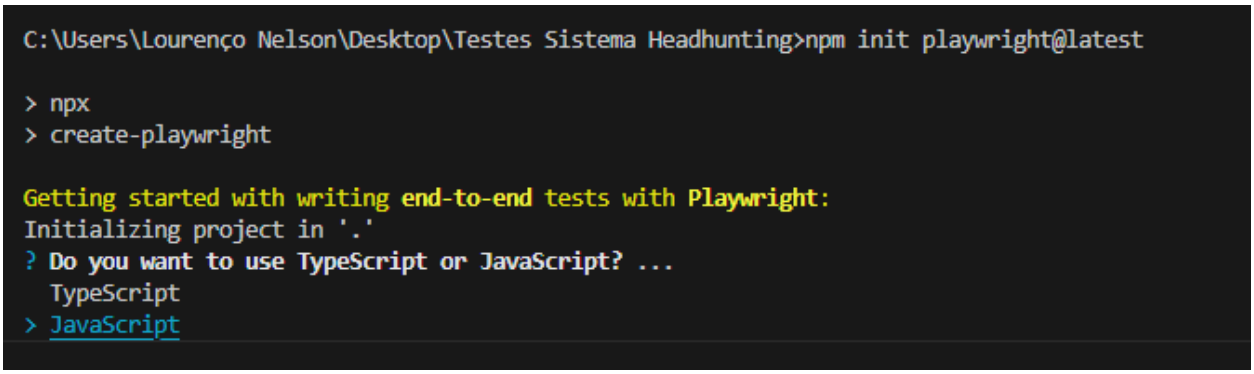

Figura 16 - Instalação do Playwright no Terminal

Fonte: Imagem de autoria do autor

Este comando inicializa um projecto com a estrutura recomendada pelo próprio Playwright, instalando as dependências necessárias, incluindo os binários dos navegadores suportados (Chromium, Firefox e WebKit), o que assegura maior controlo e consistência na execução dos testes.

Desenvolvimento do Script

Com o ambiente configurado, o projecto passou a dispor de todos os recursos necessários para o desenvolvimento dos testes automatizados.

- O editor VS Code como plataforma para escrever e organizar os scripts
- O Playwright instalado e configurado, com suporte para múltiplos navegadores;
- Um conjunto de ficheiros de configuração gerados automaticamente, incluindo playwright.config.ts, que permite definir parâmetros como o navegador a utilizar, tempo de espera, caminho para os relatórios de execução, entre outros;
- A possibilidade de escrever testes em JavaScript, usando a sintaxe recomendada pelo Playwright, com suporte a execuções em modo headless e não-headless.

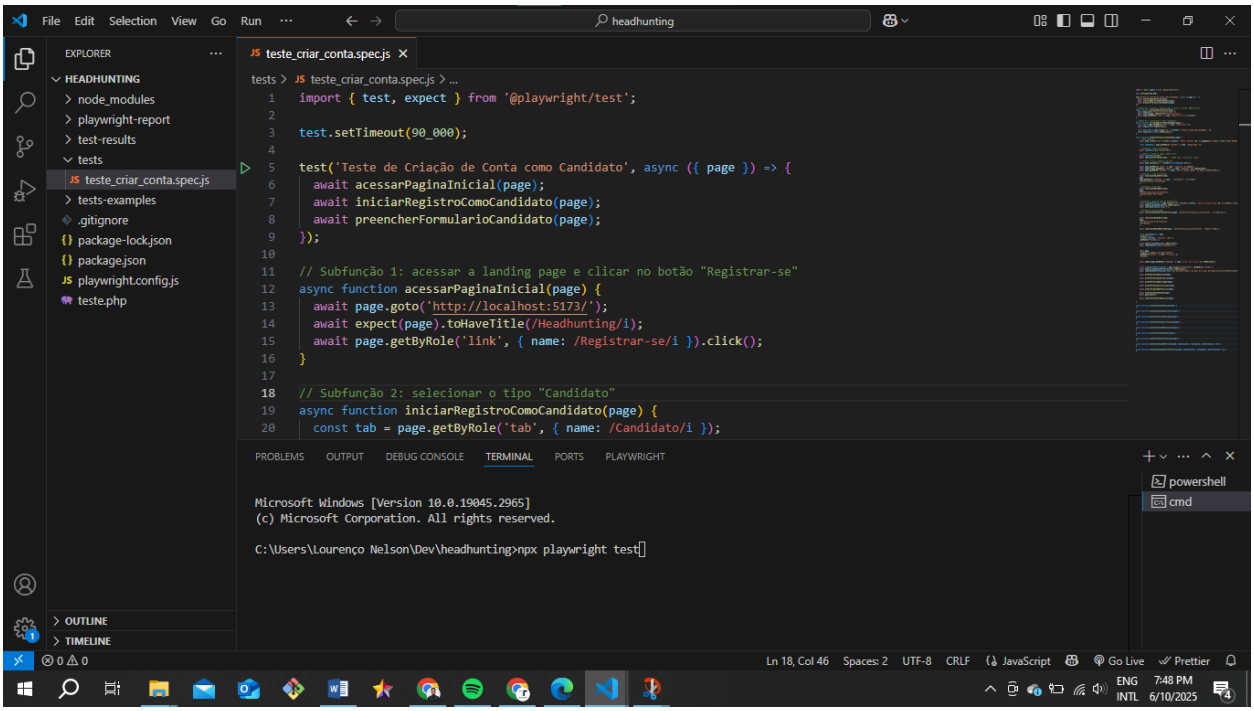

Figura 17- Script para automatizar a criação de conta de candidatos

Fonte: Imagem de autoria do autor

Este ambiente de desenvolvimento, devidamente preparado, permitiu iniciar a tradução dos cenários definidos na notação Gherkin para scripts de automação de testes, assegurando uma base sólida e eficiente para a demonstração da solução proposta, conforme ilustrado na figura. Para o efeito, foi desenvolvido um script denominado teste_criar_conta.spec.js, escrito em JavaScript, o qual implementa os passos definidos previamente, automatizando a interacção com o navegador e simulando o comportamento de um utilizador ao criar uma conta na plataforma. Este script é responsável pelo preenchimento automático dos formulários de registo, conforme especificado no cenário de teste, garantindo assim a validação do requisito funcional relativo à criação de conta.

5. Capitulo V - Apresentação dos Resultados

Neste capítulo, apresentam-se os resultados alcançados para a resolução do problema identificado nos capítulos anteriores. Com o propósito de cumprir o objectivo geral e os objectivos específicos delineados neste trabalho, foram seguidas diversas etapas metodológicas. O processo iniciou-se com a revisão da literactura, a qual proporcionou uma base teórica sólida sobre testes de software, qualidade e automação. Em seguida, realizaram-se observações sobre o processo actual de testes aplicado à plataforma Headhunting.

Este estudo permitiu identificar os principais desafios enfrentados no processo de testes manuais. Com base na análise desses constrangimentos, foi concebida uma solução centrada na automação de testes funcionais, utilizando a ferramenta Playwright. A implementação dessa solução culminou no desenvolvimento de scripts de automação que cobrem os principais fluxos funcionais do sistema.

5.1. Revisão de Literactura

A revisão da literactura teve como objectivo fundamentar teoricamente a proposta de automação de testes funcionais no sistema de recrutamento Headhunting, explorando conceitos essenciais relacionados com a qualidade de software, os testes e as ferramentas de automação. Inicialmente, evidenciou-se a crescente influência da tecnologia nas organizações e a consequente necessidade de desenvolver sistemas fiáveis, seguros e eficientes.

Foram abordadas definições sobre testes de software, bem como discutidas as diferenças entre os testes manuais e os testes automatizados, destacando-se as limitações inerentes aos primeiros. Em contrapartida, a automação de testes foi apresentada como uma solução moderna e eficaz, particularmente adequada a contextos de desenvolvimento ágil.

Por fim, procedeu-se à análise comparativa das principais ferramentas de automação de testes em navegadores, Selenium, Cypress e Playwright. Esta última foi seleccionada

como a mais adequada ao projecto, em virtude da sua arquitectura moderna, da sua facilidade de utilização, do suporte nativo a múltiplos navegadores e do elevado desempenho que proporciona.

5.2. Caso de estudo

No caso de estudo, procedeu-se à descrição da instituição Techsolutions, na qual se apresentou o seu ramo de actividade, bem como a missão, a visão e os valores organizacionais. Em seguida, foi descrita a situação actual do processo de testes da plataforma de recrutamento Headhunting, desenvolvida para o SAL Group.

Durante a análise da situação actual, foram identificados diversos constrangimentos associados à execução manual dos testes, designadamente:

- Morosidade no processo de validação, em virtude da repetição exaustiva dos testes;
- Fadiga e propensão a erros humanos, resultantes da natureza repetitiva do processo;
- Baixa cobertura de cenários de teste, sobretudo aqueles considerados mais complexos ou menos prováveis;
- Inconsistência nos resultados, devido a variações na execução por diferentes testadores;
- Custo elevado e atrasos nos cronogramas, decorrentes da forte dependência de recursos humanos;
- Dificuldade no rastreio e na documentação sistemática dos testes realizados;
- Fraca escalabilidade, sobretudo à medida que novas funcionalidades são integradas no sistema.

Estes factores evidenciam a necessidade de adoptar uma abordagem mais moderna e eficiente para o processo de validação, justificando a proposta de automação de testes apresentada neste trabalho.

5.3. Desenvolvimento da solução proposta

O desenvolvimento da solução proposta foi concretizado após uma análise cuidadadosa dos recursos disponíveis, complementada por um estudo aprofundado da literactura e pela observação directa do processo de testes actualmente praticado pela equipa da Techsolutions, na plataforma Headhunting.

A solução foi concebida com base na ferramenta Playwright, uma tecnologia moderna de automação de testes que se revelou particularmente adequada para este projecto, em virtude da sua arquitectura moderna, da sua facilidade de utilização, do suporte nativo a múltiplos navegadores e do elevado desempenho que proporciona. A escolha desta ferramenta foi fundamentada pelas suas vantagens técnicas e pela sua eficácia comprovada em ambientes de desenvolvimento ágil.

A implementação decorreu em três etapas principais, a configuração do ambiente de desenvolvimento, a modelação do cenário de teste, e a construção dos scripts de automação. O ambiente foi preparado com a instalação do Visual Studio Code e do framework Playwright, configurando-se as dependências necessárias para a criação e execução de testes.

Posteriormente, procedeu-se à tradução do cenário de teste definido na notação Gherkin para scripts escritos em JavaScript, os quais simulam a interacção de um utilizador com a plataforma. Estes scripts cobrem o processo de criação de conta com o perfil de candidato, incluindo a navegação entre páginas, o preenchimento de formulários e a submissão dos dados. A simulação é realizada no navegador Chrome, possibilitando uma experiência próxima da realidade.

A automação de testes com Playwright trouxe diversos benefícios práticos, entre os quais se destacam:

- Redução do tempo necessário para validar funcionalidades críticas da aplicação;
- Eliminação de tarefas manuais repetitivas, diminuindo o desgaste dos testadores;

- Aumento da fiabilidade e consistência nos resultados dos testes, com menor propensão a erros humanos;
- Execução multi-navegador, permitindo testar a aplicação em diferentes ambientes;
- Reutilização de scripts e possibilidade de monitoramento contínuo das funcionalidades testadas.

Esta solução contribui, assim, para a melhoria do processo de validação do sistema, oferecendo ganhos significativos em termos de qualidade, produtividade e escalabilidade.

6. Capitulo VI - Considerações Finais

6.1. Conclusões

O desenvolvimento deste trabalho permitiu compreender o papel fundamental dos testes de software, tanto para as organizações que desenvolvem sistemas como para os utilizadores finais que deles dependem. A prática de testes contribui directamente para a garantia da qualidade, fiabilidade e robustez dos sistemas, características essenciais no contexto tecnológico actual.

Face aos constrangimentos identificados durante o estágio na Techsolutions, nomeadamente o tempo excessivo despendido em testes manuais e a repetitividade das tarefas, a automação dos testes funcionais revelou-se a solução mais adequada. Esta abordagem proporcionou uma maior flexibilidade no processo de validação da plataforma Headhunting, permitindo uma cobertura mais abrangente de cenários e uma optimização significativa do tempo necessário à execução dos testes.

Dada a limitação de recursos, a automação demonstra-se especialmente vantajosa, libertando os profissionais de actividades repetitivas e susceptíveis a erro humano. Paralelamente, o aumento da quantidade e frequência de testes, possibilitado pela automação, contribuiu para uma maior fiabilidade do sistema, através de uma validação sistemática de diferentes fluxos funcionais.

Como resultado final, foi implementado um processo automatizado de testes utilizando o framework Playwright, o qual permite a simulação automática da interacção de um utilizador com o sistema, através de um navegador. Esta solução viabiliza o cumprimento do objectivo geral do trabalho, bem como dos objectivos específicos traçados.

O primeiro objectivo foi alcançado através da revisão teórica apresentada no segundo capítulo, onde se abordam os fundamentos e benefícios da automação de testes funcionais. Em seguida, procedeu-se à selecção fundamentada da ferramenta de automação, com base numa análise comparativa entre as principais alternativas disponíveis. Um dos requisitos funcionais do sistema, nomeadamente o registo de um

utilizador com o perfil de candidato, foi mapeado, documentado por meio de um diagrama de sequência e descrito utilizando a notação Gherkin. Por fim, foi desenvolvido um script em JavaScript, recorrendo ao Playwright, que automatiza o processo de registo, concretizando, assim, todas as etapas propostas para a validação da solução.

6.2. Recomendações

Com base na experiência adquirida durante a realização deste estágio profissional, recomenda-se vivamente a adopção de ferramentas de automação de testes por parte de profissionais envolvidos em actividades de verificação e validação de sistemas. Embora seja comum considerar o processo de criação inicial de scripts de automação como moroso ou complexo, constatou-se, ao longo deste trabalho, que os avanços tecnológicos tornaram esta tarefa consideravelmente mais simples, acessível e vantajosa.

Ferramentas como o Playwright oferecem uma abordagem eficiente e fiável para a execução de testes automatizados. Uma vez desenvolvidos, os scripts podem ser reutilizados de forma recorrente, sempre que necessário, o que representa uma vantagem significativa face à execução manual, que exige a repetição constante das mesmas acções, aumentando o esforço envolvido e a probabilidade de ocorrência de erros.

A adopção da automação contribui, portanto, para a poupança de tempo e de recursos, ao mesmo tempo que promove um aumento na qualidade e na consistência do producto final. Por estas razões, recomenda-se fortemente a integração de práticas de automação nas rotinas de desenvolvimento e validação de software, como forma de responder de forma mais eficaz às exigências dos ambientes tecnológicos actuais.

Bibliografia

[1] - Leloudas, P. (2023). Introduction to software testing: A practical guide to testing, design, automation, and execution.

[2] - Microsoft. (n.d.). End-to-end testing. Obtido em 9 de Junho de 2025, de Microsoft: https://microsoft.github.io/code-with-engineering-playbook/automated-testing/e2e-testing/

[3] - Gerhardt, T. E., & Silveira, D. T. (Orgs.). (2009). Métodos de pesquisa.

[4] - Gil, A. C. (2017). Como elaborar projetos de pesquisa. 6ª edição. São Paulo: Atlas.

[5]. Bartié, A. (2002). Garantia da qualidade de software: adquirindo maturidade organizacional. 13ª reimpressão. Rio de Janeiro: Elsevier.

[6]. Rocha, A. C. O. (2023). Simplificando teste de software: compartilhando experiências. 1ª edição.

[7]. IEEE Computer Society. (n.d.). O que é qualidade de software? Obtido em 9 de Junho de 2025, de https://www.computer.org/resources/what-is-software-quality

[8]. Associação Brasileira de Normas Técnicas (ABNT). (2005). NBR ISO 9000:2005 - Sistemas de gestão da qualidade - Fundamentos e vocabulário (PDF). Obtido em 9 de Junho de 2025, de https://qualidadeuniso.wordpress.com/wp-content/uploads/2012/09/nbr-iso-9000-2005.pdf

[9]. Pathak, K. (2025). Web automation testing using Playwright: End-to-end, API, accessibility, and visual testing using Playwright (1ª ed.). BPB Publications.

[10]. Ammann, P., & Offutt, J. (2017). Introduction to software testing (2ª ed.). Cambridge University Press.

[11]. Sommerville, I. (2011). Engenharia de software (9ª ed., Em português do Brasil). Pearson Universidades.

[12]. BrowserStack. (s.d.). Selenium testing. Obtido em 10 de Junho de 2025, de https://www.browserstack.com/selenium

[13]. GeeksforGeeks. (2025, 6 de janeiro). Selenium Automation tool – Software Engineering. Obtido em 10 de junho de 2025, de https://www.geeksforgeeks.org/software-engineering-selenium-an-automation-tool/#selenium-ide

[14]. García, B., del Alamo, J. M., Leotta, M., & Ricca, F. (2024). Exploring browser automation: A comparative study of Selenium, Cypress, Puppeteer, and Playwright.

[15]. BrowserStack. (s.d.). Cypress automation tutorial. Obtido em 10 de Junho de 2025, de https://www.browserstack.com/guide/cypress-automation-tutorial

[16]. GeeksforGeeks. (2025, 6 de Janeiro). Introduction to Cypress Testing Framework. Consultado em 10 de Junho de 2025, de https://www.geeksforgeeks.org/introduction-to-cypress-testing-framework/#cypress-architecture

[17]. TestingMavens. (s.d.). Architecture Breakdown: Selenium, Cypress and more. Consultado em 10 de Junho de 2025, de https://www.testingmavens.com/blogs/architecture-breakdown-selenium-cypress-and

[18]. Santos, L. D. V., & Oliveira, C. V. S. (2017). Introdução à garantia de qualidade de software. Cia do eBook.

Apêndices

Apêndice 1 – Diagrama de Sequencia

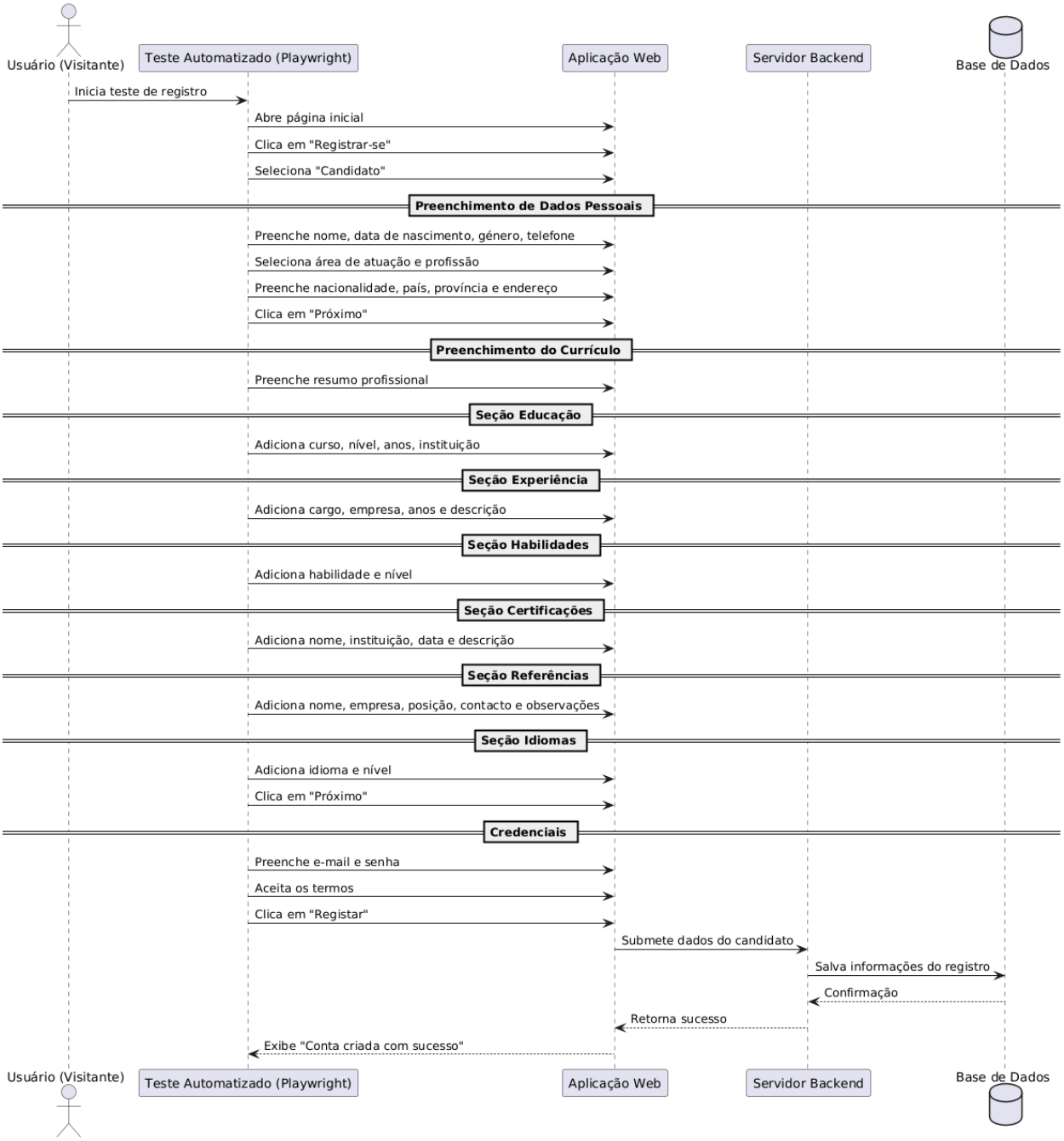

Figura A1 - Diagrama de sequencia para a criação de conta

Fonte: Imagem de autoria do autor

Apêndice 2 – Especificação de cenário na notação Gherkin

Funcionalidade de Registo de Utilizador	
Como utilizador visitante	
Quero poder registar-me como Recrutador ou Candidato	
Para aceder às funcionalidades da plataforma	
Cenário: Registo como Candidato	
Dado que estou na página inicial da plataforma	
Quando clico em "Registrar-se"	
E seleciono o perfil "Candidato"	
E preencho o formulário com:	
Nome completo	Joana Mendes
Data de nascimento	15/05/1990
Sexo	Feminino

Telefone	+258841234567
Área de atuação	Desenvolvimento de Software
Profissão	Desenvolvedor Full Stack
Anos de experiência	5
Nacionalidade	Moçambicano
País	Moçambique
Província	Maputo Cidade
Endereço	Maxaquene A
E clico no botão "Próximo"	
E preencho o resumo profissional com "Sou um profissional com mais de 5 anos de experiência em desenvolvimento de software..."	
E adiciono uma formação com os dados:	
Curso	Engenharia Informática
Nível	Licenciado

Início	2020
Fim	2025
Instituição	Universidade Eduardo Mondlane
E adiciono uma experiência com os dados:	
Cargo	Desenvolvedor de Software
Empresa	Techsolutions
Início	2020
Fim	2024
Descrição	Atuei no desenvolvimento com Vue.js e Laravel
E adiciono uma habilidade:	
Habilidade	Programação em Python
Nível	80%
E adiciono uma certificação com os dados:	

Nome	Django
Instituição	Udemy
Data de emissão	20/06/2024
Descrição	Curso intensivo com Django e APIs
E adiciono uma referência com:	
Nome	Helio Herculiano
Organização	Techsolutions
Posição	Chefe do departamento de sistemas
Telefone	+258841234567
Observações	Trabalhámos juntos 3 anos
E adiciono um idioma com:	
Idioma	Português
Nível de proficiência	AVANÇADO
E clico em "Próximo"	

E preencho as credenciais com:	
Email	lourenco.n.manhica@gmail.com
Senha	Q6v%#hFsTA!Y
E aceito os termos e condições	
E submeto o formulário de registo	
Então devo ver uma mensagem de confirmação de conta criada	

Tabela A2 - Registo de Utilizador

Fonte: Tabela de autoria do autor

Apêndice 3 – Formulário de Criação de Conta como Candidato

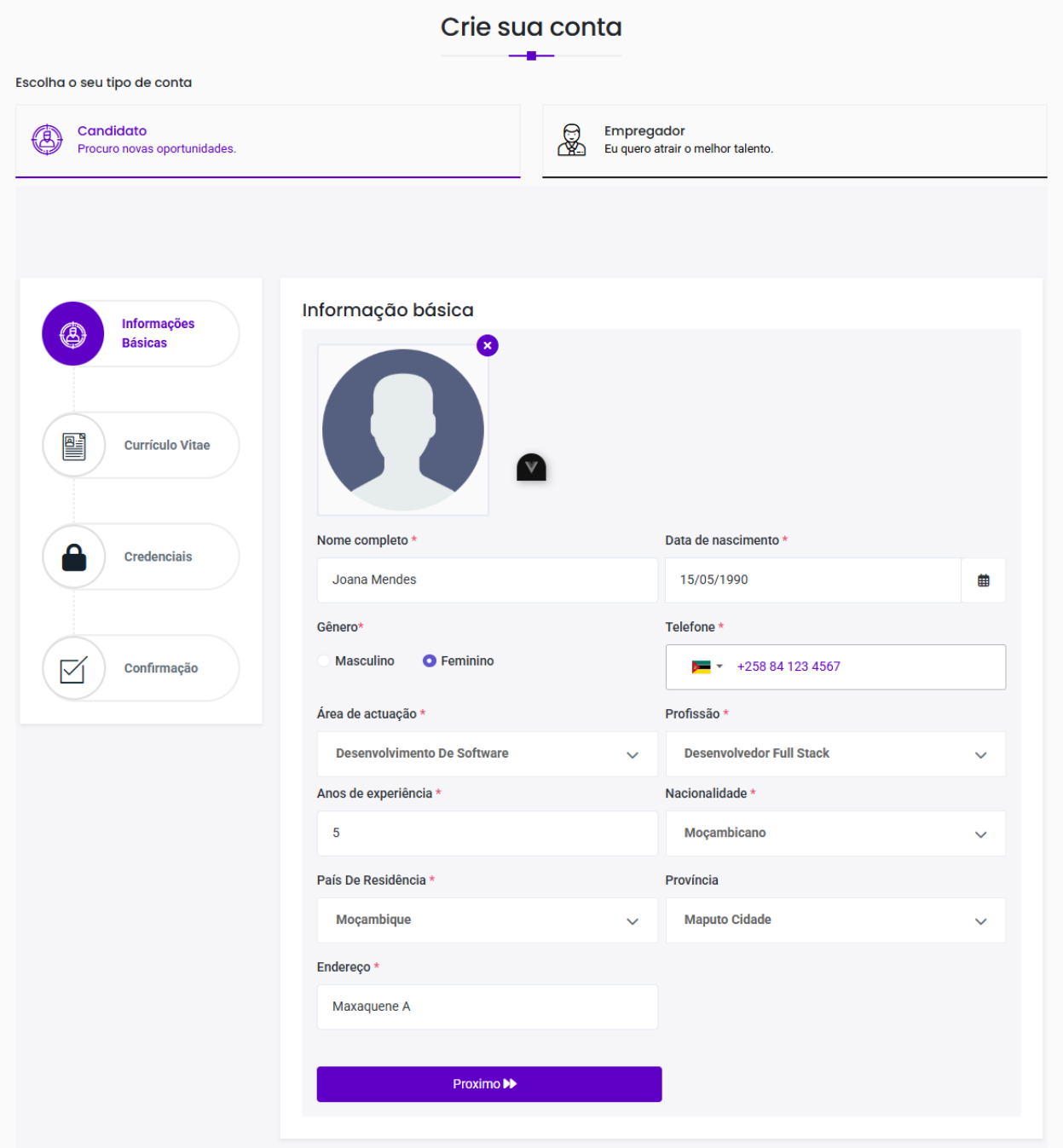

Figura A3.1 - Formulário de Registo de candidato: Informações Básicas

Fonte: Imagem de autoria do autor

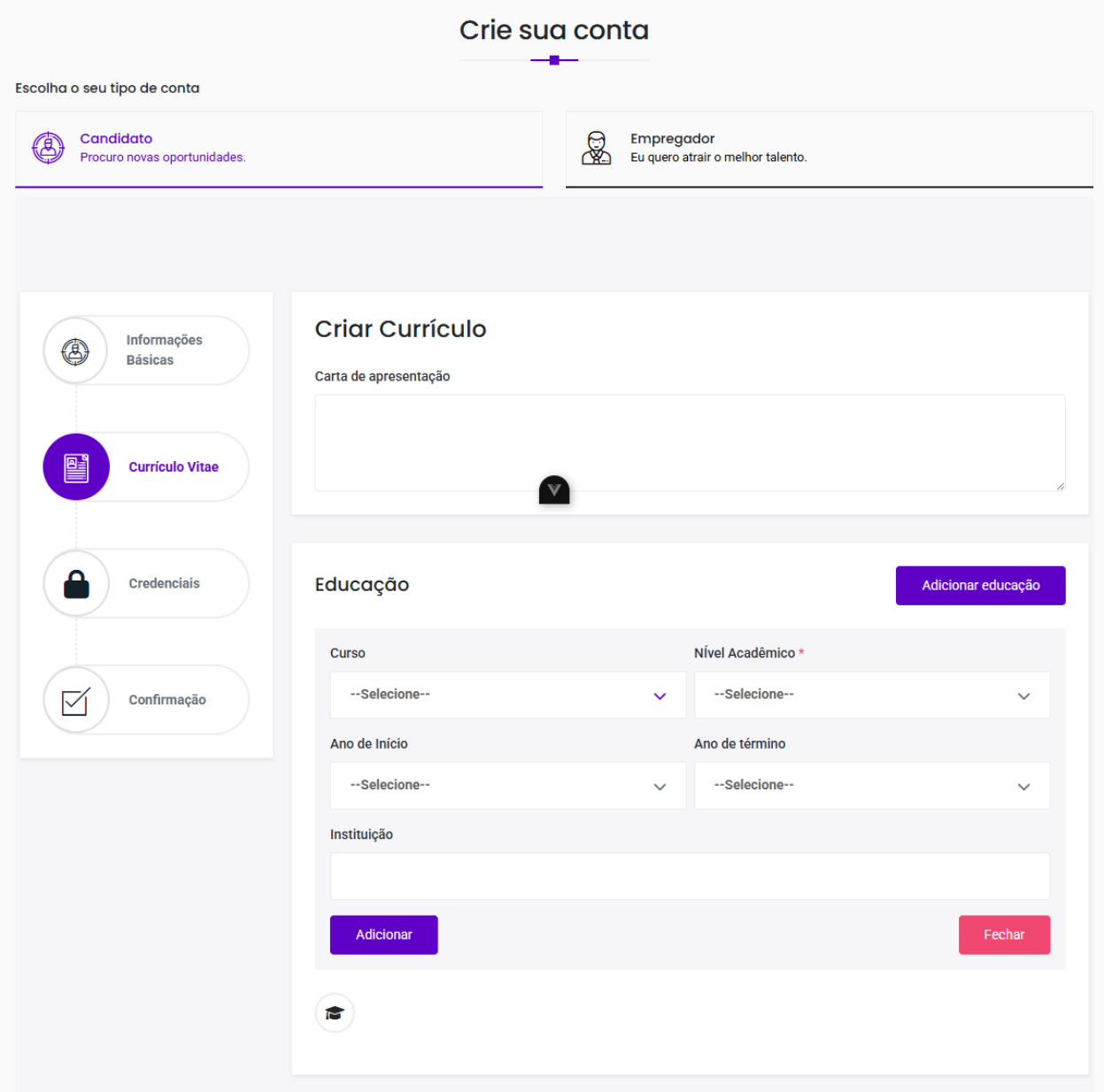

Figura A3.2 - Formulário de Registo de Candidato: Criação de Currículo

Fonte: Imagem de autoria do autor

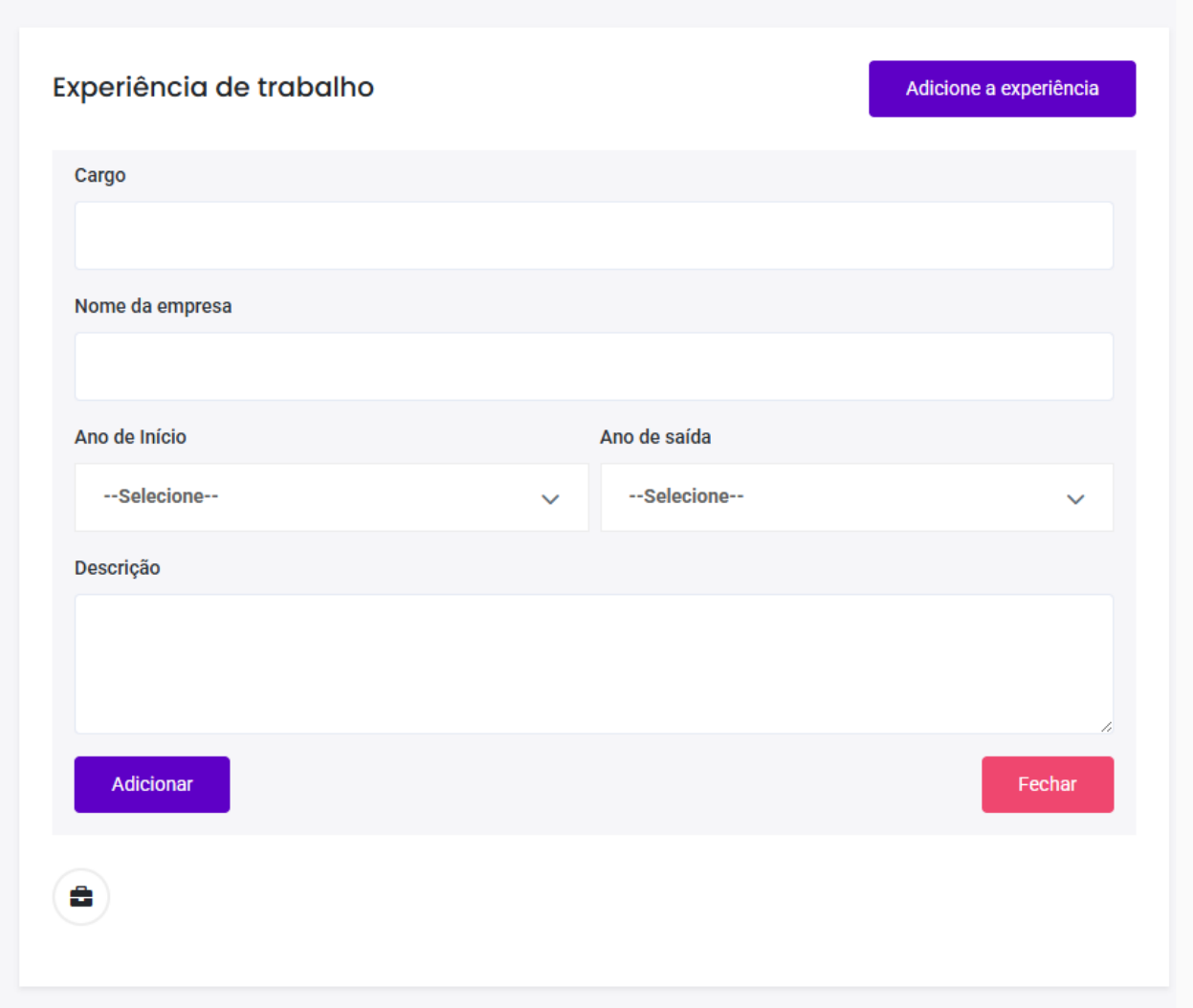

Figura A3.3 - Formulário de Registo de Candidato: Criação de Currículo

Fonte: Imagem de autoria do autor

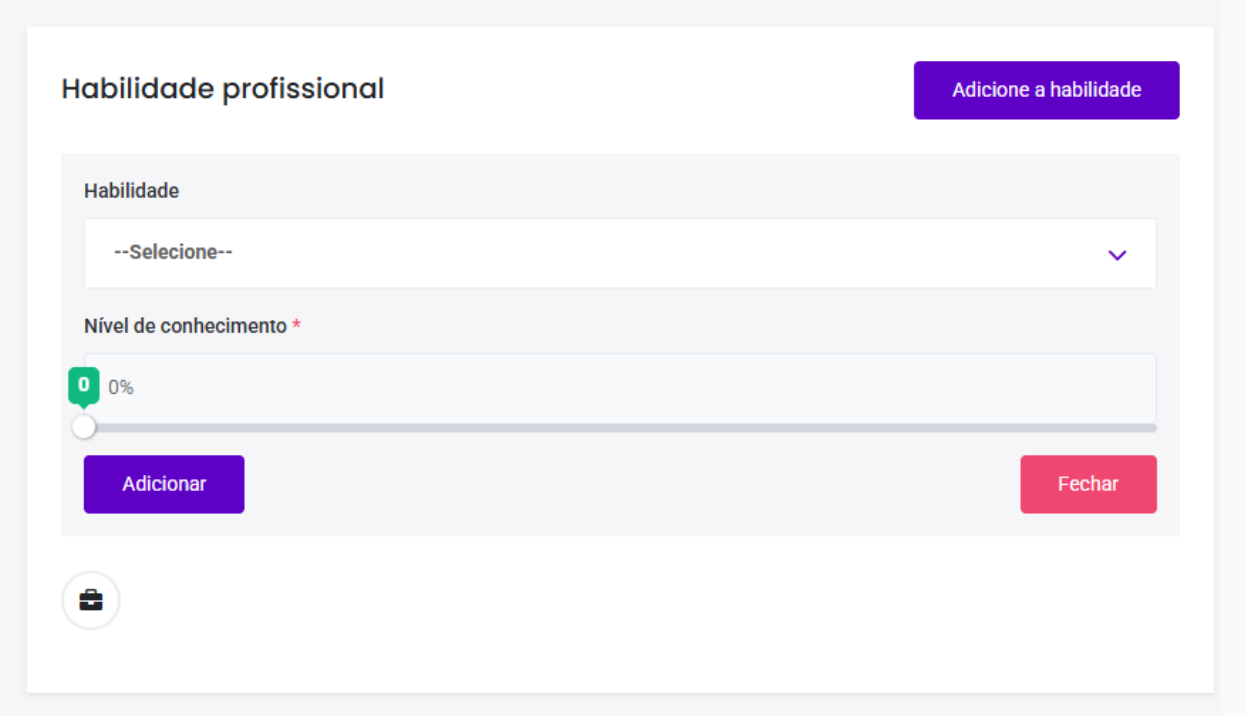

Figura A3.4 - Formulário de Registo de Candidato - Criação de Currículo

Fonte: Imagem de autoria do autor

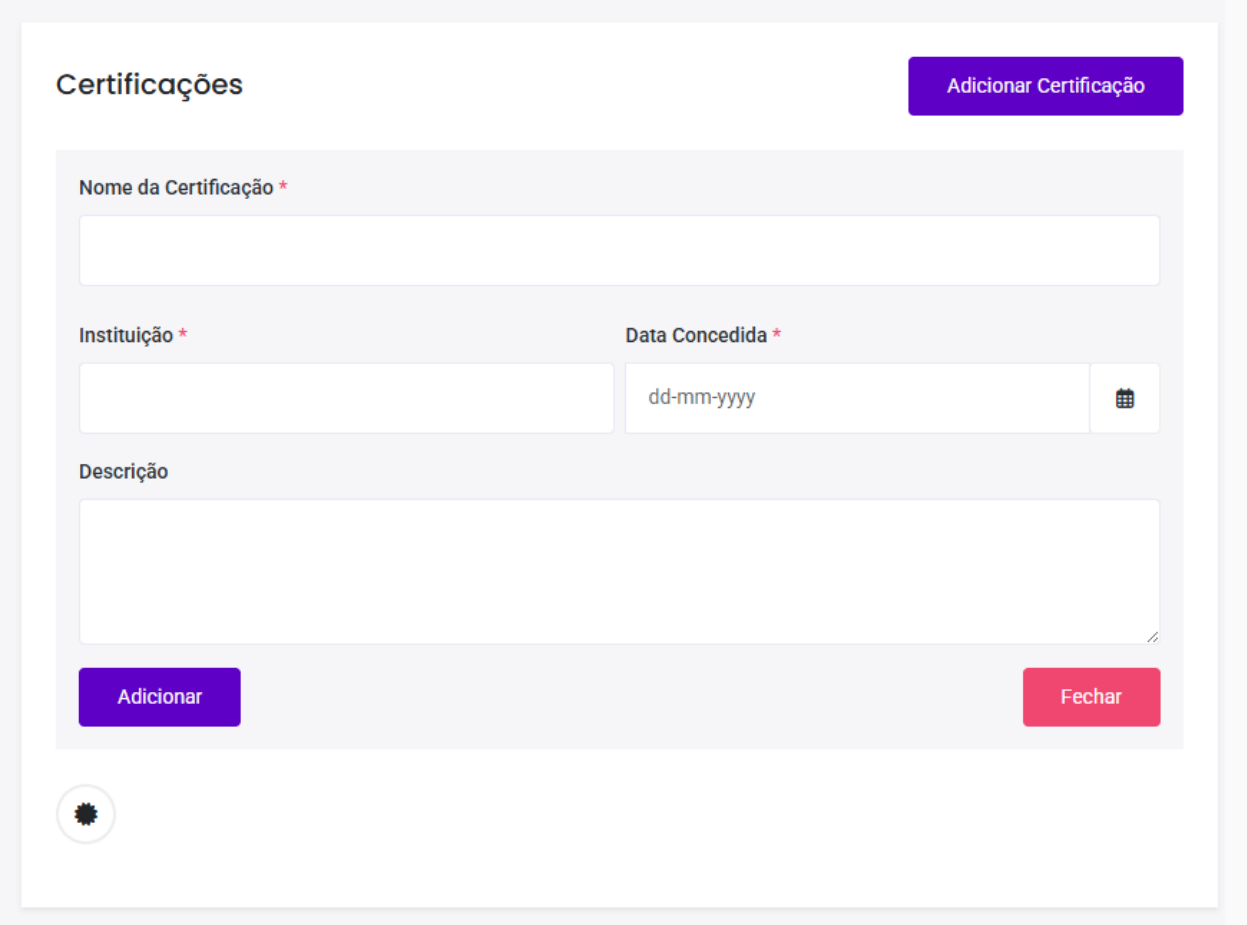

Figura A3.5 - Formulário de registo de candidato: criação de currículo

Fonte: Imagem de autoria do autor

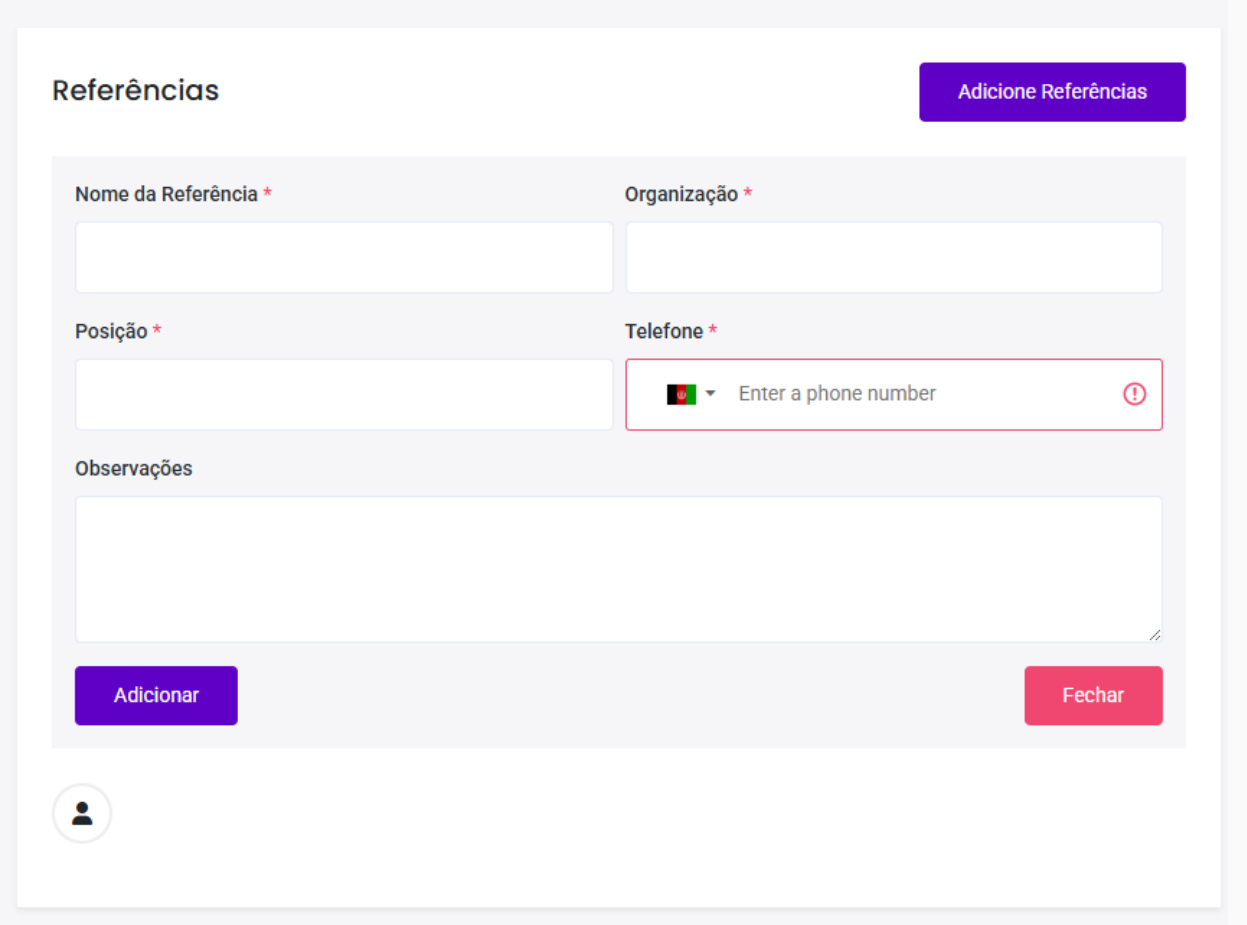

Figura A3.6 - Formulário de registo de candidato: criação de currículo

Fonte: Imagem de autoria do autor

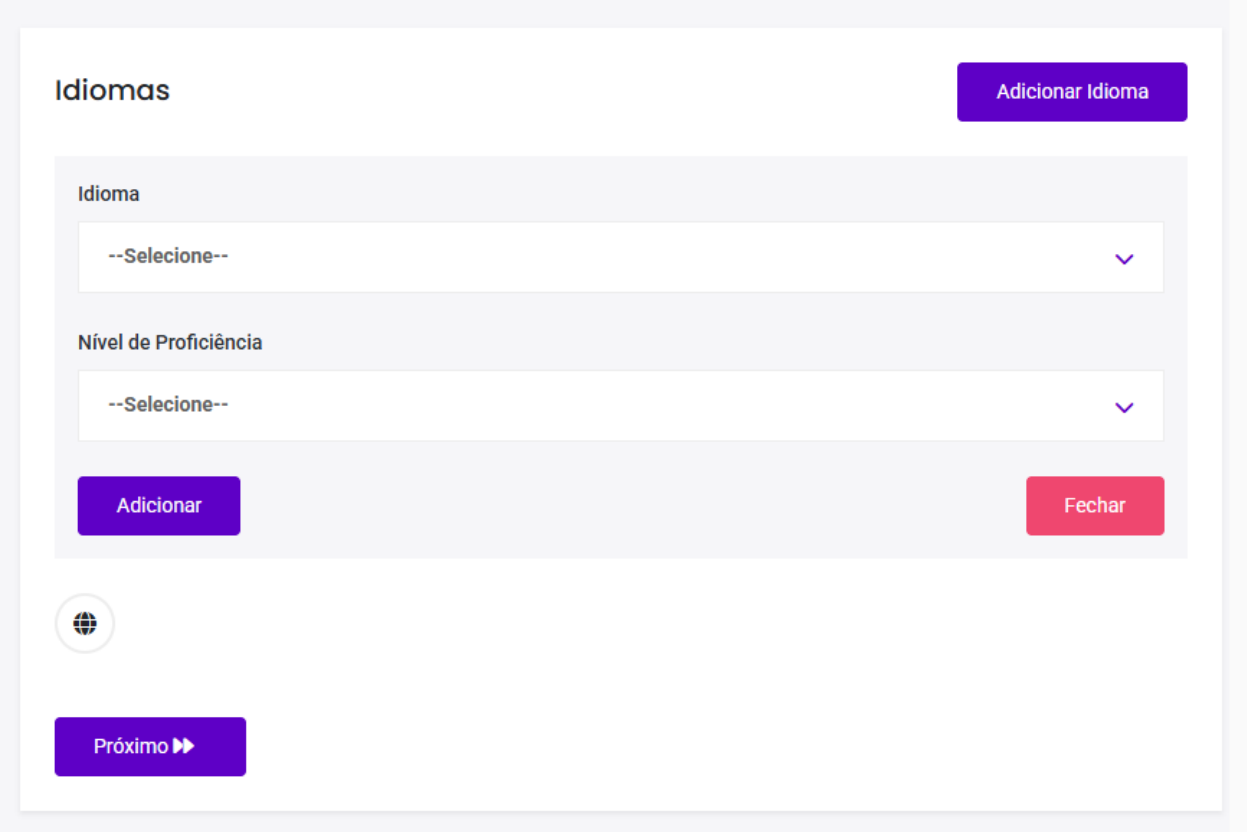

Figura A3.7 - Formulário de registo de candidato: criação de currículo

Fonte: Imagem de autoria do autor

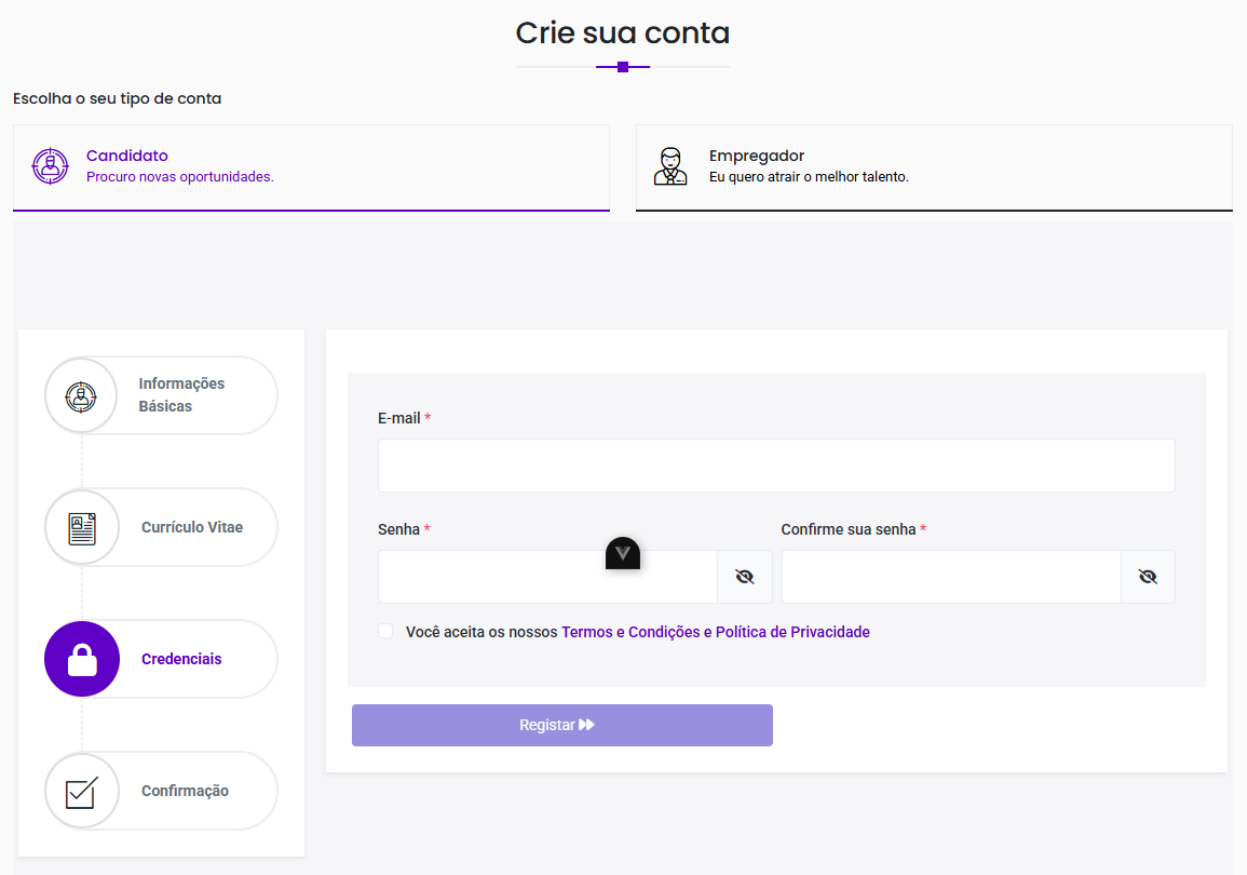

Figura A3.8 - Formulário de registo de candidato: criação credenciais

Fonte: Imagem de autoria do autor

Apêndice 4 – Ambiente de desenvolvimento Configurado

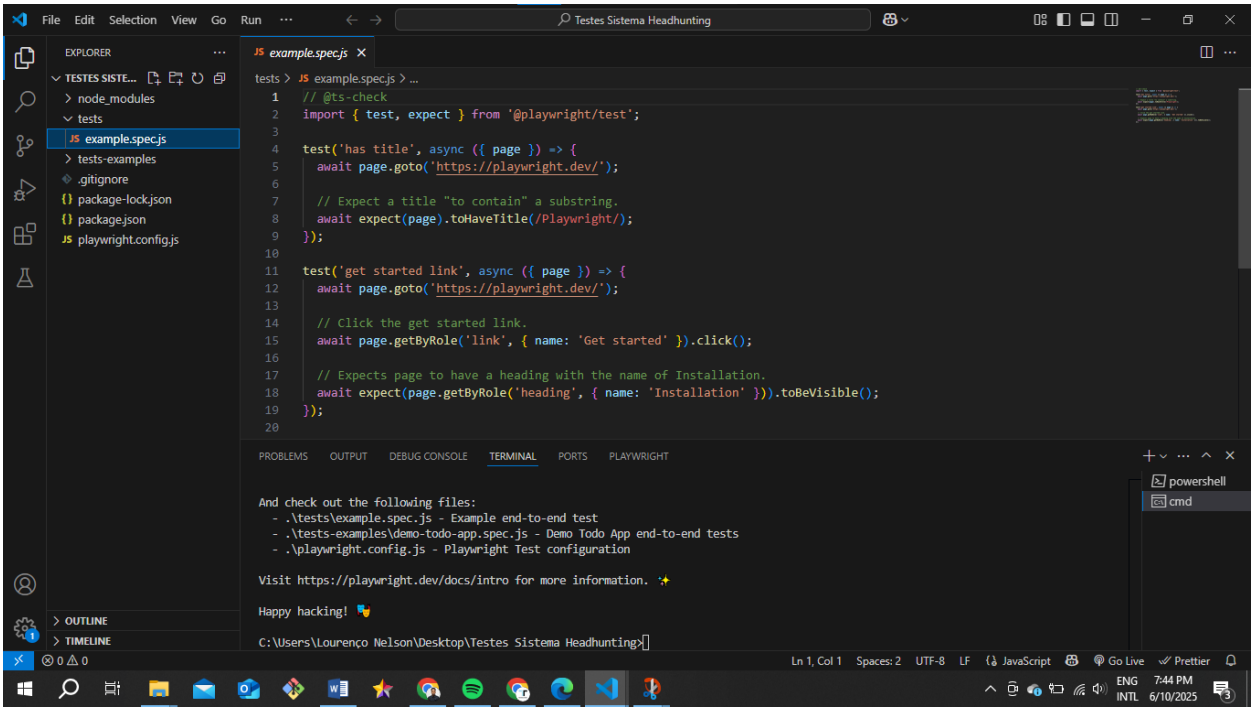

Figura A4 - ambiente de desenvolvimento com VS Code e Playwright Configurado

Fonte: Imagem de autoria do autor

Apêndice 5 – Interface de relatórios de testes executados

Figura A5 - Relatório de execução de testes no Playwright

Fonte: Imagem de autoria do autor