Automação de testes de um *software* da área da saúde

Ana Catarina Araújo Silva

UMINHO | 2023

Universidade do Minho
Escola de Engenharia

Ana Catarina Araújo Silva

Automação de testes de um *software* da área da saúde

outubro de 2023

Universidade do Minho
Escola de Engenharia

Ana Catarina Araújo Silva

Automação de testes de um *software* da área da saúde

Dissertação de Mestrado

Mestrado em Engenharia de Sistemas

Trabalho efetuado sob a orientação de

Professor Lino António Antunes Fernandes Costa

Professor João Alexandre Baptista Vieira Saraiva

Outubro 2023

AGRADECIMENTOS

Depois de terminado todo processo, cresce uma sensação de dever cumprido.

Ao longo do estágio foi-me possível um contacto com o mundo do trabalho de forma bastante desafiadora e que só foi conseguido porque tive sempre a meu lado grandes pilares, que auxiliaram e apoiaram todo o processo. É a essas pessoas que quero agradecer.

Em primeiro lugar, agradeço à Ciberbit, S.A., por me ter dado a oportunidade de estagiar com eles. A toda a estrutura e colegas que sempre me fizeram sentir em casa e me ajudaram em tudo o que conseguiam.

À Elisabete Mendes, pela orientação sempre preocupada e disponível na empresa, por todos os ensinamentos e apoio ao longo do estágio. A sua ajuda e conhecimentos de testagem e qualidade foram sem dúvida fundamentais.

Aos meus orientadores académicos, os professores João Saraiva e Lino Costa, por toda a preocupação e paciência ao longo não só do estágio como de todo o desenvolvimento deste documento. Foram importantes todas as suas transmissões de valores e conselhos sempre assertivos, assim como a sua disponibilidade para me acompanharem neste processo.

A todos os docentes que ao longo destes dois anos me incutiram valores e transmitiram os ensinamentos necessários para que conseguisse realizar este estágio.

A toda a minha família, em particular aos meus pais, por toda a paciência e por todos os esforços que fizeram e continuam a fazer, para que eu consiga alcançar os meus sonhos, por me acompanharem e motivarem a fazer mais e melhor. O orgulho que demonstram ajuda a ultrapassar tudo mais facilmente.

A todos os amigos, a quem considero família, por ao longo de todos estes anos, terem estado sempre a meu lado a percorrer este caminho e ultrapassando comigo todas as adversidades. Obrigada por nunca desistirem e estarem lá a amparar sempre que eu precisava.

DECLARAÇÃO DE INTEGRIDADE

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

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

RESUMO

Automação de testes de um *software* da área da saúde

A constante evolução tecnológica exige que os programas e todo o seu desenvolvimento sejam cada vez mais confiáveis, sendo por isso imprescindível para as empresas que exista uma aposta na área da qualidade do software. A minimização do tempo e dos gastos, tanto monetários como de recursos, trazida pela testagem automática para a empresa, são sem dúvida uma mais valia para o produto. Foi principalmente por estas razões que a Ciberbit, S.A., decidiu apostar num projeto de desenvolvimento de testes automáticos para o seu software de gestão hospitalar.

É necessária uma familiarização com o I'MTHOM, o programa que será sujeito à testagem automática, de forma a termos uma noção de quais os procedimentos prioritários para testagem e quais as etapas do teste. Foram estudadas duas ferramentas de automação, tanto de forma teórica como prática, para percebermos que o *Selenium* foi a opção que melhor se enquadrou nos requisitos da empresa. Foi então que se fizeram um total de 578 testes, principalmente relativos à abertura correta dos menus, e realizados consecutivos *builds* dos mesmos numa pipeline criada para esse intuito, de forma a perceber o comportamento dos mesmos em cada *build* tanto a nível de falhas como a nível de tempos de execução. Esta pipeline, como era executada diariamente, permite que as falhas do software sejam capturadas mais rapidamente e, consequentemente, corrigidas também com maior rapidez, melhorando assim o bom funcionamento do programa e o sucesso dos desenvolvimentos. Todos estes fatores levam ao aumento da produtividade da empresa.

PALAVRAS-CHAVE

Automação de Testes, Qualidade, *Selenium,* Testagem de *Software*

ABSTRACT

Test automation for healthcare software

Constant technological evolution requires programmes and all their development to become increasingly reliable, making it essential for companies to invest in software quality. The minimisation of time and costs, both monetary and in terms of resources, brought about by automated testing is undoubtedly an asset for the product. It was mainly for these reasons that Ciberbit, S.A. decided to invest in a project for the development of automated tests for its hospital management software.

Familiarity with I'MTHOM, the programme subject to automated testing, is necessary to understand the priority procedures for testing and the testing stages. Two automation tools were studied, both theoretically and practically, to determine that Selenium was the option that best suited the company's requirements. A total of 578 tests were then carried out, mainly related to the correct opening of menus, and consecutive builds were executed in a pipeline created for this purpose, to understand their behaviour in each build, in terms of failures and execution times. As this pipeline was run daily, it allowed software faults to be identified more promptly and, consequently, corrected more rapidly, thus improving the smooth running of the software. All these factors lead to increased company productivity.

KEYWORDS

Test Automation, Quality, Software Testing, Selenium

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABELAS

LISTA DE ABREVIATURAS, SIGLAS E ACRÓNIMOS

API- *Application Programming Interface*

CI- *Continuous Integration*

E2E- *End-to-End*

GUI- *Graphical User Interface*

QA- *Quality Assurance*

1. INTRODUÇÃO

Neste capítulo introdutório será feita uma pequena descrição de todo o enquadramento deste projeto, incluindo a motivação para o desenvolvimento deste trabalho. Em seguida, são apresentados os principais objetivos que se pretende que sejam alcançados com este estágio e ainda quais os resultados que esperamos atingir. Para terminar, é apresentada a estrutura do documento.

1.1 Enquadramento e Motivação

A indústria de *software* é cada vez mais competitiva, sendo necessário estar em constante evolução e procura de sistemas desenvolvidos no menor tempo possível, com menores custos associados. Durante o desenvolvimento de *software* é necessário passar por várias fases, sendo das mais importantes a testagem, que tem como intuito verificar se todo o desenvolvimento está a funcionar devidamente independentemente do caso de teste utilizado.

No passado, as etapas associadas a testes de *software* eram realizadas manualmente. Recentemente, foram criados mecanismos automáticos que facilitam todo o processo e, apesar da testagem automatizada possuir inconvenientes (Arasteh & Hosseini, 2022) por serem necessários casos de teste que aumentam o software (Fraser & Arcuri, 2011), sabemos que é extremamente vantajoso, devido à melhoria da eficiência e qualidade dos sistemas de *software* por ser conseguida uma melhor resposta das aplicações na interação homem-máquina (Zhong et al., 2023). A primeira etapa de testagem passa pelo indispensável planeamento dos testes, avaliando os requisitos necessários. Seguidamente, dá-se a geração de casos de teste, fundamental para identificar os dados de entrada (Korel, 1990), e são desenvolvidos os ambientes onde a testagem será realizada. Só depois destas fases poderemos passar para a execução do teste, que quanto mais trabalhado e desenvolvido for mais capacidades de encontrar erros, levando a uma constante melhoria do modelo (Cai, 2002).

O projeto de investigação desenvolve-se na empresa Ciberbit, S.A.. Esta empresa surgiu no ramo de desenvolvimento de *software* há mais de 20 anos e ao longo dos mesmos teve a missão de criar e desenvolver produtos, começando por um jogo e estando atualmente com principal foco na saúde e no retalho.

A principal centralização do estágio passa pela automação de testes realizada especificamente para um sistema existente da área da saúde, mais concretamente gestão hospitalar, o I'MTHOM (Total Hospital Management). Esta plataforma faz a gestão de unidades de saúde focando-se nos utentes, mas também

em toda a gestão do negócio, sendo de extrema importância que apresente todas as funcionalidades necessárias para ajudar os hospitais ou as clínicas a maximizar a sua resposta para com os clientes.

1.2 Objetivos

Com o desenvolvimento deste trabalho pretende-se que seja realizado um estudo acerca de testes funcionais do I'MTHOM, com o intuito de agilizar o processo de elaboração e análise de testes de *software*. Então, podemos dizer que o principal objetivo desta dissertação passa pela criação de um conjunto de mecanismos de testagem automatizada de um *software* já existente. Mais especificamente, é pedido que a fase de testes em todo o processo de desenvolvimento do *software* seja realizada de forma automática, tendo os casos de uso como ferramenta a serem gerados. Este *software* é utilizado para a área da saúde, pelo que a testagem realizada tem de ser extremamente fiável e todo o projeto desenvolvido tem de ser extremamente viável.

Para conseguirmos cumprir este objetivo final é necessário passar por outros intermédios que passam por realizar estudos e análises acerca da testagem em *software*, de maneira a percebermos as diferenças entre testagem manual e testagem automatizada. Estudar de que maneira a geração de dados será importante e quais os requisitos necessários para a mesma. Temos de familiarizar-nos com o programa de testagem e perceber se os métodos até agora utilizados são os mais vantajosos. Antes da implementação do algoritmo criado, este terá de passar pela fase de testes para perceber se respeita todas as exigências.

Ao desenvolver este projeto espera-se que o teste criado seja executado de forma automática, de maneira que o utilizador não tenha de o fazer de forma manual. Posto isto, é pretendido que com este projeto haja uma redução do tempo e dos valores até agora gastos com a testagem manual.

De acordo com os objetivos acima mencionados e de maneira a cumpri-los com exatidão foi proposto um conjunto de tarefas a realizar. Em primeiro lugar é necessário identificar qual o problema que estamos a trabalhar e, em seguida, avançamos para uma revisão bibliográfica do mesmo para que consigamos perceber um pouco mais sobre o que vamos trabalhar e que caminhos podemos seguir. Num terceiro momento será feita uma comparação entre a testagem manual e a testagem automatizada de forma a perceber quais as mais-valias de automação de testes, posteriormente é necessário definir a metodologia que vamos utilizar para passar para a aprendizagem do programa de automação a utilizar para, daí, passarmos à parte mais prática deste trabalho que é o desenvolvimento dos testes e a inserção e implementação dos mesmos em todo o projeto já existente dentro da organização. A fase final é perceber, através de uma análise de resultados, se o estudo está a funcionar devidamente.

1.3 Perguntas de Investigação

Neste capítulo vou abordar as principais perguntas de investigação para o estudo descrito no presente documento. Estas questões auxiliam a definição dos principais objetivos.

RQ1: É possível a definição de um processo automático ou semiautomático para realizar o teste do software da empresa?

Esta questão tem como intuito explorar qual a possibilidade de automatizar, quer seja totalmente ou parcialmente, o processo de testes do software da empresa. A pesquisa para responder a esta questão passará por identificar as melhores práticas, ferramentas e até mesmo metodologias que podem ser utilizadas para automatizar todo o processo de teste. Com isto também será permitido verificar quais as limitações que existirão no desenvolvimento do projeto de automação de testes.

RQ2: Qual a qualidade do sistema de testagem quando comparado com os testes criados manualmente na empresa?

Com esta pergunta queremos perceber a distinção existente entre a testagem manual e a testagem automatizada, mesmo que esta não o seja totalmente. Esta pesquisa permite analisar os indicadores de qualidade dos testes como, por exemplo, a sua precisão, os erros encontrados e ainda o tempo e dinheiro gastos ao executar cada tipo de teste. Vamos perceber com o desenvolvimento do projeto quais as principais vantagens e desvantagens dos testes automatizados relativamente aos testes manuais.

1.4 Abordagem de Investigação

Neste projeto estamos perante uma abordagem de investigação-ação, que é nada mais nada menos que a passagem para a prática, em contexto organizacional, de uma pesquisa realizada, permitindo resoluções de problemas específicos. Neste caso em concreto, conseguimos identificar que a necessidade desta empresa passa pela implementação de automação de testes de *software*.

Nesta metodologia, tal como o nome indica passamos pela fase da investigação, que remete à pesquisa de dados, artigos ou outros estudos já realizados, de forma a perceber qual a situação atual dos testes de *software* automáticos. A partir deste estudo conseguimos mais facilmente definir objetivos e critérios para que o projeto seja elaborado com sucesso.

Além disso, é necessário aperfeiçoar o conhecimento acerca das ferramentas que temos à disposição para a elaboração de testes automáticos, permitindo assim uma melhor seleção das tecnologias adequadas.

Partindo de todo o conhecimento adquirido na fase de investigação, passamos para a parte da ação, que consiste na realização de um caso prático de automação de testes de *software*, cumprindo com um desenvolvimento e implementação de um projeto no contexto de empresa. Durante todo o desenvolvimento é necessário avaliar todos os resultados obtidos e realizar uma análise crítica de todo o projeto desenvolvido, assim como uma validação do sucesso no desenvolvimento de um projeto que emergiu com o inicio da automatização de testes. Vamos ter em constante análise se seguimos os caminhos estipulados numa fase inicial ou se ao longo de toda a investigação e com as dificuldades encontradas será necessário seguir outras abordagens.

1.5 Estrutura do documento

A presente dissertação é composta por cinco capítulos. No primeiro capítulo temos uma introdução, onde apresentamos o estudo que está a ser realizado, quais os objetivos do mesmo e ainda fazemos uma pequena apresentação da Ciberbit,S.A., local onde decorreu o estágio. No segundo capítulo é efetuada uma revisão de literatura sobre testagem de *software*, mais especificamente sobre testagem automatizada, ferramentas de automação de testes e ainda comparações entre os dois tipos de testagem. No terceiro capítulo encontramos toda a implementação do projeto, isto é, descrevemos como é efetuado o fluxo de trabalho da empresa até então, são explicados todos os testes que serão realizados no projeto e ainda a implementação dos mesmos em *Selenium IDE* e ainda a passagem para *Selenium WebDriver*. No quarto capítulo apresentamos todos os resultados obtidos na execução dos testes, através de diversos gráficos gerados na *pipeline* de execução. No último capítulo deste documento encontram-se as conclusões a que chegamos e apresentamos perspetivas futuras.

2. ESTADO DA ARTE

O capítulo que agora se inicia tem como função fornecer todos os conteúdos relativos à contextualização teórica dos métodos utilizados em todo o desenvolvimento do projeto.

Para a realização deste capítulo foi necessária uma investigação e análise de livros e artigos das áreas apresentadas. No decorrer deste capítulo serão dados a conhecer melhor não só os conceitos do que é um *software*, o seu desenvolvimento, testagem e a sua automação e como pode acontecer a geração automática de testes. Além disto, ainda apresentamos as ferramentas que foram utilizadas no decorrer do projeto e ainda uma pequena apresentação do *software* que será alvo dos testes.

2.1 *Software* e Desenvolvimento de *Software*

Quando falamos de um *software* estamos a referir-nos a um conjunto de instruções interpretadas por um computador para executar uma tarefa. Podemos dizer que o *software* tem a responsabilidade de compreender e executar os comandos do utilizador.

A área de *software* está em constante crescimento e o foco das empresas de desenvolvimento de *software* é a satisfação máxima do cliente com o produto desenvolvido. Para que possamos dizer que o *software* está a ser desenvolvido corretamente temos de verificar se passa por um conjunto de etapas importantes, sendo estas a análise, o projeto, a codificação, os testes e a implementação (Alves et al., 2016). Para começarmos a fase de análise temos de, num primeiro momento, elaborar a lista com todos os requisitos que o *software* deve apresentar e só depois analisar detalhadamente esses requisitos. É a partir desta etapa que iremos construir modelos representativos do sistema. Depois de realizada esta análise passamos para a fase de projeto, onde definimos alguns parâmetros como a arquitetura do sistema, qual a linguagem de programação que será utilizada, entre outras características que serão essenciais de se definir nesta fase. Temos de ter ainda em consideração que o projeto se encontra, normalmente, dividido em dois momentos distintos. Uma atividade é o projeto de arquitetura que distribui as classes de objetos relacionados do sistema por vários subsistemas e pelos respetivos componentes. A outra atividade é o projeto detalhado que nada mais é do que a modelação das relações de cada módulo, com o intuito de ser desenvolvido o projeto de interface entre o utilizador ou a base de dados.

Passando para a fase de codificação iremos, tal como o nome indica, codificar todo o sistema na linguagem escolhida anteriormente. Nesta fase torna-se possível gerar o código executável para o desenvolvimento do *software*. Em seguida temos a fase onde testamos todo o sistema desenvolvido com o intuito de validar não só o produto como todas as suas funcionalidades. Por fim, passamos para a fase

de implementação que é, nada mais nada menos que a instalação do *software* no ambiente final. Nesta última fase realizam-se diversas tarefas desde a importação de dados até à formação dos utilizadores para que realizem um uso correto do sistema desenvolvido.

Figura 1- Etapas de desenvolvimento de software

2.2 Testes de *Software*

Como mencionado no tópico anterior, durante o desenvolvimento de um *software* este passa pela fase de testagem. Atualmente esta é considerada uma das fases importantes e tem vindo a ganhar notoriedade, uma vez que garante o bom funcionamento e qualidade do sistema. Quando uma empresa aposta na testagem de *software* está imediatamente a investir na prevenção de falhas (Claudio & Neto, 2007) e defeitos do sistema desenvolvido, uma vez que está a reduzir incertezas e, ao mesmo tempo, custos de manutenção.

A testagem deve ser realizada sempre que se termina uma fase do processo de produção, garantindo que as falhas são detetadas o mais cedo possível e que estamos perante um *software* com elevada qualidade. Para que um teste seja bem executado é necessário que passe por vários níveis, sendo que em cada nível os objetivos a serem avaliados são distintos relativamente ao tipo de falhas que queremos que nos apresente. O processo de testagem encontra-se dividido em quatro fases distintas (Curti & Dallilo, 2022).

Numa primeira fase encontramos os testes de unidade que pretendem detetar erros de lógica e implementação mais simples, sendo que cada unidade do sistema é testada isoladamente. Imediatamente a seguir a esta fase temos os testes de integração que têm como principal objetivo detetar falhas derivadas da integração interna de componentes do sistema. Nesta fase os módulos já são combinados e os testes são realizados em conjunto, resultando num sistema integrado e preparado para a próxima fase. Esta fase seguinte diz respeito aos testes de sistema, que identificam as falhas existentes nos requisitos do mesmo, estando este completamente integrado. Por fim, temos os testes de aceitação, que são considerados uma extensão dos anteriores e o principal objetivo destes é verificar se o *software* está pronto e pode ser utilizado pelo cliente.

Antes de fazer qualquer teste temos de perceber quais os tipos de teste de *software* que são necessários utilizar em cada uma das situações. Para isso temos de entender aquilo que é analisado em cada um dos testes, tendo em evidência que estes são classificados por tipo e de acordo com seu objetivo especifico.

Começamos pela técnica estrutural, mais comummente conhecido por teste de caixa-branca, que avalia todo o comportamento interno dos componentes. É evidenciada por trabalhar diretamente com o código fonte do componente de *software* e por avaliar as condições, o fluxo de dados, ciclo de teste de caminhos lógicos, entre outros (Ricca & Tonella, 2002).

Passamos agora para a outra técnica conhecida por teste funcional ou, mais vulgarmente, teste de caixa preta. Nesta técnica não olhamos para o comportamento interno de qualquer um dos níveis de teste, mas testamos se o que acontece no programa no decorrer do teste corresponde ao resultado esperado.

Em seguida temos o teste não funcional, que tem como principal objetivo verificar se existem elementos do *software* que não se relacionam com nenhuma função.

Finalmente, temos os testes de regressão que são a repetição constante de testes para verificar se a aplicação que está a ser testada sofre alterações que sejam negativas para a aplicação e se necessita que sejam realizadas correções.

Ao falarmos de testes de *software* temos também de falar de casos de teste. Os casos de teste são constituídos por vários elementos desde as entradas, restrições e resultados que serão obtidos com o teste com o objetivo de verificar se o comportamento esperado corresponde ao que deverá ser testado (Ricca & Tonella, 2002).

2.3 Testagem Manual e Automatizada

Até há bem pouco tempo as etapas de teste de *software* eram realizadas manualmente, o que despendia demasiado tempo e dinheiro às empresas. A testagem manual é dispendiosa porque é necessário definir os parâmetros de teste, levando a que a interação humana seja uma constante. Ao longo do tempo foram-se criando métodos de testagem automatizada com o intuito de agilizar o processo e de forma a utilizar os *scripts* para descobrir o bom desempenho dos mesmos passando, deste modo, a ser desnecessário uma pessoa responsável pela testagem. Além disto, os testes automatizados criam ferramentas mais eficazes e capazes de garantir sucesso de forma mais completa (Hellerstein et al., 2010).

Posto isto, conseguimos dizer que a testagem manual apresenta algumas particularidades como requerer maior esforço tanto a nível de criação como manutenção, a reutilização dos testes tem um nível baixo,

está dependente de uma linguagem muito ambígua, a sua execução é mais demorada e exige que os profissionais tenham experiência em testes. Por outro lado, podemos dizer que os testes automatizados, que também requerem grande esforço de criação e de manutenção, exigem uma maior programação nas ações realizadas, não sabem lidar com ações inesperadas e, com isto, não são capazes de lidar com situações diferentes, apesar de terem uma maior rapidez de execução, tendo capacidade de repetição, sendo que são suscetíveis a mudanças no ambiente de teste e são testes que necessitam de profissionais mais capacitados.

De forma sucinta podemos dizer que o teste manual está sujeito a erro humano, tem a vantagem de possuir uma análise humana e os testes mais utilizados nesta situação são os testes de usabilidade, já os testes automatizados como são aplicados através de ferramentas, tendo em conta cada cenário em estudo, têm resultados mais assertivos e são uma melhor opção para testes repetitivos e de longa duração. O principal objetivo da automação de testes passa pela redução de trabalho humano em atividades manuais (Jacobson et al., 1999).

2.4 Automação de Testes

Tal como vimos anteriormente, o teste de *software* pode ser trabalhoso e custoso e, por isso, o grande objetivo é automatizar o máximo possível os testes. Podemos então dizer que a automação de testes chegou para responder às necessidades das empresas de reduzir os custos e o tempo utilizados durante esta fase do desenvolvimento de *software*. Além disso permite reduzir o erro humano, tornando o teste de regressão mais fácil e ainda facilita as tarefas mais pesadas e trabalhosas, realizando o teste de vários parâmetros não funcionais (Jacobson et al., 1999).

Outra das características que encontramos na automação de testes é que estes conseguem realizar as mesmas operações mais do que uma vez, conforme forem executados, e conseguimos que o mesmo teste seja repetido várias vezes, o que fará com que sejam realizados muitos testes ao mesmo tempo e, consequentemente, ocorre uma diminuição do tempo de ciclo (Chicanelli et al., 2019).

Apesar do aparecimento da automação de testes ser extremamente vantajoso, ainda não é possível realizar todos os testes neste método, o que faz com que seja necessário que alguns continuem a ser realizados de forma manual. Ainda assim, a automação de testes está ligada a um conjunto de benefícios, sendo alguns deles: a diminuição de bugs que eram enviados para a produção, uma vez que agora a sua correção é antecipada; redução dos custos dos problemas; e a disponibilidade de produtos com mais qualidade para os clientes.

É necessário termos uma noção das vantagens e desvantagens que a automação de testes possui para percebermos se a implementação de automação se sobrepõe à hipótese de continuação de testagem manual. Sabemos que as vantagens mais relevantes da automação de testes passam pelo facto de existirem e conseguirem ser executados mais testes a cada nova versão de *software* que seja lançada, haver possibilidade de criação de testes que manualmente não eram possíveis e ainda a criação de testes com maior complexidade.

Ao automatizar os testes estamos a reduzir os erros humanos, elevando a eficácia e a eficiência dos recursos de teste, uma vez que os testes passam a ser mais assertivos e otimizados, o que permite libertar mais tempo para possíveis testes manuais que tenhamos de fazer. Com isto é possível ter um *feedback* mais rápido e organizado sobre a qualidade do *software*, permitindo uma maior confiabilidade do sistema, devido à repetição exaustiva dos vários cenários de teste criados.

Apesar de nos apresentar todas as vantagens acima descritas temos, em contrapartida, algumas desvantagens que foram aparecendo com a automação de testes.

Uma empresa para ingressar na automação de testes terá de adquirir as ferramentas necessárias para a realização deste trabalho e ainda a contratação de profissionais da área, o que leva a alguns custos. Até que todo o ambiente de automação de testes esteja apto a funcionar leva o seu tempo e esforço. Outros problemas que podem acontecer são: a complexidade dos testes devido aos requisitos e regras de automação; ocorrência de erros que serão do teste automatizado e não do recurso que está a ser testado; necessidade de constante manutenção dos testes e do ambiente de testes; e, tal como já mencionado anteriormente, nem todos os testes manuais podem ser automatizados.

2.5 Geração Automática de Casos de Teste

Quando nos referimos à geração automática de casos de testes estamos a falar de, tal como o nome indica, gerar automaticamente conjuntos de testes escritos na linguagem de programação que o utilizador entender que seja a mais apropriada consoante a ferramenta de geração que irá utilizar.

Os casos de teste gerados automaticamente podem ser utilizados para encontrar violações da especificação (Fraser & Arcuri, 2013).

Conjuntos de testes relacionados com a geração automática de casos de teste podem ter um alcance de cobertura estrutural maior do que aqueles que são criados por *testers* manuais. Um dos principais motivos para isto acontecer é que a geração automática de testes pode ser realizada de diferentes formas. Vamos apresentar algumas das formas mais utilizadas para a geração automática de testes e começamos pela geração baseada em programa. Aqui é necessário termos presente o analisador do

programa, qual o caminho que se irá seguir e ainda o gerador dos dados de teste, sendo este último elemento quem recebe o caminho a ser testado através de código.

Temos ainda a geração baseada em interface gráfica (GUI). Quando falamos em interface gráfica estamos a referir-nos aos elementos que permitem uma interação do utilizador com o programa desenvolvido de forma intuitiva. Os testes baseados em GUI têm como objetivo identificar as métricas mais usadas, de modo a conseguir formular uma métrica de cobertura para os testes (Coppola & Alégroth, 2022). Além disso, este método de geração de casos de teste permite uma cobertura bem mais abrangente, uma vez que é realizada uma simulação mais realista do funcionamento do programa para o utilizador, conseguimos com maior rapidez a elaboração de maior quantidade de testes, uma vez que conseguimos reutilizar *scripts* para diferentes cenários. Este método acaba por ser um complemento de outras técnicas de teste e tem o intuito de garantir toda a segurança, rigor e utilidade de qualquer *software* desenvolvido (Polepalle et al., 2022).

2.5.1 Monkey Test

Monkey Test, como é conhecido, é um teste realizado através de entradas de dados aleatórias com o intuito de ver até que ponto o sistema funciona e quando é que ocorre quebra do sistema por comportamentos inesperados. O responsável pela realização destes testes tem de pensar do ponto de vista dos utilizadores do programa desenvolvido, de forma a que sejam evitados o maior número de falhas possível.

Este tipo de testes não segue um padrão nem casos de teste específicos, o que significa que não temos nada que garanta uma correta identificação de todas as falhas existentes no sistema e que possui algumas limitações no que diz respeito à cobertura de teste. Por esta razão é que estes testes só são usados quando existe falta de tempo ou de recursos

Os *monkey test* podem ser divididos em duas categorias distintas (Ali et al., 2022). Uma primeira categoria, a *Dumb Monkey,* onde não é necessário conhecimento sobre a aplicação em que se está a trabalhar nem dos casos de teste gerados. Nesta categoria podemos não encontrar muitos *bugs*, mas pode encontrar *bugs* importantes que os testes direcionados não consigam alcançar. A outra categoria corresponde a teste *Smart Monkey* e é necessário ter-se uma breve ideia de como funciona toda a aplicação.

2.5.2 Web Based Test

Os testes baseados na *web*, tal como o nome indica, realizam o processo de testagem de aplicações que têm o seu acesso e execução através de um navegador *web*. O principal objetivo deste tipo de testes é

verificar e garantir que toda a aplicação funcione devidamente em todos os navegadores e seja de fácil acesso e manuseamento para o utilizador. Além disto são avaliadas outras áreas fundamentais para que o *site* seja o mais universal possível, isto é, testa se a aplicação é acessível para pessoas que careçam de algum tipo de deficiência, que é traduzido para várias linguagens, de modo a permitir que seja acedido em qualquer parte do mundo e verificar se a aplicação tem capacidade para lidar com as falhas e de que maneira ultrapassa essas falhas (Dhir & Kumar, 2019).

2.6 Cobertura de Testes

Cobertura de testes é o nome que utilizamos para determinada métrica de *software* que tem como intuito medir a qualidade dos testes realizados. Mais concretamente, quando estamos perante um teste ou conjunto de testes com vários ramos podemos, por vezes, não executar todos os ramos das instruções condicionais e portanto, não testamos a totalidade do código (Machado & Tavares, 2022). Com a cobertura de testes é possível garantir uma melhor qualidade dos testes, identificar quais as partes do código que foram abrangidas pelo teste e ainda quais os caminhos que o teste não teve capacidade de testar. Assim, podemos dizer que com a análise da cobertura de testes conseguimos prevenir as falhas existentes no nosso sistema numa fase ainda muito inicial.

Alguns casos que nos mostram a importância da cobertura de testes são, por exemplo, na existência de testes, mas que sejam repetidos e cuja única função que irão ter é a de consumir recursos e não irão ajudar a ter garantias de correção do programa.

Existem ferramentas adequadas que nos permitem saber a cobertura de testes funcionais, sendo que no nosso projeto acabamos por não implementar nenhuma dessas ferramentas, uma vez que não tínhamos acesso ao código fonte do nosso sistema, o que impossibilita a apresentação de resultados. Apesar de não termos resultados quanto a esse aspeto é possível identificar situações nos nossos testes onde a cobertura de testes não é de 100%. Na Figura 2 temos um conjunto de comandos do *Selenium IDE*, que correspondem a um teste de acesso à aplicação. Podemos ver que o teste faz o preenchimento dos campos de *username* e *password* e depois inicia um *If*, que verifica se existe a presença de um aviso de que não foi possível entrar na página e ainda um outro ciclo *if* para o caso de o aviso não estar presente e, portanto, o teste continua para abertura de um outro menu. Este é um caso em que a cobertura não é de 100% e que se trata de um teste redundante num dos ramos e o outro ramo não é sequer coberto.

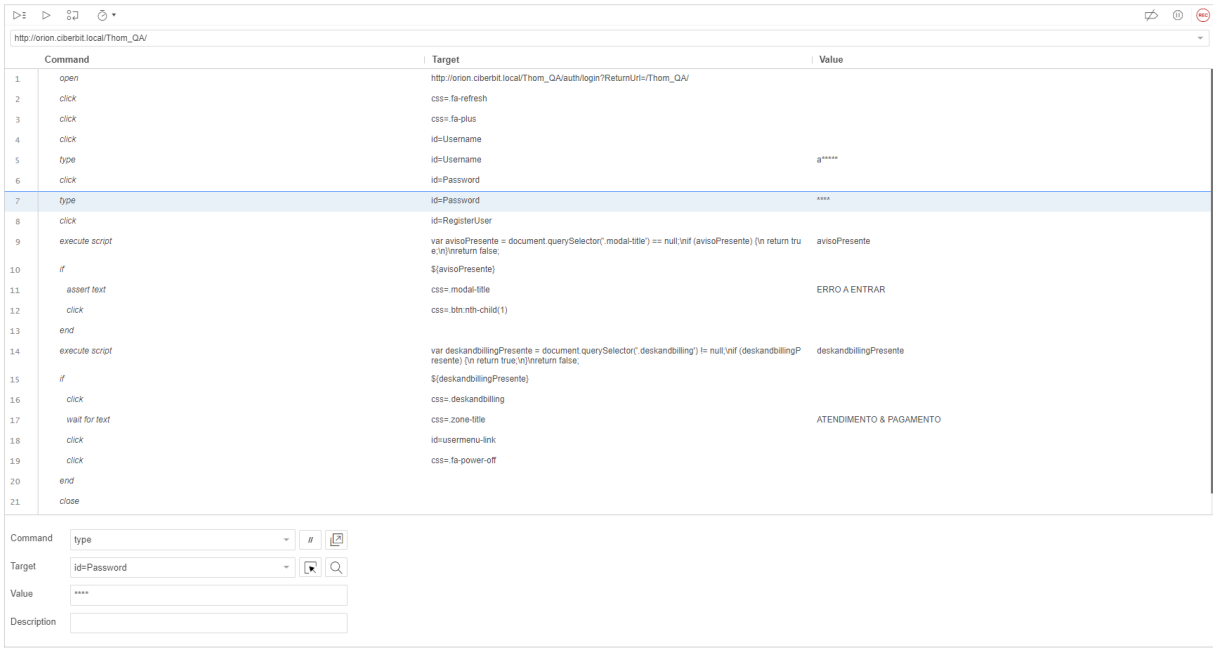


Figura 2- Teste de acesso à aplicação com If

Podemos ainda ter situações onde a cobertura não é suficiente, uma vez que esta pode estar a 100% e ainda assim o programa ter erros. Para percebermos melhor o que é isto de métricas de cobertura do código, estudamos o método M, que permite medir quais as extensões do código fonte são testadas durante a execução dos testes de *software* (Delamaro & Marcelo Rizzo Vincenzi, 2000).

2.7 Testes de Mutação

Os testes baseados em mutações são um tipo de testes caracterizados pela injeção de falhas no *software*, de maneira a avaliar se os testes criados conseguem captar a falha injetada. Quando acontece de o teste não detetar a falha inserida podemos concluir que o teste desenvolvido não é bem-sucedido e não apresenta grande qualidade, mesmo que a sua cobertura seja de 100%.

Realizar a mutação de testes consiste em realizar pequenas modificações no código, às quais chamamos de mutantes. Caso o teste esteja bem elaborado irá perceber a existência desta modificação e rejeitá-la, fazendo com que o teste falhe (Machado & Tavares, 2022).

No nosso desenvolvimento não foi possível elaborar este campo, mais concretamente realizar testes baseados em mutações, uma vez que, tal como vimos na Secção 2.6, não tínhamos acesso ao código-fonte do sistema, nem a qualquer tipo de base de dados.

2.8 Ferramentas de Automação de Testes

Existem várias ferramentas criadas com o intuito de realizar testagem automaticamente. Estas ferramentas simulam utilizadores ou até mesmo atividades humanas dispensando a ação humana (Bartié, 2002). Iremos mostrar com mais detalhe as ferramentas que foram estudadas e analisadas no decorrer da dissertação, com o intuito de perceber qual o melhor para o nosso caso em particular, tendo em consideração o que é mais vantajoso para a empresa.

O estudo de apenas duas ferramentas para uso derivou de um conjunto de discussões com a empresa, de acordo com os conhecimentos que já possuíam acerca do tema e dos critérios que já tinham pré-definidos para a elaboração deste projeto.

2.8.1 Selenium

A primeira ferramenta que será apresentada é o *Selenium*. Esta ferramenta de código aberto (Holmes & Kellogg, 2006) foi criada pelo testador *Jason Higging* no ano de 2004 e realiza testes automatizados em diversas plataformas através de *scripts* simples, que permitem executar testes diretamente no navegador. O *Selenium* testa interações que existam entre a aplicação *web* e o navegador através da simulação de ações. Os *scripts* de teste do *Selenium* são programas de computador que podem ser escritos em diversas linguagens de programação e executados em vários navegadores. Quando falamos no *Selenium* estamos a falar de um programa que possui uma arquitetura em camadas no JSON, sendo assim possível termos soluções de teste sem ser necessário aprender aprofundadamente a linguagem de *SCRIPT*.

É importante perceber que o *Selenium* é constituído por várias componentes, tendo cada uma as suas particularidades.

Começamos pelo *Selenium* IDE, que nada mais é que uma extensão dos navegadores que tem como função gravar as interações do utilizador e, em seguida, reproduzi-las. Esta ferramenta permite que se criem *scripts* dos testes gerados com a gravação. De realçar que o *Selenium* IDE é um pouco limitado e permite a elaboração de testes mais simples. Passamos agora para o *Selenium* RC (*Remote Control*) que permite aos utilizadores que a partir de linguagem de programação criem testes de interface do utilizador mais complexos. Outra componente é o *Selenium Grid*, que é uma ferramenta que facilita a execução de mais do que um teste em paralelo e em mais do que uma máquina, uma vez que a execução dos testes aqui é distribuída em vários ambientes e navegadores. Por fim, temos o *Selenium Webdriver*, que é uma API que realiza uma interação diretamente com o navegador, tendo como função criar um servidor

local que comunica com o mesmo de maneira a simular uma interação que aconteceria entre o utilizador e a aplicação (Greater Noida et al., 2015).

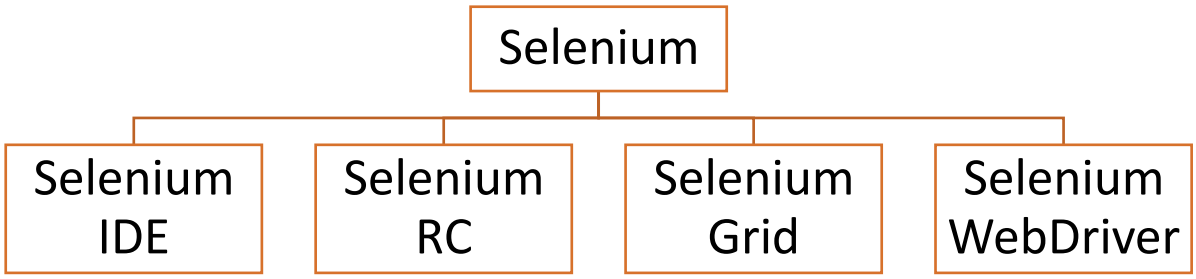


Figura 3- Componentes do Selenium

De forma a termos uma melhor perceção de como funciona a *Selenium IDE* e ver mais intuitivamente um exemplo prático de um teste elaborado com esta ferramenta. Na Figura 4 temos o ecrã de acesso à aplicação com os campos já preenchidos, é exatamente a abertura e preenchimento destes campos que o teste vai realizar.

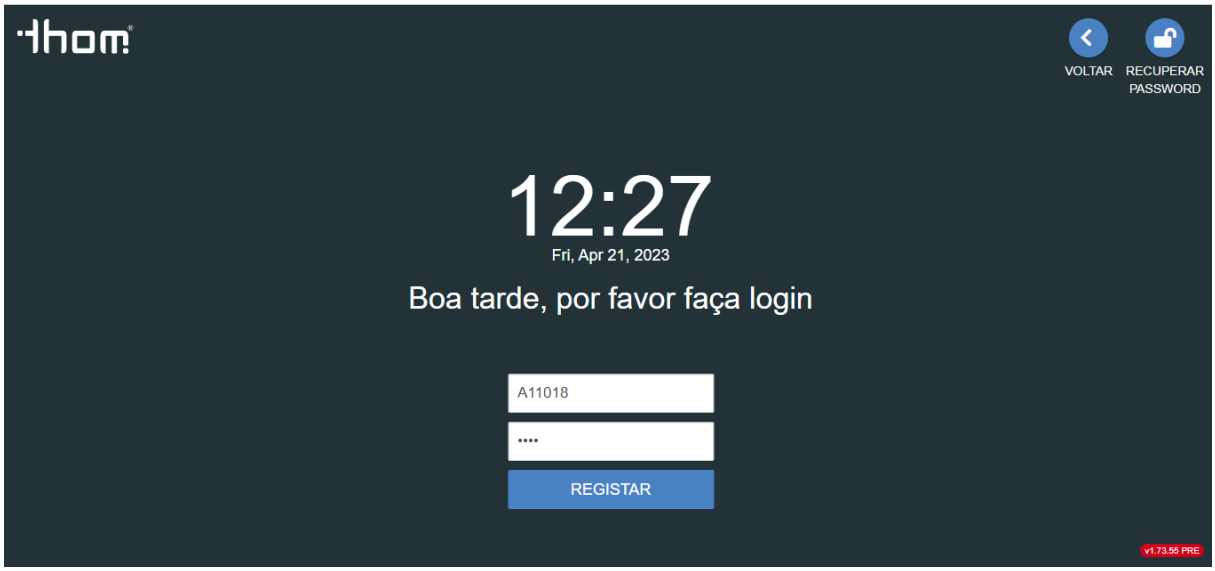


Figura 4- Ecrã de Login da aplicação

Na Figura 4 conseguimos ver ainda que nesta página é-nos dada a data e hora do momento exato em que nos encontramos e uma mensagem para realizarmos o *login* e acedermos à aplicação. Para esta execução temos presente o campo do utilizador e o campo da *password*.

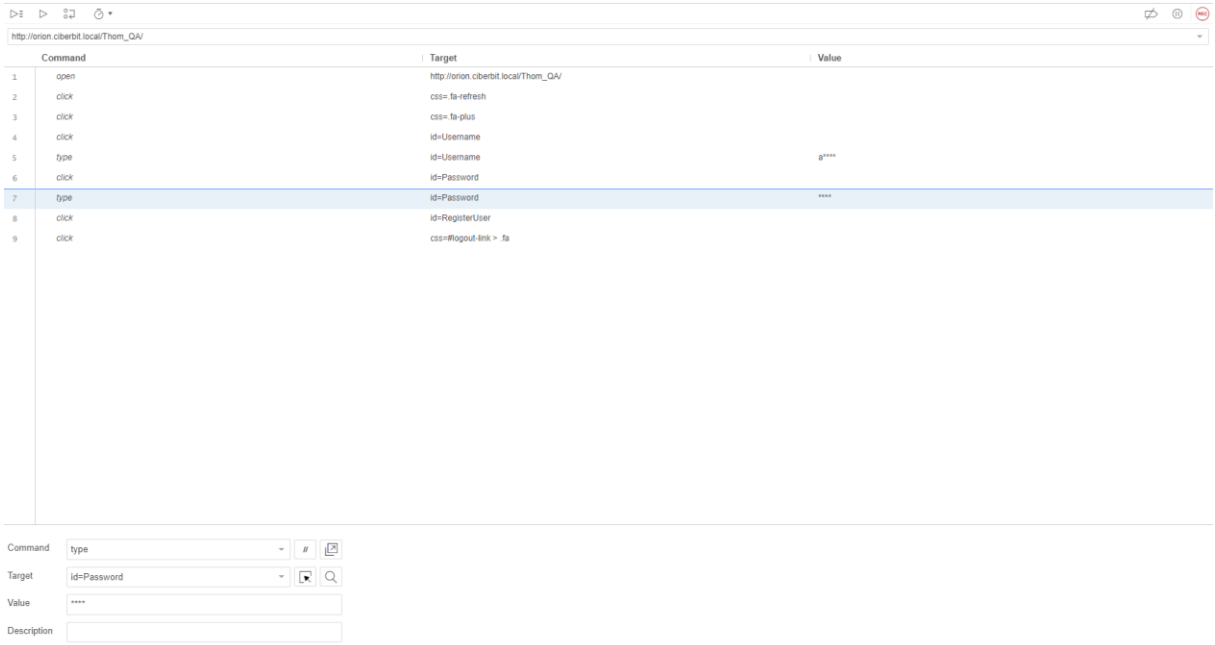


Figura 5- Teste de Login no Selenium IDE

Na Figura 5 temos presente o ecrã da ferramenta *Selenium* IDE com a elaboração do teste de *Login* do ecrã anterior. Os passos que foram executados para a criação deste teste passaram por abrir a ferramenta, clicar no símbolo "+" que nos leva para a criação de um novo teste e a partir daí começa a elaboração do teste em si. Para esta fase clicamos no símbolo da gravação, que nos abre o navegador. De seguida escrevemos o *url* do site que queremos aceder e a partir daí executamos o *login* na aplicação. Quando terminada a gravação é gerado na ferramenta o teste com os comandos realizados. Podemos ver que ele abre a página através do *link*, depois clica no campo do utilizador e preenche-o com um utilizador válido. O passo seguinte é realizar exatamente estas etapas, mas para o campo da *password*. Depois de preenchidos os dois campos ele irá clicar no botão "Registar", entrando assim na aplicação. Sempre que corrermos este teste na ferramenta irá abrir uma página do navegador e executar todos estes passos.

Podemos realçar que para cada linha do teste temos de preencher três componentes. Em primeiro lugar é necessário inserir o comando, que nos indica qual a ação a realizar. Em seguida, temos o alvo que corresponde ao elemento da página *web* que queremos interagir e, por último, o valor que esse mesmo elemento irá tomar.

É necessário ainda realçar uma particularidade do *Selenium IDE*, que será muito utilizada neste projeto: a possibilidade de extrair o teste criado para diversas linguagens (desde C#, *Java*, *Python*, entre outras).

2.8.2 Playwright

Passamos agora para a segunda ferramenta de automação de testes utilizada no desenvolvimento deste projeto. O *Playwright* é uma ferramenta criada em 2020 pela *Microsoft*, que permite a realização de testes E2E (*end-to-end*) confiáveis do nosso sistema. Esta ferramenta necessita de maior conhecimento de código para o desenvolvimento dos testes e oferece suporte a qualquer mecanismo de renderização. O *Playwright* destaca-se pelo facto de ter recursos mais avançados. Tal como no *Selenium*, o *Playwright* possui compatibilidade com vários navegadores e ainda permite a programação em várias linguagens, tendo uma arquitetura orientada a eventos.

Outras características extremamente importantes que temos a ter em conta acerca do *Playwright* é que ele possui espera automática, ou seja, espera que os elementos estejam em condições de avançar antes de executar as ações correspondentes. Se criarmos diferentes cenários em determinado contexto, o *Playwright* irá executar tudo num único teste.

Esta ferramenta acaba por, de um modo geral, tornar a automação dos testes mais acessível, rápida e confiável.

Na Figura 6 encontramos uma comparação do *Playwright* e do *Selenium* em relação a vários parâmetros.

	Playwright	Selenium
Browser support	Chromium, Firefox, and WebKit	Firefox, Edge Chromium (Selenium 4), Safari, Opera, Google Chrome, and more
Operating systems	Windows, Mac OS, and Linux	Windows, Mac OS, Linux, and Solaris
Languages supported	TypeScript, JavaScript, Python, .NET, Java	Java, Python, Ruby, C#, and JavaScript (and more with language binding)
Prerequisites & installation	Needs NodeJS to be installed, but otherwise, a straightforward process	Selenium Bindings (for your language), Browser Drivers, and Selenium Standalone Server needed
Real devices	Emulation (experimental support for real devices also available)	Offers real device support through clouds and remote servers
Community	Small but active	Big and active
Developer experience	Very good	Fair
Speed	Fast	Slower
Architecture	Event-driven architecture	Layered architecture relying on the JSON Wire Protocol

Figura 6- Comparação enntre Playwright e Selenium

2.9 I'MTHOM

O *software* que será alvo de testes no decorrer desta dissertação é o I'MTHOM (*Total Hospital Management*). Esta aplicação, tal como mencionado anteriormente, foi desenvolvida pela empresa Ciberbit S.A. e concentra-se na área da saúde. O principal objetivo da sua criação passa por gerenciar as operações que necessitam de ser efetuadas em determinada unidade de saúde. Estas operações são gerenciadas através de uma estratégia de cima para baixo, isto é, o sistema é constituído por vários subsistemas, com componentes integrados entre si e que irão exercer funções específicas.

Esta plataforma única, que otimiza todos os resultados, também consegue centralizar qualquer aspeto que esteja relacionado a qualquer fluxo de trabalho desde clínico, a administrativo ou financeiro com apenas uma exibição em tempo real.

Quando se desenvolve uma aplicação desta natureza tem de pensar-se sempre no paciente e na sua experiência de utilização do mesmo. Posto isto, é necessário estar ciente de que o paciente deve ser capaz de assumir o controlo das mais variadas ações, desde o agendamento de consultas até à documentação que será utlizada na mesma e informações de todo o processo. Neste processo temos ainda outros intervenientes que devem ver a interação com a aplicação facilitada: os médicos e enfermeiros. Isto porque só se as informações estiverem acessíveis e organizadas para os profissionais de saúde é que estes conseguirão utilizar corretamente a aplicação e inserir todas as informações necessárias para acesso do paciente.

Podemos então dizer que o I'M THOM é uma ferramenta que fornece tudo o que é necessário para que pacientes e profissionais de saúde estejam conectados, de maneira que permite que o paciente oriente as suas questões de saúde e ajude os profissionais a tratá-las consoante as necessidades.

Além disso, esta ferramenta também possibilita uma melhoria da comunicação entre os profissionais do hospital, possui um sistema de transmissão de informação sobre os doentes e ainda sobre a gestão e controlo de stocks de cada serviço dentro de cada unidade hospitalar e entre cada uma das unidades.

Para uma melhor compreensão do programa em estudo apresentamos um conjunto de ecrãs presentes no mesmo, para demonstrar algumas das funcionalidades que possui.

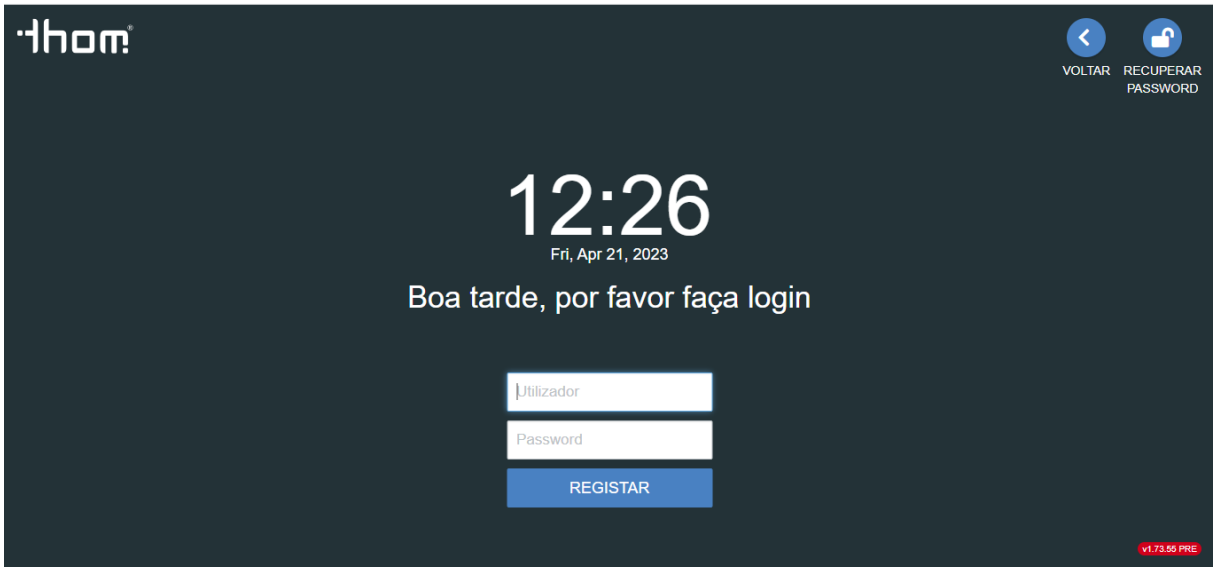


Figura 7- Ecrã de Login

Na Figura 7 é-nos apresentado o ecrã de *login* da aplicação, que já explicamos detalhadamente acima. Podemos ver todo o desenvolvimento deste teste no exemplo dado na subsecção 2.8.1.

Figura 8- Ecrã relativo à visita de determinado paciente à Urgência

Na Figura 8 conseguimos ver um *dashboard* clínico da visita de um cliente. Neste caso em concreto estamos perante um paciente que veio a uma urgência. Além do seu nome e de todos os seus dados pessoais, no *dashboard* conseguimos ainda ver o histórico do cliente, quais as suas visitas anteriores e as próximas visitas, se tiver algum agendamento marcado. Conseguimos ainda ter acesso a documentos do paciente como, por exemplo, relatórios de exames anteriormente realizados, quais os procedimentos que já passou, a sua medicação habitual, entre outras coisas.

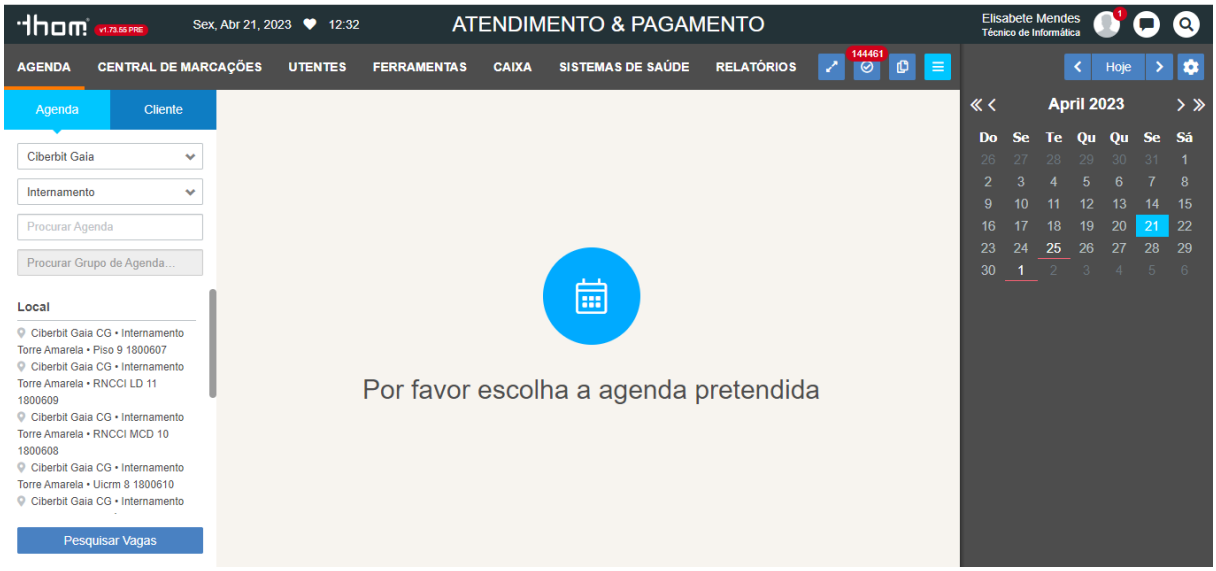


Figura 9- Ecrã da Agenda

Com a imagem da Figura 9 conseguimos verificar como é a disposição do ecrã de agenda para um paciente. Do lado direito temos um calendário disposto mensalmente, que irá mostrar, através de cores, qual a disponibilidade de marcação de algum serviço em determinado dia. Do lado esquerdo temos os

parâmetros que nos permitem realizar uma marcação, que pode ser filtrada ou pela Agenda e é possível selecionar o local onde queremos realizar a marcação ou pelo Cliente. Temos ainda o botão “Pesquisar Vagas” que nos abrirá o ecrã da central de marcações, onde podemos fazer uma seleção mais específica da marcação que pretendemos fazer.

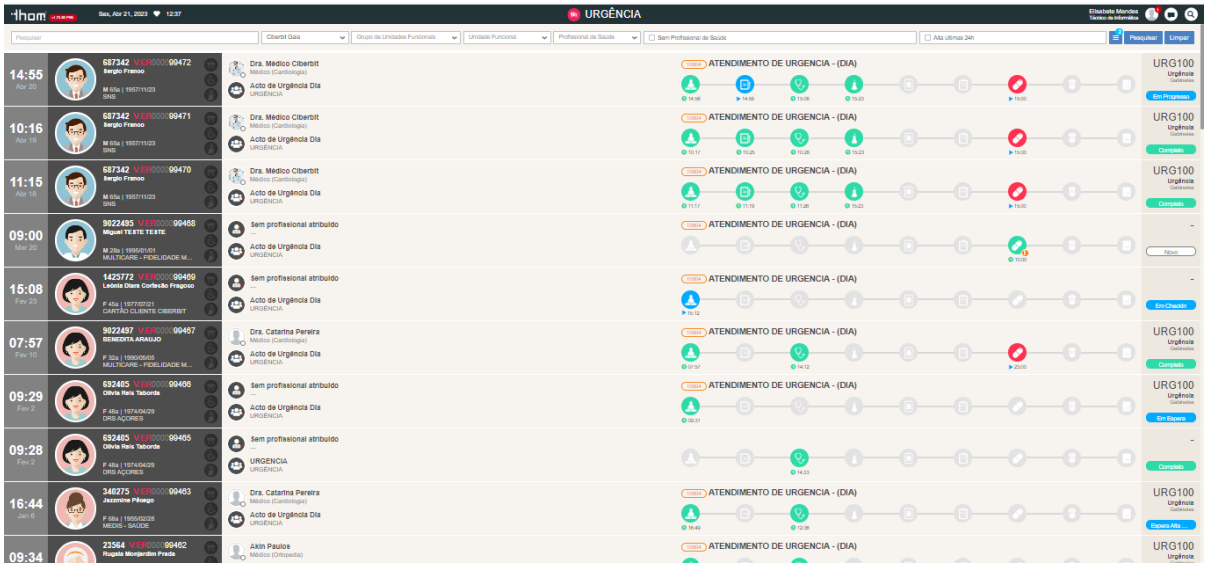


Figura 10- Ecrã de Urgência

Por fim, na Figura 10 encontramos o ecrã de Urgência. Neste ecrã conseguimos visualizar a lista de pacientes que se encontram nas urgências e ainda qual o estado da sua visita, isto é, cada bolinha mostra a etapa em que o paciente se encontra e a cor da mesma corresponde ao estado dessa fase de atendimento. Por exemplo, as bolas sinalizadas a vermelho significam que os pacientes estão em espera para serem medicados e a bolinha azul na primeira etapa do circuito significa que o paciente está a ser atendido pela administrativa neste momento para realizar a admissão.

3. ANÁLISES E DECISÕES DE DESENVOLVIMENTO

Neste capítulo é apresentado de forma mais descritiva todo o processo de funcionamento da empresa até chegar à fase de testagem. Em seguida, é apresentado todo o processo de seleção da ferramenta de testagem e, por último, toda a implementação da automação de testes.

3.1 Ciberbit, S.A.

Na Ciberbit,S.A, o método de funcionamento segue uma metodologia *Agile*, através do módulo *Kanban*. Trabalhamos a partir de *sprints* de 15 dias com tarefas pré-definidas, com gestão de tempos e de tarefas feitas a partir da ferramenta *Jira*.

A empresa é constituída por várias equipas de trabalho, como podemos ver no esquema da Figura 11.

Figura 11- Equipas que constituem a empresa

O fluxo de trabalho da empresa inicia-se com a equipa de produto a definir a lista de prioridades das tarefas a executar, onde são atribuídas sempre tarefas relativas à melhoria e outras relativas a erros. Todas as tarefas são desde início especificadas por esta equipa e, quando necessário, têm o apoio da equipa de *design*, que elabora *mockups* de forma a facilitar o trabalho tanto da equipa de desenvolvimento como da equipa de qualidade.

Quando se inicia cada *sprint* existe uma reunião de cada *team leader* com os *developers* a seu cargo, onde são planeadas todas as tarefas a executar durante essa *sprint*. Nesta reunião os *developers* têm a oportunidade de esclarecer dúvidas que possam surgir depois de lerem a especificação de cada tarefa e também podem discutir quais os melhores procedimentos a adotar, de modo a uniformizar e otimizar a

execução da tarefa. Nesta reunião são ainda atribuídos os tempos estimados a cada tarefa, com o objetivo de serem o mais realistas possível.

Durante o restante tempo de *sprint* os *developers* vão realizando as tarefas em ambiente *Dev* e entregando as mesmas à medida que as concluem, sendo entregues ao *team leader* que faz *code review*, que é nada mais nada menos do que uma revisão do que foi feito, e só depois de aceitarem o trabalho realizado é que estas podem passar para *deploy* do ambiente CI (*Continuous Integration*), que é o ambiente da chamada "primeira camada de testes".

Após *deploy*, as tarefas caem numa pilha de testes do *Jira*, que é da responsabilidade da equipa de qualidade. Cada tarefa que passa para esta fase é validada funcionalmente na aplicação de modo a verificar as alterações especificadas na tarefa. Além disso, também se verifica a funcionalidade num todo e são feitos testes de circuito de modo a garantir que a alteração pretendida não provoca danos colaterais indesejados na restante aplicação. Se durante algum teste forem detetados erros, os mesmos são reportados novamente, para que o responsável da tarefa o possa corrigir. Estas devoluções de erros são inseridas numa lista de trabalhos prioritária, onde o *developer* deve imediatamente corrigir os erros detetados e enviar a tarefa para um novo *code review* e posterior *deploy* em ambiente CI. Este ciclo repete-se tantas vezes quantos os erros que forem detetados.

Assim que todos os testes sejam validados a tarefa é colocada no estado concluído com uma versão alocada e ainda com uma descrição resumida das alterações associadas e de todos os ecrãs que tenham impacto. Assim que uma versão é validada no global, ou seja, que todas as tarefas incluídas são validadas pela equipa de qualidade com sucesso, é feito pela mesma um documento com todas as descrições resumidas e o mesmo é entregue aos clientes atuais, de modo a que tenham conhecimento de todas as melhorias e correções que a versão contém.

Quando a versão fica terminada no ambiente CI, é instalada em ambiente QA e é feita uma apresentação oficial a todos os clientes, com as melhorias de maior impacto para que seja mais claro o objetivo e funcionamento destas alterações.

3.2 Metodologia de Implementação da Automação de Testes

A proposta da empresa passava por criar um projeto de raiz que permitisse a automatização dos testes de *software* que, tal como vimos, eram realizados exclusivamente de forma manual aquando o término de um desenvolvimento. No caso dos testes automatizados, estes serão executados numa *pipeline* diariamente, para permitir que os erros sejam alcançados mais rapidamente.

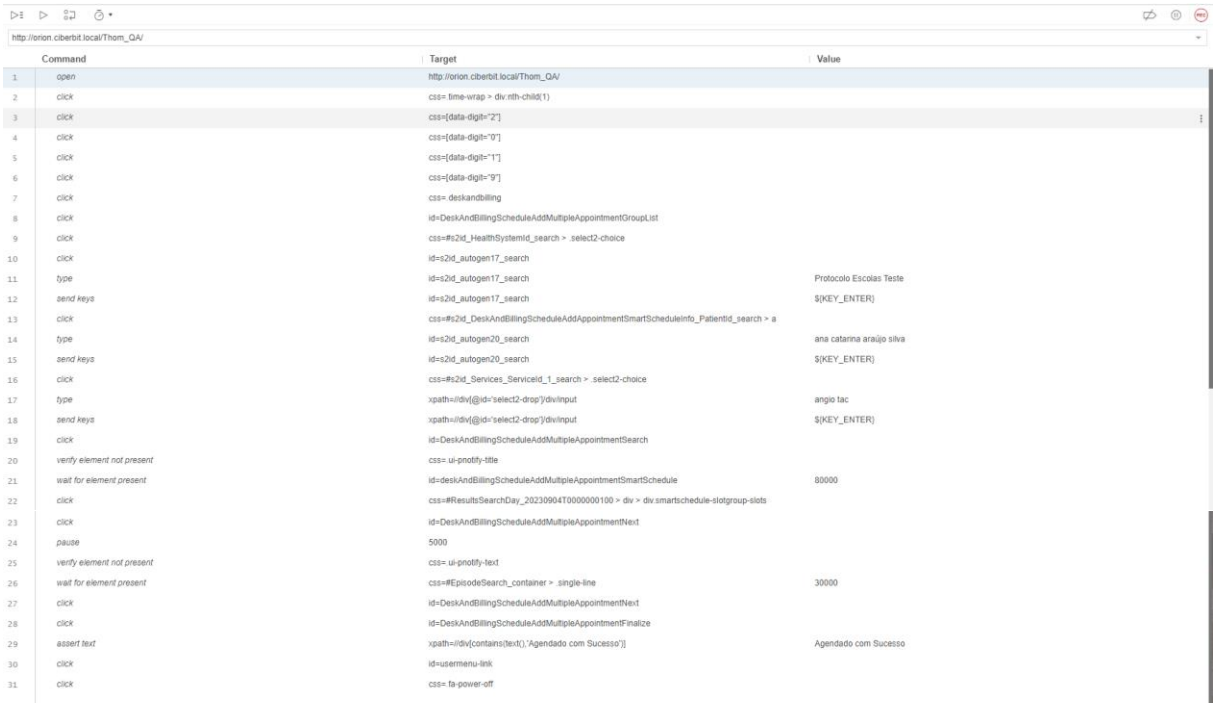
3.2.1 Seleção da Ferramenta de Testes

Antes de se iniciar qualquer projeto é necessário estudar um pouco mais sobre o tema e perceber quais são as várias abordagens que podemos seguir para conseguir avançar com o mesmo. Nesta situação seguiu-se o mesmo pensamento e um dos aspetos principais a estudar passou pelas ferramentas que eram possíveis utilizar para a criação de testes automatizados de *software*.

Numa primeira conversa com os responsáveis fomos percebendo que havia especial interesse pelo uso da ferramenta *Selenium IDE*, uma vez que era já conhecida e uma das mais utilizadas no mercado. Por outro lado, era pedido que verificasse outras possibilidades de ferramentas que correspondessem aos objetivos da empresa e que fossem intuitivas. Foi assim que surgiu a hipótese de uso do *Playwright*. Nasceu assim a primeira tarefa que passava por comparar as duas ferramentas e perceber qual a mais vantajosa para utilizar. Para iniciar esta comparação realizamos uma pesquisa mais teórica, mas também uma comparação prática. Para esta comparação pegamos num cenário de teste e desenvolvemos o respetivo em cada uma das ferramentas.

Ambas as ferramentas permitem a gravação dos testes de *software* que estamos a realizar e, portanto, realizamos um teste de Pesquisa de Vagas em cada uma das ferramentas com exatamente o mesmo cenário de teste.

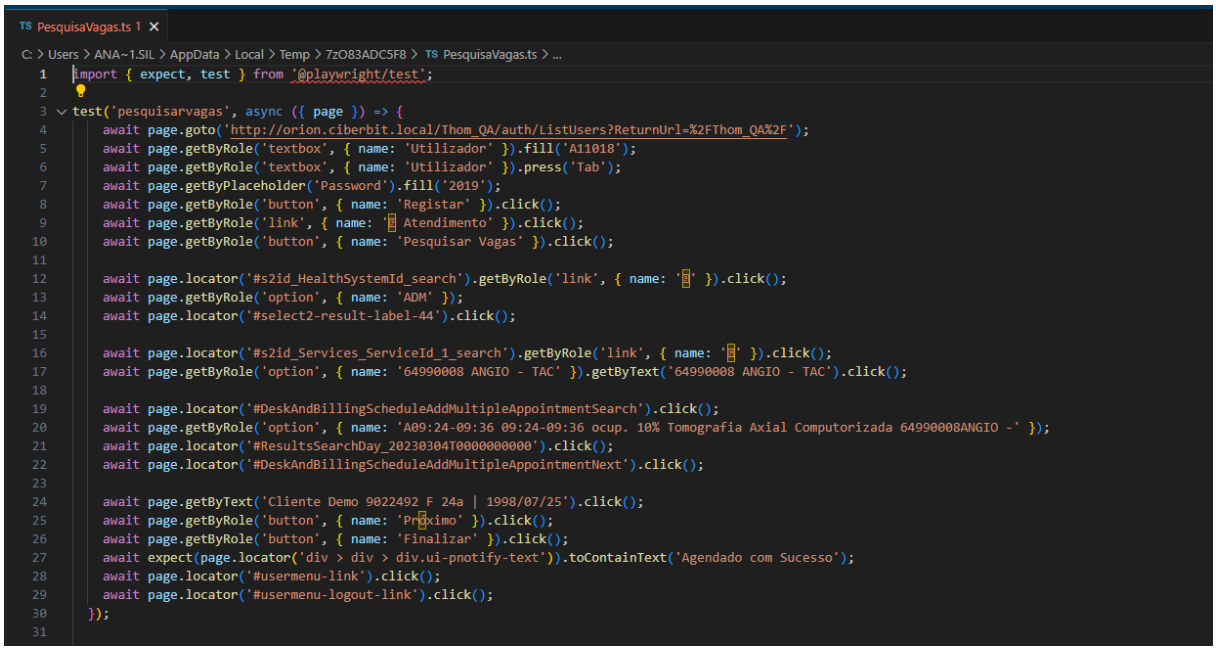
Se repararmos na Figura 12 estamos perante um conjunto de comandos que correspondem ao teste de Pesquisa de Vagas realizado na ferramenta *Selenium IDE*. Podemos ver que através da gravação é apresentado cada *click* e cada ação realizada no nosso programa, sendo que para cada elemento do campo *Target* é apresentado um conjunto de opções, desde Id, Css e Xpath. Nesta situação só temos de adicionar manualmente os campos de verificação e o preenchimento de alguns *Value*, sendo esta ferramenta bastante dinâmica e sem necessidade de conhecimento técnicos muito aprofundados.



	Command	Target	Value
1	open	http://orion.ciberbit.local/Thom_QA/	
2	click	css=.time-wrap > div:nth-child(1)	
3	click	css=[data-digit="2"]	
4	click	css=[data-digit="0"]	
5	click	css=[data-digit="1"]	
6	click	css=[data-digit="9"]	
7	click	css=.deskandbilling	
8	click	id=DeskAndBillingScheduleAddMultipleAppointmentGroupList	
9	click	css=#s2id_HealthSystemId_search > .select2-choice	
10	click	id=s2id_autogen17_search	
11	type	id=s2id_autogen17_search	Protocolo Escolas Teste
12	send keys	id=s2id_autogen17_search	${KEY_ENTER}
13	click	css=#s2id_DeskAndBillingScheduleAddAppointmentSmartScheduleInfo_PatientId_search > a	
14	type	id=s2id_autogen20_search	ana catarina araújo silva
15	send keys	id=s2id_autogen20_search	${KEY_ENTER}
16	click	css=#s2id_Services_ServiceId_1_search > .select2-choice	
17	type	xpath=//div[@id='select2-drop']/div/input	angio tac
18	send keys	xpath=//div[@id='select2-drop']/div/input	${KEY_ENTER}
19	click	id=DeskAndBillingScheduleAddMultipleAppointmentSearch	
20	verify element not present	css=.ui-pnotify-title	
21	wait for element present	id=deskAndBillingScheduleAddMultipleAppointmentSmartSchedule	80000
22	click	css=#ResultsSearchDay_20230904T0000000100 > div > div.smartschedule-slotgroup-slots	
23	click	id=DeskAndBillingScheduleAddMultipleAppointmentNext	
24	pause	5000	
25	verify element not present	css=.ui-pnotify-text	
26	wait for element present	css=#EpisodeSearch_container > .single-line	30000
27	click	id=DeskAndBillingScheduleAddMultipleAppointmentNext	
28	click	id=DeskAndBillingScheduleAddMultipleAppointmentFinalize	
29	assert text	xpath=//div[contains(text(),'Agendado com Sucesso')]	Agendado com Sucesso
30	click	id=usermenu-link	
31	click	css=.fa-power-off	

Figura 12- Teste de Pesquisa de Vagas em Selenium IDE

Por outro lado, na Figura 13 vemos o código gerado pela gravação do mesmo teste de Pesquisa de Vagas para a ferramenta *Playwright*. Apesar de ser tudo gerado pela gravação podemos ver que é tudo em formato código, sendo que existem alguns elementos que não são reconhecidos aquando da execução do teste e é necessário realizar de forma manual a identificação do elemento. Podemos ver que para identificar erros e perceber tudo o que é realizado ao longo do teste é necessário um conhecimento técnico mais aprofundado.



PesquisaVagas.ts 1

C: > Users > ANA~1.SIL > AppData > Local > Temp > 7zO83ADC5F8 > PesquisaVagas.ts > ...

```
import { expect, test } from '@playwright/test';

test('pesquisarvagas', async ({ page }) => {
    await page.goto('http://orion.ciberbit.local/Thom_QA/auth/ListUsers?ReturnUrl=%2FThom_QA%2F');
    await page.getByRole('textbox', { name: 'Utilizador' }).fill('A11018');
    await page.getByRole('textbox', { name: 'Utilizador' }).press('Tab');
    await page.getByPlaceholder('Password').fill('2019');
    await page.getByRole('button', { name: 'Registar' }).click();
    await page.getByRole('link', { name: ' Atendimento' }).click();
    await page.getByRole('button', { name: 'Pesquisar Vagas' }).click();

    await page.locator('#s2id_HealthSystemId_search').getByRole('link', { name: '' }).click();
    await page.getByRole('option', { name: 'ADM' });
    await page.locator('#select2-result-label-44').click();

    await page.locator('#s2id_Services_ServiceId_1_search').getByRole('link', { name: '' }).click();
    await page.getByRole('option', { name: '64990008 ANGIO - TAC' }).getByText('64990008 ANGIO - TAC').click();

    await page.locator('#DeskAndBillingScheduleAddMultipleAppointmentSearch').click();
    await page.getByRole('option', { name: 'A09:24-09:36 09:24-09:36 ocup. 10% Tomografia Axial Computorizada 64990008ANGIO -' });
    await page.locator('#ResultsSearchDay_20230304T0000000000').click();
    await page.locator('#DeskAndBillingScheduleAddMultipleAppointmentNext').click();

    await page.getByText('Cliente Demo 9022492 F 24a | 1998/07/25').click();
    await page.getByRole('button', { name: 'Próximo' }).click();
    await page.getByRole('button', { name: 'Finalizar' }).click();
    await expect(page.locator('div > div > div.ui-pnotify-text')).toContainText('Agendado com Sucesso');
    await page.locator('#usermenu-link').click();
    await page.locator('#usermenu-logout-link').click();
});
```

Figura 13- Teste Pesquisa de Vagas em Playwright

Ao trabalhar com ambas as ferramentas percebemos que durante a gravação dos testes a seleção de alguns elementos não era reconhecida, sendo que estas situações se repetiam mais frequentemente quando estávamos perante uma *combobox*. Depois de alguma pesquisa encontramos uma extensão do *Chrome*, a *ChroPath*, onde nos é gerado o seletor em diversos formatos (desde Css, XPath) de forma mais exata e permitia que o teste já avançasse com êxito. Esta extensão foi necessária principalmente para a utilização do *Playwright*, uma vez que o *Selenium IDE* nos dá opções de identificação de um seletor em diferentes formatos.

Além disso, percebemos que no *Playwright* tínhamos de trabalhar as *combobox* de diferentes formas dependendo se as mesmas permitiam fazer ou não *scroll* dos resultados.

Como queremos que a gravação de casos de teste seja realizada de uma maneira menos técnica, para o futuro deste projeto na empresa e para corresponder às necessidades da empresa optamos por seguir o caminho do *Selenium*.

3.2.2 Implementação da Automação

Durante esta secção vamos descrever quais os passos que foram realizados em cada um dos testes e ainda vamos mostrar todo o código de cada um dos testes em *Selenium IDE*. Durante este projeto realizamos um conjunto de testes que podemos ver no esquema da Figura 14.

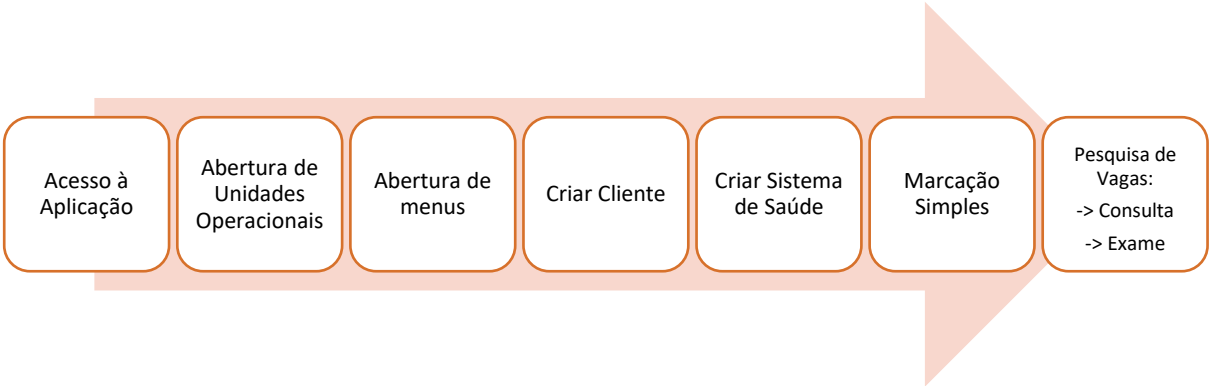


Figura 14- Conjunto de testes implementados na automação

Depois de selecionada a ferramenta a utilizar, chega o momento de passar à implementação da automação, isto é, chega o momento da criação dos casos de teste, tendo em conta os cenários possíveis em cada situação. De realçar que a seleção dos testes por ordem de prioridade foi discutida com os responsáveis da empresa.

No decorrer desta subsecção vamos descrever de forma minuciosa todo o processo pelo qual passa cada um dos casos de teste.

1. *Acesso à Aplicação*

Iniciamos com os testes de acesso à aplicação. O teste de realização de *Login* permite o acesso à aplicação através do preenchimento dos campos do número de utilizador e respetiva *password*. Em seguida, é executado o *click* no botão de "***Registar***".

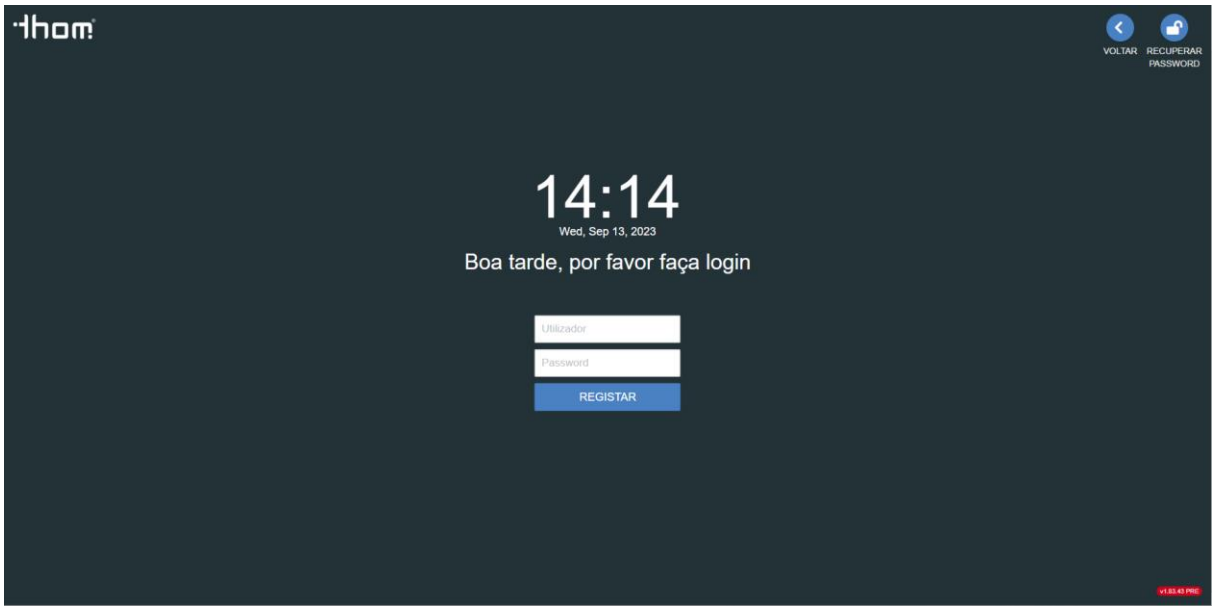


Figura 15- Ecrã de Acesso à Aplicação

Neste caso de teste verificamos todos os cenários possíveis, percebendo o comportamento do programa quando confrontado com as seguintes situações:

- A password inserida esteja incorreta
- O utilizador inserido seja inválido
- Não seja efetuado o preenchimento de algum dos campos ou de ambos

A gravação de cada um destes passos aqui descritos dá origem ao conjunto de comandos presente na Figura 16.

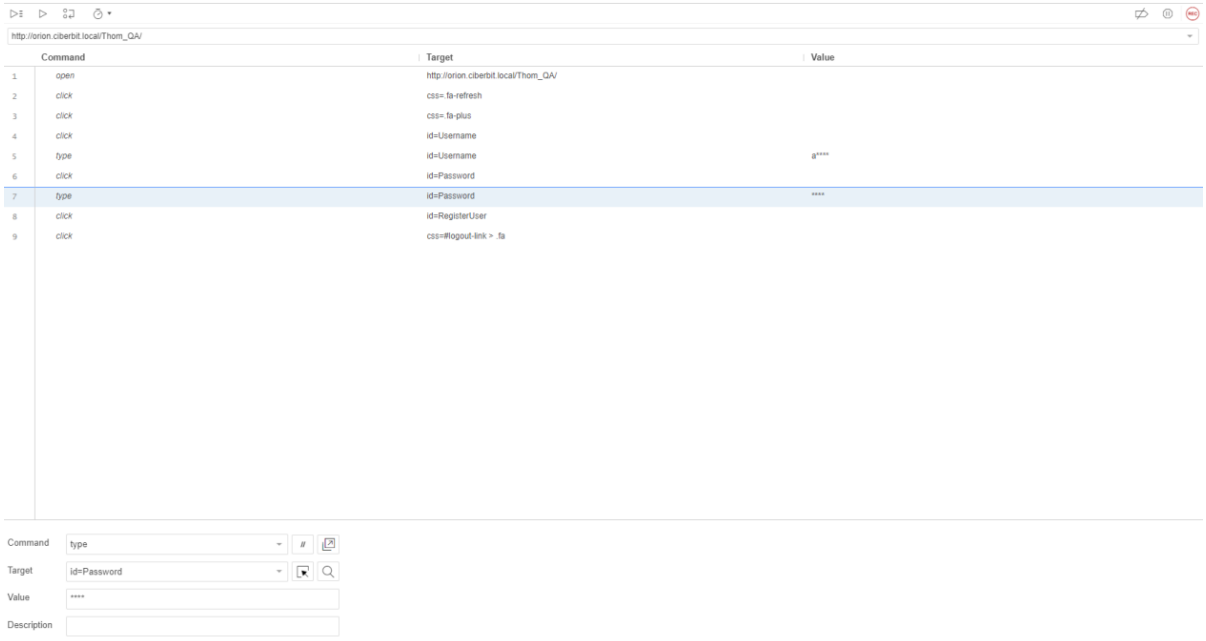


Figura 16- Teste de Acesso à Aplicação no Selenium IDE

Depois de um primeiro acesso ao sistema numa máquina, o *Login* passará a ser efetuado através de um PIN de quatro dígitos. O teste da inserção deste PIN é executado clicando nos algoritmos do PIN para o respetivo utilizador.

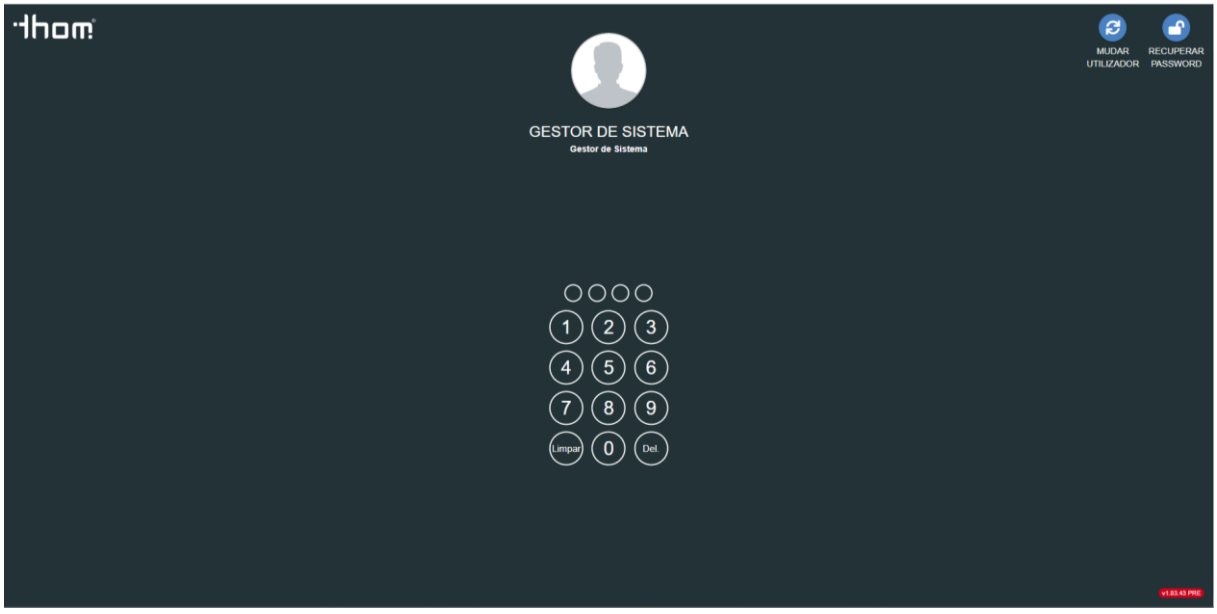


Figura 17- Ecrã de inserção do PIN de acesso

2. Abertura de Unidades Operacionais

O próximo conjunto de testes a ser executado remete aos testes que permitissem verificar se os ecrãs das respetivas unidades operacionais eram abertos com sucesso, para isso realizam-se os passos de *click* no botão respetivo a cada unidade operacional.

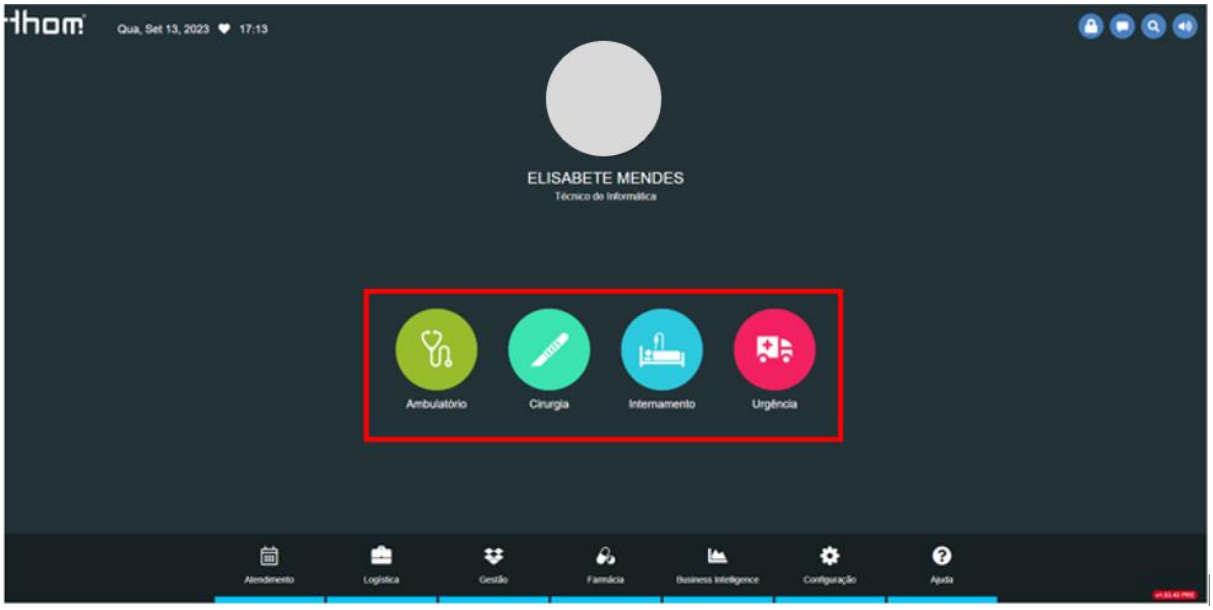


Figura 18- Ecrã inicial do I'MTHOM com realce às Unidades Operacionais

Depois de clicar na unidade operacional é feita a verificação da zona do título da página para que tenha o nome correspondente à unidade operacional selecionada.

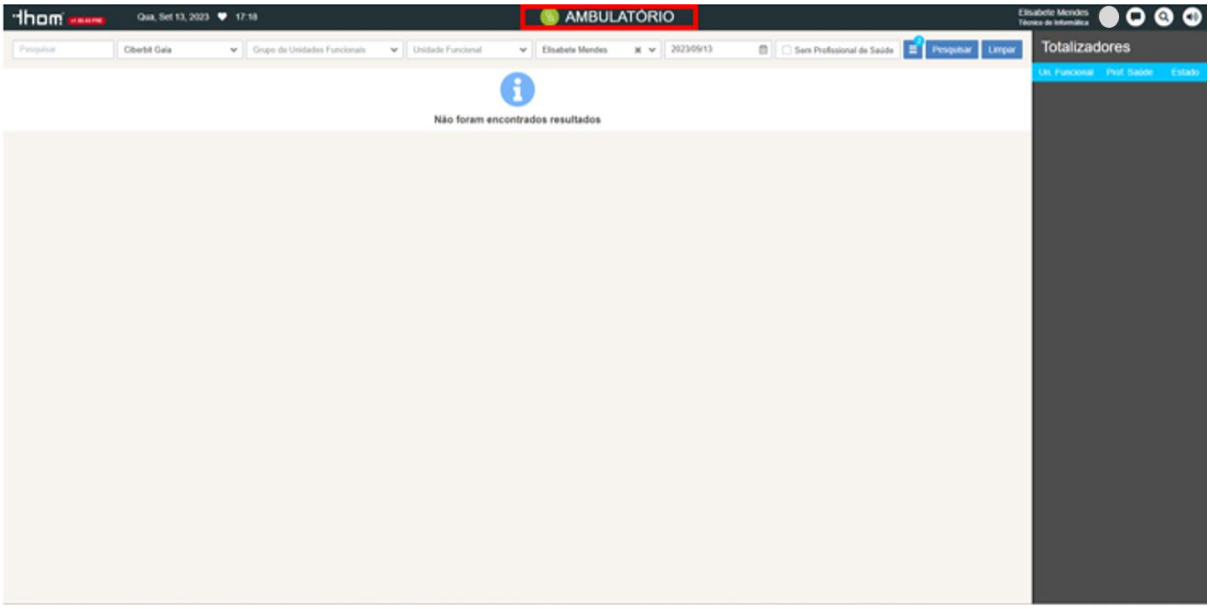


Figura 19- Ecrã da Unidade Operacional Ambulatório com realce à zona do título

Realizamos este teste para todas as unidades operacionais do nosso programa (Ambulatório, Cirurgia, Internamento e Urgência). Com a gravação destes testes na ferramenta *Selenium* IDE obtivemos códigos como o apresentado na Figura 20.

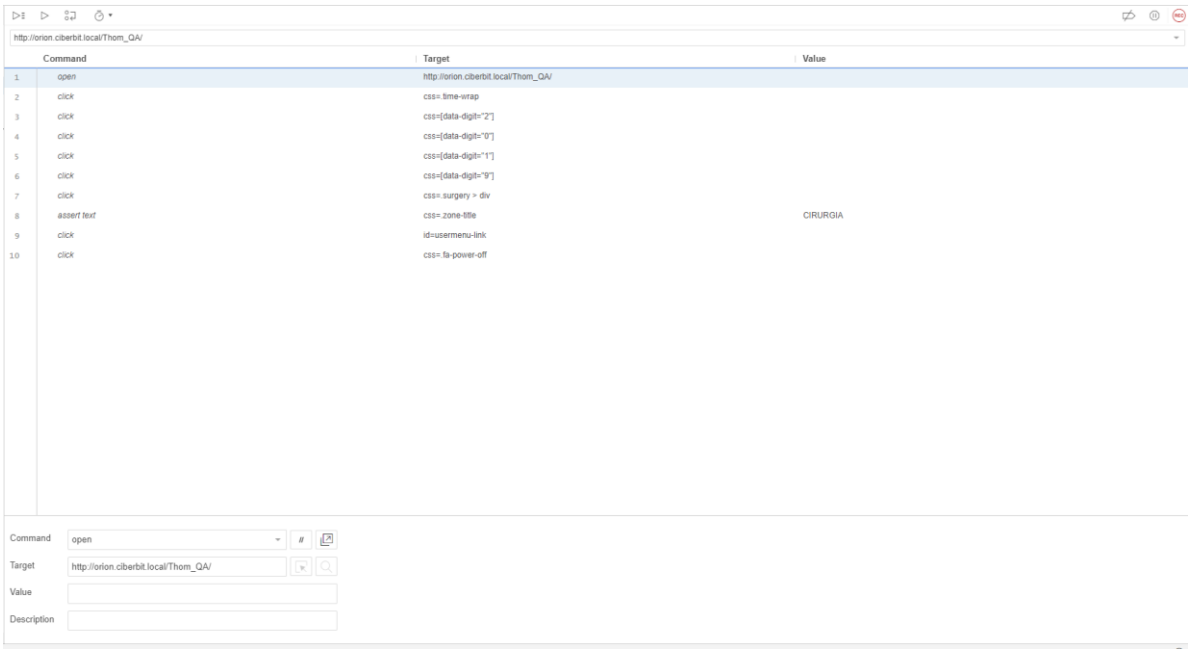


Figura 20- Teste de Abertura de Unidades Operacionais em Selenium IDE

3. Abertura de Menus

Seguidos das unidades operacionais estão todos os outros ecrãs e, portanto, é importante testar se todos os ecrãs do I'MTHOM são abertos devidamente. Para isso necessitamos de realizar um conjunto de tarefas que começam com o *click* em cada uma das opções realçadas na Figura 21.

Figura 21- Ecrã inicial do I'MTHOM com realce para os diferentes menus

Quando cada um destes ecrãs é aberto, procede-se à testagem de abertura de cada uma das opções existentes. Tal como nos descreve a Figura 22 vamos percorrer todas as opções existentes.

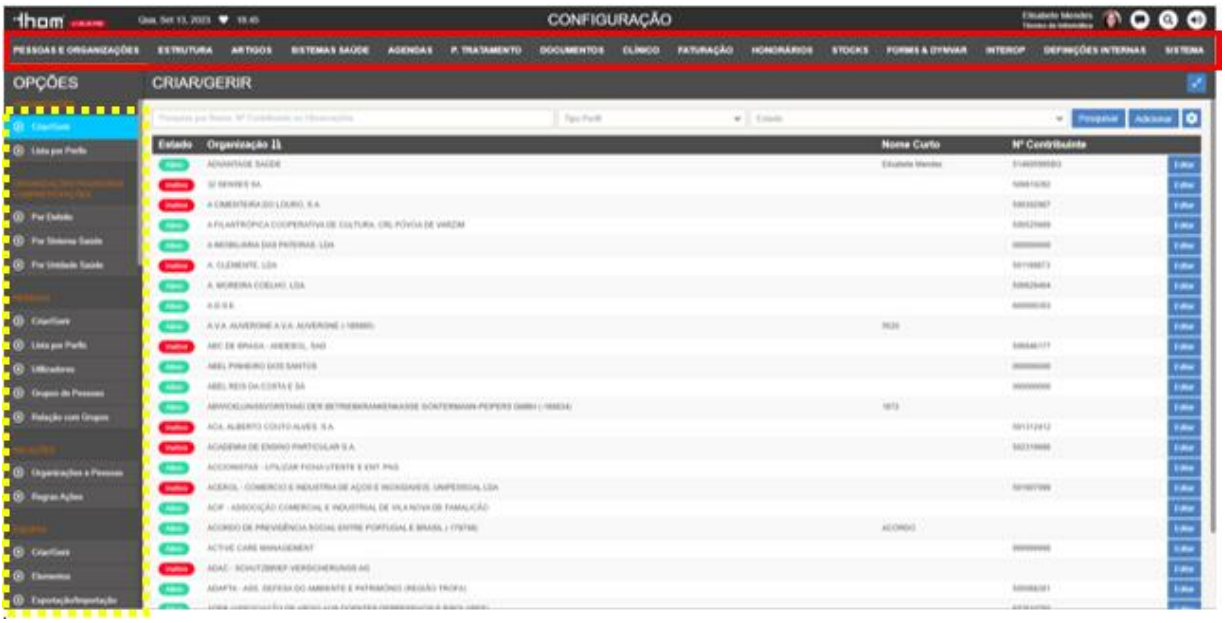
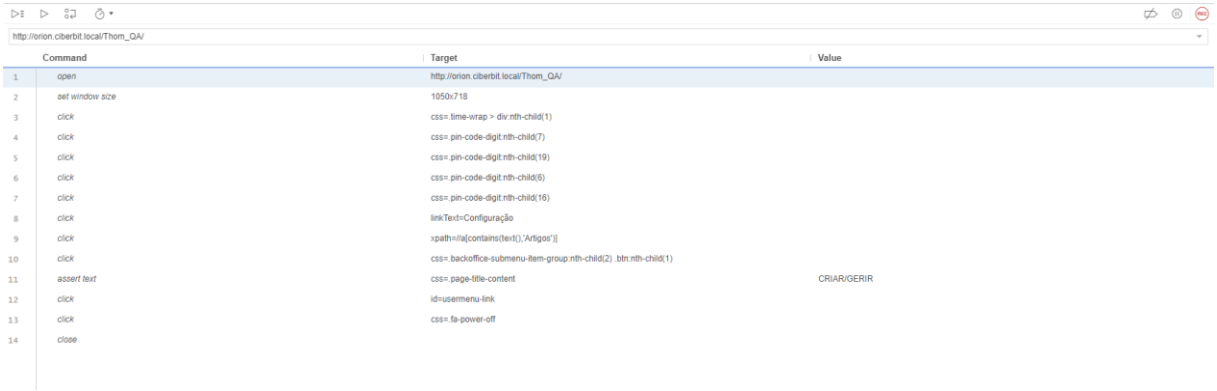


Figura 22- Ecrã de Configuração

É realizado o *click* em cada uma das opções de determinado menu e depois de clicarmos é verificado se o título da página corresponde ao título da opção.

Na imagem podemos ver o exemplo de como se procedeu para o ecrã de ***Configuração***. O menu ***Pessoas e Organizações*** está aberto e vamos clicando em cada uma das opções, estando aberta no momento da imagem a opção ***Criar/Gerir***.

Os testes em *Selenium IDE* de abertura de cada uma das opções existente em cada um dos menus terá um formato igual ao apresentado na Figura 23.



	Command	Target	Value
1	open	http://orion.ciberbit.local/Thom_QA/	
2	set window size	1050x718	
3	click	css=.time-wrap > div:nth-child(1)	
4	click	css=.pin-code-digit:nth-child(7)	
5	click	css=.pin-code-digit:nth-child(19)	
6	click	css=.pin-code-digit:nth-child(6)	
7	click	css=.pin-code-digit:nth-child(16)	
8	click	linkText=Configuração	
9	click	xpath=//a[contains(text(),'Artigos')]	
10	click	css=.backoffice-submenu-item-group:nth-child(2) .btn:nth-child(1)	
11	assert text	css=.page-title-content	CRIAR/GERIR
12	click	id=usermenu-link	
13	click	css=.fa-power-off	
14	close		

Figura 23- Teste de abertura de ecrãs de cada uma das opções em Selenium IDE

4. Criar Cliente

A partir do momento em que já temos a abertura de todos os ecrãs testada, chega a altura de passar para testes mais completos, como é o caso da criação de um cliente. Para a criação de um novo cliente é necessário que, a partir da página inicial do I'MTHOM, cliquemos no ícone cujo símbolo é uma lupa.

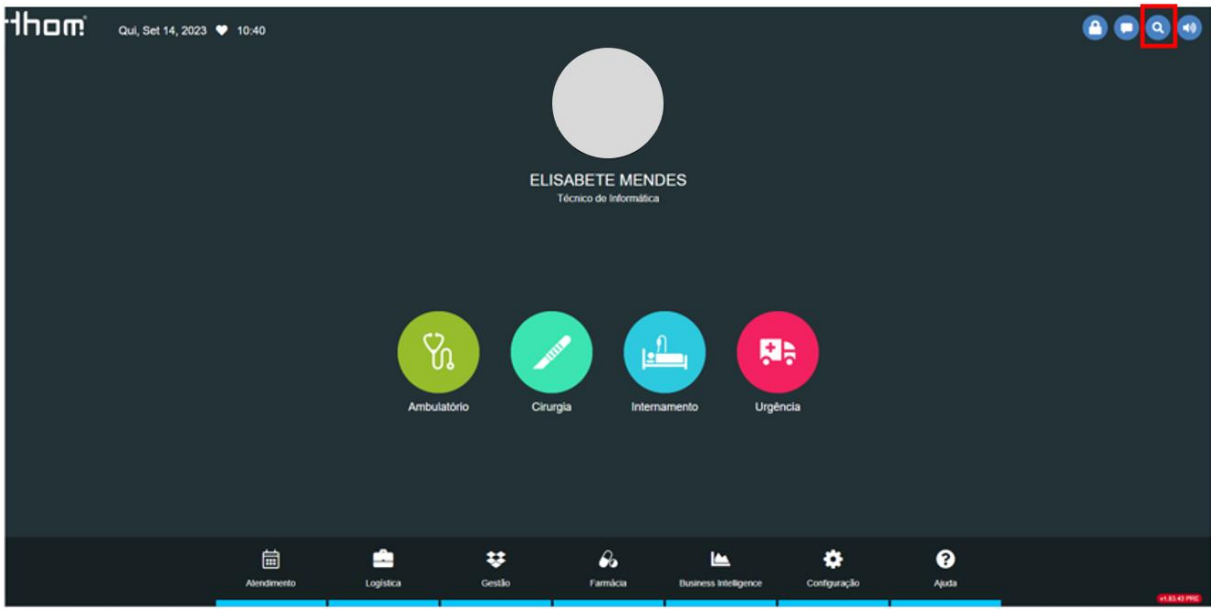


Figura 24- Ecrã inicial do I'MTHOM com realce ao ícone de Pesquisa

É aberto um menu de pesquisa, onde temos o botão ***Criar Novo Cliente,*** clicamos nele e é aberta uma nova janela de navegação. Aqui encontramos um conjunto de opções de criação e clicamos no botão com a descrição ***Criar Manual***.

Após este *click,* abre-se o ecrã de criação da ficha de cliente.

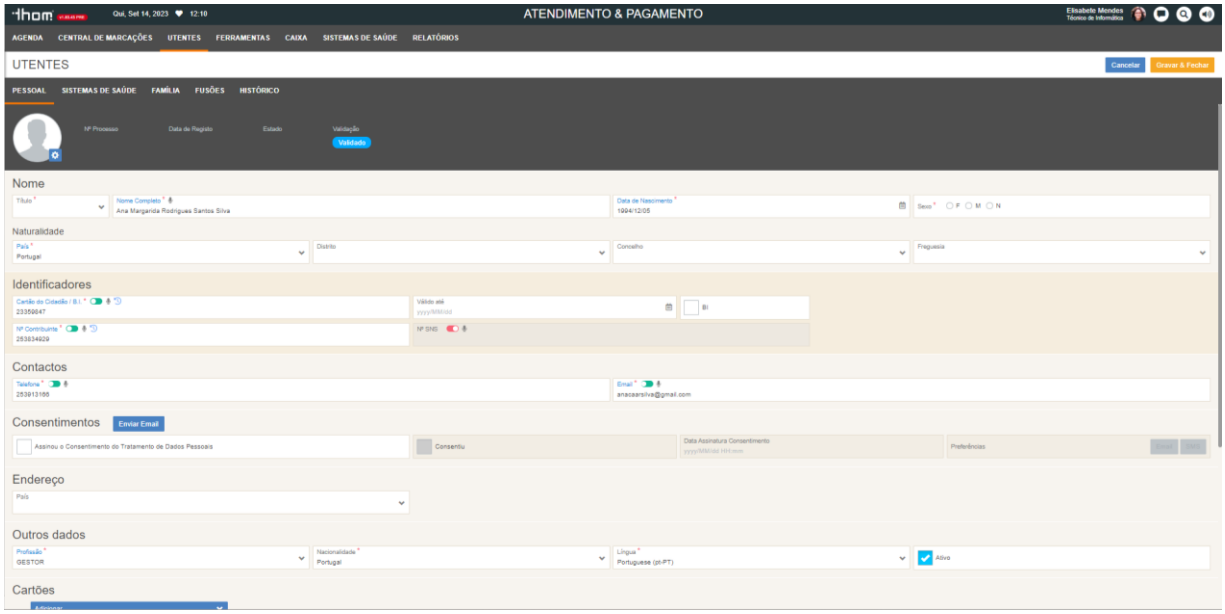


Figura 25- Ecrã de criação de ficha de cliente

Como podemos ver na Figura 25 temos um conjunto de campos de preenchimento obrigatório, sendo por isso o passo seguinte do teste preencher estes campos da seguinte forma:

- **Título:** selecionamos a opção com valor vazio
- **Nome Completo:** Ana Margarida Rodrigues Santos Silva
- **Data de Nascimento:** 1994/12/05
- **Sexo:** campo de escolha, selecionamos o F
- **País:** Portugal
- **Cartão de Cidadão/BI:** 23359847
- **Nº Contribuinte:** 253834929
- **Telefone:** 253914166
- **Email:** teste@teste.pt
- **Profissão:** Gestor
- **Nacionalidade:** Portugal
- **Língua:** Portuguesa (pt-PT)
- *Check* no campo **Ativo**
- No campo do ***Nº SNS*** o validador é desligado, indicando como ***Motivo*** que *"Não tem SNS" e check* na indicação de *"Não voltar a pedir o preenchimento deste campo".*

Ainda na ficha de cliente passamos para o menu de ***Sistemas de Saúde*** onde também são preenchidos campos da seguinte forma:

Insere-se um primeiro ***Sistema de Saúde***: Multicare-Fidelidade Mundial

- ***Nº Benificiário*** deste sistema de saúde: 1543521
- ***Data Início:*** Seleção de um dia no calendário que é gerado com o *click* no campo

Clicamos no botão ***Adicionar*** e insere-se um segundo ***Sistema de Saúde:*** SNS

- ***Nº Benificiário*** desse sistema de saúde: 7804123
- ***Data Início:*** Seleção de um dia no calendário que é gerado com o *click* no campo

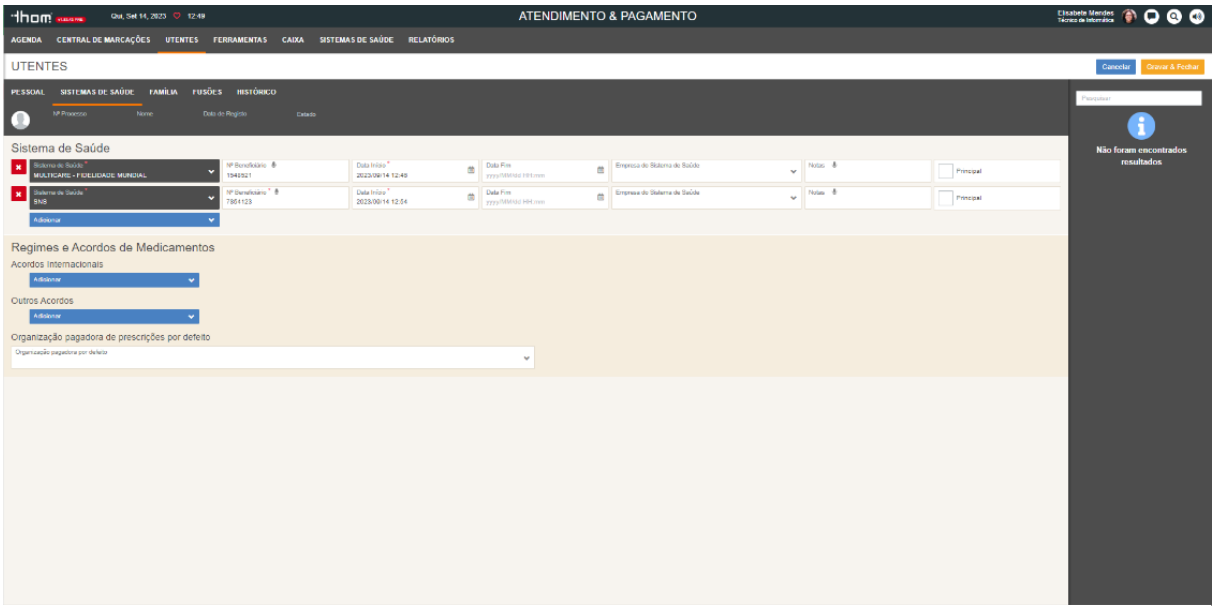
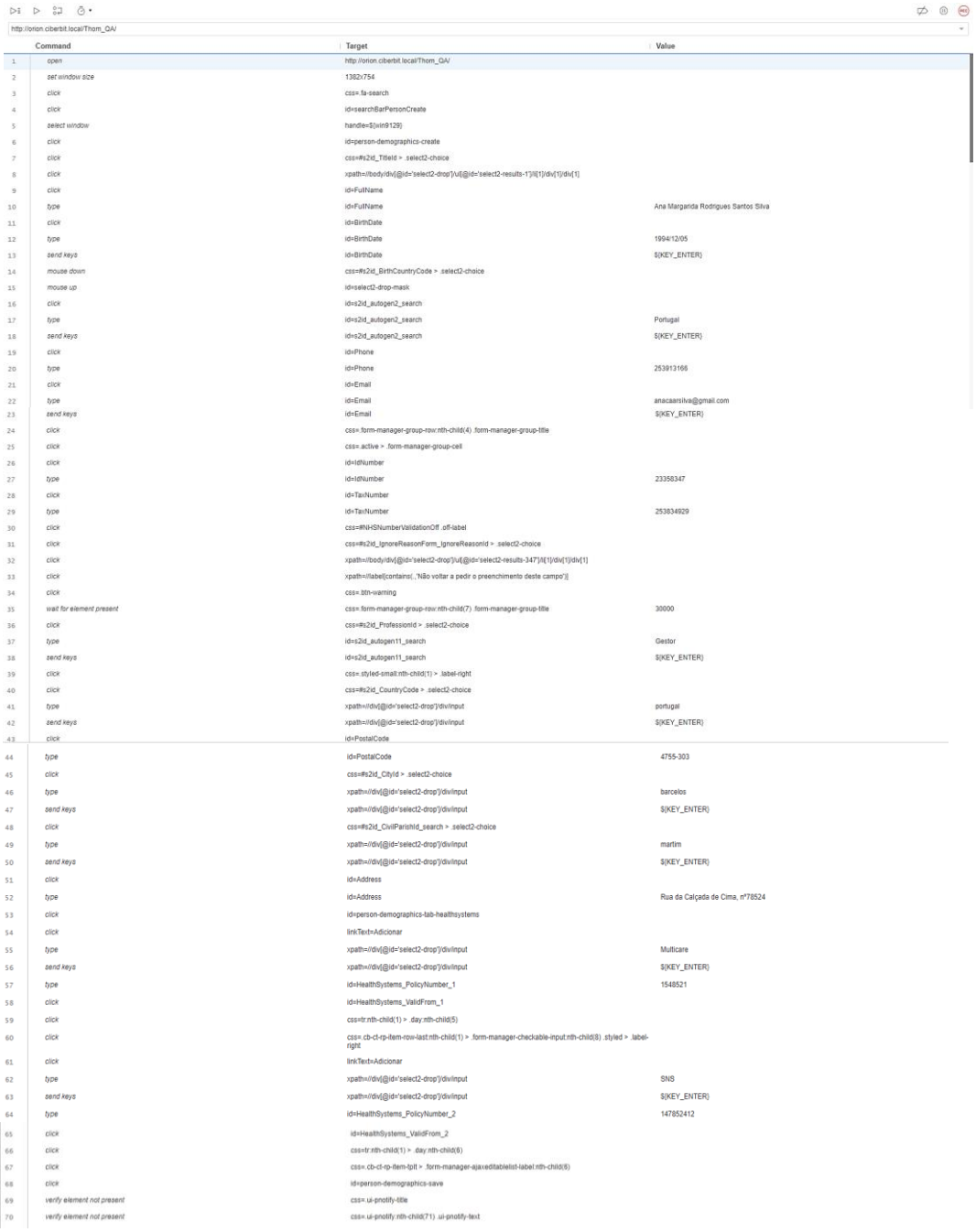


Figura 26- Menu de Sistemas de Saúde associadas ao cliente

O passo seguinte é clicar no botão ***Gravar & Fechar***, verificar que não há existência de mensagens de erro ou alertas e fica então criado o cliente. Por fim, guardamos em forma de variável o Cliente aqui criado para que seja usado em testes futuros.

Quando realizamos a gravação de todos estes passos no *Selenium IDE* obtemos um conjunto de comandos como o apresentado na Figura 27.



	Command	Target	Value
1	open	http://orion.ciberbit.local/Thom_QA/	
2	set window size	1382x754	
3	click	css=.fa-search	
4	click	id=searchBarPersonCreate	
5	select window	handle=${win9129}	
6	click	id=person-demographics-create	
7	click	css=#s2id_TitleId > .select2-choice	
8	click	xpath=//body/div[@id='select2-drop']/ul[@id='select2-results-1']/li[1]/div[1]/div[1]	
9	click	id=FullName	
10	type	id=FullName	Ana Margarida Rodrigues Santos Silva
11	click	id=BirthDate	
12	type	id=BirthDate	1994/12/05
13	send keys	id=BirthDate	${KEY_ENTER}
14	mouse down	css=#s2id_BirthCountryCode > .select2-choice	
15	mouse up	id=select2-drop-mask	
16	click	id=s2id_autogen2_search	
17	type	id=s2id_autogen2_search	Portugal
18	send keys	id=s2id_autogen2_search	${KEY_ENTER}
19	click	id=Phone	
20	type	id=Phone	253913166
21	click	id=Email	
22	type	id=Email	anacaarsilva@gmail.com
23	send keys	id=Email	${KEY_ENTER}
24	click	css=.form-manager-group-row:nth-child(4) .form-manager-group-title	
25	click	css=.active > .form-manager-group-cell	
26	click	id=IdNumber	
27	type	id=IdNumber	23358347
28	click	id=TaxNumber	
29	type	id=TaxNumber	253834929
30	click	css=#NHSNumberValidationOff .off-label	
31	click	css=#s2id_IgnoreReasonForm_IgnoreReasonId > .select2-choice	
32	click	xpath=//body/div[@id='select2-drop']/ul[@id='select2-results-347']/li[1]/div[1]/div[1]	
33	click	xpath=//label[contains(.,'Não voltar a pedir o preenchimento deste campo')]	
34	click	css=.btn-warning	
35	wait for element present	css=.form-manager-group-row:nth-child(7) .form-manager-group-title	30000
36	click	css=#s2id_ProfessionId > .select2-choice	
37	type	id=s2id_autogen11_search	Gestor
38	send keys	id=s2id_autogen11_search	${KEY_ENTER}
39	click	css=.styled-small:nth-child(1) > .label-right	
40	click	css=#s2id_CountryCode > .select2-choice	
41	type	xpath=//div[@id='select2-drop']/div/input	portugal
42	send keys	xpath=//div[@id='select2-drop']/div/input	${KEY_ENTER}
43	click	id=PostalCode	
44	type	id=PostalCode	4755-303
45	click	css=#s2id_CityId > .select2-choice	
46	type	xpath=//div[@id='select2-drop']/div/input	barcelos
47	send keys	xpath=//div[@id='select2-drop']/div/input	${KEY_ENTER}
48	click	css=#s2id_CivilParishId_search > .select2-choice	
49	type	xpath=//div[@id='select2-drop']/div/input	martim
50	send keys	xpath=//div[@id='select2-drop']/div/input	${KEY_ENTER}
51	click	id=Address	
52	type	id=Address	Rua da Calçada de Cima, nº78524
53	click	id=person-demographics-tab-healthsystems	
54	click	linkText=Adicionar	
55	type	xpath=//div[@id='select2-drop']/div/input	Multicare
56	send keys	xpath=//div[@id='select2-drop']/div/input	${KEY_ENTER}
57	type	id=HealthSystems_PolicyNumber_1	1548521
58	click	id=HealthSystems_ValidFrom_1	
59	click	css=tr:nth-child(1) > .day:nth-child(5)	
60	click	css=.cb-ct-rp-item-row-last:nth-child(1) > .form-manager-checkable-input:nth-child(8) .styled > .label-right	
61	click	linkText=Adicionar	
62	type	xpath=//div[@id='select2-drop']/div/input	SNS
63	send keys	xpath=//div[@id='select2-drop']/div/input	${KEY_ENTER}
64	type	id=HealthSystems_PolicyNumber_2	147852412
65	click	id=HealthSystems_ValidFrom_2	
66	click	css=tr:nth-child(1) > .day:nth-child(6)	
67	click	css=.cb-ct-rp-item-tpl > .form-manager-ajaxeditablelist-label:nth-child(8)	
68	click	id=person-demographics-save	
69	verify element not present	css=.ui-pnotify-title	
70	verify element not present	css=.ui-pnotify:nth-child(71) .ui-pnotify-text	

Figura 27- Testes de Criação de Cliente no Selenium IDE

De realçar que todos os dados utilizados nesta testagem são puramente fictícios para efeitos de teste e, portanto, não apresentam dados de nenhum Cliente real.

5. Criar Sistema de Saúde

Passamos agora para outro teste importante, a criação de um sistema de saúde. Para a criar um sistema de saúde começamos por, através da página inicial do I'MTHOM, fazer os *clicks* ***Configuração > Sistemas Saúde.***

Depois de estarmos no ecrã dos sistemas de saúde encontramos o botão ***Adicionar*** e clicamos nele.

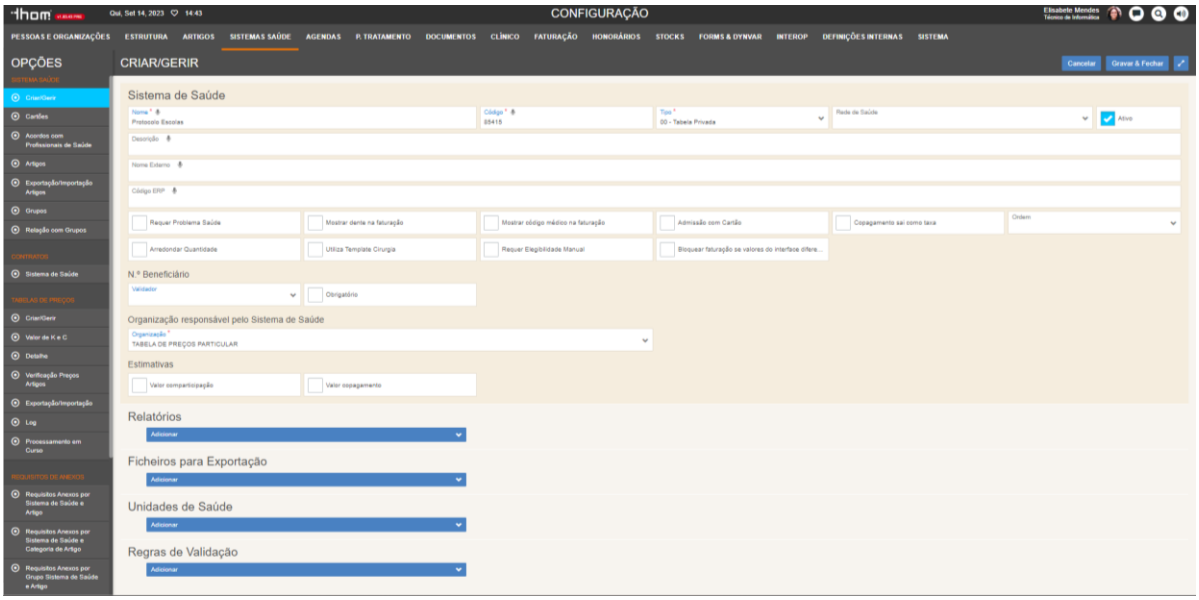


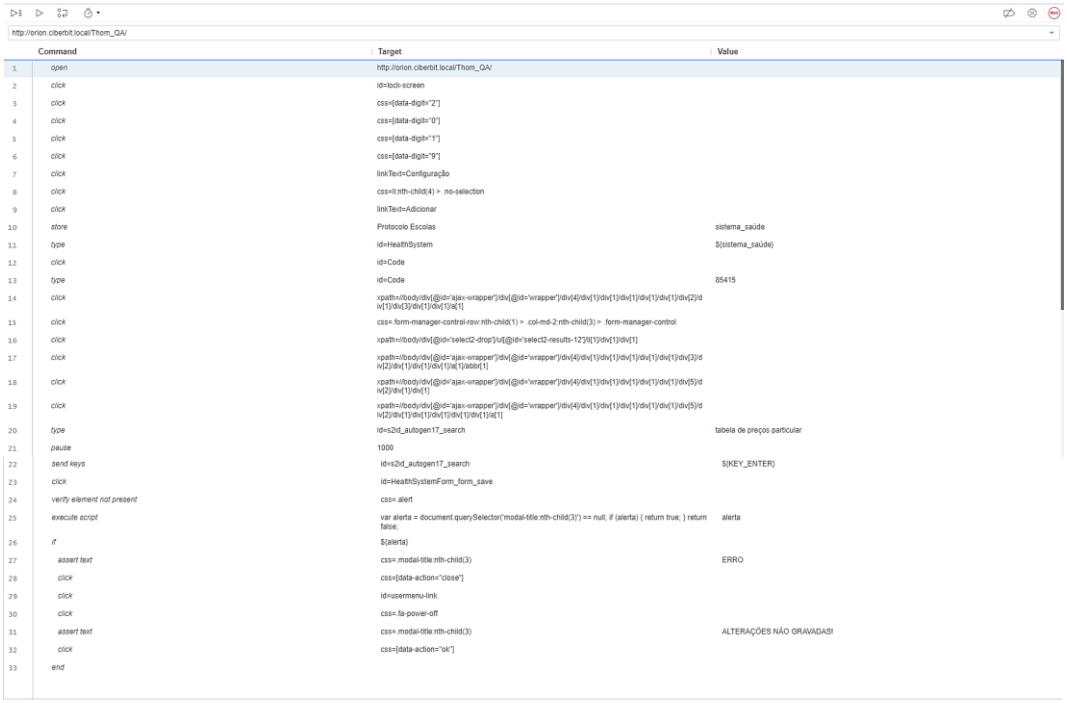
Figura 28- Ecrã de criação de Sistema de Saúde

É então aberto o ecrã relativo à criação de um sistema de saúde e passamos ao preenchimento dos campos obrigatórios.

- ***Nome:*** Protocolo Escolas Teste
- ***Código:*** 85415
- ***Tipo:*** 00-Tabela Privada
- ***Organização:*** Tabela de Preços Particular
- No campo ***Nº Benificiário*** eliminamos o preenchimento que é trazido por defeito e deixamos vazio

Depois de preenchidos todos estes campos, clicamos no botão ***Gravar & Fechar,*** é feita a verificação de que não é apresentado nenhum alerta ou mensagem de erro e é então gravado o sistema de saúde criado numa variável, para que possa ser utilizado em testes futuros.

Mais uma vez fazemos a gravação de todos estes passos com a ajuda da ferramenta *Selenium IDE* e obtemos o conjunto de comandos apresentado na Figura 29.



	Command	Target	Value
1	open	http://orion.ciberbit.local/Thom_QA/	
2	click	id=locli-screen	
3	click	css=[data-digit="2"]	
4	click	css=[data-digit="0"]	
5	click	css=[data-digit="1"]	
6	click	css=[data-digit="9"]	
7	click	linkText=Configuração	
8	click	css=li:nth-child(4) > .no-selection	
9	click	linkText=Adicionar	
10	store	Protocolo Escolas	sistema_saúde
11	type	id=HealthSystem	${sistema_saúde}
12	click	id=Code	
13	type	id=Code	85415
14	click	xpath=//body/div[@id='ajax-wrapper']/div[@id='wrapper']/div[4]/div[1]/div[1]/div[1]/div[1]/div[1]/div[2]/div[1]/div[3]/div[1]/div[1]/a[1]	
15	click	css=.form-manager-control-row:nth-child(1) > .col-md-2:nth-child(3) > .form-manager-control	
16	click	xpath=//body/div[@id='select2-drop']/ul[@id='select2-results-12']/li[1]/div[1]/div[1]	
17	click	xpath=//body/div[@id='ajax-wrapper']/div[@id='wrapper']/div[4]/div[1]/div[1]/div[1]/div[1]/div[1]/div[3]/div[2]/div[1]/div[1]/div[1]/a[1]/abbr[1]	
18	click	xpath=//body/div[@id='ajax-wrapper']/div[@id='wrapper']/div[4]/div[1]/div[1]/div[1]/div[1]/div[1]/div[5]/div[2]/div[1]/div[1]	
19	click	xpath=//body/div[@id='ajax-wrapper']/div[@id='wrapper']/div[4]/div[1]/div[1]/div[1]/div[1]/div[1]/div[5]/div[2]/div[1]/div[1]/div[1]/div[1]/div[1]/a[1]	
20	type	id=s2id_autogen17_search	tabela de preços particular
21	pause	1000	
22	send keys	id=s2id_autogen17_search	${KEY_ENTER}
23	click	id=HealthSystemForm_form_save	
24	verify element not present	css=.alert	
25	execute script	var alerta = document.querySelector('modal-title:nth-child(3)') == null; if (alerta) { return true; } return false;	alerta
26	if	${alerta}	
27	assert text	css=.modal-title:nth-child(3)	ERRO
28	click	css=[data-action="close"]	
29	click	id=usermenu-link	
30	click	css=.fa-power-off	
31	assert text	css=.modal-title:nth-child(3)	ALTERAÇÕES NÃO GRAVADAS!
32	click	css=[data-action="ok"]	
33	end		

Figura 29- Teste de criação de Sistema de Saúde no Selenium IDE

Depois de todos estes testes, verificamos dentro dos testes já mais complexos, quais as operações mais executadas no programa e criamos testes para verificar o seu bom funcionamento.

6. Marcação Simples

Iniciamos pela marcação de uma visita para o paciente. Tendo como ponto de partida a página inicial, começamos com o *click* em ***Atendimento*** abrindo automaticamente a página da ***Agenda***, que se encontra na Figura 30.

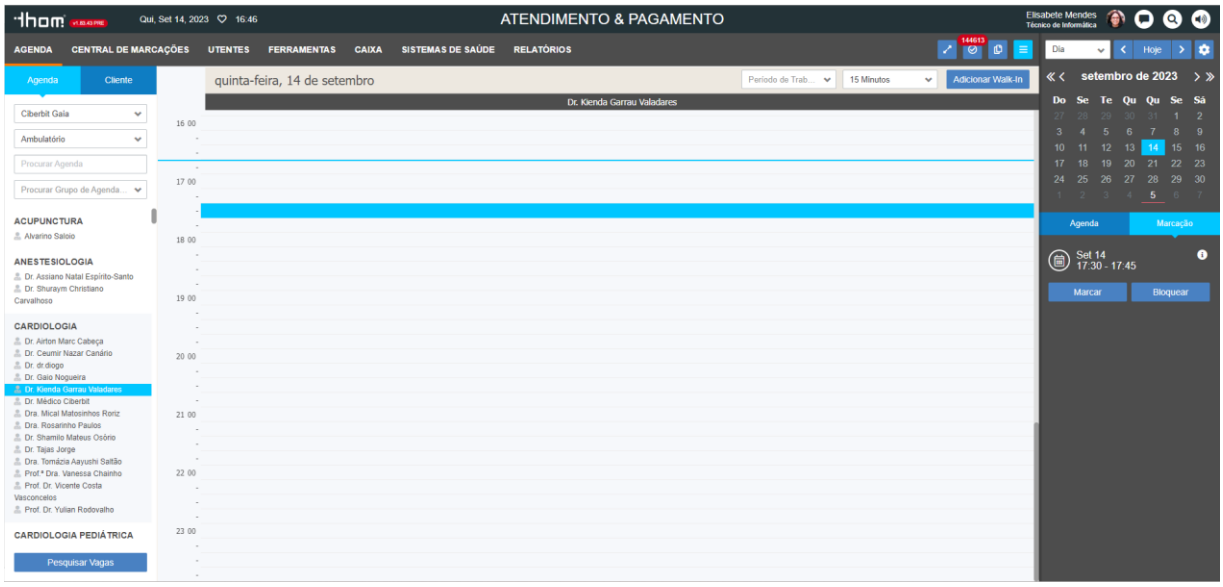


Figura 30- Ecrã da Agenda

No campo de seleção de ***Unidade Operacional*** selecionamos o Ambulatório e em seguida selecionamos o ***Profissional de Saúde*** que se encontre na 5º posição de Cardiologia. Ao clicar no profissional, aparece-nos a sua agenda, clicamos numa célula respetiva a um horário (neste caso às 17h30) e avançamos com o *click* no botão ***Marcar***. Automaticamente é aberto o *pop-up* de ***Adicionar Marcação.***

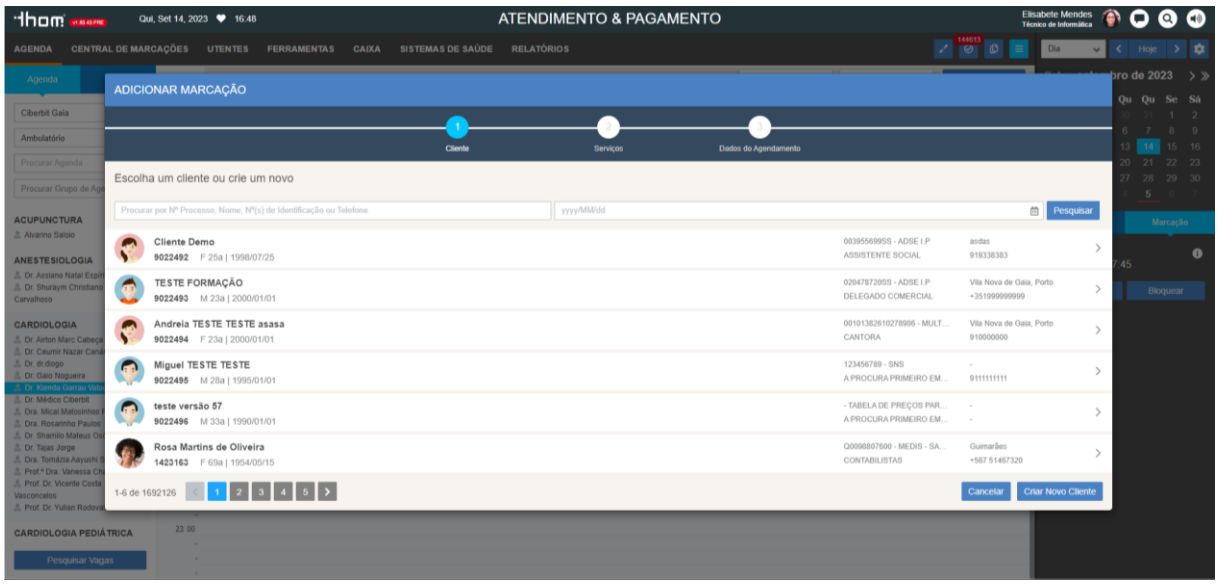


Figura 31- Pop-up de Adicionar Marcação

Neste ecrã no campo de pesquisa inserimos o valor do teste de Criar Cliente e quando aparecerem os resultados clicar no respetivo perfil. Passamos então para a fase de preenchimento dos ***Serviços*** a realizar.

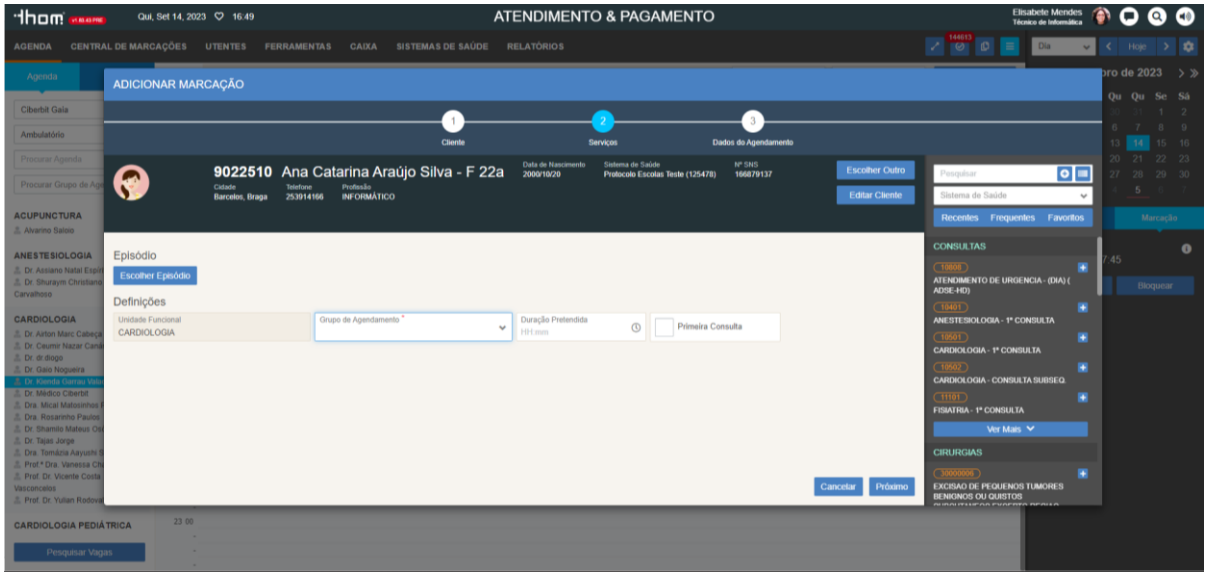


Figura 32- Etapa de seleção dos Serviços a realizar

No campo ***Grupo de Agendamento*** irão aparecer apenas as opções relativas a Cardiologia e selecionamos a opção de *Consulta de Cardiologia.* Clicamos no botão ***Próximo***, verificamos se não é existente alguma mensagem de erro e avançamos para o ecrã de ***Dados de Agendamento.*** Aqui, clicamos no botão ***Finalizar*** para que seja efetuada a marcação. Mais uma vez confirmamos se não existe nenhum alerta ou mensagem de erro e verificamos que a mensagem "Agendado com Sucesso" aparece no canto superior direito do nosso ecrã.

Depois da marcação estar realizada irá aparecer a agenda do médico, clicamos em cima da visita e do lado direito teremos um menu com o botão ***CheckIn***, clicamos nesse botão e é aberto um novo ecrã de ***CheckIn.*** Neste ecrã fazemos o seguinte:

- Verificamos se o ***Documento de Identificação*** está selecionado como Outro e, caso não esteja, selecionamos.

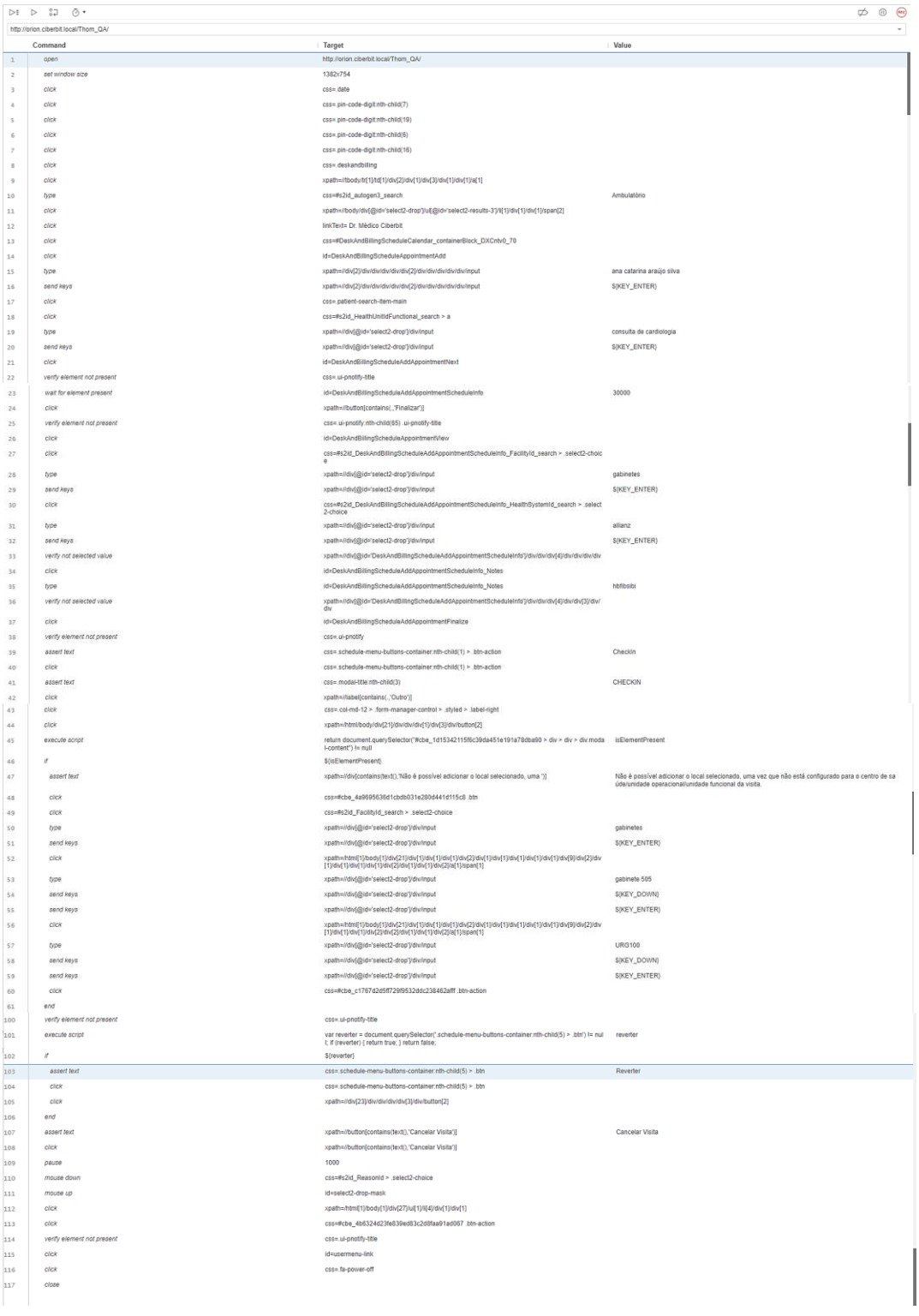
Fazemos então *click* no botão ***CheckIn*** deste novo ecrã, verificamos se não é apresentado nenhum alerta ou mensagem de erro.

Nesta situação fazemos ainda outra verificação:

- Se aparecer o alerta com o seguinte corpo de mensagem "Não é possível adicionar o local selecionado, uma vez que não está configurado para o centro de saúde/unidade operacional/unidade funcional da visita." clicamos no botão ***Ok*** e voltamos ao ecrã de ***CheckIn*** para fazer as seguintes alterações:
    - Alterar o ***FacilityID****;*
    - Alterar o ***Facility*** relativo a cada uma das visitas;
- Depois disto clicamos novamente no botão de ***CheckIn.***

Avançamos com o processo inverso que passa por reverter o ***CheckIn***, clicando então no botão ***Reverter***, situado no lado direito do nosso ecrã. É aberto um *pop-up* para confirmar esta ação e clicamos novamente num botão deste *pop-up*, que também diz reverter. Em seguida, voltamos ao lado direito do ecrã e clicamos no botão ***Cancelar Visita***. Aqui será aberto novamente um *pop-up* de validação da ação onde indicamos o ***Motivo*** de cancelamento, sendo a opção ***Outro*** a selecionada e clicamos no botão ***Cancelar Visita*** deste *pop-up*.

Passamos para a gravação a partir da ferramenta *Selenium IDE* e obtemos o conjunto de comandos apresentado na Figura 33.



http://orion.ciberbit.local/Thom_QA/

	Command	Target	Value
1	open	http://orion.ciberbit.local/Thom_QA/	
2	set window size	1382x754	
3	click	css=.date	
4	click	css=.pin-code-digit:nth-child(7)	
5	click	css=.pin-code-digit:nth-child(10)	
6	click	css=.pin-code-digit:nth-child(6)	
7	click	css=.pin-code-digit:nth-child(16)	
8	click	css=.deskandbilling	
9	click	xpath=//tbody/tr[1]/td[1]/div[2]/div[1]/div[3]/div[1]/div[1]/a[1]	
10	type	css=#s2id_autogen3_search	Ambulatório
11	click	xpath=//body/div[@id='select2-drop']/ul[@id='select2-results-3']/li[1]/div[1]/div[1]/span[2]	
12	click	linkText= Dr. Médico Ciberbit	
13	click	css=#DeskAndBillingScheduleCalendar_containerBlock_DXCntv0_70	
14	click	id=DeskAndBillingScheduleAppointmentAdd	
15	type	xpath=//div[2]/div/div/div/div/div[2]/div/div/div/div/div/input	ana catarina araújo silva
16	send keys	xpath=//div[2]/div/div/div/div/div[2]/div/div/div/div/div/input	${KEY_ENTER}
17	click	css=.patient-search-item-main	
18	click	css=#s2id_HealthUnitIdFunctional_search > a	
19	type	xpath=//div[@id='select2-drop']/div/input	consulta de cardiologia
20	send keys	xpath=//div[@id='select2-drop']/div/input	${KEY_ENTER}
21	click	id=DeskAndBillingScheduleAddAppointmentNext	
22	verify element not present	css=.ui-pnotify-title	
23	wait for element present	id=DeskAndBillingScheduleAddAppointmentScheduleInfo	30000
24	click	xpath=//button[contains(.,'Finalizar')]	
25	verify element not present	css=.ui-pnotify:nth-child(65) .ui-pnotify-title	
26	click	id=DeskAndBillingScheduleAppointmentView	
27	click	css=#s2id_DeskAndBillingScheduleAddAppointmentScheduleInfo_FacilityId_search > .select2-choice	
28	type	xpath=//div[@id='select2-drop']/div/input	gabinetes
29	send keys	xpath=//div[@id='select2-drop']/div/input	${KEY_ENTER}
30	click	css=#s2id_DeskAndBillingScheduleAddAppointmentScheduleInfo_HealthSystemId_search > .select2-choice	
31	type	xpath=//div[@id='select2-drop']/div/input	allianz
32	send keys	xpath=//div[@id='select2-drop']/div/input	${KEY_ENTER}
33	verify not selected value	xpath=//div[@id='DeskAndBillingScheduleAddAppointmentScheduleInfo']/div/div/div[4]/div/div/div/div	
34	click	id=DeskAndBillingScheduleAddAppointmentScheduleInfo_Notes	
35	type	id=DeskAndBillingScheduleAddAppointmentScheduleInfo_Notes	hbfitbsibi
36	verify not selected value	xpath=//div[@id='DeskAndBillingScheduleAddAppointmentScheduleInfo']/div/div/div[4]/div/div[3]/div/div	
37	click	id=DeskAndBillingScheduleAddAppointmentFinalize	
38	verify element not present	css=.ui-pnotify	
39	assert text	css=.schedule-menu-buttons-container:nth-child(1) > .btn-action	CheckIn
40	click	css=.schedule-menu-buttons-container:nth-child(1) > .btn-action	
41	assert text	css=.modal-title:nth-child(3)	CHECKIN
42	click	xpath=//label[contains(.,'Outro')]	
43	click	css=.col-md-12 > .form-manager-control > .styled > .label-right	
44	click	xpath=/html/body/div[21]/div/div/div[1]/div[3]/div/button[2]	
45	execute script	return document.querySelector("#cbe_1d153421 15f6c39da451e191a78dba90 > div > div > div.modal-content") != null	isElementPresent
46	if	${isElementPresent}	
47	assert text	xpath=//div[contains(text(),'Não é possível adicionar o local selecionado, uma ')]	Não é possível adicionar o local selecionado, uma vez que não está configurado para o centro de saúde/unidade operacional/unidade funcional da visita.
48	click	css=#cbe_4a96956 36d1cbdb031e280d441d115c8 .btn	
49	click	css=#s2id_FacilityId_search > .select2-choice	
50	type	xpath=//div[@id='select2-drop']/div/input	gabinetes
51	send keys	xpath=//div[@id='select2-drop']/div/input	${KEY_ENTER}
52	click	xpath=/html[1]/body[1]/div[21]/div[1]/div[1]/div[1]/div[2]/div[1]/div[1]/div[1]/div[1]/div[1]/div[1]/div[9]/div[2]/div[1]/div[1]/div[1]/div[1]/div[2]/div[1]/div[1]/div[2]/a[1]/span[1]	
53	type	xpath=//div[@id='select2-drop']/div/input	gabinete 505
54	send keys	xpath=//div[@id='select2-drop']/div/input	${KEY_DOWN}
55	send keys	xpath=//div[@id='select2-drop']/div/input	${KEY_ENTER}
56	click	xpath=/html[1]/body[1]/div[21]/div[1]/div[1]/div[1]/div[2]/div[1]/div[1]/div[1]/div[1]/div[1]/div[1]/div[9]/div[2]/div[1]/div[1]/div[1]/div[2]/div[1]/div[1]/div[2]/a[1]/span[1]	
57	type	xpath=//div[@id='select2-drop']/div/input	URG100
58	send keys	xpath=//div[@id='select2-drop']/div/input	${KEY_DOWN}
59	send keys	xpath=//div[@id='select2-drop']/div/input	${KEY_ENTER}
60	click	css=#cbe_c1767d2d5ff729f9532ddc238462afff .btn-action	
61	end		
100	verify element not present	css=.ui-pnotify-title	
101	execute script	var reverter = document.querySelector('.schedule-menu-buttons-container:nth-child(5) > .btn'); != null; if (reverter) { return true; } return false;	reverter
102	if	${reverter}	
103	assert text	css=.schedule-menu-buttons-container:nth-child(5) > .btn	Reverter
104	click	css=.schedule-menu-buttons-container:nth-child(5) > .btn	
105	click	xpath=//div[23]/div/div/div/div[3]/div/button[2]	
106	end		
107	assert text	xpath=//button[contains(text(),'Cancelar Visita')]	Cancelar Visita
108	click	xpath=//button[contains(text(),'Cancelar Visita')]	
109	pause	1000	
110	mouse down	css=#s2id_ReasonId > .select2-choice	
111	mouse up	id=select2-drop-mask	
112	click	xpath=/html[1]/body[1]/div[27]/ul[1]/li[4]/div[1]/div[1]	
113	click	css=#cbe_4b6324d23fe839ed83c2d8faa91ad067 .btn-action	
114	verify element not present	css=.ui-pnotify-title	
115	click	id=usermenu-link	
116	click	css=.fa-power-off	
117	close		

Figura 33 - Teste de Marcação Simples no Selenium IDE

7. *Pesquisa de Vagas*

Continuamos com a marcação de uma visita, mas realizada através de um método diferente, a pesquisa de vagas.

Para realizarmos a pesquisa de vagas, partindo do ecrã inicial, clicamos no ***Atendimento*** e, em seguida, no ícone relativo à pesquisa de vagas.

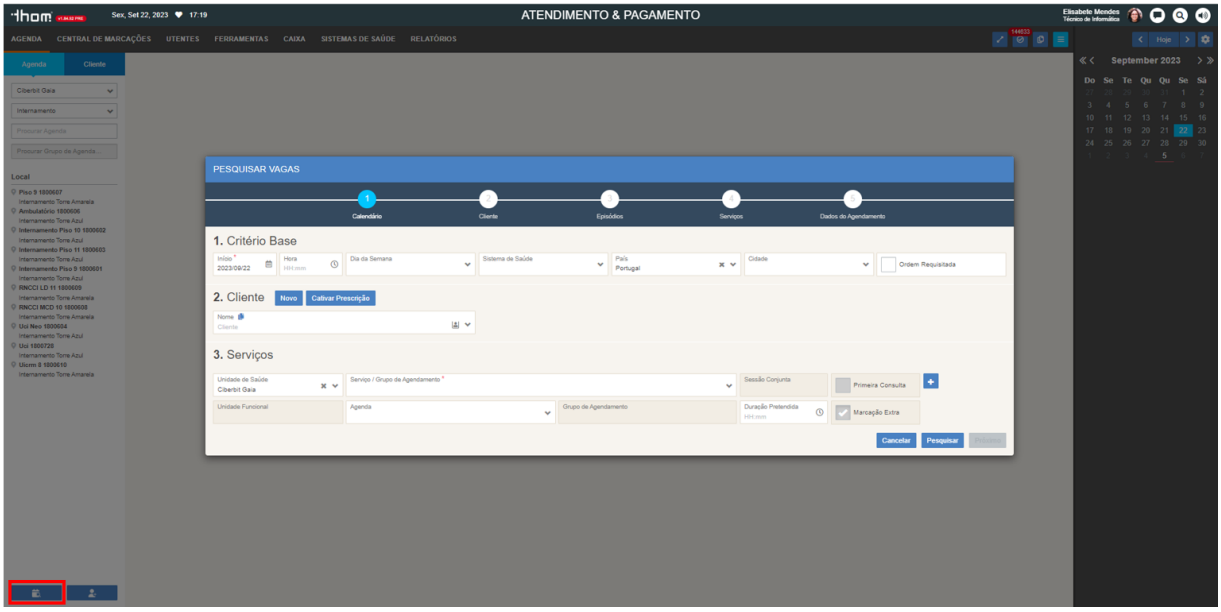


Figura 34- Pop-up relativo à Pesquisa de Vagas

Abrindo o ecrã de ***Pesquisar Vagas*** é necessário realizar o preenchimento de alguns campos:

- ***Cliente:*** Ana Margarida Rodrigues dos Santos Silva (Cliente criado no teste Criar Cliente).
- ***Serviços/Grupo de Agendamento***
    - Nesta situação como realizamos dois testes de pesquisa de vagas, uma relativa a uma consulta e outra relativa a um exame mostramos os valores com que preenchemos este campo em cada uma das situações.

Tabela 1- Grupo de Agendamento de cada caso de teste de Pesquisa de Vagas

	Consulta	**Exame**
Serviço/Grupo de Agendamento	Consulta de Fisiatria	ANGIO - TAC

Em seguida, clicamos no botão ***Pesquisar***, para que nos sejam fornecidas todas as vagas disponíveis. Temos de clicar numa das opções.

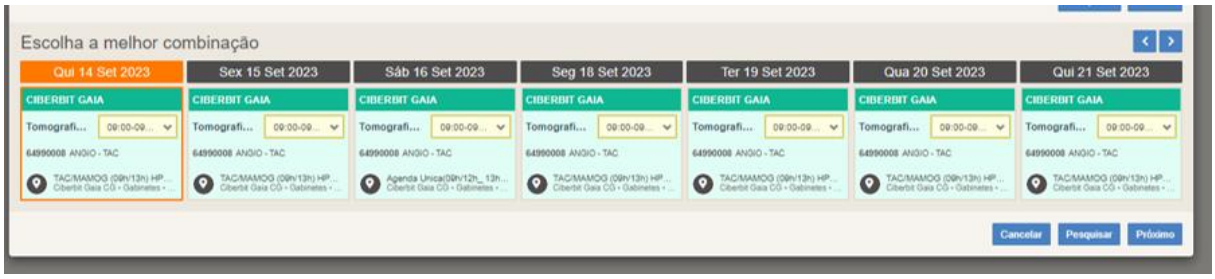


Figura 35- Opções de vagas para agendamento

Depois de selecionada a vaga, clicamos no botão ***Próximo*** para avançar os ecrãs ***Episódios*** e quando chegamos ao ecrã ***Dados de Agendamento***, clicamos no botão ***Finalizar***.

Aqui mais uma vez verificamos se não existem alertas nem mensagens de erro e ainda se a mensagem "Agendado com Sucesso" aparece no nosso ecrã, para termos a certeza de que a marcação foi realizada.

Mais uma vez realizamos a gravação de todo o teste e obtemos o conjunto de comandos presente na Figura 36.

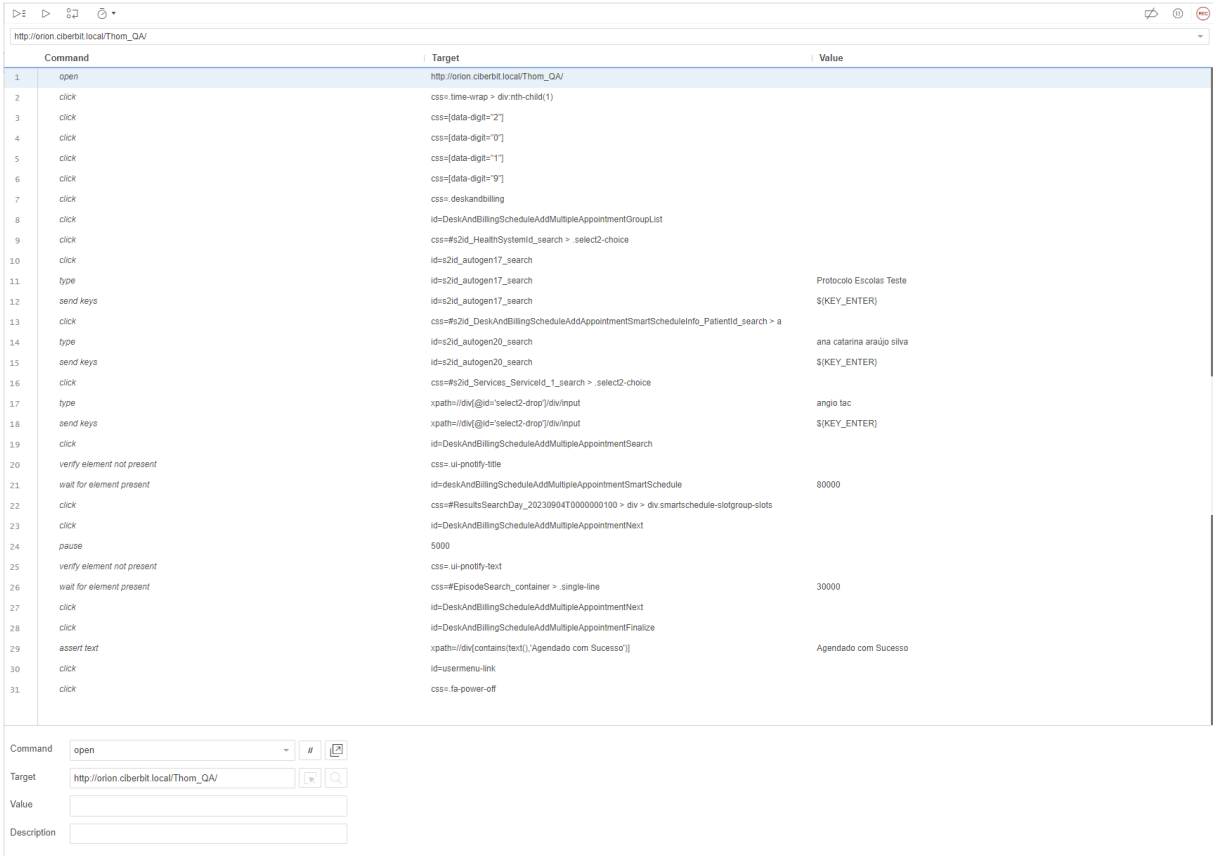


Figura 36- Teste de Pesquisa de Vagas em Selenium IDE

3.3 Implementação dos Testes em *Selenium WebDriver*

Depois de criar estes testes foi notório que havia parâmetros que podiam vir a ser usados noutros testes e ainda que podíamos ter uns testes mais simples a serem usados em outros mais complexos. Posto isto, era necessário perceber quais as capacidades que o *Selenium IDE* apresentava para corresponder a esta questão. Depois de alguma pesquisa e discussão com a empresa percebeu-se que o cenário ideal seria a leitura dos dados a serem utilizados a partir de um ficheiro *Excel* e uma outra opção seria a criação de variáveis.

Com o desenrolar do projeto e a constante criação de novos testes percebemos que num futuro próximo seria necessário utilizar variáveis no teste para inserir dados. Além disso, criar novas variáveis para, por exemplo, guardar um *click* ou até mesmo o valor de um teste para poder ser utilizado num outro teste. Verificamos que era possível fazer isso no *Selenium IDE* desde que os testes se encontrassem todos dentro da mesma *suite* de teste.

Outra abordagem que quisemos investigar passou pela possibilidade de fazer a leitura dos dados para preenchimento dos campos do teste a partir de um ficheiro *Excel*. Aqui as conclusões a que chegamos não foram tão positivas, uma vez que percebemos que no *Selenium IDE* não é imediato fazer leitura de dados de um ficheiro *Excel*. Já foi possível, mas com atualização de versões do programa perdeu essa funcionalidade e que, no momento atual, eram necessárias instalações de *plugins* específicos que não conseguimos implementar. Percebemos então que para termos o melhor dos dois mundos, isto é, conseguirmos criar variáveis e ao mesmo tempo fazer a leitura de outras variáveis através de ficheiros *Excel* só seria possível com a passagem dos testes gravados no *Selenium IDE* para o *Selenium WebDriver*, em linguagem C#, que é a linguagem utilizada nos desenvolvimentos da empresa.

No *Selenium IDE*, em cada um dos testes ou *suite* de testes conseguimos fazer *Export* e selecionar a linguagem para a qual queremos exportar os nossos testes. Como na empresa é utilizado C#, é essa a linguagem que iremos utilizar no nosso projeto. Como vemos na Figura 37, o documento exportado apresenta, em primeiro lugar todos os *packages* que são utilizados, sendo que com o desenrolar do código podem ser necessários outros mais específicos.

```
1   // Generated by Selenium IDE
2   using System;
3   using System.Collections.Generic;
4   using System.Threading;
5   using OpenQA.Selenium;
6   using NUnit.Framework;
7   using ExcelDataReader;
8   using System.IO;
9   using Selenium;
10  using NUnit.Framework.Interfaces;
11  using AberturaSubPaginas;
12
```

Figura 37- Packages utilizados nos testes

Passamos então para o restante corpo do código exportado e, tal como podemos ver na Figura 3, temos presente o [*TestFixture*] que é um conceito para testes *NUnit* e tem dentro dele os testes a serem executados, o [*SetUp*] que serve para configurar o estado inicial do teste e ainda o [*TearDown*], que tem a função de limpeza após a execução de cada teste.

No código relativo ao [*SetUp*], que acaba por ser comum a todos os testes que apresentamos anteriormente, iniciamos o *driver* e fazemos a leitura dos dados do *Excel* que iremos utilizar como variáveis no nosso teste. Neste caso temos o *url*, senha e utilizador.

Para a realização do *Login* na aplicação é necessário testar várias possibilidades, uma vez que temos diferentes cenários de teste desde falta de preenchimento de campos e dados de acesso incorretos. Como tal, no Anexo I podemos ver em cada linha do ficheiro *Excel* um caso de teste que cobrirá todos os cenários possíveis.

```
13  namespace AberturaPaginas
14  {
15
16      [TestFixture]
        0 referências
17      public class AberturaPginasTest
18      {
19          private IWebDriver driver;
            1 referência
20          public IDictionary<string, object> vars { get; private set; }
21          private IJavaScriptExecutor js;
22          private string utilizador;
23          private string senha;
24          private string url;
25
26          [SetUp]
            0 referências
27          public void SetUp()
28          {
29              driver = DriverHelper.customechromedriver();
30              driver.Manage().Timeouts().ImplicitWait = TimeSpan.FromSeconds(15);
31              js = (IJavaScriptExecutor)driver;
32              vars = new Dictionary<string, object>();
33              var caminhoArquivoExcel = FileHelper.GetURLFile("Passos.xlsx");
34              if (!string.IsNullOrEmpty(caminhoArquivoExcel))
35              {
36                  using (var stream = File.Open(caminhoArquivoExcel, FileMode.Open, FileAccess.Read))
37                  {
38                      using (var reader = ExcelReaderFactory.CreateReader(stream))
39                      {
40                          var dataSet = reader.AsDataSet();
41                          var worksheet = dataSet.Tables[0];
42
43                          utilizador = worksheet.Rows[1][1].ToString();
44                          senha = worksheet.Rows[1][2].ToString();
45                          url = worksheet.Rows[1][0].ToString();
46                      }
47                  }
48              }
49              else
50              {
51                  Console.WriteLine("Nenhum arquivo Excel encontrado.");
52              }
53          }
```

Figura 38- Código do [SetUp]

Passando agora para o [*TearDown*] vemos quais os comandos realizados no fim da execução dos testes de cada documento. Podemos ver que existe uma verificação do estado do teste e caso ele tenha falhado é realizado um *screenshot* e feito um *refresh* da página. Vemos ainda que o *screenshot* gerado será guardado numa pasta em formato *.png*, para conseguirmos ter acesso ao erro que está a originar a falha do teste.

```
[TearDown]
0 referências
public void OnTearDown()
{
    Console.WriteLine("GetEnvironment : " + Utils.GetEnvironmentName());

    if (TestContext.CurrentContext.Result.Outcome.Status == TestStatus.Failed)
    {
        CaptureScreenshot();
        driver.Navigate().Refresh();

        if (TestContext.CurrentContext.Result.Message.Contains("OpenQA.Selenium.JavaScriptException :"))
        {
            CaptureScreenshot();
            driver.Navigate().Refresh();
        }
    }
    driver.Quit();
}

2 referências
private void CaptureScreenshot()
{
    Screenshot screenshot = ((ITakesScreenshot)driver).GetScreenshot();
    string testName = TestContext.CurrentContext.Test.MethodName;
    string screenshotPath = $@"..\..\..\resources\screenshot_{testName}.png";
    screenshot.SaveAsFile(screenshotPath, ScreenshotImageFormat.Png);
}
```

Figura 39- Código do [TearDown]

Passamos então para os testes propriamente ditos e temos na Figura 40 dois testes de abertura de página de unidades operacionais. Este teste terá exatamente o mesmo comportamento dos comandos guardados no *Selenium IDE*, ou seja, realizará o *Login* na aplicação, fará os *clicks* até chegar à página pretendida e verifica se o título corresponde exatamente ao que nós queremos.

```
[Test]
0 referências
public void Ambulatorio()
{
    driver.Navigate().GoToUrl(url);
    driver.FindElement(By.Id("Username")).SendKeys(utilizador);
    driver.FindElement(By.Id("Password")).SendKeys(senha);
    driver.FindElement(By.Id("RegisterUser")).Click();
    Thread.Sleep(1500);
    driver.FindElement(By.CssSelector(".outpatient > div")).Click();
    Thread.Sleep(1500);
    Assert.That(driver.FindElement(By.CssSelector(".zone-title")).Text, Is.EqualTo("AMBULATÓRIO"));
    driver.FindElement(By.Id("usermenu-link")).Click();
    driver.FindElement(By.CssSelector(".fa-power-off")).Click();
}

[Test]
0 referências
public void Urgencia()
{
    driver.Navigate().GoToUrl(url);
    driver.FindElement(By.Id("Username")).SendKeys(utilizador);
    driver.FindElement(By.Id("Password")).SendKeys(senha);
    driver.FindElement(By.Id("RegisterUser")).Click();
    Thread.Sleep(1500);
    driver.FindElement(By.CssSelector(".urgency > div")).Click();
    Thread.Sleep(1500);
    Assert.That(driver.FindElement(By.CssSelector(".zone-title")).Text, Is.EqualTo("URGÊNCIA"));
    driver.FindElement(By.Id("usermenu-link")).Click();
    driver.FindElement(By.CssSelector(".fa-power-off")).Click();
}
```

Figura 40- Teste de abertura de Unidades Operacionais em Selenium WebDriver

Quando passamos para testes mais complexos e de maior dimensão foi necessário fazer algumas alterações na parte inicial do código, principalmente na leitura das variáveis do *Excel*. Como para cada teste necessitávamos de dados diferentes, criamos uma folha *Excel* referente a cada teste. Na Figura 41 podemos ver os comandos de leitura e extração dos dados de cada uma das variáveis criadas para o teste de Criar Cliente. Podemos ver como é elaborada a folha Excel deste teste no Anexo II.

```
string Url = planilha.Rows[row][0]?.ToString() ?? string.Empty;
string Username = planilha.Rows[row][1]?.ToString() ?? string.Empty;
string Password = planilha.Rows[row][2]?.ToString() ?? string.Empty;
string Cliente = planilha.Rows[row][3]?.ToString() ?? string.Empty;
string Data_Nascimento = planilha.Rows[row][4]?.ToString() ?? string.Empty;
string Country = planilha.Rows[row][5]?.ToString() ?? string.Empty;
string IdNumber = planilha.Rows[row][6]?.ToString() ?? string.Empty;
string TaxNumber = planilha.Rows[row][7]?.ToString() ?? string.Empty;
string Phone = planilha.Rows[row][8]?.ToString() ?? string.Empty;
string Email = planilha.Rows[row][9]?.ToString() ?? string.Empty;
string Profissao = planilha.Rows[row][10]?.ToString() ?? string.Empty;
string SS_1 = planilha.Rows[row][11]?.ToString() ?? string.Empty;
string NBenificiario_1 = planilha.Rows[row][12]?.ToString() ?? string.Empty;
string SS_2 = planilha.Rows[row][13]?.ToString() ?? string.Empty;
string NBenificiario_2 = planilha.Rows[row][14]?.ToString() ?? string.Empty;
```

Figura 41- Variáveis do teste Criar Cliente

Depois de definidas todas as variáveis, chega o momento de verificarmos o código do teste que foi exportado pelo *Selenium IDE* e realizar as alterações necessárias. Logo numa fase inicial inserimos todas as variáveis acima criadas como parâmetros de entrada. Em seguida, alteramos nos campos de preenchimento relativos a essas variáveis o valor que tínhamos no caso de teste pela variável e inserimos

esperas implícitas onde necessário, para que o teste respeite o tempo de processamento do ecrã e não falhe por incompatibilidade de tempos. Efetuadas todas estas tarefas obtemos o código presente na Figura 42.

```
[Test, TestCaseSource(nameof(GetCredentialsFromExcel))]
0 | 0 referências
public void CriarCliente(string Url, string Username, string Password, string Cliente, string Data_Nascimento, string Country, string IdNumber, string TaxNumber,
    string Phone, string Email, string Profissao, string SS_1, string NBenificiario_1, string SS_2, string NBenificiario_2)
{
    driver.Navigate().GoToUrl(Url);
    driver.FindElement(By.Id("Username")).Click();
    driver.FindElement(By.Id("Username")).SendKeys(Username);
    driver.FindElement(By.Id("Password")).Click();
    driver.FindElement(By.Id("Password")).SendKeys(Password);
    driver.FindElement(By.Id("RegisterUser")).Click();
    Thread.Sleep(5000);
    driver.FindElement(By.CssSelector(".fa-search")).Click();
    vars["WindowHandles"] = driver.WindowHandles;
    driver.FindElement(By.Id("searchBarPersonCreate")).Click();
    vars["win9129"] = waitForWindow(2000);
    driver.SwitchTo().Window(vars["win9129"].ToString());
    driver.FindElement(By.Id("person-demographics-create")).Click();
    driver.FindElement(By.CssSelector("#s2id_TitleId > .select2-choice")).Click();
    driver.FindElement(By.XPath("//body/div[@id=\'select2-drop\']/ul[@id=\'select2-results-1\']/li[1]/div[1]/div[1]")).Click();
    driver.FindElement(By.Id("FullName")).Click();
    driver.FindElement(By.Id("FullName")).SendKeys(Cliente);
    driver.FindElement(By.Id("BirthDate")).Click();
    driver.FindElement(By.Id("BirthDate")).SendKeys(Data_Nascimento);
    driver.FindElement(By.Id("BirthDate")).SendKeys(Keys.Enter);
    {
        var element = driver.FindElement(By.CssSelector("#s2id_BirthCountryCode > .select2-choice"));
        Actions builder = new Actions(driver);
        builder.MoveToElement(element).ClickAndHold().Perform();
    }
    {
        var element = driver.FindElement(By.Id("select2-drop-mask"));
        Actions builder = new Actions(driver);
        builder.MoveToElement(element).Release().Perform();
    }
    driver.FindElement(By.Id("s2id_autogen2_search")).Click();
    driver.FindElement(By.Id("s2id_autogen2_search")).SendKeys(Country);
    driver.FindElement(By.Id("s2id_autogen2_search")).SendKeys(Keys.Enter);
    driver.FindElement(By.Id("Phone")).Click();
    driver.FindElement(By.Id("Phone")).SendKeys(Phone);
    driver.FindElement(By.Id("Email")).Click();
    driver.FindElement(By.Id("Email")).SendKeys(Email);
    driver.FindElement(By.Id("Email")).SendKeys(Keys.Enter);
    driver.FindElement(By.CssSelector(".form-manager-group-row:nth-child(4) .form-manager-group-title")).Click();
    driver.FindElement(By.CssSelector(".active > .form-manager-group-cell")).Click();
    driver.FindElement(By.Id("IdNumber")).Click();
    driver.FindElement(By.Id("IdNumber")).SendKeys(IdNumber);
    driver.FindElement(By.Id("TaxNumber")).Click();
    driver.FindElement(By.Id("TaxNumber")).SendKeys(TaxNumber);
    driver.FindElement(By.CssSelector("#NHSNumberValidationOff .off-label")).Click();
    driver.FindElement(By.CssSelector("#s2id_IgnoreReasonForm_IgnoreReasonId > .select2-choice")).Click();
    driver.FindElement(By.XPath("//body/div[@id=\'select2-drop\']/ul[@id=\'select2-results-347\']/li[1]/div[1]/div[1]")).Click();
    driver.FindElement(By.XPath("//label[contains(.,\'Não voltar a pedir o preenchimento deste campo\')]")).Click();
    driver.FindElement(By.CssSelector(".btn-warning")).Click();
    {
        WebDriverWait wait = new WebDriverWait(driver, System.TimeSpan.FromSeconds(30));
        wait.Until(driver => driver.FindElements(By.CssSelector(".form-manager-group-row:nth-child(7) .form-manager-group-title")).Count > 0);
    }
    driver.FindElement(By.XPath("//body/div[@id=\'ajax-wrapper\']/div[@id=\'wrapper\']/div[3]/div[1]/div[1]/div[3]/div[2]/div[1]/div[2]/div[2]/div[2]/div[7]/div[1]" +
        "/div[2]/div[1]/div[1]/div[1]/div[1]/a[1]")).Click();
    driver.FindElement(By.Id("s2id_autogen11_search")).SendKeys(Profissao);
    driver.FindElement(By.Id("s2id_autogen11_search")).SendKeys(Keys.Enter);
    driver.FindElement(By.CssSelector(".styled-small:nth-child(1) > .label-right")).Click();
    driver.FindElement(By.Id("person-demographics-tab-healthsystems")).Click();
    driver.FindElement(By.LinkText("Adicionar")).Click();
    driver.FindElement(By.Id("s2id_autogen17_search")).SendKeys(SS_1);
    Thread.Sleep(1000);
    driver.FindElement(By.Id("s2id_autogen17_search")).SendKeys(Keys.Enter);
    Thread.Sleep(1300);
    driver.FindElement(By.Id("HealthSystems_PolicyNumber_1")).SendKeys(NBenificiario_1);
    Thread.Sleep(1000);
    driver.FindElement(By.Id("HealthSystems_ValidFrom_1")).Click();
    driver.FindElement(By.CssSelector("tr:nth-child(1) > .day:nth-child(6)")).Click();
    driver.FindElement(By.CssSelector(".cb-ct-rp-item-row-last:nth-child(1) > .form-manager-checkable-input:nth-child(8) .styled > .label-right")).Click();
    driver.FindElement(By.LinkText("Adicionar")).Click();
    Thread.Sleep(1000);
    driver.FindElement(By.XPath("/html[1]/body[1]/div[25]/div[1]/input[1]")).SendKeys(SS_2);
    Thread.Sleep(1000);
    driver.FindElement(By.XPath("/html[1]/body[1]/div[25]/div[1]/input[1]")).SendKeys(Keys.Enter);
    Thread.Sleep(2500);
    driver.FindElement(By.Id("HealthSystems_PolicyNumber_2")).SendKeys(NBenificiario_2);
    driver.FindElement(By.Id("HealthSystems_ValidFrom_2")).Click();
    driver.FindElement(By.CssSelector("tr:nth-child(1) > .day:nth-child(6)")).Click();
    Thread.Sleep(2000);
    driver.FindElement(By.Id("healthSystemCardList")).Click();
    driver.FindElement(By.Id("person-demographics-save")).Click();
    {
        var elements = driver.FindElements(By.CssSelector(".ui-pnotify-title"));
        Assert.True(elements.Count == 0);
    }
    {
        var elements = driver.FindElements(By.CssSelector(".ui-pnotify:nth-child(71) .ui-pnotify-text"));
        Assert.True(elements.Count == 0);
    }
}
```

Figura 42- Teste Criar Cliente em Selenium WebDriver

Passando agora para o próximo teste relativo à criação de um sistema de saúde, podemos ver que o processo de passagem para *Selenium WebDriver* funciona exatamente da mesma forma que o teste anterior. Começamos por alterar as variáveis que são lidas através de um ficheiro *Excel*, cujo preenchimento está explícito no Anexo III. Esta variáveis são guardadas para utilizar no teste.

```
// Ler as credenciais de login da linha atual
string url = planilha.Rows[row][0]?.ToString() ?? string.Empty;
string username = planilha.Rows[row][1]?.ToString() ?? string.Empty;
string password = planilha.Rows[row][2]?.ToString() ?? string.Empty;
string ss = planilha.Rows[row][3]?.ToString() ?? string.Empty;
string codigo = planilha.Rows[row][4]?.ToString() ?? string.Empty;
string tipo = planilha.Rows[row][5]?.ToString() ?? string.Empty;
```

Figura 43- Variáveis do teste Criar Sistema de Saúde

Em seguida, fazemos as alterações necessárias no teste, como a introdução dos parâmetros de entrada, sendo estes as variáveis mencionadas na Figura 43. Posteriormente, adicionamos ao restante código essas mesmas variáveis no local do código onde queremos que elas sejam chamadas. Por fim, adicionamos esperas implícitas onde é necessário para o bom funcionamento do teste e, caso necessário, alteramos algum seletor que não seja corretamente reconhecido, ficando o teste final com o código apresentado na Figura 44.

```
[Test, TestCaseSource(nameof(GetCredentialsFromExcel))]
0 referências
public void criarSistemaSade(string url, string username, string password, string ss, string codigo, string tipo)
{
    driver.Navigate().GoToUrl(url);
    driver.FindElement(By.Id("Username")).Click();
    driver.FindElement(By.Id("Username")).SendKeys(username);
    driver.FindElement(By.Id("Password")).Click();
    driver.FindElement(By.Id("Password")).SendKeys(password);
    driver.FindElement(By.Id("RegisterUser")).Click();
    Thread.Sleep(5000);
    driver.FindElement(By.LinkText("Configuração")).Click();
    Thread.Sleep(1000);
    driver.FindElement(By.CssSelector("li:nth-child(4) > .no-selection")).Click();
    Thread.Sleep(1500);
    driver.FindElement(By.LinkText("Adicionar")).Click();
    Thread.Sleep(1000);
    var sistema_saúde = "Protocolo Escolas Teste";
    driver.FindElement(By.Id("HealthSystem")).SendKeys(sistema_saúde);
    driver.FindElement(By.Id("Code")).Click();
    driver.FindElement(By.Id("Code")).SendKeys(codigo);
    Thread.Sleep(1000);
    driver.FindElement(By.XPath("//body/div[@id=\'ajax-wrapper\']/div[@id=\'wrapper\']/div[4]/div[1]/div[1]/div[1]/div[1]/div[1]/div[2]/div[1]/div[3]/" +
        "div[1]/div[1]/a[1]")).Click();
    Thread.Sleep(1000);
    driver.FindElement(By.XPath("//body/div[@id=\'select2-drop\']/ul[@id=\'select2-results-12\']/li[1]/div[1]/div[1]")).Click();
    driver.FindElement(By.XPath("//body/div[@id=\'ajax-wrapper\']/div[@id=\'wrapper\']/div[4]/div[1]/div[1]/div[1]/div[1]/div[1]/div[3]/div[2]/div[1]/div[1]/" +
        "div[1]/a[1]/abbr[1]")).Click();
    driver.FindElement(By.XPath("//body/div[@id=\'ajax-wrapper\']/div[@id=\'wrapper\']/div[4]/div[1]/div[1]/div[1]/div[1]/div[1]/div[5]/div[2]/div[1]/div[1]/" +
        "div[1]/div[1]/div[1]/a[1]")).Click();
    Thread.Sleep(1000);
    driver.FindElement(By.Id("s2id_autogen17_search")).SendKeys(tipo);
    Thread.Sleep(1500);
    driver.FindElement(By.Id("s2id_autogen17_search")).SendKeys(Keys.Enter);
    driver.FindElement(By.Id("HealthSystemForm_form_save")).Click();
    Assert.That(driver.FindElement(By.CssSelector(".modal-title:nth-child(3)")).Text, Is.EqualTo("ERRO"));
    driver.FindElement(By.CssSelector("[data-action=\"close\"]")).Click();
    driver.FindElement(By.Id("usermenu-link")).Click();
    driver.FindElement(By.CssSelector(".fa-power-off")).Click();
    Assert.That(driver.FindElement(By.CssSelector(".modal-title:nth-child(3)")).Text, Is.EqualTo("ALTERAÇÕES NÃO GRAVADAS!"));
    driver.FindElement(By.CssSelector("[data-action=\"ok\"]")).Click();
}
```

Figura 44- Teste Criar Sistema de Saúde em Selenium WebDriver

Terminado mais um teste com sucesso chega o momento de passarmos o teste de Pesquisa de Vagas para o *Selenium WebDriver*. Tal como acontece nos testes anteriores, o primeiro passo é definir as variáveis que são lidas no ficheiro *Excel*. Na secção anterior vimos que nesta situação estamos perante

dois casos de teste, um onde realizamos a pesquisa para uma consulta e outro onde realizamos a pesquisa para um exame. Para que este teste seja executado duas vezes e com os dados diferentes, o ficheiro Excel terá um preenchimento diferente do que foi visto até agora e podemos ver no Anexo IV que temos duas linhas da folha preenchidas, sendo que cada linha corresponde a um caso de teste.

```
string url = planilha.Rows[row][0]?.ToString() ?? string.Empty;
string username = planilha.Rows[row][1]?.ToString() ?? string.Empty;
string password = planilha.Rows[row][2]?.ToString() ?? string.Empty;
string cliente = planilha.Rows[row][3]?.ToString() ?? string.Empty;
string ss = planilha.Rows[row][4]?.ToString() ?? string.Empty;
string servico = planilha.Rows[row][6]?.ToString() ?? string.Empty;
string unidadefuncional = planilha.Rows[row][7]?.ToString() ?? string.Empty;
string grupoagendamento = planilha.Rows[row][5]?.ToString() ?? string.Empty;
```

Figura 45- Variáveis do teste Pesquisa de Vagas

No que diz respeito ao código do teste em si, podemos ver que também tem alguns detalhes. Além da inserção dos parâmetros de entrada e das esperas implícitas foram criados alguns *If*, no campo de preenchimento Unidade Funcional, Grupo de Agendamento e Serviço. Foi necessário realizar esta alteração no código porque existem algumas Unidades Funcionais que quando preenchidas preenchem automaticamente os campos do Grupo de Agendamento e Serviço, mas existem outras situações em que ambos os campos ou apenas um deles não é preenchido.

```
[Test, TestCaseSource(nameof(GetCredentialsFromExcel))]
0 | 0 referências
public void PesquisarVagasTest(string url, string username, string password,string cliente, string ss, string servico, string unidadefuncional,
    string grupoagendamento)
{
    driver.Navigate().GoToUrl(url);
    driver.FindElement(By.Id("Username")).Click();
    driver.FindElement(By.Id("Username")).SendKeys(username);
    driver.FindElement(By.Id("Password")).Click();
    driver.FindElement(By.Id("Password")).SendKeys(password);
    driver.FindElement(By.Id("RegisterUser")).Click();
    Thread.Sleep(3000);
    driver.FindElement(By.CssSelector(".deskandbilling")).Click();
    driver.FindElement(By.Id("DeskAndBillingScheduleAddMultipleAppointmentGroupList")).Click();
    Thread.Sleep(5000);
    driver.FindElement(By.XPath("//*[@id=\'s2id_HealthSystemId_search\']/a")).Click();
    driver.FindElement(By.Id("s2id_autogen17_search")).SendKeys(ss);
    Thread.Sleep(1000);
    driver.FindElement(By.Id("s2id_autogen17_search")).SendKeys(Keys.Enter);
    Thread.Sleep(1000);
    driver.FindElement(By.CssSelector("#s2id_DeskAndBillingScheduleAddAppointmentSmartScheduleInfo_PatientId_search > a")).Click();
    driver.FindElement(By.Id("s2id_autogen20_search")).SendKeys(cliente);
    Thread.Sleep(500);
    driver.FindElement(By.Id("s2id_autogen20_search")).SendKeys(Keys.Enter);

    if (unidadefuncional != null && !string.IsNullOrEmpty(unidadefuncional))
    {
        driver.FindElement(By.XPath("//*[@id=\"s2id_Services_HealthUnitId_1_search\"]/a"))
              .Click();
        driver.FindElement(By.Id("s2id_autogen26_search")).SendKeys(unidadefuncional);
        Thread.Sleep(1000);
        driver.FindElement(By.Id("s2id_autogen26_search")).SendKeys(Keys.Enter);
    }

    if (grupoagendamento != null && !string.IsNullOrEmpty(grupoagendamento))
    {
        driver.FindElement(By.XPath("//*[@id=\"s2id_Services_HealthUnitIdFunctional_1_search\"]/a"))
              .Click();
        driver.FindElement(By.Id("s2id_autogen28_search")).SendKeys(grupoagendamento);
        Thread.Sleep(1000);
        driver.FindElement(By.Id("s2id_autogen28_search")).SendKeys(Keys.Enter);
    }
    Thread.Sleep(1000);
    if (servico != null && !string.IsNullOrEmpty(servico))
    {
        driver.FindElement(By.XPath("//*[@id=\"s2id_Services_ServiceId_1_search\"]/a")).Click();
        driver.FindElement(By.XPath("//div[@id='select2-drop']/div/input")).SendKeys(servico);
        Thread.Sleep(1000);
        driver.FindElement(By.XPath("//div[@id='select2-drop']/div/input")).SendKeys(Keys.Enter);
    }

    driver.FindElement(By.Id("DeskAndBillingScheduleAddMultipleAppointmentSearch")).Click();
    Thread.Sleep(1500);
    driver.FindElement(By.XPath("//body[1]/div[15]/div[2]/div[2]/div[1]/div[3]/div[1]/div[1]/div[1]/div[2]/div[1]/div[2]/div[1]/div[3]/table[1]/tbody[1]/" +
        "tr[1]/td[6]/div[1]")).Click();
    Thread.Sleep(1500);
    driver.FindElement(By.Id("DeskAndBillingScheduleAddMultipleAppointmentNext")).Click();
    Thread.Sleep(TimeSpan.FromSeconds(2));
    var Episode = driver.FindElements(By.XPath("//div[@id='EpisodeSearch']"));
    if (Episode.Count > 0)
    {
        driver.FindElement(By.Id("DeskAndBillingScheduleAddMultipleAppointmentNext")).Click();
        Thread.Sleep(TimeSpan.FromSeconds(4));
    }
    bool alertaVisivel = driver.FindElements(By.CssSelector(".modal-title:nth-child(3)")).Count > 0;

    // Verificar se aparece o alerta
    if (alertaVisivel)
    {
        driver.FindElement(By.CssSelector(".modal-title:nth-child(3)")).Text.Should().Be("CONFIRMAÇÃO");
        driver.FindElement(By.CssSelector(".btn-warning")).Click();
    }
    Thread.Sleep(2500);
    driver.FindElement(By.Id("DeskAndBillingScheduleAddMultipleAppointmentFinalize")).Click();
    Thread.Sleep(1500);
    Assert.That(driver.FindElement(By.XPath("//div[contains(text(),\'Agendado com Sucesso\')]")).Text, Is.EqualTo("Agendado com Sucesso"));
    driver.FindElement(By.Id("usermenu-link")).Click();
    driver.FindElement(By.CssSelector(".fa-power-off")).Click();

    ClearPageState();
}
```

Figura 46- Teste Pesquisa de Vagas em Selenium WebDriver

No que diz respeito à abertura de menus realizaram-se pequenas alterações no código relativo ao [*SetUp*] e [*TearDown*], uma vez que em cada um dos ficheiros, referentes a cada uma das áreas (Atendimento, Logística, Gestão, Farmácia, BI, Configuração) tem um número de testes bastante elevado e a realização de *login* e *logout* ao fim de cada teste fazia com que a execução dos testes fosse bastante mais demorada e uma vez que já estávamos dentro do menu que íamos usar para percorrer todas as opções não fazia sentido estar a sair da aplicação. Foi então que passamos a fazer *login* apenas uma vez no início do

conjunto de testes e *logout* da mesma forma, ou seja, só no final do conjunto de testes. Ficamos então com o código representado na Figura 47.

```
[TestFixture]
0 referências
public class BILoginUncTest
{
    private IWebDriver driver;
    1 referência
    public IDictionary<string, object> vars { get; private set; }
    private IJavaScriptExecutor js;
    private string utilizador;
    private string senha;
    private string url;

    [OneTimeSetUp]
    0 referências
    public void OneTimeSetUp()
    {

        Utils.ShowPipelineEnvironment();

        driver = DriverHelper.customechromedriver();
        driver.Manage().Timeouts().ImplicitWait = TimeSpan.FromSeconds(20);
        js = (IJavaScriptExecutor)driver;
        vars = new Dictionary<string, object>();
        var caminhoArquivoExcel = FileHelper.GetEnvironmentFile("Passos.xlsx");
        using (var stream = File.Open(caminhoArquivoExcel, FileMode.Open, FileAccess.Read))
        {
            using (var reader = ExcelReaderFactory.CreateReader(stream))
            {
                var workbook = reader.AsDataSet();
                var worksheet = workbook.Tables[1];
                utilizador = worksheet.Rows[1][1].ToString();
                senha = worksheet.Rows[1][2].ToString();
                url = worksheet.Rows[1][0].ToString();
            }
        }

        Login(url, utilizador, senha);
    }

    [OneTimeTearDown]
    0 referências
    public void OneTimeTearDown()
    {
        Logout();
        driver.Quit();
    }
```

Figura 47- Código [OneTimeSetUp] e [OneTimeTearDown]

Como continuávamos a querer que fossem realizadas algumas tarefas no fim de cada um dos testes, como o caso de verificar se o teste falhava e no caso de falhar fazer uma captura de ecrã para depois percebermos o motivo da falha, inserimos na mesma um [*TearDown*] que seguia o conjunto de comandos apresentado na Figura 48.

```
[TearDown]
0 referências
public void OnTearDown()
{
    Console.WriteLine("GetEnvironment : " + Utils.GetEnvironmentName());
    if (TestContext.CurrentContext.Result.Outcome.Status == TestStatus.Failed)
    {
        CaptureScreenshot();
        driver.Navigate().Back();
        Thread.Sleep(TimeSpan.FromSeconds(3));
        By elementoSelector = By.CssSelector(".errorscreen-page-title");
        IReadOnlyCollection<IWebElement> elementosErro = driver.FindElements(elementoSelector);
        if (elementosErro.Count > 0)
        {
            driver.FindElement(By.CssSelector(".errorscreen-home-link")).Click();
            Thread.Sleep(TimeSpan.FromSeconds(5));
            driver.FindElement(By.XPath("//span[contains(.,\'Business Intelligence\')]")).Click();
            Thread.Sleep(TimeSpan.FromSeconds(2));
        }
    }
}

1 referência
private void CaptureScreenshot()
{

    #region OLD
    //Screenshot screenshot = ((ITakesScreenshot)driver).GetScreenshot();
    //string testName = TestContext.CurrentContext.Test.MethodName;
    //string screenshotPath = $@"..\..\..\resources\screenshot_{testName}.png";
    //screenshot.SaveAsFile(screenshotPath, ScreenshotImageFormat.Png);
    //TestContext.AddTestAttachment(screenshotPath);
    #endregion

    Utils.CaptureScreenshot((ITakesScreenshot)driver);
}
```

Figura 48- Código [TearDown]

Uma vez que temos um número elevado de testes, devido às imensas opções que temos em cada um dos menus, vou mostrar apenas um exemplo de dois testes desta abertura de menus e todos os outros testes se comportam da mesma forma.

```
119         [Test]
            0 | 0 referências
120         public void bIAbrirFMensalpDia()
121         {
122             driver.FindElement(By.XPath("//span[contains(.,'Faturação Mensal por Dia')]")).Click();
123             WebDriverWait wait = new WebDriverWait(driver, TimeSpan.FromSeconds(50));
124             wait.Until(driver => (bool)((IJavaScriptExecutor)driver).ExecuteScript("return jQuery.active == 0"));
125             Assert.That(driver.FindElement(By.CssSelector(".page-title-content")).Text, Is.EqualTo("FATURAÇÃO MENSAL POR DIA"));
126         }
127
128         [Test]
            0 | 0 referências
129         public void bIAbrirHonorarios()
130         {
131             WebDriverWait wait = new WebDriverWait(driver, TimeSpan.FromSeconds(50));
132             wait.Until(driver => (bool)((IJavaScriptExecutor)driver).ExecuteScript("return jQuery.active == 0"));
133             driver.FindElement(By.XPath("//span[contains(.,'Honorários')]")).Click();
134             wait.Until(driver => (bool)((IJavaScriptExecutor)driver).ExecuteScript("return jQuery.active == 0"));
135             Assert.That(driver.FindElement(By.CssSelector(".page-title-content")).Text, Is.EqualTo("HONORÁRIOS"));
136         }
137
```

Figura 49- Teste de Abertura de Menus em Selenium WebDriver

Como podemos ver o que altera neste caso é o comando do *click*, onde muda o seletor que será clicado e ainda no comando *assert*, onde temos de alterar para o valor que o título da página apresenta em cada um dos testes.

4. RESULTADOS OBTIDOS

Depois de termos todos os testes acima mencionados a serem executados corretamente, chegou o momento de incorporá-los no processo de integração contínua. A empresa utiliza o *GitLab* e foi aí que criamos um repositório onde inserimos todo o nosso projeto de testes. Em seguida, criamos uma *pipeline* no *Jenkins* ligada a este repositório e começamos a correr diversos *builds* da mesma inúmeras vezes para verificarmos os resultados que obtínhamos com os testes criados.

A *pipeline* foi configurada de forma a dar-nos um relatório de testes bastante completo, para que permitisse tirar conclusões mais válidas e fundamentadas. No relatório de testes conseguimos ver os resultados de cada um dos *builds* ocorridos. Na Figura 50 temos os resultados dos últimos 20 *builds* a serem executados, indo desde o 57 até ao 75. Podemos ver que nos 3 últimos todos os testes passaram, do 65 ao 67 todos os testes falharam e nos restantes houve situações em que apenas alguns dos testes falharam.

Na primeira linha conseguimos ver o estado de *namespace*, de modo geral se passaram ou falharam e expandindo conseguimos ver em qual dos conjuntos de testes aconteceram os erros, por exemplo, no *build* 75 conseguimos ver que foi na ***LogisticaLoginUncTest*** que houve testes a falharem.

Chart	Package/Class/Testmethod	Passed	Transitions	78	77	76	75	74	73	72	71	70	67	66	65	64	63	62	61	60	59	58	57
☐	AberturaSubPaginas	30% (81%)	7	PASSED	PASSED	PASSED	FAILED	PASSED	FAILED	PASSED	FAILED	FAILED	FAILED	FAILED	FAILED	PASSED	FAILED	FAILED	FAILED	FAILED	FAILED	FAILED	FAILED
☐	AtAberturaSubPginasTest	80% (80%)	4	PASSED	PASSED	PASSED	PASSED	PASSED	FAILED	PASSED	PASSED	PASSED	FAILED	FAILED	FAILED	PASSED	PASSED	PASSED	PASSED	PASSED	PASSED	PASSED	PASSED
☐	BlLoginUncTest	85% (85%)	2	PASSED	PASSED	PASSED	PASSED	PASSED	PASSED	PASSED	PASSED	PASSED	FAILED	FAILED	FAILED	PASSED	PASSED	PASSED	PASSED	PASSED	PASSED	PASSED	PASSED
☐	ConfLoginUncTest	65% (83%)	3	PASSED	PASSED	PASSED	PASSED	PASSED	PASSED	PASSED	FAILED	FAILED	FAILED	FAILED	FAILED	PASSED	PASSED	PASSED	PASSED	PASSED	PASSED	FAILED	FAILED
☐	FarmaciaLoginUncTest	85% (85%)	2	PASSED	PASSED	PASSED	PASSED	PASSED	PASSED	PASSED	PASSED	PASSED	FAILED	FAILED	FAILED	PASSED	PASSED	PASSED	PASSED	PASSED	PASSED	PASSED	PASSED
☐	GestaoLoginUncTest	60% (60%)	4	PASSED	PASSED	PASSED	PASSED	PASSED	PASSED	PASSED	PASSED	PASSED	FAILED	FAILED	FAILED	PASSED	FAILED	FAILED	FAILED	FAILED	FAILED	PASSED	PASSED
☐	LogisticaLoginUncTest	80% (84%)	4	PASSED	PASSED	PASSED	FAILED	PASSED	PASSED	PASSED	PASSED	PASSED	FAILED	FAILED	FAILED	PASSED	PASSED	PASSED	PASSED	PASSED	PASSED	PASSED	PASSED

Figura 50- Resultado de Teste de vários builds

Se queremos um relatório ainda mais pormenorizado, vamos a este *build* e, mais concretamente ao retângulo vermelho que nos apresenta o estado *Failed* e clicamos sobre ele, abrindo assim um relatório pormenorizado que nos indica o número total de testes e destes quantos é que falharam e quantos foram bem-sucedidos. Além desta informação ainda nos indica quais os testes que estiveram sujeitos a falha e ainda nos indica valores de duração, desde duração total do *build* como duração de cada um dos documentos de teste. Segundo a Figura 51 conseguimos perceber que além da duração, também é indicado o número tanto de *Fail* como de *Pass, Skip* e *Total* de testes.

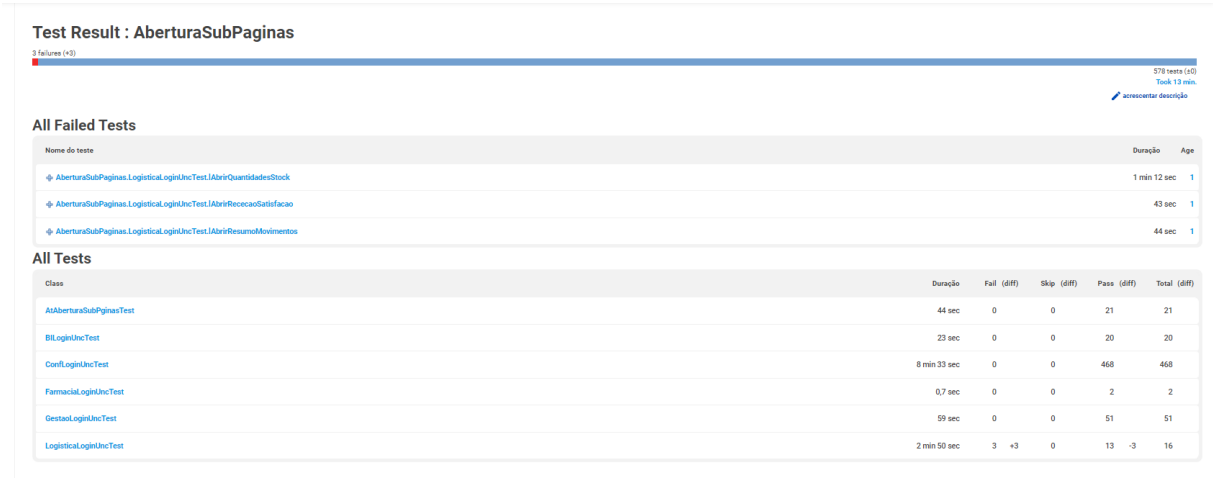


Figura 51- Resultado de teste de um build específico

Dentro da secção *All Failed Tests* temos todos os testes que falharam num *build* e se acedermos a um desses testes será aberto um ecrã com uma descrição pormenorizada da falha, permitindo-nos perceber em que linha aconteceu e qual o motivo.

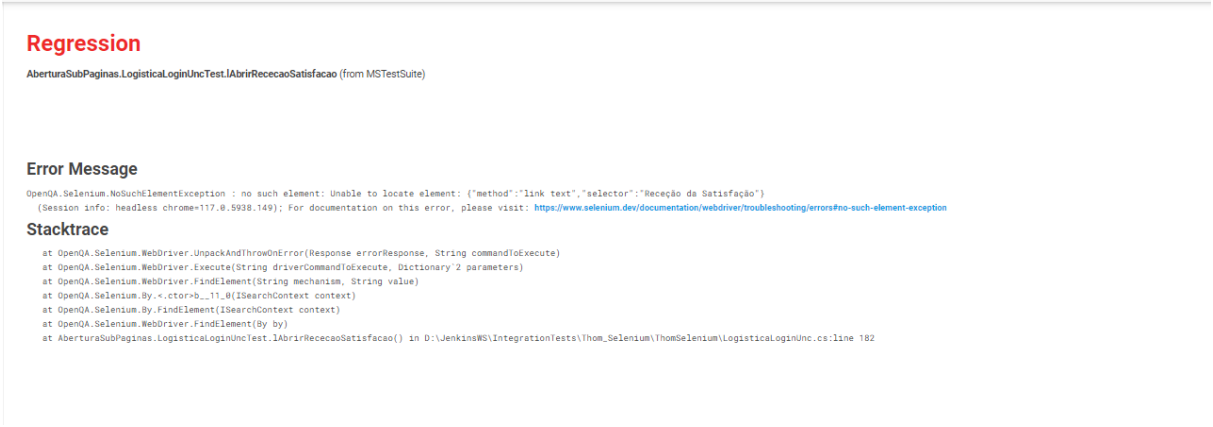


Figura 52- Descrição da falha do teste

Depois de analisados todos os resultados que a *pipeline* nos apresenta em casos onde existem falhas, chega o momento de analisarmos o último *build* executado, que diz respeito ao estado final do projeto, com todas as melhorias que vimos necessidade em realizar. Como já visto anteriormente, neste *build* não existia presença de falhas. Como isto acontece não teremos nenhuma das secções nem ecrãs descritos anteriormente que estavam relacionados às falhas de testes, sendo então que a principal análise que iremos fazer aqui é acerca da duração de execução dos testes. Podemos ver então estes resultados na Figura 53.

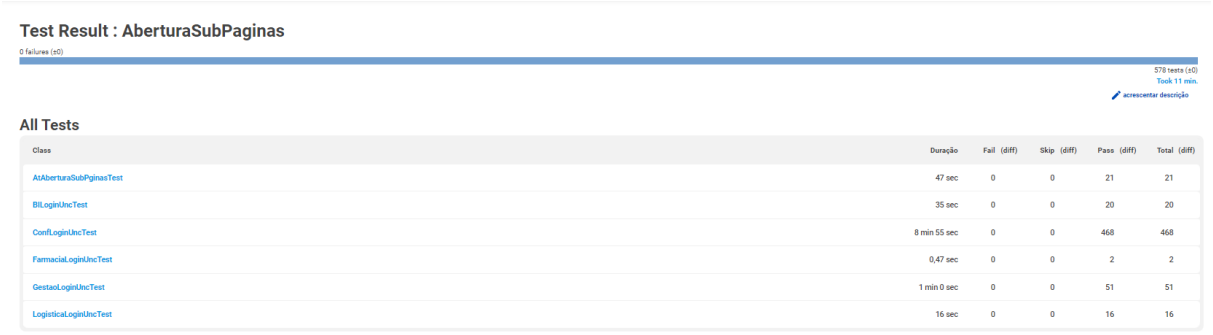


Figura 53- Resultados de teste do último build

Comparando estes resultados com os da Figura 51 verificamos que onde ocorreu maior discrepância de tempos foi na *class* ***LogisticaLoginUncTest***, que corresponde à classe onde na situação anterior ocorriam falhas, ou seja, podemos dizer que quando ocorrem falhas no teste, este demora mais tempo a ser executado. Isto pode ser justificado pelo facto de o sistema passar mais tempo a tentar responder ao que é pedido no teste antes de falhar.

Comparando agora a duração de execução com a quantidade de testes presentes em cada uma das *class,* podemos concluir que quantos mais testes a serem executados em determinada *class*, mais tempo esta demorará. Podemos ver na Figura 53 que a *class* ***ConfLoginUncTest*** possui 468 e demora um total de 8 minutos e 55 segundos. Por outro lado, aquele que apresenta menos testes também é o que demora menos tempo a ser executado.

Vamos agora para uma análise mais demonstrativa dos resultados. Através do gráfico da Figura 54 conseguimos ver qual o número de testes para cada um dos *builds*, sendo que só temos representados os *builds* desde o 67 ao 78. É importante realçar que cada uma das linhas representadas corresponde a um estado do teste. Podemos ver que o número total de testes se mantém sempre constante ao longo dos *builds*, assim como os *skipped*, que permanecem no valor 0. Os valores que vão diferir estão entre o número de testes que passam e que falham. Podemos ver que estas duas linhas vão ter comportamentos inversos, ou seja, quando a linha *Passed* está a 0, a linha *Failed* está no ponto equivalente ao *Total*. Isto acontece no *build* 67 e no caso do *build* 68 já acontece exatamente o oposto, ou seja, a linha *Failed* encontra-se no valor 0 e *Passed* encontra-se a colidir com o *Total*.

Analisando agora o *build* 73, podemos ver que existe um pequeno número de testes a falhar e que esse número também faz a linha dos *Passed* sofrer uma diminuição. Sabemos através de análises anteriores que no *build* 75 existem falhas, mas neste gráfico não conseguimos ter essa perceção uma vez que o número de falhas é muito reduzido.

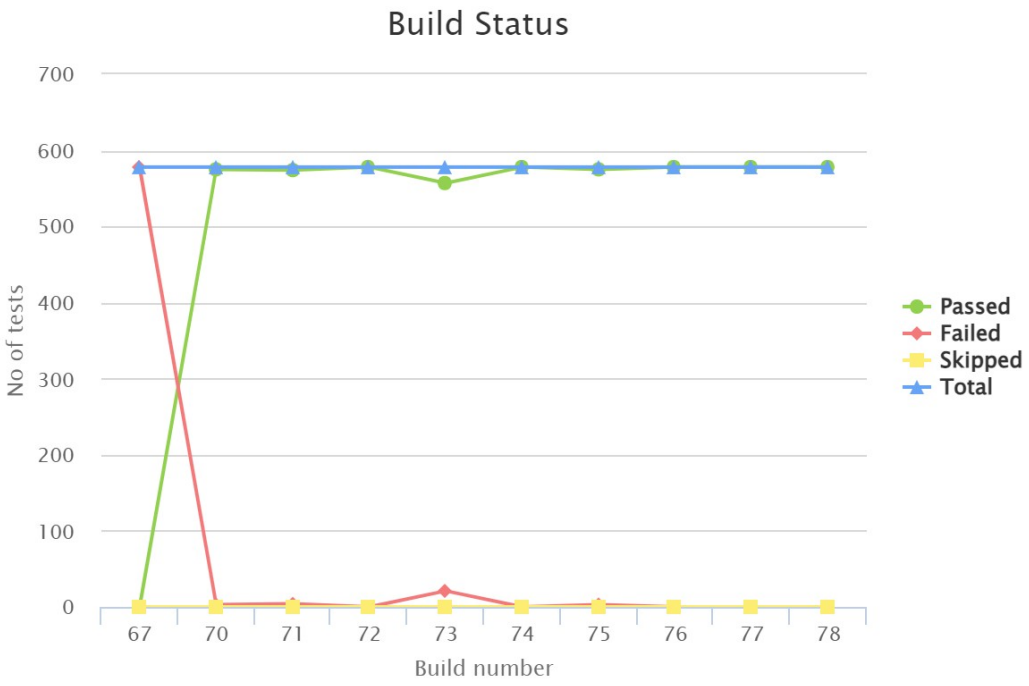


Figura 54- Gráfico do nº de testes nos diferentes estados por build

De forma a tentar ultrapassar esta questão de falha de precisão no número de testes quando este valor é muito reduzido, decidimos analisar um outro gráfico, um gráfico de barras. Neste gráfico continuamos a ter o número de testes por *build*, com a distinção de que vamos ter a barra sinalizada a vermelho com a correspondência do número de falhas e a barra a verde com a correspondência do número de sucessos. Apesar de ao olharmos para a Figura 55 não conseguirmos verificar qual o número concreto de testes que passam ou falham, é possível vermos com mais precisão quando se deram falhas e ter uma noção da proporção das mesmas.

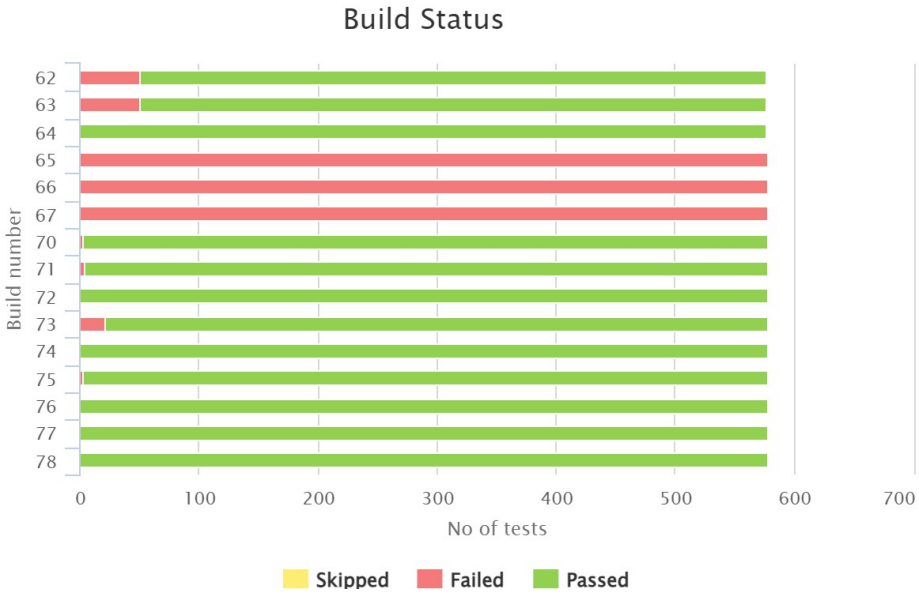


Figura 55- Gráfico de barras do nº testes por build

Além destes gráficos até agora apresentados, analisamos mais um gráfico relativo ao tempo de execução de cada *build,* em segundos. Na Figura 56 verificamos que a linha vai sofrendo algumas oscilações, isto é, o tempo de execução vai sofrendo variações. Vemos que ao longo dos *builds* as durações vão variando, mas que não é nada muito significativo, à exceção do *build* 73 onde vemos um pico bastante acentuado. Esta elevada oscilação de tempo pode justificar-se pelo facto de terem ocorrido muitas mais falhas do que o normal. Depois desse pico, as durações voltam aos intervalos de valores que mantinham anteriormente, verificando que é aqui que encontramos o *build* com a duração mais baixa. Isto pode dever-se ao facto de terem ocorrido melhorias constantes no projeto.

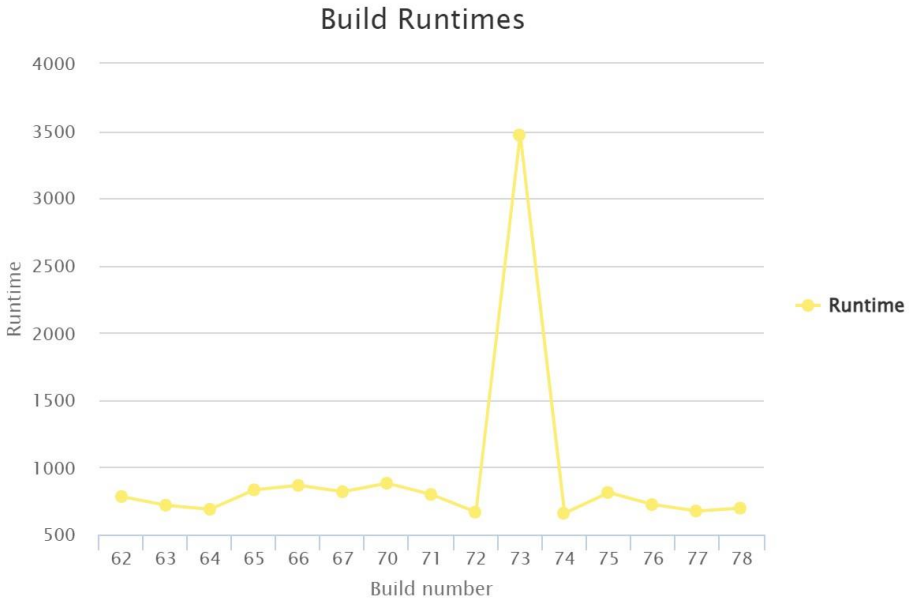


Figura 56- Gráfico da duração de execução dos testes por build

5. CONCLUSÕES E TRABALHO FUTURO

Neste capítulo será feita uma análise de todo o trabalho desenvolvido ao longo da dissertação, realçando quais os objetivos alcançados e ainda as conclusões a que chegamos. Por último serão apresentadas propostas de trabalho futuro.

Ao longo da dissertação exploramos vários métodos e ferramentas de automação de testes e chegamos ao *Selenium* como melhor ferramenta a ser utilizada no contexto da empresa. A partir daí desenvolvemos então um projeto de automação de testes para o *software* de gestão hospitalar. De acordo com o trabalho realizado podemos concluir que o objetivo de termos testes automáticos, reduzindo assim a quantidade de testes que eram realizados manualmente foi cumprida com sucesso. Apesar de não ser possível automatizar totalmente o sistema, devido ao tempo associado ao desenvolvimento do projeto e ainda ao facto de ser uma área totalmente nova, fez com que apenas automatizássemos uma parte bastante simples e de carácter inicial de testes e não todos os circuitos pretendidos. Além disso, surgiram ao longo do projeto questões que necessitaram de pesquisa para sabermos a possibilidade de realização, o que fez com que utilizássemos mais tempo para a investigação e só depois para a prática.

Apareceram ainda alguns desafios como a atualização do *Selenium IDE,* que fizeram com que tivéssemos uma abordagem alternativa para a leitura de dados a partir de ficheiros *Excel*, mas depois de desenvolvido todo o processo temos um *feedback* positivo, uma vez que percebemos que é possível automatizar eficazmente testes de *software* e que a nível da empresa esta automatização trouxe mais-valias, visto que houve uma redução do tempo que o *tester* passava a realizar testes manuais. Houve ainda uma deteção mais rápida dos erros devido ao facto de existir a execução de *builds* diários que permitiam detetar a falha antes da altura de encerramento da *sprint*, o que leva a uma melhoria na qualidade do *software*.

Depois de alguma análise sobre o trabalho desenvolvido foi possível verificar que seria possível implementar algumas melhorias ao trabalho até agora desenvolvido e ainda novos desenvolvimentos.

Em primeiro lugar, é possível a realização de mais testes, uma vez que o programa é muito extenso e não foi possível realizar testes para todas as funcionalidades do programa, mas apenas para os mais simples e ainda alguns mais prioritários, devido à elevada utilidade dessas funcionalidades. Além disso, seria útil implementar os testes em todos os ambientes com que a empresa trabalha. Esta conclusão deriva do facto dos testes serem realizados em ambiente QA, contudo em algumas situações poderá ser necessário testar as funcionalidades tanto em ambiente CI como em ambiente Dev.

Como vimos anteriormente, não nos era possível avaliar a cobertura dos testes, uma vez que não tínhamos acesso ao código fonte do programa. Posto isto, uma proposta de melhoria seria ter acesso a este código para que fosse possível implementar outras metodologias que nos permitissem confirmar com maior precisão o sucesso do teste realizado.

REFERÊNCIAS BIBLIOGRÁFICAS

Ali, S., Ibrahim, H., Nouh, K., & Fayek, M. (2022, October). An Improved Framework for Monkey GUI Testing for EDA Desktop Applications. In *2022 12th International Conference on Software Technology and Engineering (ICSTE)* (pp. 8-13). IEEE.

Altaf, I., Dar, J. A., ul Rashid, F., & Rafiq, M. (2015, October). Survey on *selenium* tool in *software* testing. In *2015 International Conference on Green Computing and Internet of Things (ICGCIoT)* (pp. 1378-1383). IEEE.

Alves, R. A., Martins, R. C., & Paulista, P. H. (2016). OPERAÇÃO E MANUTENÇÃO DE SOFTWARE: UMA ABORDAGEM TEÓRICA. *Revista Univap*, *22*(40), 766-766.

Arasteh, B., & Hosseini, S. M. J. (2022). Traxtor: an automatic software test suit generation method inspired by imperialist competitive optimization algorithms. *Journal of Electronic Testing*, *38*(2), 205-215.

Audy, J. L. N. (2007). *Desenvolvimento distribuído de software*. Elsevier.

Barbosa, E. F., Maldonado, J. C., Vincenzi, A. M. R., Delamaro, M. E., Souza, S. D. R. S. D., & Jino, M. (2000). Introdução ao teste de *software*. *Minicurso apresentado no XIV Simpósio Brasileiro de Engenharia de Software (SBES 2000)*.

Bartié, A. (2002). Garantia da qualidade de *software*.[Sl].

Candea, G., Bucur, S., & Zamfir, C. (2010, June). Automated *software* testing as a service. In *Proceedings of the 1st ACM symposium on Cloud computing* (pp. 155-160).

CHICANELLI, R., VECCHIATO, D., VENTURA, T., GOMES, R., TAKAGI, N., & MENDES, L. (2019). Aspectos sociais, humanos e econômicos da utilização de testes automatizados no desenvolvimento de sistemas. In *Décima Oitava Conferência Ibero Americana de Sistemas Cibernética e Informática*.

COLLINS, E. F.; KON, F. A Importância dos Testes Automatizados. p. 1–100, 2013.

AUTOMTECH. Desafios da automação de testes. [S. l.], 8 out. 2018. Disponível em: https://www.automtech.com.br/2018/10/08/desafios-da-automacao-de-testes/. Acesso em: 04 abr.2023

de Almeida Machado, T., & Tavares, J. F. (2022). Aumentando a Cobertura de Testes Através do Teste de Mutação. *Caderno de Estudos em Sistemas de Informação*, *7*(1).

Dhir, S., & Kumar, D. (2019). Automation *software* testing on web-based application. In *Software Engineering: Proceedings of CSI 2015* (pp. 691-698). Springer Singapore.

Dorothy Graham, Erik Van Veenendaal, and Isabel Evans. Foundations of *software* testing: ISTQB certification. Cengage Learning EMEA, 2008

Fraser, G., & Arcuri, A. (2013, March). Evosuite: On the challenges of test case generation in the real world. In *2013 IEEE sixth international conference on software testing, verification and validation* (pp. 362-369). IEEE.

Graham, D., & Fewster, M. (1999). *Software Test Automation: Effective Use of Text Execution Tools*. Addison Wesley.

Gusmão, B., Traci, H., & Talon, A. (2016). QUALIDADE DE SOFTWARE: UTILIZAÇÃO DO TESTE DE UNIDADE. *Caderno de Estudos Tecnológicos*, *4*(1).

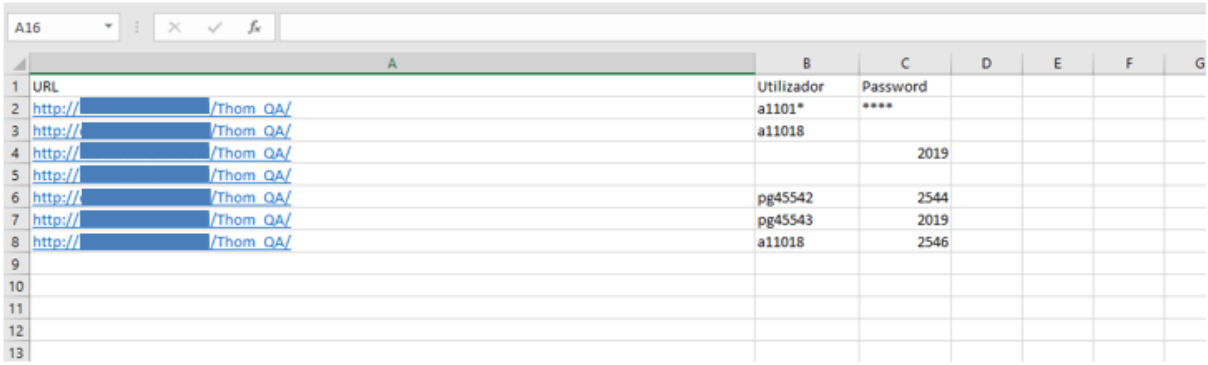
Polepalle, C., Kondoju, R. S., & Badampudi, D. (2022, February). Evidence and perceptions on GUI test automation-An Exploratory Study. In *15th Innovations in Software Engineering Conference* (pp. 1-10).

Pinheiro, S. A. M. (2015). *Estudo e implementação de testes de software em desenvolvimento ágil* (Doctoral dissertation).

Ricca, F., & Tonella, P. (2001, May). Analysis and testing of web applications. In *Proceedings of the 23rd International Conference on Software Engineering. ICSE 2001* (pp. 25-34). IEEE.

Teixeira, V. S., Delamaro, M. E., & Vincenzi, A. M. R. (2007). Fatesc-uma ferramenta de apoio ao teste estrutural de componentes. *Sessão de Ferramentas-XXI SIMPÓSIO BRASILEIRO de ENGENHARIA de SOFTWARE*, 7-12.

ANEXO I – EXCEL DE ACESSO À APLICAÇÃO



	A	B	C
1	URL	Utilizador	Password
2	http://[redacted]/Thom_QA/	a1101*	****
3	http://[redacted]/Thom_QA/	a11018	
4	http://[redacted]/Thom_QA/		2019
5	http://[redacted]/Thom_QA/		
6	http://[redacted]/Thom_QA/	pg45542	2544
7	http://[redacted]/Thom_QA/	pg45543	2019
8	http://[redacted]/Thom_QA/	a11018	2546

Figura 57- Folha de Excel do teste Acesso à Aplicação

ANEXO II – EXCEL DE CRIAR CLIENTE

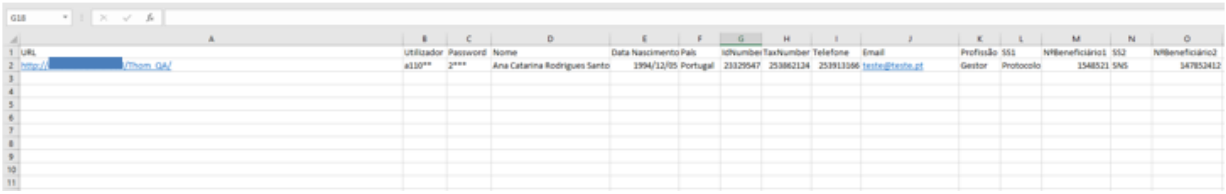
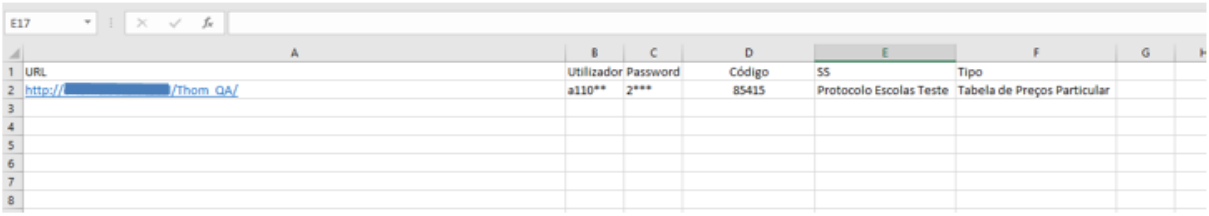


Figura 58- Folha Excel do teste Criar Cliente

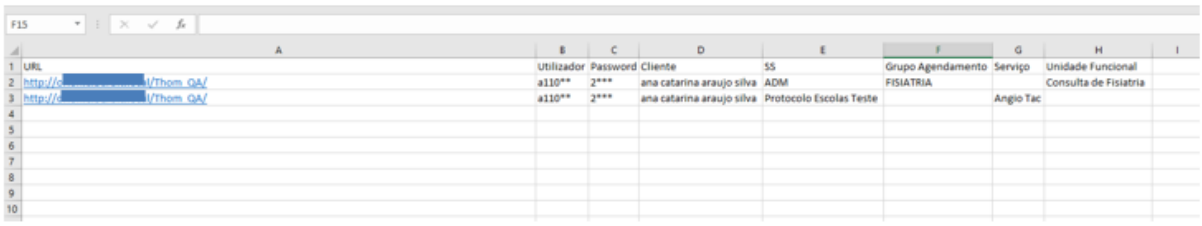
ANEXO III – EXCEL DE CRIAR SISTEMA DE SAÚDE



	A	B	C	D	E	F
1	URL	Utilizador	Password	Código	SS	Tipo
2	http://[redacted]/Thom_QA/	a110**	2***	85415	Protocolo Escolas Teste	Tabela de Preços Particular

Figura 59- Folha Excel do teste Criar Sistema de Saúde

ANEXO IV – EXCEL DE PESQUISA DE VAGAS



	A	B	C	D	E	F	G	H
1	URL	Utilizador	Password	Cliente	SS	Grupo Agendamento	Serviço	Unidade Funcional
2	http://[redacted]/Thom_QA/	a110**	2***	ana catarina araujo silva	ADM	FISIATRIA		Consulta de Fisiatria
3	http://[redacted]/Thom_QA/	a110**	2***	ana catarina araujo silva	Protocolo Escolas Teste		Angio Tac	

Figura 60- Folha Excel do teste Pesquisa de Vagas

Nestes anexos existem informações ocultadas por motivos de confidencialidade da empresa, sendo dados que não podem ser partilhados. Todos os restantes dados e variáveis utilizados são valores fictícios utilizados apenas para situações de teste.