UNIVERSIDADE DE LISBOA
FACULDADE DE CIÊNCIAS
DEPARTAMENTO DE INFORMÁTICA

QualState - Finding Website States for Accessibility Evaluation

Filipe Rosa Martins

Mestrado em Informática

Dissertação orientada por:
Carlos Alberto Pacheco dos Anjos Duarte
Letícia Seixas Pereira

2023

Resumo

Conseguir ter acessibilidade na web é, na atualidade, um tópico de grande importância. Conseguir garantir a acessibilidade a qualquer pessoa é agora crucial para qualquer web site ou aplicação web. Desta forma é necessário garantir que existem ferramentas capazes de fazer este tipo de avaliação de acessibilidade. No entanto, as ferramentas atuais estão limitadas a páginas estáticas, sem qualquer dinamismo, não tendo em consideração SPA (Single Page Application). Uma página que, através da interação de um utilizador, é transformada sem que seja necessário carregar novamente a mesma. Esta característica impossibilita a avaliação de acessibilidade da página em sua versão alterada, pois necessita de um conjunto de ações, algo que as ferramentas de avaliação atuais não são capazes de realizar.

O objetivo deste trabalho passa por desenvolver uma solução capaz de conseguir detetar o maior número de estados possível numa SPA e realizar a avaliação de acessibilidade aos mesmos com auxílio do QualWeb. Como tal, primeiramente este trabalho apresenta uma investigação ao panorama atual sobre os problemas e dificuldades na avaliação da acessibilidade, não só em SPA, mas em web sites e aplicações web. Seguidamente apresenta a solução proposta que visa combater o referido problema de exploração de SPA, o QualState. QualState é um crawler que utiliza um conjunto de configurações fornecidas pelo utilizador para explorar os diversos estados de uma SPA, através da simulação da interação de um utilizador com a página. Ainda dentro do âmbito do projeto, e de forma a tentar mitigar a limitação atual presente em avaliações de acessibilidade de SPA, o QualState foi integrado com o QualWeb, um avaliador de acessibilidade, que realiza a avaliação de acessibilidade das diferentes versões das páginas, i.e., estados, encontradas.

O QualState no fim conseguiu apresentar melhores resultados ao encontrar mais problemas de acessibilidade do que um avaliador de acessibilidade sem a capacidade de exploração.

O resultado é uma ferramenta que corresponde às espectativas de conseguir explorar SPA e realizar a avaliação de acessibilidade dos estados encontrados. Todavia, a solução final apresenta também novas dificuldades e desafios que devem ser superados para ter uma ferramenta mais completa.

Palavras-chave: Acessibilidade na Web, Crawler, Avaliação automática da acessibilidade, QualWeb, QualState

Abstract

Achieving accessibility on the web is a topic of great importance today. Being able to guarantee accessibility to anyone is now crucial for any website or web application. It is therefore necessary to ensure that there are tools capable of carrying out this type of accessibility assessment. However, current tools are limited to static pages, without any dynamism, and do not consider SPA (Single Page Application). A page that, through user interaction, is transformed without having to be reloaded, and this feature makes it impossible to evaluate the accessibility of the altered page, as it requires a set of actions, something that the current evaluators do not do.

The aim of this work is to develop a solution capable of detecting as many states as possible in a SPA and carrying out an accessibility assessment using QualWeb. As such, this work firstly presents an investigation into the current panorama of problems and difficulties in evaluating accessibility, not only in SPA, but also in web sites and web applications. It then presents the proposed solution to combat this problem of exploiting SPAs, QualState. QualState is a crawler that uses a set of settings provided by the user to explore the various states of an SPA, by simulating a user's interaction with the page. Also, within the scope of the project, and to try to combat the inability to carry out accessibility evaluations of SPAs, QualState was integrated with QualWeb, an accessibility evaluator, which carries out accessibility evaluations of the states found.

In the end, QualState was able to deliver better results by finding more accessibility problems than an accessibility evaluator without the exploration capability.

The result is a tool that lives up to the expectations of being able to explore SPA and carry out an accessibility assessment of the states found. However, the final solution also presents new difficulties and challenges that must be overcome to have a more complete tool.

Keywords: Web Accessibility, Crawler, Automated accessibility evaluation, QualWeb, QualState

Agradecimentos

Ao chegar ao fim desta dissertação tenho de agradecer à minha família e amigos e também aos meus orientadores, Carlos Duarte e Letícia Pereira pela ajuda, disponibilidade e todo o feedback dado neste último ano.

Índice

Lista de Figuras

Lista de Tabelas

1.Introdução

Nos últimos anos a utilização da internet teve um crescimento sem precedentes sem que mostre sinais de que irá abrandar, BroadbandSearch reporta que entre os anos 2000 e 2022 o uso de internet aumentou cerca de 1355%[1]. A utilização da internet está presente no nosso dia a dia estendendo-se por diversas áreas de negócio. Todos os dias surgem novas aplicações web e web sites com diversas funcionalidades. No meio deste avanço surge o conceito de Single Page Application (SPA), páginas dinâmicas que alteram a sua estrutura e ou conteúdo de acordo com as ações de um utilizador, sem que seja necessário recarregar a totalidade da página. Este conceito irá ser abordado em mais detalhe no capítulo de conceitos.

No entanto, SPAs apresentam grandes problemas para ferramentas de avaliação de acessibilidade. O conceito de acessibilidade na web não é recente. Surgiu com maior dimensão em 1996 no contexto de um projeto do W3C (Liu & Woei-Kae and Sun, 2020). Todavia, tem recentemente surgido uma maior preocupação em garantir que existe acessibilidade de forma a ter uma página que consegue ser acedida e utilizada por qualquer utilizador independentemente de este ter necessidades especiais. A grande maioria das ferramentas que fazem avaliações de acessibilidade, foram desenvolvidas para páginas estáticas, obtendo o HTML (Hypertext Markup Language) e outros elementos estáticos, como imagens e scripts JavaScript (Leithner & Simos, 2020), para depois fazerem uma avaliação de acessibilidade. Muitas ferramentas acabam também por dar uso a crawlers, que exploram uma página para encontrar links que possam ser invocados, para assim obter diferentes páginas e fazer uma avaliação de acessibilidade das mesmas. Contudo, estas ferramentas não estão preparadas para páginas com dinamismo como as SPAs, as quais são constantemente alteradas mediante ações do utilizador. Este dinamismo acaba por não ser detetado e consequentemente torna-se muito difícil, ou até mesmo impossível, fazer uma avaliação de acessibilidade que seja fiável.

1.1.Objetivos

O objetivo principal deste trabalho passa por desenvolver a ferramenta QualState, uma solução que responde ao problema de avaliações de acessibilidade em SPA, com o uso de um crawler capaz de explorar o maior número possível de estados. Aliado a este objetivo existem outros objetivos secundários, entre os quais: 1) realizar uma investigação sobre o trabalho atual de forma a compreender todos os conceitos necessários e também de tentar antecipar possíveis problemas na fase de desenvolvimento, 2) integrar o QualState com QualWeb de forma a conseguir fazer avaliação de acessibilidade de todos os estados encontrados e por fim 3) fazer uma avaliação inicial dos benefícios que o QualState representa relativamente ao QualWeb sem QualState.

[1] (https://www.broadbandsearch.net/blog/internet-statistics#post-navigation-0, 2022

1.2.Motivação

Como referido, a internet está cada vez mais presente na nossa vida quotidiana. No entanto, e apesar da grande dimensão da internet, a mesma ainda não esta disponível para todas as pessoas, pois existem diversos problemas de acessibilidade que impossibilitam que pessoas com deficiências consigam utilizar uma grande porção da internet. O foco deste trabalho passa por encontrar estados dentro de uma SPA com o objetivo de realizar uma avaliação de acessibilidade dos estados encontrados. Como tal a motivação deste trabalho passa por melhorar a acessibilidade e consequentemente a experiência de todo o tipo de utilizador numa SPA.

1.3.Contribuições

Este trabalho no fim apresenta as seguintes contribuições:

- Um crawler que consegue explorar uma SPA.
- Integrar o QualState no Qualweb de forma a realizar a avaliação de acessibilidade de uma SPA, assim expandindo as capacidades do Qualweb.

1.4.Estrutura do documento

Este documento apresenta-se dividido em seis capítulos. No Capítulo 2 são abordados diversos conceitos e tópicos, como SPA e ferramentas de avaliação de acessibilidade, necessários para a compreensão do restante documento. O Capítulo 3 serve para uma investigação sobre o panorama atual de crawlers e avaliadores de acessibilidade. O capítulo 4 apresenta a metedologia utilizada neste trabalho. O Capítulo 5 apresenta diversos componentes sobre o QualState, começando com uma breve introdução seguido de seções com mais detalhe sobre a sua arquitetura e fluxo de funcionamento. O Capítulo 6 foca-se nos testes realizados e os resultados obtidos. Por fim, o Capítulo 7 conclui o trabalho junto com uma discussão sobre trabalho futuro.

2.Enquadramento Teórico

Este capítulo irá servir para apresentar e explicar diversos conceitos para ajudar na compreensão do documento. Este capítulo explica conceitos como SPA e como estas funcionam, conceitos sobre DOM (Document Object Model) e CSSOM (Cascading Style Sheets Object Model), estado de uma página, a framework Puppeteer, e crawlers. Irá ainda ser introduzido o conceito de acessibilidade na web, os problemas atuais deste contexto, e formas de avaliar acessibilidade. Por fim irá ser também feita referência à ferramenta QualWeb e como esta funciona.

2.1. Conceitos do Browser

2.1.1. DOM e CSSOM DOM

DOM trata-se de uma API (Application Programming Interface) que define a estrutura de um documento e que permite a manipulação de todos os seus elementos (Robie, 2022). A estrutura do documento é apresentada como uma árvore em que os nós são elementos HTML. Através do DOM é possível criar, construir e navegar documentos como também adicionar, modificar ou alterar os seus elementos (Robie, 2022). CSSOM apresenta uma ideia similar ao DOM, com a diferença que este refere-se a CSS (Cascading Style Sheets). CSSOM é uma API que permite a manipulação de CSS. Um exemplo de ambos o tipo de DOM pode ser visto na Figura 2.1.

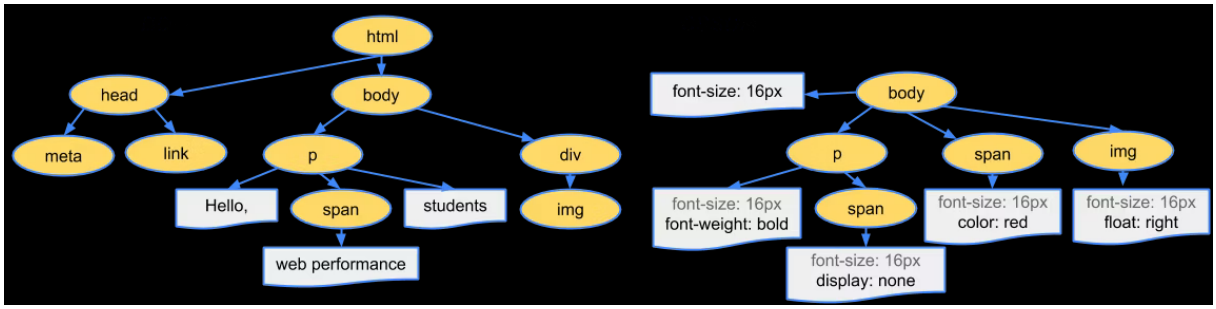

Figura 2.1 - Exemplo de DOM e CSSOM DOM, respetivamente.

2.1.2. Estado da página

O estado de uma página pode ser entendido como uma representação da estrutura e conteúdo da página, em que o web site ou aplicação web está num determinado momento. A opção mais utilizada para representar um estado é o DOM. Todavia, este conceito pode mudar de acordo necessidades do projeto. Existem abordagens que utilizam o DOM para definir o estado de uma página (Understanding the Four Principles of Accessibility, 2022), por outro lado existem também outras abordagens que utilizam uma combinação de URL (Uniform Resource Locators), links, formulários, e uma lista dos eventos existentes (Leithner & Simos, 2020).

2.2.Puppeteer

Puppeteer (Puppeteer) é uma framework node.js desenvolvida pela Google que fornece diversas API para conseguir controlar o browser através do protocolo DevTools. O Puppeteer por predefinição corre em modo *hedless* a false, sendo que este valor pode ser alterado para true. Correr um *headless* browser, significa apenas que não existe uma UI (User Interface) do browser, ou seja, o utilizador não vê o browser, pois este é para todos os efeitos "invisível". A framework disponibiliza também, para além das APIs previamente mencionadas, APIs que permitem fazer debbuging a web sites e automatizar testes.

Puppeteer atualmente é maioritariamente utilizado para automatizar testes no browser, contudo, no contexto deste projeto, irá ser utlizado para realizar ações para atingir diferentes estados.

2.3.Crawler

Web crawlers surgiram inicialmente com a intenção de explorar todos os links que uma aplicação web ou web site apresentam (Li, Han, Liu, & Fang, 2018) e (Liu & Woei-Kae and Sun, 2020). O processo inicia-se com um URL para uma aplicação web ou web site que é depois explorado com a intenção de encontrar outros links. Todos os links encontrados são depois colocados numa lista que irá também ser explorada. Atualmente, o conceito de crawler tem sofrido algumas alterações. É agora necessário fazer uma exploração mais abrangente de forma a obter mais detalhes sobre a página. Atualmente um crawler obtém o código HTML, DOM, CSS, imagens, e scripts Javascript (Leithner & Simos, 2020). Estes elementos são depois processados para isolar elementos, maioritariamente tags HTML, que permitam estender a exploração. Os elementos encontrados são depois colocados numa lista para serem estes também explorados.

2.4.SPA

Nos últimos anos surgiu uma nova forma de desenvolver aplicações web e web sites, conhecido por SPA (Almendra, Moquillaza, & Paz, 2019). Van Deursen et al. (Mesbah & van Deursen, Migrating Multi-page Web Applications to Single-page AJAX Interfaces, 2007) define SPA como "the single-page web interface is composed of individual components which can be updated/replaced independently, so that the entire page does not need to be reloaded on each user action". Em outras palavras, uma página com a qual os utilizadores interagem, sendo depois rescrita e/ou alterada mediante as ações do utilizador sem que seja necessário recarregar a totalidade da página em questão, ao contrário de outras abordagens que redirecionam os utilizadores para outras páginas mediante as ações dos mesmos.

SPAs surgiram como uma alternativa às formas mais tradicionais de desenvolvimento de web sites, que contam muito com links entre diversas páginas para navegar pela aplicação. Para atingir estas características, SPAs utilizam AJAX (Asynchronous JavaScript and XML) para carregar apenas os componentes individuais necessários de acordo com a utilização do utilizador.

2.4.1. Funcionamento de SPA

A comunicação realizada por uma SPA é inicialmente semelhante à das páginas tradicionais. Inicialmente é feito um request ao servidor, o qual devolve toda a informação e recursos que compõem o web site, como HTML, imagens, CSS, e os scripts que sejam coerentes com pedido. Seguidamente começa o DOM Processing, o processo no qual se realiza o parse que irá processar o HTML para construir o DOM e o CSSOM (Populating the page: how browsers work, 2022).

Este processo termina quando é despoletado o evento "Window: DOMContentLoaded" (Window: DOMContentLoaded event, 2022), que indica que o parse da estrutura do DOM foi concluído, apesar de existir a possibilidade de ainda estar em falta algum conteúdo que ainda esteja no processo de download, como imagens ou scripts. Em seguida, é feito a renderização da página. Este processo passa por unir o DOM e o CSSOM, de forma a construir a interface apresentada ao utilizador. Finalmente é despoletado evento "Window: load" (Window: load event, 2022), de forma a indicar que todo o conteúdo da página está carregado. SPAs distinguem-se pela forma como conseguem obter novo conteúdo.

Enquanto a página tradicional tem de repetir todo o processo novamente, SPAs têm duas formas de o fazer sem que seja necessário atualizar toda a página. Dada uma interação que o utilizador realiza, o novo conteúdo pode ser obtido através de scripts já carregados. Caso esta hipótese não seja possível, pode ser realizado um request AJAX ao servidor que devolve apenas a informação necessária a utilizar conforme a interação decorrida (Fernandes, Batista, Costa, Duarte, & Carriço, 2013). A comunicação acima descrita pode ser dividida em 3 passos e está apresentada na Figura 2.2 (Fernandes, Batista, Costa, Duarte, & Carriço, 2013):

- Request web page: Pedido para o recurso principal que define a estrutura do conteúdo que irá ser apresentado.

- Request resources: Pedido para os recursos complementares para auxiliar a construção da página.

- AJAX request: É neste passo que se foca uma SPA. Fazer um pedido AJAX de recursos que sejam necessários permitindo uma maior interatividade por parte do utilizador.

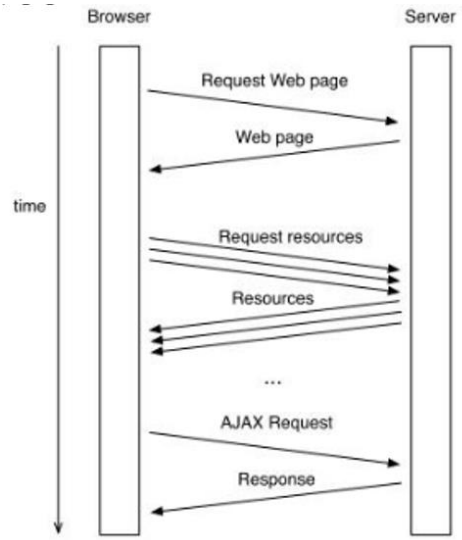

Figura 2.2 - Processamento de recursos no browser *(Fernandes, Batista, Costa, Duarte, & Carriço, 2013)*.

2.5. Acessibilidade na Web

A palavra acessibilidade é uma palavra com uma grande importância nos últimos tempos, contudo, mais pela negativa. A falta de acessibilidade tem sido um grande tópico nos últimos anos, sendo que com a pandemia do COVID-19 ganhou ainda mais importância e relevo. A recente quarentena levou a que o acesso a serviços se tornasse exclusivamente online, acabando por demonstrar inúmeros problemas que existem com a acessibilidade na grande maioria dos websites, Lazar (Lazar, 2022) reporta como este exato problema afetou inúmeras faculdades nos Estados Unidos.

W3C define acessibilidade na web como "websites, tools, and technologies are designed and developed so that people with disabilities can use them" (Lawton Henry, 2022).

Com o crescente uso da internet e a sua grande presença no nosso dia a dia, a falta de acessibilidade começa a ser colocada em questão pela forma como pode excluir certos grupos, nomeadamente daltónicos, surdos, cegos, e até mesmo idosos que com o avançar da vida ficam mais suscetíveis a mais problemas de acessibilidade (Campoverde-Molina, Luján-Mora, & García, 2020). Contudo, este problema tem de ser resolvido, pois o uso da internet está de momento incorporado no dia a dia de uma grande parte da sociedade, sendo crucial para campos desde a educação até a saúde (Campoverde-Molina, Luján-Mora, & García, 2020).

É com esta ideia que surgiram as WCAG (Web Content Accessibility Guidelines) (Campoverde-Molina, Luján-Mora, & García, 2020), um dos standards mais utilizados, servindo de referência quando se quer tornar um website mais acessível através de um conjunto de diretrizes a seguir. Existem outros standards como US Government Section 508[2] e IBM Web Accessibility Checklist [3]. No entanto, todos tiveram como inspiração e base as WCAG. Existe ainda cada vez mais

[2] https://www.section508.gov/
[3] https://www.ibm.com/able/

um esforço de entidades competentes em criar regulamentos e regras para garantir que normas de acessibilidade são cumpridas e utilizadas.

2.5.1. WCAG

As WCAG (Web Content Accessibility Guidelines), através da iniciativa WAI (Web Accessibility Initiative), foram desenvolvidas por diversas organizações à volta do mundo, com o intuito de criar um standard, com diversas diretrizes, a ser seguido, utilizado e partilhado por indivíduos e organizações (Campoverde-Molina, Luján-Mora, & García, 2020). Estas diretrizes servem de referência para estabelecer acessibilidade na web a um variado grupo de pessoas com deficiências.

A primeira versão das WCAG surgiu em 1999 com 14 diretrizes. WCAG está atualmente na versão 2.1, com 13 diretrizes distribuídas por 4 princípios, sendo que cada diretriz pode ter um ou mais critérios de sucesso.

2.5.2. Princípios

A versão 2 das WCAG organiza-se em 4 princípios: Percetível, Operável, Compreensível e Robusto (Web Content Accessibility Guidelines (WCAG) 2.0) & (Hostetler, Chen, & Blanco-Cuaresma, 2022) & (Understanding the Four Principles of Accessibility, 2022):

- **Percetível:** Toda a informação apresentada tem de ter uma forma alternativa de ser acedida. A informação apresentada não pode ser invisível a todos os sentidos, ou seja, tem de existir uma forma de apresentar a informação que seja possível receber e compreender por todos os sentidos. Cabe a quem estiver a desenvolver o software garantir que isto acontece.

- **Operável:** Qualquer utilizador, independentemente da deficiência ou problema que o afeta, tem de conseguir navegar pela interface apresentada. Por outras palavras, é necessário existirem diferentes maneiras de navegar pela interface para garantir que um utilizador não é obrigado a fazer uma interação que não lhe seja possível.

- **Compreensível:** O utilizador tem de compreender a informação que lhe é apresentada, como também a operação que realiza.

- **Robusto:** O software tem de ser robusto, ou seja, a forma como este é desenvolvido tem de ter em consideração tecnologias futuras, para assim garantir que funciona sempre com diferentes agentes de utilizador, tecnologias de assistência, ou outro tipo de ferramenta que sirva para auxiliar pessoas variadas tipos de incapacidades.

2.5.3. Critério de Sucesso

Pode se entender um critério de sucesso como um teste. Estes testes possuem o intuito de serem específicos, de forma a garantir que não existe nenhum tipo de subjetividade quando os mesmos são realizados (Understanding the Four Principles of Accessibility, 2022). Para cada critério de sucesso existe um nível de conformidade. Existem 3 níveis de conformidade: A, AA, AAA (Understanding the Four Principles of Accessibility, 2022), sendo que a diferença entre os mesmos é o nível exigido de acessibilidade. O nível A é o mínimo para atingir conformidade com as WCAG e o AAA o nível mais exigente de conformidade.

2.6.Dificuldade em implementar acessibilidade na web

Apesar do esforço existente nos últimos anos em tornar a acessibilidade na web uma realidade, ainda existem diversos problemas associados à mesma. Abuaddous et al. (Hayfa.Y, Mohd Zalisham Jali, & Nurlida Basir, 2016) resumem e agrupam os problemas em 3 grupos distintos: Standards e diretrizes, durante o desenho e desenvolvimento, e por fim, durante a avaliação de acessibilidade:

- **Standards e diretrizes** Abuaddous et al. (Hayfa.Y, Mohd Zalisham Jali, & Nurlida Basir, 2016) fazem referência a diversos problemas neste tópico, no entanto acho importante realçar 2 aspetos importantes:

    - **Ambiguidade:** Embora as WCAG sejam o standard de referência, estas ainda são alvos de alguma ambiguidade. Apesar de existir um esforço para tornar as diretrizes e critérios de sucesso objetivos, a sua interpretação acaba por ser subjetiva, o que leva a interpretações diferentes e que consequentemente levam a soluções finais mal implementadas, ou seja, problemas com a acessibilidade. De forma a combater este problema surgiram as regras ACT (Accessibility Conformance Testing) (About ACT Rules, 2022), com regras específicas de como devem ser testados certos aspetos de acessibilidade, e mitigar o problema da ambiguidade e subjetividade durante o desenvolvimento dos websites.

    - **Incompleto:** A utilização das WCAG simplesmente não é suficiente. A aplicação deste standard sozinho não serve para assegurar uma acessibilidade completa.

- **Durante o desenho e desenvolvimento:** A falta de peritos e pessoal treinado para conseguir implementar e/ou rever acessibilidade torna-se um problema pois implementação de acessibilidade é comprometida. O maior problema associado a este tópico é a falta de sensibilidade sobre acessibilidade. Não obstante todo o esforço feito até ao momento em sensibilizar a indústria, o mesmo prevalece.

- **Durante a avaliação de acessibilidade:** Considerando a relevância deste tópico para este trabalho, o mesmo será abordado em mais detalhe no capítulo 2.7 com a definição de cada método de avaliação, bem como os maiores problemas em cada método.

2.7.Avaliação de acessibilidade na web

O propósito de avaliar acessibilidade é de garantir que um determinado web site é acessível a qualquer pessoa, independentemente se esta possui alguma incapacidade. Ou seja, construir algo inclusivo, em que todas as pessoas conseguem aceder e atingir os seus objetivos sem qualquer impedimento. Harper e Yesilada (Harper & Yesilada, 2008) definem avaliação de acessibilidade na web como "[. . .] the assessment of how well a website can be used by users with disabilities". Como em qualquer desenvolvimento de software, e num cenário ideal, deve existir um processo de avaliação de acessibilidade em todas as fases de desenvolvimento, independente da metodologia de desenvolvimento utilizada, i.e., metodologia Agile. Como referido, este cenário seria o ideal, tendo em conta o aspeto crucial que a acessibilidade tem. Todavia, o mesmo nem sempre é possível. Contudo, o foco deste trabalho é na avaliação após o desenvolvimento e durante o uso do software.

2.7.1. Métodos de avaliação da acessibilidade

Como referido, o objetivo deste trabalho foca-se na avaliação após o desenvolvimento e durante a contínua utilização. Nesse cenário existem diversas abordagens em que o objetivo passa por identificar falhas de acessibilidade que possam estar presentes. Neste trabalho, focaremos nas 3 mais utilizadas: ferramentas automáticas de teste, testes manuais e/ou avaliação de peritos, testes com utilizadores (Abascal, Arrue, & Valencia, 2019) & (Harper & Yesilada, 2008) & (Nuñez, Moquillaza, & Paz, 2019):

- **Ferramentas automáticas de teste:** A avaliação de acessibilidade é um trabalho exigente que consume tempo e recursos (Hayfa.Y, Mohd Zalisham Jali, & Nurlida Basir, 2016). Como tal, começaram a surgir ferramentas que fazem avaliações de sites web de forma automática. Estas ajudam a reduzir o tempo e esforço que normalmente envolve fazer uma avaliação. Estas ferramentas podem ser utilizadas de forma online ou local (Abascal, Arrue, & Valencia, 2019) e regem-se por standards de acessibilidade, definidos por quem as desenvolveu. Estas ferramentas, no entanto, apresentam alguns problemas (Hayfa.Y, Mohd Zalisham Jali, & Nurlida Basir, 2016). O principal deles é que apenas se conseguem avaliar diretrizes que sejam possíveis codificar, ou seja, existem diretrizes que não permitem uma avaliação automática devido à sua complexidade (Abascal, Arrue, & Valencia, 2019) & (Hayfa.Y, Mohd Zalisham Jali, & Nurlida Basir, 2016). Existe também inconsistência na forma como algumas diretrizes são abordadas por diferentes programadores e consequentemente programadas, o que acaba por apresentar resultados diferentes entre diversas ferramentas de avaliação. Com isto em conta, estas ferramentas apresentam grande benefícios numa avaliação inicial, de forma a filtrar rapidamente uma grande quantidade de problemas de acessibilidade, sendo que a avaliação deve ser depois aprofundada (Abascal, Arrue, & Valencia, 2019). O funcionamento deste tipo de ferramenta irá ser abordado com mais detalhe no capítulo 2.7.2.

- **Teste manuais e/ou avaliação de peritos:** Idealmente e sempre que possível este método de avaliação deve ser utilizado (Hayfa.Y, Mohd Zalisham Jali, & Nurlida Basir, 2016). Apesar de existirem ferramentas automáticas que realizam avaliações de acessibilidade, existe sempre o fator humano que é imprescindível no que toca a acessibilidade. Os testes manuais podem ser executados por uma variedade de utilizadores desde os que tem menos conhecimento até peritos (Harper & Yesilada, 2008). O referido tipo de teste consiste em avaliar web sites contra

as diretrizes do standard selecionado para fazer a avaliação (Nuñez, Moquillaza, & Paz, 2019). Alguns testes também envolvem a avaliação de um web site usando as mesmas técnicas de interação ou dispositivos utilizados por pessoas com deficiências (Abascal, Arrue, & Valencia, 2019). Este tipo de teste tem a desvantagem de ser muito demorado e cansativo para quem os realiza, sendo que os utilizadores que realizam os testes podem ser alvo de fadiga e acabar por negligenciar alguns problemas. No fim da avaliação, na eventualidade de serem encontrados problemas de acessibilidade, estes são reportados e consequentemente corrigidos.

- **Testes com utilizadores:** São realizados com auxílio de um ou mais grupos de utilizadores finais, visando conseguir perceber se o site web possui barreiras de acessibilidade (Abascal, Arrue, & Valencia, 2019). Neste tipo de teste não é utilizada nenhuma ferramenta de avaliação, nem existem peritos para realizar os testes (Harper & Yesilada, 2008). Apesar do intuito deste tipo de teste ser a participação de utilizadores finais, é aqui que reside uma das maiores dificuldades deste tipo de teste: a necessidade de recrutar pessoas com deficiências específicas que possam participar nos testes (Hayfa.Y, Mohd Zalisham Jali, & Nurlida Basir, 2016).

2.7.2. Ferramentas de avaliação de acessibilidade

Existem várias ferramentas que testam a acessibilidade de uma página, como por exemplo AChecker[4] ou aXe[5]. Considerando o objetivo estabelecido para este trabalho, será detalhado a forma como funciona o QualWeb, um avaliador de acessibilidade.

2.7.3. QualWeb

O QualWeb[6] é um avaliador de acessibilidade automático que está em desenvolvimento desde 2008 por uma equipa de investigadores do LASIGE e está atualmente na versão 4.0. Atualmente o QualWeb existe como web site, uma extensão de browser para o Google Chrome, como um CLI (Command Line Interface), e como um package para ser integrado em projetos Node.js.

2.7.4. Arquitetura do QualWeb

A versão mais recente do QualWeb apresenta diversas diferenças na arquitetura relativamente à versão anterior, apresentando-se com uma arquitetura mais modular, o que por sua vez permite auxiliar na inclusão de futuras melhorias como também ajudar na sua manutenção. A arquitetura do QualWeb divide-se em 13 módulos (Estriga, 2020):

- **Dom:** Este módulo permite obter toda a informação da página web a ser processada.

[4] https://achecks.org/achecker/
[5] https://www.deque.com/axe/
[6] http://qualweb.di.fc.ul.pt/evaluator/

- **Evaluation:** Módulo encarregue de fazer a avaliação, utilizando diversos módulos com diferentes regras e técnicas.
    - **HTML:** Módulo que utiliza diretrizes WCAG para fazer a avaliação de acessibilidade do HTML.
    - **BP:** Best Pratice, um módulo desenvolvido pela equipa do QualWeb que realiza testes de acessibilidade sobre problemas que a equipa acha pertinente testar e assim contribuir para uma avaliação de acessibilidade mais completa.
    - **ACT:** Módulo que utiliza regras ACT para avaliar a acessibilidade.
    - **CSS:** Módulo que utiliza técnicas e regras de avaliação WCAG referentes ao CSS do web site.
- **QWPage:** O módulo QWPage lida com todas as interações a realizar sobre o Document e o objeto Window. Este módulo é o único que consegue interagir com o Document.
- **QWElement:** Módulo que interage com os objetos Element da página. Disponibiliza também diversos métodos que permitem obter propriedades e atributos da classe Element.
- **Util:** Módulo que reúne diversas funções que são úteis a diversos módulos, assim evitando repetição de código.
- **Core:** Módulo principal do QualWeb, responsável por fazer a ligação entre os diversos módulos.
- **CLI:** Módulo que representa a utilização do QualWeb com o uso de CLI (Command Line Interface).
- **Website:** Módulo que representa a utilização do QualWeb através do web site: http://qualweb.di.fc.ul. pt/evaluator.
- **EARL:** Módulo que está encarregue de apresentar os resultados. Este módulo utiliza o formato EARL (Evaluation and Report Language), um padrão desenvolvido pelo W3C para apresentar avaliações de acessibilidade (Velasco & Abou-Zahra, 2017).

2.7.5. Fluxo do funcionamento do QualWeb

De forma a utilizar o QualWeb, o utilizador tem de inserir um URL válido num dos módulos de front-end, escolher que módulo de avaliação deve ser utilizado, e qual o tipo de output pretendido. Seguidamente, o URL é enviado para o módulo DOM que obtém o HTML. Este HTML é depois enviado para os módulos de avaliação. Caso o módulo de avaliação seja um dos pretendidos, é feita uma avaliação, sendo que o resultado da mesma é adicionado a um relatório. Por fim, o relatório final é enviado para um formatter para ser convertido para o formato EARL, ou, caso este não seja o formato pretendido, o relatório final é simplesmente retornado ao utilizador. O fluxo do funcionamento do QualWeb está também apresentado como um fluxograma na Figura 2.3.

No contexto de avaliadores de acessibilidade, as SPA, devido às suas particularidades, ainda apresentam um problema que os avaliadores de acessibilidade atuais não conseguem responder.

2.8.Dificuldade na avaliação de acessibilidade de SPA

SPAs e a avaliação de acessibilidade das mesmas tem sido um tópico com crescente interesse. Como referido anteriormente, SPAs mudam os seus dados ou estrutura conforme as interações dos utilizadores, ou seja, é difícil garantir que todos os elementos que constituem a página estão presentes no momento da avaliação.

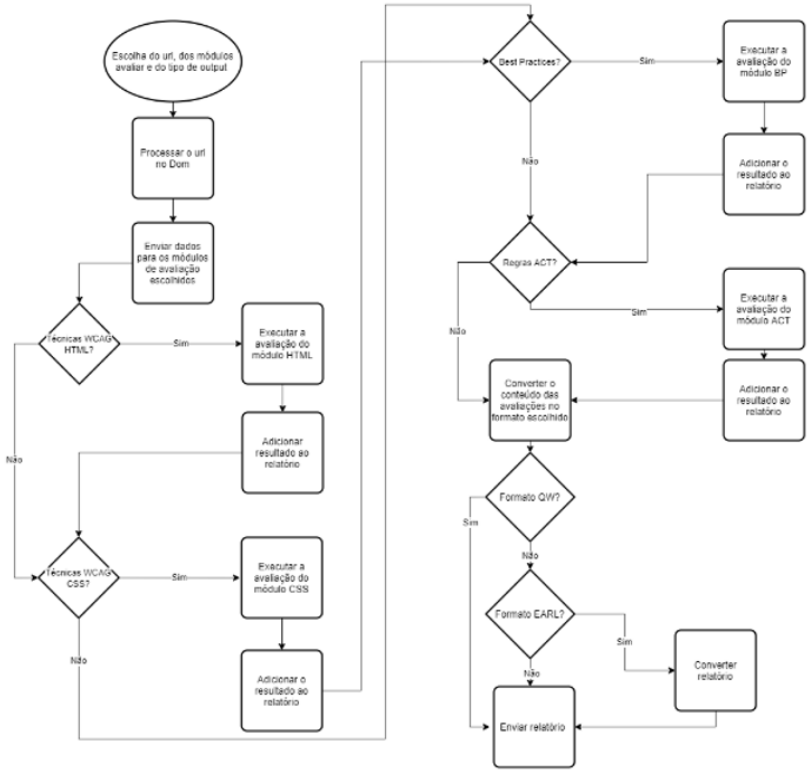

Figura 2.3 - Processamento do QualWeb *(Estriga, 2020)*s

2.8.1. Avaliadores estáticos e as suas limitações com SPA

Como mencionado na seção 2.7.1, existem ferramentas automáticas que fazem a avaliação de acessibilidade. Todavia, estes avaliadores foram desenvolvidos para avaliar páginas estáticas, o que os torna muito limitados. Estes tipos de avaliadores analisam apenas o estado original da página, sendo que não conseguem reconhecer e perceber quando o mesmo é atualizado, devido a uma interação do utilizador (Bostic, et al., 2021). Para além desta falha, estas ferramentas também não têm capacidade de invocar diferentes eventos nas SPAs e capturar novos estados para fazer a avaliação das mesmas. Como tal, não se consegue obter uma avaliação de acessibilidade de uma SPA de maneira fidedigna utilizando os avaliadores tradicionais.

2.8.2. Avaliadores dinâmicos

De forma a conseguir avaliar uma SPA na sua totalidade, ou pelo menos parcialmente, é necessário existir uma adaptação das ferramentas que existem atualmente. Recentemente têm surgido avaliadores dinâmicos, que utilizam conceitos de crawler para conseguir explorar SPAs. Este conceito irá ser abordado no capítulo seguinte. É a partir desta ideia que surge este trabalho, unir um crawler que consiga explorar todos os estados de uma SPA e fazer uma avaliação de acessibilidade de cada estado alcançado.

3.Trabalho Relacionado

A seguinte seção serve para apresentar o trabalho relacionado que foi encontrado e revisto durante a fase de investigação do trabalho. Para a investigação foi utilizada uma metodologia exploratória, em que foram utilizadas várias palavras-chave durante a pesquisa, i.e., acessibilidade na web, crawler, avaliação automática da acessibilidade, no Google Scholar. Todos os artigos encontrados foram analisados sendo que todos os tópicos relevantes estão destacados nesta seção.

3.1.Crawlers

Web crawlers têm se tornado uma tendência nos últimos tempos devido à utilidade que apresentam, neste caso para a avaliação de acessibilidade de SPA. Crawlers têm surgido na literatura com duas abordagens distintas, automáticos e semiautomáticos, sendo que ambas as abordagens apresentam vantagens e desvantagens.

3.1.1. Crawlers Automáticos

Crawlers automáticos têm a vantagem de serem ferramentas completamente autónomas, sem que seja necessário o utilizador interagir, apresentando-se assim como uma opção em que não é necessário nenhum tipo de conhecimento técnico, nem conhecimento da aplicação a ser explorada.

Crawljax[7] apresenta-se como "state-of-the-art" (Sunman, Soydan, & Sözer, 2022), e como uma das primeiras ferramentas de crawler de SPA. Crawljax é uma ferramenta automática, open-source, que, dado um URL, explora uma aplicação web de forma aleatória, introduzindo valores aleatórios em campos de input e invocando todos os elementos que identifique como clicável (Leithner & Simos, 2020). Simultaneamente, vai construindo um grafo com todos os estados encontrados (Leithner & Simos, 2020), sendo os nós os estados, e os vértices como o estado foi atingido. Crawljax apresenta a sua maior limitação no número reduzido de eventos que pode invocar, sobre um número igualmente pequeno de tags HTML. Crawljax concentra-se apenas em considerar as tags <A>, <DIV>, <INPUT> e <IMG> e os eventos "click" e "mouseOver" (Mesbah, van Deursen, & Roest, Invariant-Based Automatic Testing of Modern Web Applications, 2012). Consequentemente isto restringe o número de estados que se pode explorar, visto que existem outras tags HTML e eventos que podem gerar outros estados. Crawljax, no entanto, disponibiliza um conjunto de plugins que podem estender a sua utilidade para diversas aplicações (Liu & Woei-Kae and Sun, 2020), entre os quais a possibilidade aumentar a lista de tags e eventos que podem ser utilizados.

Demodocus (Bostic, et al., 2021) apresenta uma abordagem bastante semelhante ao Crawljax. Dado um URL, são invocadas todas as interações possíveis de forma a tentar alcançar o máximo de estados. O grafo construído por fim apresenta também uma estrutura semelhante ao que o Crawljax constrói, sendo os nós os diferentes estados atingidos e o vértice como se chega aos mesmos, ou seja, a interação que um utilizador tem na aplicação web. Demodocus tem a particularidade de ter uma lista de tags e eventos mais extensa comparativamente ao Crawljax. A maior limitação desta ferramenta é a impossibilidade de introduzir valores para preencher inputs e assim auxiliar na exploração de novos estados. Este problema torna esta abordagem muito limitada, pois, em muitos casos, uma grande

[7] https://github.com/crawljax/crawljax

porção da aplicação web aparece após interações que necessitam de inputs. Por exemplo, uma página de autenticação em que é necessário username e password. Crawljax, apesar de também ter as suas limitações, fornece alguns plugins que permitem lidar com este problema.

Por fim, Autocrawler (Li, Han, Liu, & Fang, 2018), um crawler que constrói sobre os conceitos apresentados pelo Crawljax, adicionando mecanismos que conseguem detetar se existem "pop-up windows" e invocando CSS "pseudo-classes". Autocrawler responde a 2 problemas que surgem quando se tenta explorar aplicações web: (1) "pop-up windows" que surgem e impedem que sejam invocados eventos sem que primeiro se lide com a "pop-up window", e (2) eventos CSS em "pseudo-class". Neste caso em particular, pretende-se invocar a função ":hover", a qual não é possível invocar com auxílio de JavaScript. De forma a conseguir lidar com as "pop-up windows", Autocrawler cria um estado para este elemento. Este estado é depois tratado como outro qualquer em que se identifica e se invocam todos os eventos possíveis, assim explorando o máximo de estados possíveis, até que é invocado o evento que fecha a "pop-up window" e se prossegue com a exploração dos restantes estados. O problema de invocar ações de "pseudo-class" acaba por ser resolvido através de um servidor de proxy entre o browser e a aplicação web. Este proxy faz uma análise do código recebido da aplicação web de forma a detetar se existe alguma invocação da função ":hover". Caso exista, é colocada uma função *hook* que substitui a invocação da função ":hover" por uma "OnClick", sendo que esta consegue ser invocada com auxílio de JavaScript.

Estas ferramentas, não correspondem ao objetivo deste trabalho, pois são muito limitadas nos eventos e interações que conseguem realizar, limitando os estados que conseguem atingir. É necessária uma abordagem que consiga simular de forma mais extensa as interações que um utilizador consegue realizar, assim explorando mais estados.

3.1.2. Crawlers Semiautomáticos

Crawlers semiautomáticos apresentam outra vertente nesta componente, em que é necessário a intervenção do utilizador.

Juntamente com as ferramentas acima mencionadas, existe também outra vertente em que um crawler pode ser utilizado como intermediário para atingir o resultado pretendido. AWET (Automated web application testing based on Exploratory Testing) (Sunman, Soydan, & Sözer, 2022) é uma ferramenta que auxilia no desenvolvimento de testes para aplicações web de forma automática. AWET utiliza um crawler com direções para explorar o maior número possível de estados da aplicação web e criar mais testes de forma automática. Esta ferramenta necessita inicialmente de um conjunto pré-definido de testes como input. É com estes testes que a direção do crawler acaba por ser determinada. Os testes disponibilizados são depois analisados de forma a retirar inputs, interações, e a sua ordem de execução para assim criar regras e estratégias para aplicar no crawler, assim explorando mais estados, evitando um processo desorganizado e aleatório. Esta abordagem, apesar de apresentar bons resultados comparativamente com outras ferramentas, sai um pouco do objetivo deste trabalho.

Por outro lado, existe GUIDE (Liu & Woei-Kae and Sun, 2020), novamente uma abordagem semiautomática que conta muito com a participação do utilizador. GUIDE trata-se de um crawler interativo que vai ativamente perguntando ao utilizador que valores de input deve utilizar em diferentes cenários, permitindo assim que não seja necessário o utilizador ter conhecimento da aplicação web. Durante o processo de crawling, se o crawler chegar a um estado no qual existe forma de progredir, porém sendo necessária a intervenção do utilizador, a ferramenta irá perguntar ao

utilizador como deve proceder apresentando screenshots em que os campos de input estão destacados. O utilizador pode então definir um ou mais valores que o crawler deve utilizar. A utilização de diversos valores permite criar diferentes diretivas, ou seja, diferentes opções que o crawler terá de percorrer. Este processo vai se repetindo até não existirem mais estados ou até o utilizador concluir o processo.

Estas abordagens, apesar de apresentarem bons resultados comparativamente com outras ferramentas, não se enquadram dentro do objetivo deste trabalho, pois o objetivo passa por ter uma ferramenta completamente automática, em que a única interação com o utilizador é para receber um conjunto de configurações.

3.2.Comparador de estados

Um aspeto crucial num crawler é a capacidade de comparar estados de forma a garantir que o presente estado difere de todos os estados anteriores.

Autocrawler apresenta uma versão simples para comparar estados. Considerando o HTML com uma string, esta é depois comparada com o HTML dos outros estados encontrados, utilizando o conceito de "Edit Distance". Este conceito calcula o mínimo de alterações que são necessárias realizar a uma string para esta ficar igual à outra (F Laender, Altigran, Paulo, & C Reis, 2004).

Demodocus faz a comparação de estados a utilizar com "Three-Stage Comparator Pipeline". O primeiro passo consiste em transformar o HTML numa string e comparar com todos os estados já identificados. Se o resultado mostrar que a string difere, então estamos perante um estado novo. O segundo passo consiste em comparar o DOM de cada estado. A comparação foca-se na estrutura do DOM de forma a verificar se foram adicionados ou retirados nós. Caso a comparação apresente diferenças, o estado é novo. O último passo, utiliza a técnica "Levenshtein", que compara a distância entre os nós com texto entre os diferentes estados.

3.3.Avaliador de estados

Como referido acima, a união de um crawler para obter estados e a avaliação de acessibilidade dos mesmo é um conceito relativamente novo. Como tal, não existem muitas ferramentas que façam os dois sem que seja necessário alterações ou a utilização de outras ferramentas auxiliares.

O Demodocus (Bostic, et al., 2021), mais do que realizar um crawl para obter os diversos estados, realiza também uma avaliação de acessibilidade. Para fazer a avaliação de acessibilidade são utilizados user models, representações de utilizadores que simulam a interação de utilizadores com diferentes tipos de deficiência, ou requerimentos de diretrizes de acessibilidade.

Inicialmente, na construção do grafo de estados, é utilizado um "omniuser", que consegue invocar todo o tipo de interações e assim descobrir o maior número de estados possível, concluído este processo, tem se agora um "omnigraph".

Seguidamente, são construídos diversos subgrafos com auxílio de user models. Com este subgrafo consegue-se verificar se existe algum estado que um utilizador com deficiência não conseguiria atingir e, assim, confirmar se existem ou não problemas de acessibilidade. Como todas as outras ferramentas, os problemas ou erros identificados podem depois ser corrigidos.

3.4.Discussão

No fim da fase de investigação, consegue-se ter ideias mais detalhadas do objetivo do QualState. Esta ferramenta aponta para ser completamente automática, sendo que única interação com o utilizador é receber um conjunto de configurações. O QualState aponta ainda para ser uma ferramenta de exploração com mais opções de configuração, de forma a permitir o utilizador executar uma exploração que vá mais de encontro com as suas necessidades, um sistema de exploração mais extenso que lida com um conjunto maior de elementos HTML, diferentes tipos de eventos, e simplicidade de utilização, comparativamente às ferramentas apresentadas.

4.Metodologia

4.1.Metodologia de desenvolvimento

A metodologia de desenvolvimento adotada nesta tese de mestrado abraçou um processo de desenvolvimento iterativo e ágil. Reuniões semanais com os orientadores, apresentando avanços da semana anterior e discutindo as próximas etapas. Objetivos de alto nível, semelhantes a Epics no contexto do Scrum, foram estabelecidos inicialmente, e semanalmente metas mais específicas foram definidas com base na exploração em curso. Esta abordagem promoveu a adaptabilidade e uma dinâmica responsiva ao longo das fases de pesquisa e desenvolvimento, proporcionando um ambiente propício à inovação e refinamento contínuo das metas.

4.2.Metodologia de testes

Na fase dedicada à metodologia de testes deste trabalho de mestrado, o processo iniciou-se com a identificação de um conjunto de aplicações web passíveis de serem testadas, incluindo a construção de uma aplicação web utilizando Vue.js, que permitiu o controle de diversos aspetos a serem avaliados. Outras aplicações foram selecionadas, abrangendo diferentes níveis de complexidade, para avaliar a capacidade da solução em analisar aplicações web do mundo real. Após a definição do conjunto de aplicações para testes, foram estabelecidos critérios de avaliação, contemplando tanto aspetos de desempenho quanto eficácia na deteção de barreiras de acessibilidade. Os testes foram então executados, os resultados analisados e devidamente documentados neste presente trabalho.

5.QualState

Este capítulo serve para apresentar a solução proposta, o QualState. QualState é um módulo independente que utiliza a framework Puppeter para conseguir controlar o browser e executar um conjunto de ações, que tem como objetivo explorar uma SPA e encontrar o maior número possível de diferentes estados. Devido à complexidade que existe quando se faz uma exploração de estados, o utilizador tem a possibilidade de fornecer umas algumas configurações de forma a personalizar o tipo de exploração que pretende realizar.

Uma das metas deste trabalho é, para além de desenvolver uma solução que consiga obter o maior número de estados possíveis, obter também uma avaliação de acessibilidade dos mesmos. Desta forma, foi necessário integrar o QualState com o QualWeb. Como tal, ao utilizar o QualWeb, o utilizador pode, se desejar, configurar o QualWeb para utilizar o QualState e assim ter uma avaliação de acessibilidade de todos os estados encontrados.

Este capítulo está dividido em 6 partes. Inicialmente será apresentado o contexto e partes crucias do QualState. Logo depois serão introduzidas as diversas condições e restrições sobre as opções que o utilizador pode inserir nas configurações. Seguidamente será explicada a arquitetura do QualState, seguida do fluxo de informação e a forma como o processo funciona do princípio ao fim. Logo depois serão apresentas algumas limitações conhecidas no QualState. Este capítulo termina com as alterações que foram realizadas no QualWeb de forma a conseguir integrar o QualState.

5.1.Contexto

Esta secção serve para introduzir o QualState, apresentar uma versão simplificada do seu funcionamento e alguns conceitos crucias. Como foi referido, o QualState trata-se de uma ferramenta que serve para explorar SPAs. Inicialmente o QualState começa por carregar uma página cujo URL foi fornecido pelo utilizador. Depois da página ser carregada são realizadas ações com o intuito de gerar novos estados. Depois de cada ação é verificado se o estado atual já foi encontrado ou não e consequentemente guardado. Este processo repete-se durante todo o processo do QualState até não existirem mais ações ou até o número máximo de estados ser atingido.

5.1.1. Ação

O conceito de ação é um elemento crucial no QualState. A ação representa, como o nome indica, uma ação que se deve realizar. A ação é um objeto JSON que contém informação de como uma instrução deve ser realizada. Existem dois tipos de ações: ação automática, identificada como “auto”, e ação manual, identificada como “manual”. Estas distinguem-se pela forma como são capturadas: ações automáticas são as que são encontradas no módulo **events**, ou seja, aquelas encontradas pelo QualState numa determinada página, enquanto as ações manuais são aquelas adicionadas pelo utilizador, com auxílio da opção **interactions**. Estas ações, no entanto, ainda necessitam de ser processadas. Este processamento é realizado no módulo **interactions**.

A estrutura das ações manuais irá depender da informação que o utilizador fornece. No entanto, a estrutura completa tem 4 propriedades: user, beforeAction, endAction e wait:

- **user:** Informação gerada pelo QualState que indica se a ação é manual ou automática.
- **beforeAction:** Ação que deve ser realizada antes do **endAction,** associado a realizar algum tipo de input que o utilizador pense ser necessário ou que queira testar.
- **endAction:** A última instrução a ser realizada, associada a um tipo de evento que o utilizador quer que seja realizado sobre um elemento da página.
- **wait:** Tempo de espera a ser cumprido depois de qualquer instrução realizada dentro da ação.

A Figura 5.1 ilustra uma ação manual completa.

```
{
  user: 'manual',
  wait: 2000,
  beforeAction: { fname: 'filipe', mname: 'martins' },
  endAction: { id: 'btnSubmit', eventType: 'click' }
}
```

Figura 5.1 - Estrutura de uma ação manual.

A ação automática tem sempre a mesma estrutura, pois é dependente apenas do QualState, e não de um input fornecido pelo utilizador. Uma ação automática é constituída por cinco propriedades: user, id, className, eventType:

- **user:** Informação gerada pelo QualState que indica se a ação é manual ou automática.
- **id:** Id do elemento.
- **className:** Nome da classe do elemento.
- **eventType:** Tipo de evento que deve ser realizado.
- **selector:** Seletor do elemento.

A Figura 5.2 ilustra uma ação automática completa.

```
{
  user: 'auto',
  id: 'button#btnId',
  className: 'HTMLButtonElement',
  eventType: 'click',
  selector: 'html > body:nth-child(2) > button:nth-child(7)'
}
```

Figura 5.2 - Estrutura de uma ação automática.

5.1.2. Queue

De forma a conseguir limitar o número de processos que ocorrem em simultâneo, o QualState utiliza um módulo assíncrono (*async)* que permite criar uma queue. Esta queue permite que uma ou mais execuções aconteçam de forma concorrente, tornando o processo mais rápido e mais fácil de gerir.

Uma queue trata-se de uma estrutura de dados linear que aceita elementos no formato FIFO (First In, First Out), ou seja, o primeiro elemento a entrar é o primeiro a sair, ou no caso do QualState, o primeiro elemento a entrar é o primeiro a ser executado.

5.1.3. Guardar e comparar estados

O conceito de estado já foi abordado, mas faz sentido explicar a forma como estes conceitos são abordados dentro do QualState. Durante o processo de crawling, os estados têm de passar por duas fases, comparação e persistência.

O QualState mantém a qualquer momento dois sets, uma para estados originais e outro para estados depois de serem alterados, STATES_SPA_EVALUATING (SSE) e STATES_SPA_COMPARE(SSC) respetivamente. O motivo de ter dois sets surge devido à possibilidade que o utilizador tem de ignorar elementos na comparação. É importante ressaltar que, na prática, estes elementos não são ignorados, mas simplesmente removidos, e, como tal, é necessário ter dois sets para conseguir gerir esta funcionalidade. Este processo irá ser mais detalhado na secção 5.4.

5.2.Configurações do QualState

Como referido, o utilizador tem possibilidade de fornecer um conjunto de configurações de forma a personalizar a exploração de estados e/ou adicionar instruções que devem ser executadas durante este processo. Estas configurações são uma peça crucial do QualState.

A Figura 5.3 representa as configurações na sua totalidade, com todas as opções preenchidas. Esta secção irá explicar as diversas configurações e o porquê das mesmas.

Figura 5.3 - Estrutura das configurações do QualState.

url: String que representa o URL da SPA que utilizador deseja explorar. Esta opção é a única configuração obrigatória. A Figura 5.4 representa a estrutura da configuração **url**.

Figura 5.4 – Estrutura da configuração **url.**

headless: String que representa o valor **headless** a utilizar no. Caso esta opção não seja indicada pelo utilizador, o valor true é utilizado. A Figura 5.5 representa a estrutura da configuração **headless**.

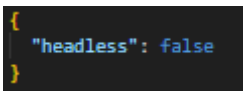

Figura 5.5 - Estrutura da configuração **headless**.

waitTime: Número apresentado em milissegundos que o utilizador quer que seja esperado antes e depois de qualquer ação realizada. O QualState, durante o seu processo, sempre que carrega uma página, aguarda que a mesma seja carregada e só depois continua com o processo. No entanto, como já foi referido, é comum uma SPA já ter toda a informação presente e apenas transformar a página. Como tal, aguardar que a página seja carregada não irá ter efeito, pois a página já foi carregada. Por isso surge a necessidade de indicar, de maneira manual, se o processo deve esperar ou não, entre a execução de ações. A Figura 5.6 representa a estrutura da configuração **waitTime**.

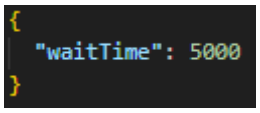

Figura 5.6 - Estrutura da configuração **waitTime.**

maxStates: Número máximo de estados que o utilizador deseja que sejam encontrados. No momento que o número máximo é atingido o processo é terminado. É importante também relembrar que este número não considera o estado inicial da página. A Figura 5.7 representa a estrutura da configuração **maxStates**.

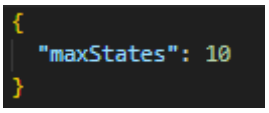

Figura 5.7 - Estrutura da configuração **maxStates.**

numberOfProcess: Número de processos que acontecem de forma concorrente. Caso esta opção não seja indicada pelo utilizador, o valor 1 é utilizado. De forma a tornar o processo de exploração mais rápido, existe a possibilidade de ter um ou mais processos a correr de forma concorrente. A Figura 5.8 representa a estrutura da configuração **numberOfProcess** .

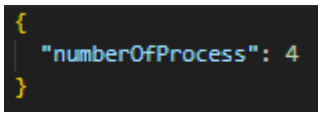

Figura 5.8 - Estrutura da configuração **numberOfProcess.**

viewport: O Puppeter permite que o viewport do browser seja ajustado de acordo com as especificações do utilizador. **Viewport** é um objeto que contem as seguintes propriedades: **mobile**, **landscape**, **userAgent** e **resolution**. Nem todos as propriedades são requeridas. Caso sejam omitidas, estas assumem o valor default. **Mobile** assume o valor de false, **landspace** assume o valor de true, enquanto **width** e **height** assumem o valor de 0. A Figura 5.9 representa a estrutura da configuração **viewport**.

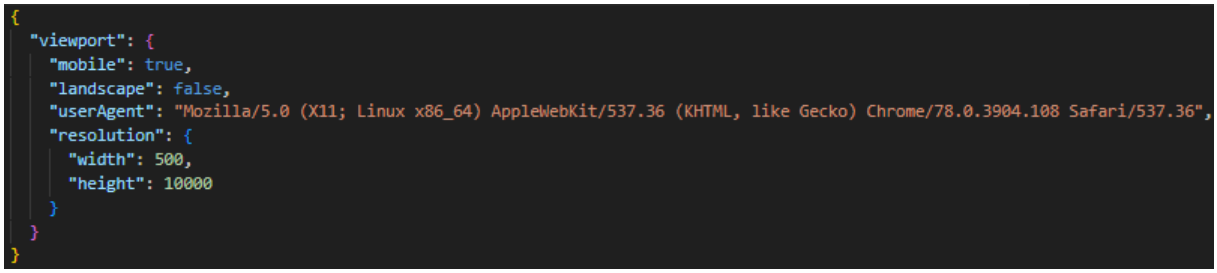

Figura 5.9 - Estrutura da configuração **viewport.**

log: O utilizador tem a opção de escolher onde quer que as informações resultantes da exploração sejam apresentadas. Como tal, existe o objeto **log** que tem duas propriedades booleana, **file** e **console,** sendo que o **file** representa se deve ou não imprimir informação num ficheiro de output, enquanto o **console** representa se a informação deve ou não ser imprimida para a consola. Por default, a informação é apenas enviada para os ficheiros de output. A Figura 5.10 representa a estrutura da configuração **log**.

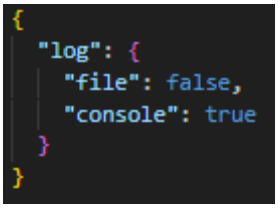

Figura 5.10 - Estrutura da configuração **log.**

i**gnore**: QualState também fornece ao utilizador a possibilidade de ignorar certos elementos presentes na página. A opção de ignorar apresenta duas propriedades: **ids_events**, que serve para ignorar elementos que tem eventos, e **ids_compare**, que serve para ignorar elementos que o utilizador não quer que sejam comparados. Ambas as opções têm a estrutura de um array de strings, e aceita o id de elementos ou qualquer valor que o elemento tenha como atributo. A Figura 5.11 representa a estrutura da configuração i**gnore**.

A opção de ignorar eventos, **ids_events**, ignora todos os elementos que tenham eventos associados na página. Esta configuração permite ao utilizador escolher quais eventos não devem ser ativados.

A opção de ignorar elementos para a comparação**, ids_compare**, surge com o intuito de não avaliar elementos que sejam repetidos ou elementos que estão constantemente a mudar e a alterar o estado. Por exemplo, um componente que apresenta as horas até aos milissegundos, sempre que comparado com estados anteriores, irá resultar num estado diferente. Neste caso, a única alteração no HTML foi nas horas, não adicionando nada relevante aos estados explorados nem à avaliação de acessibilidade. De notar que o estado que é guardado para apresentar ao utilizador tem todos os elementos originais.

Figura 5.11 - Estrutura da configuração **ignore.**

interaction:

Como mencionado anteriormente, crawlers que são limitados a “carregar” em botões tornam-se muito limitados no nível de pesquisa e procura que conseguem realizar, pois, em alguns casos, pode ser necessário algum tipo de interação de forma a conseguir expandir a procura de estados.

De forma a facilitar este processo, o QualState disponibiliza ao utilizador a possibilidade de fornecer um conjunto de dados e informações pertinentes para a pesquisa a ser realizada.

O intuito destas instruções é dar a possibilidade ao utilizador de colocar o máximo de inputs que este pense ser adequados, de forma a permitir à ferramenta explorar o maior número de estados relevantes. Estes estados podem também surgir através de input que geram erros, como por exemplo inserir uma palavra-passe errada ou valores numéricos que excedem o limite estabelecido.

Esta instruções devem ser fornecidas no objeto **interactions**. De forma a distinguir o tipo de instrução, este objeto apresenta 3 objetos distintos: **forms**, **inputs** e **directions**.

forms: Este objeto serve para adicionar ao processo uma ou mais ações associadas a formulários que estejam presentes durante o processo de crawl que é realizado. A Figura 5.12 representa a estrutura da configuração **forms**.

O objeto **forms** trata-se de um array que contém um objeto com 3 propriedades:

input: O objeto **input** é um array de objetos em que cada objeto contém uma ou mais relações propriedade valor. A propriedade é o id de um elemento e o valor corresponde ao valor que o utilizador deseja que seja inserido quando se preenche o formulário. Sempre que o objeto **input** está presente é obrigatório ter pelo menos uma relação propriedade valor.

action: O objeto **action** serve para que o utilizador possa informar o elemento na página e o tipo de evento que deve ser realizado sobre o elemento de forma que possa ser despoletado algum tipo de resultado. O objeto **action** requer sempre as propriedades **id**, string que deve ser o id do elemento, e **event**, o tipo de evento que deve ser realizado sobre o elemento com o id que o utilizador fornece.

info: O objeto **info** serve para que o utilizador possa informar sobre o id do formulário, de forma que se consiga identificar o elemento que deve utilizar a informação fornecida nas propriedades **input** e **action**. O objeto **info** contém ainda a propriedade **wait**, um número em milissegundos que representa o tempo que o processo deve esperar depois de qualquer ação dentro da propriedade **form** ser executada. O objeto **info** requer apenas a propriedade **formId**.

Cada objeto dentro do array **forms** tem de corresponder aos mínimos de campos obrigatórios: [**input**, **info**], [**action**, **info**] ou [**input**, **action**, **info**].

Figura 5.12 - Estrutura da configuração **forms.**

inputs:

O objeto **inputs** serve para que o utilizador possa ter uma maneira de conseguir lidar com inputs que possam surgir durante o processo de crawl. Desta forma o utilizador consegue testar valores que podem ser corretos ou incorretos, assim encontrando estados que surgem ao colocar input incorreto e estados que surgem quando o input correto é colocado. A Figura 5.13 representa a estrutura da configuração **inputs**.

Inputs é um array de objetos, em que cada objeto contém um objeto **value** e um objeto **info**.

Value: O objeto **value** contém uma ou mais relações propriedade valor. A propriedade é o id de um elemento e o valor corresponde ao valor que utilizador deseja que seja preenchido o input correspondente. Sempre que o objeto **inputs** esta presente é obrigatório ter pelo menos uma relação propriedade valor.

Info: O objeto **info** contém a propriedade **wait**, um número em milissegundos que representa o tempo de espera que ocorre sempre que o input é utilizado. O objeto **info** requer apenas a propriedade **wait**.

O objeto presente no array de **inputs** requer sempre o objeto **value**.

Figura 5.13 – Estrutura da configuração **inputs.**

directions:

Durante o processo de crawl, o utilizador pode sentir que não existe a necessidade de explorar certos caminhos, preferindo então utilizar o seu tempo e recursos para tentar atingir outros estados, eventualmente desconhecidos, que este tenha mais interesse. Como tal, o QualState disponibiliza ao utilizador a opção de fornecer instruções que direcionam o crawl para o caminho desejado. A Figura 5.14 representa a estrutura da configuração **directions**.

O objeto **directions** é um array de objetos, em que cada objeto é constituído por dois objetos: **actions** e **info**.

actions: **actions** é um array de objetos que contem instruções sobre como o processo de crawl deve prosseguir. Cada objeto é constituído por 3 objetos: **values**, **action** e **info**.

values: O objeto **values** é um objeto que contém uma ou mais relações propriedade valor. A propriedade é o id de um elemento e o valor corresponde ao valor que utilizador deseja que seja inserido quando utilizado. Sempre que o objeto **values** está presente é obrigatório ter pelo menos uma relação propriedade valor.

action: O objeto **action** serve para que o utilizador possa informar o elemento na página e o tipo de evento que deve ser realizado sobre o elemento de forma que possa ser despoletado algum tipo de resultado. O objeto **action** requer sempre as propriedades **id**, string que deve ser o id do elemento, e **event**, o tipo de evento que deve ser realizado sobre o elemento com o id que o utilizador fornece.

info: O objeto **info** contém a propriedade **wait**, um número em milissegundos que representa o tempo que o processo deve esperar depois de qualquer ação que é executada. O objeto **info** requer apenas a propriedade **wait**.

info: O objeto **info** contém informação sobre como o processo deve interagir quando uma direção é executada. O objeto contém dois atributos: **crawl** e **save**. O atributo **crawl** indica se o processo de crawl deve ou não continuar depois de chegar ao fim de uma direção. Este atributo assume dois possíveis valores: “stop” e “continue”. O atributo **save** indica se os

estados que são alcançados depois de cada execução durante o processo da direção devem ou não ser guardados. Este atributo assume o tipo booleano.

Figura 5.14 - Estrutura da configuração **directions.**

5.3. Arquitetura do QualState

Esta secção explica a arquitetura do QualState. Esta arquitetura tem como objetivo ser o mais modular possível, sendo assim uma solução mais escalável, fácil de reutilizar, e de manter. O QualState está divido em sete módulos: actions, events, html, interactions, logger, schema, utils e qualstate. A Figura 5.15 representa a arquitetura do QualState.

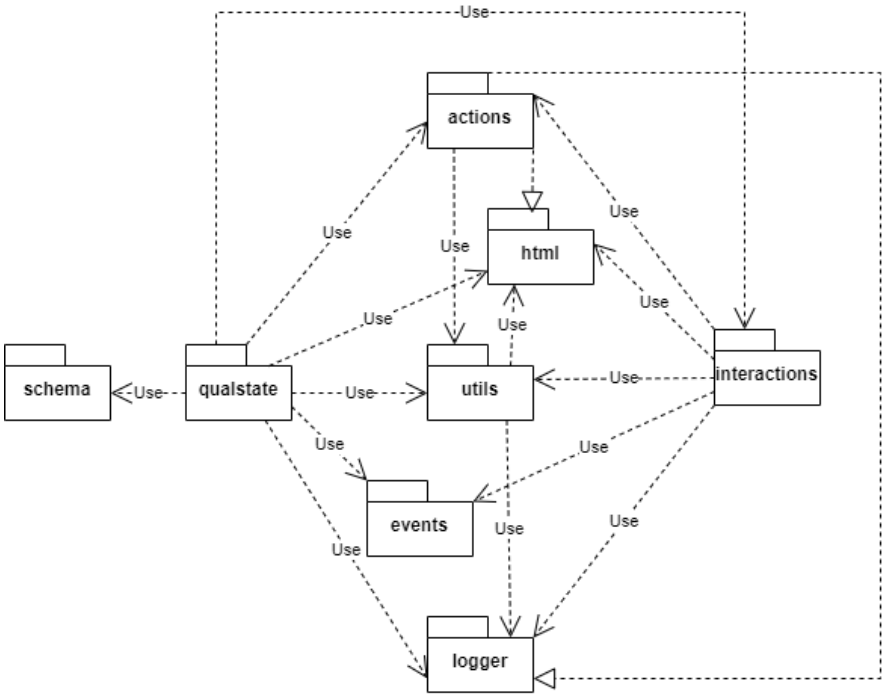

Figura 5.15 - Arquitetura do QualState.

5.3.1. Events

O módulo **events** está encarregue de encontrar todos os eventos possíveis numa determinada página, de forma a conseguir devolver um conjunto de ações a serem realizadas. A obtenção de eventos é executada em duas partes.

A primeira parte utiliza a função *DomDebugger,getEventListeners()* para todos os elementos presentes na página através do protocolo Chrome Dev Tools de forma a conseguir verificar se um elemento tem ou não um evento, e, em caso positivo, qual o tipo de evento.

A segunda parte trata-se de um processo de triagem feita para todos os elementos da página. Esta triagem filtra todos os elementos *a*, *input type = 'submited'*, pois mesmo que não tenham um evento associado são processados com se tivessem um evento *click*. Isto porque a natureza destes elementos permite que os mesmos, ainda que não tenham eventos adicionados explicitamente, tenham eventos implícitos que podem ser invocados e eventualmente gerar um estado novo.

Por fim, e depois de encontrar todos os elementos, estes são processados e cada um é transformado numa "ação".

5.3.2. Actions

O módulo **actions** é encarregue de realizar um conjunto de ações necessárias para chegar a um estado final. Para tal, o método *performAction()* é disponibilizado. O método recebe um array de ações que servem de instruções de como devem ser realizadas as ações.

O método *performAction()* realiza ações manuais e automáticas. Para as ações automáticas, é realizado um evento, enquanto para as ações manuais existe possibilidade de realizar ações de inputs.

5.3.3. HTML

De forma a conseguir identificar elementos numa determinada página é necessário ter um id único e identificável. No contexto deste projeto existem 2 tipos de ids: id de um elemento, fornecido pelo utilizador, e selector, gerado pelo QualState, uma string que permite identificar um elemento numa página. Para existir controlo total sobre elementos, o QualState faz a gestão dos seletores de todos os elementos.

O módulo **HTML** está encarregue de interagir com a página de forma a poder adicionar ou retirar o atributo *_selector*. Este módulo é constituído por dois métodos: *createSelectorOnPage()*, que calcula o seletor e gera um novo atributo *_selector* para todos os elementos presentes na página, e *removeSelectorOnPage()*, que apaga o atributo *_selector* de todos os elementos presentes na página.

5.3.4. Interactions

O módulo **interactions** processa as instruções referente a interações que o utilizador quer que aconteçam. Estas instruções são colocadas na propriedade **interactions** nas configurações do QualState:

Inputs:

O módulo **inputs** tem duas responsabilidades, verificar se o input existe na página, e processar o mesmo numa ação manual.

Inicialmente, o módulo recebe o conjunto de inputs que o utilizador forneceu através da opção **inputs** e a página em que o processo está no presente momento.

Seguidamente verifica-se se todos os elementos do conjunto de inputs estão presentes na página. Caso exista, o processo continua para a fase seguinte, e processa o input para uma ação manual. Caso se verifique que pelo menos um dos inputs não exista na página, o processo é terminado.

O **inputs** apresentado no secção 5.2 iria resultar numa ação manual, representado na Figura 5.16.

```
[
  {
    user: 'manual',
    beforeAction: { onChangeInput: 'inputOnChange' },
    wait: 2000
  }
]
```

Figura 5.16 - Ação manual resultante do input apresentado na secção 5.2.

Forms:

O módulo **forms** é responsável por processar a opção **forms**. Este módulo processa a informação de uma forma bastante semelhante ao módulo **inputs**. Inicialmente, o módulo recebe o conjunto de formulários que o utilizador inseriu na opção **forms,** e a página em que o processo está no presente momento.

Seguidamente verifica se o formulário existe na página, verificando se o id do elemento existe na página. Caso exista, o processo continua para a fase seguinte, processando o formulário para uma ação manual.

O **forms** apresentado na secção 5.2 iria resultar numa ação manual, representado na Figura 5.17.

```
{
  user: 'manual',
  beforeAction: { fname: 'filipe' },
  endAction: { id: 'btnSubmit', eventType: 'click' }
}
```

Figura 5.17 - Ação manual resultante do form apresentado na secção 5.2.

Directions:

O módulo **directions** tem a responsabilidade de receber e executar as **directions** que o utilizador fornece nas opções. O processo passa por interpretar executar cada instrução como uma ação. Este processo continua para todas as direções, criando assim uma ou mais ações manuais.

5.3.5. Logger

O módulo logger é encarregue de registar todos os *logs* que decorrem durante o processo. Os *logs* podem ser feitos para a consola ou para ficheiros. Toda a informação apresentada é dividida em duas categorias: **info** e **error**.

Os *logs* da consola apresentam a informação na consola, sendo que esta é distinguida pela categoria do tipo de informação. Caso seja **info** é utilizado o método *console.log()*. Caso a categoria seja **error**, é utilizado o método *console.error()*.

Os *logs* são repartidos por dois ficheiros: “logs.log” e “states.log”. O ficheiro “logs.log” armazena toda a informação sobre as ações realizadas, independentemente de estas serem da categoria

info ou **erro**, através de uma hash que representa o estado alcançado. O ficheiro "states.log" armazena novamente uma hash que representa o estado e a string completa do estado.

5.3.6. Schema

O módulo **schema** é encarregue de fazer a validação das configurações do QualState. Com auxílio do Ajv[8], um validador de esquemas JSON, as opções são validadas contra um JSON que contém todas as regras e condições de validação. As regras sobre as configurações estão explicadas de forma mais extensiva na secção 5.2.

5.3.7. Utils

O módulo **utils** contém um conjunto de funções utilizadas por diversos módulos. Deste módulo destacam-se diversos métodos que interagem com a página do browser e o método de comparação de estados.

5.3.8. QualState

O módulo **qualstate** funciona como cérebro de toda a operação. É neste módulo que o processo é iniciado e terminado. É também neste módulo que são delegadas todas as tarefas, entre as quais pode-se destacar: 1) validar as opções dadas pelo utilizador, 2) procurar eventos, 3) gerir a forma como os **forms**, **inputs** e **directions** são controlados, e, por fim, 4) realizar a comparação de estados.

5.4.Funcionamento do QualState

O QualState tem duas formas de ser iniciado: pode ser iniciado por um script ou através de CLI (command-line interface). Em ambos o caso é necessário ter um conjunto de configurações. Na utilização de CLI é necessário passar a localização do ficheiro de configurações. Caso a utilização seja através de um script é necessário um objeto JSON com as configurações do utilizador. Independente da forma como o processo é iniciado, o fluxo de funcionamento é igual para ambos os casos. O funcionamento do QualState está apresentado na Figura 5.18.

[8] https://ajv.js.org/

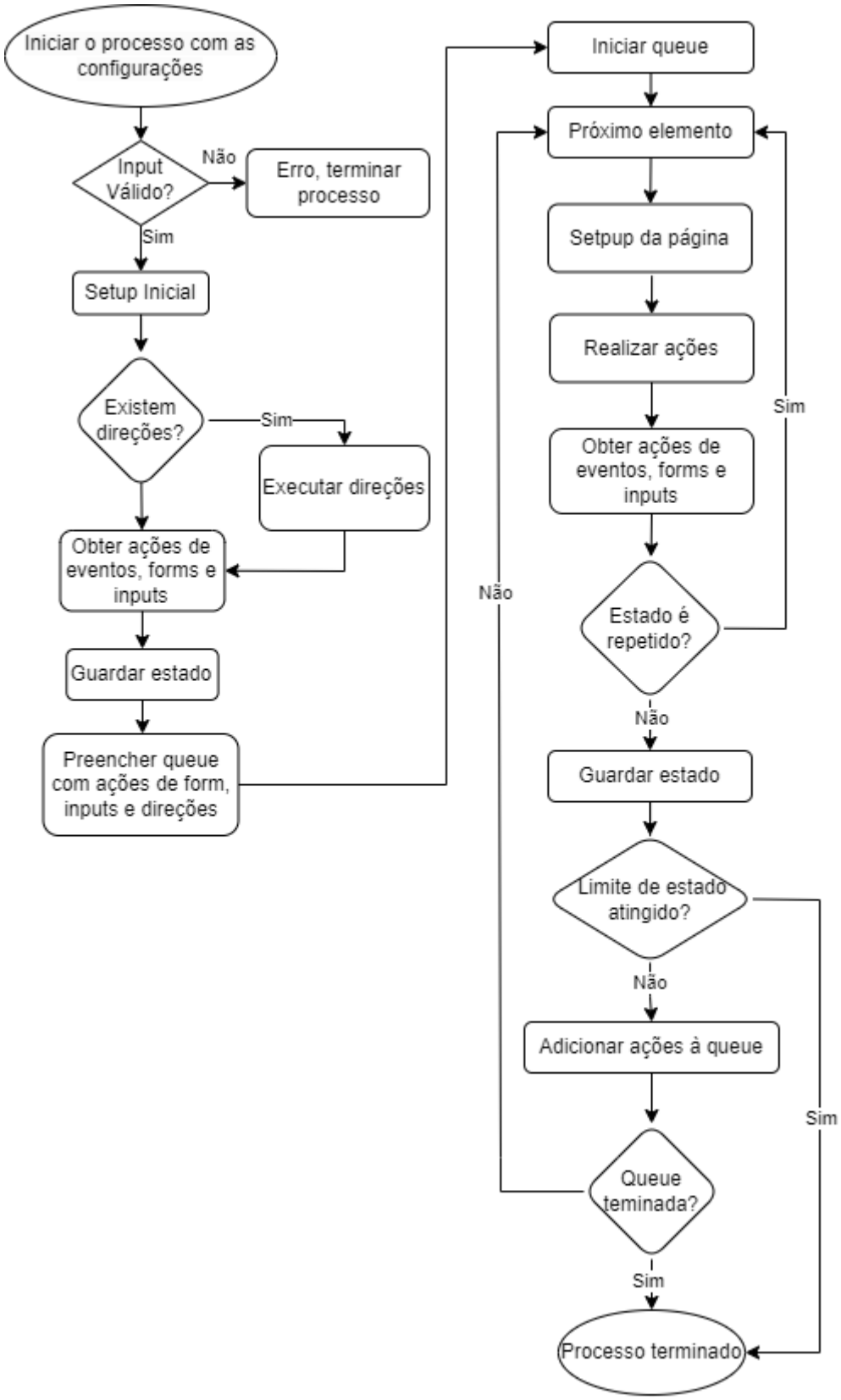

Figura 5.18 - Processamento do QualState.

Em princípio, depois de ser iniciado o processo, a primeira tarefa é verificar se as configurações estão corretas. Na eventualidade de existir algum problema com a estrutura das configurações ou com alguma opção incorreta, é apresentada uma mensagem de erro e o processo é imediatamente terminado.

Depois de validar as configurações é feito o setup inicial. Este inclui inicializar as variáveis e iniciar o browser com uma página do URL que o utilizador disponibilizou nas configurações. É importante relembrar que o browser pode ser lançado com o modo *headless* a true ou false. Independentemente de qual seja a configuração, o mesmo tem de ser iniciado.

Seguidamente, verifica-se se existem direções que devem ser executadas. Em caso positivo, é iniciado um processo secundário que percorre as mesmas, este processo está apresentado na Figura 5.19.

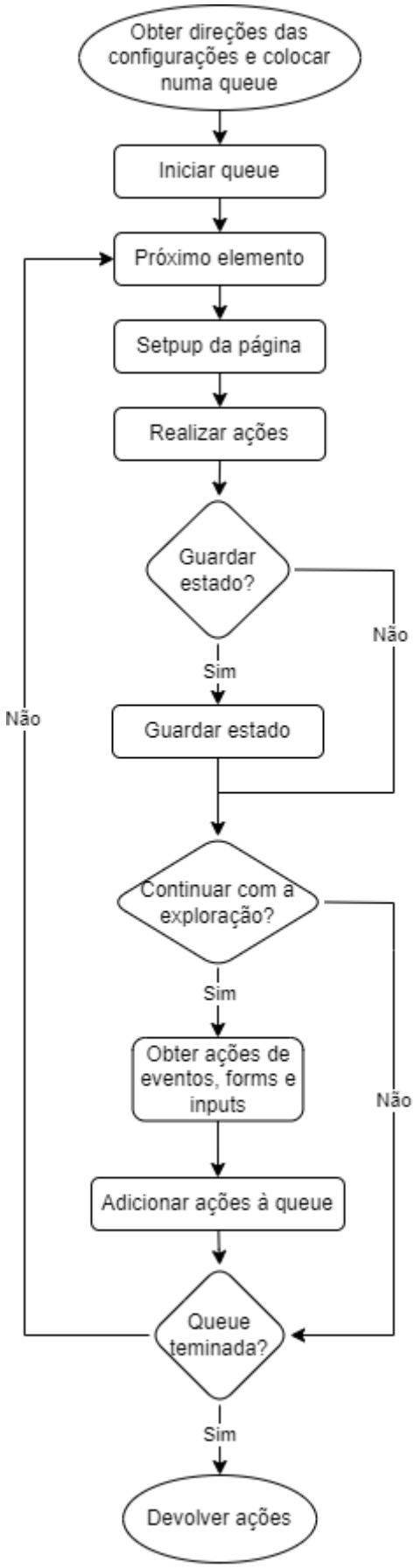

Figura 5.19 - Processamento das direções.

O processo passa por criar uma queue só para as direções antes de dar início ao processo de crawl. Durante o processo é criado um conjunto de ações manuais por cada direção que devem ser executadas. No fim de cada ação manual é verificado se o utilizador deseja ou não guardar o estado através da configuração **save** presente na configuração **directions**. Caso esta opção seja **true**, então o estado é guardado, caso seja **false,** o estado não é guardado.

O processo de guardar o estado passa por inicialmente obter o estado completo na forma de uma hash com auxílio do módulo crypto-js[9]. Seguidamente verifica-se se existem elementos que devem ser ignorados, e, em caso positivo, estes elementos são removidos da página. Logo de seguida, independentemente se foram ou não retirados elementos, obtém-se novamente o estado da página na forma de hash. Esta hash é depois comparada com as restantes que existem no set SSC. Caso a hash seja repetida, quer dizer que estamos perante um estado repetido, como tal, o processo termina. No entanto, se a hash não existir, então neste caso estamos perante um estado novo. Nesse contexto, este tem de ser persistido, colocando-se a hash alterada no SSC e a hash original no SSE junto de uma versão do estado em string. O processo de guardar o estado está apresentado na forma de fluxo na Figura 5.20.

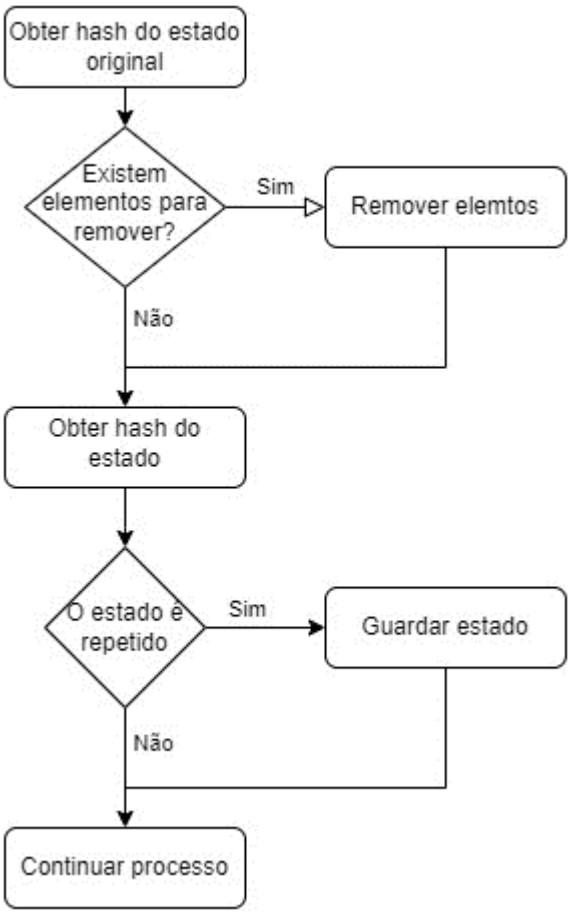

Figura 5.20- Processamento de guardar o estado.

[9] https://www.npmjs.com/package/crypto-js

No fim de executar o conjunto de direções, verifica-se a condição **crawl.** Caso seja **continue**, obtém-se os eventos, formulários e inputs da presente página. Caso seja **stop**, o processo termina. Este processo repete-se até a queue de direções seja concluída.

Após a execução das direções, obtém-se um conjunto de ações automáticas dos eventos e as ações manuais dos formulários e inputs. Seguidamente guarda-se o estado e preenche-se uma nova queue com todas as ações obtidas previamente.

Depois de preencher a queue é iniciada a exploração da SPA - para cada elemento da queue é realizado o setup da página. Este setup inclui criar uma aba dentro do browser e, caso exista a configuração **viewport**, esta é utilizada para definir o **viewport** da aba. Depois do setup da página ser realizado, são executadas as ações do presente elemento. Concluídas as ações, sejam estas manuais ou automáticas, são obtidas as novas ações do estado. Estas ações incluem as ações automática, ou seja, todos os eventos presentes na página atual, e as ações manuais, ou seja, os formulários e inputs que podem ou não existir na página.

O passo seguinte é verificar se o estado já existe e, consequentemente, decidir se o mesmo deve ou não ser guardado. O processo de confirmar se o estado existe passa por, inicialmente e caso existam elementos para remover, remover os mesmos e depois confirmar se o estado é ou não repetido, comparando-o com os estados guardados até o momento. Caso o estado não seja repetido então o mesmo é guardado e, em caso contrário, o mesmo não é guardado. Em ambos os casos o processo continua para a fase seguinte.

O último passo de cada execução dentro da queue passa por popular a queue com as novas ações encontradas na página, assim inserindo mais elementos à queue que irá continuar o seu processo até não ter mais elementos.

Durante o percorrer da queue, a mesma tem apenas uma condição de paragem: se em algum momento o número máximo estados for atingido, então a queue é terminada e passa-se para a fase final do processo.

Na fase final, o browser é fechado e as informações necessárias são apresentadas ao utilizador através da consola e/ou para um ficheiro, mediante a opção indicada nas configurações.

5.5.Limitações

O QualState, encontra-se ainda numa fase inicial de desenvolvimento e, como tal, ainda existem algumas limitações conhecidas. Estas limitações podem ser divididas em 2 partes: limitações porque se optou por não desenvolver algumas funcionalidades, pois de momento não se enquadravam com o objetivo final do projeto; e limitações porque, devido à sua complexidade, não foi possível dedicar esforço suficiente para encontrar uma solução ideal no tempo disponível para o projeto.

Uma das limitações que não se enquadravam no imediato do projeto é realizar ações em que o evento não seja *click*. De momento o QualState filtra todos os eventos que não sejam *clicks*. Apesar de não existir nenhuma limitação para o tipo de evento que o utilizador possa inserir no input, qualquer evento que não seja um *click* não será realizado.

Uma limitação para a qual não se encontrou uma solução ideal, foi para eventos *hover* criados pelo CSS. Numa SPA é comum a utilização de animações CSS, entre as quais o *hover*. No entanto, não

foi possível nem encontrar uma maneira de descobrir quais os elementos que tem este tipo de evento, nem de conseguir identificar quando o estado da página é alterado ao ser ativado este tipo de evento.

Outra limitação combinando os dois fatores, ou seja, não faz parte do escopo original e necessita de mais investigação para encontrar uma abordagem ideal, é relacionada com a identificação de estados que necessitem de mais do que uma ação para serem atingidos.

5.6. Integração com QualWeb

O QualState pode ser utilizado de forma independente. No entanto, foi também integrado com o QualWeb de forma a conseguir uma avaliação de acessibilidade dos estados encontrados. Para tal foi necessário implementar algumas alterações no QualWeb.

As alterações foram mínimas, no entanto, foi necessário abranger outros módulos que o QualWeb utiliza. As alterações ocorreram nos módulos Core, Evaluation e Type. A Figura 5.21 apresenta uma imagem do processamento do QualWeb após as alterações realizadas.

Para utilizar o QualState através do QualWeb basta colocar as configurações do QualState na propriedade “qualstate” nas configurações do QualWeb. No presente momento, a opção de geração de relatórios no formato EARL não está disponível caso se queira utilizar o QualState.

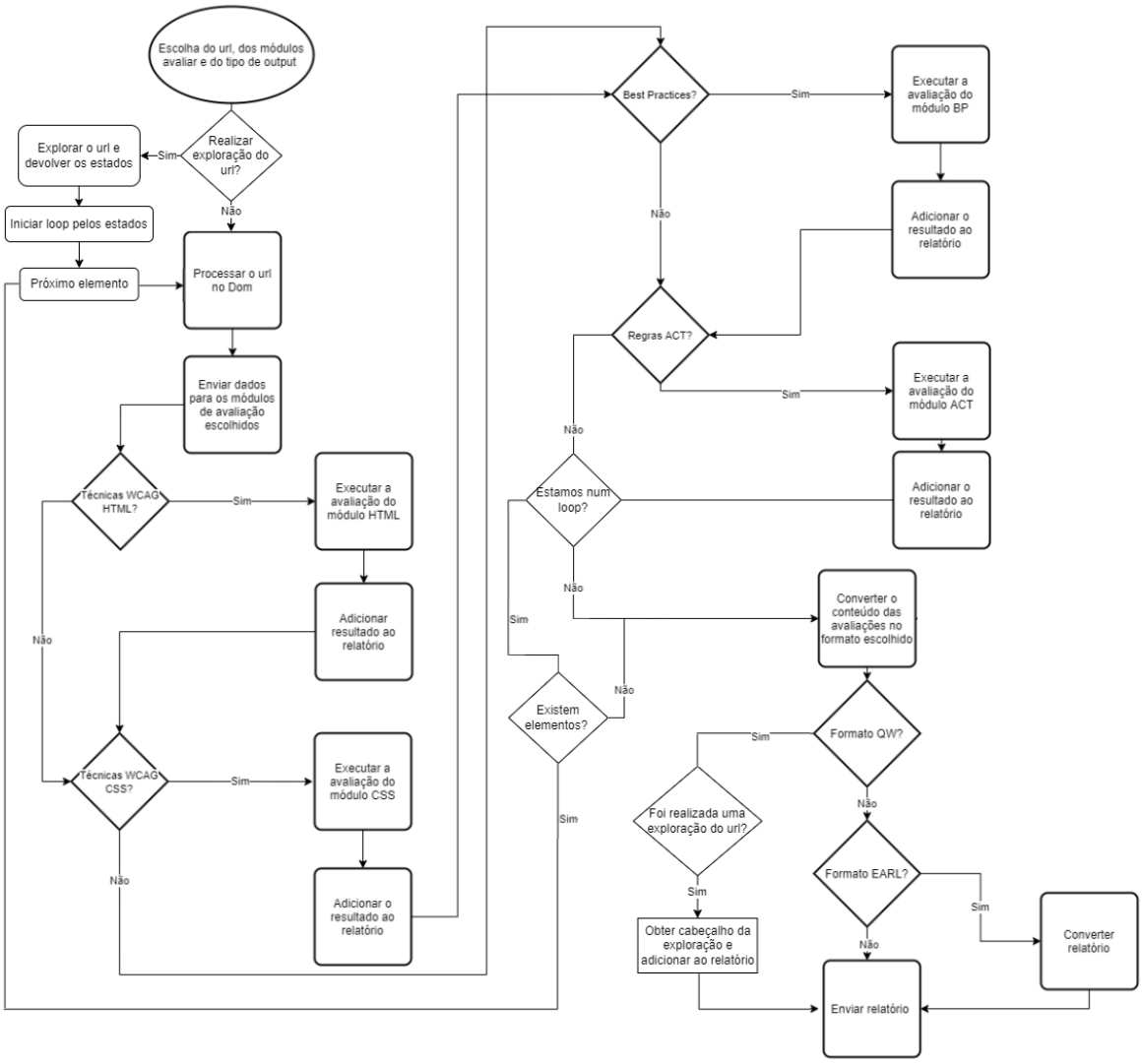

Figura 5.21 - Novo processamento do QualWeb.

Ainda no contexto das alterações realizadas, foi também alterada a estrutura do relatório final do QualWeb. Esta alteração surge apenas quando o QualWeb utiliza o QualState, e tem em consideração que podem existir uma ou mais avaliações de estados para o mesmo URL. Como tal, esta nova estrutura conta com um array, onde cada elemento contém a avaliação de acessibilidade de um estado identificado. A Figura 5.22 apresenta a estrutura do relatório final fornecido pelo QualWeb sem o QualState enquanto a Figura 5.23 apresenta a mesma estrutura, porém quando o QualState é utilizado.

Figura 5.22 - Estrutura de relatório final sem o QualState.

Figura 5.23- Nova estrutura de relatório final quando o QualState é utilizado.

6.Resultados

Com o objetivo de comparar o QualWeb **sem** o QualState com o QualWeb **com** o QualState, foram conduzidas duas sequências de testes comparativos em diferentes SPAs. Com estes testes foi possível observar os benefícios da utilização do QualState.

Todos os testes foram executados num computador com um processador Intel(R) Core(TM) i7-7700HQ a correr com 2.80GHz - 2.81 GHz, com 12 GB de RAM, no Windows 10.

Os testes foram realizados nas seguintes SPAs:

- Localhost (página local desenvolvida com auxílio das tecnologias node.js, JavaScript, HTML)

Figura 6.1- Página inicial do Localhost.

- Plant22[10]

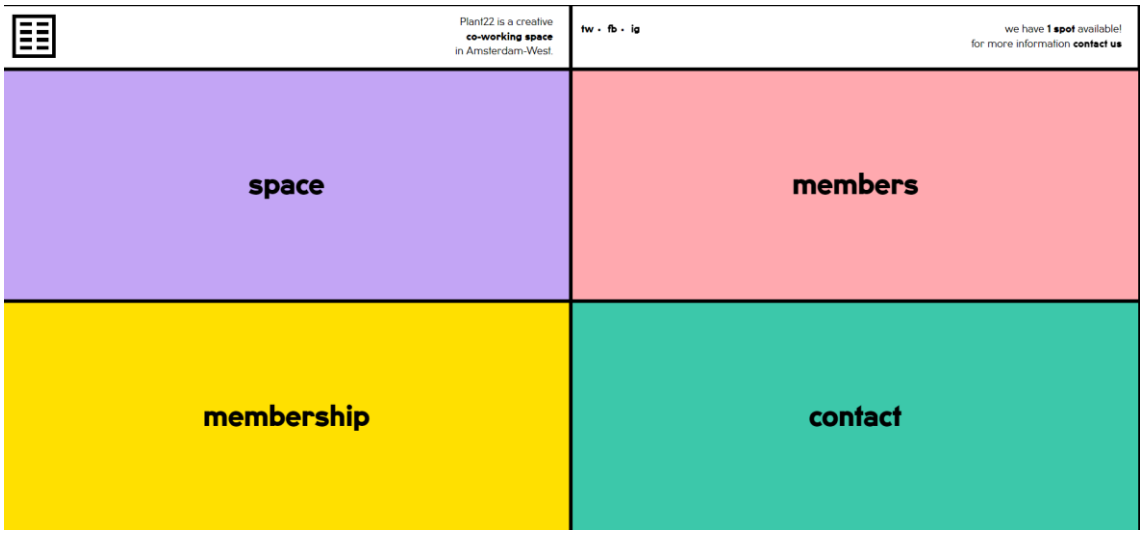

Figura 6.2 - Página inicial do Plant22.

[10] https://plant22.co/ (acedido em Setembro de 2023).

- PLACM(Prototype Large-Scale Accessibility Conformance Monitoring) [11]

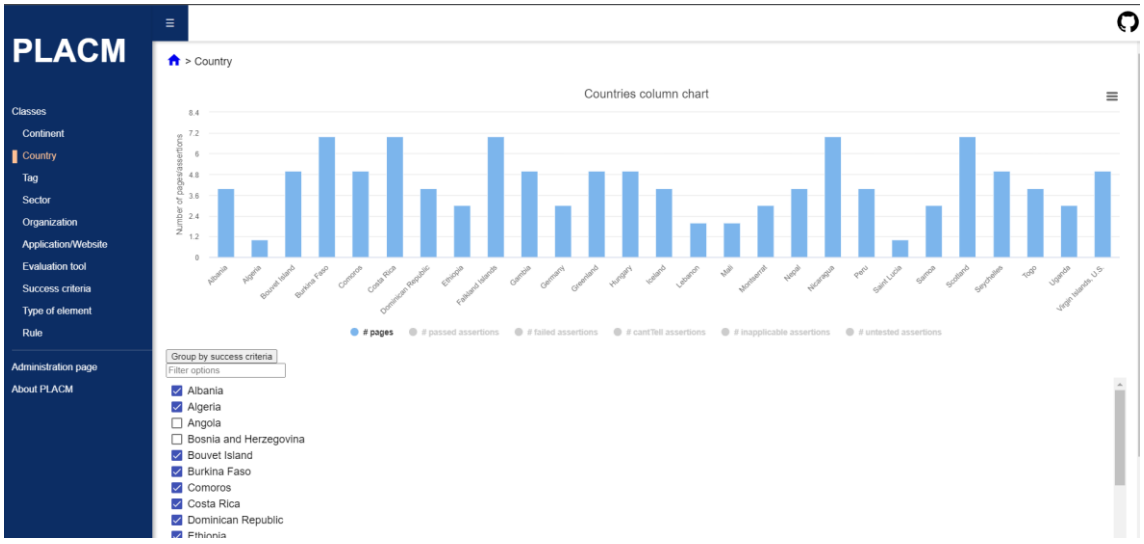

Figura 6.3 - Página inicial do PLACM.

- Observatório Português da Acessibilidade Web - Diretório: Área Governativa da Educação[12]

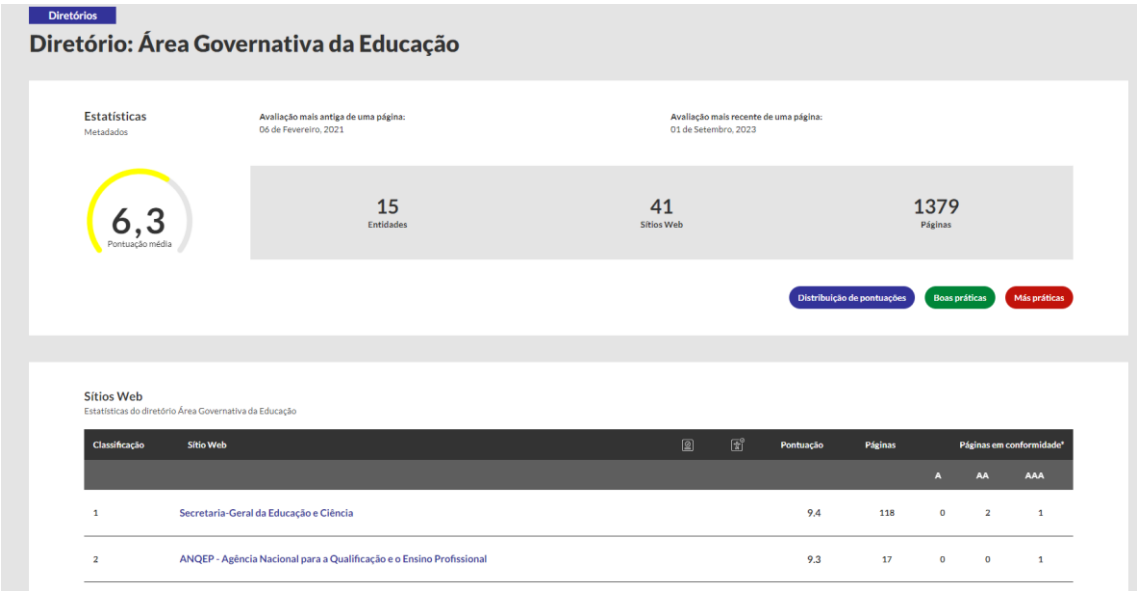

Figura 6.4 - Página inicial do Observatório Português da Acessibilidade Web - Diretório: Área Governativa da Educação.

O primeiro teste consistiu em explorar o quanto mais profunda é a avaliação de acessibilidade ao utilizar o QualWeb com QualState comparativamente ao QualWeb sem QualState. Para realizar este teste, a única configuração utilizada foi o limite de estados a serem encontrados pelo QualState. O teste consistiu em utilizar o QualWeb sem QualState contra o QualWeb com QualState com o número máximo de estados limitado a 2, 3, 5 e 10.

[11] http://qualweb.di.fc.ul.pt/placm/assertions/continent (acedido em Setembro 2023).
[12] https://observatorio.acessibilidade.gov.pt/directories/27 (acedido em Setembro 2023).

As seguintes tabelas apresentam os resultados, sob a forma dos resultados da avaliação de acessibilidade em função do número de estados avaliados. Os resultados da avaliação de acessibilidade dividem-se em 4 partes: **passed**, número de avaliações que passaram, **warning**, número de avisos encontrados, **failed**, número de falhas de avaliação de acessibilidade e, por fim, **inapplicable**, número de avaliações de acessibilidade que não foram possíveis testar.

Tabela 1 - Resultados da avaliação de acessibilidade do localhost.

	Estados				
	QualWeb sem QualState	QualWeb Com QualState			
	1	2	3	5	10
Passed	16	22	33	55	110
Warning	2	14	21	35	70
Failed	9	16	24	40	80
Inapplicable	55	112	168	280	560

Tabela 2 - Resultados da avaliação de acessibilidade do Plant22.

	Estados				
	QualWeb sem QualState	QualWeb Com QualState			
	1	2	3	5	10
Passed	11	27	38	66	126
Warning	7	16	25	43	88
Failed	3	6	10	16	35
Inapplicable	61	115	173	285	571

Tabela 3 - Resultados da avaliação de acessibilidade do PLACM.

	Estados				
	QualWeb sem QualState	QualWeb Com QualState			
	1	2	3	5	10
Passed	19	42	63	105	210
Warning	10	20	30	50	100
Failed	2	4	6	10	20
Inapplicable	51	98	147	245	490

Tabela 4- - Resultados da avaliação de acessibilidade do Observatório Português da Acessibilidade Web - Diretório: Área Governativa da Educação.

	Estados				
	QualWeb sem QualState	QualWeb Com QualState			
	1	2	3	5	10
Passed	24	51	77	131	268
Warning	9	21	33	57	120
Failed	1	7	13	25	60
Inapplicable	48	85	83	197	372

Ao analisar as tabelas com os resultados do primeiro teste, consegue-se verificar que em todos os casos o número de avaliações realizadas aumenta de acordo com o número de estados encontrados. Sendo que por cada novo estado encontrado é realizado novamente uma avaliação de acessibilidade realizada.

Com este teste, é possível verificar que a utilização do QualWeb com o QualState dá melhores resultados em todos os casos. O QualState consegue testar mais elementos e encontrar mais problemas de acessibilidade. Isto deve-se ao facto de haver agora um processo de exploração que encontra mais estados para serem testados. Deve-se, no entanto, relembrar que muitos dos erros podem ser repetidos, pois numa SPA existe a transformação do estado, sendo que erros que estavam no estado anterior podem ainda estar no estado atual. No entanto, também é importante salientar que, embora alguns dos problemas de acessibilidade sejam os mesmos, também foram encontrados novos problemas de acessibilidade em estados diferentes.

O segundo teste consistiu em tentar perceber como é que o número de processos em execução simultânea, propriedade **numberOfProcess** das configurações, influência o número de estados encontrados no determinado espaço de tempo. Como tal foram realizados um conjunto de testes nos mesmo sites em que objetivo passa por encontrar o número de estados para uma certa combinação entre o número de processos e tempo de execução.

Tabela 5 - Resultados do teste tempo/número de processos do localhost.

	Número de processos		
Tempo em minutos	1	5	10
1	16	40	59
3	30	80	116
5	41	107	152

Tabela 6 - Resultados do teste tempo/número de processos do Plant22.

Tempo em minutos	Número de processos		
	1	5	10
1	6	13	13
3	13	14	14
5	13	14	20

Tabela 7- Resultados do teste tempo/número de processos do PLACM.

Tempo em minutos	Número de processos		
	1	5	10
1	18	80	128
3	48	230	415
5	80	377	715

Tabela 8 - Resultados do teste tempo/número de processos do Observatório Português da Acessibilidade Web - Diretório: Área Governativa da Educação.

Tempo em minutos	Número de processos		
	1	5	10
1	6	21	31
3	15	68	100
5	25	120	150

Concluída a segunda fase dos testes, consegue-se verificar que um maior número de processos para o mesmo espaço de tempo resulta num número superior de estados encontrados.

No fim dos testes, conseguimos comprovar que o QualWeb com a utilização do QualState realiza uma quantidade superior de verificações de acessibilidade durante uma única avaliação. A nova funcionalidade de exploração resulta numa avaliação de acessibilidade mais completa, que não está limitada à página inicial da SPA.

7.Conclusão

Ao chegar ao fim deste projeto, consegue-se verificar que o principal objetivo, de desenvolver o QualState, uma ferramenta que faz exploração de websites de forma a encontrar estados foi cumprido.

O QualState apresenta uma proposta promissora, que pode e deve servir de ponto de partida para continuar a ser desenvolvida. Apesar dos resultados iniciais positivos, mais investigação em terminados aspetos desta ferramenta seria extremamente benéfico, incluindo a implementação de novas funcionalidades.

Este trabalho permitiu-me relembrar, aprofundar e conciliar conhecimento em tecnologias como JavaScript e node.js, aliado a ter agora conhecimentos da framewoork Puppeter, entre outras, que foram necessárias para o desenvolvimento do projeto. Consigo também agora dizer apesar de não ter lidado diretamente com os problemas de acessibilidade e avaliação, devido a todo o trabalho de investigação que foi necessário fazer, tenho agora algumas bases e conceitos sobre a avaliação de acessibilidade no contexto da web.

Em suma, o QualState serve de ponto de partida para uma ferramenta de exploração de SPAs, sendo que a mesma não é perfeita e existe ainda muito espaço para melhorar e adicionar novas funcionalidades.

7.1.Trabalho futuro

Ao concluir o trabalho, e cumprido os objetivos definidos no início da dissertação, existe ainda espaço para melhorar e adicionar novas funcionalidades, como por exemplo:

- **Desempenho:**
  - Melhorar a eficiência de todo o processo.

- **Testes:**
  - Testar o QualState com especialistas em avaliações de acessibilidade, de forma a receber feedback fidedigno que possa ajudar a compreender o que está mal e como pode ser melhorado, como também perceber o que pode ser acrescentado.

- **Configurações do utilizador:**
  - Explorar ainda mais o input fornecido pelo utilizador. Podem existir informações adicionais que o utilizador possa inserir que ajudem no processo de crawl e assim a encontrar mais estados.
  - Atualizar a nomenclatura das configurações dentro **interections**. Os nomes das propriedades podem ser mais claros, particularmente quando ser refere a input e inputs.

- **Expansão de eventos:**
  - Criar uma opção de “event” para as configurações, em que o utilizador pode definir eventos que devem ser sempre realizados em todas as páginas.

- Lidar com mais eventos, como foi referido, atualmente o QualState apenas responde a eventos do tipo “click”. No futuro, de forma a completar o processo de crawl e tentar garantir que se explorar todos os estados, seria útil considerar também outros tipos de eventos.

- **Comparação de estados:**
  - Melhorar a forma como é feita a comparação de estados. Existem estados em que a única coisa que se altera é uma imagem, pois no html o nome da imagem muda, consequentemente criando um estado novo. Existe a hipótese de o utilizador configurar a opção “ids_compare” para ignorar a imagem, acabando por não ser gerado um estado novo. No entanto, pode ser benéfico esta situação ser gerida pelo sistema sem que seja necessária interferência do utilizador.

Bibliografia

Liu, C.-H., & Woei-Kae and Sun, C.-C. 2020). GUIDE: an interactive and incremental approach for crawling Web applications. *76*. doi:10.1007/s11227-018-2335-4

Li, Y., Han, P., Liu, C., & Fang, B. (2018). Automatically Crawling Dynamic Web Applications via Proxy-Based JavaScript Injection and Runtime Analysis. *2018 IEEE Third International Conference on Data Science in Cyberspace*, (pp. 242-249).

Liu, C.-H., Chen, W.-K., & Sun, C.-C. (2020). GUIDE: an interactive and incremental approach for crawling Web applications. *The Journal of Supercomputing*.

Almendra, N., Moquillaza, A., & Paz, F. (2019). Web Accessibility Evaluation Methods: A Systematic Review. Em G. GoosJuris , & H. GoosJuris , *Lecture Notes in Computer Science* (pp. 226-237).

Mesbah, A., & van Deursen, A. (2007). Migrating Multi-page Web Applications to Single-page AJAX Interfaces. Em 1. E. Reengineering.

Fernandes, N., Batista, A., Costa, D., Duarte, C., & Carriço, L. (2013). Three Web Accessibility Evaluation Perspectives for RIA. *Proceedings of the 10th International Cross-Disciplinary Conference on Web Accessibility* (p. 9). Rio de Janeiro, Brazil: Association for Computing Machinery.

Lazar, J. (2022). Managing Digital Accessibility at Universities during the COVID-19 Pandemic. *Univers. Access Inf. Soc.*, 749–765.

Campoverde-Molina, M., Luján-Mora, S., & García, L. (2020). Empirical Studies on Web Accessibility of Educational Websites: A Systematic Literature Review. *IEEE Access*, 91676-91700.

Abascal, J., Arrue, M., & Valencia, X. (2019). Tools for Web Accessibility Evaluation. Em J. VanderdoncktQ, & Q. Liao, *Human–Computer Interaction Series* (pp. 479-503).

Nuñez, A., Moquillaza, A., & Paz, F. (2019). Web Accessibility Evaluation Methods: A Systematic Review. Em G. Goos, & J. Hartmanis, *Lecture Notes in Computer Science* (pp. 226-237).

Hostetler, T., Chen, S., & Blanco-Cuaresma, S. (2022). Web accessibility trends and implementation in dynamic web applications.

Hayfa.Y, A., Mohd Zalisham Jali, & Nurlida Basir. (2016). Web Accessibility Challenges. *International Journal of Advanced Computer Science and Applications*.

Bostic, T., Jeffrey, S., John, H., Daniel, C., Justin, B., & Brittany, T. (2021). Automated Evaluation of Web Site Accessibility Using A Dynamic Accessibility Measurement Crawler.

https://github.com/crawljax/crawljax. (s.d.). Obtido de git.

Sunman, N., Soydan, Y., & Sözer, H. (2022). Automated Web application testing driven by pre-recorded test cases. *Journal of Systems and Softwar*, 111441.

Mesbah, A., van Deursen, A., & Roest, D. (2012). Invariant-Based Automatic Testing of Modern Web Applications. *IEEE Transactions on Software Engineering*, 35-53.

Puppeteer. (s.d.). Obtido de https://pptr.dev/

F Laender, A., Altigran, S., Paulo, B., & C Reis, D. (2004). Automatic web news extraction using tree edit distance. *Proceedings of the 13th international conference on World Wide Web* (pp. 502–511). ACM.

About ACT Rules. (2022). Obtido de w3: https://www.w3.org/WAI/standards-guidelines/act/rules/about/

https://www.broadbandsearch.net/blog/internet-statistics#post-navigation-0. (2022). Obtido de broadbandsearch.

Understanding the Four Principles of Accessibility. (2022). Obtido de https://www.w3.org/WAI/WCAG21/Understanding/intro\#understanding-the-four-principles-of-accessibility

Lawton Henry, S. (2022). *What is Web Accessibility*. Obtido de w3: https://www.w3.org/WAI/fundamentals/accessibility-intro/\#what

Leithner, M., & Simos, D. (2020). XIEv: dynamic analysis for crawling and modeling of web applications.

MDN contributors. (2022). *Populating the page: how browsers work*. Obtido de MDN: https://developer.mozilla.org/en-US/docs/Web/Performance/How_browsers_work\#parsing

MDN contributors. (2022). *Window: DOMContentLoaded event*. Obtido de MDN: https://developer.mozilla.org/en-US/docs/Web/API/Window/DOMContentLoaded_event

Robie, J. (2022). *What is the Document Object Model*. Obtido de w3: https://www.w3.org/TR/WD-DOM/introduction.html

Estriga, A. (2020). QualWeb 3.0 – Avaliação automática de acessibilidade com qualidade, Faculdade de Ciências da Universidade de Lisboa.

MDN contributors. (2022). *Window: load event*. Obtido de MDN: https://developer.mozilla.org/en-US/docs/Web/API/Window/load_event

Web Content Accessibility Guidelines (WCAG) 2.0. (s.d.). Obtido de w3: https://www.w3.org/Translations/WCAG20-pt-PT/\#perceivable

Harper, S., & Yesilada, Y. (2008). Web Accessibility - A Foundation for Research. Em J. Vanderdonckt, & L. Q. Vera, *Human–Computer Interaction Series.* Springer.

Velasco, C., & Abou-Zahra, S. (2017). *Developer Guide for Evaluation and Report Language (EARL) 1.0*. Obtido de w3: https://www.w3.org/TR/EARL10-Guide/