Universidade Federal de Pernambuco

Centro de Informática - CIn UFPE

Curso de Engenharia da Computação

Extensão para auxiliar na acessibilidade para aplicativos feitos em Flutter utilizando VSCode

Trabalho de Conclusão de Curso de Graduação

por

João Rafael da Silva Faria

Orientador: Prof. Henrique Rebêlo

Recife, Setembro / 2023

João Rafael da Silva Faria

Extensão para auxiliar na acessibilidade para aplicativos feitos em Flutter utilizando VSCode

Monografia apresentada ao Curso de Engenharia da Computação, como requisito parcial para a obtenção do Título de Bacharel em Engenharia da Computação,
.

Orientador: Prof. Henrique Rebêlo

Recife, Pernambuco

2023

Ficha de identificação da obra elaborada pelo autor,
através do programa de geração automática do SIB/UFPE

Faria, João Rafael da Silva.
Extensão para auxiliar na acessibilidade para aplicativos feitos em Flutter utilizando VSCode / João Rafael da Silva Faria. - Recife, 2023.
58 p. : il., tab.

Orientador(a): Henrique Emanuel Mostaert Rebêlo
Trabalho de Conclusão de Curso (Graduação) - Universidade Federal de Pernambuco, Centro de Informática, Engenharia da Computação - Bacharelado, 2023.
Inclui referências, apêndices.

1. Flutter. 2. WCAG. 3. Acessibilidade. 4. VSCode. 5. Engenharia de Software. I. Rebêlo, Henrique Emanuel Mostaert. (Orientação). II. Título.

000 CDD (22.ed.)

Agradecimentos

Eu gostaria de agradeçer a todas as pessoas que me ajudaram durante esta jornada, especialmente para:

Meus pais, Ruy da Silva Faria e Genilsa Célia Augusta Nogueira da Silva, por todo o suporte e o conforto próprio que foi muitas vezes abdicado para que eu tivesse estudo de qualidade ao longo dos meus 24 anos.

A minha avó, Lisete Santos Faria, que não está mais presente entre nós para presenciar a minha conquista, mas que esteve por toda a caminhada atuando como uma segunda mãe, me dando muito amor e carinho.

Minha namorada, Marina Cabral do Nascimento, com quem eu divido a vida há quase uma década, e que me acalmou e me ajudou muito durante o processo de construção desse trabalho de graduação.

A todos os amigos que fiz durante a graduação, que sempre atuaram como boas influências, me puxando para cima e me motivando a aprender sempre mais.

Finalmente, eu gostaria de agradecer ao professor Henrique Rebêlo, com quem tive uma relação amistosa na construção desse trabalho, e que sempre se mostrou uma referência positiva dentro do ambiente da graduação.

Acessibilidade é essencial para construir
um mundo mais igualitário.

Tim Cook

RESUMO

Um mundo cada vez mais globalizado com o avanço da tecnologia exige que os produtos oferecidos na internet sejam utilizáveis e acessíveis para o maior número de pessoas sem perda de qualidade na experiência que eles se propõem a trazer. Na prática, uma porcentagem pequena dos sites brasileiros são considerados acessíveis para pessoas com deficiência. O trabalho proposto tem como finalidade a criação de uma extensão para a IDE do VSCode para ajudar a melhorar a acessibilidade de aplicativos construídos com o framework Flutter em tempo de desenvolvimento, através dos recursos fornecidos para construção de extensões do editor de código, trazendo informações e sugestões que adequem o código a algumas diretrizes WCAG, organizadas pela W3C, que fundamentam a construção de conteúdos digitais com qualidade e acessíveis a qualquer pessoa independente de sua deficiência e/ou habilidade.

Palavras-chave: Flutter, WCAG, Acessibilidade, VSCode, Engenharia de Software.

ABSTRACT

An increasingly globalized world with the advancement of technology demands that products offered on the internet be usable and accessible to the greatest number of people without compromising the quality of the experience they aim to provide. In practice, a small percentage of Brazilian websites are considered accessible for people with disabilities. The proposed work aims to create an extension for the VSCode IDE to help enhance the accessibility of applications built with the Flutter framework during development, through the features provided for code editor extensions. This extension will provide information and suggestions that align the code with certain WCAG guidelines, organized by the W3C, which underpin the creation of digitally high-quality and accessible content for anyone, regardless of their disability or ability.

Keywords: Flutter, WCAG, Acessibility, VSCode, Software Engineering.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE CÓDIGOS

LISTA DE SIGLAS

W3C	World Wide Web Consortium
WCAG	Web Content Accessibility Guidelines
IDE	Integrated development environment
MWPT	Movimento Web Para Todos
PWA	Progressive Web App
API	Application programming interface
UI	User interface
GUI	Graphical user interface
CLI	Command-line interface
NPM	Node package manager
LSP	Language server protocol
CPU	Central Processing Unit
SDK	Software development kit
HTML	HyperText Markup Language
JSON	JavaScript Object Notation
IPC	Inter-process communication
ADS	Análise e Desenvolvimento de Sistemas
EC	Engenharia da Computação
SI	Sistemas da Informação
CI	Continuous Integration
CD	Continuous Deployment

SUMÁRIO

1 INTRODUÇÃO

Com um mundo cada vez mais globalizado, é necessário que o ambiente digital seja acessível a todas as pessoas. Acessibilidade na internet significa garantir que pessoas com deficiência e mobilidade reduzida possam usufruir da mesma. Não somente pela interação com as informações, mas também pela possibilidade de contribuir para com as comunidades que existem.

No Brasil, pesquisas do IBGE indicaram que o número de pessoas com deficiência no Brasil é de 18,6 milhões, o que corresponde a cerca de 8,9% da população total [1]. Segundo também as pesquisas do Censo 2010, 24% da população declarou algum grau de dificuldade em pelo menos uma das habilidades investigadas: enxergar, ouvir, caminhar ou subir degraus, ou possuir deficiência mental/intelectual [2].

Essa problemática vai ainda além: no Brasil, temos a Lei Brasileira de Inclusão, que assegura que é obrigatória a acessibilidade nos sites de empresas com sede ou representação comercial no Brasil, além de órgãos públicos [3]. Na prática, menos de 1% dos sites brasileiros respeita os critérios de acessibilidade [4].

Para auxiliar na implementação de acessibilidade no desenvolvimento *web*, a W3C publicou, em 1999, o WCAG 1.0 [5]. Se tratam de diretrizes de acessibilidade para conteúdo na internet, que fundamentam a construção de conteúdos digitais com qualidade e acessíveis a qualquer pessoa independente de sua deficiência. Atualmente, essas diretrizes já estão em sua versão 2, com uma versão 3 em fase exploratória.

Considerando as problemáticas abordadas acima, esse trabalho tem o objetivo de, através das ferramentas que serão descritas mais a frente, construir uma ferramenta que tente implementar as diretrizes da WCAG no contexto de desenvolvimento *mobile*. Para criação dessa plataforma, fizemos uma análise de ferramentas que cumprem funções semelhantes para desenvolvimento *web* para definir o escopo que gostaríamos e poderíamos trazer para outro ambiente semelhante.

1.1 Motivação

A presença da evolução tecnológica na sociedade cresce exponencialmente ano após ano. Hoje, é muito difícil encontrar adultos que não possuam um *smartphone*, seja para se comunicar, trabalhar, como ferramenta de lazer ou quaisquer outras razões. E isso não

se limita apenas à navegação diretamente na internet, como era muito mais comum há anos atrás. Hoje, existem aplicativos com diversos propósitos que alugam até mais tempo da nossa atenção.

Considerando isso, é muito importante que essa preocupação com a acessibilidade na internet se extenda à acessibilidade nos *apps*. Vale Querini, em *post* num blog na internet, abordou a importância do design inclusivo. Em suas palavras, "Se não incluirmos intencionalmente, o risco é excluir involuntariamente" [6].

Além disso, produzir artefatos acessíveis transmite uma imagem melhor da marca, que auxilia na fidelização dos seus usuários. Também é fácil imaginar que aumentar o alcance do público-alvo certamente é uma chave para o crescimento e sucesso do aplicativo. É com base nessas evidências que o estudo se baseia.

1.2 **Objetivos**

1.2.1 Objetivo Geral

O objetivo geral deste trabalho é a criação de uma extensão de desenvolvimento para a IDE VSCode que auxilie o desenvolvedor a construir aplicativos mais acessíveis utilizando o *framework* de desenvolvimento móvel Flutter em tempo de desenvolvimento e de forma educativa.

1.2.2 Objetivos Específicos

Para atingir o objetivo principal, foram definidos os seguintes objetivos específicos:

- Realizar um estudo sobre a divisão e funcionamento das diretrizes de acessibilidade da WCAG e de ferramentas que auxiliem de alguma forma na acessibilidade de artefatos digitais.
- Desenvolver extensão para o VSCode que adapte em certo nível essas diretrizes para o desenvolvimento *mobile* utilizando Flutter.
- Garantir que a extensão sempre recomende melhores práticas acessíveis ao desenvolvedor em tempo de desenvolvimento.
- Garantir que a extensão tenha um teor educativo, sempre justificando o motivo por trás das recomendações.

- Criar um experimento de teste da extensão com pessoas que desenvolvam utilizando o framework e IDE propostos para avaliar usabilidade, experiência de uso e o suporte à acessibilidade da ferramenta.

1.3 Organização de texto

Quanto à estrutura e organização dos capítulos desse trabalho:

- **No capítulo 2** abordamos a análise sobre a divisão e funcionamento das diretrizes, bem como explicamos as ferramentas, conceitos teóricos e metodologias relevantes no desenvolvimento do trabalho.
- **No capítulo 3** especificamos o desenvolvimento da extensão, suas *features* e casos de uso.
- **No capítulo 4** compartilhamos os resultados do experimento de teste da ferramenta e levantamos pontos de discussão sobre os dados e *feedbacks*.
- **No capítulo 5** Trazemos uma visualização, de um ponto de vista geral, de trabalhos relacionados ao objetivo desse estudo que estão disponibilizados na rede.
- **No capítulo 6** concluímos o trabalho, pontuando dificuldades, aprendizados e pontos de melhoria para o futuro da ferramenta.

2 FERRAMENTAS, CONCEITOS TEÓRICOS E METODOLOGIAS

Neste capítulo, iremos entender melhor sobre o funcionamento das diretrizes da WCAG, bem como entender melhor sobre quais serão as ferramentas, linguagens de programação e tecnologias escolhidas como alvo e também para a criação dessa extensão, sempre justificando eventuais decisões.

2.1 Funcionamento das diretrizes da WCAG

As primeiras diretrizes de acessibilidade na *web* foi compilada por Gregg Vanderheiden e lançada em 1995 [7]. A partir daí, mais de 30 diretrizes de acessibilidade foram lançadas por diversos autores e organizações, e foram compiladas e trazidas juntas pela Universidade de Wisconsin-Maddison na 8ª versão do *Unified Web Site Accessibility Guidelines* [8], em 1998, servindo como ponto de entrada para o WCAG. Em 1999, a WCAG 1.0 foi lançada e tornou-se uma recomendação da W3C.

Do ano do lançamento até hoje, várias versões da WCAG foram publicadas, sendo a versão 2.2 agendada para ser finalizada no terceiro trimestre de 2023. Atualmente, a versão mais atualizada e completa é a versão 2.1. Na versão 2, as diretrizes são divididas dentro de quatro princípios: perceptível, operável, entendível e robusto. Os quatro princípios são divididos em 12 diretrizes, que por sua vez são divididas em inúmeros critérios de sucesso para cada um [9].

A cada critério de sucesso, é associado um nível de conformidade - que pode ser A, AA (ou duplo-A) e AAA (ou triplo-A). Esses três níveis de conformidade estão diretamente associados ao "nível de acessibilidade"de um *website*:

- *Sites* **devem** estar de acordo com os critérios de nível de conformidade A, caso contrário um ou mais grupos de pessoas não conseguirão acessar o conteúdo da página.
- De maneira semelhante *sites* **devem** estar de acordo com os critérios de nível de conformidade AA, caso contrário um ou mais grupos de pessoas encontrarão dificuldade em acessar o conteúdo da página.
- *Sites* **devem** estar de acordo com os critérios de nível de conformidade AAA para que alguns grupos de pessoas encontrem facilidade em acessar o conteúdo da página.

Vale ressaltar que nem todos os critérios de sucesso de um nível de conformidade são atingíveis apenas através do desenvolvimento do sistema; O design inclusivo é parte chave para atingir a completude. Além disso, estar de acordo com o nível de conformidade duplo-A, por exemplo, significa cumprir todos os critérios de sucesso do nível A e do nível duplo-A. O mesmo vale para o nível de conformidade triplo-A.

2.2 **Tecnologias alvo da ferramenta**

Nesta seção, falaremos um pouco sobre as tecnologias que são alvo da ferramenta que vamos construir, isto é, sobre que tecnologias a nossa ferramenta atuará.

2.2.1 VSCode

O Visual Studio Code, comumente chamado de VSCode é um ambiente de desenvolvimento integrado (IDE) de código aberto, desenvolvido pela Microsoft e lançado em 2015 [10]. Construído com o Electron Framework, tecnologia para desenvolvimento de aplicações *desktop* utilizando tecnologias web, foi projetado para ser extremamente adaptável e extensível, de forma que a tecnologia oferece suporte a muitas linguagens de programação e *frameworks*, o que a torna uma escolha vérsatil para desenvolvedores de software.

A sua interface de usuário é simples, concisa e intuitiva, e é complementada por recursos como *syntax highlighting*, *autocomplete* de código, depuração integrada e controle de versionamento. Há uma integração direta com sistemas de controle de versão, como o Git. Além de todas essas funcionalidades, possui uma enorme biblioteca de extensões desenvolvidas pela comunidade usuária da IDE, que extende mais ainda as suas funcionalidades, de forma que o usuário consegue personalizer o seu ambiente de desenvolvimento de acordo com o que precisa.

Para a construção dessas extensões, o VSCode disponibiliza uma API, que fornece uma série de recursos para facilitar o desenvolvimento das mesmas, poupando tempo e intensificando as capacidades do desenvolvedor. Além disso, vale deixar clara a escolha do desenvolvimento da extensão para a IDE do VSCode por dois motivos:

- O VSCode foi construído sobre o princípio da extensibilidade. Tudo foi pensado para ser construído como extensão. Muitos recursos que já estão carregados ini-

cialmente no uso da IDE foram construídos como extensão, e portanto, é natural imaginar que o suporte para construir extensões para o VSCode é muito grande, com documentação refinada dos recursos e muitos exemplos disponíveis na internet para acelerar o aprendizado.

- Outro recurso importante que tornou-se fator motivador e que falaremos mais para frente, proposto pela criadora do VSCode, a Microsoft, é o Language Server Protocol (LSP): trata-se de um protocolo de comunicação que padroniza a relação entre servidores que provêm recursos para linguagens de programação e editores de código. Esse protocolo permite que o esforço necessário para trazer as mesmas funcionalidades para outras IDEs que aceitem a comunicação através do protocolo seja consideravelmente reduzido.

2.2.2 Flutter

Flutter é um *framework* de desenvolvimento *mobile* de código aberto que foi lançado pelo Google em 2017 [11]. A tecnologia foi projetada para desenvolvimento de aplicativos nativos de alta qualidade e desempenho para as diversas plataformas a partir de um única fonte de código base. O *framework* permite que os desenvolvedores criem UIs atraentes e responsivas, compartilhando a maior parte do código base entre os difentes sistemas operacionais, com iOS, Android e até mesmo *web*. O Flutter baseia-se na linguagem de programação Dart, que também foi criada pela Google, e usa renderização direta para criar as suas interfaces.

Seu grande diferencial competitivo é a sua abordagem de desenvolvimento rápido, consistente e multi-plataforma. Ao utilizá-lo, os desenvolvedores criam suas UIs através de widgets personalizáveis e de alto desempenho. A renderização direta que oferece tira a dependência de componentes nativos dos sistemas operacionais, resultando num produto consistente nos mais diferentes dispositivos móveis. Além disso, a funcionalidade de *hot reload* ajuda o fluxo de criação, permitindo que os programadores vejam instantaneamente as mudanças refletidas nos seus aplicativos enquanto trabalham, acelerando assim esse processo.

O Flutter tem ganhado popularidade entre os desenvolvedores em busca de uma solução eficaz para o desenvolvimento de aplicativos multiplataforma. O Brasil tem uma das maiores comunidades do *framework* [12], tornando-se então uma das opções mais

relevantes para pessoas desenvolvedoras que se aventuram no mundo *mobile*.

2.3 **Tecnologias e Conceitos Teóricos utilizados na Construção**

Nesta seção, falaremos um pouco sobre as tecnologias e conceitos teóricos que serão diretamente utilizados na construção da ferramenta objetivo desse estudo.

2.3.1 Github

O Github é uma plataforma utilizada em todo o mundo para hospedar, gerenciar e colaborar em projetos de desenvolvimento de software [13]. Dispondo de ferramentas como o controle de versão, crucial para gerenciar os lançamentos de uma ferramenta. Utilizando o Git, que é um sistema de controle de versão distribuído, o Github permite que os programadores possuam um histórico detalhado de todas as alterações e os autores delas sobre o código base do projeto ao longo do andamento do projeto.

Ultrapassando 100 milhões de repositórios há cerca de cinco anos atrás [14], o Github está entre os clientes de Git GUI mais utilizados do mundo, por diversas milhões de empresas e colaboradores.

No que tange o gerenciamento de código versionado, o Github oferece recursos como *branches* e *pull requests*. As *branches* permitem que os desenvolvedores consigam alocar sua atenção em diferentes recursos e correções de maneira independente, mantendo essas alterações separadas até estarem aprovadas ou prontas para serem acopladas ao código base principal. Já as *pull requests* permitem que mudanças feitas numa ramificação sejam revisadas e discutidas por outros colaboradores da equipe antes de serem unidas ao código principal.

Ademais, também dispõe de *features* que rastreiam problemas, alocam tarefas e propõe discussões sobre temas específicos que sejam relevante ao projeto, que é muito relevante quando se trabalhando com lançamentos de software organizados e transparentes. A usabilidade oferecida pelo produto tornam-no uma escolha muito popular para times de desenvolvimento.

Utilizaremos o Github para gerenciar o código fonte da extensão que desenvolveremos, bem como para gerenciar as versões que iremos liberar para o *marketplace* de extensões do VSCode.

2.3.2 Node.js

O Node.js é um ambiente de tempo de execução de código aberto construído sobre o motor de JavaScript V8 do Google Chrome [15]. Ele permite que os programadores consigam executar código na linguagem de programação JavaScript do lado do servidor, ao invés de apenas nos *browsers*. Seu modelo de programação é assíncrono e orientado a eventos, o que o torna muito eficiente para lidar com operações de entrada e saída, solicitações de rede e consulta a bancos de dados. Dessa forma, rapidamente tornou-se uma escolha popular para desenvolvimento de aplicações web escaláveis e em tempo real.

Entre suas aplicações mais comuns, destacam-se aplicações web em tempo real, APIs e Serviços *web*, microsserviços, aplicações de *streaming* de mídia e ferramentas de linha de comando (CLI), como o node package manager (NPM).

Utilizaremos o Node.js não diretamente (trabalharemos numa camada acima de abstração) no cliente da nossa extensão VSCode, cujo contexto do *host* roda em Node.js, e também na implementação do *language server*, que é responsável por executar nosso código que aplica regras de validação e falaremos melhor na seção sobre a *Extension API* do VSCode.

2.3.3 TypeScript

O TypeScript é conhecido como um *superset* do JavaScript, isto é, ele provê um conjunto de ferramentas a mais que não estão originalmente presentes no JavaScript [16]. Desenvolvida em código aberto pela Microsoft, ela fornece aos desenvolvedores uma alternativa para criar códdigo de forma mais segura e escalável, minimizando erros comuns durante o desenvolvimento de software e melhorando a manuntenção de grandes projetos.

Uma das principais características é a tipagem estática. Variáveis, parâmetros de função, estrutura de objetos precisam de indicações dos tipos de dados esperados para manipulação no código. Isso ajuda a detectar erros de tipagem em tempo de compilação. Ele também provê a *feature* de autocompletar melhorada, de forma que o programador consegue visualizar com mais facilidade todos os métodos disponíveis em objetos e estruturas de dados.

Por ser transpilado para JavaScript, *browsers* e ambientes que não suportem diretamente o TypeScript ainda assim conseguem executar o código resultante da transpilação.

Utilizaremos o TypeScript como linguagem de programação para construir o código-fonte da nossa extensão.

2.3.4 Expressões regulares

Expressões regulares (ou regexes, como comumente são abreviadas) são formas concisas e flexíveis de identificar um conjunto de cadeias de caracteres de interesse [17]. Um regex age como um conjunto de passos para que os programadores procurem padrões específicos dentro de um texto. Elas são utilizadas em linguagens de programação e aplicativos para validar formatos de entradas, mascarar conteúdos e extrair informações de padrões de texto complexos.

É possível interpretá-las como um detector de padrões textuais, que permitem que se especifique não apenas palavras propriamente ditas, mas também padrões mais diversos e extensos. Por exemplo, todas as sequências de palavras que comecem com a sílaba "cha"e terminem com a sílaba "go", ou então todas as sequências de números no formato de CPF. Se trata de uma ferramenta poderosa para processar texto de maneira eficiente, poupando esforço do desenvolvedor em construir métodos mais trabalhosos para cumprir o mesmo propósito.

Utilizaremos as expressões regulares para mapear os trechos de código importantes para aplicarmos as regras de validação da extensão.

2.3.5 VSCode Extension API

VSCode foi construído com a ideia de extensibilidade em mente, e é exatamente aí que a VSCode Extension API surge. Trata-se da API fornecida pelo próprio VSCode para auxiliar desenvolvedores na criação de novas extensões [18]. Muitas das próprias funcionalidades do VSCode foram construídas propriamente como extensões, o que corrobora com os ideais da IDE. As extensões podem ser distribuídas por meio do Visual Studio Code Marketplace, onde os usuários podem instalá-las facilmente

A capacidade de interação que uma extensão pode utilizar é muito grande. Entre as principais coisas possíveis de realizar com uma extensão, destacam-se:

- Alterar o visual do VSCode através de temas de cores, e de ícones para visualização de determinados tipos de arquivo no projeto, como mostra a Figura 1

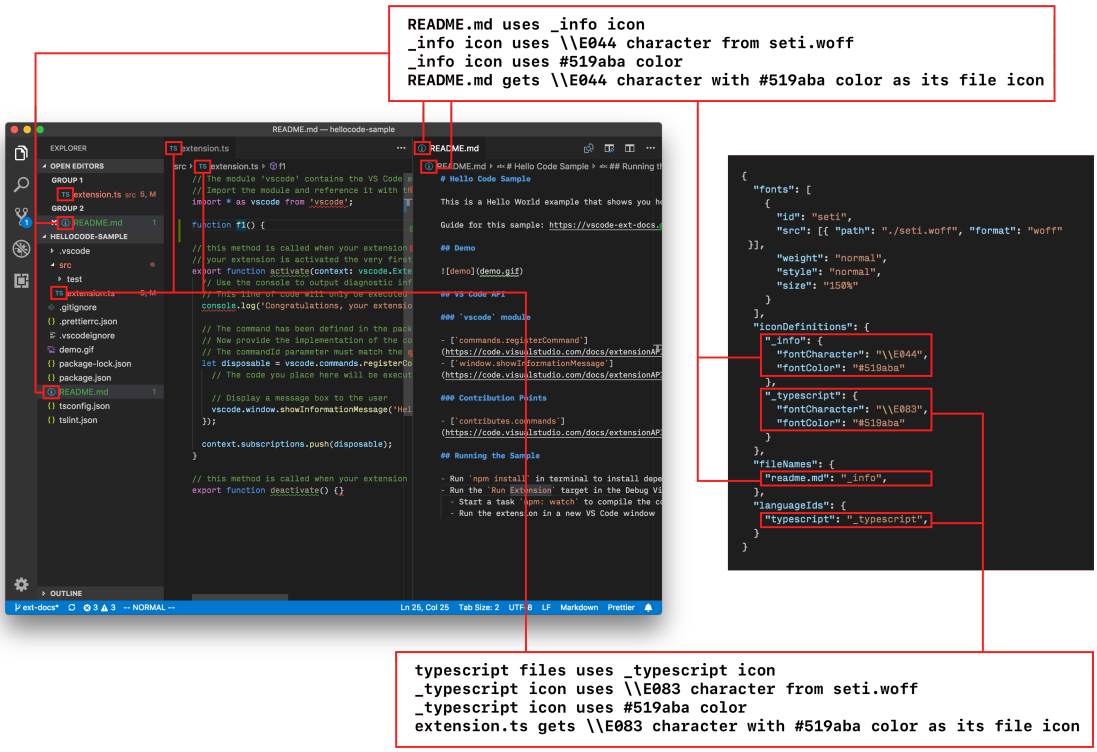

Figura 1: Exemplo de customização de ícones através de extensão no VSCode. Fonte: https://code.visualstudio.com/api/extension-capabilities/theming.

- Adicionar componentes e *views* customizáveis na interface do usuário no VSCode, como mostra a Figura 2
- Possibilita executar depurações customizáveis de código em tempo de execução.
- Dar suportes específicos a novas e também já existentes linguagens de programação. É essencialmente sobre essa funcionalidade da API que criaremos a extensão.

2.3.5.1 VSCode API Language Extensions

Extensões de linguagem são uma categoria de extensões que podem ser construídas com a API de extensões do VSCode. Com elas, é possível prover funcionalidades de edição inteligentes para as mais diferentes linguagens de programação [19]. As funcionalidades de linguagem podem aproximadamente ser definidas em dois tipos:

- **Funcionalidades declarativas de linguagem**: estas são definidas em arquivos de configuração, e agrupam, por exemplo, *syntax highlighting*, *snippets* de com-

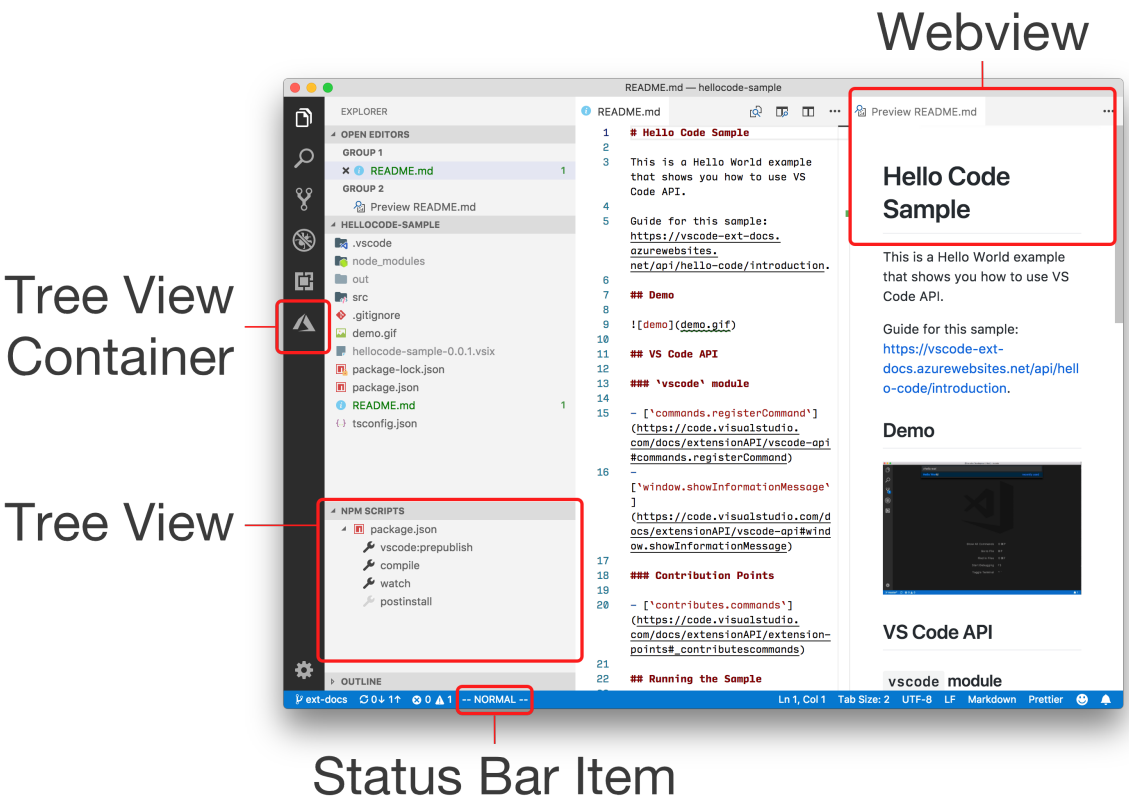

Figura 2: Exemplo de componentes customizáveis na UI do VSCode.
Fonte: https://code.visualstudio.com/api/extension-capabilities/extending-workbench.

plementação, correspondência de colchetes, auto-fechamento de colchetes, autoidentação de código, etc.

- **Funcionalidades programáticas de linguagem**: incluem funcionalidades como informação de *hover*, *autocompleting*, saltos para definições, checagem de erros, formatação, refatoração, etc. Essas são disponibilizadas por um *language server*, que nada mais é que um programa que analisa o código do projeto para prover funcionalidades.

2.3.5.2 Language Server e Language Server Protocol

Language Server é um tipo especial de extensão que auxilia a experiência de edição de linguagens de programação em geral. Com ele, você pode implementar as funcionalidades programáticas previamente descritas de acordo com as necessidades da sua extensão na **linguagem de programação que você desejar**.

Isso gera alguns problemas. Normalmente, *language servers* são implementados nas suas linguagens de programação nativas, e é um desafio integrá-los com o VSCode, que é executado sobre Node.js. Além disso, funcionalidades de linguagem podem precisar de recursos de execução intensos. Por exemplo, para validar corretamente um arquivo, o *language server* precisa analisar uma grande quantidade de arquivos, construir árvores sintáticas abstratas para elas e realizar análise estática do programa. Essas operações necessitam de demanda de CPU e memória, e é importante garantir que a performance do VSCode não seja afetada.

Ainda há outra problemática a ser resolvida que vai além da IDE do VSCode: para integrar ferramentas de múltiplas linguagens com múltiplos editores de código (além do VSCode) envolve esforço significante. Da perspectiva das ferramentas da linguagem, elas precisam se adaptar aos editores de código com variadas APIs para extensões. Isso faz com que implementar funcionalidades para **M** linguagens de programação em **N** editores de código gere um esforço de **M * N**.

Para resolver isso, a Microsoft especificou o *Language Server Protocol (LSP)*, que padroniza a comunicação entre ferramentas de linguagens e editores de código [20]. Dessa maneira, *language servers* podem ser implementados em qualquer linguagem e executar seu próprio processo para diminuir os custos de performance, e posteriormente se comunicarem com os editores de código, como o VSCode, através do LSP. Dessa maneira, um

Figura 3: Diferença do funcionamento da implementação de *Language Servers* sem LSP e com LSP.
Fonte: https://code.visualstudio.com/api/language-extensions/language-server-extension-guide.

mesmo *language server* consegue prover funcionalidades para uma linguagem para todos os editores de código que aceitarem comunicação através do LSP, como mostra a Figura 3.

2.3.5.3 Language Server no VSCode

No VSCode, um *language server* possui duas partes:

- *Language Client*: Uma extensão para VSCode normal, escrita em JavaScript ou TypeScript, que tem acesso a todas as funcionalidades da API para extensões do VSCode.
- *Language Server*: uma ferramenta de análise de linguagem rodando em um processo separado.

Como mencionado anteriormente, por rodar em um processo separado, um *language server* pode ser implementado em qualquer linguagem que possua alguma implementação do LSP para se comunicar. Além disso, rodar em um processo separado auxilia a evitar custos de performance.

Por exemplo, podemos ter uma extensão que provê funcionalidades de edição para a linguagem HTML no VSCode com um *language client* rodando em TypeScript, e um

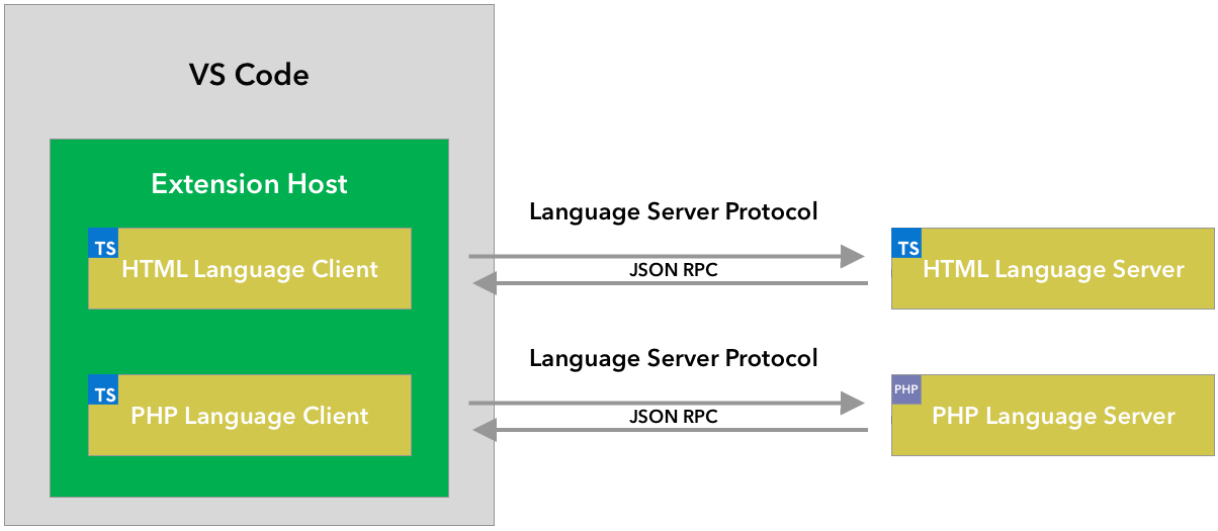

Figura 4: Exemplos de funcionamento de extensões de linguagens no VSCode. Fonte: https://code.visualstudio.com/api/language-extensions/language-server-extension-guide.

language server rodando também em TypeScript. Da mesma forma, podemos ter uma extensão provendo funcionalidades para a linguagem PHP com o *language client* rodando com TypeScript, enquanto o *language server* roda em PHP (se comunicando através do LSP), como mostra a Figura 4.

Para facilitar o desenvolvimento de extensões pelos desenvolvedores, o time do VSCode disponibiliza um SDK em Node com diversas implementações relevantes que agilizam o processo de construção:

- Um módulo pra fazer comunicações do *language client* com o *language server* numa extensão pro VSCode. O módulo auxilia a implementação sobre Node.js;
- Um módulo que implementa documentos de texto usáveis num *language server* executando sobre Node.js;
- Uma implementação do LSP em TypeScript, de forma que facilite construção de *language servers* nessa linguagem;
- Um protocolo de messageria (JSON-RPC) para comunicar o cliente e o servidor.

Ao aderir à utilização desse SDK, conseguimos aproveitar muita coisa previamente implementada pelo time do VSCode, agilizando o processo de construção da extensão que é o objetivo desse estudo.

2.4 **Metodologia de Avaliação da Ferramenta**

A experiência do usuário e utilidade de extensão é uma parte importantíssima desse estudo. Nesta seção, falaremos um pouco sobre a metodologia que utilizaremos para validar a experiência do desenvolvedor com a ferramenta que desenvolveremos nesse estudo.

2.4.1 System Usability Scale (SUS)

O SUS é o questionário padronizado mais utilizado para avaliação da usabilidade de sistemas e interfaces de usuário [21]. Desenvolvido por John Brooke em 1986, ele é composto de dez perguntas que buscam medir a percepção do usuário no que se refere à facilidade de uso de um sistema específico. Suas perguntas abrangem aspectos como aprendizado do sistema, eficiência, satisfação geral do usuário e capacidade de memorização.

Cada uma de suas perguntas é respondida numa escala de um a 5, variando de "discordo totalmente"a "concordo totalmente". Após coletar quantidade razoável de respostas, os escores são normalizados e somados para gerar uma pontuação geral de usabilidade, que pode variar de 0 a 100. Um escore acima de 68 é considerado acima da média, e pontuações mais altas indicam que houve uma percepção positiva da usabilidade do sistema.

O cálculo desse escore é feito da seguinte forma: as perguntas são divididas em dois grupos. As de numeração ímpar são realizadas de maneira direta, com o "concordo totalmente"representando a resposta mais positiva para o sistema, e o "discordo totalmente"sendo a resposta mais negativa para o sistema. Já as de numeração par, são realizadas de maneira "inversa", isto é, o "discordo totalmente", na verdade, é a resposta mais positiva para o sistema, enquando o "concordo totalmente", na verdade, significa a resposta mais negativa para o sistema.

De posse dessa informação, o cálculo é feito da seguinte forma:

$$EscoreSUS = ((S_1 - 5) + (25 - S_2)) * 2,5$$

Onde S_1 é a soma das pontuações nas perguntas de numeração ímpar, e S_2 é a soma

das pontuações nas perguntas de numeração par, e o 2,5 representa o fator de correção do questionário.

O questionário é muito valorizado por sua simplicidade e eficácia para conseguir feedback dos seus usuários no que diz respeito a um produto ou sistema, permitindo que os desenvolvedores identifiquem com facilidade pontos de melhores e tomem decisões informadas baseadas em dados para aprimorar a usabilidade.

3 DESENVOLVIMENTO DA EXTENSÃO

Diante de todo o ferramental apresentado anteriormente, neste capítulo abordaremos com mais precisão o desenvolvimento propriamente dito da extensão *Accessible Flutter*, desde a definição de *features* até a implementação em código.

3.1 Definição das Features

As seguintes *features* foram definidas com base nos critérios de sucesso atingíveis através de código e compatíveis com o fluxo de desenvolvimento com o *framework* Flutter, que constrói os componentes da UI através de *widget*, que nada mais são do que espécies de componentes.

3.1.1 Snippets

Utilizando a API de extensões do VSCode, os *snippets* são definidos através de um arquivo de configuração no formato JSON. A lista de snippets fornecidos na extensão será descrita a seguir, bem como os trechos de código das suas implementações.

- *Snipppet*: Botão semântico
  - **Palavra chave**: *SemanticsButton*
  - **Descrição**: Encapsula um botão no widget Semantics
  - **Linguagem escopo**: dart
  - **Palavra de acesso ao *snippet***: ”button”
  - **Trecho de código da implementação**:

```
{
    "SemanticsButton": {
        "scope": "dart",
        "prefix": "button",
        "body": [
            "Semantics(",
            "  label: '${1:Proposito do botao}',",
            "  child: ${2}Button(${3}),",
```

```
            ")"
        ],
        "description": "Encapsula o botao no widget
            Semantics"
    },
}
```

Trecho de Código 1: Implementação do *snippet* de botão semântico

- *Snipppet*: *Input* de texto acessível
  - **Palavra chave**: *AccessibleTextInput*
  - **Descrição**: Cria *input* de texto com label e dica de preenchimento
  - **Linguagem escopo**: dart
  - **Palavra de acesso ao *snippet***: "textfield"
  - **Trecho de código da implementação**:

```
{
    "AccessibleTextInput": {
        "scope": "dart",
        "prefix": "textfield",
        "body": [
            "Text${1}Field(",
            "  decoration: InputDecoration(",
            "    hintText: '${2}',",
            "    labelText: '${3}',",
            "  ),",
            ")"
        ],
        "description": "Cria input de texto com label
            e dica de preenchimento"
    }
}
```

Trecho de Código 2: Implementação do *snippet* de *input* acessível

3.1.2 Regras de validação

A seguir, listaremos todas as regras de validação, definidas através de princípio(s) e critério(s) de sucesso mapeados, o(s) *widget(s)* alvo, algoritmo utilizado para validar, texto de feedback que é mostrado para o usuário (se houver um), *snippet* sugerido (caso haja) e o porquê de sua importância.

- **Regra de validação**: *Label* semântica de imagens
  - **Princípio**: perceptível
  - **Critério de sucesso**: 1.1.1 (A) - Qualquer conteúdo "não textual"e relevante para compreensão da informação, deve trazer uma descrição alternativa em texto (visível ou não) para identificar o conteúdo (inclusive captcha, por exemplo)
  - ***Widgets* alvo**: Image, Image.network, Image.asset, Image.file, Image.memory.
  - **Algoritmo**: Mapeie todos os widgets alvo no documento, um por um, verifique se possui a propriedade "semanticLabel"definida, caso não, execute o diagnóstico.
  - **Feedback**: "O widget Image deveria ter a propriedade semanticLabel"ou "O construtor * do widget Image deveria ter a propriedade semanticLabel"
  - **Por que importa?** Os leitores de tela não têm como traduzir uma imagem em palavras que serão lidas para o usuário, mesmo que a imagem consista apenas de texto. Como resultado, é necessário que as imagens tenham texto alternativo curto e descritivo para que os usuários de leitores de tela entendam claramente o conteúdo e a finalidade da imagem. Se você não consegue ver, todos os tipos de informação visual são completamente inúteis, a menos que seja fornecida uma alternativa de texto digital para que os leitores de tela possam converter esse texto em som ou braille. O mesmo se aplica em vários graus para pessoas com baixa visão ou daltonismo.

- **Regra de validação**: *Label* semântica de ícones
  - **Princípio**: perceptível
  - **Critério de sucesso**: 1.1.1 (A) - Qualquer conteúdo "não textual"e relevante para compreensão da informação, deve trazer uma descrição alternativa em texto (visível ou não) para identificar o conteúdo (inclusive captcha, por exemplo)
  - ***Widget* alvo**: Icon.
  - **Algoritmo**: Mapeie todos os widgets alvo no documento, um por um, verifique se possui a propriedade "semanticLabel"definida, caso não, execute o diagnóstico.
  - **Feedback**: "O widget Icon deveria ter a propriedade semanticLabel"
  - **Por que importa?** Mesma razão da regra de *label* semântica para imagens.
- **Regra de validação**: Propósito dos botões
  - **Princípio**: robusto
  - **Critério de sucesso**: 4.1.2 (A) - Toda tecnologia assistiva faz uso das propriedades de nome, função e valor para identificar adequadamente os elementos padronizados do HTML. Qualquer componente customizado deve trazer também essas marcações de forma adequada
  - **Princípio**: operável
  - **Critério de sucesso mapeado**: 2.5.3 (A) - Rótulos em botões, ícones acionáveis ou qualquer controle interativo, devem ter uma descrição significativa tanto para quem vê, quanto para quem apenas ouve a informação.
  - ***Widgets* alvo**: ElevatedButton, FilledButton, TextButton, OutlinedButton.
  - **Algoritmo**: Mapeie todos os widgets alvo no documento, um por um, verifique se ele está dentro do widget do tipo "Semantics", caso não, execute o diagnóstico.
  - **Feedback**: "O widget * da biblioteca Material Design deveria estar encapsulado num widget Semantics que explique seu propósito"
  - **Snippet sugerido**: *SemanticsButton*

  - **Por que importa?** Os usuários de leitores de tela não conseguem discernir a finalidade de botões que não deixam claro seu propósito no seu texto interno, ou são representados de maneira não textual (apenas com ícones, por exemplo).

- **Regra de validação**: Título das páginas

  - **Princípio**: operável
  - **Critério de sucesso**: 2.4.2 (A) - Todas as telas devem ter um título principal e que descreva claramente a sua finalidade.
  - ***Widget* alvo**: Scaffold.
  - **Algoritmo**: Mapeie elementos do widget alvo, um por um, verifique se possui uma propriedade "appBar", caso não, execute o diagnóstico.
  - **Feedback**: "O widget Scaffold da biblioteca Material Design deveria conter a propriedade appBar que possa prover um título para a tela."
  - **Por que importa?** Esta regra beneficia todos os usuários ao permitir-lhes identificar de maneira rápida e fácil se a informação contida na tela é relevante para as suas necessidades. Pessoas com deficiências cognitivas, memória limitada de curto prazo e dificuldades de leitura também se beneficiam da capacidade de identificar o conteúdo pelo título. Este critério também beneficia pessoas com deficiências graves de mobilidade, cujo modo de operação depende de áudio durante a navegação entre páginas.

- **Regra de validação**: *Label* em *inputs* de texto

  - **Princípio**: robusto
  - **Critério de sucesso**: 4.1.2 (A) - Toda tecnologia assistiva faz uso das propriedades de nome, função e valor para identificar adequadamente os elementos padronizados do HTML. Qualquer componente customizado deve trazer também essas marcações de forma adequada
  - ***Widgets* alvo**: TextField, TextFormField.
  - **Algoritmo**: Mapeie elementos dos widgets alvos, um por um, verifique se possui uma propriedade "decoration", caso não, execute o diagnóstico. Caso possua, em seguida, dentro do valor passado para essa propriedade, verifique

se há uma propriedade aninhada chamada "label"ou "labelText", caso não, execute o diagnóstico.

- **Feedback**: "O widget * da biblioteca Material Design deveria conter uma label associada a ele"
- **Snippet sugerido**: *AccessibleTextInput*
- **Por que importa?** Rótulos de campos de texto eficazes são necessários para tornar os formulários acessíveis. A finalidade dos elementos do formulário costuma ser aparente para usuários com visão, mesmo que o elemento do formulário não seja rotulado programaticamente. Os usuários de leitores de tela precisam de rótulos de formulário úteis para identificar os campos do formulário. Adicionar um rótulo a todos os elementos do formulário elimina a duplicidade e contribui para um produto mais acessível. Quando os rótulos dos elementos do formulário estão ausentes, os usuários de leitores de tela não conhecem as expectativas dos dados de entrada. Os leitores de tela não podem determinar programaticamente informações sobre objetos de entrada sem um relacionamento de rótulo estabelecido.

• **Regra de validação**: Dica de preenchimento em *inputs* de texto

- **Princípio**: compreensível
- **Critério de sucesso**: Rótulos e instruções [A] - Todos os rótulos devem descrever claramente e sem ambiguidades a finalidade dos campos de formulário.
- ***Widgets* alvo**: TextField, TextFormField.
- **Algoritmo**: Mapeie elementos do widgets alvos, um por um, verifique se possui uma propriedade "decoration", caso não, execute o diagnóstico. Caso possua, em seguida, dentro do valor passado para essa propriedade, verifique se há uma propriedade aninhada chamada "hintText", caso não, execute o diagnóstico.
- **Feedback**: "O widget * da biblioteca Material Design deveria conter uma dica de preenchimento associada a ele"
- **Snippet sugerido**: *AccessibleTextInput*
- **Por que importa?** Fornecer instruções claras e corretas (como exemplos de formatos de dados esperados) ajuda todos os usuários - mas especialmente

aqueles com deficiências cognitivas, linguísticas e de aprendizagem - a inserir informações corretamente. Além disso, fornecendo instruções você pode evitar que os usuários façam envios de formulários incompletos ou incorretos, o que evita que os usuários tenham que navegar mais uma vez por uma tela/formulário para corrigir erros de envio.

3.1.3 Recursos de desabilitação de regras

Sabemos que nem todas as telas de um aplicativo devem ter um padrão (como se saísse de uma forma), e por isso, não necessariamente todas as regras de validação devem ser aplicadas em todas as telas e projetos.

Por exemplo, nem todas as telas estruturadas com o *widget* Scaffold irão "titular"a tela através de um AppBar (*widget* auxiliar que implementa uma espécie de *heading* da tela), e por isso, o alerta originado da regra de validação dos títulos das páginas pode terminar sendo desnecessária.

Por isso, a extensão dispõe de um mecanismo de desabilitação da extensão em arquivos específicos. Para isso, basta incluir um comentário no formato "// accessible-flutter disable-file"em qualquer lugar do arquivo, como demonstra a linha 7 do trecho de código 3. O *language server* se encarregará de buscar por comentários no formato antes de aplicar as regras de validação no arquivo.

```
1  import 'package:flutter/material.dart';
   void main() {
3    runApp(const MyApp());
   }
5
   class MyApp extends StatelessWidget {
7    // accessible-flutter disable-file
     const MyApp({super.key});
9    // This widget is the root of your application.

11   @override
     Widget build(BuildContext context) {
13     return MaterialApp(
         title: 'Flutter Demo',
15       theme: ThemeData(
```

```
        colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),
17      useMaterial3: true,
      ),
19    home: const Scaffold(),
    );
21  }
}
```

Trecho de Código 3: Uso do recurso para desabilitar extensão no arquivo

3.2 Seções importantes da implementação de Cliente e Servidor

Nesta seção, abordaremos trechos de códigos importantes na implementação do *language client* e *language server*, explicando o que fazem, e os seus por quês.

3.2.1 Definições importantes no manifesto da extensão

Toda extensão para o Visual Studio Code precisa de um arquivo manifesto chamado *package.json* na raiz do projeto da extensão. Entre os campos a serem definidos, vamos destacar os mais relevantes, e que estão diretamente associados ao funcionamento da extensão.

3.2.1.1 Eventos de Ativação

O campo "activationEvents"representa os eventos que vamos ouvir para ativar a nossa extensão. No nosso caso, queremos que a extensão ative toda vez que um arquivo de extensão *.dart* seja aberto. Para isso, utilizamos o evento *onLanguage*, como mostra o trecho de código 4.

```
1 "activationEvents": [
2   "onLanguage:dart"
3 ],
```

Trecho de Código 4: Eventos de ativação definidos no manifesto da extensão

3.2.1.2 Pontos de Contribuição

O campo "contributes"são declarações em JSON para extender diversas funcionalidades dentro do VSCode. Conforme visualizames na definição das *features*, queremos definir *snippets* para extender as funcionalidades do VSCode para auxiliar o desenvolvedor. Para isso, definimos a direção do arquivo que contém os snippets e a linguagem para qual eles server dentro de um campo "snippets"aninhado no campo inicial.

Também podemos definir chaves de configuração através do campo "configuration". Essas configurações podem ser alteradas pelo usuário através das configurações do usuário ou configurações do *workspace*, que podem ser definidas de forma geral no VSCode do desenvolvedor diretamente. Apenas para exemplificar, definiremos o número máximo de problemas que podem ser diagnosticados pelo servidor como uma chave de configuração.

Dessa forma, nossos pontos de contribuição podem ser visualizados no trecho de código 5.

```
"contributes": {
    "configuration": {
        "type": "object",
        "title": "Server configuration",
        "properties": {
            "acessibleFlutterServer.maxNumberOfProblems": {
                "scope": "resource",
                "type": "number",
                "default": 100,
                "description": "Controls the maximum number
                   of problems produced by the server."
            }
        }
    },
    "snippets": [
        {
            "language": "dart",
            "path": "./snippets/snippets.json"
        }
```

```
19      ]
20  }
```

Trecho de Código 5: Pontos de contribuição definidos no manifesto da extensão

3.2.2 Conexão do lado do cliente

O *language client* utiliza as funções mais básicas de criação de extensões que são fornecidas pelo SDK do VSCode. Sua comunicação com o servidor, por baixo, utiliza o IPC (inter-process communication), que são mecanismos fornecidos por um sistema operacional para gerenciar dados compartilhados [22]. As configurações do cliente e do servidor, seja executando em modo *debug* ou em modo normal podem ser definidos através de objetos, como mostra a implementação do cliente no trecho de código 6.

```
export function activate(context: ExtensionContext) {
2   // Server implementado em node
    let serverModule = context.asAbsolutePath(path.join('server', 'out', '
        server.js'));
4   // As opcoes de debug para o server
    // --inspect=6009: roda o servidor no modo de inspetor do Node de forma
        que o VS Code
6   // consegue "anexar" o servidor para debugar
    let debugOptions = { execArgv: ['--nolazy', '--inspect=6009'] };
8
    // Se a extensao eh rodada em modo debug, as opcoes do servidor de
        debug sao utilizadas
10  // Caso contrario, as opcoes de inicializacao sao utilizadas
    let serverOptions: ServerOptions = {
12    run: { module: serverModule, transport: TransportKind.ipc },
      debug: {
14      module: serverModule,
        transport: TransportKind.ipc,
16      options: debugOptions
      }
18  };

20  // Opcoes para controlar o language client
    let clientOptions: LanguageClientOptions = {
```

```
22      // Registra o servidor para documentos dart
        documentSelector: [{ scheme: 'file', language: 'dart' }],
24      synchronize: {
          // Notifica o servidor sobre mudancas nos arquivos de extensao '.
              clientrc'
26        // contidos nas pastas do projeto
          fileEvents: workspace.createFileSystemWatcher('**/.clientrc')
28      }
    };
30
    // Cria o language client e o inicia
32  client = new LanguageClient(
      'afclient', // id
34    'Accessible Flutter Client', // nome
      serverOptions,
36    clientOptions
    );
38
    // Inicia o cliente, isso tambem executa o language server
40  client.start();
}
```

Trecho de Código 6: Implementação do *language client* da extensão

3.2.3 Produção dos diagnósticos no *language server*

A execução dos diagnósticos resultantes da aplicação das regras de validação no servidor seguem um algoritmo simples. Na sua inicialização, nos inscrevemos no evento de mudança de conteúdo nos arquivos abertos na IDE do desenvolvedor. Toda vez que há qualquer mudança no conteúdo, a função de validação do documento executa mais uma vez, como mostra o trecho de código 7.

```
1  documents.onDidChangeContent(change => {
       validateTextDocument(change.document);
3  });
```

Trecho de Código 7: Inscrição no evento de edição de documentos no servidor

Em cada execução da função de validação, a primeira coisa que verificamos é se o documento em específico está habilitado ou não. Isto é, temos que buscar por comentários

no arquivo no padrão "// accessible-flutter disable-file". Caso encontremos algum *token* como esse, não executaremos as regras de validação. Essa verificação pode ser observada no trecho de código 8.

```
async function validateTextDocument(textDocument: TextDocument): Promise<
    void> {
    let settings = await getDocumentSettings(textDocument.uri);
    let m: RegExpExecArray | null;
    let problems = 0;
    let diagnostics: Diagnostic[] = [];
    let text = textDocument.getText();

    const rows = text.split("\n");
    const isFileDisabled = rows.filter((row) => row.indexOf("//
        accessible-flutter disable-file") !== -1).length > 0;
    if (isFileDisabled) {
      connection.sendDiagnostics({ uri: textDocument.uri, diagnostics });
      return;
    }
    ...
}
```

Trecho de Código 8: Checagem para verificar se o documento desabilitou as regras de validação

Caso o arquivo esteja habilitado para aplicarmos regras de validação, partimos para a execução delas. Para isso, definimos uma estrutura auxiliar para cada regra de validação. Trata-se de uma lista de objetos, onde cada um reúne uma expressão regular que captura os *tokens* que queremos observar no documento para aplicar a regra, e um método que irá verificar propriamente a regra a partir de cada token, e do documento como um todo. Chamamos eles de *pattern* e *exec*, respectivamente, como mostra o trecho de código 9.

```
export const rulesList = [
    {
        pattern: /(?:Image|Icon|Image.network|Image.asset|Image.file|
            Image.memory)\((?:[^)(]+|\((?:[^)(]*|\([^)(]*\))*\))*\)/g,
        exec: imageAltExecutor,
    },
    {
```

```
        pattern: /(?:Elevated|Outlined|Text|Filled)Button\((?:[^)
            (]+|\((?:[^)(]*|\([^)(]*\))*\))*\)/g,
        exec: buttonNameExecutor,
    },
    {
        pattern: /\bScaffold\(/g,
        exec: pageTitleExecutor,
    },
    {
        pattern: /(?:TextField|TextFormField)\((?:[^)(]+|\((?:[^)
            (]*|\([^)(]*\))*\))*\)/g,
        exec: labelExecutor,
    },
    {
        pattern: /(?:TextField|TextFormField)\((?:[^)(]+|\((?:[^)
            (]*|\([^)(]*\))*\))*\)/g,
        exec: inputSuggestionExecutor,
    },
];
```

Trecho de Código 9: Estrutura auxiliar para regras de validação

De posse dessa ferramenta auxiliar, para cada regra validação dessa lista, executamos um laço condicional, com sua condição de saída sobre 2 regras: ou quando a expressão regular da regra não encontrar mais *tokens* relevantes para aquela regra, ou quando já ultrapassamos o número máximo de problemas diagnosticados definida previamente. Para cada um, passamos para a nossa função de aplicação da regra, que nos retornará se aquele token deve ou não ter um diagnóstico. Caso deva, adicionamos à lista de diagnósticos. Caso contrário, passamos para a próxima iteração do laço. Após aplicar todas as regras de validação, enviamos de volta para o *language client* a lista de diagnósticos que devem aparecer para o desenvolvedor, como mostra o trecho de código 10.

```
for (const { pattern, exec } of rulesList) {
    while ((m = pattern.exec(text)) && problems < settings.
        maxNumberOfProblems) {
        const { hasProblem, relatedInformationMessages = [], message = ""
            } = exec(m[0], m);
        if (hasProblem) {
          problems++;
```

```
6           let diagnostic: Diagnostic = {
              severity: DiagnosticSeverity.Warning,
8             range: {
                start: textDocument.positionAt(m.index),
10              end: textDocument.positionAt(m.index + m[0].length)
              },
12            source: 'Accessible Flutter',
              message: message,
14          };
            if (hasDiagnosticRelatedInformationCapability) {
16            const relatedInformation = [];
              for (const message of relatedInformationMessages) {
18              relatedInformation.push(
                  {
20                  location: {
                      uri: textDocument.uri,
22                    range: Object.assign({}, diagnostic.range)
                    },
24                  message,
                  },
26              );
              }
28            diagnostic.relatedInformation = relatedInformation;
            }
30          diagnostics.push(diagnostic);
          }
32    }
}
34

connection.sendDiagnostics({ uri: textDocument.uri, diagnostics });
```

Trecho de Código 10: Laço de aplicação de cada regra de validação

3.3 Interface do usuário no uso das funcionalidades

Nesta seção, abordaremos brevemente a interface do usuário nas funcionalidades da extensão de modo geral. Iniciando da sua visualização no *marketplace* de extensões do VSCode, a extensão conta com uma tela de detalhes sobre a extensão muito explicativa,

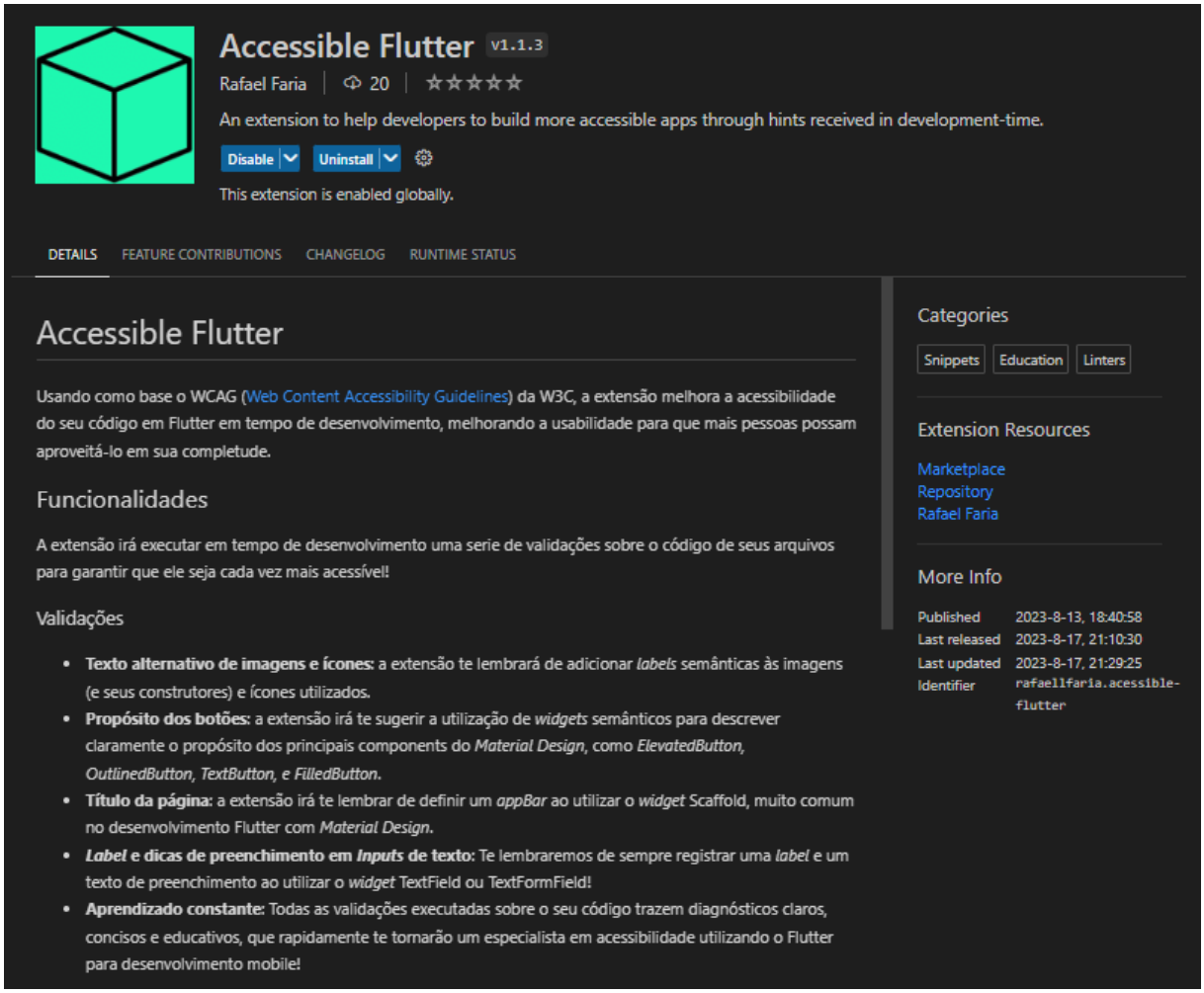

Figura 5: Tela de descrição da extensão no *marketplace* do VSCode.
Fonte: https://marketplace.visualstudio.com/items?itemName=rafaellfaria.acessible-flutter.

que aborda tudo que o usuário deve saber sobre o Accessible Flutter e como utilizá-lo, como mostra a Figura 5.

As regras de validação aparecem para os usuários destacadas como *warning*, com um sublinhado amarelo. O conteúdo textual das caixas flutuantes quando o mouse é passado por cima seguem o mesmo padrão: mostramos o feedback, o critério de sucesso mapeado, um link para ler mais sobre ele e, se houver, uma sugestão de *snippet*, como mostra a Figura 6.

Já a sugestão de *snippets* aparece num menu de seleção assim que você digita a sua palavra de acesso, como mostra a Figura 7. Após selecionar a opção com o mouse ou apertanto a tecla enter, o *snippet* é carregado no código para ser utilizado.

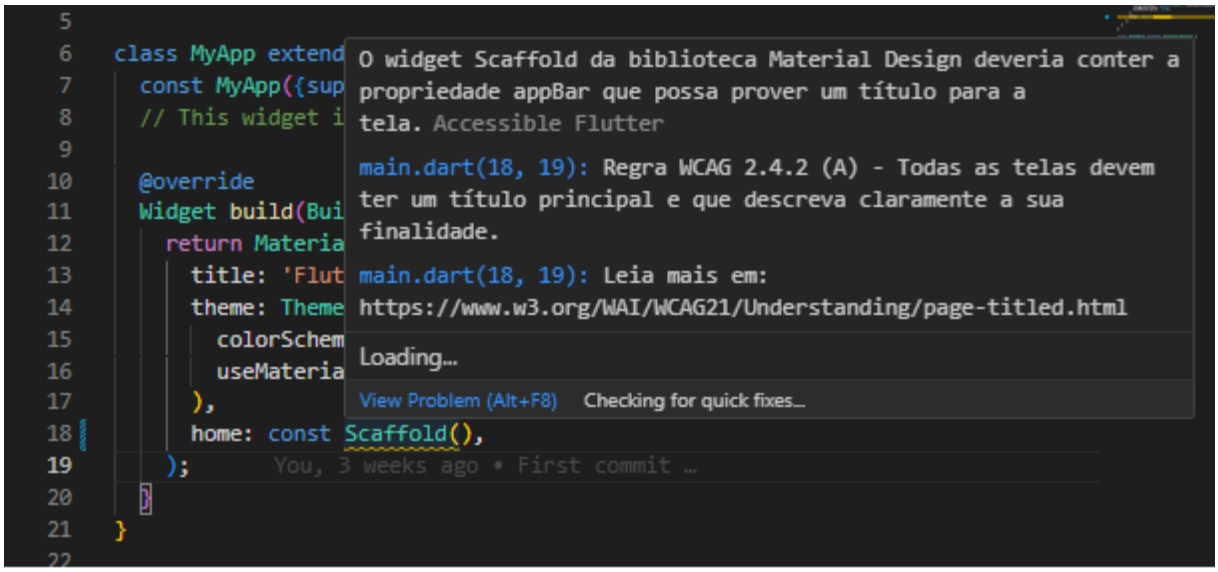

Figura 6: Exemplo da interface gráfica dos diagnósticos resultantes da aplicação das regras de validação.

Fonte: AUTOR, 2023.

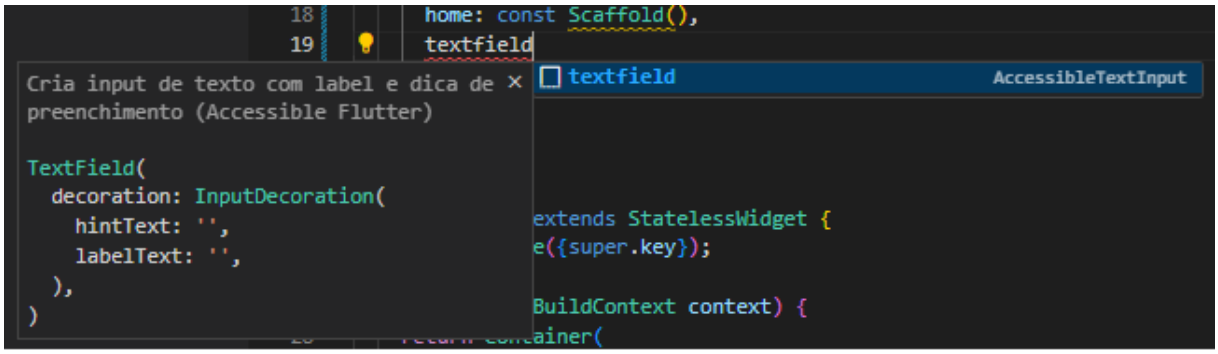

Figura 7: Exemplo de utilização de *snippet* no desenvolvimento.
Fonte: AUTOR, 2023.

4 ANÁLISE QUANTITATIVA E QUALITATIVA DA USABILIDADE DA EXTENSÃO

Neste capítulo, descreveremos o processo de teste e análise da usabilidade com usuários reais. Para análise da experiência dos usuários, fizemos uma rodada de entrevistas com o objetivo de extrair informações sobre o auxílio à usabilidade da nossa extensão. Primeiramente, os usuários participaram de uma entrevista síncrona. Nela, fizemos algumas perguntas básicas comuns à todas as entrevistas, e em seguida, concedemos acesso à extensão para o usuário realizasse alguns testes semi-direcionados. Ao final, os entrevistados responderam um formulário com perguntas do SUS.

A análise quantitativa da experiência se deu através das respostas desse formulário. A análise qualitativa se deu através da nossa percepção de uso da extensão pelos entrevistados, respostas das perguntas do roteiro e *feedbacks* relatados durante o uso da extensão.

4.1 Processo de entrevistas

As entrevistas foram feitas num intervalo de uma semana. Após as entrevistas, fizemos uma análise sobre as anotações feitas durante as mesmas. Cada um dos entrevistados já teve experiências desenvolvendo em Flutter. A avaliação qualitativa desses candidatos é importante para entender o entendimento para com a extensão e o sentimento enquanto a utiliza.

Cada entrevista foi feita individualmente em ambiente físico, onde o entrevistador e entrevistado estavam com seus computadores pessoais. Por esse motivo, não foi possível realizar gravação da entrevista, cabendo ao entrevistador registrar todos os *feedbacks* diretos e indiretos da entrevista. Entrevistamos um grupo muito limitado, de apenas 5 pessoas, com seus perfis descritos na Tabela 1.

Uma potencial brecha para falhas nessa metodologia de teste foi o critério de recrutamento dos entrevistados: todos os entrevistados são pessoas que trabalham no ecossistema de empresas de tecnologia do centro da cidade do Recife, no estado de Pernambuco, Brasil, local no qual o entrevistador está inserido. Além disso, todas as 5 pessoas conheciam previamente o entrevistador, que deve ser um ponto relevante a se considerar ao analisar o resultado dos testes.

Entrevistado	Formação	Ocupação	Nacionalidade
E1	Estudante de EC	Desenvolvedor	Brasileiro
E2	Mestre em Física	Desenvolvedor	Brasileiro
E3	Estudante de SI	Desenvolvedor	Brasileiro
E4	Estudante de EC	Desenvolvedor	Brasileiro
E5	Estudante de ADS	Desenvolvedor	Brasileiro

Tabela 1: Perfil dos entrevistados

4.2 **Roteiro da entrevista**

O roteiro da entrevista seguiu um modelo pré-definido de entrevista com um teste de usabilidade da extensão. O tempo estimado para a condução da mesma foi de cerca de 20 a 30 minutos. No primeiro momento, realizamos as seguintes perguntas básicas, comuns a todas as entrevistas conduzidas:

- Você conhece alguma ferramenta voltada para o desenvolvimento em Flutter que auxilia a acessibilidade dos produtos de forma geral?
- Você tem conhecimento de alguma boa prática de acessibilidade que você busca seguir quando desenvolvendo com o Flutter?

A partir das perguntas iniciais, fornecemos o acesso à extensão e disponibilizamos um repositório base no Github com um projeto em Flutter básico para conduzir o teste de acessibilidade. Assim que o teste começa, o entrevistado tem espaço pra dar feedbacks a qualquer momento que achar pertinente. Quando o usuário instala a extensão e inicia o repositório em seu computador, começamos então o teste semi-direcionado de usabilidade.

Nele, temos um roteiro de atividades a serem realizadas no código, para que o entrevistado desenvolva da forma como preferir. A lista é a seguinte:

- Estruture a página inicial do projeto utilizando o *widget* Scaffold.
- Adicione um título à página principal chamado "Analisador de Artes".
- Adicione uma imagem de uma arte da internet centralizada nessa tela (o entrevistador então fornece o link da imagem).
- Adicione um campo de texto para o usuário dar um nome à arte.
- Adicione um botão para que o usuário submeta a arte.

Durante cada interação com os pedidos, analisamos a usabilidade da extensão no fluxo de desenvolvimento dos entrevistados. Vale ressaltar que acompanhamos sem intervenções, respondendo apenas aos feedbacks e interações diretas do usuário. Ao fim da fase do teste de usabilidade, realizamos cinco perguntas básicas adaptadas dos conceitos do processo da estratégia de negócio Blue Ocean [23]:

- Para você, o que deveria ser eliminado da extensão?
- Para você, o que deveria ser menos explorado da extensão?
- Para você, o que deveria ser mantido na extensão?
- Para você, o que deveria ser mais explorado na extensão?
- Para você, o que deveria ser adicionado à extensão?

Após a realização das perguntas, compartilhamos o link para o formulário do SUS e finalizamos a condução da entrevista.

4.3 **Resultado do questionário SUS**

Nesta seção, apresentaremos o resultado do formulário SUS respondido em todas as entrevistas. Na Tabela 2, podemos ver as respostas para as 10 perguntas do SUS. As perguntas se encontram abaixo e para cada uma das perguntas o usuário responde de 1 a 5, sendo 1 "Discordo totalmente"e 5 "Concordo totalmente". Devido ao seu tamanho, o formulário é mais rápido de ser respondido, com uma única métrica de comparação. O resultado visto na tabela é de 88,5 pontos, o que é um bom resultado considerando a média da escala, que é 68 pontos.

- Pergunta 1: Eu acho que gostaria de usar esta extensão frequentemente.
- Pergunta 2: Eu achei a extensão desnecessariamente complexa
- Pergunta 3: Eu achei que esta extensão foi fácil de usar.
- Pergunta 4: Eu acho que eu precisaria do suporte de uma pessoa com conhecimentos técnicos para ser capaz de usar esta extensão.
- Pergunta 5: Eu achei que várias funções nesta extensão estão bem integradas.

- Pergunta 6: Eu achei que existiram várias inconsistências nesta extensão.
- Pergunta 7: Eu imagino que a maioria das pessoas aprenderia a usar esta extensão rapidamente.
- Pergunta 8: Achei a extensão muito complicada de usar.
- Pergunta 9: Me senti confiante usando a extensão.
- Pergunta 10: Eu precisaria aprender muitas coisas antes que eu pudesse me sentir a vontade em usar esta extensão.

E	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	Score SUS	Score Final SUS
E1	5	1	3	1	4	2	3	1	4	2	32	80
E2	5	1	5	1	4	1	5	1	5	1	39	97,5
E3	5	1	5	1	5	1	5	1	5	1	40	100
E4	4	1	4	1	3	2	4	2	4	1	32	80
E5	4	1	5	3	5	2	5	1	4	2	34	85
Média											35,4	88,5

Tabela 2: Respostas do questionário SUS pelos entrevistados

4.4 **Análise qualitativa das entrevistas**

Nesta seção, dissertaremos sobre os pontos de *feedback* (positivos e negativos) mais comuns que aconteceram ao longo da condução das entrevistas, bem como todos os pontos relevantes percebidos e anotados durante o teste semi-direcionado a respeito da extensão.

Dividimos eles em três grupos de acordo com sua natureza. São eles Usabilidade, Interação com as Funcionalidades, e Conhecimento e Preocupação sobre Acessibilidade.

4.4.1 Usabilidade

Entre os principais pontos relacionados à usabilidade, destacam-se:

- Em uma das entrevistas, a extensão sofreu um *crash* quando o usuário executou uma ação específica, sendo necessário recarregar o arquivo para a extensão voltar a funcionar adequadamente.

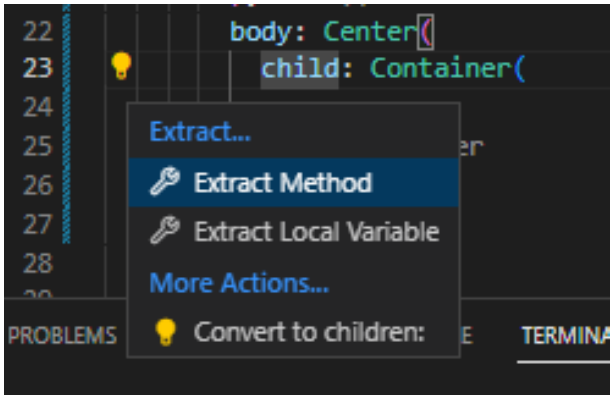

Figura 8: Exemplo da interação com a ferramenta *lightbulb* da IDE.
Fonte: AUTOR, 2023.

- Um *feedback* comum por parte dos entrevistados foi que o *link* disponibilizado no retorno para o desenvolvedor não era clicável, dificultando assim do usuário ler mais sobre os critérios de sucesso da WCAG que foram mapeados.

- Um comportamento percebido pelo entrevistador e destacado por parte dos entrevistados é o costume dos desenvolvedores de utilizar o recurso de *lightbulb* do VSCode para ajustar trechos de código com problema. Essa ferramenta, que aparece como um ícone de uma lâmpada ao lado de trechos de código destacados pelo cursor, sugere uma lista de refatorações e que através de um clique em uma das opções dessa lista, reescreve o código específico, como mostra a Figura 8. Essa é uma das limitações mais relevantes observadas nessa versão inicial da extensão, que apenas destacava a sugestão de mudança e possíveis dicas, mas não utilizando essa ferramenta disponível na IDE.

4.4.2 Interação com as funcionalidades

Entre os principais pontos de destaque com relação à interação com as funcionalidades, destacam-se:

- Um comportamento notado por parte do entrevistador em todas as 5 entrevistas foi a reação positiva às dicas oriundas da aplicação das regras de validação. O entrevistado E3, por exemplo, destacou que é muito interassante que a extensão além de sugerir mudanças, explicava o porquê e o seu embasamento. Além disso, todos constantemente demonstravam estar surpresos como mudanças tão pequenas fazem diferença quanto à acessibilidade.

- A interação dos usuários com os *snippets* da extensão destoou muito de entrevista a entrevista. Alguns reagiram muito positivamente, utilizando-as quase naturalmente. Nas palavras do entrevistado E2, essa funcionalidade é muito interessante, pois é um recurso da IDE comumente utilizado por ele. Já para o entrevistado E4, por exemplo, os *snippets* pouco ajudariam, pois usualmente, nos projetos nos quais ele trabalha, utiliza outros tipos de *widgets* para construir os botões, diferentes dos mapeados atualmente pela extensão no *snippet*.

- Outro recurso sentido em algumas entrevistas foi a falta de customização das regras habilitadas. Nas palavras do entrevistado E2, como os projetos nos quais ele trabalha têm naturezas muito distintas, e protótipos oriundos do processo de design muito particulares, algumas regras terminam não fazendo sentido em um ou outro projeto, e por isso achou a ideia de habilitar ou desabilitar apenas todo o arquivo um pouco limitante.

4.4.3 Conhecimento e Preocupação sobre Acessibilidade

Através das perguntas iniciais de todas as entrevistas, notamos que é comum a falta de conhecimento e preocupação com acessibilidade nos produtos digitais. Nenhum dos 5 entrevistados conhecia nenhuma ferramenta que auxiliasse na acessibilidade para o usuário final dos aplicativos usando Flutter. Já alguns sabiam da existência do *widget* padrão do *framework* Flutter chamado Semantics, que tem exatamente o propósito de prover camadas de contexto maiores sobre cada componente da tela.

A maioria dos entrevistados declarou que não costuma priorizar a acessibilidade no fluxo de desenvolvimento, ainda que conheçam algumas boas práticas no desenvolvimento *web* e *mobile*.

5 TRABALHOS RELACIONADOS

Neste capítulo, iremos olhar os trabalhos relacionados à temática abordada ao longo desse estudo que apresentam semelhanças em certo nível.

5.1 Entendimento de ferramentas semelhantes

Nesta subseção, falaremos um pouco sobre ferramentas e iniciativas disponíveis no mercado que buscam solucionar problemas semelhantes de acessibilidade, mesmo que de maneiras um pouco diferentes.

5.1.1 Guia WCAG

O Guia WCAG é uma iniciativa do designer Marcelo Sales de dispor, em formato de *cards* digitais, todos os critérios de sucesso das diversas diretrizes da WCAG [24]. Constantemente sincronizado com as atualizações das diretrizes da WCAG, e dispondo de um buscador, o *site* é uma ótima ferramenta de consulta rápida a critérios específicos.

Os critérios são organizados de forma que sejam facilmente assimilados, dispondo de informações como código do critério de sucesso, título do critério de sucesso, nível de conformidade (A, AA ou AAA), princípios (diferenciado por cores), recomendações (seguem as mesmas cores dos princípios), descrição resumida do critério e link para descrição completa do critério.

Como podemos verificar na Figura 9, de maneira visualmente agradável, fácil de encontrar e muito educativa, o Guia WCAG cumpre muito bem o papel informativo ao qual se propõe.

5.1.2 Movimento Web Para Todos

O Movimento Web Para Todos é uma rede que conecta organizações públicas e privadas, desenvolvedores, pessoas de design e comunicação, e pessoas com deficiência com o objetivo de mobilizar a sociedade a respeito da acessibilidade digital, transformando assim os sites brasileiros em um ambiente cada vez mais inclusivo [25].

Além de organizar eventos dos mais diversos tipos, o movimento realiza consultorias especializadas, produz relatórios técnicos sobre acessibilidade na *web* e adequa ambientes

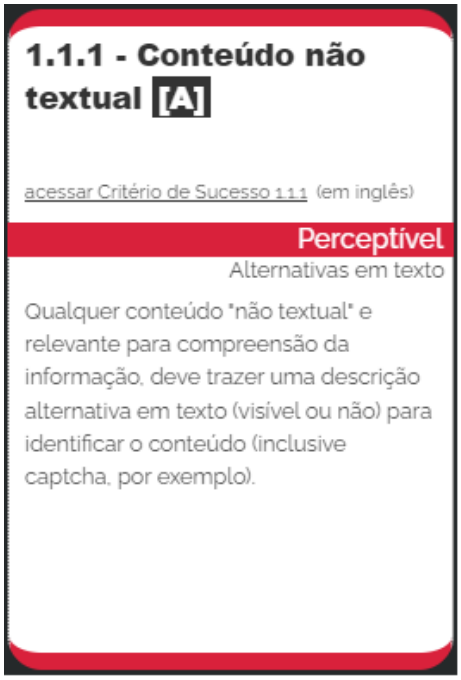

Figura 9: *Card* de critério de sucesso de uma diretriz da WCAG.
Fonte: https://guia-wcag.com/.

virtuais para receberem o Selo de Acessibilidade Digital [26], outorgado pela prefeitura da cidade de São Paulo.

Em sua plataforma online, o movimento dispõe de ferramenta automática que analisa a página principal de um site gratuitamente, indicando o nível de acessibilidade com base nas diretrizes da WCAG 2.0.

5.1.3 Lighthouse

O Lighthouse é uma ferramenta de código aberto da Google, que tem a finalidade de melhorar a qualidade de *websites*. Ela faz avaliações de performance, acessibilidade, SEO, segurança e outros fatores em pontuações que vão de 0 a 100. A ferramenta pode ser executada como uma extensão para o navegador Google Chrome, ou como um módulo Node.

Originalmente criada para analisar PWAs, terminou tendo sua funcionalidade extendida para auxiliar desenvolvedores e designers. Através de poucos cliques, a ferramenta realiza uma avaliação automatizada do *site* em várias categorias, entre elas, a de acessibilidade, como mostra a Figura 10.

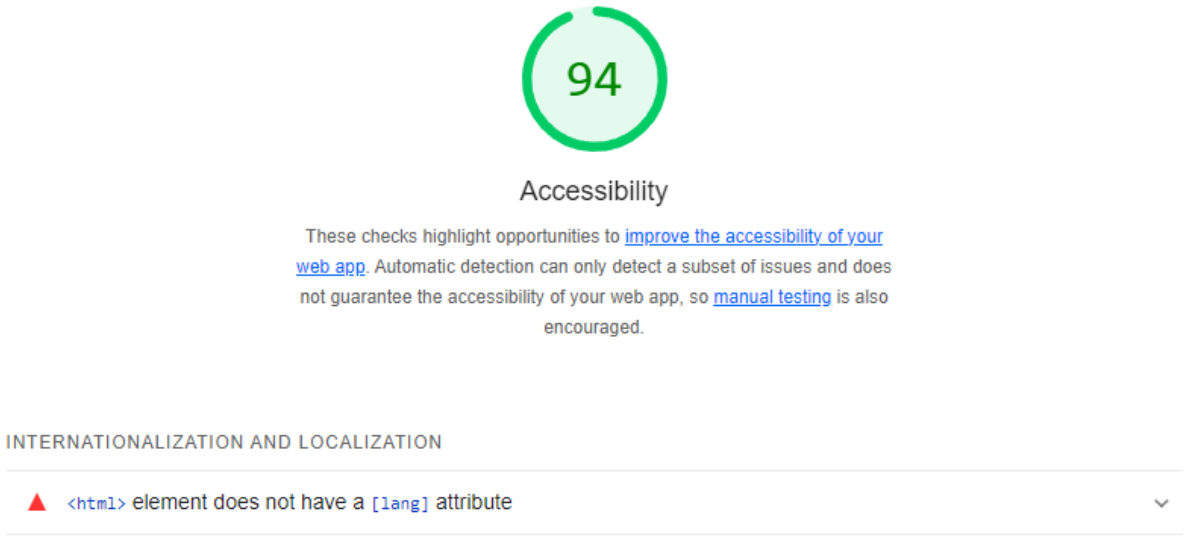

Figura 10: Análise da performance de um *site* pelo Lighthouse.
Fonte: AUTOR, 2023.

5.1.4 axe Accessibility Linter

O axe Acessibility Linter trata-se de uma extensão disponível no *marketplace* de extensões do VSCode, e que atua como uma ferramenta de análise de código estática que procura defeitos de acessibilidade em projetos de desenvolvimento *web* nos mais diversos *frameworks*, como React.js, Angular, Vue, HTML e arquivos de Markdown [27].

Suas regras de validação são todas baseadas nas diretrizes da WCAG 2.0 e WCAG 2.1. A ferramenta disponibiliza na internet uma lista com todas as regras implementadas, bem como a referência de critério de aceitação das diretrizes, explicação da importância e algoritmo [28].

O resultado da sua validação acontece em tempo real, alertando o desenvolvedor em tempo de desenvolvimento, como mostra a Figura 11.

5.2 **Reflexão sobre os trabalhos relacionados**

Como pudemos observar na seção anterior, há muitas ferramentas e iniciativas para mobilizar empresas, desenvolvedores e designers sobre a importância da acessibilidade na *web*. Algumas realizando uma avaliação posterior do *site*, outras voltadas para a educação sobre acessibilidade digital das pessoas, e também algumas que auxiliam ainda em tempo de desenvolvimento.

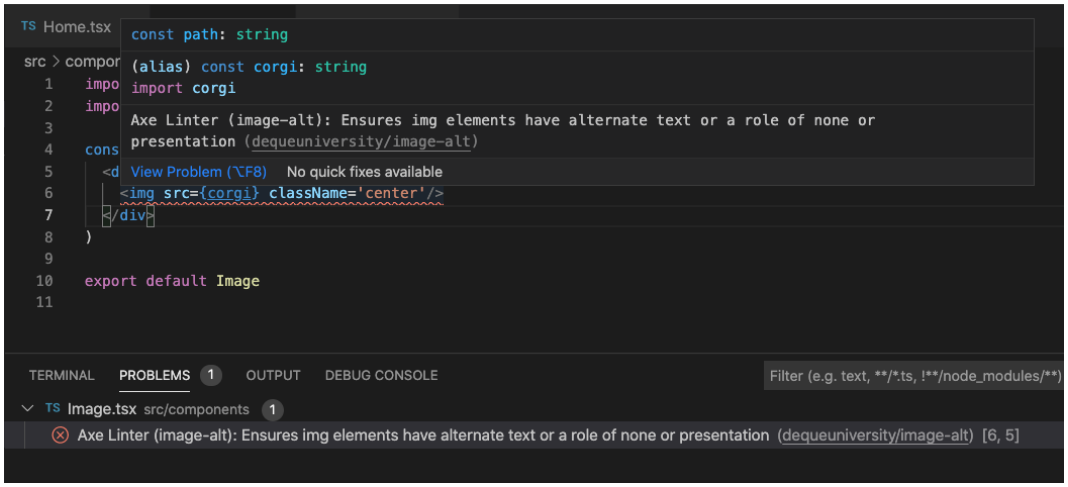

Figura 11: Exemplo de funcionamento da ferramenta axe Accessibility Linter. Fonte: https://marketplace.visualstudio.com/items?itemName=deque-systems.vscode-axe-linter.

Ao pensar sobre o que foi construído ao longo do estudo, notamos que a extensão Accessible Flutter absorve pontos positivos de cada uma delas no que se propõe a fazer, trazendo para si um diferencial competitivo de reunir uma variedade de maneiras de atacar o problema da falta de acessibilidade no ambiente *mobile*.

6 CONCLUSÃO E TRABALHOS FUTUROS

Através das entrevistas conduzidas, notamos que a usabilidade do sistema apresentou valores positivos para as pessoas desenvolvedoras de forma geral, mas que também deixaram claras as oportunidades de melhoria. No que se propõe a fazer, uma extensão que auxilia na acessibilidade para o usuário final deve preferencialmente demandar o mínimo de tempo a mais possível para o desenvolvedor do sistema, sendo idealmente nenhum tempo extra.

A extensão Accessible Flutter trouxe uma camada extra de importância às questões de acessibilidade, constantemente negligenciadas no desenvolvimento de produtos e sistemas digitais, mas ela não pode falhar nem sofrer *crashes* no seu uso pelas pessoas. Esses problemas podem ser mitigados, por exemplo, por uma cobertura de testes unitários completa sobre as funcionalidades da extensão e um sistema de *logs* de falha robustos e com pilhas de rastreamento dos erros relevante.

Do ponto de vista do uso do desenvolvedor, há muita oportunidade de melhoria: a mais relevante, obviamente, é de trazer a extensão para o inglês, por questões de acessibilidade da linguagem e que de certa forma, pelo fato de estar apenas em português, a extensão "contradiz"o seu próprio propósito. Além disso, implementar mais regras de validação baseadas em critérios de sucesso da WCAG, seja aumentando a cobertura de *widgets* verificados nessas regras (se couber validação a eles), ou aumentando a lista de regras de validação. Também podemos aumentar a quantidade de *snippets* interessantes e que acelerem o desenvolvimento da extensão, melhorar a usabilidade dos *feedbacks* tornando os links clicáveis, explorar mais recursos da API de extensões do VSCode (como o *lightbulb*), e adicionar uma camada superior de customização das regras de validação habilitadas além de habilitar tudo ou desabilitar tudo.

Para melhorar o ciclo de desenvolvimento da extensão, podemos adicionar fluxos de *Continuous Integration* (CI) e *Continuous Deployment* (CD) para acelerar o processo de lançamento de versões na loja de extensões do VSCode. Também podemos fazer o uso do protocolo de comunicação LSP proposto pela Microsoft para implementar versões da extensão para outras IDEs. Outra possibilidade a ser analisada é transformar o projeto da extensão em um *software open-source* a fim de acelerar o desenvolvimento e aumentar a qualidade das funcionalidades.

REFERÊNCIAS

[1] MIATO, B. *Brasil tem 18,6 milhões de pessoas com deficiência, cerca de 8,9% da população, segundo IBGE. Portal G1, 2023.* Disponível em: <https://g1.globo.com/economia/noticia/2023/07/07/brasil-tem-186-milhoes-de-pessoas-com-deficiencia-cerca-de-89percent-da-populacao-segundo-ibge.ghtml>. Acesso em: 27/08/2023.

[2] PESSOAS com deficiência. IBGE Educa, 2010. Disponível em: <https://educa.ibge.gov.br/jovens/conheca-o-brasil/populacao/20551-pessoas-com-deficiencia.html>. Acesso em: 27/08/2023.

[3] BRASIL. Lei nº 13.146, de 6 de julho de 2015. Institui a Lei Brasileira de Inclusão da Pessoa com Deficiência (Estatuto da Pessoa com Deficiência). Brasília, DF, [2015]. Disponível em: <https://www.planalto.gov.br/ccivil_03/_ato2015-2018/2015/lei/l13146.htm>. Acesso em: 27/08/2023.

[4] MENOS de 1% dos sites brasileiros são considerados acessíveis, diz pesquisa. Forbes, 2021. Disponível em: <https://forbes.com.br/forbesesg/2021/07/menos-de-1-dos-sites-brasileiros-sao-considerados-acessiveis-diz-pesquisa/>. Acesso em: 27/08/2023.

[5] VANDERHEIDEN, G. C.; CHISHOLM, W. A.; JACOBS, I. *Web Content Accessibility Guidelines 1.0.* Disponível em: <https://www.w3.org/TR/WAI-WEBCONTENT/>. Acesso em: 27/08/2023.

[6] QUERINI, V. *What Is Inclusive Design? A Beginner's Guide.* Disponível em: <https://careerfoundry.com/en/blog/ux-design/beginners-guide-inclusive-design/>. Acesso em: 27/08/2023.

[7] VANDERHEIDEN, G. C. *Design of HTML (Mosaic) Pages to Increase their Accessibility to Users with Disabilities Strategies for Today and Tomorrow.* Disponível em: <https://trace.umd.edu/design-of-html-mosaic-pages-to-increase-their-accessibility-to-users-with-disabilities-strategies-for-today-and-tomorrow/>. Acesso em: 26/08/2023.

[8] VANDERHEIDEN, G. C.; CHISHOLM, W. A. *Unified Web Site Accessibility Guidelines*. Disponível em: <https://www.w3.org/WAI/GL/central.htm>. Acesso em: 26/08/2023.

[9] KIRKPATRICK, A. et al. *Web Content Accessibility Guidelines (WCAG) 2.1*. Disponível em: <https://www.w3.org/TR/WCAG21/>. Acesso em: 26/08/2023.

[10] MICROSOFT. *Visual Studio Code - Code Editing. Redefined. Página inicial*. Disponível em: <https://code.visualstudio.com/>. Acesso em: 27/08/2023.

[11] GOOGLE. *Flutter - Build apps for any screen. Página inicial*. Disponível em: <https://flutter.dev/>. Acesso em: 27/08/2023.

[12] BANDEIRA, K. *Flutter: ferramenta facilita a vida de quem quer desenvolver aplicativos para smartphones. Folha de Pernambuco, 2022*. Disponível em: <https://www.folhape.com.br/colunistas/tecnologia-e-games/flutter-ferramenta-facilita-a-vida-de-quem-quer-desenvolver-aplicativos-para-smartphones/29632/>. Acesso em: 27/08/2023.

[13] MICROSOFT. *GitHub: Let's build from here. Página inicial*. Disponível em: <https://github.com/>. Acesso em: 27/08/2023.

[14] JOHNSON, K. *GitHub passes 100 million repositories. Venture Beat, 2018*. Disponível em: <https://venturebeat.com/ai/github-passes-100-million-repositories/>. Acesso em: 27/08/2023.

[15] NODE.JS. *Node.js. Página inicial*. Disponível em: <https://nodejs.org/en>. Acesso em: 27/08/2023.

[16] MICROSOFT. *TypeScript: JavaScript With Syntax For Types. Página inicial*. Disponível em: <https://www.typescriptlang.org/>. Acesso em: 27/08/2023.

[17] REGULAR expression. In: WIKIPEDIA, a enciclopédia livre. Flórida: Wikipedia Fundation, 2023. Disponível em: <https://en.wikipedia.org/wiki/Regular_expression>. Acesso em: 27/08/2023.

[18] MICROSOFT. *Visual Studio Code - Code Editing. Redefined. Extension API*. Disponível em: <https://code.visualstudio.com/api>. Acesso em: 28/08/2023.

[19] MICROSOFT. *Visual Studio Code - Code Editing. Redefined. Language Extensions Overview.* Disponível em: <https://code.visualstudio.com/api/language-extensions/overview>. Acesso em: 28/08/2023.

[20] MICROSOFT. *Official page for Language Server Protocol. Página inicial.* Disponível em: <https://microsoft.github.io/language-server-protocol/>. Acesso em: 28/08/2023.

[21] LEWIS, J. R. The System Usability Scale: Past, Present, and Future. *International Journal of Human–Computer Interaction*, v. 34, p. 577–590, Mar 2018.

[22] INTER-PROCESS communication. In: WIKIPEDIA, a enciclopédia livre. Flórida: Wikipedia Fundation, 2023. Disponível em: <https://en.wikipedia.org/wiki/Inter-process_communication>. Acesso em: 07/09/2023.

[23] BLUE Ocean Strategy. In: WIKIPEDIA, a enciclopédia livre. Flórida: Wikipedia Fundation, 2023. Disponível em: <https://en.wikipedia.org/wiki/Blue_Ocean_Strategy>. Acesso em: 07/09/2023.

[24] SALES, M. *Guia WCAG*. Disponível em: <https://guia-wcag.com/>. Acesso em: 26/08/2023.

[25] MWPT. *Movimento Web Para Todos. O Movimento.* Disponível em: <https://mwpt.com.br/movimento/>. Acesso em: 26/08/2023.

[26] PREFEITURA de São Paulo lança Selo de Acessibilidade Digital com apoio do Web para Todos. MWPT, 2018. Disponível em: <https://mwpt.com.br/web-para-todos-participa-do-lancamento-do-selo-de-acessibilidade-digital/>. Acesso em: 26/08/2023.

[27] MICROSOFT. *Visual Studio Marketplace. axe Accessibility Linter*. Disponível em: <https://marketplace.visualstudio.com/items?itemName=deque-systems.vscode-axe-linter>. Acesso em: 26/08/2023.

[28] DEQUE. *Docs. axe DevTools Linter Accessibility Rules.* Disponível em: <https://docs.deque.com/linter/4.0.0/en/axe-linter-rules>. Acesso em: 26/08/2023.