

**UNIVERSIDADE DO ESTADO DE SANTA CATARINA – UDESC
CENTRO DE EDUCAÇÃO SUPERIOR DO ALTO VALE DO ITAJAÍ – CEAVI
ENGENHARIA DE SOFTWARE**

MATEUS LUCAS CRUZ BRANDT

**INCLUSÃO E TECNOLOGIA: DESENVOLVIMENTO DE UMA FERRAMENTA DE
AUXÍLIO PARA APLICAÇÕES MÓVEIS ACESSÍVEIS EM FLUTTER**

IBIRAMA

2024

MATEUS LUCAS CRUZ BRANDT

**INCLUSÃO E TECNOLOGIA: DESENVOLVIMENTO DE UMA FERRAMENTA DE
AUXÍLIO PARA APLICAÇÕES MÓVEIS ACESSÍVEIS EM FLUTTER**

Trabalho de conclusão apresentado ao curso de Engenharia de Software do Centro de Educação Superior do Alto Vale do Itajaí (CEAVI), da Universidade do Estado de Santa Catarina (UDESC), como requisito parcial para obtenção do grau de bacharel em Engenharia de Software.

Orientador: Profa. Dra. Marília Guterres Ferreira

IBIRAMA

2024

INCLUSÃO E TECNOLOGIA: DESENVOLVIMENTO DE UMA FERRAMENTA DE AUXÍLIO PARA APLICAÇÕES MÓVEIS ACESSÍVEIS EM FLUTTER / Mateus Lucas Cruz Brandt. – Ibirama, 2024.

56 p. : il.

Orientador: Profa. Dra. Marília Guterres Ferreira.

Trabalho de Conclusão de Curso (Graduação) – Universidade do Estado de Santa Catarina, Centro de Educação Superior do Alto Vale do Itajaí, Ibirama, 2024.

1. Acessibilidade. 2. Dispositivos Móveis. 3. Flutter. 4. Análise Estática. 5. Inspeção Contínua. I. , . II. , . III. Universidade do Estado de Santa Catarina, Centro de Educação Superior do Alto Vale do Itajaí. IV. Título.

MATEUS LUCAS CRUZ BRANDT

**INCLUSÃO E TECNOLOGIA: DESENVOLVIMENTO DE UMA FERRAMENTA DE
AUXÍLIO PARA APLICAÇÕES MÓVEIS ACESSÍVEIS EM FLUTTER**

Trabalho de conclusão apresentado ao curso de Engenharia de Software do Centro de Educação Superior do Alto Vale do Itajaí (CEAVI), da Universidade do Estado de Santa Catarina (UDESC), como requisito parcial para obtenção do grau de bacharel em Engenharia de Software.

Orientador: Profa. Dra. Marília Guterres Ferreira

BANCA EXAMINADORA:

Marília Guterres Ferreira, Dra.
CEAVI - UDESC

Membros:

Matheus da Hora França, Msc.
CEAVI - UDESC

Pedro Sidnei Zanchett, Msc.
CEAVI - UDESC

Ibirama, 05 de dezembro de 2024

Aos que acreditaram no meu potencial quando
eu mesmo não acreditava.

AGRADECIMENTOS

Gostaria de expressar minha profunda gratidão, em primeiro lugar, aos meus pais, que sempre foram minha base, me ensinando o valor do trabalho árduo e da dedicação.

À minha companheira, que esteve ao meu lado em todos os momentos, oferecendo apoio, incentivo e amor incondicional. Não tenho palavras suficientes para descrever o quanto sou grato por sua presença e por tudo que fez por mim, especialmente nos momentos mais difíceis.

Um agradecimento especial também ao Fedora, minha fiel companheira felina, cuja companhia trouxe leveza e observações silenciosas (mas julgadoras, como só um gato sabe fazer) durante o desenvolvimento deste trabalho.

Por fim, à minha orientadora, que desempenhou um papel essencial nesta jornada, guiando-me com paciência e incentivando-me a superar minhas próprias limitações. Seus elogios e orientações foram fundamentais para que eu acreditasse no meu potencial. Muito obrigado por tudo.

“The only disability is when people cannot see
human potential.” (DEBRA RUH)

RESUMO

Este trabalho tem como objetivo promover a acessibilidade entre desenvolvedores, conscientizando sobre a importância de tornar aplicações acessíveis, garantindo que todas as pessoas possam usufruir do que criamos. Baseando-se nas especificações de acessibilidade das principais plataformas móveis (Android e iOS), foi desenvolvido um pacote para aplicações Flutter, capaz de fornecer sugestões em tempo real sobre potenciais problemas de acessibilidade. O pacote foi implementado em Dart, a linguagem de programação do Flutter, e disponibilizado no repositório oficial de pacotes, o pub.dev. Ele foi testado em aplicações de exemplo disponíveis em seu repositório oficial, demonstrando conformidade com os requisitos de acessibilidade estabelecidos. Dessa forma, desenvolvedores de aplicativos Flutter podem incorporar esta ferramenta em seus projetos, contribuindo para a criação de aplicações mais inclusivas.

Palavras-chave: Acessibilidade. Dispositivos Móveis. Flutter. Análise Estática. Inspeção Contínua.

ABSTRACT

This work aims to promote accessibility among developers, raising awareness of the importance of creating accessible applications to ensure that everyone can enjoy what we build. Based on the accessibility specifications of major mobile platforms (Android and iOS), a package for Flutter applications was developed to provide real-time suggestions about potential accessibility issues. The package was implemented in Dart, Flutter's programming language, and made available on the official package repository, pub.dev. It was tested in example applications available in its official repository, demonstrating compliance with the established accessibility requirements. Thus, Flutter app developers can integrate this tool into their projects, contributing to the creation of more inclusive applications.

Keywords: Accessibility. Mobile Devices. Flutter. Static Analysis. Continuous Inspection.

LISTA DE ILUSTRAÇÕES

Figura 1 – Modelo de metologia	17
Figura 2 – "Maçanetas pivotantes (à esquerda) só requerem um leve empurrão por cima. Maçanetas esféricas (à direita) necessitam de um movimento de torção com firmeza."	20
Figura 3 – Tela Principal do VSCode com a extensão ValidWeb.	24
Figura 4 – Interface principal do SonarQube.	25
Figura 5 – Exemplo de retorno do pacote “accessibility_tools” ao detectar uma inconsistência de acessibilidade.	26
Figura 6 – Interação do desenvolvedor com o plugin	31
Figura 7 – Interação do plugin com as regras de acessibilidade	32
Figura 8 – Interação do plugin com a linha de comando	33
Figura 9 – Exemplo de código fonte com a regra de acessibilidade RA1 não seguida com a IDE marcando o código fonte	40
Figura 10 – Exemplo de correção automática para a regra de acessibilidade RA1	40
Figura 11 – Exemplo da execução do comando <code>dart run custom_lint</code>	41
Figura 12 – Código fonte do arquivo <code>accessibility_lint.dart</code>	42
Figura 13 – Código fonte do arquivo <code>avoid_icon_button_without_tooltip_rule.dart</code>	43
Figura 14 – Árvore de componentes de um código fonte Dart	44
Figura 15 – Código fonte do arquivo <code>avoid_icon_button_without_tooltip_fix.dart</code>	46
Figura 16 – Código fonte do arquivo <code>pubspec.yaml</code>	48
Figura 17 – Código fonte necessário no <code>analysis_options.yaml</code>	48
Figura 18 – Pacote <code>accessibility_lint</code> publicado no <code>pub.dev</code>	49

LISTA DE TABELAS

Tabela 1 – Comparação entre as funcionalidades das ferramentas discutidas e os requisitos deste trabalho.	27
Tabela 2 – Requisitos Funcionais do TCC	29
Tabela 3 – Requisitos Não Funcionais do TCC	30
Tabela 4 – Regras de negócio do TCC	30
Tabela 5 – Estrutura de um projeto Dart utilizando o template de pacote	34
Tabela 6 – Strings de busca	37
Tabela 7 – Critérios de inclusão e exclusão	37
Tabela 8 – Quantidade de estudos encontrados	38
Tabela 9 – Regras de acessibilidade baseadas nas recomendações de ambas as plataformas	38
Tabela 10 – Estrutura de diretórios e arquivos do projeto	39
Tabela 11 – Estrutura de diretórios e arquivos para a regra de acessibilidade RA1	41
Tabela 12 – Relação dos Requisitos implementados	51
Tabela 13 – Relação dos Requisitos não Funcionais implementados	51
Tabela 14 – Relação das Regras de Acessibilidade implementadas	52

LISTA DE ABREVIATURAS E SIGLAS

TCC	Trabalho de Conclusão de Curso
PCD	Pessoa com Deficiência
LBI	Lei Brasileira de Inclusão
RF	Requisitos Funcionais
RNF	Requisitos Não Funcionais
RN	Regras de Negócio
LSP	Language Server Protocol

SUMÁRIO

1	INTRODUÇÃO	14
1.1	PROBLEMA	14
1.1.1	Objetivo Geral	15
1.1.2	Objetivos Específicos	15
1.1.2.1	<i>Revisão da Literatura</i>	15
1.1.2.2	<i>Projeto e Implementação da Extensão</i>	15
1.1.2.3	<i>Documentar e Disseminar</i>	15
1.2	JUSTIFICATIVA	16
1.3	METODOLOGIA	16
1.4	ESTRUTURA DO TRABALHO	17
2	FUNDAMENTAÇÃO TEÓRICA	19
2.1	ACESSIBILIDADE	19
2.1.1	Definição de Acessibilidade	19
2.1.2	Legislação Brasileira	20
2.1.3	Acessibilidade Digital	21
2.1.4	Estado da Acessibilidade Digital	21
2.2	FLUTTER	21
2.3	ANÁLISE ESTÁTICA	22
2.3.1	Árvore Sintática Abstrata	22
2.4	MAPEAMENTO SISTEMÁTICO DA LITERATURA	23
2.5	TESTES AUTOMATIZADOS	23
3	TRABALHOS CORRELATOS	24
3.1	VALIDAWEB: UMA FERRAMENTA PARA SUPORTE AO DESENVOLVIMENTO DE SOFTWARE ATENDENDO AOS REQUISITOS NÃO FUNCIONAIS DE ACESSIBILIDADE W3C	24
3.2	SONARQUBE: FERRAMENTA DE AUDITORIA DE CÓDIGO COM SUPORTE PARA FLUTTER E DART	25
3.3	ACCESSIBILITY_TOOLS: PACOTE PARA VALIDAÇÃO DE ACESSIBILIDADE EM APLICAÇÕES FLUTTER	26
3.4	ANÁLISE E COMPARAÇÃO DE TRABALHOS CORRELATOS	27
4	DESENVOLVIMENTO	28
4.1	MAPEAMENTO INICIAL	28
4.1.1	Requisitos Funcionais	29
4.1.2	Requisitos Não Funcionais	29
4.1.3	Regras de Negócio	30

4.2	MODELAGEM	31
4.2.1	Diagrama de Sequência	31
4.2.2	Provas de Conceito	33
4.2.2.1	<i>Extensão para Visual Studio Code</i>	33
4.2.2.2	<i>Extensão para DevTools do Flutter</i>	33
4.2.2.3	<i>Plugin para Dart utilizando o pacote analyzer_plugin</i>	34
4.2.2.4	<i>Plugin para Dart utilizando o pacote custom_lint_builder</i>	35
4.3	MAPEAMENTO SISTEMÁTICO DA LITERATURA	36
4.3.1	Protocolo de Pesquisa	36
4.3.1.1	<i>Questões de Pesquisa</i>	36
4.3.1.2	<i>Seleção de Bases de Dados</i>	36
4.3.1.3	<i>Critérios de Inclusão e Exclusão</i>	37
4.3.1.4	<i>Seleção de Estudos</i>	37
4.3.1.5	<i>Síntese dos Dados</i>	37
4.4	REGRAS DE ACESSIBILIDADE	38
4.5	PROJETO	39
4.5.1	Arquitetura de Integração com o Flutter	39
4.6	DESENVOLVENDO UMA REGRA DE ACESSIBILIDADE	40
4.6.1	Estruturando o Projeto	41
4.6.2	Implementando a Regra de Acessibilidade	42
4.6.3	Implementando a Correção Automática	45
4.7	TESTES	47
4.7.1	Testes Automatizados	47
4.8	PUBLICANDO O PACOTE NO PUB.DEV	47
5	RESULTADOS E DISCUSSÃO	49
5.1	OBJETIVOS E REQUISITOS	50
5.2	REGRAS DE ACESSIBILIDADE	51
6	CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS	53
	REFERÊNCIAS	54
	GLOSSÁRIO	56

1 INTRODUÇÃO

Este Trabalho de Conclusão de Curso (TCC) aborda o desenvolvimento de uma extensão Flutter que visa auxiliar os desenvolvedores no processo de criação de aplicações móveis acessíveis em Flutter. A motivação para este trabalho surge da crescente importância da inclusão e da acessibilidade no desenvolvimento de aplicações móveis, bem como da necessidade de simplificar e melhorar o processo de desenvolvimento de soluções acessíveis utilizando tecnologias modernas, como o Flutter.

O TCC foi desenvolvido seguindo uma metodologia baseada nas fases fundamentais da Engenharia de Software, incluindo análise (Engenharia de Requisitos), projeto, programação, testes e validação. Além disso, a execução do trabalho foi gerenciada utilizando a metodologia ágil, garantindo um desenvolvimento iterativo e incremental do projeto.

A introdução detalha o contexto, a problemática, os objetivos geral e específicos, a justificativa e a metodologia adotada no desenvolvimento do trabalho. Em seguida, o TCC é organizado em capítulos que abordam a revisão da literatura, o projeto e a implementação da extensão, os testes e a validação realizados com desenvolvedores e PCDs, e o gerenciamento do projeto. Por fim, as conclusões discutem os principais resultados, limitações e possíveis trabalhos futuros relacionados ao tema da acessibilidade em aplicações móveis.

1.1 PROBLEMA

A acessibilidade tem se tornado cada vez mais relevante no contexto do desenvolvimento de aplicações móveis, uma vez que possibilita a inclusão e a igualdade de oportunidades para pessoas com deficiência (PCDs). As aplicações móveis têm desempenhado um papel essencial no cotidiano das pessoas, e é imprescindível que elas sejam projetadas de forma inclusiva, garantindo uma experiência eficiente e satisfatória para todos os usuários. Neste cenário, o presente TCC se aplica ao desenvolvimento de aplicações móveis acessíveis utilizando o framework Flutter, que tem sido amplamente adotado devido à sua capacidade de criar aplicativos de alta qualidade e com desempenho otimizado para múltiplas plataformas. Contudo, o desenvolvimento de aplicações móveis acessíveis em Flutter ainda enfrenta desafios relacionados à complexidade e à necessidade de conhecimento especializado.

O problema abordado neste TCC é a dificuldade enfrentada pelos desenvolvedores ao implementar recursos de acessibilidade em aplicações móveis criadas com Flutter, em razão das lacunas de conhecimento e da complexidade inerente a essa tarefa. A solução proposta visa simplificar e tornar mais prática a criação de aplicações móveis acessíveis, contribuindo para a melhoria da experiência dos PCDs ao utilizar essas aplicações.

Para abordar este problema, serão investigadas as melhores práticas e técnicas disponíveis na literatura relacionadas à acessibilidade em aplicações móveis. Além disso, será desenvolvida uma aplicação de exemplo que demonstre a aplicabilidade das soluções propostas, servindo como referência para os desenvolvedores Flutter.

1.1.1 Objetivo Geral

O objetivo geral deste trabalho é apoiar o desenvolvimento de aplicativos móveis mais acessíveis. Para isso, visa-se criar uma extensão Flutter que facilite o processo de desenvolvimento de aplicações móveis acessíveis em Flutter para os desenvolvedores através da análise estática de código. A extensão fornecerá orientações e recomendações sobre a utilização de componentes e a aplicação de regras necessárias para cumprir com os requisitos de acessibilidade não funcionais. Através dessa abordagem, a extensão auxiliará os desenvolvedores a criar aplicativos mais inclusivos, melhorando a experiência dos PCDs ao utilizar essas aplicações.

Além disso, este trabalho visa aumentar a conscientização sobre a importância da acessibilidade no desenvolvimento de aplicações móveis. Para isso, serão publicados os requisitos de acessibilidade utilizados durante o desenvolvimento da extensão, a fim de promover uma compreensão mais ampla das melhores práticas e normas aplicáveis a essa área.

Dessa forma, a solução proposta contribui para resolver o problema apresentado ao facilitar a criação de aplicações móveis acessíveis em Flutter, fornecendo aos desenvolvedores as ferramentas e informações necessárias para atender aos requisitos de acessibilidade e, ao mesmo tempo, promovendo uma maior conscientização sobre a importância desse tema no contexto atual.

1.1.2 Objetivos Específicos

1.1.2.1 *Revisão da Literatura*

Realizar um mapeamento sistemático da literatura para identificar os requisitos não funcionais de acessibilidade necessários para aplicações móveis, permitindo estabelecer uma base sólida para o desenvolvimento da extensão proposta e garantir a aderência às melhores práticas e normas do campo.

1.1.2.2 *Projeto e Implementação da Extensão*

Desenvolver uma extensão Flutter que analise o código-fonte de aplicações móveis e forneça orientações e recomendações sobre a utilização de componentes e a aplicação de regras de acessibilidade. A extensão deverá ser integrada ao ambiente de desenvolvimento e ser capaz de identificar possíveis problemas de acessibilidade no código, sugerindo soluções e boas práticas para corrigi-los.

1.1.2.3 *Documentar e Disseminar*

Documentar o processo de desenvolvimento da extensão, incluindo os requisitos de acessibilidade utilizados, as técnicas e as ferramentas empregadas, e os resultados obtidos. Além disso, disseminar os resultados do trabalho por meio de artigos, apresentações e publicações, a

fim de promover a conscientização sobre a importância da acessibilidade no desenvolvimento de aplicações móveis.

1.2 JUSTIFICATIVA

Este TCC foi desenvolvido considerando a crescente importância da inclusão e acessibilidade no desenvolvimento de aplicações móveis, bem como a necessidade de simplificar e otimizar o processo de desenvolvimento de soluções acessíveis utilizando Flutter. A escolha do Flutter como tecnologia central deste trabalho se justifica por ser um framework em rápido crescimento, adotado por um número cada vez maior de desenvolvedores, devido à sua capacidade de criar aplicativos de alta qualidade e com desempenho otimizado para múltiplas plataformas. Ademais, o Flutter é mantido e promovido pelo Google, o que reforça sua relevância e potencial no cenário atual de desenvolvimento de aplicações móveis.

A escolha de uma extensão de análise estática de código como solução para o problema proposto se deve à sua capacidade de identificar problemas de acessibilidade no código-fonte e fornecer orientações e recomendações para corrigi-los. A análise estática de código é uma técnica amplamente utilizada no desenvolvimento de software para identificar possíveis falhas e melhorias no código, permitindo detectar problemas de acessibilidade em um estágio inicial do desenvolvimento e facilitar a correção desses problemas.

A justificativa para o desenvolvimento deste TCC, portanto, reside na combinação da importância de promover a acessibilidade e inclusão no desenvolvimento de aplicações móveis com a crescente adoção do Flutter como tecnologia de referência para o desenvolvimento de aplicativos. Ao criar uma extensão que facilite o desenvolvimento de aplicações móveis acessíveis em Flutter, este trabalho visa contribuir para a democratização da acessibilidade e a melhoria da experiência dos PCDs ao utilizar aplicações móveis.

1.3 METODOLOGIA

A metodologia adotada neste TCC foi baseada nas fases fundamentais da Engenharia de Software, incluindo análise (Engenharia de Requisitos), projeto, programação, testes e validação. Além disso, a execução do trabalho foi gerenciada utilizando a metodologia ágil Scrum para garantir um desenvolvimento iterativo e incremental do projeto. A seguir, detalhamos cada uma das fases e ferramentas utilizadas na metodologia:

Engenharia de Requisitos: nesta fase, foi realizada um mapeamento sistemático da literatura para identificar os requisitos não funcionais de acessibilidade necessários para aplicações móveis. As principais fontes consultadas incluíram artigos científicos, livros e diretrizes de organizações especializadas em acessibilidade. As informações coletadas serviram como base para a definição dos requisitos que orientaram o desenvolvimento da extensão proposta.

Projeto: com os requisitos de acessibilidade definidos, a próxima etapa foi projetar a extensão para Flutter. Nesta fase, foram elaborados os diagramas e especificações técnicas

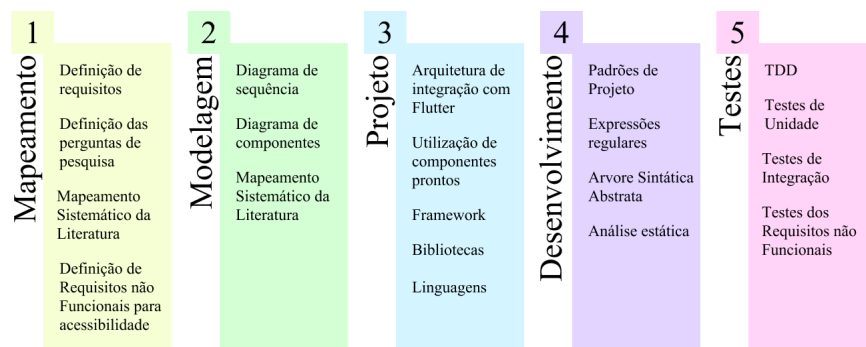
necessárias para detalhar a arquitetura e o funcionamento da extensão, incluindo a definição das funcionalidades e a interação com o ambiente de desenvolvimento.

Programação: após a conclusão do projeto, a extensão foi implementada utilizando as linguagens de programação e as ferramentas compatíveis com o Flutter, como Dart, a extensão `custom_lint_builder`. O código-fonte foi versionado e gerenciado por meio do Git e hospedado em uma plataforma de repositórios (GitHub), para facilitar a colaboração e o acompanhamento das mudanças realizadas.

Testes e validação: a extensão desenvolvida foi submetida a uma série de testes, envolvendo desenvolvedores e PCDs, a fim de avaliar sua efetividade e garantir que os requisitos de acessibilidade fossem atendidos. Os testes foram realizados em diversas etapas do desenvolvimento e incluíram testes funcionais, de usabilidade e de acessibilidade. Além disso, os requisitos não funcionais coletados na revisão da literatura foram validados por meio de testes estáticos.

Gerenciamento do projeto (Scrum): para garantir um desenvolvimento eficiente e flexível, o projeto foi gerenciado utilizando a metodologia ágil Scrum. O trabalho foi dividido em Sprints, que são períodos de tempo fixos (geralmente de 2 a 4 semanas) nos quais um conjunto específico de tarefas é realizado. As tarefas foram organizadas e priorizadas em um Product Backlog.

Figura 1 – Modelo de metodologia



Fonte: Elaborado pelo autor (2024).

A metodologia adotada permitiu um desenvolvimento estruturado e ágil do TCC, garantindo que a extensão proposta fosse construída de acordo com os requisitos de acessibilidade identificados. Além disso, a utilização do Scrum proporcionou flexibilidade e adaptabilidade durante o desenvolvimento, possibilitando ajustes e melhorias contínuas.

1.4 ESTRUTURA DO TRABALHO

O presente trabalho está organizado em capítulos que abordam os principais aspectos relacionados ao desenvolvimento da extensão proposta. A estrutura do trabalho é a seguinte:

- **Capítulo 2 - Revisão da Literatura:** apresenta uma revisão da literatura sobre acessibilidade em aplicações móveis, abordando os principais conceitos, técnicas e ferramentas relacionadas ao tema.

- Capítulo 3 - Projeto e Implementação da Extensão: descreve o projeto e a implementação da extensão proposta, detalhando a arquitetura, as funcionalidades e as tecnologias utilizadas.
- Capítulo 4 - Testes e Validação: apresenta os testes e validações realizados com desenvolvedores e PCDs para avaliar a eficácia e a usabilidade da extensão.
- Capítulo 5 - Gerenciamento do Projeto: discute o gerenciamento do projeto, incluindo a metodologia ágil Scrum, as ferramentas utilizadas e os resultados obtidos.
- Capítulo 6 - Conclusões: apresenta as conclusões do trabalho, discutindo os principais resultados, limitações e possíveis trabalhos futuros relacionados ao tema da acessibilidade em aplicações móveis.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo, são descritos e detalhados os conceitos fundamentais e as ferramentas utilizadas durante a execução deste estudo. Inicialmente, procurou-se uma compreensão mais profunda das definições de acessibilidade, acessibilidade digital e acessibilidade universal, conceitos essenciais para a compreensão do escopo deste trabalho.

Em seguida, se realiza uma discussão acerca das vantagens advindas da incorporação da acessibilidade digital no processo de desenvolvimento de aplicações destinadas a dispositivos móveis.

Destaca-se a relevância deste aspecto e o impacto positivo que pode exercer sobre a experiência do usuário final. Visando também trazer uma compreensão das definições e impactos legais que a acessibilidade digital tem para os desenvolvedores, são discutidas as principais legislações vigentes no Brasil voltadas a defesa das PCD.

Adicionalmente serão tratados quais as principais deficiências motoras, auditivas e visuais que afetam a população mundial, qual o impacto dela na vida dos portadores e quais os benefícios e relação da acessibilidade digital com a sua inclusão na sociedade.

Posteriormente, são apresentadas as tecnologias que se fizeram necessárias para a condução do desenvolvimento deste projeto. Nesta parte, dá-se ênfase à importância dos testes automatizados, procedimento que se revela fundamental para garantir a qualidade e efetividade das implementações realizadas.

Além disso, são discutidas as tecnologias específicas empregadas na elaboração de uma extensão para o Flutter que permita a análise estática de regras de acessibilidade em aplicações móveis desenvolvidas com este framework.

2.1 ACESSIBILIDADE

Conforme dados do Censo 2010 (IBGE, 2010), aproximadamente 46 milhões de brasileiros declararam ter algum grau de dificuldade em pelo menos uma das seguintes habilidades: enxergar, ouvir, caminhar ou subir degraus. Dessas, a habilidade de enxergar e ouvir são primordiais para a inclusão digital dos usuários. Portanto, se faz necessária a compreensão do conceito de acessibilidade, sua importância no mundo digital, o que pode ser implementado para melhorá-la e as respectivas questões legislativas aplicadas no Brasil.

2.1.1 Definição de Acessibilidade

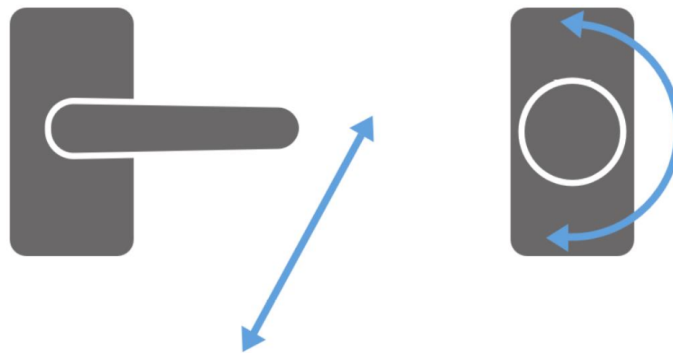
A acessibilidade é um princípio fundamental que visa garantir a todos os indivíduos, independentemente de suas limitações físicas, sensoriais, intelectuais ou psicossociais, a capacidade de acessar e utilizar adequadamente um produto, serviço ou ambiente. Neste estudo, o foco se volta à acessibilidade no âmbito digital, especialmente em relação aos dispositivos móveis.

Entretanto, é necessário que compreender que a acessibilidade é para todos, e que atos ou

soluções simples, tem um grande impacto na qualidade de vida das pessoas. Assim como define (Kalbag, 2017) “acessibilidade no mundo físico, é o nível em que um ambiente é utilizável pelo maior número de pessoas possíveis”.

Sendo assim, é simples compreender a importância e benefícios da acessibilidade no mundo físico. (Kalbag, 2017) traz um excelente exemplo de uma mudança de acessibilidade que estamos tendo em nosso mundo e acabamos por nem perceber o seu real impacto. Conforme na figura 2, é possível visualizar que maçanetas pivotantes podem ser abertas facilmente com um movimento vertical para baixo, mesmo que tenham algum grau de deficiência motora, pois podem utilizar seu corpo ou outros objetos como apoio para realizarem a abertura da porta. A princípio, esse exemplo parece não ter muita conexão com o mundo digital, porém podemos utilizá-lo como uma base para refletir a forma que construímos nossa interface, onde adicionamos complicações visando a estética, ou então apenas por decisão de produto, mas que podem ter um impacto negativo na usabilidade de PCD.

Figura 2 – "Maçanetas pivotantes (à esquerda) só requerem um leve empurrão por cima. Maçanetas esféricas (à direita) necessitam de um movimento de torção com firmeza."



Fonte: (Kalbag, 2017, p. 11)

2.1.2 Legislação Brasileira

A Lei Brasileira de Inclusão (LBI), também conhecida como Estatuto da Pessoa com Deficiência (Lei Nº 13.146, de 6 de Julho de 2015), é um marco legal que busca assegurar e promover, em condições de igualdade, o exercício dos direitos e das liberdades fundamentais por pessoa com deficiência, visando à sua inclusão social e cidadania.

A LBI aborda várias áreas importantes para a inclusão, incluindo a acessibilidade, que tem implicações diretas para a concepção e desenvolvimento de dispositivos móveis. Ela define acessibilidade como sendo a possibilidade e condição de alcance para utilização, com segurança e autonomia, de espaços, mobiliários, equipamentos urbanos, edificações, transportes, informação e comunicação, inclusive seus sistemas e tecnologias, bem como de outros serviços e instalações abertos ao público, de uso público ou privados de uso coletivo, tanto na zona urbana como na rural, por pessoa com deficiência ou com mobilidade reduzida.

Nesse contexto, a LBI define no Art. 3º inciso um que Acessibilidade é a “possibilidade e condição de alcance para utilização, com segurança e autonomia, de espaços, mobiliários, equipamentos urbanos, edificações, transportes, informação e comunicação, inclusive seus sistemas e tecnologias, bem como de outros serviços e instalações abertos ao público, de uso público ou privados de uso coletivo, tanto na zona urbana como na rural, por pessoa com deficiência ou com mobilidade reduzida”. (Brasil, 2015).

Adicionalmente, ela também define o que são tecnologias assistiva ou ajuda técnica no inciso três do Artigo 3 como sendo a “equipamentos, dispositivos, recursos, metodologias, estratégias, práticas e serviços que objetivem promover a funcionalidade, relacionada à atividade e à participação da pessoa com deficiência ou com mobilidade reduzida, visando à sua autonomia, independência, qualidade de vida e inclusão social” (Brasil, 2015).

O Capítulo II da LBI descreve quais os direitos de pessoas com deficiência. E no contexto das aplicações móveis temos no Art. 9º que a “disponibilização de recursos, tanto humanos quanto tecnológicos, que garantam atendimento em igualdade de condições com as demais pessoas.” (Brasil, 2015). Dessa forma é claro a necessidade de adequar as aplicações móveis para que não haja penalidades conforme descrito no Artigo 88 da LBI aos desenvolvedores no tangente a criação de barreiras para pessoas com deficiência.

Além da LBI, o Brasil conta com outras legislações voltadas à inclusão digital e à garantia dos direitos das pessoas com deficiência. Dentre elas, destacam-se o (Brasil, 2004), que regulamenta as Leis nº 10.048 e 10.098 e estabelece normas e critérios para a promoção de acessibilidade das pessoas com deficiência, e a Lei Nº 12.965 (Brasil, 2014), mais conhecida como Marco Civil da Internet, que estabelece princípios, garantias, direitos e deveres para o uso da Internet no Brasil.

2.1.3 Acessibilidade Digital

2.1.4 Estado da Acessibilidade Digital

A acessibilidade digital é um conceito que se refere à capacidade de pessoas com deficiência ou mobilidade reduzida de acessar e interagir com conteúdos, aplicativos e serviços digitais. Ela abrange uma ampla gama de tecnologias e práticas que visam tornar a experiência digital mais inclusiva e acessível a todos os usuários, independentemente de suas limitações.

Em uma pesquisa realizada por (adicionar iaap etc)

2.2 FLUTTER

Flutter é uma estrutura de desenvolvimento de aplicativo que usa a linguagem de programação Dart. Ela permite a criação de interfaces de usuário altamente personalizáveis e expressivas com um bom desempenho. Ao contrário de outras estruturas de desenvolvimento móvel, como o React Native, Flutter não usa pontes de JavaScript, mas compila o código diretamente no código nativo do sistema operacional, o que melhora o desempenho (Google, 2023).

Sendo assim, segundo a mantenedora o Flutter oferece melhor desempenho, capacidade de “Hot Reload” permitindo que os desenvolvedores experimentem e construam interfaces mais rapidamente, Personalização através de um rico conjunto de “Widgets” — os “blocos de construção” do Flutter — seguindo inicialmente diretrizes do Material Design para Android e Cupertino para iOS mas sem bloquear modificações do desenvolvedor e também, permite o desenvolvimento de aplicativos para diferentes plataformas partindo de uma única fonte de código.

Ademais, o Flutter oferece uma série de recursos que ajudam os desenvolvedores a criar aplicativos acessíveis. Ela suporta APIs de acessibilidade para Android e iOS, incluindo a API TalkBack para Android e a API VoiceOver para iOS. Além disso, o Flutter tem suporte para ampliação de tela, fontes maiores, contraste de cores suficiente e navegação por teclado (Google, 2023). Entretanto ainda necessitam que o desenvolvedor utilize das ferramentas providas para criar aplicações realmente acessíveis.

2.3 ANÁLISE ESTÁTICA

Análise Estática segundo (Nystrom, 2021) é uma técnica de verificação de programas que examina o código-fonte sem executá-lo, identificando possíveis erros, inconsistências e violações de regras de programação. Ela é amplamente utilizada em compiladores, ferramentas de análise de código e linters para garantir a correção e a qualidade do código.

Essa etapa no processamento de código fonte é usualmente utilizado na compilação ou interpretação de um código, entretanto, ferramentas de análise estática podem ser utilizadas para verificar a conformidade do código com padrões de codificação, identificar possíveis vulnerabilidades de segurança e otimizar a performance do código durante o processo de desenvolvimento.

Essa técnica será o pilar para a construção da extensão proposta neste trabalho, pois será responsável por analisar o código fonte de aplicações móveis desenvolvidas com Flutter e identificar possíveis violações de regras de acessibilidade e boas práticas de desenvolvimento em tempo real para auxiliar o desenvolvedor.

2.3.1 Árvore Sintática Abstrata

A construção de uma Árvore Sintática Abstrata (AST, do inglês Abstract Syntax Tree) é uma etapa fundamental no processo de compilação, sendo responsável por representar a estrutura de um programa de forma hierárquica e abstrata. De acordo com (Ullman, 2008), a AST organiza as construções da linguagem em nós que correspondem aos elementos essenciais do programa, eliminando detalhes sintáticos desnecessários, como pontuação e regras intermediárias presentes na gramática. Assim, a AST fornece uma visão condensada do programa, capturando sua lógica e estrutura de forma que facilite etapas subsequentes, como a análise semântica e a geração de código.

Com base nesses princípios, a AST e a análise estática são amplamente utilizadas não apenas em compiladores tradicionais, mas também em ferramentas modernas, como linters e otimizadores de código. Essas ferramentas aproveitam a representação abstrata fornecida pela AST para realizar diagnósticos e aplicar melhorias, garantindo que o código esteja em conformidade com padrões estabelecidos e otimizado para execução.

Com uma AST é possível fazer a análise necessária para validar regras de acessibilidade em qualquer linguagem. Para o presente trabalho, a AST será utilizada para analisar o código fonte de aplicações móveis desenvolvidas com Flutter e identificar possíveis violações de regras de acessibilidade e boas práticas de desenvolvimento móvel.

2.4 MAPEAMENTO SISTEMÁTICO DA LITERATURA

O Mapeamento Sistemático da Literatura (MSL) é uma técnica de pesquisa que visa identificar, avaliar e interpretar as evidências disponíveis em um determinado campo de estudo.

2.5 TESTES AUTOMATIZADOS

Adicionar conteúdo, e importancia

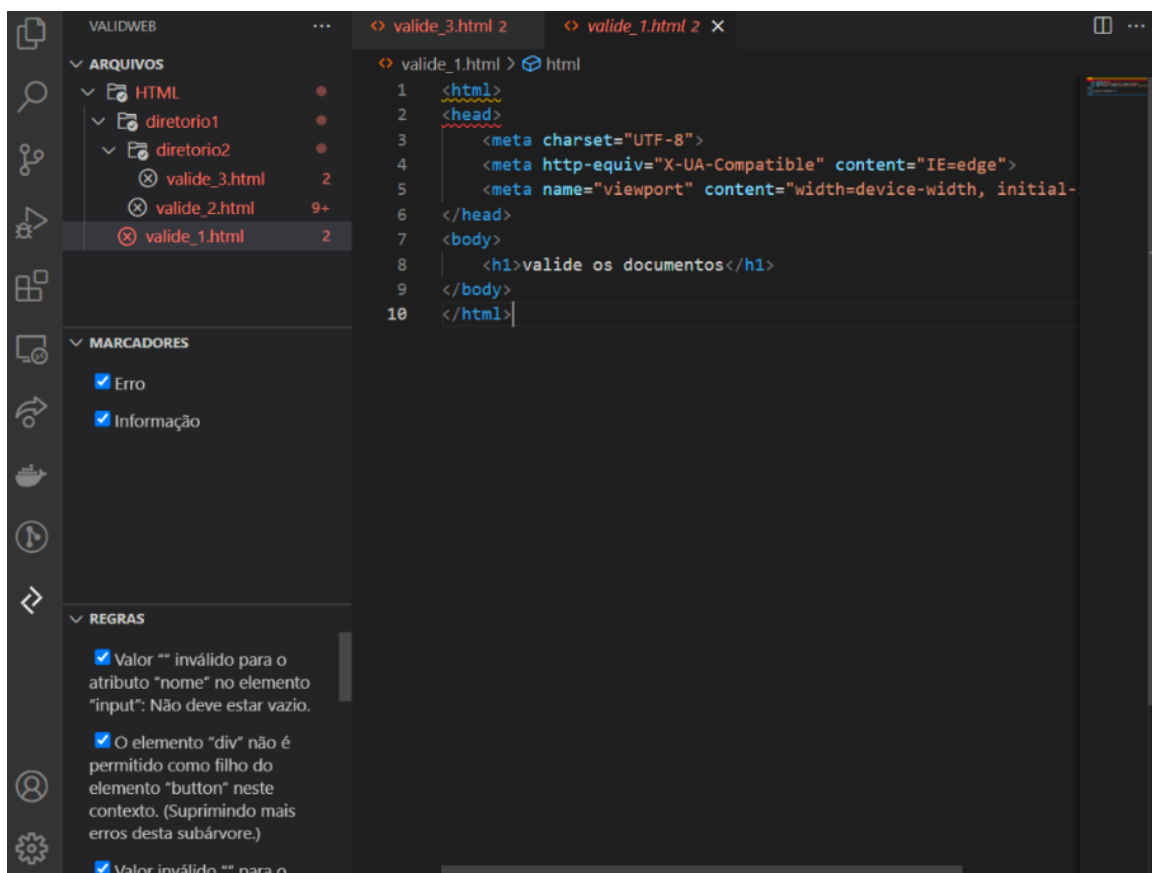
3 TRABALHOS CORRELATOS

O objetivo desta seção é realizar uma análise comparativa de aplicações que apresentam um contexto similar ao do presente TCC. Para isso, faremos uma descrição sucinta de cada trabalho correlato, seguida de uma comparação dos requisitos que são comumente oferecidos por todas essas aplicações. A intenção é explorar as similaridades e as diferenças entre elas para aprimorar nosso entendimento e guiar o desenvolvimento do nosso projeto.

3.1 VALIDAWEB: UMA FERRAMENTA PARA SUPORTE AO DESENVOLVIMENTO DE SOFTWARE ATENDENDO AOS REQUISITOS NÃO FUNCIONAIS DE ACESSIBILIDADE W3C

O ValidaWeb (Mohr, 2022), é uma ferramenta voltada ao suporte no desenvolvimento de software voltados para a Web, que valida estruturas HTML e CSS com o objetivo de retornar sugestões sobre possíveis melhorias voltadas a acessibilidade. Tem como base os requisitos da W3C para definição de quais as propriedades são necessárias para uma boa acessibilidade do software que está sendo desenvolvido.

Figura 3 – Tela Principal do VSCode com a extensão ValidWeb.



Fonte: (Mohr, 2022, p. 59)

A interface proposta pelo ValidWeb é intuitiva e prática. Por meio de destaques no texto,

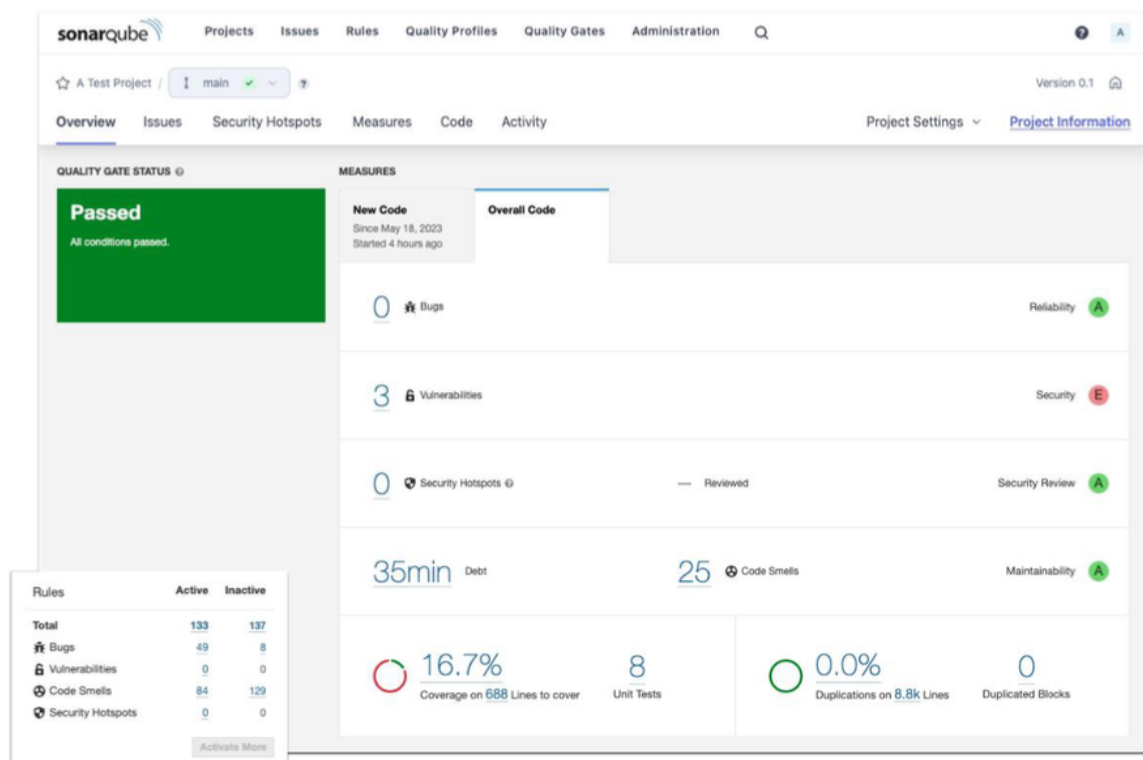
os desenvolvedores são notificados sobre inconsistências nas especificações de acessibilidade, que devem ser resolvidas para que esses destaques sejam removidos. Além disso, a ferramenta de (Mohr, 2022) também permite a geração de relatórios em PDF do estado atual do código, facilitando sua análise e compartilhamento com outros desenvolvedores, especialmente em ambientes de desenvolvimento que envolvem múltiplas equipes com controle de qualidade rigoroso.

Dessa forma, o ValidWeb e o presente TCC compartilham objetivos similares, embora tenham focos tecnológicos distintos.

3.2 SONARQUBE: FERRAMENTA DE AUDITORIA DE CÓDIGO COM SUPORTE PARA FLUTTER E DART

O SonarQube, conforme descrito pela (SonarSource, 2023), é uma ferramenta robusta de análise estática de código. Seu objetivo é simplificar a padronização de projetos para desenvolvedores. A ferramenta dispõe de uma interface gráfica intuitiva que destaca os principais pontos de atenção na qualidade estática do código. Dentre os indicadores apresentados estão: número de bugs, vulnerabilidades, "Security Hotspots", dívida técnica, "Code Smells", cobertura de testes e linhas de código duplicadas, conforme ilustrado na Figura 4.

Figura 4 – Interface principal do SonarQube.



Fonte: (Grousset, 2023)

Por padrão, a ferramenta não oferece suporte para a linguagem Dart e o framework Flutter. No entanto, graças ao esforço da comunidade, especificamente da InsideApp (Grousset, 2023),

uma empresa focada no desenvolvimento de aplicações móveis, foi criado um plugin que permite a interoperabilidade básica entre Flutter, Dart e SonarQube. Este plugin aproveita as regras padrões estabelecidas pelo Flutter para realizar a análise e fornecer sugestões de melhorias.

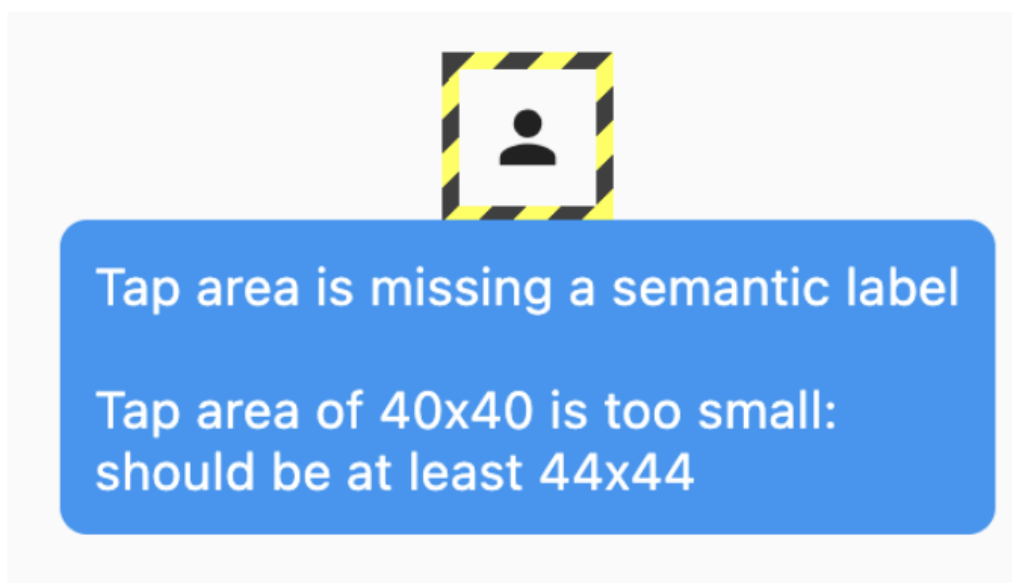
Ainda que o foco principal do SonarQube não seja a acessibilidade, a ferramenta, em conjunto com o SonarLint - também desenvolvido pela SonarSource (SonarSource, 2023) - oferece sugestões ao desenvolvedor durante o processo de codificação na IDE. Esta funcionalidade se assemelha à proposta do presente TCC, demonstrando a potencialidade de uso dessas ferramentas em conjunto para melhorar tanto a qualidade do código quanto sua acessibilidade.

3.3 ACCESSIBILITY_TOOLS: PACOTE PARA VALIDAÇÃO DE ACESSIBILIDADE EM APLICAÇÕES FLUTTER

Conforme descrito pela Rebel AppStudio, mantenedora do pacote, o accessibility_tools é uma suíte de ferramentas destinadas a melhorar a acessibilidade em aplicações desenvolvidas com Flutter. (Troshkov, 2023) destaca a importância da acessibilidade em aplicativos, ressaltando que "Criar um aplicativo acessível é extremamente importante. No entanto, é comum esquecer ou adiar essas melhorias. Este pacote garante que seu aplicativo seja acessível desde o primeiro dia, verificando a interface logo que é construída".

A Figura 5 exemplifica o funcionamento da ferramenta durante a fase de desenvolvimento de uma aplicação. A ferramenta opera em tempo de execução, enfatizando possíveis melhorias de acessibilidade por meio de mensagens exibidas diretamente na interface.

Figura 5 – Exemplo de retorno do pacote “accessibility_tools” ao detectar uma inconsistência de acessibilidade.



Fonte: (Troshkov, 2023)

Até agosto de 2023, o pacote suporta a validação de áreas mínimas de toque, semântica de botões, sobreposição de textos e descrição de campos de entrada. Embora a ferramenta

cubra apenas alguns aspectos de acessibilidade, ela representa um ponto de partida valioso para desenvolvedores Flutter interessados em melhorar a acessibilidade de suas aplicações.

A abordagem adotada no presente TCC difere em aspectos importantes. Em vez de operar em tempo de execução, o foco está na análise estática de código para fornecer sugestões diretamente na IDE. Isso elimina a necessidade de compilar e executar a aplicação para avaliar as necessidades de acessibilidade, permitindo que os desenvolvedores identifiquem e resolvam problemas de acessibilidade mais rapidamente.

3.4 ANÁLISE E COMPARAÇÃO DE TRABALHOS CORRELATOS

O universo de ferramentas disponíveis para auxiliar no desenvolvimento de aplicações acessíveis é vasto e diversificado. No entanto, as ferramentas discutidas neste capítulo foram escolhidas por sua relevância e semelhanças com a proposta deste TCC. Cada uma delas utiliza um conjunto de regras ou requisitos não funcionais de acessibilidade para estabelecer padrões e oferecer sugestões de modificações, dependendo do estado da aplicação em desenvolvimento. A identificação desses requisitos será uma etapa crucial na elaboração da proposta deste TCC.

Na Tabela 1 a seguir, é feita uma comparação entre as funcionalidades das ferramentas discutidas e os requisitos deste trabalho. Esta tabela ajuda a identificar áreas para melhoria na proposta deste TCC e também destaca as deficiências das outras ferramentas em comparação com a proposta atual.

Tabela 1 – Comparação entre as funcionalidades das ferramentas discutidas e os requisitos deste trabalho.

Funcionalidade	ValidaWeb	SonarQube	accessibility tools	Proposta Atual
Inspeção contínua	✓	✓	✗	✓
Análise estática de código	✓	✓	✗	✓
Acessibilidade em Flutter	✓	✗	✓	✓
Sugestão de correção	✗	✗	✗	✓
Disponível em IDE	✓	✓	✗	✓
Suporte para Flutter	✗	✗	✓	✓

Fonte: Elaborado pelo autor (2024).

A análise da Tabela 1 revela que, embora quase todas as funcionalidades sejam atendidas por uma ou mais das ferramentas, nenhuma delas oferece todas as funcionalidades em um único pacote. Notavelmente, a funcionalidade de sugestão de correção é uma característica distintiva da proposta atual, potencialmente agilizando o processo de desenvolvimento ao fornecer soluções rápidas para os desenvolvedores assim que identificarem a necessidade de melhorar a acessibilidade de suas aplicações.

4 DESENVOLVIMENTO

Esta seção tem como objetivo apresentar o processo de desenvolvimento da proposta atual, fundamentado nas etapas previamente descritas na seção de metodologia. Serão discutidos também os desafios encontrados durante o processo e possíveis ajustes na direção do projeto à medida que obstáculos emergem.

O processo de desenvolvimento da proposta inicia com a etapa de Mapeamento, na qual serão definidos os requisitos, formuladas as perguntas de pesquisa, realizado o mapeamento sistemático da literatura e estabelecidos os requisitos não funcionais para acessibilidade.

Em seguida, na etapa de Modelagem, serão construídos o Diagrama de Sequência, o Diagrama de Componentes, o Modelo Lógico de Dados e o Diagrama de Classes. Esses diagramas e modelos servirão como alicerce para a próxima fase.

Na fase do Projeto, será projetada a arquitetura de integração com o Flutter e selecionados os frameworks, bibliotecas e linguagens que serão usados no desenvolvimento da ferramenta.

Com a conclusão da etapa de Projeto, o processo de Desenvolvimento será iniciado, onde serão aplicados Padrões de Projeto e Expressões regulares, além de preparar a ferramenta para publicação.

Após a conclusão do desenvolvimento, a etapa de Testes será iniciada. Nela, será adotado o TDD, serão realizados Testes de Unidade e de Integração, bem como Testes dos Requisitos não funcionais de Acessibilidade.

Finalmente, na etapa de Validação, será realizada uma pesquisa com desenvolvedores e pessoas com deficiência, aplicado o SUS e executada uma prova de conceito. Essa etapa final permitirá avaliar a eficácia da ferramenta desenvolvida e identificar áreas de melhoria potencial. Cada uma dessas etapas será discutida em detalhes nas seções subsequentes deste capítulo.

4.1 MAPEAMENTO INICIAL

O processo inicial de mapeamento é fundamental para a abordagem de identificação dos Requisitos Funcionais (RF), Requisitos Não Funcionais (RNF) e Regras de Negócio (RN).

Os Requisitos Funcionais (RF) são destinados a descrever as funcionalidades que o sistema deve incorporar para atender às necessidades do usuário final. Estes requisitos são essencialmente as ações que o sistema deve ser capaz de realizar, como por exemplo, processar entradas de dados, realizar operações e fornecer saídas de informações.

Por outro lado, os Requisitos Não Funcionais (RNF) definem o conjunto de critérios que orientam o modo como o sistema operará, como as restrições e propriedades desejáveis do sistema que não estão diretamente relacionadas com o comportamento específico dele. Eles estabelecem um escopo com características particulares que o sistema a ser desenvolvido deverá seguir, tais como padrões de projeto, usabilidade, segurança, desempenho, entre outros. Em outras palavras, os RNF são os atributos de qualidade do sistema que determinam como os requisitos funcionais devem ser implementados.

As Regras de Negócio (RN), por sua vez, são diretrizes que definem ou restringem algum aspecto do negócio. Elas descrevem os detalhes das funcionalidades que o sistema deve possuir, para que estas possam ser implementadas de forma a não causar prejuízos a outras funcionalidades. Elas fornecem uma compreensão clara de como o sistema deve se comportar em determinadas circunstâncias e auxiliam na garantia de que o sistema esteja alinhado com as metas e estratégias do negócio.

Em suma, a identificação e compreensão desses três elementos - RF, RNF e RN - são cruciais para o sucesso do desenvolvimento do sistema, garantindo que ele seja construído de acordo com as necessidades do negócio e dos usuários, além de cumprir com os critérios de qualidade estabelecidos.

4.1.1 Requisitos Funcionais

Os Requisitos Funcionais (RF) identificados para o sistema proposto são:

Tabela 2 – Requisitos Funcionais do TCC

Requisito	Descrição
RF1	O sistema deve permitir a visualização de inconsistências no código
RF2	O sistema deve realizar marcações no código baseado na especificação definida
RF3	O sistema deve sugerir remover trechos de código baseado na especificação definida
RF4	O sistema deve sugerir correções automáticas baseado na especificação definida
RF5	O sistema deve permitir o usuário consultar todas as baseado os requisitos não funcionais de acessibilidade
RF6	O sistema deve permitir o usuário desabilitar requisitos não funcionais de acessibilidade
RF7	O sistema deve permitir a utilização em ambientes de integração contínua

Fonte: Elaborado pelo autor (2023).

Dos requisitos funcionais da tabela 2, destaca-se a necessidade de analisar código escrito na linguagem Dart, isso depende de uma ferramenta capaz de processar e interpretar o código fonte da aplicação desenvolvida com Flutter. A equipe do Dart provê uma API para análise estática de código, que será utilizada através do plugin analyzer. Ele já provê uma interface para acessar a árvore sintática abstrata do código fonte, que será utilizada para identificar e marcar as inconsistências no código.

4.1.2 Requisitos Não Funcionais

Os Requisitos Não Funcionais (RNF) identificados para o sistema proposto são:

Tabela 3 – Requisitos Não Funcionais do TCC

Requisito	Descrição
RNF1	O sistema deve ser escrito utilizando a linguagem Dart
RNF2	O sistema deve ser publicado no repositório pub.dev
RNF3	O sistema deve seguir as melhores práticas para pacotes publicados no repositório pub.dev
RNF4	O sistema deve possuir licença aberta para permitir alterações e melhorias
RNF5	O sistema deve possuir documentação para auxiliar na criação de novas regras de negócio de usabilidade

Fonte: Elaborado pelo autor (2024).

Dos requisitos funcionais descritos na tabela 3, destaca-se a necessidade de seguir as melhores práticas para pacotes publicados no repositório pub.dev. Isso é importante para garantir que o sistema seja facilmente acessível e utilizável por outros desenvolvedores, além de manter a qualidade e a segurança do código fonte. A equipe do Dart provê um guia de boas práticas para publicação de pacotes no repositório pub.dev, que será seguido para garantir a qualidade e a segurança do sistema. Ele também é uma das formas que o repositório classifica plugins auxiliando na escolha dos usuários.

4.1.3 Regras de Negócio

As Regras de Negócio (RN) identificadas para o sistema proposto são:

Tabela 4 – Regras de negócio do TCC

Regra de Negócio	Descrição
RN1	As regras de acessibilidade devem ser separadas por pastas para organização clara e simplificada
RN2	Cada regra de acessibilidade deve possuir testes automatizados para validar seu funcionamento
RN3	Cada regra de acessibilidade deve possuir uma descrição clara e objetiva para facilitar a compreensão do desenvolvedor
RN4	Cada regra de acessibilidade deve possuir uma sugestão de correção
RN5	Cada regra de acessibilidade deve possuir uma marcação no código para indicar a inconsistência
RN6	As marcações devem ser na cor Laranja, uma vez que a cor vermelha é utilizada para erros de compilação

Fonte: Elaborado pelo autor (2024).

Além das regras de negócio listadas na tabela 4, destaca-se a necessidade de separar as regras de acessibilidade por pastas para organização clara e simplificada. A ideia inicial era utili-

zar um Mapeamento Sistemático da Literatura para identificar as regras de acessibilidade mais utilizadas e criar um conjunto de regras de acessibilidade padrão. Porém, conforme será descrito nos próximos capítulos, não foi possível levantar um conjunto de regras de acessibilidade padrão, então optou-se por criar um conjunto de regras de acessibilidade baseado nas recomendações de ambas as plataformas.

4.2 MODELAGEM

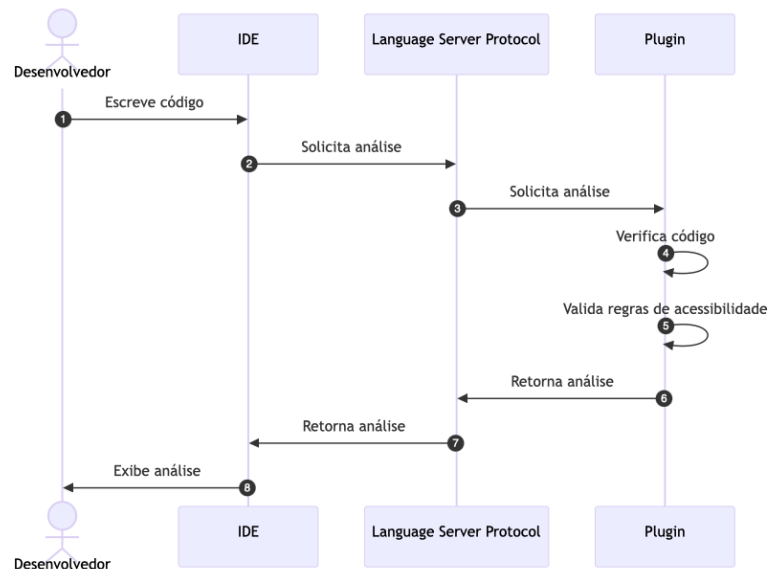
Com os requisitos identificados, a próxima etapa é a Modelagem, onde serão construídos os diagramas e modelos que servirão como base para o desenvolvimento do sistema.

4.2.1 Diagrama de Sequência

Diagramas de sequência são diagramas de interação que mostram como grupos de objetos colaboram em algum comportamento ao longo do tempo. Eles são usados para capturar o comportamento de um único caso de uso, ou seja, um cenário específico de interação entre objetos.

O diagrama de sequência da figura 6 a seguir ilustra a interação básica do desenvolvedor com o plugin:

Figura 6 – Interação do desenvolvedor com o plugin



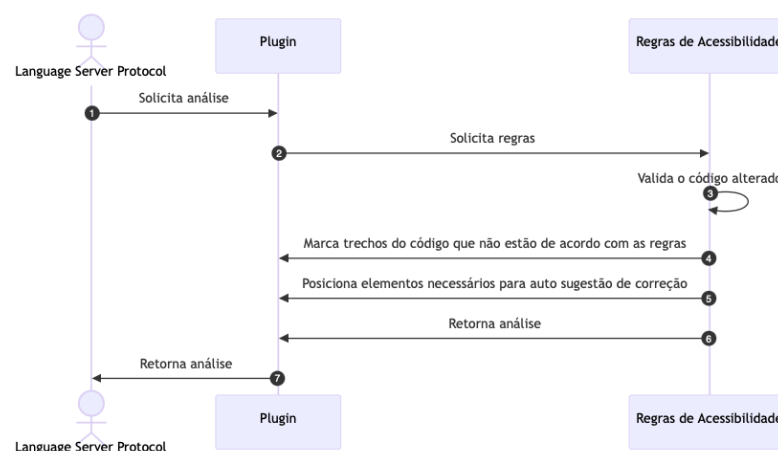
Fonte: Elaborado pelo autor (2024).

Como podemos ver na figura 6, o desenvolvedor interage com o plugin através da IDE, que por sua vez executa a análise estática do código fonte da aplicação desenvolvida com

Flutter. O plugin então marca o código fonte baseado nas regras de acessibilidade definidas e sugere correções automáticas. Isso torna todo o processo de desenvolvimento mais eficiente e permite que regras de acessibilidade sejam aplicadas desde o início do desenvolvimento sem a necessidade de compilar e executar a aplicação.

Entretanto o diagrama da figura 6 não mostra a interação do plugin com as regras de acessibilidade, para isso foi criado o diagrama de sequência da figura 7:

Figura 7 – Interação do plugin com as regras de acessibilidade



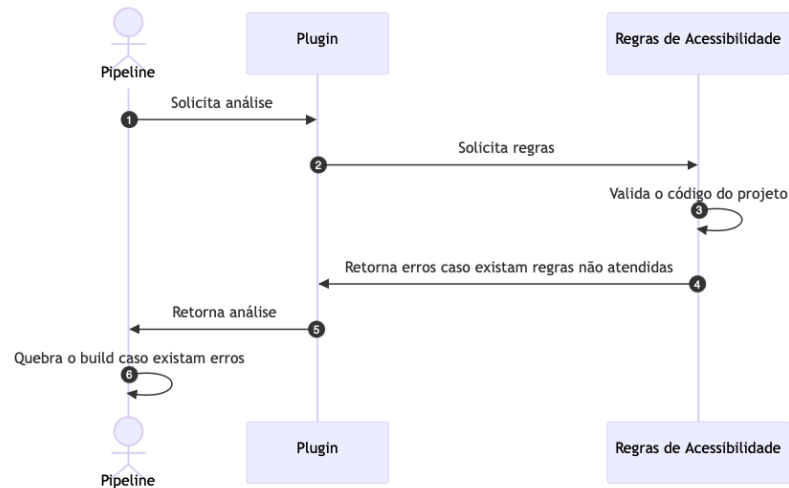
Fonte: Elaborado pelo autor (2024).

Como é possível identificar na figura 7, o ponto de partida é o LSP que envia o código fonte do desenvolvedor para o Plugin, que então irá processar o código fonte aplicando todas as regras de acessibilidade definidas de uma forma sequencial. Cada regra de acessibilidade é aplicada ao código fonte e, caso uma inconsistência seja encontrada, o plugin marca o código fonte e sugere correções automáticas.

Um dos requisitos é ser possível utilizar o plugin através de uma linha de comando possibilitando integração com ambientes de integração contínua, para isso foi criado o diagrama de sequência da figura 8:

O principal objetivo de integrar o plugin em ambientes de integração contínua é garantir que as regras de acessibilidade sejam aplicadas em todas as fases do desenvolvimento, desde a escrita do código até a compilação e execução da aplicação. Isso permite que os desenvolvedores travem a publicação de aplicações que não atendam aos requisitos de acessibilidade, garantindo que todas as aplicações desenvolvidas com Flutter sejam acessíveis a todos os usuários.

Figura 8 – Interação do plugin com a linha de comando



Fonte: Elaborado pelo autor (2024).

4.2.2 Provas de Conceito

4.2.2.1 Extensão para Visual Studio Code

Na concepção do projeto, a ideia era criar uma extensão para Visual Studio Code capaz de realizar a análise estática de código Dart e Flutter e retornar ao usuário as regras de acessibilidade que não foram seguidas. Porém, durante o desenvolvimento, percebeu-se que seria necessário desenvolver toda uma infraestrutura de análise estática de código, o que demandaria muito tempo e recursos. Além disso, a equipe do Dart já provê uma API para análise estática de código, que será utilizada através do plugin analyzer.

Ademais, uma implementação focada em Visual Studio Code limitaria o alcance do projeto, uma vez que a extensão seria específica para essa IDE. Por isso, optou-se por desenvolver um plugin para LSP, que é uma interface de comunicação entre a IDE e o plugin, permitindo que o plugin seja utilizado em qualquer IDE que suporte LSP. Porém, ainda assim esbarra-se na necessidade de desenvolver uma infraestrutura de análise estática de código, o que demandaria muito tempo e recursos.

4.2.2.2 Extensão para DevTools do Flutter

DevTools é uma ferramenta de diagnóstico para Flutter que permite visualizar e inspecionar o comportamento da aplicação em execução. A ideia era criar uma extensão para DevTools

capaz de realizar a análise estática de código Dart e Flutter e retornar ao usuário as regras de acessibilidade que não foram seguidas. Tal implementação seria agnóstica de IDE e uma vez que desenvolvida em Dart, poderia utilizar o pacote analyzer para realizar a análise estática de código e processar o código fonte da aplicação desenvolvida com Flutter.

Porém, tal abordagem não seria capaz de marcar o código fonte da aplicação desenvolvida com Flutter, uma vez que DevTools é uma ferramenta de diagnóstico e não uma IDE. Sendo assim, tal ideia foi descartada após a realização de alguns testes.

4.2.2.3 *Plugin para Dart utilizando o pacote analyzer_plugin*

Haja vista que o projeto visava criar um plugin agnóstico de IDE porém ainda capaz de se comunicar com LSP tornando possível a marcação, sugestão de correção e validação de regras de acessibilidade, foi decidido utilizar o pacote analyzer_plugin para criar um plugin para Dart. O pacote analyzer_plugin é uma API para criar plugins para o Dart Analysis Server, que é uma ferramenta de análise estática de código Dart.

Seguindo a documentação oficial do pacote, foi então iniciado o desenvolvimento de uma prova de conceito. Para isso, foi então criado um novo projeto em Dart utilizando o template de pacote. Esse template estrutura um projeto Dart com uma base ideal para o desenvolvimento de um pacote que possa ser posteriormente publicado no repositório pub.dev.

Para tal é necessário executar o seguinte comando: `dart create -t package-simple <nome_do_projeto>` que após a execução, irá gerar uma estrutura similar a apresentada na tabela 5.

Tabela 5 – Estrutura de um projeto Dart utilizando o template de pacote

```

<nome_do_projeto>
|- CHANGELOG.md
|- README.md
|- analysis_options.yaml
|- example/
|-- <nome_do_projeto>_example.dart
|- lib/
|-- <nome_do_projeto>.dart
|-- src/
--- <nome_do_projeto>_base.dart
|- pubspec.lock
|- pubspec.yaml
|- test/
-- <nome_do_projeto>_test.dart

```

Fonte: Elaborado pelo autor (2024).

Com o projeto base criado, é então necessário adicionar as dependências necessárias para o desenvolvimento do plugin. Para isso, foi necessário executar o seguinte comando: `dart pub`

add analyzer_plugin analyzer que irá adicionar a dependência do pacote analyzer_plugin e do pacote analyzer ao arquivo pubspec.yaml. Ambos são pacotes oficiais do time do Dart.

Seguindo então a documentação do (Dart, 2024) foi possível chegar em uma implementação capaz de avaliar código fonte escrito em Dart e retornar as regras de acessibilidade que não foram seguidas. Entretanto a falta de suporte da equipe para com o pacote analyzer_plugin fez com que a implementação fosse abandonada.

As principais dificuldades encontradas foram:

- Falta de documentação: A documentação oficial do pacote analyzer_plugin é escassa e não cobre todos os aspectos necessários para o desenvolvimento de um plugin para Dart.
- Falta de suporte: A equipe do Dart não fornece suporte para o pacote analyzer_plugin, o que torna difícil encontrar soluções para problemas específicos.
- Complexidade: O pacote analyzer_plugin é complexo e requer um conhecimento avançado de Dart e análise estática de código para ser utilizado corretamente.
- Falta de integração: O pacote não é capaz de integrar com o analisador padrão do Dart, o que dificulta a implementação de regras de acessibilidade personalizadas. Necessitando que o desenvolvedor que consuma o pacote tenha de executar uma série de comandos para que o plugin funcione corretamente.

4.2.2.4 *Plugin para Dart utilizando o pacote custom_lint_builder*

Após a tentativa frustrada de utilizar o pacote analyzer_plugin, foi então decidido utilizar o pacote custom_lint_builder para criar um plugin para Dart. Apesar de ele necessitar que o usuário final importe o pacote custom_lint no seu projeto, utilize arquivos diferentes para a personalização de regras de acessibilidade, e também necessitar de uma interface de linha de comando própria, sendo incompatível com o comando dart analyze do Dart, ele é atualmente a melhor opção para a criação de um plugin "linter" para Dart. Porém ele ainda possui um problema que não é simples de resolver sem envolver o time de desenvolvimento do Dart, ele não é capaz de integrar com comandos como dart analyze e dart format. Entretanto, não existe alternativa viável para a criação de um plugin "linter" para Dart que atenda esse requisito.

Tal pacote se mostrou mais simples de utilizar e mais bem documentado que o pacote analyzer_plugin. Além disso, ele é capaz de integrar com o analisador padrão do Dart, o que facilita a implementação de regras de acessibilidade personalizadas.

Para executar em um ambiente de integrado contínua, basta executar o comando dart run custom_lint que irá analisar o código fonte do projeto e retornar as regras de acessibilidade que não foram seguidas, com um comportamento similar ao comando dart analyze do Dart.

4.3 MAPEAMENTO SISTEMÁTICO DA LITERATURA

Com a conclusão da etapa de Modelagem e provas de conceito, a próxima etapa é o Mapeamento Sistemático da Literatura, onde serão identificadas as regras de acessibilidade mais utilizadas e criado um conjunto de regras de acessibilidade padrão.

4.3.1 Protocolo de Pesquisa

Como protocolo de pesquisa para a realização do Mapeamento Sistemático da Literatura, foi utilizado o descrito no apêndice 1 do relatório técnico de (Travassos, 2005). A partir dele, as etapas serão condensadas em: Definição do Problema, Questões de Pesquisa, Seleção de Bases de Dados, Critérios de Inclusão e Exclusão, Seleção de Estudos e Síntese dos Dados.

4.3.1.1 *Questões de Pesquisa*

- QP1: Quais são as regras de acessibilidade mais utilizadas em móveis?
- QP2: Existe um conjunto de regras de acessibilidade padrão para aplicações móveis?
- QP3: Como as regras de acessibilidade são aplicadas em aplicações móveis?
- QP4: Quais são as ferramentas mais utilizadas para aplicar regras de acessibilidade em aplicações móveis?

4.3.1.2 *Seleção de Bases de Dados*

Para o levantamento de dados de uma forma mais automatizada através da utilização de ferramentas de busca com suporte a consultas estruturadas, foi decidido utilizar as seguintes bases de dados:

- IEEE Xplore
- ACM Digital Library
- Google Scholar

Como *string* de busca após algumas iterações para melhor ajuste, foi definida a presente na tabela 6. Ela foi utilizada para realizar a busca nas bases de dados selecionadas.

Foi utilizado a língua inglesa para a busca, uma vez que a maioria dos artigos científicos estão escritos nessa língua. Além disso, foi definido o período de 2010 a 2023 para a busca, uma vez que a acessibilidade em aplicações móveis é um tema relativamente novo e a maioria dos artigos científicos sobre o assunto foram publicados nesse período.

Tabela 6 – Strings de busca

```
("mobile development"OR "mobile app*") AND ("accessibility"OR
"accessible design"OR "inclusive design"OR "digital inclusion") AND
("tool*"OR "guidelines"OR "legislation"OR "legal"OR
"requirements"OR "compliance"OR "challenges"OR "demand") AND
NOT("W3C"OR "web")
```

Fonte: Elaborado pelo autor (2023).

4.3.1.3 Critérios de Inclusão e Exclusão

Os critérios de inclusão e exclusão foram definidos para garantir que apenas os artigos relevantes para o tema da pesquisa fossem selecionados. Os critérios de inclusão e exclusão são apresentados na tabela 7.

Tabela 7 – Critérios de inclusão e exclusão

Critérios de Inclusão
Artigos que abordam o tema da acessibilidade em aplicações móveis
Artigos publicados entre 2010 e 2023
Artigos escritos em inglês
Artigos disponíveis gratuitamente
Regras de acessibilidade aplicáveis em aplicações móveis
Critérios de Exclusão
Artigos que não abordam o tema da acessibilidade em aplicações móveis
Artigos publicados antes de 2010 ou após 2023
Artigos escritos em outro idioma que não o inglês
Artigos pagos
Regras de acessibilidade não aplicáveis em aplicações móveis

Fonte: Elaborado pelo autor (2023).

4.3.1.4 Seleção de Estudos

Com as bases de dados selecionadas e os critérios de inclusão e exclusão definidos, foi possível realizar a busca e selecionar os estudos relevantes para a pesquisa. A busca foi realizada utilizando a string de busca definida anteriormente na tabela 6 e os critérios de inclusão e exclusão definidos na tabela 7.

Os resultados da busca obtidos estão destacados na tabela 8.

4.3.1.5 Síntese dos Dados

Talvez devido a string de busca utilizada, não foi possível encontrar estudos relevantes para a pesquisa. Alguns artigos como o de (Gama, 2021) foram encontrados e pelo seu resumo

Tabela 8 – Quantidade de estudos encontrados

Base de Dados	Total de Estudos	Estudos Incluídos	Estudos Excluídos	Estudos Selecionados
IEEE Xplore	247	17	230	0
ACM Digital Library	32	12	20	0
Google Scholar	17700	7	17693	0

Fonte: Elaborado pelo autor (2023).

pareciam ser relevantes para a pesquisa, porém não estavam disponíveis gratuitamente. Esse foi na realidade um dos maiores desafios encontrados durante a realização do Mapeamento Sistemático da Literatura, a falta de acesso a artigos científicos relevantes para a pesquisa.

Com isso, foi necessário uma mudança na abordagem do projeto, uma vez que não foi possível identificar um conjunto de regras de acessibilidade padrão para aplicações móveis. A nova abordagem consiste em criar um conjunto de regras de acessibilidade baseado nas recomendações de ambas as plataformas.

4.4 REGRAS DE ACESSIBILIDADE

Seguindo a documentação oficial da Apple (Apple, 2024) e do Google (Google, 2024), foi possível identificar um conjunto de regras de acessibilidade que são comuns a ambas as plataformas. Essas regras de acessibilidade serão utilizadas como base para a criação de um conjunto de regras de acessibilidade padrão para aplicações móveis desenvolvidas com Flutter. Podemos visualizar ela na tabela 9.

Tabela 9 – Regras de acessibilidade baseadas nas recomendações de ambas as plataformas

Regra de Acessibilidade	Descrição
RA1	Utilize rótulos descritivos para todos os elementos interativos
RA2	Forneça descrições de texto alternativas para imagens
RA3	Forneça feedback visual, auditivo ou tátil para todas as interações do usuário
RA4	Elementos interativos devem possuir um mínimo de 48pt de largura e altura
RA5	Forneça um contraste mínimo de 4.5:1 entre o texto e o plano de fundo

Fonte: Elaborado pelo autor (2024).

Tais regras de acessibilidade serão utilizadas como base para a criação de um conjunto de regras de acessibilidade padrão para aplicações móveis desenvolvidas com Flutter. Elas serão aplicadas ao código fonte da aplicação desenvolvida com Flutter e o plugin irá marcar o código fonte e sugerir correções automáticas caso uma inconsistência seja encontrada.

4.5 PROJETO

Com a conclusão da etapa de Mapeamento Sistemático da Literatura, a próxima etapa é o Projeto, onde será projetada a arquitetura de integração com o Flutter e selecionados os frameworks, bibliotecas e linguagens que serão usados no desenvolvimento da ferramenta.

4.5.1 Arquitetura de Integração com o Flutter

A arquitetura de integração com o Flutter foi projetada para ser simples e eficiente, permitindo que o plugin seja utilizado em qualquer IDE que suporte LSP. A arquitetura é composta por três componentes principais: o LSP, o Plugin e o Dart Analysis Server.

Como descrito na seção de Provas de Conceito, foi escolhido o pacote `custom_lint_builder` para criar um plugin para Dart.

Seguindo sua documentação (Invertase, 2024), foi possível criar uma estrutura de diretórios e arquivos que permite a criação de regras de acessibilidade personalizadas e a integração com o analisador padrão do Dart. Ela está disposta na tabela 10.

Uma regra de lint utilizando o pacote usualmente necessita de uma classe que estende `LintRule` e sobrescreve o método `register` para registrar a regra de acessibilidade. Ademais, podemos definir regras de correção automática criando uma nova classe que estende `DartFix`. Mais detalhes sobre como implementar serão descritos na próxima seção com um exemplo prático com uma das regras de acessibilidade implementadas.

Tabela 10 – Estrutura de diretórios e arquivos do projeto

<projeto>
- analysis_options.yaml
- lib/
-- <nome_do_projeto>.dart
-- rules/
--- <nome_da_regra>/
---- <nome_da_regra>_rule.dart
---- <nome_da_regra>_fix.dart
- pubspec.yaml

Fonte: Elaborado pelo autor (2024).

Com a estrutura da tabela 10, é possível escalar de forma simples e eficiente a criação de regras de acessibilidade personalizadas e a integração com o analisador do pacote `custom_lint`. Pode ser resumido a criação de uma nova pasta dentro de `rules` com o nome da regra de acessibilidade.

O principal objetivo é permitir que outros desenvolvedores possam ampliar o conjunto de regras de acessibilidade padrão para aplicações móveis desenvolvidas com Flutter, garantindo que todas as aplicações desenvolvidas com Flutter sejam acessíveis a todos os usuários.

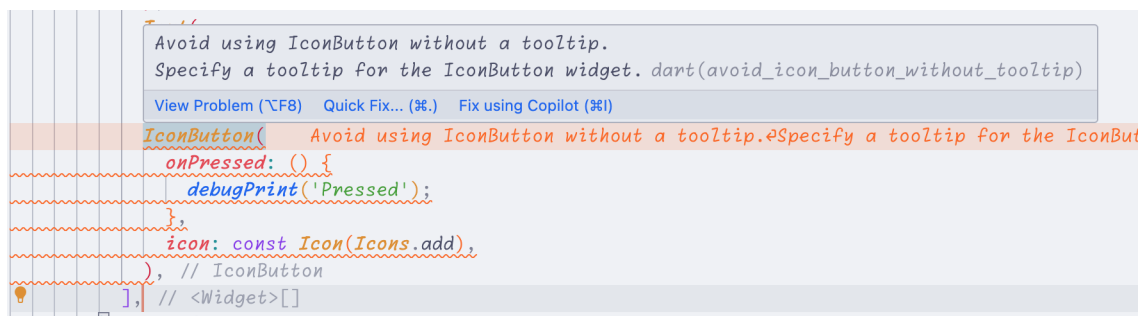
4.6 DESENVOLVENDO UMA REGRA DE ACESSIBILIDADE

Para exemplificar como desenvolver uma regra de acessibilidade utilizando o pacote `custom_lint`, foi escolhida a regra de acessibilidade RA1: Utilize rótulos descritivos para todos os elementos interativos. Essa regra de acessibilidade é comum a ambas as plataformas e é uma das regras de acessibilidade mais utilizadas em aplicações móveis.

No Flutter, seguindo sua documentação oficial (Google, 2023) podemos notar que em elementos como o `IconButton` é possível adicionar um `tooltip` que é exibido quando o usuário passa o mouse sobre o elemento. Esse `tooltip` é um rótulo descritivo que descreve a função do elemento interativo.

O objetivo final está disposto na figura 9 onde é possível visualizar que a IDE marca o código fonte e sugere correções automáticas quando um `IconButton` não possui um `tooltip`.

Figura 9 – Exemplo de código fonte com a regra de acessibilidade RA1 não seguida com a IDE marcando o código fonte



Fonte: Elaborado pelo autor (2024).

Também podemos visualizar na figura 10 que a IDE sugere correções automáticas para a regra de acessibilidade RA1. E ao aceitar a correção automática, o `tooltip` é adicionado ao `IconButton` bastando o desenvolvedor adicionar o texto da descrição do elemento interativo.

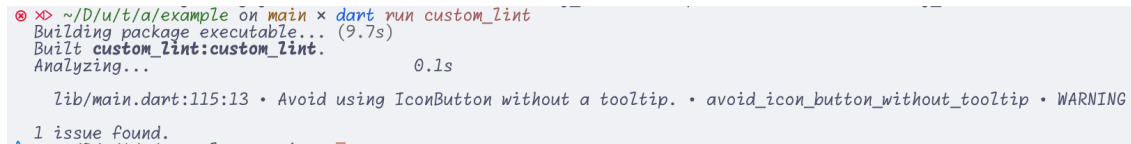
Figura 10 – Exemplo de correção automática para a regra de acessibilidade RA1



Fonte: Elaborado pelo autor (2024).

Também visando a possibilidade de utilização em ambientes de integração contínua, na figura 11 é possível visualizar a execução do comando `dart run custom_lint` que analisa o código fonte do projeto e retorna as regras de acessibilidade que não foram seguidas.

Figura 11 – Exemplo da execução do comando `dart run custom_lint`



```

~/D/u/t/a/example on main x dart run custom_lint
Building package executable... (9.7s)
Built custom_lint:custom_lint.
Analyzing... 0.1s

lib/main.dart:115:13 • Avoid using IconButton without a tooltip. • avoid_icon_button_without_tooltip • WARNING
1 issue found.
  
```

Fonte: Elaborado pelo autor (2024).

Dessa forma é possível garantir que sempre que um desenvolvedor adicione um `IconButton` em sua aplicação desenvolvida com Flutter, ele seja obrigado a adicionar um tooltip que descreva a função do elemento interativo, tornando a aplicação mais acessível a todos os usuários.

4.6.1 Estruturando o Projeto

Para implementar a regra de acessibilidade RA1: Utilize rótulos descritivos para todos os elementos interativos, foi criada a seguinte estrutura de diretórios e arquivos:

Tabela 11 – Estrutura de diretórios e arquivos para a regra de acessibilidade RA1

accessibility_lint
- analysis_options.yaml
- lib/
-- accessibility_lint.dart
-- rules/
--- avoid_icon_button_without_tooltip_rule/
---- avoid_icon_button_without_tooltip_rule_rule.dart
---- avoid_icon_button_without_tooltip_rule_fix.dart
- pubspec.yaml

Fonte: Elaborado pelo autor (2024).

Com a estrutura da tabela 11 pronta, podemos partir para a implementação de tal regra. Com a estrutura definida, precisamos alterar o arquivo inicial do projeto `accessibility_lint.dart` para registrar a regra de acessibilidade e verificar se um `IconButton` possui um tooltip. A figura 12 apresenta o código fonte.

Para criar um plugin para o pacote `custom_lint`, é necessário criar uma classe que estende `PluginBase` e sobrescrever o método `getLintRules` para retornar uma lista de regras de acessibilidade. Na linha 5 da figura 12, é definido um método `createPlugin` que retorna uma instância da classe `_AccessibilityLintPlugin`. Na linha 9 sobrescrevemos o método `getLintRules` para retornar uma lista de regras de acessibilidade, que no caso é a regra de acessibilidade `AvoidIconButtonWithoutTooltipRule` que será implementada na próxima seção.

Figura 12 – Código fonte do arquivo accessibility_lint.dart

```

1 import 'package:custom_lint_builder/custom_lint_builder.dart';
2
3 import 'src/rules/avoid_icon_button_without_tooltip/....dart';
4
5 PluginBase createPlugin() => _AccessibilityLintPlugin();
6
7 class _AccessibilityLintPlugin extends PluginBase {
8   @override
9   List<LintRule> getLintRules(
10     final CustomLintConfigs configs,
11   ) =>
12     <LintRule>[
13       const AvoidIconButtonWithoutTooltipRule(),
14     ];
15 }

```

Fonte: Elaborado pelo autor (2024).

4.6.2 Implementando a Regra de Acessibilidade

Para implementar a regra de acessibilidade, iremos primeiro alterar o arquivo `avoid_icon_button_without_tooltip_rule_rule.dart` para registrar a regra de acessibilidade e verificar se um `IconButton` possui um `tooltip`. A figura 13 apresenta o código fonte. Em seguida será destacado cada passo da implementação para melhor compreensão.

Para criar uma regra de lint utilizando o pacote `custom_lint`, é criar uma estender a classe `DartLintRule`, assim como é possível visualizar na linha 7 da figura 13. Também é necessário sobrescrever o método `run` para registrar a regra de acessibilidade e tornar visível para o pacote.

Nas linhas 10 a 14 da figura 13, é definido os textos que irão retornar ao usuário caso a regra de acessibilidade não seja seguida. Ela é um objeto da class `LintCode` que recebe o nome da regra, a mensagem de problema e a mensagem de correção. Nela definimos na linha 11 que o nome da regra é "avoid_icon_button_without_tooltip", na linha 12 a mensagem de problema é "Avoid using IconButton without a tooltip.", na linha 13 a mensagem de correção é "Specify a tooltip for the IconButton widget." e por fim na linha 14 definimos o nível de severidade do erro como `WARNING` pois a regra de acessibilidade não é um erro de compilação.

Com essa definição básica especificada, agora precisamos efetivamente construir o código que irá fazer a validação de tal regra (Invertase, 2024). Usualmente, aplicações em Flutter, possuem um esquema de definição declarativa de interface no qual as propriedades de um "Widget" são passadas como argumentos para o construtor do mesmo.

Para verificar se um `IconButton` possui um `tooltip`, é necessário verificar se o construtor do `IconButton` possui um argumento chamado `tooltip`. Para isso, é necessário percorrer a lista de argumentos do construtor e verificar se existe um argumento chamado `tooltip`. Uma

Figura 13 – Código fonte do arquivo avoid_icon_button_without_tooltip_rule.dart

```

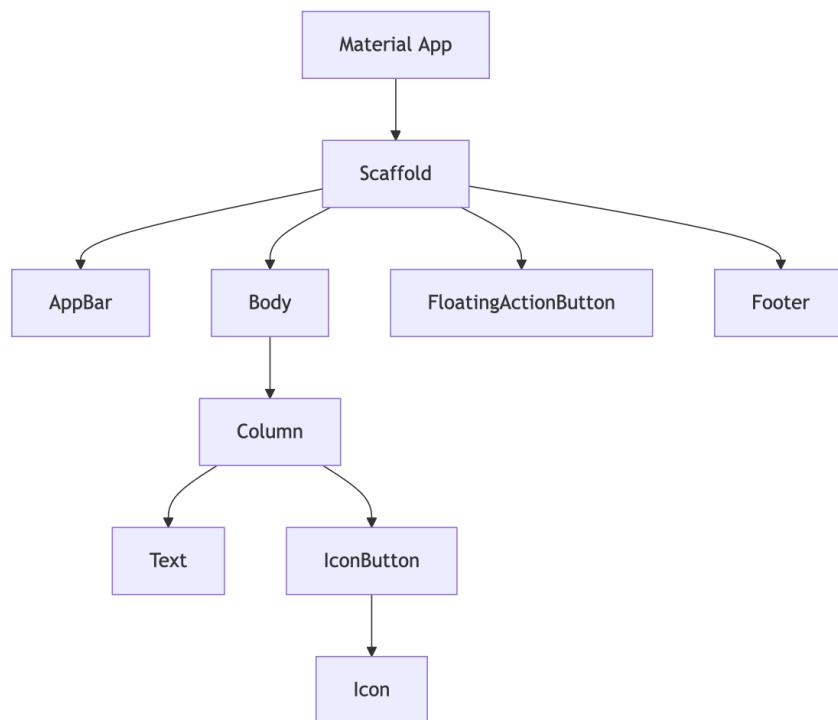
1 import 'package:analyzer_dart/ast/ast.dart';
2 import 'package:analyzer/error/listener.dart';
3 import 'package:custom_lint_builder/custom_lint_builder.dart';
4
5 import 'avoid_icon_button_without_tooltip_fix.dart';
6
7 class AvoidIconButtonWithoutTooltipRule extends DartLintRule {
8   const AvoidIconButtonWithoutTooltipRule() : super(code: _code);
9
10  static const LintCode _code = LintCode(
11    name: 'avoid_icon_button_without_tooltip',
12    problemMessage: 'Avoid using IconButton without a tooltip.',
13    correctionMessage: 'Specify a tooltip for the IconButton widget.',
14    errorSeverity: ErrorSeverity.WARNING,
15  );
16
17  @override
18  void run(
19    final CustomLintResolver resolver,
20    final ErrorReporter reporter,
21    final CustomLintContext context,
22  ) {
23    context.registry.addInstanceCreationExpression((
24      final InstanceCreationExpression node,
25    ) {
26      final String constructorName =
27        node.constructorName.type.toString();
28
29      if (constructorName != 'IconButton') return;
30
31      bool hasTooltip = false;
32      for (final Expression argument in node.argumentList.arguments) {
33        if (argument is! NamedExpression) continue;
34        final String name = argument.name.label.name;
35
36        if (name != 'tooltip') continue;
37
38        hasTooltip = true;
39        break;
40      }
41
42      if (hasTooltip) return;
43
44      reporter.atNode(node, _code);
45    });
46  }
47
48  @override
49  List<Fix> getFixes() => <Fix>[AvoidIconButtonWithoutTooltipFix()];

```

vez que o pacote `custom_lint` não fornece uma maneira de acessar o código fonte do construtor, é necessário utilizar a API do pacote `analyzer`. Ela fornece uma árvore de sintaxe abstrata (Ullman, 2008) que representa o código fonte do arquivo analisado e permite acessar os elementos do código fonte, como construtores, argumentos e propriedades.

O primeiro passo é realizado na linha 25, onde definimos uma variável `constructorName` que recebe o nome do construtor do nó da árvore que está sendo visitado naquele momento. Para facilitar a compreensão de como a árvore é construída, é possível visualizar a figura 14.

Figura 14 – Árvore de componentes de um código fonte Dart



Fonte: Elaborado pelo autor (2024).

Uma vez que estamos apenas interessados no construtor do `IconButton`, é necessário verificar se o nome do construtor é igual a `IconButton`. Isso é feito na linha 27, onde verificamos se o nome do construtor é diferente de `IconButton`. Caso não seja, a regra de acessibilidade não é aplicada e a execução já retorna na mesma linha.

Com a certeza de que o nó é um `IconButton`, agora é necessário verificar se ele já possui o atributo `tooltip` conforme definido na documentação do Flutter (Google, 2023). Para isso, é necessário percorrer a lista de argumentos do construtor e verificar se o argumento está sendo utilizado. Com isso, na linha 30 é iniciado um laço de repetição que percorre a lista de argumentos do construtor. Tais argumentos possuem o tipo `Expression` e é necessário verificar se ele é um `NamedExpression` que é um argumento nomeado. Caso não seja, a execução retorna na linha 31. Se o nome desse argumento nomeado não for `tooltip`, a execução retorna na linha 33. Caso contrário, a variável `hasTooltip` é definida como verdadeira e a execução do laço de

repetição se encerra.

Haja vista que a regra de acessibilidade é aplicada apenas quando um `IconButton` não possui um `tooltip`, é necessário verificar o estado da variável `hasTooltip`. Em caso de estar definida como verdadeira, a execução retorna na linha 35. Caso contrário, a regra de acessibilidade é aplicada e a mensagem de problema é reportada na linha 37.

Com isso, temos a primeira regra de acessibilidade implementada. A próxima etapa é implementar a regra de correção automática que irá sugerir a adição de um `tooltip` para o `IconButton`.

4.6.3 Implementando a Correção Automática

Para implementar a regra de correção automática, iremos alterar o arquivo `avoid_icon_button_without_tooltip_rule_fix.dart` para sugerir a adição de um `tooltip` para o `IconButton`. A figura 15 apresenta o código fonte.

Assim como na figura 13, a etapa inicial, seguindo a documentação (Invertase, 2024) é estender uma classe do pacote utilizado na criação do plugin. Nesse caso, é a classe `DartFix` que é utilizada para implementar regras de correção automática. Na linha 1 da figura 15, é definida a classe `AvoidIconButtonWithoutTooltipFix` que estende a classe `DartFix`. O único método que precisa ser implementado é o método `run` que é chamado quando a regra de acessibilidade é violada.

A etapa inicial, como definida na linha 12, é verificar se o erro de análise está contido no nó da árvore que está sendo visitado. Caso não esteja, a execução retorna na linha 13. Em seguida, é verificado se o nó é um `IconButton` na linha 16. Caso não seja, a execução retorna na linha 17.

Uma vez que o nó é um `IconButton`, é necessário verificar se o último token do nó é uma vírgula. Isso é necessário para garantir que a correção automática seja aplicada corretamente. Caso o último token do nó seja uma vírgula, a variável `haveCommaBeforeEndToken` é definida como verdadeira na linha 21. Caso contrário, a variável é definida como falsa na linha 19.

Com a verificação concluída, é possível sugerir a correção automática para adicionar um `tooltip` para o `IconButton`. Para isso, é necessário criar um novo `ChangeBuilder` na linha 25 que irá adicionar um novo `DartFileEdit` na linha 26. O `DartFileEdit` é responsável por adicionar a correção automática ao código fonte. Como podemos ver na documentação do (Invertase, 2024), nele é necessário informar o offset (posição) onde a correção automática será aplicada e o texto que será inserido. Uma vez que o `tooltip` é um argumento nomeado, podemos adicionar ele no final do nó, porém, atrás do último token do nó que é o `)` utilizado para fechar o construtor. Vamos utilizar a variável `haveCommaBeforeEndToken` para verificar se é necessário adicionar uma vírgula antes do `tooltip`. Caso seja necessário, a vírgula é adicionada na linha 37. Caso contrário, a vírgula é adicionada na linha 39. Em seguida, o `tooltip` é adicionado na linha 36 e a correção automática é aplicada.

Para finalizar, basta agora utilizar o builder para posicionar o cursor da IDE na posição

Figura 15 – Código fonte do arquivo avoid_icon_button_without_tooltip_fix.dart

```

1 class AvoidIconButtonWithoutTooltipFix extends DartFix {
2   @override
3   void run(
4     final CustomLintResolver resolver,
5     final ChangeReporter reporter,
6     final CustomLintContext context,
7     final AnalysisError analysisError,
8     final List<AnalysisError> others,
9   ) {
10    context.registry.addInstanceCreationExpression(
11      (final InstanceCreationExpression node) {
12        if (!analysisError.sourceRange.intersects(node.sourceRange)) {
13          return;
14        }
15
16        final String constructorName =
17          node.constructorName.type.toString();
18        if (constructorName != 'IconButton') return;
19
20        bool haveCommaBeforeEndToken = false;
21        if (node.endToken.previous?.lexeme == ',') {
22          haveCommaBeforeEndToken = true;
23        }
24
25        reporter
26          .createChangeBuilder(
27            message: 'Add tooltip property',
28            priority: 0,
29          ).addDartFileEdit((
30            final DartFileEditBuilder builder,
31          ) {
32            final int offset = node.endToken.offset - 1;
33
34            builder.addInsertion(
35              offset,
36              (final DartEditBuilder builder) {
37                if (haveCommaBeforeEndToken) {
38                  builder.write(" tooltip: ");
39                } else {
40                  builder.write(", tooltip: ");
41                }
42
43                builder..selectHere()..writeln("'",");
44              },
45            );
46          },
47        );
48      );
49    }
50  }

```

necessária para ele apenas escrever a descrição, e então adicionar ao final o ' para fechar a string e a vírgula para fechar o argumento nomeado.

Com isso, a regra de acessibilidade RA1 foi implementada com sucesso. A próxima etapa é testar a regra de acessibilidade em uma aplicação desenvolvida com Flutter e verificar se a correção automática é sugerida quando um `IconButton` não possui um `tooltip`.

4.7 TESTES

Para testar a regra de acessibilidade RA1, foi criada uma aplicação de exemplo que possui um `IconButton` sem um `tooltip`. A figura 9 apresenta o código fonte da aplicação de exemplo. Ela já foi apresentada anteriormente na seção de Provas de Conceito. Também na figura 10 é possível visualizar a correção automática sugerida pela IDE. E por último, para validar a possibilidade de utilização em ambientes de integração contínua, na figura 11 é possível visualizar a execução do comando `dartruncustom_lint` que analisa o código fonte do projeto e retorna as regras de acessibilidade que não foram seguidas.

4.7.1 Testes Automatizados

Por se tratar de um pacote com funcionalidades diferente do usual, a equipe desenvolvedora do pacote `custom_lint` fornece uma técnica de testes automatizados conforme em sua documentação (Invertase, 2024) através de comentários em trechos de código com a anotação `@lint...`

4.8 PUBLICANDO O PACOTE NO PUB.DEV

Para tornar possível que outros desenvolvedores utilizem o pacote, podemos publicá-lo no `pub.dev`. Seguindo a documentação oficial (Team, 2024), é necessário criar uma conta no `pub.dev` como publicador de pacotes. Com a conta criada, no projeto criado basta executar o comando `dartpubpublish` para publicar o pacote. As informações do pacote como nome, versão, descrição, etc, são definidas no arquivo `pubspec.yaml`. Na figura 16 é possível visualizar o arquivo `pubspec.yaml` do pacote criado.

Com o pacote publicado, ele estará disponível para ser utilizado por outros desenvolvedores através do comando `flutter pub add -dev custom_lint accessibility_lint`. O comando irá adicionar o pacote ao arquivo `pubspec.yaml` do projeto e instalar o pacote em modo de desenvolvimento no projeto que o desenvolvedor adicionar. Ademais, o desenvolvedor precisará adicionar a configuração do pacote no arquivo `analysis_options.yaml` do projeto. Na figura 17 é possível visualizar o arquivo `analysis_options.yaml` do projeto.

Figura 16 – Código fonte do arquivo pubspec.yaml

```

1 name: accessibility_lint
2 description: This package provides a handful of lint rules to help and
   guide you through your development of mobile applications.
3 version: 0.0.1
4 repository: https://github.com/MateuxLucax/accessibility-lint
5 issue_tracker: https://github.com/MateuxLucax/accessibility-lint/issues
6
7 environment:
8   sdk: ^3.5.0
9
10 platforms:
11   android:
12   ios:
13
14 dependencies:
15   analyzer: ^6.7.0
16   custom_lint_builder: ^0.7.0
17   analyzer_plugin: ^0.11.3
18
19 dev_dependencies:
20   lints: ^5.0.0

```

Fonte: Elaborado pelo autor (2024).

Figura 17 – Código fonte necessário no analysis_options.yaml

```

1 analyzer:
2   plugins:
3     - custom_lint

```

Fonte: Elaborado pelo autor (2024).

Na figura 18 é possível visualizar o pacote `accessibility_lint` publicado no `pub.dev`. Nele, o desenvolvedor pode ter uma visão geral do projeto, e também instruções e exemplos de utilização. Ademais, é possível visualizar os caminhos para o repositório público do código hospedado no GitHub e para o rastreador de problemas. Com isso, é possível que outros desenvolvedores proporcionem feedbacks e sugestões para melhorar o pacote.

5 RESULTADOS E DISCUSSÃO

Este capítulo visa apresentar os resultados obtidos durante a execução do projeto. O objetivo geral está presente na Seção 1.1.1, os objetivos específicos estão presentes na Seção 1.1.2, a metodologia utilizada está presente na Seção 1.3.

Em suma, o projeto tem por objetivo a criação de um pacote capaz de auxiliar desenvolvedores de aplicativos móveis utilizando o framework Flutter em criar aplicações mais acessíveis. Para isso, foi desenvolvido linter capaz de analisar o código-fonte de uma aplicação Flutter e identificar possíveis problemas de acessibilidade. O linter foi desenvolvido utilizando a linguagem de programação Dart e o pacote custom_lint_builder.

O mesmo, já está publicado no repositório do pub.dev e pode ser utilizado por qualquer desenvolvedor Flutter que busca criar aplicações mais acessíveis. Na figura 18 é possível visualizar o mesmo, publicado.

Figura 18 – Pacote accessibility_lint publicado no pub.dev

accessibility_lint 0.0.2

Published 6 hours ago Dart 3 compatible

SDK DART FLUTTER PLATFORM ANDROID IOS

Readme

Changelog

Example

Installing

Versions

Scores

Admin

Activity log

0

150

0%

LIKES

PUB POINTS

POPULARITY

Publisher

unverified uploader

Metadata

This package provides a handful of lint rules to help and guide you through your development of mobile applications.

[Repository \(GitHub\)](#)
[View/report issues](#)

Documentation

[API reference](#)

License

[Apache-2.0 \(license\)](#)

Dependencies

[analyzer](#), [analyzer_plugin](#), [custom_lint_builder](#)

More

[Packages that depend on accessibility_lint](#)

Accessibility Lint

license

Apache 2.0

404 badge not found

404 badge not found

ANDROID

IOS

A static analysis linter that helps developers create accessible Dart and Flutter applications by enforcing best practices for accessibility.

Features

- Avoid Icon Button Without Tooltip: Ensures all `IconButton` widgets have a `tooltip` property.

Installation

To add the accessibility linter to your project, run:

```
flutter pub add --dev custom_lint accessibility_lint
```

Then, add in your `analysis_options.yaml` file:

```
analyzer:  
  plugins:  
    - custom_lint
```

Usage

To run the accessibility linter, run:

```
dart run custom_lint
```

License

This project is licensed under the Apache License 2.0 License - see the [LICENSE](#) file for details.

Fonte: Elaborado pelo autor (2024).

A apresentação de sugestões de correção para os problemas identificados pelo `linter` é feita diretamente na sua IDE de desenvolvimento, através de mensagens de erro e avisos. A Figura 9 apresenta um exemplo de mensagem de erro gerada pelo `linter`.

5.1 OBJETIVOS E REQUISITOS

O Objetivo Geral disposto na seção 1.1.1 foi alcançado, uma vez que o pacote `accessibility_lint` foi desenvolvido e publicado no repositório do `pub.dev` e pode ser utilizado por qualquer desenvolvedor Flutter que busca criar aplicações mais acessíveis.

Referente aos objetivos específicos dispostos na seção 1.1.2, o primeiro objetivo específico 1.1.2.1 foi alcançado, através do Mapeamento Sistemático da Literatura descrito na seção 4.3. Entretanto não chegou-se no objetivo esperado de construir uma relação de requisitos de acessibilidade para as plataformas Android e iOS. Para contornar, através da documentação de ambas as plataformas (Apple, 2024), (Google, 2024), foi possível levantar alguns requisitos de acessibilidade dispostos na seção 4.4 como Regras de Acessibilidade. O segundo objetivo específico 1.1.2.2 foi concluído com a implementação conforme disposta no capítulo 4. E o terceiro objetivo específico 1.1.2.3 teve seu objetivo alcançado uma vez que com o pacote publicado, e toda a pesquisa realizada, o conhecimento foi disseminado.

Agora será feita uma revisão dos Requisitos Funcionais (seção 4.1.1), Requisitos não Funcionais (seção 4.1.2) e Regras de Negócio (seção 4.1.3).

A tabela 12 relaciona os requisitos funcionais implementados no desenvolvimento. Nela é possível verificar que quase todos os Requisitos Funcionais propostos foram implementados, com exceção do requisito RF5 que não foi implementado. O mesmo se deu por conta da dificuldade em encontrar uma fonte confiável de Requisitos de Acessibilidade para as plataformas Android e iOS conforme disposto na seção 4.3.

A tabela 13 relaciona os requisitos não funcionais implementados no desenvolvimento. Nela é possível verificar que todos os Requisitos não Funcionais propostos foram implementados. Destaca-se que o pacote foi publicado no repositório do `pub.dev` e segue as melhores práticas para pacotes publicados no repositório `pub.dev`. Com isso, todo e qualquer desenvolvedor Flutter pode utilizar o pacote para criar aplicações mais acessíveis.

Tabela 12 – Relação dos Requisitos implementados

	Requisito	Implementado
RF1 - O sistema deve permitir a visualização de inconsistências no código		✓
RF2 - O sistema deve realizar marcações no código baseado na especificação definida		✓
RF3 - O sistema deve sugerir remover trechos de código baseado na especificação definida		✓
RF4 - O sistema deve sugerir correções automáticas baseado na especificação definida		✓
RF5 - O sistema deve permitir o usuário consultar todas as baseado os requisitos não funcionais de acessibilidade		✗
RF6 - O sistema deve permitir o usuário desabilitar requisitos não funcionais de acessibilidade		✓
RF7 - O sistema deve permitir a utilização em ambientes de integração contínua		✓

Fonte: Elaborado pelo autor (2024).

Tabela 13 – Relação dos Requisitos não Funcionais implementados

	Requisito não Funcional	Implementado
RNF1 - O sistema deve ser escrito utilizando a linguagem Dart		✓
RNF2 - O sistema deve ser publicado no repositório pub.dev		✓
RNF3 - O sistema deve seguir as melhores práticas para pacotes publicados no repositório pub.dev		✓
RNF4 - O sistema deve possuir licença aberta para permitir alterações e melhorias		✓
RNF5 - O sistema deve possuir documentação para auxiliar na criação de novas regras de negócio de usabilidade		✓

Fonte: Elaborado pelo autor (2024).

5.2 REGRAS DE ACESSIBILIDADE

Tendo em vista a não execução do Requisito Funcional RF5, não foi possível construir uma relação de Requisitos de Acessibilidade para as plataformas Android e iOS. Entretanto, foi possível levantar algumas Regras de Acessibilidade para o desenvolvimento do pacote `accessibility_lint`. Os mesmos estão dispostos na tabela 9 e para facilitar visualização da implementação, a tabela 14 relaciona as Regras de Acessibilidade com as implementações realizadas.

A regra de acessibilidade RA5 não foi implementada por conta da dificuldade em encontrar uma forma de verificar o contraste entre o texto e o plano de fundo. Uma vez que elementos podem utilizar diferentes fontes para sua cor de fundo e ou texto.

Ao final, temos como resultado a implementação de 4 Regras de Acessibilidade dispostas em diferentes implementações ao longo do projeto, uma vez que as mesmas foram sub divididas

Tabela 14 – Relação das Regras de Acessibilidade implementadas

Regra de Acessibilidade	Implementado
RA1 - Utilize rótulos descritivos para todos os elementos interativos	✓
RA2 - Forneça descrições de texto alternativas para imagens	✓
RA3 - Forneça feedback visual, auditivo ou tátil para todas as interações do usuário	✓
RA4 - Elementos interativos devem possuir um mínimo de 48pt de largura e altura	✓
RA5 - Forneça um contraste mínimo de 4.5:1 entre o texto e o plano de fundo	✗

Fonte: Elaborado pelo autor (2024).

para atender melhor a Regras. Por exemplo, a regra de acessibilidade RA1 foi implementada através da criação de regras para Botões com IconButton e ElevatedButton, TextButton, OutlinedButton e FloatingActionButton, Radio, Switch e outros elementos interativos.

6 CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS

REFERÊNCIAS

- APPLE. Accessibility: Human interface guidelines. 2024. Disponível em: <https://developer.apple.com/design/human-interface-guidelines/accessibility>. Acesso em: 14 set. 2024. Citado 2 vezes nas páginas 38 e 50.
- BRASIL. Decreto nº 5.296, de 2 de dezembro de 2004: Regulamenta as leis nº 10.048, de 8 de novembro de 2000, que dá prioridade de atendimento às pessoas que especifica, e nº 10.098, de 19 de dezembro de 2000, que estabelece normas gerais e critérios básicos para a promoção da acessibilidade das pessoas portadoras de deficiência ou com mobilidade reduzida. **Diário Oficial da União**, 2004. Citado na página 21.
- BRASIL. Lei nº 12.965, de 23 de abril de 2014: Estabelece princípios, garantias, direitos e deveres para o uso da internet no brasil. **Diário Oficial da União**, 2014. Citado na página 21.
- 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). **Diário Oficial da União**, 2015. Citado na página 21.
- DART. Documentação do analyzer plugin: Plugin para análise estática de código dart. 2024. Disponível em: https://github.com/dart-lang/sdk/blob/main/pkg/analyzer_plugin. Acesso em: 12 set. 2024. Citado na página 35.
- GAMA, Victor Leal de Almeida; Kiev. Mobile accessibility guidelines adoption under the perspective of developers and designers. **IEEE**, 2021. Disponível em: <https://ieeexplore.ieee.org/document/9463302>. Acesso em: 11 ago. 2023. Citado na página 37.
- GOOGLE. Flutter: Kit de desenvolvimento de software de interface de usuário. 2023. Disponível em: <https://docs.flutter.dev>. Acesso em: 26 mai. 2023. Citado 4 vezes nas páginas 21, 22, 40 e 44.
- GOOGLE. Make apps more accessible: Android developers guide. 2024. Disponível em: <https://developer.android.com/guide/topics/ui/accessibility/apps>. Acesso em: 14 set. 2024. Citado 2 vezes nas páginas 38 e 50.
- GROUSSET, Gilles. Sonarqube flutter: Plugin para análise estática de código flutter. 2023. Disponível em: <https://github.com/insideapp-oss/sonar-flutter>. Acesso em: 29 jul. 2023. Citado na página 25.
- IBGE. Censo demográfico 2010. **Instituto Brasileiro de Geografia e Estatística**, 2010. Disponível em: <https://censo2010.ibge.gov.br/resultados.html>. Acesso em: 23 mai. 2023. Citado na página 19.
- INVERTASE. Custom lint builder. 2024. Disponível em: https://github.com/invertase/dart_custom_lint. Acesso em: 3 jul. 2024. Citado 4 vezes nas páginas 39, 42, 45 e 47.
- KALBAG, Ela/Dela Laura. **Accessibility for Everyone**. 1. ed. New York, NY: A Book Apart, 2017. Citado na página 20.
- MOHR, Danton Krieck. Validaweb: Uma ferramenta para suporte ao desenvolvimento de software atendendo aos requisitos não funcionais de acessibilidade w3c. **Universidade do Estado de Santa Catarina**, 2022. Citado 2 vezes nas páginas 24 e 25.

NYSTROM, Robert. **Crafting Interpreters**. 1. ed. Seattle, WA: Self-published, 2021. Citado na página 22.

SONARSOURCE. Sonarqube: Plataforma de análise estática de código. 2023. Disponível em: <https://www.sonarqube.org>. Acesso em: 27 jul. 2023. Citado 2 vezes nas páginas 25 e 26.

TEAM, Dart. Publishing packages: Dart package manager. 2024. Disponível em: <https://dart.dev/tools/pub/publishing>. Acesso em: 12 out. 2024. Citado na página 47.

TRAVASSOS, Jorge Biolchini; Paula Gomes Mian; Ana Candida Cruz Natali; Guilherme Horta. Systematic review in software engineering. 2005. Disponível em: <https://www.cos.ufrj.br/uploadfile/es67905.pdf>. Acesso em: 09 ago. 2023. Citado na página 36.

TROSHKOV, Alexabder. accessibility_tools: Ferramenta de análise de acessibilidade para flutter. 2023. Disponível em: https://github.com/rebelappstudio/accessibility_tools. Acesso em: 04 ago. 2023. Citado na página 26.

ULLMAN, Alfred V Aho; Monica S. Lam; Revi Sethi; Jeffrey D. **Compiladores: Princípios, Técnicas e Ferramentas**. 2. ed. São Paulo, SP: Pearson Addison-Wesley, 2008. Citado 2 vezes nas páginas 22 e 44.

GLOSSÁRIO

Árvore Sintática Abstrata: É uma representação abstrata (simplificada) da estrutura semântica de um código fonte escrito em uma certa linguagem de programação.

Análise Estática: Tem por objetivo encontrar vulnerabilidades e demais problemas na aplicação, e normalmente é executada durante a fase de revisão de código dentro do ciclo de vida de desenvolvimento de sistemas. Idealmente, tais ferramentas encontrariam falhas de segurança automaticamente e com alto grau de confiança.

Framework: É um conjunto de bibliotecas, APIs e ferramentas que auxiliam o desenvolvedor a criar aplicações de forma mais rápida e eficiente.

pub.dev: É o repositório oficial de pacotes Dart e Flutter, onde desenvolvedores podem publicar e compartilhar pacotes com a comunidade.

LSP: Language Server Protocol, é um protocolo de comunicação entre editores de código e servidores de linguagem que permite a execução de análises estáticas e outras funcionalidades de forma mais eficiente.

Plugin: É um componente de software que adiciona uma funcionalidade específica a um programa maior.

Lint: Lint é o termo de ciência da computação para uma ferramenta de análise de código estático usada para sinalizar erros de programação, bugs, erros estilísticos e construções suspeitas.

Linter: É uma ferramenta de análise estática que entrega o "Lint" para uma linguagem de programação específica.

Widget: É um componente visual que compõe a interface de usuário de um aplicativo Flutter. Podemos imaginar eles como blocos de construção que compõem a interface de usuário. São usualmente organizados em uma árvore hierárquica.