UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
SISTEMAS DE INFORMAÇÃO

RONALDO ALBERT SPRANGER

FERRAMENTA DE GERAÇÃO DE APLICATIVOS MÓVEIS HÍBRIDO/CROSS-PLATFORM BASEADA EM EDITOR DE INTERFACE

Florianópolis, SC
2019

RONALDO ALBERT SPRANGER

FERRAMENTA DE GERAÇÃO DE APLICATIVOS MÓVEIS HÍBRIDO/CROSS-PLATFORM BASEADA EM EDITOR DE INTERFACE

Trabalho de Conclusão de Curso apresentado ao curso de Sistemas de Informação da Universidade Federal de Santa Catarina, como requisito parcial para a Obtenção do grau de Bacharel em Sistemas de Informação.

Florianópolis, SC
2019

RONALDO ALBERT SPRANGER

FERRAMENTA DE GERAÇÃO DE APLICATIVOS MÓVEIS HÍBRIDO/CROSS-PLATFORM BASEADA EM EDITOR DE INTERFACE

Trabalho de Conclusão de Curso apresentado ao curso de Sistemas de Informação da Universidade Federal de Santa Catarina, como requisito parcial para a Obtenção do grau de Bacharel em Sistemas de Informação.

Florianópolis, SC, 01 de novembro de 2019

BANCA EXAMINADORA

Prof. Dr. Frank Augusto Siqueira
Orientador

Prof. Dr. Leandro José Komosinski
Banca Examinadora

Prof. Dr. Jean Carlo Rossa Hauck
Banca Examinadora

AGRADECIMENTOS

Agradeço primeiramente a minha noiva Maria Carolina, pois sem ela eu jamais teria o apoio necessário para entrar na Universidade e a força necessária para termina-la. A minha família, representada pela figura máxima de minha mãe. Mesmo nos momentos mais difíceis da minha vida, estiveram sempre ao meu lado me dando o suporte necessário para jamais desistir.

Aos mestres, que durante todo o período acadêmico me auxiliaram e sempre estiveram disponíveis. Um agradecimento especial ao meu orientador Frank Siqueira por me guiar em todo o processo de desenvolvimento deste trabalho.

Aos meus amigos que propiciaram momentos incríveis nesta longa jornada, tantos os novos amigos feito durante o curso quanto aos antigos que não se afastaram durante o tempo que estive ausente para resolver minha vida acadêmica.

Aos meus tutores e pessoas que tornaram isto possível, Samuel Gauer por me dar a oportunidade de trabalhar como programador. Clovis Rossi por sempre me mostrar que eu poderia fazer melhor, mesmo quando eu acreditava já ter feito o meu melhor. Rodrigo Branas por me mostrar o maravilhoso mundo do javascript; sem seus cursos e suas palestras o caminho até aqui teria sido bem mais difícil.

A fundação CERTI, em especial Edmar Pessoa e Alexandre Lucas que sem o suporte fornecido desta organização eu jamais conseguiria ter a disponibilidade de conseguir conciliar vida acadêmica, profissional e pessoal. Alexandre Lemos e Marcelo Frantz que sempre me fornecem valorosas lições técnicas e sempre apresentaram diferentes caminhos quando novos obstáculos apareceram.

RESUMO

Com um número cada vez maior de smartphones no mercado e as singularidades entre o desenvolvimento Android e iOS, tem se tornado cada vez mais difícil escolher o modelo correto de desenvolvimento para mobile. Cada um dos sistemas operacionais tem suas linguagens padrão de desenvolvimento e suas peculiaridades. Partindo desta dificuldade, empresas e desenvolvedores vêm buscando cada vez mais alternativas híbridas de desenvolvimento, assim como frameworks que facilitem o desenvolvimento de aplicativos móveis. O presente trabalho propõe a analisar ferramentas já utilizadas para criação de aplicativos e apresentar uma nova ferramenta de desenvolvimento de aplicações mobile. Para isso foi utilizado um editor de arrastar e soltar componentes e vincular trechos de código pré-determinados a estes. Para completar a tarefa, foram utilizadas metodologias de experiência de usuário e ferramentas consolidadas para o desenvolvimento de aplicações web. Como resultado, um editor de aplicativos móveis web foi construído. Através dele o usuário pode inserir componentes, editar propriedades e atribuir métodos pré-definidos a componentes. No final do proceso o usuário pode gerar uma aplicação multiplataforma e baixar o código gerado pela aplicação.

Palavras-chave: Aplicativos móveis. Interface de usuário. Flutter. Geração de Código. Scaffolding. Aplicativos Cross-Plataform.

ABSTRACT

The increasing number of smartphones on the market and singularities between Android and iOS development it has become increasingly difficult to choose the right model of development for mobile. Each of the operating systems has their standard development languages and their peculiarities. Starting from this difficulty, companies and developers are looking for hybrid development alternatives, as well as frameworks that facilitate the development of mobile applications. The present work proposes to analyze tools already used to create applications and present a new tool for developing mobile applications using a drag-and-drop component editor and to link pre-determined code snippets to them. To complete the task, user experience methodologies and consolidated tools for web application development will be used. At the end of this project, a mobile app editor has been built. Through it, the user can insert components, edit properties and assign predefined methods to components. No final user process can generate a multiplatform application and download the code generated by the application.

Keywords: Mobile application. User interface. Flutter. Code Generation. Scaffolding.

LISTA DE CÓDIGOS, DIAGRAMAS, FIGURAS E TABELAS

Tabela 1 — Market share sistemas operacionais smartphones no mundo 13
Tabela 2 — Lista de desafios 14
Figura 1 — Arquitetura de aplicação web simplificada 21
Tabela 3 — Comparativo de abordagens de desenvolvimento de apps 23
Figura 2 — Desenvolvimento Nativo vs. Cross vs. Web 24
Figura 3 — Arquitetura de uma nativa 25
Figura 4 — Arquitetura de uma aplicação híbrida 27
Tabela 4 — String de Busca 28
Figura 5 — App Inventor - Tela inicial 30
Figura 6 — App Inventor - Bloco de programação 31
Figura 7 — Seattle Clouds seleção de templates 33
Figura 8 — Seattle Clouds, seleção de plataformas 34
Figura 9 — Thunkable spike 35
Figura 10 — Thunkable code-blocks 36
Figura 11 — Tela inicial flutter studio 39
Figura 12 — Visualização do código da ferramenta 40
Tabela 5 — Tabela comparativa entre ferramentas 41
Tabela 6 — Análise de recursos e componentes 42
Diagrama 1 — Diagrama de casos de uso 46
Diagrama 2 — Diagrama de Sequência 1 47
Diagrama 3 — Diagrama de Sequência 3 48
Diagrama 4 — Diagrama de Sequência - 3 49
Diagrama 5 — Diagrama de sequência - Insert Node 50
Figura 13 — Protótipo de baixa fidelidade 51
Diagrama 6 — Arquitetura da aplicação 52
Figura 14 — Fluxo de dados Redux 67
Código 1 — Layout é um vetor com items como objetos 68
Código 2 — Evento disparado ao arrastar 68
Código 3 — Evento disparado ao redimensionar 68
Tabela 7 — Outras bibliotecas JavaScript utilizadas 70
Figura 15 — Estado inicial de um botão 72
Figura 16 — Menu de propriedades do componente 73
Código 4 — Koa router 74
Figura 17 — Árvore de componentes de interface Flutter 74
Código 5 — Estrutura de um nó 75
Código 6 — Template de um botão 76
77

Código 7 — Modelo padrão de código .dart 77
Figura 18 — Aplicativo 78
Figura 19 — Versão final da aplicação 79
Figura 20 — Menu com imagem dos componentes 80
Figura 21 — Configurações globais de estilo 81
Figura 22 — Configuração de interface 82
Figura 23 — Editor de propriedades do componente 83
Figura 24 — Configuração de estilo do componente 84
Figura 25 — Menu de configuração de comportamento do componente 85
Figura 26 — Componentes de recursos nativo 86
Figura 27 — Recurso de câmera 86
Figura 28 — Recurso de mapa 87
Figura 29 — Recurso de webview 87
Figura 30 — Layout editado pela ferramenta 88
Tabela 8 — Análise comparativa entre ferramentas 90

LISTA DE ABREVIATURAS E SIGLAS

2D	2 Dimensões
AOT	Ahead-of-Time
API	Application Program Interface
App	Aplicativo
CPU	Central Processing Unit
CRUD	Create Read Update Delete
DOM	Document Object Model
DSL	Domain Specific Language
GPS	Global Positioning System
GPU	Graphic Processing Unit
GUI	Graphic User Interface
HTML	Hypertext Markup Language
IBM	International Business Machines
IDE	Integrated Development Environment
IO	Input/Output
LLVM	Low Level Virtual Machine
MIT	Massachusetts Institute of Technology
MVC	Model View Controller
NDK	Native Development Kit
NFC	Near-field communication
ODM	Object Data Mapping
ORM	Object-Relational Mapper
PaaS	Platform as a Service
RIA	Rich Internet Applications
SDK	Software Development Kit
SO	Sistema Operacional
UF	Usuário Final
UI	User Interface
UIKit	User Interface Kit
VM	Virtual Machine
W3C	World Wide Web Consortium

WebApps	Aplicaticações Web
XML	Extensible Markup Language

SUMÁRIO

1 **INTRODUÇÃO** 12
1.1 **DESCRIÇÃO DO PROBLEMA** 14
1.2 **OBJETIVOS** 16
1.3 **ESCOPO** 16
1.4 **METODOLOGIA** 17
1.5 **ORGANIZAÇÃO DO TRABALHO** 18
2 **FUNDAMENTAÇÃO TEÓRICA** 19
2.1 **APLICAÇÕES WEB** 19
2.2 **GERADORES DE CÓDIGO E SCAFFOLDING** 21
2.3 **MOCKUP** 22
2.4 **APLICATIVOS MÓVEIS** 23
2.5 **DESENVOLVIMENTO NATIVO** 25
2.6 **DESENVOLVIMENTO HÍBRIDO / CROSS-PLATFORM** 26
2.7 **APLICATIVOS WEB** 27
3 **ESTADO DA ARTE** 28
3.1 **DEFINIÇÃO DO ESTUDO** 28
3.2 **TRABALHOS RELACIONADOS E FERRAMENTAS DISPONÍVEIS** 29
3.2.1 **APP INVENTOR** 29
3.2.2 **SEATTLE CLOUDS** 32
3.2.3 **THUNKABLE** 34
3.2.3.1 **REACT NATIVE** 37
3.2.3.2 **PHONEGAP/ APACHE CORDOVA** 37
3.2.3.3 **ANDROMO** 38
3.2.3.4 **FLUTTER STUDIO** 38
3.3 **ANALISE COMPARATIVA** 41
4 **A FERRAMENTA PROPOSTA** 44
4.1 **DESCRIÇÃO** 44
4.2 **LEVANTAMENTO DE REQUISITOS** 44
4.2.1 **REQUISITOS FUNCIONAIS** 44
4.2.2 **REQUISITOS NÃO FUNCIONAIS** 45
4.2.3 **CASOS DE USO** 45
4.2.4 **DIAGRAMA DE SEQUÊNCIA** 46
4.2.5 **PROTOTIPAÇÃO E ARQUITETURA** 50
4.2.5.1 **Módulos do Frontend** 53
4.2.5.2 **Módulos do Backend** 53
4.2.5.3 **Aplicações de Terceiros** 54
55

5 **DESENVOLVIMENTO DA FERRAMENTA** 55
5.1 **DART** 55
5.2 **FLUTTER** 56
5.3 **TESTES DE INTERFACE E TESTES DE UNIDADE** 58
5.4 **JAVASCRIPT** 59
5.5 **NODEJS** 60
5.6 **BIBLIOTECAS JAVASCRIPT** 63
5.6.1 **REACT** 64
5.6.1.1 **JSX** 64
5.6.1.2 **Componentes com estado (*Stateful components*)** 65
5.6.1.3 **Modelo de objeto de documento virtual (Virtual DOM)** 65
5.6.1.4 **REDUX** 65
5.6.2 **REACT-GRID-LAYOUT** 67
5.6.3 **MATERIAL-UI** 69
5.6.4 **OUTRAS BIBLIOTECAS JAVASCRIPT** 69
5.7 **FIREBASE** 70
5.8 **PROVA DE CONCEITO** 71
5.8.1 **Frontend** 71
5.8.2 **Backend** 73
5.9 VERSÃO FINAL DA FERRAMENTA 78
5.9.1 **Menu de componentes** 79
5.9.2 **CONFIGURAÇÃO DA APLICAÇÃO** 80
5.9.3 **Menu de configuração de interface** 81
5.9.4 **Configuração de componente** 82
5.9.4.1 Configuração de texto 82
5.9.4.2 Configuração de estilo 83
5.9.4.3 Configuração de comportamento de componente 84
5.9.5 **Componentes de recursos nativos** 85
5.10 **LIMITAÇÕES E FALHAS CONHECIDAS** 88
6 **CONCLUSÕES E TRABALHOS FUTUROS** 90
6.1 **CONCLUSÃO** 90
6.2 **TRABALHOS FUTUROS** 91
REFERÊNCIAS 93
APÊNDICE A — MANUAL DE AMPLIAÇÃO DA FERRAMENTA 98
APÊNDICE B — ARTIGO 103
APÊNDICE C — CÓDIGO 115

1 **INTRODUÇÃO**

Aplicativos móveis (*apps* abreviadamente) estão desempenhando um papel cada vez mais importante em nossas vidas diárias. Com a popularização dos *smartphones,* iniciada em 2008, o número de pessoas que estão conectadas através de dispositivos móveis aumenta vertiginosamente. Segundo a Globalstats (2019) estima-se que em 2020 haverá 6.1 bilhões de usuários conectados por meio de *smartphones* no mundo. Com a popularização da tecnologia e a evolução da internet móvel nos últimos anos, está cada vez mais acessível ter um dispositivo móvel inteligente, o que se tornou uma espécie de assistente para uso tanto pessoal quanto profissional. Ainda segundo o site *Globalstats, e*m maio de 2019, há mais tráfego de rede feito por dispositivos móveis (48,19%) em comparação com *desktops* / *laptops* (47%).

A introdução da *App Store*, um lugar comum para publicar aplicativos iOS[1] (*iPhone Operating System*), em 2008 foi uma ótima notícia para muitos desenvolvedores[2]. O *Google* seguiu essa nobre ideia ao criar o *Android Market,* posteriormente incorporado ao *Google Play*[3]*,* abrindo portas para desenvolvedores e empresas para alcançar usuários de *smartphones* com certa facilidade[4]. Para que seja distribuído para um grande número de usuários finais, um aplicativo móvel precisa ser programado para as duas plataformas mais populares atualmente, ou seja, *Android* e *iOS*. Evidentemente, as diferenças entre estes duas plataformas são grandes e muitas vezes exigem diferentes conjuntos de habilidades para o desenvolvimento, como as linguagens Java ou Kotlin apenas para *Android* e *Objective-C* ou *Swift* apenas para iOS. Assim, empresas e desenvolvedores muitas vezes lutam para lidar com a complexidade do desenvolvimento de aplicativos multiplataforma.

A competição entre os sistemas operacionais é o principal fator que leva os desenvolvedores a adicionar novos recursos para o sistema (Guo, 2008). Esses sistemas operacionais possuem diferenças significativas em suas arquiteturas, o que acaba forçando desenvolvedor a criar uma aplicação para cada sistema. No entanto é difícil desenvolver aplicativos móveis para uma ampla gama de *smartphones* sem um esforço de portabilidade significativo (Niyar, 2012). Segundo Bosika (2012, *apud* Moon Hui 2013) muito tempo, dinheiro, tecnologias e mão de obra são necessários para o desenvolvimento de diversos aplicativos. As estatísticas mostraram que o Android e o iOS mantiveram suas posição de liderança no mercado móvel mundial atual, como mostrado na Tabela 1.

1 https://www.apple.com/br/ios/home/
2 https://www.apple.com/newsroom/2018/07/app-store-turns-10/
3 https://play.google.com/store
4 https://googleblog.blogspot.com/2012/03/introducing-google-play-all-your.html

Tabela 1 - Market share sistemas operacionais smartphones no mundo

Sistema Operacional	
Android	75.27%
iOS	22.74%
KaiOS	0.75%
Windows	0.24%
Samsung	0.22%
Outros	0.32%

Fonte: Global Stats, 2019

Com a disparidade entre sitesmas operacionais(SOs, abreviadamento) de cada plataforma, algumas soluções surgiram para o desenvolvimento multiplataforma ou *cross-plataform*[5]*, u*sando diferentes tipos de abordagem para o tema. Aplicativos Web (*WebApps*, abreviadamente) usando JavaScript e HTML5 é uma das soluções, mas *WebApps* sendo executados no *browser* do *smartphone* apresentam problemas de desempenho que fazem com que desenvolvedores e empresas procurem por soluções híbridas ou nativas. Já o desenvolvimento híbrido conta com ferramentas como *PhoneGap, Ionic* e *React Native*, as quais serão apresentadas no capítulo 3 de análise de ferramentas do presente trabalho, que partem de abordagens diferentes para melhorar o desempenho ou até mesmo facilitar o desenvolvimento do aplicativo móvel. O desenvolvimento nativo também é uma opção entre empresas e desenvolvedores, mas segundo (CHARLAND; LEROUX, 2011) existem dois problemas com a desenvolvimento nativo. Primeiro, criar um aplicativo diferente para cada plataforma é muito caro se escrito em cada idioma nativo. Um desenvolvedor ou startup pode suportar apenas um dispositivo, provavelmente o iPhone, mas um departamento de TI terá que suportar os dispositivos que seus usuários têm que nem sempre pode ser o mais recente e melhor. Segundo, o argumento de desempenho de que aplicativos nativos são mais rápidos pode se aplicar a jogos 3D ou aplicativos de processamento de imagem, mas há uma penalidade de desempenho insignificante ou imperceptível em um aplicativo de negócios bem construído usando a tecnologia da híbridas.

Atualmente são diversas as ferramentas que se propõem a criar aplicativos na forma de arrastar e soltar, mas a grande maioria ofertada ainda é paga. Entre estas plataformas está a Seattle Clouds[6] que é uma das mais conhecidas plataformas para

5 https://www.techopedia.com/definition/17056/cross-platform
6 https://seattleclouds.com/

criar aplicativos *cross-plataform.*

Em fevereiro de 2018 a *Google* lançou o Flutter. O Flutter é um novo SDK (*Software Development Kit*) para *apps* que ajuda desenvolvedores e designers a criar aplicativos móveis modernos para *iOS* e *Android*. A estrutura moderna e reativa permite criar uma interface do usuário poderosa com animações, base de código compartilhada e visualizações nas plataformas iOS e Android. Facilita o processo de desenvolvimento, custos de desenvolvimento com implantação mínima e rápida.

1.1 **DESCRIÇÃO DO PROBLEMA**

As demandas de aplicativos móveis para *smartphones* aumentaram tremendamente nos últimos anos. Para atender às demandas de diversos usuários, os desenvolvedores de aplicativos adotam diferentes estratégias, variando de nativo, para a web e aplicativos híbridos. No entanto, cada uma das estratégias tem seus prós e contras.

Um estudo de Ahmad *et. al.* (2017) revisando 78 artigos e publicações identificou os maiores problemas segundo a bibliografia disponível, que são elencados na Tabela 2.

Tabela 2 - Lista de desafios

Desafios	Frequência (n=78)	%
Experiência do usuário	42	64
Fragmentação	36	46
Testes	34	44
Compatibilidade	36	46
Reuso de código	12	15
Falta de ferramentas de suporte	11	14
Falta de perícia técnica	11	14
Segurança	3	4

Fonte: ARSHAD et al. (2018, p. 5)

Os problemas mais comuns relatados na literatura explicitamente associada com o desenvolvimento nativo são: fragmentação (Holzer, 2012), testes (Joorabchi, 2013) e falta de reutilização de código (Wasserman, 2010). Os desafios associados ao uso de tecnologias da web são a compatibilidade (Holzer, 2012), a experiência do usuário (Charland, 2011) e a falta de acesso a recursos de hardware do dispositivo (Holzer, 2012). Os desafios associados ao desenvolvimento de aplicações móveis híbridas estão na realização de testes (Boushehrinejadmoradi, 2015), em *user experience* (Xanthopoulos, 2013) e na falta de suporte às ferramentas (Ohrt, 2012).

Organizações estão enfrentando o problema repetidamente para decidir qual estratégia vai servir e satisfazer melhor os usuários finais para atender suas requisitos desejados de forma eficiente. Todas essas decisões são tomadas com base na disponibilidade de recursos (tempo e orçamento), competências, conhecimentos e ferramentas de desenvolvimento.

Outro problema para o desenvolvimento de *apps,* segundo Perchat (2013), é a experiência do usuário, que elenca a dificuldade de alcançar o objetivo de sempre definir a melhor interface de usuário, para um uso simples e intuitivo. IDEs (*Integrated Development Environment*) como o *Android Studio*[7] permitem que o usuário arraste e solte componentes na tela para geração de layout e personalização, assim como atribuir eventos a esses componentes. A interface do usuário de um aplicativo Android consiste em um conjunto de *activities*[8]*.* Cada *activity* representa uma única tela com uma interface. Cada uma dessas interfaces é uma composição de *widgets*[9]*,* como caixas de texto, caixas de seleção e botões. Esses *widgets* são incorporados em contêineres de layout que definem a estrutura visual da interface. Ao aninhar contêineres de layout em outro contêiner, o desenvolvedor cria uma árvore de elementos da interface do usuário. Apesar de também ser possível definir a interface do usuário diretamente no código-fonte, as diretrizes para desenvolvedores do *Android* recomendam que as árvores dos elementos da interface do usuário sejam declaradas em arquivos de layout XML.

Um estudo realizado por Joorabchi *et al.* (2013) com desenvolvedores de aplicações mobile revelou que 76% deles considera como o maior desafio lidar com múltiplas plataformas móveis durante o desenvolvimento de aplicativos. Enquanto dispositivos móveis e plataformas estão se movendo em direção à fragmentação, o processo de desenvolvimento contemporâneo está perdendo a adaptação para alavancar o conhecimento de uma plataforma para outra. Os desenvolvedores atualmente tratam o aplicativo móvel para cada plataforma separadamente e verificam manualmente se a funcionalidade está preservada em várias plataformas. Ainda sobre este tema, os desenvolvedores de dispositivos móveis precisam de ferramentas de análise melhores para acompanhar as métricas de seus aplicativos durante a fase de desenvolvimento. Além disso, o teste é um desafio significativo. As atuais estruturas de teste não fornecem o mesmo nível de suporte para diferentes plataformas e as ferramentas de teste atuais não suportam recursos importantes para testes móveis, como mobilidade, serviços de localização, sensores ou diferentes gestos e entradas.

7 https://developer.android.com/studio
8 https://developer.android.com/guide/components/activities.html
9 https://canaltech.com.br/produtos/O-que-e-widget/

As abordagens de desenvolvimento *cross-platform* surgiram para enfrentar esse desafio, permitindo que os desenvolvedores implementassem seus aplicativos em uma etapa para uma variedade de plataformas, evitando repetição e aumento da produtividade. Por um lado, essas abordagens precisam oferecer generalidade adequada para permitir o fornecimento de aplicativos para várias plataformas. Por outro lado, eles ainda têm que permitir que os desenvolvedores aproveitem as vantagens específicas e possibilidades de *smartphones*.

1.2 **OBJETIVOS**

O presente trabalho se propõe a desenvolver uma ferramenta que facilite a programação e criação de layout de interfaces de aplicativos móveis multiplataforma. Esta aplicação web gerará estrutura de uma aplicação móvel fiel ao layout modelado no editor com eventos e ações selecionadas pelo usuário no momento da edição. Esta aplicação móvel deve ser fiel ao modelo feito na ferramenta web e seu comportamento também deve acontecer conforme os eventos selecionados pelo usuário.

Os objetivos específicos do trabalho são:

- Realizar estudo sobre desenvolvimento de aplicações móveis e suas diferente abordagens;
- Analisar a literatura sobre geração de código de aplicações móveis e design de interfaces utilizando bibliotecas existentes;
- Fazer uma análise do estado da arte de *frameworks* úteis para edição e geração de código de aplicativos;
- Elaborar a documentação composta por requisitos (funcionais, não funcionais), casos de uso e protótipos de telas;
- Criar uma aplicação web em formato de prova de conceito que permita edição de *layouts* de aplicações móveis multiplataforma utilizando scaffolds de componentes e realizando métodos básicos de um aplicativo;

1.3 **ESCOPO**

O artefato produzido para este trabalho consiste em uma aplicação web, desenvolvida usando *NodeJS* e JavaScript, que permite edição de *layouts* de interface para aplicativos mobile. Esta ferramenta, através de recurso de arrastar e soltar, permite a criação de interfaces de aplicações e o mapeamento de trechos de código para atribuir a estes componentes usando técnicas de *scaffolding*.

Como saída desta aplicação, uma arquitetura de aplicação *mobile*

cross-platform é gerada. Utilizando a *framework Flutter* escrita em código *Dart s*endo fiel a prototipagem realizada na ferramenta web e realizando as tarefas associadas aos *scaffolding.*

A abrangência do presente trabalho é definida em um número de componentes limitados para criação de aplicações simples afim de demonstrar em uma prova de conceito que é possível utilizar estas técnicas para desenvolvimento de aplicações, podendo esta ser estendida a novos trabalhos já que o código é aberto. Também é previsto para este trabalho a elaboração da documentação necessária para o correto uso da ferramenta.

1.4 **METODOLOGIA**

Na primeira fase, de arquitetura da aplicação, informações serão coletadas em publicações físicas e eletrônicas, assim como de maneira pessoal com profissionais da área. Após a catalogação, serão estudados os *frameworks* que melhor atendem a proposta do trabalho. Nesta etapa será elaborada uma síntese da literatura referente ao desenvolvimento de aplivcativos móveis multiplataforma, assim como tecnologias

Na segunda fase do projeto é feita uma análise sobre ferramentas existentes semelhantes a ferramenta proposta e suas questões de design e usabilidade. Esta etapa possui as seguintes atividades:

- Pesquisar as ferramentas semelhantes a ferramenta proposta e suas principais características.
- Extrair e analisar as características das interfaces de usuário das ferramentas, como os padrões de cores, principais funcionalidades e principais tecnologias.
- Comparar as soluções analisadas para analisar a viabilidade de uma nova solução.

Na terceira fase do projeto será feita a elaboração de documentos necessários para realizar a implementação da interface e da API do projeto com baseado nas informações obtidas nas etapas anteriores. Esta etapa consiste nas seguintes atividades:

- Elaborar documentação baseado nas informações resultantes nas aplicações semelhantes a ferramenta proposta.
- Implementação de uma prova de conceito tendo como referência a documentação gerada na etapa anterior.

Na quarta e última fase serão realizados testes com uma versão estável da ferramenta. Esta etapa consiste avaliar o desenvolvimento da aplicação e descrever

todo o processo de desenvolvimento. Também é previsto a escrita da documentação necessária para instalação e utilização da aplicação.

1.5 **ORGANIZAÇÃO DO TRABALHO**

Este trabalho está organizado em 6 capítulos. O capítulo 2 visa apresentar os conceitos fundamentais explorados neste projeto, dando uma visão geral das tecnologias utilizadas no desenvolvimento Web e que fundamentam a Web e aplicações móveis. O capítulo 3 descreve o estado da arte de aplicações comerciais e não comerciais relacionados ao fruto desta pesquisa. O capítulo 4 visa demonstrar como este trabalho pretende desenvolver a ferramenta proposta descrevendo em minúcias como se dará o desenvolvimento da aplicação. O capítulo 5 demonstra o desenvolvimento da ferramenta e suas características. E por fim são apresentados os resultados obtidos na criação do presente trabalho, as conclusões e os caminhos que podem ser seguidos no futuro com a ferramenta.

2 **FUNDAMENTAÇÃO TEÓRICA**

2.1 **APLICAÇÕES WEB**

As páginas Web em HTML surgiram junto com o conceito da *World Wide Web,* criado por Tim Berners-Lee em 1989, que criou o primeiro website em 1991 e modificou completamente a forma como os usuários utilizam a Internet. Desde então, a Web vem evoluindo constantemente com criações de novas tecnologias para interação e desenvolvimento de páginas dinâmicas. Segundo a organização *Website Hosting Rating*[10]*,*em janeiro de 2019 a Internet contava com mais de 4 bilhões de usuários em todo o mundo e 342 milhões de domínios registrados, nos quais são hospedados quase 2 bilhões de *websites.* Diariamente são feitos 4 milhões de *posts* em *blogs* e 5 bilhões de pesquisas no Google, sendo que 15% destas pesquisas nunca foram feitas antes.

Segundo Warren (1999), tradicionalmente a distinção entre software e dados foi bem definida, mas atualmente é mais difícil distinguir entre os dois. HTML pode ser interpretado como um programa, bem como apenas dados, para que a Web possa ser vista como um programa com alto conteúdo de dados e baixo conteúdo processual. Existem algumas interessantes conseqüências dessa interpretação. Um documento HTML é basicamente texto, que instrui o navegador (atuando como intérprete de código) a exibir texto na tela do usuário. O texto é pontuado por *tags*. O uso principal de *tags* é para indicar a inserção de imagens, definir *layout* e outros componentes externos, que podem, portanto, ser vistos como análogos a componentes de caixa preta em um programa convencional. Outro uso de *tags* é indicar *hiperlinks* para outros documentos HTML. Estas representam opções para o utilizador, que clicando em um *link* obtém uma nova página e a exibe no navegador.

O HTML é uma linguagem de marcação, usada para estruturar e apresentar conteúdo para a *World Wide Web* e uma tecnologia central da Internet, mantida pelo W3C[11]. Com a evolução da própria Web, outros elementos foram adicionados para melhorar a interação do usuário com a interface, assim como também para deixá-la mais amigável para todos os tipos de públicos. Desta forma surgiu a linguagem de script Javascript, que é interpretada pelo *browser,* adicionando elementos de forma dinâmica e permitindo maior liberdade para os programadores editarem suas interfaces; assim como o CSS[12] (*Cascading Style Sheets*), que trouxe elementos como guia de estilos para as páginas, melhorando seu visual e permitindo uma série de novas interações nas páginas que antes eram estáticas.

---

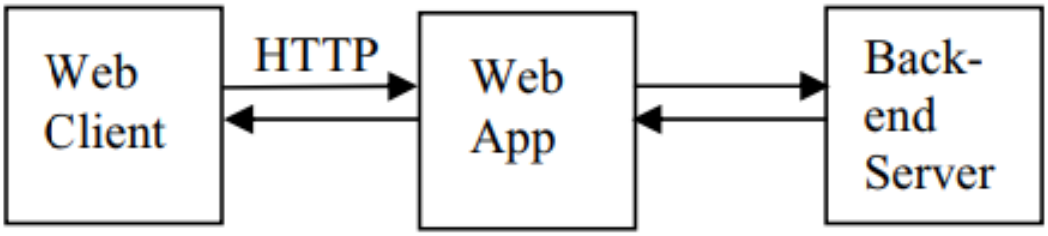
10 https://www.websitehostingrating.com/internet-statistics-facts/
11 http://www.w3.org/TR/html51/
12 https://developer.mozilla.org/pt-BR/docs/Web/CSS

Segundo Kappel (2004 *apud* Chaniotis, 2014) Aplicações Web podem ser definidas como “sistemas de software baseados em tecnologias e padrões do *World Wide Web Consortium* (W3C). Eles fornecem informações específicas da Web, recursos como conteúdo e serviços através de uma interface de usuário, o navegador da Web.”. Um aplicativo da Web, também chamado de Web App, é um programa aplicativo armazenado em um servidor remoto e entregue pela Internet através de uma interface de navegador. É uma aplicação em que todos ou algumas partes do software são baixadas da web toda vez que são executadas. Por ter um sistema de aplicação web, os usuários poderão usar navegadores como clientes para atualizar e manter aplicativos da Web sem distribuir ou instalar software em muitos computadores. Como diferencial, os aplicativos da Web suportam compatibilidade entre plataformas. Aplicações web comuns incluem*webmail*, vendas e leilões *online*, *wikis*, serviços de mensagens instantâneas e muitas outros funções (ZHENG; JIACONG, 2016).

Ainda, segundo Zheng (2016) aplicações Web podem ser subdivididas em três tipos: Baseado no navegador, Baseado no cliente e Aplicativos Web para dispositivos móveis. Do ponto de vista técnico, a Web é um ambiente altamente programável que permite a personalização em massa através do Implantação de uma ampla e diversificada gama de aplicativos para usuários globais. De forma semelhante aos navegadores da Web, que permitem que os usuários recuperem dados e interajam com conteúdo localizado em páginas da Web em um site, aplicações Web são programas de computador que permitem que os visitantes do site enviem e recuperem dados de / para um banco de dados pela Internet usando seu navegador da Web preferido. Os dados são então apresentadas ao usuário em seu navegador, pois as informações são geradas dinamicamente pela aplicação web através de um servidor web. Os documentos são gerados em um formato padrão para permitir suporte por todos os navegadores.

Uma aplicação web é composta por uma coleção de scripts, que residem em um servidor da Web e interagem com bases de dados ou outras fontes de conteúdo dinâmico. Conforme Fong (2007), usando a infra-estrutura da Internet, aplicações Web permitem que provedores de serviços e clientes compartilhem e manipulem informações em uma plataforma de maneira independente. Alguma das categorias de tecnologias de aplicação Web são protocolos de comunicação, formatos no *server-side* e linguagens de script no *client-side*, *plug-ins* do navegador e APIs do servidor Web. Uma aplicação Web tem uma arquitetura de n-camadas. Normalmente, existe um cliente (navegador da web), um servidor da Web, um servidor de aplicações (ou vários) e uma camada de persistência (banco de dados). A figura 1 apresenta uma visão simplificada de uma aplicação Web.

Figura 1 - Arquitetura de aplicação web simplificada



Fonte: Fong e Okun (2007, p. 3)

2.2 **GERADORES DE CÓDIGO E SCAFFOLDING**

A geração de código existe desde os anos 1950, tendo sua origem nos primeiros compiladores (Rich, 1988). Desde então, as organizações de software têm confiado em técnicas de síntese de código para reduzir o tempo de desenvolvimento e aumentar a produtividade (Kelly, 2008). A geração de código pode automatizar a criação de código-fonte através de quadros genéricos, classes, protótipos, modelos, aspectos e geradores de código para melhorar a produtividade dos programadores Abbas, Jeberson e Klinsega (2012). Isto é um método eficaz para obter reutilização de software na aplicação desenvolvimento e tem muitos casos de sucesso em software, aplicativos da Web e sistemas cliente-servidor distribuídos.

> "O scaffolder é basicamente uma ferramenta para automatizar a programação de tarefas repetitivas, que do contrário seriam codificadas a mão. A programação se torna repetitiva quando a aplicação precisa seguir estruturas e padrões de desenvolvimento". (O GOULART MAGNO, 2015)

Em relação ao esforço de desenvolvimento, uma das tarefas mais repetitivas no desenvolvimento de aplicativos que fazem uso intensivo de dados é a implementação de operações *Create/Read/Update/Delete* (CRUD abreviadamente) (Rodríguez-Echeverría *et al.*, 2019). Uma solução que se proponha a gerar código para execução destas tarefas repetitivas sempre vem acompanhada pela promessa de um aumento de produtividade e de um ganho significativo de tempo de desenvolvimento de aplicações complexas. A técnica de *scaffolding* cria automaticamente as estruturas mais significativas de um sistema de dados persistentes: o mapeamento entre objetos, banco de dados e interfaces CRUD. Essas estruturas estão intimamente relacionadas entre si e dependem de padrões

específicos, portanto a técnica somente poderá ser utilizada para a criação de sistemas CRUD com determinado tipo de padrão. Outras ferramentas não relacionadas a *scaffolding* permitem gerar outros componentes, tais como: camada de segurança, integração de sistemas, testes unitários, etc. (ADAMUS *et al.*, 2010 *apud* Magno 2015). Técnicas de *scaffolding* já são exploradas por diversos *frameworks* para desenvolvimento *web* e *mobile,* como *React Native* e *Ionic*[13]*.,* assim como o *scalfolding* através de *mockups,* usado por *App Inventor, PhoneGap, Firebase, etc.* Entretanto, todas esbarram em técnicas proprietárias de programação que não permitem a portabilidade deste *frontend* para outras linguagens.

2.3 **MOCKUP**

A definição de *mockup* pode ser retirada do dicionário Merrian-Webster[14] como: "um modelo estrutural de tamanho real construído para dimensionar principalmente para estudo, teste ou exibição" ou ainda "uma amostra de trabalho (como de uma revista) para revisar formato, layout ou conteúdo". Em aplicações Web, um *mockup* refere-se a um modelo em execução, navegável, parcial ou de tamanho total da aplicação, usado para elicitação de requisitos, validação e finalização. É possível verificar todos esses recursos carregando inspeções de interação do usuário, em que um especialista, avaliadores, ou mesmo a equipe de desenvolvimento, pode avaliar aspectos de design, navegação e facilidade de uso de uma interface de usuário (RIVERO; BARRETO; CONTE, 2013).

Para ajudar os engenheiros de software a identificar problemas de navegação e usabilidade, no início do processo de desenvolvimento de aplicações Web é criado um conjunto de protótipos de baixa fidelidade (ou *mockups*), que são imagens que mostram como o software ficaria depois da sua implementação (RIVERO; BARRETO; CONTE, 2013). Um *mockup* ajuda o projetista a tomar decisões finais sobre os esquemas de cores de um produto, estilo visual e tipografia. Com uma maquete, é possível experimentar com o aspecto visual do produto para ver o que parece melhor. Nesse momento, pode-se pedir *feedback* a possíveis usuários e fazer as alterações necessárias imediatamente. Isso economizará muito mais tempo do que voltar e fazer ajustes na interface do usuário após o lançamento do produto.

13 Ionic - Cross-Platform Mobile App Development: https://ionicframework.com/
14 https://www.merriam-webster.com/dictionary/mock-up

2.4 **APLICATIVOS MÓVEIS**

O desenvolvimento para dispositivos móveis geralmente requer diferentes conjuntos de habilidades e familiaridade com diferentes ambientes de tempo de execução do que aqueles mais comumente usados em PCs *desktop*. Escolher a abordagem correta de desenvolvimento para *apps* é sempre uma escolha difícil, pois cada uma delas tem suas vantagens e desvantagens (Smutný, 2012).

Três abordagens são adotadas atualmente para o desenvolvimento de aplicativos móveis: a abordagem nativa, que implica em utilizar o suporte de programação suportado nativamente pelo sistema operacional do dispositivo móvel; a abordagem Web, que consiste em desenvolver aplicações Web que poderão ser executadas em qualquer dispositivo que possua um navegador Web; e a abordagem híbrida, ou cross-plataforma, que reúne características das duas abordagens anteriores, buscando reunir as vantagens de ambas.

A Tabela 3 mostra as vantagens e desvantagens de cada abordagem:

Tabela 3 - Comparativo de abordagens de desenvolvimento de apps

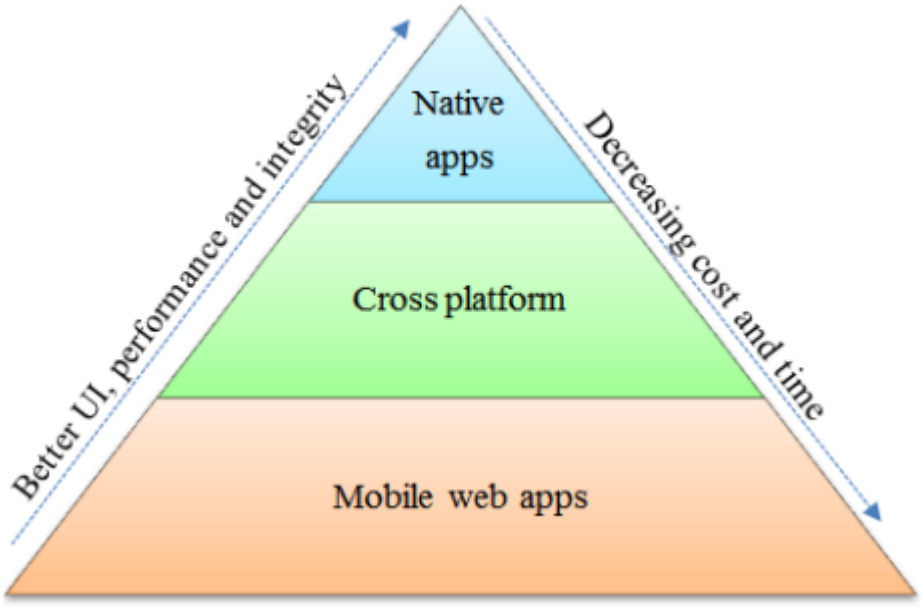
	Abordagem Nativa	Abordagem Híbrida	Abordagem Web
Acesso ao dispositivo	Total	Total	Parcial
Velocidade	Muito rápido	Velocidade nativa	Rápido
Custo de desenvolvimento	Caro	Razoável	Razoável
Processo de aprovação	Obrigatória	Baixa sobrecarga	Nenhuma
Qualidade da UX	Excelente	Não tão boa quanto nativo	Muito boa
Qualidade dos apps	Alta	Média para baixo	Média
Segurança	Alta	Não suficiente	Depende da segurança do browser
Usuários em potencial	Limitado a plataforma	Alta, alcança usuários de diferentes palataformas	Máxima, incluindo *smartphones, tablets e* telefones
Acesso a características específicas do dispositivo	Alta	Média	Baixa
Linguagem de desenvolvimento	Apenas nativa	Nativa e web ou apenas web	Apenas web
Conhecimento/ Ferramentas requeridas para desenvolviemnto	Objective-C/Swift, C#, Java/Kotlin	HTML, CSS, Javascript, Dart e/ou frameworks (como PhoneGap)	HTML, CSS, Javascript
Loja de aplicativos	Sim	Sim	Não

Fonte: Adaptado de LACHGAR e ABDALI (2017, p. 3)

Cada plataforma usa diferentes ferramentas, linguagens de programação, declarações de interface de usuário e gerenciamento de memória. Se um desenvolvedor quiser criar um aplicativo para cada plataforma, ele terá que reescrever todo o código em linguagens e APIs diferentes, uma para cada plataforma. Finalmente, ele terá que testar o aplicativo pelo menos um dispositivo para cada tipo de plataforma (ou até mesmo vários dispositivos com diferentes versões do sistema operacional). As etapas de design e implementação do aplicativo são as duas fases críticas para criar um aplicativo cross-plataforma. Na verdade, o aplicativo deve executar as mesmas funções em todas as plataforma (Perchat, 2013).

De acordo com um estudo realizado por Lahgar et. al(2017), a aplicação nativa acabou tendo melhor desempenho em comparação com aplicativos Web e híbrido. Aplicativos nativos são desenvolvidos usando uma API específica da plataforma e compilados para serem executados na plataforma, em vez de um interpretador de código da linguagem, como JavaScript. Mas o problema é que os aplicativos nativos são mais caros de implementar, são limitados a uma plataforma móvel particular, e precisam ser implementados em linguagens suportadas pela plataforma nativa. A Figura 2 compara o desenvolvimento nativo com o *cross-platform* e com aplicativos Web em relação aos fatores custo e tempo de desenvolvimento.

Figura 2 - Desenvolvimento Nativo vs. Cross vs. Web

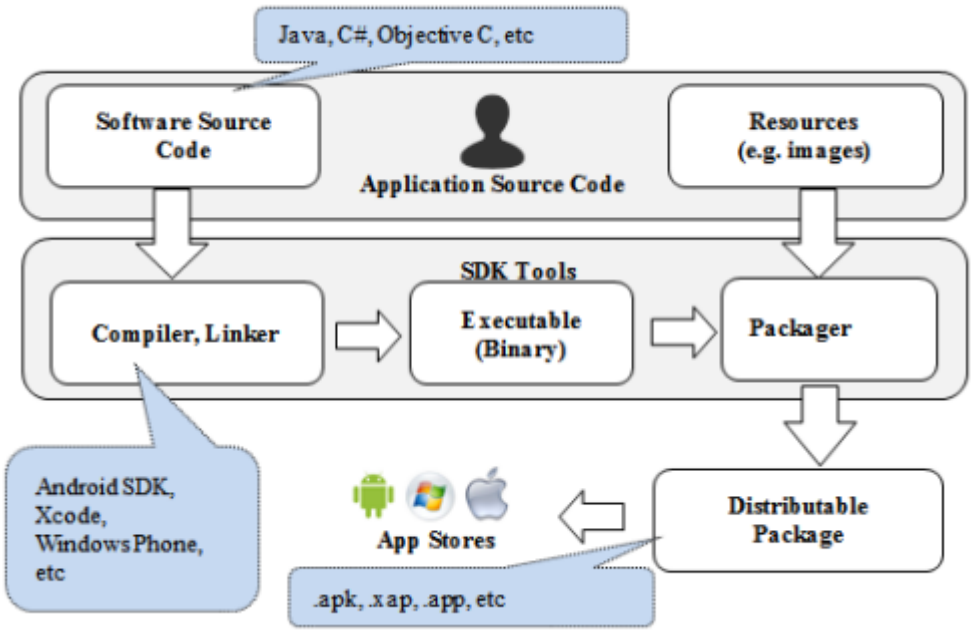


Fonte: LACHGAR e ABDALI (2017)

2.5 **DESENVOLVIMENTO NATIVO**

Aplicativos nativos referem-se a aplicativos que são escritos e desenvolvidos para um sistema operacional de dispositivo móvel específico. Os dois principais sistemas operacionais para dispositivos móveis são o Android, do Google, e o iOS, da Apple. Para escrever aplicativos nativos, devem ser usadas as linguagens Java ou Kotlin, no caso do Android, e Objective C ou Swift, no caso do iOS. As principais características das aplicações nativas são o acesso irrestrito ao hardware do dispositivo e o suporte a todas as interfaces de programação disponíveis no respectivo sistema operacional como ilustra a figura 3 de arquitetura de um aplicativo.

Figura 3 - Arquitetura de uma nativa



Fonte: LACHGAR e ABDALI (2017)

As aplicações nativas têm melhor desempenho, aparência nativa e acesso total aos recursos do dispositivo. Elas usam os recursos de hardware mais atualizados, a fim de melhorar o desempenho. As aplicações são construídas em linguagens que a plataforma suporta, como conseqüência, permitem o uso das melhores ferramentas para o desenvolvimento (IDEs), bem como uma rápida depuração do projeto. Aplicativos para Android podem ser criados no Android Studio e aplicativos para iOS podem ser criados no XCode, que tem todas as ferramentas para depurar ou para projetar as interfaces e, em seguida, verificar sua execução em emuladores ou no próprio dispositivo. No entanto, o desenvolvimento do aplicativo

nativo precisa tempo inicial para aprender as linguagens e ferramentas específicas da plataforma e, em seguida, desenvolver o aplicativo. Além disso, o aplicativo funcionará em apenas uma plataforma específica.

2.6 **DESENVOLVIMENTO HÍBRIDO / CROSS-PLATFORM**

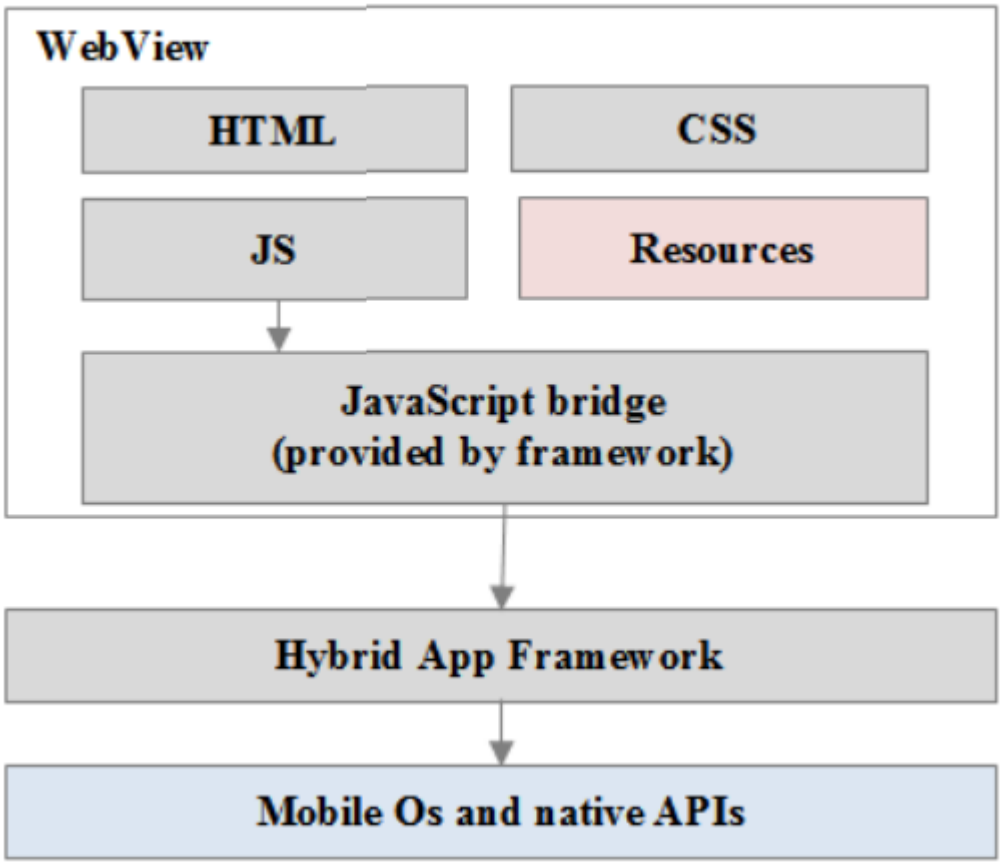
Um aplicativo da web híbrido é um aplicativo que não é verdadeiramente um aplicativo da web para dispositivos móveis nem um aplicativo nativo. É basicamente um aplicação escrita com HTML5, JavaScript APIs e CSS, mas executada dentro de um contêiner de aplicativo nativo de terceiros. As características-chave de um aplicativo híbrido são que eles são desenvolvidos com linguagens padrão da Web, mas geralmente têm acesso às APIs e ao hardware do dispositivo nativo. Alguns dos *frameworks* móveis híbridos bem conhecidos e usados são PhoneGap, Appcelerator[15]e Appspresso[16].

Segundo Heitkotter (2013) as abordagens de desenvolvimento entre plataformas surgiram para enfrentar esse desafio, permitindo que os desenvolvedores implementassem seus aplicativos em apenas uma etapa para uma variedade de plataformas, evitando repetição e aumentando a produtividade. Por um lado, essas abordagens precisam oferecer generalidade adequada para permitir o fornecimento de aplicativos para várias plataformas. Por outro lado, eles ainda têm que permitir que os desenvolvedores aproveitem as vantagens específicas e possibilidades de *smartphones*.

Os aplicativos híbridos para celular são uma solução intermediária entre o aplicativo da web e o aplicativo nativo. Esta abordagem não funciona tão bem quanto os aplicativos codificados em linguagens nativas. Mesmo que eles sejam empacotados como uma aplicação nativa, eles não são aplicativos nativos. Eles são executado em plataformas como o Webkit, em caso de Android e iOS, que é outra camada entre o usuário e o aplicativo e, assim, o desempenho não será o mesmo de aplicativos nativos (Palmier 2012 apud Lahgar, 2017). A figura 4 demonstra uma arquitetura alto nível de aplicativos cross-plataform:

15 https://www.appcelerator.com/
16 https://github.com/kthcorp/Appspresso-SDK

Figura 4 - Arquitetura de uma aplicação híbrida



Fonte: LACHGAR e ABDALI (2017)

2.7 **APLICATIVOS WEB**

Uma aplicação web móvel genérica é outro termo para versões móveis de sites. Existem várias maneiras para criar e desenvolver tecnicamente versões móveis, no entanto, a premissa usual é identificar se a requisição recebida pelo servidor Web no qual a aplicação está hospedada partiu de um dispositivo móvel. Quando um dispositivo móvel é detectado, o usuário é redirecionado para um servidor dedicado, criado para esse dispositivo específico, ou para um site que utiliza técnicas de web design responsivo que irá fornecer o mesmo conteúdo a uma variedade de dispositivos.

Segundo Corral (2012), os aplicativos da Web para dispositivos móveis são desenvolvidos usando os padrões de arquitetura da web conforme ilustra figura 5, ou seja, normalmente HTML5, JavaScript e CSS. Estes aplicativos são fáceis de desenvolver, embora tenham restrições para usar recursos de hardware, como câmera ou sensor de GPS, e aparência distinta da de um aplicativo nativo.

3 **ESTADO DA ARTE**

Este capítulo tem como finalidade identificar o estado atual em que se encontram pesquisas e ferramentas comerciais e gratuitas relacionadas ao desenvolvimento de aplicativos móveis *cross-plataform,* assim como ferramentas denominadas *app builders* com o intuito de analisar o posicionamento desta ferramenta com as outras aplicações existentes.

3.1 **DEFINIÇÃO DO ESTUDO**

Para realizar a pesquisa sobre as ferramentas e trabalhos relacionados com este projeto foram utilizadas diversas bases, e também a própria Web, a fim de encontrar ferramentas comerciais disponíveis e artigos relacionados às tecnologias empregadas no presente trabalho. Dentre as bases utilizadas estão:

- Research Gate[17].
- IEEE Explore[18].
- ACM Digital Library[19].
- Google[20].
- Medium[21].

Foram utilizadas diversas *strings* de pesquisa devido ao alto número de tecnologias e bibliotecas utilizadas neste projeto. Para seleção dos trabalhos e ferramentas foi utilizado um critério de semelhança tecnológica e funcionalidades das aplicações. A tabela 4 exibe alguns dos termos pesquisados para realização do estudo.

Tabela 4 - String de Busca

Termo	Sinônimo	Termo em Inglês
Desenvolvimento aplicativos móveis multiplataforma	Desenvolvimento apps multiplataform	Cross platform development
Desenvolvimento aplicativos híbridos	Desenvolvimento apps hibridos	Hybrid apps development
Geração de código de interface		Interface code generation
Editores/Criadores de aplicativos		Apps creators/editors

Fonte: O autor (2019)

17 https://www.researchgate.net/
18 https://ieeexplore.ieee.org
19 https://dl.acm.org/
20 https://www.google.com
21 https://medium.com/

3.2 **TRABALHOS RELACIONADOS E FERRAMENTAS DISPONÍVEIS**

3.2.1 **APP INVENTOR**

O MIT App Inventor é um ambiente baseado em blocos para criar aplicativos móveis para Android. Segundo Turbak, Wolber e I Medlock-Walton (2014) Um projeto do App Inventor consiste em um conjunto de componentes e um programa especificando seu comportamento. Os componentes incluem itens visíveis da interface do usuário (por exemplo, botões, imagens e caixas de texto) e itens não visíveis usados no aplicativo (por exemplo, câmera, reconhecedor de fala, sensor GPS). O programa está escrito em uma linguagem de blocos. Lançado em dezembro de 2013, o editor de blocos é executado em um navegador da web como um programa JavaScript com base no quadro de blocos Blockly[22].

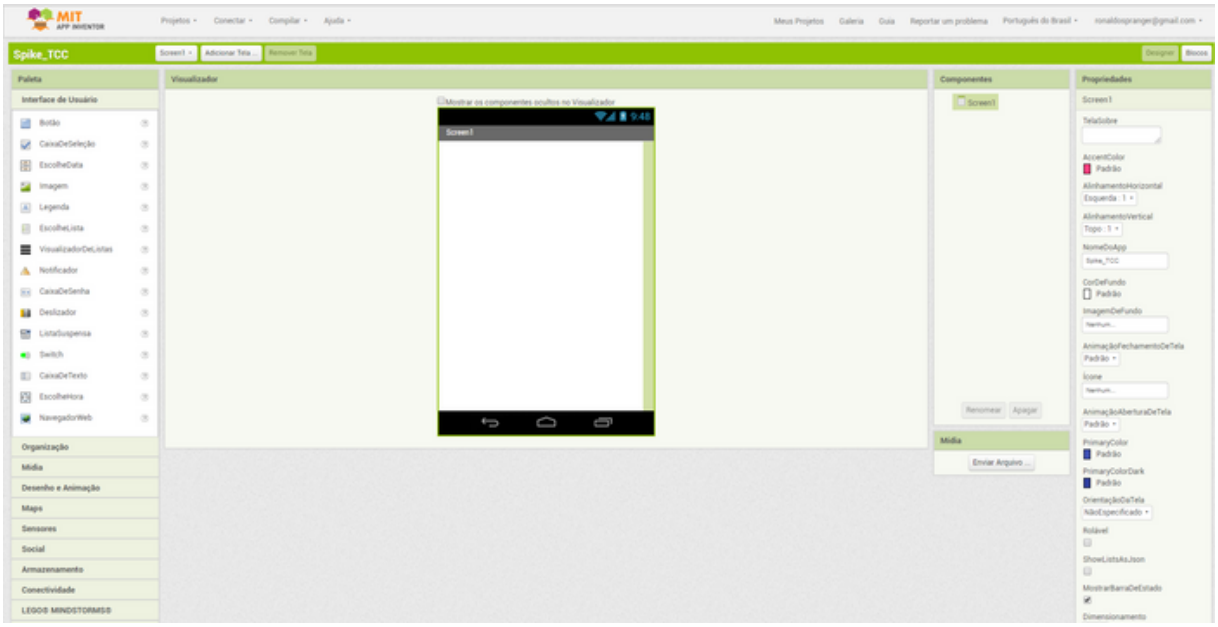
Usuários do App Inventor não precisam escrever códigos de computação tradicionais; eles desenvolvem suas produções apenas arrastando componentes e elaborando sua parte lógica através da união de blocos coloridos já codificados (HARDESTY, 2010).

Segundo Crawford Pokress e Dominguez Veiga (2013), os aplicativos do App Inventor só são executados no Android. Concentrando-se em um único sistema operacional oferece mais funcionalidade ao usuário final do que seria possível com uma solução multi-plataforma. Por exemplo, App Inventor está ligado ao sistema operacional Android de tal forma que pode enviar e receber informações de muitos dos componentes do telefone (GPS, NFC, mensagens de texto, câmera, acelerômetro, etc.). Uma ferramenta *cross-platform* não seria possível em todos os dispositivos por causa das diferentes formas em que os sistemas operacionais chamam essas funcionalidades. Além disso, é importante que os alunos possam compartilhar seus aplicativos em um local publicamente acessível sem ter que comprar uma licença de desenvolvedor. O Google Play Store permite que os aplicativos do Inventor sejam disponibilizados para download e instalação.

O App Inventor possui uma interface de arrastar e soltar componentes. Na lateral esquerda, conforme demonstra a figura 7, fica uma árvore de componentes e recursos que podem ser utilizados para criação do aplicativo. Na parte central fica o *template* no qual serão colocados os componentes selecionados. Já na parte direita da tela, ficam as propriedades do componente ou recurso selecionado pelo usuário, podendo mudar a cor, alterar tamanho, posição, etc.

22 https://developers.google.com/blockly/

Figura 5 - App Inventor - Tela inicial



Fonte: O autor (2019)

Cada componente possui seus respectivos blocos de programação, permitindo ao usuário inserir novos blocos junto aos que estão por padrão no componente. A figura 8 apresentar os blocos selecionáveis de programação.

Figura 6 - App Inventor - Bloco de programação

Fonte: O autor (2019)

O App Inventor é um projeto de código aberto. Qualquer um pode baixar o código-fonte para executar em seus próprios servidores, podendo ainda modificar, personalizar e estender cada funcionalidade do sistema. O projeto é distribuído sob a licença do MIT[23], que é permissivo em relação a modificações e comercialização. MIT disponibilizou um conjunto de documentos sobre como estender o sistema, e membros da equipe de desenvolvimento geralmente realizam sessões de vídeo remotas em que eles apresentam sobre temas técnicos e em que comunidade os membros são convidados a exibir seu trabalho. Membros da comunidade *open-source* de desenvolvedores fizeram muitas contribuições para o projeto, desde pequenas correções de *bugs* até componentes completos.

3.2.2 **SEATTLE CLOUDS**

Com mais de novecentos mil aplicativos criados em sua plataforma e grandes parceiros como IBM e Microsoft, a Seattle Clouds[24] é a mais completa ferramenta paga para criação de aplicativos Android e iOS. Altamente configurável e com uma usabilidade simples e intuitiva, a ferramenta dispõe de diversos recursos para criação de aplicativos. Existe uma série de funcionalidades implementadas que facilitam a vida do usuário, entre elas:

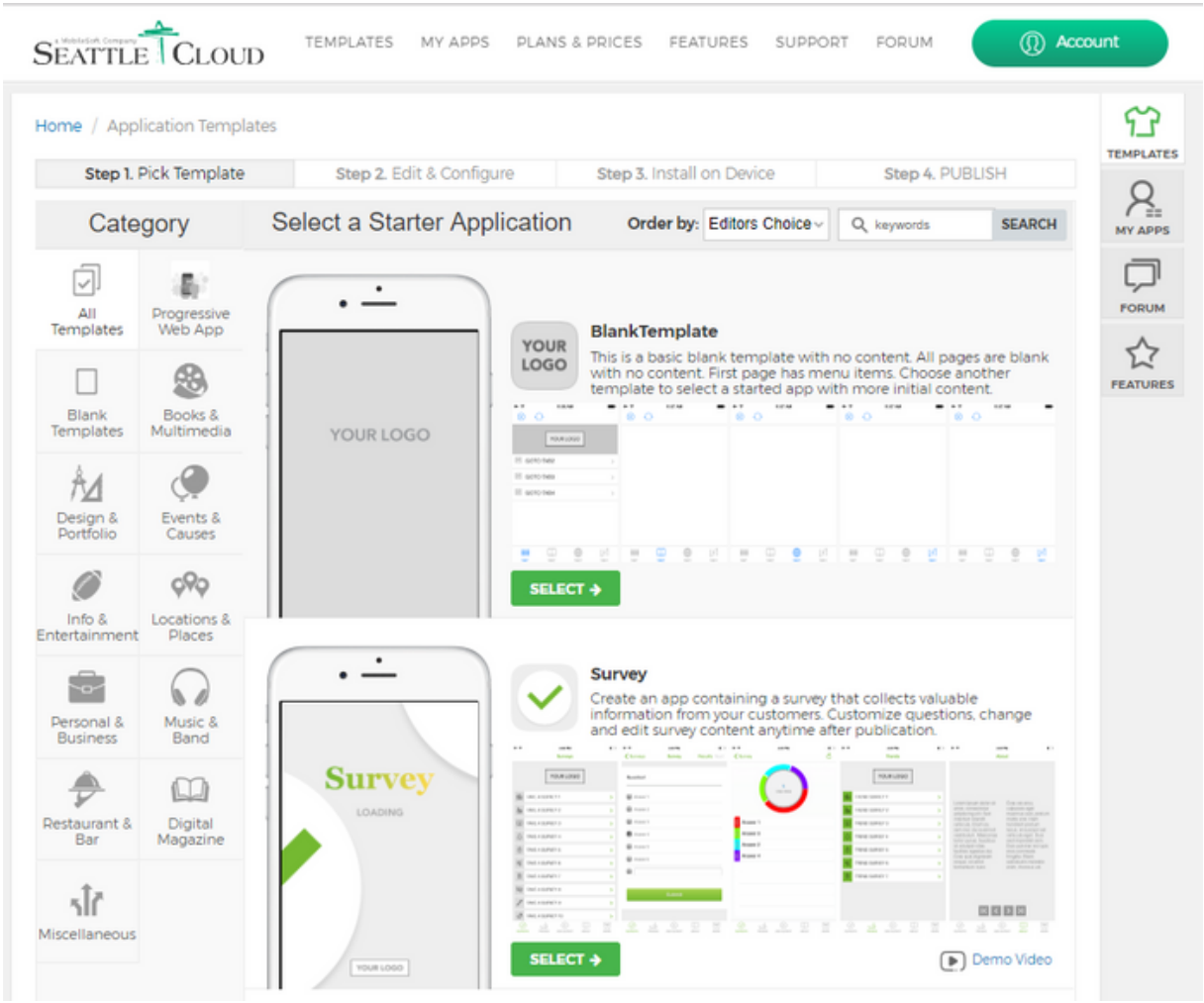
- Menu de imagem em GRID
- Assinatura e compras no aplicativo
- GPS e direções
- Mosaico de imagens
- Formulário de Comentários
- Notificações via push
- Leitor de feeds RSS
- Menu de Estilo do Botão
- Menu de ícones grandes
- Banca de jornais
- Compartilhar no Facebook
- Carrinho de compras
- Integração no Twitter
- Vídeo do youtube
- Favoritos
- Votação
- Check-in
- Efeito foto

23 https://opensource.org/licenses/MIT
24 https://seattleclouds.com

- Cupons

Na figura 9 pode-se observar uma série de grupos de temas para usar como base de modelo para o aplicativo selecionado.

Figura 7 - Seattle Clouds seleção de templates



Fonte: O autor (2019)

O Seattle Clouds também fornece muitos tipos diferentes de modelos de interface que o usuário pode adicionar a qualquer aplicativo. O utilizador pode testá-los diretamente no navegador web e verificar qual deles é mais fiel a interface que o usuário gostaria que a aplicação se parecesse.

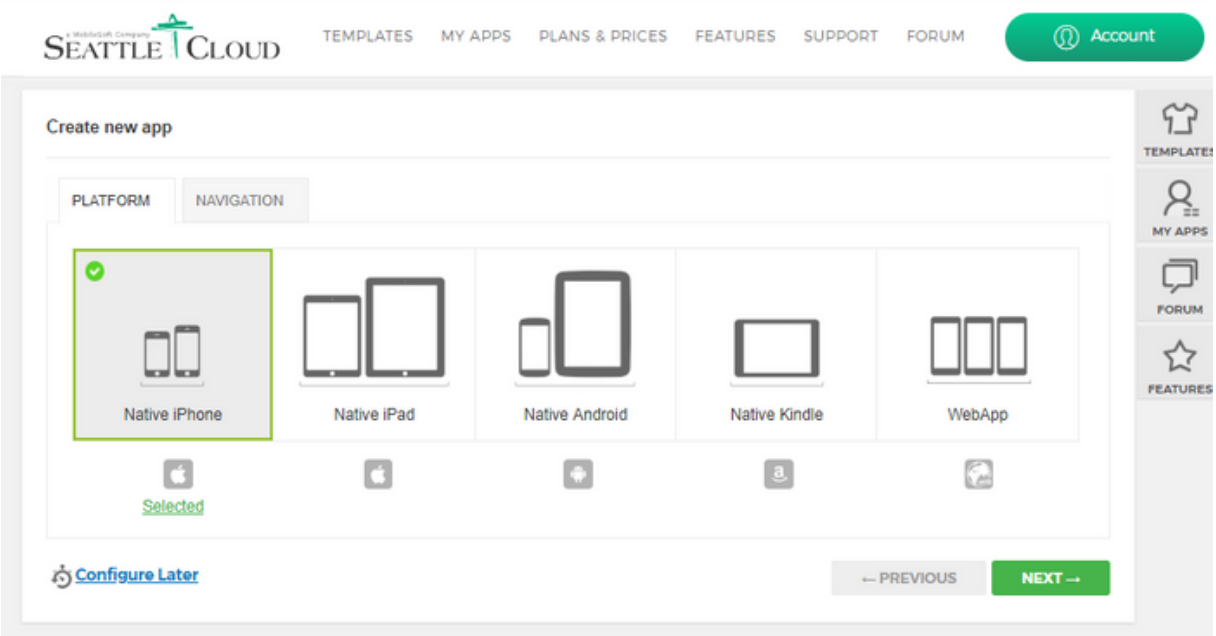
Todos os modelos de aplicativo contêm conteúdo de amostra e são fornecidos apenas para fins ilustrativos. Todas as imagens, links, planos de fundo e texto podem ser removidos e substituídos posteriormente pelo conteúdo a ser colocado na aplicação.

A desvantagem do Seattle Clouds é que o usuário não tem acesso ao código gerado, limitando toda a aplicação ao conteúdo disponibilizado pela própria

empresa.

A figura 10 demostra a seleção de plataforma para o projeto, porém todo código gerado é um *webapp* gerado em HTML em código híbrido, mesmo quando selecionado modelo de criação nativo para plataforma selecionada. Outra característica bem comum às aplicações híbridas é o limitado acesso de recursos nativos dos *smartphones*, como acelerômetro, biometria, etc.

Figura 8 - Seattle Clouds, seleção de plataformas



Fonte: O autor (2019)

3.2.3 **THUNKABLE**

O Thunkable é um construtor *drag and drop* para criação de aplicativos Android com aparência nativa e recursos interativos. Tudo é feito através de uma interface visual com os componentes e conexões disponíveis.

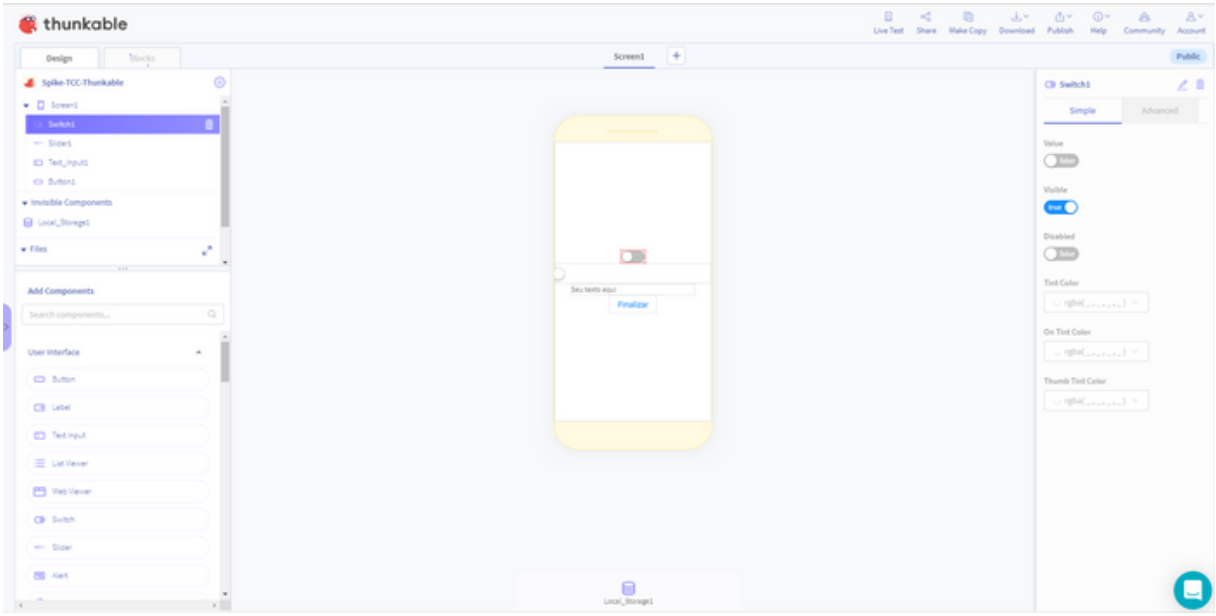
Este editor possui dois componentes principais: Designer e Blocos. No primeiro é possível criar toda a Interface de Usuário adicionando os componentes como botões, blocos de textos e elementos interativos como o Google Maps.

No Blocos fica toda a inteligência da aplicação, com a utilização de uma linguagem baseada em blocos muito semelhante ao já citado App Inventor. A ideia é facilitar a utilização de programadores iniciantes, mas ao mesmo tempo ser sofisticado o bastante para desenvolvedores experientes usarem. Tudo esta lá, funções, variáveis, *callbacks,* etc. Algumas das mais importantes características da ferramenta são as seguintes:

- Desenvolvimento *cross-plataform;*
- Programação em blocos;
- *Deploy* facilitado;
- Acesso de recursos, como mapas e outros;
- *Live debug.*

Thunkable possui uma interface fácil e intuitiva, não sendo necessária a leitura de documentação para utilizar a ferramenta. A Figura 11 mostra alguns componentes disponíveis para uso na ferramenta.

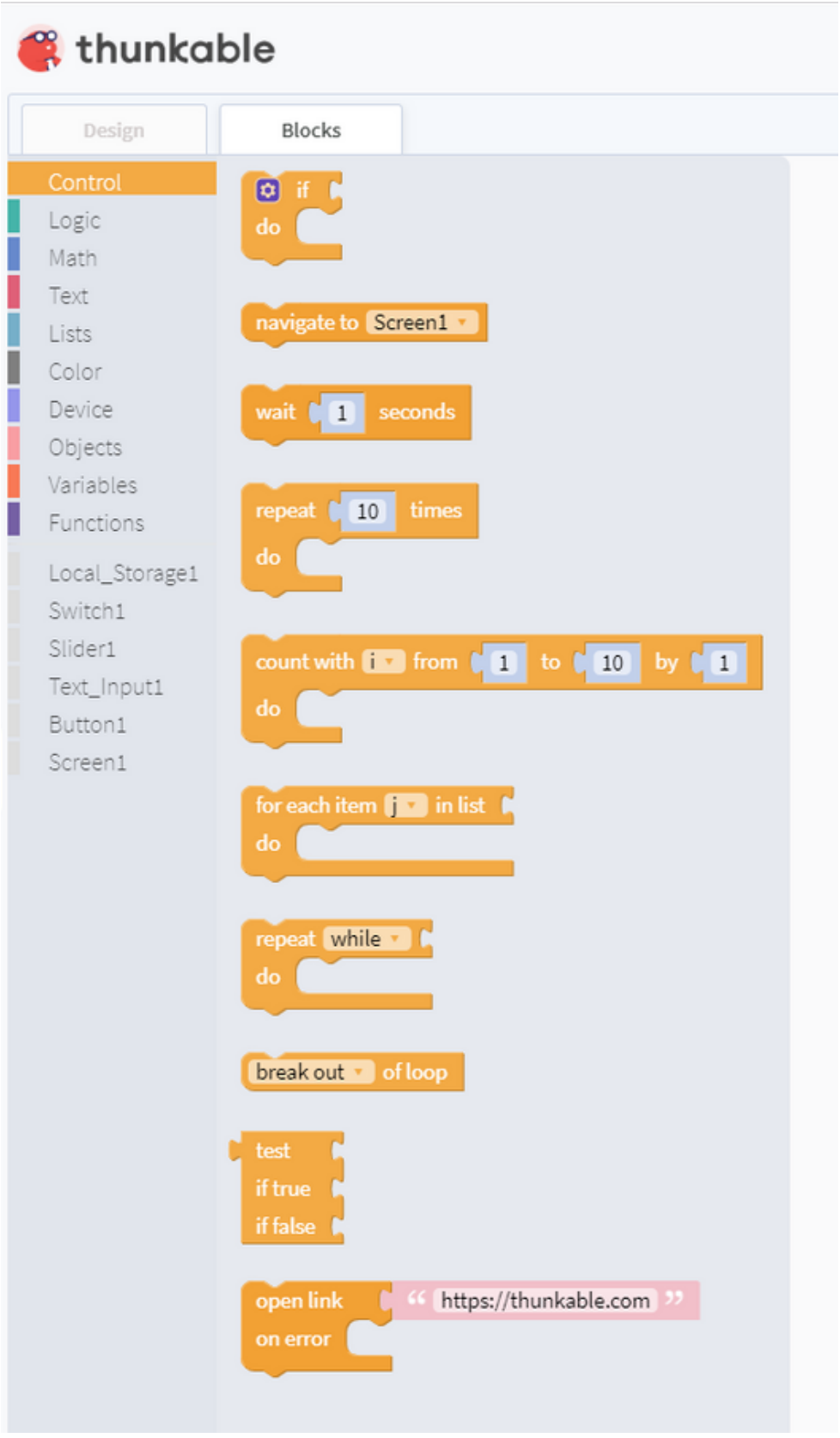
Figura 9 - Thunkable spike



Fonte: O autor (2019)

Na Figura 12 é mostrado o código em blocos gerados por esta aplicação, que podem ser editados para adicionar novas funcionalidades. Semelhante ao modelo do *App Inventor.*

Figura 10 - Thunkable code-blocks



Fonte: O autor (2019)

O Thunkable é uma ótima alternativa mais profissional que o App Inventor,

tanto para criar aplicativos sem programação, quanto para aprender os conceitos de programação de maneira interativa e fácil. Porém ainda sofre dos mesmos problemas de aplicativos baseados em web, com limitado acesso aos recursos do smartphone.

3.2.3.1 **REACT NATIVE**

O *React Native* foi desenvolvido pelo *Facebook*, lançado pela primeira vez em 2015, e agora é um projeto de código aberto no *GitHub*[25]. O *React Native* permite usar o melhor do *React* (como *re-builds* rápidos e um DOM declarativo) e ser capaz de executar uma instância do JavaScriptCore[26] dentro do aplicativo porque criam componentes específicos da plataforma (OCCHINO, 2015). A lógica da aplicação é escrita e executada em JavaScript, enquanto a interface do aplicativo é totalmente nativa. Segundo Axelsson e Carlström (2016) com o *React Native* pode-se usar JavaScript e implementação nativa, fazendo a mudança para um desenvolvimento multi-plataforma menos custoso para projetos existentes. O *React Native* permite também, se uma funcionalidade não existir no *React Native framework*, use código nativo, para que o usuário tenha a melhor experiência possível. *React Native* não tem como objetivo ser uma ferramenta "escreva uma vez, execute em qualquer lugar". Está com o objetivo de seja "aprenda uma vez, escreva em qualquer lugar". Android e iOS são dois sistemas diferentes e devem ser tratados de maneiras diferentes, portanto, aplicativos para essas plataformas não devem ser projetados da mesma maneira. Existem diferenças no componentes e características nativas especiais que tornam a experiência melhor e única. Ao aprender a sintaxe e a estrutura do *React Native*, o desenvolvedor poderá criar aplicativos para diferentes plataformas sem a necessidade de aprender um novo conjunto de técnicas. Com o mesmo código e tecnologias, um desenvolvedor será capaz de desenvolver aplicativos para Android e iOS.

3.2.3.2 **PHONEGAP/ APACHE CORDOVA**

O Adobe PhoneGap[27] é uma estrutura de código aberto para a criação de aplicativos móveis usando HTML, CSS e Javascript para iOS e Android. PhoneGap é perfeito para transformar um aplicativo da web móvel em um aplicativo nativo. É fácil de usar para desenvolvedores da web. Para usar o PhoneGap, um desenvolvedor da Web precisará aprender a construir usando um ou mais SDKs e

25 https://github.com/facebook/react-native
26 https://developer.apple.com/documentation/javascriptcore
27 http://phonegap.com/

ferramentas de dispositivos, mas todo o código do aplicativo pode ser HTML, CSS e JavaScript. Segundo Allen (2010) , um desenvolvedor deve ser bastante experiente em JavaScript para aproveitar esta plataforma. Dependendo da perspectiva do desenvolvedor, é um benefício ou uma desvantagem de fornecer pouco em termos de padrões de design para aplicativos móveis.

O PhoneGap fornece uma rica coleção de APIs JavaScript do lado do cliente com um método para hospedar seu aplicativo da web dentro de um aplicativo móvel nativo. PhoneGap é um projeto patrocinado da Nitobi (http://nitobi.com) adquirido pela Adobe em 2011[28] que renomeou para Apache Cordova[29] tornando o Cordova uma estrutura de código aberto e o PhoneGap uma ferramenta comercial mantida pela Adobe. A estrutura começou em 2008 e é gratuita para uso sob um licença MIT.

3.2.3.3 **ANDROMO**

Andromo[30] é um *framework* de desenvolvimento de apps baseado em Android. É um serviço pago com diferentes planos de assinatura. Andromo permite criar aplicações muito agradáveis visualmente que usam os desenhos de interfaces do Android, no qual, se baseia nos padrões de desenvolvimento de interface[31] para Android. Os padrões estão presentes em todas as funcionalidades da aplicação pela qual permite criação de interfaces com alta usabilidade e de acordo com os padrões estabelecidos.

Andromo é todo baseado na web, não necessitando nada mais que um browser para rodar a aplicação, sem necessidade de instalação ou configuração adicional. A interface do Andromo está dividida em diferentes telas em que cada tela representa uma etapa no desenvolvimento do aplicativo Android. O *framework* permite a criação de diversas *Activities* que permitem criar diferentes funcionalidades para os aplicativos, conectar com diversas APIs externas e também facilidade de publicação do aplicativo na Play Store.

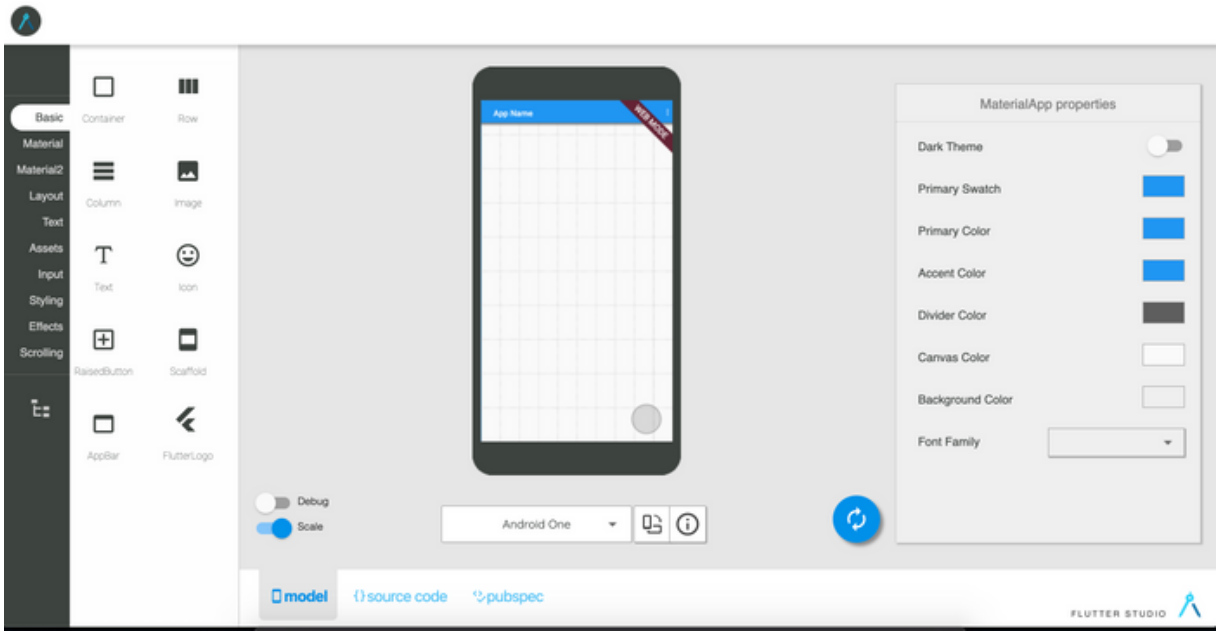
3.2.3.4 **FLUTTER STUDIO**

O *Flutter Studio*[32]*.* é uma ferramenta comunitária baseada na web para a criação de telas Flutter com o objetivo de facilitar o desenvolvimento de GUIs. O usuário tem acesso ao modo visual (model) onde pode selecionar os

28 https://techcrunch.com/2011/10/03/adobe-acquires-developer-of-html5-mobile-app-framework-phonegap-nitobi/
29 https://cordova.apache.org/
30 https://www.andromo.com/en/
31 https://developer.android.com/design
32 https://flutterstudio.app/

widgets/contêineres dos menus arrastando e soltando no canvas do dispositivo. a ferramenta permite selecionar o item que deseja usar para criar o *layout*, arrastar para a interface e soltar, a seguir fazer os ajustes usando os recursos da ferramenta. A figura 13 representa a tela inicial da ferramenta.

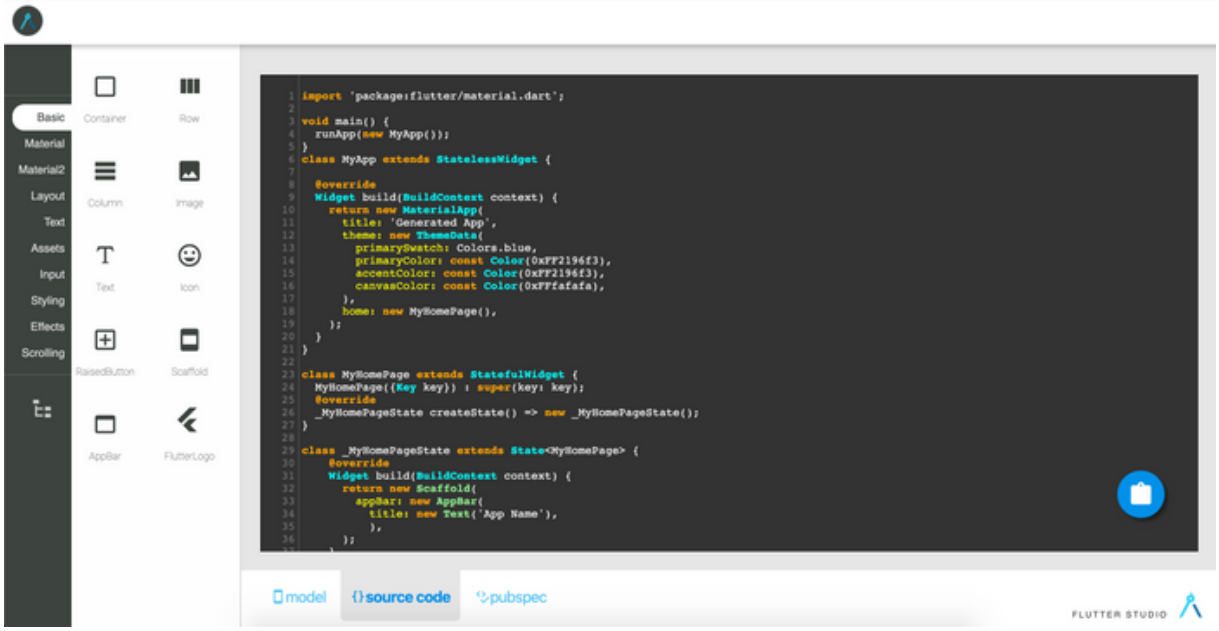
Figura 11 - Tela inicial flutter studio



Fonte: O autor (2019)

Ao clicar em um widget na tela (ou na árvore) para selecioná-lo. Os componentes selecionados são cercados por uma caixa delimitadora verde. Pode-se usar os editores para alterar suas cores, preenchimento, alinhamento, fonte, etc. O código é atualizado enquanto o usuário interage com os componentes na interface. Ao clicar na guia *Source* o usuário pode visualizar o código *Flutter* gerado. Na figura 14 é possível ver o código gerado pela edição dos componentes.

Figura 12 - Visualização do código da ferramenta



Fonte: O autor (2019)

A ferramenta *Flutter Studio* tem a proposta de ser um gerador de interface de aplicativos *Flutter*. Entre as funcionalidades da ferramenta estão:

- É responsiva.
- Mostra dispositivos Android e iOS reais com precisão na Web.
- Produz e mostra designs perfeitos de pixels.
- Fornece mais widgets (incluindo temas).
- Permite editar os widgets intuitivamente, produzindo código preciso.
- Concentra no design.

Há uma série de limitações na ferramentas que dificultam a utlização da ferramenta para criar aplicações *cross-plataform,* dentre elas estão:

- Nenhuma documentação foi encontrada além dos artigos já citados.
- Pouca referência, existem apenas 2 artigos do autor da ferramenta falando sobre a aplicação.
- Funcionou corretamente somente no navegador *Google Chrome*[33]*. Mozilla Firefox*[34]e *Opera*[35]apresentaram falhas.
- Sobreposição de componentes quando não arrastados de maneira que a ferramenta espera.
- Impossibilidade de salvar as interfaces geradas.
- Impossibilidade de baixar o código gerado pela ferramenta
- Código fechado.

33 https://www.google.com/intl/pt-BR/chrome/
34 https://www.mozilla.org/pt-BR/firefox/
35 https://www.opera.com/pt-br

- Falta de atualização, a última vez que a ferramenta foi atualizada foi em maio de 2018[36].

Flutter Studio destaca-se pela simplicidade e facilidade de uso, mas exige do usuário conhecimento prévio em edição de *layouts* em Flutter. A falta de documentação e código fechado não permite aos usuários proporem melhores ou até mesmo corrigir possíveis falhas de implementação da aplicação.

3.3 **ANALISE COMPARATIVA**

Nesta sessão é realizada uma comparação entre as ferramentas pesquisadas na sessão anterior, demonstrando a necessidade de que seja proposta uma nova ferramenta, que reúna um conjunto de características que seja mais apropriado para o desenvolvimento de *frontends* de aplicações móveis. Para realizar a análise, foram vistos pontos relevantes quanto ao desenvolvimento e uso da ferramenta. Estes elementos podem facilitar ou dificultar o uso para desenvolvedores menos experientes e também como ferramenta de ensino. A Tabela 4 ilustra as diferenças entra as ferramentas analisadas.

Tabela 5 - Tabela comparativa entre ferramentas

Ferramenta	Código-aberto	Pago	Tipo de desenvolvimento	Multiplataforma	Preço anual (R$)*	Idioma
AppInventor	Sim	Não	Híbrido	Não	--	Português / Inglês
Seattle Clouds	Não	Sim	Nativo	Sim	1619,00	Inglês
Thunkable	Não	Sim	WebApps	Sim	820,00	Inglês
Andromo	Não	Sim	Híbrido	Não	1033.00	Inglês
Flutter Studio	Não	Não	Nativo/Híbrido	Sim	--	Inglês

Fonte: O autor (2019) * Valor dolar outubro 2019 .

- **Código-aberto**: Somente o *App Inventor* possui código aberto e livre, tornando-a mais simples de personalizar, adicionar novas funcionalidades e corrigir possíveis falhas.
- **Ferramenta Paga:** Novamente somente o *App Inventor* e o *Flutter Studio* são totalmente livres de pagamentos. As outras três ferramentas possuem versões gratuitas, mas com várias limitações de escopo e desenvolvimento.
- **Tipo de Desenvolvimento:** *Flutter Studio*, *Seattle Clouds* e *Thunkable*

36 https://medium.com/@pmutisya/flutter-studio-version-2-41cce10fcf3d

permitem o desenvolvimento para diferentes plataformas, já as outras duas ferramentas pesquisadas permitem somente a criação de aplicativos para a plataforma *Android.*

- **Preço:** As ferramentas *Seattle Clouds, Thunkable* e *Andromo* possuem valores nada convidativos para o uso de desenvolvedores iniciantes e para o uso educacional. Já as licenças destas ferramentas são individuais e voltadas para o desenvolvimento comercial. A ferramenta *Thunkable* possui uma versão educacional.
- **Idioma:** Somente o *App Inventor* possui versão em português, tornando-a mais simples para os desenvolvedores brasileiros.

A Tabela 5 faz uma análise mais técnica sobre sobre os recursos e componentes de cada ferramenta.

Tabela 6 - Análise de recursos e componentes

Ferramenta	Edição de código	Inserção de novos componentes	Extensão de componentes	Guia de estilos	Tipo de sintaxe
AppInventor	Sim	Código aberto	Código aberto	Material	Programação em Blocos
Seattle Clouds	Sim	Não possui	Não possui	Material/Cupertino	DSL proprietária
Thunkable	Sim	Não possui	Não possui	Não possui	Programação em Blocos
Andromo	Sim	Não possui	Não possui	Material	Java/ Android
Flutter Studio	Parcial	Não possui	Não possui	Material	Dart/Flutter

Fonte: O autor (2019)

- **Edição de código:** Todas as ferramentas analisadas possuem edição de código, seja por interface proprietária, como é o caso do *Seattle Clouds*, programação em blocos como o *App Inventor* e o *Thunkable*, edição na linguagem de destino, como no *Andromo,* que tem um editor para escrever código em *Java* para *Android,* ou edição após a geração de código. A ferramenta Flutter Studio foi marcada como parcial, pois somente é possível copiar e colocar o código gerado pela interface.
- **Inserção e extensão de componentes:** Somente o *App Inventor* possui essas funcionalidades. Por ser uma ferramentas de código aberto, ela

permitem a personalização da estrutura do projeto.

- **Guia de estilos:** As ferramentas seguem os guias de estilo feitos por *Google* e *Apple* para suas plataformas. Nesse quesito a única ferramenta que suporta os dois guias de estilo é o *Seattle Clouds.*
- **Sintaxe:** *App Inventor* e *Thunkable* têm sintaxe de programação em blocos, mas estas não permitem muitas personalizações. As outras ferramentas pesquisadas e o artefato proposto neste trabalho permitem a edição do código gerado, permitindo assim personalização completa da aplicação gerada.

Concluído o processo de análise comparativa entre todas as soluções pesquisadas, fica demonstrada a necessidade de uma ferramenta gratuita para o desenvolvimento de aplicativos móveis multiplataforma. Apesar da ferramenta Flutter Studio preencher parte dos requisitos de gratuidade e desenvolvimento multiplaforma, suas limitações quanto a recursos e ausência de documentação e escopo(se limita somente a gerar código de interface). A solução proposta neste trabalho se posiciona com recursos de ambas ferrametas, além de possuir grande parte das funcionalidades das plataformas pagas e gratuitas. Possui ainda a vantagem de ser de código aberto, o que permite fácil inclusão de novas funcionalidades pela comunidade ou por seus usuários. O que será demonstrado no próximo capítulo.

4 **A FERRAMENTA PROPOSTA**

4.1 **DESCRIÇÃO**

A ferramenta produto deste trabalho se propõe a criar código-fonte de aplicação mobile híbrida através de um editor web de *drag and drop* no qual componentes são arrastados para uma interface gráfica com a representação da tela de um smartphone. O usuário pode criar diversas interfaces e escolher um modelo de navegação entre elas. Clicando nos componentes adicionados no modelo, o usuário tem a opção selecionar ações deste componente. Por exemplo, ao colocar um campo de texto ele pode nomear o campo para adicioná-lo a uma tabela no banco de dados. Quando selecionar um botão e vincular uma ação, como gravar no banco de dados.

Ao término, o usuário pode gerar o código resultante da sua edição. A aplicação Web cria um projeto com toda a arquitetura padrão do *framework Flutter, ou seja, com tod*a estrutura de pastas e arquivos, assim como os arquivos gerados pelo editor de layout. O usuário, com o código gerado, usa o editor da sua preferência para compilar e rodar o código em sua máquina.

4.2 **LEVANTAMENTO DE REQUISITOS**

Nesta sessão são abordados os requisitos da aplicação, sua arquitetura e protótipos a fim de guiar o desenvolvimento da ferramenta. O levantamento de requistos foi realizado através da análise das aplicações existentes e a documentação exposta nas referidas ferramentas. Os requisitos funcionais foram escritos partindo da premissa dos recursos mínimos necessários para o funcionamento da ferramenta de geração de aplicativos multiplataforma. Os requisitos não funcionais partiram de conhecimento tácito do autor sobre aplicações web.

4.2.1 **REQUISITOS FUNCIONAIS**

RF-1: Selecionar componente: O usuário poderá selecionar e arrastar qualquer componente para a área de edição de template respeitando as margens do template.

RF-2: Incluir scaffold: O usuário poderá incluir ações em trechos de códigos na linguagem DART que depois podem ser utilizados para geração posterior do código-fonte da aplicação.

RF-3: Editar scaffold: O usuário poderá editar os códigos adicionados pela **RF-2**.

RF-4: Excluir scaffold: O usuário poderá excluir os códigos adicionados pela **RF-2**.

RF-5: Atribuir scaffold a componente: O usuário tem a opção de atribuir os trechos de códigos criados pela **RF-2**, editados pela **RF-3,** além dos códigos já armazenados na base de scaffolding nos componentes já incluídos no template.

RF-6: Adicionar imagens: O usuário tem a opção de adicionar imagens que ficarão salvas no banco de dados para que possa usar no template.

RF-7: Salvar Template: O usuário tem a opção de salvar o template como modelo, ou como trabalho. Este requisito servirá tanto para o reuso em novas aplicações como para continuar um trabalho salvo.

RF-8: Excluir template: O usuário poderá editar os códigos adicionados pela **RF-7**.

RF-9: Salvar aplicação: O usuário poderá excluir os códigos adicionados pela **RF-7**.

RF-10: Gerar código: Quando o usuário decidir, ele deve poder gerar o código da aplicação criada no editor. Esta ação gerará uma aplicação em código Dart na framework Flutter com toda a estrutura de arquivos e pastas pronto para a compilação.

4.2.2 **REQUISITOS NÃO FUNCIONAIS**

RNF-1: Aplicacão deve rodar sem nenhum tipo de travamento no Google Chrome em sua versão 78.0.3.

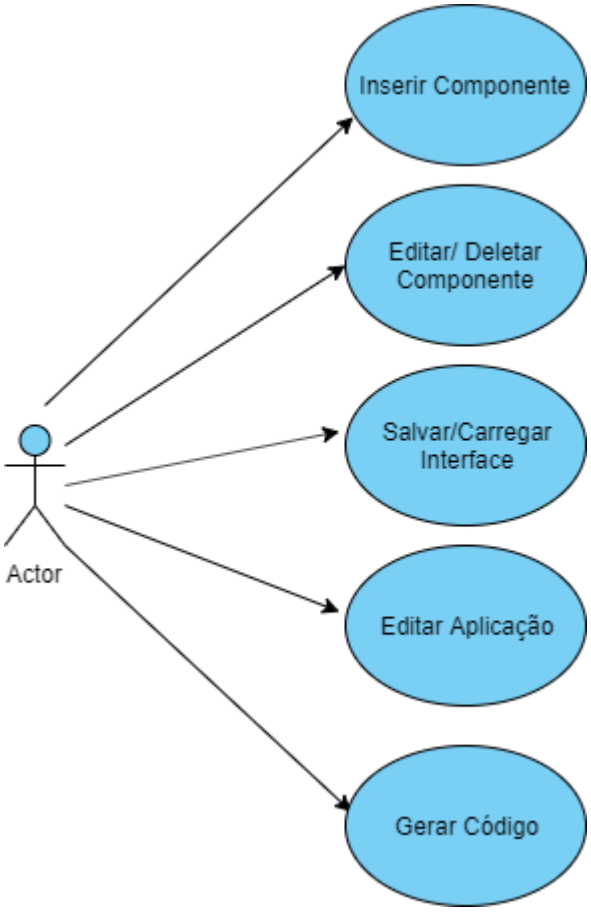
RNF-2: Não deve ser necessária a instalação de plugins.

RNF-3: Não será permitida a transposição de componentes. Componentes não podem ser colocado no mesmo ponto da interface.

4.2.3 **CASOS DE USO**

Foram identificados 5 casos de uso principais da ferramenta através da análise dos recursos mínimos para a geração de código multiplataforma e também do estudo realizado entre as ferramentas semelhantes a proposta. O diagrama 1 demonstra a iteração entre o ator e o sistema.

Diagrama 1 - Diagrama de casos de uso



Fonte: O autor (2019)

UCS-01 - Inserir Componente: O usuário poderá inserir componentes na interface a partir de uma listagem de componentes exibida na tela.

UCS-02 - Editar/ Deletar Componente: Ao selecionar um componente inserido na interface, o usuário poderá editar ou deletar este componente utilizando de uma menu de propriedades.

UCS-03 - Salvar/ Carregar Interface: Será permitido salvar o trabalho realizado e carregar posteriormente para continuação do mesmo.

UCS-04 - Editar Aplicação: Será permitido a edição do nome das interfaces, escolha de temas e estilo a partir de um menu de configuração.

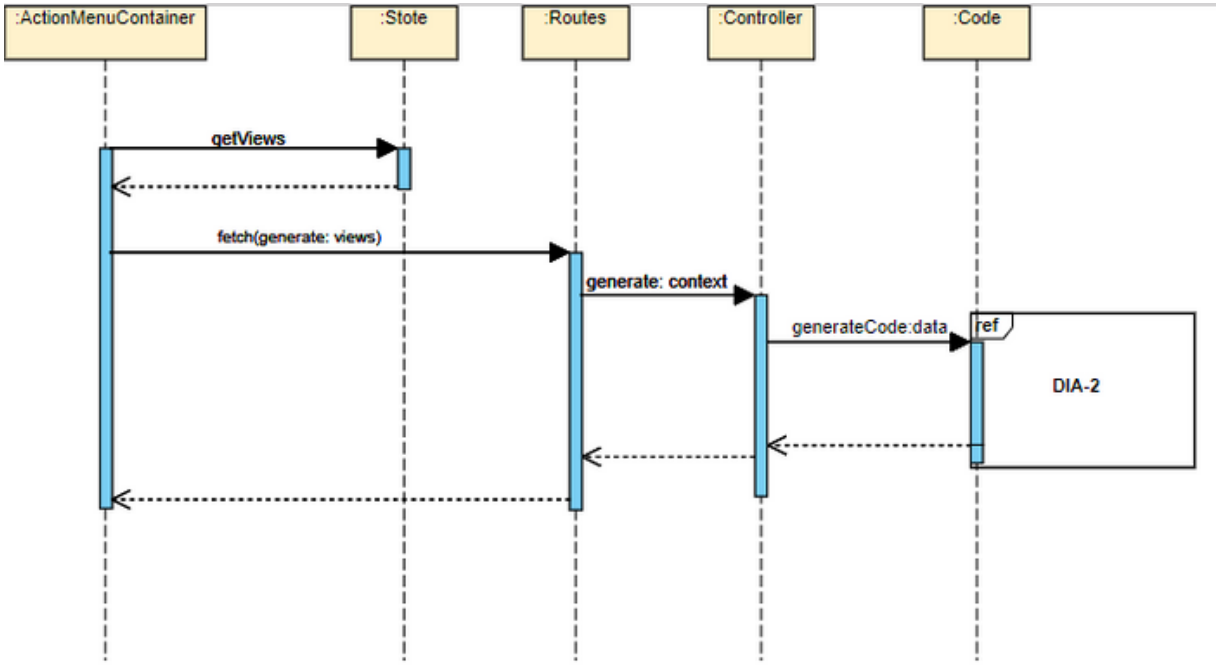
UCS-05 - Gerar Código: Ao usuário clicar em na opção "Gerar código"

4.2.4 **DIAGRAMA DE SEQUÊNCIA**

O caso de uso **UCS-05 - Gerar Código**, devido a sua alta complexidade e iteração entre cliente e servidor, foi constatada a necessidade um diagrama que explique seu fluxo. A sequência de diagramas a seguir ilustra por partes a execução

do **UCS-05.** O Diagrama 2 ilustra o fluxo inicial do *frontend* até a função de geração de código no *backend*. Com o retorno do aplicativo e do código gerado disponibilizando estes para *download*.

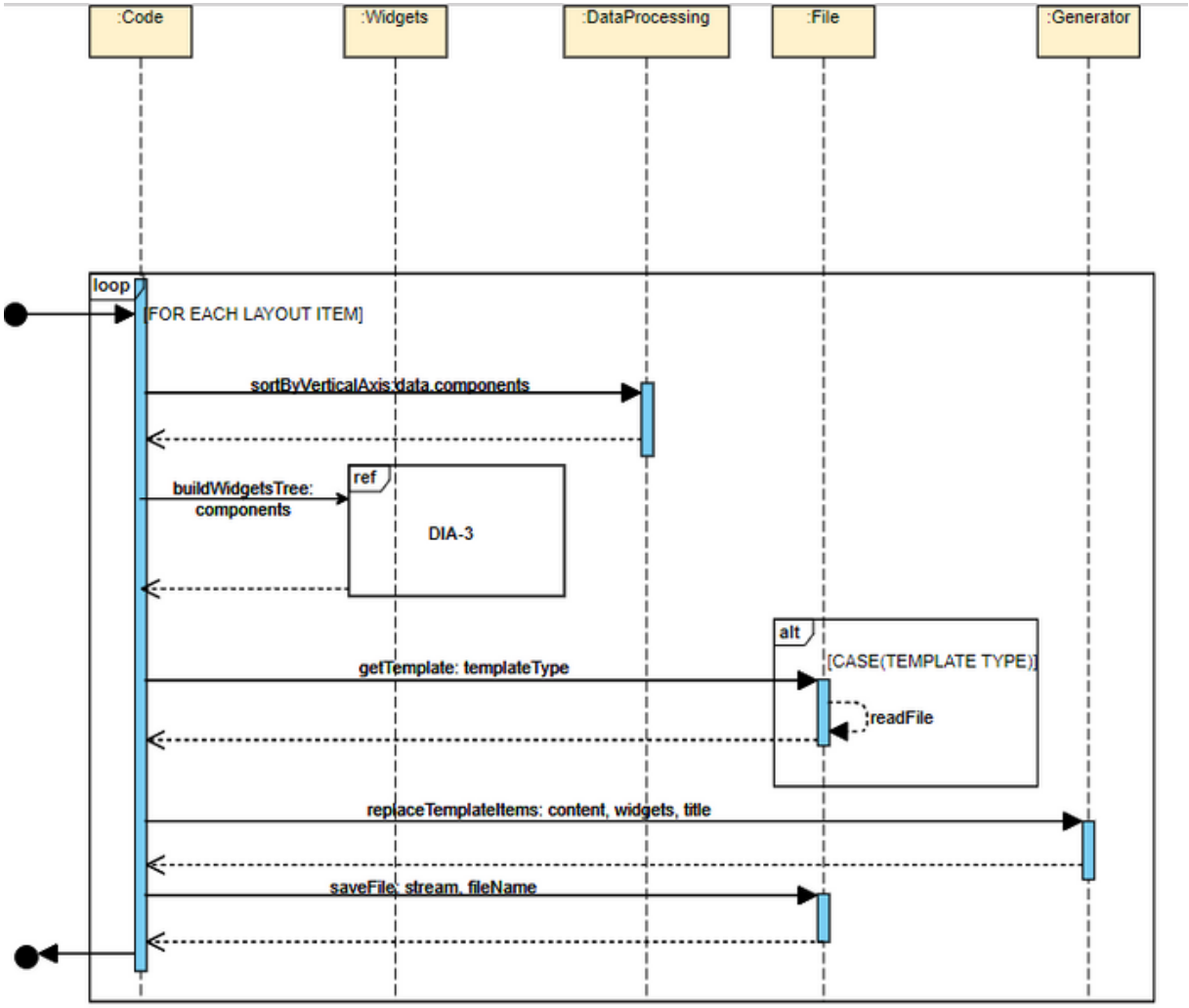
Diagrama 2 - Diagrama de Sequência 1



Fonte: O autor (2019)

O Diagrama 3 representa a continuação do fluxo inicial, desta vez ilustrando a função de geração de código. A função faz uma interação para cada item de interface criado pelo usuário. Na sequência ordena todos os componentes dentro deste layout e envia para o DIA-3 (Diagrama de sequência -3) no qual constrói a árvore de componentes que serão gerados. Após o término da construção da árvore, a função busca o *scaffold* específico para cada componente inserindo-os em um vetor de *scaffold`s.* Depois que o vetor de componentes estiver completo, a função enviar para o gerador que vai substituir as referências no template de arquivo .dart pelo código contido no vetor de componentes. A seguir, a aplicação salva o arquivo no contêiner *Docker.*

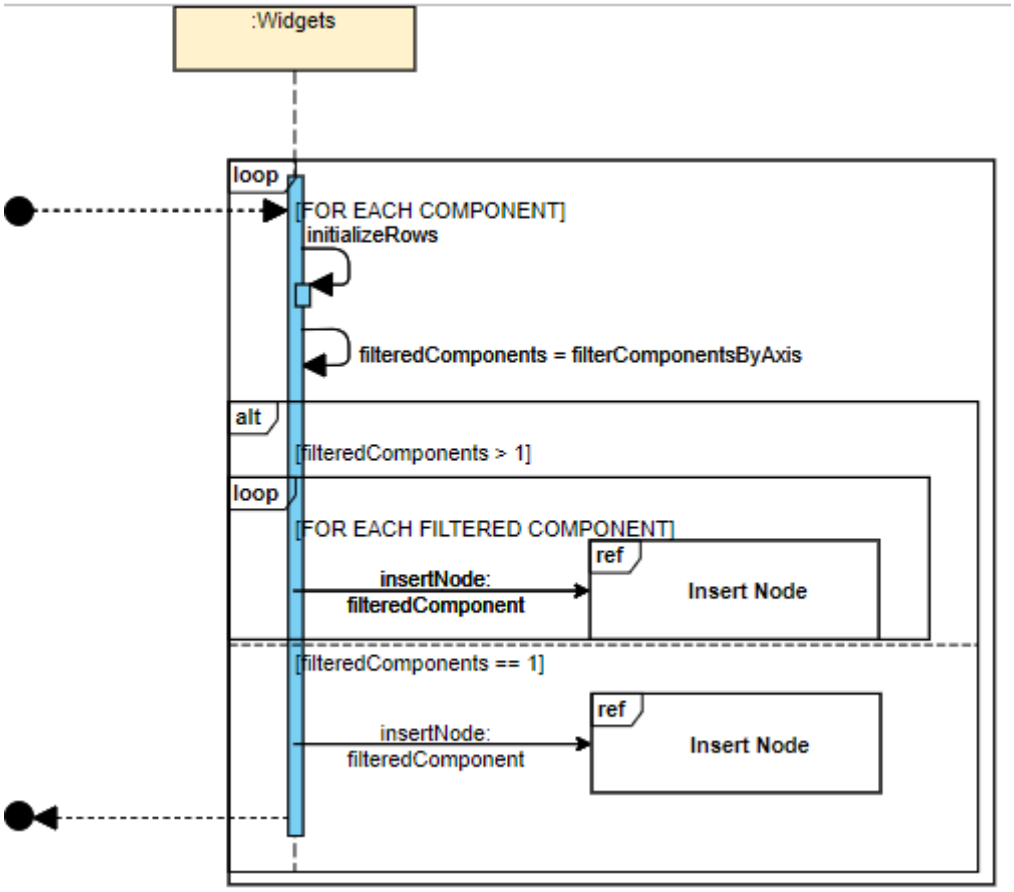
Diagrama 3 - Diagrama de Sequência 3



Fonte: O autor (2019)

O Diagrama 4, DIA-3 exibe o processo de construção da árvore de componentes. Este processo está detalhado na sessão 5.2.3 do presente trabalho e demonstra a abordagem utilizada para construção do layout na *framework Flutter.*

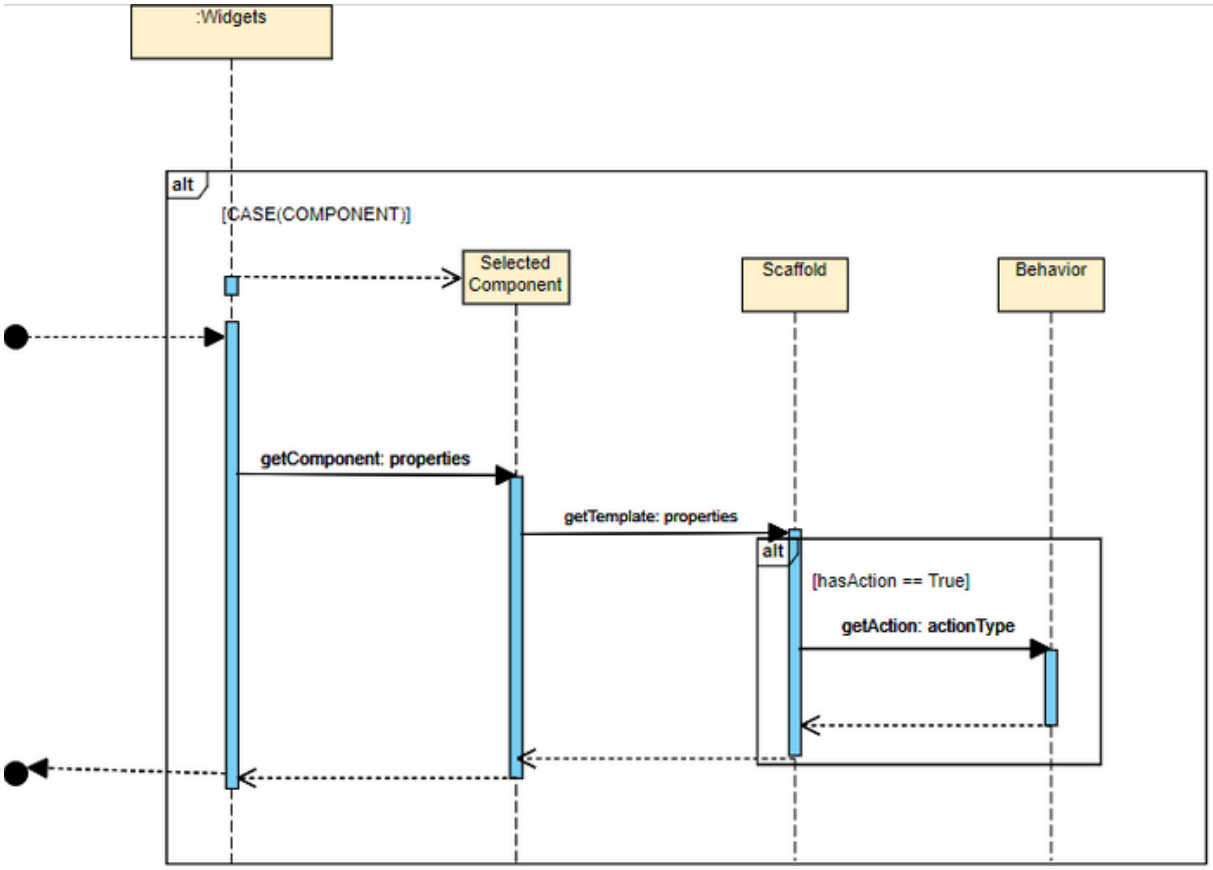
Diagrama 4 - Diagrama de Sequência - 3



Fonte: O autor (2019)

O Diagrama 4, "*Insert Node"* exibe o fluxo de um componente genérico. Esse comportamento é replicado para cada tipo de componente existente na aplicação, assim a função "getComponent" é um nome genérico que quando a chamada é por exemplo de um botão, a função é chamada como "getButton".

Diagrama 5 - Diagrama de sequência - Insert Node

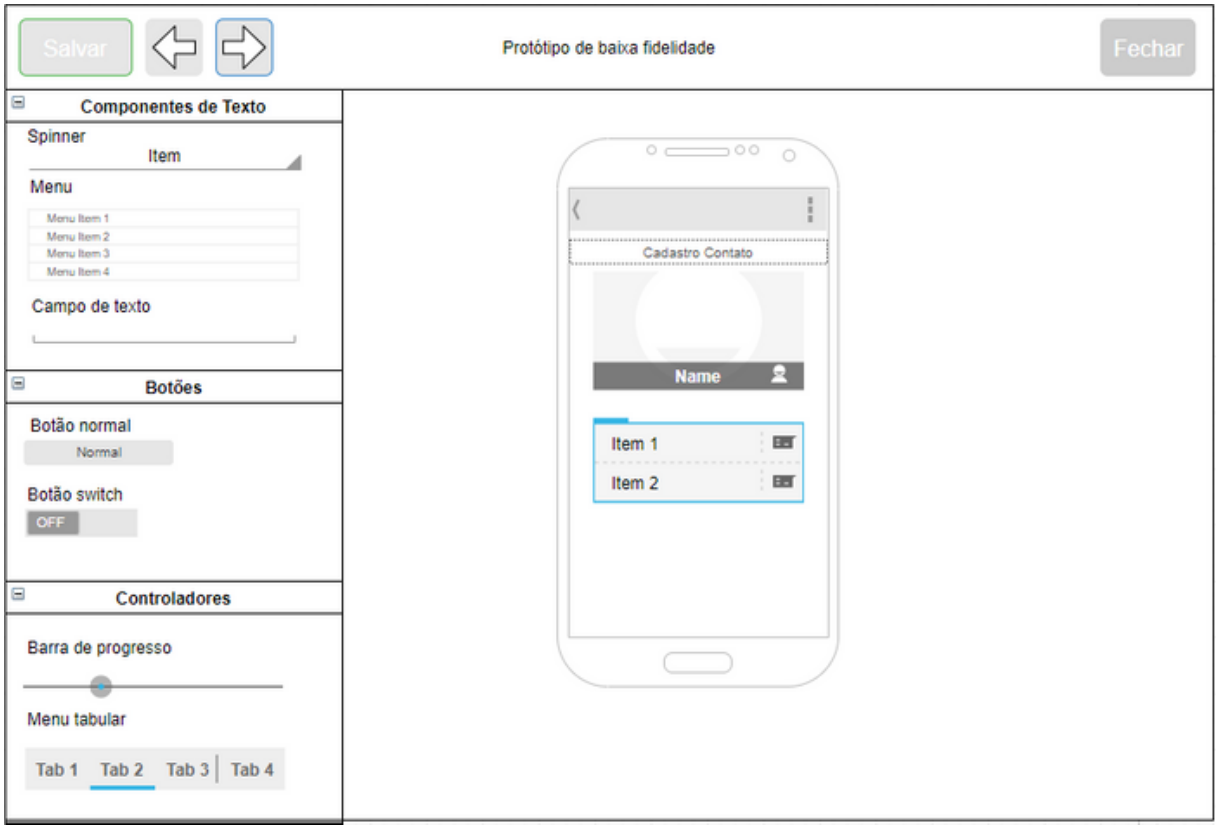


Fonte: O autor (2019)

4.2.5 **PROTOTIPAÇÃO E ARQUITETURA**

A figura 15 exibe um protótipo de baixa fidelidade da tela de editor da aplicação web. À **esquerda** é mostrado o menu de componentes, no qual se encontram todos os *widgets* disponíveis para o usuário criar as interfaces da aplicação. No **centro** fica o *mockup* do *smartphone*, onde o usuário poderá arrastar e soltar os componentes para criação do app. Na parte **superior** estão os menus de ações da aplicação web, por meio dos quais o usuário poderá salvar, editar, abrir novo *template,* desfazer e refazer ações.

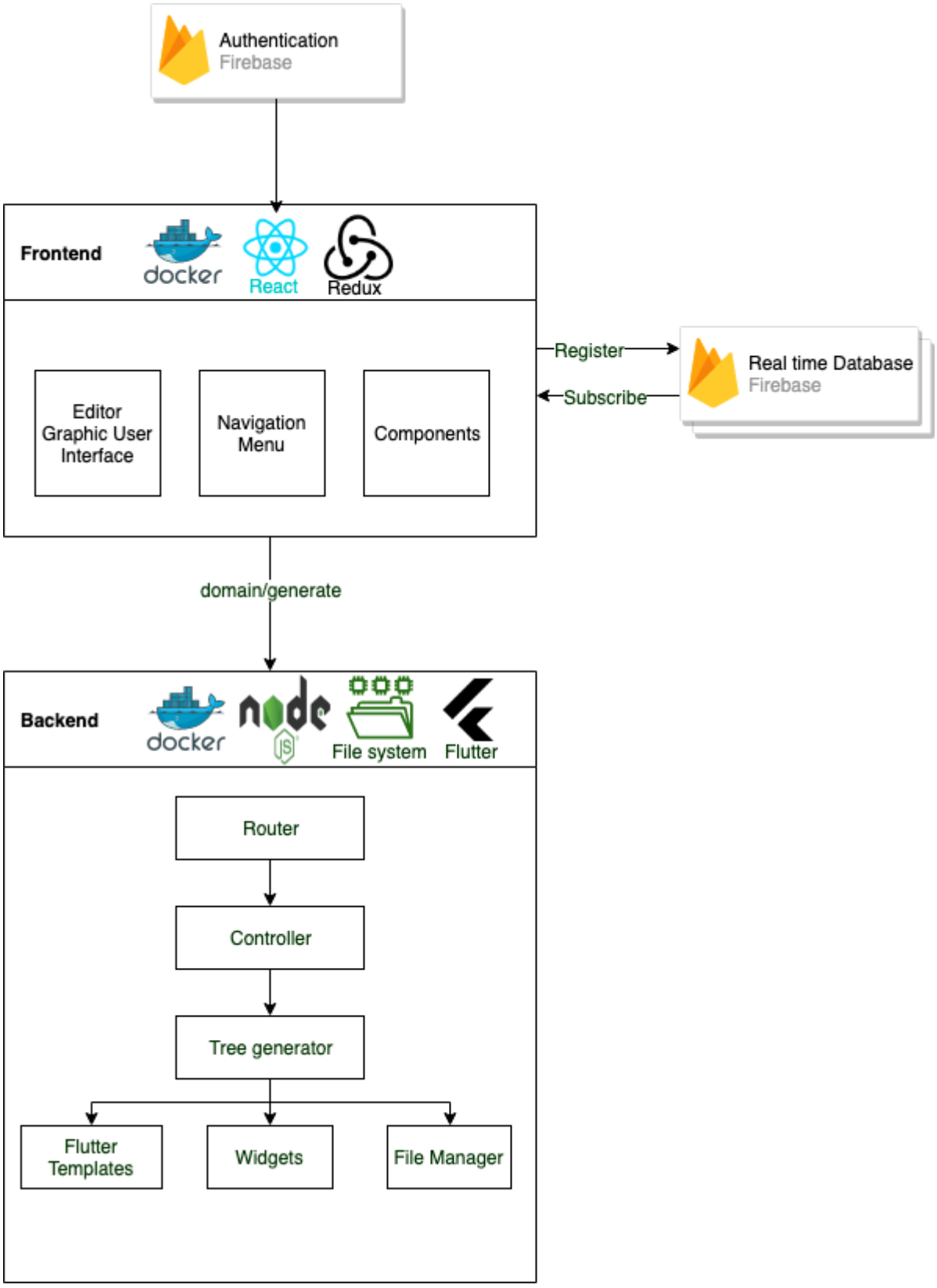
Figura 13 - Protótipo de baixa fidelidade



Fonte: O autor (2019)

O Diagrama 6 mostra a arquitetura básica da aplicação, que está dividida em duas camadas principais e duas conexões a APIs de terceiros. No frontend da aplicação ficam os módulos do Menu, Editor e componentes que são renderizados ao mesmo tempo.

Diagrama 6 - Arquitetura da aplicação



Fonte: O autor (2019)

4.2.5.1 **Módulos do Frontend**

O *frontend* da aplicação feita tem uma estrutura de pastas e arquivos bem robustas, para fins de explicação nenhum módulo de configuração ou de pacotes foi adicionado ndo diagrama de arquitetura. Partindo deste princípios os módulos principais do *frontend* são:

- **Components:** Neste módulo está contido toda a lógica de renderização, configurações iniciais, listagem e atributos característicos de cada componente existente na aplicação.
- **Editor GUI:** Este módulo é responsável por toda parte gráfica da aplicação, nesta parte do projeto é realizada a renderização de interface do dispositivo móvel. Também neste módulo está o editor de propriedades da aplicação, de interface e cada componente, sendo capaz de realizar customização das partes do *layout* desejado.
- **Navigation Menu:** Os menus superiores da ferramenta estão implementados neste módulo. Toda a lógica de adição de interface, conexão com *backend* e aplicações de terceiros fica nos menus correspondentes as ações desejadas.

4.2.5.2 **Módulos do Backend**

Os módulos do *backend* são responsáveis pela conversão do *layout* editado no frontend em código Flutter. Também é no *backend* que o código Flutter é gerado. O back*end* é acessado através do *frontend* a partir de um método POST e através dos controladores distribui para as camadas seguintes.

- **Router:** O módulo de roteamento tem a função de somente receber os dados no seu *endpoint* específico e enviar para o respectivo controlador.
- **Controller:** Os controladores tem como principal função receber os dados das rotas e distribuir para os módulos de geração de código.
- **Tree Generator:** Este módulo é responsável por gerenciar todo o ciclo de vida do processo de geração de código no *backend.* O gerador faz a iteração e conversão dos dados recebidos e faz a chamada para os módulos específicos de cada parte da geração da aplicação móvel. Ao final do processo este módulo retorna uma pasta compactada com o código gerado e o aplicativo para ser disponibilizado pelo *frontend.*
- **Flutter Templates:** Neste módulo ficam salvos os templates escrito em código Dart/Flutter que serão substituídos pelos dados provenientes do *frontend* respeitando as regras de cada template.

- **Widgets:** Todos os scaffold's dos componentes Flutter e seus comportamentos ficam armazenados no módulo de *widgets.* Estes widget ficam armazenados pelo tipo correspondente e cada vez que é chamado retorna o código em Flutter contido no módulo.
- **File Manager:** O *file manager* faz toda escrita e leitura de arquivos está implementada no módudo de gerenciamento de arquivos.

4.2.5.3 **Aplicações de Terceiros**

Usando um serviço de Plataforn as a Service[37] (PaaS abreviadamente) foi possível fazer persistência dos layouts gerados sem a necessidade de um banco de dados local. Como solução de PaaS foi selecionado o *Firebase* para realizar autenticação e bando de dados em tempo real sem a necessidade de criação de módulos específicos para utilizar estes recursos. Os recursos disponibilizados pelo *Firebase* são explanados na sessão 5.2 do presente trabalho.

37 https://azure.microsoft.com/pt-br/overview/what-is-paas/

5 **DESENVOLVIMENTO DA FERRAMENTA**

O desenvolvimento da ferramenta foi concebido seguindo os recursos de engenharia de software expostos no capítulo anterior. A implementação da solução foi realizado em 2 etapas. Sendo a primeira uma prova de conceito com o intuito de demonstrar a viabilidade de desenvolvimento. Na segunda etapa foi desenvolvida a ferramenta final aproveitando parte do código fonte construído para a prova de conceito. Neste capítulo serão abordadas questões pertinentes ao desenvolvimento da ferramenta, assim como tecnologias adotadas, metodologias de implementação e bibliotecas selecionadas.

5.1 **DART**

Como relata Sikora (2015), a linguagem Dart foi revelada na conferência GOTO[38] em Aarhus, em outubro de 2011. Seu objetivo principal, a longo prazo, é substituir o JavaScript como a única linguagem nos navegadores. Embora o JavaScript seja muito fácil de usar para aplicativos menores, com a crescente complexidade e a necessidade de dimensionar os projetos de hoje, torna-se rapidamente muito difícil de manter. JavaScript não foi projetado para isso, e usá-lo para escrever aplicativos maiores é uma tarefa difícil. O Dart foi criado como uma nova linguagem com a sintaxe no estilo C; é orientada a objetos e baseada em classe com um único modelo de herança com *mixins*[39]. Ele oferece muitos recursos utilizados em outras linguagens, como classes abstratas, encapsulamento, reflexão, exceções e assim por diante. Além disso, é possível fazer uso de verificação de tipo estático opcional. O Dart usa uma abordagem muito fácil de entender, que permite que o desenvolvedor se concentre em escrever seus aplicativos em vez de lidar com a própria linguagem.

Em novembro de 2013, o Dart chegou ao seu primeiro lançamento estável, 1.0, e ainda está em desenvolvimento ativo. Em março de 2015, a equipe Dart lançou a versão 1.9, que simplificou significativamente o trabalho com APIs assíncronas e é considerada a versão mais importante desde a versão 1.0. No final de abril de 2015 o Google realizou o primeiro Dart Summit, revelando planos de usar o Dart como uma linguagem para o desenvolvimento móvel multi-plataforma com seu novo runtime, chamado Fletch.

Segundo Ortiz (2012), *Javascript* é bom para adicionar interatividade básica para páginas da Web, mas não é necessariamente melhor quando aplicativos da

38 https://blog.gotocon.com/

39 https://medium.com/flutter-community/https-medium-com-shubhamhackzz-dart-for-flutter-mixins-in-dart-f8bb10a3d341

Web incham em milhares de linhas de código. Linguagens de script são leves e permitem que os desenvolvedores escrevam código rapidamente. No entanto, essas linguagens fazem com que pequenos scripts evoluam para grandes programas sem estrutura aparente. Isso dificulta a compreensão, a manutenção, a depuração e a colaboração no desenvolvimento das aplicações. Dart, por sua vez, permite que aplicativos sejam mais fáceis de entender, manter e depurar. Isso também permite que os desenvolvedores escrevam no estilo de script, e não apenas aplicações maiores. Dart é projetado para rodar tanto no cliente quanto no servidor, mas a única maneira para executar o código Dart do lado do cliente atualmente é usando um *cross-compiler* para JavaScrip**t.**

5.2 **FLUTTER**

Flutter é uma *framework* de desenvolvimento de aplicativos do Google para criar aplicativos móveis *cross-platform* (para iOS e Android), que visa tornar o desenvolvimento mais fácil, mais rápido e mais eficiente. Características como o *Hot Reload*, um vasto catálogo de *widgets*, um desempenho muito bom e uma sólida comunidade contribuem para atingir esse objetivo e fazem do Flutter uma boa alternativa.

A origem do Flutter foi semelhante à de muitos softwares famosos. Inicialmente, o Flutter era um experimento criado por desenvolvedores do Google que tentavam remover alguns suportes de compatibilidade do Chrome com o intuito de fazer com que ele rodasse mais suavemente. Depois de algumas semanas e de muitos dos suportes de compatibilidade serem removidos, os desenvolvedores descobriram que tinham algo 20 vezes mais rápido do que o Chrome e viram que ele tinha potencial para ser algo grande. O Google criou uma estrutura em camadas que se comunicava diretamente com a CPU e a GPU para permitir que o desenvolvedor personalizasse os aplicativos o máximo possível (MAINKAR; GIORDANO, 2019).

O Flutter pode se tornar um fator de mudança no setor devido à arquitetura exclusiva, com o próprio mecanismo de renderização, ciclo de desenvolvimento rápido e um ambiente estável. (AGOSTINI.TECH, 2019)

Segundo Kobets (2019), a arquitetura do Flutter (ilustrada na Figura 6) possui várias camadas:

- Framework: linguagem Dart + Widgets;
- Flutter engine em C++;
- Embedder: integrações específicas da plataforma.

Arquitetura FlutterAgostini.Tech (2019)

Framework DART e Widgets

Widgets são o núcleo da camada das interfaces gráficas da *framework* Flutter. O bloco de interface do usuário básico é chamado *Widget*. Cada *widget* encapsula a apresentação, a configuração e o estado da interface do usuário. O Flutter tem sua própria implementação de componentes de interface do usuário. Para iOS, é chamado Cupertino e basicamente tem todos os componentes do UIKit (User Interface kit).

Flutter desenha os elementos do usuário em seu próprio contexto, de modo que quaisquer alterações nas bibliotecas nativas (por exemplo, UIKit) não afetam a estrutura. Imagine que a Apple mude o UIKit na nova versão do iOS e todo o código que funciona em torno desta API precise ser reescrito. Isso não ocorre com o Flutter, que conta com seu próprio motor 2D. Portanto, o Flutter contém dependências mínimas para o sistema.

Os widgets são criados por meio de composição e permitem que os widgets tenham vários comportamentos, como o *ScrollableandAnimatable*. Por meio de composição, fica mais fácil projetar uma interface de usuário flexível.

Flutter engine

Segundo um dos criadores do Flutter Ian Hickson (2018), o Flutter Engine é um executor de tempo de execução portável para aplicativos móveis de alta qualidade. Ele implementa as principais bibliotecas do Flutter, incluindo animação e gráficos, E / S de arquivos e redes, suporte à acessibilidade, arquitetura de plug-ins e um tempo de execução e toolchain do Dart para desenvolvimento, compilação e execução de aplicativos Flutter.

Flutter possui um sistema próprio de interface do usuário, independente dos elementos nativos. Ele usa a biblioteca de gráficos 2D Skia, que é usada no Chrome, Android e no Mozilla Firefox como um mecanismo de renderização. O uso do próprio mecanismo de renderização oferece o mais alto desempenho em renderização e animações. De acordo com sua documentação, o Flutter tem como objetivo fornecer um desempenho de 60 quadros por segundo (fps) ou um desempenho de 120 fps em dispositivos capazes de atualizações de 120Hz.

Hot Reload

Um recurso útil do Flutter é o *hot reload*. O desenvolvedor não precisa recompilar o aplicativo inteiro para ver uma pequena alteração na interface do usuário ou corrigir um *bug*. O código injetado pode ser recarregado no Dart Virtual Machine, salvando o arquivo editado com o aplicativo em execução. É preciso apenas instalar o *plugin* do Microsoft Visual Studio Code[40] e executar o aplicativo no modo de depuração.

Embedder

40 https://code.visualstudio.com/

Segundo Hickson (2018) a engine Flutter não cria nem gerencia suas próprias *threads*. Em vez disso, é responsabilidade do *embedder* criar e gerenciar encadeamentos (e seus loops de mensagens) para o mecanismo Flutter. O *embedder* fornece aos *task runners*[41] os segmentos que ele gerencia. Além dos encadeamentos gerenciados pelo *embedder* para o mecanismo, o Dart Virtual Machine(VM abreviadamente) também possui seu próprio *thread pool*[42].

Nem o motor Flutter nem o embedder têm acesso aos threads neste pool.

A engine do Flutter requer que o *embedder* forneça referências a 4 t*ask runners*. O mecanismo não se importa se as referências são para o mesmo executor de tarefas ou se vários *task runners* são atendidos na mesma *thread*. Para um ótimo desempenho, o *embedder* deve criar um *thread* dedicado por *task runner*. Embora a engine não se importe com as *threads* em que os *task runners* estão ocupando, ele espera que a configuração de encadeamento permaneça estável durante toda a vida útil do processo. Ou seja, uma vez que o *embedder* decide atender a um executor de tarefas em uma determinada *thread*, ele deve executar tarefas para esse *task runner* apenas nessa thread(até o processo ser encerrado).

Os principais *task runners* são:

- *Platform Task Runner*: *Task runners* da plataforma específica de desenvolvimento
- *UI Task Runner*: *Tasks* pertencentes a interface de usuário
- *GPU Task Runner*: *Tasks* de processamento gráfico
- *IO Task Runner*: *Tasks* de entrada e saída

5.3 **TESTES DE INTERFACE E TESTES DE UNIDADE**

Teste da GUI é o teste do sistema de um software que possui um front-end da interface gráfica do usuário (GUI). Como o teste do sistema implica que todo o sistema de software, incluindo a interface do usuário, seja testado como um todo, durante o teste da GUI, casos de teste - modelados como sequências de eventos de entrada do usuário (BANERJEE; NGUYEN, 2013). Testes de interface são importantes para capturar erros durante a execução do software e a interação do usuário com a aplicação.

O teste de unidade representa uma atividade essencial no desenvolvimento e manutenção de software. Os conjuntos de testes com alta qualidade interna facilitam as atividades de manutenção, como compreensão de código e teste de regressão (BAVOTA; QUSEF; OLIVETO, 2012). Segundo A. Ynchausti (2001) algumas

41 Task Runners são automatizadores de tarefas.
42 Um pool de threads é uma coleção de threads disponíveis para realizar tarefas. http://www.dsc.ufcg.edu.br/~jacques/cursos/map/html/threads/pool.html

práticas-chave do teste de unidade são: Os testes de unidade são automatizados para facilitar a execução e execute novamente para validar o sistema de software de produção.

- Os testes de unidade são automatizados para facilitar a execução e execute novamente para validar o sistema de software de produção.
- Os testes de unidade são automatizados para facilitar a execução e execute novamente para validar o sistema de software de produção.

· Testes de unidade são criados para todas as classes no sistema de produção.

· Testes de unidade são implementados para todos os métodos que poderia quebrar no sistema de produção.

· Os testes de unidade são codificados da forma mais simples e direta caminho possível.

· Testes de unidade são criados para que conceitos e construções são expressas apenas uma vez.

· Cada teste de unidade retorna um valor indicando que o teste aprovado ou falhou.

· Todos os testes de unidade devem passar antes do lançamento do novo código.

· Os testes de unidade são atualizados com o código de produção e usado por todos os desenvolvedores que trabalham nos programas.

5.4 **JAVASCRIPT**

JavaScript é uma linguagem de programação interpretada, multi-paradigma, com uma sintaxe muito parecida com a sintaxe de linguagens como C++/C e Java, além de diversas inspirações na linguagem Perl, FLANAGAN (2006). O nome oficial da linguagem, segundo a especificação ECMA-262, é ECMAScript, pois a linguagem foi padronizada e estabilizada pela associação *European Computer Manufacturers Association* (ECMA), e conta com diversas implementações do padrão. Segundo (FLANAGAN, 2006), desde a sua introdução, o JavaScript cresceu e amadureceu muito mais do que se esperava inicialmente. Agora é possível codificar aplicativos da Web poderosos, aplicativos móveis, aplicativos do Windows e também servidores inteiros com JavaScript. A natureza versátil dessa linguagem atingiu seu ápice - agora ela é usada para monitorar e controlar um telefone celular, robótica, as luzes da sua casa, movimento de mão e dedo com o *Leap Motion*[43], realidade aumentada com o Google Glass e muito mais.

Conforme (JAIMINI, 2016) o que fez com que o JavaScript se difundisse tão

43 https://www.leapmotion.com/

rapidamente foi a sua facilidade de aprendizado. Não é necessário conhecimento prévio ou experiência de programação para codificar em JavaScript. Além disso, basta ter um navegador da Web instalado em uma máquina para executar as aplicações. Isso significa que faz muito mais sentido incluir essa linguagem em dispositivos que eventualmente visam fazer parte da Internet. Existe uma grande comunidade de desenvolvedores colaborativos que contribuem para aumentar o conjunto de recursos disponível sobre a linguagem, como artigos, tutoriais, vídeos do YouTube e até mesmo sites inteiros dedicados a melhorar o entendimento das pessoas sobre a linguagem.

A natureza orientada a eventos do JavaScript se encaixa perfeitamente com a forma de interação dos usuários com a Web, que ocorre de forma simples e natural, visto que os usuários agem e reagem a diferentes eventos em suas vidas diárias (JAIMINI; DHANIWALA, 2016).

5.5 **NODEJS**

O Node.js é uma plataforma JavaScript do lado do servidor que usa um modelo de E / S sem bloqueio, orientado a eventos, permitindo que os usuários criem aplicativos de dados intensivos, rápidos e escaláveis, em execução em tempo real (HERRON, 2018). Com JavaScript sendo uma das tecnologias mais usadas na Web para programação do lado do cliente, a possibilidade de uso do JavaScript do lado do servidor, a partir da criação do Node.js em maio de 2009, ofereceu o potencial de integração de programação cliente-servidor, usando a mesma linguagem de ponta a ponta (IOANNIS; KYRIAKOU; TSELIKAS, 2015).

Cada plataforma tem sua própria filosofia - ou seja, um conjunto de princípios e diretrizes - alguns dos quais surgem da própria tecnologia, alguns são capacitados por seu ecossistema, alguns são apenas tendências na comunidade e outros são evoluções de diferentes ideologias. No Node.js, alguns desses princípios vêm diretamente de seu criador, Ryan Dahl (2009), das pessoas que contribuíram para o núcleo, de figuras de destaque na comunidade de desenvolvedores, e alguns dos princípios são herdados da cultura JavaScript ou são influenciados pela filosofia Unix. Suas características são?

Núcleo pequeno (Small core)

O próprio núcleo do Node.js, segundo a Node.js Foundation (2017), tem suas bases construídas em alguns princípios; um deles é ter o menor conjunto de funcionalidades, deixando o restante para a chamada *userland* (ou *userspace*), o ecossistema de módulos que vivem fora do núcleo. Esse princípio tem um enorme

impacto na cultura do Node.js, pois dá liberdade para a comunidade experimentar e iterar rapidamente em um conjunto mais amplo de soluções dentro do escopo dos módulos do usuário, em vez de ser imposta com uma solução em evolução lenta que é embutida no núcleo, mais rigidamente controlado e estável. Manter o conjunto básico de funcionalidades no mínimo, não apenas se torna conveniente em termos de manutenção, mas também em termos do impacto cultural positivo que isso traz à evolução de todo o ecossistema. Módulos pequenos (Small modules)

O Node.js usa o conceito de módulo como meio fundamental para estruturar o código de um programa. É o tijolo para criar aplicações e bibliotecas reutilizáveis chamadas pacotes ou *packages* (um *package* também é freqüentemente chamado apenas de módulo, já que, geralmente, ele possui um único módulo como um ponto de entrada). No Node.js, um dos princípios mais evangelizados é projetar pequenos módulos, não apenas em termos de tamanho de código, mas o mais importante, em termos de escopo (Node.js Foundation, 2017). Este princípio tem suas raízes na filosofia Unix, particularmente em dois de seus preceitos, que são os seguintes:

"Pequeno é bonito."

"Faça cada programa fazer uma coisa bem."

O Node.js trouxe esses conceitos para um novo nível. Juntamente com a ajuda do Node Package Manager (npm abreviadamente), seu gerenciador de pacotes oficial, o Node.js ajuda a resolver o problema de *dependency hell*[44], certificando-se de que cada pacote instalado terá seu próprio conjunto separado de dependências, permitindo que um programa dependa de muitos pacotes sem incorrer em conflitos. Essa característica estimula a reusabilidade, e faz com que os aplicativos sejam compostos de um grande número de dependências pequenas e bem focadas. Embora isso possa ser considerado impraticável ou totalmente inviável em outras plataformas, no Node.js essa prática é incentivada. Como conseqüência, não é raro encontrar pacotes npm contendo menos de 100 linhas de código ou que possuem apenas uma função. Além da clara vantagem em termos de reutilização, um pequeno módulo também é considerado o seguinte:

- Mais fácil de entender e usar
- Mais simples de testar e manter
- Perfeito para compartilhar com o navegador

Ter módulos menores e mais focados permite que todos compartilhem ou reutilizem até mesmo o menor código; é o princípio Don't Repeat Yourself (DRY) aplicado em um nível totalmente novo.

Pequena área de superfície (Smalll surface area)

Além de serem pequenos em tamanho e escopo, os módulos do Node.js geralmente também têm a característica de expor apenas um conjunto mínimo de

44 https://www.techopedia.com/definition/27701/dependency-hell

funcionalidades. A principal vantagem aqui é uma maior usabilidade da API, o que significa que a API se torna mais clara para uso e está menos exposta a uso incorreto. Na maioria das vezes, na verdade, o usuário de um componente está interessado apenas em um conjunto muito limitado e focado de recursos, sem a necessidade de estender sua funcionalidade ou explorar aspectos mais avançados.

No Node.js, um padrão muito comum para a definição de módulos é expor apenas uma parte da funcionalidade, como uma função ou um construtor, ao permitir que aspectos mais avançados ou recursos secundários se tornem propriedades da função ou construtor exportado. Isso ajuda o usuário a identificar o que é importante e o que é secundário. Outra característica de muitos módulos do Node.js é o fato de que eles são criados para serem usados em vez de estendidos. Bloquear os componentes internos de um módulo, proibindo qualquer possibilidade de extensão, pode parecer inflexível, mas na verdade tem a vantagem de reduzir os casos de uso, simplificar sua implementação, facilitar sua manutenção e aumentar sua usabilidade.

Simplicidade e pragmatismo (Simplicity and pragmatism)

Richard P. Gabriel, um proeminente cientista da computação, cunhou o termo em seu ensaio "A ascensão do pior é melhor"[45]. Segundo o autor, "O design deve ser simples, tanto na implementação quanto na interface. É mais importante que a implementação seja simples do que a interface. A simplicidade é a consideração mais importante em um projeto."

Projetar um software simples, em oposição a um software perfeito, é uma boa prática por vários motivos: é preciso menos esforço para implementar, permite um envio mais rápido com menos recursos, é mais fácil de adaptar e é mais fácil de manter e entender. Esses fatores estimulam as contribuições da comunidade e permitem que o próprio software cresça e melhore. No Node.js, esse princípio também é proporcionado pelo JavaScript, que é uma linguagem muito pragmática. Não é raro, na verdade, ver funções simples, closures e literais de objeto substituindo hierarquias de classes complexas. Projetos puramente orientados a objetos frequentemente tentam replicar o mundo real usando os termos matemáticos de um sistema de computador sem considerar a imperfeição e a complexidade do próprio mundo real. A verdade é que software é sempre uma aproximação da realidade e provavelmente haveria mais sucesso em tentar fazer com que algo funcionasse mais cedo e com razoável complexidade, em vez de tentar criar um software quase perfeito com um enorme esforço e toneladas de código para manter.

Segundo Herron (2018), a arquitetura do Node.js parte de uma escolha típica feita por outras plataformas de aplicativos, nas quais os encadeamentos são amplamente usados para dimensionar um aplicativo para preencher a CPU. O Node.js evita encadeamentos devido à sua complexidade inerente. Alega-se que,

45 http://dreamsongs.com/WorseIsBetter.html

com arquiteturas orientadas a eventos de thread único, o footprint de memória é baixo, a taxa de transferência é alta, o perfil de latência sob carga é melhor e o modelo de programação é mais simples. A plataforma Node.js está em uma fase de rápido crescimento, e muitos a vêem como uma alternativa convincente às arquiteturas tradicionais de aplicativos da Web usando Java, PHP, Python ou Ruby on Rails.

5.6 **BIBLIOTECAS JAVASCRIPT**

A ferramenta proposta faz uso de diversas bibliotecas Javascript, que são abstrações que facilitam o desenvolvimento e trazem soluções para muitos problemas conhecidos nos browsers. À medida que o escopo do uso de uma linguagem de programação aumenta, torna-se extremamente importante que um método de distribuição de código seja usado, não apenas para simplificar o desenvolvimento, mas também para permitir a reutilização do código pelas massas. As bibliotecas JavaScript resolvem esse problema muito bem e existem dezenas, senão centenas, de bibliotecas disponíveis que atendem a vários casos de uso[46].

Essas várias bibliotecas fornecem código JavaScript pré-escrito, o que facilita a execução de tarefas comuns ou complexas. As bibliotecas JavaScript geralmente têm como alvo tarefas específicas, como manipulação de DOM, configuração de estrutura e manipulação de AJAX. Segundo a organização *WebPlatform* o uso de bibliotecas javascript trás diversos benefícios, entre eles:

- Interface uniforme para código compatível com vários navegadores: Enquanto os navegadores modernos estão se tornando cada vez mais semelhantes em suas implementações dos recursos de idioma e DOM, ainda existem inúmeras pequenas diferenças. O problema se torna substancial quando a compatibilidade com navegadores antigos é necessária. Algumas bibliotecas JavaScript tentam reduzir o código padrão que os desenvolvedores precisam escrever para solucionar esse problema. Eles fornecem uma API uniforme para o desenvolvimento, na qual é implementada com compatibilidade entre todos os navegadores.
- Níveis mais altos de abstração: Muitas vezes, o desenvolvimento requer implementações de recursos bastante populares na indústria. Complementação automática, elementos gráficos e interface do usuário são alguns exemplos de áreas em que as bibliotecas podem ser um grande benefício. A maioria dos esforços pode ser focada na personalização.
- *Frameworks*: Embora a linguagem em si seja bastante poderosa e extremamente fácil de usar, o desenvolvimento e a manutenção de grandes

46 https://webplatform.github.io/docs/concepts/programming/javascript/libraries/

bases de código podem se tornar desafiadores. Existem várias estruturas disponíveis, como bibliotecas JavaScript que implementam padrões de projeto, como o MVC, para escrever o código de uma maneira mais estruturada.

5.6.1 **REACT**

Também conhecido como React, é uma biblioteca JavaScript de código aberto para criação de interfaces de usuário. É usado para lidar com a camada de visualização em aplicações de página única (SPA -) e no desenvolvimento de aplicativos móveis. O React é mantido por um grupo de desenvolvedores do Facebook e do Instagram, e conta também com contribuição da comunidade de desenvolvedores e outras empresas. O React se esforça para fornecer velocidade, simplicidade e escalabilidade. Algumas de suas características mais notáveis são JSX, componentes com estado e modelo de objeto de documento virtual (virtual DOM).

O React.js[47], também conhecido como React, é uma biblioteca JavaScript de código aberto para criação de interfaces de usuário. É usado para lidar com a camada de visualização em aplicações de página única (SPA - *Single Page Applications*) e no desenvolvimento de aplicativos móveis. O React é mantido por um grupo de desenvolvedores do Facebook e do Instagram, e conta também com contribuição da comunidade de desenvolvedores e outras empresas. O React se esforça para fornecer velocidade, simplicidade e escalabilidade. Algumas de suas características mais notáveis são JSX, componentes com estado e modelo de objeto de documento virtual (virtual DOM).

5.6.1.1 **JSX**

JavaScript XML (JSX) é uma extensão da sintaxe do ECMAScript sem nenhuma semântica definida. (JSX 2014[48]). O React adota o fato de que a lógica de renderização é inerentemente acoplada a interface do usuário. Em vez de separar tecnologias, o React usa unidades fracamente acopladas, chamadas componentes, que contêm ambos. JSX é opcional e não é necessário para usar o React. No entanto, o JSX é uma boa ajuda visual ao trabalhar com a interface do usuário dentro do JavaScript. JSX também permite que o React mostre mensagens de erro e avisos mais úteis.

47 https://pt-br.reactjs.org/
48 https://facebook.github.io/jsx/

5.6.1.2 **Componentes com estado (*Stateful components*)**

O React permite que os usuários dividam a interface do usuário em partes independentes e reutilizáveis, chamadas *React Components*. Os componentes do React implementam um método de renderização que pega os dados de entrada e retorna o que exibir. Cada componente possui vários métodos de ciclo de vida que podem ser substituídos para executar o código em momentos específicos durante o processo. Os métodos podem ser chamados usando a API Reacts.

State é um objeto JavaScript simples usado para registrar e reagir a eventos do usuário. Cada classe de componente definido possui seu próprio objeto de estado. Sempre que um estado do componente é alterado, o componente e todos os seus componentes filhos são renderizados imediatamente. Estados mantêm valores em todo o componente e podem ser transmitidos para componentes filhos como parâmetros.

5.6.1.3 **Modelo de objeto de documento virtual (Virtual DOM)**

O DOM HTML foi originalmente destinado a páginas estáticas e, portanto, não foi otimizado para a criação de UI dinâmica. Quando o DOM é atualizado, ele precisa atualizar todos os nós e renderizar novamente a página com o CSS e o layout correspondentes. É comum que um aplicativo de página única contenha milhares de nós gerados dinamicamente que têm ouvintes de eventos anexados a eles. Em páginas dinâmicas, o HTML DOM deve verificar alterações em todos os dados do nó em um intervalo regular. Isso reduz consideravelmente o desempenho do aplicativo. O DOM Virtual foi inventado como uma solução para essa ineficiência[49].

O DOM virtual é uma abstração do DOM em HTML. É leve e desacoplado do navegador, podendo ser atualizado sem afetar o DOM real. O React possui o DOM Virtual integrado em um módulo chamado ReactDOM. Quando as atualizações são fornecidas, o React usa um processo chamado reconciliação, usando um algoritmo que compara e contrasta as alterações para saber quais elementos precisam de atualização. O React, então, apenas atualiza esses elementos que tiveram alteração de estado, deixando todos os outros elementos não afetados sem a necessidade de uma nova renderização.

5.6.1.4 **REDUX**

Redux é um contêiner de estado previsível para aplicativos JavaScript. A

49 https://reactjs.org/docs/faq-internals.html

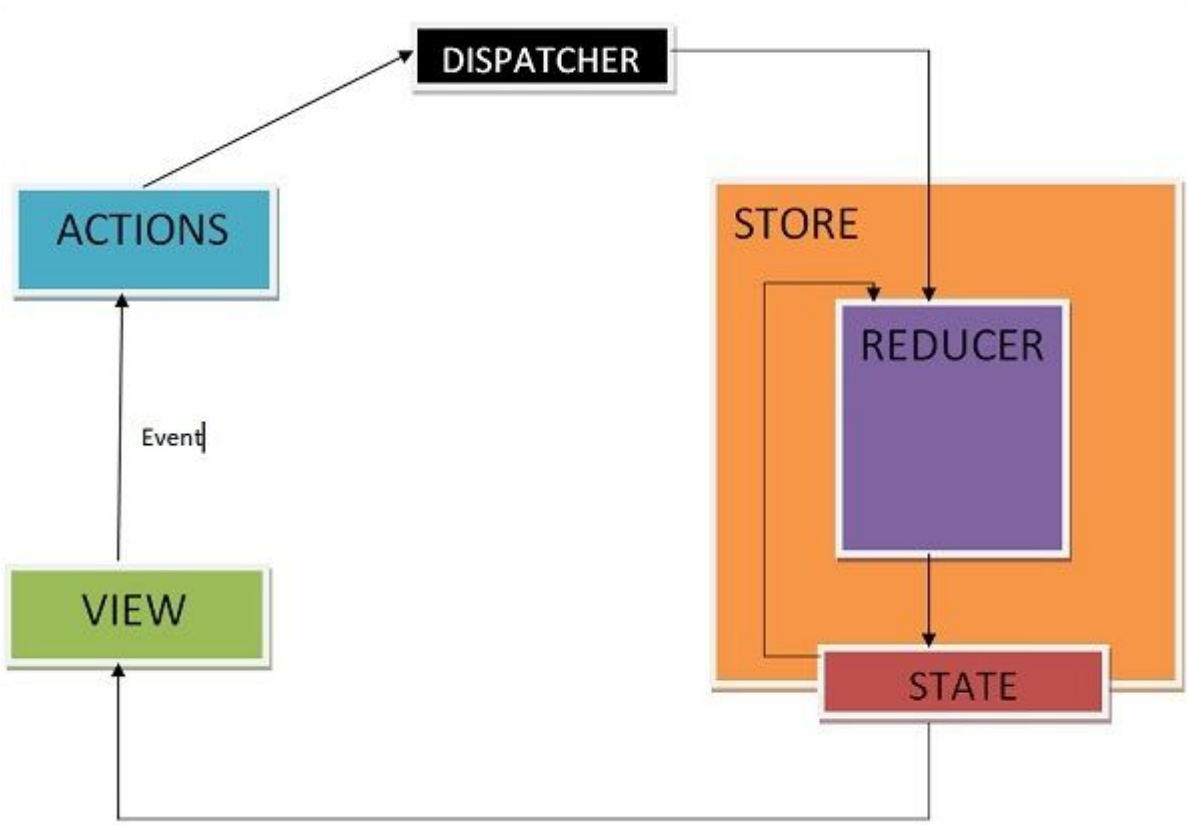
gestão do estado é uma dos aspectos mais difíceis do desenvolvimento de software baseado em React. A má administração dos estados é uma fonte de erros bastante significativa. Redux é uma implementação simplificada da arquitetura *Flux* do *Facebook*, que é uma estrutura MVC, e que diminui a complexidade usando *reducers*. *Reducers* Redux são funções sem efeitos colaterais que computam o estado do aplicativo(Geary 2016).

O Redux é baseado em três princípios:

- O estado do aplicativo é armazenado em um único objeto. Redux armazena o estado em um único objeto JavaScript para facilitar o mapeamento e a transmissão de dados por todo o aplicativo. Centralizar o estado em um único objeto também torna os processos de testes e de depuração mais rápidos.
- O estado do aplicativo é imutável. No Redux, os estados não podem ser modificados. A única maneira de mudar o estado é executando uma ação. Ações são objetos JavaScript imutáveis que descrevem as mudanças de estado. As ações são executadas para impedir condições de corrida.
- Os *reducers* especificam como a ação transforma o estado. *Reducers* são funções JavaScript que criam um novo estado com o estado e a ação atuais. Eles centralizam mutações de dados e podem atuar em todo ou em parte do estado. *Reducers* também podem ser combinados e reutilizados.

A figura 16 demonstra o fluxo de ações na arquitetura React/Redux. Quando um evento é chamado na View, uma *Action* é disparada para o *Store* através de um *Dispatcher*. Esta ação realiza a alteração do estado por meio dos *Reducers* e quando essa alteração de estado é realizada, a View é atualizada.

Figura 14 - Fluxo de dados Redux



Fonte: Tutorials Point

Este tipo de arquitetura aumenta muito a escalabilidade para aplicativos grandes e complexos. Também permite poderosas ferramentas de desenvolvedor, porque é possível rastrear todas as mutações na ação que causou o comportamento no componente inspecionado. Com um estado e uma ação, o próximo estado do aplicativo pode ser previsto com absoluta certeza.(Redux[50]).

5.6.2 **REACT-GRID-LAYOUT**

React Grid Layout[51] (RGL abreviadamente) fornece um layout de *grid* de arrastar e soltar e redimensionável com pontos de interrupção responsivos para o React.js. Suporta pontos de interrupção (responsivos) que podem ser fornecidos pelo usuário ou gerados automaticamente. É compatível com aplicativos renderizados pelo servidor, vem com *widgets* arrastáveis e redimensionáveis, pacotes configuráveis podendo redimensionar de forma vertical, ou horizontal. Também é possível definir deixar o componente de forma estática não havendo redimensionamento ou arrasto destes componentes. É permitido ao usuário

50 https://redux.js.org/
51 https://github.com/STRML/react-grid-layout

adicionar e remover *widgets* sem necessidade reconstruir a grade, tornando a biblioteca extremamente performatica.

RGL mantém uma matriz de layout com cada uma das alturas da grade (h), largura (w), coordenadas (x, y) e uma chave (i) e também com parâmetros extras de altura mínima e máxima (minH, maxH) e largura mínima e máxima (minW, maxW). Os códigos 1,2 e 3 exemplificam como fica a estrutura e o uso da biblioteca no contexto de sintaxe de programação.

Código 1 - Layout é um vetor com items como objetos

```
    layout : [
        {i: '1', x: 0, y: 0, w: 1, h: 2, minH: 2, maxH: 2},
        {i: '2', x: 1, y: 0, w: 1, h: 2, minH: 2, maxH: 2},
      ],
```

Fonte: O autor (2019)

Ao arrastar e soltar no grid, o evento *onLayoutChange* é acionado e retorna o *layout* atual e todos os *layouts*.

Código 2 - Evento disparado ao arrastar

```
onLayoutChange = (layout) => {
  this.setState({layout});
}
```

Fonte: O autor (2019)

O evento *onResize* é acionado quando um item do grid é redimensionado, mas também retorna o layout semelhante ao de *onLayoutChange*.

Código 3 - Evento disparado ao redimensionar

```
onResize = (layouts) => {
  this.setState({
    layout: layouts
  });
};
```

Fonte: O autor (2019)

5.6.3 **MATERIAL-UI**

Material-UI[52] é a implementação do *Google* do *Material Design*[53] para o React.js. O *Material Design* é uma linguagem de design desenvolvida em 2014 pelo Google e é muito popular para aplicativos da Web e móveis. O *Material Design* é inspirado no mundo físico e em suas texturas, incluindo como eles refletem a luz e projetam sombras. Com os componentes da biblioteca da interface do usuário do Material, é muito fácil usar os elementos do *Material Design* em aplicações *Web* ou em aplicativos móveis criados com o *React*. (Eschweiler, 2018)

Segundo o Google, o *material design* segue alguns princípios[54], entre eles:

- Material é uma metáfora: O material é fundamentado em uma realidade mais tátil (conectada ao sentido e ao toque) e é inspirado com base no estudo do papel e da tinta. É mais aberto e também imaginativo com o comportamento. Ao contrário do papel real, o material pode se dividir, reorganizar e pode ser movido quando necessário. No momento da interação com os materiais, fornece uma experiência mais realista aos usuários. Atributos como sombras, bordas e dimensionalidade fornecem mais pistas visuais. O uso de atributos táteis familiares ajuda os usuários a entender rapidamente as possibilidades (dicas sobre como o objeto deve ser usado).
- Negrito, intencional e gráfico: Os elementos como tipografia, grades, espaço, escala e cor criam hierarquia, significado e foco. A experiência do usuário aumenta devido às opções de cores, imagens de ponta a ponta e também espaço em branco intencional.
- Movimento fornece significado: A animação é a melhor parte do *material design*, pois não interrompe a experiência do usuário de forma alguma. A animação reforça o fato de o usuário ser o principal motor. As ações principais do usuário são pontos de inflexão que iniciam o movimento, transformando todo o design. O movimento cai em cascata a partir dos pontos de contato e o *feedback* visual parece realmente conectado ao que o usuário fez. A animação torna a experiência do usuário mais natural e imersiva.

5.6.4 **OUTRAS BIBLIOTECAS JAVASCRIPT**

Na tabela 6 é possível visualizar outras bibliotecas que auxiliam no

52 https://material-ui.com/
53 https://material.io/design/
54 https://material.io/design/introduction/#principles

desenvolvimento de aplicação.

Tabela 7 - Outras bibliotecas JavaScript utilizadas

Nome	Função	Site
Redux-form	Formulários como componente do Redux	https://redux-form.com/
Lodash	Diversas funções para manipulação de coleções	https://lodash.com/
React-i18next	Internacionalização e Localização	https://react.i18next.com/
KOA	Framework de servidor em node	https://koajs.com/
Jest	Testes unitários	https://jestjs.io/
shelljs	Permite a chamada de comando de kernel no servidor	https://github.com/shelljs/shelljs
React-Color	Color picker para ReactJS	https://casesandberg.github.io/react-color/

Fonte: O autor (2019)

5.7 **FIREBASE**

O Firebase é um BaaS (*Backend as a Service*) para aplicações Web e Mobile do Google, lançado em 2004, que com o passar dos anos cresceu muito, se tornando uma ferramenta que hoje para alguns projetos é a melhor opção, devido à quantidade de serviços oferecidos por ele, além da facilidade de implementação. (Orlandi, 2019)

O *Firebase Realtime Database*[55] é um banco de dados hospedado na nuvem. Os dados são armazenados como JSON e sincronizados em tempo real com todos os clientes conectados. Quando o usuário cria apps cross-plataforma com SDKs para iOS, Android e JavaScript, todos os clientes compartilham uma instância do *Realtime Database* e recebem automaticamente atualizações com os dados mais recentes.

Os principais recursos do Firebase, segundo o Google, são:

- Funcionamento em tempo real: Em vez de requisições HTTP típicas, o *Firebase Realtime Database* usa a sincronização de dados. Sempre que os dados são alterados, todos os dispositivos conectados recebem essa atualização em milissegundos. Com isso, o desenvolvedor pode criar experiências colaborativas e imersivas sem se preocupar com códigos de comunicação via rede.
- *Off-line*: Os apps do Firebase permanecem responsivos mesmo *off-line*,

55 https://firebase.google.com/docs/database?hl=pt-br

pois o SDK do *Firebase* mantém seus dados em disco. Quando a conectividade é restabelecida, o dispositivo cliente recebe as alterações perdidas e faz a sincronização com o estado atual do servidor.

- Acessível em dispositivos clientes: O Firebase pode ser acessado diretamente de um dispositivo móvel ou navegador da Web, sem um servidor de aplicativos. A segurança e a validação de dados estão disponíveis por meio de regras de segurança baseadas em expressões do *Firebase Realtime Database*, executadas quando os dados são lidos ou gravados.
- Escalabilidade: O Firebase oferece suporte em grande escala às necessidades de dados de apps. Para isso, os dados podem ser divididos entre várias instâncias de banco de dados no mesmo projeto do Firebase.
- Autenticação simplificada: o *Firebase Authentication* simplifica a autenticação de usuários nas instâncias de banco de dados. O acesso às informações em cada banco de dados é controlado com regras personalizadas do Firebase para cada instância de banco de dados.

5.8 **PROVA DE CONCEITO**

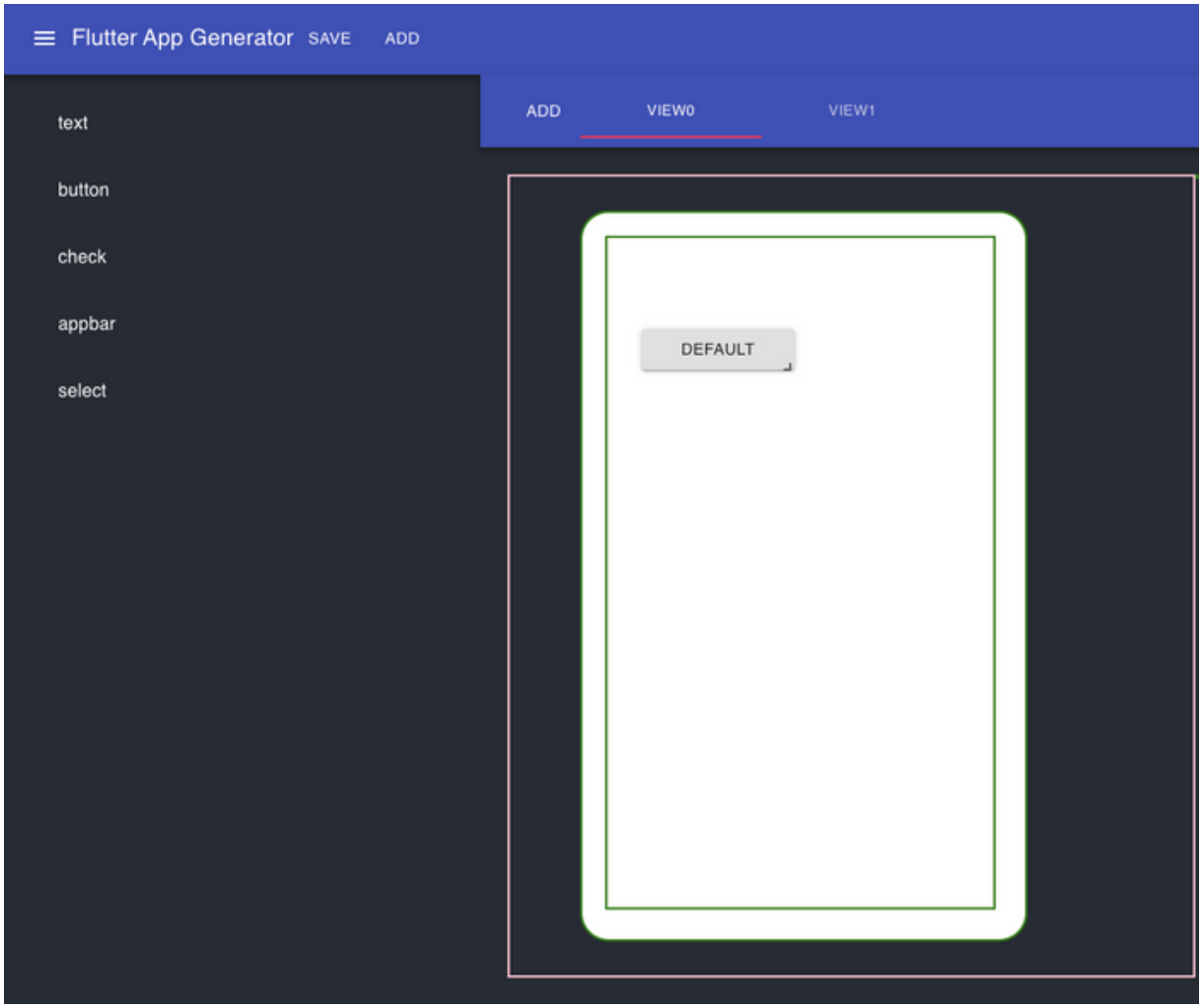
Para testar a viabilidade do projeto foi desenvolvida uma prova de conceito[56] da aplicação dividida em três partes: *frontend*, geração de código e criação do aplicativo móvel. Nenhuma folha de estilo foi aplicada nesta etapa do processo, que visa somente validar o funcionamento básico da aplicação.

5.8.1 **Frontend**

Na primeira etapa foi criada a aplicação web para validar o uso da biblioteca React-Grid-Layout, assim como a estrutura de estado gerenciado pela biblioteca Redux. Nessa etapa foi criada uma listagem de componentes básicos seguindo o protótipo desenvolvido com o editor e uma janela de propriedades para edição básica de componentes. As Figuras 17 e 18 demonstram, respectivamente, a adição de um botão na interface gráfica da aplicação e a definição de propriedades de componentes.

56 https://www.techopedia.com/definition/4066/proof-of-concept-poc

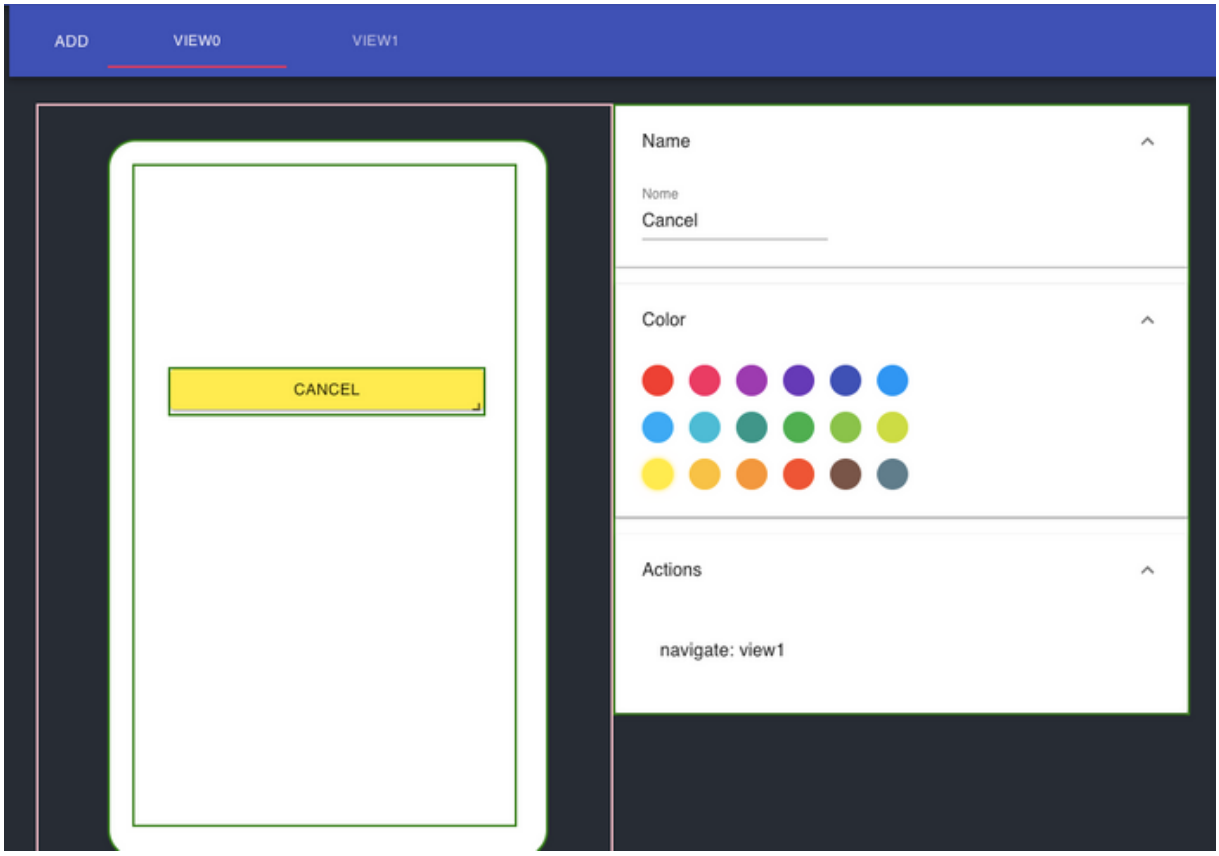
Figura 15 - Estado inicial de um botão



Fonte: O autor (2019)

Ao selecionar um componente, um menu lateral abre à direita para edição do componente e atribuição de ações a este componente. O componente ganha uma borda verde para mostrar que este é o componente selecionado. A figura abaixo mostra o efeito de clique em um componente no editor.

Figura 16 - Menu de propriedades do componente



Fonte: O autor (2019)

5.8.2 **Backend**

No *backend* foi criada uma API usando Node.js com o *framework* Koa[57], que foi criado pela mesma equipe que criou o Express[58] que visa ser uma base menor, mais expressiva e mais robusta para aplicativos da Web e APIs. Ao aproveitar as funções assíncronas, o Koa permite que o usuário evite retornos de chamada e aumente bastante o tratamento de erros. Para fazer o roteamento entre o *frontend* e a lógica da aplicação foi criada uma rota na API REST com a qual o cliente faz uma requisição usando o método POST, passando como parâmetro a estrutura de dados com as interfaces criadas no *frontend* juntamente com seus respectivos componentes e configurações. Com servidor em Node.js também é mais simples ler e escrever arquivos, que é uma premissa básica para um projeto de geração de código. O código 4 demonstra um exemplo de uma rota criada no framework Koa.

57 https://koajs.com/
58 https://expressjs.com/pt-br/

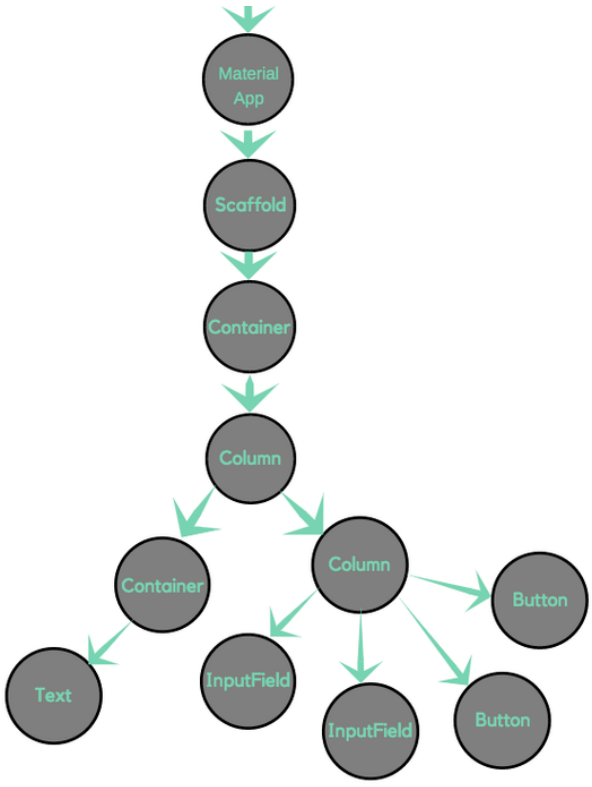
Código 4 - Koa router

```
module.exports = (router) => {
    router.prefix('/v1')
    router.use('/generate', require('./generate'))
}
```

Fonte: O autor (2019)

A arquitetura de uma aplicação escrita em Dart/Flutter parte de uma abordagem de Pai e Filhos, segundo a qual um primeiro componente é instanciado e, a partir daí, um ou mais filhos são gerados dependendo do componente raiz. O funcionamento é semelhante à representação de uma árvore B[59] tendo em vista que todos os componentes abaixo do primeiro têm uma ligação direta com o nodo raiz acima deles. A figura 19 ilustra um exemplo de como fica a árvore de componentes em Dart/Flutter.

Figura 17 - Árvore de componentes de interface Flutter



Fonte: N Bisarahalli (2018)

A estrutura de árvore permitiu uma modelagem da solução baseada na criação e inserção de nós, em que cada nó tem como propriedade o número mínimo

59 https://ime.usp.br/~pf/estruturas-de-dados/aulas/B-trees.html

e máximo de filhos, e os filhos permitidos para o componente. O objeto nó possui as seguintes propriedades:

- template: Scaffold do código na linguagem Dart (arquivo .dart).
- minChildren: o número mínimo de filhos que esse nó pode conter.
- maxChildren: o número máximo de filhos que esse nó pode conter.
- children: vetor de componentes que podem ser inseridos como filhos deste nó.

O código 5 representa as propriedades de um nó *Button* (Botão).

Código 5 - Estrutura de um nó

```
const Button = {
    template: getTemplate(),
    minChildren: 0,
    maxChildren: 1,
    children: ['Text', 'Decorator', 'Icon']

}
```

Fonte: O autor (2019)

A estrutura de árvore foi concebida partindo do principio que cada nó da árvore é um componente a ser inserido no *layout*. Cada componente tem suas propriedades e suas regras de inserção, facilitando a validação e a construção da mesma.

Quando um nó é inserido, a função "getTemplate" é acionada. Essa função é responsável por buscar o *scaffold* específico do componente recém inserido na árvore, preenchendo a propriedade "template" do objeto nó com o código pertencente àquele componente. O código 6 exibe o template do nó botão inserido anteriormente.

Código 6 - Template de um botão

```
return `FlatButton(
        color: ${colorString},
        textColor: Colors.white,
        disabledColor: Colors.grey,
        disabledTextColor: Colors.black,
        splashColor: Colors.blueAccent,
        onPressed: () {
            ${actionPressed}
        },
        child:
            Text(
                "${text}",
                style: TextStyle(fontSize: 20.0),
            ),
    )`;
```

Fonte: O autor (2019)

Conforme demonstrado nos diagramas 2, 3, 4 e 5 da sessão 4.2.4, a aplicação itera por cada interface criada e a seguir por cada componente. A aplicação monta uma árvore de componentes com *scaffolds* seguindo as propriedades definidas pelo usuário na edição de *layout* no *frontend.* Quando esta árvore está finalizada, a aplicação busca um modelo de template adequado para a interface que vai ser gerada. Este modelo contém textos de marcação que serão posteriormente substituídos pelos *scaffolds* obtidos no processo de conversão. O código 7 exibe um modelo padrão de template Dart. As palavras iniciadas com o símbolo "$" são as que serão substituídas durante a geração do código.

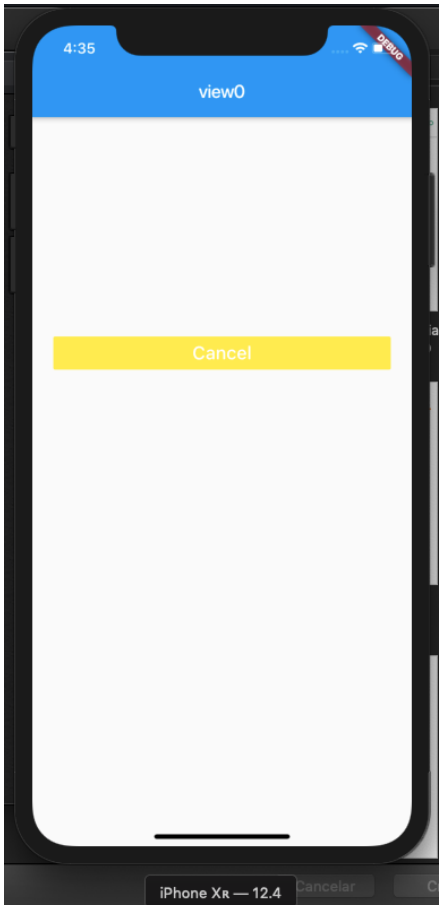
Código 7 - Modelo padrão de código .dart

```
import 'package:flutter/material.dart';
$import

class $title extends StatefulWidget {
  $title({Key key}) : super(key: key);

  @override
  _$titleState createState() => _$titleState();
}

class _$titleState extends State<$title> {
  $variavel
  @override
  Widget build(BuildContext context) {
    return Scaffold(
        appBar: AppBar(
          title: Text("$title"),
        ),
        body: Container(
            padding: EdgeInsets.all(15),
            child: Column(
              mainAxisAlignment: MainAxisAlignment.spaceBetween,
              children: <Widget>[$components],
            ),
        )
    );
  }
}
```

Fonte: O autor (2019)

Após o processo de substituição de variáveis, o código Dart está pronto para ser compilado. Para a prova de conceito, o processo de construção do aplicativo foi feito de forma manual, gerando os arquivos em uma pasta de uma aplicação Flutter e executado o comando "flutter run" para construir o aplicativo. A Figura 20 exibe o aplicativo já no simulador de iOS com o botão gerado no processo.

Figura 18 - Aplicativo



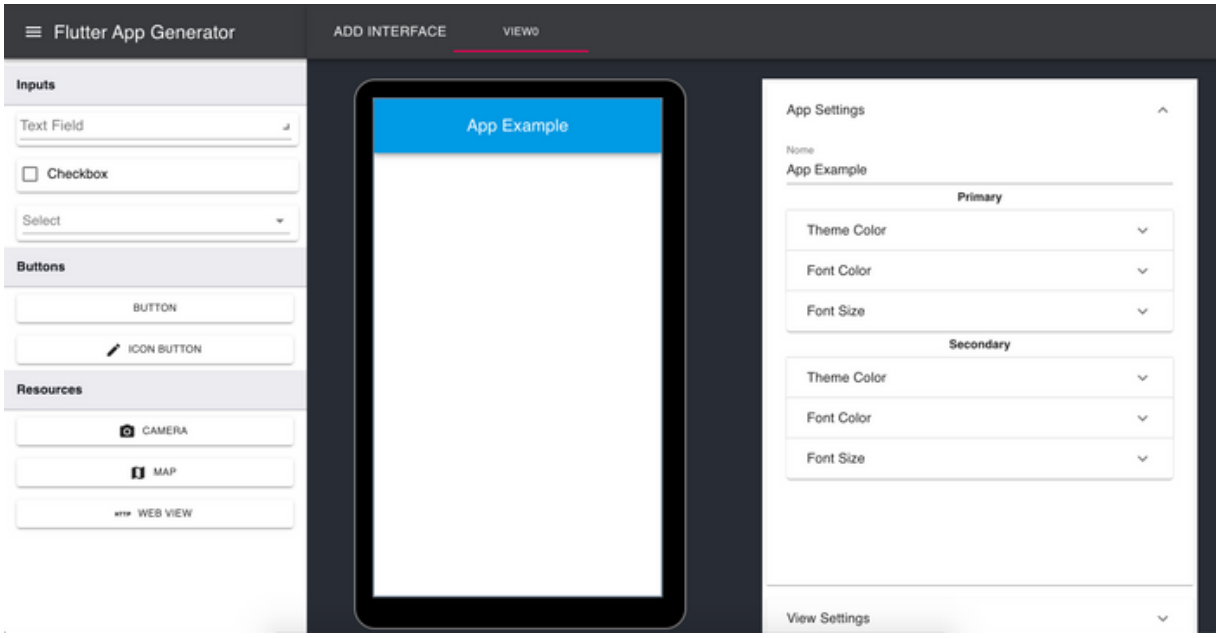
Fonte: O autor (2019)

5.9 VERSÃO FINAL DA FERRAMENTA

Neste capítulo é abordada a fase final de construção da aplicação, assim como as imagens finais do desenvolvimento e da estrutura do projeto. Foi usado como base na aplicação final o código escrito na prova de conceito da aplicação, utilizando da arquitetura descrita anteriormente.

O escopo definido para a aplicação final foi de aumentar a personalização de estilo da aplicação e as propriedades de configurações de componentes realizadas na prova de conceito. Devido a uma limitação de tempo para desenvolvimento da ferramenta, optou-se por uma abordagem de viabilizar uma aplicação básica, com navegação entre interfaces, inserção de componentes e utilização de recursos que são nativos das plataformas *Android* e *iOS.* Utilizando esta abordagem, o trabalho abrange todo o ciclo básico de desenvolvimento de uma aplicação *cross-plataform*. A figura 21 demonstra a tela inicial da aplicação.

Figura 19 - Versão final da aplicação

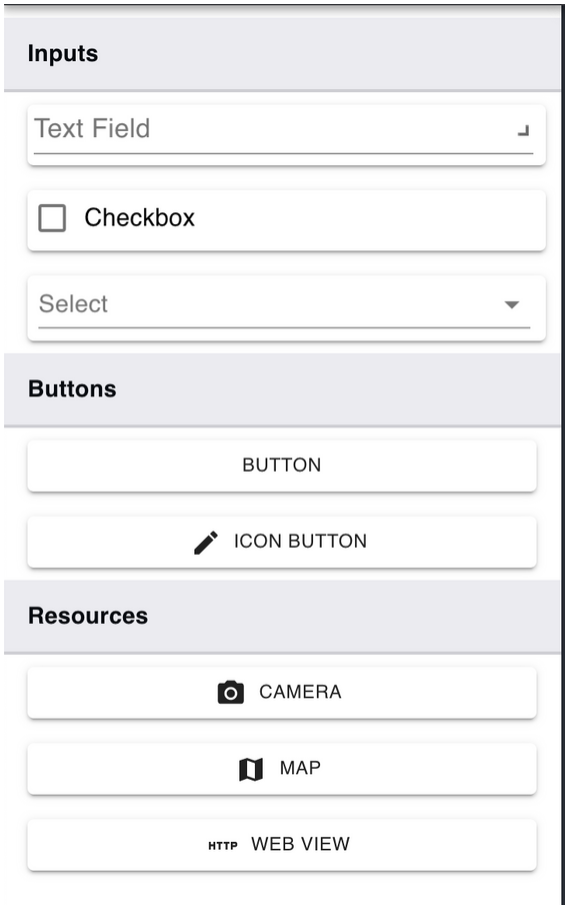


Fonte: O autor (2019)

5.9.1 **Menu de componentes**

Foram realizadas diversas melhorias de interface na aplicação tendo como objetivo melhoria na usabilidade. Uma delas foi categorizar os componentes por tipo e utilizar imagem do componente a ser adicionado junto com seu nome para facilitar o entendimento do usuário, exibindo uma pré-visualização do componente que está para ser selecionado. A figura 22 demonstra o resultado final da lista de componentes categorizada e renderizada com suas respectivas imagens.

Figura 20 - Menu com imagem dos componentes



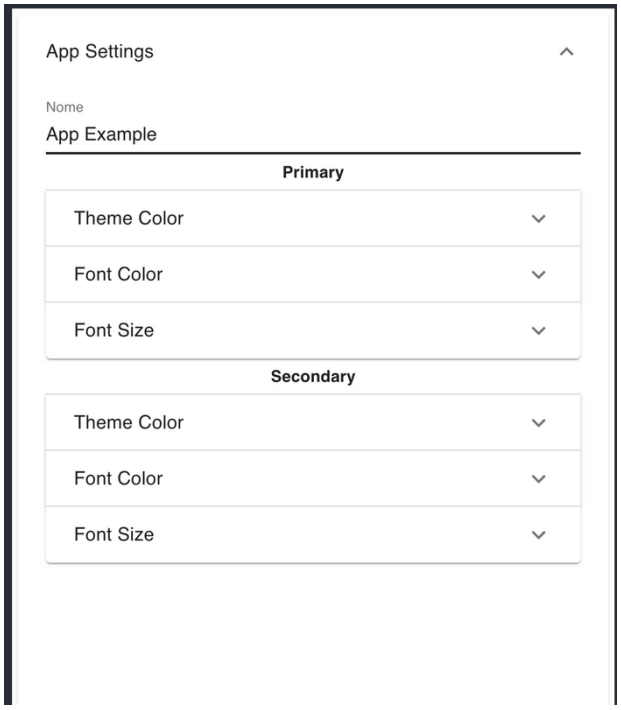
Fonte: O autor (2019)

5.9.2 **CONFIGURAÇÃO DA APLICAÇÃO**

Uma sessão de configuração global da aplicação foi adicionada nas propriedades da primeira interface do editor de aplicativos. Nela é possível registrar um nome para a aplicação e mudar os estilos primários e secundários seguindo as guias de estilo e cores do Material Design[60]. Foram adicionadas configurações de estilo globais para aplicação. Nelas é possível mudar a cor do tema, a cor da fonte e o tamanho da fonte. A figura 23 exibe a aba de configuração global com os menus expansíveis de edição de tema.

60 https://material.io/design/color/the-color-system.html#

Figura 21 - Configurações globais de estilo



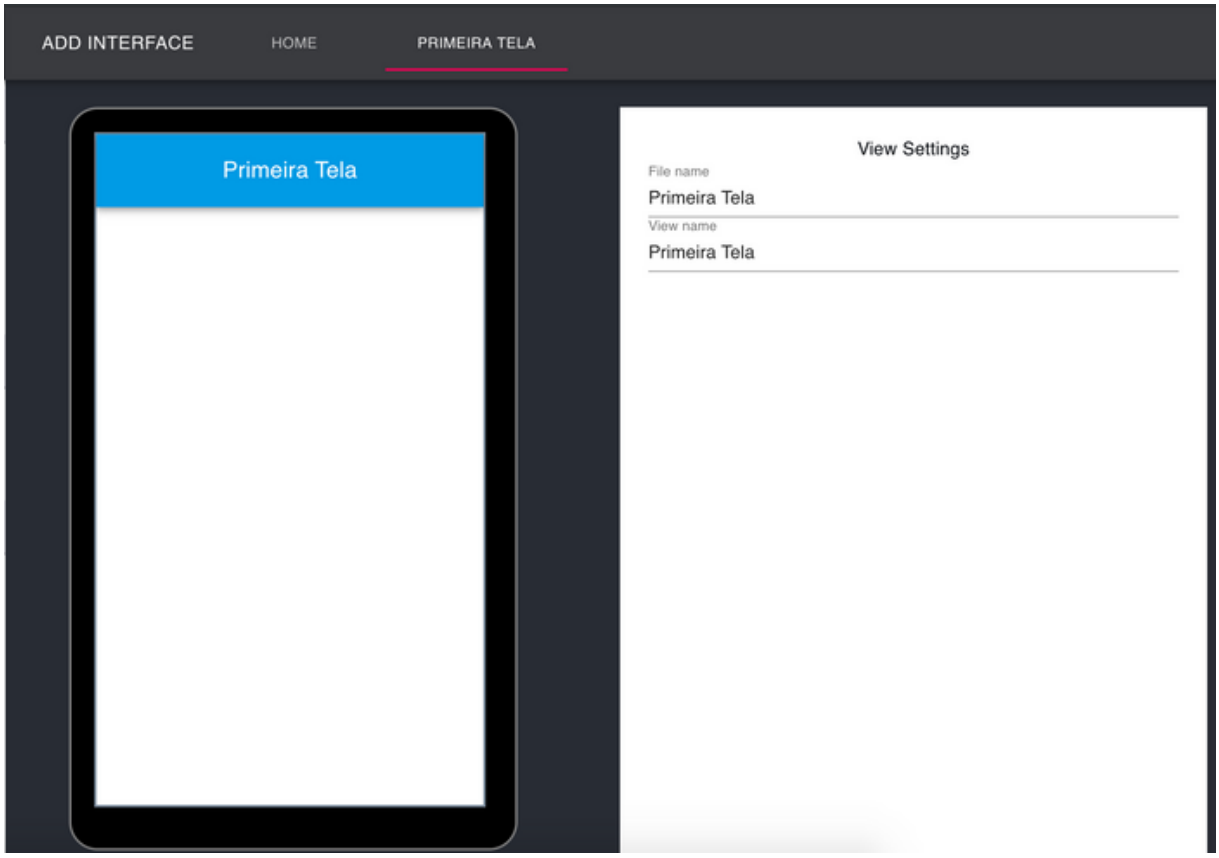
Fonte: O autor (2019)

5.9.3 **Menu de configuração de interface**

Uma aba de configuração de interface foi adicionada para nomear cada nova interface adicionada na aplicação. Na figura 24 é possível verificar que o nome inserido no campo de configuração da interface foi inserida na *AppBar*[61] e na guia[62] que identifica a interface.

61 https://material-ui.com/pt/components/app-bar/
62 https://material-ui.com/pt/components/tabs/

Figura 22 - Configuração de interface



Fonte: O autor (2019)

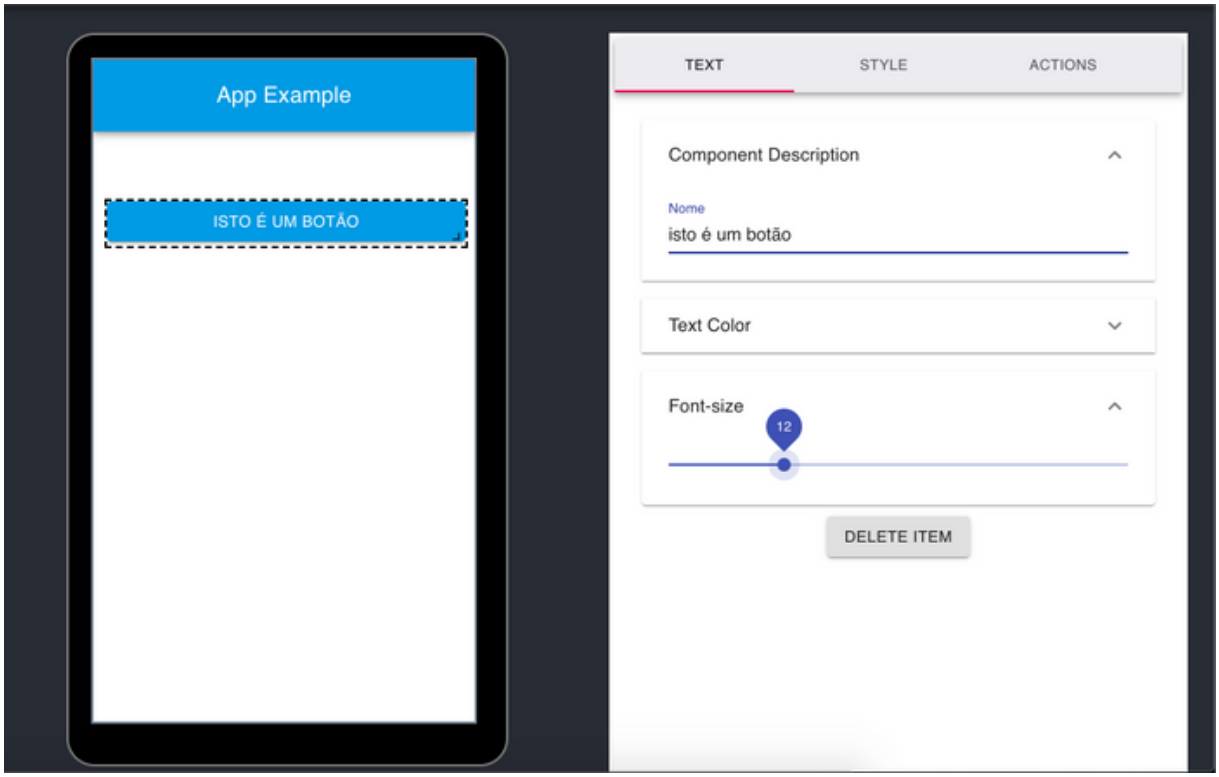
5.9.4 **Configuração de componente**

Toda vez que um componentes é inserido e selecionado, o menu lateral de propriedades muda o aspecto para atender as configurações do componente em questão. Neste menu é possível mudar propriedades de texto, estilo e ações para atender as expectativas do usuário.

5.9.4.1 Configuração de texto

Na aba de texto da guia de configurações é possível mudar o nome de exibição do componente, a cor do texto e também o tamanho da fonte. Estas ações são executadas em tempo real, não havendo a necessidade de nenhuma ação para salvar estas alterações. A figura 25 exibe o texto inserido nas configurações inseridas automaticamente no componente selecionado.

Figura 23 - Editor de propriedades do componente

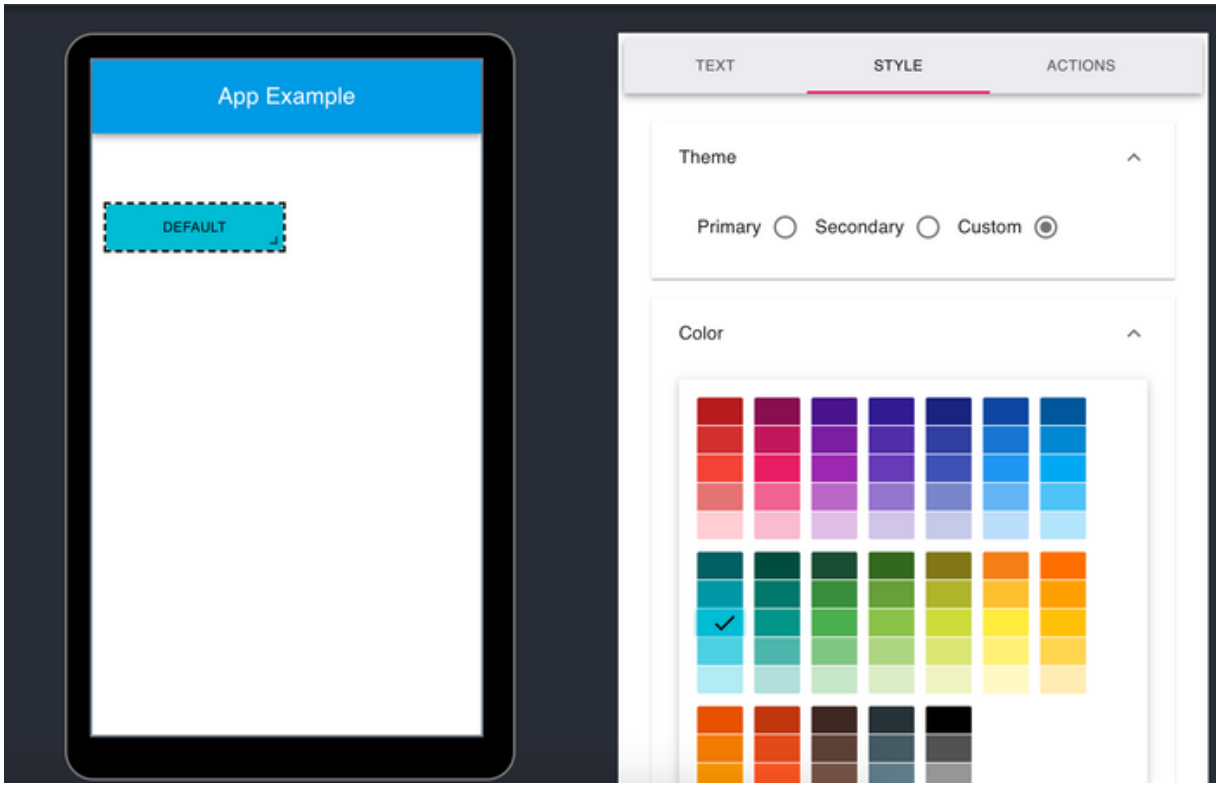


Fonte: O autor (2019)

5.9.4.2 Configuração de estilo

Na guia de configuração de estilo é possível customizar as cores do componente. No menu de seleção de temas é possível escolher entre três tipos de estilo. O primário e secundário utilizam das cores de tema selecionados no menu de configuração global da aplicação, enquanto o customizado permite que usuário selecione a cor para o componente na paleta de cores. A figura 26 demonstra a customização de cor do componente.

Figura 24 - Configuração de estilo do componente

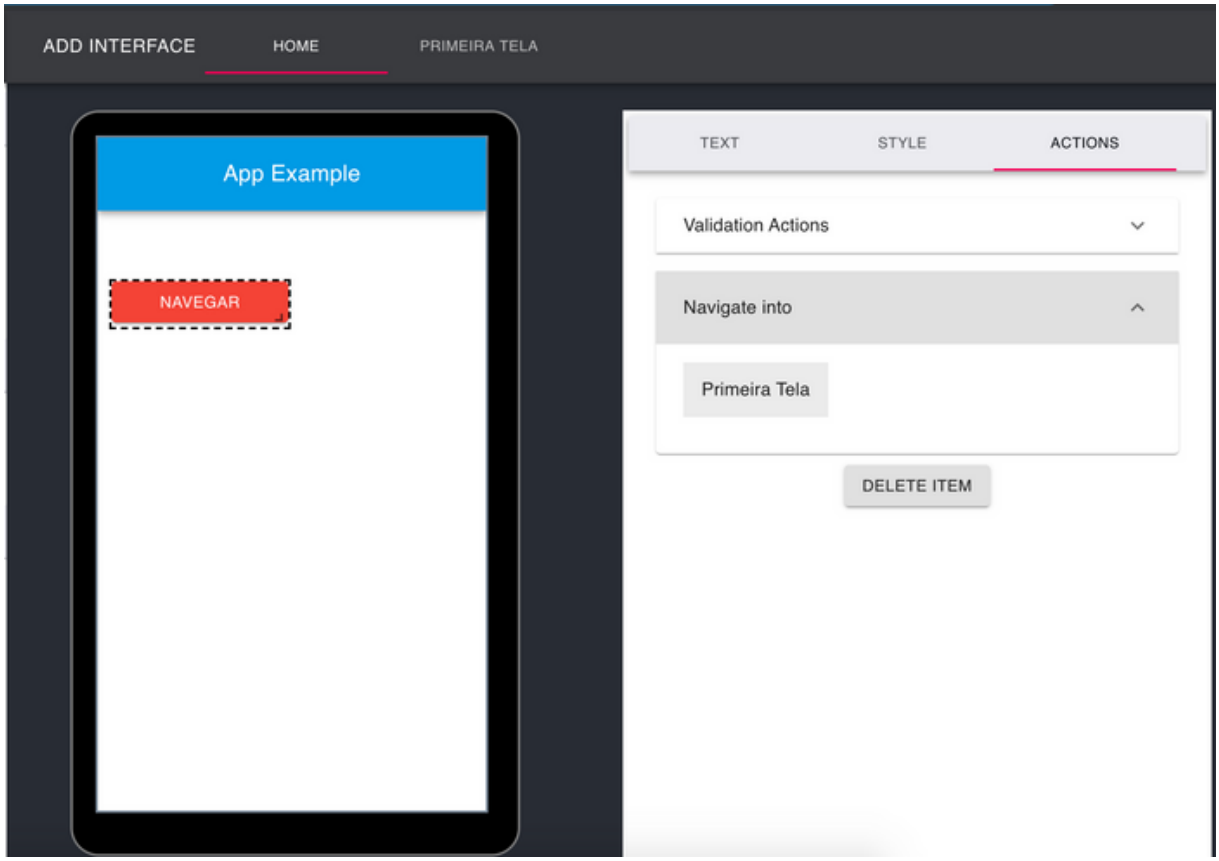


Fonte: O autor (2019)

5.9.4.3 Configuração de comportamento de componente

Na guia de ações é possível inserir ações que alteram comportamentos da aplicação em componentes de permitem este tipo de ação e também é permitido inserir expressões regulares para validação em campos de texto. A figura 27 exibe o comportamento de navegação atribuído ao componente selecionado.

Figura 25 - Menu de configuração de comportamento do componente



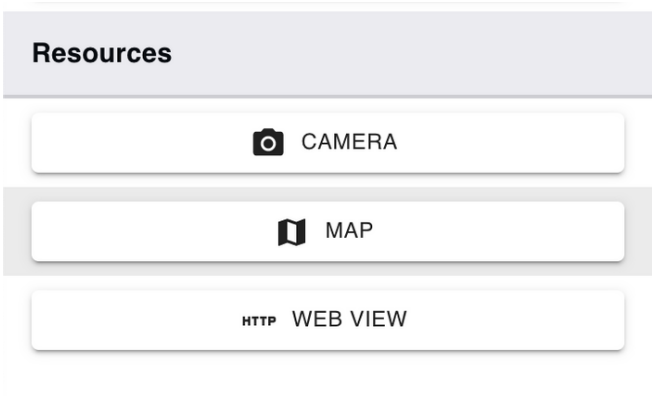
Fonte: O autor (2019)

5.9.5 **Componentes de recursos nativos**

O desenvolvimento de uma aplicação cross-platform tem como uma das premissas o fácil acesso a recursos nativos das plataformas. Segundo (XANTHOPOULOS; XINOGALOS, 2013, p. 2) o objetivo final do desenvolvimento de aplicativos móveis cross-plataforma é alcançar o desempenho nativo do aplicativo e executar recursos nativos da plataforma. Foram adicionados como componentes da aplicação três recursos nativos das plataformas móveis para utilizar na construção do aplicativo. Os recursos selecionados para integrar a plataforma foram recursos de mapa e de câmera. Também foi adicionado um recurso de *webview*[63] para acesso de conteúdo externo da *web*. A figura 28 exibe os itens do menu de lista de recursos nativos.

63 https://developer.chrome.com/multidevice/webview/overview

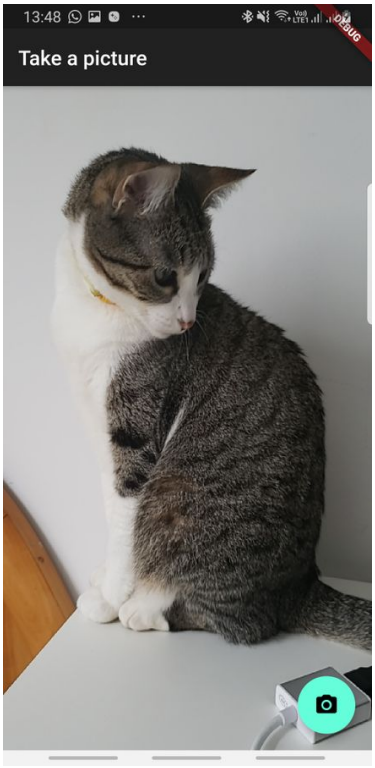
Figura 26 - Componentes de recursos nativo



Fonte: O autor (2019)

As figuras 29, 30 e 31 demonstram os componentes de recursos em um aplicativo móvel gerado pela ferramenta funcionando em um celular *Android.*

Figura 27 - Recurso de câmera



Fonte: O autor (2019)

Figura 28 - Recurso de mapa

Fonte: O autor (2019)

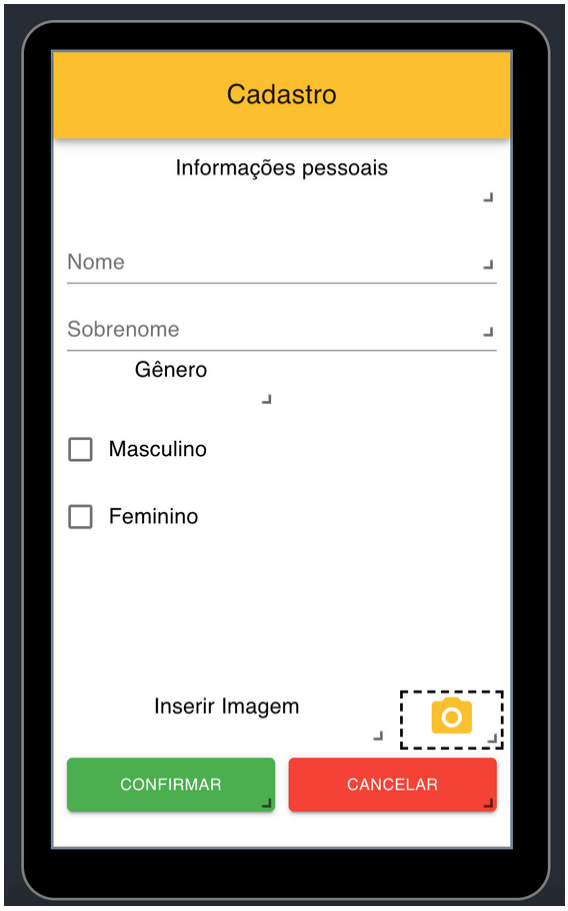
Figura 29 - Recurso de webview

Fonte: O autor (2019)

Foi disponibilizado na aplicação web uma sessão de documentação e utilização da ferramenta com o intuito de esclarecer possíveis dúvidas do usuário quanto à forma correta de uso da aplicação *web* e suas principais funcionalidades.

Como resultado de todo o processo de criação de layouts é possível criar diferentes modelos de interface para aplicativos móveis. A figura 32 exemplifica um cadastro simples criado na ferramenta de edição de layouts.

Figura 30 - Layout editado pela ferramenta



Fonte: O autor (2019)

5.10 **LIMITAÇÕES E FALHAS CONHECIDAS**

Assim como toda versão inicial de uma ferramenta, o artefato fruto deste trabalho contém algumas limitações que podem ser corrigidas em versões futuras. Assim como bugs encontrados durante os testes, a seguir serão listados alguns desses problemas. Os erros descritos foram capturados em sua grande maioria em testes de interface realizados pelos usuários da ferramenta e reportados através de contato pessoal. Todos os problemas listados podem vir a serem corrigidos em versões futuras.

- Alinhamento de textos: Pela escolha do layout em grade, existem algumas limitações de alinhamento de componentes e textos.
- Número limitado de ícones: Há uma limitação no número de ícones conhecidos do *Material Design* disponibilizados pela ferramenta.
- Desalinhamento do layout: Em algumas situações, quando se insere um novo componente na interface, os componentes já inseridos desalinham, de forma que precisam ser realinhados manualmente pelo usuário.
- Layout flexível: Em alguns modelos de aparelhos, quando o teclado é exibido, há uma quebra de layout devido ao número de linhas de tamanho fixo aplicadas no layout em grid.
- Geração de aplicação para iOS: Para criar uma aplicação que rode em um dispositivo iOS, é necessário uma conta de desenvolvedor da Apple para submeter a aplicação para aprovação dos moderadores da loja. Por isso somente é possível rodar aplicações iOS no modo de simulador.

6 **CONCLUSÕES E TRABALHOS FUTUROS**

6.1 **CONCLUSÃO**

O desenvolvimento de aplicativos para dispositivos móveis é um campo em evolução que atrai grande interesse. O tipo mais comum de aplicativos móveis era, até recentemente, aplicativos móveis nativos. Os aplicativos móveis nativos são desenvolvidos para uma plataforma específica e oferecem, sem dúvida, a melhor experiência do usuário. Nos aplicativos móveis nativos, o código-fonte é eficiente, com desempenho rápido, aparência e comportamento consistentes e acesso total aos dados e hardware da plataforma subjacente. O desenvolvimento de um aplicativo móvel nativo para várias plataformas requer uma implementação distinta para cada plataforma, um fenômeno conhecido como fragmentação.

A proposta do presente trabalho em criar uma ferramenta de geração de aplicativos *cross-platform* para resolver um problema bem conhecido quanto ao desenvolvimento para diversas plataformas foi concluída com êxito. O objetivo de criar um gerador de aplicativos móveis multiplataforma foi alcançado. Mesmo não contendo uma gama de componentes e configurações muito grande, todos os principais recursos de uma aplicação móvel está contida no editor. É possível através da ferramenta gerar aplicativo que roda em iOs e Android, capaz de acessar recursos nativos em ambos sistemas operacionais, realizar operações como navegação e validação de campos de texto, e construir um layout de aplicativo móvel que seja fiel a interface criada no editor online.

A ferramenta web foi projetada e construída com intuito de ser facilmente escalável. Idealizada em módulos e componentes, sua arquitetura comporta mudanças e atualizações de maneira simples, sem necessidade de realizar grandes mudanças no código existente. Utilizando recursos de ambientes web modernos, como banco de dados em nuvem, contêineres para garantir execução em qualquer ambiente e integração contínua, a aplicação web cumpre seu objetivo de manter um ambiente de criação de aplicativos utilizando o editor de interface.

Uma análise comparativa com as principais características da ferramenta desenvolvida comparada com as ferramentas estudas no capítulo 3 deste trabalho. Esta análise tem como objetivo principal posicionar a aplicação no contexto da análise feita anteriormente. A tabela 8 demonstra as principais características entre as ferramentas analisadas e a ferramenta proposta.

Tabela 8 - Análise comparativa entre ferramentas (continua)

Ferramenta	Tipo de Licença	Tipo de Desenvolvimento	Multiplataforma	Gratuita

Tabela 8 - Análise comparativa entre ferramentas (conclusão)

Ferramenta	Tipo de Licença	Tipo de Desenvolvimento	Multiplataforma	Gratuita
AppInventor	Código Aberto	Híbrido	Não	Sim
Seattle Cloud	Código Fechado	Nativo/ Híbrido	Sim	Não
Thunkable	Código Fechado	WebApp	Sim	Não
Andromo	Código Fechado	Híbrido	Não	Não
Flutter Studio	Código Fechado	Nativo/ Híbrido	Sim	Sim
Ferramenta Proposta	Código Aberto	Nativo/ Híbrido	Sim	SIm

Fonte: O autor (2019)

A ferramenta proposta é a única das soluções que é gratuida, multiplataforma e de código aberto. Por ser uma aplicação gratuita e de código aberto, a solução pode ser melhorada por qualquer pessoa que assim desejar. A ampliação do número de componentes e estilização são necessárias para deixar a ferramenta mais robusta. Estas mudanças tornam a ferramenta não só útil para gerar aplicações multiplataforma simples, como a torna também um ambiente de aprendizagem para desenvolvimento de aplicativos móveis.

Por fim, este trabalho proporcionou uma grande base de conhecimento em desenvolvimento de aplicações web e móveis cross-plataforma*,* geração de código através de *scaffolds,* soluções de geração de aplicações e tecnologia no geral e diversas tecnologias que compuseram todo o ambiente de desenvolvimento necessário para produzir este trabalho.

6.2 **TRABALHOS FUTUROS**

A ferramenta foi concebida com o intuito de ser estendida e continuada. Por isso, todo o código está disponível em um servidor web na forma de um projeto no Github[64]. O código fonte do projeto está disponível em https://github.com/RonaldoDev/flutterappgenerator. Neste endereço, na aba "*Issues*", podem ser colocadas dúvidas quanto à execução e ao código. O apêndice A do presente trabalho contém um guia de como adicionar um novo componente na aplicação.

Existem diversas melhorias que podem ser feitas na ferramenta, entre as principais são:

- Inserir mais componentes para dar a opção de criar aplicativos mais ricos;
- Inserir mais propriedades de componentes permitindo assim uma melhor estilização dos aplicativos gerados;
- Inserir mais tipos de comportamentos, pois atualmente a ferramenta só

64 https://github.com/about

conta com navegação e validação de campos de texto;

- Usar um serviço de PaaS ou banco de dados para persistir dados dos aplicativos gerados pela ferramenta;
- Melhoria de visual da ferramenta para melhor usabilidade dos usuários;
- Operações de "Undo" e "Redo" na aplicação" ;
- "*Live Coding", ou seja,* geração de código em tempo real com a opção de o usuário editar esse código a partir do *frontend* e não somente após o aplicativo gerado;
- Realizar *benchmark* entre abordagens nativas de desenvolvimento e aplicativos gerados pela plataforma.
- Refatoração do código;

A melhorias / extensões podem ser feitas tanto no *frontend* quanto no *backend*. Concluídas estas melhorias, a ferramenta proposta se igualaria às melhores soluções pagas e gratuitas existentes atualmente para criação de frontends de aplicativos móveis.

REFERÊNCIAS

A. YNCHAUSTI, Randy. Integrating Unit Testing Into A Software Development Team's Process. **Bountiful**, 2001.

ABBAS, A. S.; JEBERSON, W. ; KLINSEGA, V.. A literature review and classification of selected software engineering researches. **International Journal of Engineering and Technology**, v. 2, n. 7, p. 1, 2012.

ABRAMOVA , Veronika; BERNARDINO, Jorge. NoSQL Databases: MongoDB vs Cassandra . *In:* PROCEEDINGS OF THE INTERNATIONAL C* CONFERENCE ON COMPUTER SCIENCE AND SOFTWARE ENGINEERING. 2013, Porto, Portugal: ACM, 2013. 14-22 p.

AGOSTINI.TECH. **Flutter – an overview of cross-platform framework**. **Agostini.Tech**. 2019. Disponível em: https://agostini.tech/2019/04/14/flutter-overview-cross-platform-framework/. Acesso em: 20 Jun. 2019.

ARSHAD, A. *et al*. An Empirical Study of Investigating Mobile Applications Development Challenges. **SPECIAL SECTION ON SOFTWARE STANDARDS AND THEIR IMPACT IN REDUCING SOFTWARE FAILURES**, Beijing , China, 2018.

AXELSSON , Oscar; CARLSTRÖM, Fredrik . **Evaluation Targeting React Native in Comparison to Native Mobile Development**. Lund, Sweden, 2016. Dissertação (INTERACTION DESIGN) - LUND UNIVERSITY, 2016.

BANERJEE, Ishan; NGUYEN, Bao. Graphical user interface (GUI) testing: Systematic mapping and repository. *In:* INFORMATION AND SOFTWARE TECHNOLOGY. 55. ed. 2013. 1679-1694 p.

BAVOTA, Gabriele; QUSEF, Abdallah; OLIVETO, Rocco. An empirical analysis of the distribution of unit test smells and their impact on software maintenance.. *In:* IEEE INTERNATIONAL CONFERENCE ON SOFTWARE MAINTENANCE,. 2012. 2012. 56-65 p.

BOUSHEHRINEJADMORADI, N. *et al*. ‘Testing cross-platform mobile app development frameworks (T). *In:* PROC. 30TH IEEE/ACM INT. CONF. AUTOM. SOFTW. ENG. 2015. 2015. 441–451 p.

CHARLAND, A.; LEROUX, B.. Mobile application development: Web vs. native. **Commun. ACM**, v. 54, n. 49-53, 2011.

CHARLAND, Andre; LEROUX, Brian. Mobile Application Development: Web vs. Native. **Communications of the ACM**, Nova Iorque, v. 54, n. 5, p. 49-53, Maio 2011.

CORRAL, Luis; JANES, Andrea; REMENCIUS, Tadas. Potential advantages and disadvantages of multiplatform development frameworks: A vision on mobile environments . **Procedia Computer Science**, Bolzano, v. 10, 2012.

CRAWFORD POKRESS, Shaileen; DOMINGUEZ VEIGA, José Juan. MIT App Inventor: Enabling Personal Mobile Computing. **Human-Computer Interaction** ,

2013.

DAHL, Ryan. Node.js*. In:* . 2009.

ESCHWEILER, Sebastian. **Getting Started With Material-UI For React (Material Design For React)**. **Medium**. 2018. Disponível em: https://medium.com/codingthesmartway-com-blog/getting-started-with-material-ui-for-react-material-design-for-react-364b2688b555. Acesso em: 25 Nov. 2019.

FLANAGAN, David. **JavaScript**: the definitive guide. 3. ed. O'Reilly Media., 2006.

FONG, Elizabeth; OKUN, Vadim. Web Application Scanners: Definitions and Functions*. In:* 40TH ANNUAL HAWAII INTERNATIONAL CONFERENCE ON SYSTEM SCIENCES (HICSS'07). 2007, Waikoloa, HI, USA, 2007.

GARCIA-PEÑALVO, F. J.; GARCIA-HOLGADO, A.. Open Source Solutions for Knowledge Management and Technological Ecosystems*. In:* IGI GLOBAL. 2016.

GEARY , David . **Introducing Redux**: Get a grip on your JavaScript application state. **IBM Developer**. 2016. Disponível em: https://developer.ibm.com/tutorials/wa-manage-state-with-redux-p1-david-geary/. Acesso em: 25 Nov. 2019.

GLOBAL STATS. **Desktop vs Mobile vs Tablet Market Share Worldwide**. 2019. Disponível em: http://gs.statcounter.com/platform-market-share/desktop-mobile-tablet. Acesso em: 25 Nov. 2019.

GUO, J.; MUKUNDAN, R.; BILLINGHURST, M. “Developing mobile phone AR applications using J2ME*. In:* 23RD INTERNATIONAL CONFERENCE ON IMAGE AND VISION COMPUTING NEW ZEALAND, New Zealand, 2008.

HARDESTY, L.. **The MIT roots of Google’s new software**. **MIT News**. Massachusetts, 2010. Disponível em: http://news.mit.edu/2010/android-abelson-0819. Acesso em: 25 Nov. 2019.

HEITKÖTTER , H.; HANSCHKE , S.; MAJCHRZAK , T.A.. Evaluating Cross-Platform Development Approaches for Mobile Applications*. In:* CORDEIRO . 2013, Berlin: Springer.

HERRON, David . **Node.js Web Development**. 4. ed. Packt, 2018.

HICKSON, Ian. **The Engine architecture**. **Github**. 2018. Disponível em: https://github.com/flutter/flutter/wiki/The-Engine-architecture. Acesso em: 28 Jun. 2019.

HOLZER, A.; ONDRUS, J.. ‘Mobile app development: Native or Web?*. In:* PROC. WORKSHOP EBUS. (WEB). 2012. 2012.

IOANNIS, Ioannis; KYRIAKOU, Kyriakos-Ioannis; TSELIKAS, Nikolaos. Is Node.js a viable option for building modern web applications?: A performance evaluation study. **Computing**, Tripolis, Greece, 15 Mar 2015.

JAIMINI, Utkarshani; DHANIWALA, Mayur. JavaScript empowered Internet of Things*.*

In: 2016 3RD INTERNATIONAL CONFERENCE ON COMPUTING FOR SUSTAINABLE GLOBAL DEVELOPMENT (INDIACOM). 2016, New Delhi, India, 2016.

JAVASCRIPT libraries. Disponível em: https://webplatform.github.io/. Acesso em: 25 Nov. 2019.

JOBE, Willian. Native Apps vs. Mobile Web Apps . **International Journal of Interactive Mobile Technologies**, Stockholm, 2013.

JOORABCHI, M.E.; MESBAH , A.; KRUCHTEN, P.. Real Challenges in Mobile App Development*. In:* ACM / IEEE INTERNATIONAL SYMPOSIUM ON EMPIRICAL SOFTWARE ENGINEERING AND MEASUREMENT. 2013, Baltimore, 2013. 15-24 p.

KELLY, Steven; TOLVANEN, Juha-Pekka. **Domain-Specific Modeling**: Enabling Full Code Generation. 1. ed. New Jersey: John Wiley & Sons, 2008.

LACHGAR, Mohamed; ABDALI, Abdelmounaïm. Decision Framework for Mobile Development Methods. **A) International Journal of Advanced Computer Science and Applications**, Marrakech, v. 8, 2017.

MAINKAR, Prajyot . *In:* MAINKAR, Prajyot ; GIORDANO, Salvatore . **Google Flutter Mobile Development** : Quick Start Guide. 1. ed. Packt, 2019.

MAINKAR, Prajyot ; GIORDANO, Salvatore . **Google Flutter Mobile Development** : Quick Start Guide. 1. ed. Packt, 2019.

N BISARAHALLI, Manoj. **Flutter hands-on**: Building a Live Location Sharing App. 2018. Disponível em: https://blog.geekyants.com/flutter-hands-on-building-a-live-location-sharing-app-14b67ef17404. Acesso em: 27 Set. 2019.

NIYAR, Noor Ali ; HUSSAIN, Mansoor. **Cross Platform Mobile Application Development Framework**. Karachi, Pakistan. Monografia () - Fast-NUCES (Karachi Campus).

NODE.JS FOUNDATION. **Keeping the Node.js core small**. **Medium**. 2017. Disponível em: https://medium.com/the-node-js-collection/keeping-the-node-js-core-small-137f83d18152. Acesso em: 20 Jun. 2019.

O GOULART MAGNO, Danillo. **APLICAÇÃO DA TÉCNICA DE SCAFFOLDING PARA A CRIAÇÃO DE SISTEMAS CRUD**. 2015. Dissertação (Sistemas de Computação) - Unifei, 2015.

OCCHINO, T.. **React Native**: Bringing modern web techniques to mobile. **Code Facebook**. 2015. Disponível em: https://code.fb.com/android/react-native-bringing-modern-web-techniques-to-mobile/. Acesso em: 28 Jun. 2019.

ORLANDI, Claudio. **Firebase**: serviços, vantagens, quando utilizar e integrações. **Rocketseat**. 2019. Disponível em: https://blog.rocketseat.com.br/firebase/. Acesso em: 25 Nov. 2019.

ORTIZ, Sixto. Computing Trends Lead to New Programming Languages. **IEEE**

Magazine, v. 45, 2012.

PHONEGAP*. In:* ALLEN, Sarah; GRAUPERA, Vidal; LUNDRIGAN, Lee. **Pro Smartphone CrossPlatform Development** . 1. ed. Springer Science, 2010, p. 131-138.

RICH, C.; WATERS, R.C. . Automatic programming: myths and prospects. **Computer**, v. 21, p. 40-51, Agosto 1988. Disponível em: http://ieeexplore-ieee-org.ez46.periodicos.capes.gov.br/stamp/stamp.jsp?tp=&arnumber=75&isnumber=8. Acesso em: 22 Mai. 2019.

RIVERO, L.; BARRETO, R.; CONTE, T.. Characterizing Usability Inspection Methods through the Analysis of a Systematic Mapping Study Extension. **Latin-american Center for Informatics Studies Electronic Journal**, v. 16, n. 1, 2013.

RODRIGUEZ-ECHEVERRIA, Roberto *et al*. A Pattern-Based Development Approach for Interaction Flow Modeling Language,*. In:* SCIENTIFIC PROGRAMMING. 2019.

SIKORA, Martin. **Dart Essentials**. 2. ed. Packt, v. 1, 2015.

SMUTNÝ, P.. Mobile development tools and cross-platform solutions*. In:* PROCEEDINGS OF THE 13TH INTERNATIONAL CARPATHIAN CONTROL CONFERENCE . 2012, High Tatras, 2012. 653-656 p.

TURBAK, Franklyn; WOLBER, David; L MEDLOCK-WALTON, Paul. The Design of Naming Features in App Inventor 2*. In:* IEEE SYMPOSIUM ON VISUAL LANGUAGES AND HUMAN-CENTRIC COMPUTING (VL/HCC). 2014. 2014.

TUTORIALS POINT. **Redux - Data Flow**. Disponível em: https://www.tutorialspoint.com/redux/redux_data_flow.htm. Acesso em: 25 Nov. 2019.

VÁZQUEZ-INGELMO, Andrea; CRUZ-BENITO, Juan; GARCÍA-PEÑALVO, Francisco J.. Proceedings of the 5th International Conference on Technological Ecosystems for Enhancing Multiculturality*. In:* . 2017. 89. ed. 2017. ádiz, Spain p.

WARREN, P.; BOLDYREFF, C.; MUNRO, M.. The evolution of Websites. **Proceedings Seventh International Workshop on Program Comprehension**, Pittsburgh, p. 178-185, 199. Disponível em: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=777757&isnumber=16862. Acesso em: 30 Mai. 2019.

WASSERMAN, A. I. . ‘Software engineering issues for mobile application development,*. In:* PROC. FOSER. 2010, Santa Fe, NM, USA, 2010. 397–400. p.

WEB SITE HOSTING RATING. **INTERNET STATISTICS AND FACTS FOR 2019**. Disponível em: https://www.websitehostingrating.com/internet-statistics-facts/. Acesso em: 29 Mai. 2019.

XANTHOPOULOS, S.; XINOGALOS, S.. ‘A comparative analysis of crossplatform development approaches for mobile applications,*. In:* N PROC. BCI. 2013. 2013. 213-220 p.

ZHENG, Hui; JIACONG, S. Xu. **Workflow Web Application Design**. Worcester , 2016. Dissertação (Computer Science) - Worcester Polytechnic Institute, 2016.

APÊNDICE A — MANUAL DE AMPLIAÇÃO DA FERRAMENTA

O conteúdo deste apêndice destina-se a um guia de como ampliar a ferramenta. Um guia de instalação e execução do projeto em servidor local pode ser encontrado no endereço do projeto nos servidores do Github: https://github.com/RonaldoDev/flutterappgenerator/blob/develop/README.md

Inserção de novos componentes: Para adicionar um novo componente na aplicação é necessário realizar edições de código no frontend e no backend.

Passo 1: é adicionar o componente na lista de componentes no arquivo de configuração no caminho "config/initialStore.js". O código abaixo demonstra os componentes separadas por tipos. O desenvolvedor deve decidir em qual categoria o componente se encaixa e colocar o nome do item no vetor correspondente.

```
const dataDisplays = ['text'];
const inputs = ['textField', 'checkBox', 'select'];
const buttons = ['button', 'íconButton'];
const resources = ['camera', 'map', 'webview'];

export const componentsList = [
    'Data Displays',
    ...dataDisplays,
    'Inputs',
    ...inputs,
    'Buttons',
    ...buttons,
    'Resources',
    ...resources];
```

O nome selecionado para o novo componente deve ser único para evitar conflitos na hora de renderização da aplicação.

Passo 2: Os arquivos que tratam do menu de componentes devem ser alterados para aceitar o novo item. São três estruturas de decisão que devem ser alteradas para funcionamento. Duas delas localizadas no módulo de "component" e outra no módulo "editorGUI".

- Deve ser adicionado um novo arquivo na pasta "component/components" com o nome do componente no modle "novoComponente.js". Este arquivo é

um *React Component* do item desejado. Também é necessário editar o arquivo "component/components/components.js" com as propriedades de componente do React Grid Layout e adicionar o novo item com as propriedades correspondentes. O código abaixo exibe o exemplo de um componente "button" e suas respectivas propriedades que podem ser editador pela aplicação. Cabe ressaltar que as propriedades desenvolvidas para um componente devem estar em consonância com as propriedades possíveis de um componente Flutter.

```
export default function RenderButton(props) {
  const { item, selectItem } = props
  const { widget, id, selected } = item;
  const { color, text, textColor, fontSize, theme } = widget;
  let backgroundColor = color === "primary" ?  null : color
  let componentColor = color === "primary" ?  "primary" : "inherit"
  switch(theme) {
    case "custom":
        return (
          <div key={id} className={selected ? " selected" : ""} onClick={() => selectItem(id)}>
            <Button
              fullWidth
              style={{
                backgroundColor,
                color: textColor,
                fontSize : fontSize,
                height: "100%"
              }}
              className="protect"
              variant="contained">
              {text}
            </Button>
          </div>);
```

- No arquivo "component/componentList.jsx˜, inserir mais um case no *switch statement* com o nome do componente escolhido no Passo 1. Esta adição é responsável pela renderização do item na lista de componentes exibida a esquerda da aplicação.

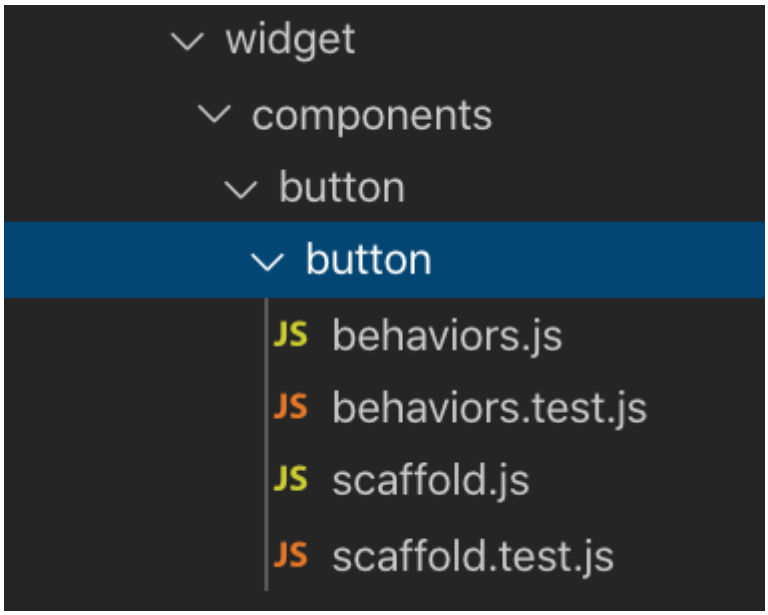
```
switch (item) {
  case "text":
      return (<ListItem style={{ width: "100%" }} key="0" button onClick={() => this.handleAddComponent("text")}>…
  case "button":
    return (<ListItem style={{ width: "100%" }} key="1" button onClick={() => this.handleAddComponent("button")}>…
```

- No módulo "editorGUI/editorGUI.jsx" na função "renderLisItems" o *switch statement* deve ser alterado para contemplar o novo componente a ser renderizado no layout que representa a interface *mobile.* Este caso de escolhe deve chamar a função de componente desenvolvida em "component/components/novoComponente.jsx". O código a seguir representa o *statment* do editorGUI.

```
switch(item.type) {
  case 'button':
    return renderButton({ item, selectItem: this.selectItem });
  case 'iconButton':
    return renderIconButton({ item, selectItem: this.selectItem });
```

Terminando o passo 2, deve ser possível adicionar o novo componente e editar o componente no editor de interface.

Passo 3: No servidor é necessário adicionar uma nova estrutra em "code/widget/components/" com o nome do novo componente respeitando a estrutura de arquivos existentes. A imagem abaixo exibe como é a estrutra de arquivos de um componente no *backend.*



- **"behaviors.js"** é responsável por todos os métodos e ações permitidas para o componentes. É sempre importante verificar se o comportamento esperado existe nas funções do Flutter. O código a seguir demonstra uma ação de navegação para uma próxima tela, comportamento que pode ser atribuído a um botão/

```
function getAction(action) {
    switch(action.type) {
        case 'navigate':
            return `Navigator.push(
                context,
                MaterialPageRoute(
                  builder: (context) => ${action.value}(),
                  ),

              );`
        default:
            return '/*...*/'
    }
}

module.exports = {
    getAction
}
```

- **"scaffold.js" é** o arquivo que contém o template do componente escrito em Flutter, neste arquivo acontece a conversão das propriedades editadas no *frontend* da aplicação. O código abaixo é o scaffold de um botão com suas respectivas propriedades

```
const { getAction } = require('./behaviors');

const getTemplate = (properties) => {
    const { color, text, action } = properties;
    const colorString = color === 'default'? 'Colors.blue' : `Color(0xff${color.replace('#', '')
    const actionPressed = getAction(action);
    return `FlatButton(
            color: ${colorString},
            textColor: Colors.white,
            disabledColor: Colors.grey,
            disabledTextColor: Colors.black,
            splashColor: Colors.blueAccent,
            onPressed: () {
                ${actionPressed}
            },
            child:
                Text(
                    "${text}",
                    style: TextStyle(fontSize: 20.0),
                ),
        )`;
};
                (parameter) properties: any
const button = (properties) => ({
    template: getTemplate(properties),
    minChildren: 0,
    maxChildren: 1,
    children: ['Text', 'Decorator', 'Icon']

})
module.exports = {
    button
}
```

No caminho "code/widget/index.js" deve ser adiciona na estrutura de decisão o novo componente assim como o caminho para as funções de criação do template e seus respectivos comportamentos.

Ao finalizar o passo 3, um novo componente poderá ser adicionado na aplicação e gerado como aplicativo.

APÊNDICE B — ARTIGO

FERRAMENTA DE GERAÇÃO DE APLICATIVOS MÓVEIS HÍBRIDO/CROSSPLATFORM BASEADA EM EDITOR DE INTERFACE

Ronaldo Albert Spranger

(UFSC) Caixa Postal 476 - 88040-900 - Florianópolis - SC - Brazil

Abstract. The present work proposes to analyze tools already used to create applications and present a new tool for developing mobile applications using a drag-and-drop component editor and to link pre-determined code snippets to them. To complete the task, user experience methodologies and consolidated tools for web application development will be used. At the end of this project, a mobile app editor has been built. Through it, the user can insert components, edit properties and assign predefined methods to components. No final user process can generate a multiplatform application and download the code generated by the application.

Resumo. O presente trabalho propõe a analisar ferramentas já utilizadas para criação de aplicativos e apresentar uma nova ferramenta de desenvolvimento de aplicações mobile. Para isso foi utilizado um editor de arrastar e soltar componentes e vincular trechos de código pré-determinados a estes. Para completar a tarefa, foram utilizadas metodologias de experiência de usuário e ferramentas consolidadas para o desenvolvimento de aplicações web. Como resultado, um editor de aplicativos móveis web foi construído. Através dele o usuário pode inserir componentes, editar propriedades e atribuir métodos pré-definidos a componentes. No final do proceso o usuário pode gerar uma aplicação multiplataforma e baixar o código gerado pela aplicação.

1 **INTRODUÇÃO**

Aplicativos móveis (*apps* abreviadamente) estão desempenhando um papel cada vez mais importante em nossas vidas diárias. Com a popularização dos *smartphones,* iniciada em 2008, o número de pessoas que estão conectadas através de dispositivos móveis aumenta vertiginosamente. Segundo a Globalstats (2019) estima-se que em 2020 haverá 6.1 bilhões de usuários conectados por meio de *smartphones* no mundo. Com a popularização da tecnologia e a evolução da internet móvel nos últimos anos, está cada vez mais acessível ter um dispositivo móvel inteligente, o que se tornou uma espécie de assistente para uso tanto pessoal quanto profissional. Ainda segundo o site *Globalstats, e*m maio de 2019, há mais tráfego de rede feito por dispositivos móveis (48,19%) em comparação com *desktops* / *laptops* (47%).

A competição entre os sistemas operacionais é o principal fator que leva os desenvolvedores a adicionar novos recursos para o sistema (Guo, 2008). Esses sistemas operacionais possuem diferenças significativas em suas arquiteturas, o que acaba forçando desenvolvedor a criar uma aplicação para cada sistema. No entanto é difícil desenvolver aplicativos móveis para uma ampla gama de *smartphones* sem um esforço de portabilidade significativo (Niyar, 2012). Segundo Bosika (2012, *apud* Moon Hui 2013) muito tempo, dinheiro, tecnologias e mão de obra são necessários para o desenvolvimento de diversos aplicativos.

Atualmente são diversas as ferramentas que se propõem a criar aplicativos na forma de arrastar e soltar, mas a grande maioria ofertada ainda é paga. Entre estas plataformas está a Seattle Clouds[65] que é uma das mais conhecidas plataformas para criar aplicativos *cross-plataform.*

Em fevereiro de 2018 a *Google* lançou o Flutter. O Flutter é um novo SDK (*Software Development Kit*) para *apps* que ajuda desenvolvedores e designers a criar aplicativos móveis modernos para *iOS* e *Android*. A estrutura moderna e reativa permite criar uma interface do usuário poderosa com animações, base de código compartilhada e visualizações nas plataformas iOS e Android. Facilita o processo de desenvolvimento, custos de desenvolvimento com implantação mínima e rápida.

2 **ESCOPO**

O artefato produzido para este trabalho consiste em uma aplicação web, desenvolvida usando *NodeJS* e JavaScript, que permite edição de *layouts* de

65 https://seattleclouds.com/

interface para aplicativos mobile. Esta ferramenta, através de recurso de arrastar e soltar, permite a criação de interfaces de aplicações e o mapeamento de trechos de código para atribuir a estes componentes usando técnicas de *scaffolding*.

Como saída desta aplicação, uma arquitetura de aplicação *mobile cross-platform* é gerada. Utilizando a *framework Flutter* escrita em código *Dart s*endo fiel a prototipagem realizada na ferramenta web e realizando as tarefas associadas aos *scaffolding.*

A abrangência do presente trabalho é definida em um número de componentes limitados para criação de aplicações simples afim de demonstrar em uma prova de conceito que é possível utilizar estas técnicas para desenvolvimento de aplicações, podendo esta ser estendida a novos trabalhos já que o código é aberto. Também é previsto para este trabalho a elaboração da documentação necessária para o correto uso da ferramenta.

3 **FUNDAMENTAÇÃO TEÓRICA**

3.1 Aplicações Web

As páginas Web em HTML surgiram junto com o conceito da *World Wide Web,* criado por Tim Berners-Lee em 1989, que criou o primeiro website em 1991 e modificou completamente a forma como os usuários utilizam a Internet. Desde então, a Web vem evoluindo constantemente com criações de novas tecnologias para interação e desenvolvimento de páginas dinâmicas. Segundo a organização *Website Hosting Rating,*em janeiro de 2019 a Internet contava com mais de 4 bilhões de usuários em todo o mundo e 342 milhões de domínios registrados, nos quais são hospedados quase 2 bilhões de *websites.* Diariamente são feitos 4 milhões de *posts* em *blogs* e 5 bilhões de pesquisas no Google, sendo que 15% destas pesquisas nunca foram feitas antes.

Segundo Zheng (2016) aplicações Web podem ser subdivididas em três tipos: Baseado no navegador, Baseado no cliente e Aplicativos Web para dispositivos móveis. Do ponto de vista técnico, a Web é um ambiente altamente programável que permite a personalização em massa através do Implantação de uma ampla e diversificada gama de aplicativos para usuários globais. De forma semelhante aos navegadores da Web, que permitem que os usuários recuperem dados e interajam com conteúdo localizado em páginas da Web em um site, aplicações Web são programas de computador que permitem que os visitantes do site enviem e recuperem dados de / para um banco de dados pela Internet usando seu navegador da Web preferido. Os dados são então apresentadas ao usuário em seu navegador,

pois as informações são geradas dinamicamente pela aplicação web através de um servidor web. Os documentos são gerados em um formato padrão para permitir suporte por todos os navegadores.

3.2 Geradores de Código E Scaffolding

A geração de código existe desde os anos 1950, tendo sua origem nos primeiros compiladores (Rich, 1988). Desde então, as organizações de software têm confiado em técnicas de síntese de código para reduzir o tempo de desenvolvimento e aumentar a produtividade (Kelly, 2008). A geração de código pode automatizar a criação de código-fonte através de quadros genéricos, classes, protótipos, modelos, aspectos e geradores de código para melhorar a produtividade dos programadoresAbbas, Jeberson e Klinsega (2012). Isto é um método eficaz para obter reutilização de software na aplicação desenvolvimento e tem muitos casos de sucesso em software, aplicativos da Web e sistemas cliente-servidor distribuídos.

3.3 Mockup

A definição de *mockup* pode ser retirada do dicionário Merrian-Webster como: "um modelo estrutural de tamanho real construído para dimensionar principalmente para estudo, teste ou exibição" ou ainda "uma amostra de trabalho (como de uma revista) para revisar formato, layout ou conteúdo". Em aplicações Web, um *mockup* refere-se a um modelo em execução, navegável, parcial ou de tamanho total da aplicação, usado para elicitação de requisitos, validação e finalização. É possível verificar todos esses recursos carregando inspeções de interação do usuário, em que um especialista, avaliadores, ou mesmo a equipe de desenvolvimento, pode avaliar aspectos de design, navegação e facilidade de uso de uma interface de usuário (RIVERO; BARRETO; CONTE, 2013).

3.4 Aplicativos Móveis

O desenvolvimento para dispositivos móveis geralmente requer diferentes conjuntos de habilidades e familiaridade com diferentes ambientes de tempo de execução do que aqueles mais comumente usados em PCs *desktop*. Escolher a abordagem correta de desenvolvimento para *apps* é sempre uma escolha difícil, pois cada uma delas tem suas vantagens e desvantagens (Smutný, 2012).

Três abordagens são adotadas atualmente para o desenvolvimento de aplicativos móveis: a abordagem nativa, que implica em utilizar o suporte de programação suportado nativamente pelo sistema operacional do dispositivo móvel; a abordagem Web, que consiste em desenvolver aplicações Web que poderão ser executadas em qualquer dispositivo que possua um navegador Web; e a abordagem

híbrida, ou cross-plataforma, que reúne características das duas abordagens anteriores, buscando reunir as vantagens de ambas.

3.5 Desenvolvimento de aplicativos híbridos

Um aplicativo da web híbrido é um aplicativo que não é verdadeiramente um aplicativo da web para dispositivos móveis nem um aplicativo nativo. É basicamente um aplicação escrita com HTML5, JavaScript APIs e CSS, mas executada dentro de um contêiner de aplicativo nativo de terceiros. As características-chave de um aplicativo híbrido são que eles são desenvolvidos com linguagens padrão da Web, mas geralmente têm acesso às APIs e ao hardware do dispositivo nativo.

Segundo Heitkotter (2013) as abordagens de desenvolvimento entre plataformas surgiram para enfrentar esse desafio, permitindo que os desenvolvedores implementassem seus aplicativos em apenas uma etapa para uma variedade de plataformas, evitando repetição e aumentando a produtividade. Por um lado, essas abordagens precisam oferecer generalidade adequada para permitir o fornecimento de aplicativos para várias plataformas. Por outro lado, eles ainda têm que permitir que os desenvolvedores aproveitem as vantagens específicas e possibilidades de *smartphones*.

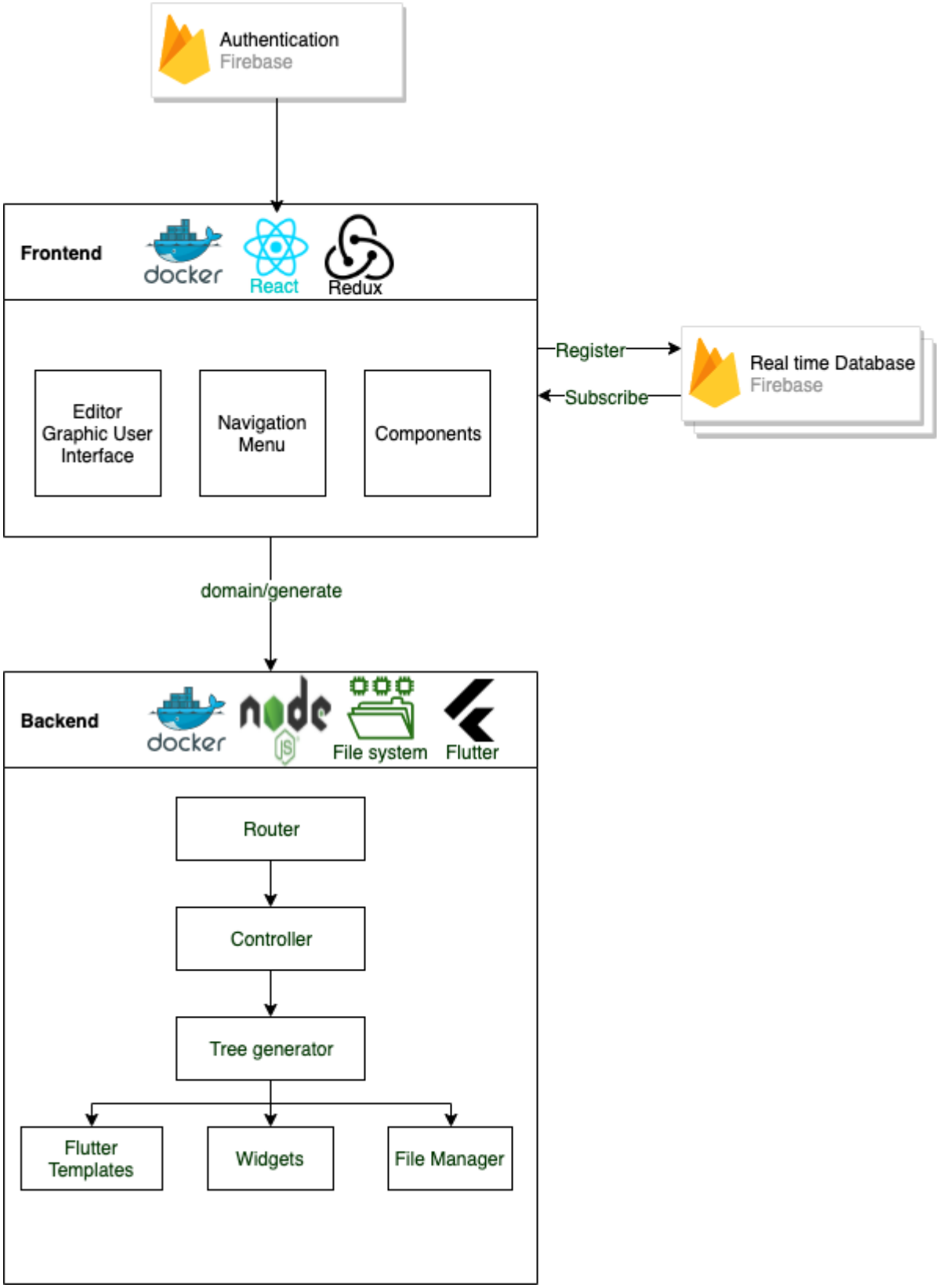
4 DESENVOLVIMENTO

A ferramenta produto deste trabalho se propõe a criar código-fonte de aplicação mobile híbrida através de um editor web de *drag and drop* no qual componentes são arrastados para uma interface gráfica com a representação da tela de um smartphone. O usuário pode criar diversas interfaces e escolher um modelo de navegação entre elas. Clicando nos componentes adicionados no modelo, o usuário tem a opção selecionar ações deste componente. Por exemplo, ao colocar um campo de texto ele pode nomear o campo para adicioná-lo a uma tabela no banco de dados. Quando selecionar um botão e vincular uma ação, como gravar no banco de dados.

Ao término, o usuário pode gerar o código resultante da sua edição. A aplicação Web cria um projeto com toda a arquitetura padrão do *framework Flutter, ou seja, com toda* estrutura de pastas e arquivos, assim como os arquivos gerados pelo editor de layout. O usuário, com o código gerado, usa o editor da sua preferência para compilar e rodar o código em sua máquina.

4.1 Arquitetura

O Diagrama 1 mostra a arquitetura básica da aplicação, que está dividida em duas camadas principais e duas conexões a APIs de terceiros. No frontend da aplicação ficam os módulos do Menu, Editor e componentes que são renderizados ao mesmo tempo.



4.2 PROVA DE CONCEITO

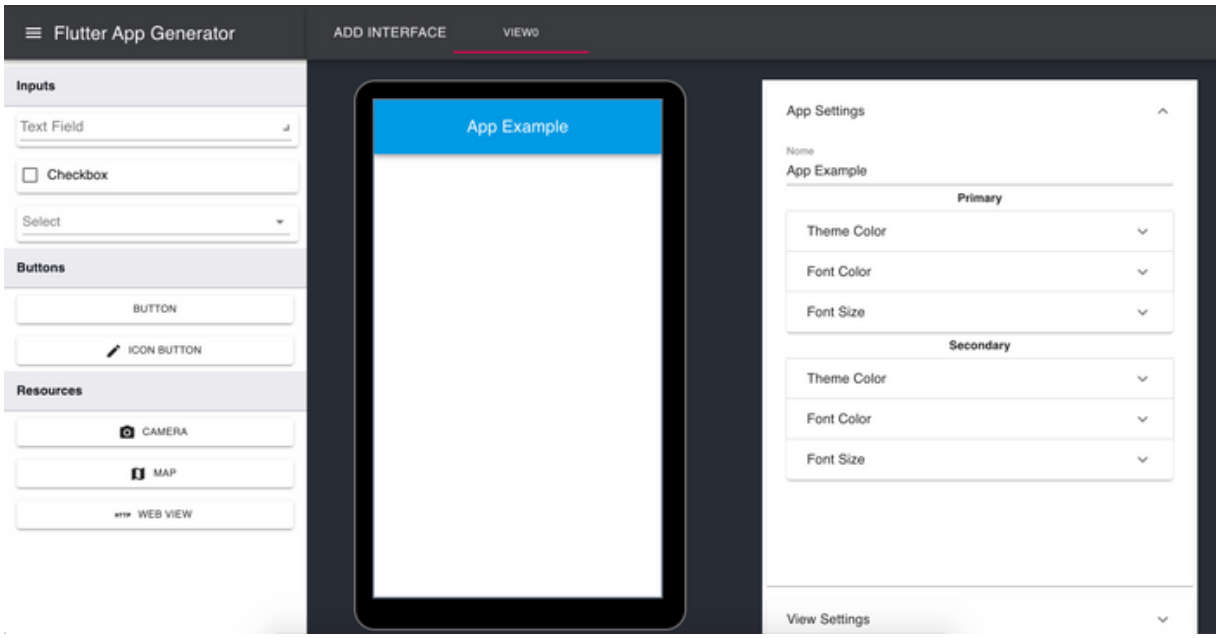
Para testar a viabilidade do projeto foi desenvolvida uma prova de conceito da aplicação dividida em três partes: *frontend*, geração de código e criação do aplicativo móvel. Nenhuma folha de estilo foi aplicada nesta etapa do processo, que visa somente validar o funcionamento básico da aplicação.

4.3 VERSÃO FINAL DA FERRAMENTA

Neste capítulo é abordada a fase final de construção da aplicação, assim como as imagens finais do desenvolvimento e da estrutura do projeto. Foi usado como base na aplicação final o código escrito na prova de conceito da aplicação, utilizando da arquitetura descrita anteriormente.

O escopo definido para a aplicação final foi de aumentar a personalização de estilo da aplicação e as propriedades de configurações de componentes realizadas na prova de conceito. Devido a uma limitação de tempo para desenvolvimento da ferramenta, optou-se por uma abordagem de viabilizar uma aplicação básica, com navegação entre interfaces, inserção de componentes e utilização de recursos que são nativos das plataformas *Android* e *iOS.* Utilizando esta abordagem, o trabalho abrange todo o ciclo básico de desenvolvimento de uma aplicação *cross-plataform*. A figura 21 demonstra a tela inicial da aplicação.

Figura 19 - Versão final da aplicação



Fonte: O autor (2019)

Foram realizadas diversas melhorias de interface na aplicação tendo como objetivo melhoria na usabilidade. Uma delas foi categorizar os componentes por tipo e utilizar imagem do componente a ser adicionado junto com seu nome para facilitar o entendimento do usuário, exibindo uma pré-visualização do componente que está para ser selecionado. A figura 22 demonstra o resultado final da lista de componentes categorizada e renderizada com suas respectivas imagens.

5 **CONCLUSÃO**

O desenvolvimento de aplicativos para dispositivos móveis é um campo em evolução que atrai grande interesse. O tipo mais comum de aplicativos móveis era, até recentemente, aplicativos móveis nativos. Os aplicativos móveis nativos são desenvolvidos para uma plataforma específica e oferecem, sem dúvida, a melhor experiência do usuário. Nos aplicativos móveis nativos, o código-fonte é eficiente, com desempenho rápido, aparência e comportamento consistentes e acesso total aos dados e hardware da plataforma subjacente. O desenvolvimento de um aplicativo móvel nativo para várias plataformas requer uma implementação distinta para cada plataforma, um fenômeno conhecido como fragmentação.

A proposta do presente trabalho em criar uma ferramenta de geração de aplicativos *cross-platform* para resolver um problema bem conhecido quanto ao desenvolvimento para diversas plataformas foi concluída com êxito. O objetivo de criar um gerador de aplicativos móveis multiplataforma foi alcançado. Mesmo não contendo uma gama de componentes e configurações muito grande, todos os principais recursos de uma aplicação móvel está contida no editor. É possível através da ferramenta gerar aplicativo que roda em iOs e Android, capaz de acessar recursos nativos em ambos sistemas operacionais, realizar operações como navegação e validação de campos de texto, e construir um layout de aplicativo móvel que seja fiel a interface criada no editor online.

A ferramenta web foi projetada e construída com intuito de ser facilmente escalável. Idealizada em módulos e componentes, sua arquitetura comporta mudanças e atualizações de maneira simples, sem necessidade de realizar grandes mudanças no código existente. Utilizando recursos de ambientes web modernos, como banco de dados em nuvem, contêineres para garantir execução em qualquer ambiente e integração contínua, a aplicação web cumpre seu objetivo de manter um ambiente de criação de aplicativos utilizando o editor de interface.

Por fim, este trabalho proporcionou uma grande base de conhecimento em desenvolvimento de aplicações web e móveis cross-plataforma, geração de código

através de *scaffolds,* soluções de geração de aplicações e tecnologia no geral e diversas tecnologias que compuseram todo o ambiente de desenvolvimento necessário para produzir este trabalho.

Referências

ARSHAD, A. et al. An Empirical Study of Investigating Mobile Applications Development Challenges. SPECIAL SECTION ON SOFTWARE STANDARDS AND THEIR IMPACT IN REDUCING SOFTWARE FAILURES, Beijing , China, 2018.

AXELSSON , Oscar; CARLSTRÖM, Fredrik . Evaluation Targeting React Native in Comparison to Native Mobile Development. Lund, Sweden, 2016. Dissertação (INTERACTION DESIGN) - LUND UNIVERSITY, 2016.

BANERJEE, Ishan; NGUYEN, Bao. Graphical user interface (GUI) testing: Systematic mapping and repository. In: INFORMATION AND SOFTWARE TECHNOLOGY. 55. ed. 2013. 1679-1694 p.

CORRAL, Luis; JANES, Andrea; REMENCIUS, Tadas. Potential advantages and disadvantages of multiplatform development frameworks: A vision on mobile environments . Procedia Computer Science, Bolzano, v. 10, 2012.

FLANAGAN, David. JavaScript: the definitive guide. 3. ed. O'Reilly Media., 2006.

FONG, Elizabeth; OKUN, Vadim. Web Application Scanners: Definitions and Functions. In: 40TH ANNUAL HAWAII INTERNATIONAL CONFERENCE ON SYSTEM SCIENCES (HICSS'07). 2007, Waikoloa, HI, USA, 2007.

GARCIA-PEÑALVO, F. J.; GARCIA-HOLGADO, A.. Open Source Solutions for Knowledge Management and Technological Ecosystems. In: IGI GLOBAL. 2016.

Global Stats. Desktop vs Mobile vs Tablet Market Share Worldwide. 2019. Disponível em: http://gs.statcounter.com/platform-market-share/desktop-mobile-tablet. Acesso em: 25 Nov. 2019.

HEITKÖTTER , H.; HANSCHKE , S.; MAJCHRZAK , T.A.. Evaluating Cross-Platform Development Approaches for Mobile Applications. In: CORDEIRO . 2013, Berlin: Springer.

HOLZER, A.; ONDRUS, J.. 'Mobile app development: Native or Web?. In: PROC. WORKSHOP EBUS. (WEB). 2012. 2012.

JOBE, Willian. Native Apps vs. Mobile Web Apps . International Journal of Interactive Mobile Technologies, Stockholm, 2013.

JOORABCHI, M.E.; MESBAH , A.; KRUCHTEN, P.. Real Challenges in Mobile App Development. In: ACM / IEEE INTERNATIONAL SYMPOSIUM ON EMPIRICAL SOFTWARE ENGINEERING AND MEASUREMENT. 2013, Baltimore, 2013. 15-24 p.

KELLY, Steven; TOLVANEN, Juha-Pekka. Domain-Specific Modeling: Enabling Full Code Generation. 1. ed. New Jersey: John Wiley & Sons, 2008.

LACHGAR, Mohamed; ABDALI, Abdelmounaïm. Decision Framework for Mobile Development Methods. A) International Journal of Advanced Computer Science and Applications, Marrakech, v. 8, 2017.

MAINKAR, Prajyot . In: MAINKAR, Prajyot ; GIORDANO, Salvatore . Google Flutter Mobile Development : Quick Start Guide. 1. ed. Packt, 2019.

MAINKAR, Prajyot ; GIORDANO, Salvatore . Google Flutter Mobile Development : Quick Start Guide. 1. ed. Packt, 2019.

N BISARAHALLI, Manoj. Flutter hands-on: Building a Live Location Sharing App. 2018. Disponível em: https://blog.geekyants.com/flutter-hands-on-building-a-live-location-sharing-app-14b67ef17404. Acesso em: 27 Set. 2019.

NIYAR, Noor Ali ; HUSSAIN, Mansoor. Cross Platform Mobile Application Development Framework. Karachi, Pakistan. Monografia () - Fast-NUCES (Karachi Campus).

Node.js Foundation. Keeping the Node.js core small. Medium. 2017. Disponível em: https://medium.com/the-node-js-collection/keeping-the-node-js-core-small-137f83d18152. Acesso em: 20 Jun. 2019.

O GOULART MAGNO, Danillo. APLICAÇÃO DA TÉCNICA DE SCAFFOLDING PARA A CRIAÇÃO DE SISTEMAS CRUD. 2015. Dissertação (Sistemas de Computação) - Unifei, 2015.

RICH, C.; WATERS, R.C. . Automatic programming: myths and prospects. Computer, v. 21, p. 40-51, Agosto 1988. Disponível em: http://ieeexplore-ieee-org.ez46.periodicos.capes.gov.br/stamp/stamp.jsp?tp=&arnumber=75&isnumber=8. Acesso em: 22 Mai. 2019.

RIVERO, L.; BARRETO, R.; CONTE, T.. Characterizing Usability Inspection Methods through the Analysis of a Systematic Mapping Study Extension. Latin-american Center for Informatics Studies Electronic Journal, v. 16, n. 1, 2013.

RODRIGUEZ-ECHEVERRIA, Roberto et al. A Pattern-Based Development Approach for Interaction Flow Modeling Language,. In: SCIENTIFIC PROGRAMMING. 2019.

SMUTNÝ, P.. Mobile development tools and cross-platform solutions. In: PROCEEDINGS OF THE 13TH INTERNATIONAL CARPATHIAN CONTROL CONFERENCE . 2012, High Tatras, 2012. 653-656 p.

VÁZQUEZ-INGELMO, Andrea; CRUZ-BENITO, Juan; GARCÍA-PEÑALVO, Francisco J.. Proceedings of the 5th International Conference on Technological

Ecosystems for Enhancing Multiculturality. In: . 2017. 89. ed. 2017. ádiz, Spain p.tp=&arnumber=777757&isnumber=16862. Acesso em: 30 Mai. 2019.

WASSERMAN, A. I. . 'Software engineering issues for mobile application development,. In: PROC. FOSER. 2010, Santa Fe, NM, USA, 2010. 397–400. p.

Web Site Hosting Rating. INTERNET STATISTICS AND FACTS FOR 2019. Disponível em: https://www.websitehostingrating.com/internet-statistics-facts/. Acesso em: 29 Mai. 2019.

XANTHOPOULOS, S.; XINOGALOS, S.. 'A comparative analysis of crossplatform development approaches for mobile applications,. In: N PROC. BCI. 2013. 2013. 213-220 p.

ZHENG, Hui; JIACONG, S. Xu. Workflow Web Application Design. Worcester , 2016. Dissertação (Computer Science) - Worcester Polytechnic Institute, 2016.

APÊNDICE C — CÓDIGO

O código completo gerado pode ser encontrado em https://github.com/RonaldoDev/flutterappgenerator. Seguem as funções principais:

#reducer.js

```
import {
  ADD_COMPONENT,
  UPDATE_COMPONENTS,
  SELECT_COMPONENT,
  EDIT_COMPONENT,
  SELECT_TAB,
  ADD_VIEW,
  SAVE_VIEWS,
  FETCH_VIEWS,
  CHANGE_THEME
} from "./actionTypes";

import { componentsList, firstView, theme } from '../config/initialStore';

const initialState = {
  allIds: [],
  byIds: {},
  compId: 0,
  components: componentsList,
  componentsRender: [],
  views: [firstView],
  currentTab: firstView,
  theme: theme
};

export default function(state = initialState, action) {
  switch (action.type) {
    case ADD_COMPONENT: {
      const { id, content } = action.payload;
        const tab = { ...state.currentTab, componentId : content.id, components:
[...state.componentsRender, content]};
      const newViews = state.views.reduce((total, view) => {
        if (view.id === tab.id) {
```

```
      total.push({...view, components : tab.components});
      return total;
    }
    total.push(view);
    return total;
  }, []);
  return {
    ...state,
    allIds: [...state.allIds, id],
    byIds: {
      ...state.byIds,
      [id]: {
        content,
        completed: false
      }
    },
    compId: ++state.compId,
    componentsRender: [...state.componentsRender, content],
    currentTab: tab,
    views: newViews
  };
};
case UPDATE_COMPONENTS: {
  const { content } = action.payload;
  const tab = { ...state.currentTab, components: [...content]};
  const newViews = state.views.reduce((total, view) => {
    if (view.id === tab.id) {
      total.push({...view, components : tab.components});
      return total;
    }
    total.push(view);
    return total;
  }, []);
  return {
    ...state,
    componentsRender: [...content],
    currentTab: tab,
    views: newViews
```

```
  };
}
case SELECT_COMPONENT: {
  const { content } = action.payload;
  return {
    ...state,
    currentTab: {...state.currentTab, selectedComponent: content}
  };
}
case EDIT_COMPONENT: {
  const { content } = action.payload;
  return {
    ...state,
    currentTab: {...state.currentTab, selectedComponent: content}
  };
}
case SELECT_TAB: {
  const { content } = action.payload;
  return {
    ...state,
    currentTab: content
  };
}
case ADD_VIEW: {
  const { content } = action.payload;
  return {
    ...state,
    views: [...state.views, content]
  };
}
case SAVE_VIEWS: {
  const { content } = action.payload;
  return {
    ...state,
    views: [...content]
  };
}
case FETCH_VIEWS: {
  const content = action.payload;
```

```
        return {
          ...state,
          compId: content.compId,
          views: content.views,
          theme: { ...content.theme },
          currentTab: { ...content.views[0]}
        };
      }
      case CHANGE_THEME: {
        const { content } = action.payload;
        return {
          ...state,
          theme: {
            ...state.theme,
            ...content
          }
        }
      }
      default:
        return state;
    }

  }
```

 # components.js

```
  export default (id, item, position) => {
      const id_axis = id++;
      switch(item) {
          case 'checkBox':
              return { i: `${id_axis}`, x: 0, y: position, w: 4, h: 1, isResizable: false}
          case 'textField':
                return { i: `${id_axis}`, x: 0, y: position, w: 4, h: 1, minW: 2, maxW: 4,
maxH: 1, minH: 1}
          case 'text':
                return { i: `${id_axis}`, x: 0, y: position, w: 4, h: 1, minW: 2, maxW: 4,
maxH: 1, minH: 1}
```

```
            case 'button':
                return { i: `${id_axis}`, x: 0, y: position, w: 2, h: 1, minW: 2, maxW: 4,
maxH: 1}
            case 'iconButton':
                return { i: `${id_axis}`, x: 0, y: position, w: 2, h: 1, minW: 1, maxW: 4,
maxH: 1}
            case 'appbar':
                    return { i: `${id_axis}`, x: 0, y: position, w: 4, h: 1, static: true,
isDraggable: false, isResizable: false }
            case 'select':
               return { i: `${id_axis}`, x: 0, y: position, w: 4, h: 1, isResizable: false }
            default:
               return { i: `${id_axis}`, x: 0, y: position, w: 4, h: 10, static: true };
         }
      }

      export const widget = (id, componentType) => ({
         color: "primary",
         text: "default",
         theme: "primary",
         textColor: "black",
         fontSize: 12,
         cssClass: "",
         icon: "add",
         id: id,
         hasAction: componentType === "button",
         action: { type: "", value: "" },
         type: componentType,
         items: [
            { key: 1, label: "one" },
            { key: 2, label: "two" },
            { key: 3, label: "three" }
         ],
         keyboardType: "Alphabetical",
         website: ""
      });
```

 # componentList.container.jsx

```
      import { arrayOf, func, number, string } from 'prop-types';
```

```
import React, { Component } from 'react';
import { connect } from 'react-redux';
import { bindActionCreators } from 'redux';
import { getComponentList, getTheme } from '../../reducers/selectors';
import { getPositionsToRenders } from '../editorGUI/editorGUI.selectors';
import { getLastId } from '../nav/nav.selectors';
import ComponentList from './componentList';
import { addComponent } from './components.actions';
import getComponent, { widget } from './components/components';
import { Box, Snackbar, IconButton } from '@material-ui/core';
import CloseIcon from '@material-ui/icons/Close';

class ComponentListContainer extends Component {
 constructor() {
  super();
  this.state = {
   layoutFull: false,
   errorMessage: ""
  }
  this.handleClose = this.handleClose.bind(this);
  this.handleAddComponent = this.handleAddComponent.bind(this);
 }
 handleAddComponent = (componentType) => {
  const { lastId, positionsToRender } = this.props;
  const resources = ["map", "camera", "webview"];
   if (resources.includes(componentType) && positionsToRender.length !==
10) {
   this.setState({ errorMessage: `To insert ${componentType} component, the
layout must be clean` });
   this.setState({ layoutFull: true });
  } else if (!positionsToRender.length) {
   this.setState({ errorMessage: "Layout is full" });
   this.setState({ layoutFull: true });
  } else {
            const component = getComponent(lastId, componentType,
positionsToRender[0]);
   const comp = {
    layoutItem: component,
```

```
      id: component.i,
      type: componentType,
      selected: false,
      widget: widget(component.i, componentType)
    };
    this.props.addComponent(comp);
  }
};
handleClose() {
  this.setState({ layoutFull: false });
}
render() {
  const { componentList } = this.props;
  return (
    <Box>
      <Snackbar
        anchorOrigin={{
          vertical: 'top',
          horizontal: 'center',
        }}
        open={this.state.layoutFull}
        autoHideDuration={2000}
        onClose={this.handleClose}
        ContentProps={{
          'aria-describedby': 'message-id',
        }}
        message={<span id="message-id">{this.state.errorMessage}</span>}
        action={[
          <IconButton
            key="close"
            aria-label="close"
            color="inherit"
            // className={classes.close}
            onClick={this.handleClose}
          >
            <CloseIcon />
          </IconButton>,
        ]}
      />
```

```
                    <ComponentList components={componentList} addComponent=
{this.handleAddComponent} />
            </Box>)
          }
        }

        ComponentListContainer.propTypes = {
          componentList: arrayOf(string),
          addComponent: func,
          lastId: number
        }

        const mapStateToPros = state => {
          const positionsToRender = getPositionsToRenders(state);
          const componentList = getComponentList(state);
          const compId = getLastId(state);
          const theme = getTheme(state);

          return { positionsToRender, componentList, lastId: compId, theme };
        }

        const mapDispatchToProps = dispatch =>
          bindActionCreators({ addComponent }, dispatch);

        export default connect(mapStateToPros, mapDispatchToProps)
(ComponentListContainer);
```

 # componentList.jsx

```
        import React, { Component } from 'react';
        import { func, arrayOf, string } from 'prop-types';
        import {
          Box,
          Button,
          Divider,
          List,
          ListItem,
          ListItemText,
```

```
    Checkbox,
    Select,
    TextField,
    Typography,
    FormControlLabel,
    FormControl,
    InputLabel

  } from '@material-ui/core'
  import {
    CameraAlt as CameraArtIcon,
    Edit as EditIcon ,
    Map as MapIcon,
    Http as HttpIcon,
  } from '@material-ui/icons';
  import i18n from 'i18next';


  class ComponentList extends Component {
    constructor() {
      super();
      this.handleAddComponent = this.handleAddComponent.bind(this)
    }
    handleAddComponent(componentName) {
      const { addComponent } = this.props
      addComponent(componentName);
    }

    renderListItems(item, index) {
      switch (item) {
        case "text":
            return (<ListItem style={{ width: "100%" }} key="0" button onClick={() =>
this.handleAddComponent(item)}>
                    <Button
                     variant="contained"
                     color="default"
                       style={{ backgroundColor: "#FFFFFF", width: "100%", fontSize:
12}}
                    >
```

```
                    {i18n.t("text")}
                  </Button>
                </ListItem>);
            case "button":
              return (<ListItem style={{ width: "100%" }} key="1" button onClick={() =>
this.handleAddComponent(item)}>
                  <Button
                variant="contained"
                color="default"
                style={{ backgroundColor: "#FFFFFF", width: "100%", fontSize: 12}}
              >
                {i18n.t("button")}
              </Button>
            </ListItem>)

            case "textField":
              return (
                  <ListItem style={{ width: "100%" }} key="2" button onClick={() =>
this.handleAddComponent(item)}>
                <TextField style={{ pointerEvents: "none", boxShadow: "0px 3px 1px
-2px rgba(0,0,0,0.2), 0px 2px 2px 0px rgba(0,0,0,0.14), 0px 1px 5px 0px
rgba(0,0,0,0.12)" }} fullWidth label={i18n.t("input-text")}/>
              </ListItem>);
            case "checkBox":
              return (
                  <ListItem style={{ width: "100%" }} key="3" button onClick={() =>
this.handleAddComponent(item)}>
                <FormControlLabel
                  style={{ color: "grey", width: "100%", marginLeft: "4px", boxShadow:
"0px 3px 1px -2px rgba(0,0,0,0.2), 0px 2px 2px 0px rgba(0,0,0,0.14), 0px 1px 5px
0px rgba(0,0,0,0.12)"}}
                  control={
                    <Checkbox fullWidth style={{ pointerEvents: "none" }} />
                  }
                  label={i18n.t("checkbox")}
                />
              </ListItem>);
            case "select":
```

```
            return (
                    <ListItem style={{ width: "100"}} key="4" button onClick={() =>
this.handleAddComponent(item)}>
                  <FormControl style={{ width: "100%", boxShadow: "0px 3px 1px -2px
rgba(0,0,0,0.2), 0px 2px 2px 0px rgba(0,0,0,0.14), 0px 1px 5px 0px rgba(0,0,0,0.12)"
}} >
                          <InputLabel  htmlFor="select-01">{i18n.t("select-input")}
</InputLabel>
                    <Select
                      style={{width:'100%', pointerEvents: "none"}}
                      inputProps={{
                      id: "select-01",
                      }}
                    >


                    </Select>
                  </FormControl>
                </ListItem>);

            case "iconButton":
              return (
                  <ListItem className="icon-button" style={{ width: "100%" }} key="5"
button onClick={() => this.handleAddComponent(item)}>

                <Button
                  variant="contained"
                  color="default"
                  style={{ backgroundColor: "#FFFFFF", width: "100%", fontSize: 12}}
                  startIcon={<EditIcon />}
                >
                  {i18n.t("icon-button")}
                </Button>

                </ListItem>);
            case "camera":
              return (
                  <ListItem className="icon-button" style={{ width: "100%" }} key="6"
button onClick={() => this.handleAddComponent(item)}>
```

```
                    <Button
                      variant="contained"
                      color="default"
                      style={{ backgroundColor: "#FFFFFF", width: "100%", fontSize: 12}}
                      startIcon={<CameraArtIcon />}
                    >
                      {i18n.t("camera")}
                    </Button>

                  </ListItem>);
              case "map":
                return (
                   <ListItem className="icon-button" style={{ width: "100%" }} key="7"
button onClick={() => this.handleAddComponent(item)}>

                    <Button
                      variant="contained"
                      color="default"
                      style={{ backgroundColor: "#FFFFFF", width: "100%", fontSize: 12 }}
                      startIcon={<MapIcon />}
                    >
                      {i18n.t("map")}
                    </Button>

                  </ListItem>);
            case "webview":
                return (
                   <ListItem className="icon-button" style={{ width: "100%" }} key="8"
button onClick={() => this.handleAddComponent(item)}>

                    <Button
                      variant="contained"
                      color="default"
                      style={{ backgroundColor: "#FFFFFF", width: "100%", fontSize: 12}}
                      startIcon={<HttpIcon />}
                    >
                      {i18n.t("web-view")}
```

```
                </ListItem>);

            default:
              return (
                <div style={{ backgroundColor: "#ebebf0" }}>
                <ListItem key={`${item}${index}`}>
                  <ListItemText primary={
                        <Typography type="body2" style={{ color: '#090C12', fontWeight:
"bold", fontSize: 15 }}>
                   {i18n.t(item)}
                  </Typography>} ></ListItemText>

                </ListItem>
                  <Divider style={{ height: 2, color:"#828282" }} />

                </div>

              );
            }

          }
          render() {
            const { components } = this.props;
            return (

              <Box style={{ backgroundColor: "#FFFFFF", width: "100%" }}>
                <List style={{ fontSize: "1rem" }} component="nav">
                  {components.map((item, index) => this.renderListItems(item, index))}

                </List>
              </Box>

            );
          }
        }

        ComponentList.propTypes = {
```

```
  components: arrayOf(string),
  addComponent: func
}
export default ComponentList;
```

 # EditorGui.container.jsx

```
import React, { Component } from 'react';
import { Grid } from '@material-ui/core';
import { connect } from 'react-redux';
import { bindActionCreators } from 'redux';
import { updateComponent, selectComponent } from './editorGUI.actions';
import EditorGUI from './editorGUI';
import PropertyMenuContainer from './propertyMenu/propertyMenu.container';
import { getComponentsToRender } from './editorGUI.selectors';
import { arrayOf, func, object } from 'prop-types';
import { getTheme } from '../../reducers/selectors';
import { ThemeProvider } from '@material-ui/core/styles';
import { createMuiTheme } from '@material-ui/core/styles';
import { getCurrentTab } from '../nav/nav.selectors';

class EditorGUIContainer extends Component {
  constructor() {
    super();
    this.handleUpdateComponents = this.handleUpdateComponents.bind(this);
    this.handleSelectComponent = this.handleSelectComponent.bind(this);
  }
  handleUpdateComponents = (components) => {
    const { updateComponent } = this.props;
    updateComponent(components);
  }

  handleSelectComponent = (component) => {
    const { selectComponent } = this.props;
    selectComponent(component);
  }

  render() {
    const { componentList, currentTab, theme } = this.props;
```

```
    const muiTheme = createMuiTheme({
      typography: theme.typography,
      palette: theme.palette
    });
    return (
      <Grid container spacing={0}>
        <Grid item md={6}>
          <ThemeProvider theme={muiTheme}>
            <EditorGUI
              components={componentList}
              updateComponent={this.handleUpdateComponents}
              selectComponent={this.handleSelectComponent}
              name={currentTab.id === 0 ? currentTab.appName : currentTab.title}
            />
          </ThemeProvider>
        </Grid>
        <Grid item xs={6}>
          <PropertyMenuContainer />
        </Grid>
      </Grid>);
  }
}

EditorGUIContainer.propTypes = {
  componentList: arrayOf(object),
  updateComponent: func.isRequired,
  selectComponent: func.isRequired
}

const mapStateToPros = state => {
  const componentList = getComponentsToRender(state);
  const theme = getTheme(state);
  const currentTab = getCurrentTab(state);
  return { componentList, theme, currentTab };
}

const mapDispatchToProps = dispatch =>
  bindActionCreators({ updateComponent, selectComponent }, dispatch);
```

export default connect(mapStateToPros, mapDispatchToProps)(EditorGUIContainer);

 # EditorGui.jsx

```
import React, { Component } from 'react';
import { AppBar, Toolbar, Typography, Container } from '@material-ui/core';
import ReactGridLayout from 'react-grid-layout';
import "react-grid-layout/css/styles.css";
import './style.css';
import { arrayOf, func, number, object, string } from 'prop-types';
import renderButton from '../component/components/button';
import renderCheckbox from '../component/components/checkbox';
import renderTextField from '../component/components/inputText';
import renderText from '../component/components/text';
import renderSelect from '../component/components/select';
import renderIconButton from '../component/components/iconButton';
import renderResource from '../component/components/resource';


class EditorGUI extends Component {
  constructor() {
    super();
    this.state = {
      selected: false,
      isLoading: false,
      todos: {}
    }
    this.onLayoutChange = this.onLayoutChange.bind(this);
  }
  onLayoutChange = (layout) => {
    const { components, updateComponent } = this.props
    const componentPositions = components.reduce((layouts, component) => {
      layout.forEach(item => {
        if (component.id === item.i) {
          layouts.push({ ...component, layoutItem: item });
        }
```

```
    })
    return layouts;
  }, []);
  updateComponent(componentPositions)
};

selectItem = (id) => {
  const { components, selectComponent } = this.props;

  components.forEach(comp => {
    if (comp.id === id) {
      comp.selected = true;
      selectComponent(comp.widget);
    }
    else
      comp.selected = false;
  });
  this.setState({ isSelected: true })
}
renderItems(item) {

  switch (item.type) {
    case 'button':
      return renderButton({ item, selectItem: this.selectItem });
    case 'iconButton':
      return renderIconButton({ item, selectItem: this.selectItem });
    case 'textField':
      return renderTextField({ item, selectItem: this.selectItem });
    case 'checkBox':
      return renderCheckbox({ item, selectItem: this.selectItem });
    case 'text':
      return renderText({ item, selectItem: this.selectItem });
    case 'select':
      return renderSelect({ item, selectItem: this.selectItem });
    default:
      return renderResource({ item, selectItem: this.selectItem });
  }
};
```

```
  render() {

    const { components, name } = this.props;
    const layout = components.map(item => item.layoutItem);
    return (
      <Container maxWidth="lg">
        <div className="phone">
          <div className="screen">

            <AppBar color="primary" position="static">
              <Toolbar>
                <Typography variant="h6" style={{ flexGrow: 1 }}>
                  {name}
                </Typography>

              </Toolbar>
            </AppBar>
            <ReactGridLayout cols={4}
              onLayoutChange={this.onLayoutChange}
              rowHeight={45}
              width={340}
              static={false}
              isDraggable={true}
              useCSSTransforms
              compactType={null}
              preventCollision={true}
              isResizable={true}
              className="layout" maxRows={9} layout={layout} >
              {components.map(item => this.renderItems(item))}
            </ReactGridLayout>
          </div>
        </div>
      </Container>
    )
  }
}

EditorGUI.defaultProps = {
  components: [],
```

```
  className: "layout",
  items: 20,
  rowHeight: 30,
  onLayoutChange: func,
  cols: 4
}
EditorGUI.propTypes = {
  components: arrayOf(object),
  updateComponent: func.isRequired,
  selectComponent: func.isRequired,
  className: string,
  items: number,
  rowHeight: number,
  onLayoutChange: func,
  cols: number
}

export default EditorGUI;
```

 # PropertyMenu.container.jsx

```
import React, { Component } from 'react';
import { connect } from 'react-redux';
import { bindActionCreators } from 'redux';
import PropertyMenu from './propertyMenu';
import { getComponentsToRender, getSelectedComponent } from
'../editorGUI.selectors';
import { getCurrentTab } from '../../nav/nav.selectors';
import { getViews } from '../../../selectors/view';
import { updateComponent, selectComponent } from '../editorGUI.actions';
import { func, object } from 'prop-types';
import { Box } from '@material-ui/core'
import Settings from './settings.container';
import Snackbar from '@material-ui/core/Snackbar';
import IconButton from '@material-ui/core/IconButton';
import CloseIcon from '@material-ui/icons/Close';

class PropertyMenuContainer extends Component {
```

```
        constructor() {
          super();
          this.handleClose = this.handleClose.bind(this);
          this.state = {
            theme: "primary",
            themeChanged: false
          }
        }
        handleUpdateComponents = (value) => {

          const { components, updateComponent, component } = this.props
          if (Object.keys(value)[0] === "theme") {
            this.setState({ theme: Object.values(value)[0] });
            this.setState({ themeChanged: true });
          }
              const componentPositions = components.reduce((edited_components,
comp) => {
            if(comp.id === component.id) {
              comp.widget[Object.keys(value)[0]] = Object.values(value)[0];
            }
            edited_components.push(comp);
            return edited_components;
          }, []);
          updateComponent(componentPositions)
        }
        deleteItem = () => {
           const { components, updateComponent, component, selectComponent } =
this.props
          const componentPositions = components.reduce((componentList, comp) =>
{

            if (component.id !== comp.id) {
              componentList.push(comp);
            }

            return componentList;
          }, []);
          updateComponent(componentPositions);
```

```
    selectComponent({});
  }
  handleClose() {
    this.setState({ themeChanged: false });
  }
  render() {
    const { component, views, currentTab } = this.props;
    const viewsFiltered = views.filter(view => view.id !== currentTab.id)
    if (component.id)
      return (
        <Box className="prop-menu">
          <Snackbar
            anchorOrigin={{
              vertical: 'center',
              horizontal: 'center',
            }}
            open={this.state.themeChanged}
            autoHideDuration={2000}
            onClose={this.handleClose}
            ContentProps={{
              'aria-describedby': 'message-id',
            }}
              message={<span id="message-id">Component theme changed to
{this.state.theme}</span>}
            action={[
              <IconButton
                key="close"
                aria-label="close"
                color="inherit"
                // className={classes.close}
                onClick={this.handleClose}
              >
            <CloseIcon />
          </IconButton>,
        ]}
      />

          <PropertyMenu
```

```
            handleUpdateComponents={this.handleUpdateComponents}
            deleteItem={this.deleteItem}
            component={component}
            views={viewsFiltered.map(v => v.title)}
          />
        </Box>);
    return (
      <Box className="prop-menu">
        <Settings
          view={views.filter(v => v.id === currentTab.id)[0]}
        />
      </Box>);
  }
}

PropertyMenuContainer.propTypes = {
  component: object,
  updateComponent: func.isRequired
}

const mapStateToPros = state => {
  const component = getSelectedComponent(state);
  const components = getComponentsToRender(state);
  const views = getViews(state);
  const currentTab = getCurrentTab(state);
  return { component, components, views, currentTab };
}

const mapDispatchToProps = dispatch =>
   bindActionCreators({ updateComponent, selectComponent }, dispatch);


export default connect(mapStateToPros, mapDispatchToProps)
(PropertyMenuContainer);
```

 # PropertyMenu.jsx

```
import React from 'react';
import PropTypes from 'prop-types';
```

```
import { makeStyles } from '@material-ui/core/styles';
import {
  AppBar,
  Tabs,
  Tab,
  Typography,
  Box,
  Button
} from '@material-ui/core';

import i18n from 'i18next';

import TextMenu from './componentMenu/textMenu';
import StyleMenu from './componentMenu/styleMenu';
import ActionMenu from './componentMenu/actionMenu';


function TabPanel(props) {
  const { children, value, index, ...other } = props;

  return (
    <Typography
      component="div"
      role="tabpanel"
      hidden={value !== index}
      id={`simple-tabpanel-${index}`}
      aria-labelledby={`simple-tab-${index}`}
      {...other}
    >
      <Box p={3}>{children}</Box>
    </Typography>
  );
}

TabPanel.propTypes = {
  children: PropTypes.node,
  index: PropTypes.any.isRequired,
  value: PropTypes.any.isRequired,
```

```
};

function a11yProps(index) {
  return {
    id: `simple-tab-${index}`,
    'aria-controls': `simple-tabpanel-${index}`,
  };
}

const useStyles = makeStyles(theme => ({
  root: {
    flexGrow: 1,
    backgroundColor: theme.palette.background.paper,
  },
}));

export default function SimpleTabs(props) {
  const classes = useStyles();
  const [value, setValue] = React.useState(0);
  const { handleUpdateComponents, deleteItem, component, views } = props;
  const handleChange = (event, newValue) => {
    setValue(newValue);
  };
  const renderButton = (deleteItem) => {
     return ( <Button style={{ marginTop: 10, fontSize: 14 }} variant="contained"
onClick={deleteItem}>{i18n.t("delete-item")}</Button>);
  }
  return (
    <div className={classes.root}>
          <AppBar style={{ backgroundColor: "#ebebf0", color: "#090C12" }}
position="static">
        <Tabs value={value} onChange={handleChange} aria-label="simple tabs
example">
          <Tab label={i18n.t("text")} {...a11yProps(0)} />
          <Tab label={i18n.t("style")} {...a11yProps(1)} />
                                { ['button', 'iconButton', 'webview', 'select',
'textField'].includes(component.type) && <Tab label={i18n.t("actions")}
{...a11yProps(2)} />}
        </Tabs>
```

```
      </AppBar>
      <TabPanel value={value} index={0}>
        <TextMenu
          handleUpdateComponents={handleUpdateComponents}
          component={component}
        />
        {renderButton(deleteItem)}

      </TabPanel>
      <TabPanel value={value} index={1}>
        <StyleMenu
          handleUpdateComponents={handleUpdateComponents}
          component={component}
        />
        {renderButton(deleteItem)}
      </TabPanel>
      <TabPanel value={value} index={2}>
        <ActionMenu
          handleUpdateComponents={handleUpdateComponents}
          component={component}
          views={views}
        />
        {renderButton(deleteItem)}
      </TabPanel>
    </div>
  );
}
```

 # Server > index.js

```
const Koa = require('koa')
const Router = require('koa-router')
const Logger = require('koa-logger')
const Cors = require('@koa/cors')
const BodyParser = require('koa-bodyparser')
const Helmet = require('koa-helmet')
const respond = require('koa-respond')
```

```
const app = new Koa()
const router = new Router()

app.use(Helmet())

if (process.env.NODE_ENV === 'development') {
  app.use(Logger())
}

app.use(Cors())
app.use(BodyParser({
  enableTypes: ['json'],
  jsonLimit: '5mb',
  strict: true,
  onerror: function (err, ctx) {
    ctx.throw('body parse error', 422)
  }
}))

app.use(respond())

// API routes
require('./routes')(router)
app.use(router.routes())
app.use(router.allowedMethods())
app.use(require('koa-static')('./build'))

module.exports = app
```

 # widget > index.js

```
const { insertNode } = require('./components');

function buildWidgetsTree(components) {
   let widgets = initializeRows();
   components.forEach((c) => {
      index = c.layoutItem.y;
                  const filteredComponents = components.filter(comp =>
comp.layoutItem.y === c.layoutItem.y);
```

```
            if (filteredComponents.length > 1) {
                let items = []
                filteredComponents.forEach(component => {
                    items.push(insertNode(component, widgets));
                })
                widgets[index] = row(items);
            }
            else
                widgets[index] = row(insertNode(c, widgets))
        });
        return widgets;
    }


    function initializeRows() {
        rows = ["","","","","","","","","",""];
        return rows.map(_ => row(null));
    }

    const row = (component) => {
            return component ? `Row(children: <Widget>[${component}])` :
`Row(children: <Widget>[Container(height: 50)])`;
    }

    module.exports = {
        buildWidgetsTree
    }
```

 # components > index.js

```
    const { getButton } = require('./button');
    const { getCheckbox } = require('./checkbox');
    const { text } = require('./text/scaffold');
    const { flexible } = require('./layout/flexible/scaffold');
    const { getInput } = require('./input');
    const { select } = require('./select/select/scaffold');

    function insertNode(component) {
```

```
    switch (component.type) {
        case 'button':
            return flexible(getButton(component));
        case 'iconButton':
            return flexible(getButton(component));
        case 'textField':
            return flexible(getInput(component));
        case 'checkBox':
            return flexible(getCheckbox(component));
        case 'text':
            return flexible(text(component.widget).template);
        case 'select':
            return flexible(select(component.widget).template);
    }
}

module.exports = {
    insertNode
}
```

 # button > scaffold.js

```
const { getAction } = require('./behaviors');

const getTemplate = (properties) => {
    const { color, text, action, theme } = properties;
    const colorString = theme === 'primary' ?
    'color: Theme.of(context).primaryColor,' : theme === 'secondary' ?
    'color: Theme.of(context).accentColor,' : theme === 'custom' ?
    `color: Color(0xff${color.replace('#', '')}),` : '';
    const actionPressed = getAction(action);
    return `FlatButton(
            ${colorString}
            textColor: Colors.white,
            disabledColor: Colors.grey,
            disabledTextColor: Colors.black,
            splashColor: Colors.blueAccent,
            onPressed: () {
                ${actionPressed}
```

```
            },
            child:
                Text(
                    "${text}",
                    style: TextStyle(fontSize: 20.0),
                ),
        )`;
};

const button = (properties) => ({
    template: getTemplate(properties),
    minChildren: 0,
    maxChildren: 1,
    children: ['Text', 'Decorator', 'Icon']

})
module.exports = {
    button
}
```

 # button > behavior.js

```
function getAction(action) {
            const navigate = action.value.toLowerCase() === "camera" ? `${action.value}Page(camera: widget.camera)` : `${action.value}Page()`;
    switch(action.type) {
        case 'navigate':
            return `Navigator.push(
                context,
                MaterialPageRoute(
                    builder: (context) => ${navigate},
                ),

            );`
        default:
            return '/*...*/'
```

```
    }
}

module.exports = {
    getAction
}
```

 # code > index.js

```
const { sortByVerticalAxis } = require('./dataProcessing');
const { getTemplate, saveFile } = require('./file');
const { buildWidgetsTree } = require('./widget');
const { makeCodeConversion } = require('./generator');

async function generateCode (data) {
    data.forEach((view, index) => {
        const { components } = view;
                                const template = components.filter(f => ['camera','map','webview'].includes(f.type))
        if (template.length) {
            let fileContent = getTemplate(template[0].type);
            if (template[0].type === 'webview') {
                                            fileContent = fileContent.replace('$url', `'${template[0].widget.website}'`);
            }
            const stream = makeCodeConversion(fileContent, view.title);
            saveFile(stream, view.title);
        } else {
            if(index === 0) {
                let fileMainContent = getTemplate('firstPage');
                fileMainContent = fileMainContent.replace('$firstPage', view.title);
                        fileMainContent = fileMainContent.replace('$colorPrimary', `Color(0xff${view.palette.primary.main.replace('#', '')})`);
                    fileMainContent = fileMainContent.replace('$colorSecondary', `Color(0xff${view.palette.secondary.main.replace('#', '')})`);
```

```
                                fileMainContent = fileMainContent.replace('$appName',
view.appName);
                            fileMainContent = fileMainContent.replace('$import', `import
'${view.title}.dart';`);
                        saveFile(fileMainContent, 'main');
                    }
                    const orderedComponents = components.sort(sortByVerticalAxis);
                    const widgets = buildWidgetsTree(orderedComponents);
                    const fileContent = getTemplate('home');
                      const stream = makeCodeConversion(fileContent, view.title, widgets,
orderedComponents);
                    saveFile(stream, view.title);
                }
            })
        }

        module.exports = {
            generateCode
        }
```