

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Matheus Felipe Gonçalves Oliveira de Araújo

Solução de Software para Sistema de Doações

Uberlândia, Brasil

2021

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Matheus Felipe Gonçalves Oliveira de Araújo

Solução de Software para Sistema de Doações

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, Minas Gerais, como requisito exigido parcial à obtenção do grau de Bacharel em Sistemas de Informação.

Orientador: Profa. Dra. Maria Adriana Vidigal de Lima

Universidade Federal de Uberlândia – UFU

Faculdade de Ciência da Computação

Bacharelado em Sistemas de Informação

Uberlândia, Brasil

2021

Matheus Felipe Gonçalves Oliveira de Araújo

Solução de Software para Sistema de Doações

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, Minas Gerais, como requisito exigido parcial à obtenção do grau de Bacharel em Sistemas de Informação.

Trabalho aprovado. Uberlândia, Brasil, 24 de maio de 2021:

**Profa. Dra. Maria Adriana Vidigal de
Lima**
Orientador

**Profa. Dra. Christiane Regina Soares
Brasil**

Prof. Dr. Humberto Luiz Razente

Uberlândia, Brasil
2021

Dedico este trabalho a minha tia-avó Maria Sueli Gonçalves Alves, que sempre incentivou meu talento para tecnologia, me apoiava a cada desafio da minha vida e sempre teve orgulho de mim.

Agradecimentos

Quero agradecer, primeiramente, a minha professora orientadora, Maria Adriana Vidigal de Lima, na qual nunca me ministrou uma matéria, mas nos conhecemos ao longo da minha graduação e agora me orientou neste terceiro projeto. Muito obrigado por ter me incentivado nos dois projetos, por todo conhecimento transmitido e ter tido tanta paciência e empenho em trabalhar comigo.

A empresa Ene Soluções, por me darem a oportunidade e meios de executar este trabalho.

À minha namorada e amor da minha vida, Daiane Cristina da Silva Borges, por ter sempre me incentivado e estado ao meu lado.

Aos professores Ana Paula Fernandes, Pedro Frosi Rosa e Autran Macedo por terem me ajudado a superar minhas dificuldades quando eu achava que não iria conseguir.

À minha amiga Ivana Brito Bonfim, por ter sido o meu “Jean” na graduação. Sempre serei profundamente grato a tudo que fez por mim.

À Graça Cobalt que me apoia desde os meus primeiros passos na UFU, que foi no programa PIBIC-Júnior em 2014.

À minha amiga Sayonara Martins, por sempre me dar apoio e aguentar meus devaneios. Isso não é para qualquer um.

À minha amiga Luana de Paula, por não me deixar me auto-sobrecarregar e dizer que tudo vai ficar bem. Essas ações simples fizeram toda a diferença para mim.

Aos meus colegas, Tiago Henrique, Elisangela Rithiely, Ana Victória Teixeira, Carolina Alves, Juliana Rodrigues e Ademar Neto, obrigado por todos os momentos de descontração e por tornarem este capítulo da minha vida único e memorável.

“Talk is cheap. Show me the code.” (Linus Torvalds)

Resumo

Ações cívico-sociais como doações e serviços voluntários são temas sempre em alta para se obter contribuições para diversas causas. A evolução das tecnologias digitais, o acesso à uma gama cada vez maior da população à internet, o surgimento de plataformas para contribuir com causas cívico-sociais e a disseminação de informações pelas redes sociais, fizeram com que as doações realizadas pela internet ganhassem impulso. Este trabalho tem como objetivo construir um aplicativo para dispositivos móveis multiplataforma e uma página *web*, visando contribuir para execução de ações particulares de doações sem envolver dinheiro para entidades filantrópicas/beneficentes. Os *softwares* propostos foram desenvolvidos usando o formato MVP e a biblioteca JavaScript React.

Palavras-chave: Ações Sociais, Sistema de Doação, Campanhas, React, Multiplataforma.

Lista de ilustrações

Figura 1 – Capturas de tela do aplicativo “Solidarius”. Fonte: (MORESI et al., 2017)	17
Figura 2 – Capturas de tela do aplicativo “Paraná Solidário”. Fonte: (PORTAL, 2020)	18
Figura 3 – Capturas de tela do aplicativo “Doarti”. Fonte: (DOARTI, 2020)	19
Figura 4 – Capturas de tela do aplicativo “Uma Ajuda Boa”. Fonte: (ITEGRA, 2020)	20
Figura 5 – Definições breves de MVP e propósito. Fonte: (LENARDUZZI; TAIBI, 2016)	23
Figura 6 – Relação de dependência entre as classes do padrão arquitetural de <i>software</i> MVC. Fonte: (VALENTE, 2020)	26
Figura 7 – Relação entre componentes de diferentes plataformas usando componentes nativos do <i>React Native</i> como base de comparação. Fonte: (REACTNATIVE, 2020)	27
Figura 8 – Telas de <i>onboarding</i> que compõem a introdução ao aplicativo para dispositivos móveis.	31
Figura 9 – Demonstração do espaço de trabalho dentro da ferramenta Trello com os cinco <i>sprints</i> representados em quadros.	35
Figura 10 – Demonstração de um quadro Kanban com tarefas representados por cartões dispostos em colunas.	36
Figura 11 – Representação de uma tarefa selecionada na forma de um cartão maximizado para interação do usuário.	37
Figura 12 – Visão geral da <i>Styleguide</i> do projeto.	38
Figura 13 – Trecho do código fonte do arquivo de valores de <i>design tokens</i> e uma porção de valores de estilo da <i>styleguide</i> .	40
Figura 14 – Tela de doação com campos de preenchimento sendo exibidos e ocultados.	41
Figura 15 – Medidas e valores de estilo de um componente botão de uma tela na ferramenta Zeplin.	42
Figura 16 – Seções de telas na ferramenta Zeplin e Figma.	43
Figura 17 – Partes do código fonte do processo de confirmar o recebimento de uma doação pela instituição.	45
Figura 18 – Pastas de componentes da tela de perfil no sistema de arquivos ao lado do trecho do código fonte da utilização de dois deles.	46
Figura 19 – Trechos de código fonte de dois componentes: um com gerenciamento de estado próprio e o outro sem.	47

Figura 20 – Diagrama conceitual de direção de fluxo de dados da biblioteca MobX. Fonte: (WESTSTRATE, 2021b)	48
Figura 21 – Trecho principal do código fonte da loja raiz de gerenciamento de dados.	50
Figura 22 – Diagrama comparativo do fluxo de dados cliente-servidor tradicional e da plataforma Firebase. Fonte: (STEVENSON, 2018)	51
Figura 23 – Diagrama do modelo e estrutura de dados proposto na forma de docu- mentos.	53
Figura 24 – Captura de telas do métodos de autenticação no aplicativo.	54
Figura 25 – Diagrama da hierarquia de pastas e arquivos do Cloud Storage.	55
Figura 26 – Versões do <i>layout</i> da <i>landing-page</i> para diferentes resoluções de tela.	56
Figura 27 – Captura das tela de uma requisição de junção de <i>commit</i> da <i>branch</i> da funcionalidade para a de homologação.	58
Figura 28 – Barras de navegação para as visões do aplicativo	61
Figura 29 – Captura de tela da seção de início da visão do doador.	62
Figura 30 – Captura de tela do formulário de doação preenchido.	63
Figura 31 – Captura da tela de doações na modalidade Buscar e Levar.	64
Figura 32 – Capturas da tela de detalhes de uma campanha de doação e do formu- lário de doação para campanha.	65
Figura 33 – Captura da tela de detalhes de antes e depois de realizar uma colabo- ração na modalidade Buscar e Levar.	66
Figura 34 – Captura das telas, seção instituições e perfil da instituição.	67
Figura 35 – Captura de tela da seção Conheça.	68
Figura 36 – Captura de tela da seção de início da visão do doador.	69
Figura 37 – Captura de tela da seção de início da visão da instituição.	70
Figura 38 – Captura das telas de confirmar o recebimento de uma doação e os detalhes dela.	71
Figura 39 – Captura de tela do formulário de criar nova campanha.	72
Figura 40 – Capturas de tela da seção de Campanhas e detalhes de uma campanha.	73
Figura 41 – Captura de tela da seção de Perfil na visão do usuário instituição.	74
Figura 42 – Captura de tela da janela modal de cadastro de gestor de doações.	75
Figura 43 – Captura da tela de editar o perfil do usuário instituição.	76
Figura 44 – Captura da tela de editar configurações de conta e senha do usuário instituição.	77
Figura 45 – Captura de tela da seção de Início da <i>landing-page</i>	78
Figura 46 – Captura de tela da seção Crie Sua Conta da <i>landing-page</i>	78
Figura 47 – Captura de tela do formulário de cadastro de instituição da <i>landing-page</i> .	79

Lista de abreviaturas e siglas

iOS	iPhone Operational System
macOS	Macintosh Operational System
CRM	Customer Relationship Management
BI	Business Intelligence
MVP	Minimum Viable Product
API	Application Program Interface
DOM	Document Object Model
HTML	Hyper Text Markup Language
XML	Extensible Markup Language
JSON	JavaScript Object Notation
IAM	Identity Access Management
DB	Data Base
BaaS	Back-end as a Service
SDK	Software Develepoment Kit
SSL	Secure Sockets Layer
CDN	Content Delivery Network

Sumário

1	INTRODUÇÃO	12
1.1	Objetivos	13
1.2	Organização do trabalho	14
2	METODOLOGIA	15
2.1	Modelagem e prototipação	15
2.2	Projeto da arquitetura de software e desenvolvimento	15
3	TRABALHOS CORRELATOS	17
3.1	Aplicações de <i>software</i> levantadas	17
3.1.1	Solidarius	17
3.1.2	Paraná Solidário	18
3.1.3	Doarti	19
3.1.4	Uma Ajuda Boa	20
3.2	Análises e considerações	21
4	FUNDAMENTAÇÃO TEÓRICA	22
4.1	<i>Minimum Viable Product</i> (MVP)	22
4.2	Conceitos e ferramentas de criação de telas protótipos e MVP	23
4.3	Arquitetura de software	24
4.4	Tecnologias de desenvolvimento do <i>front-end</i>	26
4.5	Tecnologias de desenvolvimento do <i>back-end</i>	29
5	DESENVOLVIMENTO	31
5.1	Gerenciamento do desenvolvimento	31
5.1.1	Estado inicial do projeto	31
5.1.2	<i>Personal Scrum</i>	32
5.2	Organização & Modelagem	38
5.2.1	<i>Styleguide</i>	38
5.2.2	<i>Design tokens</i>	39
5.2.3	Aplicativo & <i>landing-page</i>	40
5.3	Implementação <i>front-end</i> & <i>back-end</i>	43
5.3.1	Aplicativo	43
5.3.1.1	Padrões de projeto	44
5.3.1.2	Gerenciamento de estado	48
5.3.2	Banco de dados & Serviços	50

5.3.2.1	Cloud Firestore	51
5.3.2.2	Firebase Authentication	53
5.3.2.3	Cloud Storage	55
5.3.3	<i>Landing-page</i>	56
5.4	Controle de versionamento de código	57
6	RESULTADOS OBTIDOS	59
6.1	Conceitos do aplicativo	59
6.1.1	Instituição	59
6.1.2	Doador voluntário	59
6.1.3	Campanha	60
6.1.4	Doação	60
6.1.5	Buscar e levar	60
6.2	Seções do aplicativo	60
6.2.1	Visão doador	61
6.2.1.1	Início	61
6.2.1.2	Instituições	67
6.2.1.3	Conheça	68
6.2.1.4	Perfil	68
6.2.2	Visão instituição	70
6.2.2.1	Início	70
6.2.2.2	Campanhas	72
6.2.2.3	Perfil	74
6.3	Seções da <i>landing-page</i>	77
7	CONCLUSÃO	80
7.1	Trabalhos futuros	80
	REFERÊNCIAS	82
	APÊNDICES	85
	APÊNDICE A – LISTA DE TODAS AS TAREFAS DOS CINCO SPRINT RELACIONADOS A PRODUÇÃO DOS SOFTWARES DESTE TRABALHO	86
	APÊNDICE B – DIAGRAMA DE ENTIDADE E RELACIONAMENTO	90
	APÊNDICE C – DIAGRAMA DE CASO DE USO DA VISÃO GE- RAL DO SISTEMA	91

1 Introdução

A Organização das Nações Unidas (ONU), instituiu o ano de 2001 como o Ano Internacional do Voluntariado, promovendo discussão sobre o tema e valorizando a participação da sociedade civil nos inúmeros problemas sociais que são desafios em várias nações. No Brasil, de acordo com o Art. 1º da Lei Nº 9.608 de 18 de fevereiro de 1998, considera-se serviço voluntário a “atividade não remunerada prestada por pessoa física a entidade pública de qualquer natureza ou a instituição privada de fins não lucrativos que tenha objetivos cívicos, culturais, educacionais, científicos, recreativos ou de assistência à pessoa”. Estas iniciativas, em conjunto com os inúmeros problemas sociais enfrentados pelo Brasil, justificam o fato de a sociedade brasileira encontrar-se em um período de valorização e ampliação do espaço da sociedade civil, tendo em vista o surgimento de novas organizações sociais, a ampliação da quantidade de voluntários e de espaços para a prática do voluntariado nas mais diversas áreas da sociedade ([SELLI; GARRAFA, 2005](#)).

As tecnologias digitais desde sua emergente evolução a partir século XXI, vêm se tornando cada vez mais aliadas nos mais diversos aspectos do cotidiano da vida humana. Para [Fé e Moura \(2008\)](#), o cenário é de uma revolução digital, onde os dispositivos móveis como telefones celulares e *tablets*, por meio das tecnologias móveis que são as redes de conexão sem fio, tornaram-se instrumentos convergentes para quase todas as soluções das necessidades do homem. Por exemplo, um dispositivo móvel como um telefone celular permite realizar ligações, capturar fotos, pagar contas, assistir conteúdos que antes eram somente da televisão, ouvir música, enviar *e-mails* ou mensagens instantâneas, comprar basicamente qualquer coisa e ainda contribuir com causas humanitárias por meio de aplicações cívico-sociais.

As doações realizadas na forma virtual pela internet são o próximo patamar da digitalização e ganham impulso pela força e disseminação das redes sociais. Um exemplo é a plataforma Vakinha de financiamento coletivo que surgiu de forma tímida em 2009 no Brasil e alcançou a marca histórica em 2018 com R\$ 50 milhões arrecadados e 150 mil “vaquinhas” criadas. De acordo com Nicolle Stad (criadora da Inti, uma plataforma de gestão, CRM, BI, ingressos, inscrições para cursos e doações), as doações para organizações têm ganhado volume: “Quando começou a compra *online*, existia um pouco de desconfiança. Cada vez mais o brasileiro está se familiarizando com o consumo digital, e isso acaba espelhando em doação também” ([PAMPLONA, 2019](#)).

Uma pesquisa da revista Brasil Giving Report ([CAF, 2018](#)) revelou que de 1022 pessoas entrevistadas no Brasil de 2017 a 2018, 57% realizariam mais doações se tivessem mais dinheiro, pois a doação em dinheiro é um dos métodos mais utilizados pelas pessoas

(68%). A pesquisa identificou que a principal razão das pessoas entrevistadas realizarem doações é “porque faz com que se sintam bem”, seguido de “preocupar-se com a causa” e “querer ajudar as pessoas menos favorecidas”. Realizar uma doação é uma atitude cidadã para somar esforços e contribuir para uma sociedade melhor. Desta forma, não considerando fazer doações em dinheiro, mas ressaltando as outras possibilidades de doação, esta proposta visa promover mais engajamento cívico para a causa do voluntariado, que está estagnada no mesmo patamar desde 2017, conforme apontado pela pesquisa.

Este trabalho é inspirado em uma iniciativa de um aplicativo para viabilizar campanhas e ações particulares de doações, concebida na empresa Ene Soluções da cidade de Uberlândia-MG. A iniciativa do projeto despertou a atenção para o fato de ser uma nova proposta de auxílio à pessoas carentes, sempre em destaque, independentemente da ocorrência de calamidades, promovendo o uso de tecnologia para ações mais nobres.

Os *softwares* que serão desenvolvidos neste trabalho visam propor uma nova solução para promover ações sociais de caridade e filantropia, com a criação de campanhas para arrecadar recursos materiais ou serviços relevantes para caridade e filantropia. O aplicativo para dispositivos móveis proverá também o acompanhamento de todos os passos do processo referentes a uma doação, desde a obtenção da intenção da doação, com campanhas criadas pelas instituições, até a entrega do objeto ou serviço doado.

A aplicação será desenvolvida para dispositivos móveis como *smartphones* e *tablets*, para os sistemas operacionais Android e iOS. Será construída uma página *web* no formato *landing-page* com foco no cadastramento de instituições, na divulgação e na apresentação da aplicação. A *landing-page* terá a finalidade de engajar as pessoas a serem doadoras voluntárias, converter os visitantes da página em voluntários e a realizarem o *download* do aplicativo para dispositivos móveis. A página possuirá um formulário para cadastro de informações, onde as instituições deverão preencher para se cadastrarem. A proposta é que a plataforma seja multilinguagem, com suporte aos idiomas português, inglês e espanhol.

1.1 Objetivos

O objetivo geral deste trabalho é desenvolver uma aplicação de *software* cívico-social para a criação de campanhas de doações, onde o foco é doar recursos materiais e não dinheiro. A aplicação de software tem em vista viabilizar uma plataforma de engajamento social para além dos padrões de contribuição monetária, permitir que pessoas que querem fazer alguma contribuição a sociedade tenham essa possibilidade. Sendo mais otimista, a aplicação pode ampliar a atuação das pessoas e instituições em causas sociais através da reunião de campanhas e de solicitações, chamando a atenção das pessoas que desejam fazer parte de atividades que promovam o bem comum.

Os objetivos principais deste trabalho são:

- Desenvolver a solução proposta a fim de atender os requisitos levantados resumidos no diagrama do Apêndice C, utilizando, para isso, tecnologias comumente utilizadas no mercado.
- Disponibilizar a criação de campanhas por instituições para aquisição de recursos materiais ou serviços;
- Disponibilizar a criação de doações de itens e serviços por usuários;
- Disponibilizar solicitação do cadastro de instituições por meio da *landing-page* da aplicação e o cadastro de usuários por meio do aplicativo para dispositivos móveis;
- Implementar a consulta e visualização de instituições que possuem ações de doações ativas (campanhas);
- Implementar o controle de entrega e recebimento dos materiais doados.

1.2 Organização do trabalho

Para uma melhor separação e compreensão do conteúdo, os próximos capítulos deste trabalho estão organizados da seguinte maneira:

- O Capítulo 2 descreve a metodologia proposta para o desenvolvimento do trabalho;
- O Capítulo 3 apresenta uma breve análise dos trabalhos correlatos;
- O Capítulo 4 apresenta os conceitos básicos necessários para compreensão do trabalho;
- O Capítulo 5 mostra como o trabalho foi desenvolvido desde a etapa da modelagem até a parte de implementação do projeto. Nele estão presentes diagramas e trechos de código-fonte do projeto;
- O Capítulo 6 contém as principais telas, funcionalidades dos *softwares* desenvolvidos neste trabalho e suas respectivas descrições;
- O Capítulo 7 expõe as conclusões e considerações finais do trabalho, além de propostas para continuação do mesmo.

2 Metodologia

Para que os objetivos apresentados na Seção 1.1 sejam alcançados, os seguintes passos foram propostos para o desenvolvimento deste trabalho:

2.1 Modelagem e prototipação

- Levantamento bibliográfico sobre o tema de doações para estabelecer embasamento teórico e estado da arte atual, impacto social e formas de aplicação;
- Estudar ferramentas de gestão de doações em instituições;
- Levantamento dos requisitos do sistema tendo como base os artefatos produzidos na etapa de análise inicial do sistema. Adequação dos requisitos para a metodologia ágil Scrum;
- Organizar o *wireframing* de guia visual para estabelecer a estrutura básica do aplicativo;
- Estudo de referencial teórico de desenvolvimento de software para dispositivos móveis considerando diferentes sistemas operacionais;
- Verificação da compatibilidade através dos dispositivos móveis para identificar padrões e criar modelo de estruturas de dados.

2.2 Projeto da arquitetura de software e desenvolvimento

- Estudo de trabalhos relacionados;
- Aplicar a metodologia de desenvolvimento ágil de software Scrum distribuindo os requisitos no *backlog* do produto entre cinco *sprints*: (i) revisão da configuração base do projeto, (ii) desenvolvimento das telas do *front-end*, (iii) desenvolvimento dos serviços de consumo de dados com regras de negócio (*back-end*), (iv) integração do *front-end* com o *back-end* e (v) testes de software e aprimoramentos;
- Realizar modelagem do projeto do sistema de banco de dados;
- Desenvolvimento do aplicativo para dispositivos móveis e da *landing-page* no formato MVP (*Minimum Viable Product*) com as seguintes tecnologias:

- *Front-end* utilizando a biblioteca JavaScript *React* tanto para o aplicativo para dispositivos móveis e a *landing-page*. Em particular o aplicativo para dispositivos utilizará também o *React Native* e a *landing-page* o *React Virtual DOM*;
 - *Back-end* utilizando um sistema de banco de dados NoSQL orientado a documentos (Cloud Firestore), comunicação de dados em lotes, eventos entre a aplicação e o banco de dados usando o Cloud Functions da plataforma Google Firebase.
- Testes iniciais do software para verificações de garantia de qualidade.

3 Trabalhos correlatos

Neste capítulo serão apresentados alguns trabalhos que abordam aplicações desenvolvidas com semelhante próxima à deste trabalho. Serão citadas as aplicações de *software* Solidarius, Paraná Solidário, Doarti e Uma Ajuda Boa, que promovem ações de doações e trabalho voluntário no Brasil.

3.1 Aplicações de *software* levantadas

3.1.1 Solidarius

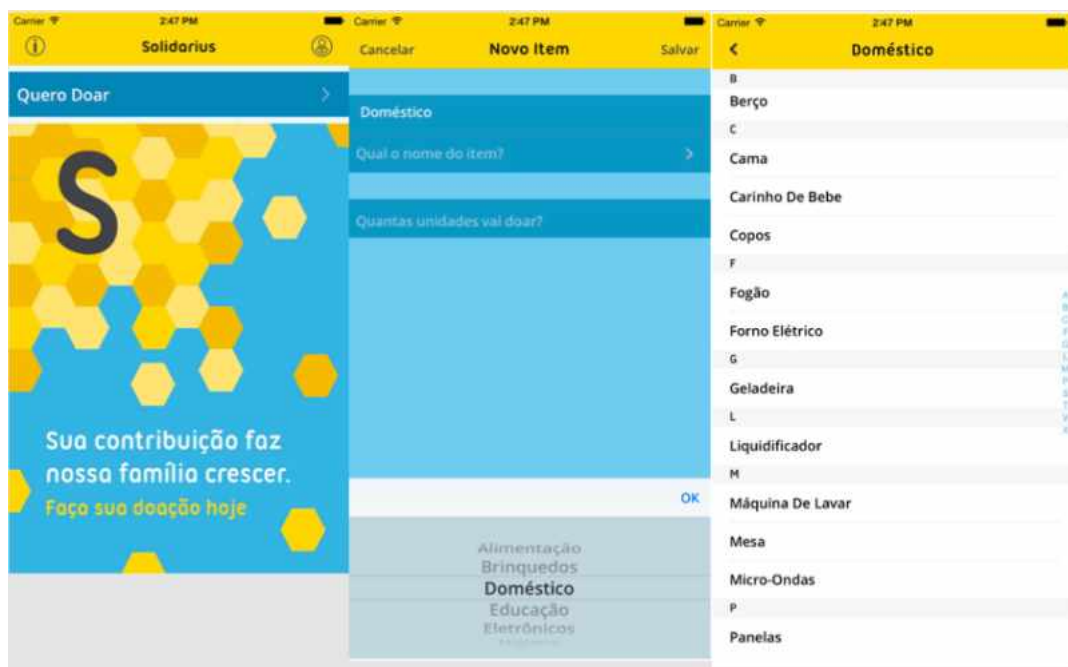


Figura 1 – Capturas de tela do aplicativo “Solidarius”. Fonte: (MORESI et al., 2017)

Uma aplicação de *software* brasileira com os objetivos e funcionalidades que mais se assemelham com o desse trabalho, foi um aplicativo desenvolvido no projeto BEPiD (Brazilian Education Program for iOS Development) da Universidade Católica de Brasília, publicado oficialmente para consumo somente nas lojas de aplicativos oficiais dos dispositivos móveis da plataforma iOS, a App Store e iTunes Store em abril de 2016. O objetivo do projeto foi o de formular um triângulo de comunicação, entre usuários que tem condições de ajudar com produtos à uma instituição de caridade, por segundo as próprias instituições de caridade, e como terceiro uma instituição sendo o intermediador que transporta os produtos para as famílias necessitadas, assim como instituições de caridade referidas (MORESI et al., 2017).

O Solidarius possuía uma parte externa de administração, uma página na *web* para gerenciamento das doações, controle dos dados e elaboração de relatórios.

Até o momento da escrita deste trabalho, o aplicativo Solidarius ainda não se encontra disponível na loja de aplicativos App Store, assim também na iTunes Store, pois a mesma foi descontinuada pela empresa desenvolvedora, a Apple Inc.

3.1.2 Paraná Solidário

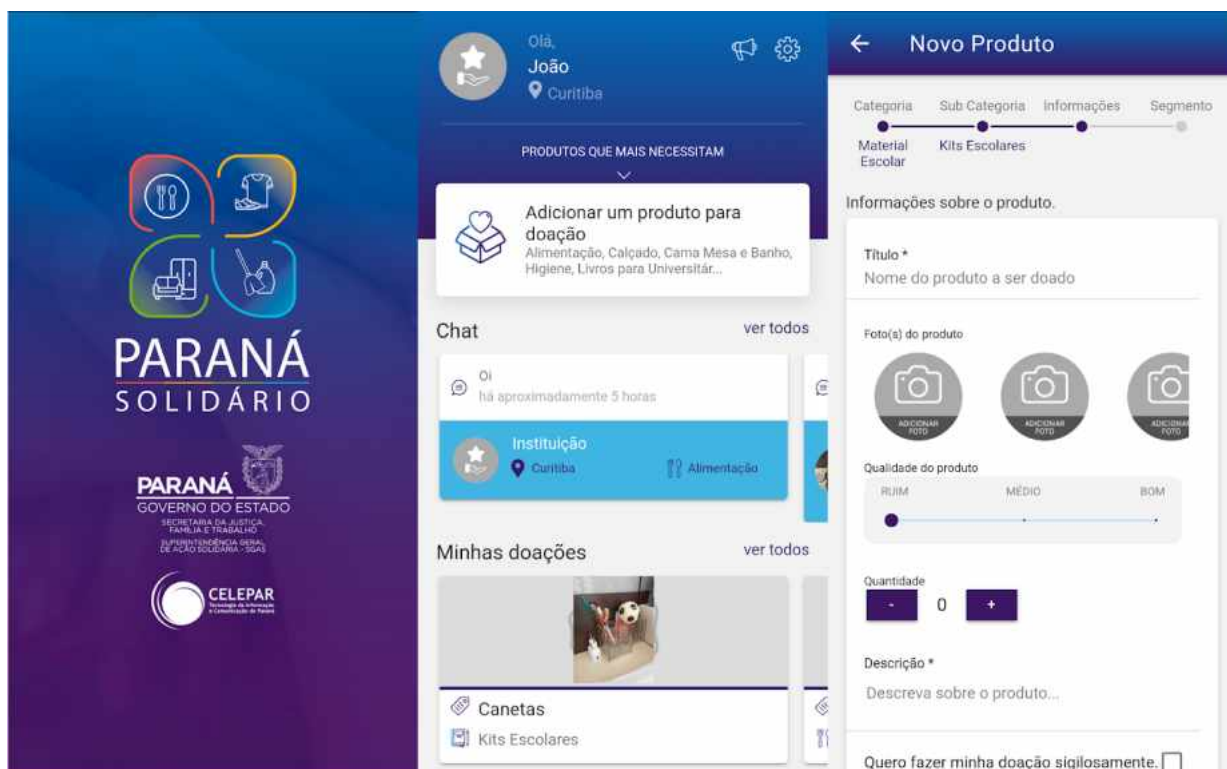


Figura 2 – Capturas de tela do aplicativo “Paraná Solidário”. Fonte: (PORTAL, 2020)

Um aplicativo muito recente que contém uma gama de funcionalidades semelhantes ao proposto neste trabalho é o Paraná Solidário. O aplicativo foi criado pelo Governo do Estado do Paraná para ajudar a causa das doações, que haviam sofrido quedas drásticas no volume de doações provocadas pela pandemia da COVID-19. Lançado em 24 de junho de 2020, para promover a doação direta entre os cidadãos e as entidades beneficentes (PARANÁ, 2020).

A ferramenta é uma ponte entre empresas e pessoas físicas que queiram doar para instituições de caridade produtos como alimentos, roupas, móveis, eletrodomésticos, ração para animais, entre outros. Todas as instituições de caridade que estão no aplicativo, estão cadastradas juntas ao Governo do Estado do Paraná. Os doadores que pretendem realizar uma doação, precisam se cadastrar na plataforma, encaminhar informações e

fotos dos objetos e escolher qual instituição irá receber esses produtos (PORTAL, 2020). A uma funcionalidade chamada “Serviços Voluntários”, no qual o usuário já cadastrado, pode complementar seu cadastro para se tornar um voluntário e deixar registrado uma disponibilidade para instituições que necessitam de algum auxílio presencial.

Este aplicativo até a escrita deste trabalho está disponível para as plataformas de dispositivos móveis Android e iOS.

3.1.3 Doarti

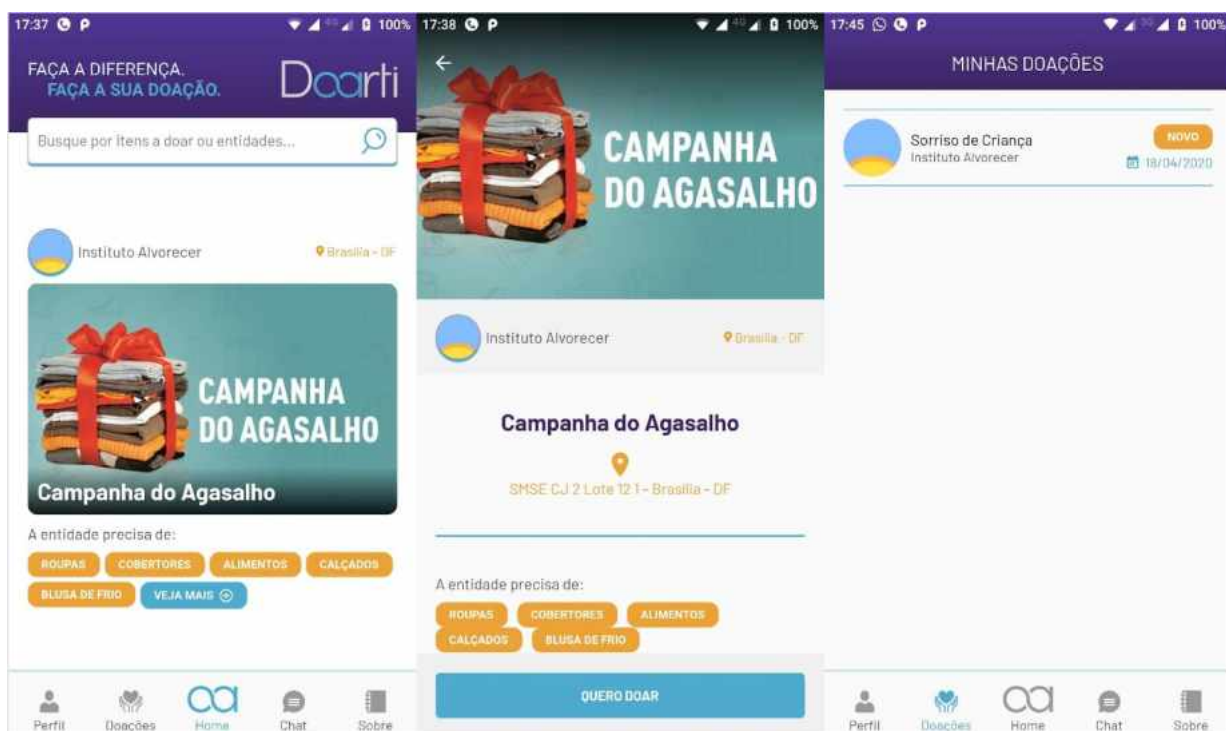


Figura 3 – Capturas de tela do aplicativo “Doarti”. Fonte: (DOARTI, 2020)

Outro aplicativo tão recente quanto o anterior, também brasileiro e para a causa de doações impactadas pela pandemia da COVID-19 é o Doarti. Uma iniciativa que desenvolveu uma página *web* e um aplicativo na Universidade de Brasília (UnB) por uma equipe de 16 pessoas, formada por professores da Faculdade UnB Gama (FGA), ex-alunos do curso de Engenharia de Software da Universidade, profissionais do mercado, pela empresa Main Class e da Rede Solidária Anjos do Amanhã que é um programa de voluntariado da Vara da Infância para aumentar o fluxo de doações no Distrito Federal (DOARTI, 2020).

O aplicativo para dispositivos móveis possui funcionalidades como cadastro de doadores, consulta de entidades que possuem ações de doações ativas, envio de notificação aos doadores sempre que uma nova ação é registrada, tipos de doações que podem ser realizadas, controle de entrega e recebimento de doações, entre outras. Na documentação

apresentada em [Doarti \(2020\)](#) constam todas as informações e critérios que as entidades deverão cumprir para participarem da iniciativa, assim como as funcionalidades de cadastro e atualização dos dados. O aplicativo e a página *web* são complementares um ao outro. O primeiro é direcionado a doadores e entidades, o segundo, apenas a entidades.

Até o momento da escrita deste trabalho, o cadastramento de instituições de doações está limitado às regiões administrativas do Distrito Federal (DF) e à cidade de Brasília.

O lançamento da página *web* ocorreu no dia 21 de abril de 2020. Já o aplicativo para dispositivos móveis foi lançado no dia 27 de abril de 2020. Este aplicativo até a escrita deste trabalho está disponível para as plataformas de dispositivos móveis Android e iOS.

3.1.4 Uma Ajuda Boa

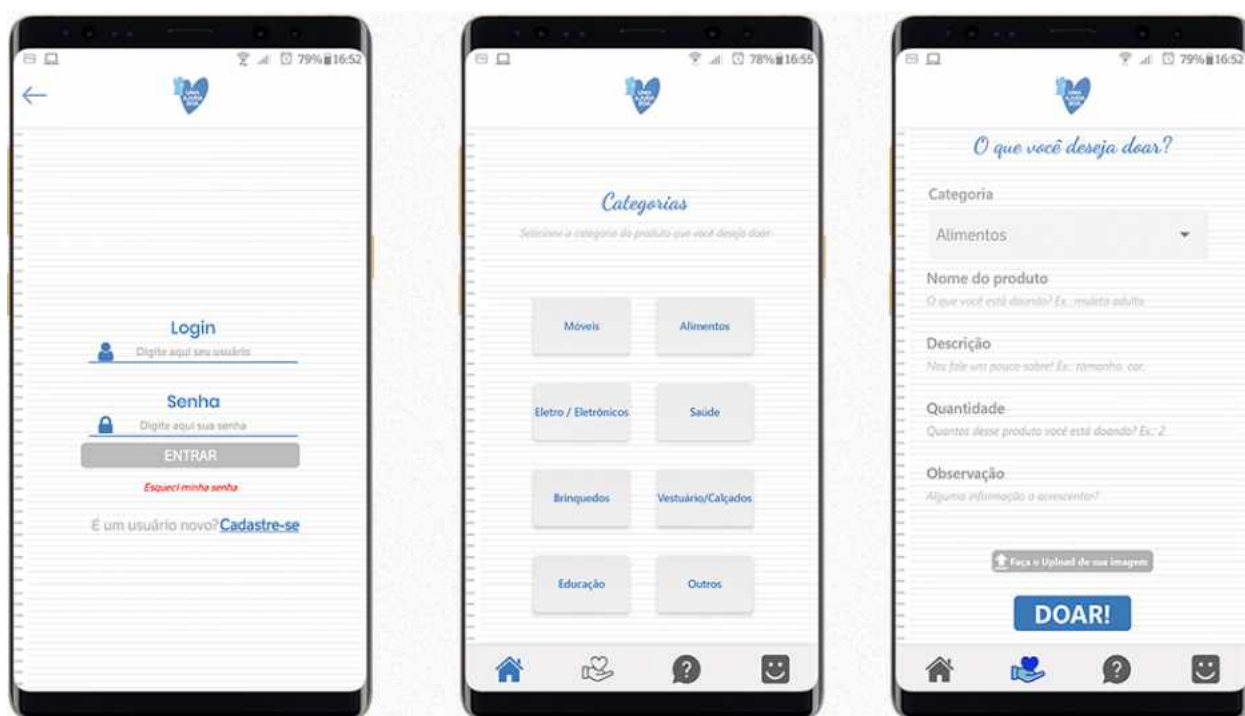


Figura 4 – Capturas de tela do aplicativo “Uma Ajuda Boa”. Fonte: ([ITEGRA, 2020](#))

O aplicativo Uma Ajuda Boa é um aplicativo brasileiro de uma iniciativa da Associação Anjo Rafael, desenvolvido em 2019 pela empresa Itegra Desenvolvimento. O aplicativo possui a finalidade de conectar doadores regulares e pessoas que queriam fazer uma doação, com instituições de auxílio previamente cadastradas que precisam de ajuda ([ITEGRA, 2020](#)).

Com um cadastro simples, com apenas informações mínimas, qualquer um pode doar de forma segura algum item que se tem em casa como: roupas, móveis, livros, computadores, material de escritório e material de construção por exemplo. Em poucos passos, o usuário consegue disponibilizar o que quer doar e as instituições cadastradas recebem uma notificação. Aquela que aceitar primeiro, fica responsável para buscar o item onde o doador indicar.

As instituições cadastradas passam por um processo rigoroso de análise, e precisam cumprir uma série de exigências, tudo isso para garantir a segurança de todo esse processo.

Este aplicativo até a escrita deste trabalho, está disponível para as plataformas de dispositivos móveis Android e iOS.

3.2 Análises e considerações

As quatro aplicações de *softwares* descritas anteriormente (sendo que somente três estão disponíveis atualmente até escrita deste trabalho) possuem um aplicativo para dispositivos móveis com funcionalidades em comum, como cadastro de instituições/entidades beneficentes ou filantrópicas, cadastro de usuários, criação de campanhas por instituições e realização de doações pelo usuários. Também possuem uma página *web* como *landing-page*, para solicitações de cadastros ou para gerenciamento externo da aplicação.

Até o momento da escrita deste trabalho somente o aplicativo Doarti da Seção 3.1.3 possuía o método de doação na forma serviços, que também está planejado para ser implementado no *software* deste trabalho. Nenhuma das aplicações disponíveis atualmente citadas anteriormente, oferece uma solução como opção, para um método de doação por meio da realização do transporte do material doado da localidade do usuário doador até a instituição destinatária.

Levando em consideração o que foi constatado, há algumas aplicações de *software* disponíveis no mercado brasileiro para contribuir com a causa de doações de recursos materiais. A aplicação de *software* proposta neste trabalho ainda apresenta relevância com potencial para melhorar a causa de doação no Brasil.

4 Fundamentação teórica

Neste capítulo são abordados os conceitos e tecnologias acerca da temática de sistemas de doações utilizadas para o desenvolvimento deste trabalho. Eles estão agrupados nas seguintes seções: *Minimum Viable Product*, conceitos e ferramentas de criação de protótipos e MVP, arquitetura de *software*, tecnologias de desenvolvimento do *front-end* e tecnologias de desenvolvimento do *back-end*.

4.1 *Minimum Viable Product* (MVP)

O termo *minimum viable product* é comumente utilizado por organizações do tipo *startup* para concepção de um produto ou serviço. Para (RIES, 2009 apud LENARDUZZI; TAIBI, 2016) MVP é uma versão de um novo produto, que permite a uma equipe coletar o máximo de aprendizado validado sobre os clientes com o mínimo esforço. É um processo iterativo de geração de ideias, prototipagem, apresentação, coleta de dados, análise e aprendizagem. Outros tipos de organizações, principalmente no setor da tecnologia, têm adotado este conceito para criação de seus produtos com objetivo de produzi-los assertivamente para um segmento de clientes alvo para mais chances de sucesso. Tendo em vista os fatores apresentados, este é o formato que os softwares deste trabalho serão produzidos.

A Figura 5 mostra a proposta e as definições importantes para um MVP:

Mínimo	Propósito principal do MVP
Funcionalidades Mínimas	Permite o produto ser implantado Atingir oportunidades do mercado Criar um produto viável para o cliente Testar as hipóteses fundamentais de negócio Permite testar o produto no mercado Coletar <i>feedback</i> do cliente Identificar as funcionalidades mais viáveis pela experimentação interativa do mercado
Requisitos Mínimos	Conhecer as necessidades dos primeiros adotantes
Mínimo de implementação possível	Motivar os clientes a fornecer <i>feedback</i> e recomendar para outros clientes Trazer valor agregado para o cliente
Esforço Mínimo	Coletar a quantidade máxima de conhecimento validado dos clientes Testar as hipóteses fundamentais de negócio Desenvolver um produto que inclui apenas as funcionalidades necessárias para permitir os primeiros adotantes a prover <i>feedback</i> Reunir <i>feedback</i> do cliente
Valor mínimo para organização	Acelerar vendas para os primeiros clientes

Figura 5 – Definições breves de MVP e propósito. Fonte: (LENARDUZZI; TAIBI, 2016)

4.2 Conceitos e ferramentas de criação de telas protótipos e MVP

A concepção e modelagem inicial da aplicação, foi obtida primeiramente através de um *wireframe* de baixa fidelidade com rascunhos de elementos da interface, como botões, seções de conteúdo, campos de texto, formulários e entre outros.

Para a modelagem das telas do protótipo e MVP da aplicação foi utilizado uma ferramenta de criação de interface e *design* de experiência do usuário, chamada Sketch. Esta ferramenta desenvolvida pela Sketch B.V para dispositivos com sistema operacional macOS, é usada primariamente para o desenho vetorial de interfaces de usuários tanto para dispositivos móveis quanto páginas *web* (SKETCH, 2020). A ferramenta tem foco em criação de protótipos de forma simples e prática, com vantagens de se trabalhar com tipografia em relação a ferramentas concorrentes como o Adobe XD (ADOBE, 2021). A vantagem dessa ferramenta em relação a tipografia, é facilidade de manipulação de elementos e propriedades textuais durante a modelagem de uma tela de *software* e na exportação da mesma.

Outra ferramenta comumente utilizada em conjunto com o Sketch é o Zeplin. Esta é uma ferramenta colaborativa entre *designers* e desenvolvedores e auxilia o desenvolvedor em visualizar, inspecionar e exportar todos os elementos criados para uma tela de *software* e suas propriedades de *layout* (ZEPLIN, 2020).

Para se ter um padrão de estilo do layout da aplicação tanto para *design* quanto para desenvolvimento, uma *Style Guide* de interface de usuário foi criada. *Style Guide* é um conjunto de símbolos, artefatos, regras e propriedades de elementos visuais e textuais que a aplicação deve conter. Elementos esses como cores, tipografia e espaçamentos, formatos de elementos e comportamentos.

A modelagem da *Style Guide* e das telas da aplicação foram construídas influenciadas pelas heurísticas de Nielsen ou princípios de usabilidade de Nielsen, seguem-nas sendo.

1. Visibilidade do status do sistema;
2. Combinação do sistema entre o mundo real;
3. Controle do usuário e liberdade;
4. Consistência e padrões;
5. Prevenção de erros;
6. Reconhecimento em vez de recordação;
7. Flexibilidade e eficiência de uso;
8. *Design* estético e minimalista;
9. Ajudar os usuários a reconhecer, diagnosticar e se recuperar de erros;
10. Suporte e documentação.

São portanto 10 princípios gerais de usabilidade para criação de interfaces para dispositivos digitais, com a finalidade de proporcionar interfaces mais acessíveis e amigáveis ao usuário (NIELSEN, 2005). O estudo desses princípios heurísticos foram fundamentais para definição da tipografia da aplicação com a parametrização de medidas e espaçamentos específicos para que a fonte e elementos do *software* aumentem de tamanho, com objetivo de estabelecer um aspecto harmônico visual para o usuário, proporcionando uma usabilidade fluída da aplicação. A parametrização de pesos de fontes e espaçamentos entre elementos é essencial para que o usuário seja capaz de identificar grupos de informações na tela.

4.3 Arquitetura de software

As aplicações e sistemas de *software* vêm se tornando cada vez maiores e mais complexos desde a popularização do uso de dispositivos digitais. A maior motivação vem

de uma crescente necessidade de transformar ou reformular atividades do cotidiano da vida humana com o uso de *softwares* para torná-las menos trabalhosas e mais rápidas de se realizar. Como consequência do aumento da complexidade das aplicações de *software*, cresce também a importância da construção de um código-fonte organizado, documentado e com divisões bem definidas com relação às funcionalidades e à lógica do problema. Desta forma, garante-se que a codificação esteja legível para todos os envolvidos na construção e na manutenção do software.

Aplicações de *software* não são entidades únicas com partes indivisíveis, elas são formadas por componentes e camadas que devem ser relevantes para que a mesma atinja seus objetivos. Tradicionalmente, são compostas por uma camada de interface de interação com usuário, camada de processamento da regra de negócio e uma de sistema de banco de dados para persistir os mesmos. Tendo em vista esta perspectiva, uma aplicação de *software* necessita ter sua concepção projetada previamente ao desenvolvimento em si.

De acordo com (VALENTE, 2020), arquitetura de *software* trata da organização de um sistema em um nível de abstração mais alto do que aquele que envolve classes ou construção semelhante. Projetar a arquitetura de um *software* inclui as decisões de projeto mais importantes em um sistema. Essas decisões uma vez feitas, dificilmente poderão ser revertidas no futuro. Projetar uma arquitetura não é apenas definir um conjunto de módulos, mas um conjunto de decisões a partir do que se quer construir, como a escolha das linguagens de programação, bibliotecas externas, sistema de banco de dados que serão utilizados no desenvolvimento. É fato que, uma aplicação de *software* se implementada utilizando um determinado sistema de banco de dados, dificilmente consegue-se migrar para outro.

As camadas que constituem uma aplicação de *software* como um todo podem estar desacopladas por uma questão de organização ou necessidade, mas devem estar aptas a comunicarem entre si. Cada camada possui responsabilidades e como podem estar desacopladas e funcionando independentes, é necessário definir uma organização arquitetural. Ainda de acordo com (VALENTE, 2020) padrões arquiteturais propõem uma organização de mais alto nível para aplicações de *software*, incluindo seus principais módulos e as relações entre eles. Essas relações definem, por exemplo, que um módulo A pode (ou não pode) utilizar os serviços de um módulo B. Alguns padrões de *software* populares são: Arquitetura em Camadas, Arquitetura Model-View-Controller, Arquitetura orientada a Mensagens e Arquitetura Publish/Subscribe.

A base da aplicação de *software* deste trabalho será projetada utilizando os padrões Arquitetura em Camadas em conjunto com o padrão Arquitetura Model-View-Controller ou popularmente conhecido padrão MVC. Foram previstas duas camadas, uma para visão e controle da interface do usuário (*front-end*) e outra para manipulação e persistência dos dados da aplicação (*back-end*). A Figura 6 demonstra a relação de dependência

entre as classes do padrão arquitetural de *software* MVC. As classes da camada Visão e Controladores compõem a estrutura de base da Interface Gráfica. Pode-se observar que a Interface Gráfica também pode depender do Modelo, porém classes de Modelo não têm dependências para classes da Interface Gráfica.

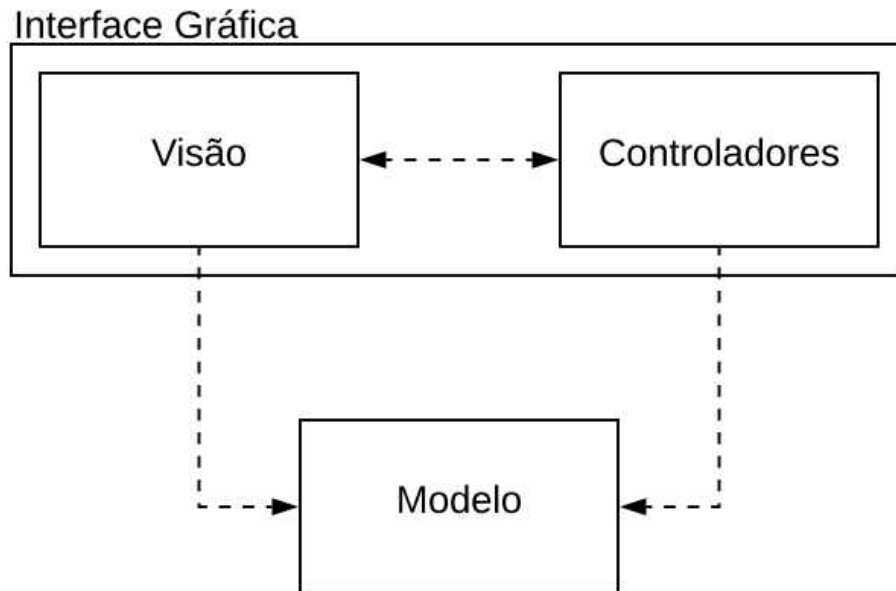


Figura 6 – Relação de dependência entre as classes do padrão arquitetural de *software* MVC. Fonte: (VALENTE, 2020)

Os padrões arquiteturais de software são o resultado do trabalho e da experiência de desenvolvedores em resolver questões importantes na criação de software com qualidade, e sintetizam conjuntos de regras e boas práticas que podem ser seguidos na engenharia de software. Ainda, a separação clara de camadas e os demais aspectos arquiteturais trazem diversos benefícios para a aplicação em geral, como testabilidade e manutenibilidade.

4.4 Tecnologias de desenvolvimento do *front-end*

A linguagem JavaScript é o coração da aplicação, tanto do aplicativo para dispositivos móveis quanto da *landing-page*. A linguagem é uma das três principais tecnologias do desenvolvimento de animações e interações de páginas *web*. Em 2015 a linguagem sofreu reformulações em sua sintaxe e funcionalidades possibilitando assim uma implementação e manipulação de comportamentos e eventos mais robusta para as páginas *web*. Criar páginas *web* e aplicativos para dispositivos móveis são tarefas complexas, mesmo após a evolução do JavaScript e criação de novas linguagens (Swift para o iOS e Kotlin para o Android) para desenvolvimento de aplicativos para os dispositivos móveis. Com a

finalidade de tornar o desenvolvimento de páginas *web* menos complexo e mais performáticos, começaram a surgir *frameworks* reativos e progressivos como *Angular* e *Vue*, além de bibliotecas como o *React* (EISENMAN, 2015).

Cross-platform (multiplataforma) é um termo popular no contexto desenvolvimento de aplicativos móveis. É uma abordagem que não segue o modo tradicional de desenvolvimento de aplicativos em plataformas nativas, mas que são desenvolvidos em plataformas, *frameworks* ou bibliotecas que fornecem aplicativos para diferentes plataformas, como por exemplo a plataforma Xamarin que fornece aplicações para Android, iOS e Windows com base no *framework* .NET na linguagem C# (MICROSOFT, 2020) e o *React* que em particular tem a API *React Virtual DOM* para desenvolvimento de páginas *web* e a *React Native* que fornece aplicativos para Android e iOS usando a linguagem JavaScript.

A Figura 7 exemplifica a equivalência de um componente nativo criado em *React Native* com componentes nativos criados nas plataformas nativas dos sistemas operacionais Android e iOS. Uma coluna adicional apresenta uma analogia a elementos da linguagem de marcação HTML, utilizados em páginas *web*:

REACT NATIVE UI COMPONENT	ANDROID VIEW	IOS VIEW	ANALOGIA WEB	DESCRIÇÃO
<View>	<ViewGroup>	<UIView>	Sem rolagem <div>	Um contêiner que suporta um leiaute com flexbox, estilo, eventos de interação e controles de acessibilidade
<Text>	<TextView>	<UITextView>	<p>	Exibição, estilo, cadeias de texto e manipula eventos de interação
<Image>	<ImageView>	<UIImageView>		Exibe diferentes tipos de imagens
<ScrollView>	<ScrollView>	<UIScrollView>	<div>	Um contêiner genérico com rolagem que pode conter múltiplos componentes e views
<TextInput>	<EditText>	<UITextField>	<input type="text">	Permite o usuário entrar com texto

Figura 7 – Relação entre componentes de diferentes plataformas usando componentes nativos do *React Native* como base de comparação. Fonte: (REACTNATIVE, 2020)

O desenvolvimento do código fonte do *front-end*, que resultará na interface interativa do usuário com a aplicação que será desenvolvida utilizando a biblioteca JavaScript chamada *React*. Para o desenvolvimento da *landing-page* será utilizado a API declarativa do *React* de manipulação da árvore de elementos (DOM) do HTML nos navegadores de páginas *web*, chamada *React Virtual DOM* (REACT, 2020c). Para o desenvolvimento do aplicativo para dispositivos móveis, será realizado utilizando a API declarativa também

do *React*, o *React Native*. O *React Native* implementa o paradigma declarativo de interface de usuário (*React's declarative UI paradigm*) em conjunto com JavaScript permitindo criar componentes *React* que compartilham a interação com as API's nativas dos sistemas operacionais Android e iOS ([REACTNATIVE, 2020](#)).

A biblioteca JavaScript *React* e suas API's foram desenvolvidas pela empresa Facebook Inc. A utilização da API *ReactDOM* para criar, manipular e renderizar componentes *React*, em navegadores de páginas *web* é realizado por meio de uma linguagem denominada JSX. A JSX é uma extensão de sintaxe para JavaScript semelhante a uma linguagem de *template*, que permite ao desenvolvedor escrever códigos da linguagem de marcação HTML dentro do código JavaScript ([REACT, 2020b](#)).

O padrão de arquitetura de *software* deste projeto (MVC) será utilizado em conjunto com o padrão *Component Pattern* do *React*. O padrão MVC define uma organização das camadas de um *software*, separando em camadas denominadas *Model*, *View* e *Controller*. A camada *Model* é responsável por manipular os dados do *software*, a *View* é responsável pela definição e disposição dos elementos visíveis para o usuário interagir e a camada *Controller* tem o papel de detectar e propagar eventos de alteração para a camada *Model* advindas da camada *View* ([GAMMA; JOHNSON, 2007](#)). O *React* adota o conceito de componentes para criar estruturas complexas da aplicação de forma eficiente. Um componente no *React* é uma unidade de lógica de renderização visual pouco acoplado com uma lógica de manipulação de interface de usuário. Dividir um *software* em componentes menores para se ter partes independentes e reutilizáveis com vários níveis de isolamento destes componentes ([REACT, 2020a](#)). Neste contexto nota-se que o uso do *React* permite a união das camadas *Controller* e *View*. Esse acoplamento permite que, internamente no componente exibido na interface do usuário, já se encontre a definição de suas ações e como elas são transmitidas para a camada de dados (*Model*).

A criação dos componentes seguirá o padrão *Container Component Pattern* que é utilizado para criar componentes chamados *stateful components*, no qual são compostos por componentes que gerenciam seus próprios estados além de renderizar interface independente da lógica encapsulada, passando dados para os componentes filhos na forma de propriedades. O padrão *Presentational Component Pattern* também foi utilizado para criar componentes a partir de uma função pura que são denominados sem estado (*stateless components*), pois não gerenciam seu próprio estado. São responsáveis apenas por renderizar uma composição de interface preenchida com dados do componente pai ([REACT, 2020a](#)).

4.5 Tecnologias de desenvolvimento do *back-end*

A camada *back-end* de uma aplicação é uma abstração da relação da comunicação e controle de dados entre a interface de usuário (*front-end*) e o banco de dados. É uma parte oculta ao usuário que contém a implementação da lógica da regra de negócio que permite o funcionamento da aplicação. O código fonte do *back-end* e o sistema de banco de dados geralmente residem e são processados em um servidor. A camada *back-end* tem o papel fundamental de proporcionar diversos pontos-chaves de comunicação com a camada *front-end* de acordo com a necessidade da aplicação de consumir e persistir dados no banco de dados para finalmente recuperar informações e exibi-las na interface de usuário de maneira apropriada.

No mercado há diversas linguagens de programação e plataformas de desenvolvimento para construir um *back-end*, já que é uma camada com implementação desacoplada do contexto da aplicação. BaaS (*Back as a Service*) é o termo para se referir a uma plataforma que integra e coordena serviços de *back-end*, implementados em diversas linguagens e permite administrar recursos, como configurações de servidor, sistema de *push notifications*, armazenamento e gerenciamento de arquivos, integração com banco de dados, e autenticação de usuários. De acordo com (SHARMA; DAND, 2019) utilizar uma plataforma BaaS simplifica o desenvolvimento de uma aplicação de *software*, fazendo com que o desenvolvedor não tenha que se preocupar em implementar todas particularidades que formam um *back-end*, que é considerado um processo dispendioso e complexo. Um BaaS gerencia suas próprias API's para serviços de forma independente em um ambiente integrado. Algumas plataformas existentes com essa proposta são: Amazon AWS, Firebase, Parse e Kinvey.

As tecnologias utilizadas na implementação *back-end* deste trabalho, para assegurar a infraestrutura de armazenamento e parte da manipulação dos dados da aplicação serão construída com recursos da plataforma Firebase da empresa Google. É uma plataforma digital no modelo BaaS, que disponibiliza bibliotecas com arquitetura cliente/servidor para facilitar a criação de aplicações de páginas *web* e aplicativos para dispositivos móveis, com o objetivo de melhorar o rendimento das aplicações com a implementação de diversas funcionalidades que tornam a aplicação mais segura, maleável e acessível para o usuários (FIREBASE, 2020a). A motivação para o uso da plataforma Firebase como camada *back-end* no *software* deste trabalho se deve a disponibilização de uma gama de serviços que um *back-end* geralmente possui, com a finalidade de facilitar o desenvolvimento de protótipos e MVP, pois é possível focar no desenvolvimento *front-end* sem tem que se preocupar com todas as questões de criar um *back-end* do zero, tornando o processo mais rápido e simples.

É preciso definir um sistema de banco de dados no *back-end* para armazenar e recuperar os dados da aplicação. Os sistemas de bancos de dados SQL são tradicionalmente

utilizados, eles consistem em estruturar dados de forma relacional com apresentação de dados em tabelas no modelo linhas e colunas. Tem ganhado cada vez mais notoriedade os sistemas de banco de dados NoSQL que estruturam dados de maneira não relacional. Para ([SADALAGE; FOWLER, 2013](#)) os sistemas de banco de dados NoSQL possuem quatro categorias principais de modelagem de dados baseados em:

1. Chave-valor, por exemplo o Oracle NoSQL e Amazon DynamoDB;
2. Famílias de colunas, por exemplo o Apache Cassandra;
3. Grafos, por exemplo o Arango DB e Amazon Neptune;
4. Documentos, por exemplo o MongoDB e o Cloud Firestore;

O sistema de banco de dados escolhido para o desenvolvimento do *back-end* deste trabalho foi o Cloud Firestore, que é otimizado para armazenar grandes coleções de documentos pequenos. O modelo de dados do Cloud Firestore não utiliza esquemas, proporcionando assim total liberdade sobre quais campos inserir em cada documento e inferir o tipo de dados que se pode armazenar nesses campos. Os documentos se assemelham muito com documentos JSON, o que facilita a comunicação e manipulação desses dados no *front-end* da aplicação ([FIREBASE, 2020b](#)).

Por todos esses aspectos, as aplicações necessitam garantir segurança dos dados que gerenciam. O Cloud Firestore contém definições personalizadas de regras de segurança, na quais pode-se controlar o acesso a documentos e coleções do banco de dados. Usando a sintaxe de regras flexíveis, pode-se criar regras que correspondem a várias operações, de todas as gravações em todo o banco de dados até operações em um documento específico. O acesso aos recursos da aplicação pode ser controlado com o gerenciamento de identidade e acesso (IAM). Com o IAM, é possível adotar o princípio de segurança do menor privilégio para conceder apenas o acesso necessário aos recursos ([FIREBASE, 2020c](#)). É crucial que a comunicação entre o *back-end* e o *front-end* ocorra na forma de que alguém esteja autenticado na aplicação com um usuário específico e que somente os dados desse usuário estejam sendo manipulados. De maneira a garantir essa segurança será utilizado o Firebase Authentication, um serviço que providencia uma conta autora de autenticação para acessar a aplicação.

5 Desenvolvimento

Este capítulo apresenta o ambiente de desenvolvimento *front-end* e *back-end*, padrões de projetos, *frameworks* e a arquitetura utilizada durante o processo de implementação dos *softwares* deste trabalho (aplicativo para dispositivos móveis para os sistemas operacionais Android e iOS e uma *landing-page* para navegadores de páginas de internet).

5.1 Gerenciamento do desenvolvimento

5.1.1 Estado inicial do projeto

A codificação deste trabalho originou-se a partir de um estado inicial produzido pela equipe de desenvolvedores da empresa Ene Soluções incluindo o autor desde trabalho nesta equipe. O estado inicial do projeto continha apenas três telas de *onboarding* (apresentadas na Figura 8, que compõem a introdução do usuário ao aplicativo), a tela inicial do aplicativo (visão do doador), componentes básicos (campos de entrada de texto, campo de seletor de opções, janela dinâmica de seleção de opções (*dropdown*), alternador de opção lógico (*toggle*), botões com suas variações e especializações, cartões de exibição de informações, janela de diálogo/conteúdo sobreposto dinâmico (modal)), configurações prévias das plataformas Android e iOS, como, algumas fontes de texto, valores de tema padronizados (*design token*) e processos de *build*. Exceto as três telas de *onboarding* e os valores que compõem o *design token*, todos os componentes já existentes e configurações citados acima sofreram severas refatorações e aprimoramentos ao longo do desenvolvimento.

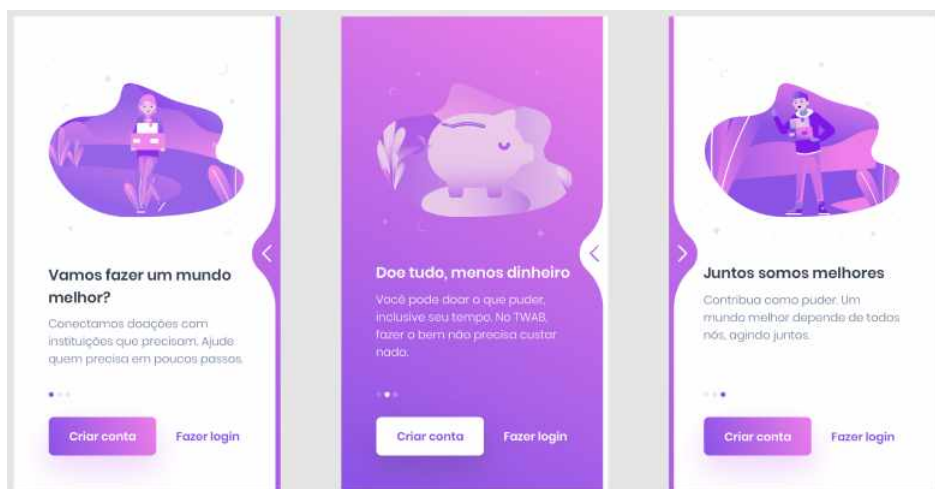


Figura 8 – Telas de *onboarding* que compõem a introdução ao aplicativo para dispositivos móveis.

5.1.2 *Personal Scrum*

Foi adotado um processo de *software*, nos moldes e princípios da metodologia ágil Scrum, para auxiliar o gerenciamento da produção do *software* deste trabalho: a *Personal Scrum* (PRUITT, 2011). De forma geral, o *framework* Scrum trabalha com ciclos curtos de desenvolvimento e entrega de software. Em consequência, seu *feedback* sobre o resultado pode ser obtido rapidamente, garantindo a qualidade do produto e a satisfação do cliente, que passa a fazer parte do processo e receber os resultados mais rapidamente (BONFIM, 2013). O *framework* Scrum é composto de:

- Pessoas: *product owner*, *Scrum master* e time Scrum;
- Atividades: *sprint*, planejamento do *sprint*, *daily's*, revisão do *sprint*, retrospectiva do *sprint*, duração do *sprint*;
- Artefatos: *product backlog*, *sprint backlog*, definição de pronto, incremento do produto, refinamento do *product backlog* e *burndown chart*.

O *Personal Scrum* é a adoção de algumas práticas, atividades e artefatos do *framework* Scrum em projetos onde só há uma pessoa. Neste trabalho foram utilizados os seguintes conceitos:

- *Sprint*: conjunto de atividades que em um determinado período de tempo são realizadas para desenvolver as funcionalidades do produto;
- *Product Owner*: representa o papel do cliente no projeto, sendo o responsável pelo valor agregado no produto construído, além da priorização dos requisitos. No *Personal Scrum* é o próprio desenvolvedor;
- *Scrum Master*: representa o papel do responsável por garantir que o Scrum seja seguido pelo time, sendo também responsável por remover os impedimentos encontrados pelo time na hora do desenvolvimento. No *Personal Scrum* é o próprio desenvolvedor;
- Planejamento do *sprint*: reunião com o time Scrum onde são escolhidos quais requisitos do *Product Backlog* serão desenvolvidos durante determinado *sprint*. No *Personal Scrum* o planejamento de *sprint* ocorre de forma similar ao tradicional, a única mudança é que deixaria de ser uma reunião, e entraria apenas como a atividade que o desenvolvedor realiza, escolhendo os requisitos a serem desenvolvidos no próximo *sprint*;
- Duração da *Sprint*: define o tempo de duração que terá o *sprint*;

- Refinamento do *Product Backlog*: ajuste do *Product Backlog*, para mantê-lo em ordem e pronto para que os requisitos sejam escolhidos no planejamento do *sprint*;
- *Product Backlog*: documento que contém uma lista com todos os requisitos do produto, e qualquer alteração ou novo requisito deve ser acrescentado neste documento;
- *Sprint Backlog*: documento que contém uma lista com todos os requisitos do produto que devem ser desenvolvidos durante um determinado *sprint*.

Foram planejados cinco *sprints* para a produção dos *softwares* deste trabalho contendo as tarefas listadas no Apêndice A e no diagrama de caso de uso geral do sistema mostrado no Apêndice C. Os itens a seguir resumem, para cada *sprint* as atividades, duração e prazo:

- *Sprint 1* - duração: 8 dias, período: 28/12/2020 à 04/01/2021
 1. Criar uma diagrama de entidade e relacionamento a partir de requisitos fomentados;
 2. Revisar telas, configurações, componentes e funcionalidades já existentes;
 3. Atualizar versões de dependências das plataformas Android e iOS;
 4. Revisões dos componentes existentes após atualizações e aprimoramentos ou refatoração necessários;
 5. Desenhar fluxograma de navegação entre telas do aplicativo a partir de artefatos existentes na ferramenta Figma.
- *Sprint 2* - duração: 25 dias, período: 05/01/2021 à 30/01/2021
 1. Desenvolvimento do *front-end* de todas as telas e componentes do aplicativo ainda restantes;
 2. Realizar testes do fluxo de navegação entre telas;
 3. Revisões dos componentes existentes após atualizações e aprimoramentos/refatorações necessárias;
 4. Realizar teste dos componentes com comportamentos nativo em suas devidas plataformas.
- *Sprint 3* - duração: 15 dias, período: 01/02/2021 à 16/02/2021
 1. Estudo do gerenciamento de estado interno da aplicações de *software*;
 2. Integração do projeto com dependência a Firestorter;

3. Criação das lojas de gerenciamento de estado da aplicação usando a biblioteca reativa Mobx;
 4. Estudo de como aplicar múltiplas estruturas no Cloud Firestore usando práticas de armazenamento apropriadas e execução de consultas para recuperação de dados;
 5. Criação da estrutura de dados do diagrama de entidade relacionamento no Cloud Firestore.
- *Sprint 4* - duração: 25 dias, período: 17/02/2021 à 13/03/2021
 1. Desenvolvimento do *back-end*;
 2. Criar consultas para recuperar dados do banco;
 3. Integrar o aplicativo com Firebase Performance e Firebase Crashlytics;
 4. Integrar o aplicativo e a *landing-page* com Firebase Storage.
 - *Sprint 5* - duração: 21 dias, período: 15/03/2021 à 04/04/2021
 1. Criar diagrama para representar a hierarquia resultante de arquivos implementada no Firebase Storage;
 2. Implementar *front-end* da *landing-page*;
 3. Desenvolver o *back-end* da *landing-page*;
 4. Hospedar *landing-page* no Firebase Hosting;
 5. Realizar testes de fluxo básico de realizar uma doação na visão do doador e recebimento da doação na visão da instituição;
 6. Realizar resolução de *bugs* e aprimoramentos nos *softwares*;
 7. Complementar fluxograma com os novos componentes e telas desenvolvidas. .

A ferramenta utilizada para a organização e o acompanhamento dos processos da *Personal Scrum* foi a plataforma Trello da empresa Atlassian Corporation Plc. A ferramenta possui um portfólio vasto de modelos para gerenciamento de diversos tipos de projetos, incluindo-se o Modelo Kanban. Criou-se inicialmente um espaço de trabalho dentro da plataforma Trello, como mostra a Figura 9, contendo quadros Kanban para anotação de atividades e configurações de cada *sprint*.

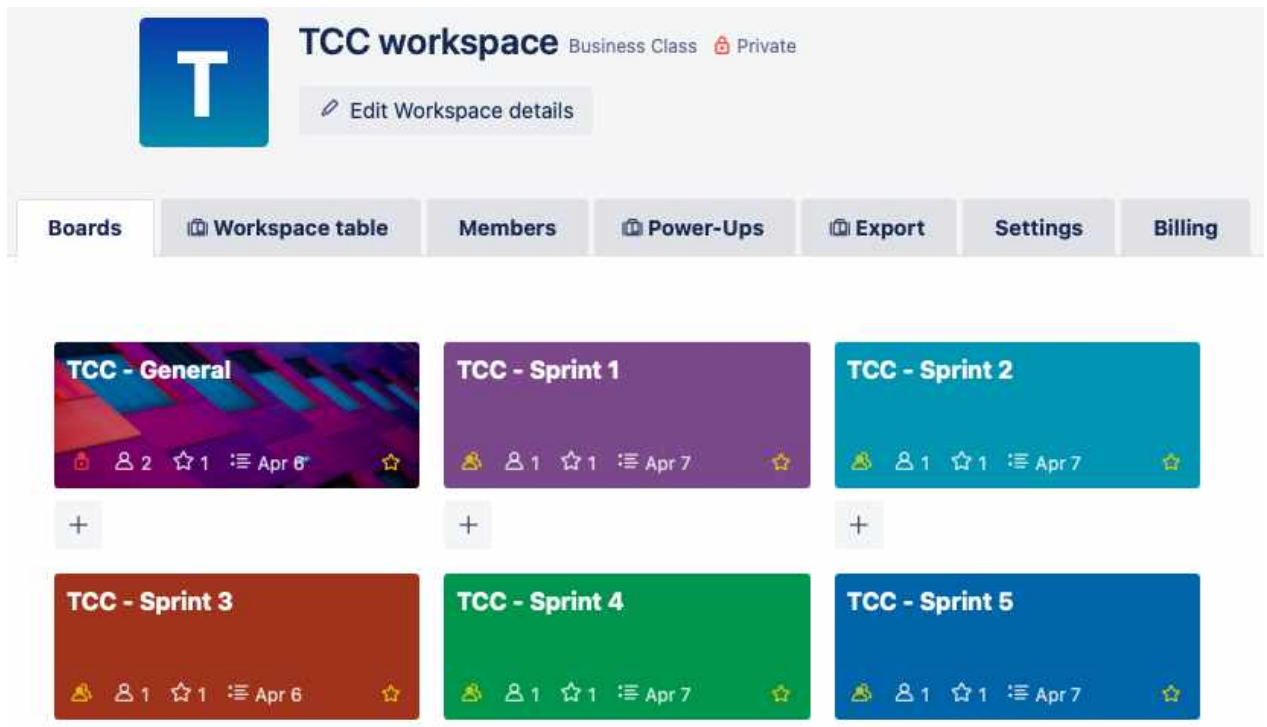


Figura 9 – Demonstração do espaço de trabalho dentro da ferramenta Trello com os cinco *sprints* representados em quadros.

A forma escolhida e comum para organizar e acompanhar as atividades da metodologia Scrum e *Personal Scrum* foi o Modelo Kanban. O Kanban permite a criação de tarefas representadas por cartões com uma prévia de informações do que deve ser feito, na forma de elementos textuais resumidos em colunas nomeadas com o status atual daquela tarefa, conforme mostra a Figura 10. Em um quadro Kanban as colunas são nomeadas de acordo com as etapas que cada tarefa deve seguir desde a sua criação até a sua conclusão. Para este trabalho foram criadas as seguintes colunas:

1. *To Do*: representa uma tarefa do *product backlog*, mas que não foi escolhida ainda para fazer parte do *sprint backlog*;
2. *Backlog*: representa uma tarefa do *sprint backlog*;
3. *Doing*: representa o estado da tarefa que está sendo desenvolvida atualmente;
4. *Testing*: representa o estado da tarefa para a qual está sendo realizado testes;
5. *Code Review*: representa o estado da tarefa para a qual está sendo realizada a revisão do código fonte no sistema de versionamento;
6. *Done*: representa o estado da tarefa concluída, ou seja, que foi codificada, testada, revisada e teve sua versão do código fonte atrelada à versão de código fonte do *software*.

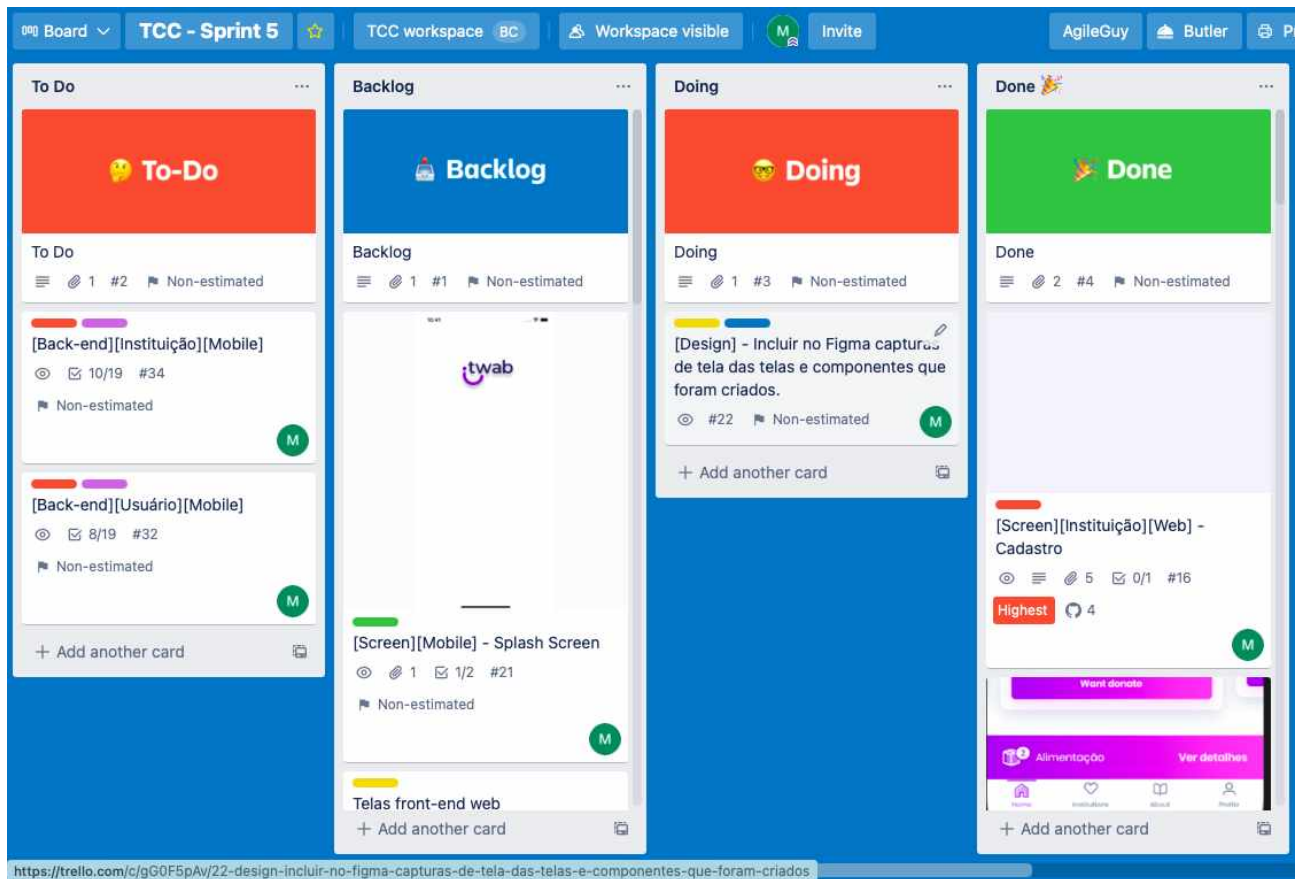


Figura 10 – Demonstração de um quadro Kanban com tarefas representados por cartões dispostos em colunas.

Cada cartão de uma determinada coluna do quadro representa uma tarefa com seus atributos na forma de elementos textuais dinâmicos dentro do quadro, expressando de forma veemente uma funcionalidade do *software*. A versão do cartão que é disposto nas colunas de um quadro contém uma síntese de informações na forma de elementos visuais resumidos em: título, descrição, prioridade, rótulo, a quem está atribuída, entre outros. Quando o cartão é selecionado, a sua versão maximizada é exibida de forma que todas as informações de uma tarefa que a aplicação e o usuário definiram sejam destacadas e fiquem disponíveis para interação. Como representado na Figura 11, notam-se os campos de título, descrição (*description*), a quem está atribuída a tarefa (*members*), rótulos (*labels*), rastreamento do tempo gasto na tarefa (*tracking*), anexos (*attachments*), informação dos ramos (*branches*) que representam o código fonte daquela tarefa do sistema de versionamento, data prevista para finalização da tarefa (*due date*), localização da tarefa no quadro (*in list Done*), definição de pontos de prioridade (*priority*) e listagem de sub-tarefas ou pontos de verificação relacionados à tarefa em exibição (*checklist*).

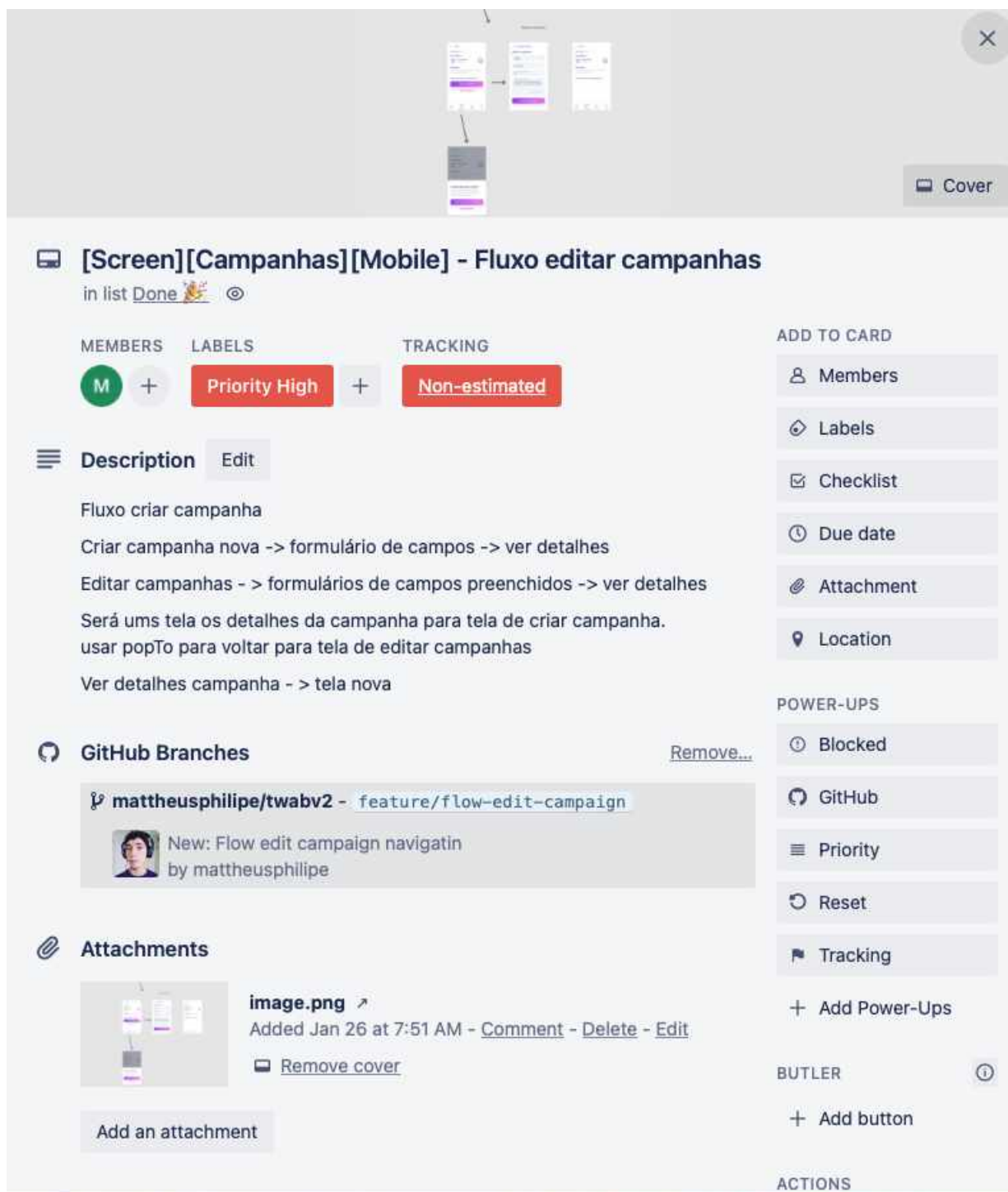


Figura 11 – Representação de uma tarefa selecionada na forma de um cartão maximizado para interação do usuário.

Outras ferramentas importantes no mercado como Jira Software também da Atlassian Coporation Plc, Phabricator da empresa Phacility e Notion da empresa Notion Labs Inc disponibilizam de maneira muito semelhante as funcionalidades descritas nesta seção na aplicação do Scrum na forma do modelo de quadros Kanban. Todas as funcionalidades

descritas nesta sub-seção e representadas pelas tarefas indicadas no Apêndice A para a aplicação dos conceitos da metodologia ágil do Scrum na plataforma Trello, não representam uma unanimidade universal de como deve ser utilizado por outras plataformas ou serviços, por mais que sigam um modelo com características em comum baseadas nos conceitos do modelo de quadros Kanban e do Scrum, cada uma possui suas singularidades para disponibilização e manuseio das funcionalidades.

5.2 Organização & Modelagem

A modelagem e produção de quase todos os conceitos, conteúdos e layouts de interface dos *softwares* MVP deste trabalho foram concebidos pela equipe de *designers* gráficos e da equipe de conteúdo da empresa Ene Soluções.

5.2.1 Styleguide

O guia de padrões de estilos do layout consistente de um projeto de *software* é de suma importância e tem como propósito guiar e facilitar a codificação, principalmente em uma equipe com muitos desenvolvedores e *designers*.

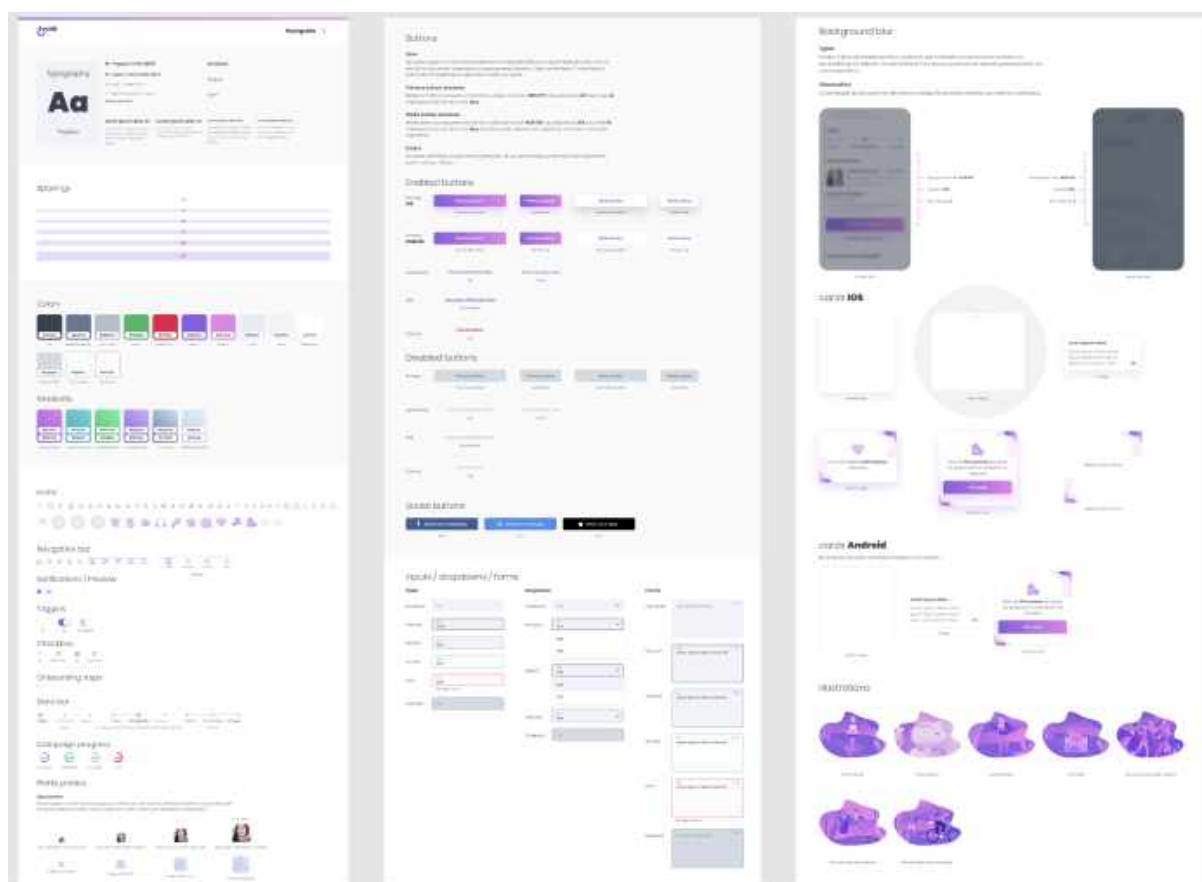


Figura 12 – Visão geral da *Styleguide* do projeto

A Figura 12 representa uma *styleguide* que contém todas as definições principais de conjuntos de símbolos, artefatos, regras de espaçamentos entre outras propriedades de elementos visuais e textuais, incluindo cores, tipografia, efeitos de animações, definição do estilo base de componentes individuais, variações de estados visuais de componentes e ícones que devem ser utilizados na codificação.

5.2.2 *Design tokens*

A partir do guia de padrões de estilos definidos, foi possível criar um tema global para o aplicativo para ser utilizado em todos os componentes. O tema criado foi configurado de tal forma que todo componente que necessita de uma estilização possa acessar seus valores estáticos facilmente sem a necessidade de importações extras. O tema implementado contém um único um arquivo composto com *design tokens* que são coleções de atributos visuais estáticos, como valores hexadecimais de cores, valores de espaçamentos, valores da tipografia (família, tamanho, peso e estilo da fonte) e entre outros.

A Figura 13 mostra como uma porção de estilos definidos na *styleguide*, foi utilizado para implementar os *design tokens* no código fonte do aplicativo. O lado esquerdo da imagem contém trechos do código fonte do arquivo com os valores de *design tokens*, por exemplo, valores hexadecimais de cores, tipografia e espaçamento de margens. O lado direito da figura contém uma porção da *styleguide* com algumas definições da tipografia, cores e espaçamento de margens.

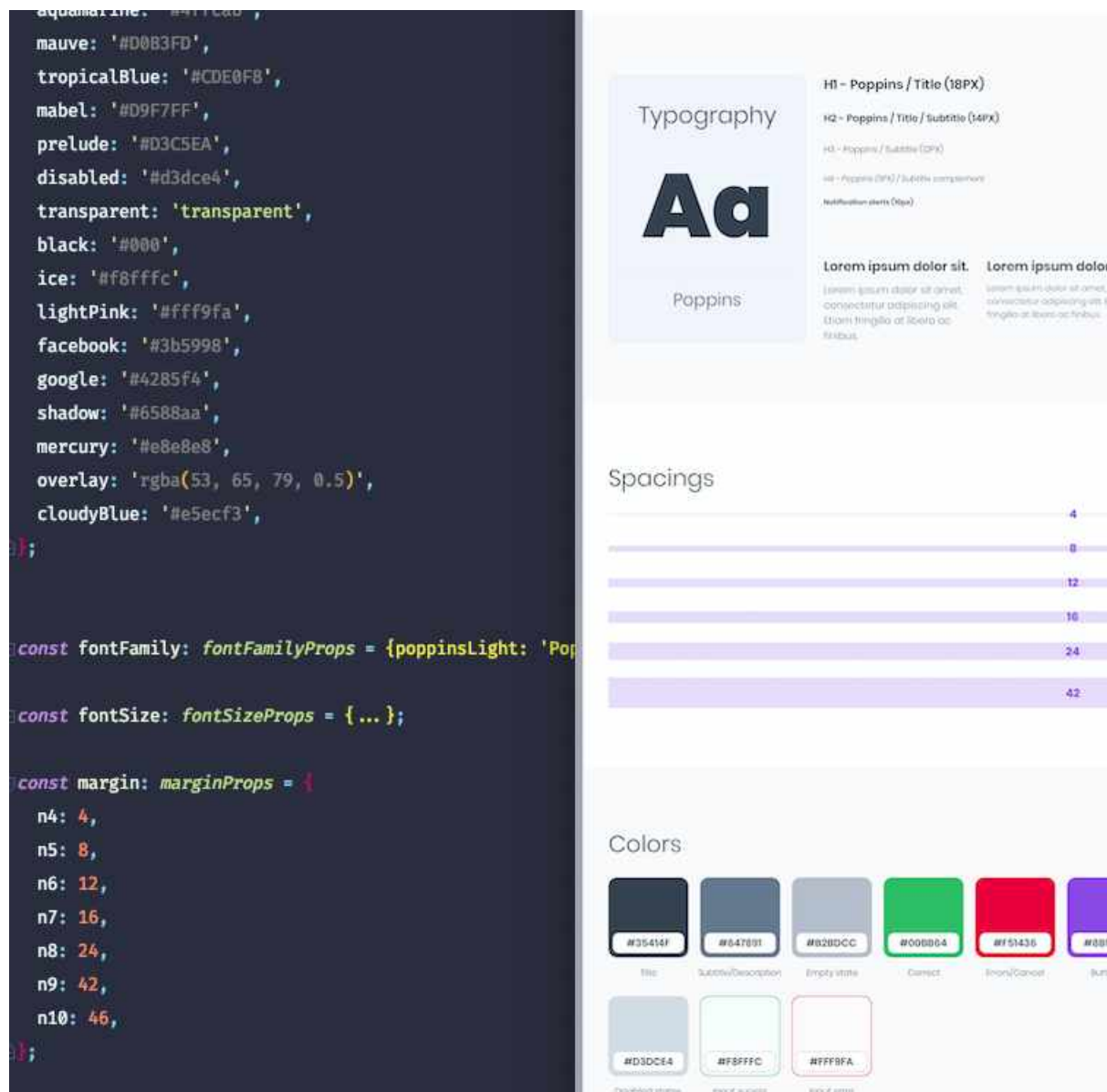


Figura 13 – Trecho do código fonte do arquivo de valores de *design tokens* e uma porção de valores de estilo da *styleguide*.

5.2.3 Aplicativo & landing-page

O *design* das telas do aplicativo para dispositivos móveis e da *landing-page* foram produzidos sob influência das heurísticas (princípios de usabilidade) de Nielsen (NIELSEN; MOLICH, 1990), em conjunto com o conceito “*design clean*” que tem sido uma tendência no mercado. O “*design clean*” norteia a produção de telas com menos elementos, resultando em pouca interferência visual, onde somente o necessário para ser utilizado na execução de alguma tarefa naquela tela esteja sendo exibido para o usuário.

A Figura 14 exemplifica como o “*design clean*” foi aplicado na tela de **Realizar uma doação**. Em ambos os lados da figura nota-se o texto **Vou levar a minha doação**

até a instituição seguido de um componente alternador de valor lógico (*booleano*) para indicar verdadeiro ou falso. No lado direito da figura percebe-se que o componente está indicando falso demonstrando a intenção do usuário de indicar que não irá levar sua doação até a instituição e campos de preenchimento complementares estão sendo exibidos. Em contrapartida o lado esquerdo da figura mostra o componente indicando verdadeiro e escondendo os campos desnecessários para o contexto da opção indicada pelo usuário.

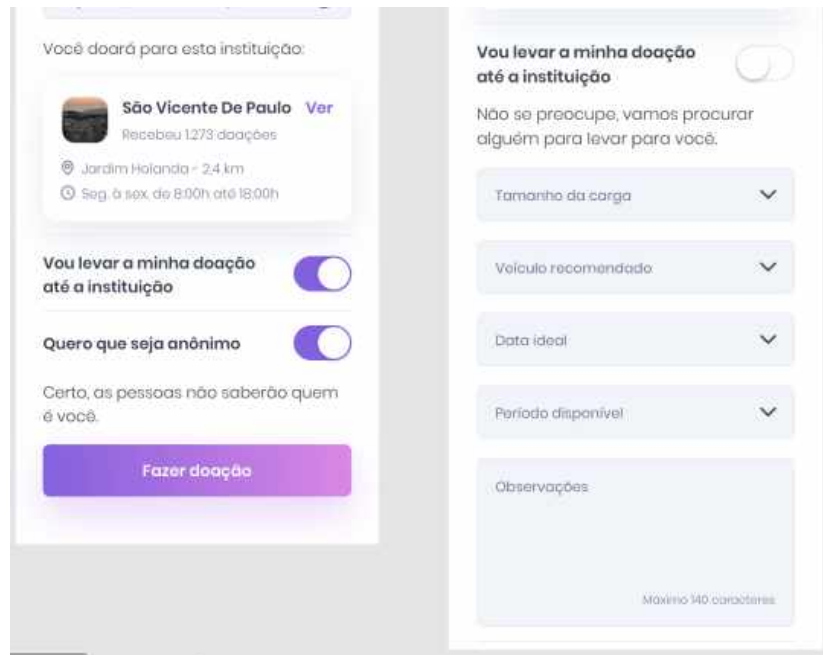


Figura 14 – Tela de doação com campos de preenchimento sendo exibidos e ocultos.

Toda a fase de concepção do *design* da *styleguide* e das telas do aplicativo e da *landing-page* foi construída dentro da ferramenta Sketch e exportada para a ferramenta Zeplin. A ferramenta Zeplin foi essencial para a obtenção de quase toda a informação para se produzir o código, e a partir dela foi possível inspecionar uma tela (Figura 15) e obter as seguintes informações e elementos que compõem a mesma:

1. Medidas de espaçamentos e parametrização entre elementos;
2. Valores de cores;
3. Exportar imagens e ícones em diferentes formatos;
4. Nome e valores de propriedades de um elemento da tela na linguagem de estilo em cascata para navegadores de páginas de internet.

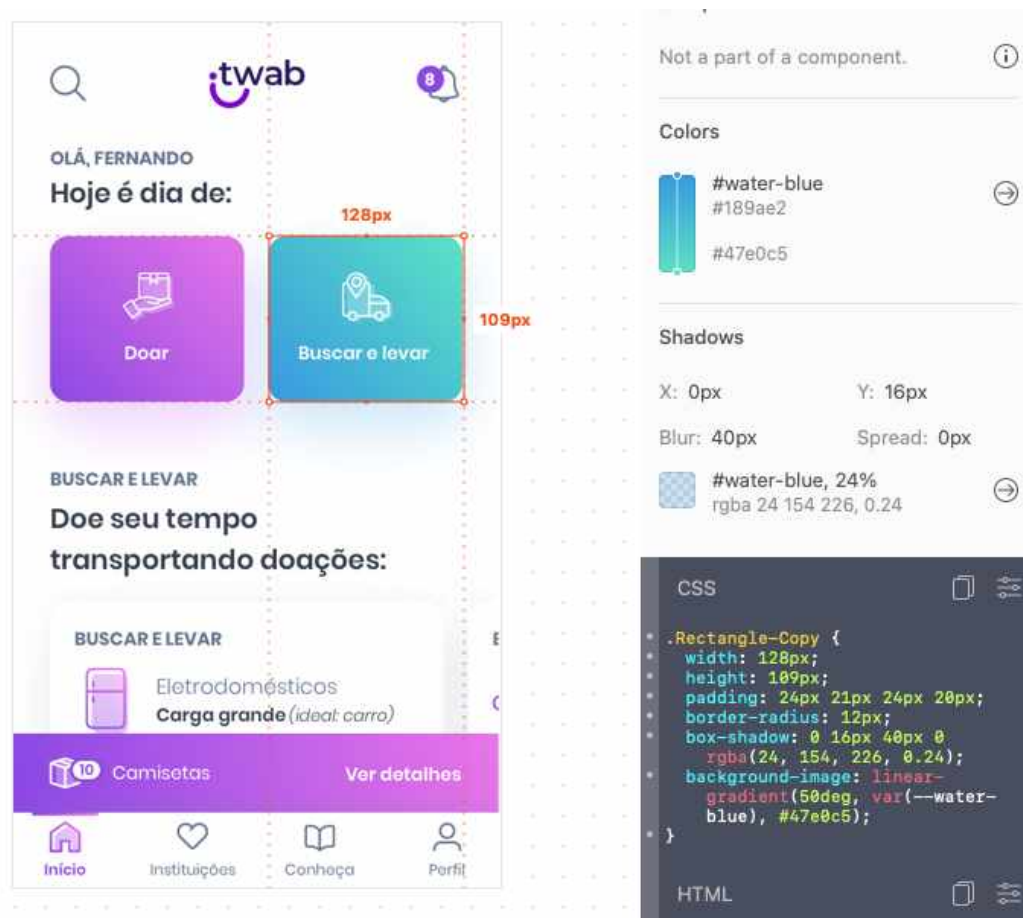


Figura 15 – Medidas e valores de estilo de um componente botão de uma tela na ferramenta Zeplin.

Houve ainda a necessidade de se utilizar outra ferramenta de criação de *design* de interface, denominada Figma, com o objetivo de obter uma visualização aprimorada do fluxo entre as diferentes telas do aplicativo e suprir a necessidade de confeccionar novos ícones e imagens de forma mais rápida a partir de existentes. A ferramenta Zeplin permite organizar um conjunto de telas em seções e nomeá-las, mas a ferramenta Figma permite uma organização menos engessada deixando livre além de nomeação e criação de seções, a utilização de outros elementos visuais em conjunto com as telas.

A Figura 16 representa uma comparação em relação à organização de conteúdo entre as ferramentas de modelagem de *design* de interface usadas para guiar o desenvolvimento das telas deste trabalho. A parte superior da figura exhibe como são dispostas as telas dentro de uma seção na ferramenta Zeplin e na parte inferior tem-se o mesmo na ferramenta Figma. Nota-se na parte inferior que nos retângulos que representam as telas estão organizados em conjunto com outros elementos gráficos além das próprias telas, textos e desenhos gráficos no formato de setas para a indicar direção de fluxo.

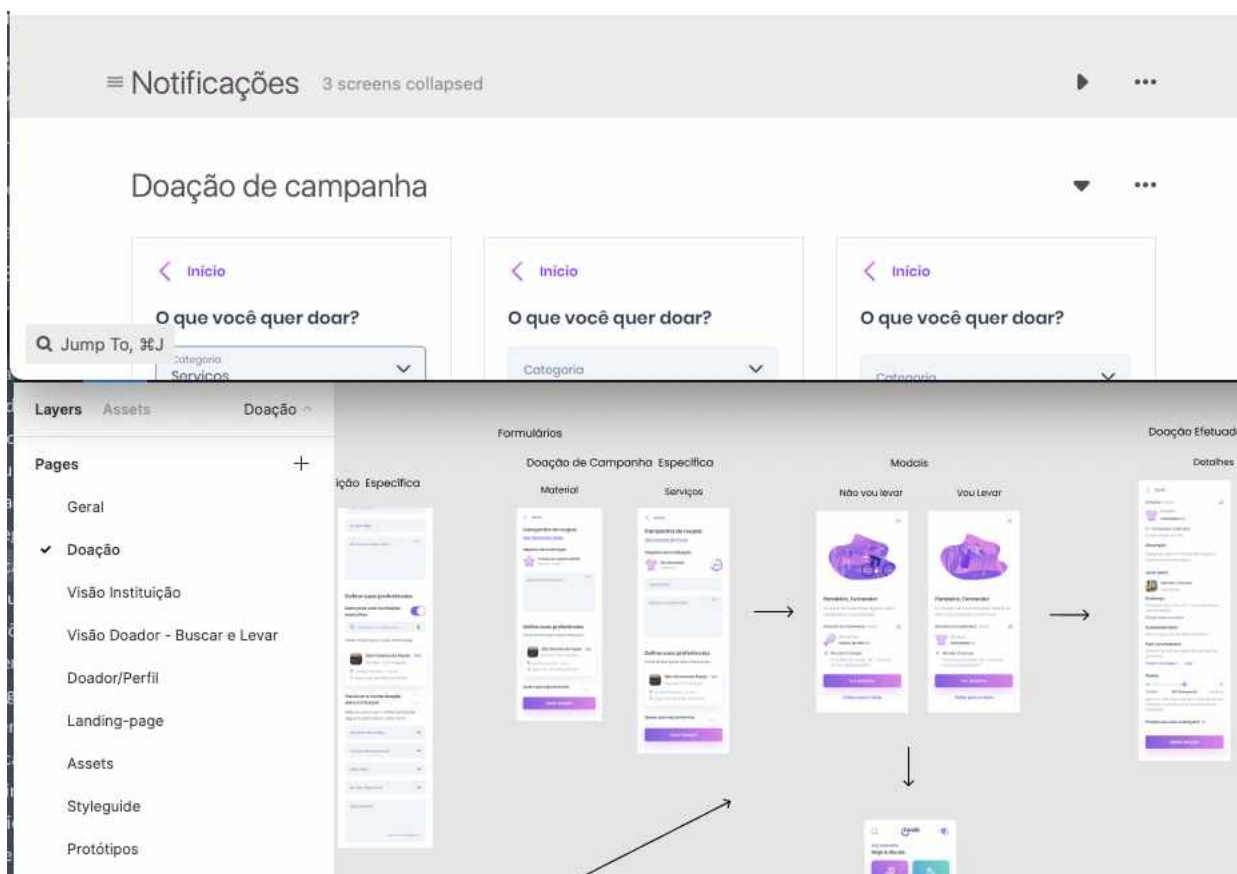


Figura 16 – Seções de telas na ferramenta Zeplin e Figma.

Toda o processo de criação do *design* e organização das telas foi essencial para a parte de codificação dos *softwares* deste trabalho, pois foi realizada de forma a garantir que todas as decisões visuais já estivessem acertadas cabendo assim ao programador implementar o mais próximo possível do que foi planejado.

5.3 Implementação *front-end* & *back-end*

5.3.1 Aplicativo

O principal objetivo deste projeto é criar um aplicativo para dispositivos móveis que tenha acesso às funcionalidades internas do *hardware* de um dispositivo móvel (celular ou *tablet*), como serviços de localização pela antena GPS, captura de fotos, captura de áudio e recebimento de notificações. Atualmente existem dois sistemas operacionais para dispositivos móveis dominantes no mercado, com o iOS da Apple estando com 27% e o Android da Google com 72% de fatia do mercado (STATCOUNTER, 2021). Tendo estas estatísticas em vista, a maneira evidente de prosseguir é de produzir o mesmo aplicativo para a linguagem Swift ou Objective-C do iOS e Kotlin ou Java do Android.

Em termos mais técnicos, produzir dois códigos fontes completamente diferentes para resultar em um mesmo *software*, pelo menos no quesito interface gráfica. Essa primeira opção é a forma tradicional de se desenvolver para dispositivos móveis que possuem estes dois sistemas operacionais, e esta opção oferece a vantagem para o desenvolvedor de ter a total liberdade para criar telas, componentes e funcionalidades, além explorar o máximo de possibilidades do seu aplicativo poderá fazer dentro dos respectivos sistemas operacionais, com destaque por exemplo, de obter o melhor desempenho possível e controle de consumo de energia. Mas essa opção possui dois desafios, que é ter domínio de duas linguagens de programação completamente diferentes e a manutenção de dois códigos fontes. Se uma nova funcionalidade ou alteração precisar ser feita, ela terá que ser implementada de maneiras diferentes para cada uma das duas linguagens.

A segunda forma e a escolhida para desenvolver o aplicativo deste projeto foi utilizar uma biblioteca de desenvolvimento para dispositivos móveis *cross-plataform*, chamada *React Native*. Existem outras bibliotecas e *frameworks* conceituados no mercado com a mesma finalidade que é produzir *softwares* para diferentes plataformas com apenas um código fonte, por exemplo, Flutter, Ionic, Corona (Solar2D) e Xamarin. Estes *frameworks* fornecem ao desenvolvedor a manutenção de apenas um código fonte para ambos os sistemas operacionais citados. Dos *frameworks* citados anteriormente, o Flutter, é o que mais se aproxima de permitir o controle para o desenvolvedor de seu aplicativo junto ao *hardware* do dispositivo móvel para obter os benefícios de desempenho e gasto energético. No entanto, a escolha pelo *React Native* se deu primeiramente pela facilidade de aprendizagem da biblioteca, rapidez de desenvolvimento, pela sintaxe de escrita do código fonte e constante suporte e evolução da biblioteca. Outra vantagem é que a sintaxe do *React* é mais simples de aprender e visualmente mais organizada que a linguagem Dart do *framework* Flutter, além de ser semelhante a de aplicações para páginas de internet. Um aplicativo feito em *React Native* permite que o código fonte seja escrito usando uma linguagem chamada JSX que é um JavaScript com “poderes”, possibilitando escrever o código JavaScript com elementos na notação XML em um mesmo arquivo. Considerando-se a possibilidade de explorar o *hardware* do dispositivo, um aplicativo feito com *React Native* possui dois lados, um lado nativo e o lado JavaScript, ambos se comunicam através de um recurso chamado *React Native Bridge*, em que o código escrito em JavaScript é executado em uma máquina virtual controlando os componentes nativos e renderizando no *hardware* do dispositivo ([REACTNATIVE, 2020](#)). Este recurso permite criar componentes específicos para cada uma das duas plataformas.

5.3.1.1 Padrões de projeto

O *React Native* por ser uma biblioteca declarativa para se criar componentes e não um *framework*, permite maior liberdade ao desenvolvedor de poder estruturar sua aplicação, como escolher a arquitetura e fluxos de gerenciamento de estado. Ele não dita

regras e sim indica de princípios de *design* e padrões de desenvolvimento que se aplicarão adequadamente ao *software* construído com ele.

O aplicativo foi implementado com base no padrão de arquitetura de *software* MVC na camada *front-end*. Neste projeto com *React*, o padrão MVC é aplicado de forma não tradicional (figura 17). Assim, ao invés de serem definidas as três camadas *Model*, *View* e *Controller* separadas, têm-se as camadas *View* e *Controller* unidas em um único componente, ficando a cargo do programador a tentativa de desacoplar essas duas camadas via composição de componentes (Figura 18) e Componentes de Ordem Superior (HOC, do inglês *High Order Components*).

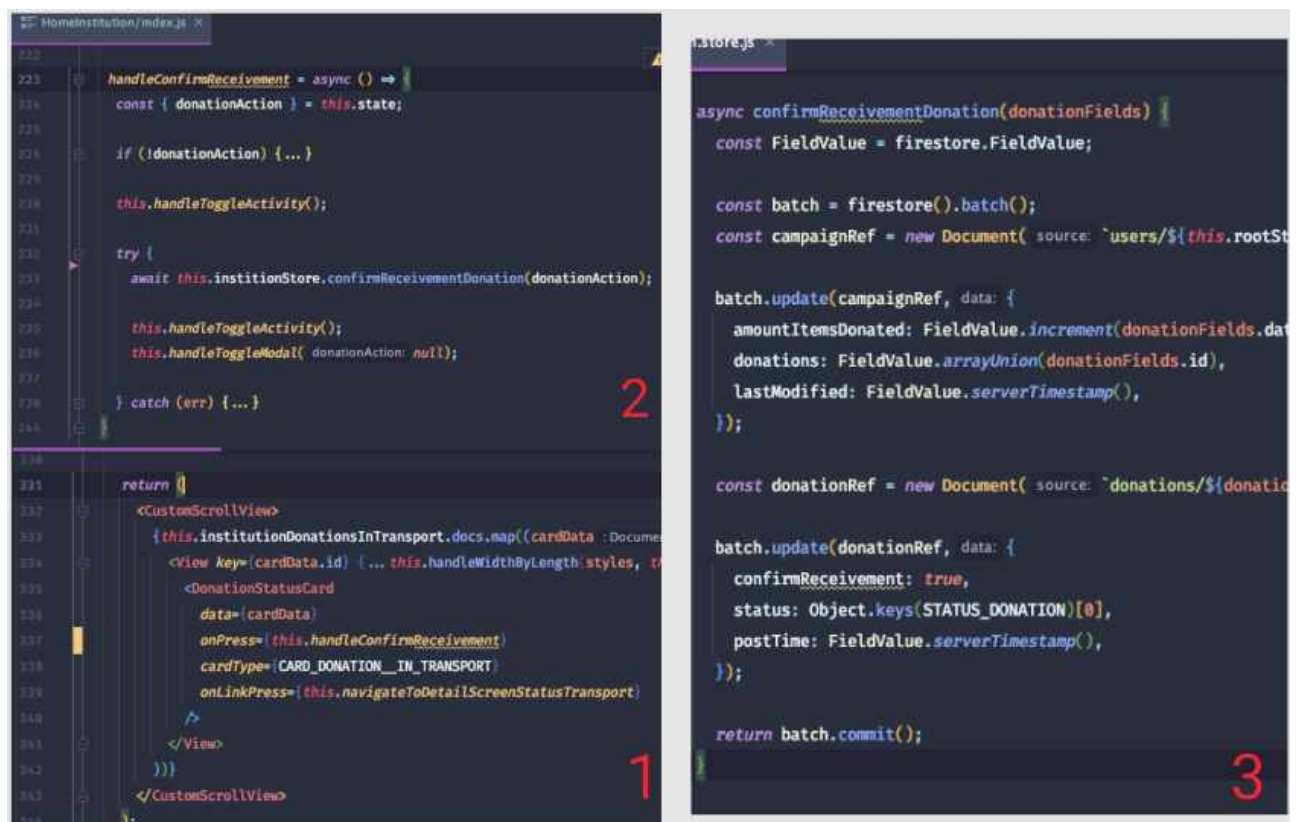


Figura 17 – Partes do código fonte do processo de confirmar o recebimento de uma doação pela instituição.

A Figura 17 contém trechos do código fonte do aplicativo que realizam o procedimento em que o usuário de tipo instituição confirma o recebimento de uma doação. Estes trechos são utilizados para mostrar como o padrão MVC está codificado. A parte da figura que contém os algarismos 1 e 2 representam respectivamente as camadas *View* e *Controller*, que estão localizadas no mesmo arquivo. Na sequência, o algarismo 3 representa a camada *Model*. O usuário, ao interagir com a *View*, dispara um evento (confirmar o recebimento de uma doação) e a camada *Controller* recebe este evento com os dados atrelados

àquela *View*, repassando-os à camada *Model* que realiza a formatação e persistência em base de dados, retornando sucesso ou erro.

Como já mencionado anteriormente, construir um *software* em componentes no *React* é a sua característica essencial. A composição de componentes dividindo o *software* em componentes menores para se ter partes independentes e reutilizáveis é o seu principal princípio de *design*. A Figura 18 retrata a composição de alguns componentes que compõem a tela de perfil. O lado esquerdo da figura mostra a hierarquia de pastas e arquivos que abrigam os componentes que fazem parte da tela de perfil do usuário. O lado direito da figura mostra como é utilizado no código dois destes componentes no componente pai que é a tela de perfil. Se por acaso o componente **AboutMeInfo** ou **PresentationUser** fosse utilizado dentro de outro componente, ele seria declarado de maneira semelhante ao retratado na figura.

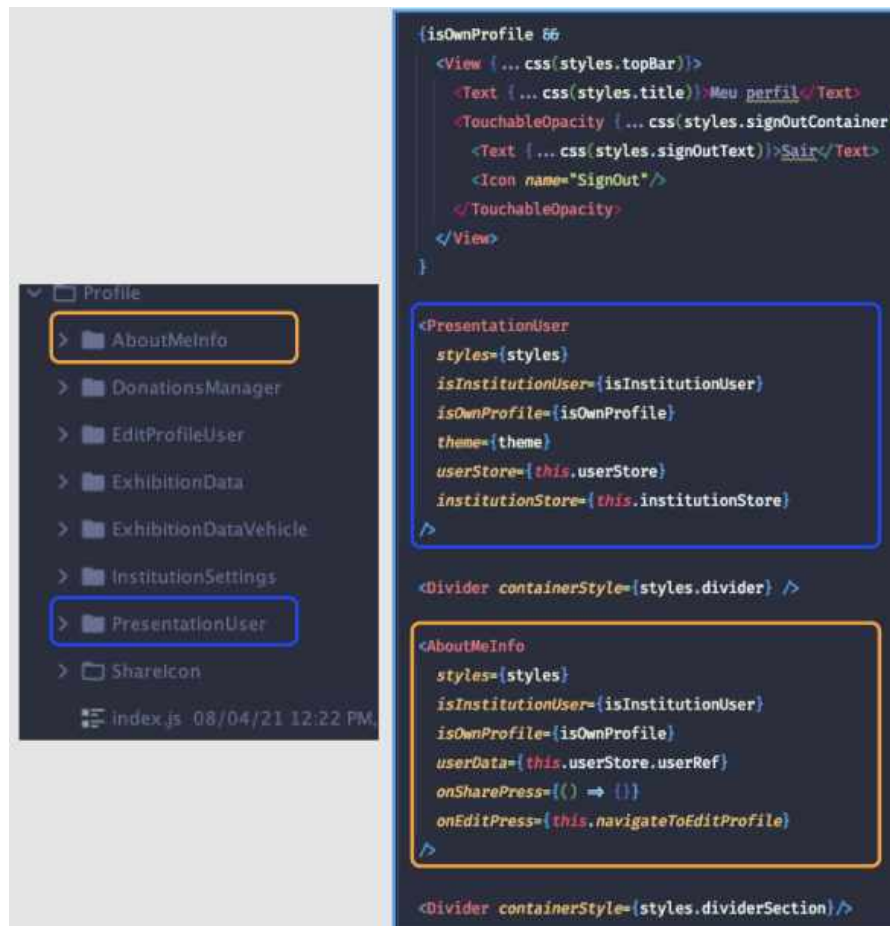


Figura 18 – Pastas de componentes da tela de perfil no sistema de arquivos ao lado do trecho do código fonte da utilização de dois deles.

Componentes de Ordem Superior são técnicas avançadas derivadas do padrão natural da composição de componentes do *React*. Em uma aplicação com MVC os componentes com essa técnica podem realizar o papel de *Controller* de outros componentes

que são responsáveis apenas por renderizar um conteúdo, ou seja, o papel da *View*. O padrão MVC está aplicado neste trabalho em conjunto com o padrão *Container Component* e o *Presentational Component* que formam o *Component Pattern* que é a principal característica do *React*.

A Figura 19 contém trechos principais de código de dois tipos de componentes do aplicativo. A parte superior é um componente codificado com o padrão *Container Component* que contém seu próprio controle de estado (1) e a responsabilidade de repassar o seu atual valor armazenado em seu estado ao componente pai a qual está acoplado (2). A parte inferior da figura é um componente codificado com o padrão *Presentational Component*, que não possui nenhum estado próprio, tendo a responsabilidade de receber propriedades do componente pai ao qual está acoplado e renderizá-las.



```
function SlideUploadPhotos({styles, containerStyle, onSelectImages, imagesURLs, clearImages,
const [images, setImages] = React.useState({ initialState: [] }); 1

const onSelectPhoto = ({name, path, creationDate}) => {
  const newData = [ ... images, {creationDate, name, path}];
  setImages(newData);
  onSelectImages(newData); 2
};

const handleContent = () => {...}

return (
  <CustomScrollView scrollViewStyle={{paddingLeft: noPaddingLeft ? 0 : 14}} containerStyle
    {handleContent()}
  </CustomScrollView>
);
}

function UserInfoData({styles, userName, city, avatarUrl}) {
  return (
    <View { ... css(styles.directionLine)}>
      <Avatar avatarUri={avatarUrl}/>
      <View { ... css(styles.userDonationInfoWrapper)}>
        <Text { ... css(styles.userDonationInfo)}>{userName}</Text>
        <Text { ... css(styles.dateDonation)}>{city}</Text>
      </View>
    </View>
  );
}
```

Figura 19 – Trechos de código fonte de dois componentes: um com gerenciamento de estado próprio e o outro sem.

5.3.1.2 Gerenciamento de estado

A forma mais simples e utilizada de se repassar dados manualmente entre os componentes da uma aplicação com *React* é via propriedades (comumente usa-se o termo *props*) desses componentes. Na medida em que a aplicação vai tendo níveis mais profundos de acoplamento de componentes dentro de outros componentes, que podem ter seus próprios ciclos de vida e seus próprios gerenciamentos de estados, a complexidade da aplicação aumenta. A partir do momento em que os componentes (*View* e *Controller*) já estão acoplados e recebendo dados de serviços (*Model*) além de interações do usuário, existe o risco de renderizações desnecessárias e *loops* infinitos entre os componentes caso o trajeto dessas informações não esteja bem planejado e em harmonia.

Encontrar *bugs* ingerenciados dentro de um *software* e resolvê-los é uma tarefa dispendiosa. Assim, à medida que o acoplamento entre componentes vai ficando complexo é necessário ter planejado quais valores de estados da aplicação irão ser utilizados em mais de um lugar, quais *Controllers* e *Views* irão se comunicar com quais *Models*. Para resolver este problema de planejar o tráfego de valores de estados entre componentes da aplicação e garantir uma maior escalabilidade de funcionalidades no futuras na aplicação, foi utilizada uma biblioteca para gerenciamento de estado, chamada MobX.

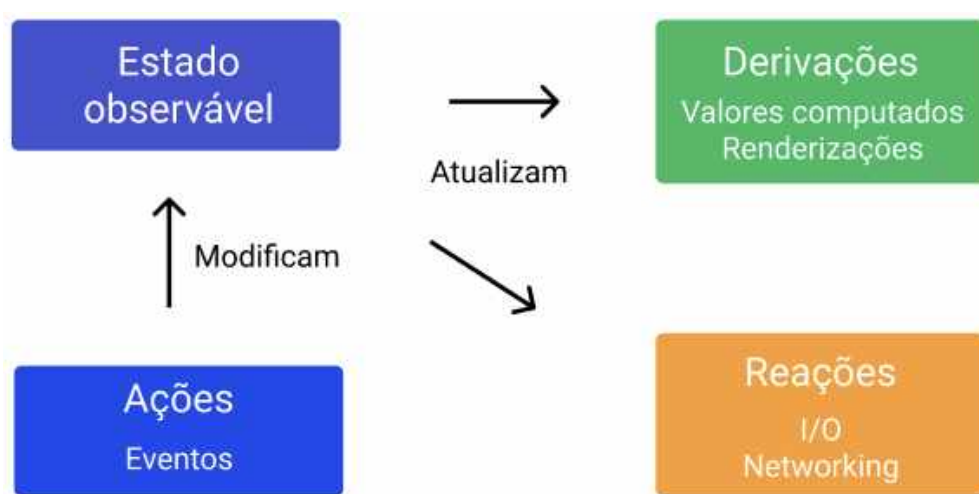


Figura 20 – Diagrama conceitual de direção de fluxo de dados da biblioteca MobX.
Fonte: (WESTSTRATE, 2021b)

A biblioteca MobX é inspirada nos princípios de programação reativa que provê uma solução de gerenciamento de estado simples e escalável que conceitualmente trata uma aplicação de *software* como uma planilha (Figura 20):

1. Primeiramente, há um “estado da aplicação”. Grafos de objetos, matrizes, valores primitivos, referências formam o modelo da aplicação. Esses valores são as “células de dados” da aplicação;

2. Em segundo lugar, há “derivações”. Basicamente, qualquer valor pode ser calculado automaticamente a partir do estado da sua aplicação. Essas derivações, ou valores calculados, podem variar de valores simples, como o número de tarefas inacabadas, a coisas complexas, como uma representação HTML visual de tarefas. Em termos de planilha: estas são as fórmulas e gráficos do seu aplicativo;
3. Em terceiro lugar, há “reações”, que são muito semelhantes às derivações. A principal diferença é que essas funções não produzem um valor. Em vez disso, elas são executadas automaticamente para rodar alguma tarefa. Geralmente isto é relacionado a operações de entrada e saída (I/O). Elas garantem que os elementos da interface gráfica sejam atualizados ou que as solicitações de rede sejam feitas automaticamente no momento certo;
4. Finalmente há “ações”. Ações são todas as coisas que alteram o estado. O MobX garante que todas as alterações no estado do aplicativo causadas por ações sejam processadas automaticamente por todas as derivações e reações de forma síncrona e sem falhas.

A biblioteca MobX não dita como deve ser a estrutura de gerenciamento de valores do estado de uma aplicação, ela permite que se gerencie o estado do aplicativo fora de qualquer estrutura de interface do usuário ([WESTSTRATE, 2021b](#)). Isso torna o código fonte dissociado, portátil e, acima de tudo, facilmente testável. Mas assim como o *React*, o MobX recomenda a utilização de engenharia adequadas, como encapsular lógica em métodos, organizá-los em lojas ou controladores (*controllers*), *views-models* e etc. Muitos padrões arquitetônicos diferentes podem ser aplicados, e alguns são discutidos detalhadamente na documentação oficial da biblioteca. Neste contexto, pode-se destacar que a biblioteca funciona adequadamente com o padrão MVC.

Considerando-se as recomendações de melhores práticas do MobX ([WESTSTRATE, 2021a](#)) para a construção de projetos escaláveis futuramente, foi implementado o conceito de lojas. Neste projeto, a camada *Model* funciona como loja para abrigar o armazenamento de valores derivados do estado geral do *software*. A principal responsabilidade das lojas é possuir a lógica e o estado dos valores que serão utilizados em mais de um lugar na aplicação. Como o aplicativo desenvolvido neste projeto não contém muitos fluxos muito complexos, foram implementadas duas lojas (Figura 21) com o papel de armazenar e manipular dados de domínio (banco de dados) e alguns dados de interface de usuário.

```
class RootStore {
  isHydrated = true;

  constructor() {
    this.userStore = new UserStore(this);
    this.institutionStore = new InstitutionStore(this);
  }

  init(uid, isInstitutionUser) {
    this.userStore.setUid(uid);
    this.userStore.setInstitutionUser(isInstitutionUser);
  }
}

export default new RootStore();
```

Figura 21 – Trecho principal do código fonte da loja raiz de gerenciamento de dados.

A Figura 21 contém o trecho de código que define a loja raiz que instancia todas as outras lojas e compartilha as referências das instâncias entre elas. As vantagens de implementá-las com esse padrão é que, é simples de configurar e facilita os testes unitários, já que é preciso instanciar uma única loja raiz. Na figura, a primeira loja definida, chamada de **UserStore**, manipula todas as informações do usuário autenticado no aplicativo e instancia objetos de domínio relacionados às operações de dados que serão realizadas pelo usuário doador como: visualizar instituições, campanhas de doações e realizar/gerenciar doações. A segunda loja, chamada **InstitutionStore**, manipula todas as informações relacionadas às operações que o usuário instituição poderá realizar, como criar campanhas de doações e gerenciar as próprias doações. As duas lojas também implementam seus métodos necessários para persistir informações no banco de dados, ou seja o *back-end*.

5.3.2 Banco de dados & Serviços

A camada *back-end* de uma aplicação de *software* tradicionalmente é composta pela lógica da regra de negócio da aplicação e persistência de informações em um banco de dados, que reside em um servidor para suprir a necessidade da camada *front-end* de consumir dados, ou seja, o modelo cliente-servidor. Neste trabalho para a camada *back-end* utilizou-se uma plataforma BaaS chamada Firebase. É relevante apontar a diferença entre o modo tradicional de desenvolver uma camada *back-end* e de se utilizar uma plataforma que disponibiliza serviços *back-end*. No modelo cliente-servidor a camada

front-end apenas invoca pontos de comunicação de uma API exposta pelo *back-end*, e o *back-end* fica responsável por processar os dados aplicando a lógica da regra de negócio sobre os SDK's como persistir dados na nuvem ou em um banco de dados local. Contudo, a plataforma Firebase disponibiliza SDK's de seus serviços para serem utilizados a partir da lógica da regra de negócio no código fonte do cliente. Outra forma de acessar os serviços da plataforma é através de uma interface gráfica na forma de páginas *web* chamada Firebase Console.

A Figura 22 contém um diagrama comparativo do fluxo de dados entre o modelo cliente-servidor tradicional e a plataforma Firebase que provê serviços de *back-end*.

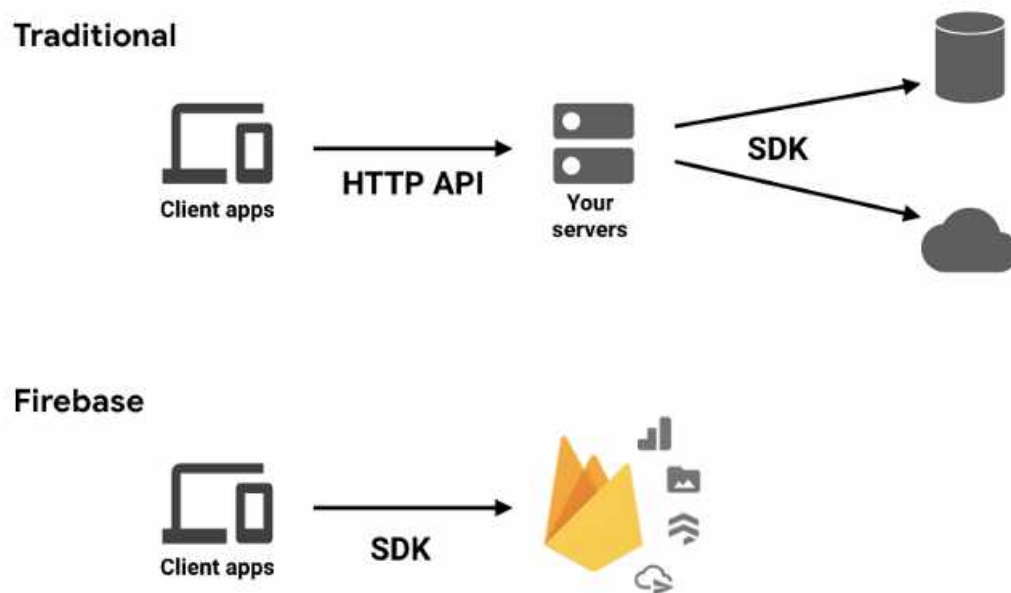


Figura 22 – Diagrama comparativo do fluxo de dados cliente-servidor tradicional e da plataforma Firebase. Fonte: (STEVENSON, 2018)

5.3.2.1 Cloud Firestore

A plataforma Firebase disponibiliza dois serviços de banco de dados: o Real Time Database e o Cloud Firestore. Ambos são banco dados NoSQL, orientados a documentos e com atualização de dados em tempo real. O banco de dados escolhido para persistir os dados dos *softwares* deste trabalho foi o Cloud Firestore. A motivação para escolher o Cloud Firestore ao invés do Real Time Database se deve ao fato do Cloud Firestore ter sido desenvolvido com foco para aplicações *web* e dispositivos móveis. Ele também possui um modelo de dados mais intuitivo, possibilidade de criar consultas de recuperação de dados mais avançadas e rápidas, além de melhor escalabilidade que o Real Time Database (FIREBASE, 2021b).

A Figura 23 contém o diagrama do modelo e estrutura de dados na forma de documentos e coleções obtido inicialmente através do modelo de entidade e relacionamento do apêndice B. O Cloud Firestore é um banco NoSQL orientado à documentos, e também não dita regras de como os dados devem ser estruturados. Ele possui um modelo de dados com campos em documentos com tipos primitivos (Número, String, Booleano) e customizados (Geopoint, Timestamp, Reference, Blob, Map e Array) dentro de coleções com referências. O Cloud Firestore em sua documentação oficial ([FIREBASE, 2021a](#)) possui a recomendação de três formas de estruturar dados: Dados Aninhados em Documentos, Subcoleções e Coleções Nível Raiz. A forma de estruturar os dados dos *softwares* deste trabalho foi uma combinação de Subcoleções a partir de Coleções Nível Raiz. A vantagem de se estruturar uma parte dos dados em Subcoleções é que conforme os documentos crescem o tamanho do documento pai não muda, mas há algumas limitações em construir consultas de Subcoleções. A vantagem de se estruturar uma parte dos dados em Coleções de Nível Raiz é a versatilidade de se escrever consultas avançadas e são boas para representar relações muitos para muitos.

A partir do diagrama de entidade e relacionamento (Apêndice B) criaram-se três coleções nível raiz chamadas de **fetch_and_deliver**, **users** e **donations**, além de duas subcoleções dos documentos da coleção chamadas **campaigns** e **donations_managers**. A coleção **users** armazena os dados do usuário, a coleção **donations** armazena dados de doações de itens e a **fetch_and_deliver** armazenara dados de uma doação na modalidade buscar e levar. Entretanto, na Figura 23 que contém o diagrama do modelo e a estrutura de dados, percebe-se que não há a coleção de nível raiz chamada **fetch_and_deliver**. Esta coleção teve seus documentos removidos e seus campos movidos para dentro da coleção **donations**.

A plataforma Firebase possui uma lista de fatores que geram cobranças e fazem atingir limites para o uso de seus recursos, que foi a motivação para remodelagem realizada, levando em consideração a análise dos seguintes fatores: complexidade de consultas $\times$ preço $\times$ número de operações. Com a remodelagem, diminuiu-se a quantidade de documentos a serem recuperados, sendo agora um subconjunto de dados em um documento que cabe ao código saber se este subconjunto de campos deve fazer parte ou não do documento.

Ao começar a implementar a integração com o banco de dados notou-se uma quantidade expressiva de operações de leitura de documentos da coleção **donations**, o que já era esperado. Porém, dependendo do fluxo de uma doação de itens, ela poderia necessitar de um transporte e esses dados de transporte iriam para um documento em uma segunda coleção. Esta estratégia acarretaria num aumento expressivo da quantidade de operações de leitura, isso sem levar em consideração outras operações que ocorrem com menos frequência que podem ser inserção, atualização e remoção de dados.

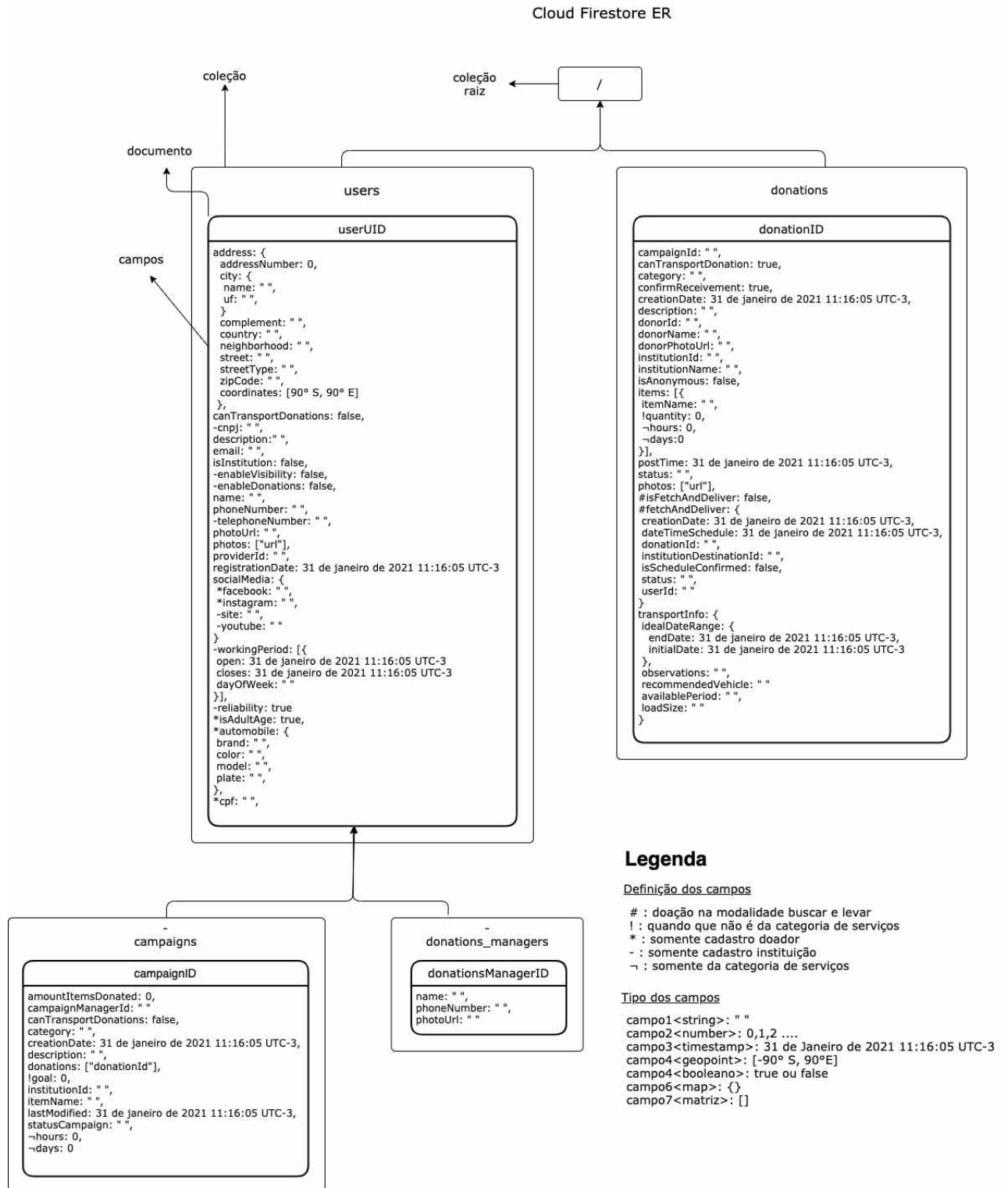


Figura 23 – Diagrama do modelo e estrutura de dados proposto na forma de documentos.

5.3.2.2 Firebase Authentication

Para manter os usuários autenticados em uma sessão local no aplicativo foi utilizado o serviço Firebase Authentication. Este é um serviço independente sem relação direta

com os bancos de dados de dentro da plataforma, com seus próprios meios de prover gerenciamento de usuários através de seus serviços além de uma seção no Firebase Console. O Firebase Authentication fornece serviços para cadastrar, atualizar, re-autenticar, recuperação de senha e remover usuários dentro do seu SDK, conhecido como *easy-to-use*, do inglês, fácil-de-usar. Este serviço suporta autenticação usando *e-mail* com senha, número de telefone além provedores de identidade de autenticações muito utilizados como Google, Facebook, Apple, Twitter e entre outros. A Figura 24 contém capturas de telas dos respectivos provedores de identidade para autenticação no aplicativo, com *e-mail* e senha (1), Facebook (2), Google (3) e Apple ID (4).

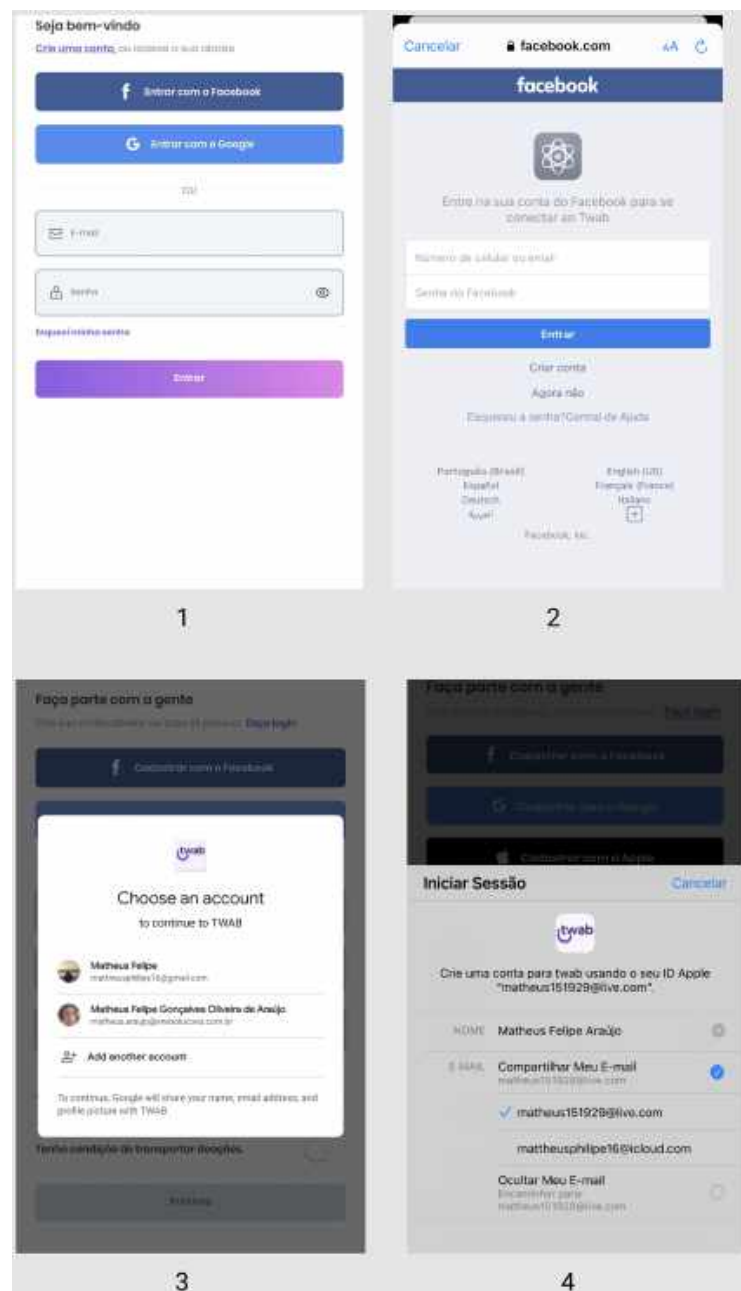


Figura 24 – Captura de telas do métodos de autenticação no aplicativo.

O Firebase Authentication contém seu próprio *back-end* que verifica as credenciais de usuário obtidas pelos métodos de provedores de identidades citados anteriormente e retorna uma resposta ao cliente, que é o *software* que está usando o SDK. Ele também é utilizado por outros serviços como o Cloud Firestore, Real Time Database e Cloud Storage para identificar se um usuário está autenticado para poder executar operações.

5.3.2.3 Cloud Storage

Os bancos de dados SQL e NoSQL podem armazenar diferentes tipos de dados simples como um campo com número inteiro a mais complexos dados binários e data e hora. Os diferentes tipos de arquivos são armazenados diretamente no sistema de arquivos de um servidor ou em um serviço de entrega de conteúdo pela rede que é largamente utilizada para essa finalidade, conhecida pela sigla CDN (*Content Delivery Network*). Um CDN detém a responsabilidade de armazenar arquivos e disponibilizá-los através de um meio de acesso fácil como uma URL. O Cloud Storage, assim como o Amazon S3, é um serviço da plataforma Firebase que a finalidade de armazenar arquivos e prover uma forma de acessá-los com níveis de segurança, ficando a organização e estruturação a cargo do desenvolvedor.

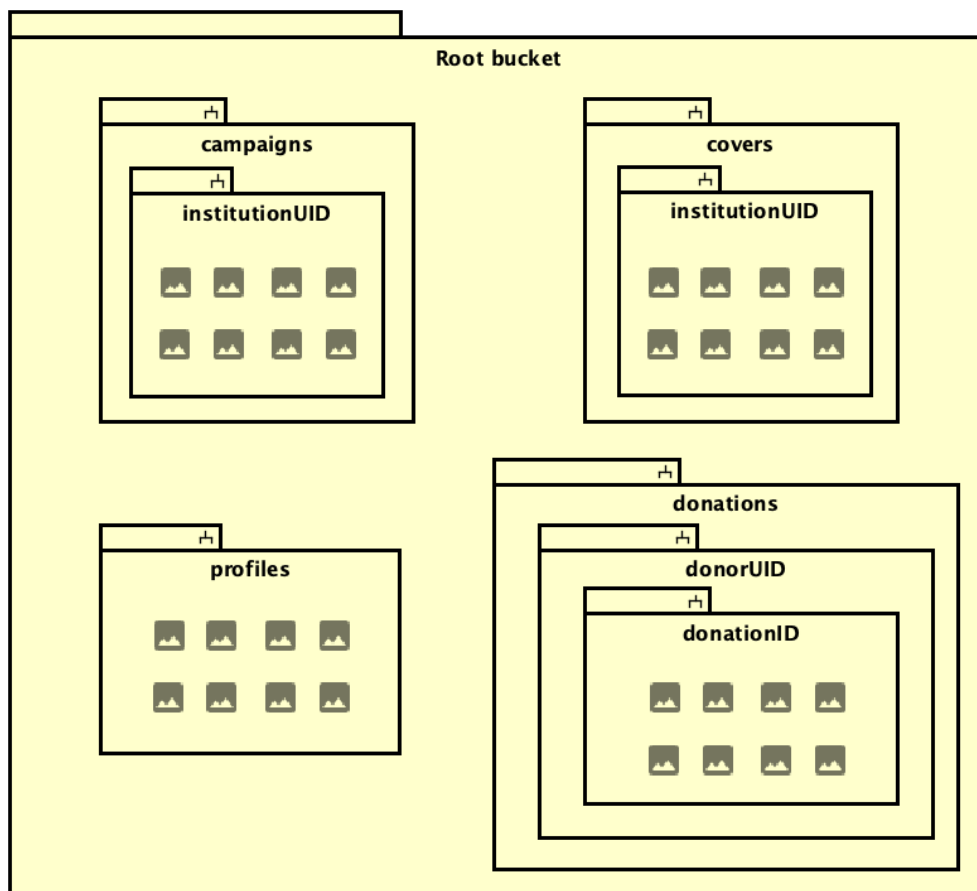


Figura 25 – Diagrama da hierarquia de pastas e arquivos do Cloud Storage.

A Figura 25 contém o diagrama de como estão estruturados os arquivos que atualmente são carregados para o Cloud Storage. Assim como no Amazon S3, este serviço trabalha com o conceito de buckets (do inglês, baldes) e dentro deles é que ficam as pastas e arquivos. Como o único tipo de arquivo que é carregado dos *softwares* deste projeto para este serviço são imagens, foi criado somente um *bucket* que contém pastas para armazenar fotos de campanhas, doações, fotos de perfil do usuário e fotos de capa do perfil do usuário.

5.3.3 Landing-page

A *landing-page* tem por objetivo engajar os visitantes a serem doadores voluntários e permitir a solicitação de cadastro das instituições interessadas em se envolver com a iniciativa desta aplicação.

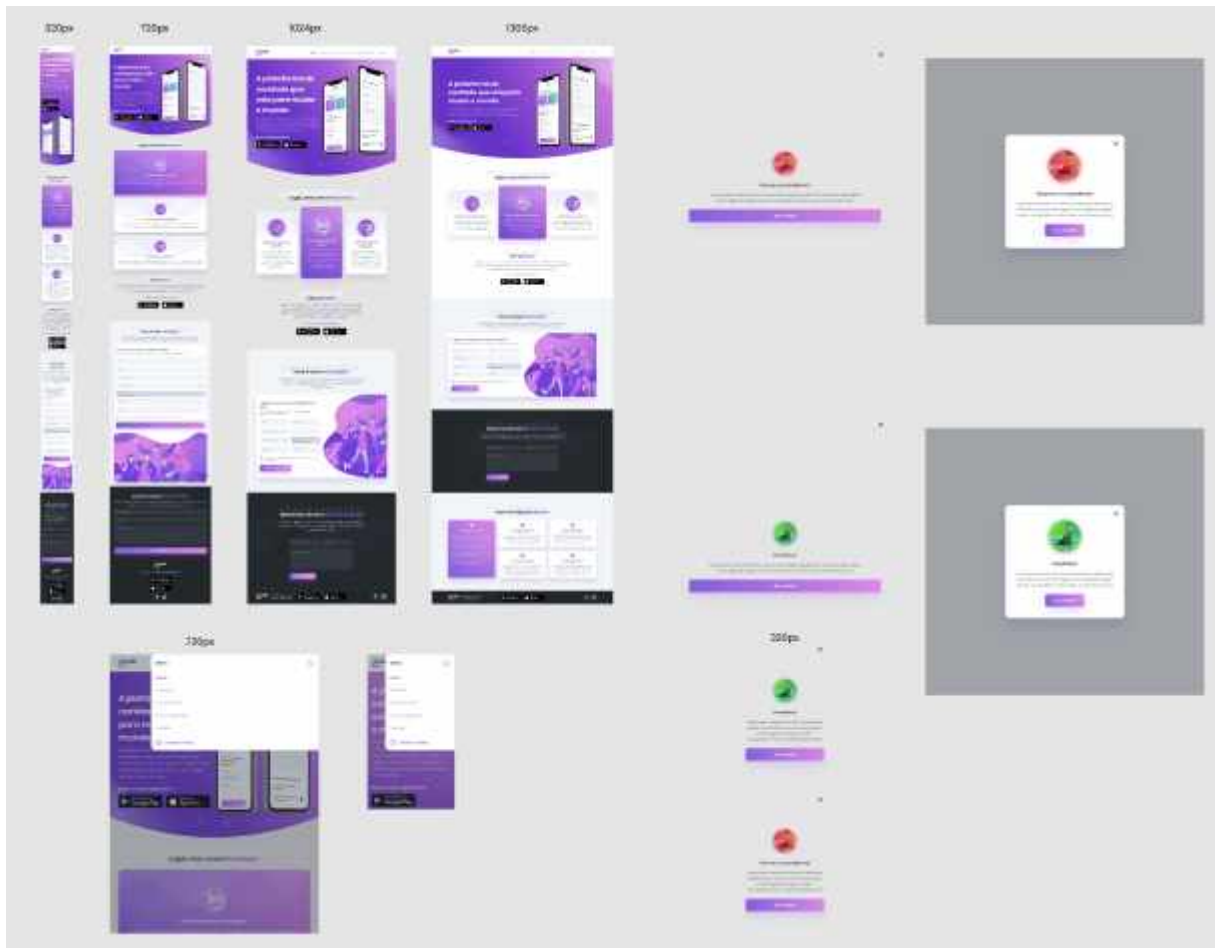


Figura 26 – Versões do *layout* da *landing-page* para diferentes resoluções de tela.

Foram desenvolvidas duas páginas *web*, conforme a Figura 26, com a API declarativa do *React* para a manipulação da árvore de elementos do conteúdo HTML, chamada

React Virtual DOM. O DOM (*Document Object Model*) é uma representação estruturada de todos os elementos da página HTML e é utilizado para facilitar as modificações que podem ocorrer nesses elementos. O *React* permite ter um DOM virtualizado para ser manipulado extensivamente dentro do código JavaScript (JSX). A estilização desses elementos é feita com a linguagem de estilos em cascata chamada CSS.

Para que a *landing-page* possa realizar a solicitação de cadastro, estão integrados à ela os seguintes serviços da plataforma Firebase: Cloud Storage, Cloud Firestore, Firebase Authentication e ela está hospedada no serviço Firebase Hosting. Este último serviço mencionado é um serviço global de hospedagem de páginas *web*, que funciona de forma adequada para entregar conteúdo estático como HTML, CSS e JavaScript. As páginas *web* são hospedadas com navegação segura através de um certificado de segurança SSL.

5.4 Controle de versionamento de código

O controle de versionamento de códigos fonte é necessário para gerenciar versões diferentes de um mesmo projeto. Os *softwares* deste projeto seguiram o padrão de controle de versionamento chamado Git Flow ([DRIESSEN, 2010](#)). Este é um padrão que resumidamente induz o desenvolvedor a ir compondo seu *software* através de funcionalidades distribuídas em *commits* dentro de *branches*.

Conforme mostra a Figura 27, ao construir uma nova funcionalidade para um *software*, o desenvolvedor deve criar uma *branch* com um nome curto da funcionalidade (3) que se está pretendendo implementar e realizar a operação de um ou mais *commits* (4). Quando a funcionalidade estiver finalizada deve-se realizar a junção desses *commits* na *branch* mestre, que é a operação que leva os *commits* daquela *branch* (que representa a funcionalidade para a *branch* principal de desenvolvimento). *Commits* são compostos por arquivos alterados e mensagens descrevendo-os. No Git Flow é recomendado ter uma *branch* de código fonte para homologação de funcionalidades (*develop*) (2) e uma principal para produção (*master*). A ferramenta utilizada para realizar o controle de versionamento do código fonte é o GitHub da empresa Github, Inc.



Figura 27 – Captura das tela de uma requisição de junção de *commit* da *branch* da funcionalidade para a de homologação.

6 Resultados Obtidos

Esse capítulo apresenta as principais telas e funcionalidades dos *softwares* desenvolvidos neste trabalho: aplicativo para dispositivos móveis para os sistemas operacionais Android e iOS e uma *landing-page* para navegadores de páginas de internet. As imagens das figuras das subseções seguintes podem ter algumas variações de tamanho e resolução, pois foram obtidas de dispositivos e sistemas operacionais diferentes.

Uma versão de teste do aplicativo está disponível para *download* em dispositivos móveis com sistema operacional Android em: <<https://appdistribution.firebase.dev/i/7a0ed6ddc278f64e>>. A *landing-page* construída neste trabalho encontra-se hospedada no endereço: <<https://twab-hom.web.app>>.

6.1 Conceitos do aplicativo

Nesta seção são explicados os conceitos principais para a compreensão do funcionamento dos *softwares* deste trabalho.

6.1.1 Instituição

O primeiro tipo de usuário participante do aplicativo é o usuário instituição. Para se ter um cadastro aprovado para usar o aplicativo, as instituições precisam ser socialmente reconhecidas como entidades filantrópicas/beneficentes. O usuário instituição pode:

- Gerenciar seu próprio cadastro: solicitar a criação de sua conta via formulário na *landing-page*, editar dados e excluir conta;
- Cadastrar e excluir gerenciadores de doação;
- Gerenciar campanhas de doação: criar, consultar detalhes, editar e remover;
- Gerenciar doações: consultar detalhes, confirmar recebimento, buscar uma doação e solicitar uma doação;

6.1.2 Doador voluntário

O segundo tipo de usuário participante do aplicativo é o usuário doador voluntário. Os doadores voluntários do aplicativo são pessoas físicas que querem ajudar instituições na doação de itens e serviços. O usuário doador voluntário pode:

- Gerenciar seu próprio cadastro: solicitar via aplicativo, editar dados e excluir conta;

- Gerenciar doações: criar, consultar detalhes, editar e cancelar;
- Consultar campanhas de doação;
- Consultar instituições.

6.1.3 Campanha

Uma campanha é uma ação da instituição de solicitar um conjunto de recursos materiais ou serviços relevantes para caridade e filantropia. Como por exemplo, criar uma campanha de arrecadação de agasalhos ou uma campanha para obter voluntários para ajudar a servir alimentos no local da instituição.

6.1.4 Doação

Uma doação é uma ação dos doadores voluntários de contribuir com itens materiais ou serviços para instituições. Uma doação pode ser destinada a uma instituição específica, campanha específica (nesse caso a campanha já é atrelada a uma instituição) ou criar intenção de doação sem campanha ou instituição definidas. Neste caso, a doação fica disponível para que instituições da região/localidade do doador possam solicitá-la.

6.1.5 Buscar e levar

Outra ação particular que o usuário doador pode realizar é a de buscar e levar uma doação. Essa é uma modalidade de doação em que o doador consulta itens destinados à campanhas ou instituições específicas em que outro doador sinalizou que não possui meios próprios de transportar a doação até a instituição. Esta ação permite que o doador realize o transporte da doação no endereço determinado até a instituição de destino.

6.2 Seções do aplicativo

O aplicativo para dispositivos móveis está organizado em duas visões baseadas em usuário: instituição e doador. Para cada visão foi associada uma barra de navegação contendo quatro seções de atividades, que podem ser visualizadas na Figura [28](#).



Figura 28 – Barras de navegação para as visões do aplicativo

6.2.1 Visão doador

6.2.1.1 Início

A seção de Início contém a primeira e principal tela do aplicativo, apresentada na Figura 29. Esta tela é a mesma tanto na visão do usuário doador quanto na do usuário instituição e contém as seguintes funcionalidades: fazer doação, buscar/levar e ver doações mais recentes.

A opção **Doar** aparece na tela inicial e abre uma nova tela (Figura 30) contendo: um formulário de campos da doação e um botão para a tela (Figura 31) de contribuições da modalidade buscar/levar considerando a localidade atual do usuário autenticado. Abaixo do formulário é exibida uma lista, com rolagem horizontal, de cartões com informações de campanhas de doações das instituições daquela localidade (nome da instituição, foto de perfil da instituição, descrição, objetivos, nome do item ou serviço desejado, meta de quantidade, progresso atingido), um ícone de caminhão para indicar se a instituição possui meios próprios de buscar as doações daquela campanha, um botão para compartilhar a campanha de doação e um botão para acessar mais informações da mesma.

Uma segunda lista, também de rolagem horizontal, exibe os cartões de doações que aguardam transporte, contendo informações como: categoria da doação, tamanho do item, tipo de veículo recomendado para realizar o transporte, parte do endereço da localização da doação, horário de funcionamento da instituição destino, botão para compartilhar a doação representada no cartão e um botão para ver mais informações.

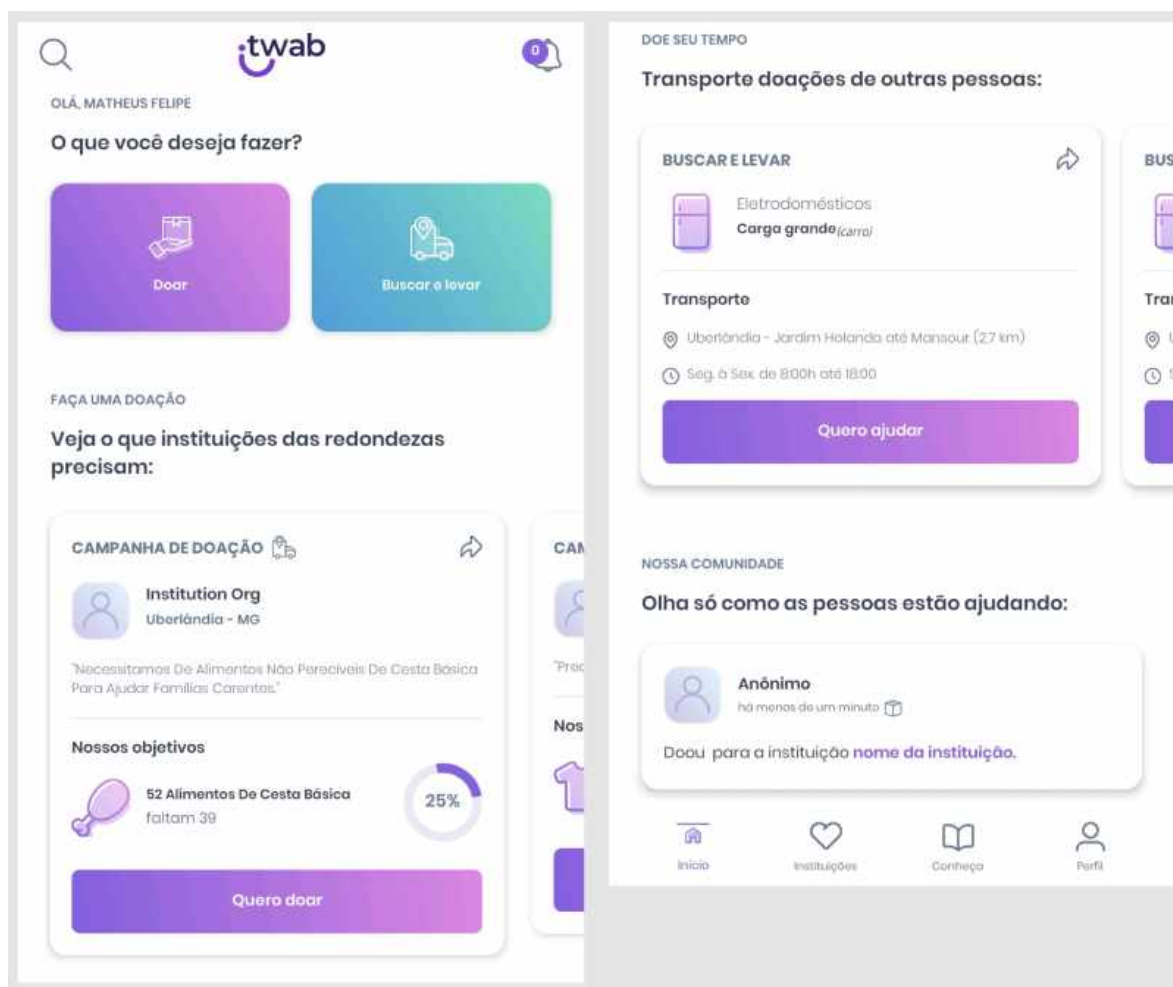


Figura 29 – Captura de tela da seção de início da visão do doador.

A terceira e última parte da seção de início é responsável por exibir as doações mais recentes da localidade do usuário, com no máximo cinco cartões. Por questões de usabilidade e desempenho, as duas listas de cartões (campanhas e doações aguardando transporte) exibem a quantidade de no máximo dez cartões. Assim, quando o usuário chega no final destas duas listas há um décimo primeiro cartão que indica a quantidade total de cartões da lista e um botão que leva à uma listagem com todos os cartões daquele contexto.

Esta seção de início permite ao usuário navegar para as seguintes telas:

- Tela de Nova Doação (Figura 30): tela que contém um formulário com todos os campos necessários para se criar uma doação. No formulário pode-se especificar informações do item ou serviço a ser doado como: sua categoria, nome do item ou serviço, quantidade, descrição do item, selecionar uma instituição de destino, informar se deseja doar de forma anônima, se irá levar a doação até instituição ou não e fotos do item. O formulário tem algumas variações de acordo com certas opções selecionadas. Por exemplo, se o usuário selecionar a opção de doar anonimamente,

sua identidade não será revelada para outros usuários do aplicativo, mas caso ele selecione a opção de que não pode levar a doação até a instituição então mais campos (disponibilidade e informações para transporte) são exibidos no formulário e somente o usuário que for buscar sua doação poderá ver suas informações de identificação. Se a opção de categoria for de **Serviços** então o campo de **Fotos do item** não será exibido.

A imagem mostra duas telas de um aplicativo de doação. A tela da esquerda, intitulada 'Nova doação', contém o formulário principal. No topo, há uma barra de navegação com 'Início' e 'Nova doação'. O formulário começa com a pergunta 'O que você quer doar?' e inclui campos para 'Categoria' (selecionado 'Outros'), 'Nome do item' ('Livros'), 'Quantidade' ('4') e uma descrição ('Livros de histórias infantis'). Abaixo, há uma seção 'Fotos do item' com uma galeria de imagens. A seção 'Defina suas preferências' contém um toggle para 'Doar para uma instituição específica' (ativado) e uma barra de busca para encontrar instituições. Uma lista de instituições é exibida, com 'Institution Org' selecionada. A tela da direita, intitulada 'Vou levar a minha doação até a instituição', contém campos para 'Tamanho da caixa' ('Pequena'), 'Valor recomendado' ('Carro'), 'Período disponível' ('Manhã'), 'Data inicial' ('8/4/2021') e 'Data final' ('12/4/2021'). Há também um campo para 'Observações' e um toggle para 'Quero que seja anônimo' (desativado). No final, há um botão 'Fazer doação'.

Figura 30 – Captura de tela do formulário de doação preenchido.

- Tela do Buscar e Levar (Figura 31): tela que contém duas listas, que juntas totalizam os cartões que representam uma prévia das informações de uma doação na modalidade Buscar e Levar. A primeira lista é deslizável horizontalmente e contém as doações mais próximas da localidade atual do usuário com um limite de exibi-

ção de dez doações. A última composição da tela é uma segunda lista vertical que contém o restante das doações da localidade atual do usuário.

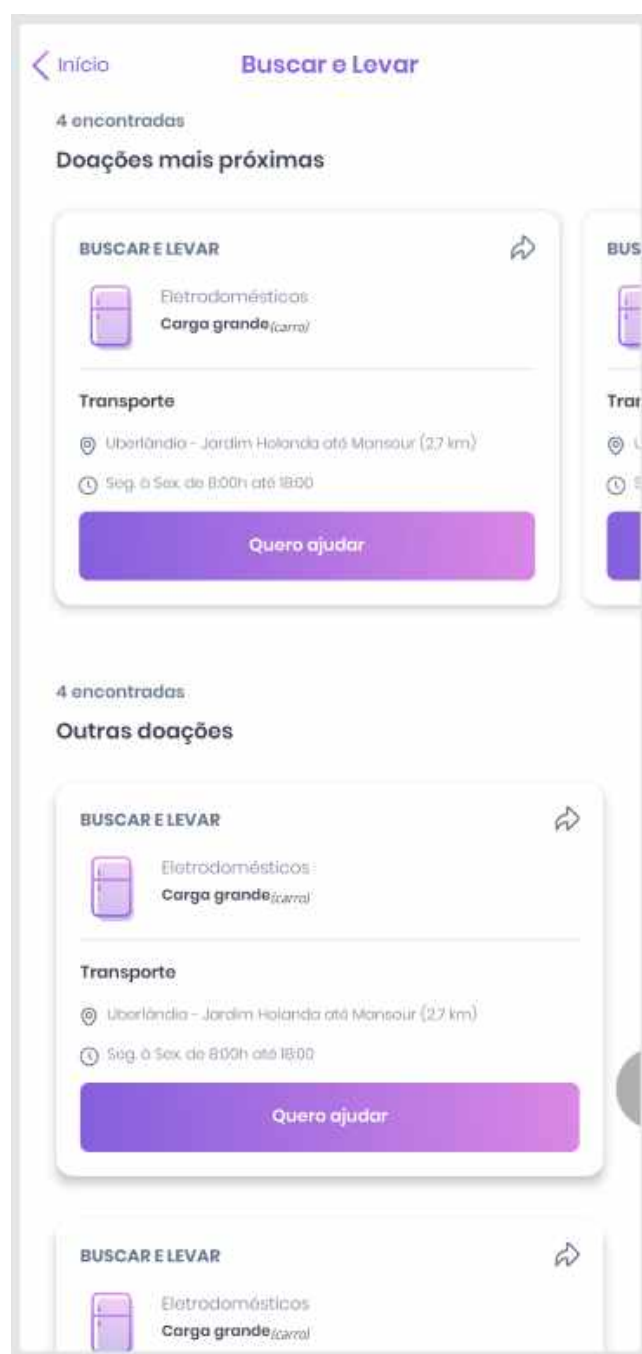


Figura 31 – Captura da tela de doações na modalidade Buscar e Levar.

- Tela de detalhes da Campanha de Doação (Figura 32): tela que contém todas as informações de uma campanha de doação como foto da campanha, identificador da campanha, botão de compartilhar, informações do item desejado na campanha (nome, quantidade, meta atingida e ícone da categoria), data de criação, informações da instituição (nome, cidade, UF, telefone, botão para acessar perfil), descrição, disponibilidade de meios próprios para buscar doações da campanha e um botão

que navega para uma tela que contém um formulário específico para doação de campanha. Esse formulário (lado direito da Figura 32) já contém informações da instituição e do item ou serviço desejado na campanha.

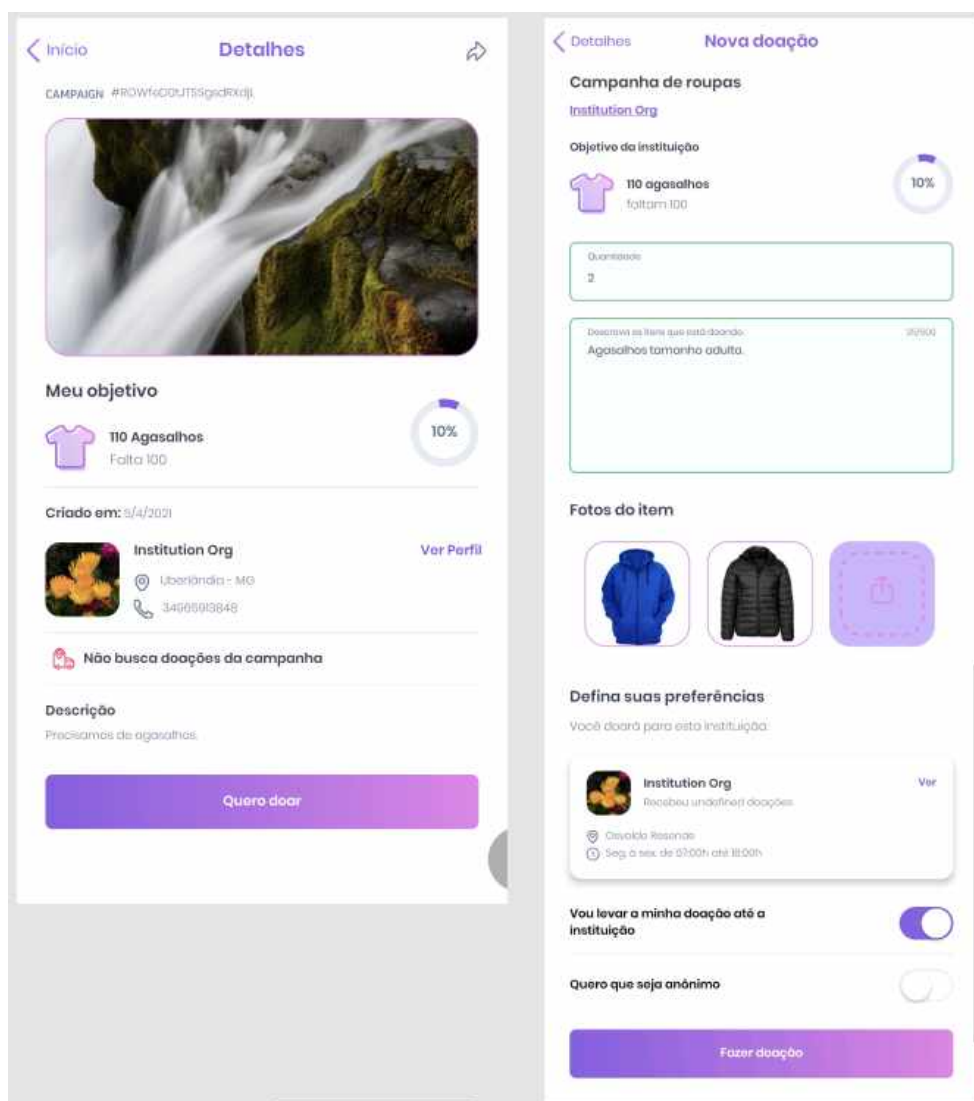


Figura 32 – Capturas da tela de detalhes de uma campanha de doação e do formulário de doação para campanha.

- Tela de detalhes da modalidade Buscar e Levar (Figura 33): O lado esquerdo da Figura 33 contém as informações de uma doação que aguarda transporte: identificador da doação, fotos do item, categoria, nome do doador, data da doação, instituição (nome, endereço, horário de funcionamento e ponto de contato), tipo da carga, veículo recomendado, doador que fez o cadastro da doação (nome, parte do endereço, disponibilidade de período), botão de compartilhar e um botão que confirma a intenção do usuário de realizar o transporte da doação.

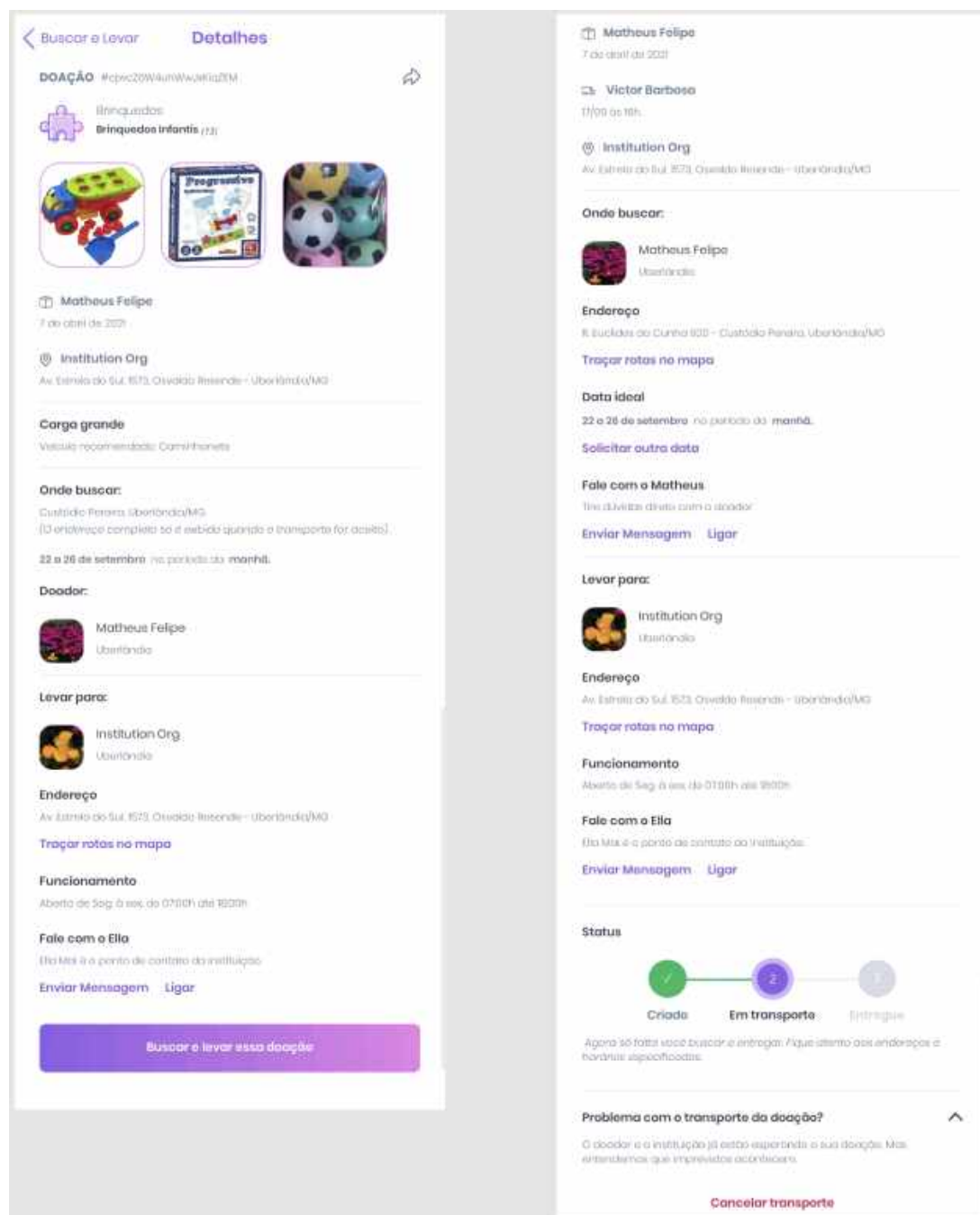


Figura 33 – Captura da tela de detalhes de antes e depois de realizar uma colaboração na modalidade Buscar e Levar.

O lado direito da Figura 33 é a mesma tela descrita anteriormente só que com informações adicionais: status da etapa do transporte, botão de cancelar o transporte, botão para solicitar outra data e horário, endereço completo de onde se localiza o item da doação.

6.2.1.2 Instituições

Esta seção do aplicativo é responsável por listar todas as instituições da cidade em que o usuário está localizado. Assim como na tela de Buscar e Levar, são exibidas duas listas com cartões que contêm informações do perfil de uma instituição (nome, quantidade de doações recebidas, bairro, distância em quilômetros da localização atual do usuário até o da instituição, dias e horários de funcionamento). Ao pressionar um cartão, o usuário navega para a tela de perfil da instituição (lado direito da Figura 34), podendo assim ver todas as informações da mesma: se transporta doações, horário de funcionamento, descrição, número de telefone, redes sociais, e-mail, quantidade de doações recebidas, campanhas ativas, lista de doações recentemente recebidas, lista com campanhas ativas, botão de compartilhar, e um botão para o formulário de nova doação.

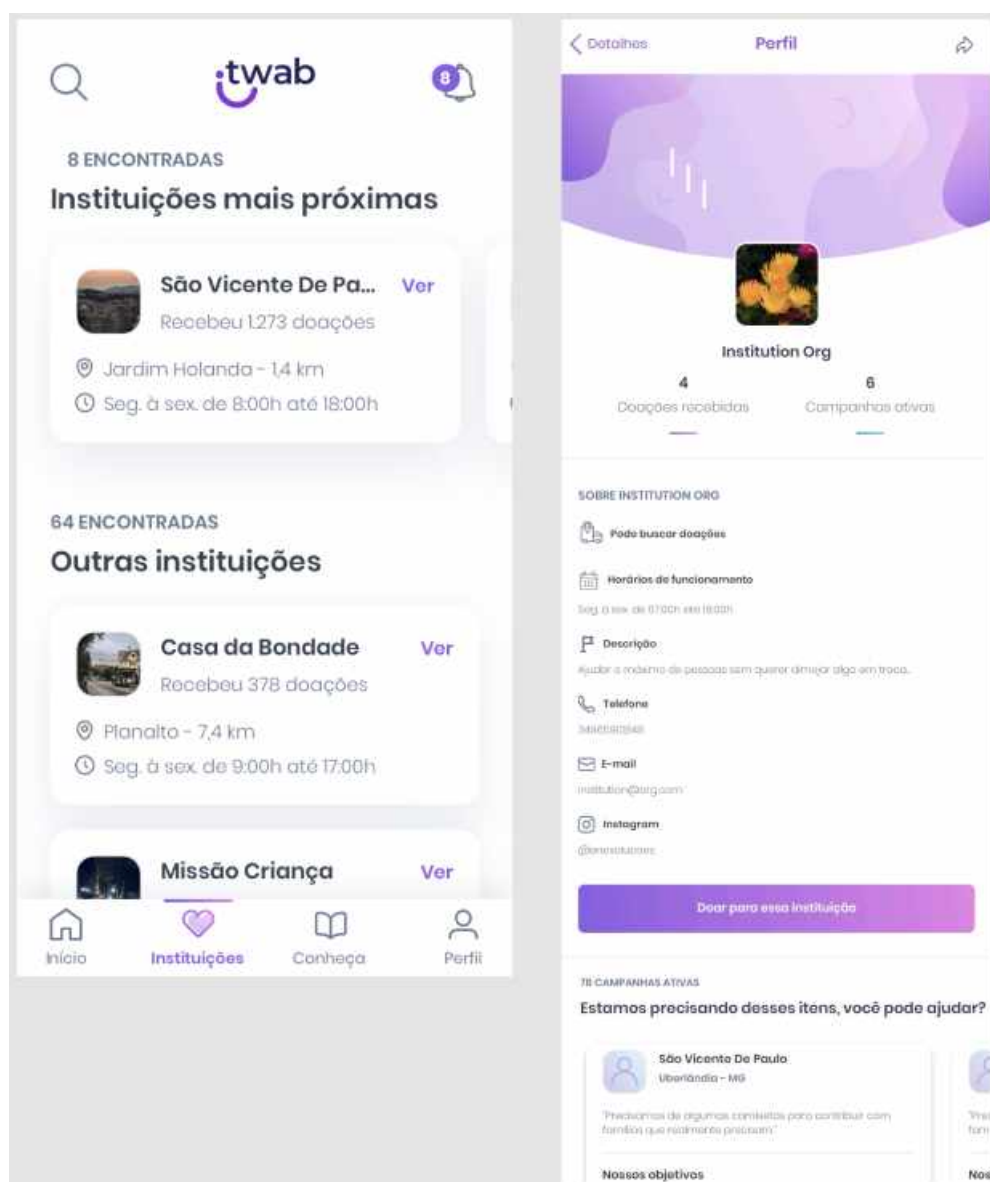


Figura 34 – Captura das telas, seção instituições e perfil da instituição.

6.2.1.3 Conheça

Esta seção contém uma tela (Figura 35) com informações estáticas, com o objetivo de informar ao usuário (doador e instituição) quais categorias seguidas de uma breve descrição motivacional, estão disponíveis para selecionar na hora de fazer doação ou criar campanha.



Figura 35 – Captura de tela da seção Conheça.

6.2.1.4 Perfil

Esta seção é a segunda mais importante para o usuário, tanto doador quanto instituição, porque nela são gerenciadas as informações do seu perfil (foto de perfil, foto de

capa, descrição, redes sociais e telefone) e edição de configurações da conta (nome, e-mail, senha, endereço e CPF). O usuário doador pode consultar suas atividades de doação (em andamento e concluídas) de materiais ou serviços e contribuições na modalidade Buscar e Levar. Na parte superior acima da foto de capa na Figura 36, há um botão para finalizar a sessão de autenticação do usuário.

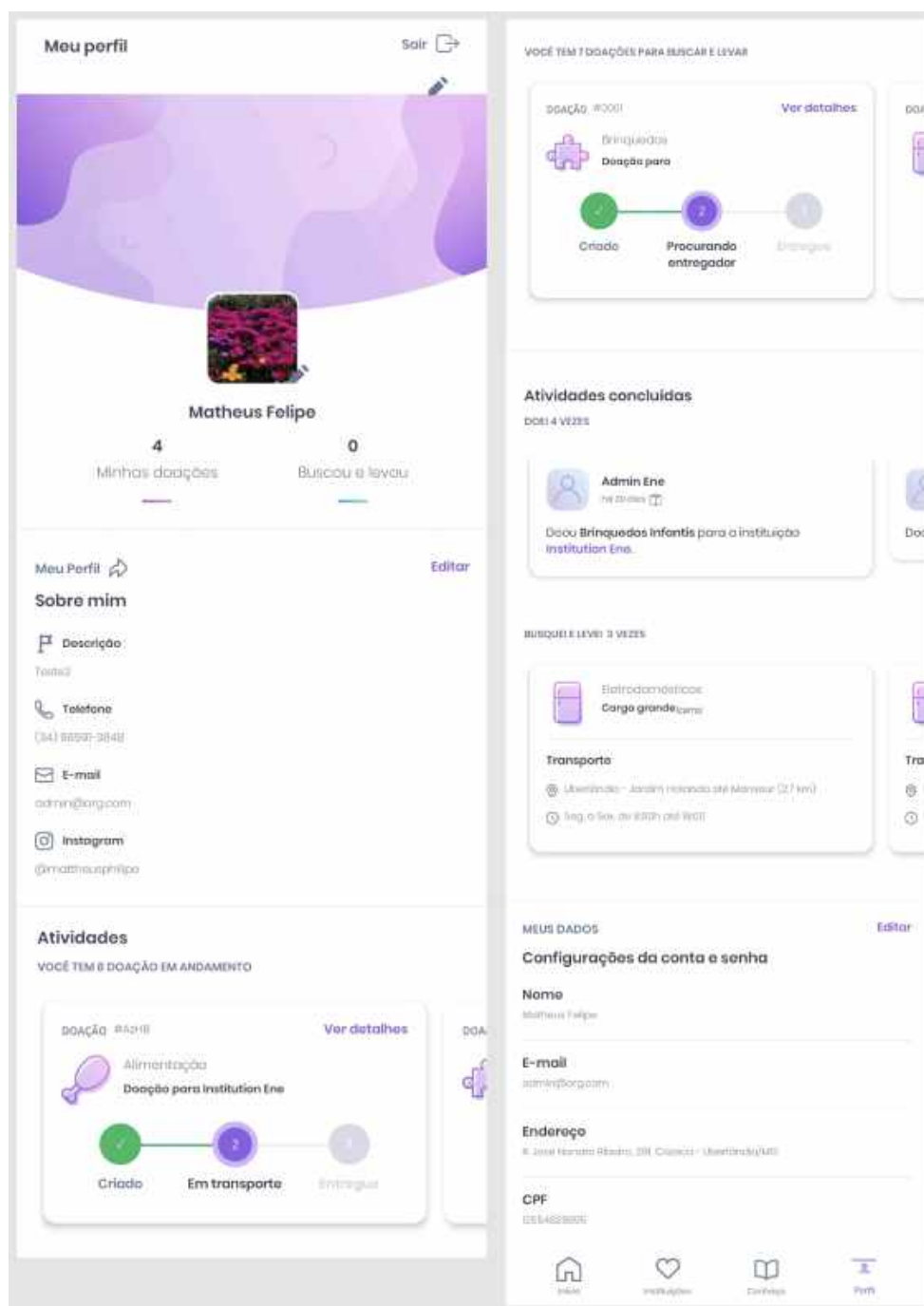


Figura 36 – Captura de tela da seção de início da visão do doador.

6.2.2 Visão instituição

6.2.2.1 Início

A seção inicial provê o ambiente no qual o usuário instituição pode gerenciar e realizar ações sobre as doações destinadas a si.

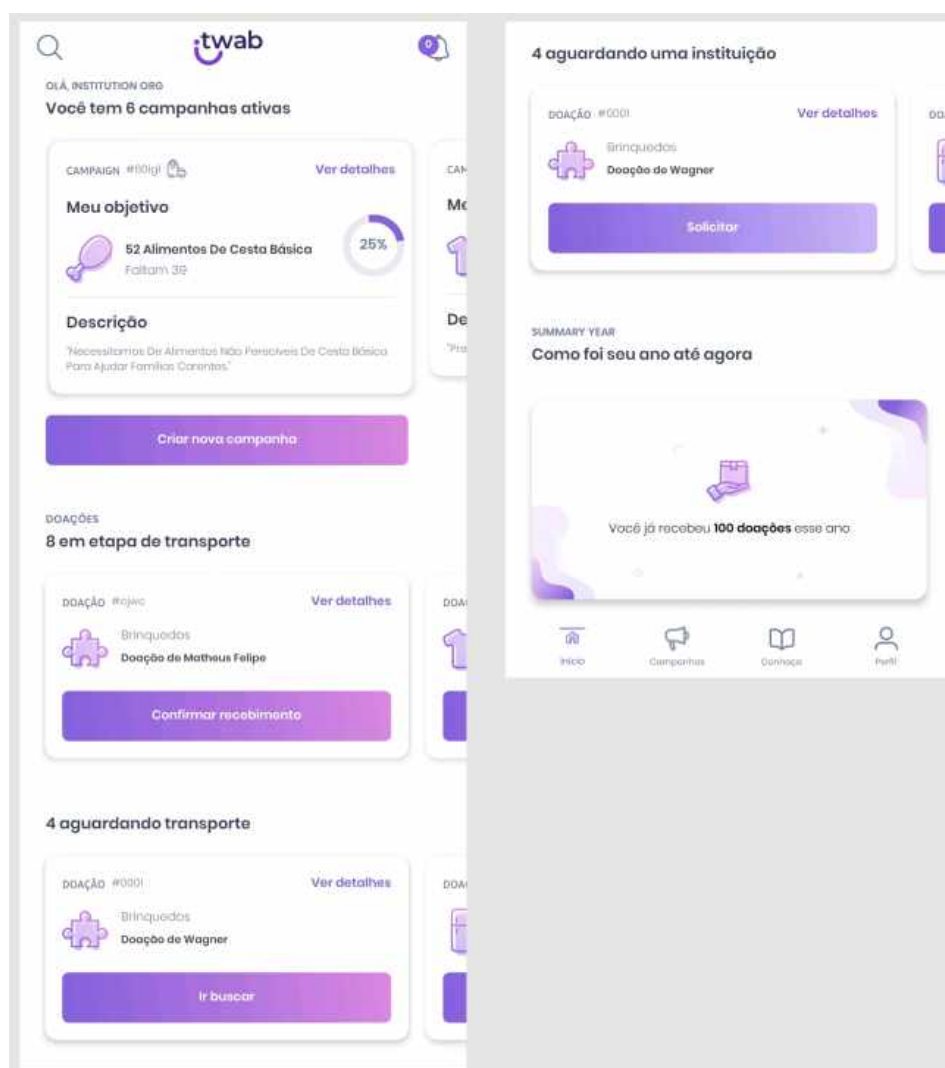


Figura 37 – Captura de tela da seção de início da visão da instituição.

A seção é composta por uma tela (Figura 37) que contém quatro listas horizontais, sendo a primeira de campanhas ativas e as outras três de cartões de doações com um resumo das informações (identificador da doação, categoria, ícone da categoria, nome do doador, botão para navegar para a tela de detalhes (lado direito da Figura 38) e um botão de ação) e os seguintes status:

- Etapa de transporte: doações que a instituição está aguardando o recebimento em seu endereço. O botão de ação dos cartões dessa lista é para confirmar o recebimento da doação. Neste processo é aberta uma caixa de diálogo (lado esquerdo da

Figura 38) perguntando ao usuário instituição se deseja confirmar a ação. É neste momento que a doação é contabilizada para a campanha se a mesma faz parte de uma, e o status dela é alterado para concluído e o status da doação como entregue;

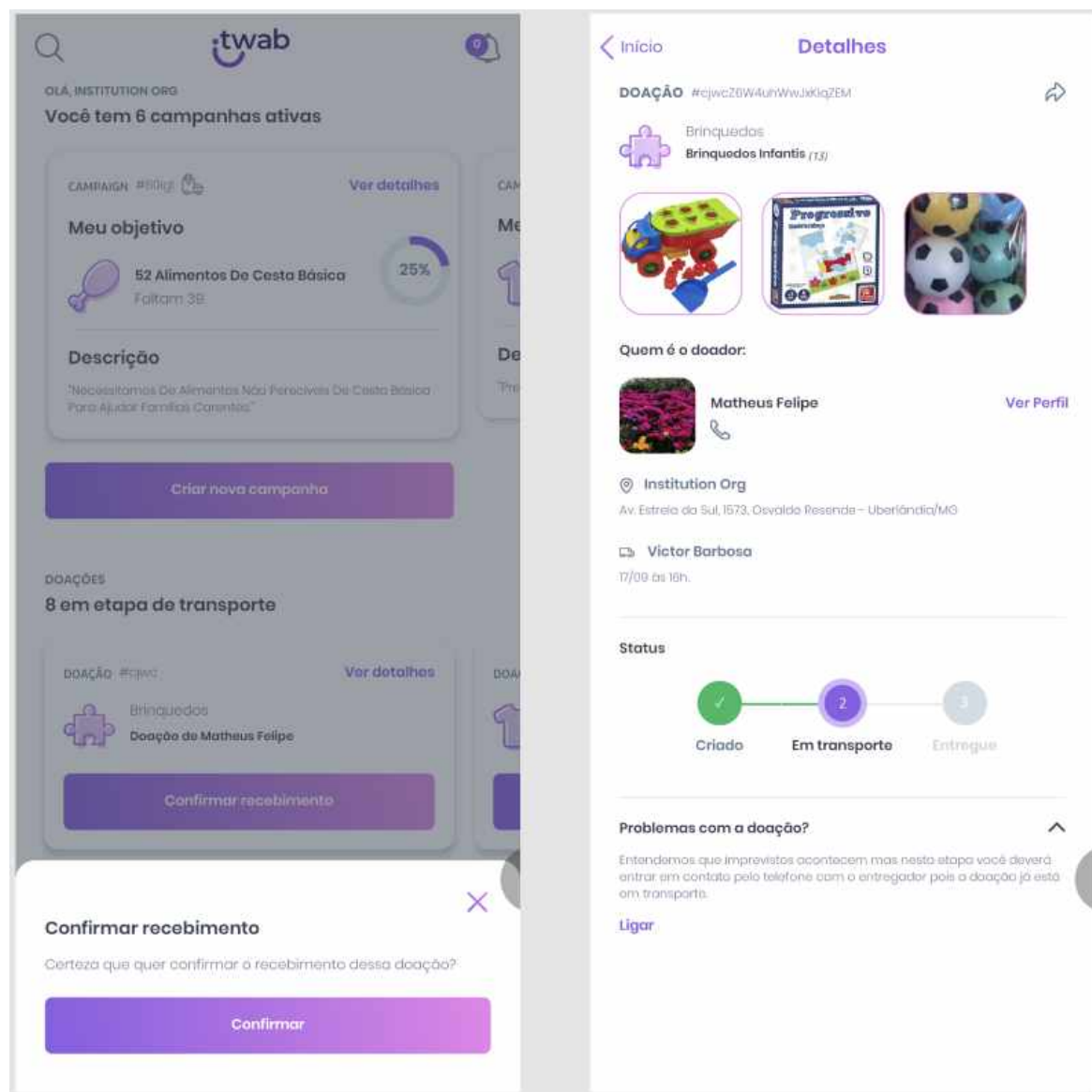


Figura 38 – Captura das telas de confirmar o recebimento de uma doação e os detalhes dela.

- Aguardando transporte: doações para as quais os usuários doadores sinalizaram não possuir meios próprios de levar até a instituição. O botão de ação dos cartões dessa lista levam ao processo de Buscar e Levar demonstrado na Figura 33;
- Aguardando uma instituição: doações que não possuem instituições de destino e foram criadas por usuários que estão localizados na mesma região da instituição que está acessando o aplicativo.

Por fim a seção de início contém um botão de criar campanhas, logo abaixo da lista de campanhas ativas. A Figura 38 mostra o formulário com os campos da nova campanha: categoria, nome do item, meta, gestor de doações daquela campanha, descrição e sinalização do transporte (se pode transportar ou não as doações daquela campanha).



A captura de tela mostra a interface de criação de uma nova campanha. No topo, há uma barra de navegação com um ícone de seta para trás e o texto 'Início', e o título da tela 'Nova campanha'. Abaixo, há uma seção para upload de imagem com um ícone de câmera e o texto 'Coloque uma foto de capa para a campanha'. Segue-se a pergunta 'O que a instituição precisa?' seguida por cinco campos de entrada: 'Categoria' (menu suspenso), 'Nome do item', 'Meta (quantidade de itens)', 'Gestor da campanha' (menu suspenso) e 'Escreva uma descrição..'. Abaixo dos campos, há uma opção 'Disponibilizar transporte' com um ícone de caminhão e um interruptor. No rodapé, há um botão laranja com o texto 'Criar campanha'.

Figura 39 – Captura de tela do formulário de criar nova campanha.

6.2.2.2 Campanhas

A tela da seção de campanhas (lado esquerdo da Figura 40), da mesma forma que a tela Buscar e Levar (Figura 31), possui duas listas de cartões.

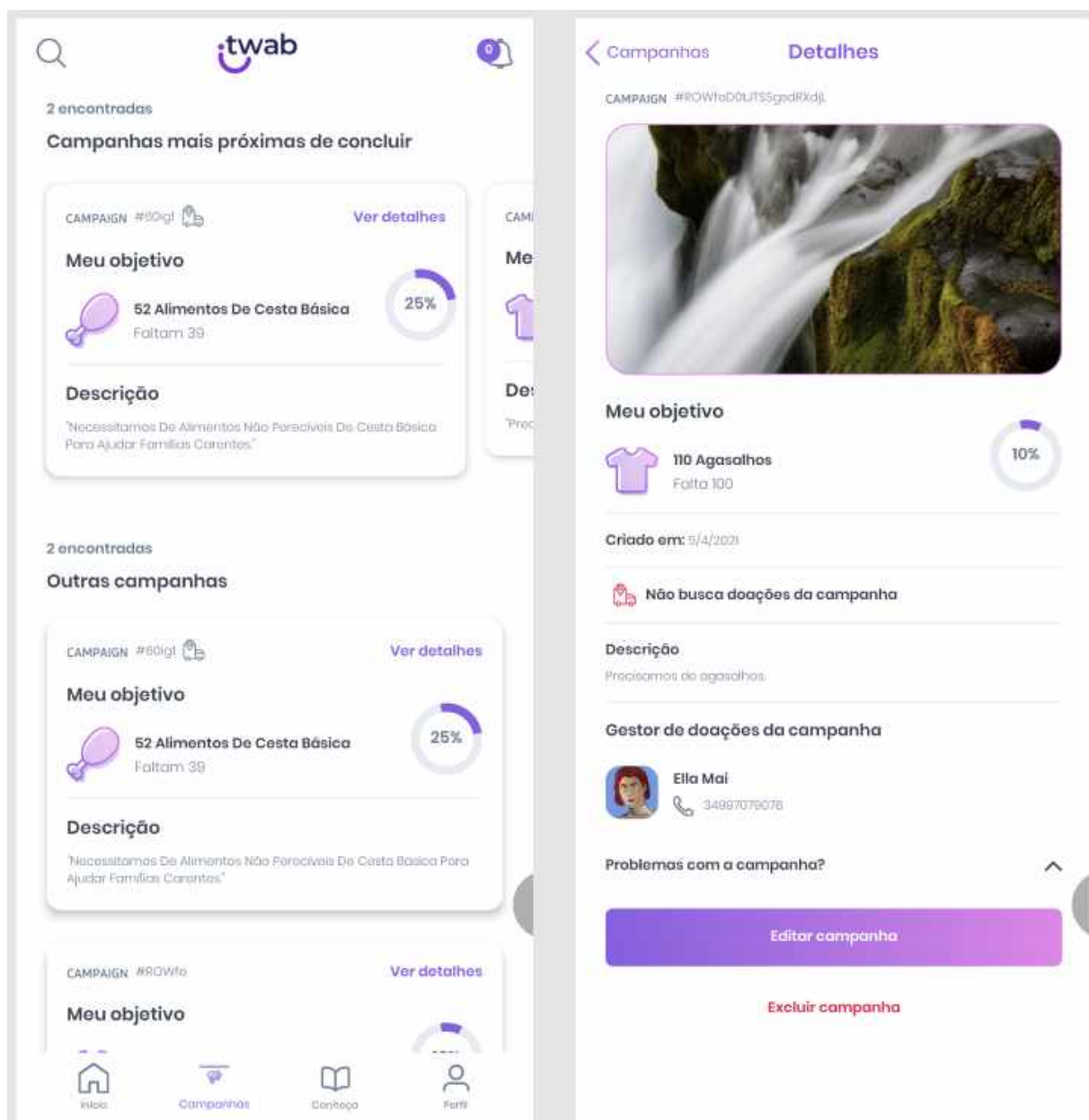


Figura 40 – Capturas de tela da seção de Campanhas e detalhes de uma campanha.

A primeira lista, com rolagem horizontal, exibe cartões de campanhas que estão próximas de serem concluídas com as informações: identificador da campanha, descrição, nome do item ou serviço desejado, meta (quantidade de itens no caso da campanha não ser da categoria de serviços, caso contrário são exibidos os campos de horas e dias no lugar), gráfico indicando de forma visual e numérica (porcentagem) o quão a campanha já atingiu seu objetivo e um botão que permite navegar para a tela de detalhes da campanha. A segunda lista possui o restante das campanhas que não estão tão próximas de serem concluídas, orientadas em uma rolagem deslizável verticalmente.

O lado direito da Figura 40 mostra todas as informações de uma campanha e as

ações que podem ser executadas, como excluir a campanha e navegar para tela de edição da campanha, que é a tela de criar nova campanha (Figura 39), mas com os campos já previamente preenchidos.

6.2.2.3 Perfil

Os componentes e elementos visuais da tela da seção de perfil (Figura 41) são os mesmos para o usuário doador e instituição, mudando apenas o contexto e algumas funcionalidades.

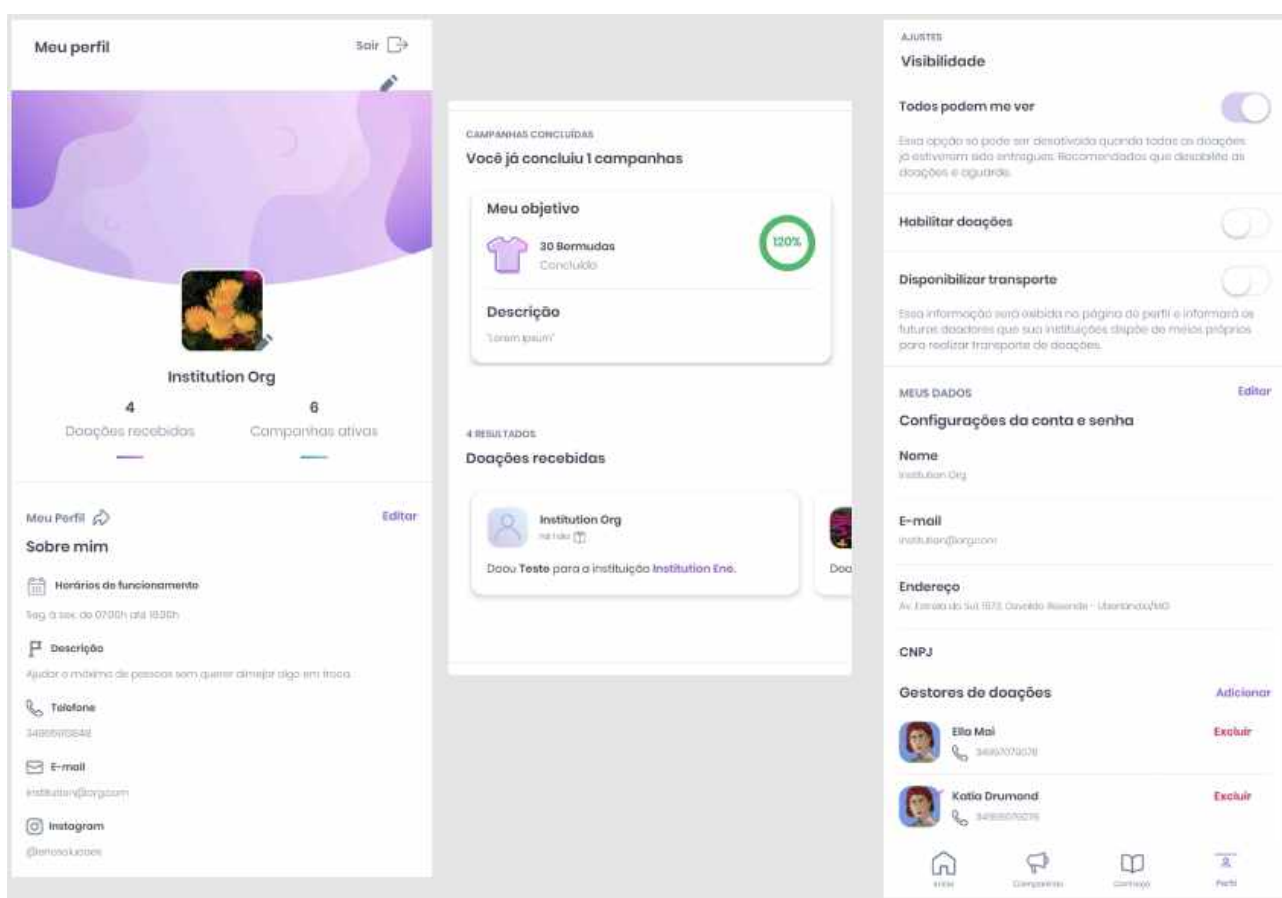


Figura 41 – Captura de tela da seção de Perfil na visão do usuário instituição.

A seção na visão do usuário instituição além de possuir as partes da tela **Sobre mim** e **configurações da conta e senha** em comum, possui também uma parte para ajustes de **visibilidade** da instituição dentro do aplicativo, duas listagens de campanhas concluídas e doações recebidas e por fim a parte de gestores de doação.

Essa seção permite ao usuário ir para as seguintes telas:

- Modal de adicionar gestor (Figura 42): janela modal para se cadastrar um gestor de doações. É obrigatório se ter pelo menos um gestor para as campanhas, que é

um funcionário da instituição, disponível para o usuário doador entrar em contato caso necessário;

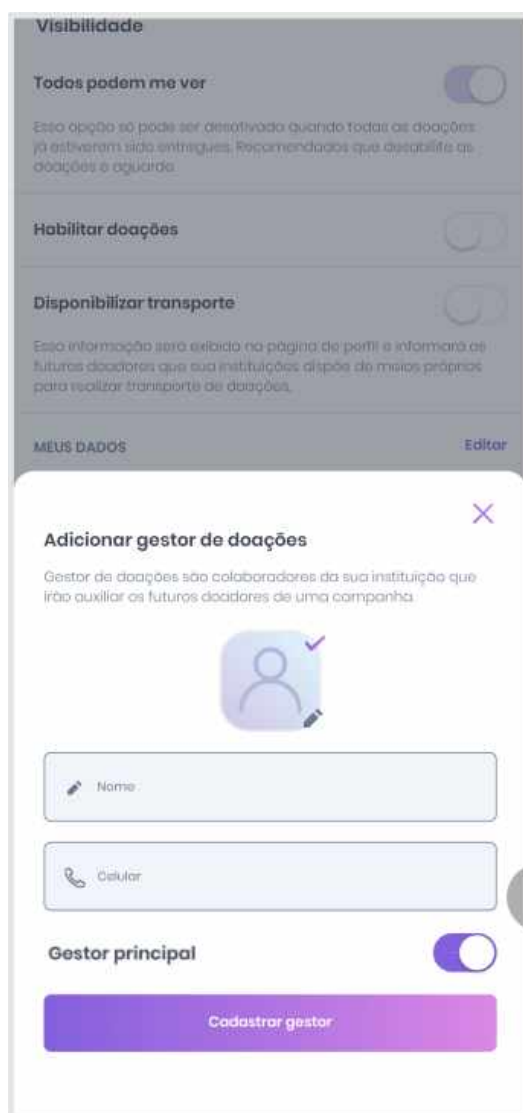


Figura 42 – Captura de tela da janela modal de cadastro de gestor de doações.

- Tela de Editar Perfil (Figura 43): tela com formulário de campos para se editar informações do perfil como descrição, telefone, redes sociais e no caso do usuário instituição dias e horários de funcionamento;

A imagem mostra a interface de edição de perfil de uma instituição. O layout é dividido em duas colunas. A coluna da esquerda, intitulada 'MEU PERFIL' e 'Editar perfil', contém um campo de texto para 'Sobre mim' com o placeholder 'Ajudar o máximo de pessoas sem querer cimejar algo em troca.', campos de entrada para 'Telefone' (34965913848), 'Instagram' (@linesolucoes), 'Facebook', 'Youtube' e 'Site'. A coluna da direita, intitulada 'Horários de funcionamento', apresenta uma lista de dias da semana com interruptores de ativação e campos de horário 'Abre' e 'Fecha'. Os dias Segunda, Terça, Quarta, Quinta e Sexta estão ativados com o mesmo horário (07:00 a 18:00). Sábado e Domingo estão desativados. Um botão 'Salvar alterações' está na base da segunda coluna.

Horários de funcionamento	Segunda	Terça	Quarta	Quinta	Sexta	Sábado	Domingo
Abre	07:00	07:00	07:00	07:00	07:00		
Fecha	18:00	18:00	18:00	18:00	18:00		

Figura 43 – Captura da tela de editar o perfil do usuário instituição.

- Tela de Editar Perfil (Figura 44): tela com formulário para editar configurações da conta como nome, *e-mail*, CPF (usuário doador), CNPJ (usuários instituição), senha (lado direito da figura 44) e endereço.

The image displays two side-by-side screenshots of a mobile application interface for editing a user's account. Both screens have a header with a back arrow, the text 'Perfil', and the title 'Editar conta'.

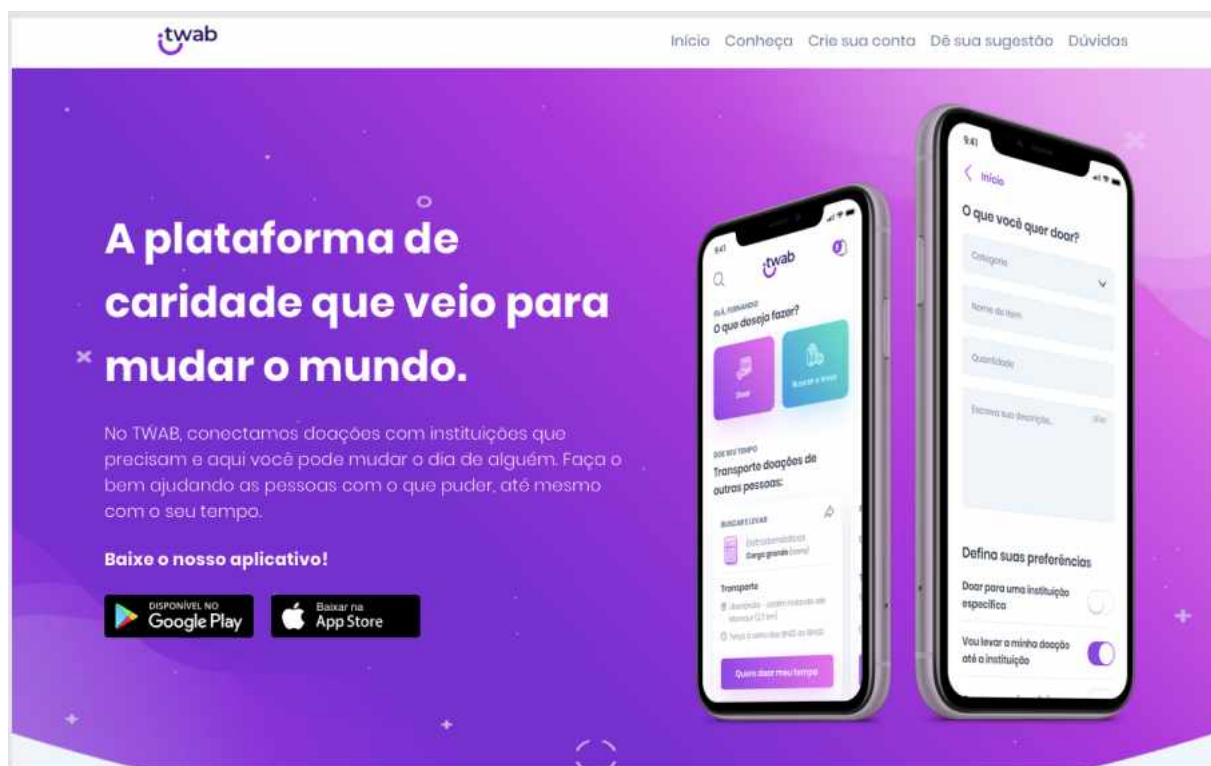
The left screenshot shows the 'Configurações da conta' (Account Settings) section. It includes a 'MEUS DADOS' (My Data) header, followed by a 'Configurações da conta' section with a collapse arrow. This section contains three input fields: 'Nome' (Name) with the value 'Institution Org', 'E-mail' (Email) with the value 'institution@org.com', and 'CNPJ' (CNPJ). Below these fields is a purple button labeled 'Salvar dados da conta' (Save account data). Below this is a link 'Alterar senha' (Change password). The 'Endereço' (Address) section is collapsed, indicated by a downward arrow.

The right screenshot shows the 'Endereço' (Address) section expanded. It includes a 'MEUS DADOS' (My Data) header, followed by a 'Configurações da conta' section with a collapse arrow. This section contains three input fields: 'Nome' (Name) with the value 'Mathias Felipe', 'E-mail' (Email) with the value 'admin@org.com', and 'CPF' (CPF) with the value '12554826695'. Below these fields is a purple button labeled 'Salvar dados da conta' (Save account data). Below this is a link 'Alterar senha' (Change password). The 'Endereço' section is expanded, showing a 'CEP' (384003489) and 'Número' (1573) field, a 'Tipo' (Avenida) dropdown, an 'Endereço' (Estrela do Sul) field, a 'Bairro' (Osvaldo Resende) field, a 'Cidade' (Uberlândia) field, a 'UF' (MG) field, and a 'Complemento' field. Below these fields is a purple button labeled 'Salvar endereço' (Save address). Below this is a link 'Alterar senha' (Change password). The 'Alterar senha' section is expanded, showing a 'Senha' (Password) field, a 'Confirme a senha' (Confirm password) field, a password strength indicator (green checkmark), and a purple button labeled 'Confirmar alteração' (Confirm change).

Figura 44 – Captura da tela de editar configurações de conta e senha do usuário instituição.

6.3 Seções da *landing-page*

A seção de Início da *landing-page* é o chamariz que contém um título e uma descrição contendo o nome do aplicativo (TWAB) e a proposta dele. Na Figura 45 aparecem dois botões que, quando pressionados, levam respectivamente às lojas de aplicativo Google Play (Android) e App Store (Apple).

Figura 45 – Captura de tela da seção de Início da *landing-page*.

A seção Crie Sua Conta contém informações para que os usuários de uma instituição possam solicitar seu cadastro por um formulário. Há um botão na seção que, quando pressionado, encaminha o usuário para a página (Figura 47) do formulário de cadastro. Uma vez finalizado o cadastro, o mesmo segue para um processo de análise e aprovação.

Figura 46 – Captura de tela da seção Crie Sua Conta da *landing-page*.

Cadastro de instituição



Colocar uma imagem de capa para instituição



Perfil

Perfil

Nome da instituição*	CNPJ*
Telefone*	Celular*
Descrição	
☐ Posso buscar doações(opcional).	

Endereço

Selecione o endereço no mapa



CEP*

Bairro*

Rua*

Número*

Tipo de logradouro* Estado* Cidade*

Selecione ↓ Selecione ↓ Selecione ↓

Complemento*

Autenticação

E-mail*

Senha*

Repita a senha*

☐ *Declaro que a sou a instituição mencionada acima e respondo pela mesma.

[Solicitar minha conta](#)

Figura 47 – Captura de tela do formulário de cadastro de instituição da *landing-page*.

7 Conclusão

Estamos atualmente vivendo um cenário em nossas vidas que sofreu uma revolução na forma de como as pessoas lidam com os mais diversos aspectos do cotidiano, através da evolução das tecnologias digitais, disseminação do acesso a internet e a confecção de dispositivos móveis mais acessíveis para diferentes classes sociais. Um dispositivo digital móvel se tornou parte integral das vidas das pessoas para suprir a necessidade desde de realizar desde tarefas simples como realizar uma ligação e capturar uma foto, desde a tarefas que mais complexas como realizar uma transação bancária e comprar comida em um restaurante e receber na porta da sua casa em poucos passos através de uma aplicação de *software* em um dispositivo móvel.

O tema de doações e voluntariado com o objetivo de contribuir para ações sociais está sempre em destaque independentemente de calamidades e desastres estarem ocorrendo. Desde 2017 as ações sociais no Brasil como doações estão estagnadas. Uma pesquisa identificou que as pessoas realizariam mais doações se tivessem mais dinheiro, já que apontado pela pesquisa é o método mais utilizado ([CAF, 2018](#)).

As doações pela internet têm ganhado mais impulso com o surgimento de novas plataformas para contribuir nas causas de ações sociais e pela disseminação das informações dessas necessidades de ações nas redes sociais. Dessa forma, neste trabalho foram construídas duas aplicações de *software* com a proposta de promover mais engajamento para a causa de ações sociais com enfoque em outras possibilidades de contribuição que não sejam em dinheiro. Criou-se uma página *web* para permitir o engajamento das pessoas as serem doadoras voluntárias e cadastrar instituições e um aplicativo multiplataforma para dispositivos móveis que permite um usuário doador voluntário a realizar ações de doações de recursos materiais e serviços para instituições, e usuários instituições a criação e gerenciamento de suas campanhas e doações.

7.1 Trabalhos futuros

Os *softwares* construídos neste trabalho são uma nova proposta para causa de ações sociais na forma de um MVP. Tendo isso em vista eles trazem uma abordagem prévia de como os usuários irão realizar essas ações. A principal necessidade para se validar a execução dessas ações é realizar teste de campo dos *softwares* com um grupo seletivo de usuários doadores e instituições para obter informações de *feedback*. Com essas informações em mãos será possível moldar a aplicação ao mais próximo da necessidade da causa, que é o intuito de um *software* no formato MVP. Há uma série de aspectos para se lidar na causa de ações sociais envolvendo a relação entre pessoas e instituições. Por mais

que toda a parte *front-end* com todas as telas previstas e não previstas tenham sido implementadas, há algumas funcionalidades que precisam de finalização e aprimoramentos, além dos seguintes aspectos ainda precisarem ser trabalhados futuramente:

- Buscar e levar: sem dúvidas esta é a funcionalidade mais desafiadora do aplicativo. Envolve a relação de três usuários, um doador de item, uma instituição e um doador para ir buscar a doação (no local indicado pelo doador) do item e levar até a instituição. Logo de início há o desafio de como lidar com questões de segurança, compartilhamento de endereço e proporcionar confiabilidade através da plataforma;
- Privacidade: aplicação da Lei Geral de Proteção de Dados (LGPD) N° 13.709/18 nos *softwares* de maneira a deixar ciente ao usuário quais dos seus dados podem ser obtidos, destinação e a disponibilização da solicitação dos mesmos;
- Elaborar e aplicar casos de testes funcionais, desempenho e de segurança;
- Incluir aprimoramentos diversos a partir do *feedback* de testes;
- Transformar a *landing-page* numa plataforma para gerenciamento da aplicação;
- Criar o processo de análise e aprovação do cadastro de instituições;
- Finalizar a integração de funcionalidades entre a camada *front-end* e *back-end*.

Referências

- ADOBE. *Desenvolvidos para a sua maneira de trabalhar*. 2021. Disponível em <<https://www.adobe.com/br/products/xd/features.html>>. Online, acessado em 24/05/2021. Citado na página 23.
- BONFIM, M. *Introdução ao Scrum*. 2013. Disponível em <<https://www.devmedia.com.br/apresentacao-do-scrum/27887>>. Online, acessado em 09/04/2021. Citado na página 32.
- CAF, R. *Giving Report 2019 Brasil - Um retrato da doação no Brasil*. 2018. Disponível em: <<https://www.idis.org.br/caf-brazil-report-2018/>>. Citado 2 vezes nas páginas 12 e 80.
- DOARTI. *O Projeto*. 2020. Disponível em <<https://doarti.com.br/projeto.html>>. Online, acessado em 29/10/2020. Citado 3 vezes nas páginas 7, 19 e 20.
- DRIESSEN, V. *A successful Git branching model*. 2010. Disponível em <<http://nvie.com/posts/a-successful-git-branching-model/>>. Online, acessado em 24/04/2021. Citado na página 57.
- EISENMAN, B. *Learning React Native: Building Native Mobile Apps with JavaScript*. [S.l.]: O'Reilly Media, Inc., 2015. ISBN 1491929073, 9781491929070. Citado na página 27.
- FÉ; MOURA, A. L. D. *Tecnologias móveis e vida pessoal: Uma pesquisa sobre o impacto da comunicação sem fio no tempo de trabalho e nas demais esferas da vida*. Tese (Doutorado) — PUC-SP, Curso de Comunicação e Semiótica, Signo e Significado das Mídias, São Paulo - SP, 2008. Disponível em: <<https://tede2.pucsp.br/handle/handle/5073>>. Citado na página 12.
- FIREBASE, G. *A comprehensive app development platform*. 2020. Disponível em <<https://firebase.google.com>>. Online, acessado em 19/08/2020. Citado na página 29.
- FIREBASE, G. *Cloud Firestore Data model*. 2020. Disponível em <<https://firebase.google.com/docs/firestore/data-model>>. Online, acessado em 19/08/2020. Citado na página 30.
- FIREBASE, G. *Structuring Cloud Firestore Security Rules*. 2020. Disponível em <<https://firebase.google.com/docs/firestore/security/rules-structure>>. Online, acessado em 19/08/2020. Citado na página 30.
- FIREBASE, G. *Choose a data structure*. 2021. Disponível em <<https://firebase.google.com/docs/firestore/manage-data/structure-data>>. Online, acessado em 14/04/2021. Citado na página 52.
- FIREBASE, G. *Choose a database: Cloud Firestore or Realtime Database*. 2021. Disponível em <<https://firebase.google.com/docs/firestore/rtdb-vs-firestore>>. Online, acessado em 14/04/2021. Citado na página 51.

GAMMA, H.; JOHNSON, V. *Padrões de projeto: soluções reutilizáveis de software orientado a objetos*. [S.l.]: Porto Alegre: Bookman, 2007. 20–22 p. ISBN 9788577800469. Citado na página 28.

ITEGRA, D. L. *Aplicativo: Uma Ajuda Boa*. 2020. Disponível em <<https://anjarahael.com/projetos/>>. Online, acessado em 29/10/2020. Citado 2 vezes nas páginas 7 e 20.

LENARDUZZI, V.; TAIBI, D. MVP explained: A systematic mapping study on the definitions of minimal viable product. In: *42th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. Limasol, Chipre: [s.n.], 2016. Citado 3 vezes nas páginas 7, 22 e 23.

MICROSOFT. *What is Xamarin?* 2020. Disponível em <<https://docs.microsoft.com/pt-br/xamarin/get-started/what-is-xamarin>>. Online, acessado em 18/08/2020. Citado na página 27.

MORESI, E. A. D. et al. Tecnologia social e empoderamento: o caso do aplicativo Solidarius. *Revista Iberoamericana de Sistemas, Cibernética e Informática*, v. 14, p. 473–478, 2017. Disponível em: <<http://www.iiisci.org/Journal/riSCI/FullText.asp?var=&id=CB065CG17>>. Citado 2 vezes nas páginas 7 e 17.

NIELSEN, J. *10 Usability Heuristics for User Interface Design*. 2005. Disponível em <<https://www.nngroup.com/articles/ten-usability-heuristics/>>. Online, acessado em 19/08/2020. Citado na página 24.

NIELSEN, J.; MOLICH, R. Heuristic evaluation of user interfaces. In: *Proceedings of the SIGCHI conference on Human factors in computing systems*. [S.l.: s.n.], 1990. p. 249–256. Citado na página 40.

PAMPLONA, P. *Brasileiro opta por doar em dinheiro, mas tecnologia pode motivar mudança*. Folha de São Paulo, UOL, 2019. Disponível em: <<https://www1.folha.uol.com.br/empreendedorsocial/2019/04/brasileiro-opta-por-doar-em-dinheiro-mas-tecnologia-pode-motivar-mudanca.shtml>>. Citado na página 12.

PARANÁ, G. do Estado do. *Aplicativo Paraná Solidário - Secretaria da Justiça, Família e Trabalho* -. 2020. Disponível em <<http://www.justica.pr.gov.br/Pagina/Aplicativo-Parana-Solidario>>. Online, acessado em 29/10/2020. Citado na página 18.

PORTAL, P. *Doação de roupas e cobertores poderá ser feito por aplicativo no Paraná*. 2020. Disponível em <<https://paranaportal.uol.com.br/cidades/doacao-roupas-cobertores-aplicativo/>>. Online, acessado em 29/10/2020. Citado 3 vezes nas páginas 7, 18 e 19.

PRUITT, J. *Personal Scrum*. 2011. Online, acessado em 24/04/2021. Disponível em: <<https://jgpruitt.wordpress.com/2011/04/10/personal-scrum/>>. Citado na página 32.

REACT. *A JavaScript library for building user interfaces*. 2020. Disponível em <<https://reactjs.org>>. Online, acessado em 18/08/2020. Citado na página 28.

REACT. *Introducing JSX*. 2020. Disponível em <<https://reactjs.org/docs/introducing-jsx.html>>. Online, acessado em 18/08/2020. Citado na página 28.

- REACT. *Virtual DOM and Internals*. 2020. Disponível em <<https://reactjs.org/docs/faq-internals.html>>. Online, acessado em 18/08/2020. Citado na página 27.
- REACTNATIVE. *Core Components and Native Components*. 2020. Disponível em <<https://reactnative.dev/docs/intro-react-native-components>>. Online, acessado em 18/08/2020. Citado 4 vezes nas páginas 7, 27, 28 e 44.
- RIES, E. *What is the minimum viable product?* 2009. Disponível em <<https://venturehacks.com/minimum-viable-product>>. Online, acessado em 02/11/2020. Citado na página 22.
- SADALAGE, P. J.; FOWLER, M. *NoSQL distilled: a brief guide to the emerging world of polyglot persistence*. [S.l.]: Pearson Education, 2013. Citado na página 30.
- SELLI, L.; GARRAFA, V. Bioética, solidariedade crítica e voluntariado orgânico. *Revista de Saúde Pública, SciELO Public Health*, v. 39, p. 473–478, 2005. Citado na página 12.
- SHARMA, D.; DAND, H. Firebase as baas for college android application. *International Journal of Computer Applications*, v. 178, p. 1–6, 06 2019. Citado na página 29.
- SKETCH, B. *The ultimate design toolkit*. 2020. Disponível em <<https://www.sketch.com/for-designers/>>. Online, acessado em 18/08/2020. Citado na página 23.
- STATCOUNTER. *Mobile Operating System Market Share Worldwide - March 2021*. 2021. Disponível em <<https://gs.statcounter.com/os-market-share/mobile/worldwide>>. Online, acessado em 12/04/2021. Citado na página 43.
- STEVENSON, D. *What is Firebase? The complete story, abridged*. 2018. Disponível em <<https://medium.com/firebase-developers/what-is-firebase-the-complete-story-abridged-bcc730c5f2c0>>. Online, acessado em 14/04/2021. Citado 2 vezes nas páginas 8 e 51.
- VALENTE, M. T. *Engenharia de Software Moderna: Princípios e Práticas para Desenvolvimento de Software com Produtividade*. [S.l.]: UmLivro, 2020. Citado 3 vezes nas páginas 7, 25 e 26.
- WESTSTRATE, M. *Defining data stores*. 2021. Disponível em <<https://mobx.js.org/defining-data-stores.html>>. Online, acessado em 13/04/2021. Citado na página 49.
- WESTSTRATE, M. *Ten minute introduction to MobX and React*. 2021. Disponível em <<https://mobx.js.org/getting-started>>. Online, acessado em 13/04/2021. Citado 3 vezes nas páginas 8, 48 e 49.
- ZEPLIN. *Delightful products require teamwork*. 2020. Disponível em <<https://zeplin.io/why-zeplin>>. Online, acessado em 18/08/2020. Citado na página 23.

Apêndices

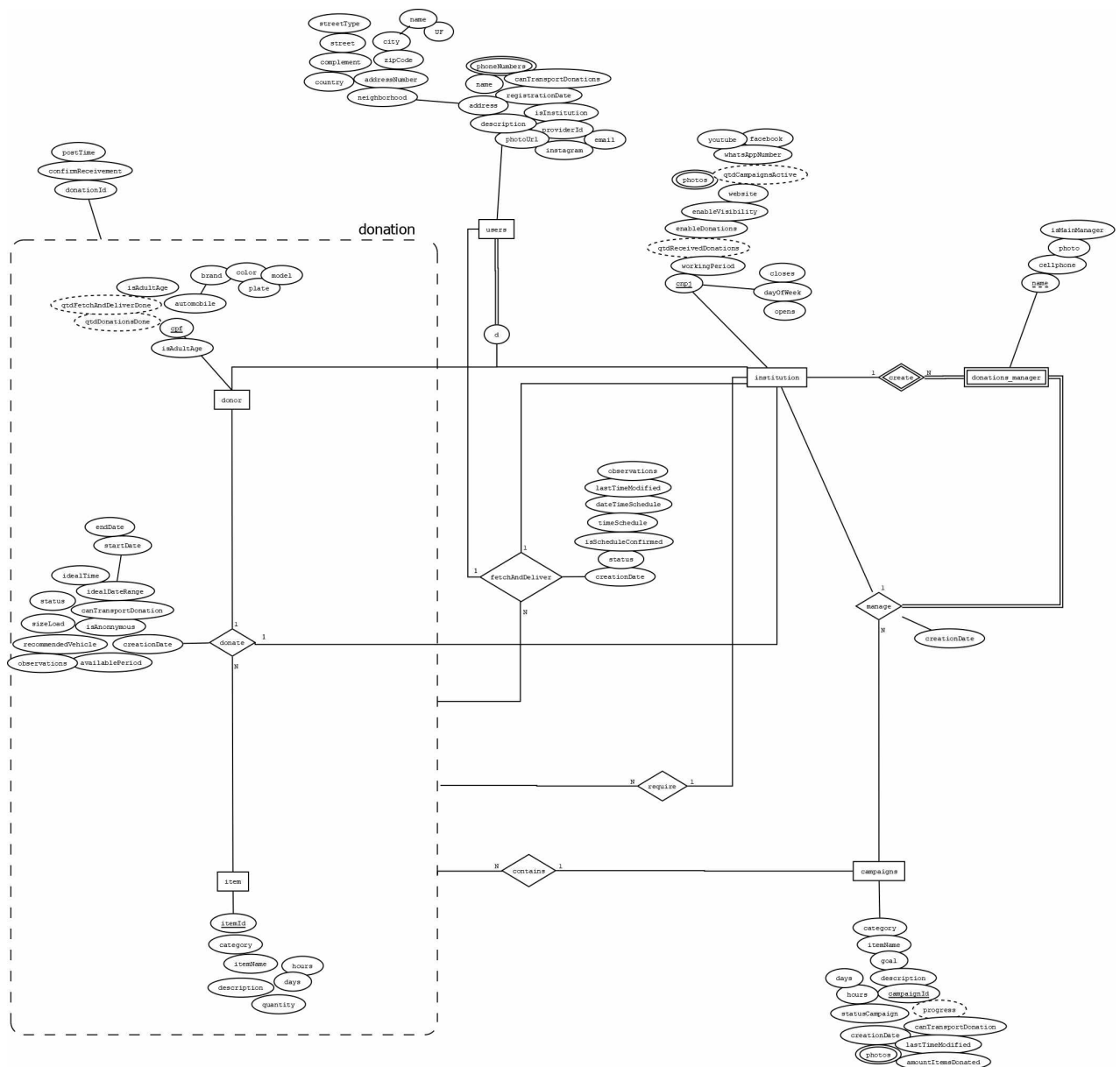
APÊNDICE A – Lista de todas as tarefas dos cinco *sprint* relacionados a produção dos *softwares deste trabalho*

TCC - Sprint 1 × TCC - Sprint 2 × TCC - Sprint 3 × TCC - Sprint 4 × TCC - Sprint 5 ×		
CARD	LIST	LABELS
Modelagem do banco de dados	Done 🐛	Priority High
Revisar telas e configurações e funcionalidades já existentes	Done 🐛	Priority High
Atualizar versão do React e React Native	Done 🐛	Priority Medium
Revisão de realização de Build de produção e homologação	Done 🐛	Priority High
Revisões dos componentes existentes após atualizações e	Done 🐛	Priority High
Desenhar fluxograma do aplicativo a partir das telas existentes	Done 🐛	Priority Medium
[Screen][Doação][Mobile] - Doação campanha específica	Done 🐛	Priority High
[Flow][Doação][Mobile] - Realizar fluxo navegação de doação	Done 🐛	Priority High
[Modal][Buscar e levar][Mobile] - Formulário para solicitar	Done 🐛	Priority High
[Modal][Agendar][Mobile] - Agenda confirmada.	Done 🐛	Priority Medium
[Mobile] - criar ícones: calendário, transporte e suas variações	Done 🐛	Priority Low
[Modal][Agendar][Mobile] - Solicitar outra data	Done 🐛	Priority High
[Modal][Busca][Mobile] - Atrair botão de busca da home	Done 🐛	Priority Low
[Modal][Doação][Mobile] - Vou levar/ Não vou levar	Done 🐛	Priority High

CARD	LIST	LABELS
 [Modal][Instituições][Mobile] - Excluir campanha	Done 	Priority Medium
 [Screen][Agendar][Mobile] - Não consigo levar.	Done 	Priority High
 [Screen][Campanhas][Mobile] - Fluxo editar campanhas	Done 	Priority High
 [Screen][Campanhas][Mobile] - Listagem de cards de cam	Done 	Priority High
 [Screen][Conheça][Mobile]	Done 	Priority Low
 [Screen][Doação][Mobile] - Buscar e levar - Inicio	Done 	Priority High
 [Screen][Doação][Mobile] - Ajudar a buscar e levar/Aceitei	Done 	Priority High
 [Screen][Perfil][Mobile] -Formulário para cadastro de pont	Done 	Priority High
 [Screen][Doação][Mobile] - Detalhes Vou levar/Não vou lev	Done 	Priority High
 [Screen][Doação][Mobile]	Done 	Priority High
 [Screen][Doador][Mobile] - Tela de instituições	Done 	Priority Low
 [Screen][Instituição][Mobile] - Detalhes em transporte/agu	Done 	Priority High
 [Screen][Instituições][Mobile] - Criar campanha	Done 	Priority High
 [Screen][Instituições][Mobile] - Inicio	Done 	Priority High
 [Screen][Instituições][Mobile] -editar campanha	Done 	Priority Medium
 [Screen][Notificações][Mobile]	Done 	Priority Low
 [Screen][Perfil - Campanha][Mobile] - Sinalizar se a institu	Done 	Priority High
 [Screen][Perfil][Mobile] - Editar dados do perfil e da conta	Done 	Priority Medium
 [Screen][Perfil][Mobile] - Instituições	Done 	Priority Low
 [Screen][Perfil][Mobile] - Usuário	Done 	Priority Low
 [Skeleton Screen][Home][Mobile]	Done 	Priority Low

CARD	LIST	LABELS
 [Skeleton Screen][Home][Mobile]	Done 🏆	Priority Low
 [Bug][Detalhes doação][Mobile] - ícone de seta voltando p	Done 🏆	Priority Low
 Resolver visualização de doação sem instituição e campani	Done 🏆	Priority High
 [Screen][Mobile] - Diagramar tela conheça.	Done 🏆	Priority Low
 Estudo React context API vs Mobx para gerenciamento d	Done 🏆	Priority High
 Integração da dependência Firestarter ao aplicativo.	Done 🏆	Priority Medium
 Criação das lojas de gerenciamento de estado da aplicação	Done 🏆	Priority High
 Estudo de como aplicar múltiplas estruturas no Cloud Fires	Done 🏆	Priority High
 Criação da estrutura de dados do DER no Firebase Cloud F	Done 🏆	Priority High
 [Database] - Colocar campo "description" na tabela users	Done 🏆	Priority Low
 [Back-end][Mobile] - Integrar com Firebase Perfomance	Done 🏆	Priority Low
 [Back-end][Mobile] - Integrar com Firebase Crashlytics	Done 🏆	Priority Low
 [Back-end][Mobile] - Integrar com Firebase Storage	Done 🏆	Priority Medium
 [Back-end][Instituição][Mobile]	To Do	Priority High Previous Sprint
 [Back-end][Usuário][Mobile]	To Do	Priority High Previous Sprint
 [Screen][Mobile] - Splash Screen	Backlog	Priority Low
 Telas front-end web	Backlog	Priority Medium
 [Testes][Mobile] - Realizar testes no aplicativo tanto Andro	Backlog	Priority Medium Validações
 [Doação][Mobile] - Ver fotos da doação em tela cheia	Backlog	Priority Medium
 [Back-end][Doação][Mobile] -Formulário com instituição p	Backlog	Priority High
 [Screen][Instituições][Mobile] - empty state tela campanha	Backlog	Priority Low

CARD	LIST	LABELS
[Back-end][Doação][Mobile] - Formulário com instituição p	Backlog	Priority High
[Screen][Instituições][Mobile] - empty state tela campanha	Backlog	Priority Low
[Screen][Home][Mobile] - Colocar cards "see all no final d	Backlog	Priority Low
[Screen][Mobile][Empty-states] - Revisar empty states em	Backlog	Priority Medium
[Screen][Buscar e levar e Perfil][Mobile] - Formulário para	Backlog	Priority Medium
[Screen][Perfil][Mobile] - Se número for apontado no cada	Backlog	Priority Medium
[Revisão][Mobile] - Padronizar o botão de voltar para tela a	Testing	Priority Medium
[Design] - Incluir no Figma capturas de tela das telas e con	Doing	Priority Medium Re-Check
[Screen][Instituição][Web] - Cadastro	Done 🐛	Priority High
[Toaster][Doação][Mobile] - Barra flutuante na tela de inici	Done 🐛	Priority Medium
[Bug][Modal][Mobile] - Teclado no iOS sobrepondo conteú	Done 🐛	Priority Medium
[Diagrama] - Estrutura de dados no Firebase Cloud Firestor	Done 🐛	Priority High Re-Check
[Diagrama] - Estrutura de arquivos do Firebase Storage	Done 🐛	Priority Low Re-Check Valid
[Mobile] - Colocar troca de senha em um modal.	Done 🐛	Priority Low
[Web][Bug] - Ajustar processo de upload de fotos com pro	Done 🐛	Priority Medium back-end
[Screen][Campaign][Mobile] - Rever os detalhes da camp	Done 🐛	Priority Medium
[Modal][Mobile] - Opções de escolha de foto.	Done 🐛	Priority Medium
[Mobile][Doação] - Alterar fluxo de ao clicar em doação pa	Done 🐛	Priority High
[Campanha][Mobile] - Cadastrar foto de capa para criação	Done 🐛	Priority Medium
[Doação][Mobile] - Cadastrar fotos de itens na doação	Done 🐛	Priority Medium
[Alteração][Doador][Mobile] - Alterar fluxo de doador para	Done 🐛	Priority High



APÊNDICE C – Diagrama de caso de uso da visão geral do sistema

