Severino: uma aplicação mobile para contratação de serviços

João Daniel de Aquino Rufino

Universidade Federal de Uberlândia
Faculdade de Computação
Trabalho de conclusão de curso

Uberlândia
2021

João Daniel de Aquino Rufino

Severino: uma aplicação mobile para contratação de serviços

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 Ciência da Computação.

Área de concentração: Ciência da Computação

Orientador: Prof. Dr. Marcelo Rodrigues de Sousa

Uberlândia
2021

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
Faculdade de Engenharia Elétrica
Av. João Naves de Ávila, 2121, Bloco 3N - Bairro Santa Mônica, Uberlândia-MG, CEP 38400-902
Telefone: (34) 3239-4701/4702 - www.feelt.ufu.br - feelt@ufu.br

ATA DE DEFESA - GRADUAÇÃO

Curso de Graduação em:	Ciência da Computação				
Defesa de:	GBC082 - Projeto de Graduação 2				
Data:	25/06/2021	Hora de início:	16h	Hora de encerramento:	16h55min
Matrícula do Discente:	11621BCC033				
Nome do Discente:	João Daniel Aquino Rufino				
Título do Trabalho:	Serverino: uma aplicação mobile para contratação de serviços				

Reuniu-se, por meio da plataforma de Conferência Web (Google Meet) - utilizada como alternativa à reunião presencial, considerando as recomendações do Ministério da Saúde e a pandemia COVID-19, a Banca Examinadora, designada pelo Colegiado do Curso de Graduação em Ciência da Computação: Bacharelado - Integral, assim composta: (Professores: Dr. Igor Santos Peretta - FEELT/UFU; Dr. Márcio José da Cunha - FEELT/UFU e Dr. Marcelo Rodrigues de Sousa - FEELT/UFU orientador do candidato.

Iniciando os trabalhos, o presidente da mesa, Dr. Marcelo Rodrigues de Sousa, apresentou a Comissão Examinadora e o candidato(a), agradeceu a presença do público, e concedeu ao discente a palavra, para a exposição do seu trabalho. A duração da apresentação do discente e o tempo de arguição e resposta foram conforme as normas do curso.

A seguir o(a) senhor(a) presidente concedeu a palavra, pela ordem sucessivamente, aos(às) examinadores(as), que passaram a arguir o(a) candidato(a). Ultimada a arguição, que se desenvolveu dentro dos termos regimentais, a Banca, em sessão secreta, atribuiu o resultado final, considerando o(a) candidato(a):

(X) Aprovado(a) Nota 85 (Somente números inteiros)

OU

( ) Aprovado(a) sem nota.

Nada mais havendo a tratar foram encerrados os trabalhos. Foi lavrada a presente ata que após lida e achada conforme foi assinada pela Banca Examinadora.

Documento assinado eletronicamente por **Marcelo Rodrigues de Sousa**, **Professor(a) do Magistério Superior**, em 29/06/2021, às 19:01, conforme

horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Igor Santos Peretta**, **Professor(a) do Magistério Superior**, em 29/06/2021, às 19:29, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Marcio José da Cunha**, **Professor(a) do Magistério Superior**, em 29/06/2021, às 20:29, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site https://www.sei.ufu.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **2850117** e o código CRC **024A7B3F**.

Referência: Processo nº 23117.039743/2021-09 SEI nº 2850117

Este trabalho é dedicado aos meus pais e amigos, que sempre me apoiam e me alegram todos os dias.

Agradecimentos

Agradeço aos meus amigos Pedro Henrique Faria e Gabriel Mendes Santiago, que ajudaram com a ideia desse projeto e em sua construção também.

Agradeço aos meus pais por sempre estarem do meu lado, me motivando até o útlimo dia.

Agradeço à cada professor que tive a oportunidade de ver as aulas, pois em cada uma, fiquei mais motivado e inspirado sobre o curso de Ciência da Computação.

"Não tenha medo de pensar diferente dos outros. Tenha medo de pensar igual e descobrir que todos estão errados."

(Eça de Queiroz)

Resumo

Nos dias de hoje é comum pessoas terem celulares em suas mãos, não importa a sua posição social, a maioria dos brasileiros hoje em dias já tem o devido acesso à tecnologia e internet. Dado isso, surgiu a oportunidade de propor uma aplicação para que pessoas busquem profissionais prestadores de serviços e possam contratá-las. Para se ter uma abordagem mais bem exemplificada, foi levantado os requisitos e prototipagem do aplicativo, afim de trazer o que era necessário para a criação do software e que atendessem aos usuários de uma maneira fácil de se entender. Nessa plataforma de nome Severino, o usuário pode se cadastrar e visualizar um aglomerado de serviços com os profissionais e suas informações, aplicando filtros como o de localidade, para que possam escolher o melhor trabalhador disponível em sua região.

Palavras-chave: Aplicação, Internet, Serviços, Profissionais, Região, Celular.

Abstract

Nowadays it is common for people to have cell phones in their hands, no matter their social position, most Brazilians nowadays already have due access to technology and the internet. Given this, the opportunity arose to propose an application for people to seek professional service providers and to hire them. In order to have a better exemplified approach, the requirements and prototyping of the application were raised, in order to bring what was necessary for the creation of the software and that would serve the users in an easy way to understand. In this platform called Severino, the user can register and view a cluster of services with professionals and their information, applying filters such as the location, so that they can choose the best worker available in their region.

Keywords: Application, Internet, Services, Professionals, Region, Cell phone.

Lista de ilustrações

Lista de siglas

ABNT Associação Brasileira de Normas Técnicas

API Application Programming Interface

CSS Cascading Style Sheets

CLI Command Line Interface

DVCS Distributed Version Control System)

HTTP Hypertext Transfer Protocol

HTML HyperText Markup Language

iOS iPhone OS

JSON JavaScript Object Notation

JSX JavaScript XML

NBR Denominação de norma da Associação Brasileira de Normas Técnicas

OTA Over the Air

OS Operating System

REST Representational state transfer

SDK Software Development Kit

UX User Experience

UI User Interface

Sumário

Capítulo 1

Introdução

Diante do alto índice de desemprego no Brasil, várias pessoas migraram para o trabalho autônomo, como: motoristas de aplicativos e entregadores, já que, de acordo com dados do IBGE, houve um aumento de 138% nos últimos oito anos no número de motoristas de aplicativos no Brasil, visto que haviam cerca de 4 milhões de brasileiros nesta atividade, em 2019 (FRAGA, 2020). Além disso, devido ao contexto atual em que estamos enfrentando uma pandemia, o número de trabalhadores autônomos atingiu a marca de 24,5 milhões em 2020, visto que mais pessoas estão buscando formas alternativas para ter uma renda por mês (BRIDI, 2020). Assim, identificou-se a oportunidade de criar uma aplicação de fácil uso tanto para quem busca o serviço, quanto para quem quer ofertá-lo.

Por mais que existam muitas pessoas que precisam de serviços rápidos como: instalações elétricas, hidráulicas, limpeza doméstica ou até mesmo professores particulares, é necessário buscar indicações com amigos e familiares com fim de encontrar profissionais qualificados e com boas recomendações. Assim, uma das propostas deste trabalho é centralizar os trabalhadores autônomos em um aplicativo capaz de informar de forma prática os serviços realizados por um dado profissional, bem como suas avaliações e seu número de contato.

Decidiu-se implementar uma aplicação para solucionar esses problemas, ela será denominada Severino, como uma espécie de agente. O nome faz referência ao personagem Severino, o faz tudo, do programa Zorra Total da Globo TV. Tal aplicação será construída em conjunto por três alunos do curso de Ciência da Computação na Universidade Federal de Uberlândia, sendo eles: João Daniel, Pedro Teixeira e Gabriel Mendes. A aplicação proposta busca amenizar o problema do desemprego juntamente com a dificuldade de encontrar profissionais qualificados e confiáveis. Com uma interface simples e amigável, será possível encontrar o trabalhador desejado com apenas dois cliques, resolvendo a complicação da demora demasiada dos meios convencionais de procura. Do ponto de vista do profissional, também será simplificado o seu meio de ingresso na aplicação, servindo principalmente como uma forma de divulgação de seu trabalho.

Cada integrante deste projeto ficará responsável por uma parte específica do desenvol-

vimento, correspondente a habilidade individual de cada um, que consiste na construção de dois sites, sendo eles um portal do cliente e um portal do trabalhador, um aplicativo *mobile* e um *back-end* que fornecerá os dados tanto para os sites quanto para a aplicação *mobile*.

O foco dessa monografia é apresentar o processo da criação do aplicativo *mobile*, que será a interface que o cliente empregador utilizará para pesquisar por profissionais que realizam o serviço desejado.

1.1 A aplicação Severino

1.1.1 Clientes

Os clientes dessa aplicação são de duas categorias: empregadores e prestadores de serviço.

O cliente empregador é a pessoa que necessita contratar um profissional para realizar algum serviço. Esse irá entrar na aplicação em busca de um trabalhador, pesquisará pelos serviços e então visualizará os perfis dos profissionais para que assim encontre um que lhe atenda melhor.

O cliente prestador de serviço é a pessoa que está em busca de divulgar seu trabalho e pretende ser contratada por um cliente empregador. Esse irá se cadastrar pelo portal do profissional e editar seu perfil, colocando informações relacionadas as suas experiências como trabalhador, serviços em que consegue atuar e informações para contato.

1.1.2 As dores dos clientes

A dificuldade do cliente empregador se dá quando acontece algum imprevisto no seu dia a dia ou se ele necessita que um serviço seja realizado e possui dificuldade para encontrar algum profissional para exercer tal trabalho, pois é necessário pedir indicações de amigos e familiares ou então buscar por informações em listas telefônicas, panfletos para achar alguém qualificado e de confiança para exercer o trabalho, gastando tempo para finalmente encontrar um trabalhador que se encaixa na expectativa do cliente.

Por outro lado, a dificuldade encontrada pelo prestador de serviço é a divulgação do seu trabalho para que ele tenha mais chances de ser contratado por alguém, a falta de confiança sobre a qualidade do seu serviço e sua relação interpessoal, pois não existe uma plataforma para que os clientes empregadores possam dar e ver *feedbacks* de um profissional, para gerar uma maior confiança sobre seu trabalho e sobre sua pessoa.

1.1.3 Contribuições da aplicação

A contribuição dessa aplicação visa ajudar ambas as partes que irão utilizar esse sistema, tanto os clientes empregadores quanto os prestadores de serviços cadastrados, pois irá simplificar a busca e contratação de profissionais prestadores de serviços com sua facilidade e simplicidade de uso, além de ajudar também na divulgação do trabalho dos profissionais cadastrados, aumentando a renda dos prestadores de serviços e contribuindo para seu ingresso no meio digital.

1.1.4 Mercado potencial e concorrência

Devido a aplicação proposta possuir dois tipos clientes, os clientes empregadores e os clientes prestadores de serviço, foi realizado o estudo do mercado potencial para cada um deles. Primeiramente foi analisado o mercado para os trabalhadores e segundos dados do IBGE, no ano de 2019 a porcentagem do trabalho informal no Brasil atingiu 41,6% (AGENCIABRASIL, 2019), o que representa 39.3 milhões de pessoas. Já o estudo de mercado dos clientes empregadores foi feito através de pesquisas em aplicações que possuem propostas similares com a do Severino, como por exemplo o GetNinjas, que realiza cerca de 2.8 milhões de contratos anuais, com 500 mil profissionais cadastrados em sua base, atendendo mais de 3 mil cidades (CAVALLINI, 2020), ou até mesmo a aplicação Triider, que finalizou o ano de 2020 com um crescimento de 20% na procura de seus serviços, entre eles o Marido de Aluguel, além de ter realizado mais de 35 mil serviços (POVO, 2020).

1.1.5 Motivação

Dado o mercado potencial e a concorrência, como GetNinjas e Triider, é possível identificar um ambiente propício para o novo produto Severino, visto que a proposta principal é ser um aplicativo que possua uma interface de uso mais intuitivo e simples em comparação aos demais, além disso, ambos aplicativos concorrentes mencionados possuem nomes cujas palavras têm origem em idioma estrangeiro, o que dificulta a pronúncia e o entendimento por pessoas que não tenham domínio sobre a língua, já a palavra Severino, por ser em português e devido ao significado do uso coloquial da mesma, pode proporcionar uma correlação com o produto oferecido e possibilitar uma maior propagação do aplicativo.

1.1.6 Homenagem ao grande ator Paulo Silvino

O nome desta aplicação é uma homenagem ao ator Paulo Silvino, que ganhou grande destaque com um de seus personagens famosos de humor do programa *Zorra Total*: o "Severino". Este, trabalhava como o porteiro, mas que, além de exercer as suas funções, desempenhava também outras tarefas operacionais, sendo visto como o "quebra galho"para

todo tipo de serviço. Desta forma, o nome Severino para a aplicação trará a ideia de que será possível encontrar profissionais para todo tipo de trabalho.

1.2 O desafio de desenvolver a aplicação Severino

1.2.1 Arquitetura da solução

A aplicação Severino é formada por quatro componentes: uma aplicação mobile, dois sites, um para o cliente empregador e um para o prestador de serviço, e por fim um *back-end*.

A aplicação *mobile* será voltada somente para o cliente empregador, possuindo uma interface amigável e intuitiva. Nela será possível o usuário pesquisar por diversos serviços e escolher o profissional, que mais se encaixa em suas expectativas, além se ser possível favoritar um trabalhador e ver o perfil de cada um, em que será disponibilizado informações de contato, avaliações e comentários feitos por clientes que contrataram seu serviço.

O site do cliente empregador possui a mesma funcionalidade que a aplicação *mobile*, porém ele será disponibilizado como uma página *web* responsiva, atendendo os diversos tamanhos de telas e também sendo possível acessar pro *smartphones*.

O site voltado para o trabalhador tem como objetivo o seu cadastro na aplicação, para que seus dados possam aparecer nas aplicações voltadas para o cliente empregador. Será disponibilizado um formulário simples para que o profissional possa preencher e então ser cadastrado como um prestador de serviço na aplicação Severino.

No *back-end* da aplicação é onde ficará a manipulação dos dados recebidos pelos dois sites e pela aplicativo *mobile*, salvando em um banco de dados as informações que devem ser persistidas. Além disso, é também seu papel fornecer os dados para as aplicações voltadas para os clientes empregadores, tanto os dois portais *web* quanto o *app*.

A figura 1 exemplifica a arquitetura geral da solução.

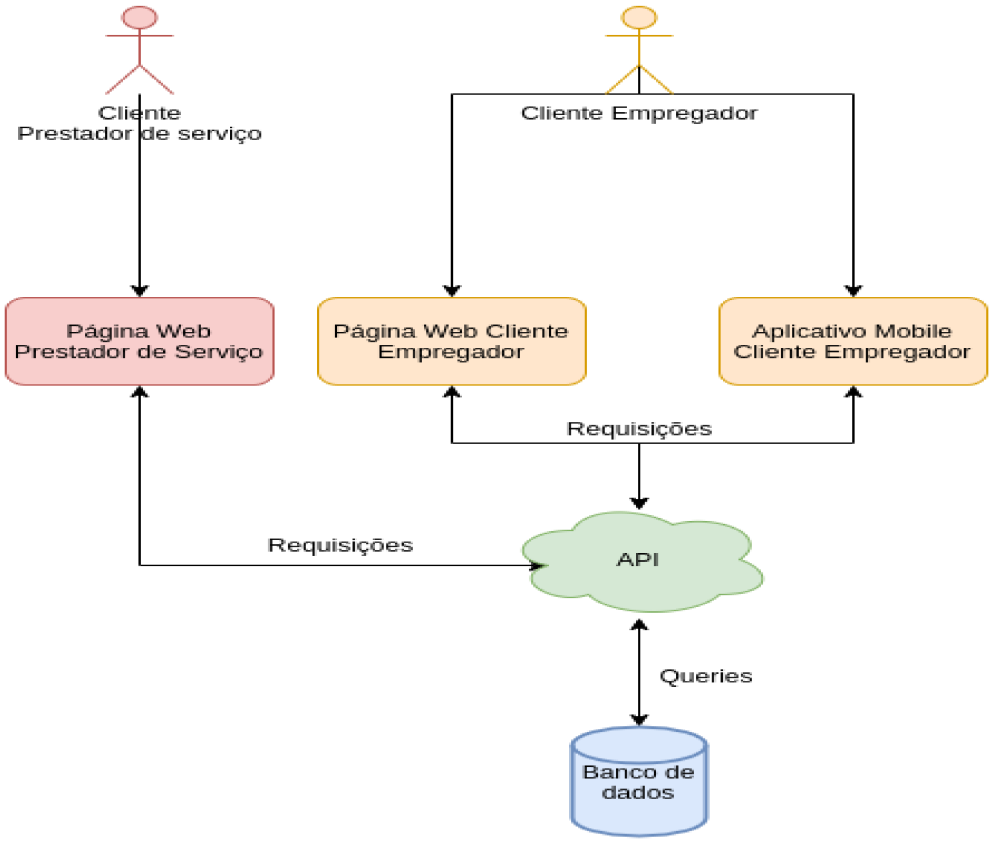

Figura 1 – Arquitetura do projeto

1.2.2 Divisão de trabalho e gestão de desenvolvimento

Esse trabalho está sendo realizado por um grupo de três integrantes sendo eles: João Daniel Aquino Rufino, Gabriel Mendes e Pedro Teixeira. Visto que todos geram uma importância crucial para o desenvolvimento do Severino, cada um ficou responsável por uma parte do projeto.

A divisão de trabalho foi feita da seguinte forma: o integrante João Daniel ficou para o desenvolvimento aplicativo mobile, já o integrante Gabriel Mendes para os dois sites, tanto o do cliente empregador quanto o do prestador de serviço, e por fim, o último integrante, Pedro Henrique, ficou responsável pelo *back-end*.

Para a gestão de desenvolvimento, foi decidido utilizar o Kanban, explicado na seção 2.1.2, por ser um método bastante utilizado em empresas que atuam na área tecnológica, além de ser um método fácil de gerenciar e objetivo.

1.2.3 Integração e testes

A integração das aplicações *front-ends* com o *back-end* será feita através de requisições HTTP utilizando o modelo de arquitetura REST explicado na seção 2.6.3. O servidor *back-end* fornecerá *endpoints* para que seja possível realizar a comunicação *front-end*/*back-end*, sendo essas feitas com métodos GET, POST, PUT e DELETE.

Para testar a aplicação, serão feitos testes unitários para cada módulo e componente da aplicação, com o fim de testá-los separadamente e garantir a acertividade de seu funcionamento, além disso, a cada funcionalidade nova, será pedido a familiares para usarem as aplicações e testarem a sua usabilidade e com os *feedbacks* obtidos a partir desta pesquisa, será feita alterações para que no final o usuário tenha a uma experiência melhor.

1.3 Organização do trabalho

Esta monografia está organizada da seguinte forma:

- O segundo capítulo contempla as metodologias de gestão aplicadas e as tecnologias utilizadas no desenvolvimento do projeto;
- O terceiro capítulo apresenta a desenvolvimento do aplicativo móvel para o tema em questão, descrevendo os pontos mais importantes da solução;
- O quarto capítulo aborda a conclusão, os desafios encontrados durante o desenvolvimento e os futuros trabalhos a serem desenvolvidos a partir desse.

Capítulo 2

Fundamentação Teórica

Neste capítulo são apresentadas diferentes tecnologias e conceitos utilizados durante a construção deste projeto. Estão listados respectivamente em metodologias de gestão de trabalho, ferramentas para o desenvolvimento de protótipos e tecnologias de desenvolvimento mobile.

2.1 Metodologias de gestão aplicadas

2.1.1 Metodologia ágil

A metodologia ágil é um conjunto de técnicas e práticas de gerenciamento de projetos que fornecem maior velocidade, eficiência e flexibilidade. O objetivo original era simplificar o desenvolvimento de software, mas esses métodos agora estão revolucionando o gerenciamento de negócios em todas as áreas além da área técnica. Lançado oficialmente em 2001 (AGILEMANIFESTO, 2001), foi definido quatro princípios fundamentais para seu gerenciamento, são eles: colaboração do cliente na negociação do contrato, respondendo à mudança seguindo um plano, software que trabalha sobre uma documentação completa, indivíduos e interações sobre processos e ferramentas. Dentre as várias metodologias ágeis de desenvolvimento de software, foi escolhido utilizar o Kanban nesse projeto, devido a sua facilidade e simplicidade de uso.

2.1.2 Kanban

Desenvolvido pela Toyota, o Kanban é um método de gerenciamento de fluxo de trabalho. O principal objetivo é organizar visualmente o seu trabalho, maximizar a eficiência e melhorar a continuidade das atividades. Esse sistema funciona através de um quadro com colunas e cartões coloridos, onde as colunas representam os produtos ou os status do produto e as cores representam o nível de urgência para a produção de cada produto.

A primeira coluna do quadro é o *backlog*. Nele estão indicadas as principais ideias sobre o produto, para que elas possam ser refinadas e movidas para a próxima coluna do

quadro. A etapa seguinte e denominada de *to do*, onde as tarefas mais urgentes para fazer, e que já passaram pelo processo de refinamento, feito na etapada de *backlog*. Quando está realizando uma atividade, ela deve ficar na coluna de *doing*, e quando finalizar, deve ficar na coluna de *done*.

Para uma maior adequação com a realidade enfrentada no projeto, é possível criar colunas adicionais, como uma coluna de *blocked*, onde uma atividade está bloqueada por algum fator externo. Também é comum ter colunas de *waiting deploy*, *testing*, e entre outras (KANBANIZE, 2021).

Neste projeto foi utilizado o trello como ferramenta ágil de fluxo de trabalho, ele se baseia em um quadro kanban para disponibilizar as atividades a serem feitas no seu projeto.

2.2 Conceitos e Ferramentas de Criação de Protótipos

Prototipagem é a arte de transferir ideias do âmbito conceitual para a realidade. É todo e qualquer objeto físico ou virtual que simula uma interação para validar uma ideia. As equipes constroem protótipos de vários graus de fidelidade para capturar conceitos de design e testar os usuários. Com protótipos, é possível refinar e validar seus projetos para que sua marca possa lançar os produtos certos.

O trabalho na fase de Prototipagem consiste na produção de uma versão inicial reduzida e de baixo custo de um produto, por meio dos chamados protótipos. Seu objetivo é revelar problemas de design, usabilidade ou adequação. Deve-se considerar a prototipagem desde o início, se adequando aos *feedbacks* obtidos dos usuários para ajudar a orientar o desenvolvimento (FOUNDATION, 2019). Na figura 2 temos um exemplo de como funciona a prototipação de telas.

As vantagens da prototipagem são:

1. Reduz o *time-to-market* de soluções.

2. Permite que a solução chegue ao mercado validada por potenciais consumidores, reduzindo custos de posteriores ajustes.

3. Incentiva a equipe a tangibilizar conceitos, auxiliando na defesa de uma ideia de negócio para instâncias superiores.

4. Poder adaptar as mudanças antecipadamente, evitando assim o compromisso com uma versão única e falsamente ideal, ficando preso em máximos locais de UX e, posteriormente, causando custos pesados devido a descuidos.

5. Mostrar o protótipo aos seus usuários para que eles possam dar *feedback* para ajudar a identificar quais elementos / variantes funcionam melhor e se uma revisão é necessária.

6. Fornecer um senso de propriedade a todas as partes interessadas, promovendo, portanto, o investimento emocional no sucesso final do produto.

Figura 2 – Exemplificação da prototipagem de uma tela, retirado de (PEHAM, 2015).

Nesse projeto, foi utilizado o Figma como ferramente de prototipagem, visto que ele é um software de prototipagem colaborativa, ou seja, é possível criar designs de interface de uma forma multitarefa, permitindo que equipes multidisciplinares possam explorar o mesmo projeto juntas vendo as alterações em tempo real. Ele fornece todas as ferramentas necessárias para fase de design do projeto e também permite geração de código para transferência (ENVATOTUTS, 2021). Na seção Protótipo teremos as telas prototipadas para o aplicativo.

2.3 Usabilidade

Os profissionais definem usabilidade como o atributo de qualidade que avalia a facilidade de uso de uma interface de sistema. A usabilidade contribui para a eficácia, eficiência e satisfação em que usuários especificados alcançam objetivos específicos.

Muitas pessoas confundem usabilidade com design UX e vice-versa; no entanto, a usabilidade do aplicativo móvel é um aspecto da UX que atua no relacionamento geral entre um usuário e o produto. UX define todos os aspectos da percepção de um usuário de um aplicativo móvel, incluindo usabilidade.

A usabilidade do aplicativo móvel promove a aprendizagem. Um aplicativo bem-sucedido precisa ser intuitivo e deve levar muito pouco tempo para o usuário atingir um

certo grau de familiaridade com a interface. Se um usuário encontrar um problema, uma solução deve ser facilmente detectável. A integração de aplicativos móveis é uma maneira eficaz de orientar os usuários por meio de um aplicativo móvel, melhorar a usabilidade e se recuperar de erros (CLEARBRIDGEMOBILE, 2021).

Muitos fatores contribuem para a usabilidade de um aplicativo móvel que afetará a eficiência e eficácia geral em que um usuário conclui seus objetivos. Uma interface utilizável deve ter três resultados principais:

1. A usabilidade do aplicativo móvel torna mais fácil para o usuário se familiarizar com a interface do usuário (UI).

2. Deve ser fácil para os usuários atingirem seus objetivos ao usar o aplicativo;

3. Os aplicativos precisam estar livres de erros. Se seu aplicativo não funcionar corretamente, o resto de seus elementos de experiência do usuário estão mudos.

Jakob Nielsen descreveu dez heurísticas de usabilidade para orientar o design da interface do usuário, mas é importante observar que elas servem como uma regra geral, em vez de diretrizes de usabilidade específicas. No entanto, eles melhoram drasticamente a usabilidade de produtos móveis (NIELSEN, 1993).

1. **Visibilidade do status do sistema:** o ato de tornar visíveis os elementos e estruturas do sistema, para que o usuário tenha consciência do contexto. O sistema deve sempre manter os usuários informados sobre o que está acontecendo por meio de *feedback* apropriado dentro de um prazo razoável;

2. **Uma correspondência entre o sistema e o mundo real:** o sistema deve falar a língua dos usuários, com palavras, frases e conceitos familiares a eles. Evite usar qualquer jargão técnico ou orientado para o sistema. É vital seguir as convenções do mundo real, fazendo com que as informações apareçam em uma ordem natural e lógica;

3. **Controle e liberdade do usuário:** as vezes, os usuários podem realizar ações em um aplicativo que não pretendiam realizar. Quando um usuário comete um erro, ele precisa de uma "saída de emergência"marcada sem diálogos ou explicações demoradas. Os aplicativos móveis precisam oferecer suporte às funções desfazer e refazer;

4. **Use padrões de consistência:** os usuários nunca devem se perguntar se palavras, situações ou ações diferentes significam a mesma coisa em todo o produto. A consistência é um princípio fundamental de um design UX excelente porque reduz a confusão. Embora seja essencial usar elementos padrão de outros aplicativos, também é imperativo manter a aparência geral de seu aplicativo móvel consistente.

Este princípio inclui consistência visual, consistência funcional e consistência externa. Tipos de letra, fontes, botões e rótulos precisam ser uniformes; os elementos interativos devem funcionar de maneira previsível em todas as partes do seu aplicativo e as decisões de design devem ser consistentes em vários produtos. Ao manter a consistência, os usuários podem aplicar o conhecimento existente ao usar seu aplicativo móvel;

5. **Prevenção de erros:** ainda melhor do que boas mensagens de erro é um design cuidadoso que evita a ocorrência de um problema. Elimine as condições propensas a erros ou verifique-as e apresente aos usuários uma opção de confirmação antes de se comprometer com uma ação;

6. **Reconhecimento em vez de lembrar:** minimize a carga cognitiva do usuário tornando objetos, ações e opções visíveis. O usuário não deve ter que se lembrar de informações de uma parte do aplicativo para outra. As instruções de uso do sistema devem ser visíveis ou facilmente acessíveis sempre que apropriado.

7. **Flexibilidade e eficiência de uso:** aceleradores - invisíveis para o usuário novato - podem frequentemente acelerar a interação do usuário experiente de forma que o sistema possa atender tanto a usuários inexperientes quanto experientes. Permitir que os usuários personalizem ações frequentes;

8. **Design Estético e Minimalista:** as telas não devem conter informações irrelevantes ou raramente necessárias. Cada unidade extra de informação em uma tela compete com as unidades relevantes de informação e diminui sua visibilidade relativa. A desordem é um dos piores inibidores de um bom design. Organizar a interface do usuário do seu aplicativo móvel é uma maneira eficaz de reduzir a carga cognitiva de um produto. Cada botão, imagem e ícone adicional torna a tela e o fluxo do usuário do produto mais complicados. É essencial eliminar tudo o que não seja necessário para o design do aplicativo móvel. É uma prática recomendada manter o conteúdo e os elementos da interface em um nível mínimo e apresentar ao usuário apenas o que ele precisa saber. Opte sempre por um design simples e intuitivo;

9. **Manipulação de erros:** O tratamento de erros tem um impacto considerável na UX. O tratamento inadequado de erros em conjunto com mensagens de erro pouco claras também pode causar frustração e levar o usuário a abandonar um aplicativo. Nunca presuma que um usuário é experiente em tecnologia o suficiente para descobrir erros. Sempre diga aos usuários o que há de errado em uma linguagem simples e concisa. Cada mensagem de erro deve indicar:

    a) O que deu errado e possivelmente o por quê;

b) Quais etapas o usuário deve seguir para corrigir o problema.

10. **Ajuda e documentação:** embora seja sempre melhor interagir com um sistema sem documentação, pode ser necessário fornecer ajuda e documentação. Essas informações devem ser acessíveis para pesquisa, focadas em tarefas específicas, listar etapas concretas a serem realizadas e não ser muito extensas.

Com os princípios heurísticos de (NIELSEN, 1993) em mente, a seguir está uma lista de sete práticas recomendadas para superar problemas comuns de usabilidade de aplicativos móveis (CLEARBRIDGEMOBILE, 2021).

1. **Compatibilidade de plataforma:** Uma frustração comum para usuários móveis é não ter um aplicativo que funcione para seu modelo específico de smartphone. Android e iOS, por exemplo, são duas plataformas drasticamente diferentes. Você não pode simplesmente clonar um aplicativo iOS para Android e vice-versa. Cada sistema operacional adere a considerações de programação, design e interface totalmente diferentes. Por exemplo, a navegação para cada plataforma difere drasticamente. Com isso em mente, você deve criar seu aplicativo para ter uma sensação nativa, para que seus usuários possam interagir de forma intuitiva. É uma boa ideia fazer um orçamento para otimizar seu aplicativo para os *smartphones* Android e iOS mais comuns;

2. **Forneça valor imediatamente:** Se você deseja que novos usuários voltem ao seu aplicativo, você precisa se certificar de que eles descobrem o valor logo no início, de preferência durante o processo de integração. Se você não convencer os usuários a ficar na primeira semana, é provável que os perca para sempre. Milhões de aplicativos saturam o mercado, todos competindo pela atenção do usuário, por isso é fundamental garantir que você ofereça valor imediato. Uma ótima integração do usuário não apenas reduz as taxas de abandono, mas também pode ajudar a impulsionar as métricas de sucesso de longo prazo, como retenção de usuário e valor de vida útil do usuário;

3. **Navegação Simples:** Um dos principais problemas que os usuários enfrentam ao usar aplicativos móveis é a navegação ruim. Quando os usuários fazem o download de seu aplicativo pela primeira vez, eles precisam entender claramente como navegar para cumprir sua meta, seja agendar um compromisso, comprar um produto ou encontrar informações. Sua navegação deve ter o mínimo de barreiras possível. Muitos aplicativos incluem recursos exclusivos, mas lutam para encaixá-los e fazer sentido para o usuário. A navegação deve ser compreensível para o usuário para que ele não acabe perdido em uma tela aleatória. Ao focar na usabilidade, certifique-se de que seus produtos ou serviços sejam fáceis de encontrar. Quanto mais complicado for para os usuários navegar no aplicativo, maior será a taxa de abandono do usuário;

4. **Conteúdo claro e conciso:** É de conhecimento comum simplificar o conteúdo ao projetar para dispositivos móveis. No entanto, você precisa apenas de conteúdo suficiente, que é essencial para o usuário cumprir uma meta. Essa prática é particularmente verdadeira para um processo de compra. Os consumidores ainda precisam de informações completas para fazer sua compra, e a retenção de informações básicas resultará em uma taxa de conversão menor. Você deve adaptar seu conteúdo para celular, em vez de copiá-lo literalmente da web. Incluir muitas informações em seu aplicativo móvel, sem dúvida, resultará em uma UX pobre, com usuários frustrados procurando por conteúdo específico. Torne o mais fácil possível consumir seu conteúdo com o mínimo de beliscões e zoom, apresentando as informações de forma clara e concisa;

5. **Minimize o número de etapas:** Sempre que seu usuário precisar concluir uma ação, verifique se há uma maneira mais direta de tornar a experiência dele mais natural. Considere cada ação que você exige de seu usuário como uma barreira adicional. Quanto menos etapas você incluir, mais perto seus usuários estarão de concluir suas metas. A desordem é um dos piores inibidores de um bom design. Organizar a interface do usuário do seu aplicativo móvel é uma maneira eficaz de reduzir a carga cognitiva de um produto. Cada botão, imagem e ícone adicional torna a tela e o fluxo do usuário do produto mais complicados. É essencial eliminar tudo o que não seja necessário para o design do aplicativo móvel. É uma prática recomendada manter o conteúdo e os elementos da interface em um nível mínimo e apresentar ao usuário apenas o que ele precisa saber. Opte sempre por um design simples e intuitivo.Para aplicativos de comércio eletrônico ou de varejo, em particular, fazer o check-out pode ser frustrante. Você deve digitar seu endereço, endereço de e-mail e confirmar que selecionou o produto certo usando uma tela pequena. Uma abordagem prática é simplificar a criação de uma conta com um *login* do Facebook. Projetar seu aplicativo com uma opção conveniente de check-out como convidado também incentivará os usuários a fazer mais compras em menos tempo. Outros recursos notáveis a serem considerados durante o desenvolvimento são o preenchimento automático e botões grandes de checkout. Esses elementos de design resultarão em um caminho perfeito para a compra e satisfação geral com a experiência do usuário;

6. **Reduzir a rolagem:** A priorização de conteúdo também contribui significativamente para a usabilidade de um aplicativo. Os usuários devem obter a maioria - senão todas - as informações de que precisam para decidir dentro dos limites de sua tela. Rolar para baixo às vezes é inevitável e necessário em alguns casos, mas sempre tente evitar o deslocamento lateral. Se um usuário precisa rolar para o lado, normalmente oculta um conteúdo valioso;

7. **Considere a orientação da paisagem:** Ao desenvolver um aplicativo móvel, muitas pessoas não consideram a orientação paisagem necessária. Um bom aplicativo móvel deve ser projetado para retrato e paisagem para acomodar usabilidade e UX ideais, especialmente para um aplicativo que contém conteúdo de vídeo.

Após seguir essas regras a coisa mais importante que deve ser feita é a aplicação de testes para validar a usabilidade deve-se usar plataformas de teste A / B móveis. O teste A / B permite comparar duas ou mais variações de um design ou *layout* de aplicativo específico. Por exemplo, é possível testar a eficácia dos botões e como eles diferem na geração de conversões. Em vez de adivinhar o que os usuários preferem em seu aplicativo móvel, é necessário utilizar testes para validar essas suposições.

E por fim, sucesso de um aplicativo móvel depende de apenas uma coisa principal: como os usuários percebem o produto. A usabilidade contribui diretamente para a forma como o usuário se sente em relação ao seu aplicativo, pois considera a facilidade de uso, a percepção do valor, a utilidade e a eficiência da experiência geral. Usabilidade é o que ajudará a converter usuários em clientes leais e de longo prazo, por sua vez, gerando mais receita para seu aplicativo.

2.4 Aplicação Mobile

Um aplicativo móvel, mais comumente referido como um aplicativo, é um tipo de software projetado para ser executado em um dispositivo móvel, como um *smartphone* ou *tablet*. Os aplicativos móveis frequentemente servem para fornecer aos usuários serviços semelhantes aos acessados em computadores. Os aplicativos são geralmente unidades de software pequenas e individuais com funções limitadas. Esse uso de software de aplicativo foi originalmente popularizado pela *Apple Inc.* e sua *App Store*, que oferece milhares de aplicativos para *iPhone*, *iPad* e *iPod Touch* (TECHOPEDIA, 2016).

Os aplicativos móveis estão se distanciando dos sistemas de software integrados geralmente encontrados em computadores. Em vez disso, cada aplicativo fornece funcionalidade limitada e isolada, como um jogo, calculadora ou navegação na *web* móvel. Embora os aplicativos possam ter evitado a multitarefa devido aos recursos limitados de *hardware* dos primeiros dispositivos móveis, sua especificidade agora faz parte de seu desejo, pois permitem que os consumidores escolham a dedo o que seus dispositivos são capazes de fazer.

Os aplicativos móveis mais simples pegam os aplicativos baseados em computador e os transportam para um dispositivo móvel. À medida que os aplicativos móveis se tornam mais robustos, essa técnica fica um pouco ausente. Uma abordagem mais sofisticada envolve o desenvolvimento específico para o ambiente móvel, aproveitando suas limitações e vantagens. Por exemplo, aplicativos que usam recursos baseados em localização são

inerentemente construídos a partir do zero, visando a mobilidade, uma vez que o usuário não está preso a um local, como no computador, este é um caso encontrado nesse projeto.

Os aplicativos *mobile* são divididos em duas grandes categorias: aplicativos nativos e aplicativos híbridos. Os aplicativos nativos são desenvolvidos para um sistema operacional móvel específico, geralmente iOS ou Android, desfrutando de um melhor desempenho e de uma interface de usuário (IU) mais ajustada e, geralmente, precisam passar por um processo de desenvolvimento e garantia de qualidade muito mais rigoroso antes de serem lançados (TECHOPEDIA, 2016). Já os aplicativos híbridos, possuem apenas um desenvolvimento tanto para iOS quanto para Android, com pequenas customizações de algumas particularidades de uma plataforma para outra, porém, seu desempenho pode ser inferior em comparação ao código nativo.

2.5 Ferramentas e Tecnologias de Desenvolvimento

2.5.1 Git e GitHub

Git é um sistema de controle de versão distribuído usado principalmente para desenvolvimento de software, mas pode ser usado para registrar o histórico de alterações para todos os tipos de arquivos. O Git foi inicialmente projetado e desenvolvido por Linus Torvalds em 2005 para o desenvolvimento do kernel Linux, mas foi adotado por muitos outros projetos. Além de ser distribuído, o Git foi projetado com desempenho, segurança e flexibilidade (ATLASSIAN, 2021).

GitHub é uma plataforma de hospedagem de código para controle de versão e colaboração, que utiliza Git como base e permite que você e outras pessoas trabalhe em conjunto de qualquer lugar. Tendo amplo controle do código e sabendo onde cada alteração foi feita e por quem foi realizada (GUIDES, 2021).

Neste projeto, a ferramenta foi utilizada para facilitar o armazenamento do código e auxiliar no controle de novas versões realizada durante todo o processo de construção.

2.5.2 Yarn

O Yarn é um gerenciador de pacotes para aplicar comandos prontos ao código de uma aplicação. Por ser uma ferramenta de código aberto, há uma comunidade de colaboradores experientes e qualificados que, continuamente, contribuem com novas adições de códigos, gerando pacotes variados. Assim, é possível utilizá-los nas mais diversas possibilidades. Armazena em cache cada pacote que baixa para que nunca precise baixá-lo novamente. Ele também paraleliza as operações para maximizar a utilização de recursos, de forma que os tempos de instalação sejam mais rápidos do que nunca. Usando uma soma de verificação para verificar a integridade de cada pacote instalado antes da execução do código. O Yarn usa um formato de arquivo de bloqueio detalhado, mas conciso, e um

algoritmo de instalação determinístico para garantir que uma instalação realizada em um sistema se comporte como qualquer outro sistema. (YARN, 2021).

Há outros gerenciadores de pacotes comumentes mais utilizados, porém o Yarn possui maior desempenho e por este motivo foi utilizado neste projeto.

2.5.3 React Native

React Native oferece uma maneira de construir aplicativos móveis usando React e JavaScript. Em vez da primitiva *span*, que temos na web, React Native oferece a primitiva *Text*. Se estivermos construindo um aplicativo iOS, o React Native garantirá que o Text resulte em um UIView iOS nativo contendo o texto.

Se estivermos construindo um aplicativo Android, isso resultará em um TextView nativo. Isto é muito importante. Embora estejamos construindo nosso aplicativo usando JavaScript, não obtemos um aplicativo da web embutido no shell de um móvel. O resultado é um aplicativo iOS ou Android nativo real.

Existem dois *threads* importantes em execução em cada aplicativo React Native. Um deles é o *thread* principal, que também é executado em cada aplicativo nativo padrão. Ele lida com a exibição dos elementos da interface do usuário e processa os gestos do usuário. A outra é específica para React Native. Sua tarefa é executar o código JavaScript em um mecanismo JavaScript separado. O JavaScript lida com a lógica de negócios do aplicativo. Ele também define a estrutura e as funcionalidades da interface do usuário (EVKOSKI, 2017).

Esses duas *threads* nunca se comunicam diretamente e nunca se bloqueiam. Na imagem 3 temos uma representação de como as *threads* são.

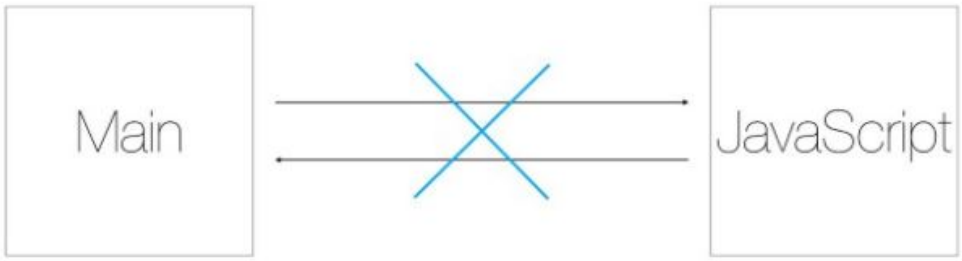

Figura 3 – Representação das *threads* do React Native.

Entre as *threads* está a chamada ponte, que é o núcleo do React Native. A ponte possui três características importantes.

1. **Assíncrono:** Ele permite a comunicação assíncrona entre os *threads*. Isso garante que eles nunca se bloqueiem.

2. **Em lote:** Ele transfere mensagens de um tópico para outro de forma otimizada.

3. **Serializável:** Os dois *threads* nunca compartilham ou operam com os mesmos dados. Em vez disso, eles trocam mensagens serializadas.

Na figura 4 temos a representação da ponte e como termina é feita a arquitetura do React Native.

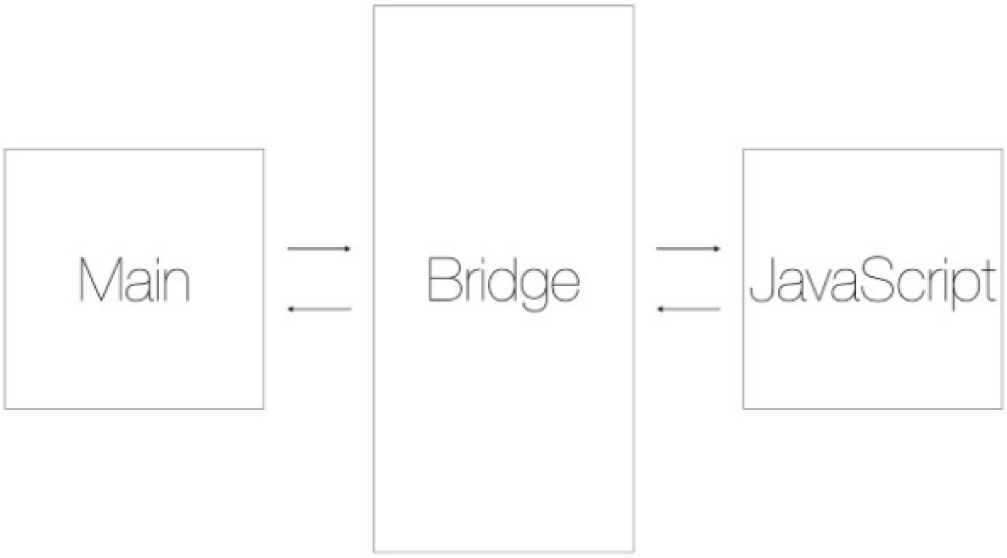

Figura 4 – Representação da *arquitetura* do React Native

Além disso, é possível utilizar a sintaxe chamada JSX, que permite escrever HTML dentro do React Native, deixando a escrita de um componente mais amigável.

2.5.4 StyleSheet

Um StyleSheet é uma abstração semelhante a CSS. Em vez de criar um novo objeto de estilo todas as vezes, StyleSheet ajuda a criar objetos de estilo com um *ID* (identificador) que é usado posteriormente para fazer referência, em vez de renderizá-lo novamente. Mover o código para fora do *render()* ajuda a obter um melhor entendimento do código e adiciona significado aos componentes de baixo nível. A folha de estilo é enviada apenas uma vez pela ponte, ao contrário do objeto de estilo normal dentro de *render()* (DUA, 2018).

2.5.5 Expo

Expo é um *framework* e uma plataforma para aplicações React universais. É um conjunto de ferramentas e serviços criados em torno de plataformas React Native e nativas que ajudam a desenvolver, construir, implantar e iterar rapidamente em aplicativos iOS,

Android e web a partir da mesma base de código JavaScript / TypeScript (EXPO, 2018).

Vantagens de se usar o expo:

- ❏ Instalação rápida e simples do projeto (em apenas alguns minutos);
- ❏ Um utilitário Expo CLI conveniente que abre em um navegador e ajuda a verificar o status do aplicativo, dispositivos em que ele é executado, escanear o código QR ou enviar o link por e-mail para abrir o aplicativo no cliente Expo, alternar o modo de produção / desenvolvimento e publicar seu aplicativo no servidor Expo;
- ❏ O cliente Expo é um aplicativo instalado a partir do Google Play e da Apple Store em seu telefone. Ele permite abrir projetos durante o desenvolvimento sem compilar via XCode ou Android Studio. Com o cliente Expo, você pode enviar seu aplicativo a outras pessoas para revisão, o que é muito útil durante o teste, pois você pode ver todas as alterações no código no cliente Expo sem criar arquivos apk ou ipa;
- ❏ Expo SDK oferece um conjunto de soluções prontas, como trabalhar com acelerômetro do aparelho, câmera, notificações, geolocalização, etc. Você pode vê-las na seção Referência API do SDK. A equipe da Expo atualiza regularmente o SDK com novas soluções;
- ❏ Over the Air - recurso Expo muito útil para atualizar seu aplicativo pelo ar sem implantação repetida no Google Play ou Apple Store. Quando os usuários abrem seu aplicativo, ele atualiza automaticamente as novas alterações no código JavaScript. No entanto, itens como ícone do aplicativo, seu nome e outras configurações não são atualizados via OTA;
- ❏ Você pode desenvolver aplicativos para iOS sem macOS, apensa ter um iOS e testá-los com o cliente Expo;
- ❏ Gerenciamento automático de certificados e assinaturas de aplicativos.

Algumas desvantagens:

- ❏ Não é possível adicionar módulos nativos escritos em Objective-C, Swift, Java, Kotlin;
- ❏ Não é possível usar pacotes com idiomas nativos que exigem vinculação;
- ❏ O aplicativo tem um tamanho grande, pois é construído com todas as soluções Expo SDK, mesmo aquelas que não se usa. Um aplicativo com Hello World pesa 25 MB;
- ❏ Geralmente tudo funciona bem no cliente Expo durante o teste, mas certos problemas podem ocorrer em um aplicativo independente.

Expo foi escolhido para ajudar na implementação do *app* levando em consideração tais informações obtidas a partir deste *post* (KRUHLYK, 2019).

2.5.6 NodeJS

O Node.js é um ambiente servidor executado por uma máquina V8 JavaScript, seu código é *opensource* (código aberto). O mecanismo V8 pega o código e o converte em um código de baixo nível, tal código é executável pelo computador sem a necessidade de interpretação o que aumenta a sua eficiência.

Node.js esta presente em vários sistemas operacionais, como Windows, Linux, Mac OS X, entre outros. Com a ampliação do JavaScript para algo que poderia ser executado em uma máquina como aplicativo independente, parou-se de ser executável somente em navegadores (NODE.JS, 2021).

2.5.7 TypeScript

TypeScript é uma linguagem que se basea no JavaScript, nela é adicionado uma forma de se descrever um objeto, essa descrição é feita por meio do tipo estático. Esse tipo, faz com que a linguagem possua uma melhor documentação e validação do código, acarretando em um melhor desempenho (TYPESCRIPTLANG, 2021). A inferência do tipo permite obter muito poder sem escrever código adicional, porém é opcional.

2.5.8 Axios

Para se fazer a conexão com o *back-end* é preciso usar uma ferramenta. O Axios faz exatamente isso, ele é um cliente HTTP que funciona tanto em ambiente Node.js quanto em navegadores, fornecendo uma API única para se lidar com as chamadas HTTP. O Axios é uma das maneiras mais simples de se fazer uma conexão HTTP, dada a baixa curva de aprendizagem que é preciso para utilizá-lo, muitos projetos o aderem em algum momento.

2.5.9 Expo/vector-icons

Expo/vector-icons é uma biblioteca que disponibiliza ícones populares, utilizando importações ES6 que permitem incluir apenas os ícones que seu projeto esteja usando.

2.5.10 Yup

Inspirada na ferramenta Joi, o Yup permite a construção de esquemas JavaScript para a validação e análise de valores, mais especificamente, formulários. Com ele, é possível

definir esquemas de uma forma enxuta e extremamente expressiva, permitido modelar validações complexas e interdependentes.

Tendo como foco ser fácil de usar, ele separa as funções de análise e validação em etapas separadas. Para transformar, validar e verificar os dados, o Yup possui uma funcionalidade chamada cast(). Cada dado pode ser executado separadamente como dados desserializados de APIs confiáveis ou agrupado como validação de formulários HTML (YUP, 2021).

2.6 Arquitetura

2.6.1 Características Principais

1. Padrão de nomenclatura *CamelCase*, em que cada palavra é iniciada com maiúsculas e unidas sem espaços.

2. Uso de React Hooks ao invés de classes.

3. Tipagem de variáveis com Typescript.

4. Api que fornece os dados é RESTful e respostas sempre em JSON.

2.6.2 React Hooks

Hooks permitem o uso de estados e outros recursos do React sem escrever uma classe. Eles foram desenvolvidos para resolver alguns problemas que a estrutura de classes tinha, como a dificuldade de reutilizar a lógica entre componentes *stateful*, a complexidade dos métodos de ciclo de vida do React, o uso do *this* e a verbosidade das classes.

2.6.3 API REST

API ou *Application Programming* Interface, trata-se de um conjunto de requisições as quais permitem a comunicação de dados entre aplicações. Para isso, a API utiliza requisições HTTP responsáveis pelas operações básicas necessárias para a manipulação dos dados (SOUZA, 2020). As principais requisições são:

- ❏ POST: criar dados no servidor;
- ❏ GET: leitura de dados no host;
- ❏ DELETE: excluir as informações;
- ❏ PUT: atualizações de registros.

Representational State Transfer (REST), é um conjunto de restrições utilizadas para que as requisições HTTP atendam as diretrizes definidas na arquitetura. Basicamente, as restrições determinadas pela arquitetura REST são:

- ❏ cliente-servidor: as aplicações existentes no servidor e no cliente devem ser separadas;
- ❏ sem estado: as requisições são feitas de forma independente, ou seja, cada uma executa apenas uma determinada ação;
- ❏ cache: a API deve utilizar o cache para evitar chamadas recorrentes ao servidor;
- ❏ interface uniforme: agrupa outros quatro conceitos em que determina que os recursos devem ser identificados, a manipulação dos recursos deve ser por meio de representação, com mensagens autodescritivas e utilizar links para navegar pelo aplicativo.

Uma das vantagens de utilizar o modelo REST API é a separação entre as aplicações *front-end* e *back-end*. Isso é importante para proteger o armazenamento de dados, pois não há o tratamento de regras de negócio, ou seja, é feita apenas a troca de informações seja para recuperar dados, seja para inserir ou deletar novos registros (ERINç, 2020).

Neste projeto, todas as conexões com o *back-end* foram feitas seguindo essas técnicas para ter uma melhor performance e estabilidade, facilitando também a organização dos dados e na autenticação do usuário.

Capítulo 3

Desenvolvimento

3.1 Protótipo

Para implementar as funcionalidades propostas no aplicativo e disponibilizá-las de forma mais simples possível ao usuário, foi feito um estudo de usabilidade com o objetivo de testar e melhorar a interface proposta para *app*. No estudo previamente citado foi usado um protótipo de interface com alta fidelidade ilustrando as telas do sistema.

No desenvolvimento do protótipo foram adotadas boas práticas de *design* de interfaces para a orientação do usuário, fundamentadas por (NIELSEN, 1993) e apresentados como as 10 heurísticas de usabilidade na seção 2.3. No processo de criação da interface foram usadas algumas dessas heurísticas como guias de *design* e desenvolvimento do protótipo, sendo elas parte do produto final e podendo ser destacadas e analisadas. Cinco das heurísticas previamente citadas foram amplamente usadas na interface do sistema, sendo elas:

1. Visibilidade do status do sistema;
2. Correspondência entre o sistema e o mundo real:;
3. Controle e liberdade para o usuário;
4. Padrões de consistência;
5. *Design* Estético e Minimalista.

Para a prototipagem do projeto, usamos a ferramenta Figma apresentada na seção 2.2 para a construção das telas do aplicativo. Sendo elas:

- Telas de autenticação 5: Essas são a telas que permitem o usuário realizar a criação de sua conta, realizar o *login* após a criação da mesma e efetuar a mudança de senha se necessário.

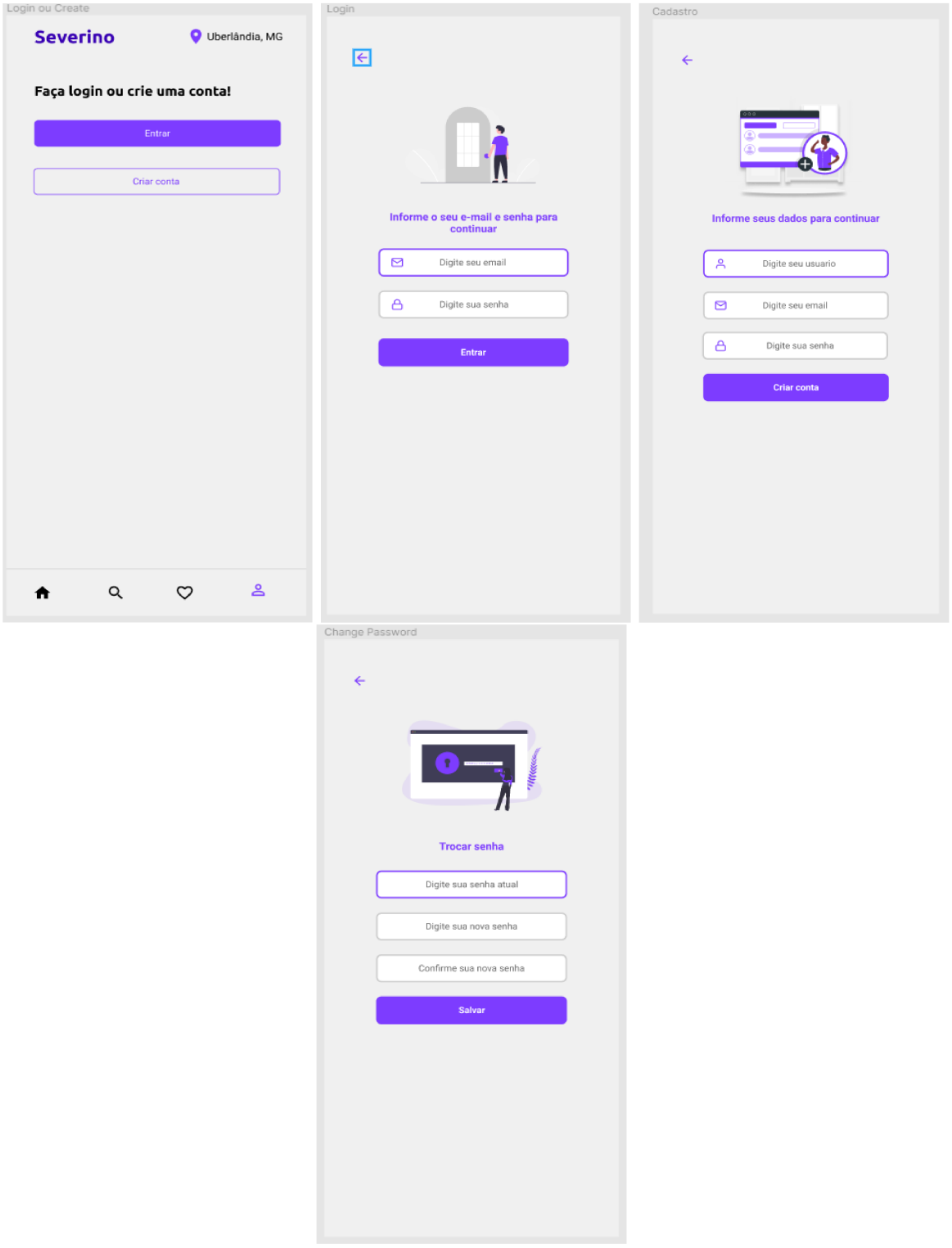

Figura 5 – Telas de autenticação referenciadas nas figuras

❑ Tela de perfil do usuário e tela de profissionais favoritados 6: A tela de perfil do usuário contempla apenas um botão que irá levar para a página de mudança de senha e um botão de *logout* da aplicação.

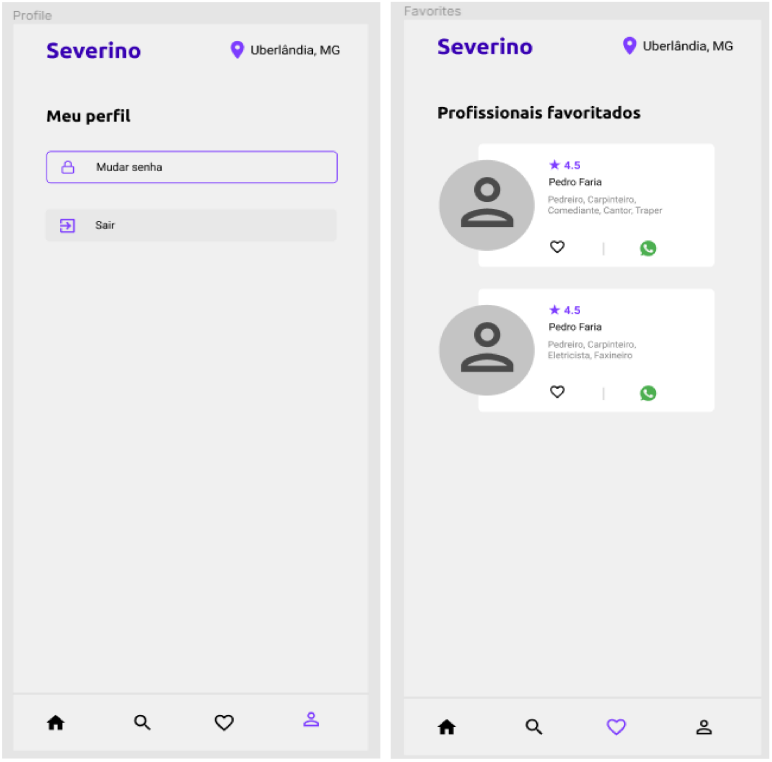

Figura 6 – Tela de perfil do usuário e tela de profissionais favoritados

❏ Telas principais 7: Nestas telas é possível que o usuário faça uma busca pelo serviço desejado além de conter *cards* com serviços existentes e profissionais disponíveis na plataforma. Ao selecionar o serviço desejado será listado todos os trabalhadores que prestam o serviço da região em que o usuário se localiza, sendo possível também fazer a filtragem por três modos diferentes.

❏ Telas dos filtros 8: Aqui temos três telas onde o usuário efetua a filtragem dos serviços, a primeira mostra um filtro contendo a definição especifica do serviço, a segunda é a filtragem por avaliação e a terceira por região.

❏ Tela de perfil do profissional referenciada na figura 9: Essa é a tela do perfil do profissional, nela contem todas as informações necessárias para que o usuário entre em contato com o profissional, o avalie e também veja as avaliações do mesmo, contendo também algumas fotos com descrições de seus trabalhos já realizados anteriormente.

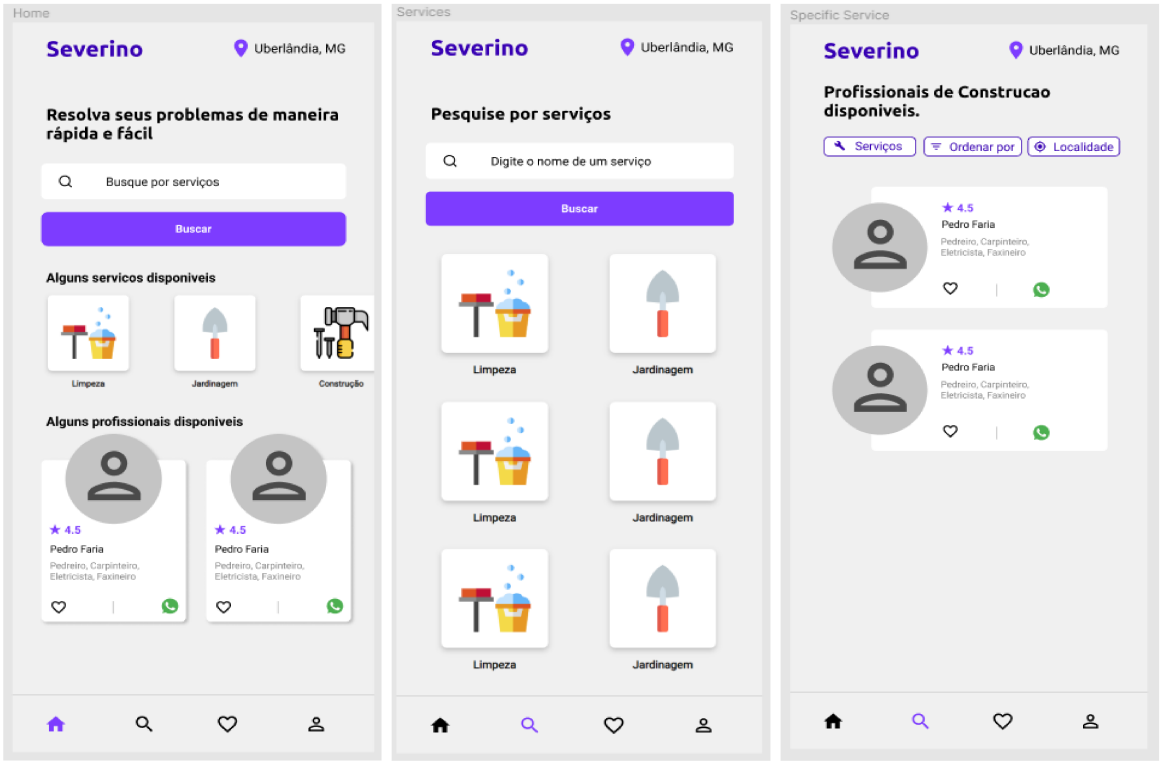

Figura 7 – Telas principais

3.2 APIs externas utilizadas

3.2.1 IBGE

Para buscar todas os estados e cidades do Brasil, foi utilizada a API de localidades do IBGE, disponível em API IBGE. Foram utilizados dois *endpoins*, um para retornar todos os estados: *api/v1/localidades/estados* e o outro para retornar as cidades do estado selecionado: */api/v1/localidades/estados/ESTADO/municipios*.

3.2.2 Big Data Cloud

Para localizar a cidade e estado em que o usuário se encontra, foi utilizada a API de geolocalização Big Data Cloud, que se basea na latitude e longitude capturadas pela ferramenta de geolocalização que o Expo fornece, disponível em: API Big Data.

3.3 Implementação

Após a prototipagem e os testes de usabilidade feitos, começou-se a implementação *app*.

Esta aplicação é a parte *front-end* do sistema, responsável pela interface visual e interação do usuário. É através dela que o usuário faz requisições à API do *back-end*

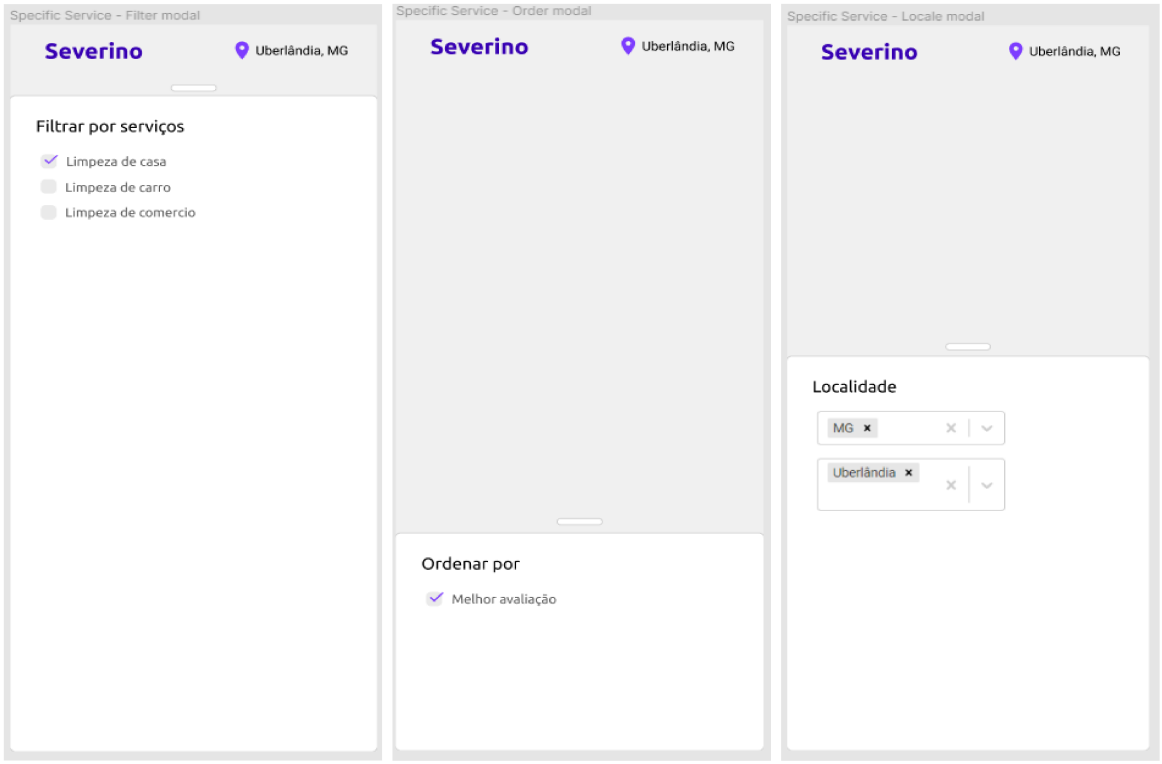

Figura 8 – Telas de filtros

desenvolvido. Esta, por sua vez, é o ponto de comunicação entre o *app* e o banco de dados, onde ocorre o armazenamento e manipulação dos dados do sistema.

A abstração que a API oferece para a aplicação *front-end* é interessante no ponto de vista em que não é necessário a implementação da comunicação com o banco de dados e nem a manipulação dos dados. Isso traz um foco em somente desenvolver o *front-end* e não se preocupar em como os dados são armazenados ou tratados, acarretando uma simplificação no código da aplicação.

Tendo em vista que a comunicação entre o *app* e a API é feita através do protocolo HTTP, na figura 10 temos o fluxograma do sistema:

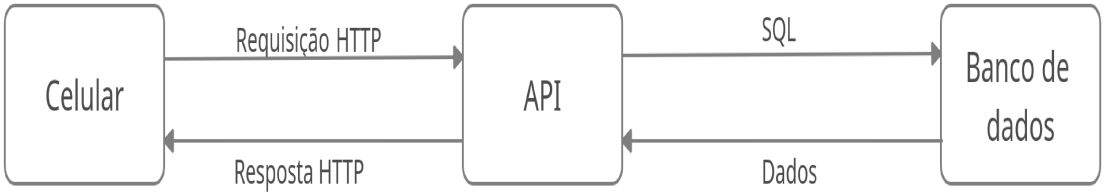

Figura 10 – Fluxograma do sistema

Para o desenvolvimento do *app*, foi utilizada a biblioteca React Native juntamente com

Figura 9 – Tela de perfil do profissional

o framework Expo. A utilização destes, permite criar apenas um código fonte tanto para o sistema iOS quanto para o Android. Além do esforço reduzido para acessar alguns recursos nativos de cada sistema como: câmera, localização, acesso aos contatos, fotos, o Expo também disponibiliza meios para simular a aplicação diretamente no dispositivo físico, gerando assim uma melhor usabilidade e facilidade nos testes, sem ter que se preocupar em ter um Mac OS para simular um telefone iOS ou algum emulador de Android.

A Figura 11 representa as telas da aplicação e as hierarquia entre elas.

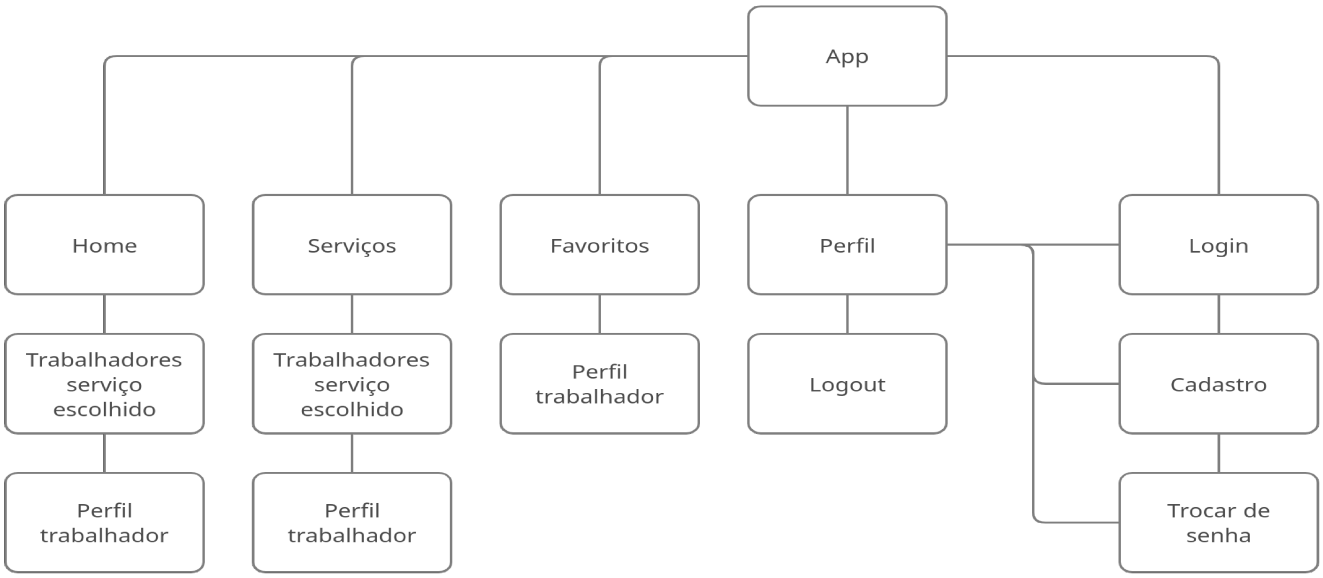

Figura 11 – Fluxograma do App

Para implementar essa estrutura, foi usado um dos principais recursos do React Native, que é a utilização de componentes. Um componente é formado por um arquivo TypeScript juntamente com o StyleSheet, para a estilização do componente. O código fonte a seguir 3.3 exemplifica um componente.

Componente React Native

```
import React from 'react';
import { StyleSheet, View, Text } from 'react-native';

const Component = () => (
    <View style={styles.container}>
        <Text style={styles.text}>Hello World</Text>
    </View>
)

const styles = StyleSheet.create({
    container: {
        flex: 1,
        alignItems: 'center',
        justifyContent: 'center'
    },
    text: {
        color: 'blue'
```

```
    }
});

export default Component;
```

Para a comunicação com a API, foi utilizado o cliente HTTP Axios. Os trechos de códigos a seguir exemplificam a criação da instância do Axios 3.3 e a utilização de métodos HTTP Post e Put, para a autenticação e mudança de senha 3.3.

Criação da instância do axios

```
import axios from 'axios';

const api = axios.create({
  baseURL: 'http://192.168.1.102:3333',
});

export default api;
```

Utilização do axios para realizar requisições HTTP para o servidor

```
import api from './api';

class Auth {
  async signIn(email: string, password: string) {
    const response = await api.post('session/customer', { email, password });
    return response.data;
  }

  async signUp(name: string, email: string, password: string) {
    const response = await api.post('customers', { name, email, password });
    return response.data;
  }

  async changePassword(id: string, oldPassword: string, newPassword: string) {
    const response = await api.put('customers', { id, oldPassword, newPassword
        });
    return response.data;
  }
}

export default new Auth();
```

Um recurso importante que o React Native possui são os contextos. Eles permitem compartilhar informações para os componentes desejados sem ter que passar manualmente para cada componente em cada nível.

O trecho de código a seguir 3.3, demonstra o contexto de autenticação do *app*, em que os dados compartilhados são: variável que identifica se o usuário está logado, dados o *login* do usuário, funções que permitem a realização de *login*, *logout* e mudança de senha e uma variável que indica o *loading* do processamento do contexto.

Contextos React Native

```
import React, { createContext, useState, useEffect } from 'react';
import AsyncStorage from '@react-native-async-storage/async-storage';

import api from '../services/api';
import Auth from '../services/Auth';

interface Customer {
  id: string;
  email: string;
  name: string;
}

interface AuthContextData {
  signed: boolean;
  customer: Customer | null;
  signIn(email: string, password: string): Promise<void>;
  signOut(): void;
  changePassword(oldPassword: string, newPassword: string): Promise<void>;
  loading: boolean;
}

const AuthContext = createContext<AuthContextData>({} as AuthContextData);

export const AuthProvider: React.FC = ({ children }) => {
  const [customer, setCustomer] = useState<Customer | null>(null);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    async function loadStorageData() {
      const customer = await AsyncStorage.getItem('@Severino:customer');
      const token = await AsyncStorage.getItem('@Severino:token');
```

```
      if (customer && token) {
        api.defaults.headers.Authorization = `Bearer ${token}`;
        setCustomer(JSON.parse(customer));
      }

      setLoading(false);
    }

    loadStorageData();
  }, []);

  async function signIn(email: string, password: string) {
    const response = await Auth.signIn(email, password);

    const { customer, token } = response;

    setCustomer(customer);

    await AsyncStorage.setItem('@Severino:customer', JSON.stringify(customer));
    await AsyncStorage.setItem('@Severino:token', token);
    api.defaults.headers.Authorization = `Bearer ${token}`;
  }

  async function signOut() {
    await AsyncStorage.clear();
    setCustomer(null);
    api.defaults.headers.Authorization = undefined;
  }

  async function changePassword(oldPassword: string, newPassword: string) {
    if (customer) {
      await Auth.changePassword(customer.id, oldPassword, newPassword);
    }
  }

  return (
    <AuthContext.Provider value={{ signed: !!customer, customer, signIn,
        signOut, changePassword, loading }}>
      {children}
    </AuthContext.Provider>
  );
};
```

```
export default AuthContext;
```

3.4 Genymotion

Para conseguir simular o aplicativo no computador, é necessário utilizar um emulador para isto. Foi utilizado o emulador de Android Genymotion, pois ele é um emulador Android completo para Windows, macOS e Linux. Pode-se usar essa ferramenta para emular uma dúzia de dispositivos Android, sendo capaz de detectar automaticamente seu teclado, *mouse* e conexão com a Internet.

Além de ser muito fácil de usar, inclui também funções avançadas para desenvolvedores e usuários experientes, incluindo um conjunto completo de sensores e recursos para interagir com um ambiente Android virtual. Com Genymotion Desktop, é possível testar seus aplicativos em uma ampla gama de dispositivos virtuais para fins de desenvolvimento, teste e demonstração. Ele é rápido, simples de instalar e poderoso graças aos *widgets* de sensor e recursos de interação fáceis de se lidar.

A escolha dessa plataforma de emulação se deu por conta de algumas vantagens que ela traz em relação as outras, sendo elas:

- Compatibilidade:
  - Compatível a toda a estrutura de teste baseada em ADB Appium, Espresso, Robotium, etc.
  - Compatível com soluções populares de integração contínua CircleCI, Bitrise, Terraform, etc.
- Performance:
  - Sem virtualização aninhada para acelerar seus testes e operações com ou sem uma GPU dedicada.
- Escalabilidade:
  - Acesso instantâneo a dispositivos virtuais ilimitados que podem ser executados simultaneamente para fragmentação de teste ou teste paralelo.
- Escolha:
  - Disponível a partir do Android 4.4 até as versões mais recentes, em todos os tamanhos de tela e em uma grande variedade de plataformas, AWS, Google, Azure e Alibaba.

E não para por ai, o Genymotion ainda oferece uma variedade de sensores singulares para que seu aplicativo fique completo, sendo eles:

- Bateria
- Rotação
- Injeção de câmera e som
- GPS
- Rede e banda base
- Registros ao vivo
- *Upload* de arquivo
- Identificadores

3.5 Seções do aplicativo

O aplicativo é dividido em seções, algo comum entre os aplicativos mais atuais, tornando mais amigável a experiência do usuário ao navegar pelas telas, visto que fica mais intuitivo de se entender que, em cada uma das diferentes seções, ele estará realizando tipos de atividades diferentes. A utilização de ícones, também facilita o entendimento do propósito de cada parte do *app*. A Figura 12 demonstra esta parte desenvolvida.

Figura 12 – Seções do *app*

Existem quatro seções diferentes e são elas:

❏ Seção Home 13

❏ Seção Busca de serviços 3.5.2

❏ Seção Profissionais favoritados 3.5.3

❏ Seção Perfil usuário 16

3.5.1 Seção Home

A seção Home é a tela principal do aplicativo. Nela já é mostrado para o usuário alguns serviços disponíveis em um listagem horizontal. O *card* do serviço é apresentado com a sua foto e seu nome e, que, ao clicar nele, o usuário será redirecionado para a tela em que apresenta os profissionais que prestam o serviço escolhido.

Também em listagem horizontal, é apresentado alguns profissionais melhores avaliados na região em que o usuário se encontra. Cada *card* do trabalhador contém uma foto do profissional, sua avaliação, seu nome, os serviços prestados, um botão para favoritar o profissional e um botão que abre automaticamente o *WhatsApp* do usuário para mandar uma mensagem para o trabalhador escolhido. Ao clicar nesse *card*, o usuário será redirecionado para a tela de perfil do trabalhador, explicado na subseção 18. A Figura 13 exemplifica a tela Home.

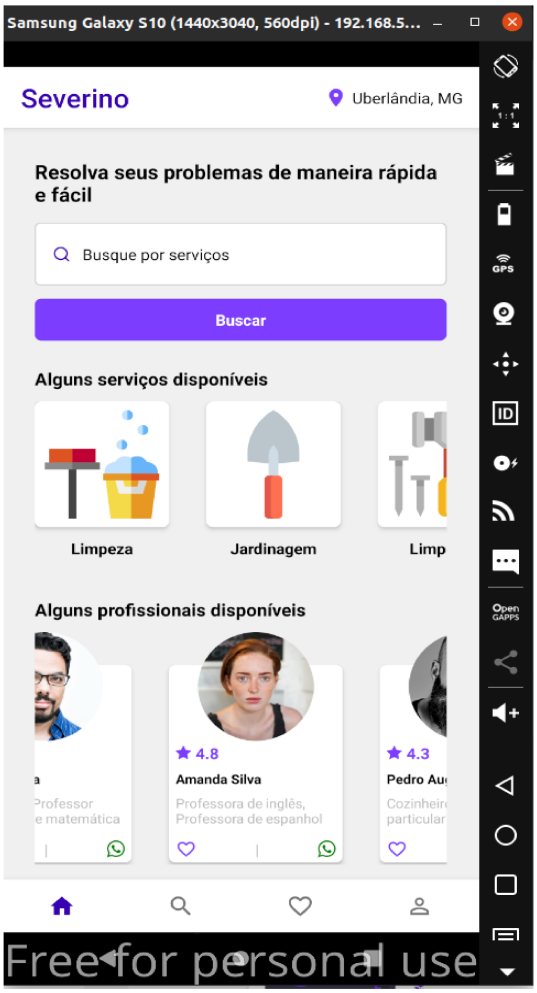

Figura 13 – Seções Home

3.5.2 Seção Busca de serviços

Nesta seção, é listado todos os serviços disponíveis. Cada *card* de serviço é composto de sua imagem e seu respectivo nome. Ao clicar no *card*, o usuário é redirecionado para a tela em que apresenta os profissionais que prestam o serviço escolhido. A Figura 14 exemplifica a tela de Busca de serviços.

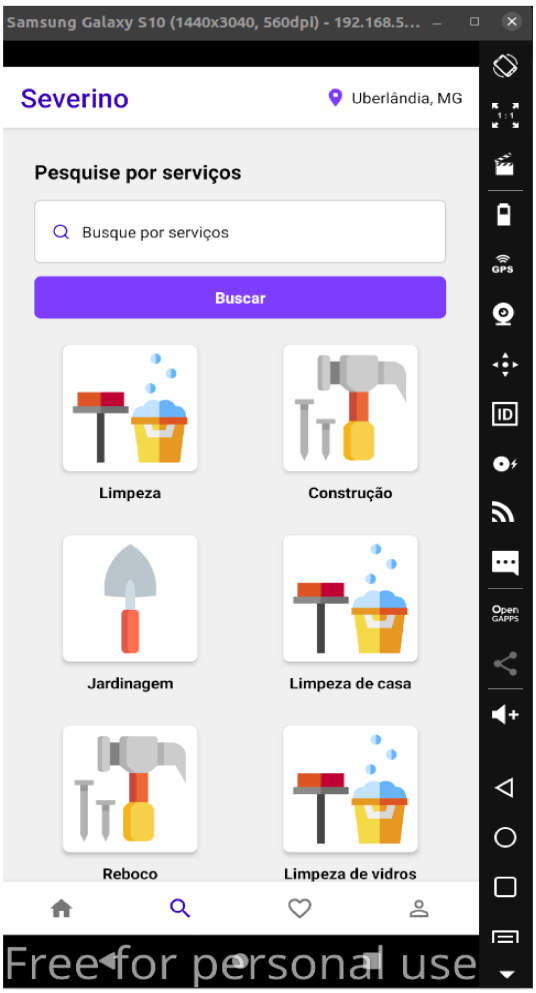

Figura 14 – Seções Busca de serviços

3.5.3 Profissionais favoritados

A seção de profissionais favoritados exibe uma lista vertical de cada trabalhador que o usuário favoritou até o momento. Cada *card* apresenta uma foto do profissional, sua avaliação, seu nome, os serviços prestados, um botão para favoritar o profissional e um botão que abre automaticamente o *WhatsApp* do usuário para mandar uma mensagem para o trabalhador escolhido. A Figura 15 exemplifica a seção profissionais favoritados.

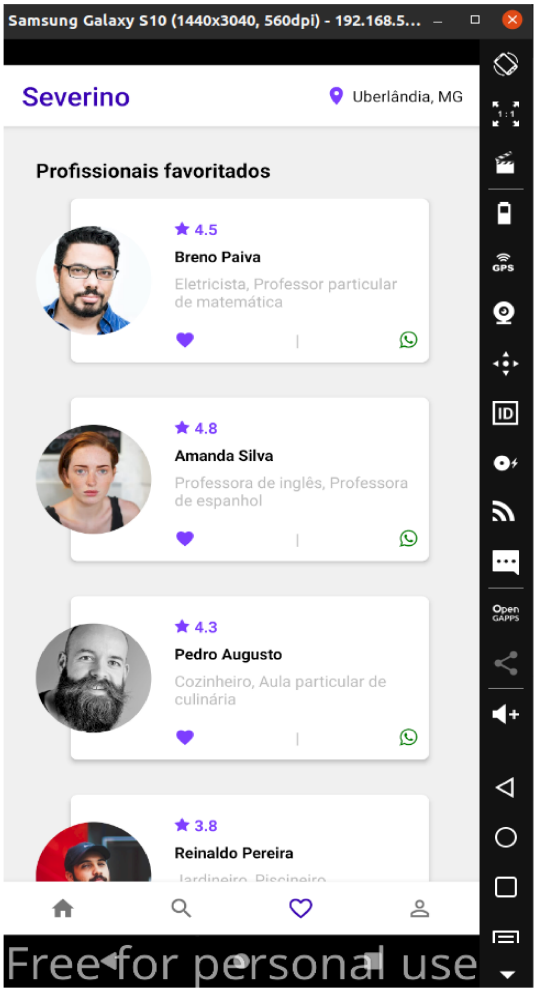

Figura 15 – Seções Profissionais favoritados

3.5.4 Seção Perfil usuário

Existem dois fluxos nessa seção. Um se o usuário não está logado e outro se ele já está autenticado no aplicativo. No primeiro fluxo, é apresentado para o usuário dois botões, um para logar no aplicativo, que redirecionará para a tela de login, e um outro botão que permite o usuário criar uma conta, entrando com seu nome, email e senha desejada. Já no segundo fluxo, existe um botão para efetuar a mudança de senha, e um outro para realizar o logout da aplicação. A Figura 16 exemplifica a seção de perfil do usuário.

3.5.5 Telas sem seção

Para contemplar todas as telas do aplicativo, criou-se esta subseção que trata as telas em que só é possível acessar por meio da navegação do usuário a partir das seções explicadas anteriormente. Dentre elas, estão as telas:

- ❏ Tela de Listagem de profissionais por serviço selecionado 3.5.5.1

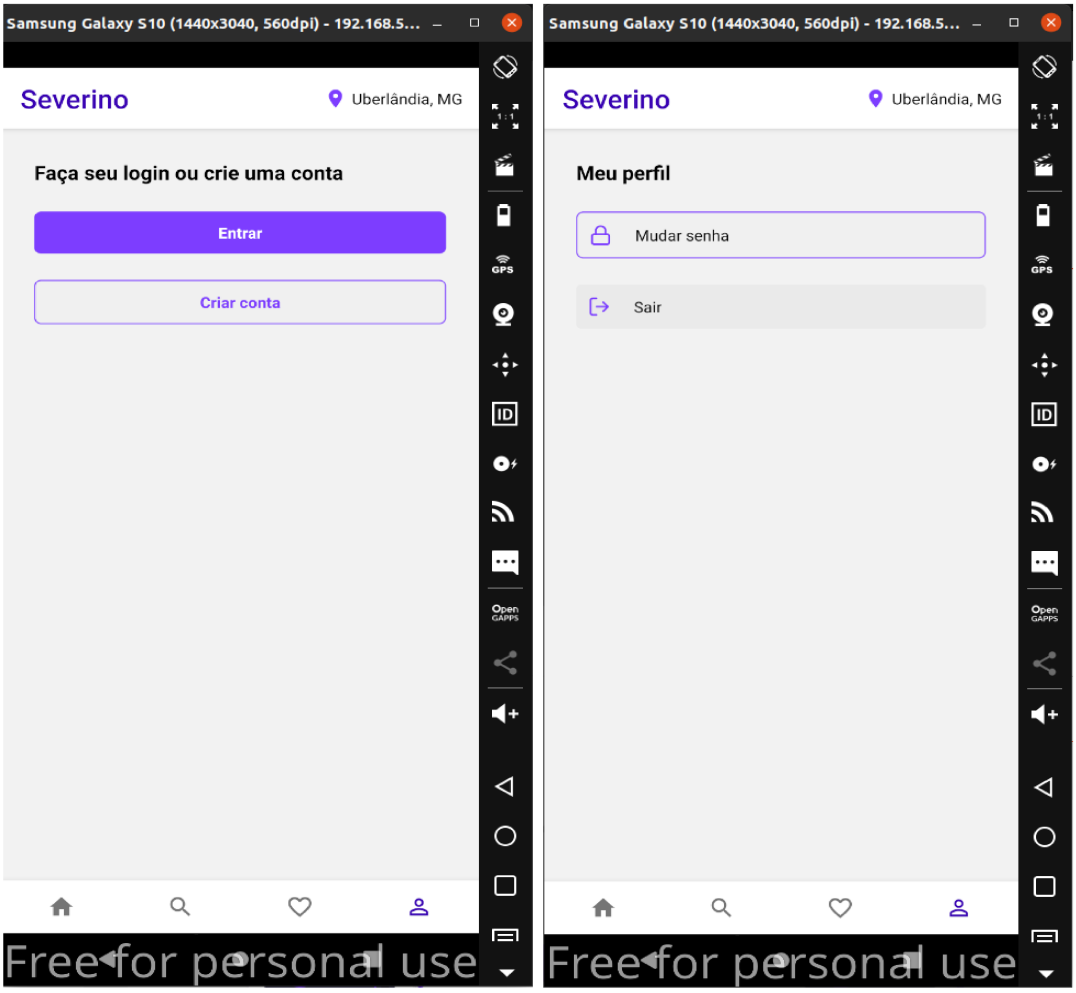

Figura 16 – Seção Perfil usuário

- ❏ Tela de Perfil do trabalhador 3.5.5.2

3.5.5.1 Tela de Listagem de profissionais por serviço selecionado

Nesta tela, é apresentado para o usuário, em forma de *card*, todos os profissionais que prestam o serviço escolhido pelo usuário. Cada *card* do trabalhador contém uma foto do profissional, sua avaliação, seu nome, os serviços prestados, um botão para favoritar o profissional e um botão que abre automaticamente o *WhatsApp* do usuário para mandar uma mensagem para o trabalhador escolhido. Também, existem três tipos de filtros que o usuário pode escolher. São eles:

- ❏ Filtragem por serviço específico
- ❏ Ordenação por melhor avaliação
- ❏ Filtragem por localidade

Quando o usuário escolher, por exemplo, profissionais que prestam serviços de limpeza, será apresentado para ele todos os trabalhadores que realizam todos os tipos de limpezas possíveis. Por isso faz-se necessário a opção de selecionar apenas os serviços específicos desejados, como por exemplo listar apenas profissionais que prestam limpeza de vidros. Isso gera uma filtragem dos trabalhadores, facilitando a escolha do profissional.

Por padrão, a ordem que os profissionais listados nessa seção é coerente com a ordem em que foram cadastrados no banco de dados. Se o usuário quiser que os trabalhadores mais bem avaliados sejam listados primeiro, basta escolher essa opção no botão "Ordenar por".

A filtragem por localidade faz-se necessária para mostrar profissionais apenas das regiões selecionadas. Por padrão, será listado trabalhadores presentes na mesma localização compartilhada pelo usuário na execução inicial do aplicativo, mas é possível selecionar qualquer cidade do Brasil. Se não existir algum prestador de serviço na região informada, aparecerá na tela que não existem profissionais na região escolhida.

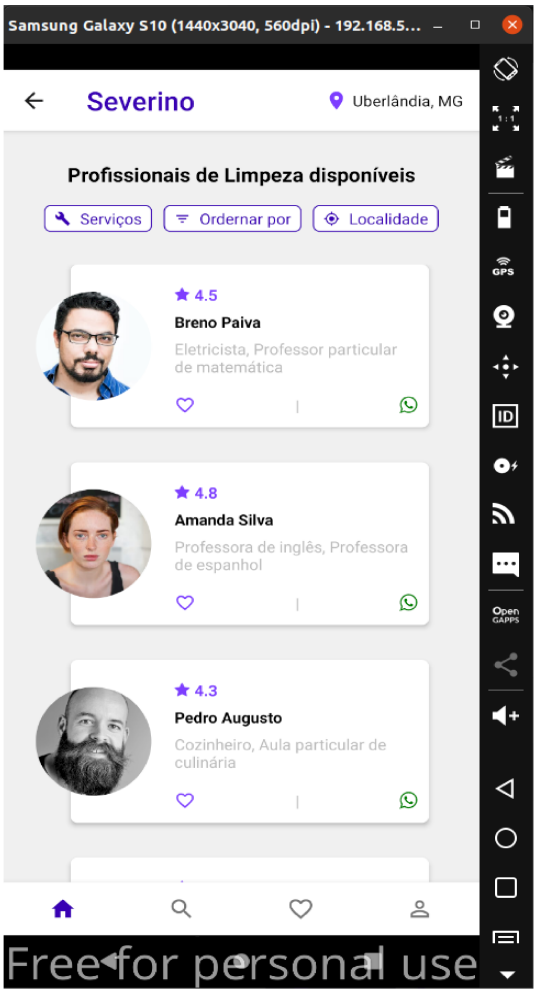

Figura 17 – Tela de Listagem de profissionais por serviço selecionado

3.5.5.2 Tela Perfil do trabalhador

A tela Perfil do trabalhador mostra para o usuário todas as informações importantes sobre o profissional escolhido. É possível chegar nessa tela ao clicar em qualquer *card* de um trabalhador. O perfil do prestador de serviço contém as seguintes informações a respeito dele: foto de perfil, média de avaliações, número de telefone, localidade do trabalhador, serviços prestados, uma descrição e uma seção para apresentar seu currículo e portfólio.

No currículo é possível conter várias informações, como: um portfólio para apresentar imagens de trabalhos realizados, uma seção para apresentar suas experiências de trabalhos, formação acadêmica e competências.

Por fim, existe uma seção em que é mostrado comentários sobre o profissional. Cada comentário existe contém a avaliação numérica, um título do comentário e o respectivo comentário. É possível ordená-los em avaliações positivas e negativas, bem como também mostrar todos.

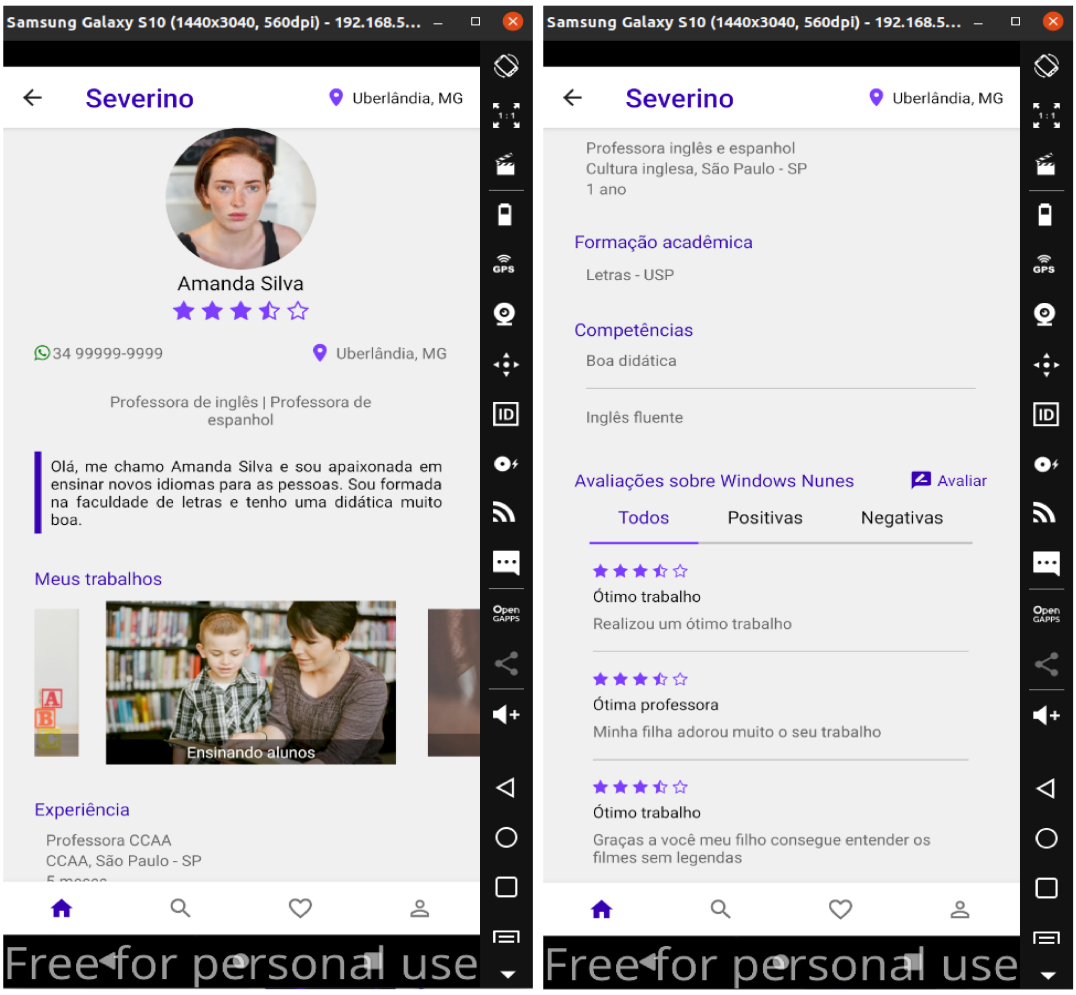

Figura 18 – Telas perfil trabalhador

Capítulo 4

Conclusão

É muito comum hoje em dia vermos diversas pessoas de diferentes faixas salariais com um *smartphone* em mãos, o acesso a tecnologia esta cada vez mais próximo dos brasileiros e ao mundo, com isso várias empresas tem focado o seus negócios voltados para aplicativos *mobile*. O Severino não é diferente, visto que o público que acessar a plataforma terão condições de navegação adequada, porque iriam preferir buscar profissionais por métodos convencionais como folhetos, agendas telefônicas, portais de informações, sendo que ele poderia ter uma plataforma simples, descomplicada e de fácil entendimento para que pudesse contratar um profissional para realizar as suas atividades em pendências à poucos toques de uma tela.

Dado isso, veio a ideia de transformar a plataforma do Severino em um aplicativo de celular, simples e direto, para que os usuários não tenham nenhuma dificuldade em manuseá-lo. Construído com telas responsíveis com técnicas de UX/UI para que se ajustem a qualquer tipo de celular, junto com uma tecnologia híbrida que atende tanto para *smartphones* com o sistema operacional Android e quanto iOS, com um código robusto e otimizado para a melhor performance da plataforma em diferentes tipos de celulares.

4.1 Desafios encontrados

Durante todo o desenvolvimento e estudo do projeto foram encontrado diversos desafios. Algumas questões iniciais foi no levantamento de requisitos e usabilidade da aplicação, como por exemplo as definições das regras de negócio, pois é uma característica crucial para ser um diferencial no mercado.

Na prototipagem, foi encontrado dificuldades em fazer interfaces amigáveis e de fácil usabilidade para o usuário final, dado que não existia nenhuma experiência prévia de construção de telas que realizam a interação do usuário com o sistema. Com isso, algumas telas foram refeitas várias vezes até chegarem ao produto final, pois a falta de conhecimento em aplicar técnicas de *User Experience* e *User Interface* dificultaram a prototipagem.

Outra dificuldade encontrada, foi construir um código responsivo para qualquer dimensão de tela de *smartphones* disponíveis no mercado. Foi necessário emular diversos dispositivos no computador para conseguir atingir uma boa responsividade das telas, dado que o código construído utilizando React Native é híbrido, ou seja, atende tanto iOS quanto Android, foi preciso realizar configrações específicas no código para que atendessem igualmente ambas as plataformas.

4.2 Trabalhos Futuros

Este projeto de graduação representa uma versão inicial do Severino e que já pode ser utilizado em larga escala por todo o Brasil, porém, como todo *software* que busca a excelência, este contará com algumas *features* adicionais para levar um diferencial para os clientes e profissionais da plataforma, como:

- Contagem de acessos ao perfil do trabalhador, para ele saber se seu perfil está sendo notado.
- Criar um serviço para salvar *logs* de erros em produção, para que seja possível rastrear e corrigir problemas sem a necessidade de relatórios por parte dos usuários.
- Criar uma página para acompanhamento das denúncias e sugestões de usuários, para tornar esse acesso mais rápido, sem a necessidade de consultar o banco de dados.
- Desenvolver uma página para gerenciamento do conteúdo das duas plataformas, como imagens e textos, para que futuros colaboradores do Severino consigam editar esses conteúdos sem precisar de conhecimentos de programação.
- Desenvolver *script* para automação do *deploy* no AWS S3, já que isso é feito manualmente no projeto atual.
- Caso mais desenvolvedores façam parte de projetos futuros, também há a necessidade de criar testes unitários para diminuir a chance de ocorrer *side effects* sem que sejam percebidos.
- Elaboração de um plano de negócio para avaliar possibilidade de transformar a aplicação em um produto e vendê-lo.

Referências

AGENCIABRASIL. **IBGE: imformalidade atinge 41,6% dos trabalhadores no país em 2019**. 2019. Agenciabrasil. Disponível em: <https://agenciabrasil.ebc.com.br/economia/noticia/2020-11/ibge-informalidade-atinge-416-dos-trabalhadores-no-pais-em-2019>. Acesso em: 08 jun 2021.

AGILEMANIFESTO. **Manifesto for Agile Software Development**. 2001. Agilemanifesto. Disponível em: <http://agilemanifesto.org/>. Acesso em: 21 abr 2021.

ATLASSIAN. **What is Git**. 2021. Atlassian. Disponível em: <https://www.atlassian.com/git/tutorials/what-is-git>. Acesso em: 15 abr 2021.

BRIDI, C. **Em meio à pandemia, profissionais autônomos buscam alternativas para ter renda**. 2020. CNN. Disponível em: <https://www.cnnbrasil.com.br/nacional/2020/04/01/em-meio-a-pandemia-profissionais-autonomos-buscam-alternativas-para-ter-renda>. Acesso em: 22 abr 2020.

CAVALLINI, M. **Pandemia aumenta busca por profissionais autônomos e freelancers no país; veja serviços com maior demanda**. 2020. G1. Disponível em: <https://g1.globo.com/economia/concursos-e-emprego/noticia/2020/10/25/pandemia-aumenta-busca-por-profissionais-autonomos-e-freelancers-no-pais-veja-servicos-com-maio
ghtml>. Acesso em: 22 abr 2020.

CLEARBRIDGEMOBILE. **7 Best Practices to Overcome Mobile App Usability Issues**. 2021. Clearbridgemobile.com. Disponível em: <https://clearbridgemobile.com/7-best-practices-to-overcome-mobile-app-usability-issues/#:~:text=What%20is%20Mobile%20App%20Usability,specified%20users%20achieve%20specified%20goals.> Acesso em: 08 may 2021.

DUA, Y. **https://medium.com/mindorks/everything-to-know-about-styling-in-react-native-7e30aed53ad: :text=A2018. Mediun.com. Disponível em: <https://medium.com/mindorks/everything-to-know-about-styling-in-react-native-7e30aed53ad#:~:text=A%20StyleSheet%20is%20an%20abstraction,reference%20instead%20rendering%20it%20again.> Acesso em: 10 may 2021.**

ENVATOTUTS. **What is Fisma?** 2021. Envatotuts. Disponível em: <https://webdesign.tutsplus.com/articles/what-is-figma--cms-32272>. Acesso em: 16 abr 2021.

ERINç, Y. K. **The Benefits of Going RESTful – What is REST and Why You Should Learn About It**. 2020. Freecodecamp. Disponível em: <https://www.freecodecamp.org/news/benefits-of-rest/>. Acesso em: 22 abr 2021.

EVKOSKI, B. **React Native: What it is and how it works**. 2017. Medium. Disponível em: <https://medium.com/we-talk-it/react-native-what-it-is-and-how-it-works-e2182d008f5e>. Acesso em: 05 marc 2021.

EXPO. **Introduction to Expo**. 2018. Medium. Disponível em: <https://docs.expo.io/>. Acesso em: 05 marc 2021.

FOUNDATION, I. D. **What is Prototyping?** 2019. Interaction-design.org. Disponível em: <https://www.interaction-design.org/literature/topics/prototyping>. Acesso em: 11 may 2021.

FRAGA, C. **Motoristas de aplicativos paralisam por melhores tarifas**. 2020. Extraclasse. Disponível em: <https://www.extraclasse.org.br/movimento/2021/02/motoristas-de-aplicativos-paralisam-por-melhor-remuneracao/>. Acesso em: 22 abr 2021.

GUIDES, G. **Hello World Git Hub?** 2021. GitHub Guides. Disponível em: <https://guides.github.com/activities/hello-world/#:~:text=GitHub%20is%20a%20code%20hosting,%2C%20commits%2C%20and%20Pull%20Requests.> Acesso em: 16 abr 2021.

KANBANIZE. **What is Kanban?** 2021. Kanbanize. Disponível em: <https://kanbanize.com/kanban-resources/getting-started/what-is-kanban>. Acesso em: 16 abr 2021.

KRUHLYK, Y. **Introduction to Expo**. 2019. Apiko.com. Disponível em: <https://apiko.com/blog/expo-vs-vanilla-react-native/#:~:text=Expo%20is%20a%20toolchain%20built,party%20native%20React%20Native%20components.> Acesso em: 06 marc 2021.

NIELSEN, J. **Usability Engineering**. San Francisco, CA, USA: Morgan Kaufmann: Inc., 1993.

NODE.JS. **Intro to Node.js**. 2021. Node.js. Disponível em: <https://www.w3schools.com/nodejs/nodejs_intro.asp>. Acesso em: 14 abr 2021.

PEHAM, T. **5 dicas para você melhorar os mockups e os protótipos de seu site**. 2015. Imasters.com.br. Disponível em: <https://imasters.com.br/design-ux/5-dicas-para-voce-melhorar-os-mockups-e-os-prototipos-de-seu-site>. Acesso em: 11 may 2021.

POVO, C. do. **Triider registra crescimento de 20prestadores de serviços em várias áreas**. 2020. Correio do Povo. Disponível em: <https://www.correiodopovo.com.br/blogs/planodecarreira/triider-registra-crescimento-de-20-na-procura-de-prestadores-de-servi%C3%A7os-em-v%C3%A1rias-%C3%A1reas-1.593366>. Acesso em: 22 abr 2020.

SOUZA, I. de. **Entenda o que é Rest API e a importância dele para o site da sua empresa**. 2020. Rockcontent. Disponível em: <https://rockcontent.com/br/blog/rest-api/>. Acesso em: 22 abr 2021.

TECHOPEDIA. **What is a Mobile Application**. 2016. Techopedia.com. Disponível em: <https://www.techopedia.com/definition/2953/mobile-application-mobile-app>. Acesso em: 08 may 2021.

TYPESCRIPTLANG. **What is TypeScript?** 2021. Typescriptlang. Disponível em: <https://www.typescriptlang.org/>. Acesso em: 14 abr 2021.

YARN, C. **FAST, RELIABLE, AND SECURE DEPENDENCY MANAGEMENT.** 2021. Classic Yarn. Disponível em: <https://classic.yarnpkg.com/en/>. Acesso em: 15 abr 2021.

YUP. **Framework Yup**. 2021. Yup. Disponível em: <https://github.com/jquense/yup#:~:text=Yup%20is%20a%20Java%20schema,an%20existing%20value%2C%20or%20both.&text=Yup's%20API%20is%20heavily%20inspired,as%20its%20primary%20use%2Dcase.> Acesso em: 14 abr 2021.