Severino: uma aplicação web para encontrar profissionais

Gabriel Mendes de Souza Santiago

Universidade Federal de Uberlândia
Faculdade de Computação
Trabalho de conclusão de curso em Ciência da Computação

Uberlândia
2021

Gabriel Mendes de Souza Santiago

Severino: uma aplicação web para encontrar profissionais

Trabalho de Conclusão de Curso apresentado ao Programa de Conclusão de Curso da Faculdade de Computação da Universidade Federal de Uberlândia como parte dos requisitos para a obtenção do título de Graduado em Ciência da Computação.

Área de concentração: Ciência da Computação

Orientador: Prof. Dr. Marcelo Rodrigues de Sousa

Uberlândia
2021

Gabriel Mendes de Souza Santiago

Severino: uma aplicação web para encontrar profissionais

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.

Trabalho aprovado. Uberlândia, Brasil, 25 de junho de 2021

_____

Prof. Dr. Marcelo Rodrigues de Sousa

Orientador

_____

Prof. Dr. Igor Santos Peretta

_____

Prof. Dr. Márcio José da Cunha

Uberlândia
2021

Dedico este trabalho aos meus pais e irmãos, que sempre me incentivaram e me deram todas as condições para ter bons estudos. Dedico também aos meus amigos que tornam meus dias mais alegres.

Agradecimentos

Gostaria de agradecer imensamente à minha família que me apoiou muito e me incentivou durante a faculdade, mesmo quando eu pensei em desistir e iniciar outro curso, me dando forças e condições para continuar.

Agradeço também aos meus grandes amigos Pedro H. Faria e João D. Aquino pela contribuição na ideia e no desenvolvimento do Severino, além de tornar meus dias na faculdade mais alegres.

Por fim, gostaria de agradecer ao meu melhor amigo Yan, aos seus pais, que também são grandes amigos, Nikson e Kelma e à minha namorada Ysla por todo o apoio emocional que me deram durante esta jornada.

"Nós poderíamos ser melhores se não quiséssemos ser tão bons"
(Sigmund Freud)

Resumo

Com o fim das listas telefônicas para a busca de profissionais e o inicio de uma era digital, surgiram alguns sistemas para facilitar a procura por profissionais *online*, contudo, essas plataformas oferecem um sistema que, além de ter uma usabilidade complexa, necessitam de cadastro e do preenchimento de vários formulários, o que afasta muitas pessoas dessas aplicações. Visto isso, decidiu-se propor um novo *software*, chamado Severino, que possa ser usado de forma fácil e didática pelos clientes, que irão conseguir encontrar profissionais em sua cidade com apenas dois cliques e pelos profissionais, que irão conseguir se cadastrar na plataforma com um pequeno formulário.

Para a realização da aplicação Severino, foram utilizadas as tecnologias de *front-end* no Estado da Arte, tendo o React como biblioteca *core* da aplicação.

Palavras-chave: Severino, Contratação de Serviços, Web, Sistema Front-end, React.

Abstract

With the end of telephone directories to search for professionals and the beginning of a digital age, some systems have emerged to facilitate the search for professionals online, however, these platforms offer a system that, in addition to having complex usability, require registration and filling out various forms, which drives many people away from these applications. In view of this, it was decided to propose a new software, called Severino, which can be used in an easy and didactic way by clients, who will be able to find professionals in their city with just two clicks, and by professionals, who will be able to register on the platform with a small form.

For the realization of the Severino application, the front-end technologies in the State of the Art were used, with React as the core library of the application.

Keywords: Severino, Service Contracting, Web, Front-end System, React.

Lista de ilustrações

Lista de siglas

API Application Programming Interface

AWS Amazon Web Services

CSS Cascading Style Sheets

DVCS Distributed Version Control System

DDD Discagem direta a distância

DOM Document Object Model

ES ECMAScript

HTML HyperText Markup Language

HTTP Hypertext Transfer Protocol

HTTPS Hypertext Transfer Protocol Secure

IBGE Instituto Brasileiro de Geografia e Estatistica

JSX JavaScript XML

JSON Javascript Object Notation

NPM Node Package Manage

OIT Organização Internacional do Trabalho

REST Representational state transfer

S3 Simple Storage Service

SPA Single Page Application

UX User Experience

UI User Interface

URI Identificador uniforme de recurso

Sumário

Capítulo 1

Introdução

Pessoas que oferecem serviços domésticos, como diaristas, eletricistas, jardineiros e pedreiros são comumente requisitados por diversas famílias brasileiras, dado que a mão de obra doméstica representa uma parte significativa da força de trabalho brasileiro, formal ou informal. Segundo estimativas da OIT no ano de 2013, existiam cerca de 67 milhões de trabalhadores domésticos, sendo que este é uma das ocupações com menor nível de remuneração no mundo e as médias salariais estão abaixo da metade do salário médio no mercado de trabalho (TRABALHO, 2021).

Por mais que seja um mercado muito grande e comum no Brasil e que haja muitas empresas que fornecem esses trabalhos de forma terceirizada, são poucas que fornecem uma solução tecnológica e de fácil acesso. Dado isso, surgiu a oportunidade de criar uma aplicação que possibilita a entrada desses trabalhadores domésticos no meio digital, fornecendo um mercado de trabalho mais aberto e com mais opções de clientes.

Assim, o foco principal dessa aplicação é facilitar a entrada de profissionais no meio digital e disponibilizar um *software* simples de utilizar para os clientes encontrarem esses trabalhadores. A aplicação consiste em dois portais, um para o cliente empregador, que é a pessoa que necessita contratar alguém para realizar algum serviço em sua residência ou empresa e outro para os profissionais prestadores de serviço, que será onde eles farão inscrição e criação do perfil.

1.1 Nome da Aplicação

A aplicação, em homenagem ao saudoso ator e humorista Paulo Ricardo Campos Silvino, será denominada Severino, já que ele interpretava o personagem Severino no programa Zorra Total da Rede Globo, que era conhecido como um "Faz-Tudo"que realizava diversos serviços de forma humorística, também chamado de "quebra-galho".

1.2 Motivação e Dores dos Clientes

Ao realizar pesquisas de satisfação sobre as aplicações existentes no mesmo nicho da aplicação Severino e ao conversar com amigos e parentes, constatou-se que algumas pessoas não conheciam aplicações que ajudassem os usuários a encontrarem profissionais autônomos e que passavam muito tempo buscando profissionais com boas recomendações. Além disso, algumas dessas pessoas conheciam somente o GetNinjas, porém elas reclamaram da necessidade de realizar cadastro e da obrigação de preencher formulários para enfim iniciar a busca por profissionais, que também não é feita de forma direta, ou seja, deve-se esperar que os trabalhadores entrem em contato com o cliente via *e-mail* ou WhatsApp.

Portanto, a motivação para a construção dessa aplicação foi a dificuldade de encontrar plataformas simples de utilizar e que resolvessem o problema de conectar clientes a prestadores de serviço de forma rápida e eficaz.

1.3 Segmento de Clientes

Diante do exposto anteriormente, fica evidente que os usuários da aplicação Severino são:

1. Clientes: pessoas que estão em busca de prestadores de serviços variados, como eletricistas, jardineiros, faxineiros e até mesmo professores particulares.

2. Profissionais: pessoas que prestam serviços e que querem ingressar ou se manter no ambiente digital, por meio de plataformas *online*.

1.4 Mercado e Concorrência

Visto que existem dois tipos de usuários da plataforma, foi feita uma análise de mercado primeiramente para o público de profissionais, que pode chegar a até 24,5 milhões, já que essa é a estimativa da quantidade de trabalhadores autônomos no Brasil em 2020, segundo o IBGE (BRIDI, 2020). Já para o público de clientes empregadores, é difícil estimar uma quantidade de usuários para a plataforma, já que essa estatística não é feita de forma oficial, contudo sabe-se que a demanda por trabalhadores autônomos no GetNinjas foi de 2,4 milhões entre março e setembro de 2020, que representa um aumento de 86% em relação ao mesmo período em 2019 (CAVALLINI, 2020). Além disso, o Triider, que é outra plataforma que busca conectar clientes a profissionais, também registrou um crescimento de 42% de pedidos de orçamentos no primeiro trimestre de 2021 em comparação com o mesmo período em 2020.(MAN, 2021)

Dado o constante crescimento das aplicações que ajudam clientes a encontrarem profissionais, fica evidente que existe um cenário propício para a entrada da aplicação Severino. Espera-se que essa aplicação seja bem aceito tanto pelos profissionais, quanto pelos clientes, já que terá uma usabilidade mais simples e o contato com os profissionais se dará de forma mais rápida do que o das empresas concorrentes, como o Triider, GetNinjas e o Crafty.

1.5 Organização Geral e Gestão da Solução

De forma geral, essa aplicação será desenvolvida em três partes:

1. Dois portais *web*, um para os clientes e um para os profissionais.
2. Um aplicativo *mobile* hibrido que funcionará para Android e iOS, mas que, inicialmente será somente para os clientes.
3. Um *back-end* para servir a parte *web* e a parte **mobile**.

Nessa monografia, serão demonstrados o desenvolvimento, as tecnologias utilizadas e a solução do desafio da parte *web*. O *back-end* da aplicação será desenvolvido pelo aluno Pedro Henrique Faria Teixeira em outro trabalho e o *mobile* será desenvolvido pelo aluno João Daniel Aquino Rufino, também em outro trabalho.

Como houve uma divisão entre as partes da aplicação, será utilizado o aplicativo Discord para a comunicação com os outros alunos e o Trello para a divisão das tarefas de cada um. A parte de gestão do desenvolvimento será abordada detalhadamente no próximo capítulo.

A figura 1 ilustra uma arquitetura geral da solução. Vale salientar que a parte *web* foi desenvolvida seguindo o princípio de *Desktop First*, ou seja, busca atender primeiramente os usuários de *desktops* e *notebooks*, mas também possui um *layout* responsivo, que é capaz de se adaptar perfeitamente em dispositivos *mobile*.

1.6 Objetivos e Desafios

Objetiva-se criar uma aplicação *web* que irá possibilitar a entrada dos trabalhadores autônomos no meio digital, além de facilitar o contato dos clientes com os profissionais. Os clientes terão um portal web com listas de diversos serviços disponíveis em sua região, podendo filtrá-los por vários modos, como tipo do serviço, por ranque de avaliações e localidade.

A aplicação também fornecerá a análise do perfil do prestador de serviços, mostrando os trabalhos realizados, meios de contato, comentários e avaliações de clientes, para que

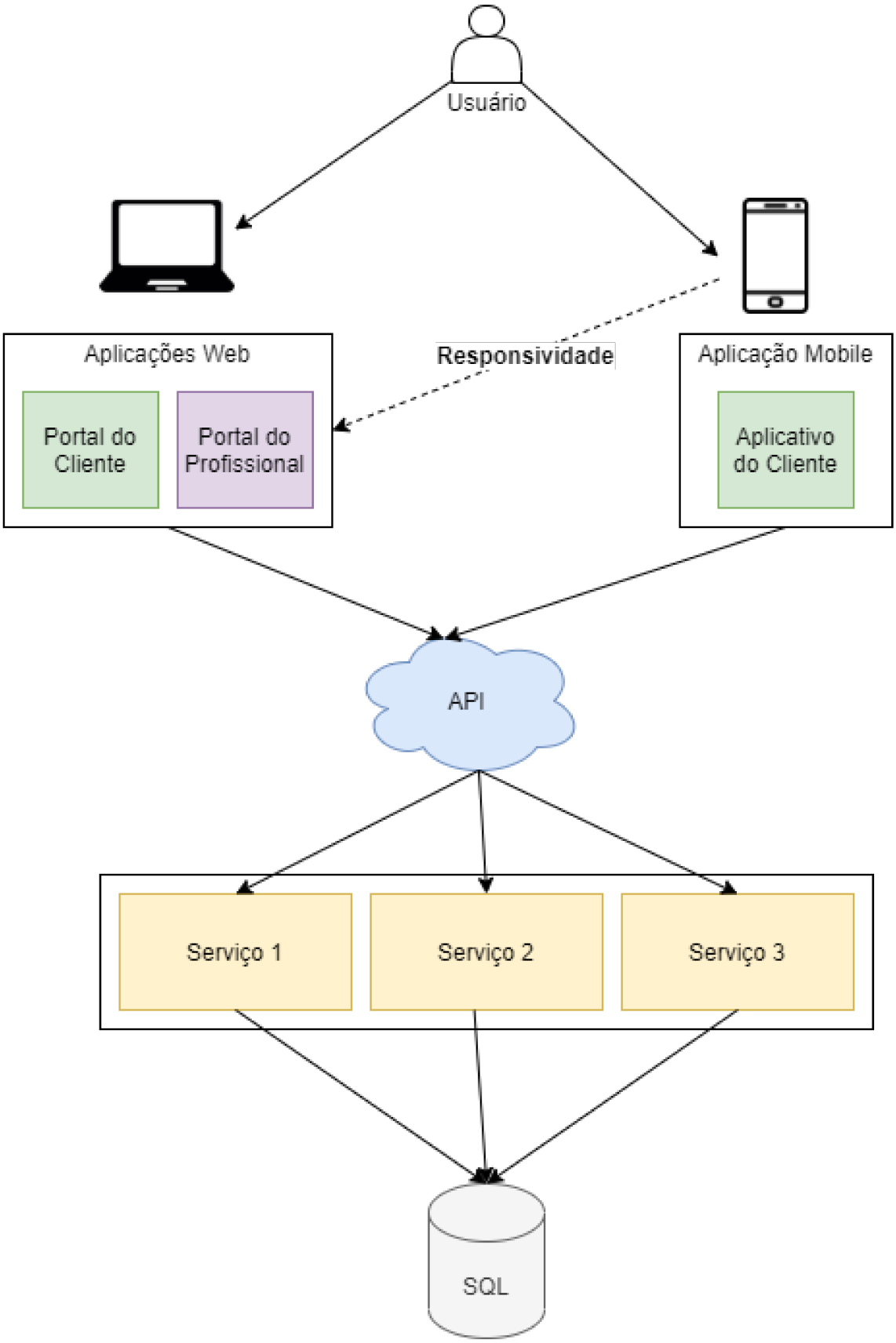

Figura 1 – Arquitetura geral da solução.

o cliente empregador se sinta seguro ao contratar uma pessoa desconhecida, resolvendo assim, as dores dos usuários da aplicação.

Como desafios, a inserção da aplicação no mercado, se as pessoas irão realmente se acostumar e aderir a seu uso. Além disso, a integração de todas as tecnologias que serão usadas para o desenvolvimento do *front-end*, bem como a integração com o servidor também irão representar um grande desafio.

1.7 Metodologia

Para que os objetivos apresentados anteriormente sejam realizados, os seguintes passos serão seguidos para o desenvolvimento dos portais *web*:

- ❑ Criar um protótipo de solução para o problema apresentado:

    1. Estudar o mercado ao qual a aplicação será utilizada;
    2. Fazer o levantamento de requisitos e usabilidade da aplicação, isso inclui a realização de entrevistas com profissionais prestadores dos serviços que poderão ser incluídos na base de dados e também a clientes empregadores que utilizarão o *website*;
    3. Estudo e definição de quais ferramentas serão utilizadas para construção do portal do usuário e dos profissionais;
    4. Fazer a modelagem das telas em um software especializado em experiência de usuário;
    5. Designar as tarefas a serem realizadas em um quadro *kanban*.

- ❑ Programar a solução prototipada:

    1. Subir um servidor local com Node.js;
    2. Enumerar as APIs externas que serão consumidas;
    3. Fazer as chamadas de APIs utilizando Axios.js;
    4. Controlar o versionamento dos códigos utilizando Git e GitHub;
    5. Implementar as páginas utilizando TypeScript e React.js;
    6. Fazer a estilização das páginas com Styled Components e Material-UI;
    7. Implementar os formulários e suas validações utilizando Yup e Formik;
    8. Criar testes unitários;

1.8 Organização do trabalho

Esta monografia está organizada da seguinte forma:

- No primeiro capítulo apresentou-se o nome da aplicação, as dores dos clientes, o mercado e uma visão geral da organização desta monografia.
- No segundo capítulo apresenta-se como foi realizada a gestão do desenvolvimento da solução, explicitando conceitos como desenvolvimento de software ágil, engenharia de software, usabilidade e prototipação.
- No terceiro capítulo apresenta-se a arquitetura e as tecnologias utilizadas para o desenvolvimento das aplicações *web*.
- No quarto capítulo apresenta-se o desenvolvimento da aplicação e todas as páginas implementadas.
- No quinto capítulo apresenta-se a aplicabilidade da solução proposta, qual sua relação com soluções parecidas e quais os possíveis desdobramentos futuros dessa aplicação.

Capítulo 2

Gestão do Desenvolvimento da Aplicação

Neste capítulo são apresentadas diferentes tecnologias e conceitos utilizados durante a gestão do desenvolvimento da aplicação e que são aplicadas no mercado de trabalho. Alguns desses conceitos foram estudados durante a graduação no curso de ciência da computação e outros foram estudados e aplicados ao ingressar no meio empresarial.

2.1 Engenharia de Software

É importante salientar que para produzir a aplicação Severino, foi necessário aplicar a engenharia de *software* e não só simplesmente codificar todas as páginas necessárias para seu funcionamento. Isso porque a engenharia de *software* trata-se da criação e utilização de princípios de engenharia aplicados de forma sistemática ao desenvolvimento de *software*, para que este seja feito de maneira econômica, confiável e que funcione em ambientes reais (MOHAPATRA; RATH, 2020).

Dessa forma, ao planejar e desenvolver novas funcionalidades para a aplicação, foram utilizadas algumas áreas da engenharia de *software*, como o desenvolvimento ágil, buscando sempre desenvolver funcionalidades que agregassem valor à aplicação mais rapidamente, mas codificadas de forma estável e escalável.

2.2 Metodologia Ágil e Kanban

Assim como em grandes empresas, utilizou-se a metodologia ágil para auxiliar na gestão e no desenvolvimento dessa aplicação. Essa metodologia foi lançada formalmente em 2001, quando 17 tecnólogos redigiram o Manifesto Ágil. Eles escreveram quatro princípios fundamentais para o gerenciamento ágil de projetos, com o objetivo de desenvolver um *software* mais eficiente: 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. (AGILEMANIFESTO, 2001)

Existem várias metodologias ágeis de desenvolvimento de *software*, porém nesse trabalho foi utilizado o Kanban, que é um método para gerenciar equipes e tarefas, afim de melhorar as entregas continuas de funcionalidades de uma aplicação. O principal objetivo é organizar visualmente o trabalho, maximizar a eficiência e melhorar a continuidade das atividades.

O Kanban pode ser visto como um quadro, que geralmente é separado em três partes básicas, sendo elas (MESH, 2020):

1. *To Do*: Tarefas prontas para serem inicializadas pelos membros da equipe.
2. *In Progress*: Tarefas que estão sendo desenvolvidas.
3. *Done*: Tarefas finalizadas e aprovadas pela equipe ou pelo cliente.

Para implementar o Kanban foi utilizado o Trello como ferramenta ágil de fluxo de trabalho, visando também a separação das tarefas entre os outros desenvolvedores da aplicação Severino. Na figura 2 tem-se um exemplo de atividades que foram criadas para a realização das tarefas. Atente-se que não foi usada apenas as colunas tradicionais, adicionou-se também uma lista de ideias e um *backlog*, para separar as tarefas que ainda estavam em discussão.

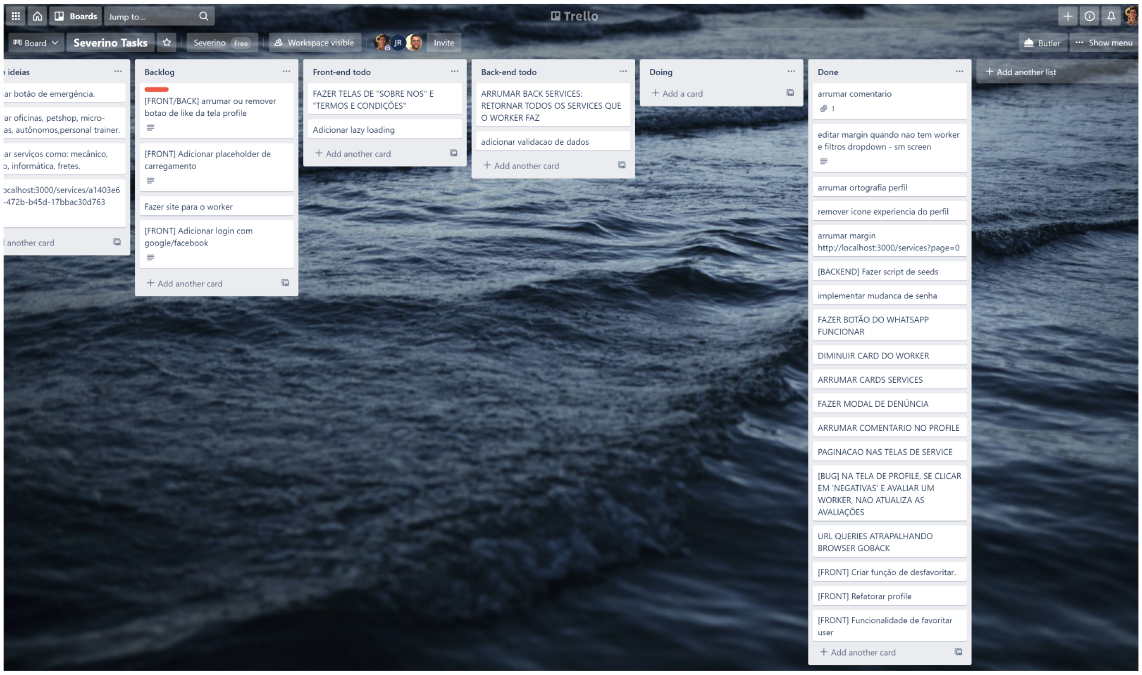

Figura 2 – Atividades criadas no Trello para a construção da aplicação.

Para decidir se uma tarefa do quadro Kanban iria para desenvolvimento, discutiu-se com os outros participantes do desenvolvimento da aplicação a usabilidade e o valor que essa tarefa agregaria à aplicação.

2.3 Usabilidade

A usabilidade é um termo utilizado para definir a facilidade com que as pessoas aprendem a manusear um sistema. Por isso, a intenção ao planejar o desenvolvimento desse *front-end* é fazer com que os clientes e os profissionais tenham uma experiência agradável, a ponto de optarem pelo Severino ao invés de outras aplicações com o mesmo conceito. Para que isso fosse possível, antes de implementar um recurso, esse foi cuidadosamente elaborado e discutido entre os desenvolvedores, amigos e familiares.

É muito importante ter uma boa usabilidade em uma aplicação, principalmente em sua versão *web*, pois a resolução da tela é alta e o usuário pode se perder mais facilmente procurando os botões de ação. Assim, ao planejar essa aplicação, investiu-se muito tempo pensando em onde colocar cada ação e se ela deveria ser colocada em vários lugares, para ficar mais fácil de ser localizada. Além disso, pensou-se também nas cores de cada titulo, texto e botão, para chamar mais atenção quando necessário, garantindo assim, uma boa usabilidade do *website*, para prender a atenção dos usuários e assegurar a resolução de seus problemas.

As regras principais a serem seguidas para garantir uma boa usabilidade e facilitar o aprendizado dos usuários:

1. Clareza na arquitetura da informação: essencial para a distinção do que é prioritário e o que é secundário no site;

2. Facilidade de navegação: o usuário deve conseguir acessar a informação desejada com no máximo 3 cliques;

3. Simplicidade: evitar qualquer exagero, dando ao usuário calma para analisar as informações;

4. A relevância do conteúdo: evitar mensagens promocionais ou publicitárias, deixar o texto o mais conciso e objetivo possível, páginas curtas e deixar as informações secundárias em páginas de suporte;

5. Consistência: a aplicação deve ter um projeto único de interface com o usuário, assim ele não precisa se preocupar a respeito do que irá acontecer pois as coisas acontecerão sempre da mesma maneira;

6. Tempo suportável: o tempo de carregamento das páginas deve ser o menor possível, para que os usuários não percam o interesse, além de ser visível que o carregamento está acontecendo;

7. Foco nos usuários: é simplesmente retirar tudo que não seja de interesse do usuário, para que ele possa realizar o que quiser de forma rápida.

Após seguir essas regras e antes de subir o sistema no ar, deve-se realizar sucessivos ciclos de análises e testes, para avaliar a qualidade das interações, levando em conta os resultados para uma nova versão das interfaces. Realizar o teste de usuário também contribui para a análise de comportamento ao utilizar o *software*, a fim de extrair os pontos positivos e negativos.

Quanto mais cedo essas regras forem aplicadas à aplicação, mais benefícios elas trarão, reduzindo riscos de falhas conceituais e garantindo que a cada ciclo o sistema responda cada vez melhor às expectativas e necessidades dos usuários.

Durante a análise de usabilidade, foram criados protótipos para que houvesse uma melhor visualização das funcionalidades, como elas seriam utilizadas pelos usuários e se seriam aceitas ou não, levando em conta críticas de amigos e familiares. A vantagem é que os protótipos são rápidos de implementar e alterar além de ajudarem na visualização do sistema como um todo.

2.4 Conceitos e Ferramentas de Criação de Protótipos

A prototipagem é um processo experimental em que as equipes de *design* implementam ideias em formas concretas, do papel ao digital. As equipes constroem protótipos de vários graus de fidelidade para capturar conceitos de *design* e testar a usabilidade por parte dos usuários. Com protótipos, é possível refinar e validar projetos para que uma marca possa lançar aplicações que agradem os clientes.

Prototipar o projeto é essencial para a experiência do usuário (UX) e para uma melhor visualização das regras de negócio que a equipe elaborou e selecionou, visando filtrar ou criar ideias que possam resolver as necessidades dos usuários. Na prototipagem, é criado um modelo experimental simples da aplicação proposto para que se possa verificar se ele corresponde ao que os usuários desejam por meio do *feedback* que fornecem. Deve-se considerar a prototipagem desde o início, usando a prototipagem em papel, se apropriando aos *feedbacks* obtidos dos usuários para ajudar a orientar o desenvolvimento (FOUNDATION, 2019). Na figura 3, tem-se um exemplo de como funciona a prototipação de telas.

As vantagens da prototipagem são:

1. Ter uma base sólida para visualizar necessidades de melhorias, dando a todas as partes interessadas uma imagem clara das regras de negocio, benefícios, riscos e custos potenciais associados a uma ideia.

Figura 3 – Exemplificação da prototipagem de uma tela (PEHAM, 2015).

2. Poder adaptar as interfaces rapidamente, o que poupa tempo e evita o compromisso com uma versão única e falsamente ideal, levando à redução de custos e aumentando o rendimento de tempo.

3. Ter uma fonte universal para os desenvolvedores se basearem no momento de construção do código da aplicação e evitar mal-entendidos em relação às regras de negócio e evitar que programadores tenham que pensar em como deixar uma tela mais agradável para o usuário, já que o trabalho da prototipagem geralmente fica com um profissional especializado em *User Experience* e *User Interface*.

Tendo em vista a importância e as vantagens da prototipação, optou-se por utilizar uma ferramenta de prototipação, o Figma, devido ao conhecimento prévio, vindo do ambiente de trabalho e por ser uma ferramenta utilizada pelas empresas atualmente.

2.4.1 Figma

Figma é um aplicativo de design de interface que funciona no navegador mas também possui uma versão *desktop* para Windows e Mac Os. Ele fornece todas as ferramentas necessárias para a fase de design do projeto, incluindo ferramentas vetoriais, *frames*, formas geométricas e outros. Possui também recursos de prototipagem e geração de código *css* (**??**). Uma das funcionalidades do Figma mais utilizadas durante a gestão e desenvolvimento dessa aplicação, foi a de compartilhamento de projetos, assim, os outros alunos que estavam envolvidos conseguiam visualizar e contribuir com a prototipagem.

O uso das principais funcionalidades do Figma é gratuito, mas também existem funcionalidades pagas, mas que são utilizadas somente em times com muitas pessoas, portanto, utilizou-se a versão gratuita para prototipar o Severino.

Após entender melhor os conceitos e ferramentas de gestão e prototipagem, iniciou-se o levantamento de requisitos, juntamente com o desenho dos protótipos para deixar as ideias e regras de negócio mais concretas.

Capítulo 3

Levantamento de Requisitos

A primeira etapa para o desenvolvimento da aplicação foi o levantamento de requisitos, em que realizou-se a pesquisa por problemas apresentados em plataformas com o mesmo conceito do Severino, como reclamações de pessoas em relação ao modo de navegar e encontrar profissionais nesses outros *websites*. Além disso, realizou-se uma pesquisa, perguntando a familiares e amigos o que gostariam de encontrar na plataforma e dicas de melhorias gráficas e de acessibilidade. Por fim, estudou-se sistemas bem sucedidos, como iFood, Uber, Rappi, GetNinjas e outros para encontrar elementos de UI e UX que fizeram com que todas elas prosperassem e fossem bem aceitas pelos usuários.

Assim, como foi decidido que a parte *web* seria dividida em dois sistemas, um para o cliente empregador e um para os profissionais, fez-se o mapeamento das páginas e componentes essenciais a serem acrescentadas no fluxo e nas regras de negócio para cada um dos portais, sendo elas também os principais requisitos. Abaixo estão listados os requisitos das paginas e dos principais componentes de cada portal e seus protótipos feitos no Figma:

3.1 Portal do Cliente Empregador

O Portal do Cliente contém as páginas abaixo:

3.1.1 Páginas

1. **Home:** essa é a primeira página a ser apresentada para o usuário, nela contém uma barra de pesquisa, onde o cliente poderá pesquisar por serviços e também já possui alguns *cards* apresentando os principais serviços presentes na ferramenta. Logo abaixo, na tela, terá um texto explicando brevemente como o *website* funciona. A Figura 4 ilustra essa página.

2. **Login:** essa página é onde o cliente colocará seu e-mail e senha para autenticar-se no *website* e então seguir para a página anterior ou para *Home*, caso não exista uma

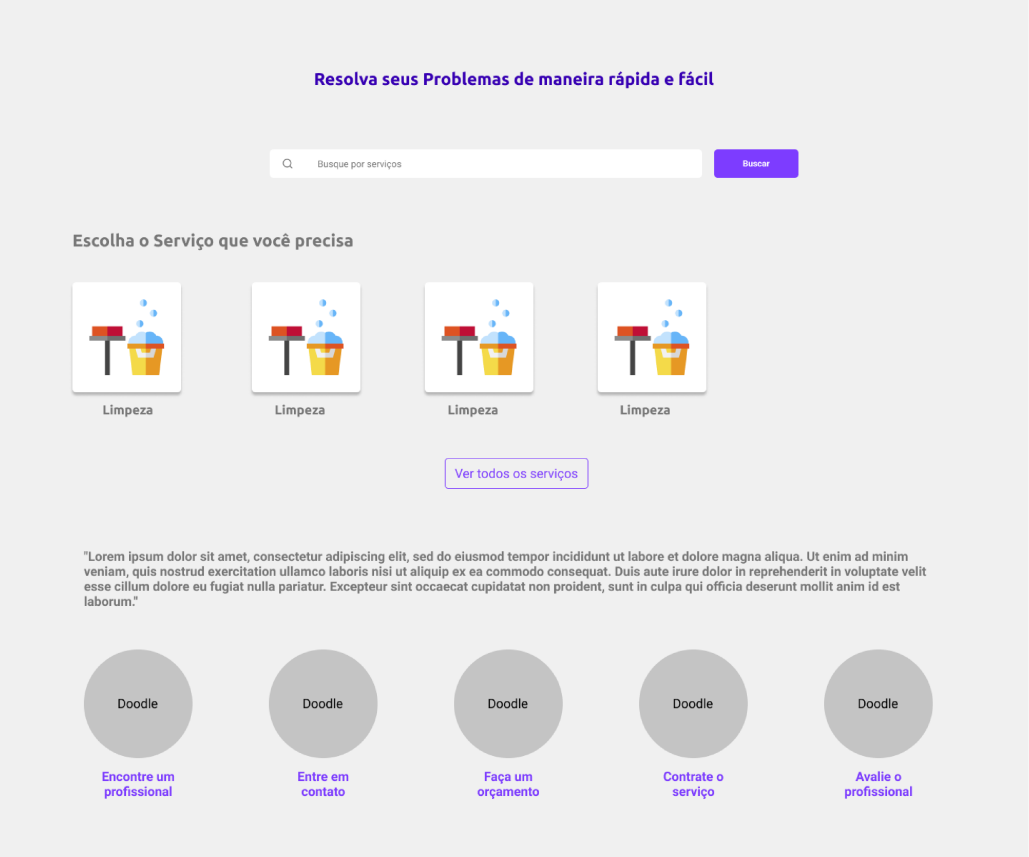

Figura 4 – Página *home*

página anterior. A Figura 5 ilustra essa página.

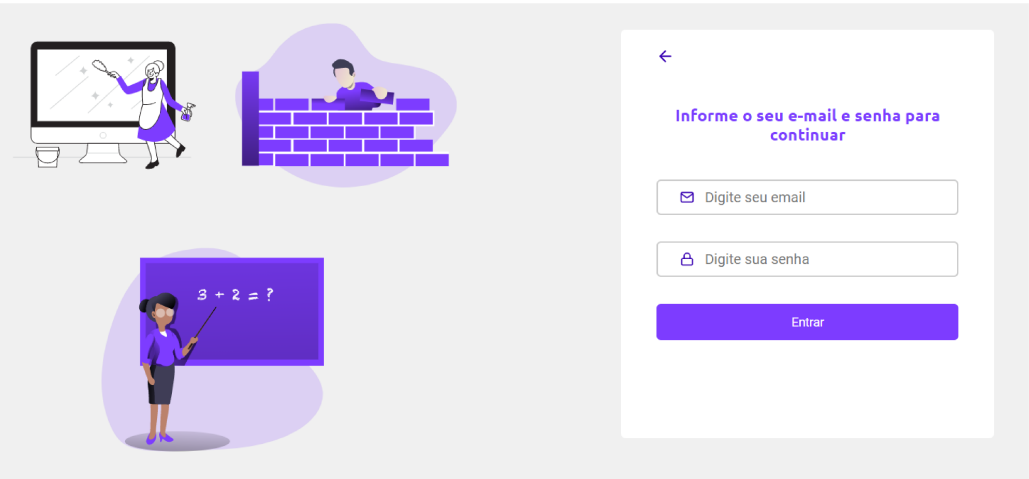

Figura 5 – Página de login.

3. **Cadastro:** a página de cadastro será onde o cliente irá colocar seu e-mail, nome e senha para realizar cadastro, após esse passo a conta será criada, o login será efetuado automaticamente, redirecionando o usuário para a página anterior ou para a *Home*, caso não exista uma página anterior. Os três campos citados são obrigatórios.

A Figura 6 ilustra essa página.

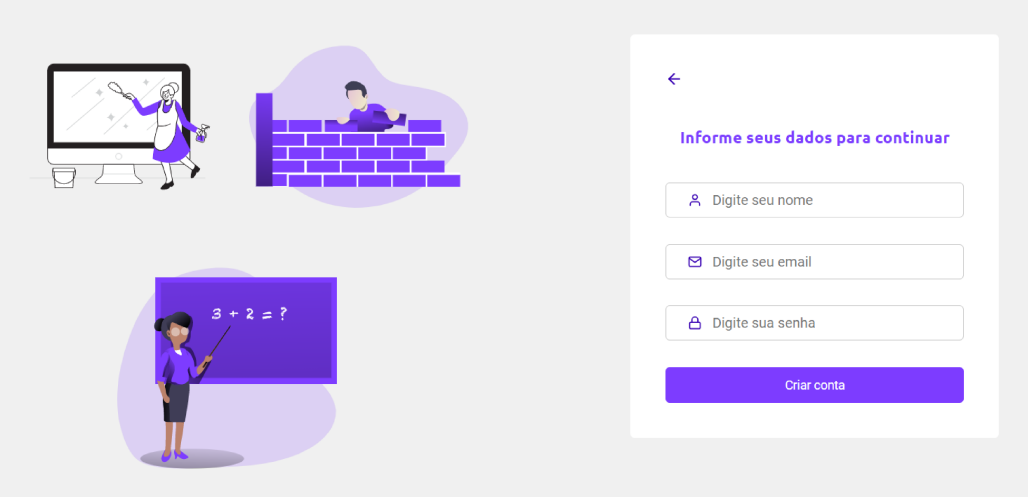

Figura 6 – Página de cadastro.

4. **Trocar Senha:** página responsável por fazer a troca da senha do cliente, onde deve ser informado a senha atual, a nova senha e a confirmação dela, sendo os três campos obrigatórios. A Figura 7 ilustra essa página.

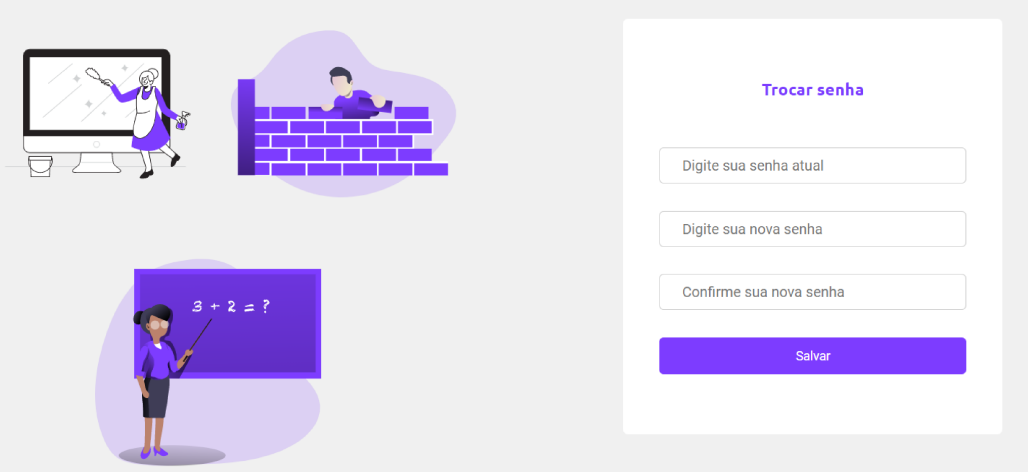

Figura 7 – Página de troca de senha.

5. **Serviços gerais:** nessa página terá um campo de busca de serviços e alguns *cards* de serviços gerais, além de conter uma paginação para carregar mais serviços. É importante ressaltar que os serviços foram divididos em **gerais** e **específicos**, em que o geral seria, por exemplo um serviço de aulas particulares e os específicos seriam, aulas particulares de matemática, português, etc. A Figura 8 ilustra essa página.

6. **Serviços específicos:** essa é a página em que se encontram as principais regras de negócio da aplicação, contendo filtros por serviços específicos, melhores avaliações

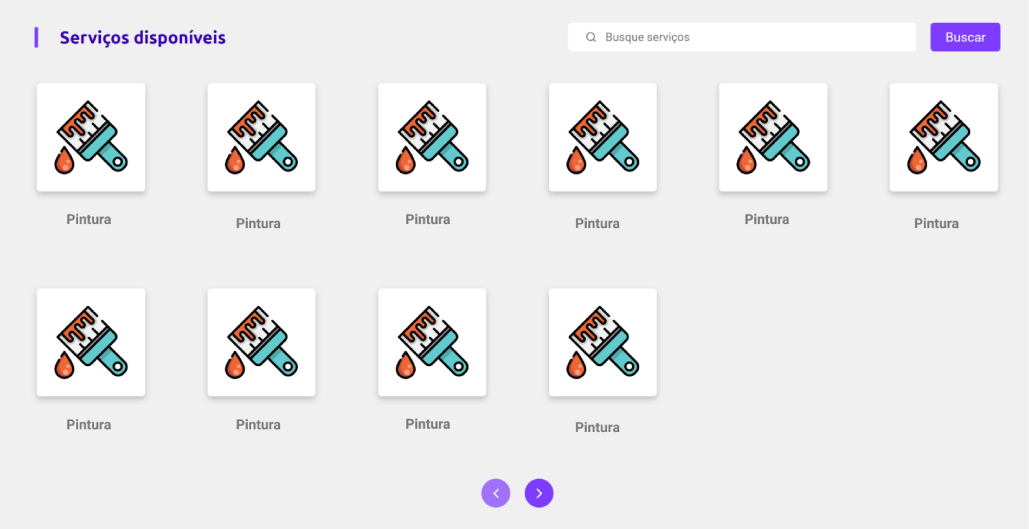

Figura 8 – Página de serviços gerais.

e localidade. Nela também é apresentado inúmeros *cards* contendo as informações dos profissionais. A Figura 9 ilustra essa página.

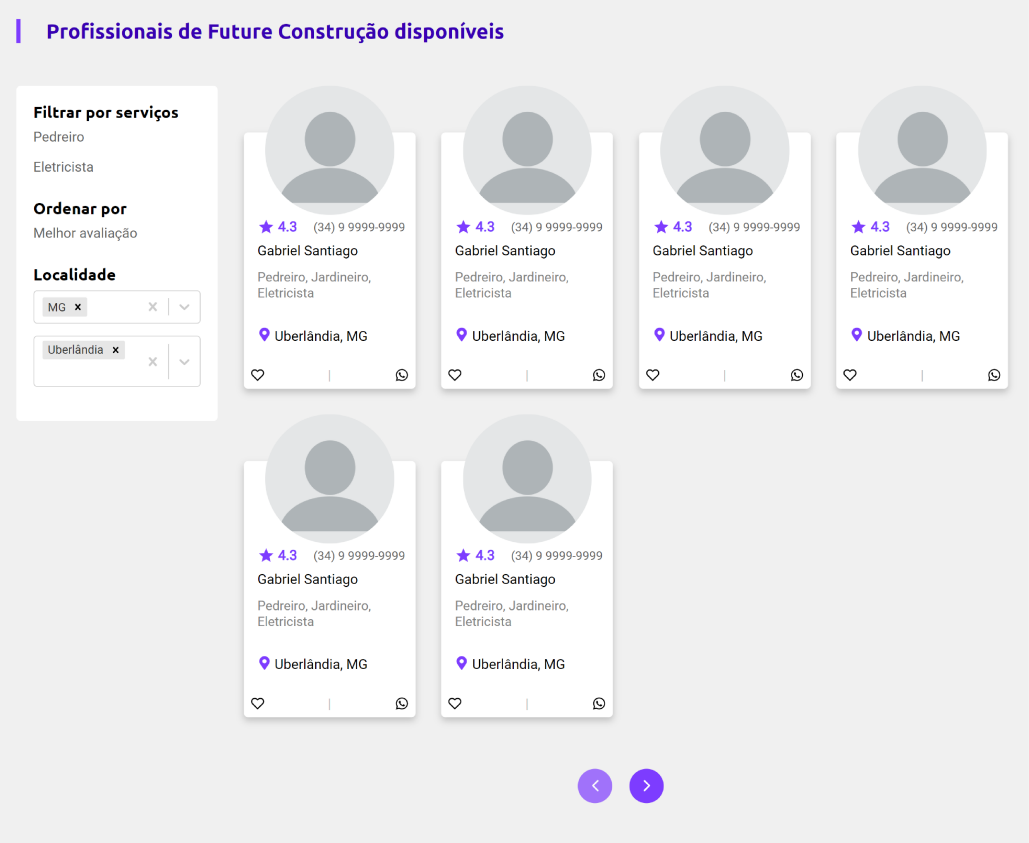

Figura 9 – Página de serviços específicos.

7. **Perfil do profissional:** A Figura 10 ilustra essa página, que irá conter as principais informações do trabalhador com subseções, sendo elas:

a) Apresentação da foto do trabalhador e botões de ação, sendo eles, botão para redirecionamento ao *WhatsApp*, botão de favoritar, botão que irá rolar a página para seção de avaliação, botão de avaliar, o qual abrirá um modal de avaliação e botão de denúncia que irá abrir outro modal;

b) Apresentação o nome do prestador de serviço, avaliação média, número de telefone, localização, serviços prestados e uma descrição de sua escolha;

c) Apresentação de um *carroussel* com fotos e legendas de seus trabalhos realizados escolhidas pelo trabalhador, mostrando também o seu currículo contendo experiências profissionais, formação acadêmica e competências.

d) Por ultimo, a seção de avaliações, que irá irá mostrar a média de avaliações do prestador de serviços, bem como as avaliações classificadas em positivas, negativas ou então todas elas, junto os comentários dos clientes, mostrados de forma paginada.

3.1.2 Componentes

1. ***Cards* com informações dos trabalhadores:** nesses *cards* estarão contidos: uma foto do profissional, avaliação media, telefone com DDD, nome, serviços prestados, cidade, um botão para favoritar e um para redirecionar o cliente para o *WhatsApp* do profissional. Se for um novo profissional da plataforma, sua avaliação aparecerá como "Novo Usuário". Ao clicar no *card*, o cliente será redirecionado para o perfil do profissional.

2. **Modal de avaliação:** irá conter uma lista com 5 estrelas em que o usuário deverá selecionar de 1 a 5, título do comentário (opcional) e por fim o comentário (opcional).

3. **Modal de denúncia:** irá apresentar os seguintes campos: e-mail para contato (opcional) e motivo da denúncia (obrigatório).

4. **Comentários:** Os comentários presentes na lista de comentários sempre irão conter a quantidade de estrelas, título e uma descrição (opcional).

3.2 Portal do Profissional

O Portal do Profissional contém as páginas abaixo:

3.2.1 Telas

1. **Home:** página que irá conter um carrossel com vários componentes, em que cada um deles terá uma imagem e uma frase para atrair e incentivar o profissional a se

cadastrar na plataforma. Logo abaixo, terá uma seção chamativa com um botão que irá redirecionar para a pagina de cadastro. Após essa seção, terá outra explicando o que é o Severino e os benefícios de fazer parte da plataforma. A Figura 11 ilustra essa página.

2. **Login:** essa página é onde o profissional colocará seu e-mail e senha para autenticar-se no sistema e então seguir para a página anterior ou para *Home*, caso não exista uma página anterior. Ambos os campos são obrigatórios. A Figura 12 ilustra essa página.

3. **Cadastro:** nessa página, o profissional terá que preencher os seguintes campos, todos obrigatórios: e-mail, nome, sobrenome, celular, estado, cidade, senha e confirmação de senha. Além disso, terá um *checkbox* para o usuário informar se este telefone tem *WhatsApp*. Todos os campos, exceto o *checkbox* são obrigatórios. A Figura 13 ilustra essa página.

4. **Trocar Senha:** página responsável por fazer a troca da senha do profissional, onde deve ser informado a senha atual, a nova senha e a confirmação dela, sendo os três campos obrigatórios. A Figura 14 ilustra essa página.

5. **Perfil:** A Figura 15 ilustra essa página, que será responsável pela visualização e edição dos dados do profissional, sendo dividida em 6 partes:

    a) **dados gerais**, contendo nome, número de telefone, localização, serviços prestados e descrição

    b) **fotos dos trabalhos**, contendo as fotos e legendas de serviços já realizados;

    c) **experiências profissionais**, onde o usuário poderá descrever a empresa que já trabalhou, o tempo e a localidade daquela experiência

    d) **formação acadêmida**, que contem a instituição e o tempo de estudo

    e) **competência**, onde o usuário poderá descrever suas *soft skills* e por ultimo, uma parte para acompanhar sua avaliação e os comentários que recebe em seu perfil.

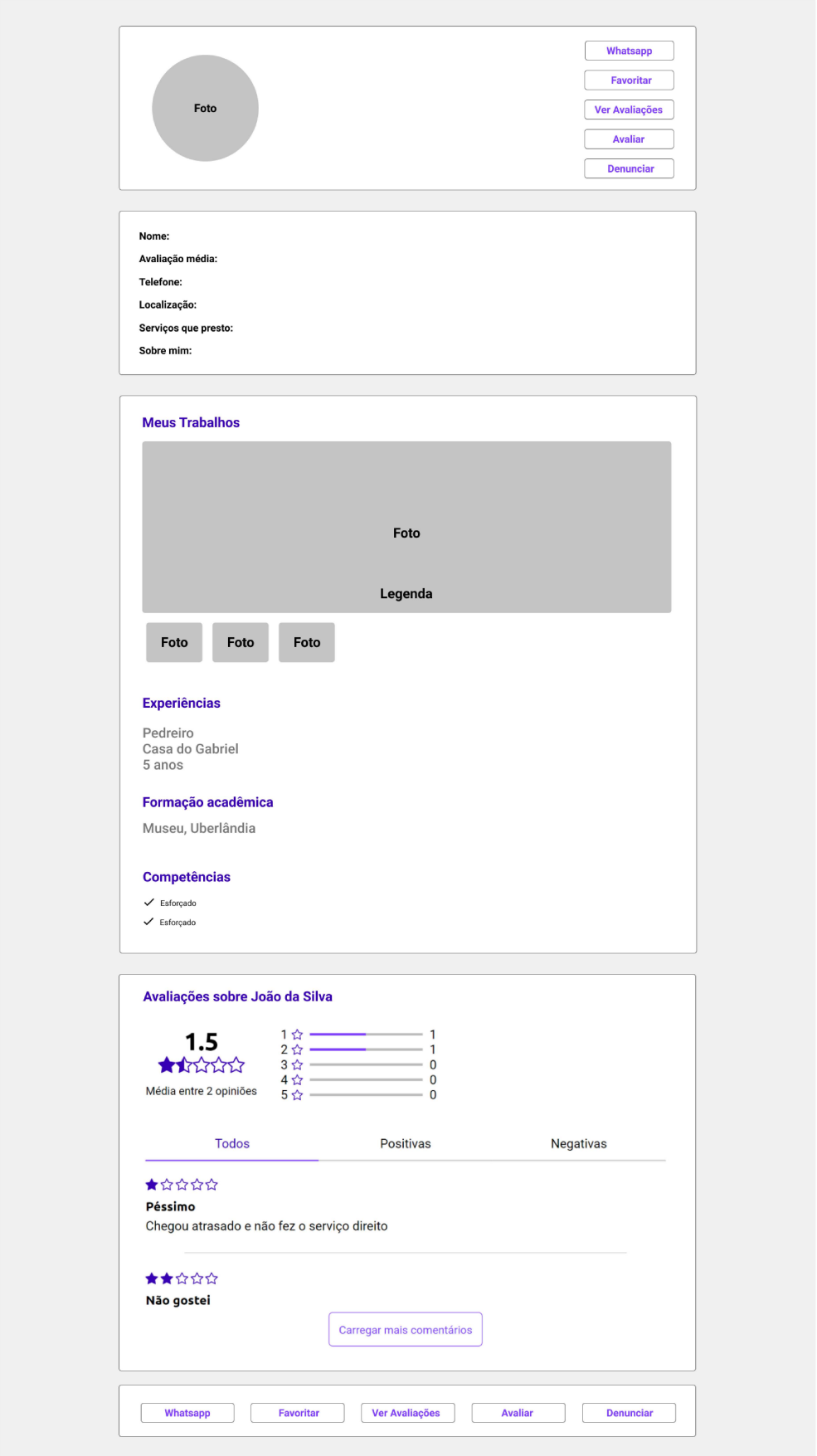

Figura 10 – Página de perfil do profissional

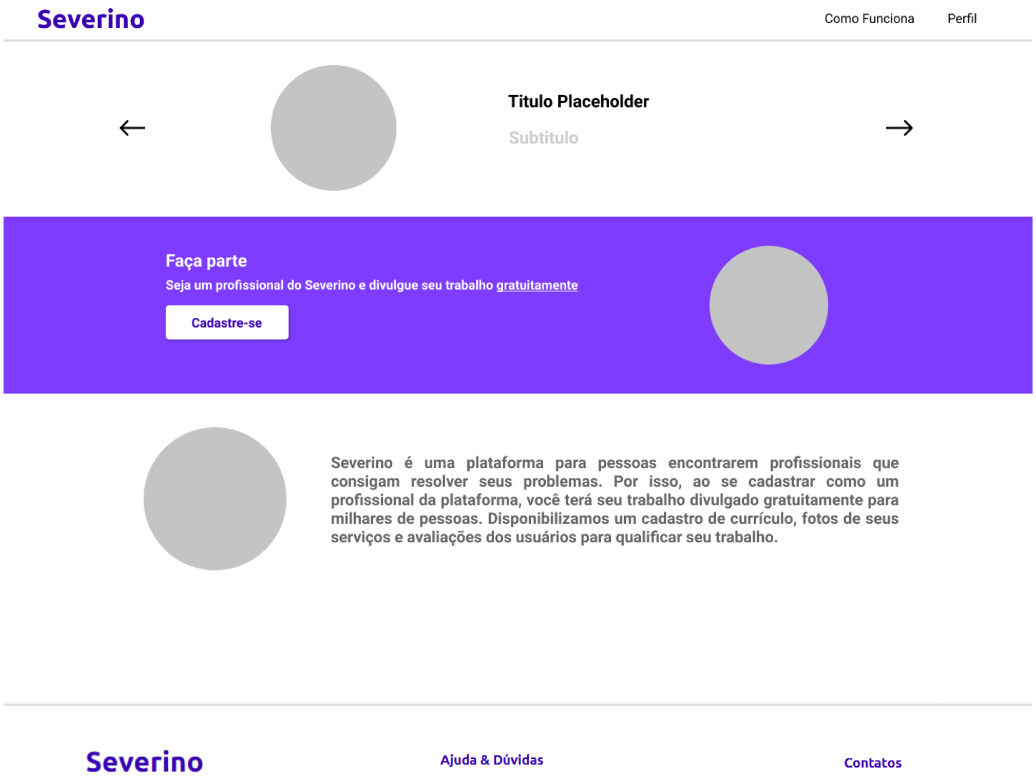

Figura 11 – Página *home*

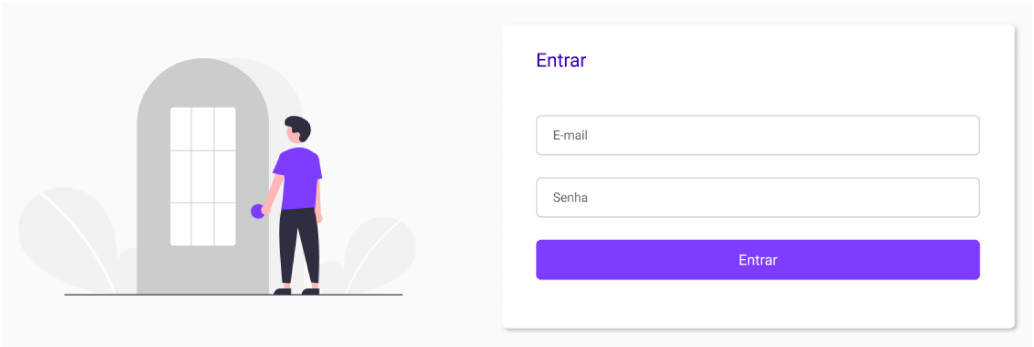

Figura 12 – Página de *login*

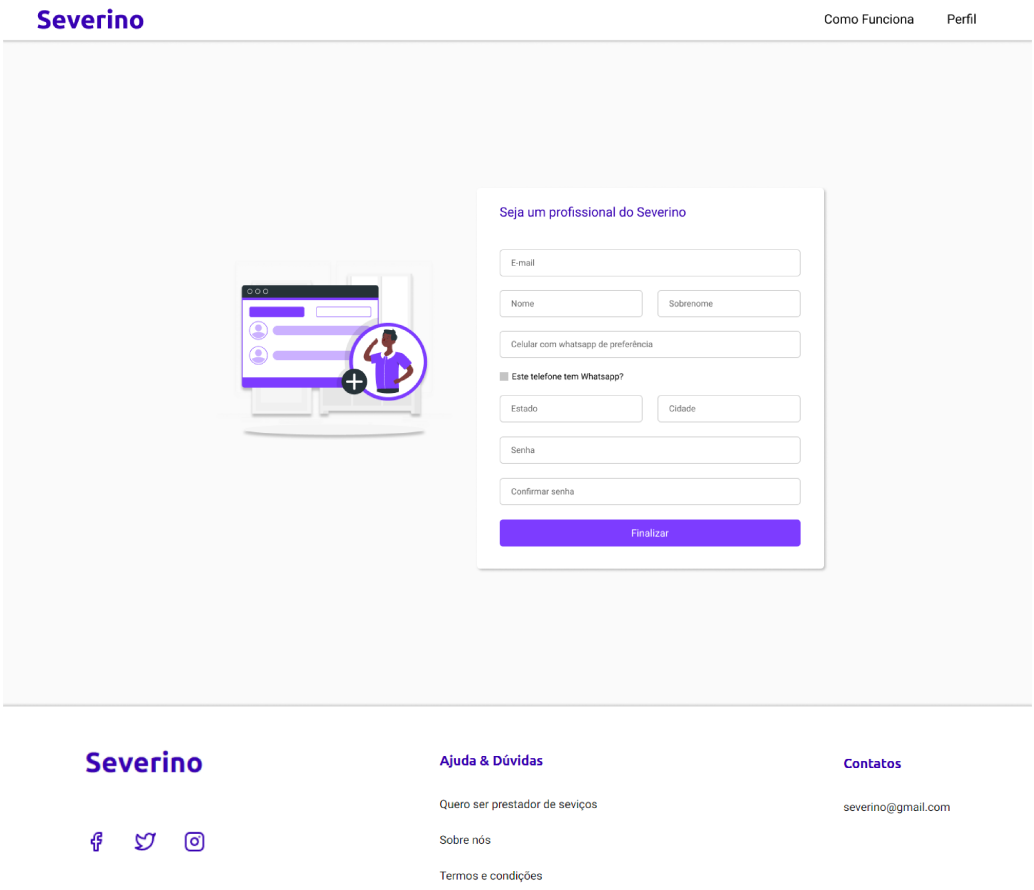

Figura 13 – Página de cadastro

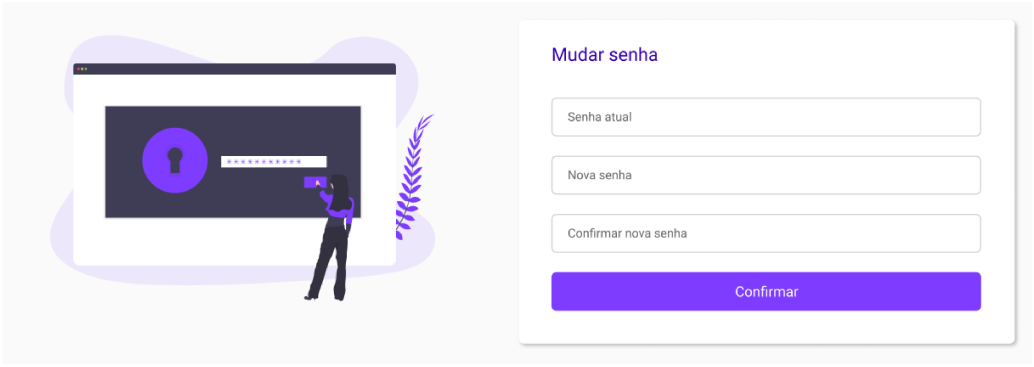

Figura 14 – Página de troca de senha

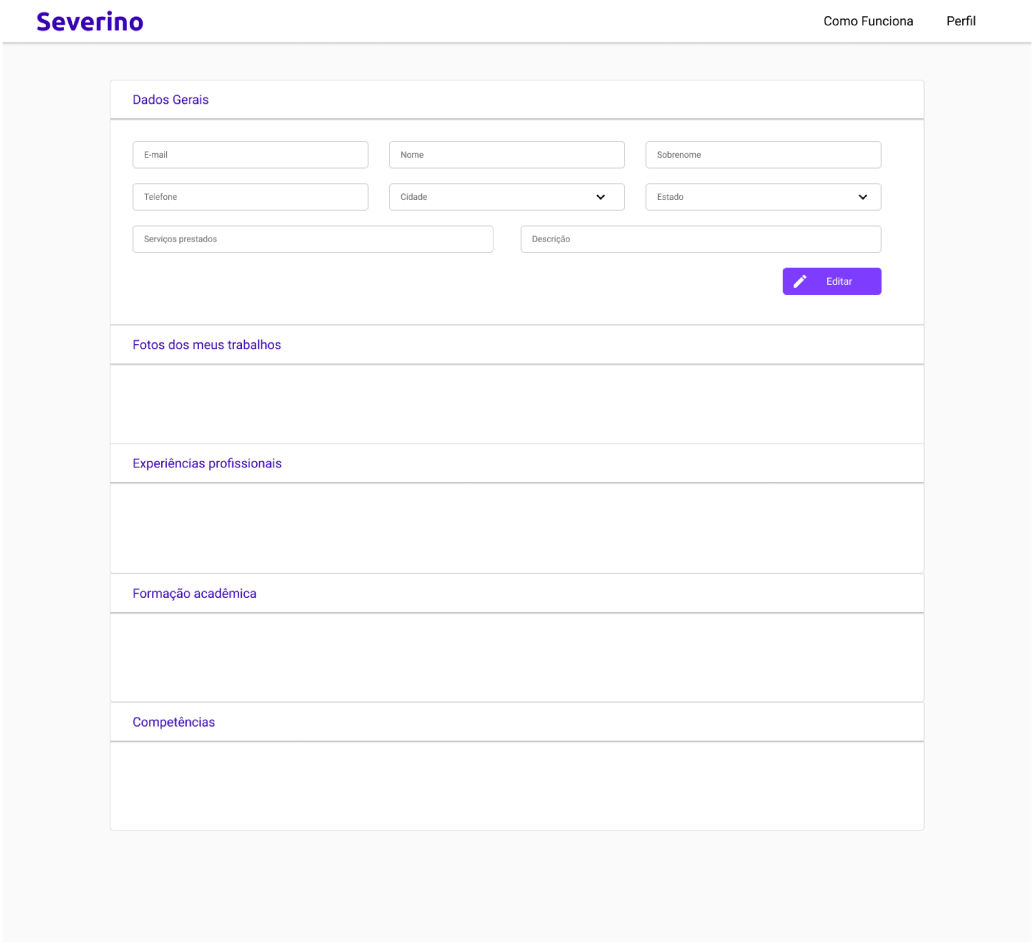

Figura 15 – Página de perfil e edição de informações do profissional

Capítulo 4

Arquitetura e Tecnologias

Após definir os requisitos de cada tela e seus respectivos *layouts*, definiu-se também a arquitetura e as principais tecnologias que seriam usadas na aplicação. Já que este trabalho trata-se da parte *web* da aplicação Severino, será apresentado o conceito de aplicações web antes de explicitar-se a arquitetura e as tecnologias.

4.1 Aplicações Web

Uma aplicação *web* é uma solução que é executada diretamente no navegador por meio de um servidor *web*, utiliza de tecnologias *web* para atuar no papel de cliente de um sistema cliente-servidor em que não é preciso realizar nenhuma instalação na máquina do usuário além do navegador de internet (JAZAYERI, 2007).

As aplicações *web* são desenvolvidas com o uso de algumas tecnologias, como HTML (*HyperText Markup Language*) para a marcação, CSS (*Cascading Style Sheets*) para a construção das páginas e estilização, HTTP (*Hypertext Transfer Protocol*) ou HTTPS (*Hypertext Transfer Protocol Secure*) para a comunicação com o servidor e *script languages* como o JavaScript para a construção logística da ferramenta.

HTML é a linguagem de marcação padrão para a criação de páginas da *web*, descreve a estrutura do *website* e consiste em uma série de elementos, os quais informam ao navegador como exibir o conteúdo, identificam partes de conteúdo como "isto é um título", "isto é um parágrafo", "isto é um link", etc. Sua versão atual é a 5, que apresenta um conjunto maior de tecnologias, elementos e recursos (W3SCHOOL, 2013) e por isso essa versão foi a escolhida pra ser utilizada no desenvolvimento da aplicação Severino.

Como a linguagem HTML é feita somente para marcação, precisa-se do CSS para definir a estilização de uma página *web*. A linguagem CSS descreve como os elementos HTML devem ser exibidos na tela, no papel ou em outra mídia. A figura 16 representa a saída de código escrito em HTML e estilizado usando o CSS.

Entrada: HTML:

```
<h1>Olá mundo</h1>
```

CSS:

```
h1 {
    background-color: red;
    color: blue;
}
```

Saída:

Figura 16 – Exemplo de estilização com CSS e HTML.

Com o HTML e o CSS consegue-se criar páginas estáticas, ou seja, os usuários não conseguem interagir com as páginas. Para que essa interação aconteça, precisa-se de uma linguagem capaz de modificar o *Document Object Model* (DOM) dinamicamente. O DOM nada mais é do que o conjunto de dados presentes no navegador e em uma página *web*, como os elementos HMTL, suas estilizações, o armazenamento do navegador, o tamanho da janela, entre outros.

Para tornar as páginas interativas, utiliza-se a linguagem de programação JavaScript, que também é responsável por realizar requisições HTTP, conduzindo a comunicação com o servidor e descrevendo a lógica do cliente. Além disso, como na maioria das linguagens de programação, existem bibliotecas e *frameworks* de JavaScript, criados para facilitar e agilizar o trabalho dos desenvolvedores. Na aplicação Severino, por exemplo, utilizou-se como principal biblioteca o React.js, desenvolvido pelos mesmos desenvolvedores do Facebook e utilizado por grandes empresas, como Uber, NEtflix e iFood.

4.2 Biblioteca Core da Aplicação

4.2.1 React.js

React é uma biblioteca JavaScript declarativa criada pela mesma equipe do Facebook e é eficiente e flexível, desenvolvido no intúito de facilitar a construção de *Single Page Applications* utilizando ECMAScript 6 e JSX, que nada mais é que uma sintaxe usada para escrever código HTML em um arquivo de JavaScript. Além disso, o React permite

criar interfaces complexas a partir de pequenos pedaços isolados de código chamados de componentes (REACTJS.ORG, 2021). O código abaixo representa um componente de *input* reutilizável criado para a aplicação Severino, que tem como finalidade a inserção de dados e que pode ter um ícone opcional. Link para o código

Nota-se que mesmo que o código acima pertença a um arquivo JavaScript, escreveu-se uma sintaxe HTML no retorno da função e isso é possível graças ao JSX citado anteriormente. Assim, pode-se utilizar o componente de *input* para gerar um componente visual na tela do usuário, em que a entrada é dada abaixo e a saída é representada na figura 17.

```
<Input
    placeholder='Busque serviços'
    icon={FiSearch}
/>
```

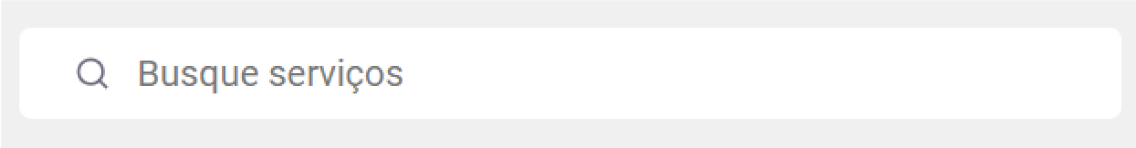

Figura 17 – Saída do componente de *input*

A escolha do React para o desenvolvimento da aplicação Severino foi devido ao seu alto desempenho na criação de *Single Page Applications*, que serão explicadas mais adiante, e pelo fato de ser uma biblioteca e não um *framework*. A diferença entre trabalhar com bibliotecas e *frameworks* é que com o segundo, existem regras e limitações a serem seguidas para atender a uma interface, já com o primeiro, os desenvolvedores são os responsáveis por ditar as regras.

4.2.2 Single Page Application

Um Single Page Application (SPA) é um tipo de *website* que carrega todos os recursos necessários para navegar por todo o sistema no primeiro carregamento da página. À medida que o usuário clica em links e interage com a página, o conteúdo subsequente é carregado dinamicamente, por meio de código JavaScript (DEVELOPERS.GOOGLE.COM, 2021). O React ajuda nesse processo e atualiza a URL na barra de endereço para emular a navegação tradicional nas páginas. No entanto, outra solicitação de página completa nunca é feita, já que o React utiliza o JSX e o Virtual DOM para atualizar o DOM real da página, atualizando somente os componentes que tiveram seus estados alterados e seus respectivos filhos, como mostra a Figura 18.

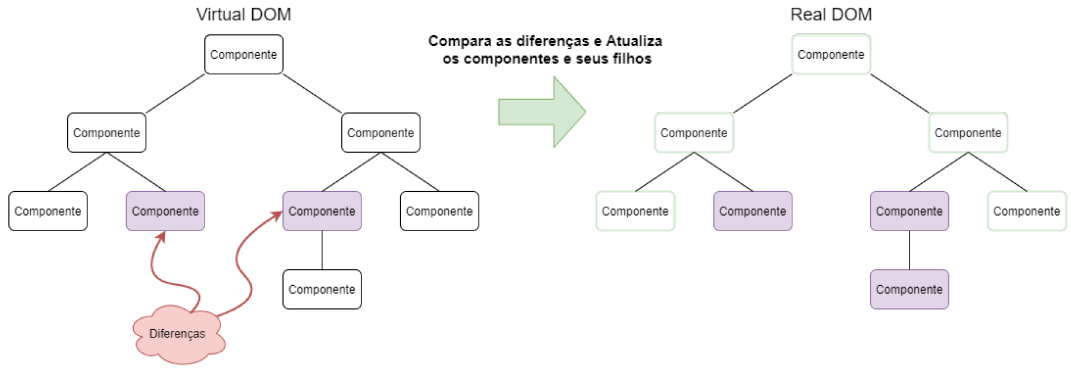

Figura 18 – Exemplo Virtual DOM

4.2.3 ECMAScript 6

Para codificar aplicações React, utiliza-se o ECMAScript 6 (ES6), que surgiu em 2015 e é a sexta edição da especificação da linguagem JavaScript, a qual foi lançada em 1997. Essa edição tem o propósito de fornecer melhor suporte para grandes aplicações, criação de bibliotecas e para uso do JavaScript como um destino de compilação para outras linguagens. Algumas de suas principais melhorias incluem módulos, declarações de classe, escopo de bloco lexical, iteradores e geradores, *Promises* ou promessas de programação assíncrona, padrões de desestruturação e chamadas de cauda adequadas. (INTERNATIONAL, 2015).

4.2.4 TypeScript

Por mais que a linguagem padrão para escrever aplicações React seja o JavaScript, existe a possibilidade de utilizar o Typescript, que é uma linguagem de código aberta baseada no JavaScript, que adiciona definições de tipo estático, facilitando o desenvolvimento devido à inferência de tipos. Tipos fornecem uma maneira de descrever a forma de um objeto, fornecendo uma melhor documentação e permitindo que o TypeScript valide o código em tempo de compilação para que funcione corretamente. Contudo, como a linguagem padrão utilizada pelo navegador é o JavaScript, todo código escrito em TypeScript é compilado para JavaScript antes de ser referenciado no navegador. (TYPESCRIPTLANG, 2021).

Para que fosse possível utilizar a inferência de tipos, utilizou-se o TypeScript como linguagem padrão para o desenvolvimento da aplicação Severino. A figura 17 mostra um exemplo da criação de uma interface para o componente de *Input*, em que este espera um ícone do tipo *React.ComponentType<IconBaseProps>*.

4.3 Arquitetura

Após definir qual será a biblioteca *core* da aplicação, criou-se a arquitetura da solução, bem como definiu-se os padrões de projeto, estrutura de pastas, ferramentas de integração, e principais bibliotecas a serem utilizadas:

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.
5. O *website* será uma Single Page Application.
6. Uso de carregamento lento (*Lazy Loading*) em páginas que não forem muito acessadas.
7. Idioma do código: Inglês.
8. Ferramenta de integração será o GitHub utilizando o Git.
9. O gerenciador de pacotes será o Yarn.
10. Para a estilização será usado a biblioteca Styled Components.

4.3.1 Estrutura de pastas e componentes

```
root
├── public                (Arquivos estáticos)
├── src
│   ├── assets            (Ícones e imagens - svg, png, jpg)
│   ├── components        (Componentes compartilhados e reutilizaveis)
│   ├── contexts          (Contextos globais)
│   ├── hooks             (Custom Hooks)
│   ├── interfaces        (Arquivos de tipagem)
│   ├── pages             (Telas em que são aplicadas as regras de negócio)
│   ├── services          (Arquivos de comunicação com Apis)
│   ├── styles            (Estilos globais e temas)
│   ├── App.js            (Componente root)
│   └── index.js          (Entrypoint da aplicação)
├── .env                  (Variáveis de ambiente)
├── README.md             (Documentação resumida)
└── ...                   (Outros arquivos de configuração)
```

4.4 Principais Bibliotecas e Tecnologias

4.4.1 React Hooks

Hooks foram adicionados na versão 16.8 do React e permitem o uso de estados e outros recursos do React sem escrever uma classe. Eles resolvem alguns problemas que a estrutura de classes tinha, como a dificuldade de reutilizar a logica entre componentes *stateful*, a complexidade dos métodos de ciclo de vida do React, o uso do *this* e a verbosidade das classes. (REACT.JS, 2021) Além disso, como os *Hooks* são recomendados pela documentação do *React*, optou-se por utilizá-los na aplicação Severino.

4.4.2 Carregamento lento

O carregamento lento ou *lazy loading* é uma técnica que adia o carregamento de recursos não críticos no momento do carregamento da página. Em vez disso, esses recursos são carregados no momento da necessidade, ou seja, quando a página é acessada. No que diz respeito às imagens, "não crítico"costuma ser sinônimo de "fora de tela" (DEVELOPER.MOZILLA.ORG, 2021). Exemplo, na aplicação *web* do Severino, a página de mudar senha não é acessada com muita frequência, por isso ela pode ter seu carregamento adiado.

O código abaixo mostra um exemplo em que o primeiro componente é importado da forma convencional e é carregado mesmo se o usuário nunca acessar sua rota e o segundo componente é importado usando o *lazy loading* e somente é carregado se o usuário acessar sua rota. Link para o código

4.4.3 Styled Components

Outra biblioteca importante que será utilizada na construção da aplicação Severino, é o Styled Components, uma biblioteca criada para estilizar componentes React usando a linguagem JavaScript, o que facilita a importação e exportação de arquivos e estilos. Essa biblioteca gera nomes de classes automaticamente, evitando que haja conflitos gerados por nomes iguais, colaborando também com a manutenção dos componentes e seus estilos. Além disso, uma das *features* mais interessantes do Styled Components é que a biblioteca gera estilos *cross-browser* automaticamente.(DOCUMENTATION, 2021)

O código abaixo exemplifica um componente criado com *Styled Componentes*, o qual cria uma *tag div* estilizada e utiliza um botão já existente para ser estilizado também. Link para o código

4.4.4 Node.js

Para que fosse possível desenvolver a aplicação Severino com um servidor local, utilizou-se também o Node.js, que é um ambiente servidor JavaScript de código aberto, executado

por uma *WebAssembly engine*, chamada de V8, o qual é um mecanismo que recebe um código JavaScript e o converte em um código de máquina mais rápido, capaz de ser executado fora do navegador (NODE.JS, 2021). Outra razão para utilizar o Node.js, é que este se faz necessário para utilizar o gerenciador de pacotes Yarn.

4.4.5 Yarn

Existem dois principais gerenciadores de pacotes para projetos que utilizam Node.js, o NPM e o Yarn, que foi o escolhido para a aplicação Severino. O Yarn armazena em cache cada pacote que baixa para que não precise baixá-lo novamente em instalações futuras, além de paralelizar as operações para maximizar a utilização de recursos, de forma que os tempos de instalação sejam otimizados. Utiliza somas de verificação para verificar a integridade de cada pacote instalado antes que seu código seja executado. O Yarn garante que uma instalação que funcionou em um sistema funcionará exatamente da mesma forma em qualquer outro sistema, pois é gerado um arquivo chamado *yarn.lock* que guarda o histórico das dependências instaladas (YARN, 2021).

O Yarn foi escolhido em detrimento do NPM devido à sua maior performance, como mostra a figura 19

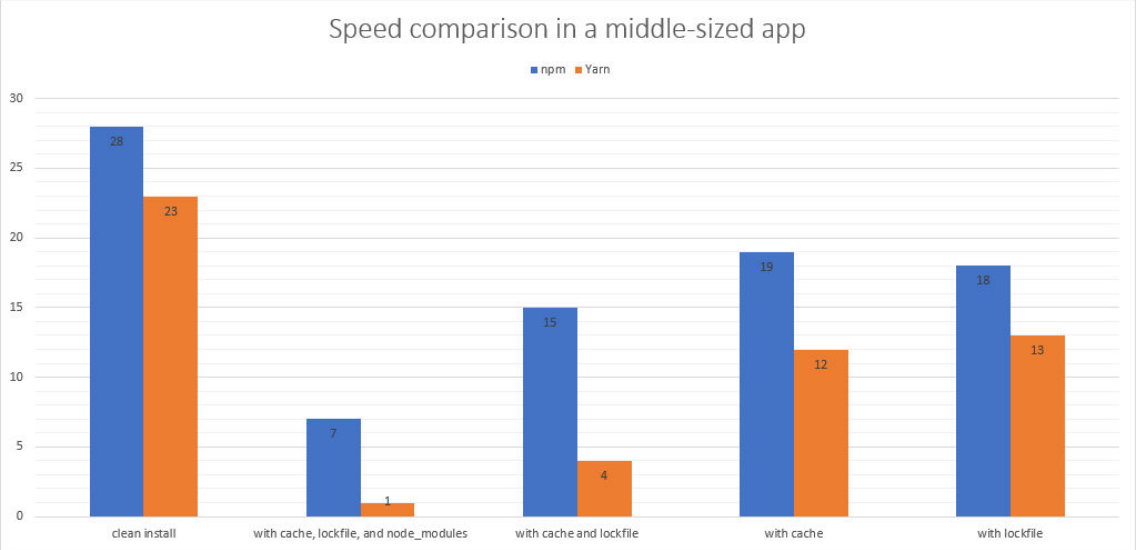

Figura 19 – Desempenho do Yarn vs. NPM (KRYUKOV, 2019)

4.4.6 API REST

É importante salientar que para que haja um padrão de definição dos *endpoints* de requisição à APIs, a aplicação *web* da aplicação Severino consome uma API REST.

Uma API é um conglomerado de definições e protocolos para construir e integrar softwares de aplicativos, criando um contrato entre um provedor de informações e um usuário de informações. REST significa "Transferência de Estado Representacional" e

determina a aparência da API, ou seja, é um conjunto de regras que os desenvolvedores seguem ao criar uma API (MASSE, 2011). Isso ajuda no desenvolvimento, pois, mesmo que o *back-end* não esteja pronto, é possível estabelecer quais serão as rotas a serem requisitadas pelo *front-end*.

4.4.7 Axios

Para que a conexão com a API fosse feita, utilizou-se a biblioteca Axios, que é um cliente HTTP baseado em *Promises* que funciona tanto em navegador quanto em ambientes Node.js, fornece uma única API para lidar com chamadas HTTP (XMLHttpRequests) (TO, 2021). Projetos dinâmicos precisam ter interface com uma API em algum ponto, e usar o Axios facilita essa conexão, além de possibilitar a customização de chamadas HTTP, a criação de *interceptors* e colaborar com a tratativa de erros. O código abaixo, retirado da aplicação Severino, exemplifica a facilidade de configurar e utilizar o Axios para realizar chamadas HTTP. Link para o código

4.4.8 Git e GitHub

Git é um projeto de código aberto desenvolvido por Linus Torvalds em 2005, projetado para facilitar o versionamento e compartilhamento códigos. É um exemplo de DVCS (Distributed Version Control System), pois ao invés de ter apenas um único lugar para o histórico de versão completa do software, cada ambiente de trabalho dos desenvolvedores do código se torna um repositório que pode conter o histórico completo de todas as alterações anteriores. Além de ser distribuído, o Git foi projetado para ter um grande 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 o usuário e outras pessoas trabalhem em conjunto de qualquer lugar. Tendo amplo controle do código e sabendo as linhas de código e por quem cada alteração foi feita. (GUIDES, 2021).

Durante a definição das principais tecnologias e ferramentas a serem utilizadas para a construção da aplicação Severino, optou-se por utilizar o GitHub, por ser gratuito e de conhecimento de todos os outros desenvolvedores da aplicalção.

4.4.9 Amazon S3

O Amazon Simple Storage Service (Amazon S3) é um serviço de armazenamento de objetos que oferece escalabilidade, disponibilidade de dados, segurança e performance a um preço escalável, que aumenta ou diminui dependendo do quanto é usado (AMAZON, 2018). Assim, optou-se por utilizar esse serviço para fazer o *deploy* da aplicação devido ao baixo custo e à alta escalabilidade, caso a plataforma atinja muitos acessos.

4.4.10 ReactIcons

Todos os ícones da aplicação Severino foram disponibilizados pela biblioteca ReactIcons, que nada mais é que uma biblioteca que disponibiliza ícones no formato SVG, como os ícones do Material Design e do Font Awesome (ICONS, 2021).

4.4.11 Formik

Formik é uma biblioteca React que foi criada para facilitar a validação de *inputs*, formatação, máscara, *input* de *arrays* e manipulação de erros. Contudo, outras bibliotecas de formulário fazem muita abstração e por isso tem uma customização complexa, por isso optou-se pelo Formik para gerenciar os formulários da aplicação Severino, já que é uma pequena biblioteca e permite que os desenvolvedores lidem com a maior parte da manipulação do formulário. (FORMIK, 2020)

4.4.12 Yup

Para validar os formulários da aplicação, utilizou-se o Yup, que é um construtor de esquemas JavaScript para validação e análise de valores. Com ele é possível definir um esquema, transformar valores correspondentes e validar a forma de um valor existente. Esquemas Yup são extremamente expressivos permitindo modelar validações complexas e interdependentes ou transformações de valores (YUP, 2021). Além disso, é possível utilizar o Yup como *plugin* de validação para o Formik.

4.4.13 Visual Studio Code

O Visual Studio Code (VSCode) é um editor de código destinado ao desenvolvimento de aplicações web lançado em 2015 pela Microsoft, é uma ferramenta leve e multiplataforma disponível para Windows, Mac OS e Linux, atendendo uma ampla gama de projetos como: ASP.NET, Node.js e também tendo suporte para diversas linguagens como Python, Ruby, C, C++, etc. Essa ferramente é de código aberto, tendo todo seu ele disponível no GitHub, permitindo que a comunidade contribua com o seu desenvolvimento, o que gera diversas novas funcionalidades como as extensões, as quais são ferramentas criadas para auxiliar os desenvolvedores em projetos, como: indentação de código, debug, edição colaborativa, etc (MEDIA, 2021).

Toda a estrutura do código da aplicação Severino foi criada usando essa ferramenta e suas extensões para facilitar o desenvolvimento. Algumas extensões utilizadas: Prettier, Dracula, ESLint e PostgreSQL Explorer.

Capítulo 5

Desenvolvimento

Após criar a arquitetura e definir as principais tecnologias a serem usadas no desenvolvimento da parte *web* da aplicação Severino, iniciou-se a implementação. Este capítulo apresenta as fases do desenvolvimento da aplicação Severino, explica-se os principais fluxogramas e mostra-se o a implementação e apresentação das páginas *web*, bem como algumas bibliotecas escolhidas durante esse processo, tanto para o Portal do Cliente Empregador como para o Portal do Profissional.

5.1 APIs Externas

Como na aplicação Severino é necessário saber a localização do usuário ou a cidade em que se deseja buscar por profissionais, utilizou-se algumas APIs externas para auxiliar no processo de obtenção das cidades e estados Existentes no Brasil e também para localizá-las baseando-se na latitude e longitude do usuário.

5.1.1 IBGE

Foi utilizada a API de localidades do IBGE para buscar as cidades e os estados nas duas aplicações web. A documentação da API está disponível em: API IBGE e os **endpoints** usados para a busca foram: *api/v1/localidades/estados* para buscar todos os estados do Brasil e */api/v1/localidades/estados/${ESTADO}/municipios* para buscar todas as cidades de um dado estado.

5.1.2 Big Data Cloud

Utilizou-se também a API de geolocalização do Big Data Cloud, disponível em: API Big Data para localizar a cidade e o estado do usuário baseando-se em sua latitude e longitude, capturados por uma API nativa de geolocalização dos próripos *broswers*. O *endpoint* é */data/reverse-geocode-client?latitude=${LATITUDE} &longitude=${LONGITUDE} &localityLanguage=en*

5.2 Fluxogramas

Os principais fluxogramas que representam as regras de negócio definidas para a aplicação Severino são mostrados na seção de Implementação. A leitura das figuras se dá da seguinte forma: as caixas verdes e azuis representam as telas, contudo as azuis são referentes à autenticação. As caixas que contém um cadeado, são aquelas que precisam de autenticação para serem acessadas e as que contém um ícone de localização, utilizam uma lógica específica para pedir a localização do usuário antes de serem acessadas, exibindo um modal caso o cliente ainda não tenha informado sua localidade anteriormente e explicando a importância dessa informação. As linhas lisas representam o acesso via plataforma e ss linhas pontilhadas representam o acesso à páginas via *link*, seguindo a seguinte regra: ações de filtros ou paginação sempre devem refletir na URL do navegador, pois dessa maneira os usuários são capazes de receber e enviar links com o estado atual da tela para outras pessoas.

5.3 Implementação

5.3.1 Portal do Cliente

Após a prototipação e validação das interfaces, iniciou-se a implementação das páginas *web*, começando pelo Portal do Cliente, responsável por explicar os objetivos da aplicação para os usuários e instruí-los a utilizá-la. A figura 20 representa o fluxograma do Portal do Cliente e a hierarquia entre as páginas.

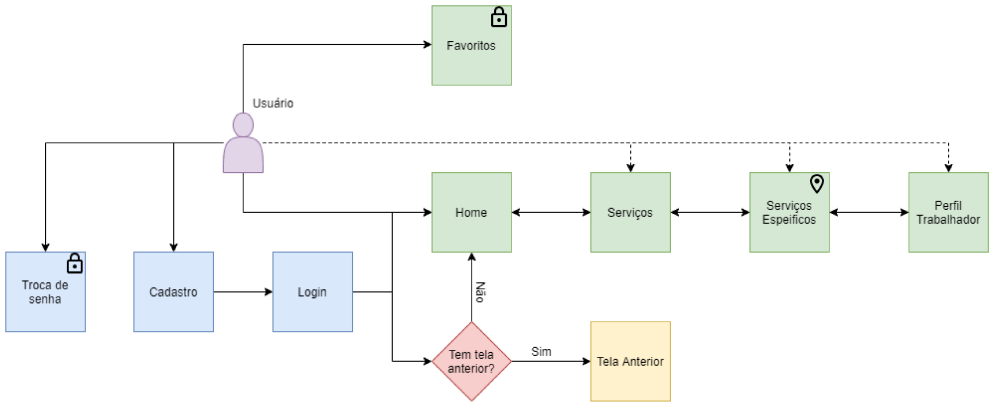

Figura 20 – Fluxograma explicativo do acesso à página *web* do cliente.

Para implementar a navegação entre as páginas, utilizou-se a biblioteca React Router DOM, exemplificada no código abaixo. É importante salientar que durante a implementação das rotas, criou-se rotas customizadas, que podem exigir que o usuário esteja logado ou que exija que ele informe sua localização para a aplicação. Link para o código

Como foi mostrado, existem rotas que precisam de autenticação para serem acessadas, como a página de Favoritos. Ao acessar uma rota protegida por autenticação, seu conteúdo somente é carregado se o usuário estiver logado. Caso não esteja, ele é redirecionado automaticamente para a página de login. Ao realizar login, o usuário é novamente desviado de forma automática para a página em que estava anteriormente e seus dados são persistidos no *local storage* do navegador e no contexto da aplicação para serem lembrados em futuros acessos à páginas protegidas.

Para implementar a autenticação, utilizou-se a Context-API do React e a biblioteca Axios para realizar as requisições à API de autenticação e resgatar o *token* JWT, que será utilizado no contexto da aplicação e no *Header* das requisições HTTP, para que o *back-end* e o *front-end* identifiquem que o usuário está logado ao realizar requisições ou ao acessar uma rota protegida. O fluxograma de autenticação é mostrado na figura 21 e o código da implementação da autenticação é mostrado abaixo.

Criação da instância do Axios

Criação do contexto de autenticação

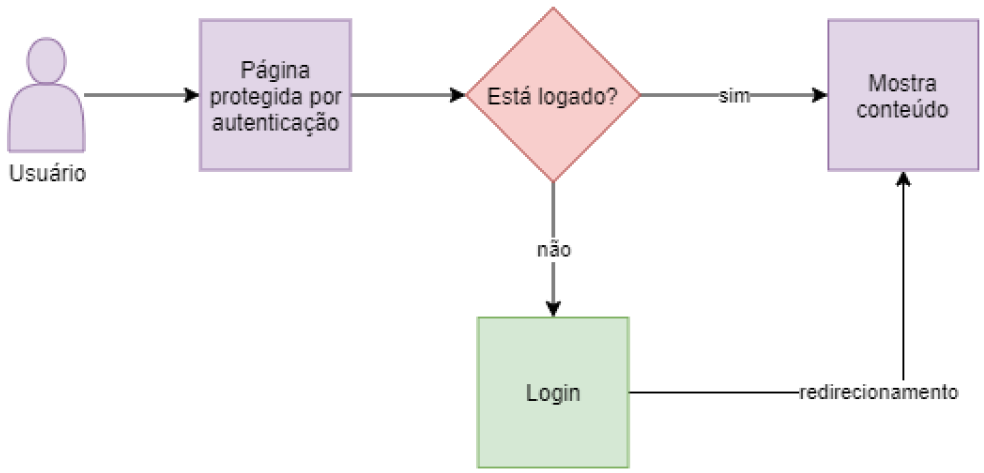

Figura 21 – Fluxograma de rota protegida.

Além da implementação das rotas, do serviço de comunicação com a API e do contexto te autenticação, outra implementação importante foi a da página de busca por serviços específicos, que contém as principais regras de negócio da aplicação e que demonstra bem o conceito de componentes explicado anteriormente, já que no código abaixo, pode ser notado somente a instância dos componentes, passando as propriedades para que eles funcionem, ou seja, a lógica de negócio é implementada no componente pai, mostrado abaixo, e a parte visual e a lógica de aplicação são implementadas nos componentes filhos, que não serão exemplificados neste trabalho. Link para o código

5.3.2 Portal do Profissional

Finalizando a implementação do Portal do Cliente, iniciou-se a implementação do Portal do Profissional, que é responsável por realizar o cadastro e manutenção do perfil dos usuários, além de explicar o conceito e os benefícios da aplicação para eles. A figura 22 representa o fluxograma do Portal do Profissional e a hierarquia entre as páginas.

A implementação da navegação, do contexto de autenticação e da comunicação com a API é muito semelhante à do Portal do Cliente, portanto será mostrado a implementação da criação de formulários, utilizando a biblioteca Formik e suas validações, utilizando a biblioteca Yup. O código abaixo exemplifica essas implementações. É importante salientar que algumas validações do Yup, como a de telefone e de confirmação de senha, não são nativas da biblioteca e por isso criou-se validações customizadas, como é mostrado no objeto *validations*, na chave *confirmPassword*. Link para o código

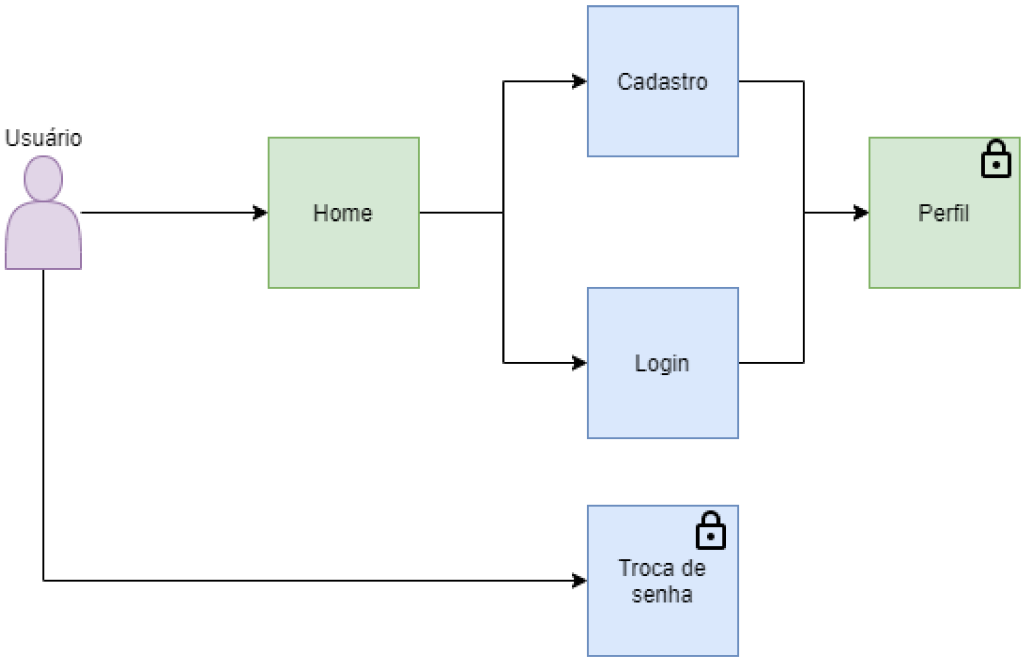

Figura 22 – Fluxograma explicativo do acesso à página *web* do profissional.

5.4 Páginas dos Portais

5.4.1 Portal do Cliente

5.4.1.1 Home

A página Home é a página inicial do aplicativo, responsável por, logo no início, convidar o usuário a buscar profissionais na plataforma, mostrar alguns serviços disponíveis na plataforma, explicar o conceito da plataforma e como utilizá-la. A figura 23 exemplifica a página Home.

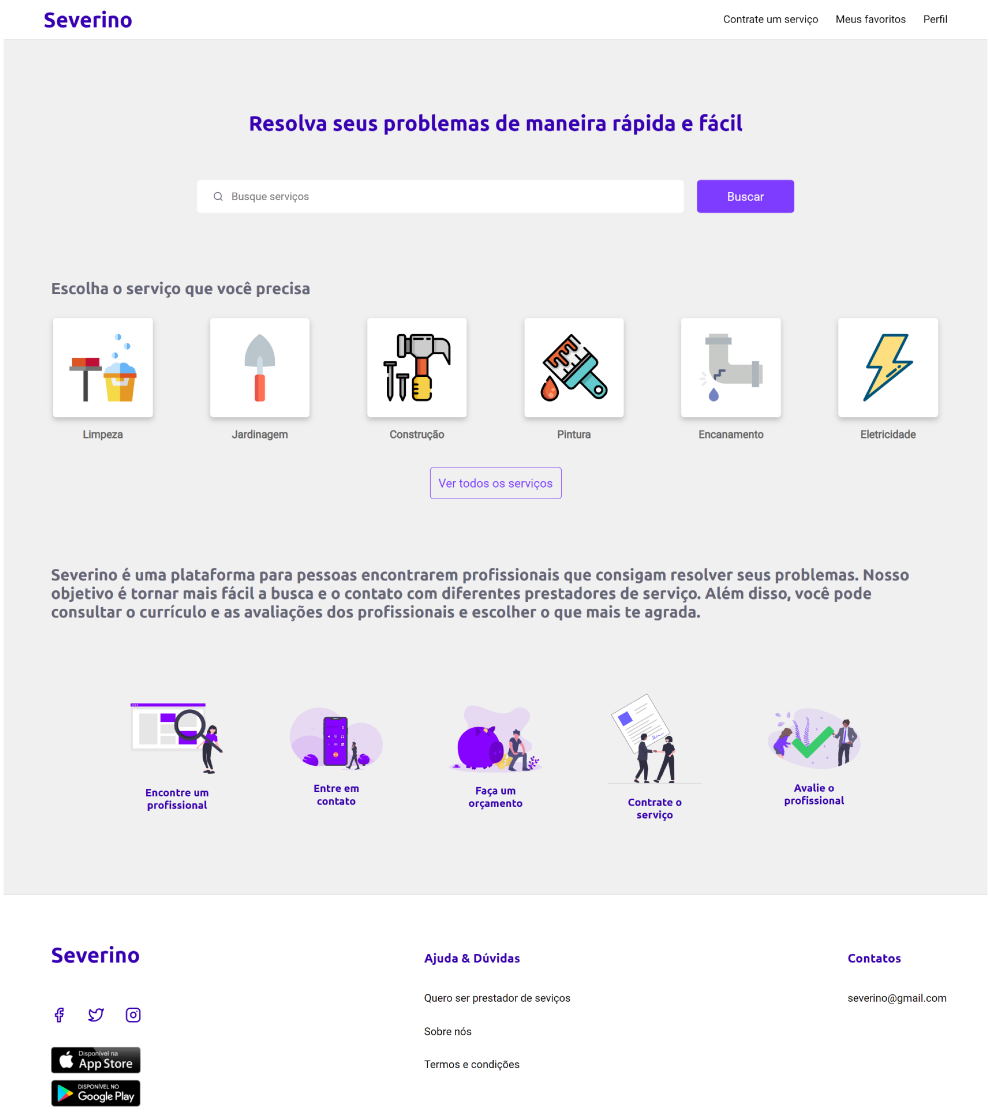

Figura 23 – Página Home do Portal do Cliente.

5.4.1.2 Favoritos

A página de Favoritos é responsável por mostrar os profissionais favoritados pelo usuário. É possível acessar o perfil dos profissionais clicando nos *cards*, retirá-lo da lista de favoritos clicando no ícone de coração e iniciar uma conversa via WhatsApp clicando no ícone do WhatsApp. A figura 24 exemplifica a página Favoritos.

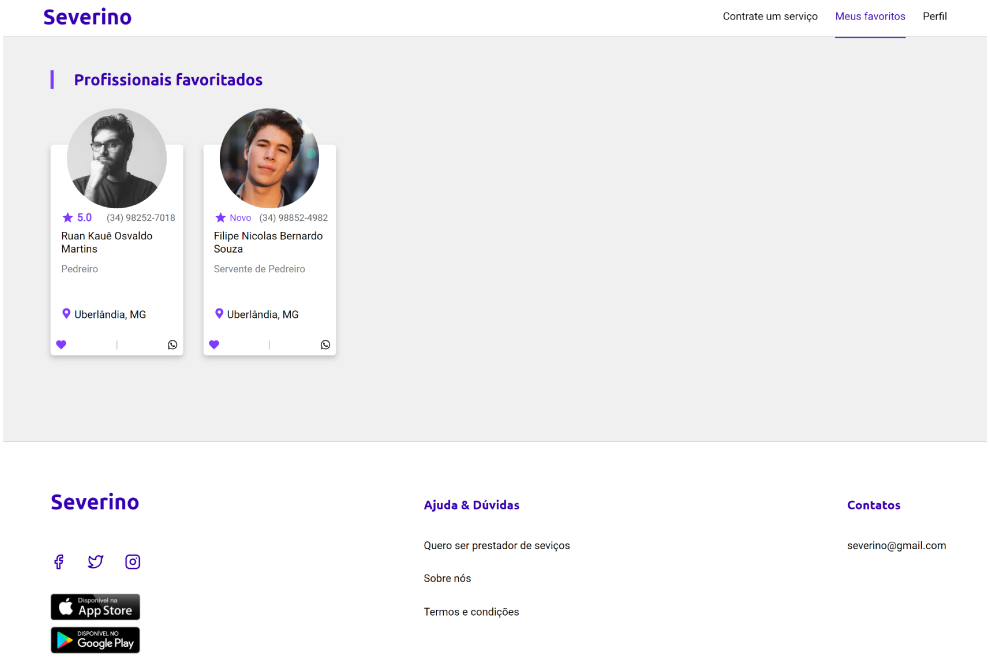

Figura 24 – Página Favoritos do Portal do Cliente.

5.4.1.3 Serviços Gerais

A página de Serviços Gerais é responsável por mostrar os serviços genéricos disponíveis na plataforma. Ao clicar em um serviço, o usuário é redirecionado para a página de Serviços Específicos do serviço clicado, por exemplo, ao clicar sobre o serviço de limpeza, será possível visualizar todos os profissionais que realizam serviços específicos de limpeza, como limpeza de carro, limpeza de casa, entre outros. A figura 25 exemplifica a página Serviços Gerais.

5.4.1.4 Serviços Específicos

A página de Serviços Específicos é responsável por mostrar os profissionais que realizam os serviços buscados. Pode-se filtrar os profissionais por localidade e por serviços que realizam, além de ser possível ordená-los por melhor avaliação. A figura 26 exemplifica a página Serviços Específicos.

5.4.1.5 Login

A página Login é onde os clientes conseguem se autenticar na aplicação, utilizando *e-mail* e senha. A figura 27 exemplifica a página Login.

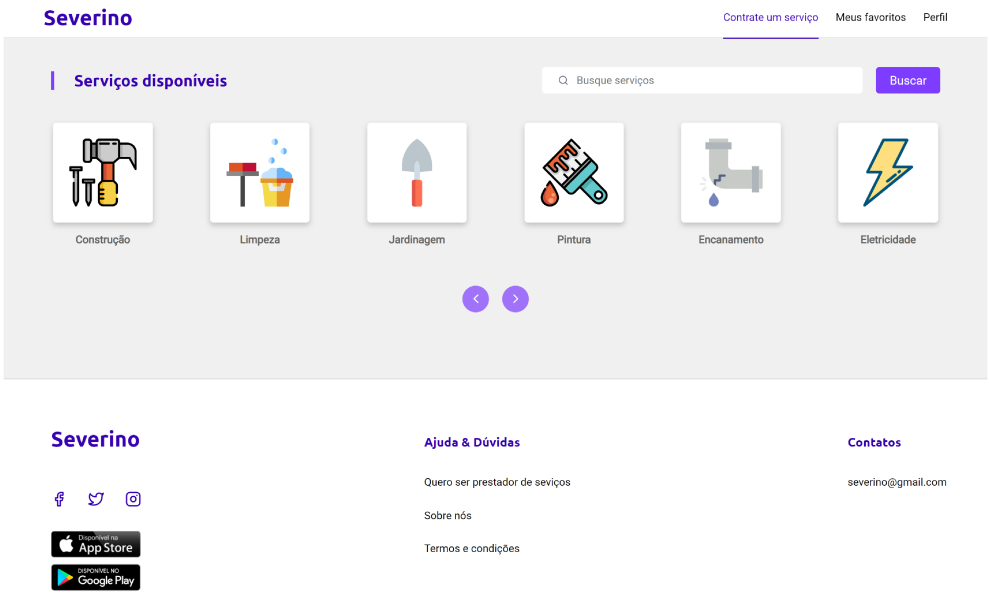

Figura 25 – Página Serviços Gerais do Portal do Cliente.

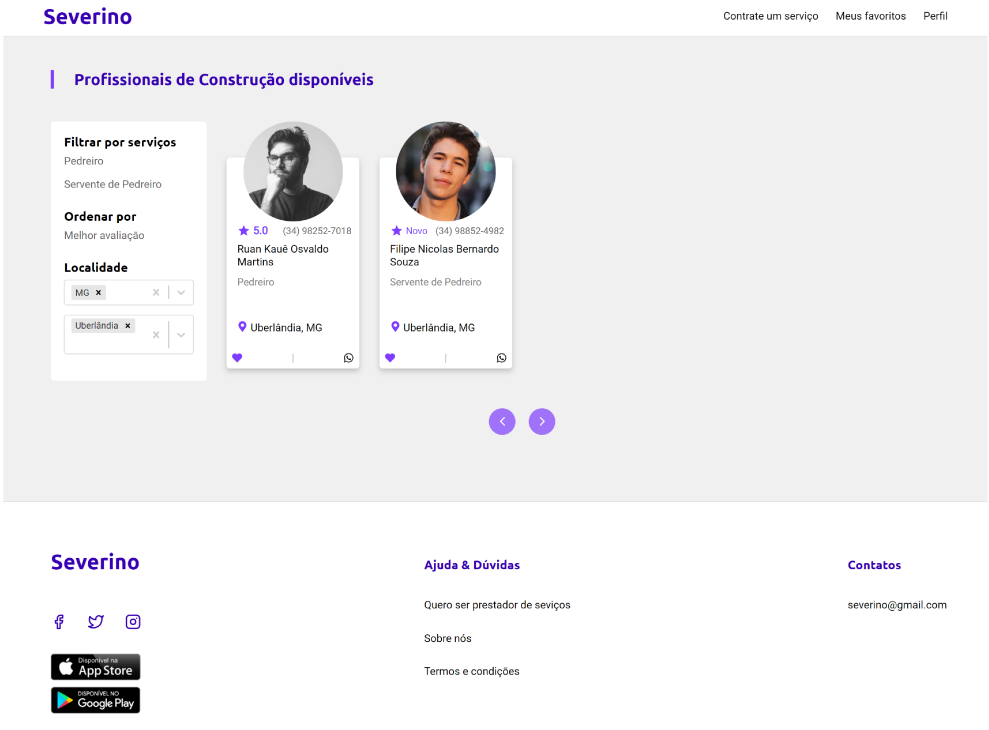

Figura 26 – Página Serviços Específicos do Portal do Cliente.

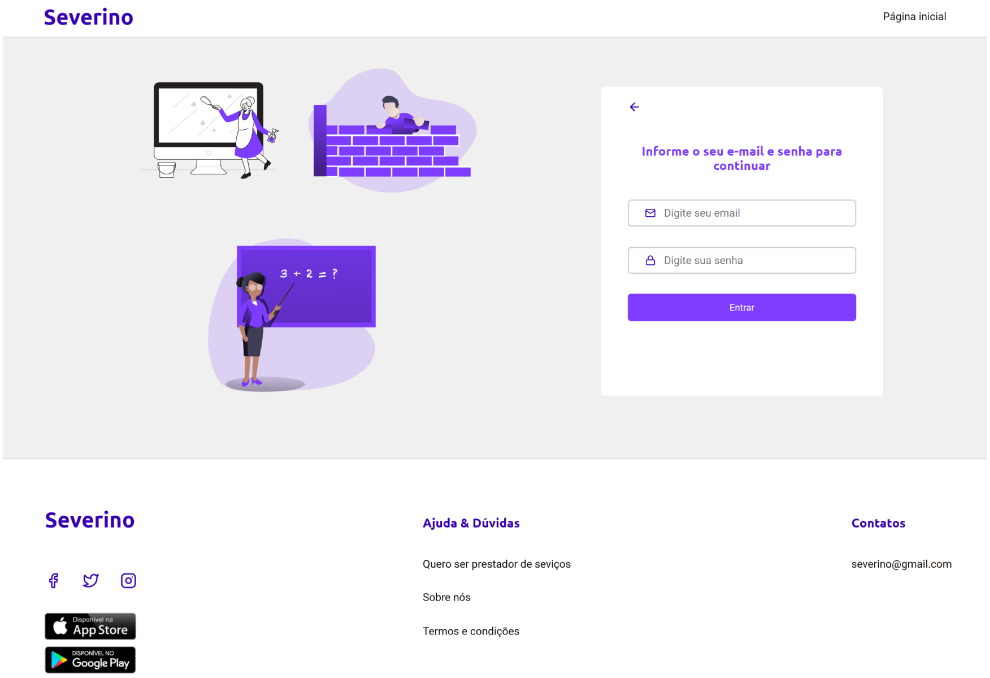

Figura 27 – Página Login do Portal do Cliente.

5.4.1.6 Cadastro

A página Cadastro é responsável por realizar o cadastro dos clientes na aplicação Severino. É necessário preencher todos os campos para realizar o cadastro. A figura 28 exemplifica a página Cadastro.

5.4.1.7 Troca de Senha

A página Troca de Senha é onde os usuários conseguem alterar sua senha. A figura 29 exemplifica a página Troca de Senha.

5.4.1.8 Perfil do Profissional

A página Perfil do Profissional será onde os clientes irão visualizar todos os dados do profissional, como foto, nome, avaliação média, serviços que ele realiza, fotos de serviços que já realizou, entre outros. Além disso, nessa página é possível adicionar comentários sobre o profissional ou denunciá-lo. A figura 30 exemplifica a página Perfil do Profissional.

5.4.2 Portal do Profissional

5.4.2.1 Home

A página Home é responsável por explicar o conceito da aplicação e por mostrar os benefícios de fazer parte do Severino, buscando atrair os profissionais, para que eles se

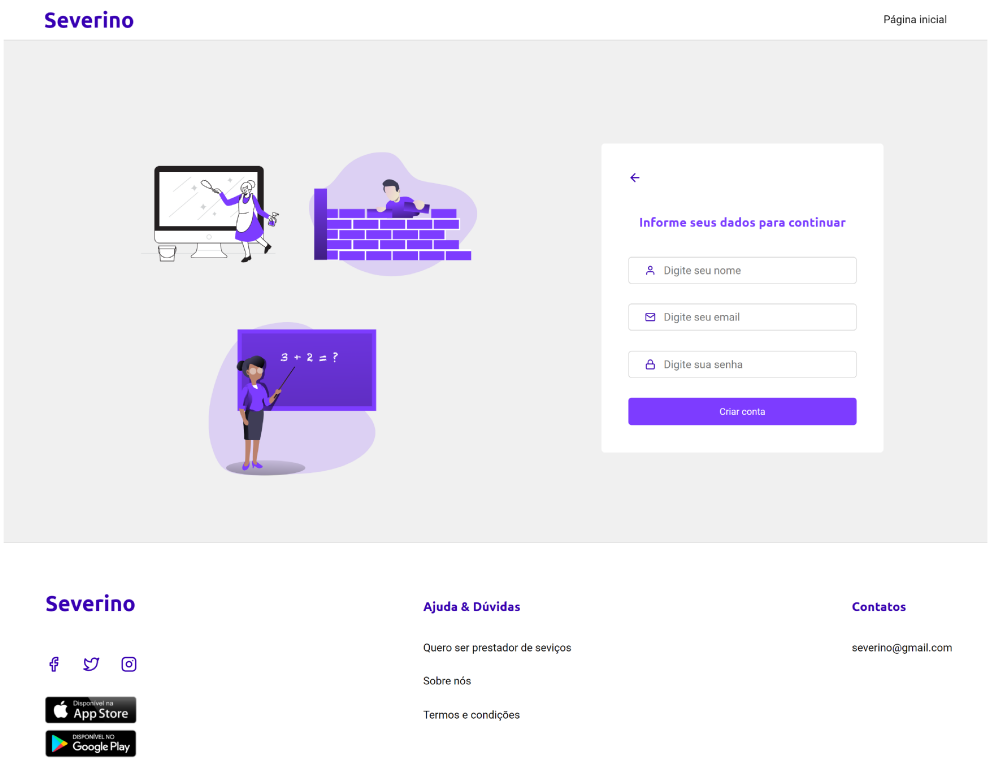

Figura 28 – Página Cadastro do Portal do Cliente.

cadastrem na aplicação. A figura 31 exemplifica a página Troca de Senha.

5.4.2.2 Login

A página Login é onde os profissionais conseguem se autenticar na aplicação, utilizando *e-mail* e senha. A figura 32 exemplifica a página Login.

5.4.2.3 Cadastro

A página Cadastro é responsável por realizar o cadastro dos clientes na aplicação Severino. É necessário preencher todos os campos para realizar o cadastro. A figura 33 exemplifica a página Cadastro.

5.4.2.4 Troca de Senha

A página Troca de Senha é onde os usuários conseguem alterar sua senha. A figura 34 exemplifica a página Troca de Senha.

5.4.2.5 Perfil

A página Perfil é onde os profissionais conseguem visualizar suas informações e editá-las. Inicialmente a aba de Dados Gerais fica aberta, mostrando os dados e para editá-los

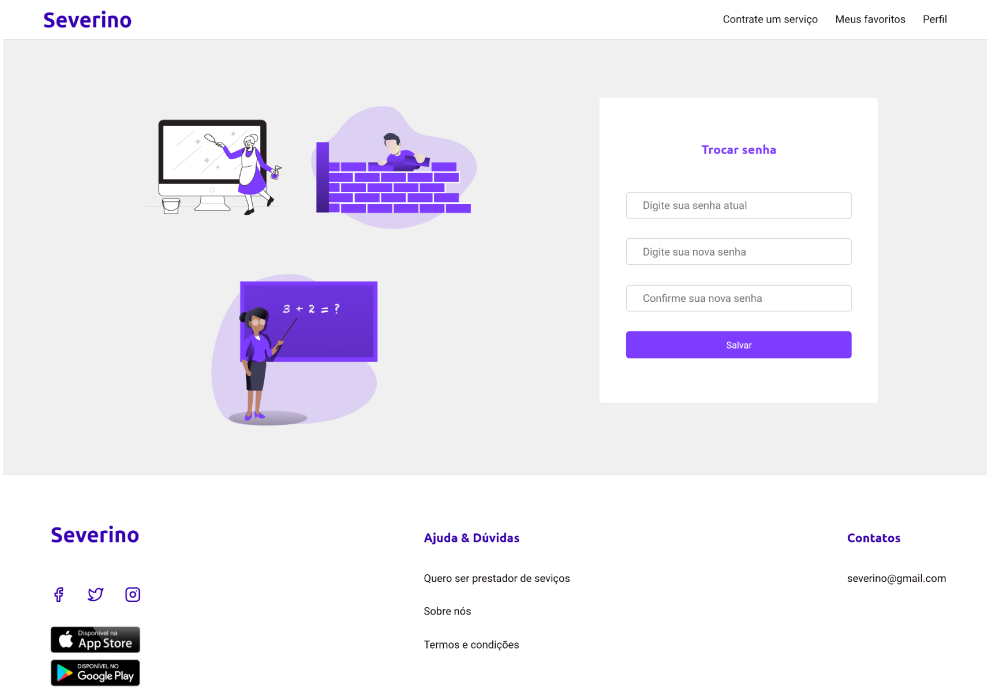

Figura 29 – Página Troca de Senha do Portal do Cliente.

é necessário clicar em Editar, assim, os *inputs* se tornam editáveis e o botão troca sua *label* para Salvar. Só uma aba pode ser aberta por vez, assim, ao abrir uma aba, a que está aberta é fechada automaticamente. A figura 35 exemplifica a página Perfil.

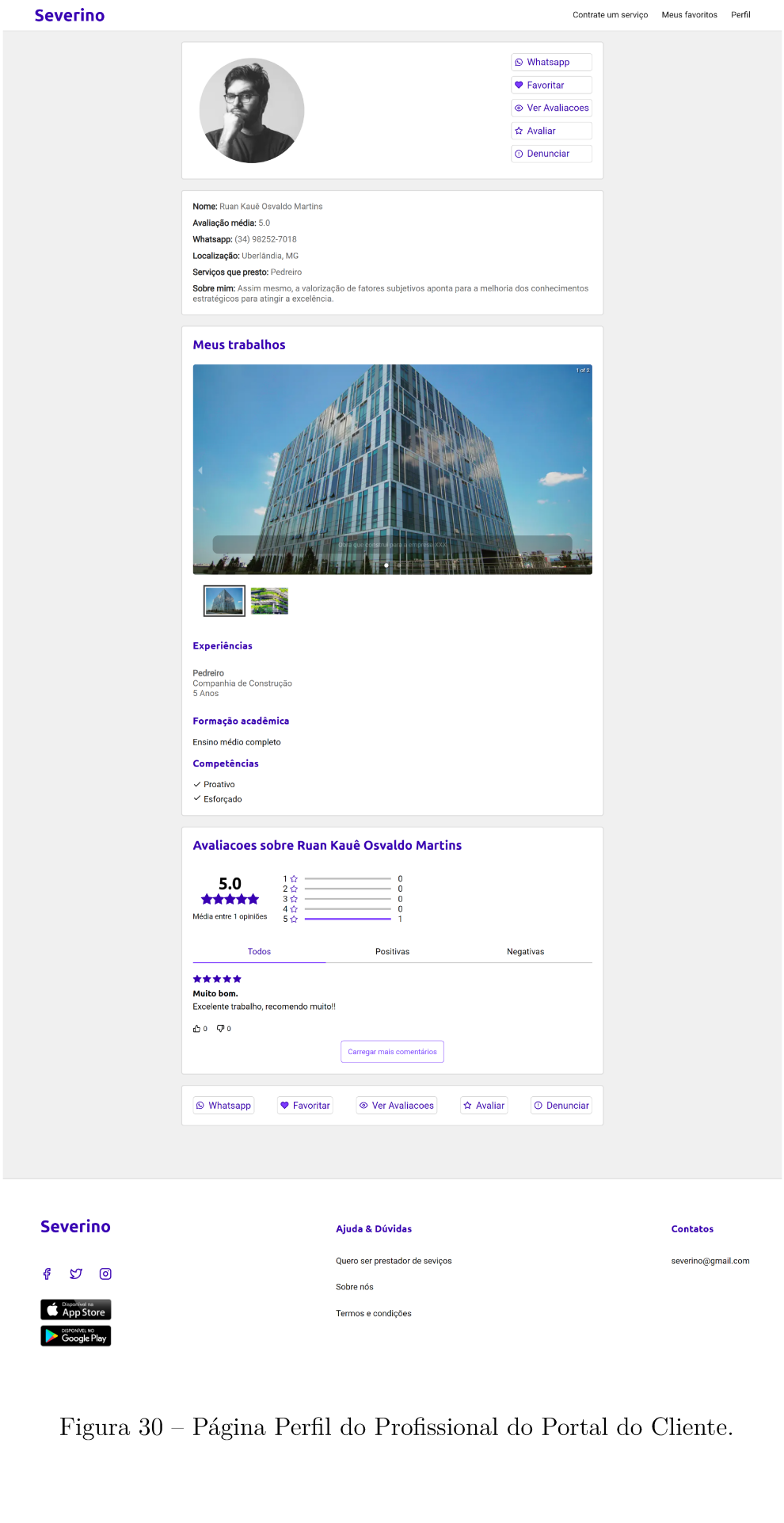

Figura 30 – Página Perfil do Profissional do Portal do Cliente.

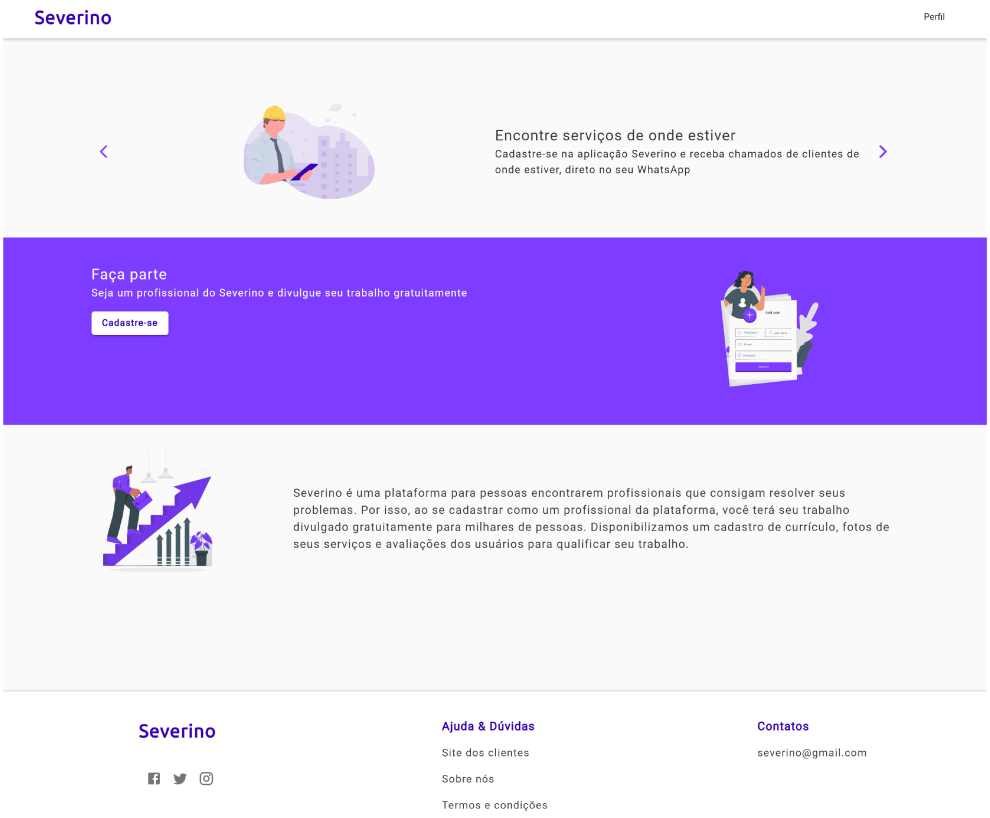

Figura 31 – Página Home do Portal do Profissional.

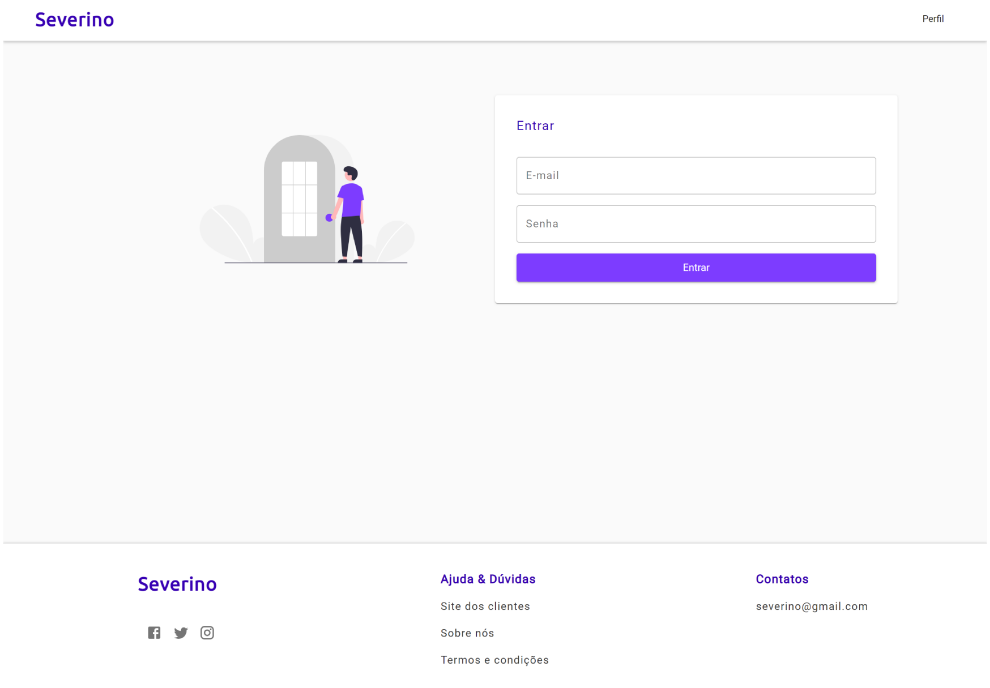

Figura 32 – Página Login do Portal do Profissional.

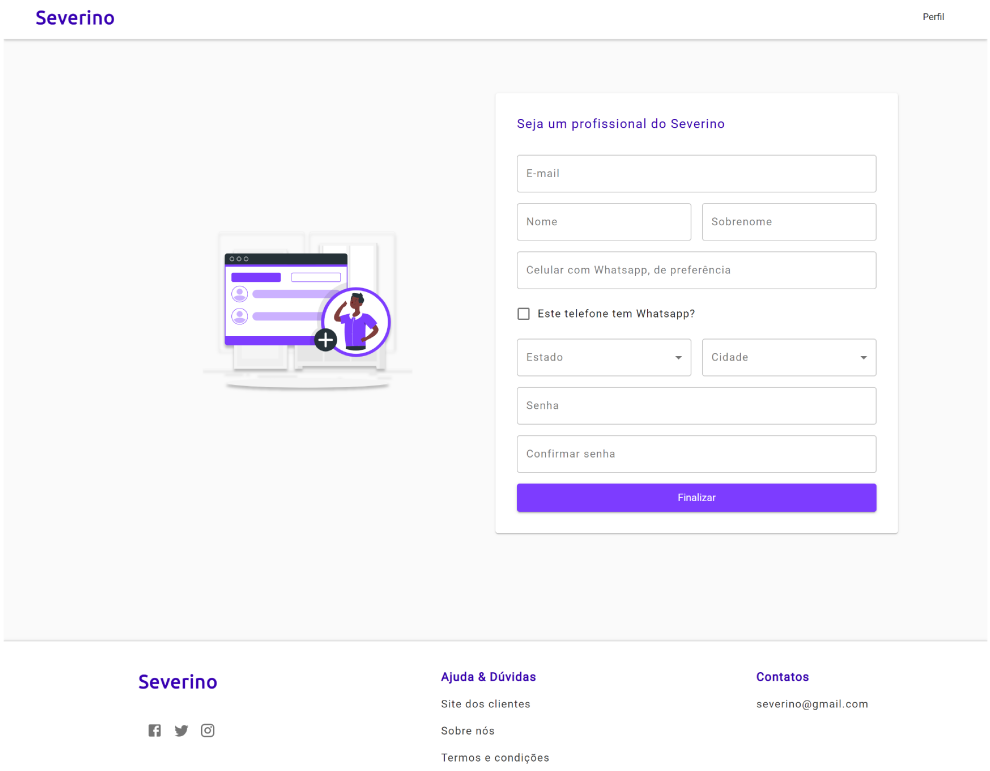

Figura 33 – Página Cadastro do Portal do Profissional.

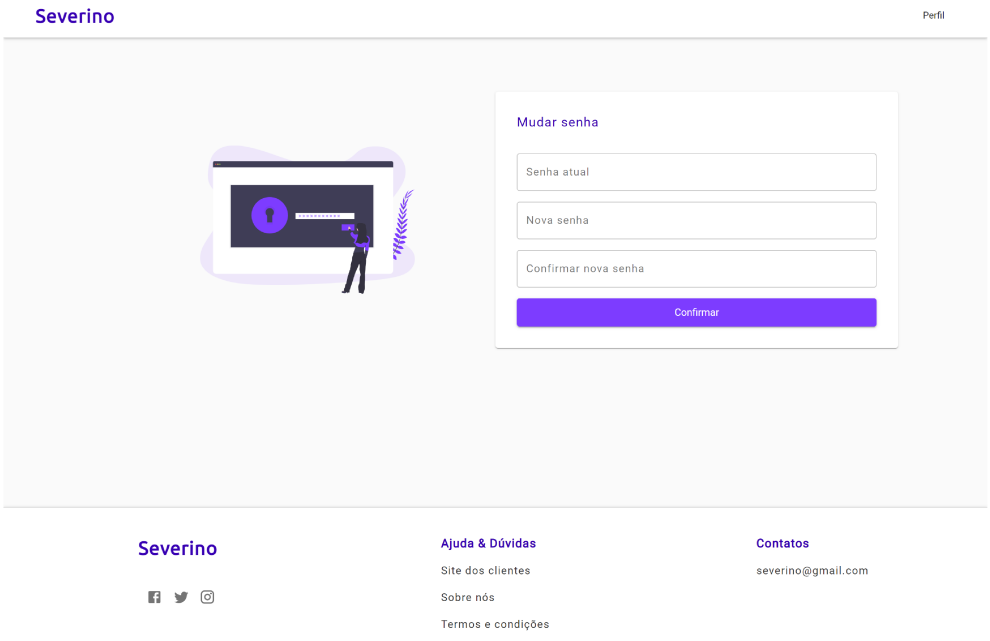

Figura 34 – Página Troca de Senha do Portal do Profissional.

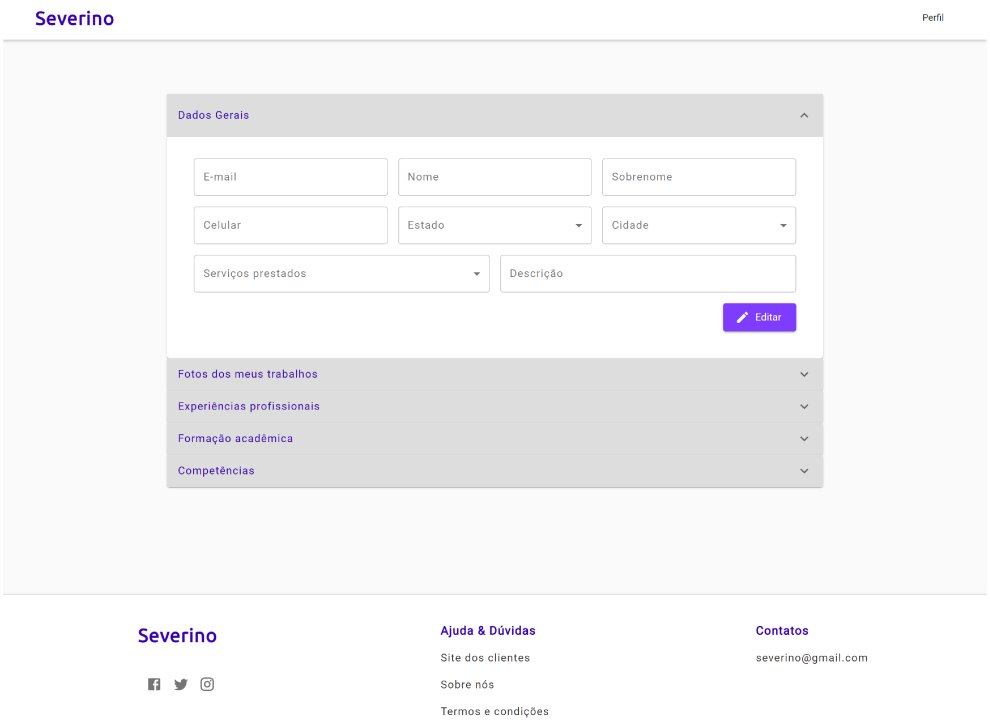

Figura 35 – Página Perfil do Portal do Profissional.

Capítulo 6

Conclusão

Com a adesão da população à sistemas digitais, muitas pessoas deixaram de utilizar a lista telefônica, que era um ótimo meio de encontrar profissionais e de anunciar serviços. Com isso, surgiram algumas plataformas para substituí-la na era digital, contudo a maioria delas necessita de cadastro e do preenchimento de vários formulários, bem como a confirmação de *e-mails* e outras etapas complicadas para enfim encontrar o profissional desejado, o que afasta muitos usuários desses meios, pois nem todas as pessoas conseguem chegar ao fim desse processo, seja por falta de paciência ou por falta de conhecimento tecnológico.

Visto isso, identificou-se a necessidade e oportunidade de criar um sistema fácil e rápido de utilizar, que, com apenas dois cliques, consegue encontrar profissionais na mesma cidade do cliente, sem necessidade de cadastro prévio. Além disso, para os profissionais, também foi facilitado a entrada na plataforma, com uma interface didática e que possui um cadastro ínfimo para ingressar na base de dados do sistema. Assim, a praticidade das plataformas do cliente e do profissional se torna um enorme diferencial frente a outros concorrentes no mercado, criando uma maior possibilidade de aceitação e migração do publico para um novo *software*.

6.1 Desafios encontrados

Durante o planejamento e desenvolvimento do *software* algumas dificuldades foram encontradas, principalmente em relação à definição de regras de negócio, estilização e usabilidade, já que esses pontos são os que mais iriam diferenciar esse sistema dos demais existentes no mercado. Graças à disciplina de Engenharia de Software, foi mais fácil modelar e criar diagramas para entender como a construção do sistema se daria de forma geral, mesmo que fossem só rascunhos.

Elaborar o *layout* e os protótipos do sistema também representou uma dificuldade devido ao pouco conhecimento na área de *design* e de experiência do usuário, que não foram muito abordadas durante o curso. Contudo, o aprendizado que veio tanto do

estágio facultativo quanto do obrigatório, ajudou muito nesse ponto, devido ao contato com sistemas reais e com boas interfaces.

Além disso, criar uma arquitetura e um padrão de projeto utilizando tantas bibliotecas e tecnologias também foi um desafio superado, já que precisou-se não só do conhecimento desconjuntado de cada um, mas sim de encontrar uma maneira de ligar todos eles buscando criar um código que fosse ao mesmo tempo escalável e de fácil entendimento. Todavia, graças ao conhecimento adquirido em várias matérias do curso, como, Análise e Estrutura de Dados, Modelagem de Software, Programação Orientada a Objetos e Programação para Internet, bem como conhecimentos externos à faculdade, essa parte técnica foi superada com mais facilidade do que as outras citadas.

6.2 Trabalhos Futuros

Este projeto de graduação representa uma versão inicial da aplicação Severino e que já pode ser utilizada em larga escala por todo o Brasil, porém, como todo *software* que busca a excelência, esse contará com algumas funcionalidades 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 na aplicação atual.
- Caso mais desenvolvedores façam parte de funcionalidades futuras, 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

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

AMAZON. **AWS S3**. 2018. Amazon. Disponível em: <https://aws.amazon.com/pt/s3/>. Acesso em: 23 mai 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. Cnnbrasil.com.br. Disponível em: <https://www.cnnbrasil.com.br/nacional/2020/04/01/em-meio-a-pandemia-profissionais-autonomos-buscam-alternativas-para-ter-renda>. Acesso em: 20 abr 2021.

CAVALLINI, M. **Pandemia aumenta busca por profissionais autônomos e freelancers no país; veja serviços com maior demanda**. 2020. G1.globo.com. 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: 20 abr 2021.

DEVELOPER.MOZILLA.ORG. **Lazy loading**. 2021. Https://developer.mozilla.org/en-US/docs/Web/Performance/Lazy*loading.Disponívelem* : <>. Acesso em: 18 abr 2021.

DEVELOPERS.GOOGLE.COM. **Single Page Application Measurement**. 2021. Https://develo
page-applications?hl=en. Disponível em: <https://developers.google.com/analytics/devguides/collection/analyticsjs/single-page-applications?hl=en>. Acesso em: 16 abr 2021.

DOCUMENTATION, S. C. **Styled Components**. 2021. Styled-components.com. Disponível em: <https://styled-components.com/docs/>. Acesso em: 14 abr 2021.

FORMIK. **Formik**. 2020. Formik. Disponível em: <https://formik.org/>. Acesso em: 23 mai 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.

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.

ICONS react. **React Icons**. 2021. React-icons. Disponível em: <https://react-icons.github.io/react-icons/>. Acesso em: 14 abr 2021.

INTERNATIONAL, E. **ES6**. 2015. 262.ecma-international.org. Disponível em: <https://262.ecma-international.org/6.0/>. Acesso em: 10 may 2021.

JAZAYERI, M. Some trends in web application development. In: IEEE. **Future of Software Engineering (FOSE'07)**. [S.l.], 2007. p. 199–213.

KRYUKOV, D. **Yarn vs. npm in 2019: Choosing the Right Package Manager for the Job**. 2019. Https://soshace.com/. Disponível em: <https://soshace.com/yarn-package-manager-in-2019-should-we-keep-on-comparing-yarn-with-npm/>. Acesso em: 19 abr 2021.

MAN, M. **Triider planeja faturar R$ 12 milhões em 2021**. 2021. Dci.com.br. Disponível em: <https://www.dci.com.br/tecnologia-e-games/triider-startup-pequenas-reformas-faturamento-2123152/>. Acesso em: 20 abr 2021.

MASSE, M. **REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces**. [S.l.]: "O'Reilly Media, Inc.", 2011.

MEDIA, D. **What is Git**. 2021. Dev Media. Disponível em: <https://www.devmedia.com.br/introducao-ao-visual-studio-code/34418>. Acesso em: 15 abr 2021.

MESH, J. **Kanban 101: How Any Team Can Be More Agile**. 2020. Https://blog.trello.com/. Disponível em: <https://blog.trello.com/kanban-101>. Acesso em: 16 abr 2021.

MOHAPATRA, H.; RATH, A. K. **Fundamentals of Software Engineering: Designed to provide an insight into the software engineering concepts**. [S.l.]: BPB Publications, 2020.

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.

REACT.JS. **Hooks React.js**. 2021. React.js. Disponível em: <https://reactjs.org/docs/hooks-intro.html>. Acesso em: 14 abr 2021.

REACTJS.ORG. **React A JavaScript library for building user interfaces**. 2021. Https://reactjs.org/. Disponível em: <https://reactjs.org/>. Acesso em: 16 abr 2021.

TO, D. **What is Axios?** 2021. Dev to. Disponível em: <https://dev.to/veewebcode/what-is-axios-and-how-to-use-it-4an1>. Acesso em: 15 abr 2021.

TRABALHO, O. I. do. **Trabalho Doméstico**. 2021. Organização Internacional do Trabalho. Disponível em: <https://www.ilo.org/brasilia/temas/trabalho-domestico/lang--pt/index.htm>. Acesso em: 20 abr 2021.

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

W3SCHOOL. **Introduction to HTML**. 2013. W3schoold. Disponível em: <https://www.w3schools.com/html/html_intro.asp>. Acesso em: 24 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.