UNIVERSIDADE EVANGÉLICA DE GOIÁS - UNIEVANGÉLICA
ENGENHARIA DE SOFTWARE

GABRIEL HENRIQUE SILVA
HENRIQUE VALEIRO DE SOUZA
JÚLIO CÉSAR CAIXETA RODRIGUES DA CUNHA
VINICIUS REIS CAMPOS

DESENVOLVIMENTO DE UM MÍNIMO PRODUTO VIÁVEL PARA ARRECADAÇÃO E DISTRIBUIÇÃO DE VESTIMENTAS PARA PESSOAS NECESSITADAS

Anápolis
2022

UNIVERSIDADE EVANGÉLICA DE GOIÁS - UNIEVANGÉLICA
ENGENHARIA DE SOFTWARE

GABRIEL HENRIQUE SILVA
HENRIQUE VALEIRO DE SOUZA
JÚLIO CÉSAR CAIXETA RODRIGUES DA CUNHA
VINICIUS REIS CAMPOS

DESENVOLVIMENTO DE UM MÍNIMO PRODUTO VIÁVEL PARA ARRECADAÇÃO E DISTRIBUIÇÃO DE VESTIMENTAS PARA PESSOAS NECESSITADAS

Trabalho apresentado ao Curso de Engenharia de Software da Universidade Evangélica de Goiás – UniEVANGÉLICA, da cidade de Anápolis-GO como requisito parcial para obtenção do Grau de Bacharel em Engenharia de Software.

Orientador: Prof. M.e. Alexandre Moraes Tannus

Anápolis
2022

UNIVERSIDADE EVANGÉLICA DE GOIÁS - UNIEVANGÉLICA

ENGENHARIA DE SOFTWARE

GABRIEL HENRIQUE SILVA
HENRIQUE VALEIRO DE SOUZA
JÚLIO CÉSAR CAIXETA RODRIGUES DA CUNHA
VINICIUS REIS CAMPOS

DESENVOLVIMENTO DE UM MÍNIMO PRODUTO VIÁVEL PARA ARRECADAÇÃO E DISTRIBUIÇÃO DE VESTIMENTAS PARA PESSOAS NECESSITADAS

Monografia apresentada para Trabalho de Conclusão de Curso de Engenharia de Software da Universidade Evangélica de Goiás - UniEVANGÉLICA, da cidade de Anápolis-GO como requisito parcial para obtenção do grau de Engenheiro(a) de Software.

Aprovado por:

_____

(ORIENTADOR)

_____

(AVALIADOR)

Anápolis

2022

FICHA CATALOGRÁFICA

SILVA, Gabriel Henrique; SOUZA, Henrique Valeiro; CUNHA, Júlio César Caixeta Rodrigues; CAMPOS, Vinicius Reis; **Desenvolvimento de um Mínimo Produto Viável para Arrecadação e Distribuição de Vestimentas para Pessoas Necessitadas.** Anápolis, 2022. (Universidade Evangélica de Goiás – UniEVANGÉLICA, Engenheiro(a) de Software, 2022).
Monografia. Universidade Evangélica de Goiás, Curso de Engenharia de Software, da cidade de Anápolis-GO.
1. MVP. Vestimentas. Doação.

REFERÊNCIA BIBLIOGRÁFICA

SILVA, Gabriel Henrique; SOUZA, Henrique Valeiro; CUNHA, Júlio César Caixeta Rodrigues; CAMPOS, Vinicius Reis; **Desenvolvimento de um Mínimo Produto Viável para Arrecadação e Distribuição de Vestimentas para Pessoas Necessitadas.** Anápolis, 2022. 61 páginas. Monografia - Curso de Engenharia de Software Universidade Evangélica de Goiás - UniEVANGÉLICA.

CESSÃO DE DIREITOS

NOMES DOS AUTORES: GABRIEL HENRIQUE SILVA, HENRIQUE VALÉRIO DE SOUZA, JÚLIO CÉSAR CAIXETA RODRIGUES DA CUNHA, VINICIUS REIS CAMPOS

TÍTULO DO TRABALHO: DESENVOLVIMENTO DE UM MÍNIMO PRODUTO VIÁVEL PARA ARRECADAÇÃO E DISTRIBUIÇÃO DE VESTIMENTAS PARA PESSOAS NECESSITADAS

GRAU/ANO: Graduação / 2022

_____
Gabriel Henrique Silva

_____
Henrique Valério de Souza

_____
Júlio César Caixeta Rodrigues da Cunha

_____
Vinicius Reis Campos

RESUMO

Ações sociais como doações e serviços voluntários são temas sempre em alta para se obter contribuições para diversas causas, sejam elas naturais ou não. A evolução das tecnologias digitais e o acesso a uma gama cada vez maior da população à internet, possibilita o surgimento de plataformas para contribuir com causas sociais e a disseminação de informações pelas redes sociais. Vendo que diversas pessoas têm dificuldade de encontrar lugares para doações de vestimentas devido a não conhecer pessoas que realizam esse tipo de ação. Este trabalho tem como objetivo desenvolver um software para a arrecadação e distribuição de vestimentas para pessoas necessitadas. O software será desenvolvido usando o formato Mínimo Produto Viável - MVP.

Palavras-chave: Ações Sociais; Doações; Campanhas; Vestimentas; MVP.

ABSTRACT

Social actions such as donations and voluntary services are always a hot topic to obtain contributions to various causes, whether natural or not. The evolution of digital technologies and the access to an increasing range of the population to the internet, makes possible the emergence of platforms to contribute to social causes and the dissemination of information through social networks. Seeing that many people have difficulty finding places for clothing donations due to not knowing people who carry out this type of action. This work aims to develop a software for the collection and distribution of clothing to people in need. The software will be developed using the Minimum Viable Product - MVP format.

Keywords: Social Actions; Donations; Campaigns; Clothing; MVP.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

Siglas	Descrição
API	Application Programming Interface
CSS	Cascading Style Sheets
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
JSON	JavaScript Object Notation
MVP	Minimum Viable Product
ONG	Organização Não Governamental
TDD	Test Driven Development
URLs	Uniform Resource Locators
WEB	World Wide Web

SUMÁRIO

1 INTRODUÇÃO

Seja por questões de administração econômica, acontecimentos climáticos ou desastres causados pela humanidade, sempre há um número alto de pessoas que necessitam de doações. Alimentos, moradia, cobertores e vestimentas, são exemplos de recursos que são arrecadados para ajudar pessoas que passam por dificuldades como as acima citadas. Órgãos governamentais, ONGs, pessoas jurídicas e físicas criam ações para arrecadamento de suprimentos, buscando ajudar ao próximo. Como por exemplo da prefeitura de Campinas, que lançou a Campanha do Agasalho 2022, teve início no mês de maio e já distribuiu 751 cobertores às pessoas em vulnerabilidade (ACIDADEON, 2022).

As campanhas para arrecadação de doações ocorrem normalmente através de veículos de comunicação como panfletos, noticiários, redes sociais e comunicação verbal, mas entidades relatam dificuldades nas realizações campanhas de doações causadas pela pandemia do novo coronavírus (GUIMARÃES; PUCHTA; ANDRADE, 2021). No entanto, com o avanço das tecnologias digitais, novas formas de divulgação podem surgir e serem mais eficazes, otimizando o alcance da divulgação e arrecadação de produtos.

Considerando todo o impacto da internet nos tempos atuais, o desenvolvimento de um software para a arrecadação e distribuição de vestimentas para pessoas necessitadas se faz relevante? Com isso, a proposta deste trabalho é desenvolver um sistema web com o foco de cadastrar instituições para a arrecadar e distribuir vestimentas. A página vai mostrar um mapa tendo todas as informações para realizar a doação.

2 OBJETIVOS

2.1 Objetivo Geral

Desenvolver um software no formato MVP para facilitar a dinâmica de arrecadar e distribuir vestimenta para pessoas necessitadas.

2.2 Objetivos Específicos

- Realizar um formulário para verificar a viabilidade e a relevância do software;
- Realizar o levantamento de requisitos necessários para o desenvolvimento do software;

- Mapear de forma lógica a base de dados para a armazenagem correta dos dados;
- Prototipar as telas para ter uma visualização prévia de como será a interface;
- Desenvolver o backend e frontend da aplicação;
- Realizar testes para verificar o nível de qualidade do sistema;

3 JUSTIFICATIVA

A doação de vestimentas é algo que ocorre no mundo todo. Como exemplo, a cidade de Anápolis, teve no último dia 10 de junho, uma arrecadação de 500 itens por uma iniciativa da Ala 2 da Base Aérea de Anápolis utilizando como canal principal de veiculação o site da prefeitura (ANÁPOLIS, 2021). Partindo para a capital de Goiás, Goiânia utilizou seu canal de comunicação oficial para divulgar a campanha de doação de agosto de 2021 e fez uso de número fixo como canal principal para doações (GOIÂNIA, 2021).

Tendo em vista o esforço manual para a divulgação e realização de doações, foi realizada uma pesquisa em diversas plataformas como a Google, Play Store, Apple Store, Microsoft Store, Samsung Store, na intenção de encontrar uma aplicação que divulgasse locais para doar e receber vestimentas. Após a realização de uma busca, foram encontrados alguns aplicativos que cumprem parcialmente a ideia proposta neste estudo, um exemplo é o aplicativo Paraná Solidário.

O aplicativo Paraná Solidário é um sistema com o intuito de promover a doação direta entre os cidadãos e as entidades beneficentes, fazendo uma ponte com empresas e pessoas físicas que tenham interesse em fazer doações (PARANÁ, 2019). Entretanto, o aplicativo possui alcance apenas para a localização do Paraná, enquanto o sistema que será desenvolvido, procura abranger uma área maior de alcance, podendo futuramente ser utilizado em qualquer localidade do país.

Diante desse cenário, esta pesquisa busca ajudar pessoas necessitadas por meio do desenvolvimento de um software para arrecadação, distribuição e divulgação de locais para doar e receber vestimentas. Por meio desse software, surge a oportunidade de ampliar o canal de comunicação entre o doador e as partes interessadas em realizar a coleta e a distribuição.

Com o acesso ao sistema, a pessoa poderá buscar entidades que arrecadarão vestimentas nas proximidades da região. Pessoas, empresas e órgãos locais serão avaliados e cadastrados no sistema por um administrador em primeiro momento, trazendo ao sistema e ao local de doação mais segurança e confiabilidade.

4. FUNDAMENTAÇÃO TEÓRICA

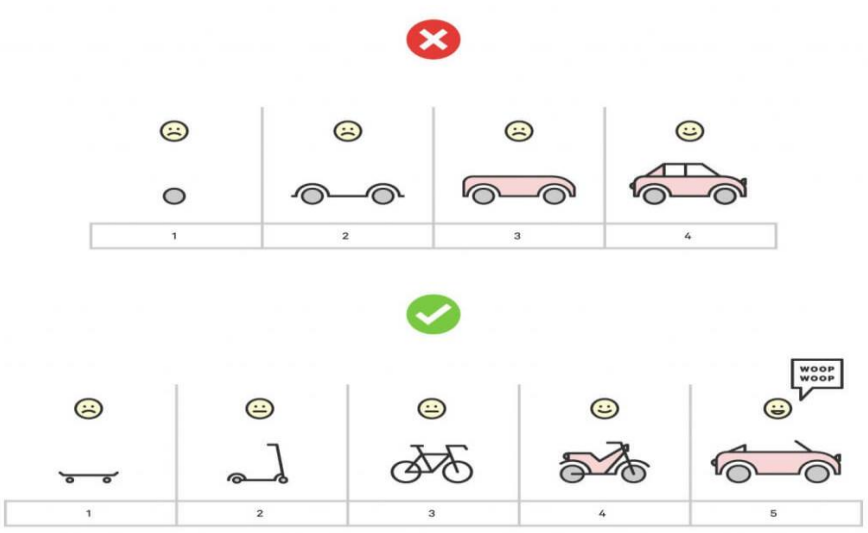
Durante o desenvolvimento do capítulo, será falado sobre o mínimo produto viável, o que ele é e seu conceito, também será apresentado os processos de desenvolvimento e o desenvolvimento Web. Onde será apresentado as ferramentas e os meios que serão utilizados.

4.1 Mínimo produto viável (MVP)

O surgimento do termo mínimo produto viável ou do inglês *minimum viable product* - MVP, vem de encontro com o conceito de *Startup* Enxuta, ou *Lean Startup*. De acordo com Ries (2011), o MVP é aquela versão do produto que permite uma volta completa do ciclo construir-medir-aprender, com o mínimo de esforço e o menor tempo de desenvolvimento".

O conceito mais básico e propagado de um MVP é a construção de um carro, ao qual inicia o desenvolvimento do produto realizando da forma mais simples possível, incluindo apenas rodas e um suporte para que possa ficar em cima. Após lançar essa versão, são colhidos os *feedbacks* dos usuários e então um novo ciclo será realizado para aprimorar o produto, como vemos a seguir na Figura 1.

Figura 1 - MVP – Modo correto e modo incorreto



Fonte: MVP - Serasa Empreendedor (2018)

De acordo com Ries (2009), "O mínimo produto viável é a versão de um novo produto que permite que uma equipe colete o máximo de aprendizado validado sobre os clientes com o

mínimo esforço". O MVP não se trata de construir um produto final rápido, mas sim uma primeira versão do produto que irá a público e com isso será validado se é viável ou não continuar com o desenvolvimento do projeto.

4.2 Processos de desenvolvimento

O *Scrum* é um *framework* ágil, um conjunto de boas práticas usadas no gerenciamento de projetos complexos e tem como base as equipes pequenas, as reuniões constantes e a colaboração dos envolvidos (SCHWABER; SUTHERLAND, 2013).

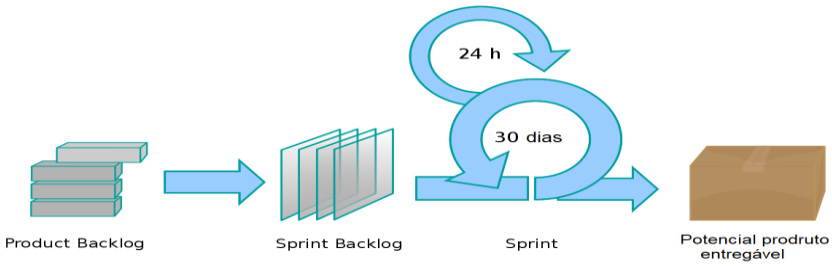
Segundo Prikladnicki (2014) o *Scrum* maximiza efetivamente a entrega de software, adaptando-se às novas realidades. Desenvolve os recursos mais valiosos com antecedência, enquanto considera se os recursos de prioridade mais baixa são necessários. Se mudanças forem necessárias, a equipe ágil pode facilmente alterar a prioridade. Sua principal motivação é que o desenvolvimento de software envolve muitas variáveis técnicas, como requisitos, recursos e tecnologia, que podem mudar no decorrer do processo, assim tornando-o imprevisível e precisando acompanhar as mudanças com flexibilidade.

A essência do *Scrum* é a *Sprint*, elas têm durações de acordo com o esforço de desenvolvimento da equipe de desenvolvimento. *Sprint* são os ciclos do projeto, ou seja, cada ciclo é um *Sprint* e uma se inicia assim que uma anterior é encerrada. É um evento *time-boxed* de até um mês, durante o incremento potencialmente utilizável do produto é criado, conforme mostrado na Figura 2, e as *sprints* têm uma duração consistente durante todo o esforço de desenvolvimento. O Schwaber e o Sutherland (2013), define as 3 funções principais dentro do Scrum, sendo elas:

- O *Product Owner* é o responsável pelo *backlog* do produto (uma lista de itens sobre os quais a equipe de desenvolvimento trabalhará no decorrer do projeto), por ajustar funcionalidades e prioridades a cada *sprint* e decidir a data de liberação e conteúdo das atualizações.
- O Time de desenvolvimento é um grupo de profissionais, responsável pelo desenvolvimento do produto, e a cada *sprint* gera um incremento do produto e que significa um avanço visível para os clientes do projeto.
- O *Scrum Master* é o responsável por facilitar o trabalho da equipe e garantir que o *Scrum* seja compreendido e aplicado, é a pessoa que dita os prazos, além de ser quem facilita e auxilia,

mas não é o gerente do projeto, garante a remoção dos impedimentos que dificultam o desenvolvimento do projeto.

Figura 2 - Ciclo da Sprint



Fonte: Autores

Com o processo ágil do *Scrum*, facilitando na identificação das tarefas e gestão das mesmas, será utilizado o TDD, que é o Desenvolvimento Orientado por Testes (*Test Driven Development*). De acordo Guedes (2020), os desenvolvedores criam um teste que irá falhar de todo jeito. Afinal, o mesmo ainda não foi implementado. Em seguida, a equipe desenvolve um código que satisfaça esse teste e então reaplicar a ele. Se o resultado é satisfatório, a equipe implementa o novo recurso ao código, e então partem para o desenvolvimento de um novo teste.

Essa primeira implementação deverá satisfazer imediatamente o teste que foi escrito no ciclo anterior. Ele se baseia em um ciclo curto de repetições, no qual para cada funcionalidade do sistema, um teste é criado antes. Esse ciclo é repetido até o final do projeto, quando o programa ou aplicativo é finalizado.

Com o TDD os testes serão escritos antes do código de produção e para cada novo ciclo, haverá um teste. Isto garante mais qualidade no desenvolvimento do código, já que eles deverão ser sempre refeitos e melhorados. Também dá mais segurança à equipe de desenvolvimento, com menos chances de o software apresentar algum bug ao final do processo. Segundo Kanner (2004), a automação mudou fundamentalmente a função dos testes em todo o processo de desenvolvimento.

O local para manter o repositório sempre atualizado e muito bem dividido, será utilizado o *Git*, é um sistema de gerenciamento e controle de versões. Desenvolvido em 2005 por Linus Torvalds, o mesmo criador do *kernel* do *Linux*. Muitos projetos utilizam o *Git* para versionar seus códigos. Com ele podemos criar projetos onde diversos usuários podem criar, editar, adicionar e excluir arquivos sem que tenham riscos de suas modificações serem sobrescritas.

O software de controle de versão armazena o registro de todas as alterações realizadas no código. Caso seja cometido um erro, os desenvolvedores podem voltar, restaurar uma versão

mais antiga, onde não existia o erro e ajudar a corrigir o código e ao mesmo tempo, reduzir a interferência para todos os membros da equipe.

Segundo Martin (2020), o tempo de verificação usando o *Git* foi reduzido a quase nada, a ideia nem existe mais. Assim, podendo fazer o comando *commit* que vai obter instantaneamente qualquer versão sempre que achar necessário. Com isso, é possível criar um conjunto de testes abrangente e rápido para testar quase tudo.

Pensando em agilizar ainda mais o processo com o *Git*, foi decidido utilizar o *GitFlow*, que é um tipo alternativo de *branch* do *Git* que consiste em ramificações de recursos e diversas ramificações primárias. Podemos definir o *GitFlow* como uma metodologia de trabalho, com foco na entrega de projetos. Seus ideais não geram nada de novo ao *Git*. Este modelo usa *branches* fixas. A *branch master* é utilizada como ponte entre o código principal e o de desenvolvimento. A *branch development* é utilizada para inserir novas modificações no código de desenvolvimento. Caso não haja erros, a modificação é inserida na *branch master*.

4.3 Desenvolvimento Web

Desde o começo da web, que vem crescendo de forma exponencial, algumas décadas atrás, a web fornecia apenas sites estáticos, simples e construídos usando somente uma linguagem de marcação que é chamada de HTML. Esses sites possuíam unicamente textos e imagens. Ao longo dos anos, a internet continuou a progredir, seguindo-se o aparecimento de outras tecnologias que contribuíram para o desenvolvimento do site.

Atualmente, a web não é mais apenas um ambiente de páginas simples e passou a ser tornar uma plataforma de aplicações. Como a revolução da web e com o surgimento de novas ferramentas para o desenvolvimento, as aplicações web passaram a se tornar mais difíceis. Assim o número de usuários e servidores disponíveis crescia substancialmente, a busca por aplicações cada vez melhores se fez indispensável, consequentemente, com o aumento de novas tecnologias envolvidas, foram exigindo cada vez mais de profissionais na área. Segundo Coulouris (2013), a web evoluiu bastante sem mudar sua arquitetura básica que são divididas em três básicos componentes:

- HTML (*HyperText Markup Language*), que é uma linguagem de marcação aplicada para estruturar elementos da página, como textos, tabelas, imagens e até vídeos.
- URLs (*Uniform Resource Locators*), é o endereço virtual de uma página Web, como exemplo: www.unievangelica.edu.br.

- Arquitetura cliente-servidor que utilizam o padrão de comunicação através do protocolo HTTP (*HyperText Transfer Protocol*), que os navegadores usam para se comunicar e buscar informações, como documentos e outros recursos.

Portanto, para o desenvolvimento de sistemas web junto com os componentes citados, é também necessário o uso de linguagens de programação voltadas para web, o que faz com que se reparte em duas partes na sua construção, denominadas de *Front-end* (lado do cliente) e *Back-end* (lado do servidor).

4.3.1 *Front-End* e suas tecnologias

Andrade (2018) define *front-end* como sendo todo o código da aplicação responsável pela apresentação do software. Para Viana (2017) o *front-end* resultará na interação do usuário com o sistema. O *front-end* é o responsável por exibir e coletar as informações da interface gráfica do sistema. Dentro desta parte trabalha-se principalmente com HTML, CSS, JavaScript, entre outras tecnologias para a composição de interfaces gráficas (páginas web).

Segundo Miletto (2014), o HTML é uma linguagem de marcação utilizada para criação das páginas de internet. É através do HTML que é definido o formato que as informações devem se apresentar ao usuário nos navegadores de internet, como Google Chrome, Mozilla Firefox, Internet Explorer. Conforme Eis e Ferreira (2012, p.26), o "HTML é baseado no conceito de Hipertexto, que define um conjunto de elementos ligados por conexões, e esses elementos podem ser textos, vídeos, imagens, gifs e documentos". Todos os elementos que compõem a página são inseridos por meio de comandos específicos da linguagem, nomeados de TAGs. Para Miletto (2014, p.72), "TAG como palavra específica, definida em HTML, envolta por sinais de "menor que" (<) e "maior que" (>). De um modo geral, as TAGs aparecem em pares, uma indicando o início e a outra indicando o fim da marcação", conforme é mostrado na Figura 3:

Figura 3 - Início e fim da tag

Fonte: Miletto (2014)

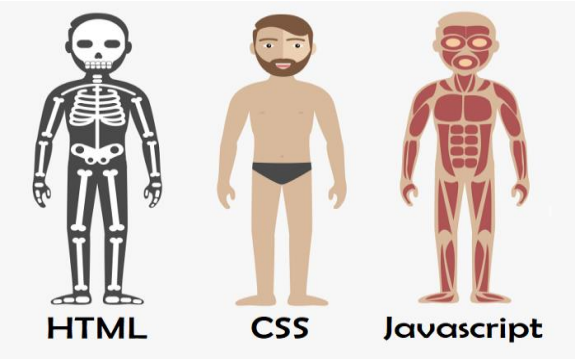
As páginas HTML não têm detalhes, cores ou estilos. Conforme Oliveira (2020, p.35) esta transformação pode acontecer usando “as Folhas de Estilo em Cascata (CSS ou *Cascading Style Sheet*), permitem customizar a formatação dos elementos HTML”. O CSS é o responsável por organizar todos os tipos de informações presentes na web, para que sejam exibidas da melhor maneira aos usuários. A tecnologia CSS tem uma função exclusiva de estilo de página, através da estilização pode ser atribuído alguns efeitos. Por exemplo, é o uso de atributos, como ao passar o mouse em uma estrutura ocorre a mudança da cor das letras.

Por último, temos o JavaScript que é uma linguagem de programação orientada a objetos semelhante às linguagens C++, C e Java. O nome oficial da linguagem, segundo a especificação ECMA-262, é ECMAScript, pois a linguagem foi padronizada e estabilizada pela associação European Computer Manufacturers Association(ECMA), e conta com diversas implementações do padrão (MILETTO, 2014). Por exemplo, movimentos do mouse, pressionar botão, arrastar e soltar etc.

O JavaScript conta com um interpretador que executa operações em tempo real sem a necessidade de enviar dados para o servidor, executando no próprio navegador. Portanto, por meio do JavaScript, podemos realizar a validação de dados e outras operações, esta possibilidade de processar informações no próprio navegador permite que o servidor não se sobrecarregue.

Portando, na Figura 4 a seguir comparando as tecnologias citadas (HTML, CSS e JavaScript) com o corpo humano:

Figura 4 - O corpo humano como uma página web



Fonte: Front-end - Alura (2021)

Para o desenvolvimento do *front-end* (interface do usuário) será utilizado a biblioteca complementar do JavaScript React. Criada em 2013 pela empresa Facebook, esta biblioteca é

voltada para a construção de interfaces web, sendo capaz de manipular elementos visuais de forma que se possa criar componentes reutilizáveis.

O JavaScript, por não ser uma linguagem tipada, será utilizado em conjunto com a biblioteca Typescript. De acordo com o site oficial, “TypeScript é uma linguagem de programação fortemente tipada que se baseia em JavaScript, oferecendo melhores ferramentas em qualquer escala. (MICROSOFT, 2022c)”

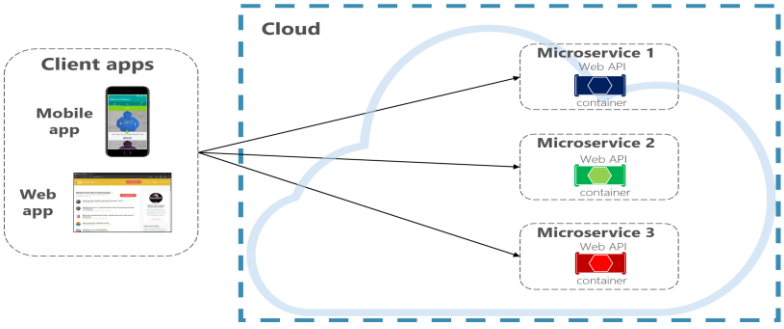
O motivo de utilizar a biblioteca React neste trabalho é a performance dessa tecnologia, pois trabalhar com *Virtual Document Object Model* (DOM), isto é, o React trabalha com uma cópia virtual do DOM da página web, sendo copiado para a memória do computador, e por meio dele é possível fazer a manipulação da árvore de elementos do HTML (REACT, 2021c).

4.3.2 Back-End e suas tecnologias

A camada de *back-end* é uma aplicação de abstração do relacionamento e da comunicação e controle de dados entre a interface de usuário (*front-end*) e o banco de dados. É uma parte que não é visível para os usuários, contendo a lógica de negócio relativo ao projeto solicitado que é considerado uma informação privada e restrita ao negócio (dados pessoais de usuários, informações de funcionários etc.) além de ser de suma importância para o funcionamento correto de software. O código fonte do *back-end* e o sistema de banco de dados geralmente residem e são processados em um servidor.

O *back-end* tem o papel de proporcionar diversos pontos chaves de comunicação com a camada *front-end* de acordo com a necessidade da aplicação de consumir e persistir dados do banco de dados para que o *front-end* possa recuperar (pegar) informações e exibi-las na interface de usuário de maneira apropriada, mostrado na Figura 5.

Figura 5 - Comunicação direta do cliente para o serviço

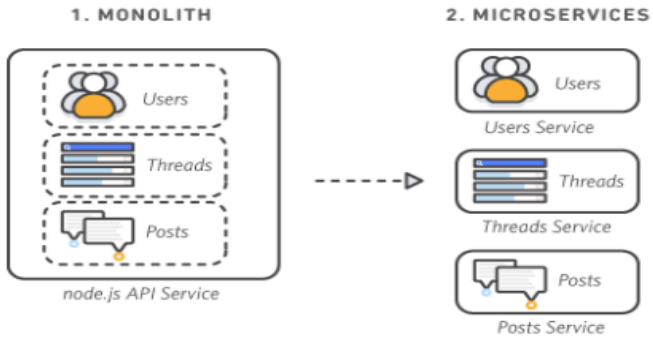


Fonte: Microsoft (2021)

A Figura 6 é uma imagem representativa de um sistema de baixa confiabilidade de uma comunicação entre o *front-end* e microsserviços de *back-end* utilizados para fazer a comunicação entre os mesmos, “microsserviços são uma abordagem arquitetônica e organizacional do desenvolvimento de software na qual o software consiste em pequenos serviços independentes que se comunicam usando APIs bem definidas.” (AWS, 2021).

Podemos definir um sistema entre arquiteturas monolítica e de microsserviços, no monolítico todas as funções do negócio estão implementadas em um único processo e caso uma função tenha uma alta de acesso e necessite que seja escalada, será feita de todo processo por ter suas funcionalidades todas interligadas entre si em um único processo. Já os microsserviços tem suas funcionalidades divididas em processos separados e independentes, possibilitando que seja escalado só a funcionalidade que está tendo uma demanda maior, na Figura 6 abaixo será mostrado a divisão de um aplicativo monolítico em microsserviços.

Figura 6 - Divisão de um aplicativo monolítico em microsserviços



Fonte: Microsserviços - AWS (2021)

Os monólitos e microsserviços tem suas vantagens e desvantagens, as vantagens do um sistema monolítico e que o código tem uma curva de aprendizado melhor pois todas as suas funções estão vinculadas em um único sistema, além de sua manutenção ser feita em único local, porém caso algum problema todo o conjunto do processo será prejudicado e sairá do ar. Já os microsserviços têm sua curva de aprendizagem mais difícil por ter suas funções separadas em processos diferentes além de sua manutenção ser feita em vários locais, porém caso tenha algum problema um dos processos só o mesmo não funcionará, porém os demais continuará funcionando normalmente e quando ele for corrigido tudo funcionará como antes.

Levando em considerações todos os pontos já referidos sobre os tipos de arquiteturas e conceito de um MVP foi definido a utilização de um desenvolvimento em monolito por entregar um projeto íntegro que possa cumprir com todas as necessidades presente no desenvolvimento

deste projeto, sendo possível a implementação de uma arquitetura de micros serviços futuramente sem ter um problema na integridade do mesmo, assim possibilitando entregarmos um projeto funcional em um menor tempo.

No mercado há diversas linguagens de programação e plataformas de desenvolvimento para as tecnologias do *back-end*, já que é uma camada com implementação desacoplada do contexto da aplicação. Elixir é uma linguagem de programação funcional executada na máquina virtual do Erlang. “Erlang é uma linguagem de programação usada para construir sistemas soft em tempo real altamente escaláveis com requisitos de alta disponibilidade.” (WINBLAD, 2021), sendo utilizada em vários setores de TI como banco de dados, comércio eletrônico, setores de telecomunicações, telefonia por computador e mensagens instantâneas.

A escalabilidade é gerada graças ao elixir ser executada dentro de *threads* leves de execução (chamados de processos) trocando informações por meio isolado utilizando mensagens. Possibilitam uma máquina fazer execução de centenas de milhares de processos simultaneamente sem pausas em todo sistema podendo utilizar a de forma mais eficiente o possível graças ao isolamento de seus processos além de trabalhar com a imutabilidade diminuindo possíveis conflitos de informação. A imutabilidade apresenta como conceito algo que não pode ser alterado como objetos e variáveis que precisam estar em um estado imutável após serem criados, porém não garante que os valores serão sempre os mesmos, na Figura 7 abaixo temos um exemplo de um processo.

Figura 7 - Processo Elixir

```
current_process = self()

# Spawn an Elixir process (not an operating system one!)
spawn_link(fn ->
  send(current_process, {:msg, "hello world"})
end)

# Block until the message is received
receive do
  {:msg, contents} -> IO.puts(contents)
end
```

Fonte: Recursos da plataforma - Elixir (2021)

Com a necessidade de fazer uma definição de um banco de dados para fazer a armazenagem, modificações e requisições de informações para que possa ter uma interação completa com o *back-end* e *front-end*. Os sistemas de bancos de dados SQL são tradicionalmente utilizados, por consistir em uma estrutura de dados de forma relacional apresentando os dados em tabelas no modelo linhas e colunas.

O sistema de banco de dados escolhido para o desenvolvimento deste *back-end* utilizado neste trabalho foi o PostgreSQL, ele é um sistema que consegue lidar bem com altos volumes de solicitações e com cargas de trabalho grandes, funcionando bem com sites que tenham uma intensa demanda de acesso, trazendo um desempenho otimizado para um sistema que recebem simultâneos acessos, apresentando uma boa escolha para se trabalhar com a linguagem Elixir que trabalha com milhares de processos simultâneos.

Além de todos esses aspectos o PostgreSQL é um sistema de banco de dados relacional de objeto de código aberto. De acordo com o site https://www.postgresql.org/, o sistema tem mais de 30 anos e possui uma ótima gama de documentos e informações na internet. Atualmente, existem vários profissionais com conhecimento na área, o que facilita a busca de informações para um melhor desempenho dentro das necessidades de cada projeto a ser desenvolvido.

5. METODOLOGIA

Esta pesquisa possui uma natureza aplicada, pois busca gerar o conhecimento para a aplicação prática e dirigida à solução de um problema específico. Tem como objetivo o desenvolvimento de uma aplicação web, que poderá contribuir para causas humanitárias, trazendo um alcance maior de visibilidade para pessoas que visam arrecadar ou realizar doações.

Esta pesquisa busca solucionar um problema da sociedade, por meio de estudos sistemáticos, pesquisas e investigação. Nesse sentido, a classificação do ponto de vista dos objetivos, é de uma pesquisa exploratória, visto que sua finalidade é levantar dados sobre como será arrecadado e distribuída as vestimentas. Utilizar-se-á o método de levantamento de dados e, a partir dessas informações serão realizados estudos significativos a fim de analisar a sua facilidade ou não de adesão ao produto.

Primeiramente será realizada a definição dos papéis para essa escolha, tem-se como base os tipos aos quais os autores as classificam em duas modalidades: Tradicionais - visam na documentação do desenvolvimento de cada etapa do projeto; e ágeis - que pode ser considerada um novo método de desenvolvimento, onde prioridade será nas demandas surgidas dos requisitos, ambiente e reduzir as formalidades.

Seguindo pela metodologia ágil, onde usa de suas habilidades e práticas para satisfazer todos os envolvidos no projeto. Nesse padrão, existe um *framework* muito popular, que é o

Scrum, onde propõe uma abordagem baseada na experiência e observação, aplicando alguns conceitos da teoria da gestão e controle de processos.

De acordo com os cargos dentro do *Scrum*, ficou determinado para cada integrante dentro da equipe um cargo. A seleção foi feita também utilizando apenas 3 funções, sendo eles o *Scrum Master*, *Product Owner* e o *Team Development*. A divisão de funções da forma como pode ser visto na Tabela 1.

Tabela 1 - Stakeholders

CARGO	RESPONSABILIDADE	NOME
Scrum Master	**Responsável pelo projeto**	**Gabriel Henrique**
Product Owner	**Responsável pelos requisitos do projeto**	**Vinicius Reis**
Team Development	**Responsável pelo desenvolvimento do *back-end***	**Júlio César**
Team Development	**Responsável pelo desenvolvimento do *front-end***	**Henrique Valeiro**

Fonte: Autores

Vale ressaltar que indiferente do cargo dos integrantes, todos participarão da escrita do código fonte do software. Finalizando as nomeações, parte-se para a criação e distribuição das tarefas a serem desenvolvidas. Essa atividade será realizada com a utilização do software conhecido como Trello, permitindo a criação de um *board*, *columns* e *cards*, assim sendo, possibilitando uma organização das tarefas de acordo com o estado que ela se encontra. A Figura 8 mostra o quadro no Trello definido inicialmente.

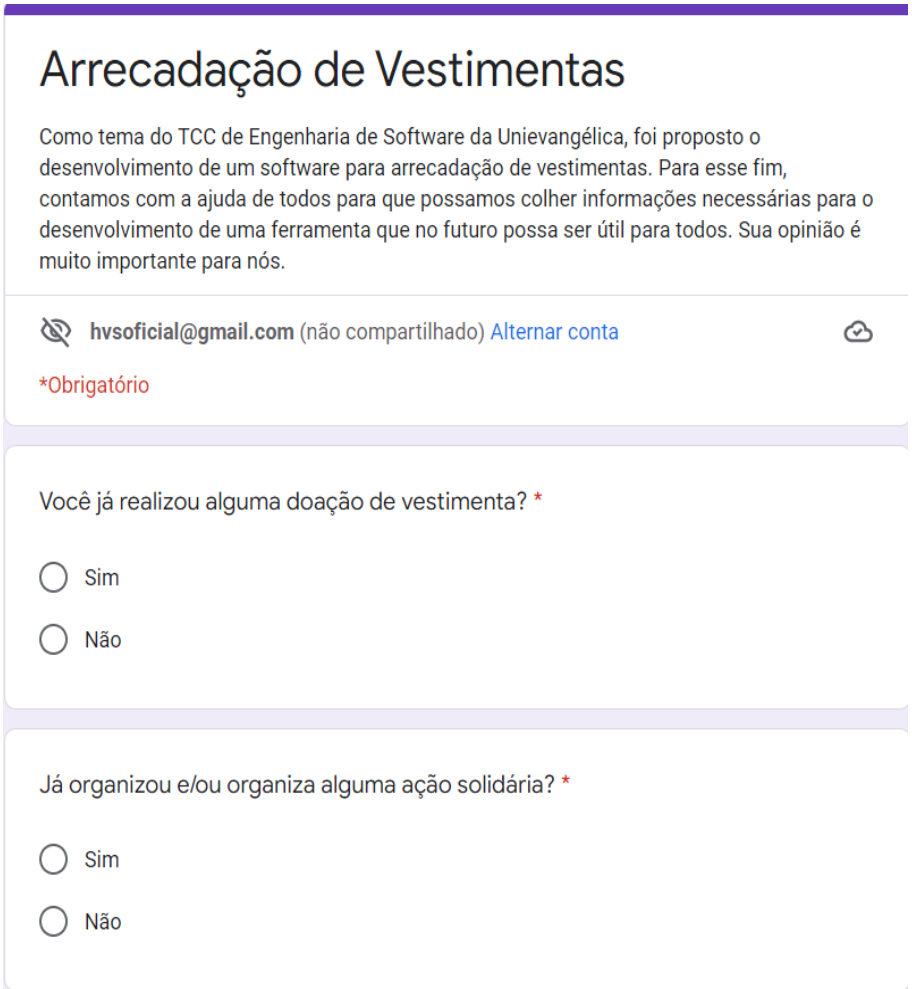
Figura 8 - Quadro do Trello

Fonte: Autores

Em seguida, foi desenvolvido um formulário para levantamento de requisitos para auxiliar na concepção do projeto, foi realizada a concepção de um formulário web para obter uma direção de qual o caminho a tomar para o desenvolvimento do MVP. Foram levantadas seis questões com o objetivo de obter uma visão geral esperada pelos usuários em potencial. Dentre as questões, quatro são objetivas e duas subjetivas, conforme mostram as Figuras 9 e 10.

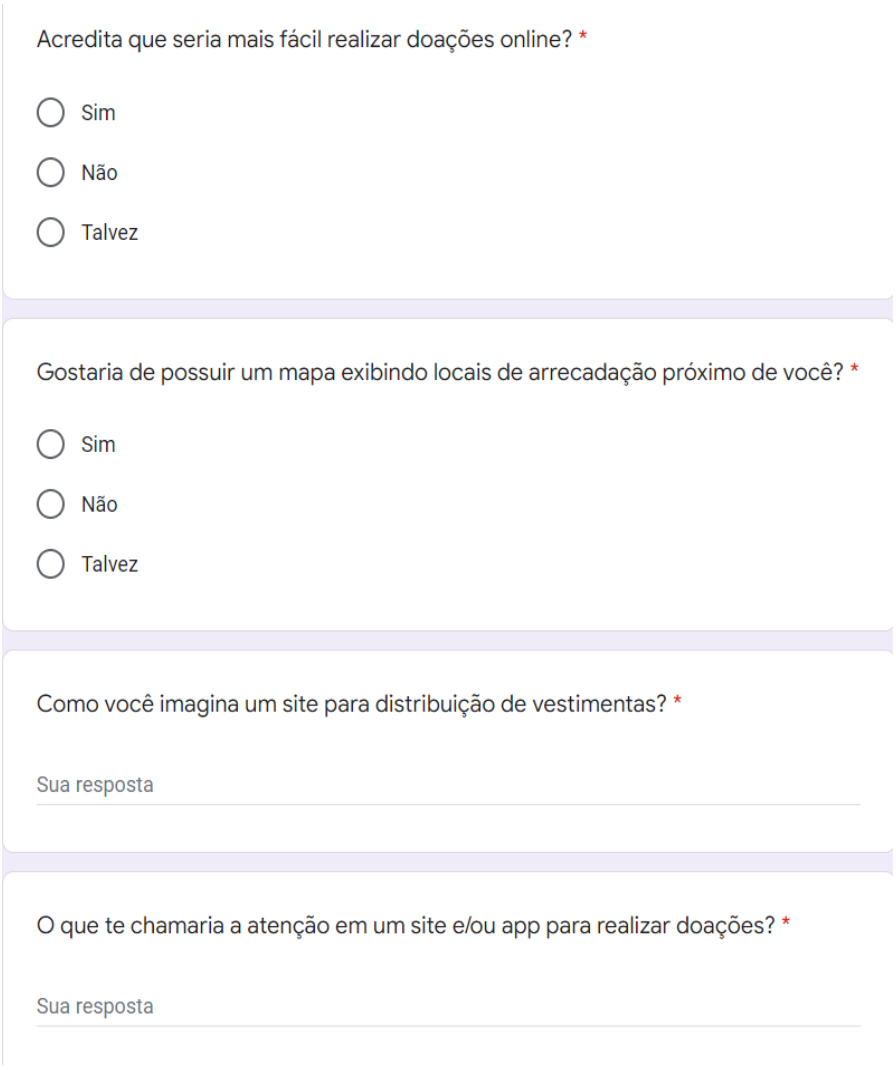
O formulário foi disponibilizado entre os dias 6 e 30 de novembro de 2021, através do endereço URL: https://forms.gle/qP1BWs4G9qGxRLz26. A pesquisa foi realizada com os moradores da cidade de Vianópolis - GO e de Anápolis - GO. Com base neste formulário foram extraídas as histórias de usuários necessárias para a continuidade no projeto.

Figura 9 - Parte 1 formulário para levantamento de requisitos



Fonte: Autores

Figura 10 - Parte 2 formulário para levantamento de requisitos



Fonte: Autores

6. DESENVOLVIMENTO

Neste capítulo apresenta todos os artefatos e as tecnologias utilizadas para o desenvolvimento do MVP.

6.1 Requisitos do Software

Os requisitos de um sistema definem seu funcionamento tanto na parte visual (*front-end*), quanto na parte da retaguarda (*back-end*). Segundo o Nishi (2019) Se o sistema não

atender ao usuário não importa se ele é bem codificado ou bem testado, ele se tornará um esforço ineficaz. Para isso existe a engenharia de requisitos, ela é aplicada nas fases iniciais do projeto, para que o produto final atenda o usuário alvo. Diante disso podemos destacar os seguintes requisitos.

6.1.1 Requisitos Funcionais

RF 01 - Realizar cadastro
RF 02 - Realizar login
RF 03 - Localizar um estabelecimento mais próximo
RF 04 - Mostrar o mapa com os estabelecimentos
RF 05 - Localizar local de doação no sistema
RF 06 - Avaliar o estabelecimento de coleta
RF 07 - Disparar uma mensagem quando o login é mal sucedido
RF 08 - Apagar credenciais de login quando o login for mal sucedido
RF 09 - Mensagem de local de descarte não encontrado
RF 10 - Validar campos não preenchidos corretamente

6.1.2 Requisitos Não Funcionais

- Requisitos não funcionais de desenvolvimento
    RNF 01 - Desenvolvimento do *Front-end* - TypeScript, React
    RNF 02 - Desenvolvimento do *Back-end* - Elixir
    RNF 03 - Banco de dados - PostgreSQL

- Requisitos não funcionais de usabilidade
    RNF 04 - Simplicidade de interface
    RNF 05 - Facilidade de aprendizado
    RNF 06 - Facilidade em usar

- Requisitos não funcionais de manutenibilidade
    RNF 07 - Código simples e boas práticas de desenvolvimento
    RNF 08 - Arquitetura facilitada para manutenção de erros

- Requisitos não funcionais de confiabilidade

RNF 09 - Disponibilidade para uso em qualquer hora e circunstância

RNF 10 - Capaz de continuar funcionando após falha

- Requisitos não funcionais de segurança

RNF 11 - Segurança de dados pessoais dos usuários

RNF 12 - Segurança de dados na localização dos usuários

6.2 Casos de Uso

De acordo com os requisitos levantados, iniciou-se uma nova fase do projeto chamada Casos de Uso, na qual desenvolveu-se um diagrama de caso de uso e as histórias de usuário. Fowler (2011) define o caso de uso sendo uma técnica para documentar com o máximo de detalhes uma funcionalidade do sistema. A documentação descreve quem usará o recurso, quem são os atores, o tráfego que será usado, quais são os pré-requisitos e o resultado.

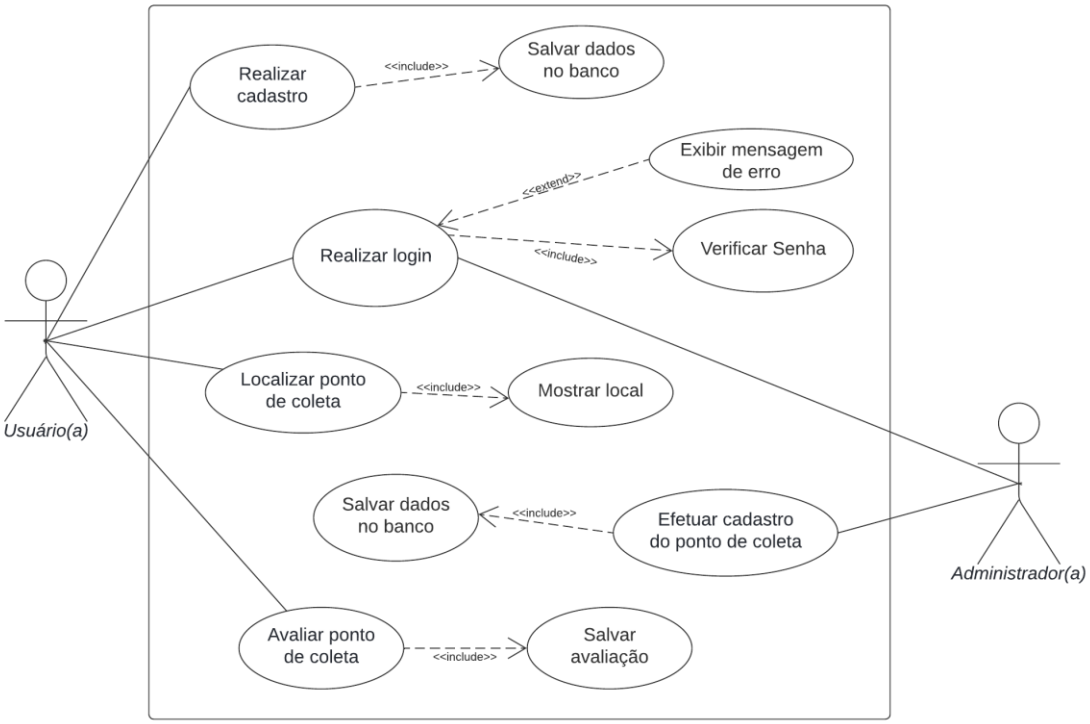
Essa etapa é dividida em duas partes, sendo o diagrama de casos de uso, que é responsável por representar as funções que cada usuário pode utilizar no sistema, e as especificação dos casos de uso (história de usuário), que descrevem cada uma dessas funções. Assim, ao final desta etapa, temos uma prévia de como será o sistema.

6.2.1 Diagrama de Casos de Uso

No diagrama de casos de uso são apresentados os atores e as principais funcionalidades do sistema por meio de ilustração. A Figura 11 apresenta dois atores, sendo eles:

- O ator usuário, onde é quem poderá localizar os pontos de coletas, mas só será necessário realizar o cadastro e o login caso usuário queira avaliar os pontos de coleta.
- O ator administrador é um cadastro que terá acesso na funcionalidade de cadastrar o ponto de coleta e gerenciar um ponto de coleta em específico.

Figura 11 - Diagrama de caso de uso



Fonte: Autores

6.2.2 Especificação dos Casos de Uso (História de Usuário)

Com a pesquisa realizada com moradores da cidade de Vianópolis, foi possível extrair algumas histórias de usuário, para assim se ter uma melhor percepção dos requisitos desejados pelos usuários.

HU – História de Usuário

CA – Critério de Aceitação

Tabela 2 - Realizar Cadastro

HU.001	**Realizar Cadastro**
História de Usuário:	**COMO** usuário **QUERO** me cadastrar **PARA** salvar minhas informações;
CA.001.01	**Campos preenchidos corretamente**
	DADO que estou na tela de cadastro; **QUANDO** clico em salvar; **E** todos os campos estão preenchidos corretamente; **ENTÃO** o cadastro será salvo no banco de dados;
CA.001.02	**Campos não preenchidos corretamente**

	DADO que estou na tela de cadastro; **QUANDO** clico em salvar; **E** caso houver algum campo preenchido incorretamente ou não preenchido; **ENTÃO** de um alerta informando "Campo preenchido de forma incorreta ou não preenchido. Favor verificar novamente"

Fonte: Autores

Tabela 3 - Realizar *login*

HU.002	**Realizar *Login***
História de Usuário:	**COMO** usuário **QUERO** efetuar *login* **PARA** ter a autorização para avaliar os estabelecimentos;
CA.002.01	**Acesso ao sistema**
	DADO que estou na tela do sistema; **E** já exista um cadastro no banco de dados; **QUANDO** informo os dados para autenticação; **E** clico em entrar; **ENTÃO** os dados são validados; **E** visualizo que a autenticação foi realizada com sucesso.
CA.002.02	**Problema na validação dos dados de *login***
	DADO que estou na tela do sistema; **E** já exista um cadastro na banco de dados; **QUANDO** cliquei em entrar; **E** ao validar os dados ocorre um erro; **ENTÃO** as informações de *login* são apagados; **E** uma mensagem é exibida "Dados inválidos, tente novamente".

Fonte: Autores

Tabela 4 - Localizar Estabelecimento

HU.003	**Localizar estabelecimento**
História de Usuário:	**COMO** usuário; **QUERO** localizar os estabelecimentos cadastrados; **PARA** efetuar a doação das vestimentas.
CA.003.01	**Encontrar estabelecimento**
	DADO que estou na tela de localização dos estabelecimentos; **QUANDO** procuro por um estabelecimento; **ENTÃO** aparece o mapa com os locais dos estabelecimentos marcados;
CA.003.02	**Local de ponto de coleta não localizado**

	DADO que estou na tela de localização dos estabelecimentos; **QUANDO** procuro um estabelecimento; **E** nenhum foi possível de localizar; **ENTÃO** uma mensagem é exibida “Desculpe, nenhum estabelecimento foi encontrado nas proximidades. Por favor tente novamente mais tarde”.

Fonte: Autores

Tabela 5 - Avaliar Estabelecimento

HU.004	**Avaliar estabelecimento**
História de Usuário:	**COMO** usuário **QUERO** avaliar o estabelecimento **PARA** deixar a minha opinião sobre o local.
CA.004.01	**Deixar opinião**
	DADO que estou na tela de avaliação; **E** estou logado com minha conta de usuário; **QUANDO** clico em avaliar; **ENTÃO** a minha avaliação e salva;

Fonte: Autores

Tabela 6 - Efetuar Cadastro do Estabelecimento

HU.005	**Efetuar cadastro do estabelecimento**
História de Usuário:	**COMO** administrador **QUERO** cadastrar um estabelecimento **PARA** exibir aos usuários os locais de doação;
CA.005.01	**Campos preenchidos corretamente**
	DADO que estou na tela de cadastrar o estabelecimento; **E** logado como administrador; **QUANDO** clico em salvar; **E** todos os campos estão preenchidos corretamente; **ENTÃO** o estabelecimento será salvo na banco de dados;
CA.005.02	**Campos não preenchidos corretamente**
	DADO que estou na tela de cadastrar o estabelecimento; **E** logado como administrador **QUANDO** clico em salvar; **E** caso algum campo não for preenchido corretamente; **ENTÃO** uma mensagem é exibida “Campo preenchido de forma incorreta ou não preenchido. Favor verificar novamente”

Fonte: Autores

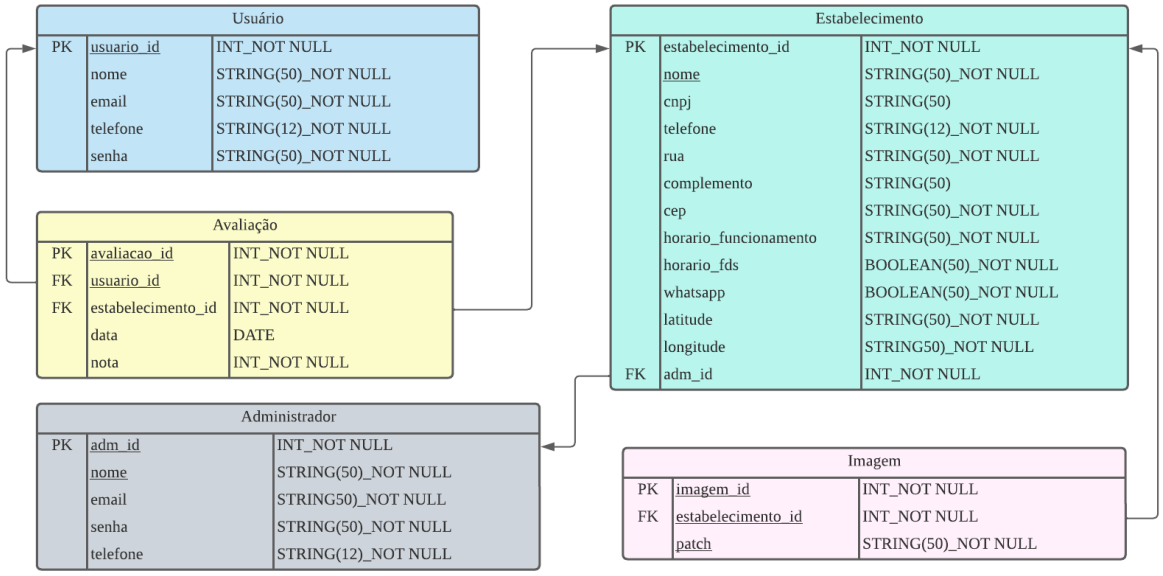
6.3 Banco de Dados

O banco de dados é uma parte essencial do desenvolvimento de projetos de software, assim os dados informados pelos usuários e administradores são armazenados. Com o levantamento de requisitos já realizado e feito os casos de uso, deu-se início a modelagem, sendo um banco de dados relacional clássico.

6.4.1 Modelagem Lógica

Pode ser observado na Figura 12 o modelo lógico do projeto, onde além da tabela e relacionamentos, contém propriedades e seus tipos de dados definidos. Com isso o banco de dados é demonstrado com mais detalhes.

Figura 12 - Modelo lógico do banco de dados



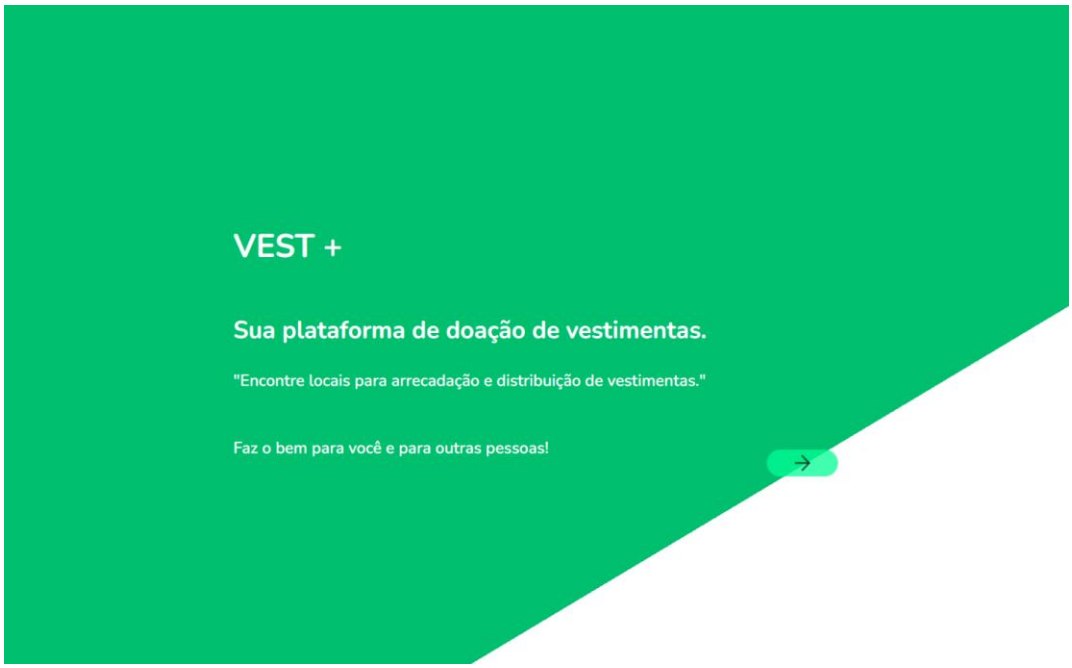
Fonte: Autores

6.4 Protótipos

Protótipos são modelos estáticos ou dinâmicos que servem para demonstrar como será o produto final. Fazer a prototipagem de maneira correta, diminui as dúvidas acerca da aparência e requisitos, assim podendo evitar prejuízos desnecessários. Para o projeto foi

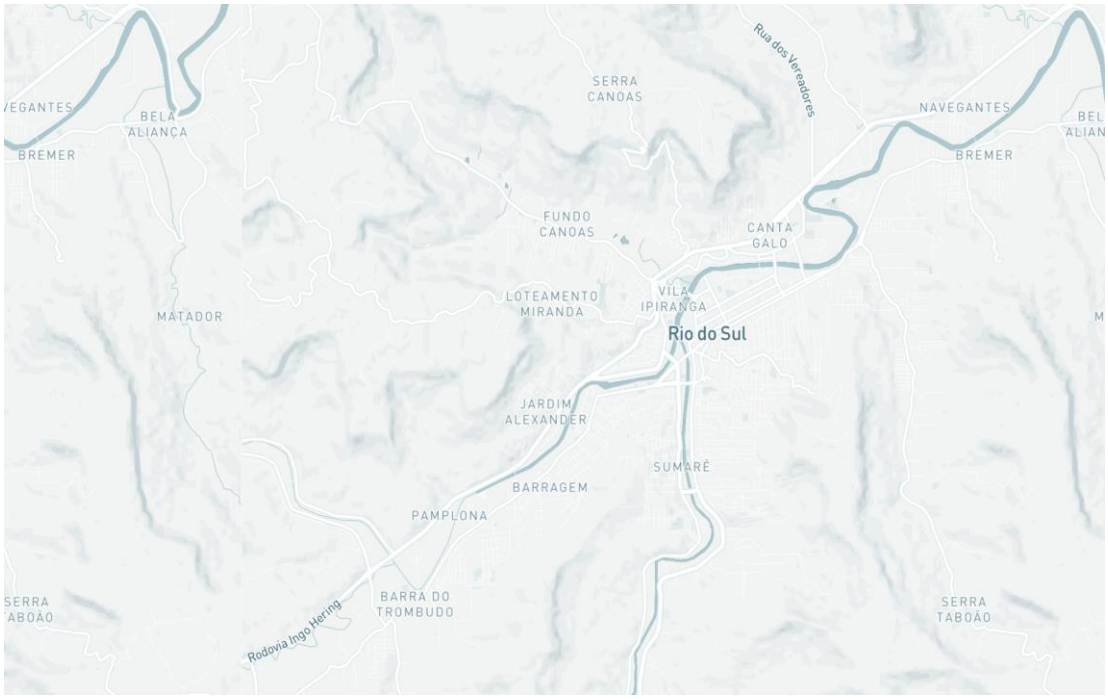
desenvolvido a prototipagem das telas, um esboço de como seria o projeto final. Podendo ser observado nas Figuras 13 a 15 abaixo.

Figura 13 - Tela inicial



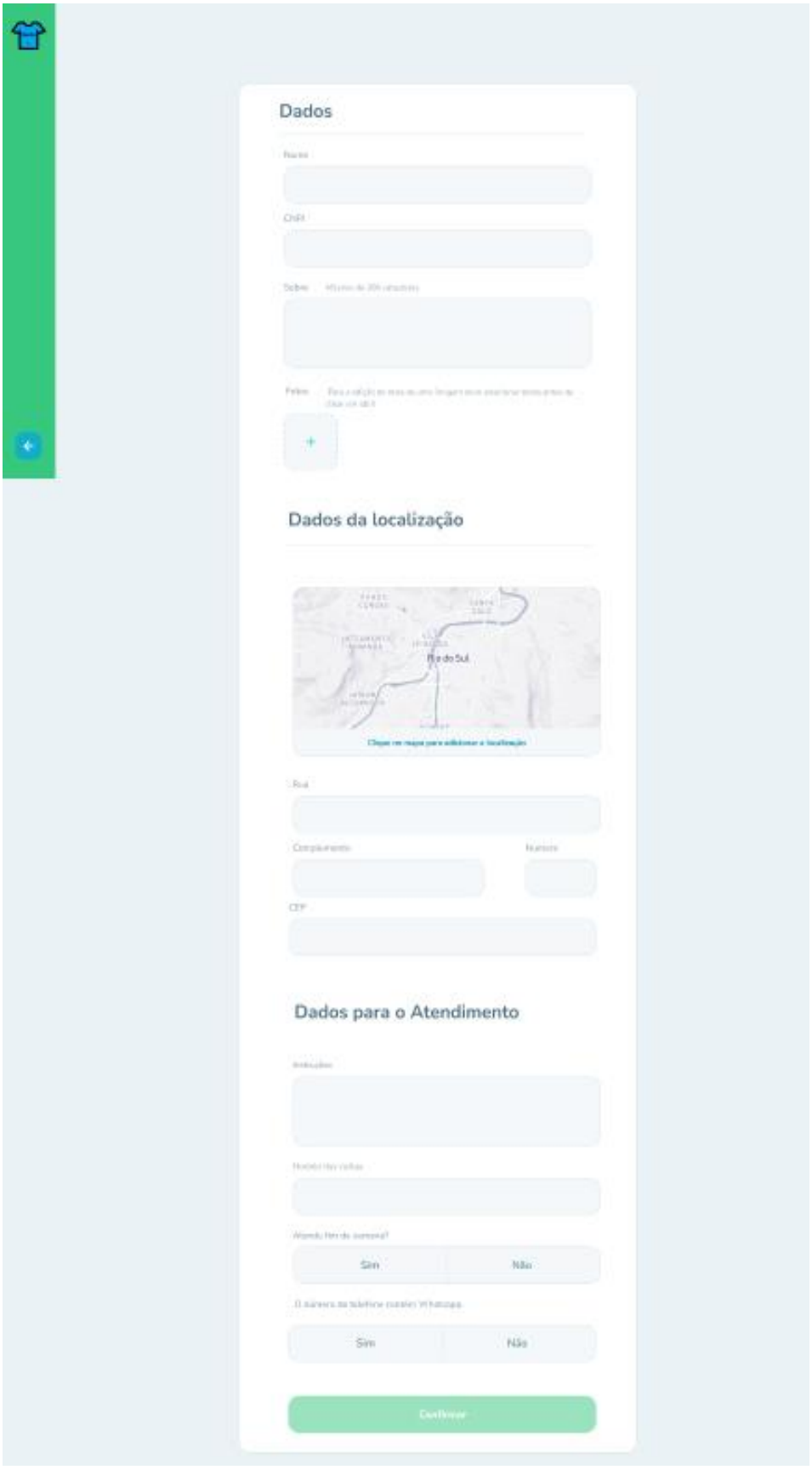
Fonte: Autores

Figura 14 - Mapa



Fonte: Autores

Figura 15 - Cadastro



Fonte: Autores

6.5 Back-End

Em paralelo com a criação do protótipo e tendo a modelagem do banco pronto, foi-se iniciado o desenvolvimento do código em *Elixir* e seu *framework Phoenix* para servir os dados da aplicação. Partindo do pressuposto que já possuir instalado a linguagem e seu *framework*, e seguindo como base o início do guia Phoenix, para iniciar o projeto, basta rodar o comando presente na Figura 16.

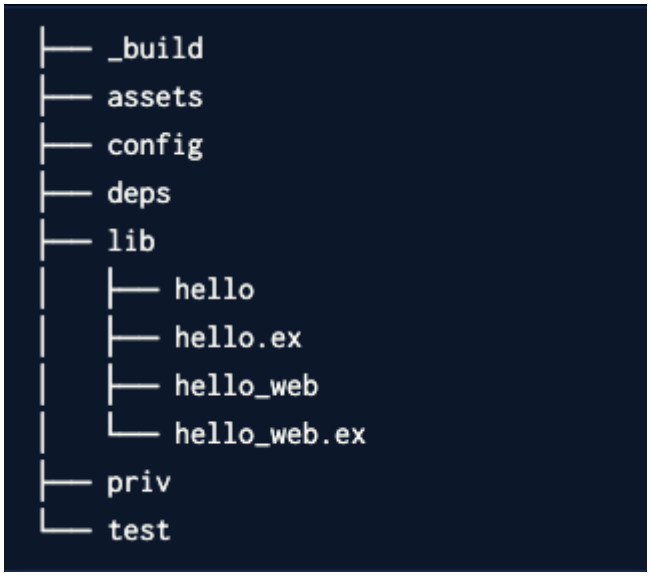
Figura 16 - Comando de início do projeto

```
mix phx.new vestemais --no-assets  --no-html --no-live
```

Fonte: Autores

Executado o comando e de acordo com a documentação no *Hexdoc*, o *Phoenix* irá gerar toda a estrutura inicial necessária para rodar uma aplicação, tendo toda a pré-configuração do banco de dados, *entrypoint*, preparação para a geração de *migrations* entre outros. A estrutura de pastas geradas pode ser vista na Figura 17, sendo as pastas *lib*, *config*, *priv* e *test* as mais utilizadas durante o desenvolvimento pelo desenvolvedor (PHOENIX, 2022c).

Figura 17 - Estrutura de pastas

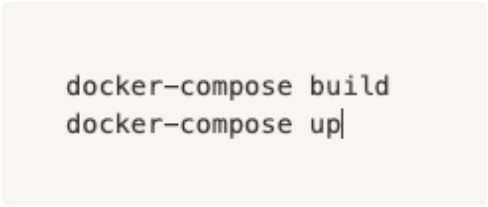


Fonte: Autores

Com o projeto criado, foi necessária a configuração dos arquivos *Dockerfile* e *docker-compose*. De acordo com sua documentação oficial, um *Dockerfile* é um documento de texto que contém todos os comandos que um usuário pode chamar na linha de comando para montar uma imagem (DOCKER, 2021c). Ambos os arquivos podem ser encontrados nos respectivos Apêndice A e Apêndice B ou no repositório *GitHub*, disponível no endereço URL: https://github.com/gabshs/vestemais.

Após a criação dos arquivos é necessário rodar o comando descrito na Figura 16 para realizar o processo de *build* do projeto e assim que finalizar com sucesso, para inicializar a aplicação, basta rodar o segundo comando da mesma figura tendo assim cumprido os requisitos básicos para levantar a aplicação utilizando contêineres.

Figura 18 - Criação dos arquivos necessários



Fonte: Autores

Concluídas as etapas anteriores, tem-se por objetivo a instalação de dependências que são cruciais para o projeto. Para o MVP, foram selecionadas as seguintes dependências:

- Credo: ferramenta de análise estática de código para a linguagem Elixir com foco no ensino e na consistência do código.
- Excoveralls: uma biblioteca que relata estatísticas de cobertura de teste, com a opção de postar no serviço coveralls.io.
- ExMachina: facilita a criação de dados e associações de teste. Funciona muito bem com o Ecto, mas é configurável para funcionar com qualquer biblioteca de persistência.

A inserção de uma dependência é feita através do arquivo mix.exs que fica na raiz do projeto. Para que possa instalar elas de fato no projeto, é necessário rodar o comando da Figura 19 que, por sua vez, irá baixar as dependências do repositório oficial provido pelo Elixir. A utilização do comando docker-compose run –rm app se fará necessário antes do comando desejado, para que possa ser refletido diretamente dentro do contêiner.

Figura 19 - Comando para inserção de dependência

```
docker-compose run --rm app mix deps.get
```

Fonte: Autores

Dando continuidade ao projeto, foi realizada a configuração da base de dados. Por padrão, o Phoenix utiliza o Postgresql como banco de dados. O Elixir por sua vez possui o Ecto, que é descrito pelo ElixirSchool (2021c) como um projeto oficial da Elixir que fornece um *wrapper* de banco de dados e uma linguagem de consulta integrada.

Dentro da pasta config, existem 3 arquivos que configuram 3 ambientes distintos, sendo eles dev.exs (ambiente de desenvolvimento), test.exs (ambiente de teste) e prod.exs (ambiente de produção). Os ambientes utilizados no desenvolvimento, como o nome sugere, é o dev e o test que é utilizado para rodar testes automatizados.

Ambos os ambientes possuem macros de configuração como descritos na Figura 20 e se faz necessário para atribuir as variáveis de conexão com a base de dados. Conforme configurado no docker-compose, foi utilizado o termo db como *host* e as *environments* *POSTGRES_USER* e *POSTGRES_PASSWORD* para a autenticação no banco.

Figura 20 - Variáveis de conexão de banco

```
config :vestemais, Vestemais.Repo,
 username: "postgres",
 password: "postgres",
 hostname: "db",
 database: "vestemais_dev",
 show_sensitive_data_on_connection_error: true,
 pool_size: 10
```

Fonte: Autores

Assim que configurado o acesso a base de dados, tendo como referência a modelagem lógica, foi criado uma *migration* para representar a entidade estabelecimento (Por padrão, todo

o código foi realizado em inglês, sendo a entidade nomeada como *Establishment*). As Figuras 21 e 22 representam respectivamente o script para a geração da *migration* e sua estrutura.

Figura 21 - *Migration* entidade estabelecimento

```
mix ecto.gen.migration create_establishment_table
```

Fonte: Autores

Figura 22 - Modelo de criação do banco de dados

```
defmodule Vestemais.Repo.Migrations.CreateEstablishmentTable do
 use Ecto.Migration


 def change do
   create table(:establishments) do
     add :name, :string
     add :cnpj, :string
     add :phone, :string
     add :lat, :string
     add :long, :string


     timestamps()
   end
 end
end
```

Fonte: Autores

O padrão do arquivo gerado pela *migration* é a *timestamp* do horário que o arquivo foi gerado e o nome passado como argumento no script separados por *underscore*(_). O Ecto possui um macro chamado *Migration* que permite a criação da tabela e seus campos de acordo com o tipo inserido. A utilização das *migrations* é ideal caso precise alterar o driver do banco de dados,

sendo necessário apenas a alteração do driver utilizado no arquivo de configuração como descrito acima neste documento. Por fim basta rodar o comando apresentado na Figura 23 para que seja refletido no banco de dados.

Figura 23 - Código para criar a tabela no banco de dados

```
mix ecto.migrate
```

Fonte: Autores

Com a *migration* já inclusa no banco de dados, foi desenvolvido em seguinte a parte de modelos. De acordo com (BALTHAZAR et al., 2006), é no *Model* que se tem, a comunicação com a camada responsável pelo acesso ao banco, ou com outras que processem os dados. No caso do Elixir, a lógica da camada de negócios fica dentro da pasta *Lib* e por convenção usamos a nomenclatura do objeto referenciado no plural. No caso deste artigo, foi criada uma pasta com nome *Establishments* fazendo referência às ações do estabelecimento.

O primeiro arquivo gerado será o *schema* do estabelecimento, ao qual é referenciado dentro da função macro com mesmo nome, foi inserido os campos que fazem referência às colunas na base de dados. Conforme pode ser visto na Figura 24, além dos campos criados anteriormente, foi adicionado à função *timestamps()* a qual tem por função gerenciar colunas auto geradas pelas *migrations* que referenciam o horário de criação e atualização de um registro.

Figura 24 - Colunas do banco de dados

```
schema "establishments" do

  field :name, :string

  field :cnpj, :string

  field :phone, :string

  field :lat, :string

  field :long, :string


  timestamps()

end
```

Fonte: Autores

Dentro deste mesmo arquivo também possui uma função cujo nome é *changeset*. Essa função é importada de dentro do Ecto e deve ser sobrescrita para realizar validações e também o *cast* dos dados serializados em *JSON* para a estrutura de módulo, que é referenciada. O arquivo completo pode ser encontrado no Apêndice C.

A camada que intermediará entre o cliente e a lógica da aplicação é chamada de *controller*. Ele terá como função receber uma requisição e encaminhará para a classe responsável por executar determinada ação. Para este projeto foi definido como partes importantes a criação e visualização de estabelecimentos, tendo então o *controller* com a estrutura apresentada no Apêndice D.

Dentro do *controller*, quando chamado pela *route* que será descrita em breve, a função possui como parâmetros de entrada a *connection*, possuindo todas as informações da conexão, sendo importante também para a inserção de status e também retornar uma informação quando executado e como segundo parâmetro, dados enviados no corpo da requisição.

Foi desenvolvido também uma classe *fallback* que tem por finalidade retornar uma mensagem de erro caso ocorra algum problema com a solicitação. Desta forma, é possível definir a tratativa de erros genérica, atribuindo então a camada de lógica definir qual o erro foi gerado durante a requisição conforme a Figura 25.

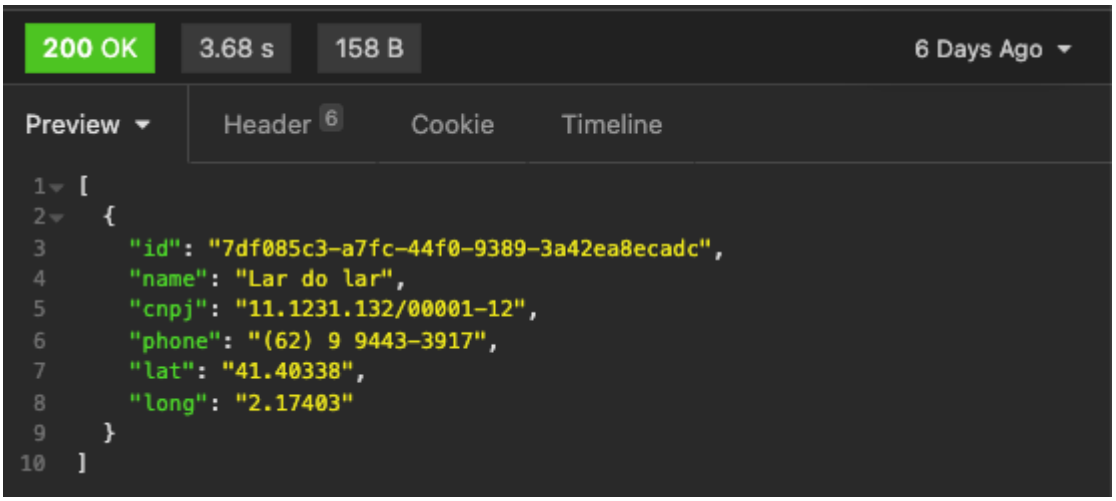
Figura 25 - *Fallback_Controller.ex*

```
defmodule VestemaisWeb.FallbackController do
 use VestemaisWeb, :controller


 alias Vestemais.Error
 alias VestemaisWeb.ErrorView


 def call(conn, {:error, %Error{status: status, result: result}}) do
   conn
   |> put_status(status)
   |> put_view(ErrorView)
   |> render("error.json", result: result)
 end
end
```

Fonte: Autores

Caso a requisição ocorra com sucesso, o controller atribui então um código de sucesso de acordo com o tipo da requisição e renderizar um arquivo, que por se tratar de uma api, retorna um JSON ao qual cada função retorna um json específico, tendo como exemplo a listagem de estabelecimentos presente na Figura 26.

Figura 26 - Requisição listando todos os estabelecimentos cadastrados



Fonte: Autores

A última camada da aplicação, sendo a camada de entrada, é o *router*. Ele tem como objetivo definir as rotas da aplicação, além de apontar para qual *controller* receberá a conexão de acordo com a rota determinada. As rotas seguem o padrão REST, possuindo os verbos, *GET*, *POST*, *PUT* e *DELETE*. A implementação no Elixir pode ser facilitada utilizando o método *resource*, passando como argumentos a rota e o controller conforme a Figura 27.

Figura 27 - Referenciar a imagem router.ex

```
        scope "/api", VestemaisWeb do

          pipe_through :api



            resources "/establishments", EstablishmentsController, except: [:new,
:edit]

        end
```

Fonte: Autores

Para assegurar o funcionamento do código, foram desenvolvidos testes unitários utilizando a biblioteca *Excoveralls* juntamente com a biblioteca de testes já integrada no Elixir, o *ExUnit*. Os arquivos ficam dentro da pasta *test* localizada na raiz da aplicação, possuindo dentro dela, estrutura semelhante à da pasta *lib*, com a inclusão do arquivo *test_helper.exs* e a pasta *support*.

Foram criados testes para cada método na camada de lógica e também testes para o *controller*, simulando seu funcionamento e definindo valores esperados de acordo com os parâmetros inseridos. Com a biblioteca *ExMachina*, foi possível criar o que são chamadas de *factories*, que são responsáveis por retornar estruturas de dados pré-moldadas para serem utilizadas.

Com o *Excoveralls*, é possível verificar a cobertura do código desenvolvido, sendo que o projeto atual atingiu 96.8% de cobertura. Para verificar esse dado, foi necessário rodar o comando "*docker-compose run –rm app mix test –cover*" ao qual retornou no terminal o relatório que pode ser visto na Figura 28.

Figura 28 - Relatório de cobertura de testes

```
warning: unused import Ecto.Repo
  test/vestemais/establishments/get_test.exs:4

..............

Finished in 1.7 seconds (1.7s async, 0.00s sync)
14 tests, 0 failures

Randomized with seed 911480
----------------
COV    FILE                                        LINES RELEVANT   MISSED
100.0% lib/vestemais.ex                              8        3        0
100.0% lib/vestemais/error.ex                       12        2        0
100.0% lib/vestemais/establishments/create.ex       17        3        0
100.0% lib/vestemais/establishments/establishme     25        2        0
100.0% lib/vestemais/establishments/get.ex          17        5        0
100.0% lib/vestemais_web/controllers/establishm     32        6        0
100.0% lib/vestemais_web/controllers/fallback_c     13        1        0
 83.3% lib/vestemais_web/views/error_view.ex        35        6        1
100.0% lib/vestemais_web/views/establishments_v     20        3        0
[TOTAL]  96.8%
----------------
```

Fonte: Autores

6.6 Front-End

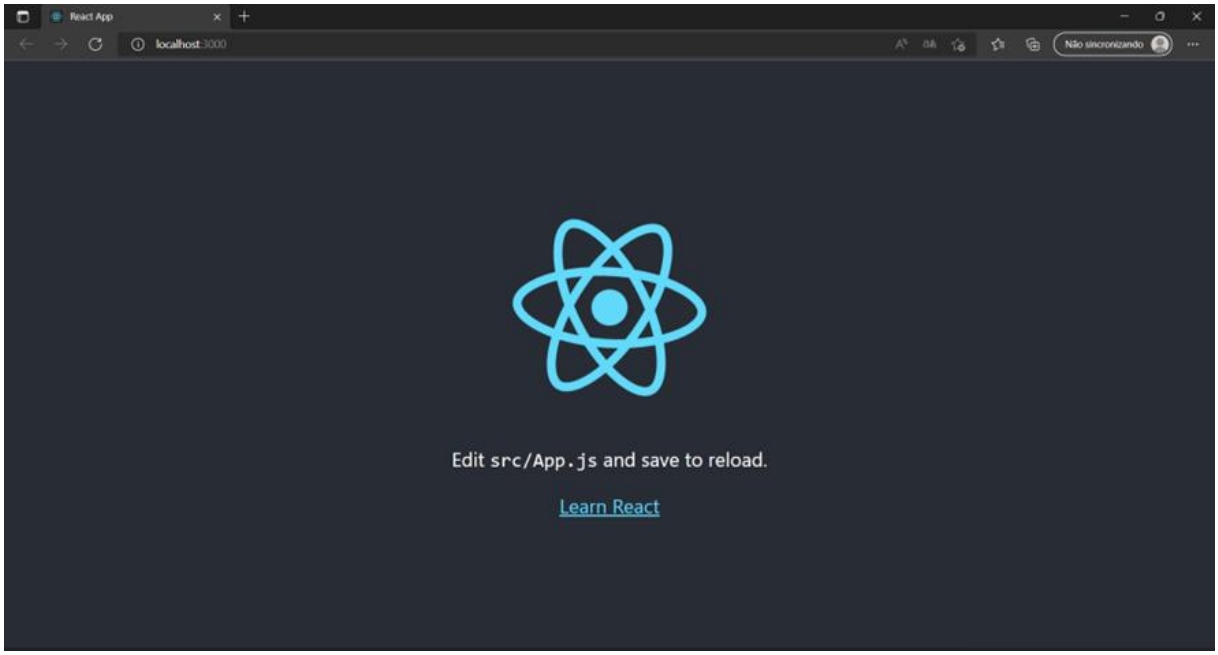
Em paralelo com a criação do protótipo e do desenvolvimento do *back-end*, foi iniciado o desenvolvimento do *front-end* em *React*. Partindo do pressuposto de já possuir instalado o Node e o NPM, para assim dar início ao projeto, será instalado a CLI (*command-line interface*) utilizando o código no terminal, de acordo com a Figura 29.

Figura 29 - Comando instalar a CLI

Fonte: Autores

Após a instalação da CLI, a base do projeto será criada utilizando o comando "*create-react-app web_vest*". A pasta principal contém a estrutura padrão de introdução do *React*. Rodando o comando "*npm start*" após abrir o projeto em um editor de código, será iniciado um servidor local, redirecionando para a página inicial como podemos ver na Figura 30.

Figura 30 - Página inicial



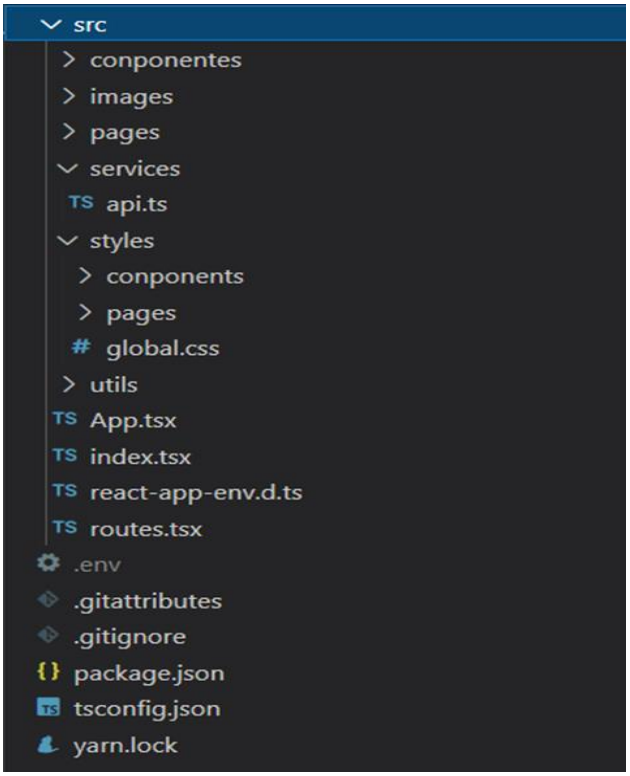
Fonte: Autores

Após a criação da estrutura base do projeto seguindo as instruções do site oficial do *React*, foi adicionado como dependência o Axios para realizar o consumo de API e o *react-leaflet* para auxiliar na utilização de mapas fornecidos pelo *Leaflet*. Através dos seguintes comandos as dependências foram instaladas com sucesso:

- *npm install axios*
- *npm install react-leaflet*

Após a instalação das dependências citadas acima, o documento do *front-end* foi repartido de acordo com a Figura 31.

Figura 31 - Estrutura de pastas



Fonte: Autores

Na pasta src estão todos os componentes necessários para o desenvolvimento deste projeto, sendo eles a pasta componentes onde estão os *scripts* que são utilizados em mais de uma página, assim facilitando na compreensão e em correções e manutenções futuras. A pasta de imagens está destinada às imagens e figuras utilizadas no projeto. *Pages* contém os *scripts*

das páginas que serão apresentadas ao usuário. O componente services contém o api.ts, esse *script* é responsável por fazer conexão com API do *back-end* utilizando o axios, esse *script* está disponível na Figura 32.

Figura 32 - *Scrypt* do api.ts

```
import axios from 'axios';

const api = axios.create({

   // O baseURL: recebe o endereço para conectar com a API do back-end

   baseURL: process.env.REACT_APP_API_URL,

   // E utilizado process.env.REACT_APP_API_URL por necessitar de uma variável para subir o código no Heroku
   // Caso o arquivo não necessite de ser subido para o ambiente de produção como o Heroku pode ser utilizado só
   // baseURL:http://localhost:3338, em vez de baseURL: process.env.REACT_APP_API_URL,
   // A variável process.env.REACT_APP_API_URL recebe informação da mesma que está localizado no arquivo .env
   // O arquivo .env na variável process.env.REACT_APP_API_URL recebe http://localhost:3338 que e o endereço para
   //conectar com a API do back-end


})

export default api;
```

Fonte: Autores

A pasta *styles* contém todo o css de estilização necessário para as páginas e componentes. Em utils está localizado os *scripts* do ícone da localização, como suas dimensões. O arquivo App.tsx é responsável por repassar as rotas de quais páginas necessitam da renderização das rotas do projeto. Já o index.tsx é o *script* principal responsável pela renderização das páginas que são repassadas pelo App.tsx. O react-app-env.d.ts contém as referências dos tipos de *script*. Por último o routes.tsx que é o responsável pela designação de quais são as rotas existentes e quais componentes serão repassados de acordo com a rota solicitada.

Para a apresentação e consumo da API do mapa precisamos importar alguns componentes do *react-leaflet*, como o *Map, TitleLayer, Marker e Popup*. Com o *Map* podemos fornecer informações da renderização do mapa, já o *TitleLayer* tem como parâmetro receber a URL da API que vai ser utilizado na para renderizar o mapa e por último o *Marker* e *Popup* como perceptível pelo nome dois, suas funções respectivamente são marcar um ponto no mapa e gerar um *Popup*. Abaixo a Figura 33 está representada a importação dos componentes do *react-leaflet*.

Figura 33 - Importação dos componentes do *react-leaflet*

```
import { Map, TileLayer, Marker, Popup } from 'react-leaflet'
```

Fonte: Autores

Na Figura 34 abaixo será apresentado o código utilizado no projeto para demonstrar o mapa com os pontos de coletas, tendo explicações em forma de comentário nas classes principais para o desenvolvimento desta funcionalidade.

Figura 34 - Código para a apresentação dos estabelecimentos

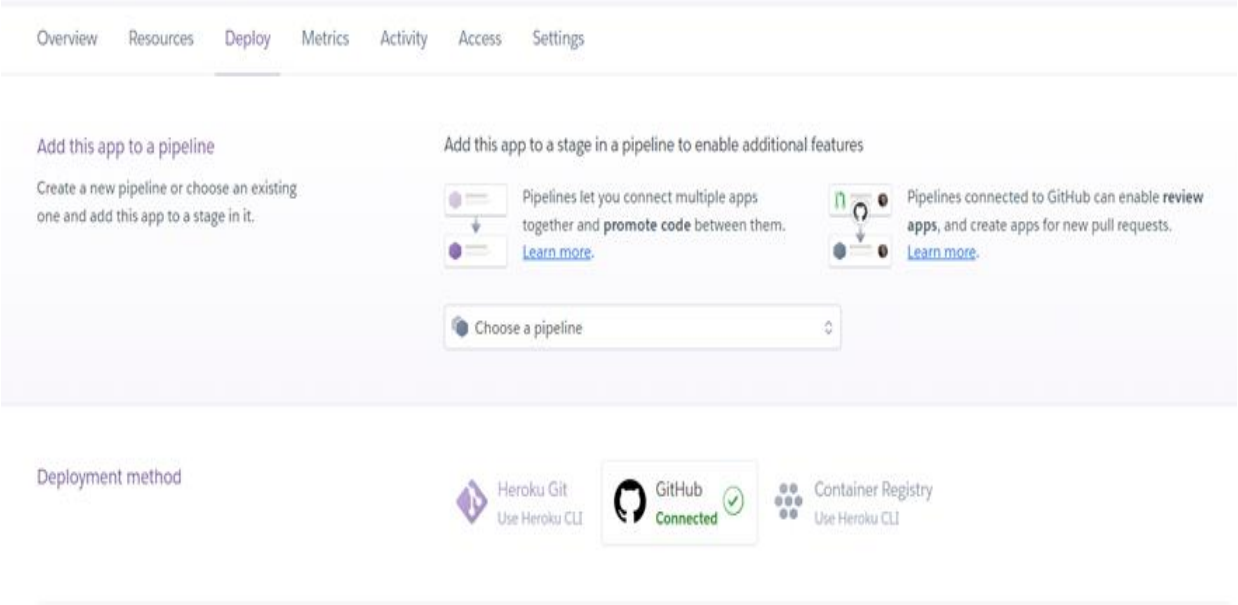
```
<Map center={[-16.3325529, -48.9549526]} zoom={14} style={{ height: '100%', width: '100%' }}>
{/* A classe Map necessita da propriedade center que receber como parâmetro dois números representados por x e y
o x recebe a latitude e o y a longitude que são utilizados para definir o centro do mapa ao qual será renderizado
a zoom define o foco e style recebe a estilo que o mapa aparece */}
    <TileLayer url={`https://api.mapbox.com/styles/v1/mapbox/light-v10/tiles/256/{z}/{x}/{y}@2x?access_token=
    ${process.env.REACT_APP_MAPBOX_TOKEN}`} />
    {/* O TileLayer recebe a propriedade url que cotem a url da api que fornece o mapa
    nesta url e contido o access_token que e o validador de permissão de acesso, para poder utilizar os dados do mapa,
    sem um access_token valido não e possível renderizar o mapa */}
    {estabelecimentos.map(estabelecimento => (
        <Marker key={estabelecimento.id} position={[estabelecimento.latitude, estabelecimento.longitude]} icon={mapIcon} >
            {/* O Marker tem a função de marcar postos no mapa necessitando da Key que é o identificador do ponto,
            o position que recebe a posição do ponto a ser marcado necessitando dos mesmos parâmetro que o certer do Map
            e o icon que recebe as informações do ícone que o ponto recebera */}
            <Popup closeButton={false} minWidth={240} maxWidth={240} className="map-popup">
                {/* O Popup gera um popup após ser clicado em um Marker */}
                {estabelecimento.name}
                <Link to={`/estabelecimentos/${estabelecimento.id}`}>
                    <FiActivity size={20} color="#fff" />
                </Link>
            </Popup>
        </Marker>
```

Fonte: Autores

Para colocarmos o projeto em um ambiente de produção ao qual é possível ser acessado por qualquer computador que tenha acesso a internet, e se torne viável para que um usuário possa acessá-lo. É necessário subir o projeto para um repositório online ao qual foi utilizado o *GitHub*.

Após subir o arquivo para o repositório e estar logado em uma conta da *Heroku*, é necessário a criação de um novo app. Na aba *Deploy* você terá três opções de para subir o arquivo como pode ser visto na Figura 35 abaixo.

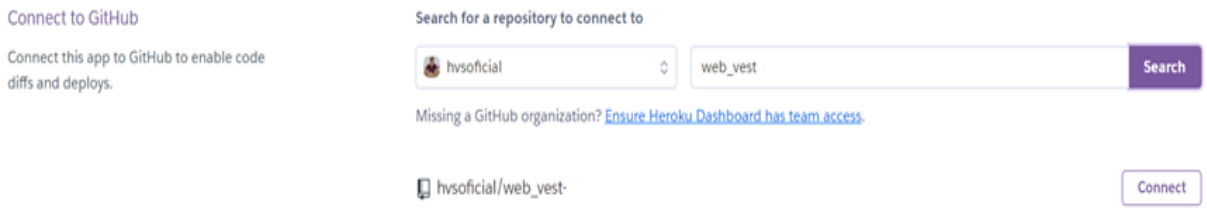
Figura 35 - Aba *Deploy* no *Heroku*



Fonte: Autores

A primeira opção é subir o arquivo utilizando a CLI do *Heroku*, a segunda é pelo *GitHub* e por último utilizando container. Como já temos o arquivo no *GitHub* é só necessário vincular-se com a conta com o *Heroku*.

Figura 36 - Conexão com o GitHub



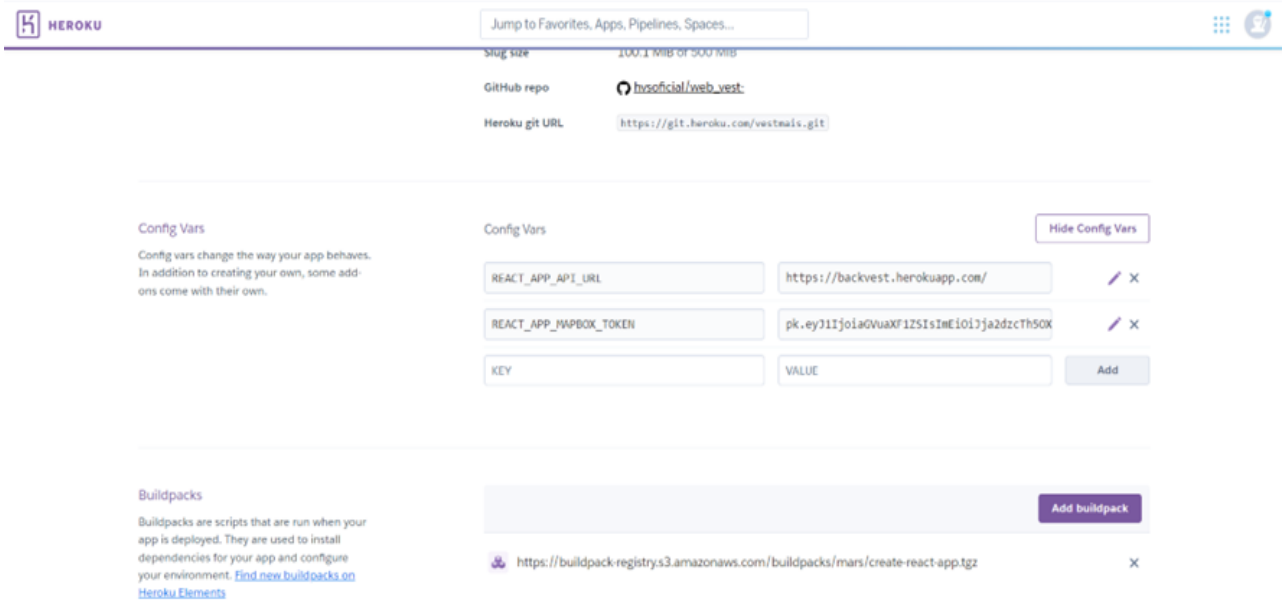
Fonte: Autores

Na Figura 36 é mostrado como é selecionado o repositório para que possa ser usado no *Heroku*, após conectar com o repositório será possível fazer dois tipos de *deploy* que é o manual e o automático, no manual toda a vez que ser feita uma alteração no repositório é necessário que manualmente seja solicitado um *deploy* para que as alterações seja efetuadas, já no automático quando e feito uma alteração no arquivo passado pela a ramificação principal

(*main*) o *heroku* efetua um *deploy* automático para que seja implementada as alterações feitas no projeto de maneira eficiente.

Antes de ser feito o *deploy* são necessárias algumas configurações para que não ocorra erro no *heroku* ao tentar rodar o código. Em *Settings* temos que configurarmos as variáveis que nossa API irá necessitar para que tenha um funcionamento correto.

Figura 37 - Aba *Settings* no *Heroku*



Fonte: Autores

Como demonstrado na Figura 37, o projeto necessita da configuração de duas variáveis e um *buildpack* (Pacotes de compilação). A variável REACT_APP_API_URL recebe o endereço para o consumo da API do *back-end* que pode ser vista na Figura 32, já o REACT_APP_MAPBOX_TOKEN recebe o *token* de acesso para que possamos utilizar o mapa como pode ser visto na Figura 34.

O *buildpack* e necessário para que possa ser compilado o projeto em *react*, para isso e necessário adicionar o pacote "https://buildpack-registry.s3.amazonaws.com/buildpacks/mars/create-react-app.tgz" para que o *heroku* possa rodar o projeto. Após a adição destes componentes podemos dar o *deploy*, o *deploy* gera um registro de criação (*Build Log*) que podemos observar.

Ao finalizar o *deploy* o código está pronto para ser acessado por um navegador na internet passando o link de acesso ao projeto. Para acessar o projeto é através do endereço URL: https://vest-mais.herokuapp.com, na barra de pesquisa de seu navegador de preferência.

7. RESULTADOS

Com o decorrer do desenvolvimento do projeto, alguns resultados esperados foram alcançados. O desenvolvimento de toda estrutura organizacional para iniciar o desenvolvimento do MVP, bem como o sucesso do questionário para validar se seria viável desenvolver um site que suprisse a necessidade das pessoas em sua maioria.

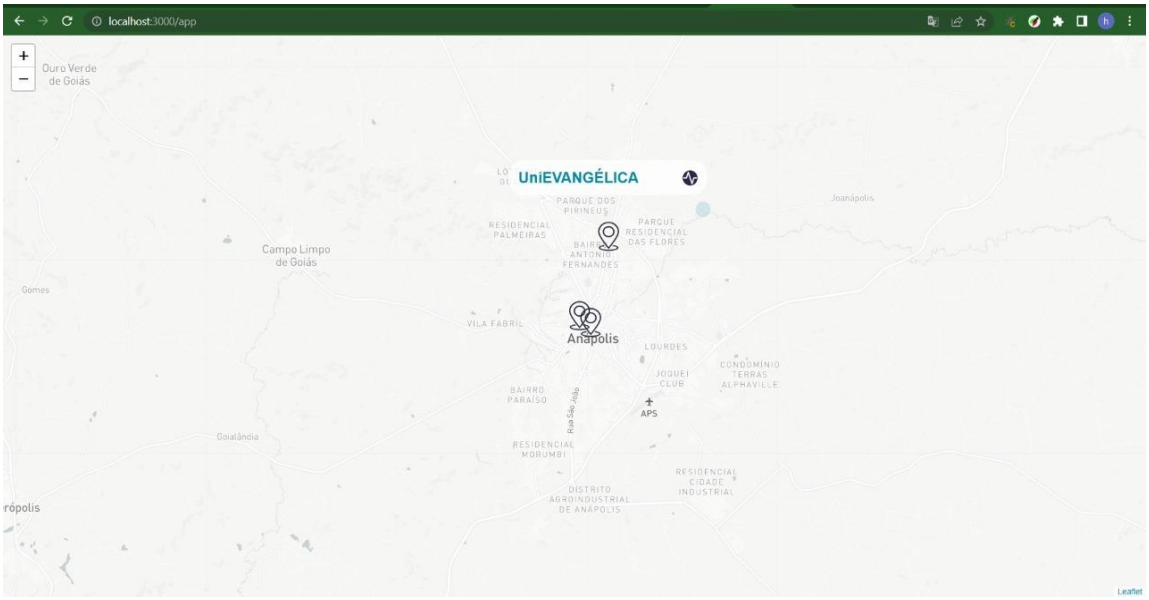
Os dados coletados na pesquisa de campo através do questionário, mostrou que dos 47 participantes do formulário foi retratado a possibilidade de desenvolver um MVP mostrando um interesse de utilizar o site proposto. 38,3% das pessoas acreditam que com um sistema online irá ajudar e facilitar que as pessoas doem vestimentas, já 76,6% acham que um mapa que exiba os locais de arrecadação próximos, aumentaria ainda mais.

Obteve-se também um feedback, ao qual 48,9% das pessoas ficaram indecisas e acabaram escolhendo a opção talvez, nos dando uma margem de uma futura aceitação. Com essa quantidade elevada, existe a possibilidade de converter isso em satisfação. Para isso, deve-se procurar focar na experiência de usuário, em uma interface chamativa e simples. Com o grande avanço na estruturação da equipe e com o desenvolvimento pessoal de cada integrante, foi definido e iniciado o desenvolvimento do MVP, sendo ele o objetivo do próximo desenvolvimento do projeto.

O desenvolvimento do MVP foi bastante satisfatório, tendo em vista que todos os objetivos primários foram alcançados para a utilização do sistema que auxilia o usuário na localização de ponto de coleta e de doação. Outro ponto a ser analisado foi a questão da cobertura de testes ter atingido uma pontuação superior a 90%, trazendo uma maior segurança para a hora de subir o projeto para produção.

Como telas obtidas durante o desenvolvimento deste projeto podemos destacar as do mapa e detalhe, que compõem o principal objetivo deste projeto possibilitando que o usuário possa escolher um dos pontos de coleta já cadastrado por um administrador para ter acesso a localidade e informações do mesmo.

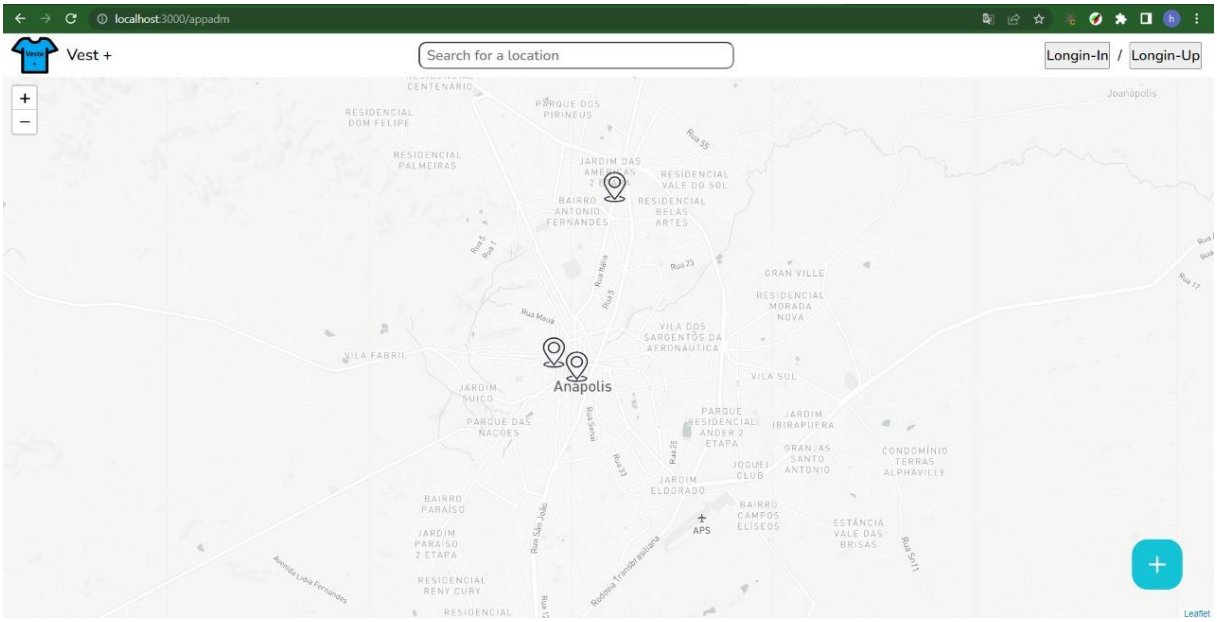
Figura 38 - Página do mapa



Fonte: Autores

A função de cadastro de estabelecimentos, pode ser gerenciado por um administrador, onde o mesmo é previamente cadastrado direto no banco de dados do sistema, tendo acesso a uma tela especial, como pode ser visto na Figura 39.

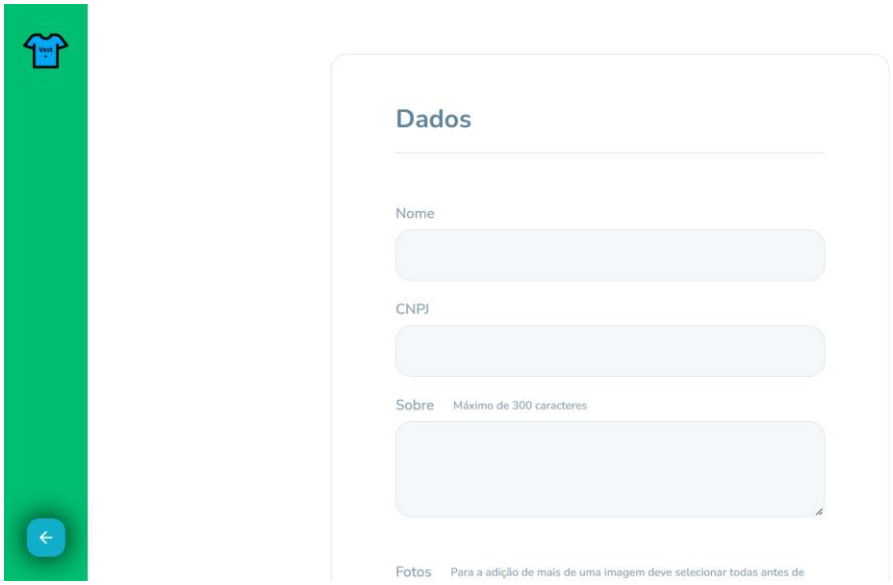
Figura 39 - Página do mapa para administrador



Fonte: Autores

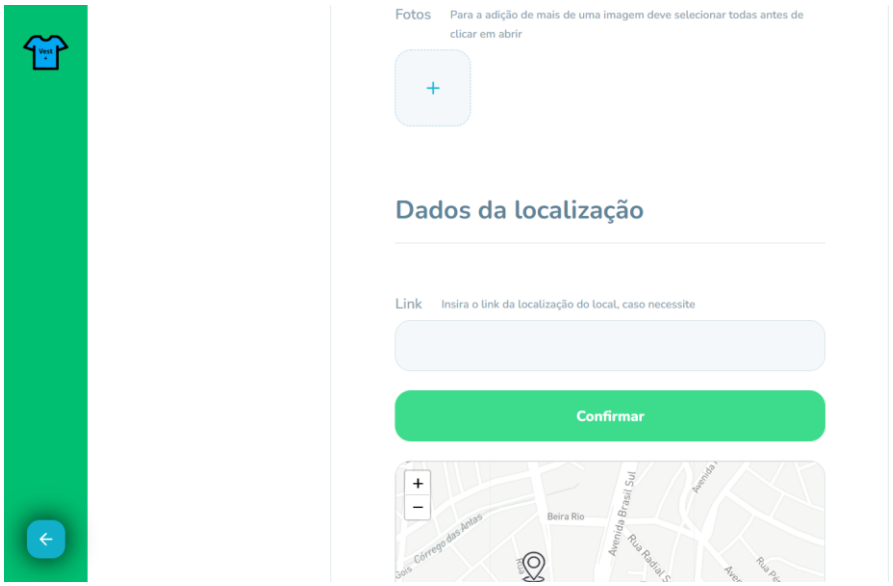
Ao clicar no botão “mais”, no canto inferior direito da tela, o administrador é direcionado a uma tela de cadastro de estabelecimentos, onde o mesmo fornecerá informações do local, como nome, imagens, descrição, localização, telefone para contato, horário de atendimento e outras informações. As figuras 40 a 43 mostram um exemplo desses campos.

Figura 40 - Parte 1 página de criação de estabelecimento



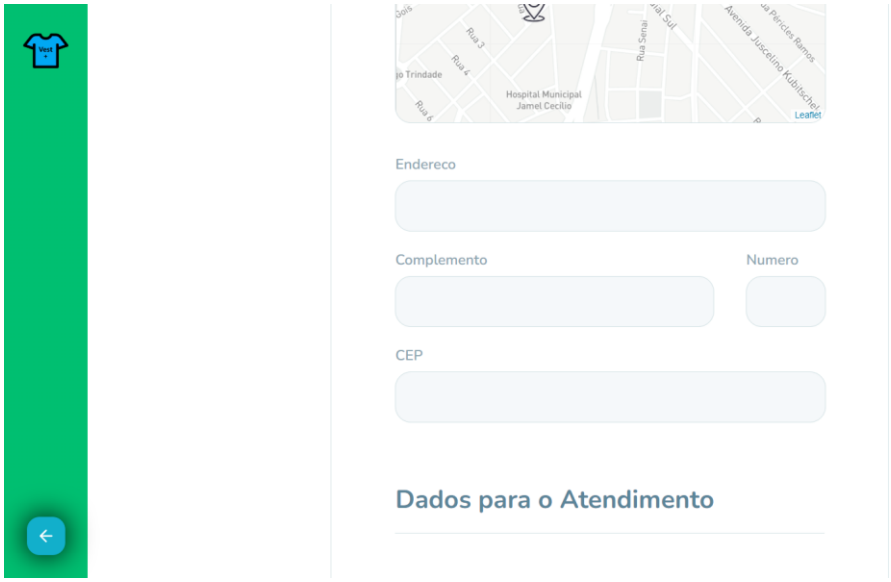
Fonte: Autores

Figura 41 - Parte 2 página de criação de estabelecimento



Fonte: Autores

Figura 42 - Parte 3 página de criação de estabelecimento



Fonte: Autores

Figura 43 - Parte 4 página de criação de estabelecimento

Fonte: Autores

Após a criação do estabelecimento pelo administrador, o novo estabelecimento já fica visível para o usuário no mapa de estabelecimentos. Ao clicar em um estabelecimento já cadastrado, as informações do local são apresentadas ao usuário, como nas Figuras 44 a 46.

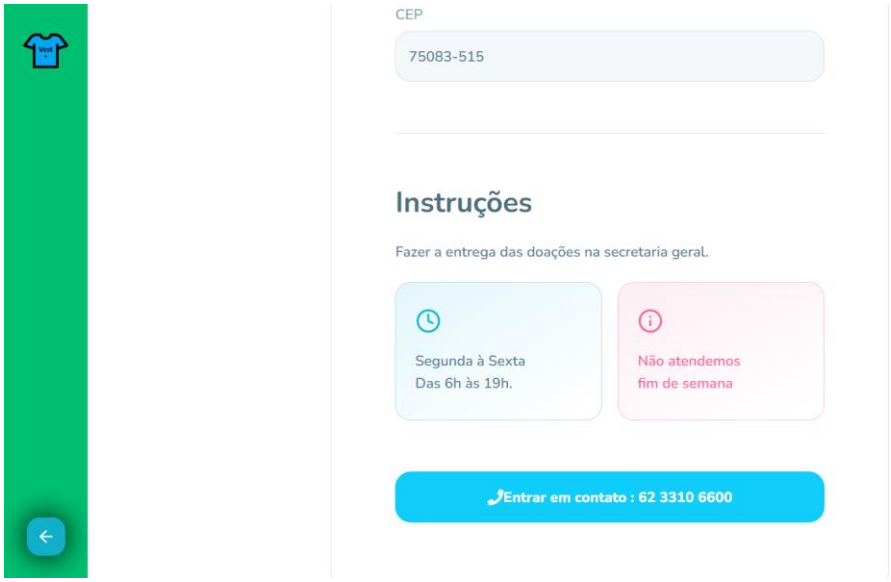
Figura 44 - Parte 1 página de informação do estabelecimento

Fonte: Autores

Figura 45 - Parte 2 página de informação do estabelecimento

Fonte: Autores

Figura 46 - Parte 3 página de informação do estabelecimento



Fonte: Autores

8. CONCLUSÃO

O projeto desenvolvido apresenta um site para arrecadação e doação de vestimentas. O site foi planejado para facilitar modificações futuras, onde será adicionado novas funcionalidades ao sistema atual. O site foi desenvolvido para ser usado em servidores remotos, que faz com que ele seja acessível de qualquer lugar que tenha acesso à internet.

Para dar início ao desenvolvimento do trabalho foi realizada uma pesquisa para analisar a necessidade do desenvolvimento do sistema. Essa pesquisa é de natureza aplicada e classificada como uma pesquisa exploratória, a fim de levantar dados para a construção da aplicação web.

O *back-end* foi desenvolvido com a linguagem *elixir* e o *front-end* desenvolvido em *React*. O desenvolvimento do *back-end* em *elixir*, foi escolhido por ser uma linguagem de fácil manutenibilidade, sendo de fácil desenvolvimento de novas funcionalidades, fácil manutenção e correção de bugs. O *React* foi escolhido por ter uma grande biblioteca e ser de fácil aprendizado, como a sintaxe é um pouco parecida com *javascript* foi de grande auxílio para quem já estava familiarizado com a linguagem. Por ser uma linguagem flexível, foi um fator decisivo para se utilizar, já que poderia de forma fácil se integrar com o *back-end*.

Com a pesquisa realizada, foi desenvolvida aplicação, quando o usuário não está cadastrado tem acesso apenas ao mapa dos locais de arrecadação das vestimentas, quando o usuário está cadastrado, mais funções ficam disponíveis para o mesmo acessar, como a avaliação dos locais de coleta, histórico de pesquisa e locais favoritos.

A utilização da aplicação web auxilia o usuário a encontrar locais onde receba vestimentas, avalie os locais para futuros usuários terem mais confiança nesses estabelecimentos cadastrados. Com essas informações reunidas em um só lugar, o usuário pode se sentir mais seguro em doar vestimentas e saber que chegaram às mãos de pessoas necessitadas.

Vale salientar que a ideia proposta foi o desenvolvimento do MVP, não sendo desenvolvido todas as funcionalidades definidas no planejamento da aplicação como um todo, mas a parte inicial para que possa ser lançado e validado pelos usuários e assim possa ter uma maior compreensão do que fora proposto supre a demanda e expectativa dos usuários.

Para projetos futuros, a ideia seria cadastrar usuários e mensurar a avaliação dos mesmos acerca dos estabelecimentos, tendo assim uma possível forma de encontrar estabelecimentos fraudulentos, além de desenvolver uma forma de pontuação para incentivar os usuários a realizarem doações.

REFERÊNCIAS

ACIDADEON. **Frio:** Campinas lança Campanha do Agasalho com mais de 100 pontos de coleta - ACidadeON Campinas. 13 maio 2022. Disponível em: https://www.acidadeon.com/campinas/cotidiano/cidades/NOT,0,0,1760977,frio-campinas-lanca-campanha-do-agasalho-com-mais-de-100-pontos-de-coleta.aspx. Acesso em: 16 maio 2022.

ANÁPOLIS, Prefeitura de. Ala 2 – **Base Aérea de Anápolis realiza a entrega de quase 500 itens arrecadados na Campanha do Agasalho**, 2021. Disponível em https://www.anapolis.go.gov.br/ala-2-base-aerea-de-anapolis-realiza-a-entrega-de-quase-500-itens-arrecadados-na-campanha-do-agasalho/. Acesso em: 06 set. 2021.

ANDRADE, Thiago Faria de. **Programação: Back-end vs Front-end vs Fullstack:** Escolha o seu futuro como programador! 2018. Disponível em: https://blog.algaworks.com/back-end[1]front-end-full-stack/. Acesso em: 01 nov. 2021.

AWS, Amazon. **Microsserviços**, 2021. Disponível em https://aws.amazon.com/pt/microservices/ Acesso em: 06 nov. 2021

BALLERINI , Rafaella. **HTML, CSS e Javascript, quais as diferenças?**, 25 fev. 2021. Disponível em: https://www.alura.com.br/artigos/html-css-e-js-definicoes. Acesso em: 21 out. 2021.

BALTHAZAR, Glauber et al. **Uma Abordagem Prática sobre a Aplicação do padrão MVC com o Framework Struts**. 2ª Edição Revista Eletrônica, p. 10, 31 dez. 2006. Disponível em: http://re.granbery.edu.br/artigos/MjQy.pdf. Acesso em: 23 maio 2022.

COULOURIS, George .; DOLLIMORE, Jean .; KINDBERG, Tim .; BLAIR, Gordon. **Sistemas Distribuídos**, 2013. 9788582600542. Disponível em: https://integrada.minhabiblioteca.com.br/#/books/9788582600542/. Acesso em: 23 out. 2021.

DEVMEDIA. **Guia de PostgreSQL**, c2021. Disponível em: https://www.devmedia.com.br/guia/guia-de-postgresql/34328. Acesso em: 06 nov 2021.

DOCKER. **Dockerfile reference**, 2021c. Disponível em: https://docs.docker.com/engine/reference/builder/. Acesso em: 23 maio 2022.

EIS, Diego; FERREIRA, Elcio. **HTML5 e CSS3:** com farinha e pimenta. São Paulo: Clube de Autores, 2012. P. 219

ELIXIR. **Elixir is a dynamic, functional language for building scalable and maintainable applications**, 2021. Disponível em: https://elixir-lang.org/. Acesso em: 14 out. 2021.

ELIXIRSCHOOL. **Lessons: Ecto**, 2021c. Disponível em: https://elixirschool.com/en/lessons/ecto#:~:text=Ecto%20is%20an%20official%20Elixir,Repo. Acesso em: 23 maio 2022.

FOWLER, Martin. **UML Essencial.**, 2011. Disponível em: https://integrada.minhabiblioteca.com.br/#/books/9788560031382/. Acesso em: 21 mai. 2022.

GOIÂNIA, Prefeitura de. **Aquecendo Corações é exemplo para demais municípios**, 2021. Disponível em https://www.goiania.go.gov.br/aquecendo-coracoes-e-exemplo-para-demais-municipios/. Acesso em: 06 set. 2021

GUEDES, Marylene. **Afinal, o que é TDD?** 2020. Disponível em: https://www.treinaweb.com.br/blog/afinal-o-que-e-tdd. Acesso em: 10 fev. 2022.

GUIMARÃES, Carolina; PUCHTA, Fabiana; ANDRADE, Maria. **Pandemia agrava falta de doações e voluntários**. 24 abr. 2021. Disponível em: https://medium.com/labjorfaap/pandemia-agrava-falta-de-doações-e-voluntários-afa8e9a98b8c. Acesso em: 15 maio 2022.

HEFLO. **Não confunda mais: Agile, Scrum e Kanban**, 2017. Disponível em: https://www.heflo.com/pt-br/agil/agile-scrum-e-kanban/. Acesso em: 21 out. 2021.

KANER, C. **The ongoing revolution in software testing**, 2004. Disponível em: <http://www.kaner.com/pdfs/TheOngoingRevolution.pdf>. Acesso em: 05 nov. 2021.

MARTIN, Robert. C. **Desenvolvimento Ágil Limpo**. Editora Alta Books, 2020. 9788550816890. Disponível em: https://integrada.minhabiblioteca.com.br/#/books/9788550816890/. Acesso em: 05 nov. 2021.

MICROSOFT. **Comunicação front-end de cliente**, 2021. Disponível em: https://docs.microsoft.com/pt-br/dotnet/architecture/cloud-native/front-end-communication. Acesso em: 15 set. 2021.

MICROSOFT. **JavaScript With Syntax For Types**. 2022c. Disponível em: https://www.typescriptlang.org/. Acesso em: 1 jun. 2022.

MILETTO, Evandro. M .; BERTAGNOLLI, Silvia.de. C. **Desenvolvimento de Software II** . Grupo A, 2014. 9788582601969. Disponível em: https://integrada.minhabiblioteca.com.br/#/books/9788582601969/. Acesso em: 01 nov. 2021.

NISHI, Fernando. **REQUISITOS DE SOFTWARE**: PROPOSTA, ESPECIFICAÇÃO E OTIMIZAÇÃO DE UM METAFRAMEWORK PARA A DOCUMENTAÇÃO. 2019. 123 p. CURSO SUPERIOR DE TECNOLOGIA EM GESTÃO DE TECNOLOGIA DA INFORMAÇÃO — INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE SANTA CATARINA, Florianópolis, 2019. Disponível em: https://repositorio.ifsc.edu.br/bitstream/handle/123456789/1216/TCC-FernandoMFNishi.pdf?sequence=1&isAllowed=y. Acesso em: 12 abr. 2022.

PARANÁ, G. do Estado do. **Paraná Solidário já cadastrou 460 doadores e 240 entidades, 2019**. Disponível em https://www.aen.pr.gov.br/modules/noticias/article.php?storyid=104774. Acesso em: 06 set. 2021.

PHOEINX. **Directory structure**, 2022c. Disponível em: https://hexdocs.pm/phoenix/directory_structure.html. Acesso em: 23 maio 2022.

POSTGRESQL. **The World's Most Advanced Open Source Relational Database**, 2021. Disponível em: https://www.postgresql.org/. Acesso em: 06 nov. 2021.

PRIKLADNICKI, Rafael.; WILLI, Renato.; MILANI, Fabiano. **Métodos Ágeis para Desenvolvimento de Software**. Grupo A, 2014. 9788582602089. Disponível em: https://integrada.minhabiblioteca.com.br/#/books/9788582602089/. Acesso em: 04 nov. 2021.

REACT. **Virtual DOM and Internals**, 2021. Disponível em https://reactjs.org/docs/faq-internals.html#gatsby-focus-wrapper. Acesso em: 06 nov. 2021

RIES , Eric. **A Startup Enxuta:** Como os empreendedores atuais utilizam inovação contínua para criar empresas extremamente bem-sucedidas. São Paulo: Leya, 2011.

RIES, Eric - Caroli. **Minimum Viable Product**, 2009. Disponível em https://www.caroli.org/en/eric-ries-on-mvp-2009/. Acesso em: 06 set. 2021.

SCHWABER, Ken; SUTHERLAND, Jeff. **Guia do Scrum - Um guia definitivo para o Scrum:** As regras do jogo. 1 jul. 2013. Disponível em: https://scrumguides.org/docs/scrumguide/v1/Scrum-Guide-Portuguese-BR.pdf. Acesso em: 13 abr. 2022.

SERASA EMPREENDEDOR. **O que você precisa saber sobre MVP**, 2018. Disponível em: https://empresas.serasaexperian.com.br/blog/o-que-e-mvp-ou-melhor-produto-minimo-viavel-e-como-utiliza-lo/. Acesso em: 15 set. 2021.

SOUZA, Ivan de - Rockcontent. **PostgreSQL:** saiba o que é, para que serve e como instalar**,** 2020. Disponível em https://rockcontent.com/br/blog/postgresql/. Acesso em: 06 nov. 2021

VENTURA, Plínio. **Entendendo definitivamente o que é um Caso de Uso**, 2016. Disponível em: https://www.ateomomento.com.br/o-que-e-caso-de-uso/. Acesso em : 03 Abr. 2022

VIANA, Daniel. **O que é front-end e back-end?** 2017. Disponível em: https://www.treinaweb.com.br/blog/o-que-e-front-end-e-back-end/. Acesso em: 01 nov. 2021.

WINBLAD, Kjell - Elang. **Practical functional programming for a parallel world**, 2021. Disponível em https://www.erlang.org/. Acesso em: 06 nov. 2021

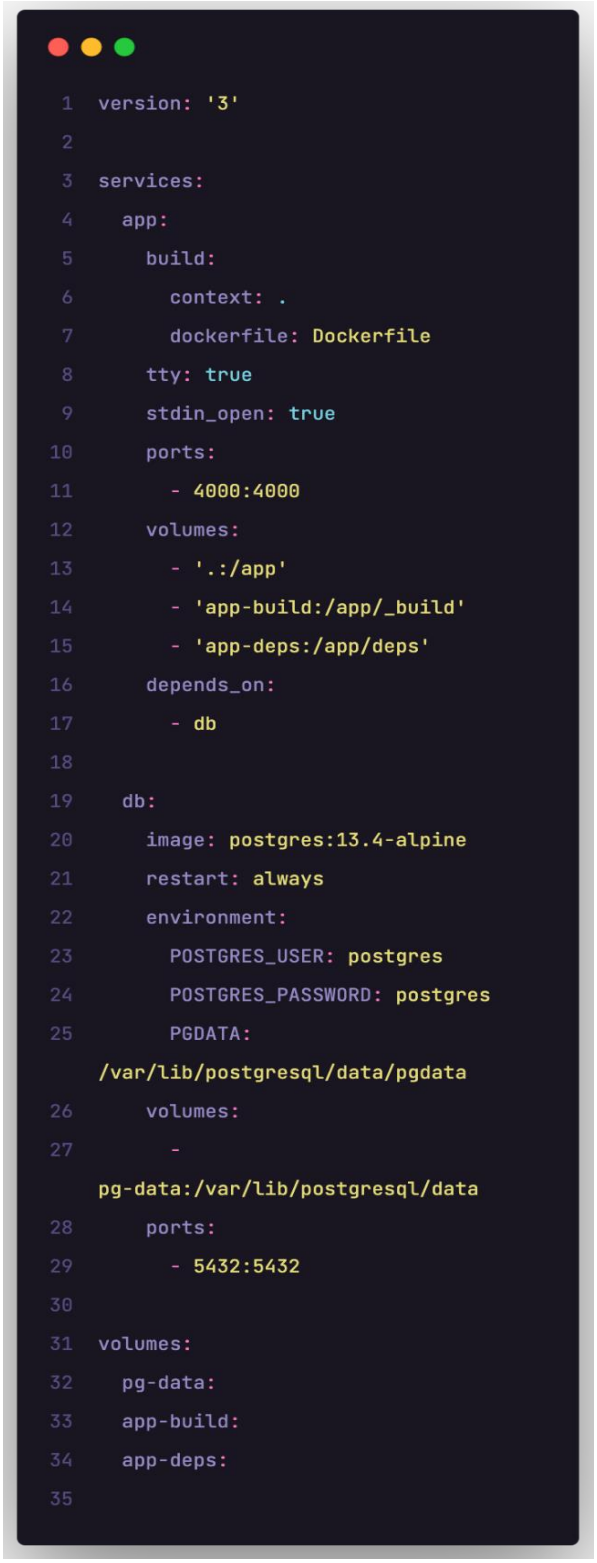
APÊNDICE

APÊNDICE A - Dockerfile

```
FROM elixir:1.12.2-alpine

RUN mkdir -p /opt/app && \
  chmod -R 777 /opt/app

RUN apk update && \
  apk add inotify-tools git build-base && \
  rm -rf /var/cache/apk/*

WORKDIR /app

COPY . /app

RUN
 mix do local.hex --force, local.rebar --force

COPY config/ /app/config/
COPY mix.exs /app/
COPY mix.* /app/

RUN mix do deps.get, deps.compile

RUN mix phx.digest

RUN mix compile
RUN mix phx.digest

EXPOSE 4000

CMD ["mix", "phx.server"]
```

Fonte: Autores

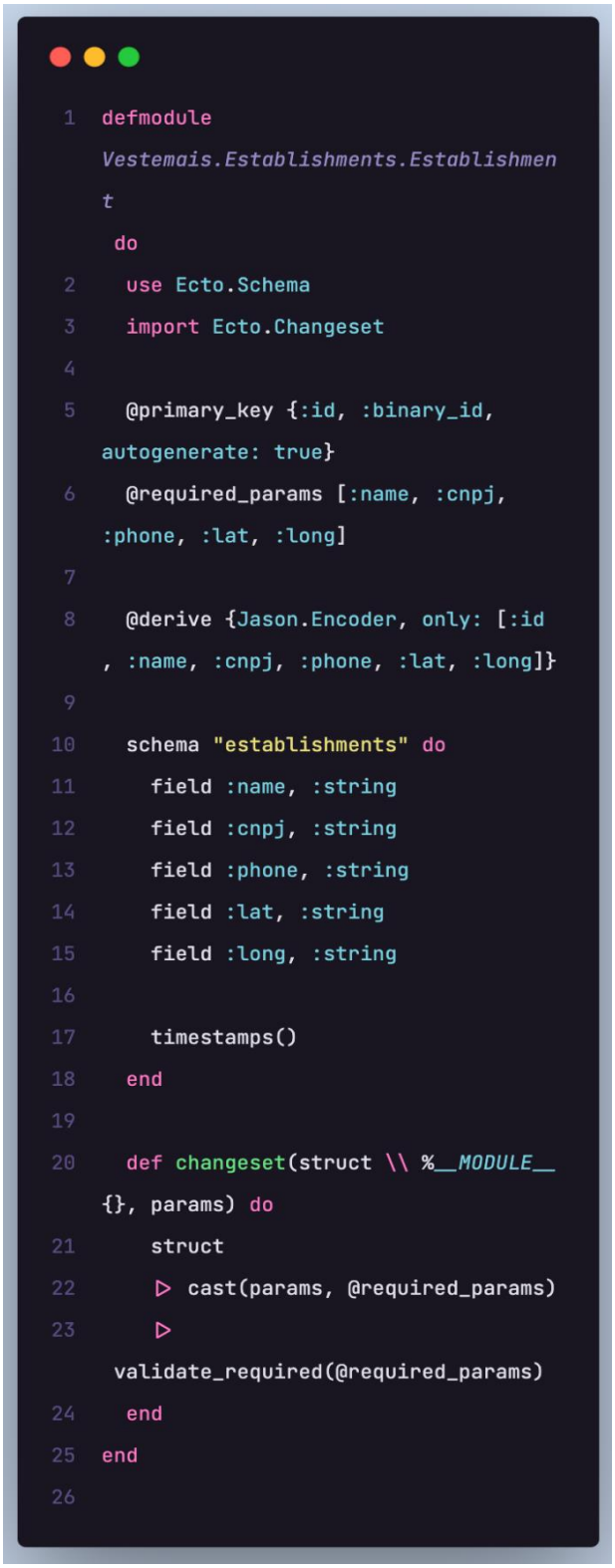
APÊNDICE B - Docker-Compose.yml



```yaml
version: '3'

services:
  app:
    build:
      context: .
      dockerfile: Dockerfile
    tty: true
    stdin_open: true
    ports:
      - 4000:4000
    volumes:
      - '.:/app'
      - 'app-build:/app/_build'
      - 'app-deps:/app/deps'
    depends_on:
      - db

  db:
    image: postgres:13.4-alpine
    restart: always
    environment:
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: postgres
      PGDATA:
/var/lib/postgresql/data/pgdata
    volumes:
      -
pg-data:/var/lib/postgresql/data
    ports:
      - 5432:5432

volumes:
  pg-data:
  app-build:
  app-deps:

```

Fonte: Autores

APÊNDICE C - Schema de Establishment.ex



```elixir
defmodule Vestemais.Establishments.Establishment do
  use Ecto.Schema
  import Ecto.Changeset

  @primary_key {:id, :binary_id, autogenerate: true}
  @required_params [:name, :cnpj, :phone, :lat, :long]

  @derive {Jason.Encoder, only: [:id, :name, :cnpj, :phone, :lat, :long]}

  schema "establishments" do
    field :name, :string
    field :cnpj, :string
    field :phone, :string
    field :lat, :string
    field :long, :string

    timestamps()
  end

  def changeset(struct \\ %__MODULE__{}, params) do
    struct
    |> cast(params, @required_params)
    |> validate_required(@required_params)
  end
end
```

Fonte: Autores

APÊNDICE D - Establishment_Controller.ex

```elixir
defmodule VestemaisWeb.EstablishmentsController do
  use VestemaisWeb, :controller

  alias Vestemais.Establishments.Establishment
  alias VestemaisWeb.FallbackController

  action_fallback FallbackController

  def index(conn, _params) do
    with {:ok, establishments} <- Vestemais.get_all_establishments() do
      conn
      |> put_status(:ok)
      |> render("index.json", establishments: establishments)
    end
  end

  def create(conn, params) do
    with {:ok, %Establishment{} = establishment} <- Vestemais.create_establishment(params) do
      conn
      |> put_status(:created)
      |> render("create.json", establishment: establishment)
    end
  end

  def show(conn, %{"id" => id}) do
    with {:ok, establishment} <- Vestemais.get_establishment_by_id(id) do
      conn
      |> put_status(:ok)
      |> render("establishment.json", establishment: establishment)
    end
  end
end
```

Fonte: Autores