CENTRO ESTADUAL DE EDUCAÇÃO TECNOLÓGICA PAULA SOUZA

ETEC ZONA LESTE

Ensino Médio com Habilitação Profissional de Técnico em Desenvolvimento de Sistemas - AMS

Lucas Rodrigues Batista

Lucas Wernek Garcia dos Santos

Nelson Samuel Gomes Nóbrega

PEAT: Carteira de vacinação digital para cães e gatos

São Paulo

2022

Lucas Rodrigues Batista
Lucas Wernek Garcia dos Santos
Nelson Samuel Gomes Nóbrega

PEAT: Carteira de vacinação digital para cães e gatos

Trabalho de Conclusão de Curso apresentado ao Curso do Ensino Médio com Habilitação Profissional de Técnico em Desenvolvimento de Sistemas AMS da Etec Zona Leste, orientado pelo Professor Ediney Ciasi Barreto, como requisito final para obtenção do título de Técnico em Desenvolvimento de Sistemas.

São Paulo
2022

AGRADECIMENTOS

Nós agradecemos ao professor Rogério Bezerra Costa e a professora Vilma Cardoso dos Santos, pela orientação na criação do projeto de conclusão de curso, aos professores Ediney Ciasi Barreto e Wagner de Oliveria Lucca por ajudar na construção da base do projeto, aos professores Jeferson Roberto de Lima e Carlos Alberto Pereira da Silva por nos guiar na definição de tecnologias. Nós também gostaríamos de agradecer a Juliana Justiço Mioshi, por ter nós fornecido uma entrevista para levantamento de requisitos do nosso sistema.

RESUMO

O objetivo deste projeto de conclusão de curso é apontar a necessidade de uma ferramenta que seja capaz de monitorar o andamento veterinário de animais de estimação. Devido ao aumento continuo desses pets, os donos acabam se descuidando no acompanhamento veterinário de todos os animais que possui, gerando assim possíveis novos problemas de saúde. Com isso em mente, utilizamos as tecnologias *React Native* e Firebase, para criar um aplicativo *mobile*, que será capaz de ajudar esses donos a terem controle do registro de vacinas de seus cães e gatos. Este aplicativo possuirá as opções de cadastro de *pets*, veterinários e petshops, ele possuirá uma *API* de mapa, que irá mostrar para o usuário o endereço dos estabelecimentos parceiros do nosso aplicativo, estabelecimentos esses que irão fornecer serviços como banho e tosa para os clientes, estes que podem ser agendados pelos para o cliente pelo aplicativo, e por último e possível criar uma ficha de vacinação digital para cada animal de estimação. Em conjunto ao aplicativo *mobile*, criamos um site utilizando as tecnologias HTML, CSS, Bootstrap e Javascript, para que esses estabelecimentos possam visualizar os agendamentos de serviços que foram realizados pelos usuários. Acreditamos que com o uso continuo desse aplicativo, em soma as campanhas de vacinação do governo, as taxas de vacinação entre os cães e gatos irão aumentar, melhorando assim a qualidade de vida desses animais.

PALAVRAS-CHAVE: Aplicativo, Animais de estimação, Petshops, Veterinários, Saúde, Ficha de vacinação digital, Agendamentos, Vacinas.

ABSTRACT

The objective of this course conclusion project is to point out the need for a tool that is capable of monitoring the veterinary progress of pets. Because the number of these pets is constantly increasing, the owners are ending up neglecting the veterinary monitoring of all the animals they have, what will possibly generate new health problems. With that in mind, we used React Native and Firebase technologies to create a mobile app that will be able to help these owners to take control of the vaccination record of the theirs cats and dogs. This application will have the options for registering pets, veterinarians and petshops, it will have a map API, which will show the user the address of the establishments partner of our application, establishments who will provide services like bath and groom, these that can be scheduled by the cliente through the application and it will create a digital vaccine form for each pet. Together with the mobile application, we created a website using HTML, CSS, Bootstrap and Javascript Technologies, so these establishements can view the servisce appointmentes that were made by users. We believe that with the continued use of this application, in addition to government vaccination campaigns, the vaccination rates among dogs and cats will increase, resulting in improviment on the quality of life for these animals.

KEYWORDS: Application, Pets, Pet Shops, Veterinarians, Health, Vaccine, Digital vaccination form, Schedules.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

Application (APP)

Application Programming Interface (API)

Cascading Style Sheets (CSS)

Conselho Regional de Medicina Veterinária do Estado de São Paulo (CRMV-SP)

Extensible Markeup Language (XML)

Hypertext Markup Language (HTML)

Hypertext Reference (HREF)

Instituto Brasileiro de Geografia e Estatística (IBGE)

Identidade (ID)

Java Serialization to XML (JSX)

Uniform Resource Locator (URL)

Unified Modeling Language (UML)

SUMÁRIO

1. INTRODUÇÃO

Devido ao constante aumento de animais de estimação nas casas brasileiras, sentimos necessário a criação de uma ferramenta que fosse capaz de monitorar o acompanhamento veterinário desses *pets* e pudesse recomendar bons veterinários e *petshops*, pois com o aumento exponencial desses estabelecimentos, a população dona de *pet* fica indecisa e insegura na escolha de um bom provedor de serviços aos animais de estimação.

Já que esse aumento vai gerar diversos novos veterinários e *petshops*, alguns destes serão de boa e má qualidade, fazendo com que o único meio para se identificar se o estabelecimento é recomendável ou não é testando, entretanto não é nada recomendável testar se um local é de boa qualidade colocando a saúde do animal de estimação em risco. Por este motivo o nosso aplicativo possuirá um sistema de recomendação de *petshops* e veterinários bem avaliados e de qualidade, que serão capazes de cuidar corretamente do seu *pet*.

Outro ponto que foi levado em conta para a criação do aplicativo veio após analisar o apontamento feito pelo IBGE, onde os níveis de vacinação contra a raiva em cães e gatos estão diminuindo constantemente desde 2013, onde os níveis estavam em 75%, já em 2019 eles foram reduzidos para 72% e está porcentagem está continuamente abaixando. Isto mostra como a população está continuamente deixando a vacinação dos seus *pets* de lado, podendo assim gerar futuros problemas de saúde aos animais.

Com todos esses benefícios possíveis para os animais de estimação, decidimos optar pela tecnologia *React Native* para criar a base para o nosso aplicativo. Pois como ele possui em sua constituição *HTML, JavaScript* e *CSS* embutidos, a criação das telas gráficas e as funções do *app* serão criadas em *React Native*. Na criação do banco de dados, que servirá para armazenar os dados cadastrados, como animais de estimação, *petshops* e veterinários, optamos por utilizar o *Firebase*. O motivo pela escolha desse Sistema de Gerenciamento de Banco de Dados, foi decorrente da sua excelente compatibilidade com aplicativos de celular, em conjunto a característica “não relacional” deste banco de dados, ele é capaz de armazenar a imensa quantidade de dados que serão gerados todos os dias pelo nosso *app*.

Todas essas funcionalidades irão facilitar a vida dos donos de animais de estimação na hora de verificar se os seus *pets* precisam de algum auxílio médico, consequentemente gerando um aumento na qualidade de vida desses animais.

2. REFERENCIAL TEÓRICO

Neste capítulo, iremos apresentar as tecnologias e ferramentas que foram utilizadas para a criação deste projeto de conclusão de curso.

2.1. HTML

O *HyperText Markup Language (HTML)* é uma linguagem de marcação de hipertexto, de acordo com Silva (2019, p.9) “hipertexto como todo o conteúdo inserido em um documento para a web e que tem como principal característica a possibilidade de se interligar a outros documentos web”. Ao longo dos anos ele evoluiu, e atualmente está na versão *HTML* 5, que tem capacidades adaptativas para funcionar em qualquer navegador web.

No *HTML* você pode indicar um valor de atributo correspondente a aquele valor, como disse Flatschart (2011, p.43) “Dentro de cada atributo é indicado um valor que pode ser textual, numérico ou booleano”. Os elementos booleanos têm apenas dois valores possíveis, exemplo disso são: sim ou não.

Os atributos que são comuns que estão presentes em todos os elementos do *HTML* são: *accesskey*, que indica um atalho do teclado para acessar alguma função, temos também as *class*, que são as classes no *HTML*, geralmente são destinadas para utilização de *CSS*, outro atributo global e o *contenditable*, este indica de o conteúdo digitado nele pode ou não ser modificado, e por último temos o *contextmenu*, este que tem o propósito de indicar um conteúdo em um menu.

2.1.1.Tags

O *HTML* é baseado em *tags*, onde os sinais <> e </> identificam as funções de cada elemento digitado, a Figura 1 contêm um código em *HTML,* este que cria um formulário de cadastro de um usuário. Onde as *tags* e outras diversas funções dessa linguagem de programação serão postas em prática:

Figura 1 - Código em HTML

```
<!DOCTYPE html>
  <html lang="pt-br">
  <head>
  <meta charset="UTF-8" />
  <title> Formulário HTML</title>
   <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/font-awesome/4.7.0/css/font-
awesome.min.css">
   <link rel="stylesheet" href="..\Exemplo HTMLCSSJS\css.css">
   </head>

   <body>
   <div class='container'>
     <div class='card'>
      <div>
       <h1> Cadastrar </h1>
        <img src="..\Exemplo HTMLCSSJS\logo 2.png" alt="logo" id="imag">
       </div>

       <div id='msgError'></div>
       <div id='msgSuccess'></div>

               <div class="label-float">
                  <input type="text" id="nome" placeholder=" " required>
                  <label id="labelNome" for="nome">Nome</label>
               </div>

               <div class="label-float">
                  <input type="text" id="usuario" placeholder=" " required>
                  <label id="labelUsuario" for="usuario">Usuário</label>
               </div>

               <div class="label-float">
                  <input type="password" id="senha" placeholder=" " required>
                  <label id="labelSenha" for="senha">Senha</label>
                  <i id="verSenha" class="fa fa-eye" aria-hidden="true"></i>

               </div>

               <div class="label-float">
                  <input type="password" id="confirmSenha" placeholder=" " required>
                  <label id="labelConfirmSenha" for="confirmSenha">Confirmar Senha</label>
                  <i id="verConfirmSenha" class="fa fa-eye" aria-hidden="true"></i>
               </div>

               <div class='justify-center'>
                  <button onclick='cadastrar()'>Cadastrar</button>
               </div>
        </div>
    </div>
  </body>
</html>
```

Fonte: Autoria própria, 2022

Nesta Figura 1, podemos observar diversas *tags*. Começando pelas responsáveis por identificar se os códigos de programação são *HTML* <*html*>, <*meta*>, em seguida temos a *tag* <*head*>, o objetivo dela e posicionar o conteúdo na parte superior da página web.

Então é apresentado o <*title*>, que serve para dar o nome da página web, embaixo dela está o <*body*>, dentro desta *tag* irá ficar todo o conteúdo do site. Uma das *tags* mais importantes e utilizadas no *HTML* é a <*div*>, esta que armazena as funções do

site, então cada nova utilidade que colocada no site, fica dentro da sua própria <*div*>, é estas separações ajudam a reduzir a quantidade de erros que possam vir a ocorrer, durante a adição de novas funções e tecnologias.

Os <*h1*>, <*h2*> e <*h3*> tem o propósito de mudar o tamanho da fonte dos textos escritos nela, ou seja, o <*h1*> é responsável pela maior fonte, usualmente utilizada para títulos, o <*h2*> já possui um tamanho menor de texto, servindo para subtítulos e botões, e por fim o <*h1*> que possui a menor fonte entre eles, seguindo essas *tags* para escrita temos o <*p*>, que serve exclusivamente para escrever textos.

Segundo Pinheiro (1997, p.6) “Para colocar um arquivo ou página dentro do seu código *HTML* se utiliza o <*A HREF*=”endereço”> Texto do *Link*<*a*/>.” O <input> serve para receber dados de um usuário em um formulário. Já *tag* <*label*> funciona em conjunto com o <input>, direcionando um elemento a outro, e serve para esconder botões do *layout* com o *CSS*. E por fim a *tag* <*button*>, a função dela e criar um botão na página *web*, entretanto esse botão só irá funcionar se a tecnologia *JavaScript* for utilizada em conjunto.

Essas são apenas algumas *tags* dentro do *HTML*, dentre milhares de opções que podem ser encontradas para projetar o seu site da forma que você desejar.

O resultado deste código em *HTML* está apresentado na Figura 2:

Figura 2 - Resultado HTML

Fonte: Autoria própria, 2022

Como é possível observar, o *HTML* possui capacidade para montar o *website* da forma que o usuário desejar, e se somar a isto a sua fácil conexão com diversas tecnologias

diferentes, as possibilidades para criação de páginas *web* distintas são praticamente infinitas.

Tornando assim o *HTML* uma ferramenta importantíssima para criação do *front-end* do nosso aplicativo.

2.2. CSS

O *Cascading Style Sheet (CSS)*, é um mecanismo para dar um estilo a uma página *web*, ele é muito utilizado para complemento do *HTML*, pois com ele você pode personalizar o *site* da forma que desejar.

Para Silva (2015, p.129) *CSS* tem funcionalidades destinas a manipular e definir *boxes* para modelar o *layout* da página *web*, ou seja, definir como esses *boxes* se localizam na tela.

Com ele você pode personalizar as cores de um elemento específico, alterar a fonte, as dimensões e margens, o espaçamento entre as *boxes*, a localização dos *boxes* na página web e por último adicionar animações ao seu site.

O *CSS* fornece três posicionamentos diferentes ao programador, o primeiro deles e o *normal flow* (fluxo normal), o segundo é o *float* (flutuado) e o último e o *absolute* (absoluto).

O *normal flow* serve para posicionar os elementos de forma padrão na vertical ou horizontal, já o *float* e quando o box permanece no fluxo padrão, mas ele e colocado em posições onde ele parece estar flutuando em meio do texto. O último deles o *absolute* e utilizado para posicionar os boxes em locais diferentes do fluxo normal, fazendo isso por coordenadas da página *web*.

2.2.1. Propriedades

Para criar documentos em *CSS* e necessário utilizar a sintaxe própria dele, mesmo que ele seja usualmente combinado ao *HTML*. Ele é separado em três setores: seletor: onde o programador seleciona quais elementos do *HTML* serão modificados, propriedade: aqui irão ser postos as características que você deseja modificar, e o último, valor: neste se delimita os valores desejados para o elemento assumir.

A Figura 3 a seguir possui códigos em *CSS* que irão transformar o aspecto visual da página web em *HTML* da Figura 2 de algo desleixado e não formatado, para um site organizado e bonito:

Figura 3 - Código em CSS

```css
@import url('https://fonts.googleapis.com/css2?
family=Roboto:ital,wght@0,100;0,300;0,400;0,500;0,700;0,900;1,
100;1,300;1,400;1,500;1,700;1,900&display=swap');

* {
  margin: 0;
  padding: 0;
  font-family: 'Roboto', sans-serif;
}
body {
  background: linear-gradient(0, #383677, #000000);
  background-color: #757575;
  background-repeat: no-repeat;
  background-size: cover;
  background-attachment: fixed
}
.container {
  display: flex;
  justify-content: center;
  width: 100%;
  margin-top: 100px;
}

.card {
  background-color: white;
  margin-top: center;
  padding: 30px;
  border-radius: 4%;
  box-shadow: 3px 3px 1px 0px #00000060;
  width: 400px;
}

#imag {
  width: 100px;
  display: inline;
  margin-left: 300px;
  position: absolute;
  top: 101px;
}

h1{
  text-align: center;
  margin-bottom: 20px;
  color: #757575;
}

.label-float input{
  width: 100%;
  padding: 5px 5px;
  display: inline-block;
  border: 0;
  border-bottom: 2px solid #757575;
  background-color: transparent;
  outline: none;
  min-width: 180px;
  font-size: 16px;
  transition: all .3s ease-out;
  border-radius: 0;
}

.label-float{
  position: relative;
  padding-top: 13px;
  margin-top: 5%;
  margin-bottom: 5%;
}

.label-float input:focus{
  border-bottom: 2px solid #4038a0;
}

.label-float label{
  color: #757575;
  pointer-event: none;
  position: absolute;
  top: 0;
  left: 0;
  margin-top: 13px;
  transition: all .3s ease-out;
}

.label-float input:focus + label,
.label-float input:valid + label{
  font-size: 13px;
  margin-top: 0;
  color: #4038a0
}

button{
  background-color: transparent;
  border-color: #757575;
  color: #757575;
  padding: 7px;
  font-weight: bold;
  font-size: 12pt;
  margin-top: 20px;
  border-radius: 4px;
  cursor: pointer;
  outline: none;
  transition: all .4s ease-out;
}

button:hover{
  background-color: #0d009c;
  color: #fff;
}

.justify-center{
  display: flex;
  justify-content: center;
}

.fa-eye{
  position: absolute;
  top: 15px;
  right: 10px;
  cursor: pointer;
  color: #0d009c;
}

#msgError{
  text-align: center;
  color: #ff0000;
  background-color: #ffbbbb;
  padding: 10px;
  border-radius: 4px;
  display: none;
}

#msgSuccess{
  text-align: center;
  color: #00bb00;
  background-color: #bbffbe;
  padding: 10px;
  border-radius: 4px;
  display: none;
}
```

Autor: Autoria própria, 2022

Podemos observar neste código que os três aspectos que definem a linguagem *CSS* estão presentes neste exemplo, o primeiro deles é o seletor, que pode ser identificado

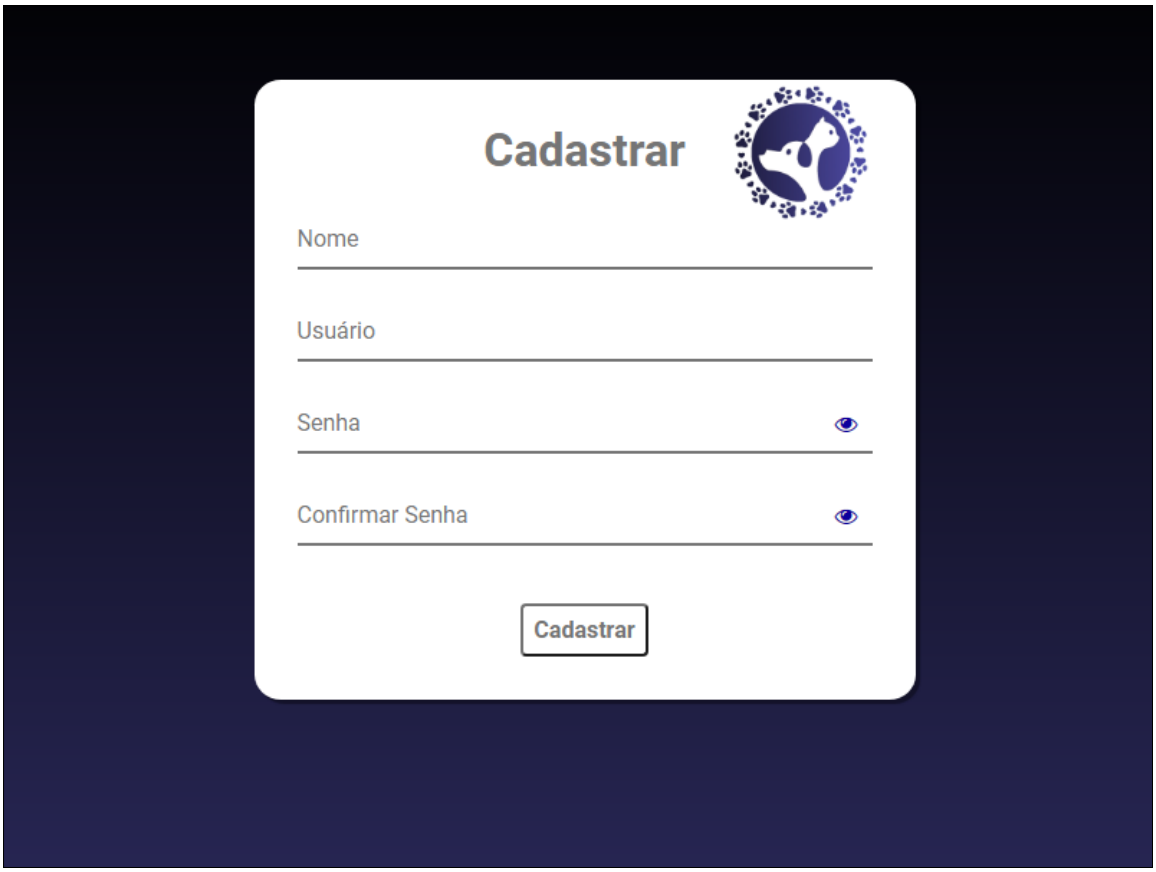
na Figura 3, pelo título dos elementos, pôr a cor acinzentada e pela presença de chaves ({ }), esse seletor irá selecionar um elemento que o usuário deseja modificar do código em *HTML* ou *JavaScript.* Em seguida temos o aspecto propriedade, que pode ser identificado pelas cores amarelas na Figura 3, todos os parâmetros com essas cores realizam diferentes modificações no *website:*

- *Import*: Ele faz a importação de uma fonte de texto da *internet* para o seu projeto.
- *Margin*: Delimita quais são as margens que cada elemento deve ter.
- *Padding*: Determina qual será a distância que ficara as informações de um elemento das bordas dele.
- *Font*: Todos os elementos da família *font*, servem para alterar o tamanho ou o tipo de fonte que é utilizada.
- *Background*: A família *background* altera os parâmetros do fundo de um elemento do *website,* modificando o tamanho, a cor, a quantidade de vezes que ele se repete.
- *Display*: O display serve para determinar como o conteúdo dentro dela deve se posicionar, tendo a possibilidade se ficar em linha, blocos, ou uma tabela.
- *Justify*: Justifica um texto.
- *Width*: Modifica a largura de um elemento.
- *Box*: A família *box* irá modificar diversas características de um elemento, esses podendo ser o tamanho, largura ou altura, a adição de uma sombra, entre diversos aspectos.
- *Position*: Irá alterar a posição que um elemento na página *web*.
- *Color*: Altera a cor de um elemento.
- *Border*: Ela adiciona bordas aos elementos.
- *Transition*: Cria para animações em páginas *HTML*.
- *Outline*: Ele é parecido com o *border* entretanto, o *outline* altera como as bordas de um elemento estão posicionadas e pode adicionar mais uma borda a um mesmo elemento.
- *Text*: Essa propriedade vai modificar como um texto irá se posicionar, ou o tamanho dele, como ele está alinhado a algum elemento entre diversas outras modificações que podem ser realizadas.

E por último temos a característica valor, está que pode ser identificada nas Figuras 8 e 9 pela cor cinza e verde claro, e pelo ponto e vírgula. Esses valores vão ser delimitar a quantidade de um determinado elemento, ou qual será o tipo de alteração que o usuário deseja fazer.

Na Figura 4, está o resultado de todas as modificações que foram adicionadas ao código *HTML* da Figura 2:

Figura 4 - Resultado HTML + CSS



Fonte: Autoria Própria, 2022

De acordo com Silva (2007, p.67) o *CSS* é formado de três formas: autor, usuário e agente de usuário. O autor cria o estilo para o *website*, o usuário e facultado a criar o *CSS* para sua utilização, e o navegador tem um *CSS* próprio.

E essa cascata possui uma ordem, primeiramente ele precisa verificar se o estilo está de acordo com o documento do *CSS*, depois ele deve seguir a ordem de prioridade que foi citada acima, autor, usuário e navegador. Após todos esses fatores serem levados em conta a ordem e delimitada.

Como disse Scheidt (2015, p.5) os Seletores são criados conforme valores são determinados nos elementos, ou seja, para o elemento *text*, nós podemos definir o tipo da fonte e o seu tamanho.

Esses valores e atributos são posicionados em colchetes, o que facilita na programação, pois a necessidade de posicionar classes para esses elementos e retirada da codificação. Tornando os códigos de *CSS* muito limpos e curtos.

Com todas essas funcionalidades, o *CSS* se torna uma excelente linguagem para embelezar o *front-end* do nosso aplicativo, em conjunto com o *HTML* eles irão determinar o visual do *app.*

2.3. Bootstrap

De acordo com Lima (2022) o *Bootstrap* é um framework que possui diversas estruturas de CSS previamente criadas, que são utilizadas para construir o *front-end* de telas de sites para desktop e celular. O *Bootstrap* usufrui das principais tecnologias para criação de *websites,* podemos confirmar isso a partir da afirmação feita por Bootstrap (2022) o *Bootstrap* se utiliza de diversos estilos globais, estas que são guiadas a normalização e estilos que funcionam entre diversos navegadores. Entre as tecnologias principais que estão presentes no *Bootstrap* temos o *HTML*, *CSS* e *Javascript*, ele as reúne para estilizar e adicionar funções aos seus componentes.

Um dos pontos fortes do *Bootstrap* é sua capacidade de tornar os sites responsivos, ou seja, esses *websites* se adaptem a qualquer tamanho de tela, tanto de computadores, como de celulares, fornecendo assim mais liberdade aos desenvolvedores *front-end* para utilizar todos os componentes presentes nessa tecnologia, sem que aconteça nenhuma quebra de responsividade.

Como foi dito por Mariano(2022, p.15) “O sistema de grade (*grid system*) do Bootstrap permite posicionar elementos com precisão. Para isso, Bootstrap divide a página em 12 colunas.” Essa divisão permite que todos os elementos do site sejam colocados nas posições desejas de forma mais fácil, reduzindo a chance de conflitos entre <*div*>, que podem acabar se sobrepondo e assim estragando o visual do site. A seguir na Figura 5, está um exemplo de como *Boostrap* é constituído:

Figura 5 - Código do Booststrap

```
<title>Peat</title>
  </head>
  <body class="bg-light">
    <div class="container-fluid">
      <div class="row">
        <div class="col-10 offset-1 text-center mt-5 pb-4 text-dark">
          <h1 class="h1">Peat</h1>
        </div>
        <div class="col-10 offset-1 bg-white shadow mb-5 border border-success">
          <div class="row">
            <div
              id="loginLogo"
              class="col-6 p-4 bg-success d-flex justify-content-center">
              <img src="./images/logo3.png" alt="Logo Peat" />
            </div>
            <div class="col-6 p-5">
              <div id="loginForm">
                <h2 class="h2 text-center text-dark mb-3">Acesso ao Peat</h2>
                <div class="form-group">
                  <label for="email">Email</label>
                  <input
                    type="email"
                    name="email"
                    id="email"
                    class="form-control"/>
                </div>
                <div class="form-group">
                  <label for="password">Senha</label>
                  <input
                    type="password"
                    name="password"
                    id="password"
                    class="form-control"
                  />
                </div>
                <button
                  type="button"
                  class="btn btn-outline-success"
                  onclick="login()">
                  Login
                </button>
                <button
                type="button"
                class="btn btn-outline-success"
                onclick="trocaTelaCadastro()"
                >Não tem Conta?</button>
              </div>
            </div>
          </div>
        </div>
      </div>
    </div>
```

Fonte: Autoria própria, 2022

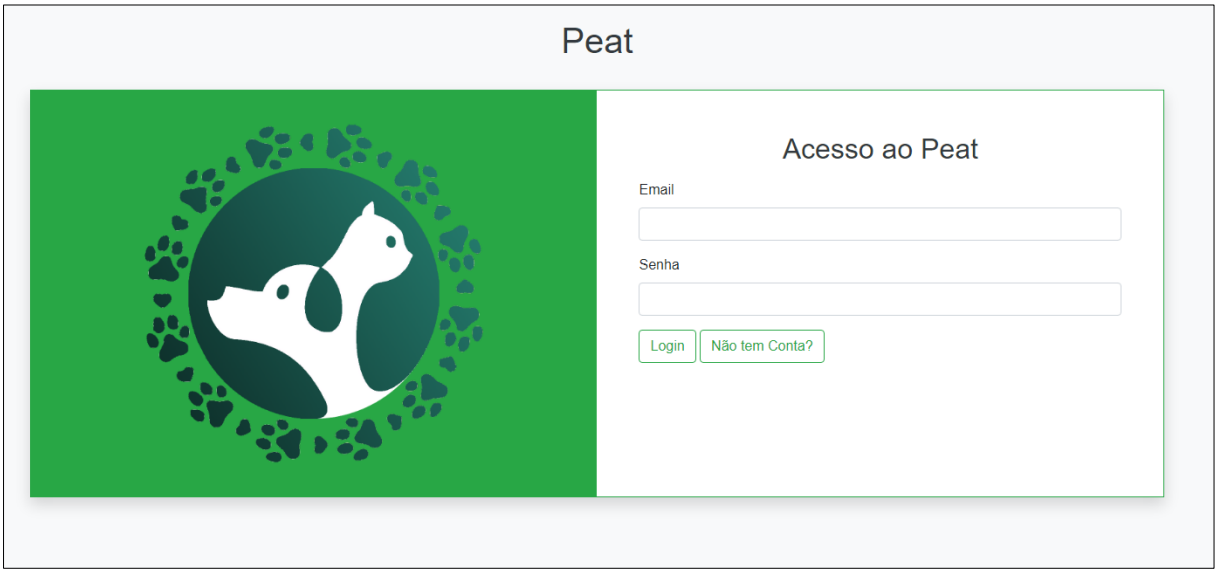
Como podemos observar na Figura 5, o *Bootstrap* adiciona diversos componentes de estilização para o nosso site, todas essas tags estão sendo explicadas a seguir:

- *Container-fluid*: Ele serve para criar um contêiner que armazenara todas as informações da tela.
- *Row*: E utilizado para alinhar os elementos da página.
- *Col*-10: Ele é responsável por definir a largura de uma coluna.

- *Text-center:* Centraliza um texto na página.
- *Text-dark*: Define uma cor para um texto.
- *Border-sucess:* É definido a cor da borda de um *container* para verde.
- *Bg*-white shadow: Adiciona uma borda com sombra a um *container*.
- *Col-6:* Delimita o tamanho da largura de uma coluna.
- *Justify-content-around*: Define o alinhamento de itens na página.
- *Form-control:* Faz uma atualização a campos textuais, adicionando estilos a eles.
- *Btn-outline-sucess*: Adiciona uma borda de cor verde em volta de um botão.

Ao analisar a Figura 5, observamos que temos diversas novas *tags* do *Bootstrap,* entretanto a outra maioria delas continua sendo *HTML*¸ mostrando assim como o essas tecnologias estão conectadas e funcionando em perfeita harmonia. Abaixo na Figura 6, está o resultado do código da Figura 5:

Figura 6 - Resultado Bootstrap



Fonte: Autoria própria, 2022

Podemos observar na Figura 6, que o *Bootstrap* consegue tornar uma página normal *HTML*, em algo totalmente novo, com um visual profissional, mas sem dificuldade para sua aplicação, tornando assim essa tecnologia perfeita para estilização do nosso projeto de conclusão de curso em conjunto ao *CSS.*

2.4. Javascript

Para adicionar funções a um *website* é necessário utilizar o *JavaScript*, ele é uma linguagem de programação *web* não tipada, ela funciona muito bem em conjunto outros estilos de programação orientados a objetos, o *HTML* e *CSS* são exemplos disto. A padrão das linguagens de programação está presente no *JavaScript*, é por ser uma vertente do *Java*, ela trabalha com valores e variáveis.

> Uma variável define um valor, este que pode ser utilizado pelo nome da variável futuramente. Essas que são divisíveis em duas categorias: primitivos e objetos. Os primitivos são os números, *strings*, *booleanos, null* e indefinido, qualquer valor que não estiver neste espectro é considerado um objeto. (FLANAGAN, 2013, p.28)

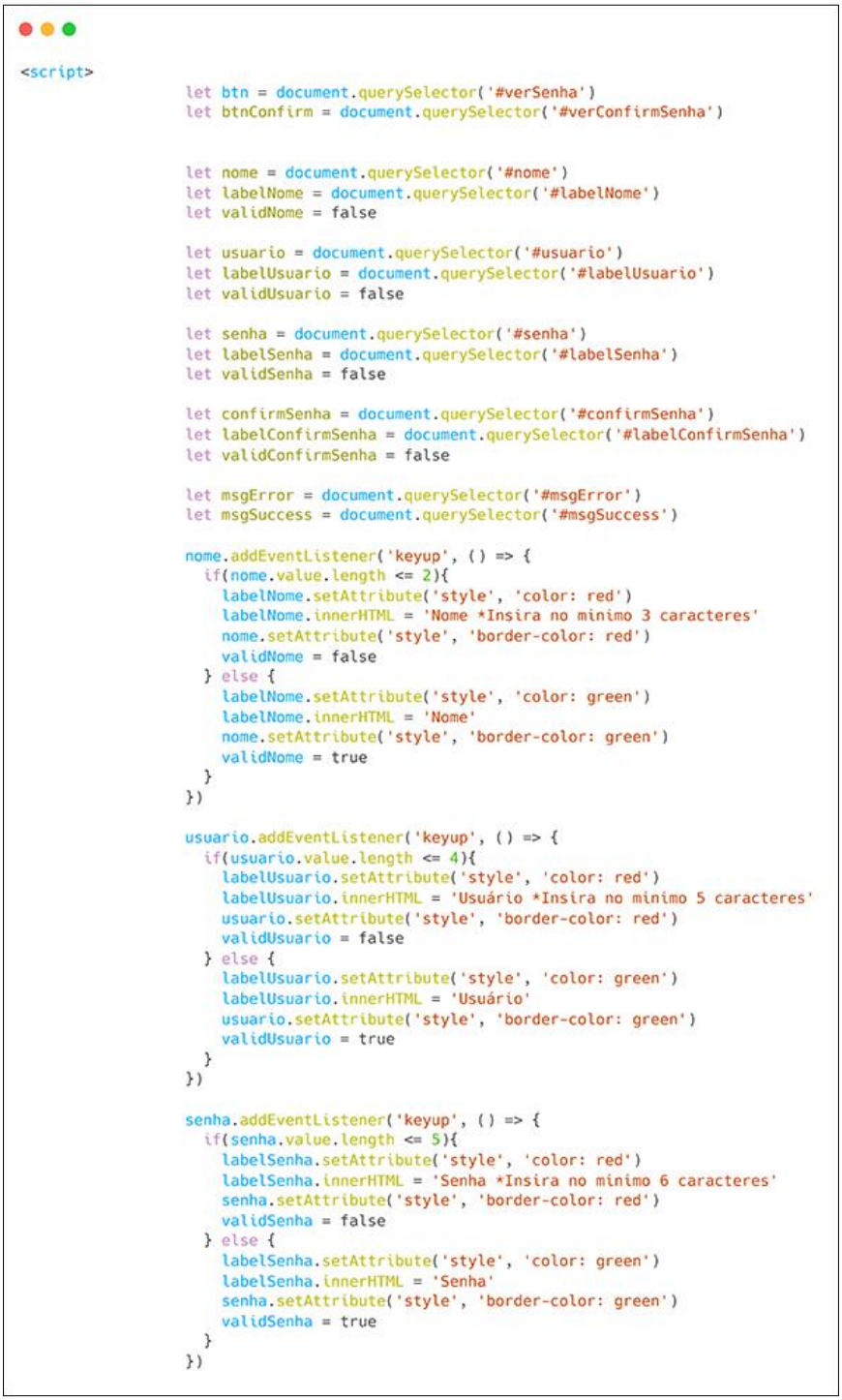
O *JavaScrip*t trabalha com declarações, estás que são linhas de comandos que declaram o que cada variável irá fazer. As principais declarações são: *var*: que como disse Silva (2020, p.31) ela se destina a declarar uma ou mais variáveis, onde estes podem ter vários valores, e eles tem que ser separados por vírgula.

A segunda declaração mais utilizada e o *function,* ele tem o objetivo de definir uma função do *JavaScript*, como por exemplo, executar o cálculo de uma variável*.* Segundo Gillo (2008, p.18) os objetos podem conter diversos valores e atributos diferentes armazenados, estes possuirão funções que podem executar esses valores. Para criação executar um atributo que está em um objeto e necessário utilizar a função *this*.

O *JavaScript* trabalha igual diversas linguagens de programação, ele utiliza operadores aritméticos, como: +, -, /, comparativos, como: ==, !=, >, <, *bit a bit*, como: &, ^, >>, <<, atribuição, como: =, +=, -=, lógicos, como: &&, ! e estruturas de controle, como*: if ... else*, *switch ... case*, *while*, *do ... while*, *for e for ... in*.

Abaixo nas Figuras 7 e 8 está um código de *JavaScript*, nele foram feitas as certificações do formulário, estas que verificam se o usuário digitou a quantidade certa de caracteres que é necessário para se cadastrar:

Figura 7 - Código em JavaScript



```
<script>
                let btn = document.querySelector('#verSenha')
                let btnConfirm = document.querySelector('#verConfirmSenha')


                let nome = document.querySelector('#nome')
                let labelNome = document.querySelector('#labelNome')
                let validNome = false

                let usuario = document.querySelector('#usuario')
                let labelUsuario = document.querySelector('#labelUsuario')
                let validUsuario = false

                let senha = document.querySelector('#senha')
                let labelSenha = document.querySelector('#labelSenha')
                let validSenha = false

                let confirmSenha = document.querySelector('#confirmSenha')
                let labelConfirmSenha = document.querySelector('#labelConfirmSenha')
                let validConfirmSenha = false

                let msgError = document.querySelector('#msgError')
                let msgSuccess = document.querySelector('#msgSuccess')

                nome.addEventListener('keyup', () => {
                  if(nome.value.length <= 2){
                    labelNome.setAttribute('style', 'color: red')
                    labelNome.innerHTML = 'Nome *Insira no minimo 3 caracteres'
                    nome.setAttribute('style', 'border-color: red')
                    validNome = false
                  } else {
                    labelNome.setAttribute('style', 'color: green')
                    labelNome.innerHTML = 'Nome'
                    nome.setAttribute('style', 'border-color: green')
                    validNome = true
                  }
                })

                usuario.addEventListener('keyup', () => {
                  if(usuario.value.length <= 4){
                    labelUsuario.setAttribute('style', 'color: red')
                    labelUsuario.innerHTML = 'Usuário *Insira no minimo 5 caracteres'
                    usuario.setAttribute('style', 'border-color: red')
                    validUsuario = false
                  } else {
                    labelUsuario.setAttribute('style', 'color: green')
                    labelUsuario.innerHTML = 'Usuário'
                    usuario.setAttribute('style', 'border-color: green')
                    validUsuario = true
                  }
                })

                senha.addEventListener('keyup', () => {
                  if(senha.value.length <= 5){
                    labelSenha.setAttribute('style', 'color: red')
                    labelSenha.innerHTML = 'Senha *Insira no minimo 6 caracteres'
                    senha.setAttribute('style', 'border-color: red')
                    validSenha = false
                  } else {
                    labelSenha.setAttribute('style', 'color: green')
                    labelSenha.innerHTML = 'Senha'
                    senha.setAttribute('style', 'border-color: green')
                    validSenha = true
                  }
                })
```

Fonte: Autoria própria, 2022

Abaixo na Figura 8 está a continuação do código em *JavaScript* da Figura 7, nesta parte a validação da senha é realizada e se ela não estiver no padrão requisitado, o

sistema irá recusar a senha. Já na parte final do código à função que realiza o cadastramento do usuário no formulário está sendo construída:

Figura 8 - Continuação código JavaScript

```
        confirmSenha.addEventListener('keyup', () => {
        if(senha.value != confirmSenha.value){
        labelConfirmSenha.setAttribute('style', 'color: red')
        labelConfirmSenha.innerHTML = 'Confirmar Senha *As senhas não conferem'
        confirmSenha.setAttribute('style', 'border-color: red')
        validConfirmSenha = false
        } else {
        labelConfirmSenha.setAttribute('style', 'color: green')
        labelConfirmSenha.innerHTML = 'Confirmar Senha'
        confirmSenha.setAttribute('style', 'border-color: green')
        validConfirmSenha = true
        }})

        function cadastrar(){
        if(validNome && validUsuario && validSenha && validConfirmSenha){
        let listaUser = JSON.parse(localStorage.getItem('listaUser') || '[]')

        listaUser.push(
        {
        nomeCad: nome.value,
        userCad: usuario.value,
        senhaCad: senha.value
        })
        localStorage.setItem('listaUser', JSON.stringify(listaUser))

        msgSuccess.setAttribute('style', 'display: block')
        msgSuccess.innerHTML = '<strong>Cadastrando usuário...</strong>'
        msgError.setAttribute('style', 'display: none')
        msgError.innerHTML = ''

        setTimeout(()=>{
        window.location.href = 'https://cdpn.io/thicode/debug/ZELzYxV/dXAqBaRyvwJk'
        }, 3000)

        } else {
        msgError.setAttribute('style', 'display: block')
      msgError.innerHTML = '<strong>Preencha todos os campos corretamente antes de cadastrar</strong>'
        msgSuccess.innerHTML = ''
        msgSuccess.setAttribute('style', 'display: none')
         }
        }

        btn.addEventListener('click', ()=>{
        let inputSenha = document.querySelector('#senha')

        if(inputSenha.getAttribute('type') == 'password'){
        inputSenha.setAttribute('type', 'text')
        } else {
        inputSenha.setAttribute('type', 'password')
        }})

        btnConfirm.addEventListener('click', ()=>{
        let inputConfirmSenha = document.querySelector('#confirmSenha')

        if(inputConfirmSenha.getAttribute('type') == 'password'){
        inputConfirmSenha.setAttribute('type', 'text')
        } else {
        inputConfirmSenha.setAttribute('type', 'password')
       }})
</script>
```

Fonte: Autoria própria, 2022

A seguir estão explicadas todas as propriedades dos códigos que foram mostrados nas Figuras 7 e 8 em *JavaScript:*

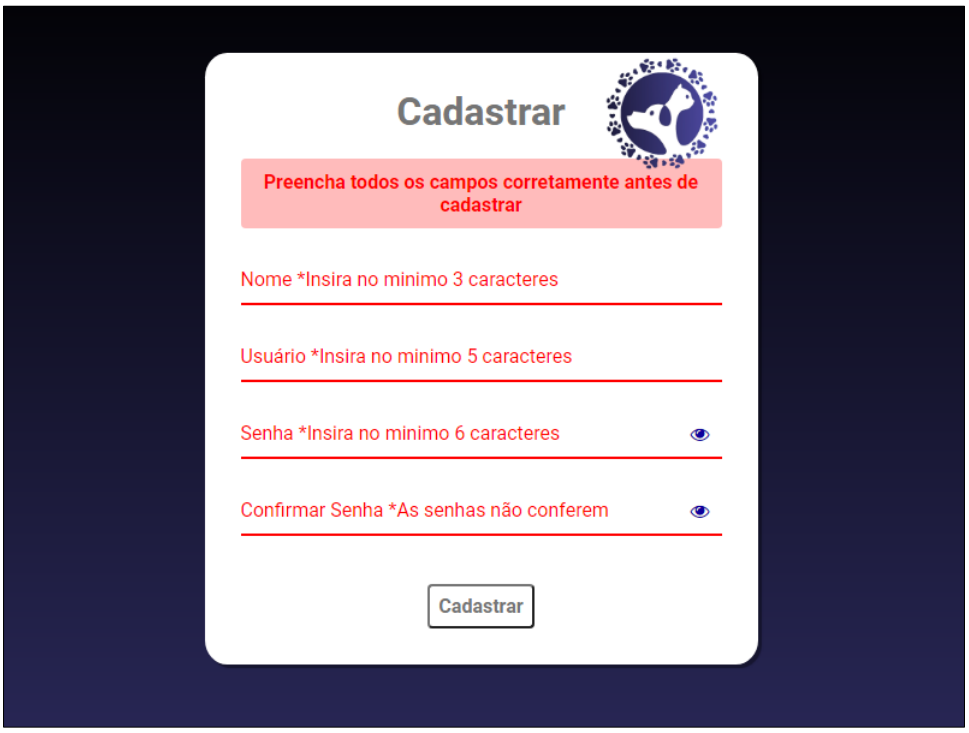
- *Script*: Serve para incluir ou referenciar um script executável.

- *Document.querySelector*: Retorna o primeiro elemento dentro do documento que corresponde ao grupo especificado de seletores.
- *AddEventListener*: Registra uma única espera de evento em um único alvo, O alvo do evento pode ser um único elemento em um documento.
- *If*: É uma estrutura condicional que executa a afirmação, se determinada condição for verdadeira.
- *Else*: É uma estrutura condicional que permite definir fluxos diferentes de execução a partir de determinadas condições definidas.
- *setAttribute*: É utilizado para adicionar um atributo ou modificar o valor de um atributo já existente em um elemento especifico.
- *InnerHTML*: Define ou obtém a sintaxe HTML ou XML descrevendo os elementos descendentes.
- *Values*: Retorna um array em que os elementos são os valores das propriedades enumeradas encontradas no objeto.
- *Length*: Indica quantos caracteres a função espera.
- *Style*: Permite você criar programas que mudam o estilo de um documento de uma forma mais dinâmica.
- *GetAttribute*: Retorna o valor de um argumento especifico do elemento.
- *Location.href*: Redireciona você para uma página interna ou URL externa.

Podemos observar nesse exemplo que o *JavaScript* trouxe grande parte de sua programação do Java, para sua forma de linguagem, o que torna a adaptação aos iniciantes em *Java* muito suave para o *JavaScript.*

Observando a Figura 9 como ficou o resultado das mensagens de erro que foram postas código de *JavaScript,* elas não permitem que o usuário realize o cadastro na página, já que estão faltando campos para serem preenchidos corretamente:

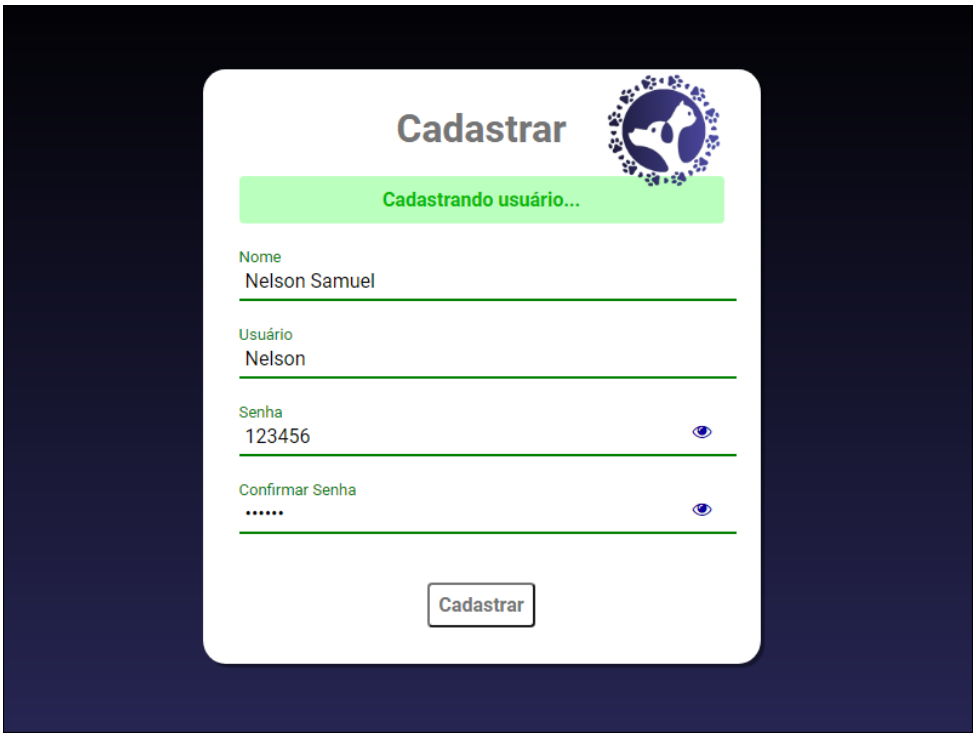
Figura 9 - JavaScript sinais de erro



Fonte: Autoria própria, 2022

Já na Figura 10, demonstrasse o formulário funcionando da forma correta, onde todos os campos de cadastro foram preenchidos corretamente, e assim o usuário foi cadastrado com sucesso:

Figura 10 - JavaScript sinais de aceitação



Fonte: Autoria própria, 2022

Ao observar todas as funções incríveis que o *JavaScript* proporciona, nós o adicionamos ao nosso trabalho de conclusão de curso, ele será o principal responsável por criar todas as funcionalidades do aplicativo em conjunto ao *React Native*.

2.5. React Native

Como foi dito por Escudelario (2020, p.13) *React Native* é o *framework* com as ferramentas de criação de um aplicativo *mobile* para *Android* e *iOS* mais atual e atualizada do mercado de desenvolvimento *front-end*. O *React Native* é a parte focada em criação *mobile* da biblioteca *React*, este que se utiliza de *JavaScript* para criar uma das melhores ferramentas para criação de *apps mobile* do mercado.

Umas das principais bibliotecas para importação de funções do *JavaScript* se chama *DOM*, contudo essa biblioteca e muito problemática, podendo acarretar instabilidades constantes nas aplicações, a solução que que foi criada pela *React Native* para que fosse possível a utilização desta biblioteca, foi adotar uma variante dela, chamada *DOM Virtual*, esta que e ao contrário de sua versão padrão, permite que as alterações realizadas no código de uma aplicação feita em *React Native*, fiquem desabilitadas, até que todas as modificações sejam feitas, assim o programa será executado com todas as funções ativadas de uma só vez, evitando possíveis erros e travamentos durante a criação do código.

> O *React Native* usa uma linguagem e extensão do *JavaScript* que se chama *JSX*, este que vai criar uma série de classes customizadas, que irão proporcionar os componentes da interface, que irão se adaptar dependendo de qual plataforma o usuário esteja utilizando. (MCCOMB, 2015)

De acordo com as palavras de Stefanov (2016, p.24) o *JSX* é um substituto para as funções, ele faz com que as várias chamadas do método *return()* não sejam mais necessárias, tornando o código mais limpo e fluido de se entender. O *JSX* não é obrigatório para programas em *React Native*, mas ele muito prático, e pode ser utilizado no lugar das funções padrões.

Entretanto o *JSX* tem suas falhas, ele é uma tecnologia que não é interpretada pelos navegadores e plataformas, ou seja, para que seja possível implementar ele funcionalmente, é necessário utilizar a *babel.js*, que irá converter o código puro de *JavaScript* para um elemento interpretável pela aplicação utilizada.

Abaixo está a figura 11, que representa como o *React Native* se baseia em telas e caixas de texto editáveis:

Figura 11 - Código React Native

```
import {TextInput,TouchableOpacity,StyleSheet} from 'react-native';

export default function App() {
    <view style={container}>
      <TextInput>
        placeholder="Email"
        autoCorrect={false}
        onChangeText={()=> {}}
      </TextInput>

      <TextInput>
        placeholder="Senha"
        autoCorrect={false}
        onChangeText={()=> {}}
      </TextInput>

      <TouchableOpacity style={styles.btnSubmit}>
        <text style={styles.submitText}>Acessar</text>
      </TouchableOpacity>

      <TouchableOpacity style={styles.btnRegister}>
        <text style={styles.registerText}>Criar Conta Gratuita</text>
      </TouchableOpacity>
    </view>
}
```

```
background:{
    flex:1,
    alignItems: 'center',
    justifyContent: 'center',
    backgroundColor: '#191919'
  },
  containerLogo: {
    flex: 1,
    justifyContent: 'center',
    backgroundColor: 'red'
  },
  container: {
    flex:1,
    alignItems: 'center',
    justifyContent: 'center',
    width: '90%'
  },
  input: {
    backgroundColor: '#FFF',
    width: '90%',
    marginBottom:15,
    color: '#222',
    fontSize: 17,
    borderRadius: 7,
    padding: 10
  },
  btnSubmit: {
    backgroundColor: '#35AAFF',
    width: '90%',
    height: 45,
    alignItems: 'center',
    justifyContent: 'center',
    borderRadius: 7,
  },
  submitText: {
    color: '#FFF',
    fontSize:18
  },
  btnRegister: {
    marginTop: 10
    },
    registerText:{
      color: '#FFF'
    }
});
```

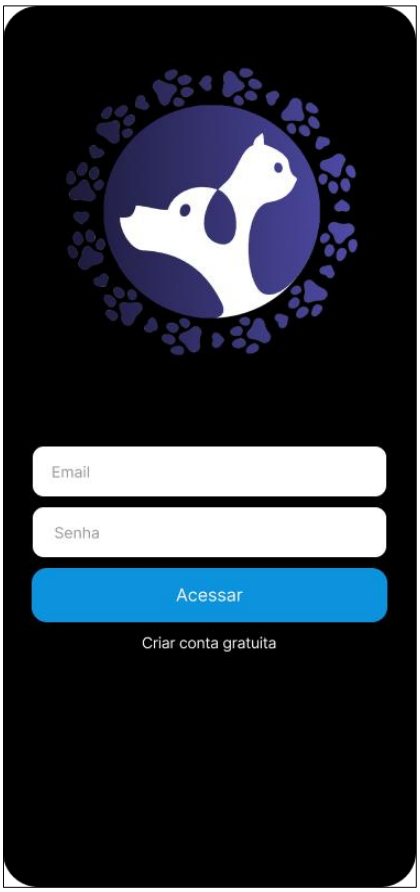
Fonte: Autoria própria, 2022

Os elementos de código da Figura 11 funcionam dá seguinte forma, o *import* importa bibliotecas do *React Native* para a sua aplicação, estas que vão adicionar novas funções ao código. O *export* irá importar todo o conteúdo que o usuário deixar na pasta que a aplicação está localizada. A *view style* irá criar um elemento de layout para a aplicação, onde as funções do *app* ficaram localizadas. Os elementos *textInput* irão criar as caixas de informação, onde o usuário que vá utilizar o aplicativo devera escrever seu *email* e senha para se cadastrar no *app*. O *placeholder* vai configurar onde e como ficarão as animações na aplicação. Já o *autoCorrect* manipula onde ficara posicionado um campo de texto. O *onChangeText* tem a função de registrar o texto digitado pelo usuário. O *touchableOpacity* é responsável por tornar uma *view* responsiva a toques, fazendo com que ela seja diminuída ou aumentada dependo da

escolha do programador. E completando, temos o *text style*, que dá a um texto um novo estilo.

Abaixo na Figura 12 está o resultado desse aplicativo em *React Native*:

Figura 12 - Tela do Aplicativo em React Native



Autor: Autoria própria, 2022

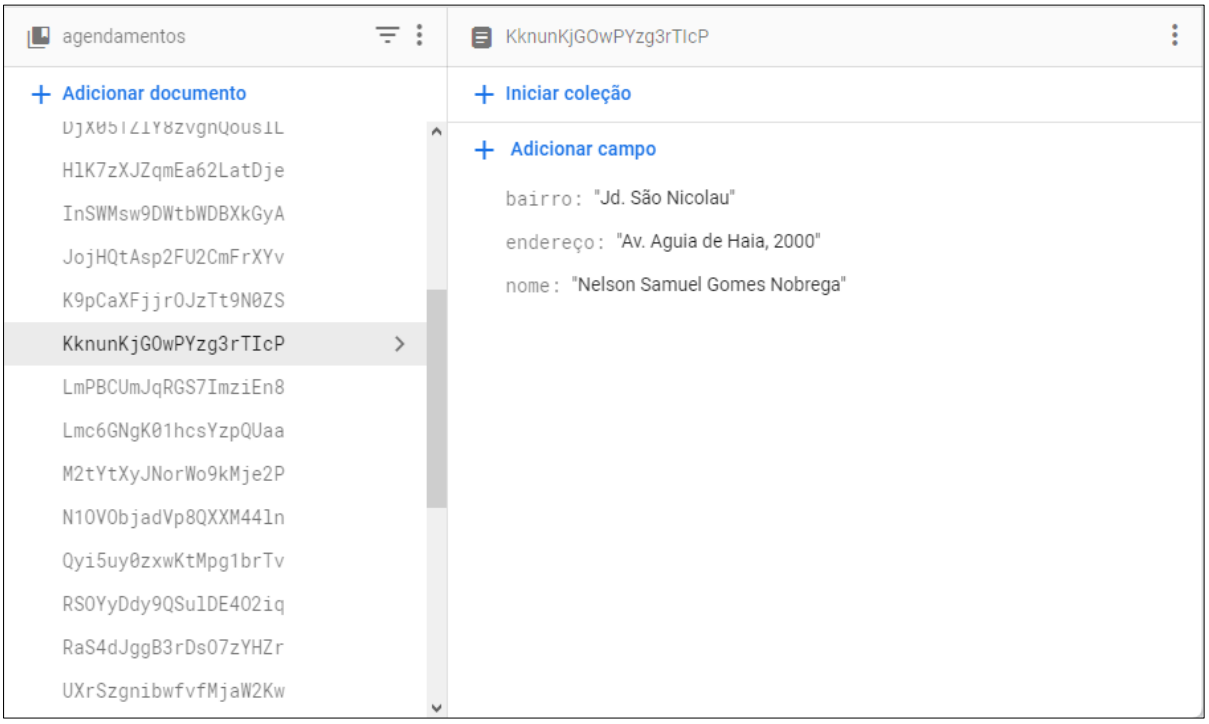
Com isso concluímos que os códigos da Figura 12, que o *React Native* faz uso de outras linguagens como *HTML, CSS* e *JavaScript* em sua programação. Isso torna essa tecnologia muito versátil e inclusiva, já que essas linguagens são base para os programadores, fazendo com que eles consigam iniciar muito facilmente no *React Native*. Todas essas qualidades fizeram a escolha do *React Native* a melhor tecnologia possível para criação do *back-end* do nosso aplicativo.

2.6. Banco de Dados Firebase

O Firebase é um sistema de banco de dados NoSQL, que significa não relacional. E como foi dito pela Microsoft (2022) diferentemente dos bancos de dados comuns, os não relacionais não utilizam o esquema de linhas e colunas nas tabelas do banco.

Essa não utilização de tabelas e colunas faz com que o Firebase seja o banco de dados perfeito para se conectar com aplicativos, pois os usuais sistemas que adotam SQL (Standart Query Language) não funcionam com aplicativos para celular. A Figura 13 exemplifica como ficam ordenadas as informações no banco de dados Firebase:

Figura 13 - Banco de Dados Firebase



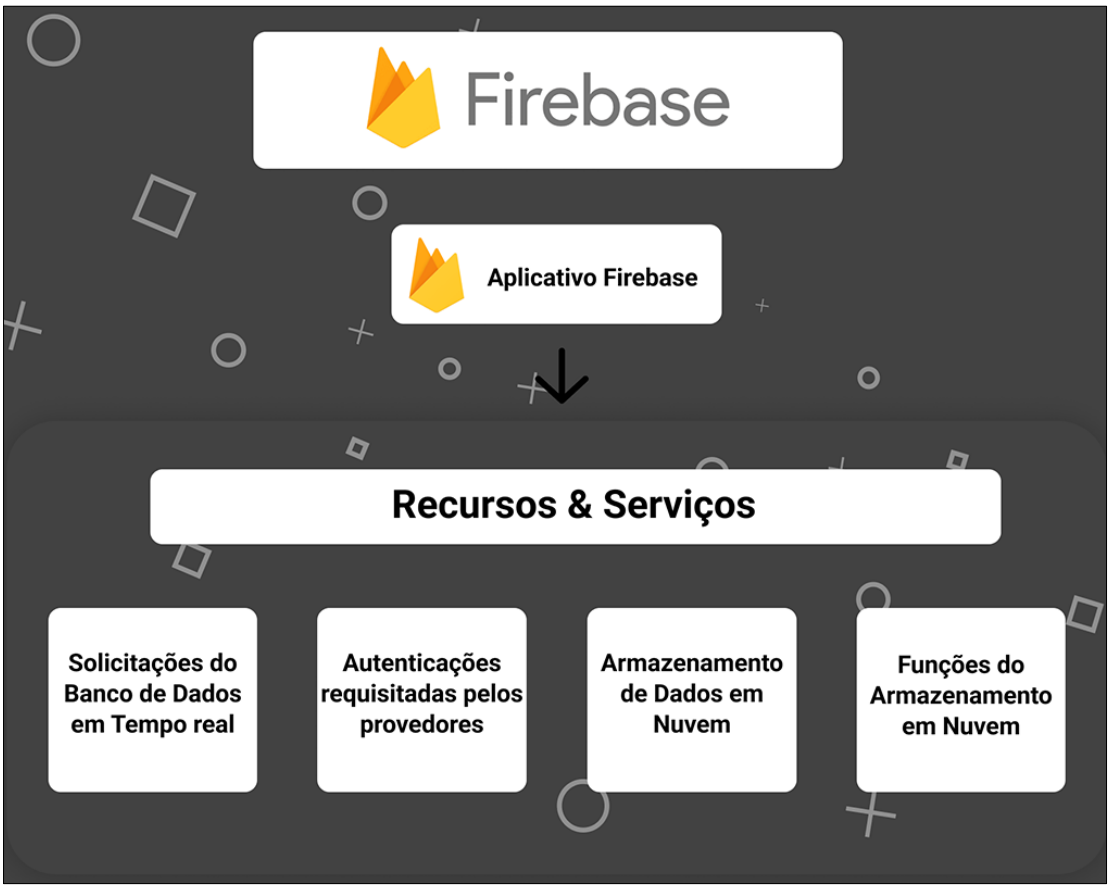
Fonte: Autoria própria, 2022

Ao observar atentamente a Figura 13, podemos entender que o banco não possui nenhum parâmetro para salvar as informações, elas só são postas em ordem aleatória no banco, mesmo assim o sistema consegue identificar e utilizar cada informação quando e requisitada.

O projeto Firebase e baseado em um contêiner, onde todas as informações serão armazenadas, sendo assim, versões diferentes de um mesmo aplicativo podem coexistir, e todos esses apps terão o mesmo poder de acesso as informações e possuem compartilhamento delas entre si.

Na Figura 14, é explicado quais são os recursos e serviços que o *Firebase* utiliza para administrar o funcionamento dos seus bancos de dados:

Figura 14 - Recursos e Serviços Firebase



Fonte: Autoria própria, 2022

2.6.1. Identificadores

Ao criar um banco de dado em Firebase, alguns identificadores serão direcionados a este projeto, o primeiro deles será o nome de projeto, que é o mais básico. Em seguida temos o número do projeto, que segundo o Firebase (2022) este código identificador e único para cada projeto, logo utilize-se dele para realizar integrações a novas *APIs* dentro dos bancos de dados *Firebase*.

E o último entre eles é o ID do projeto, ele se parece com o número do projeto, onde o próprio Firebase fornece o código, entretanto no ID, e possível futuramente a alteração do código pelo usuário, porém se recursos do Firebase forem adicionados ao banco, este ID não poderá mais ser modificado.

Essa adaptabilidade e essencial para empresas que precisam lidar com alto nível de dados, os aplicativos de redes sociais são um exemplo perfeito disto. Segundo Machado (2021, p.35) os dados do banco são organizados em coleções de documentos, onde conforme o aumento exponencial de dados chega, o sistema consegue notificar este aumento para a empresa.

Além de que todo esse sistema e controlado pela *Cloud Firestore*, que administra e gerencia a evolução do banco de dados, possibilitando assim com que a empresa que utilize o seu sistema, não precise se preocupar com gerenciamento da infraestrutura dos seus bancos *Firebase*, já que todas as questões técnicas ficaram por responsabilidade da *Google*, que é a empresa dona do *Firebase*.

Toda essa adaptabilidade e conexão, fazem com que esse seja o tipo de banco de dados mais adequado para a necessidade do nosso *app*, já que sua velocidade de pesquisa e capacidade de armazenamento são de altíssimo nível, ele será capaz de lidar com a alta demanda que os usuários exigirão do banco e como não precisaremos nós preocupar com as questões técnicas, poderemos focar em melhorar cada vez mais o nosso aplicativo.

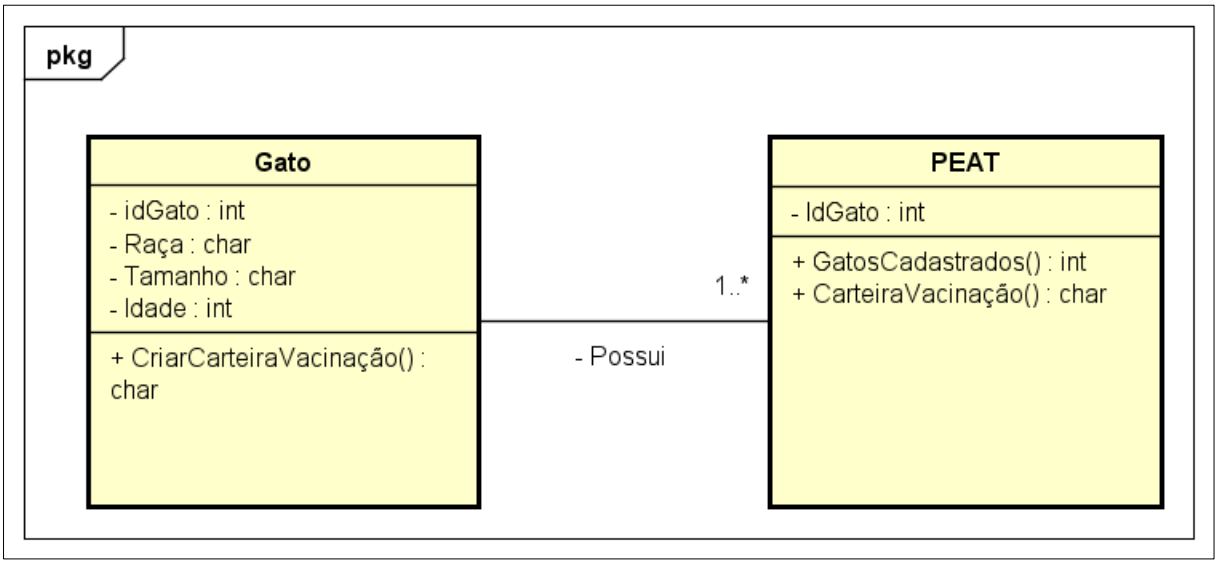
2.7. UML

A *Unified Modeling Language* (UML) é uma linguagem unificada de modelagem orientada a objetos, que serve para criação de gráficos de especificação, construção e documentação de softwares. Ou seja, a *UML* é utilizada para criação de diagramas para mapear as funções é classes de um software, assim os programadores podem planejar as limitações e o objetivos da aplicação.

A estrutura da *UML* é baseada em nos seguintes blocos: itens, relacionamentos e diagramas. Conforme foi dito por Booch (2006, p.18) “Os itens são as abstrações identificadas como cidadãos de primeira classe em um modelo; os relacionamentos reúnem esses itens; os diagramas agrupam coleções interessantes de itens.” Esses blocos ditam como devem ser construída as funções dos diagramas, entretanto, para criação de uma *UML* deve seguir um padrão em suas estruturas, que conforme foi apontado pela IBM (2021) os diagramas *UML* necessitam apresentar os aspectos existentes em uma aplicação, estes sendo relacionamentos, comportamento, estruturas e funcionalidades.

A seguir na Figura 15, podemos encontrar todos os aspectos que formam um diagrama de classe *UML*:

Figura 15 - Diagrama de Classe



Fonte: Autoria própria, 2022

Com este diagrama da Figura 15, é possível encontrar os itens Gato e PEAT, os aspectos de relacionamento, representados pela seta entre os dois objetos, o comportamento entra no que é passado de um objeto para o outro, este que é o IdGato: Int, em seguida temos as estruturas, que são os componentes de cada objeto e por último as funcionalidades, estas que são o cadastro de gatos, a listagem desses animais e a criação de uma ficha médica para cada gato.

Contudo, a *UML* não se limita em apenas um tipo de diagrama, ao total, ele conta com 13 modelos diferentes de diagrama, sendo destes 6 modelos que modificam as estruturas *UML* e os outros 7 diagramas trabalham com modificações comportamentais.

Entre todos os modelos de diagramas *UML*, o que mais se destaca entre eles se chamada diagrama de classes, segundo Lucidchart (2022) diagramas de classe fazem parte da classificação de estruturas *UML*, pois eles descrevem quais são os detalhes que devem estar no sistema que o diagrama está sendo modelado. A Figura 15 é um exemplo de diagrama de classe.

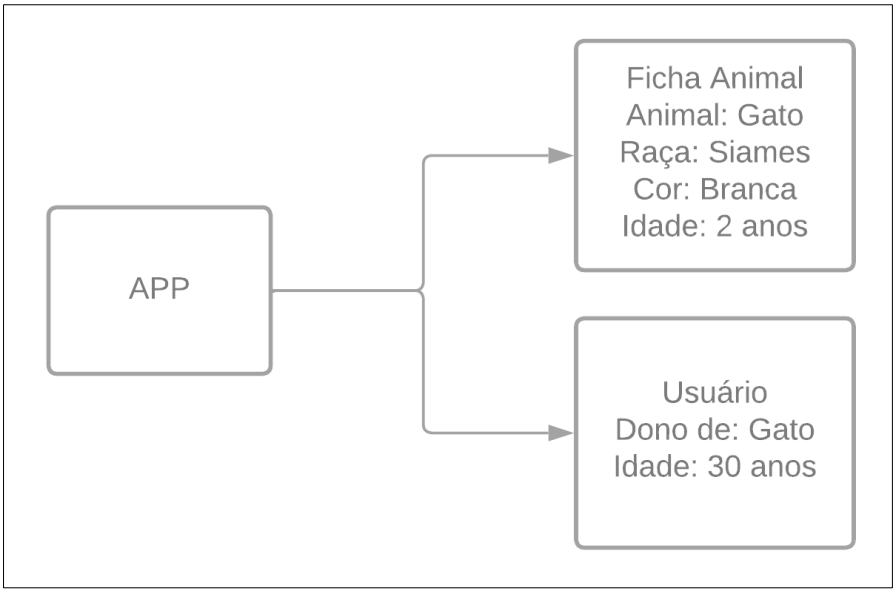
Os diagramas de componente, são essenciais para implementações de componentes em sistemas, possibilitando assim o crescimento desses softwares, a partir de um projeto pequeno.

O diagrama de estrutura composta é bem básico, ele serve para mostrar as estruturas de uma classe.

O diagrama de implementação trabalha descrevendo quais são as implementações físicas que um projeto necessita, ou seja, utilizando-se de nós de processamento, esse diagrama permite implementações físicas em uma arquitetura de software.

Já os diagramas de objetos, buscam apresentar um conjunto de objetos e o relacionamento entre eles. Eles buscam abranger a perspectiva dos diagramas de classe, introduzindo mais realismo e comparações a protótipos. Abaixo segue a Figura 16, que exemplifica um diagrama de objeto:

Figura 16 - Diagrama de Objetos



Fonte: Autoria própria, 2022

E por fim o diagrama de pacotes, este que é o último diagrama que modifica as estruturas do *UML*. Ele se utilizando de pacotes, faz a apresentação de diversas camadas de um sistema.

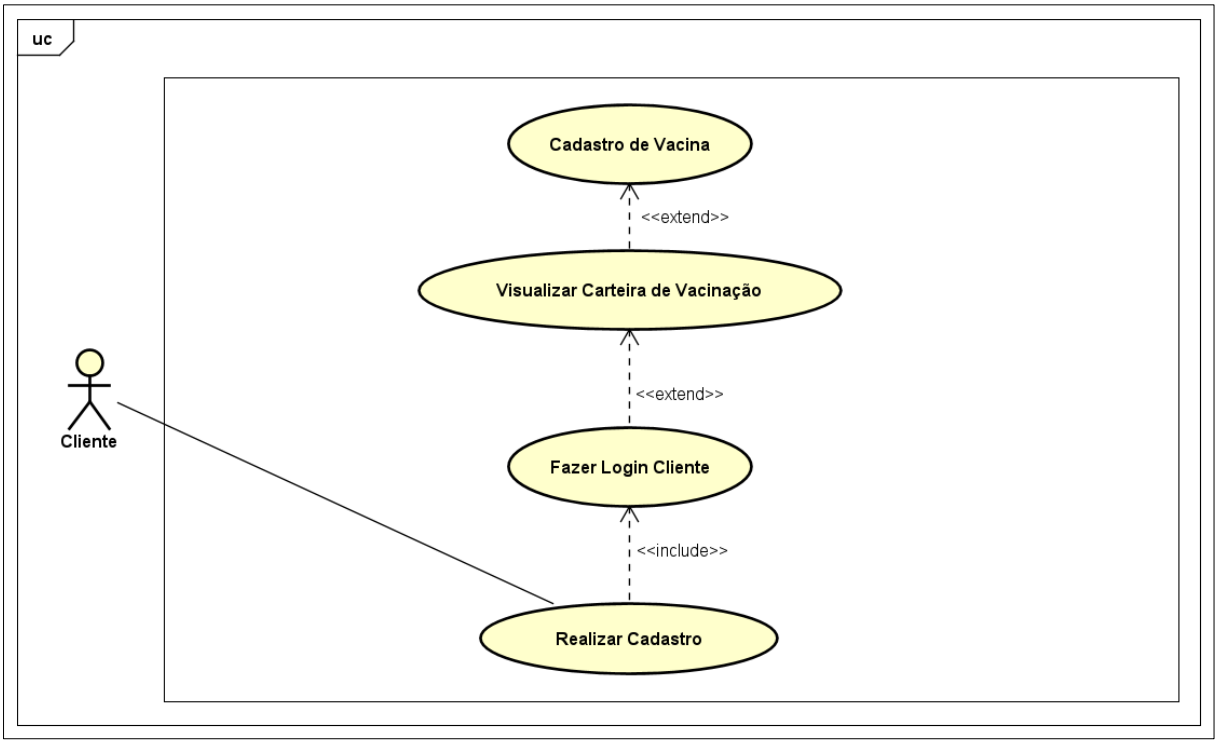
Em seguida temos os diagramas *UML* comportamentais, o primeiro entre eles é o diagrama de atividade, ele é muito utilizado para criar um fluxo de controle das atividades realizadas para construção de um software.

Os diagramas da visão geral da interação controlam a ordem de funcionamento de todos os diagramas *UML* que se baseiam em interações. Entre eles temos o diagrama de comunicação serve para a troca de mensagens entre objetos, proporcionando assim uma interação entre eles, focando sempre na estrutura de dados que as mensagens trafegam. Já o diagrama de sequência e muito parecido com o diagrama de comunicação, entretanto ele foca na sequência de tempo entre as mensagens.

Então temos os diagramas que cuidam do tempo e das atividades dos diagramas *UML*, começando pelo diagrama de máquinas de estados, que segundo Booch (2006, p.27) “Os diagramas de gráfico de estados exibem uma máquina de estados, formada por transições, eventos e atividades.” Em seguida temos o diagrama de tempo, este que é responsável por monitorar o comportamento de um objeto em um determinado período, observando as alterações que foram geradas ao longo do tempo.

E por último temos o diagrama de casos de uso, este que é de extrema importância, pois com ele é possível criar relacionamentos entre controladores e funcionalidades, proporcionando assim uma forma de observar onde cada função do sistema está empregada, e como ela se comporta ao ser utilizada. A Figura 17 abaixo, exemplifica como funciona os diagramas de caso de uso, utilizando como exemplo um dono de *pet,* utilizando um app para cadastro de vacinas em uma carteira de vacinação digital:

Figura 17 - Diagrama de Casos de Uso



Fonte: Autoria própria, 2022

Todos esses modelos diferentes de diagrama, proporcionam uma gama de modelos diferentes, para que possamos assim apresentar e exemplificar como cada parte do nosso software irá funcionar, promovendo mais entendimento sobre o projeto ao leitor que seja leigo sobre a área de construção de aplicativos.

3. DESENVOLVIMENTO

Neste capítulo, iremos apresentar e explicar os digramas *UML* e as telas do nosso projeto de conclusão de curso.

3.1. UML

A seguir iremos apresentar todos os diagramas *UML*¸ que em conjunto irão explicar detalhadamente como o nosso projeto funciona.

3.1.1. Levantamento de Análise e Requisitos

Para iniciar a criação do nosso projeto, primeiramente fizemos um levantamento de analises e requisitos que o nosso sistema deveria ter. Nosso primeiro passo foi visitar um centro veterinário, para consultar um profissional que atua nesta área, para descobrir quais eram as funções mais essenciais para um aplicativo que envolvesse animais.

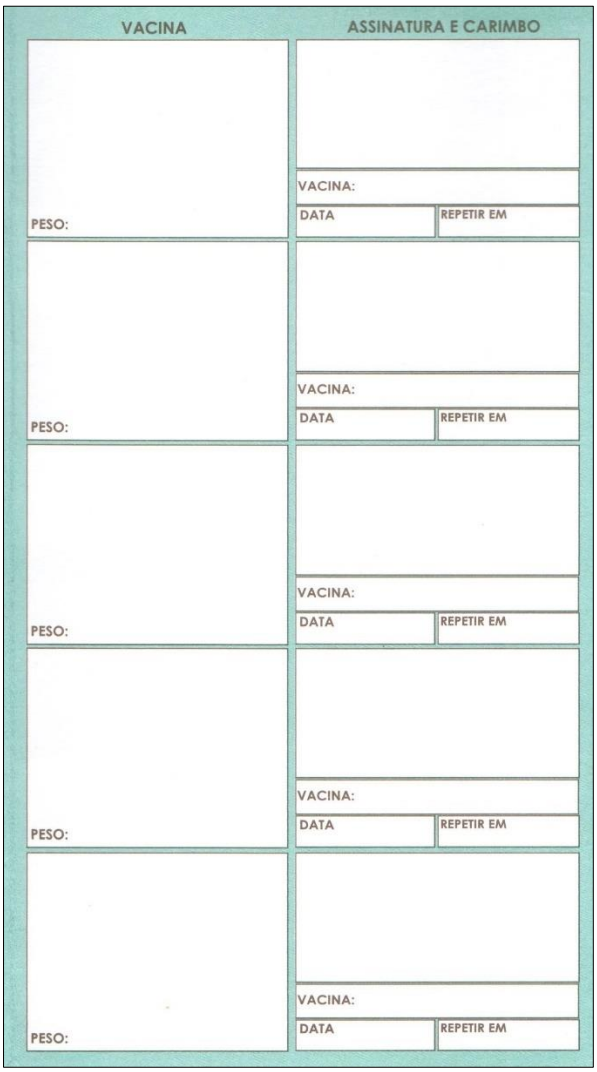
A médica veterinária Juliana Justino Mioshi, detentora do número de inscrição no Conselho Regional de Medicina Veterinária do Estado de São Paulo (CRMV-SP) 50161, nós ajudou a escolher quais funções seriam essenciais para o aplicativo. Entre os pontos discutidos, os que mais fizeram sentido para serem aplicados ao sistema foram a implementação de uma carteira de vacinação digital, estas que são dificilmente encontras no mercado de aplicativos atuais e um sistema de agendamento de banho e tosa, já que são serviços de rotina para petshops e veterinários, a criação de um *app* para que concentre esses agendamentos, facilitaria o , assim facilitando a rotina desses estabelecimentos, agrupando todos os agendamentos em apenas um lugar. Nas seguintes Figuras 18 e 19, possuem a carteira de vacinação fornecida pelo centro veterinário Mioshi, que serviu de inspiração para criação da carteira de vacinação digital.

Figura 18 - Carteira de Vacinação

Fonte: Autoria própria, 2022

Na Figura 18, podemos observar todas as informações que devem ser requisitadas para criação de uma carteira de vacinação, como nome, espécie do animal, raça, sexo e data de nascimento. Abaixo na Figura 19, está a parte onde as vacinas são marcadas:

Figura 19 - Cadastro de Vacinas



Fonte: Autoria própria, 2022

A partir das Figuras 18 e 19, nós conseguimos estruturar a carteira de vacinação digital do aplicativo, nós levando ao próximo passo do projeto, levantar os requisitos funcionais, requisitos funcionais e regras de negócio.

Tabela 1 - Requisitos Funcionais

Código	Descrição dos requisitos funcional
RF1	O sistema deve cadastrar clientes.
RF2	O sistema deve cadastrar veterinários.
RF3	O sistema deve logar clientes.
RF4	O sistema deve logar veterinários.
RF5	O cliente deve poder cadastrar vacinas na carteira de vacinação.
RF6	O cliente deve poder visualizar a carteira de vacinação.
RF7	O cliente deve poder agendar serviços.
RF8	O sistema só exibi carteira de vacinação no aplicativo.
RF9	O sistema só exibi os agendamentos de serviços no site.
RF10	O sistema deve permitir o agendamento de diversos serviços.

Fonte: Autoria própria, 2022

Os requisitos funcionais da Tabela 1, explicam quais serão as funções que o nosso sistema pode executar. Abaixo na Tabela 2, estão os requisitos não funcionais:

Tabela 2 - Requisitos Não Funcionais

Código	Descrição dos requisitos não funcional
RNF1	O design do aplicativo e minimalista para facilitar a compreensão do usuário.
RNF2	O design do site e simples para facilitar a utilização dele pelos veterinários.
RNF3	O aplicativo possui responsividade, para que ele seja capaz de se adaptar a qualquer tamanho de celular.
RNF4	O site é equipado com responsividade, para que ele se adapte a qualquer tamanho de monitor.

Fonte: Autoria própria, 2022

Na Tabela 2, está os requisitos não funcionais, eles mostram as funções que o sistema deve executar. A seguir está a Tabela 3, exibindo os requisitos não funcionais:

Tabela 3 - Regras de Negócio

Código	Descrição dos requisitos não funcionais
RN1	O acesso ao site e exclusivo para veterinários.
RN2	Os veterinários e os clientes podem cadastrar vacina no sistema.
RN3	O veterinário pode escolher se ele aceita ou recusa o agendamento.
RN4	Os agendamentos só podem ser visualizados pelos veterinários.
RN5	O aplicativo é para uso exclusivo de clientes.

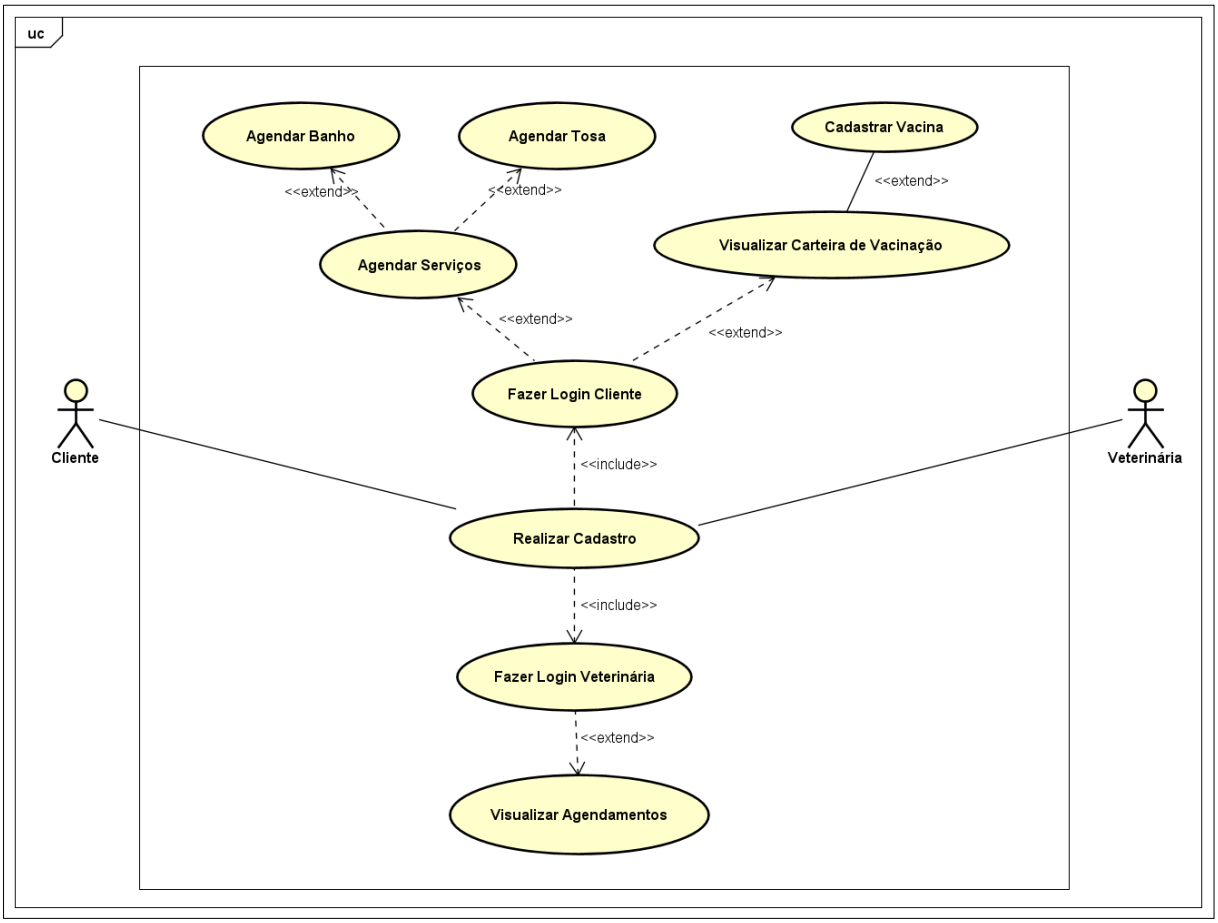
Fonte: Autoria própria, 2022

Por último temos a Tabela 3, aqui temos as regras de negócio do sistema, elas são responsáveis por declarar a forma que o negócio deve seguir. Todo esse levantamento de requisito foi essencial para guiar a produção do nosso projeto de conclusão de curso.

3.1.2. Diagrama de Casos de Uso

Para exemplificar e facilitar o entendimento do projeto, criamos o diagrama de casos de uso, que irá generalizar todas as ações que os usuários podem realizar no nosso projeto. Na Figura 20, está o nosso diagrama de casos de uso:

Figura 20 - Diagrama de Casos de Uso



Fonte: Autoria própria, 2022

Conseguimos observar no diagrama de casos de uso da Figura 20, que o nosso sistema fornece um cadastro e um login específico para cada tipo de usuário, sendo ele cliente do aplicativo, ou um veterinário.

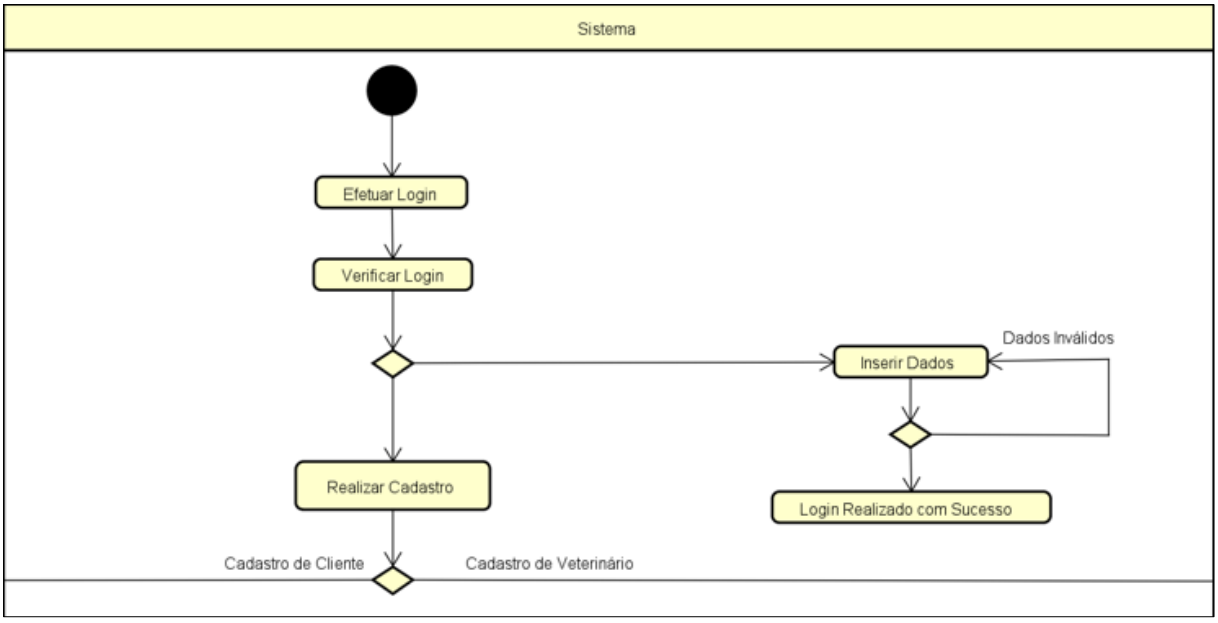
Analisando a Figura 20, podemos observar que todos os balões constituem ações que podem ser realizadas no sistema, as linhas pontilhadas que possuem escrito <<*include*>>, apontam ações que obrigatoriamente precisam ser realizadas, já as linhas que possuem <<*extend*>> mostram que ações que podem ser executas, em decorrência de outra ação.

As ações os clientes podem realizar no sistema são cadastro, este que obrigatoriamente leva o usuário a efetuar o login, em seguida ele tem a escolhe de agendar serviços, como banho e tosa, e visualizar a carteira de vacinação do pet. Já o veterinário e capaz de efetuar cadastro, assim o levando a realizar o login, após isso ele pode visualizar os agendamentos de serviços que foram feitos para o estabelecimento dele.

3.1.3. Diagrama de Atividade

O diagrama de atividade realiza uma expansão do diagrama de casos de uso, mostrando todos os passos que os usuários fazem ao entrar no sistema, e as escolhas que ele possuem. Nesse diagrama de atividade, as bolas pretas representam o início da utilização do sistema, os balões mostram as ações que podem ser realizadas, as setas mostram a ordem que as ações podem ser executadas, os losangos são responsáveis por representar as decisões que os usuários podem escolher na utilização do sistema, por último os pontos pretos com círculo em volta, mostram o final da utilização do sistema. Abaixo na Figura 21, podemos observar o começo do nosso diagrama de atividades:

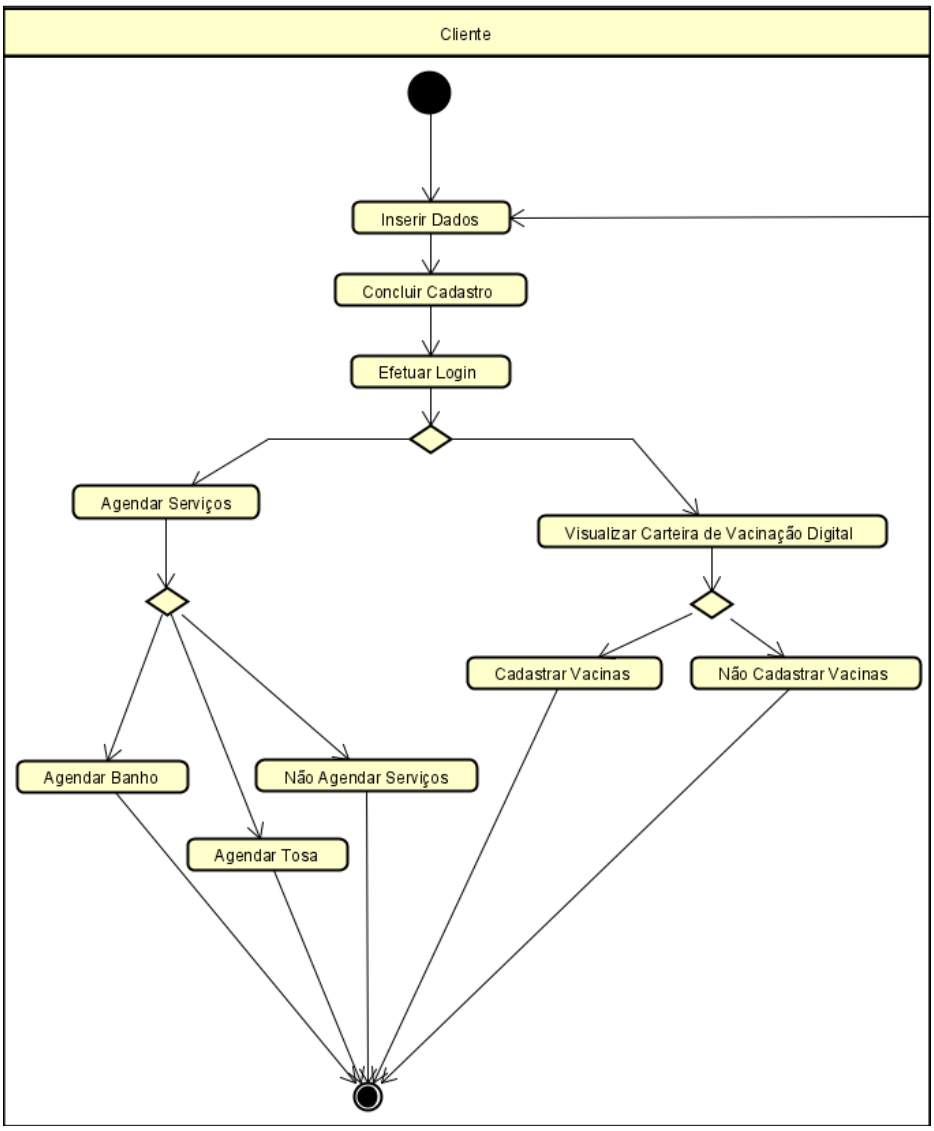
Figura 21 - Diagrama de Atividade - Sistema



Fonte: Autoria própria, 2022

Esse diagrama da Figura 21 é divido em 3 partes, a primeira sendo a do sistema, que mostra as funções que ele realiza, sendo elas de realizar o login de um usuário, este que se digitar os seus dados incorretamente terá de refazer o login, e a função de cadastro, que e separada para clientes e veterinários. A segunda parte do diagrama de atividade está na Figura 22 abaixo:

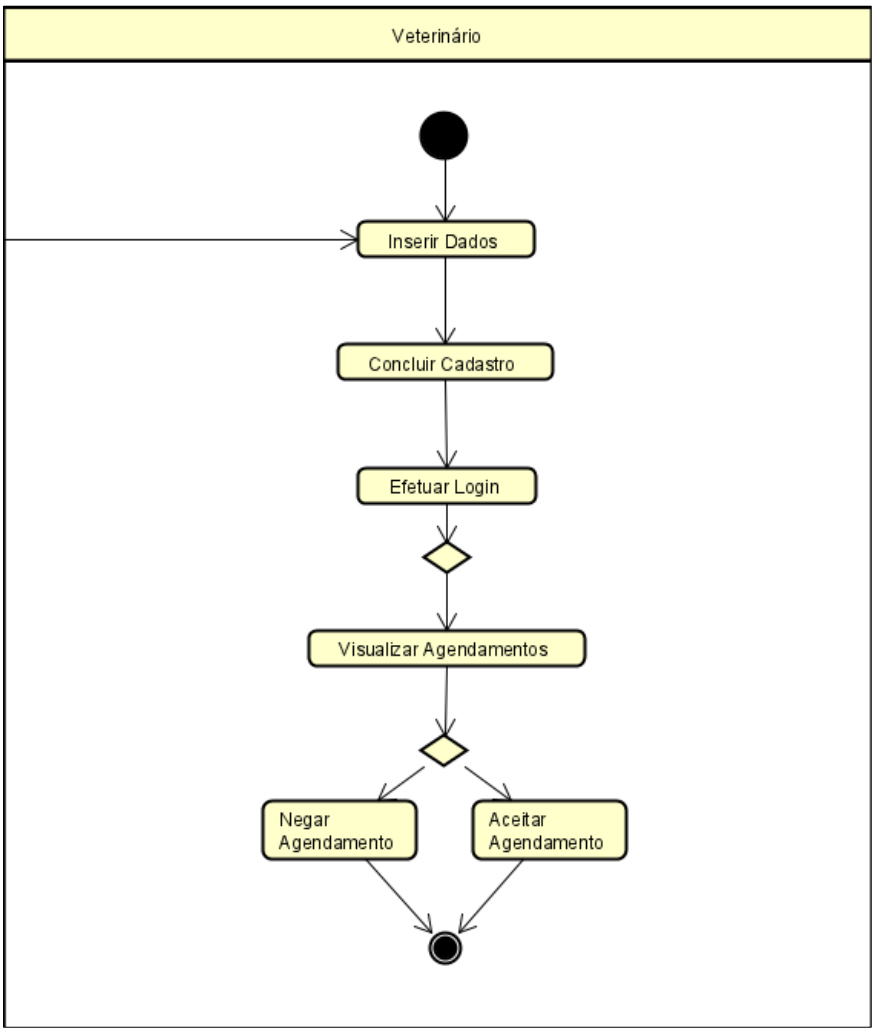
Figura 22 - Diagrama de Atividade - Cliente



Fonte: Autoria própria, 2022

Como podemos ver na Figura 22, as atividades que os clientes podem efetuar o cadastro, que o leva a efetuar o login, após efetuar o login, o cliente tem a escolha de agendar um serviço ou visualizar a carteira de vacinação digital, ao escolher agendar um serviço, ele pode optar por agendar banho, tosa, ou não agendar nada, então acaba a atividade. A última parte do diagrama está na Figura 23, onde as atividades do veterinário são explicadas:

Figura 23 - Diagrama de Atividade - Veterinário



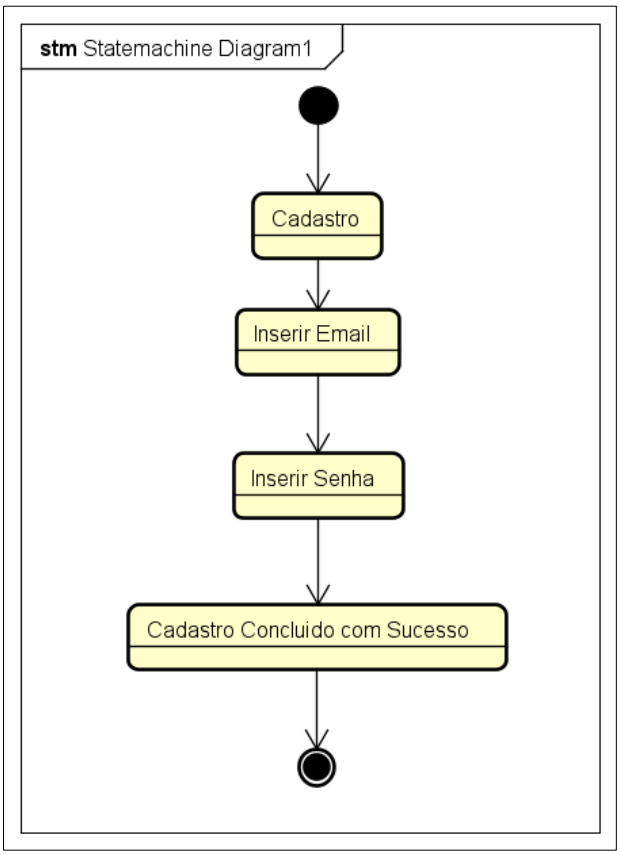
Fonte: Autoria própria, 2022

Neste diagrama da Figura 23, o veterinário pode realizar as seguintes ações, realizar cadastro, em seguida efetuar o login, após isso o veterinário pode visualizar os agendamentos que foram feitos pelos clientes, esses agendamentos que ele pode escolher aceitar ou negar.

3.1.4. Diagrama de Estado

O diagrama de estado expande o diagrama de atividade, mostrando todos as possibilidades e casos que podem acontecer no sistema. Abaixo na Figura 24, está parte que explica como o cadastro funciona:

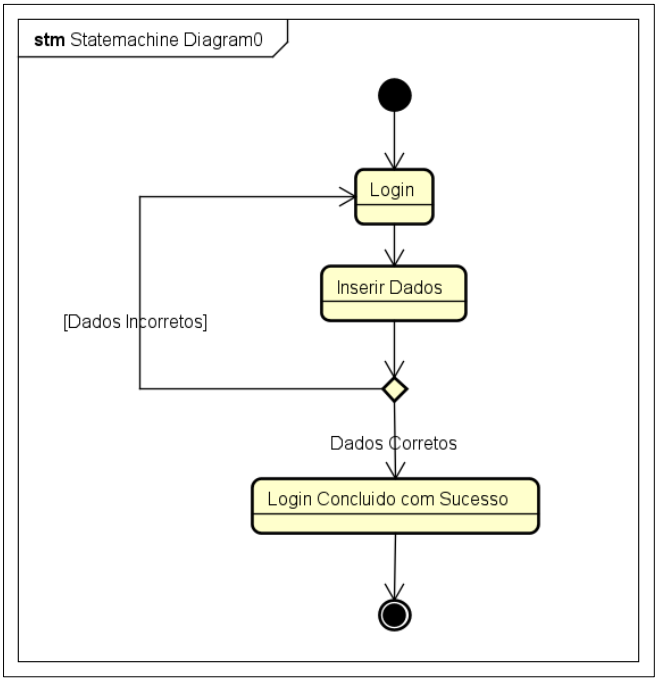
Figura 24 - Diagrama de Estado Cadastro



Fonte: Autoria própria, 2022

Em seguida na Figura 25, temos o diagrama de estado explicando como o *login* do projeto funciona:

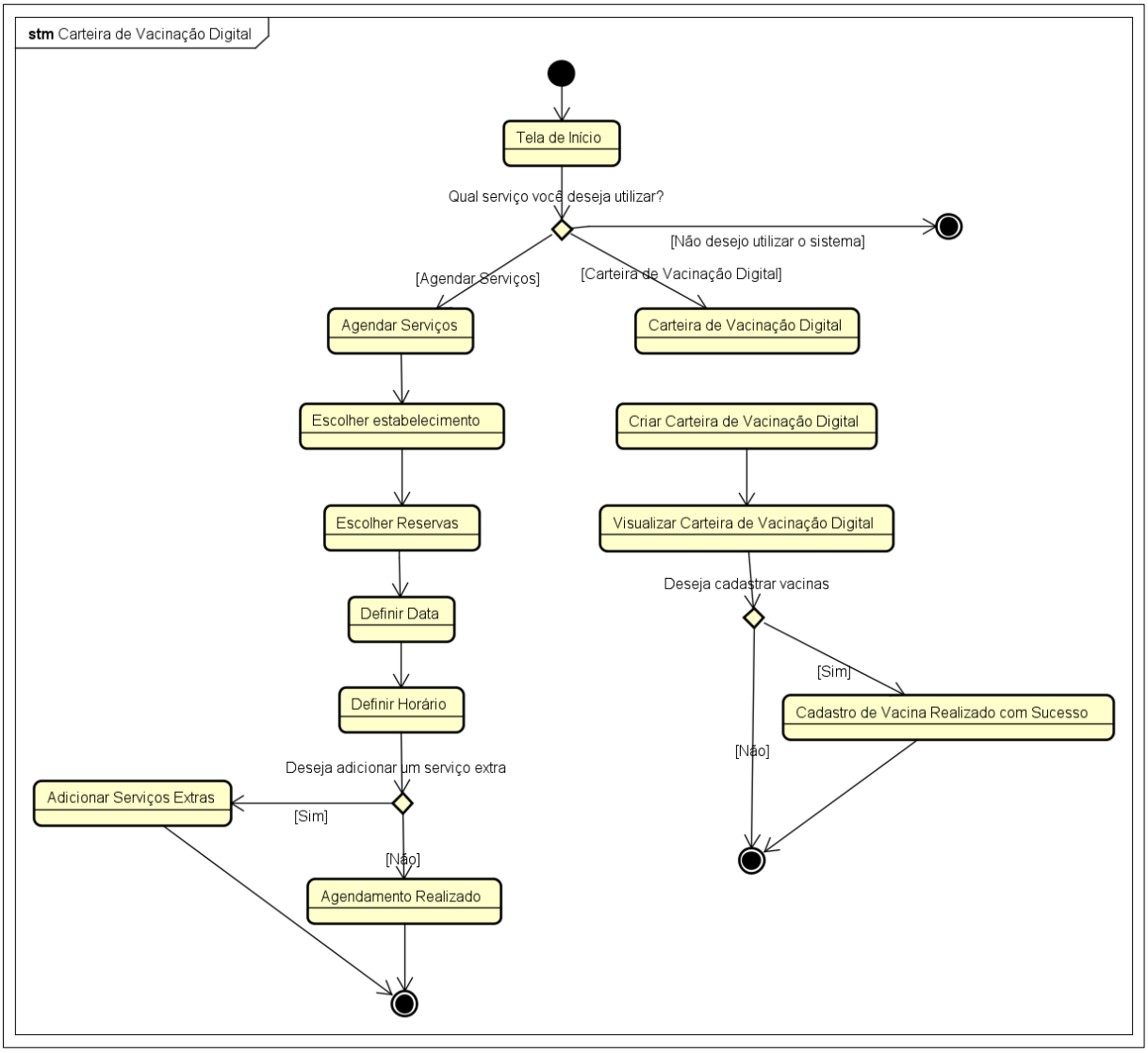
Figura 25 - Diagrama de Estado Login



Fonte: Autoria própria, 2022

Agora em relação ao aplicativo, a seguinte Figura 26, consegue evidenciar todos as ações que podem acontecer:

Figura 26 - Diagrama de Estado Aplicativo

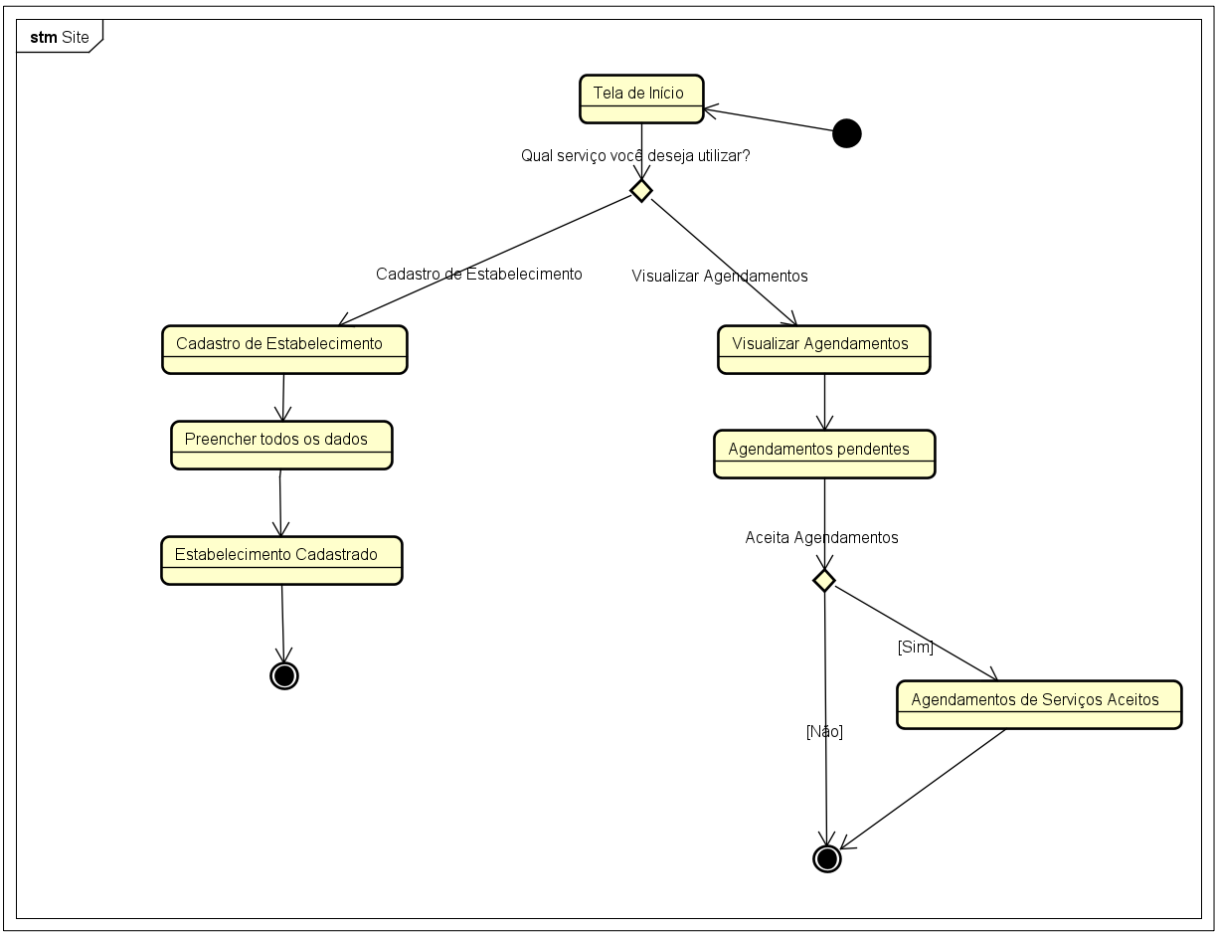


Fonte: Autoria própria, 2022

Nesse diagrama da Figura 26, nós podemos observar que existem duas ações principais, agendar serviços e carteira de vacinação digital, essas ações constituem as funções que estão presentes no aplicativo.

Os veterinários irão utilizar o site do nosso projeto, como e possível visualizar na Figura 27, temos todas as ações que podem ser realizadas no site:

Figura 27 - Diagrama de Estado Site

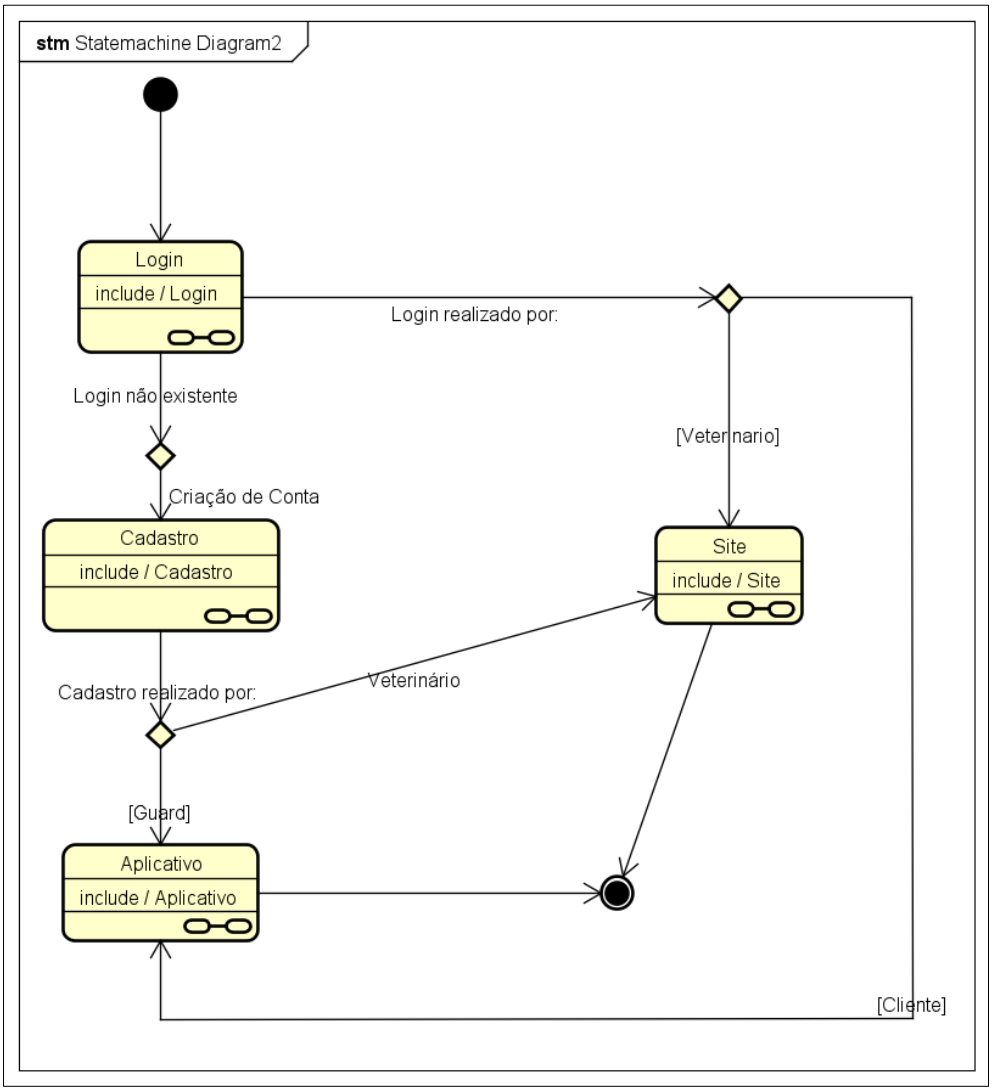


Fonte: Autoria própria, 2022

E possível observar nessa Figura 27, que os diagramas descrevem as duas principais ações que podem ser feitas no site, eles são cadastrar o estabelecimento dos veterinários e visualizar os agendamentos que foram efetuados pelos clientes.

Por último, como pode ser visto pela Figura 28, o diagrama de estados completo, reunindo todas as funções do projeto:

Figura 28 - Diagrama de Estado



Fonte: Autoria própria, 2022

Ao analisar o diagrama final da Figura 28, podemos ver duas bolinhas de marcação dentro dos balões de estado, elas servem para referenciar as ações do sistema, então o balão de login está referenciando o diagrama de estado login, que está presente na Figura 24. Essas referencias seguem em todos os balões do sistema.

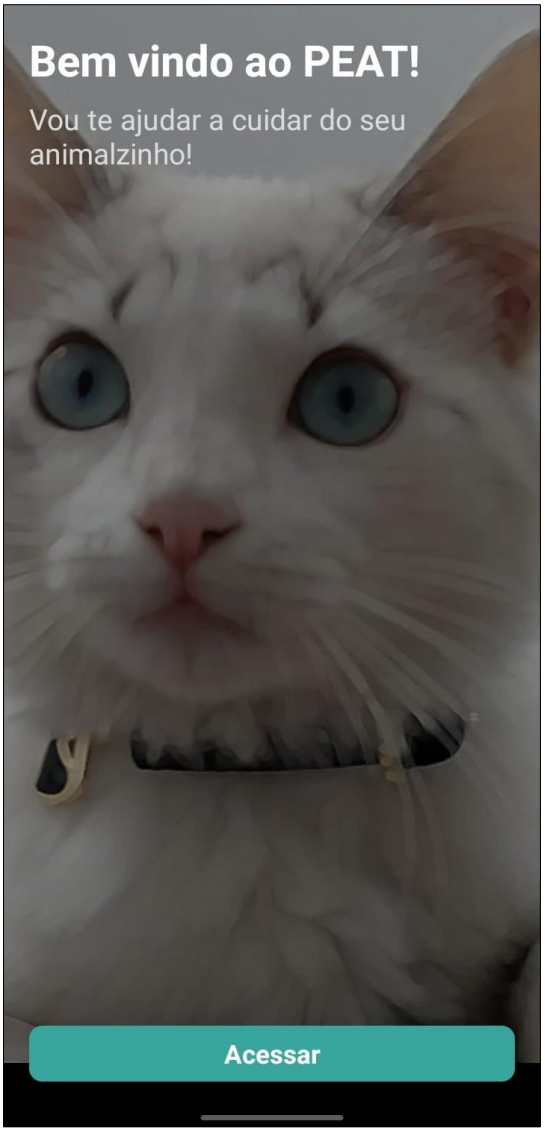
A construção do diagrama de estado, possibilitou que todas as funções e ações do nosso projeto fossem postas em um único lugar, facilitando assim a visualização e o entendimento de como o projeto funciona.

3.2. Telas do Aplicativo Peat

Neste capítulo, as telas do nosso aplicativo *mobile* estão sendo apresentas e explicadas. A primeira entre elas e a tela início do aplicativo, apresentada na Figura 29, ele será a primeira tela a ser executada ao abrir o aplicativo. Sua função é apenas

estética. Ela apresenta um botão que te direciona para o login, mas se o cliente já estiver logado, ela o direciona para a *home*:

Figura 29 - Tela de Início



Fonte: Autoria própria, 2022

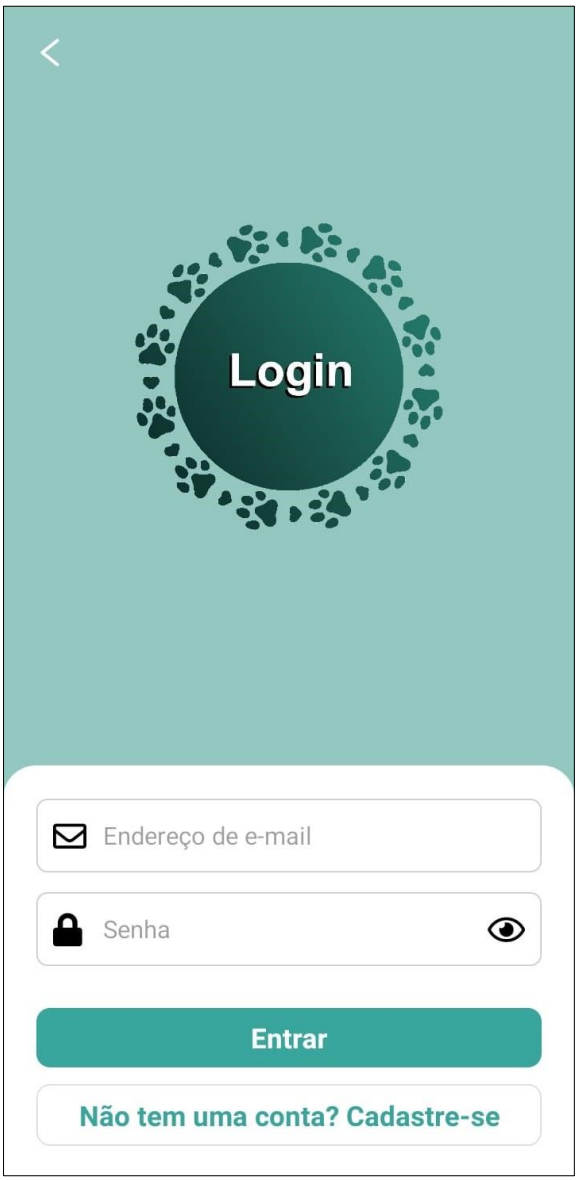
A tela da Figura 30, apresenta o cadastro do aplicativo, ela que requere o nome completo, *email* e senha do usuário, caso o usuário já possua uma conta, ao clicar em “Já tem uma conta? Entre” ele será redirecionado a tela de login:

Figura 30 - Tela de Cadastro

Fonte: Autoria própria, 2022

A próxima tela a ser apresentada no aplicativo é a Figura 31, está que a tela de login, ela irá requisitar *email* e senha, caso essas informações estejam corretas, o botão “Entrar” irá efetuar o login, caso o usuário não possua conta, ele pode clicar em “Não tem uma conta? Cadastre-se” para ser redirecionado a tela de cadastro:

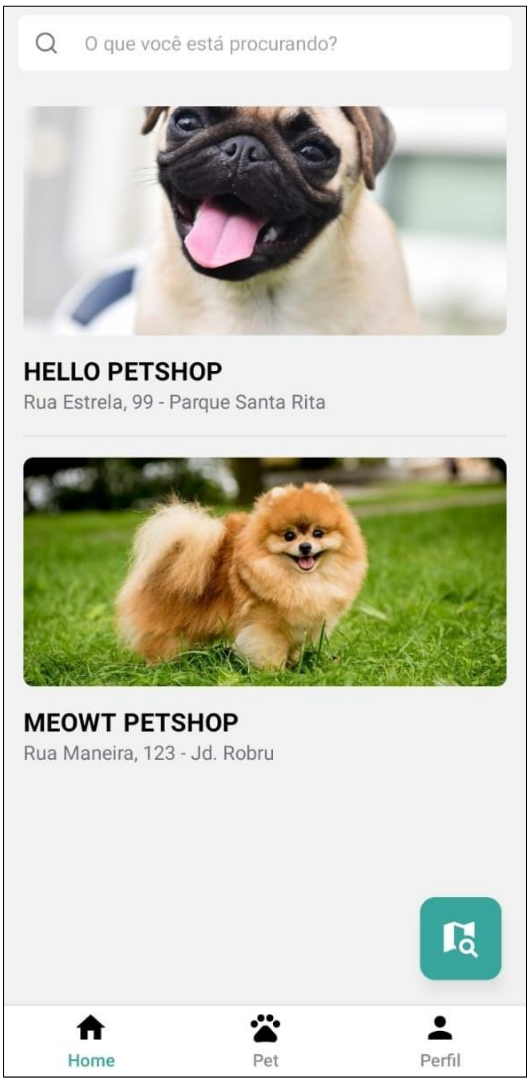
Figura 31 - Tela de Login



Fonte: Autoria própria, 2022

Ao realizar o login no sistema, encontramos a tela da Figura 32, onde usuário é encaminhado a tela *home* do *app*, está que é a tela onde os estabelecimentos cadastrados no banco de dados estão localizados em forma de lista. Ela apresenta uma função de busca no topo, onde o usuário pode pesquisar por um petshop ou veterinária específico. Ao clicar em um dos estabelecimentos, essa tela também apresenta um botão que o direciona ao mapa do sistema:

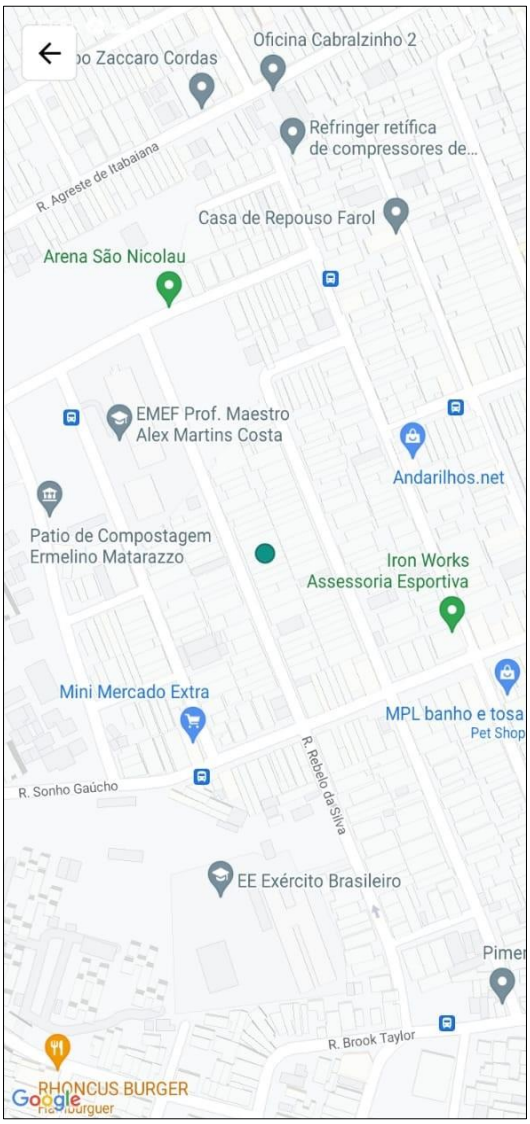
Figura 32 - Tela Home



Fonte: Autoria própria, 2022

Está tela de mapa localizada na Figura 33, tem a função de mostrar a posição em tempo real do usuário, em conjunto a isto, os estabelecimentos próximos ao cliente serão exibidos no mapa, para que assim ele consiga ter uma noção da sua distância até o *petshop* ou veterinário mais próximo. Ao acessar a tela a primeira vez, ela irá requisitar previamente permissão para uso da localização do usuário:

Figura 33 - Tela de Mapa



Fonte: Autoria própria, 2022

Podemos visualizar que na Figura 34, está a tela que é aberta ao usuário escolher um estabelecimento na tela *home*, será exibido a logo, nome e endereço do local, também é possível escolher os serviços que o usuário deseje reservar ao clicar nos botões com a escrita “Reservar”, eles irão direcionar o usuário a uma outra tela para concluir a reserva dos serviços:

Figura 34 - Tela do Estabelecimento

Fonte: Autoria própria, 2022

A tela que finaliza a reserva de um serviço está apresentada na Figura 35, nela o usuário pode escolher os horários e datas da reserva ao clicar no botão que contém o ícone de um calendário, ele também pode visualizar os serviços que estão sendo reservados, assim podendo adicionar outros serviços que o estabelecimento ofereça:

Figura 35 - Tela de Reservas

Fonte: Autoria própria, 2022

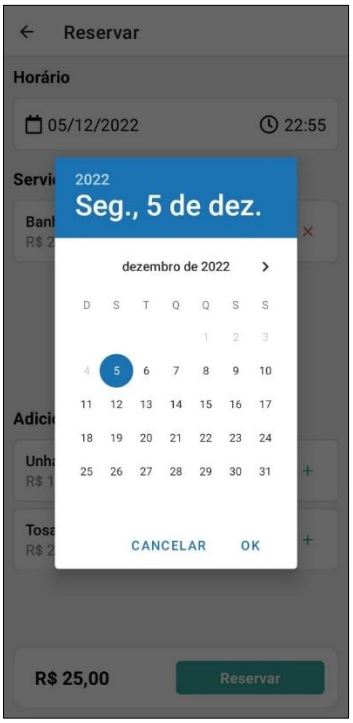
Ao clicar no botão com um ícone de calendário, a tela da Figura 36 é aberta, aqui o usuário pode escolher qual será o horário desejado para os serviços contratados:

Figura 36 - Tela de Horário

Fonte: Autoria própria, 2022

Em seguia ao selecionar o horário desejado, o usuário e enviado a tela representada na Figura 37, onde se é possível escolher as datas desejadas para a reserva:

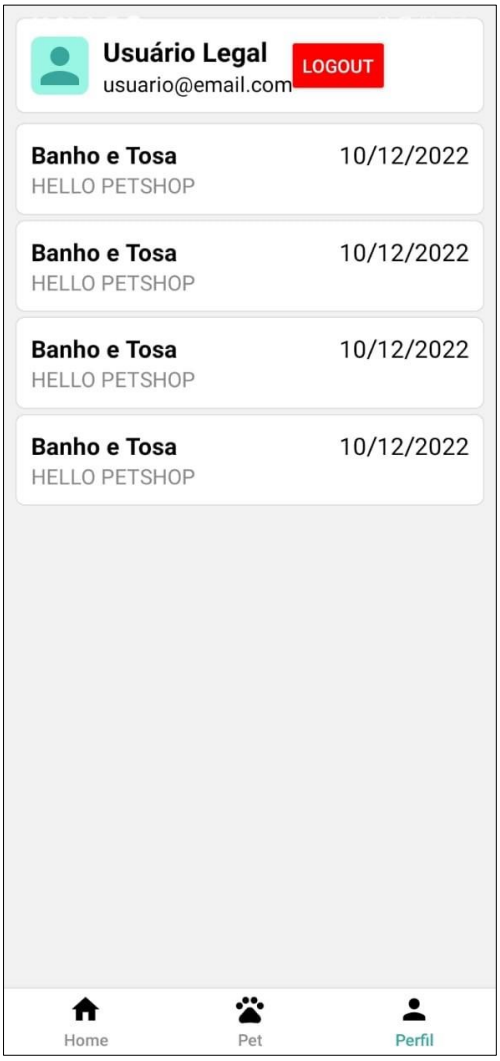
Figura 37 - Tela de Datas



Fonte: Autoria própria, 2022

Ao realizar a reserva de algum serviço, essas informações dela serão exibidas na tela perfil do usuário, que está representada pela Figura 38, aqui o usuário poderá visualizar as reservas que ele possui é conseguira realizar o *logout* do aplicativo ao clicar no botão vermelho escrito “LOGOUT”:

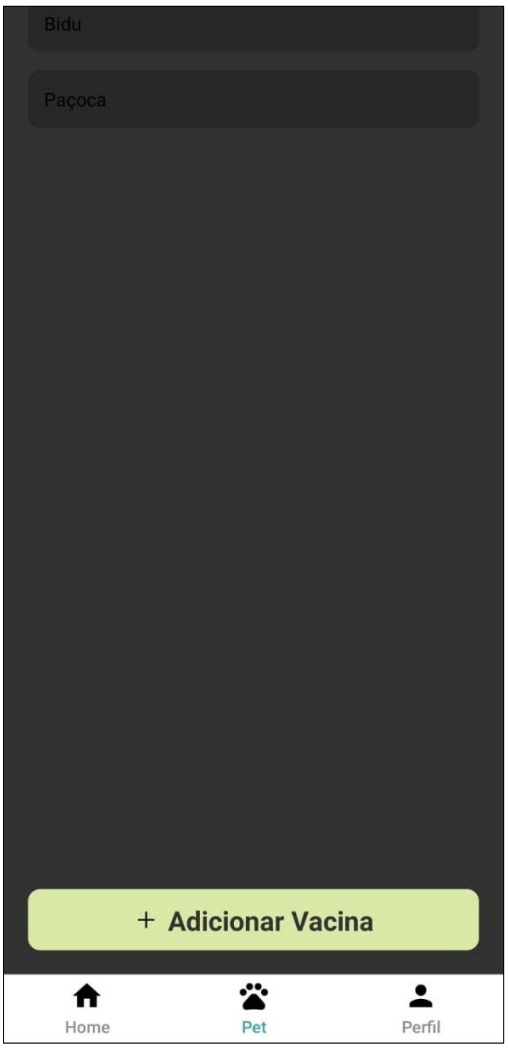
Figura 38 - Tela de Perfil



Fonte: Autoria própria, 2022

E por último temos a tela da Figura 39, está que é a ficha de vacinação digital, aqui o usuário tem a opção de cadastrar uma vacina para o seu animal. O botão "Adicionar Vacina" se abre uma barra na parte inferior da tela, onde uma caixa de texto e apresentada, para que o usuário possa inserir o nome da vacina. Ao adicionar ela, ela irá ser armazenada localmente no celular do usuário:

Figura 39 - Tela da Carteira de Vacinação Digital



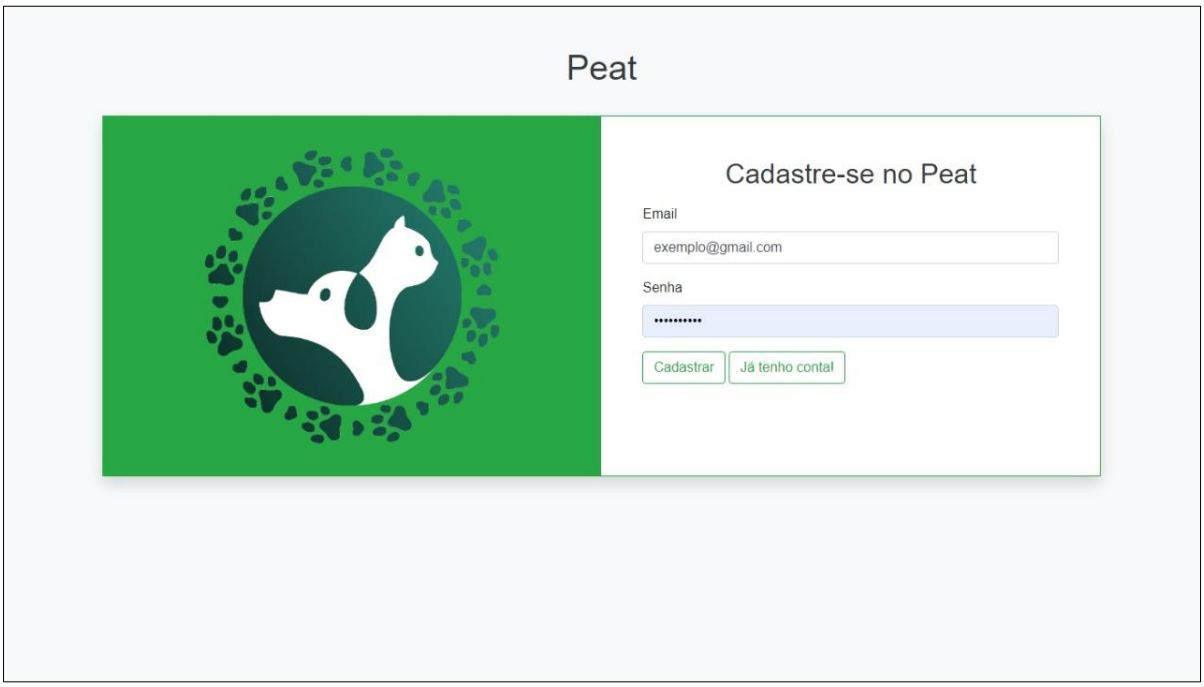
Fonte: Autoria própria, 2022

Todas essas telas constituem o nosso aplicativo mobile, ele que será majoritariamente de uso para os clientes que são donos de cães e gatos.

3.3. Telas do Site Peat

Nesse capítulo as telas do site estão sendo apresentadas e explicadas. A primeira tela que os veterinários se deparam é a tela de cadastro, está que pode ser visualizada na Figura 40, aqui o veterinário devera realizar o cadastro ao colocar o *email* e senha, e apertar no botão “Cadastrar”, entretanto se o veterinário já possuir um *login,* basta clicar em clicar em “Já tenho conta!” ele será redirecionado a tela de login:

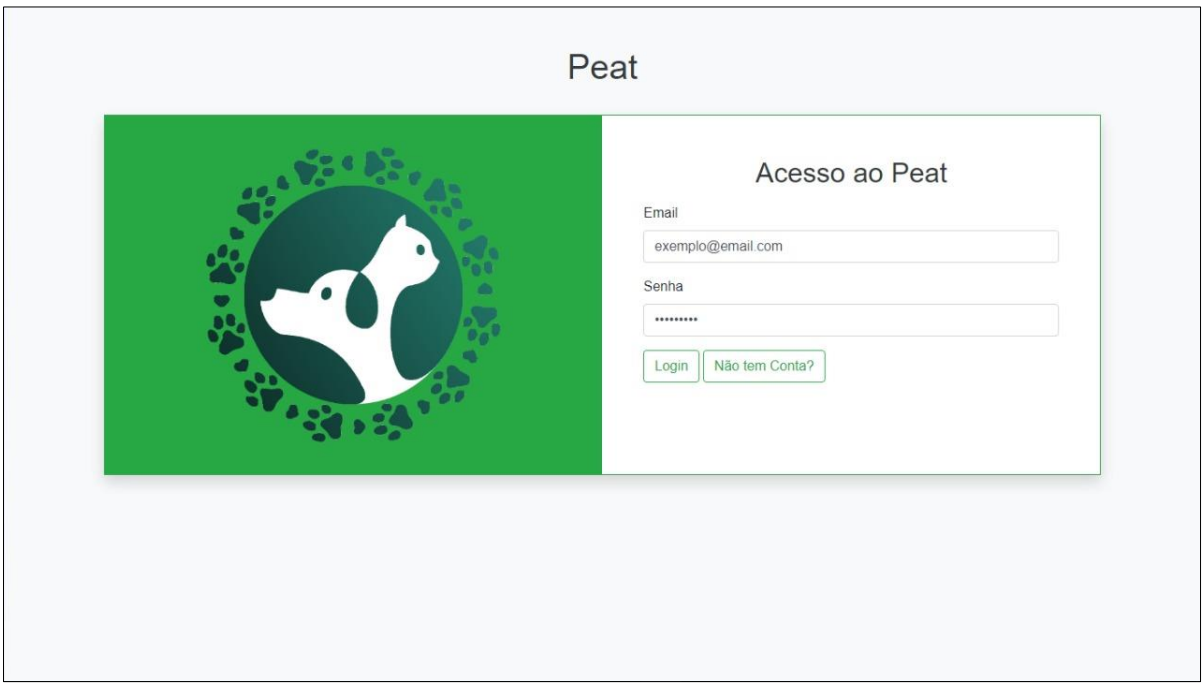
Figura 40 - Tela de Cadastro Site



Fonte: Autoria própria, 2022

A segunda tela entre elas está na Figura 41, é a tela de login no site, para realizar login ela irá requisitar *email* e senha, caso essas informações estejam corretas, ao pressionar o botão nomeado de “Login”, o veterinário irá logar no site. Caso ele não possua *login,* é só pressionar o botão “Não tem Conta?”, para ser direcionado a tela de cadastro:

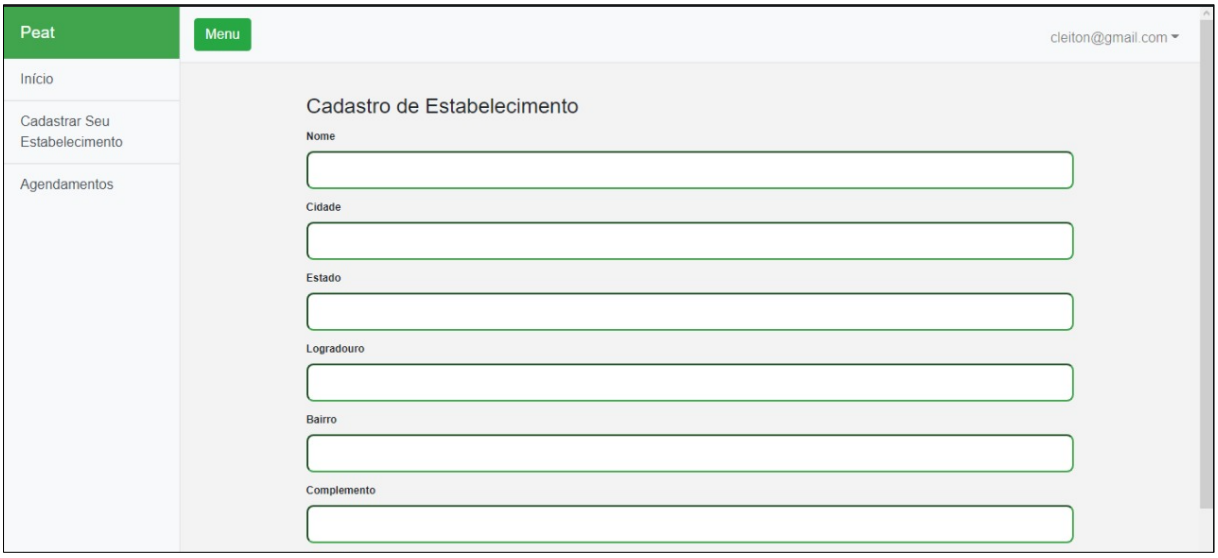
Figura 41 - Tela de Login Site



Fonte: Autoria própria, 2022

Ao adentrar o sistema o veterinário tem duas opções, ao observar a Figura 42 e 43, podemos entender que e os veterinários fazem o cadastro de seus estabelecimentos pela tela de cadastro de estabelecimentos que o site possui, aqui ele irá preencher os campos de nome, cidade, estado, logradouro, bairro, complemento, descrição número e foto, ao preencher todos os campo e só apertar em “Salvar” para cadastrar o estabelecimento do veterinário ou *petshop* no aplicativo, para que assim os usuários possam utilizar seus serviços:

Figura 42 - Tela de Cadastro de Estabelecimentos - Parte 1



Fonte: Autoria própria, 2022

A seguir na Figura 43 está a continuação da página de cadastro de estabelecimentos:

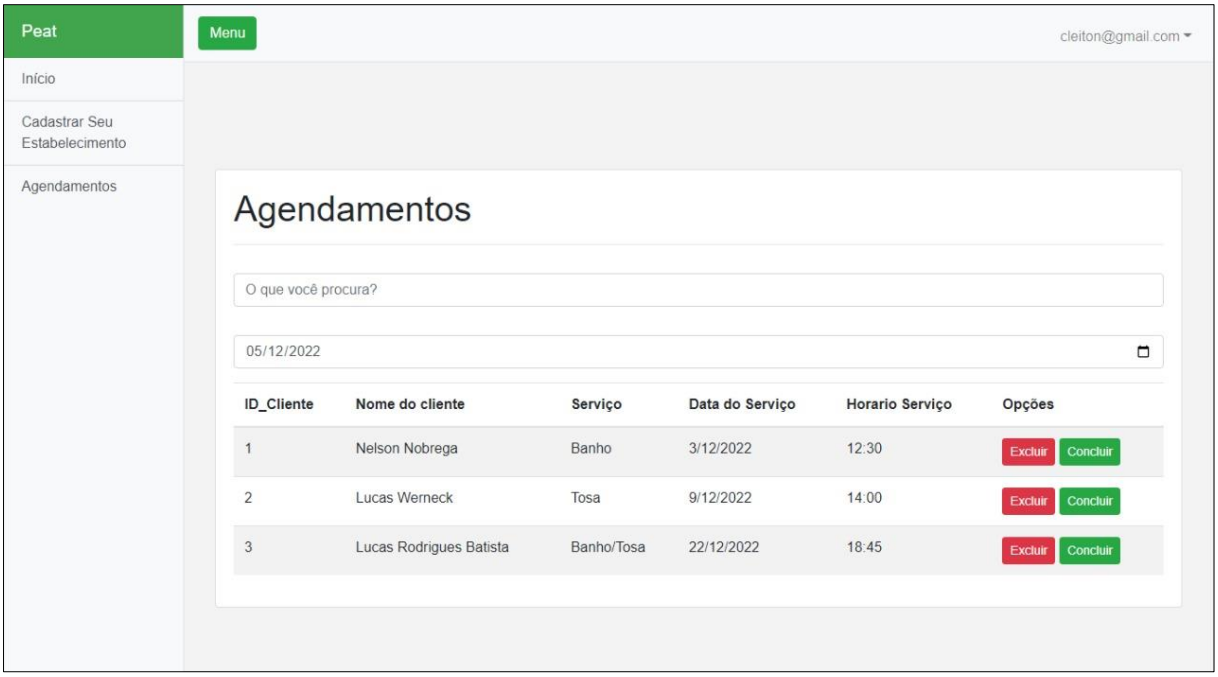
Figura 43 - Tela de Cadastro de Estabelecimentos - Parte 2

Fonte: Autoria própria, 2022

E a última tela que o site possui é de visualizar os agendamentos feitos pelos usuários do *app*, está sendo apresentada na Figura 44, os agendamentos realizados pelos usuários no aplicativo mobile se encontram aqui, possibilitando que os estabelecimentos possam analisar se podem atender a requisição de serviço, assim

aceitando ou cancelando o pedido, se utilizando do botão “Concluir” para marcar serviços já realizados e o botão “Excluir” para cancelar as reservas:

Figura 44 - Tela de Agendamentos



Fonte: Autoria própria, 2022

Com isso concluímos a apresentação e explicação de todas as telas que constituem o nosso projeto de conclusão de curso.

4. Conclusão

As tecnologias que nós utilizamos para construção da aplicação mobile e do site foram decisivos para o funcionamento de todo o projeto. O *React Native* possibilitou que os sistemas de criação da ficha de vacinação e de agendamento de serviços fossem possíveis. Ao juntar essa tecnologia com o banco de dados não-relacional Firebase, conseguimos criar uma *API* capaz de lidar com os agendamentos de serviços. Já para criação do site, nós utilizamos tecnologias mais usuais como *HTML, CSS, Bootstrap* e *Javascript*, essas que podem ser mais simples que o *React Native*, mas que ainda sim desempenham um excelente resultado para estilização e funcionalidades do site.

Contudo, esse sistema só foi capaz de ser estruturado de forma correta, ao utilizarmos dos estudos em *UML*, que foram decisivos para o planejamento do funcionamento de toda a aplicação, os diagramas de casos de uso, atividade e estados fizeram toda a diferença a criação do projeto, já que nós conseguimos identificar todos os pontos que precisavam ser realizados dentro do projeto.

Outro ponto que deu sustentação ao projeto, foi a entrevista concedida pela Juliana Justiço Mioshi, onde nós discutimos quais ideias poderiam funcionar ou não, a partir desse ponto, o nosso projeto teve um novo estopim, pois reunimos todos os requisitos funcionais, requisitos não funcionais e regras de negócio, então nós tínhamos em mente todas as funcionalidades que o sistema deveria possuir, fazendo com que ele fosse prático para usuários e para veterinários.

Com a conclusão do projeto, nós buscamos colocar o aplicativo mobile na loja de aplicativos dos aparelhos *android*, para que assim o máximo de pessoas podem se aproveitar dele. Com isso planejamos criar monetização a partir de anúncios que são exibidos após o cadastro de novas vacinas e ao realizar novos agendamentos.

Nós esperamos ansiosamente que o Peat tenha algum futuro no mercado de aplicativos, já que isso a influenciaria positivamente as taxas de vacinação entre cães e gatos, pois os donos desses pets sempre terão sua carteira de vacinação a postos em seus smartphones.

REFERÊNCIAS

BOOCH, Grady; RUMBAUGH, James. **UML**: guia do usuário. Rio de Janeiro: Elsevier, 2006.Disponível em: https://www.google.com.br/books/edition/UML/ddWqxcDKGF8C?hl=pt-BR&gbpv=0. Acesso em: 24 jun. 2022.

BOOTSTRAP. **Introdução**. Disponível em: https://getbootstrap.com.br/docs/4.1/content/reboot/. Acesso em: 01 dez. 2022.

ESCUDELARIO, Bruna; PINHO, Diego. **React Native**: desenvolvimento de aplicativos mobile com react. São Paulo: Casa do Código, 2020. Disponível em: https://www.google.com.br/books/edition/React_Native/keDdDwAAQBAJ?hl=pt-BR&gbpv=0. Acesso em: 24 jun. 2022.

FIREBASE. **Documentação do Firebase**. Disponível em: https://firebase.google.com/docs. Acesso em: 24 jun. 2022.

FLATSCHART, Fábio. **HTML 5 - Embarque Imediato**. Rio de Janeiro: Brasport, 2011. Disponível em: https://www.google.com.br/books/edition/HTML_5_Embarque_Imediato/_cgsCgAAQBAJ?hl=pt-BR&gbpv=1. Acesso em: 24 jun. 2022.

FLANAGAN, David. **JavaScript: o guia definitivo**. Porto Alegre: Bookman Editora, 2013. Disponível em: https://www.google.com.br/books/edition/JavaScript/zWNyDgAAQBAJ?hl=pt-BR&gbpv=0. Acesso em: 24 jun. 2022.

FOWLER, Martin. **UML Essencial**: um breve guia para linguagem padrao. Porto Alegre: Bookman Editora, 2005. Disponível em: https://www.google.com.br/books/edition/UML_Essencial_Um_Breve_Guia_para_Linguag/xxoXcuh0oS0C?hl=pt-BR&gbpv=1. Acesso em: 24 jun. 2022

GRILLO, Filipe del Nero; FORTES, Renata Pontin de Mattos. **Aprendendo JavaScript**. 2008. Disponível em: https://firebase.google.com/docs. Acesso em: 24 jun. 2022.

IBGE. **Pesquisa Nacional de Saúde**: informações sobre domicílios, acesso e utilização dos serviços de saúde. Rio de Janeiro: Ibge, 2019. Disponível em: https://biblioteca.ibge.gov.br/visualizacao/livros/liv101748.pdf. Acesso em: 24 jun. 2022.

IBM. **Modelos e Diagramas UML**. 2021. Disponível em: https://www.ibm.com/docs/pt-br/rsas/7.5.0?topic=models-uml-diagrams. Acesso em: 24 jun. 2022.

JOBSTRAIBIZER, Flávia. **Criação de sites com o CSS**: desenvolva páginas web mais leves e dinâmicas em menos tempo. São Paulo: Universo dos Livros Editora, 2009. Disponível em: https://www.google.com.br/books/edition/Criação_de_sites_com_o_CSS/Bdq5_oBRHqUC?hl=pt-BR&gbpv=0. Acesso em: 24 jun. 2022.

LIMA, Guilherme. **Bootstrap**: o que é, documentação, como e quando usar. O que é, Documentação, como e quando usar. 2022. Disponível em: https://www.alura.com.br/artigos/bootstrap#bootstrap:-o-que-e?. Acesso em: 01 dez. 2022.

MACHADO, Kheronn Khennedy. **Angular 11 e Firebase**: construindo uma aplicação integrada com a plataforma do google. São Paulo: Casa do Código, 2021. Disponível em: https://www.google.com.br/books/edition/Angular_11_e_Firebase/cRi9DwAAQBAJ?hl=pt-BR&gbpv=0. Acesso em: 24 jun. 2022.

MARIANO, Diego. **Bootstrap 5**: guia rápido para iniciantes. Lagoa Santa: Alfahelix Publicações, 2022. Disponível em: https://www.google.com.br/books/edition/Bootstrap_5_Guia_Rápido_para_Iniciantes/1idnEAAAQBAJ?hl=pt-BR&gbpv=0. Acesso em: 01 dez. 2022.

LARMAN, Craig. **Utilizando UML e Padrões**: uma introdução à análise e ao projeto orientados a objetos e ao desenvolvimento iterativo. Santama: Bookman, 2007. Disponível em: https://www.google.com.br/books/edition/Utilizando_UML_e_Padrões/hzl2tmT8QkUC?hl=pt-BR&gbpv=0&kptab=sideways. Acesso em: 24 jun. 2022.

LUCIDCHART. **O que é um diagrama UML?** Disponível em: https://www.lucidchart.com/pages/pt/o-que-e-uml. Acesso em: 24 jun. 2022.

MCCOMB, Matt. **React Native - Desenvolvendo aplicações nativas com JavaScript**. 2015. Disponível em: https://www.infoq.com/br/news/2015/04/facebook-announces-react-native/. Acesso em: 24 jun. 2022.

MOZILA. **CSS básico**. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Learn/Getting_started_with_the_web/CSS_basics>. Acesso em: 24 jul. 2022.

PINHEIRO, Paulo César da Costa. **DESENVOLVIMENTO DE UM TUTORIAL HIPERTEXTO EM HTML1**. Disponível em: https://www.researchgate.net/profile/Paulo-Cesar-Pinheiro/publication/334971400_DESENVOLVIMENTO_DE_UM_TUTORIAL_HIPERTEXTO_EM_HTML/links/5d4846ee92851cd046a353dd/DESENVOLVIMENTO-DE-UM-TUTORIAL-HIPERTEXTO-EM-HTML.pdf. Acesso em: 24 jun. 2022.

SCHEIDT, Felippe Alex. **Fundamentos de CSS**: criando design para sistemas web. Foz do Iguaçu: Outbox Livros Digitais, 2015. Disponível em: https://www.google.com.br/books/edition/Fundamentos_de_CSS_criando_design_para_s/04cbCgAAQBAJ?hl=pt-BR&gbpv=0. Acesso em: 24 jun. 2022.

SILVA, Maurício Samy. **HTML5**: a linguagem de marcação que revolucionou a web. São Paulo: Novatec Editora, 2019. Disponível em: https://www.google.com.br/books/edition/HTML5/tDG-DwAAQBAJ?hl=pt-BR&gbpv=1. Acesso em: 14 jun. 2022.

SILVA, Maurício Samy. **Construindo Sites com CSS e (X)HTML**: sites controlados por folhas de estilo em cascata. São Paulo: Novatec Editora, 2007. Disponível em: https://www.google.com.br/books/edition/Construindo_Sites_com_CSS_e_X_HTML/BxKWAwAAQBAJ?hl=pt-BR&gbpv=0. Acesso em: 24 jun. 2022.

SILVA, Maurício Samy. **Fundamentos de HTML5 e CSS3**. São Paulo: Novatec Editora, 2018. Disponível em: https://www.google.com.br/books/edition/Fundamentos_de_HTML5_e_CSS3/ZyJyDwAAQBAJ?hl=pt-BR&gbpv=0. Acesso em: 24 jun. 2022.

SILVA, Maurício Samy. **JavaScript - Guia do Programador**: guia completo das funcionalidades de linguagem javascript. São Paulo: Novatec Editora, 2020. Disponível em: https://www.google.com.br/books/edition/JavaScript_Guia_do_Programador/6DfnDwAAQBAJ?hl=pt-BR&gbpv=1. Acesso em: 24 jun. 2022.

SILVA, Maurício Samy. **React Aprenda Praticando**: desenvolva aplicações web reais com uso da biblioteca react e de seus módulos auxiliares. São Paulo: Novatec Editora, 2021. Disponível em: https://www.google.com.br/books/edition/React_Aprenda_Praticando/MWkOEAAAQBAJ?hl=pt-BR&gbpv=0. Acesso em: 24 jun. 2022.

STEFANOV, Stoyan. **Primeiros passos com React**: construindo aplicações web. São Paulo: Novatec Editora, 2016. Disponível em: https://www.google.com.br/books/edition/Primeiros_passos_com_React/ff6ZDwAAQBAJ?hl=pt-BR&gbpv=0. Acesso em: 24 jun. 2022.

TEJADA, Zoiner. **Dados não relacionais e NoSQL**. Disponível em: https://learn.microsoft.com/pt-br/azure/architecture/data-guide/big-data/non-relational-data#feedback. Acesso em: 24 jun. 2022.