



Ministério da Educação Instituto Federal de Educação, Ciência e Tecnologia do Pará – Campus

Altamira

Graduação em Análise e Desenvolvimento de Sistemas

Disciplina Programação Web II

Professor Obedio Albuquerque

MEIKE DIONE LIBORIO BRILHANTE

PROTÓTIPO DE BLOG

PROJETO FINAL

Altamira

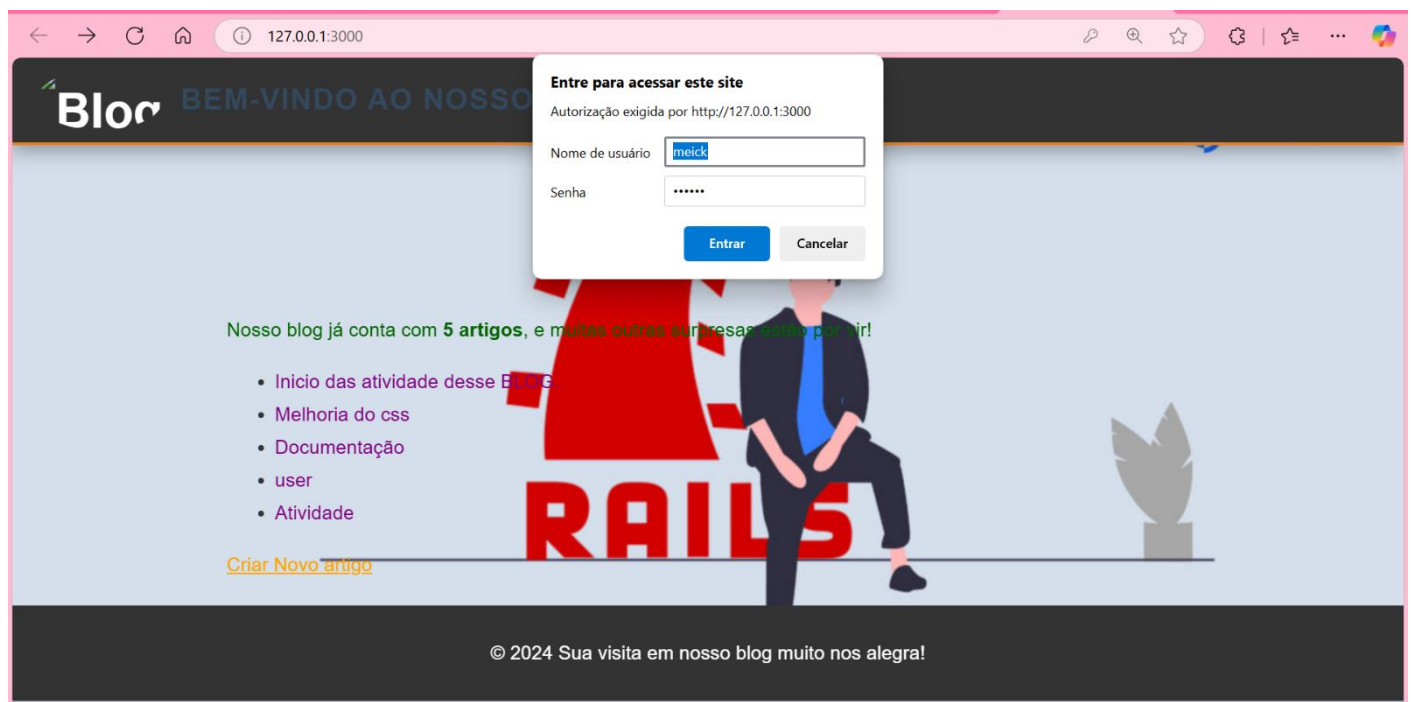
2024

# Sumário

1. Introdução.....	3
1.1 Objetivo do projeto.....	3
2. Desenvolvimento.....	4
2.1 Estrutura Geraldo Projeto .....	4
2.1.1 Frontend: HTML, e CSS.....	4
2.1.2 Backend: Controladores, Modelos e Banco de Dados .....	5
2.2 Funcionalidades Desenvolvidas .....	6
2.2.1 Gerenciamento de Artigos .....	7
2.2.2 Sistema de Autenticação e Controle de Acesso .....	8
2.2.3 Estilização e Usabilidade .....	9
2.2.4 Interações Dinâmicas no Formulário de Artigos.....	10
3. Conclusão .....	11
4. Referências .....	12

## 1. INTRODUÇÃO.

O plano contempla recursos fundamentais, como o registro de usuários com autenticação segura, a habilidade de criar e administrar publicações, além da opção de fazer comentários nas publicações de outros usuários. Este projeto, além de oferecer uma experiência prática em desenvolvimento web, também tem como objetivo ampliar minha compreensão em autenticação de usuários, gestão de dados e design de interfaces. Com o auxílio vscode e o framework Ruby on Rails e da gem Devise, tentei criar uma solução que não só cumpra os requisitos técnicos, mas também proporcionasse uma interface intuitiva e de fácil acesso. Durante o trabalho, vou explicar as características do blog, as tecnologias empregadas, os obstáculos encontrados ao longo do processo e os resultados alcançados.



### 1.1 Objetivo do Projeto:

Desenvolver um protótipo de blog com funcionalidades de criar, ler, atualizar e deletar.

## 2. DESENVOLVIMENTO.

### 2.1 Estrutura Geral do Projeto

O projeto foi desenvolvido em Ruby on Rails, aplicando o padrão MVC (Model-View-Controller). Abaixo, detalha-se a implementação de cada camada.

#### 2.1.1 Frontend: HTML e CSS

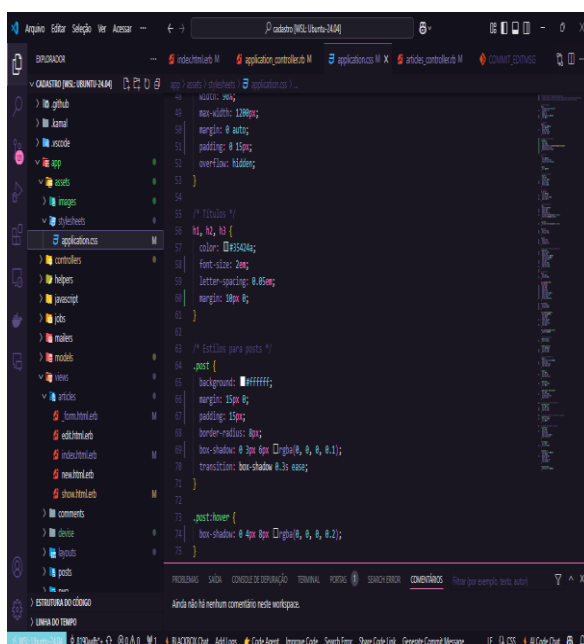
O frontend foi desenvolvido utilizando majoritariamente criado no Visual Studio Code, permitindo gerar HTML dinamicamente. Além disso, foram aplicados estilos CSS personalizados, que garantem uma interface visual consistente.

#### Principais arquivos do frontend:

- Views: Local onde estão os arquivos ERB que geram as páginas.

```
app > views > articles > index.html.erb
1 <p style="color: gray"><%= notice %></p>
2 <h1 style="color: darkblue">Articles</h1>
3
4 <!-- Exibe a contagem correta dos artigos públicos com pluralização dinâmica -->
5 <p style="color: darkgreen">Nosso blog já conta com <strong><%= pluralize(Article.public_count, 'artigo', 'artigos') %></strong>, e muitas outras
6
7
8 <ul>
9   <%= @articles.each do |article| %>
10     <% unless article.archived? %> <!-- Verifica se o artigo não está arquivado -->
11       <li>
12         <%= link_to article.title, article, style: "color: purple" %> <!-- Link para visualizar o artigo -->
13       </li>
14     <% end %>
15   <% end %>
16 </ul>
17
18 <%= link_to "Criar Novo artigo", new_article_path, style: "color: orange" %> <!-- Link para criar um novo artigo
19
```

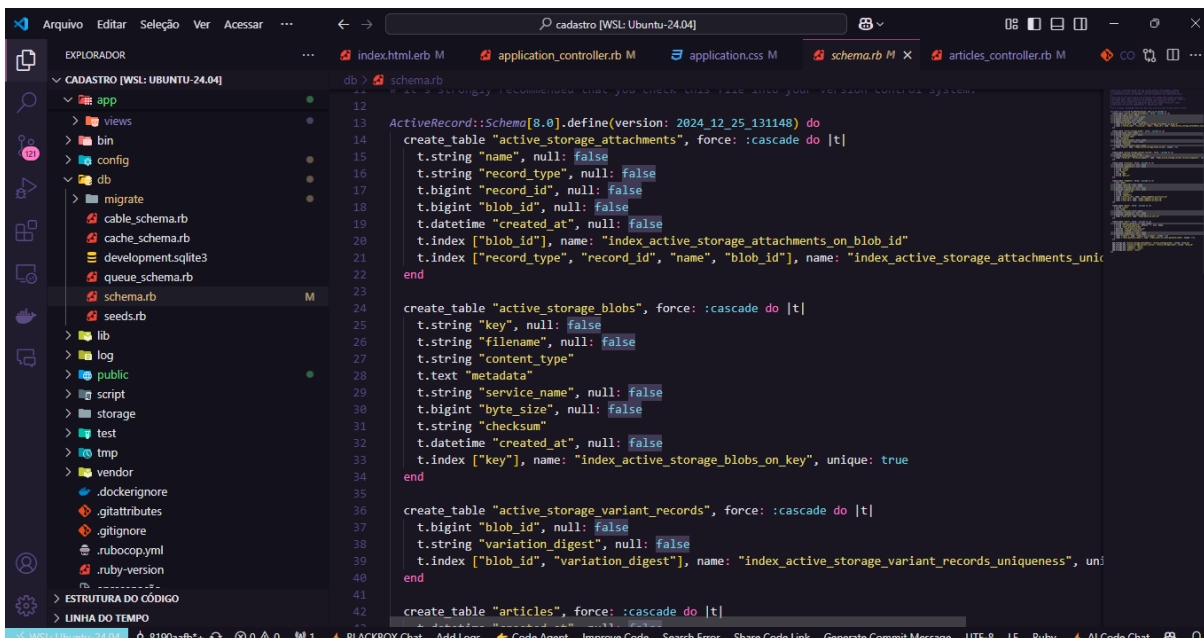
- CSS: Local onde estão definidos os estilos que foram aplicados nos botões, formulários e layout geral do site.



```
app > views > layouts > application.html.erb
1 <!DOCTYPE html>
2 <html lang="pt-BR">
3   <head>
4     <meta charset="UTF-8">
5     <meta name="viewport" content="width=device-width, initial-scale=1">
6     <meta name="apple-mobile-web-app-capable" content="yes">
7     <meta name="mobile-web-app-capable" content="yes">
8     <%= csrf_meta_tags %>
9     <%= csp_meta_tag %>
10
11     <title><%= content_for(:title) || "Cadastro" %></title>
12
13     <!-- Favicon e ícones -->
14     <link rel="icon" href="/icon.png" type="image/png">
15     <link rel="icon" href="/icon.svg" type="image/svg+xml">
16     <link rel="apple-touch-icon" href="/icon.png">
17
18     <!-- Estilos -->
19     <%= stylesheet_link_tag :app, "data-turbo-track": "reload" %>
20     <%= stylesheet_link_tag 'application', "data-turbo-track": "reload" %>
21
22     <!-- Scripts -->
23     <%= javascript_importmap_tags %>
24     <%= javascript_include_tag "turbo", type: "module" %>
25
26     <!-- Conteúdo adicional no cabeçalho -->
27     <%= yield :head %>
28
```

## 2.1.2 Backend: Controladores, Modelos e Banco de Dados

No backend, foi seguido o padrão MVC. Os modelos representam as tabelas do banco de dados, os controladores contêm a lógica de negócio e as views geram as páginas exibidas ao usuário.



### PRINCIPAIS COMPONENTES DO BACKEND:

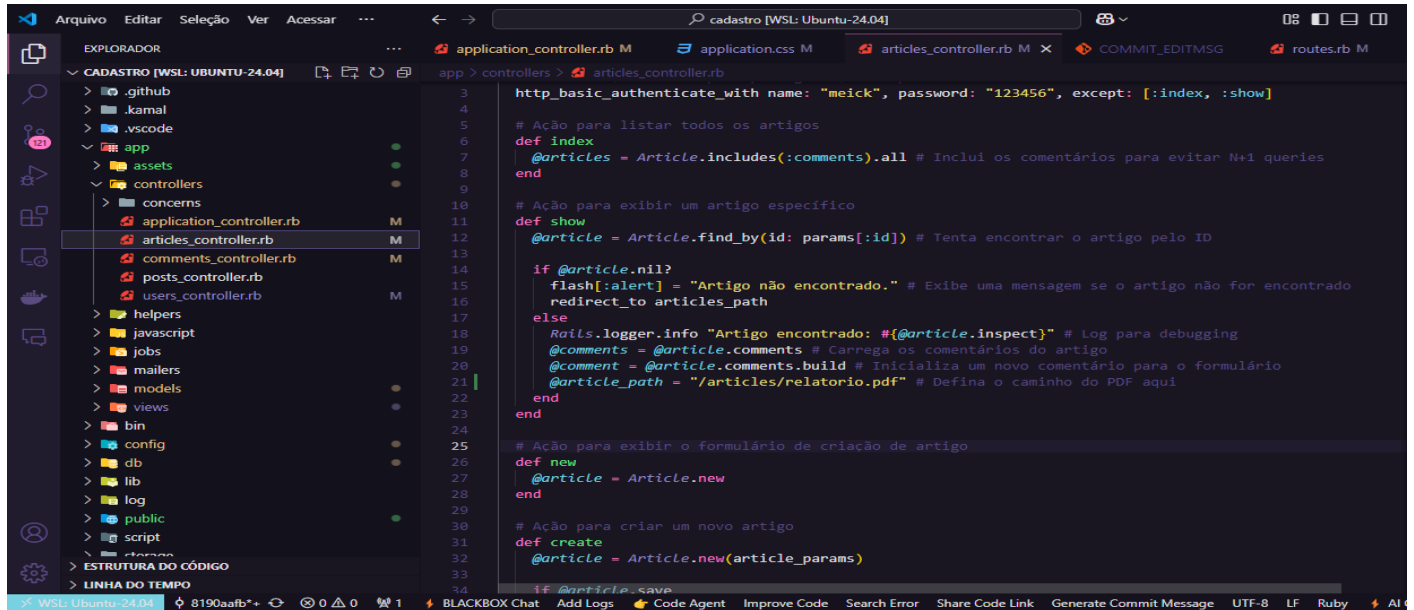
- **Modelos:** artigo.rb, usuario.rb, comentario.rb, que representam as entidades.

```
app > models > article.rb
6 # end
7
8 class Article < ApplicationRecord
9   include Visible
10
11   has_many :comments, dependent: :destroy
12
13   validates :title, presence: true
14   validates :body, presence: true, length: { minimum: 10 }
15   validates :status, inclusion: { in: ['public', 'private', 'archived'] }
16
17   def self.public_count
18     where(status: 'public').count
19   end
20 end
21 #module Visible
```

```
app > models > user.rb
1 class User < ApplicationRecord
2
3   # Include default devise modules. Others available are:
4   # Inclua os módulos padrão do Devise. Outros disponíveis são:
5   # :confirmable, :lockable, :timeoutable, :trackable and :omniauthable
6   # Devise modules
7   devise :database_authenticatable, :registerable,
8         :recoverable, :rememberable, :validatable
9
10  # Associações
11  has_many :posts
12  has_many :comments
13  has_secure_password
14  validates :username, presence: true, uniqueness: true
15 end
```

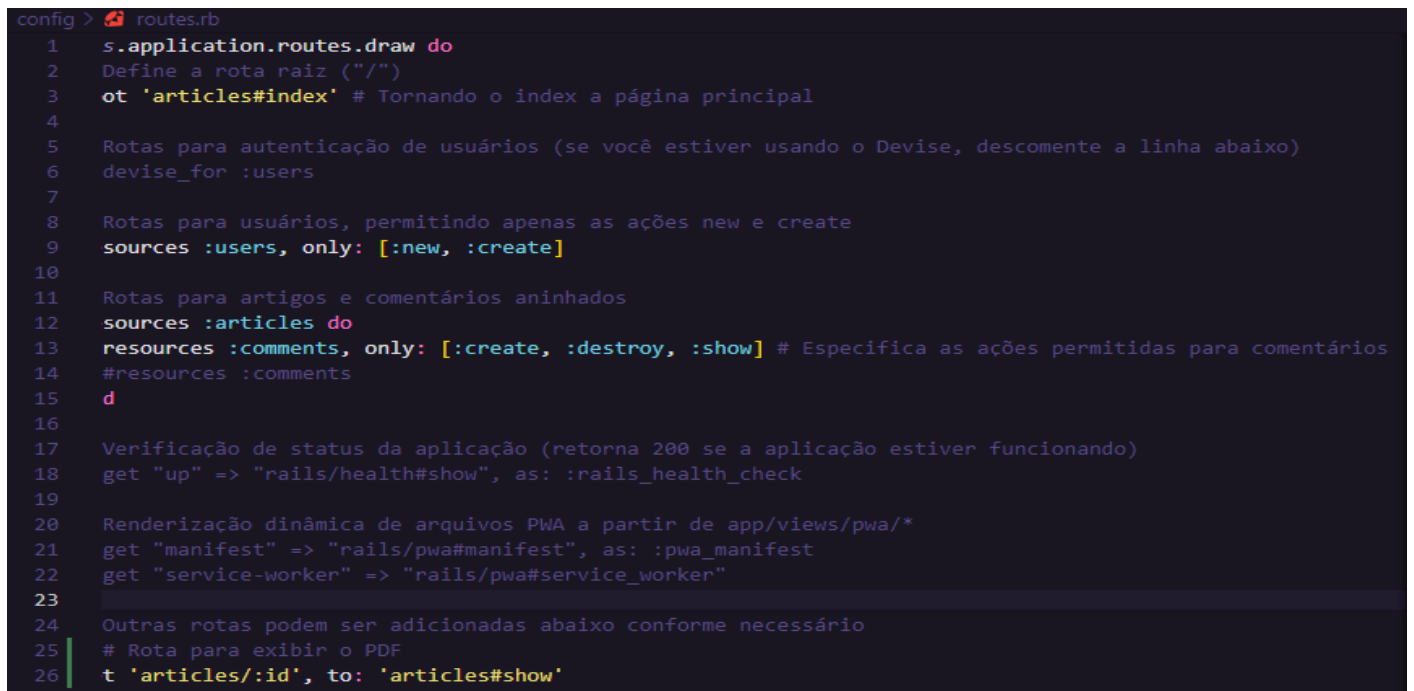
```
app > models > comment.rb
1 class Comment < ApplicationRecord
2   # belongs_to :article
3   # belongs_to :user
4   # belongs_to :post
5   #end
6   include Visible
7
8   belongs_to :article
9   belongs_to :user
10 end
```

**Controladores:** `artigos_controller.rb`, `usuarios_controller.rb`, `comentarios_controller.rb`, que definem ações de CRUD.



The screenshot shows the VS Code interface with the Explorer sidebar on the left displaying the project structure. The main editor window shows the `articles_controller.rb` file. The code includes a `http_basic_authenticate_with` call, an `index` action to list all articles with comments, a `show` action to display a specific article and its comments, and a `new` action to show the form for creating a new article. The `create` action is also partially visible.

- **Rotas:** Configuradas para definir as URLs associadas a cada ação.



The screenshot shows the `routes.rb` file in the VS Code editor. It defines the application routes, including the root route (`articles#index`), routes for user authentication (`devise_for :users`), routes for users (`sources :users, only: [:new, :create]`), and nested routes for articles and comments (`sources :articles` and `resources :comments`). It also includes a health check route (`get "up"`) and dynamic PWA routes.

## 2.2 Funcionalidades Desenvolvidas.

A seguir, apresentamos as funcionalidades da aplicação, detalhando o funcionamento do backend e frontend.

### 2.2.1 Gerenciamento de Artigos

O sistema permite que usuários autenticados criem, editem e excluam artigos.

## Backend:

- **Modelo artigo.rb:** Define o esquema de artigos e as validações.

```
8 class Article < ApplicationRecord
9   include Visible
10
11   has_many :comments, dependent: :destroy
12
13   validates :title, presence: true
14   validates :body, presence: true, length: { minimum: 10 }
15   validates :status, inclusion: { in: ['public', 'private', 'archived'] }
16
17   def self.public_count
18     where(status: 'public').count
19   end
20 end
```

```
app > views > articles > index.html.erb
1 <p style="color: gray"><%= notice %></p>
2 <h1 style="color: darkblue">Articles</h1>
3
4 <!-- Exibe a contagem correta dos artigos públicos com pluralização dinâmica -->
5 <p style="color: darkgreen">Nosso blog já conta com <strong><%= pluralize(Article.public_count, 'artigo', 'artigos') %></strong>, e muitas outras surpresas estão por vir!</p>
6
```

**Controlador artigos\_controller.rb:** Gerencia as ações de criação, leitura, atualização e exclusão de artigos.

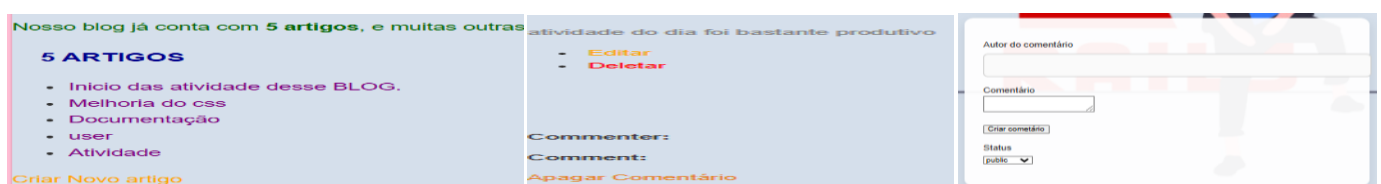
```
app > models > comment.rb
1 class Comment < ApplicationRecord
2   # belongs_to :article
3   # belongs_to :user
4   # belongs_to :post
5   #end
6   include Visible
7
8   belongs_to :article
9   belongs_to :user
10 end
```

- **Rotas:** Mapeiam URLs para as ações do controlador de artigos.

```
config > routes.rb
1 Rails.application.routes.draw do
10
11   # Rotas para artigos e comentários aninhados
12   resources :articles do
13     resources :comments, only: [:create, :destroy, :show] # Especifica as ações permitidas para comentários
14     #resources :comments
15   end
16
```

## Frontend:

- **Views:** da página como new.html.erb, edit.html.erb, show.html.erb. Estruturam as páginas de criação, edição e visualização.



- **Formulário de Artigo: (\_formulario.html.erb)** Cria o formulário e edição de artigos, com validações e estilização.

```
app > views > articles > _form.html.erb
1  <%= form_with model: article do |form| %>
2
3  <div>
4    <%= form.label :status %><br>
5    <%= form.select :status, Visible::VALID_STATUSES, selected: article.status || 'public' %>
6  </div>
7
8  <div>
9    <%= form.label :title %><br>
10   <%= form.text_field :title %>
11   <% article.errors.full_messages_for(:title).each do |message| %>
12     <div><%= message %></div>
13   <% end %>
14 </div>
15
16 <div>
17   <%= form.label :body %><br>
18   <%= form.text_area :body %><br>
19   <% article.errors.full_messages_for(:body).each do |message| %>
20     <div><%= message %></div>
21   <% end %>
22 </div>
23
24 <div>
25   <%= form.submit %>
26 </div>
```

### 2.2.2 Sistema de Autenticação e Controle de Acesso

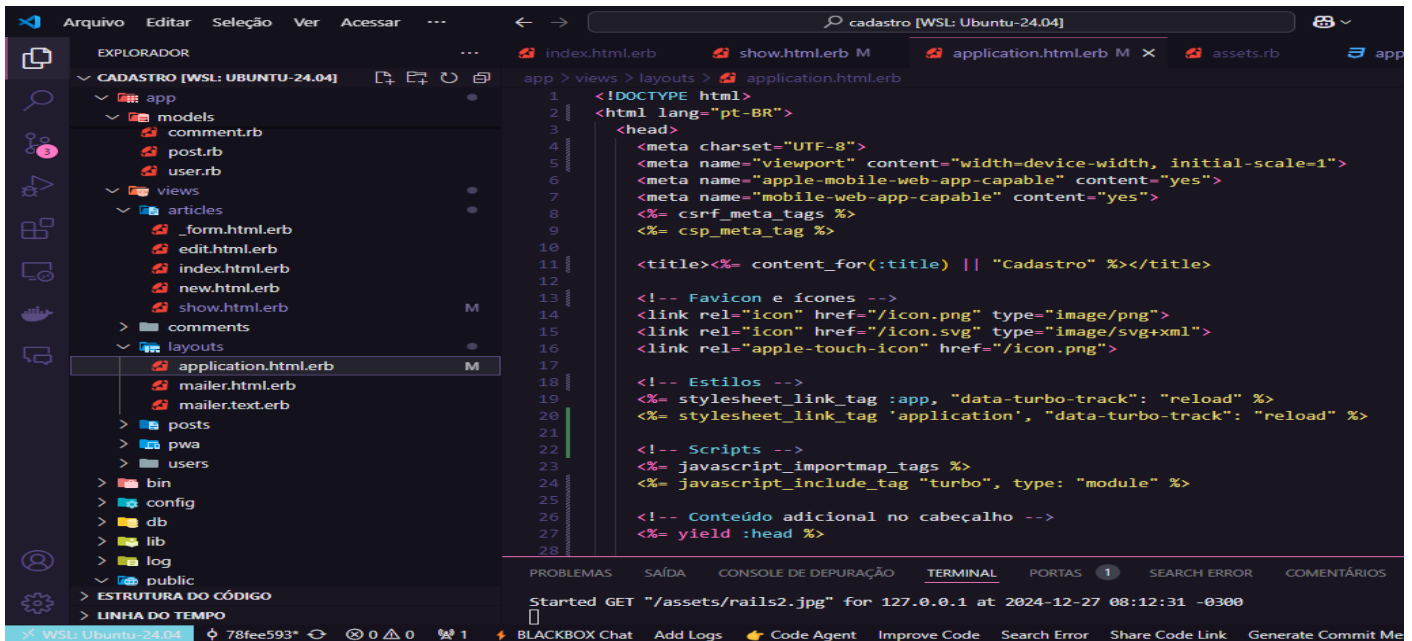
Para o controle de acesso, utilizou-se a gem Devise, que permite que apenas usuários autenticados acessem determinadas ações.

```
5  gem "rails", "~> 8.0.0.rc2"
6  # Adiciona a gem Devise para autenticação de usuários
7  gem 'devise'
8  # The modern asset pipeline for Rails [https://github.com/rails/propshaft]
9  gem "propshaft"
10 # Use sqlite3 as the database for Active Record
```



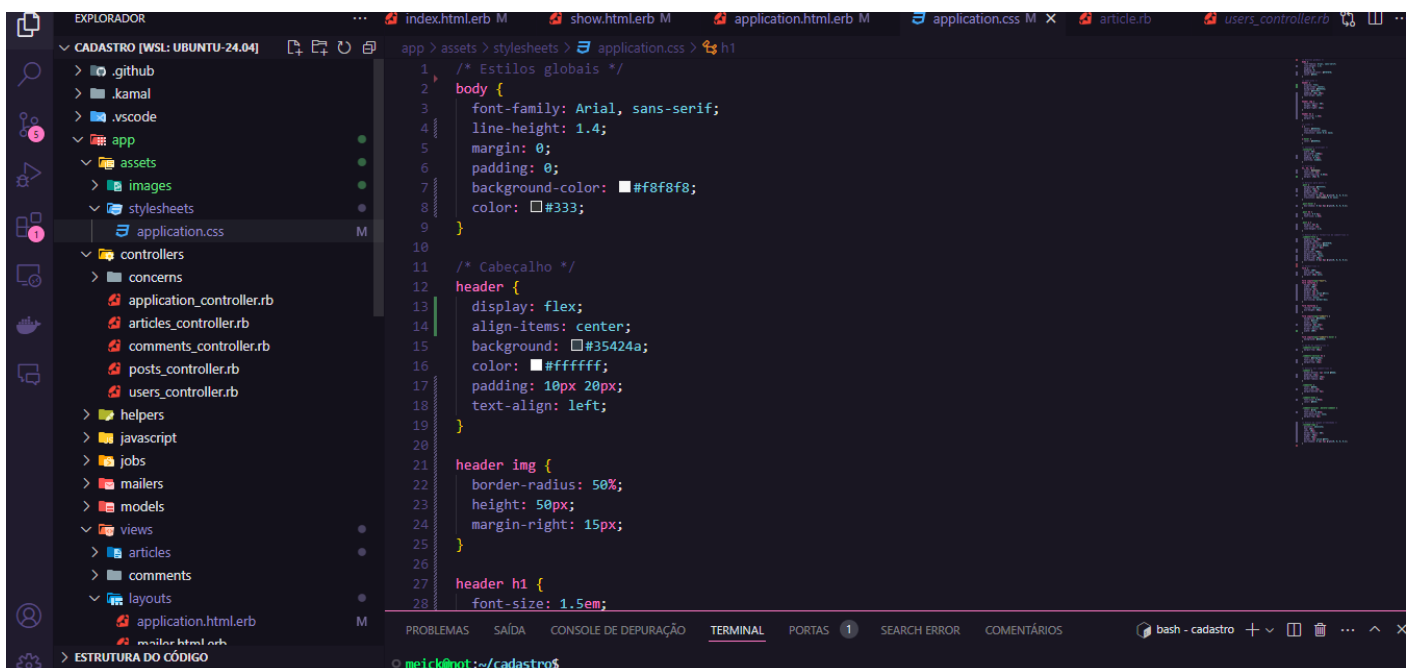
### 2.2.3 Estilização e Usabilidade

O CSS (application.css, application.html.erb) foi aplicado para garantir uma aparência uniforme, com foco em usabilidade. Utilizamos classes CSS personalizadas para os botões, mensagens e formulários.



```
1 <!DOCTYPE html>
2 <html lang="pt-BR">
3   <head>
4     <meta charset="UTF-8">
5     <meta name="viewport" content="width=device-width, initial-scale=1">
6     <meta name="apple-mobile-web-app-capable" content="yes">
7     <meta name="mobile-web-app-capable" content="yes">
8     <%= csrf_meta_tags %>
9     <%= csp_meta_tag %>
10
11     <title><%= content_for(:title) || "Cadastro" %></title>
12
13     <!-- Favicon e ícones -->
14     <link rel="icon" href="/icon.png" type="image/png">
15     <link rel="icon" href="/icon.svg" type="image/svg+xml">
16     <link rel="apple-touch-icon" href="/icon.png">
17
18     <!-- Estilos -->
19     <%= stylesheet_link_tag :app, "data-turbo-track": "reload" %>
20     <%= stylesheet_link_tag 'application', "data-turbo-track": "reload" %>
21
22     <!-- Scripts -->
23     <%= javascript_importmap_tags %>
24     <%= javascript_include_tag "turbo", type: "module" %>
25
26     <!-- Conteúdo adicional no cabeçalho -->
27     <%= yield :head %>
28   </head>
29   <body>
30   </body>
31 </html>
```

Essas camadas trazem um dinamismo a pagina do sistema, uma vez editável da para deixa bem elegante a interface.



```
1 /* Estilos globais */
2
3 body {
4   font-family: Arial, sans-serif;
5   line-height: 1.4;
6   margin: 0;
7   padding: 0;
8   background-color: #f8f8f8;
9   color: #333;
10 }
11
12 /* Cabeçalho */
13 header {
14   display: flex;
15   align-items: center;
16   background-color: #35424a;
17   color: #ffffff;
18   padding: 10px 20px;
19   text-align: left;
20 }
21
22 header img {
23   border-radius: 50%;
24   height: 50px;
25   margin-right: 15px;
26 }
27
28 header h1 {
29   font-size: 1.5em;
30 }
```

Frontend:

### 2.2.4 Interações Dinâmicas no Formulário de Artigos

No formulário, foram aplicadas mensagens de erro dinâmicas e validações para melhorar a experiência do usuário.



The image shows a web interface for a blog. In the background, there is a large red gear graphic. The word "ATIVIDADE" is written in blue at the top of the gear. The word "COMENTARIOS" is written in green across the middle of the gear. The word "ADICIONE COMENTARIO:" is written in purple at the bottom of the gear. On the left side of the gear, there is a list of activities: "Atividade do dia foi bastante produtivo" followed by a bulleted list with "Editar" (in orange) and "Deletar" (in red). Below this, there are labels "Committer:" and "Comment:" followed by a text input field. Below the input field, there is a link "Apagar Comentário" in orange. In the foreground, there is a white comment form. The form has a label "Autor do comentário" above a text input field. Below that is a label "Comentário" above a larger text input field. Below the input fields is a button labeled "Criar comentário". Below the button is a label "Status" above a dropdown menu showing "public" with a downward arrow. At the bottom of the page, there is a footer that says "© 2024 Sua visita em nosso blog muito nos alegra!"

Como podemos ver na imagem, quando cadastrado e alongado na página do blogue pose ser fazer cometário e editá-los até mesmo excluí-los de uma forma bem pratica

## **CONCLUSÃO.**

O projeto foi majoritariamente criado no Visual Studio Code, empregando a linguagem Ruby e a estrutura Ruby on Rails para o lado do servidor. A gem Devise foi utilizada para autenticar os usuários. A configuração do banco de dados pode ser feita com alternativas como SQLite ou PostgreSQL. No lado do frontend, utilizamos HTML e CSS para organizar e personalizar a interface do blog, a execução do projeto requer as seguintes etapas:

Primeiramente, a preparação do ambiente Rails, seguida pela instalação da gem Devise para autenticação. Depois, os modelos e controllers necessários são criados, juntamente com a implementação das visualizações. Finalmente, o projeto é decorado com CSS no sistema para aprimorar a aparência visual.

Nos próximos passos, pretende-se melhorar a interface, incorporar funcionalidades avançadas de interação entre os usuários e introduzir novas técnicas de personalização e segurança, com o objetivo de tornar o blog ainda mais completo e funcional.

#### REFERÊNCIAS:

GUIDES RUBY ON RAILS. *Getting started with Rails*. Disponível em: [https://guides.rubyonrails.org/getting\\_started.html](https://guides.rubyonrails.org/getting_started.html). Acesso em: 22 dez. 2024.

YOUTUBE. *Ruby on Rails from scratch*. Disponível em: <https://www.youtube.com/watch?v=kbWbRx3-9IY>. Acesso em: 22 dez. 2024.

HEARTCOMBO. *Devise: flexible authentication solution for Rails with Warden*. GitHub, 2024. Disponível em: <https://github.com/heartcombo/devise>. Acesso em: 26 dez. 2024.