UNIVERSIDADE DE SÃO PAULO
ESCOLA POLITÉCNICA

PMR 3304 - Sistemas de Informação

Docentes: Marcos Tsuzuki, Luis de Sa, Daniela Damasceno, Agesinaldo Matos

Relatório Segunda Entrega de Laboratório

Gustavo Loiola dos Santos - 13681784

SÃO PAULO - SP

Novembro, 2024

SUMÁRIO

1. Introdução

Este relatório apresenta o desenvolvimento de uma aplicação CRUD (Create, Read, Update, Delete) utilizando o framework Django, com o objetivo de construir um blog dinâmico que permite a criação, edição e remoção de conteúdo diretamente pela interface da aplicação web. A proposta do projeto foi desenvolver uma plataforma de blog, site de notícias ou portfólio, onde os usuários podem gerenciar o conteúdo sem a necessidade de acessar o código fonte, proporcionando uma experiência mais acessível e amigável a usuários não técnicos.

O escopo da aplicação abrange funcionalidades adicionais, como um sistema de comentários, que possibilita aos leitores interagirem com o conteúdo publicado, e um sistema de categorização de posts, que organiza o conteúdo em diferentes categorias temáticas. Essas funcionalidades visam tornar a aplicação mais interativa e facilitar a navegação, promovendo uma experiência mais completa para os usuários.

Para o desenvolvimento, foi utilizado o framework Django, que fornece ferramentas robustas para a criação de aplicações web dinâmicas e escaláveis. Com o uso de modelos, views e templates do Django, foi possível construir uma estrutura modular e organizada, em que cada parte da aplicação possui uma função específica. O sistema de templates permitiu a criação de uma interface visualmente consistente, enquanto o uso de views genéricas e personalizadas facilitou a implementação das operações CRUD de forma eficiente e reutilizável.

Além da construção da aplicação, o projeto inclui o uso do Git para o versionamento de código. Cada funcionalidade foi desenvolvida em branches específicos, e posteriormente integrada ao branch principal, promovendo um fluxo de desenvolvimento organizado. Ao final, a aplicação foi implantada na plataforma Render, tornando-a acessível na web e permitindo a interação com usuários externos.

Este relatório descreve detalhadamente as etapas de desenvolvimento da aplicação, incluindo a criação dos modelos e views, a implementação das funcionalidades de comentários e categorização, e o processo de implantação. Adicionalmente, apresenta consultas SQL realizadas no banco de dados para extrair informações específicas, acompanhadas de exemplos e resultados. Este documento serve como um guia para o entendimento do processo de desenvolvimento da aplicação, bem como para a avaliação de seu funcionamento e das funcionalidades implementadas.

a. Links importantes

Link para o repositório no Github: Github - Blog Dinâmico com Django
Endereço da página hospedada no Render: Render - Blog Dinâmico

2. Objetivo

O objetivo deste projeto é desenvolver uma aplicação web dinâmica utilizando o framework Django, que permita a criação, leitura, edição e exclusão de conteúdo de forma prática e acessível para usuários finais, sem a necessidade de conhecimentos técnicos de programação. A aplicação visa ser uma plataforma de blog, site de notícias ou portfólio, onde o conteúdo é gerenciado diretamente pela interface da aplicação.

Para atender a esses objetivos, o projeto foi estruturado em três etapas principais:

1. **Implementação das Funcionalidades CRUD**: Criar uma estrutura de blog onde os usuários possam adicionar, visualizar, atualizar e remover posts. Cada post deve conter um título, conteúdo e data de postagem, armazenados no banco de dados e exibidos de maneira organizada na interface.
2. **Sistema de Comentários**: Adicionar uma funcionalidade que permita aos usuários interagirem com o conteúdo por meio de comentários. Esses comentários devem ser associados aos posts e exibidos em ordem cronológica inversa, possibilitando uma comunicação bidirecional entre o criador do conteúdo e os leitores.
3. **Categorização de Conteúdo**: Implementar um sistema de categorias para organizar os posts em temas específicos, facilitando a navegação e a busca de informações. Além disso, criar uma página de listagem de posts por categoria, ampliando a capacidade de organização do conteúdo.

Além disso, o projeto visa garantir que a aplicação esteja devidamente documentada e que o banco de dados seja configurado de forma a suportar consultas SQL específicas. Com isso, busca-se criar uma aplicação completa e escalável, com uma interface amigável e funcionalidades que promovam a interação e organização do conteúdo, tornando a experiência do usuário intuitiva e satisfatória. A aplicação final foi implantada na plataforma Render para facilitar o acesso e a interação de usuários externos.

3. Metodologia

Para o desenvolvimento deste projeto, a metodologia adotada seguiu uma abordagem incremental, dividida em três etapas principais, com o uso do framework Django. Cada etapa focou em um conjunto específico de funcionalidades, implementadas e testadas de forma independente antes de serem integradas. A seguir, são descritas as etapas e as decisões metodológicas tomadas para cada uma delas.

a. Implementação das Funcionalidades CRUD

Na primeira etapa, foi implementada a funcionalidade CRUD (Create, Read, Update, Delete) para o gerenciamento de notícias no site. Foi criado um modelo de dados chamado Noticia, com campos para o título, resumo, conteúdo, data de postagem e uma URL da notícia. O conteúdo de cada notícia foi configurado para ser armazenado em HTML, permitindo uma renderização direta no template com o uso do filtro safe. O campo data_postagem, configurado como DateTimeField, registrou automaticamente a data e a hora de criação da notícia.

Três versões das views foram desenvolvidas para essa funcionalidade, representando abordagens distintas de manipulação de dados. A primeira versão das views implementou as operações CRUD de forma simples, sem a utilização de Django forms e sem validação de dados. Na segunda versão, foram introduzidos os Django forms, proporcionando uma melhor validação de dados e simplificação dos formulários. Finalmente, a terceira versão utilizou Class-Based Views (CBVs) do Django, como ListView, DetailView, CreateView, UpdateView e DeleteView, resultando em uma estrutura de código mais eficiente e organizada. Para a navegação, foi criado um template base, com páginas para listagem, detalhamento, criação, edição e confirmação de remoção de notícias.

b. Adição de Sistema de Comentários

Na segunda etapa, foi implementado um sistema de comentários, permitindo que os usuários interajam diretamente nas páginas de notícias. Para isso, foi criado o modelo Comment, com os campos noticia (referenciando o modelo Noticia), autor (referenciando o modelo User do Django), texto, e data_postagem. O campo data_postagem foi configurado para registrar a data e hora do comentário automaticamente, e cada comentário foi associado a uma notícia e a um usuário específico.

A exibição dos comentários foi implementada na página individual de cada notícia, onde os comentários aparecem em ordem cronológica inversa, logo abaixo do conteúdo da notícia. Foi desenvolvida uma view de criação de comentários, com um formulário embutido na própria página de cada notícia, permitindo que usuários autenticados comentem diretamente. Um superusuário foi criado no Django Admin para facilitar a criação e o gerenciamento de usuários adicionais para simular diferentes interações.

c. Implementação de Categorias para as Notícias

Na terceira etapa, foi adicionado um sistema de categorização das notícias, permitindo que cada notícia fosse associada a uma ou mais categorias, facilitando a organização por temas. Para isso, o modelo Categoria foi criado com os campos nome e descricao. No modelo Noticia, foi adicionada uma relação many-to-many com Categoria, permitindo que uma notícia pudesse pertencer a múltiplas categorias. A implementação foi feita em um branch chamado categories, que foi criado após a fusão das funcionalidades de comentários no branch principal.

Duas views foram desenvolvidas para o sistema de categorias: uma para listar todas as categorias e outra para exibir as notícias de uma categoria específica. Na página individual de cada notícia, todas as categorias associadas à notícia foram listadas com links que direcionam o usuário para a página de cada categoria. O template de listagem de notícias foi reutilizado para a visualização individual de categorias, com ajustes no cabeçalho para exibir o nome da categoria selecionada.

d. População do Banco de Dados e Consultas SQL

Após a implementação de todas as funcionalidades, o banco de dados db.sqlite3 foi configurado para armazenar as informações de notícias, categorias e comentários, possibilitando a execução de consultas SQL para a extração de dados específicos. O banco de dados foi populado com dados fictícios, incluindo várias notícias, categorias e comentários, com o objetivo de simular o uso real do sistema. Cada notícia foi associada a uma ou mais categorias, e algumas foram comentadas por diferentes usuários.

Foram então elaboradas consultas SQL para a obtenção de informações específicas, como a listagem de todas as notícias ordenadas por data, a extração de todos os comentários de uma notícia específica, a visualização dos comentários com informações complementares da notícia associada, a listagem de todas as notícias de uma categoria específica, e a identificação de

categorias que possuíam duas ou mais notícias associadas. Essas consultas foram realizadas no banco de dados populado, verificando a correta extração e organização dos dados.

e. Controle de Versão e Implantação

Finalmente, o projeto foi versionado usando Git e hospedado em um repositório no GitHub, conforme boas práticas de controle de versão. Após testes locais, a aplicação foi implantada na plataforma Render, permitindo o acesso público ao sistema e possibilitando que terceiros testassem as funcionalidades desenvolvidas. Esta implantação consolidou o projeto como uma aplicação web funcional e acessível online.

4. Resultados

a. Implementação do CRUD de Notícias

Na primeira etapa do projeto, foi desenvolvida uma aplicação CRUD para gerenciar notícias no site. O modelo Noticia foi utilizado para armazenar as informações de cada notícia, como título, resumo, conteúdo, data de postagem e URL da imagem. A aplicação inclui uma página de listagem de notícias, onde cada título é um link para a página individual da respectiva notícia. O CRUD foi implementado em três versões diferentes para as views: uma com views funcionais sem Django forms, outra com views funcionais utilizando forms, e finalmente uma versão com views baseadas em classes (CBVs) utilizando ListView, DetailView, CreateView, UpdateView e DeleteView.

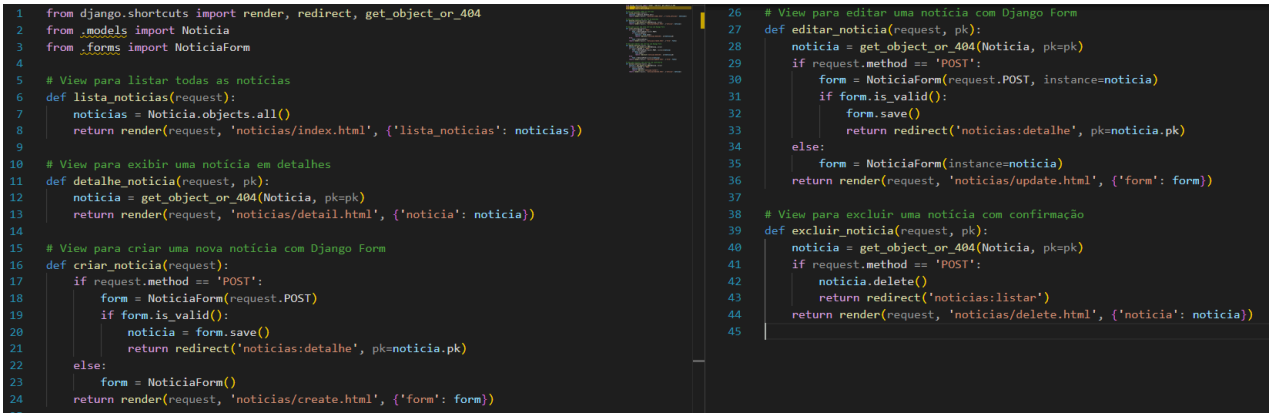
- Versão 1: Views funcionais sem Django forms.

```python
from django.shortcuts import render, redirect, get_object_or_404
from .models import Noticia

# View para listar todas as notícias
def lista_noticias(request):
    noticias = Noticia.objects.all()
    return render(request, 'noticias/index.html', {'lista_noticias': noticias})

# View para exibir uma notícia em detalhes
def detalhe_noticia(request, pk):
    noticia = get_object_or_404(Noticia, pk=pk)
    return render(request, 'noticias/detail.html', {'noticia': noticia})

# View para criar uma nova notícia
def criar_noticia(request):
    if request.method == 'POST':
        titulo = request.POST.get('titulo')
        resumo = request.POST.get('resumo')
        conteudo = request.POST.get('conteudo')
        noticia_url = request.POST.get('noticia_url')
        noticia = Noticia.objects.create(
            titulo=titulo,
            resumo=resumo,
            conteudo=conteudo,
            noticia_url=noticia_url
        )
        return redirect('noticias:detalhe', pk=noticia.pk)
    return render(request, 'noticias/create.html')
```

```python
# View para editar uma notícia
def editar_noticia(request, pk):
    noticia = get_object_or_404(Noticia, pk=pk)
    if request.method == 'POST':
        noticia.titulo = request.POST.get('titulo')
        noticia.resumo = request.POST.get('resumo')
        noticia.conteudo = request.POST.get('conteudo')
        noticia.noticia_url = request.POST.get('noticia_url')
        noticia.save()
        return redirect('noticias:detalhe', pk=noticia.pk)
    return render(request, 'noticias/update.html', {'noticia': noticia})

# View para excluir uma notícia com confirmação
def excluir_noticia(request, pk):
    noticia = get_object_or_404(Noticia, pk=pk)
    if request.method == 'POST':
        noticia.delete()
        return redirect('noticias:listar')
    return render(request, 'noticias/delete.html', {'noticia': noticia})

```

Figura 1: Código versão 1 Views

- Versão 2: Views funcionais utilizando Django forms.

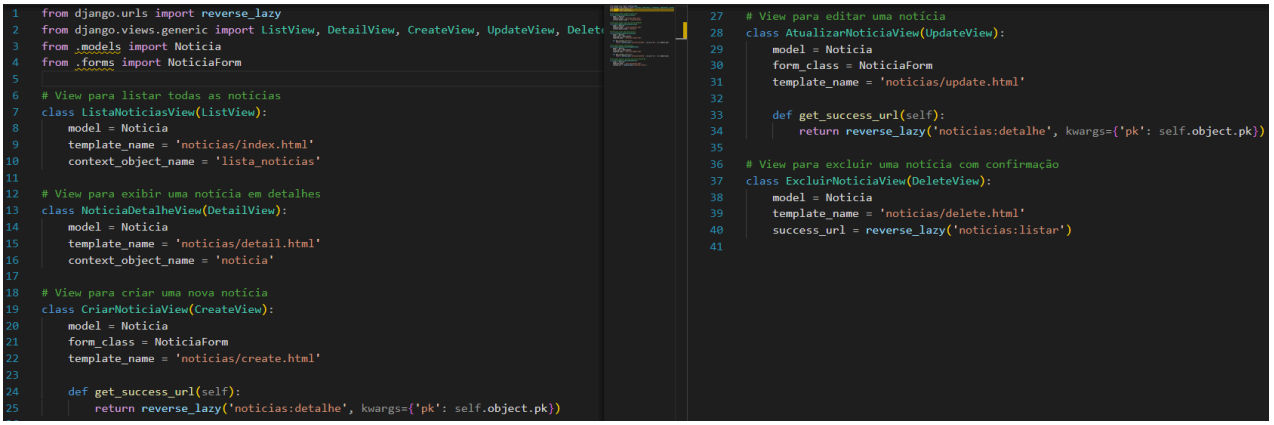


```python
from django.shortcuts import render, redirect, get_object_or_404
from .models import Noticia
from .forms import NoticiaForm

# View para listar todas as notícias
def lista_noticias(request):
    noticias = Noticia.objects.all()
    return render(request, 'noticias/index.html', {'lista_noticias': noticias})

# View para exibir uma notícia em detalhes
def detalhe_noticia(request, pk):
    noticia = get_object_or_404(Noticia, pk=pk)
    return render(request, 'noticias/detail.html', {'noticia': noticia})

# View para criar uma nova notícia com Django Form
def criar_noticia(request):
    if request.method == 'POST':
        form = NoticiaForm(request.POST)
        if form.is_valid():
            noticia = form.save()
            return redirect('noticias:detalhe', pk=noticia.pk)
    else:
        form = NoticiaForm()
    return render(request, 'noticias/create.html', {'form': form})
```

```python
# View para editar uma notícia com Django Form
def editar_noticia(request, pk):
    noticia = get_object_or_404(Noticia, pk=pk)
    if request.method == 'POST':
        form = NoticiaForm(request.POST, instance=noticia)
        if form.is_valid():
            form.save()
            return redirect('noticias:detalhe', pk=noticia.pk)
    else:
        form = NoticiaForm(instance=noticia)
    return render(request, 'noticias/update.html', {'form': form})

# View para excluir uma notícia com confirmação
def excluir_noticia(request, pk):
    noticia = get_object_or_404(Noticia, pk=pk)
    if request.method == 'POST':
        noticia.delete()
        return redirect('noticias:listar')
    return render(request, 'noticias/delete.html', {'noticia': noticia})
```

Figura 2: Código versão 2 Views

- Versão 3: Views utilizando CBVs (ListView, DetailView, CreateView, UpdateView e DeleteView).



```python
from django.urls import reverse_lazy
from django.views.generic import ListView, DetailView, CreateView, UpdateView, Delet
from .models import Noticia
from .forms import NoticiaForm

# View para listar todas as notícias
class ListaNoticiasView(ListView):
    model = Noticia
    template_name = 'noticias/index.html'
    context_object_name = 'lista_noticias'

# View para exibir uma notícia em detalhes
class NoticiaDetalheView(DetailView):
    model = Noticia
    template_name = 'noticias/detail.html'
    context_object_name = 'noticia'

# View para criar uma nova notícia
class CriarNoticiaView(CreateView):
    model = Noticia
    form_class = NoticiaForm
    template_name = 'noticias/create.html'

    def get_success_url(self):
        return reverse_lazy('noticias:detalhe', kwargs={'pk': self.object.pk})
```

```python
# View para editar uma notícia
class AtualizarNoticiaView(UpdateView):
    model = Noticia
    form_class = NoticiaForm
    template_name = 'noticias/update.html'

    def get_success_url(self):
        return reverse_lazy('noticias:detalhe', kwargs={'pk': self.object.pk})

# View para excluir uma notícia com confirmação
class ExcluirNoticiaView(DeleteView):
    model = Noticia
    template_name = 'noticias/delete.html'
    success_url = reverse_lazy('noticias:listar')
```

Figura 3: Código versão 3 Views

Capturas de Tela:

- Página de Listagem de Notícias

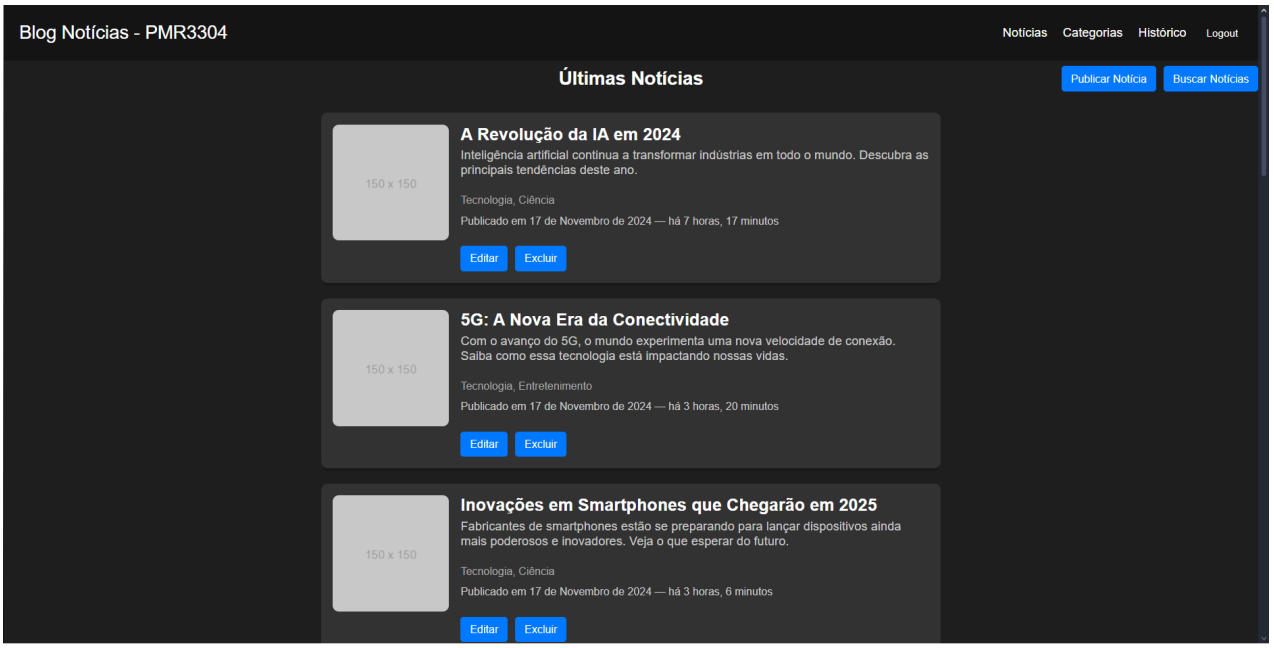


Figura 4: Página Visível Listagem Notícias

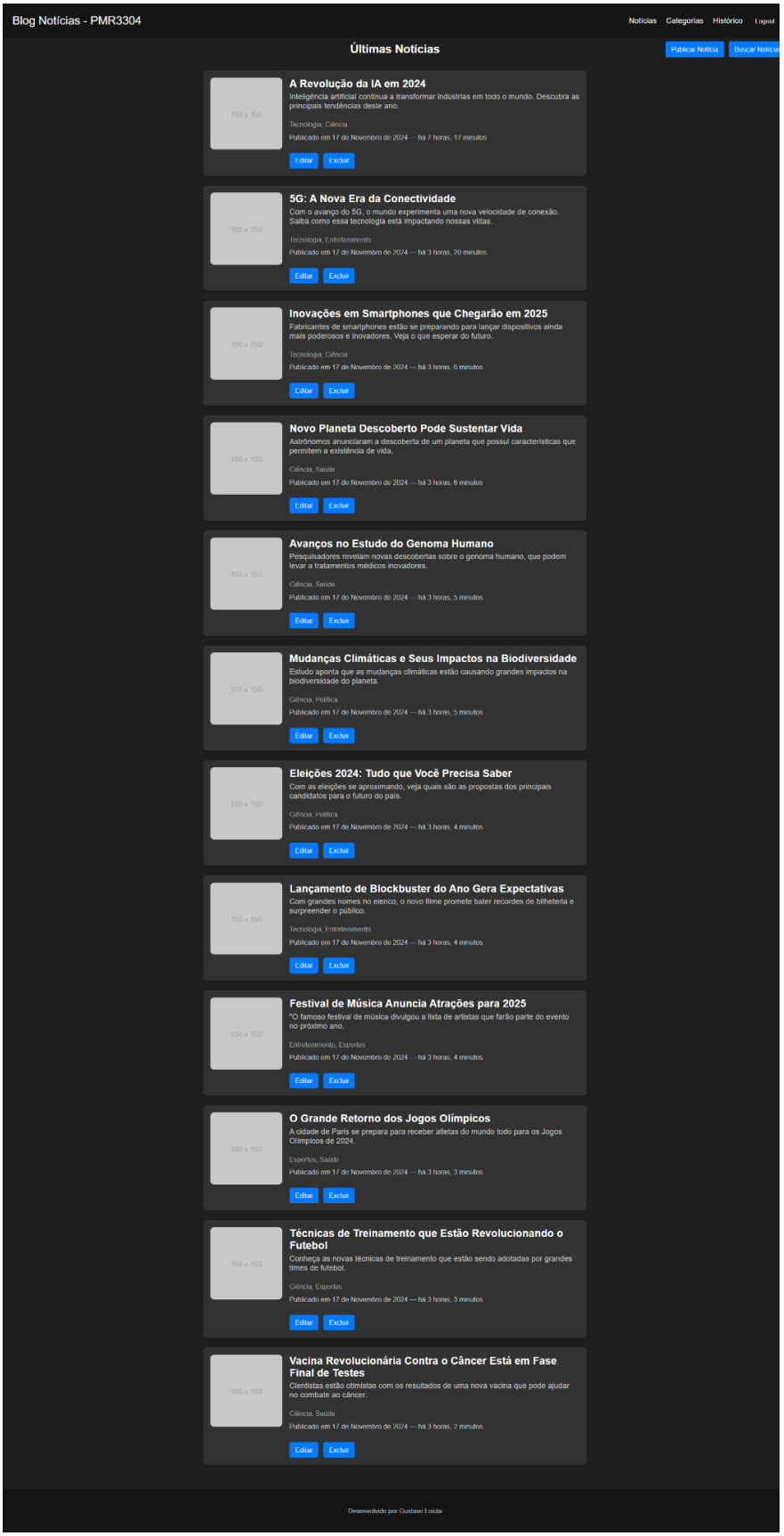


Figura 5: Página Inteira Listagem Notícias

- Página Individual de Post

Figura 6: Página Visível Individual Notícia

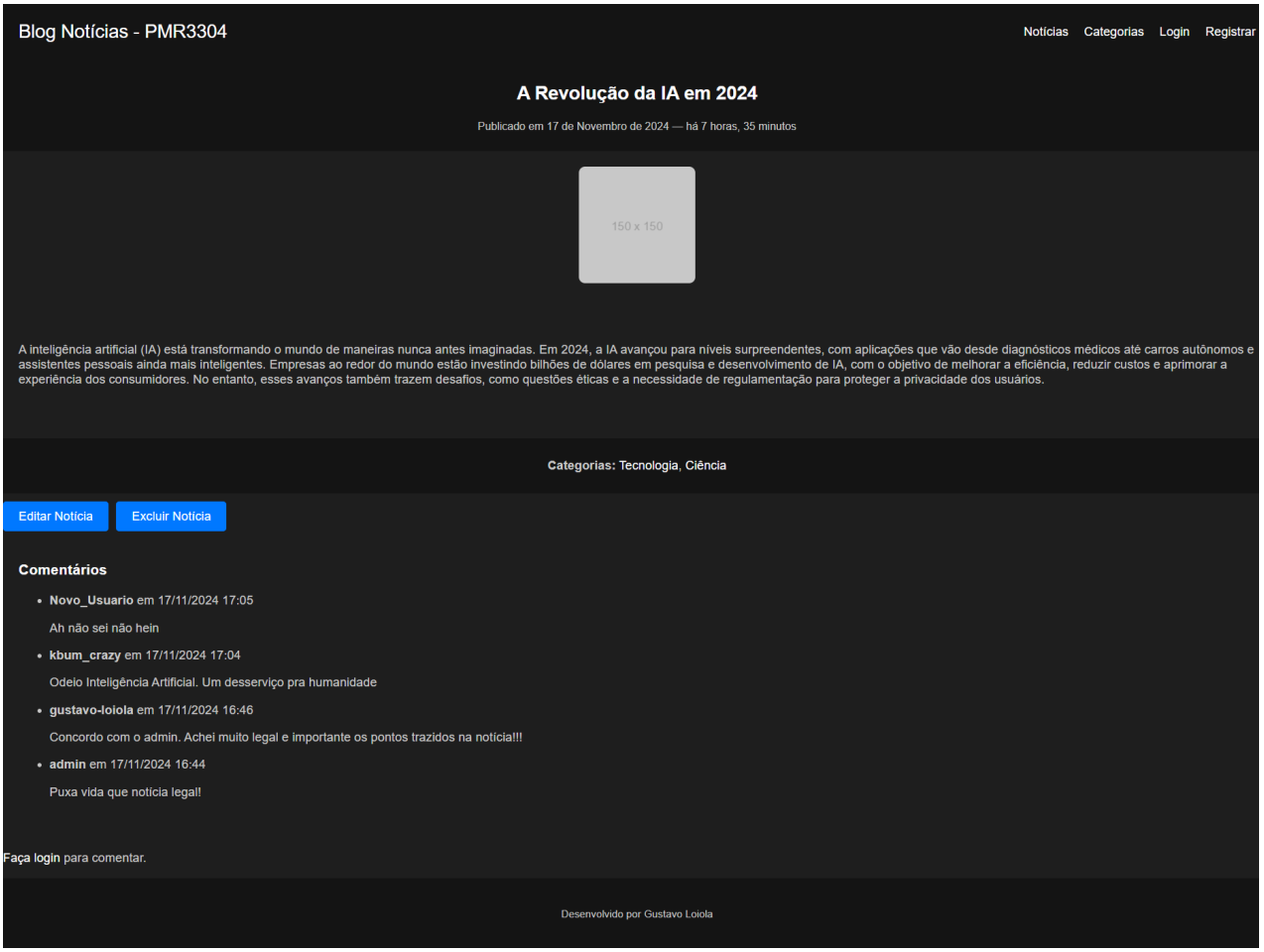


Figura 7: Página Inteira Individual Notícia

- Página de Criação de Post

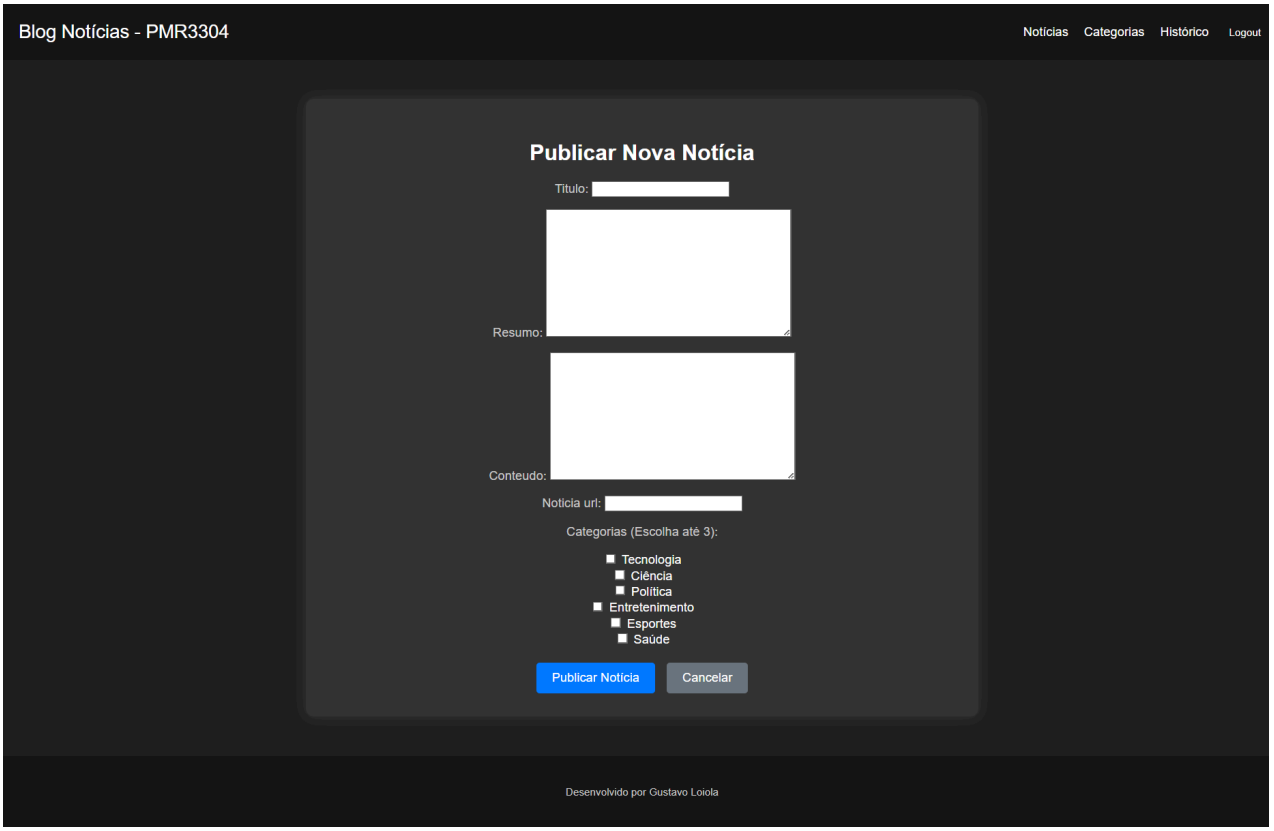


Figura 8: Página Inteira Publicar Notícia

- Página de Edição de Post

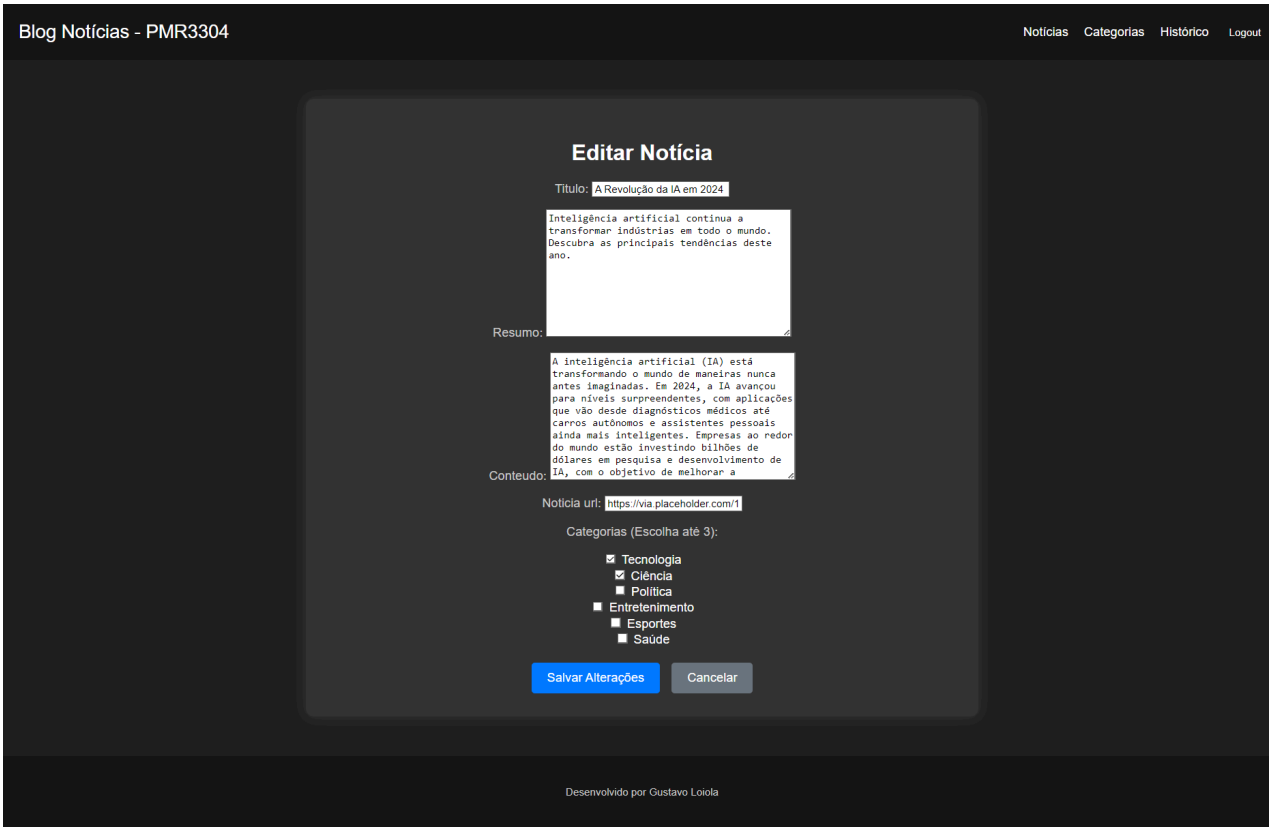


Figura 8: Página Inteira Editar Notícia

- Página de Confirmação de Remoção de Post

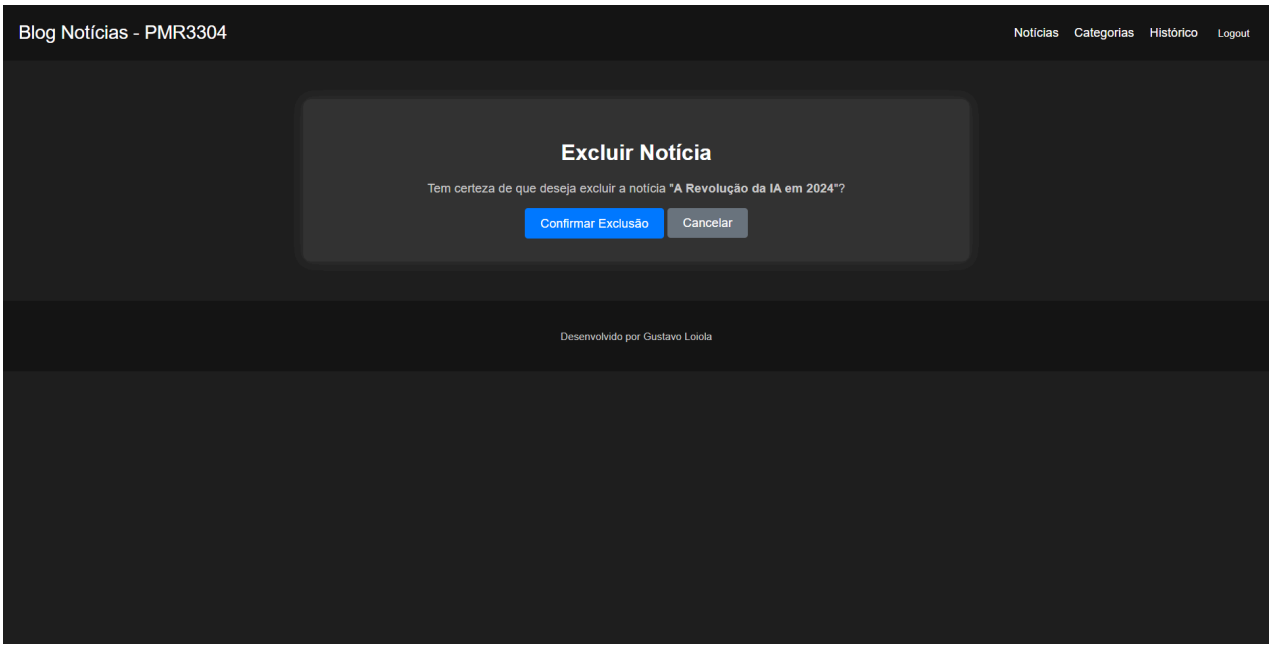


Figura 9: Página Inteira Excluir Notícia

b. Implementação do Sistema de Comentários

Para a segunda etapa, foi adicionado um sistema de comentários, permitindo que os usuários autenticados comentem diretamente na página de detalhes de cada post. O modelo Comment foi utilizado para armazenar o autor, texto e data de cada comentário. A view de criação de comentários permite que o usuário poste um comentário associado a uma notícia específica. Os comentários são exibidos abaixo do conteúdo do post, ordenados do mais recente ao mais antigo.

```
26  # Model de Comentário
    You, 9 hours ago | 1 author (You)
27  class Comment(models.Model):
28      noticia = models.ForeignKey(Noticia, on_delete=models.CASCADE, related_name='comments')
29      autor = models.ForeignKey(User, on_delete=models.CASCADE)
30      texto = models.TextField()
31      data_postagem = models.DateTimeField(auto_now_add=True)
32
33      def __str__(self):
34          return f'Comentário de {self.autor} na notícia {self.noticia}'
```

Figura 10: Código do Modelo Comment

```python
# View para exibir detalhes de uma notícia e registrar o acesso no histórico
You, 7 hours ago | 1 author (You)
class NoticiaDetalheView(DetailView):
    model = Noticia
    template_name = 'noticias/detail.html'
    context_object_name = 'noticia'

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        context['comments'] = Comment.objects.filter(noticia=self.object).order_by('-data_postagem')
        context['form'] = CommentForm()
        return context

    def dispatch(self, request, *args, **kwargs):
        if request.user.is_authenticated:
            noticia = self.get_object()
            HistoricoAcesso.objects.create(
                usuario=request.user,
                noticia=noticia,
                url=self.request.build_absolute_uri(reverse('noticias:detalhe', kwargs={'pk': noticia.pk}))
            )
        return super().dispatch(request, *args, **kwargs)

    def post(self, request, *args, **kwargs):
        self.object = self.get_object()
        form = CommentForm(request.POST)
        if form.is_valid():
            comentario = form.save(commit=False)
            comentario.noticia = self.object
            comentario.autor = request.user
            comentario.save()
            messages.success(request, "Comentário adicionado com sucesso!")
            return redirect('noticias:detalhe', pk=self.object.pk)
        return self.get(request, *args, **kwargs)
```

Figura 11: Código da View de Detalhamento da Notícia com uso do modelo Comment e Forms

```python
class CommentForm(forms.ModelForm):
    You, yesterday | 1 author (You)
    class Meta:
        model = Comment
        fields = ['texto']
        labels = {'texto': 'Comentário'}
        widgets = {
            'texto': forms.Textarea(attrs={'rows': 3, 'placeholder': 'Escreva seu comentário...'}),
        }
```

Figura 12: Código do CommentForm

Capturas de Tela:

- Página Individual de Post com Comentários

  Uma observação interessante aqui nesta tela é que, por não estar logado com um usuário, no canto inferior esquerdo é dada a opção de fazer login para comentar. Caso o usuário já esteja logado, a seção de comentários aparece como na figura 14.

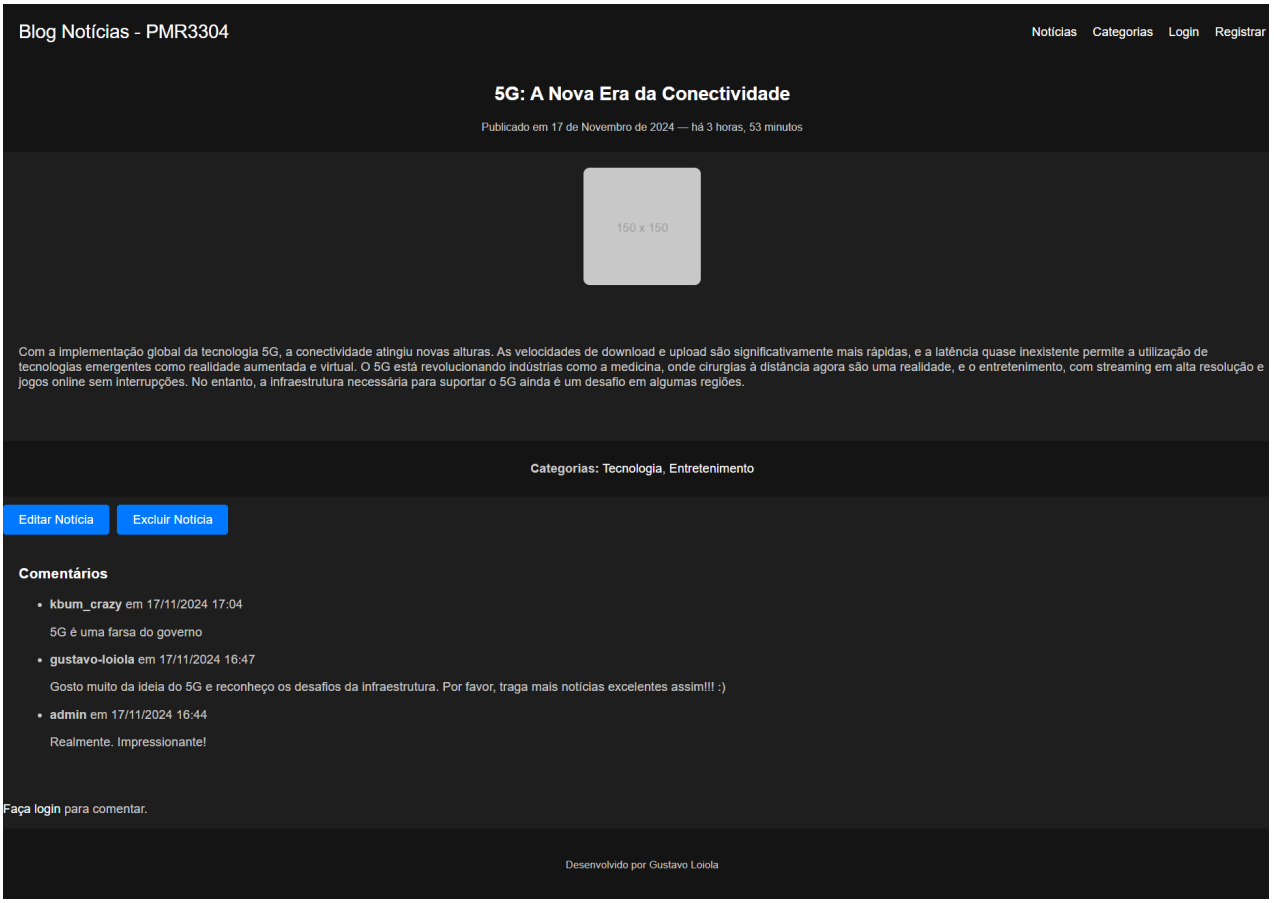


Figura 13: Página Inteira de Notícia com Comentários (deslogado)

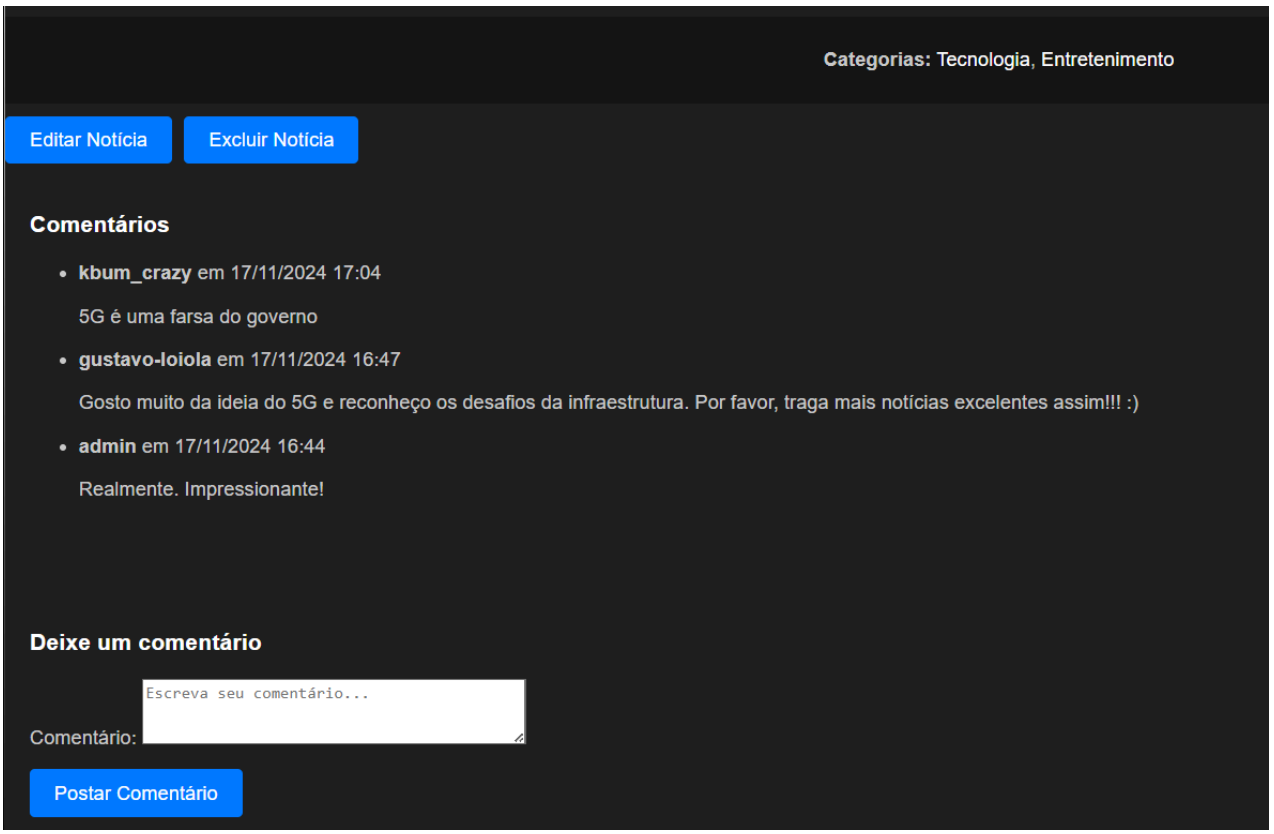


Figura 14: Seção de Comentários (logado)

- Página de Criação de Comentários

Criação de comentários é feita na própria página da notícia e só é permitido com usuário logado. Visualização dessa parte está na figura 14 acima.

c. Implementação de Categorias para Notícias

Na terceira etapa, foi implementado um sistema de categorização de notícias. Cada notícia pode ser associada a múltiplas categorias. O modelo Categoria foi criado com os campos de nome e descrição para armazenar as informações de cada categoria. Foram desenvolvidas views para listar todas as categorias e exibir os posts pertencentes a uma categoria específica. Na página individual de cada notícia, são exibidas as categorias às quais ela pertence, com links para as páginas individuais dessas categorias.

```python
# Model de Categoria
You, yesterday | 1 author (You)
class Categoria(models.Model):
    nome = models.CharField(max_length=50, unique=True)
    descricao = models.TextField(blank=True, null=True)

    def __str__(self):
        return self.nome
```

Figura 15: Código do modelo Categoria

```python
# View para listagem de categorias
You, 7 hours ago | 1 author (You)
class CategoriaListView(ListView):
    model = Categoria
    template_name = 'noticias/categoria_list.html'
    context_object_name = 'categorias'

# View individual de categoria para listar os posts da categoria
You, 7 hours ago | 1 author (You)
class CategoriaDetailView(DetailView):
    model = Categoria
    template_name = 'noticias/categoria_detail.html'
    context_object_name = 'categoria'

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        context['posts'] = Noticia.objects.filter(categorias=self.object)
        return context
```

Figura 16: Código das Views de Categoria

Capturas de Tela:

- Página Individual de Post Mostrando as Categorias

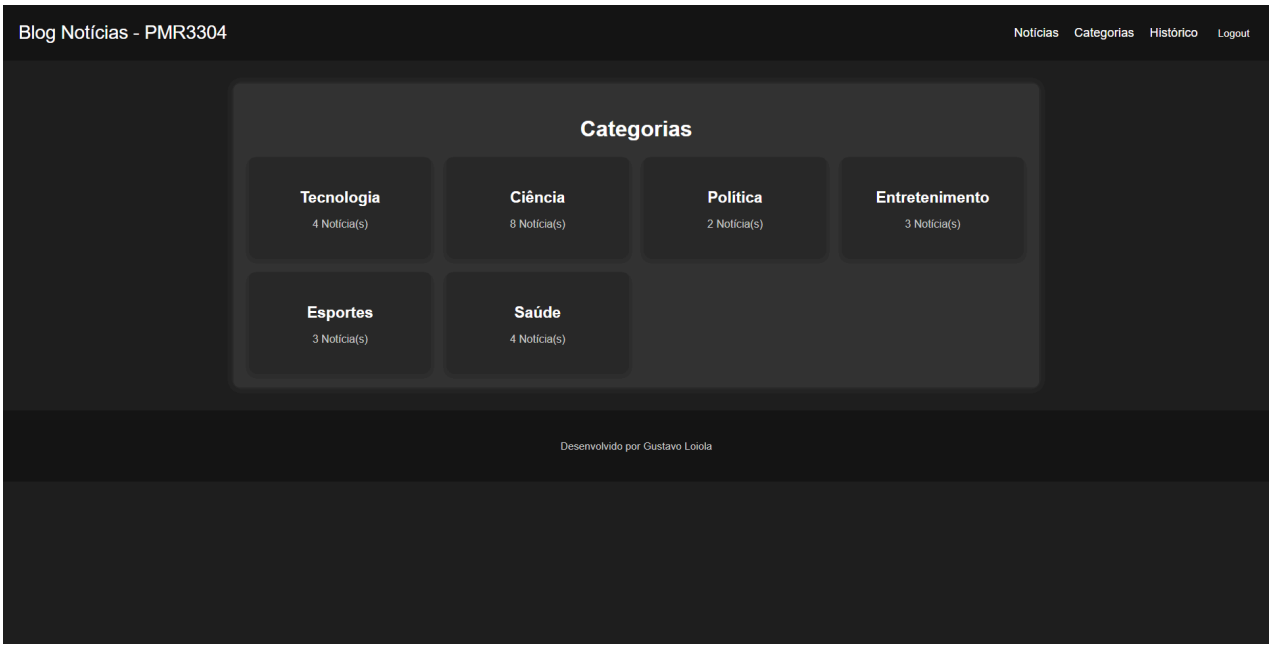


Figura 17: Página Inteira da Listagem de Categorias

- Página Individual de Categoria com Todos os Posts da Categoria

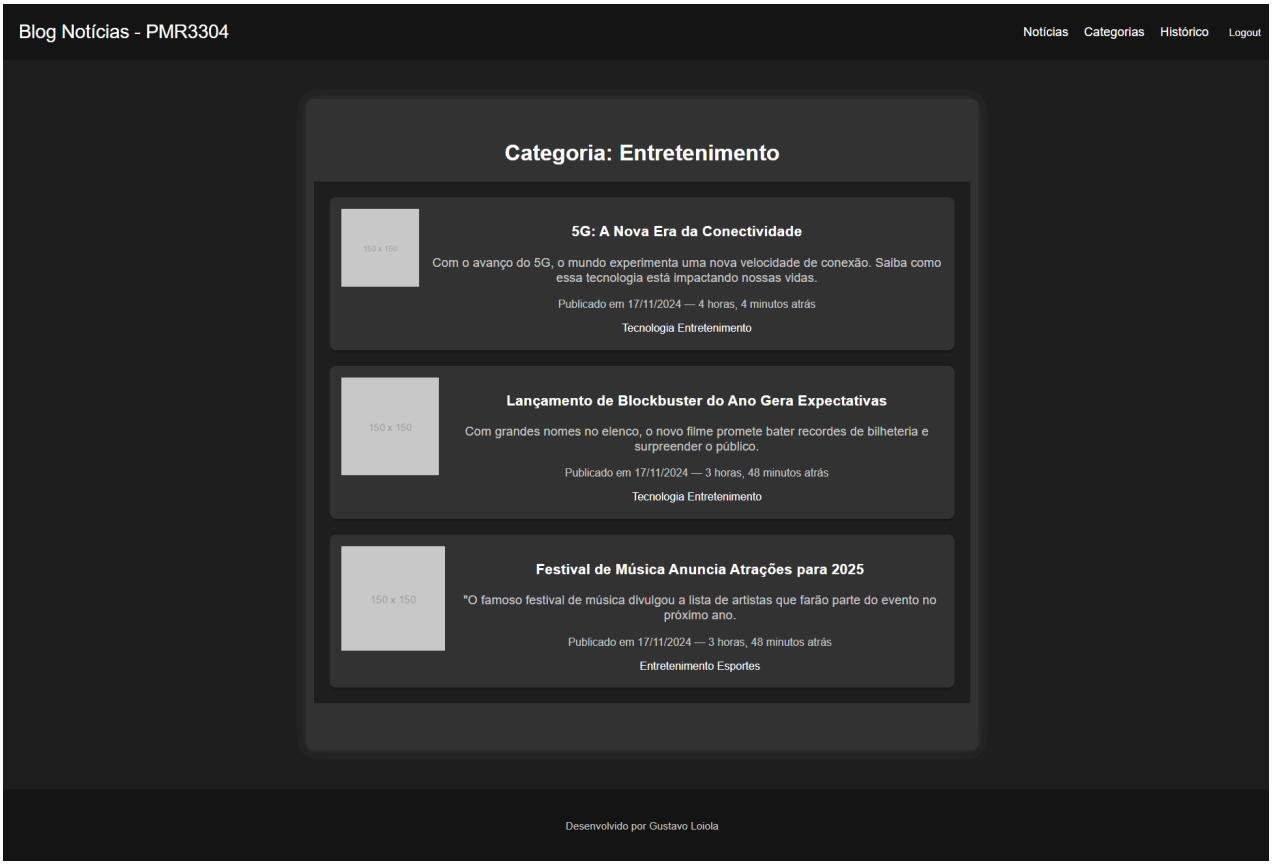


Figura 18: Página Inteira de Notícias da Categoria Entretenimento

d. Implementação de Histórico de Acesso

Para melhorar a experiência do usuário, foi implementado um sistema de histórico de acessos. Cada vez que um usuário autenticado acessa uma notícia, o sistema registra automaticamente a data, a URL e o título da notícia no modelo HistoricoAcesso. O histórico é exibido na página de histórico do usuário, mostrando todas as notícias acessadas em ordem cronológica decrescente. A view HistoricoAcessoView foi implementada para listar os acessos do usuário logado, e o template foi adaptado para exibir os detalhes de cada acesso.

```
114  # View para exibir o histórico de notícias acessadas pelo usuário
     You, 8 hours ago | 1 author (You)
115  class HistoricoAcessoView(LoginRequiredMixin, ListView):
116      model = HistoricoAcesso
117      template_name = 'noticias/historico.html'
118      context_object_name = 'historico_acesso'
119      login_url = 'noticias:login'
120
121      def get_queryset(self):
122          return HistoricoAcesso.objects.filter(usuario=self.request.user).order_by('-data_acesso')
```

Figura 19: Código View HistoricoAcessoView

Captura de Tela:

Figura 20: Página de Histórico de Acessos do Usuário

e. Implementação de Histórico de Buscas

Além do histórico de acessos, foi implementado um sistema de histórico de buscas, que registra cada termo pesquisado por um usuário autenticado no modelo HistoricoBusca. Na view busca_noticias, o termo de busca é salvo no histórico sempre que uma pesquisa é realizada. Este histórico é exibido na página de busca, permitindo ao usuário ver suas buscas anteriores. Caso o usuário não tenha realizado nenhuma busca, uma mensagem é exibida indicando a ausência de histórico.

```
52  # View para buscar notícias
53  def busca_noticias(request):
54      query = request.GET.get('query', '')
55      noticias_encontradas = Noticia.objects.filter(titulo__icontains=query) if query else Noticia.objects.all()
56
57      # Salva o termo de busca no histórico do usuário autenticado
58      if query and request.user.is_authenticated:
59          HistoricoBusca.objects.create(usuario=request.user, termo=query)
60
61      # Obtém o histórico de buscas do usuário autenticado
62      historico_buscas = (
63          HistoricoBusca.objects.filter(usuario=request.user).order_by('-data_busca')
64          if request.user.is_authenticated else []
65      )
66
67      context = {
68          'lista_noticias': noticias_encontradas,
69          'query': query,
70          'historico_buscas': historico_buscas,
71      }
72      return render(request, 'noticias/search.html', context)
```

Figura 21: Código da view de histórico de buscas

Captura de Tela:

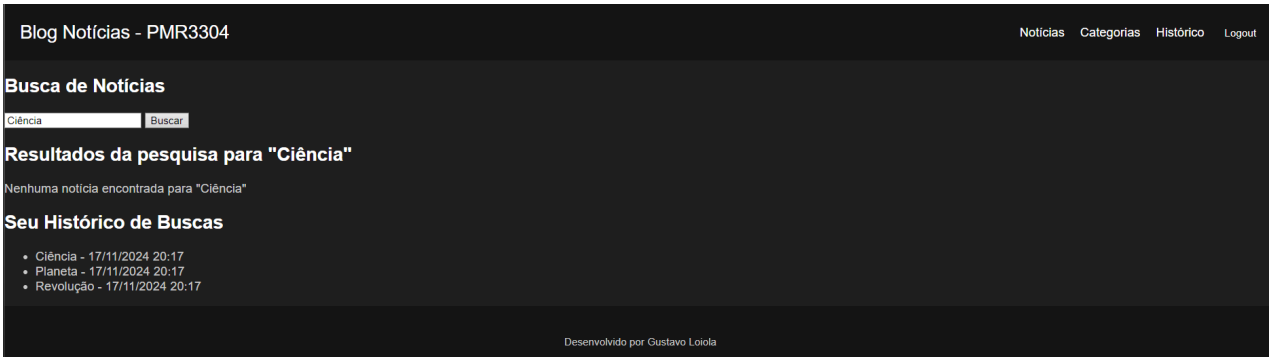


Figura 22: Página de Busca com Histórico de Buscas do Usuário

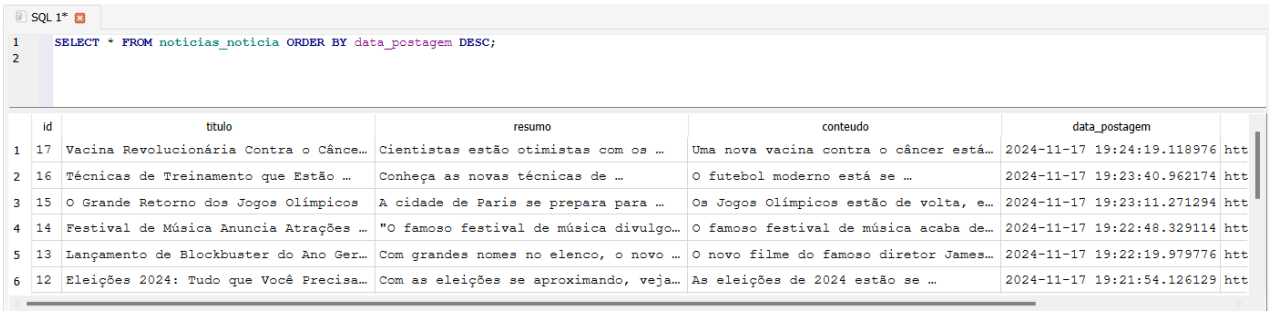
f. Exercícios com SQL

Após o desenvolvimento do sistema de notícias, comentários, categorias e históricos, o banco de dados foi preenchido com os seguintes dados:

- Doze posts no total, que podem ser editados ou removidos.
- Há posts sem comentários e outros com quantidade variando entre um e cinco comentários.
- Seis categorias foram criadas, cada post associado a pelo menos uma categoria.

Consultas SQL foram realizadas para extrair dados do banco db.sqlite3. As queries e os respectivos resultados estão apresentados a seguir:

- **Consulta 1**: Todos os posts, ordenados por data, sendo o mais recente primeiro



SQL 1*

```
SELECT * FROM noticias_noticia ORDER BY data_postagem DESC;
```

	id	titulo	resumo	conteudo	data_postagem	
1	17	Vacina Revolucionária Contra o Câncer…	Cientistas estão otimistas com os …	Uma nova vacina contra o câncer está…	2024-11-17 19:24:19.118976	htt
2	16	Técnicas de Treinamento que Estão …	Conheça as novas técnicas de …	O futebol moderno está se …	2024-11-17 19:23:40.962174	htt
3	15	O Grande Retorno dos Jogos Olímpicos	A cidade de Paris se prepara para …	Os Jogos Olímpicos estão de volta, e…	2024-11-17 19:23:11.271294	htt
4	14	Festival de Música Anuncia Atrações …	"O famoso festival de música divulgo…	O famoso festival de música acaba de…	2024-11-17 19:22:48.329114	htt
5	13	Lançamento de Blockbuster do Ano Ger…	Com grandes nomes no elenco, o novo …	O novo filme do famoso diretor James…	2024-11-17 19:22:19.979776	htt
6	12	Eleições 2024: Tudo que Você Precisa…	Com as eleições se aproximando, veja…	As eleições de 2024 estão se …	2024-11-17 19:21:54.126129	htt

Figura 23: Query e Resultados da Consulta 1

- **Consulta 2**: Para um post com comentários, todos os seus comentários

SQL 1*

```
SELECT * FROM noticias_comment WHERE noticia_id = 6;
```

	id	texto	data_postagem	autor_id	noticia_id
1	4	Puxa vida que notícia legal!	2024-11-17 19:44:16.548833	1	6
2	6	Concordo com o admin. Achei muito …	2024-11-17 19:46:52.316033	4	6
3	8	Odeio Inteligência Artificial. Um …	2024-11-17 20:04:06.948590	5	6
4	10	Ah não sei não hein	2024-11-17 20:05:31.586101	6	6

Figura 24: Query e Resultados da Consulta 2

- **Consulta 3**: Para um post com comentários, todos os seus comentários, incluindo título e data de publicação

```
SELECT noticias_noticia.titulo, noticias_comment.texto, noticias_comment.data_postagem
FROM noticias_comment
JOIN noticias_noticia ON noticias_comment.noticia_id = noticias_noticia.id
WHERE noticias_noticia.id = 6
```

	titulo	texto	data_postagem
1	A Revolução da IA em 2024	Puxa vida que notícia legal!	2024-11-17 19:44:16.548833
2	A Revolução da IA em 2024	Concordo com o admin. Achei muito …	2024-11-17 19:46:52.316033
3	A Revolução da IA em 2024	Odeio Inteligência Artificial. Um …	2024-11-17 20:04:06.948590
4	A Revolução da IA em 2024	Ah não sei não hein	2024-11-17 20:05:31.586101

Figura 25: Query e Resultados da Consulta 3

- **Consulta 4**: Para uma categoria, todos os posts pertencentes a ela, incluindo colunas de categoria e post

```
SELECT noticias_categoria.nome, noticias_noticia.titulo
FROM noticias_categoria
JOIN noticias_noticia_categorias ON noticias_categoria.id = noticias_noticia_categorias.categoria_id
JOIN noticias_noticia ON noticias_noticia.id = noticias_noticia_categorias.noticia_id
WHERE noticias_categoria.nome = 'Tecnologia'
```

	nome	titulo
1	Tecnologia	5G: A Nova Era da Conectividade
2	Tecnologia	A Revolução da IA em 2024
3	Tecnologia	Inovações em Smartphones que Chegarã…
4	Tecnologia	Lançamento de Blockbuster do Ano Ger…

Figura 26: Query e Resultados da Consulta 4

- **Consulta 5**: Todas as categorias que possuem dois ou mais posts

```sql
SELECT noticias_categoria.nome, COUNT(noticias_noticia_categorias.noticia_id) AS num_posts
FROM noticias_categoria
JOIN noticias_noticia_categorias ON noticias_categoria.id = noticias_noticia_categorias.categoria_id
GROUP BY noticias_categoria.nome
HAVING COUNT(noticias_noticia_categorias.noticia_id >= 2);
```

	nome	num_posts
1	Ciência	8
2	Entretenimento	3
3	Esportes	3
4	Política	2
5	Saúde	4
6	Tecnologia	4

Figura 27: Query e Resultados da Consulta 5

g. Repositório GitHub e Implantação no Render

Após o término do desenvolvimento, o código foi versionado e armazenado em um repositório GitHub. A aplicação foi então implantada na plataforma Render, tornando-a acessível publicamente.

- Link para o repositório no Github: Github - Blog Dinâmico com Django
- Endereço da página hospedada no Render: Render - Blog Dinâmico