Luiza Ramos Pascuotte

Relatório de Auditoria de Qualidade e Testes de Software

1. Análise Estática do Código Fonte

1.1 Boas Práticas de Programação

O código fonte avaliado segue boas práticas de programação, demonstrando um padrão consistente na nomenclatura das propriedades, tornando o código de fácil leitura e manutenção.

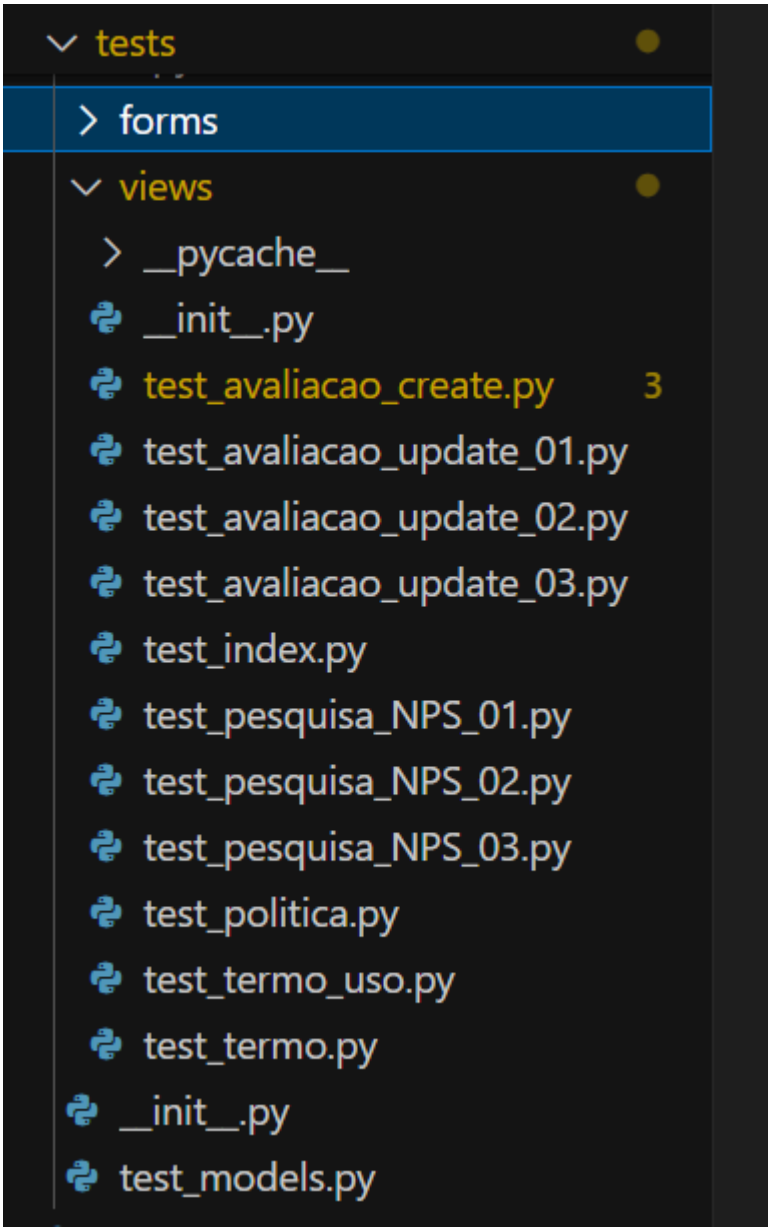
```python
class AvaliacaoModel(models.Model):
    user = models.ForeignKey(get_user_model(), on_delete=models.CASCADE)
    slug = models.SlugField(max_length=200, default='')
    nome_pesquisa = models.CharField(default='pesquisa_de_satisfação', blank=False,
                                     max_length=40)
    titulo = models.CharField(default='Pesquisa de Satisfação', blank=False,
                                     max_length=40)
    nota_explicativa = models.CharField(default='O que achou de nosso atendimento ?', blank=False,
                                     max_length=250)
    mostrar_pesquisa = models.BooleanField(default=True)
    opcao_nota_5 = models.CharField(default='Muito Satisfeito', blank=False, max_length=40)
    opcao_nota_4 = models.CharField(default='Satisfeito', blank=False, max_length=40)
    opcao_nota_3 = models.CharField(default='Indiferente', blank=False, max_length=40)
    opcao_nota_2 = models.CharField(default='Insatisfeito', blank=False, max_length=40)
    opcao_nota_1 = models.CharField(default='Muito Insatisfeito', blank=False, max_length=40)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
```

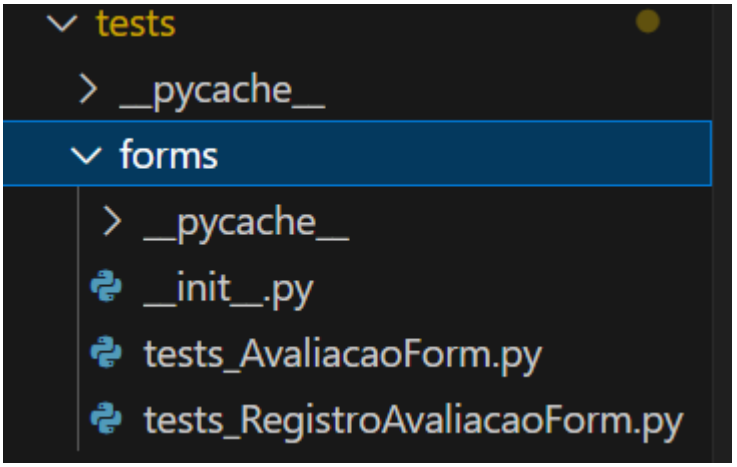
No entanto, observou-se que, em algumas partes do código (particularmente em views), algumas variáveis apresentam nomenclaturas menos claras, como "tu" e "pp". Embora essas variáveis possam ser identificadas com a leitura das requisições, recomenda-se que os nomes sejam mais explícitos, de modo a melhorar a legibilidade do código.

```python
def perfil_completo(function=None, redirect_url='user_edit_name'):
    def wrap(request, *args, **kwargs):
        if request.user.is_authenticated:
            tu = request.user.aceitou_termos_uso
            pp = request.user.aceitou_politica_privacidade
            completo = request.user.perfil_preenchido
            if tu and pp and completo:
                return function(request, *args, **kwargs)
        return redirect(redirect_url)
    wrap.__doc__ = function.__doc__
    wrap.__name__ = function.__name__
    return wrap

def index(request):
    return render(request, "index.html")

def pesquisa_NPS_01(request, slug):
    pesquisa = get_object_or_404(AvaliacaoModel, slug = slug)
    utm_source = request.GET.get('utm_source','unknow')
    if pesquisa.mostrar_pesquisa:
        context = {'form': pesquisa, 'utm_source':utm_source}
        return render(request, "pesquisaNPS.html", context)
    return redirect('core:index')
```

1.2 Testes Automatizados

A análise identificou que o projeto conta com testes automatizados bem organizados, separados por views e formulários. Essa separação facilita a localização e a manutenção dos testes. Além disso, os testes vão além de verificações simples de respostas HTTP, contemplando fluxos de uso completos, como a criação de um usuário e a execução de uma avaliação, o que contribui para uma cobertura mais eficaz do sistema.


tests
forms
views
__pycache__
__init__.py
test_avaliacao_create.py 3
test_avaliacao_update_01.py
test_avaliacao_update_02.py
test_avaliacao_update_03.py
test_index.py
test_pesquisa_NPS_01.py
test_pesquisa_NPS_02.py
test_pesquisa_NPS_03.py
test_politica.py
test_termo_uso.py
test_termo.py
__init__.py
test_models.py


tests
__pycache__
forms
__pycache__
__init__.py
tests_AvaliacaoForm.py
tests_RegistroAvaliacaoForm.py

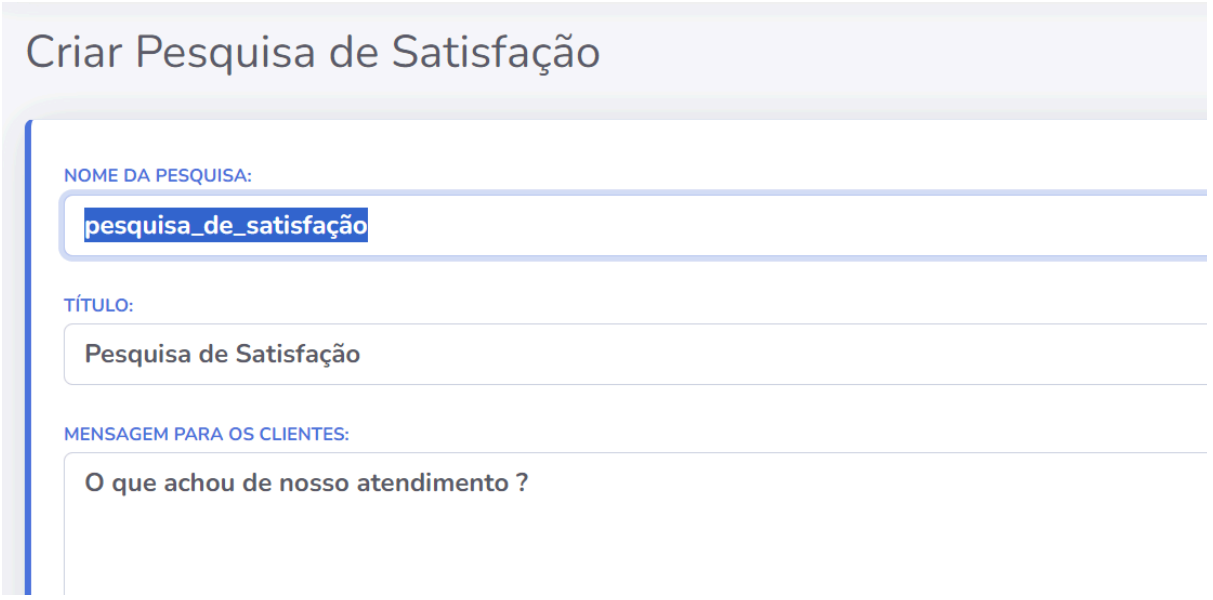
```python
class AvaliacaoCreate_Auth_GET_Test(TestCase):
    def setUp(self):
        User.objects.create_user(username='testuser@gmail.com', password='password123',
                                 aceitou_termos_uso=True, aceitou_politica_privacidade=True,
                                 perfil_preenchido=True)
        data = {'username':'testuser@gmail.com','password':'password123'}
        self.resp = self.client.post(r('login'), data)
        self.resp = self.client.get(r(visited_url))
        self.resp2 = self.client.get(r(visited_url), follow=True)

    def test_context(self):
        form = self.resp.context['form']
        self.assertIsInstance(form, AvaliacaoForm)
        self.assertFalse(form.is_bound)

    def test_created(self):
        self.assertFalse(AvaliacaoModel.objects.exists())

    def test_status_code(self):
        self.assertEqual(self.resp.status_code , HTTPStatus.OK)
        self.assertEqual(self.resp2.status_code , HTTPStatus.OK)

    def test_template_used(self):
        self.assertTemplateUsed(self.resp2, 'avaliacao_create.html')
        self.assertTemplateUsed(self.resp2, 'HEADER.html')
        self.assertTemplateUsed(self.resp2, 'SIDEBAR.html')
        self.assertTemplateUsed(self.resp2, 'TOPBAR.html')
        self.assertTemplateUsed(self.resp2, 'FOOTER.html')
```

1.3 Bugs Identificados

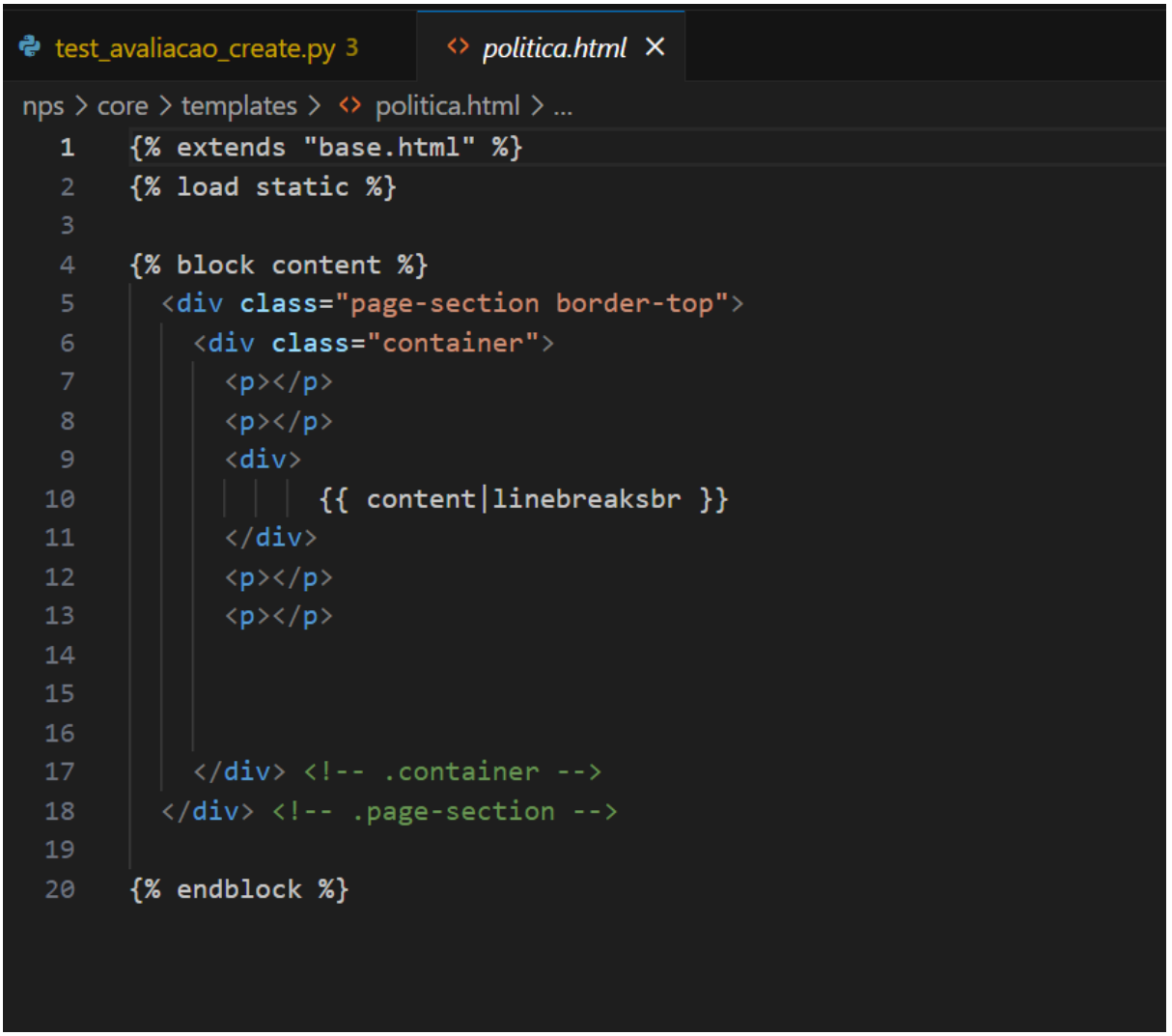
Foram identificados alguns problemas no sistema, como:

- O campo de número de WhatsApp na edição de perfil aceita letras, o que exige a implementação de uma validação mais rigorosa.

- Na página HTML, foi utilizado texto escrito diretamente em um componente de edição. Recomenda-se o uso da propriedade `placeholder`, para evitar que o usuário precise apagar manualmente o texto padrão.



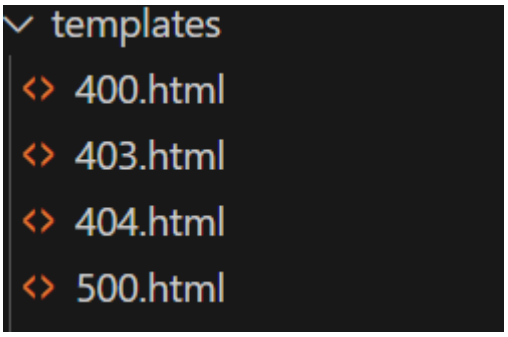
- A página de política de privacidade está vazia, caracterizando uma inconsistência.



1.4 Recomendações para a Equipe Técnica

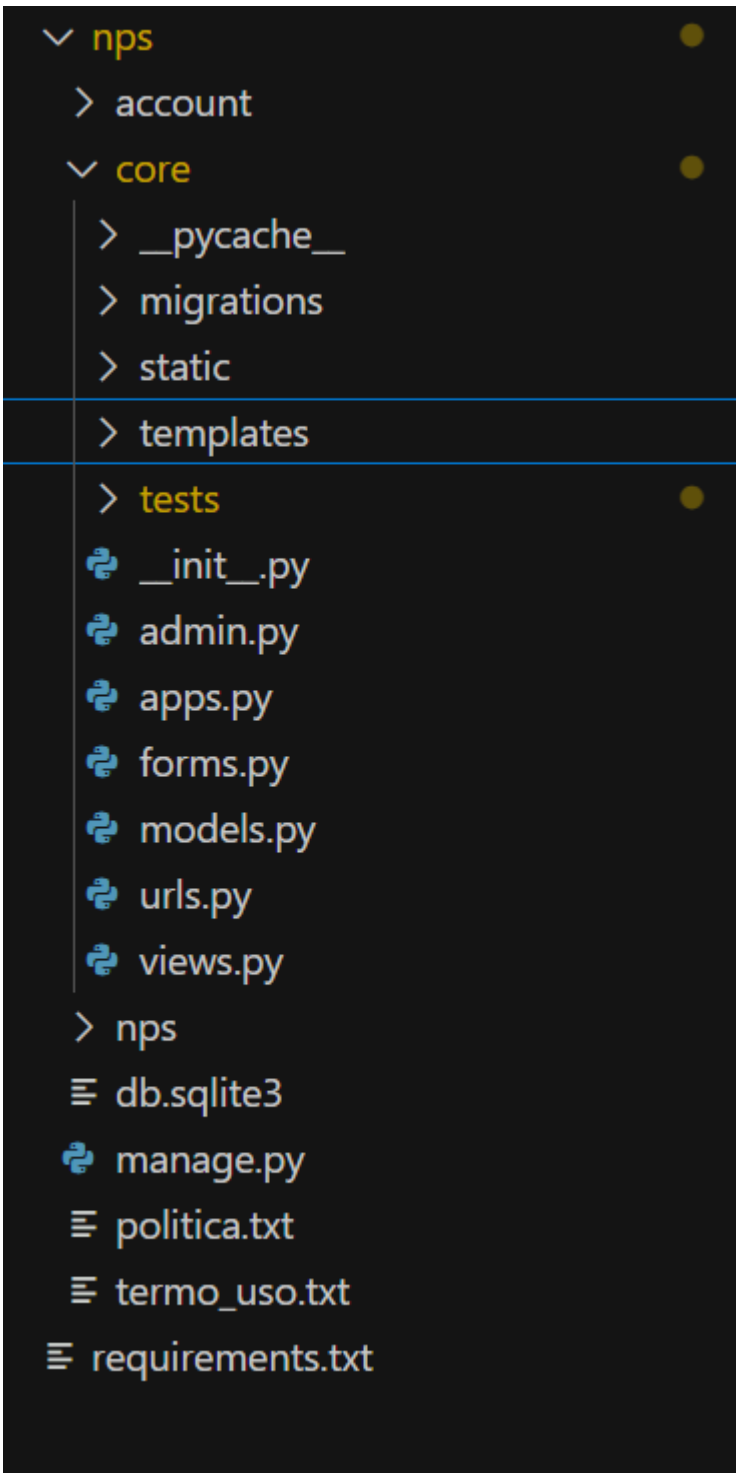
Sugere-se a implementação do uso de serializers para tratamento de erros. Também seria interessante criar uma pasta específica para respostas HTTP, o que organizaria melhor o projeto e facilitaria a manutenção do código.

Serializers são usados principalmente para converter dados complexos, como **querysets** e **modelos** do Django, em formatos que podem ser facilmente renderizados em JSON, XML ou outros formatos para serem utilizados em APIs, especialmente quando você está utilizando o Django REST Framework (DRF). Além disso, eles também podem transformar dados recebidos via requisições HTTP em objetos Python para validação e manipulação.



1.5 Estrutura de Pastas do Projeto

A estrutura de pastas está bem organizada e segue o padrão do framework Django. Pastas como "migrations", "static", "templates" e "tests" estão devidamente configuradas, o que contribui para a fácil compreensão e manutenção do projeto.



---

2. Análise do Programa em Homologação

2.1 Avaliação da Usabilidade da Plataforma

A usabilidade da plataforma foi considerada satisfatória, mas existem áreas a serem melhoradas. Sugere-se:

- A utilização de ícones que ajudem o usuário a identificar ações rapidamente, como criação e edição de pesquisas.
- A adição de recursos de acessibilidade, como a conversão de texto em fala para pessoas com deficiência visual.
- O uso de pop-ups para informar ao usuário sobre erros, como no caso de senha ou e-mail incorreto.
- A inclusão de um modo escuro para usuários que prefiram essa visualização.
- A adição de um título "Descrição" na seção "Dados do Registro" para maior clareza.

2.2 Pontos Fortes e Fracos da Usabilidade

- **Pontos Fortes**: A plataforma é responsiva, fácil de visualizar e simples de usar.
- **Pontos Fracos**: Falta de ícones em funções importantes, ausência de recursos de acessibilidade para deficientes visuais, impossibilidade de alterar a senha ou deletar pesquisas, e falta de pop-ups informativos.

2.3 Recomendações para a Equipe Técnica

Recomenda-se a inclusão de funcionalidades de acessibilidade, como aumento de fontes e leitura de textos para deficientes visuais. Além disso, sugere-se que o aceite dos termos de uso e política de privacidade seja colocado no processo de criação de conta, e não após a criação.

2.4 Funcionalidades de Acessibilidade

Atualmente, o sistema não conta com funcionalidades de acessibilidade, como comandos por voz, modo escuro ou aumento de fonte. Essas melhorias devem ser consideradas em versões futuras.

---

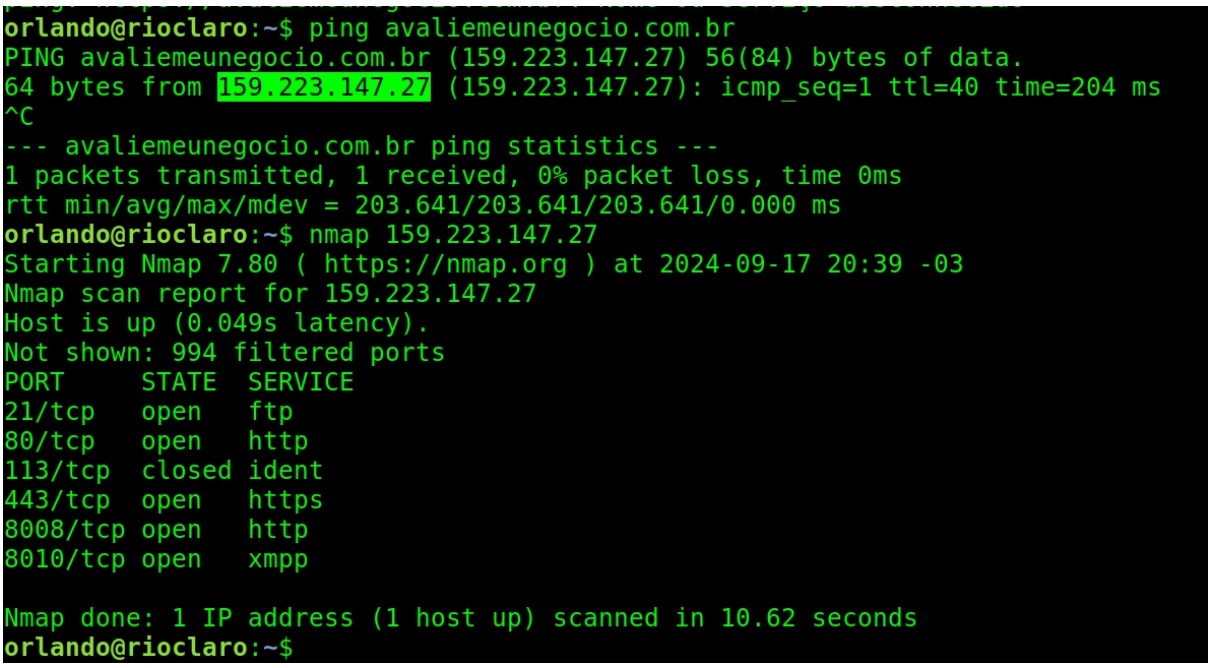
3. Análise do Servidor

3.1 Portas TCP Abertas

A análise do servidor identificou as seguintes portas TCP abertas:

- **22/tcp**: FTP - Utilizada para transferência de arquivos.
- **80/tcp**: HTTP - Usada para tráfego web não seguro.
- **443/tcp**: HTTPS - Usada para tráfego web seguro.
- **8008/tcp**: HTTP - Porta alternativa para tráfego web.
- **8010/tcp**: XAMPP - Associada ao serviço de gerenciamento de servidores XAMPP.

Essas portas abertas sugerem a exposição de serviços importantes, como o acesso FTP e as portas de comunicação HTTP/HTTPS, o que requer atenção especial para segurança. Além disso, a exposição das portas 8008 e 8010 (ligadas ao serviço XAMPP) pode representar um risco se não forem protegidas adequadamente.



Essas portas revelam que o servidor está disponibilizando serviços essenciais, como FTP, HTTP e HTTPS, além de outras portas que podem estar relacionadas a serviços de gerenciamento de servidores, o que demanda precaução.

3.2 Proteção do Servidor

A proteção do servidor foi avaliada, verificando a segurança dos serviços expostos ao público e identificando possíveis pontos de falha na infraestrutura de segurança.

3.3 Softwares e Versões

Foi possível verificar quais softwares e suas respectivas versões estão instalados no servidor, assegurando que estejam atualizados e livres de vulnerabilidades conhecidas.

3.4 Recomendações para a Equipe Técnica

Recomenda-se uma revisão contínua das configurações de segurança, monitoramento ativo de portas e serviços, bem como a verificação regular de atualizações dos softwares instalados no servidor.