INSTITUTO FEDERAL DE RONDÔNIA
***CAMPUS* PORTO VELHO CALAMA**
CURSO SUPERIOR DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

MARYA CLARA OLIVEIRA COSTA

AVALIAÇÃO DA IMPLEMENTAÇÃO DE HTTP SECURITY HEADERS EM APLICAÇÕES WEB DE PEQUENO PORTE

Porto Velho/RO
2025

MARYA CLARA OLIVEIRA COSTA

AVALIAÇÃO DA IMPLEMENTAÇÃO DE HTTP SECURITY HEADERS EM APLICAÇÕES WEB DE PEQUENO PORTE

Artigo entregue como Trabalho de Conclusão de Curso ao Instituto Federal de Educação, Ciência e Tecnologia de Rondônia (IFRO), *Campus* Porto Velho Calama, como requisito parcial para obtenção do grau de Tecnólogo, junto ao Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas.

Orientador: Leandro Ferrarezi Valiante

Porto Velho/RO
2025

Ficha catalográfica elaborada pelo Sistema Gerador de Ficha Catalográfica do IFRO.

Costa, Marya Clara Oliveira.
Avaliação da Implementação de HTTP Security Headers em Aplicações Web de Pequeno Porte / Marya Clara Oliveira Costa. - Porto Velho, 2025.
28 f. : il.

Orientador(a): Prof. Dr. Leandro Ferrarezi Valiante.

Trabalho de Conclusão de Curso (Superior de Tecnologia em Análise e Desenvolvimento de Sistemas) – Instituto Federal de Educação, Ciência e Tecnologia de Rondônia - IFRO, Porto Velho, 2025.

1. DJANGO. 2. HTTP SECURITY HEADERS. 3. SEGURANÇA WEB. 4. VULNERABILIDADES. I. Valiante, Leandro Ferrarezi (orient.). II. Instituto Federal de Educação, Ciência e Tecnologia de Rondônia - IFRO. III. Título.

Bibliotecário(a) Responsável: Miria Santana Veiga, CRB-11/898

MARYA CLARA OLIVEIRA COSTA

AVALIAÇÃO DA IMPLEMENTAÇÃO DE HTTP SECURITY HEADERS EM APLICAÇÕES WEB DE PEQUENO PORTE

A banca examinadora, abaixo listada, aprova o Trabalho de Conclusão de Curso “AVALIAÇÃO DA IMPLEMENTAÇÃO DE HTTP SECURITY HEADERS EM APLICAÇÕES WEB DE PEQUENO PORTE” elaborado por “MARYA CLARA OLIVEIRA COSTA” como requisito parcial para obtenção do grau de Tecnólogo em Análise e Desenvolvimento de Sistemas, pelo Instituto Federal de Educação, Ciência e Tecnologia de Rondônia.

Porto Velho/RO, 12/12/2025

Comissão Examinadora

LEANDRO FERRAREZI VALIANTE:82694095291

Assinado de forma digital por LEANDRO FERRAREZI VALIANTE:82694095291
Dados: 2025.12.22 16:18:17 -04'00'

Leandro Ferrarezi Valiante - IFRO
(Orientador)

gov.br Documento assinado digitalmente
FERNANDO DALL IGNA
Data: 22/12/2025 15:19:52-0300
Verifique em https://validar.iti.gov.br

Prof. Fernando Dall Igna - IFRO

gov.br Documento assinado digitalmente
CELSO GUEDES GOMES
Data: 22/12/2025 17:08:20-0300
Verifique em https://validar.iti.gov.br

Prof. Celso Guedes Gomes - IFRO

AVALIAÇÃO DA IMPLEMENTAÇÃO DE HTTP SECURITY HEADERS EM APLICAÇÕES WEB DE PEQUENO PORTE

RESUMO: Este trabalho investiga a efetividade dos HTTP *Security Headers* na proteção de aplicações *web* de pequeno porte, frequentemente expostas a vulnerabilidades decorrentes da ausência de políticas explícitas de segurança. Considerando que sistemas simples, como *blogs*, portfólios e *websites* pessoais, frequentemente priorizam funcionalidade em detrimento da segurança, desenvolveu-se um protótipo em Django representativo desse perfil. Após implantação em ambiente de produção, realizou-se diagnóstico inicial utilizando a ferramenta Security Headers, identificando-se a ausência de mecanismos essenciais e definindo-se uma linha de base para comparação. Em seguida, foram implementados *Strict-Transport-Security*, *Content-Security-Policy* e *Permissions-Policy*. A nova análise, sob as mesmas condições experimentais, registrou aumento da classificação de C para A+, evidenciando impacto positivo das políticas adotadas. Os resultados indicam que, mesmo em projetos com infraestrutura limitada e baixa maturidade em práticas de segurança, a implementação adequada de HTTP *Security Headers* amplia a resiliência de aplicações *web* e fornece subsídios práticos a desenvolvedores.

PALAVRAS-CHAVE: DJANGO. HTTP SECURITY HEADERS. SEGURANÇA *WEB*. VULNERABILIDADES.

ABSTRACT: This study investigates the effectiveness of HTTP Security Headers in protecting small-scale web applications, which are often exposed to vulnerabilities resulting from the absence of explicit security policies. Considering that simple systems such as blogs, portfolios, and personal websites frequently prioritize functionality over security, a Django-based prototype representative of this profile was developed. After deployment in a production environment, an initial assessment was conducted using the Security Headers tool, identifying the absence of essential mechanisms and establishing a baseline for comparison. Subsequently, Strict-Transport-Security, Content-Security-Policy, and Permissions-Policy were implemented. A second analysis, under the same experimental conditions, showed an improvement in the security rating from C to A+, demonstrating the positive impact of the adopted policies. The results indicate that even in projects with limited infrastructure and low security maturity, the proper implementation of HTTP Security Headers enhances the resilience of web applications and provides practical guidance for developers.

KEYWORDS: DJANGO. WEB SECURITY. HTTP SECURITY HEADERS. VULNERABILITIES.

1 INTRODUÇÃO

A ampla adoção de aplicações *web* e serviços online, aliada à crescente conectividade global, intensificou a exposição de sistemas a ameaças cibernéticas, tornando a segurança um elemento essencial para a integridade, confiabilidade e disponibilidade desses serviços (Nawrocki; Kołodziej, 2024). Nos últimos anos, observou-se um aumento significativo no volume e na sofisticação de ataques direcionados a aplicações *web* e Interfaces de Programação de Aplicações (*Application Programming Interfaces* — APIs), evidenciando que o ambiente digital se tornou alvo constante de agentes maliciosos (Technologies, 2025).

As aplicações *web* atraem desenvolvedores e usuários pelo baixo custo e alcance global, favorecendo a criação de sites pessoais, blogs e sistemas informais que, pela simplicidade, muitas vezes operam sem práticas de segurança efetivas (Silva, 2011). Estudos indicam que vulnerabilidades relacionadas à lógica da aplicação e a falhas de configuração representam parcela significativa dos problemas encontrados (Nawrocki; Kołodziej, 2024). Relatórios do “Open Worldwide Application Security Project” (OWASP) apontam o controle de acesso inadequado e falhas de configuração entre os riscos mais críticos, reforçando a necessidade de incorporar segurança desde as fases iniciais de desenvolvimento (OWASP Foundation, 2021b).

Soma-se a isso o fato de que grande parte das vulnerabilidades exploradas decorrem não apenas de falhas complexas de lógica, mas de descuidos ou falhas de configuração, ou seja, barreiras “básicas” que poderiam ser evitadas. Dessa forma, o risco torna-se ainda maior em aplicações web de pequeno porte, entendidas neste trabalho como sistemas desenvolvidos para atender a um número reduzido de usuários, geralmente operando em escala limitada, com infraestrutura simplificada, baixa ou inexistente distribuição de carga e recursos técnicos restritos, como ausência de equipes dedicadas à segurança ou de processos formais de auditoria. Nesse contexto, a limitação de recursos técnicos, aliada à possível ausência de práticas sistemáticas de segurança, amplia a exposição a vulnerabilidades (Nawrocki; Kołodziej, 2024).

Esse cenário reforça a necessidade de investigar, em nível experimental, como boas práticas de segurança, especialmente no transporte e comunicação por meio do Protocolo de Transferência de Hipertexto (*Hypertext Transfer Protocol* - HTTP), podem elevar a resiliência dessas aplicações e proteger tanto desenvolvedores quanto usuários das consequências potenciais dessas vulnerabilidades.

Diretrizes internacionais indicam que a adoção de boas práticas de segurança desde as fases iniciais do desenvolvimento é essencial para reduzir vulnerabilidades em aplicações *web*, especialmente no controle do comportamento entre navegador e servidor (OWASP Foundation, 2021b). Nesse contexto, os Cabeçalhos de Segurança

HTTP (HTTP *Security Headers*) atuam como políticas declarativas inseridas na resposta HTTP para restringir a execução de recursos, suas origens e permissões no cliente, funcionando como uma camada preventiva contra configurações inadequadas (Mozilla Foundation, 2023).

Embora a segurança seja um requisito fundamental para sistemas em produção, aplicações *web* de pequeno porte frequentemente são desenvolvidas com foco predominante em funcionalidade e entrega rápida, deixando em segundo plano práticas de proteção. Esse cenário é crítico quando os cabeçalhos de segurança HTTP não são configurados adequadamente, pois a aplicação passa a operar com políticas permissivas que não comunicam ao navegador restrições mínimas de comportamento. Como resultado, páginas simples, como *blogs* ou *sites* institucionais, tornam-se vulneráveis à exploração por agentes externos. Assim, emerge o problema que orienta este trabalho: compreender em que medida a adoção de políticas de segurança via *HTTP Security Headers* pode elevar o nível de proteção de aplicações *web*, mitigando riscos decorrentes de configurações padrão ou ausentes, sem exigir mudanças profundas na arquitetura do sistema.

A escolha por investigar a aplicação de *HTTP Security Headers* em sistemas de pequeno porte justifica-se pelo fato de esse perfil representar parcela expressiva do ecossistema *web* atual, sendo amplamente empregado por profissionais independentes, estudantes, microempresas e iniciativas pessoais que raramente dispõem de equipes ou recursos avançados de segurança. Diferentemente de grandes plataformas comerciais, sustentadas por times especializados, auditorias frequentes e investimentos contínuos, esses projetos costumam ser desenvolvidos por indivíduos ou equipes reduzidas, que recorrem às configurações padrão de *frameworks*, servidores e serviços de hospedagem, assumindo que tais medidas são suficientes para garantir a proteção do sistema.

Esse contexto torna essas aplicações mais suscetíveis a vulnerabilidades evitáveis. A ausência de controles explícitos expõe o sistema a riscos que poderiam ser mitigados por mecanismos simples, como os cabeçalhos de segurança HTTP. Ao demonstrar, de forma prática, como essas políticas podem ser aplicadas e avaliadas, este trabalho busca oferecer um caminho acessível para o fortalecimento da segurança de projetos reais, difundindo conhecimento.

1.1 Objetivos

1.1.1 Objetivo geral

Avaliar a efetividade da implementação de *HTTP Security Headers* na mitigação de vulnerabilidades comuns em uma aplicação *web* de pequeno porte desenvol-

vida em Django, por meio da análise comparativa entre o comportamento do sistema antes e depois da aplicação desses mecanismos de segurança, visando identificar sua contribuição para o fortalecimento da proteção e da resiliência do ambiente *web*.

1.1.2 Objetivos específicos

1. Identificar os principais *HTTP Security Headers* e compreender sua função na mitigação de vulnerabilidades em aplicações *web*.
2. Desenvolver um protótipo *web* em Django, representativo de aplicações de pequeno porte, para fins de experimentação.
3. Implementar os *HTTP Security Headers* selecionados no protótipo, seguindo boas práticas consolidadas de segurança.
4. Realizar testes comparativos antes e depois da implementação dos *headers*, utilizando ferramentas adequadas de análise.
5. Avaliar os resultados obtidos, verificando o impacto das configurações na redução da superfície de ataque.
6. Documentar o processo de implementação e os resultados, com o intuito de fornecer referências práticas a futuros desenvolvedores.

2 FUNDAMENTAÇÃO TEÓRICA

Nesta seção, são apresentados os fundamentos teóricos que sustentam o desenvolvimento e a análise desta pesquisa. Serão abordados os principais conceitos relacionados ao funcionamento da *Web*, às tecnologias utilizadas na construção da aplicação, bem como às ferramentas, métodos e diretrizes empregados na implementação e avaliação de mecanismos de segurança. A partir desses elementos, estabelece-se a base conceitual necessária para compreender o contexto, os procedimentos e as decisões adotadas ao longo do trabalho.

A evolução da Internet possibilitou o surgimento de novas formas de organizar, acessar e compartilhar informações, sendo a *World Wide Web* (WWW) um dos marcos mais significativos nesse processo. A WWW constitui um espaço de informação distribuída e inter-relacionada por meio de documentos hipermídia, como textos, imagens, vídeos e outros recursos, conectados por *links* e acessíveis via rede global (Bryant, 2020).

Com o avanço da Internet, as aplicações *web* consolidaram-se como principal modelo de serviços digitais, sendo sistemas executados em servidor e acessados por

navegadores por meio de tecnologias baseadas em HTTP. Elas integram interface, lógica de negócio, comunicação cliente-servidor e gerenciamento de dados, o que as torna flexíveis e aplicáveis a diversos domínios (Li *et al.*, 2024).

O *Hypertext Transfer Protocol* (HTTP) é o protocolo central da *Web*, responsável pela comunicação entre clientes e servidores por meio do modelo de requisições e respostas. Ele padroniza a transferência de documentos e recursos, utilizando métodos, cabeçalhos e códigos de *status* para garantir controle, interoperabilidade e flexibilidade no transporte de conteúdo digital (Gourley *et al.*, 2002).

Dando continuidade ao entendimento sobre HTTP, o Protocolo de Transferência de Hipertexto Seguro (*Hypertext Transfer Protocol Secure* — HTTPS), surge como uma extensão necessária para tornar essa comunicação mais segura. Ele utiliza uma camada adicional de proteção para impedir que os dados transmitidos entre cliente e servidor possam ser lidos ou alterados por terceiros. Segundo Tariq, Zainab e Hussain (2024), essa proteção torna o tráfego mais confiável, especialmente em aplicações que lidam com informações sensíveis.

Um servidor *web* desempenha papel fundamental na comunicação entre cliente e servidor, sendo responsável por armazenar, processar e entregar os recursos de um *site* ou aplicação sempre que uma requisição é realizada pelo navegador. Segundo Awati e Gillis (2024), ele reúne hardware e software para receber requisições HTTP ou HTTPS, localizar arquivos como páginas HTML, *scripts*, estilos e imagens, e enviá-los ao cliente para exibição do conteúdo (Khaire *et al.*, 2022).

Os Cabeçalhos de Resposta HTTP (HTTP *Response Headers*) constituem metadados transmitidos pelo servidor ao cliente juntamente com o corpo da resposta, determinando instruções essenciais sobre como o conteúdo deve ser interpretado, armazenado ou executado pelo navegador. Além de transportar atributos técnicos como tipo de mídia, *status* da requisição ou diretrizes de cache, os cabeçalhos podem estabelecer políticas que restringem comportamentos ou definem limites de interação entre o usuário e o sistema, o que os torna elementos fundamentais para a segurança de aplicações *web* (Fielding; Reschke, 2014).

Os *HTTP Security Headers* consistem em campos de resposta definidos pelo servidor que instruem o navegador sobre como lidar com determinadas funcionalidades ou restrições, reduzindo vulnerabilidades e outras ameaças comuns em aplicações *web* (Project, 2025). Além disso, um estudo recente que analisou milhares de *websites* populares em escala global evidenciou que a adoção sistemática desses *headers* está associada a uma postura de segurança mais rigorosa, o que contribui para a reduzir explorações no lado do cliente (Kishnani; Das, 2024).

O HTML (*HyperText Markup Language*) é a linguagem responsável por orga-

nizar o conteúdo de páginas *web* por meio de elementos semânticos e hierárquicos. Segundo Duckett (2011), o HTML define a forma como textos, imagens, *links*, listas, tabelas e demais componentes são apresentados ao navegador, permitindo que as páginas se tornem documentos navegáveis compostos por blocos de informação interligados.

O CSS (*Cascading Style Sheets*) possui a função de controlar a forma como conteúdos estruturados em HTML são exibidos ao usuário, permitindo o gerenciamento de cores, tipografia, espaçamentos e *layouts* de interface (Meyer; Weyl, 2023).

Segundo Flanagan (2020), o JavaScript opera diretamente no navegador, permitindo alterar a estrutura visual representada pelo *Document Object Model*, reagir a eventos de usuário, atualizar conteúdos de forma assíncrona e executar lógicas que tornam a aplicação responsiva e mais próxima da experiência de software nativos (Flanagan, 2020).

Segundo Lutz (2023), Python é uma linguagem de programação de alto nível, multiparadigma e interpretada, projetada para ser expressiva e eficiente tanto para iniciantes quanto para profissionais experientes. O autor enfatiza que sua filosofia prioriza a redução da complexidade desnecessária, o que se reflete em uma sintaxe direta e em um amplo conjunto de bibliotecas e módulos que possibilitam a solução de tarefas variadas (Lutz, 2023).

O Django é um *framework web* em Python que visa acelerar o processo de construção de aplicações ao fornecer uma arquitetura estruturada e componentes pré-configurados. Segundo Baker (2024), o Django adota o padrão MTV (*Model–Template–View*), no qual a lógica de dados, a camada de apresentação e o controle de fluxo são separados de maneira clara, favorecendo manutenção, modularidade e organização do código (Baker, 2024).

A “Open Web Application Security Project” (OWASP) é uma fundação internacional sem fins lucrativos dedicada ao avanço da segurança de software, oferecendo recursos e diretrizes para o desenvolvimento de aplicações mais seguras. Entre suas iniciativas, destaca-se o “OWASP Secure Headers Project” (OSHP), que reúne recomendações sobre o uso de cabeçalhos HTTP para mitigar riscos no navegador, restringir comportamentos indesejados e reduzir vulnerabilidades por meio de mecanismos preventivos simples e configuráveis (OWASP Foundation, 2025e).

O “Security Headers”, originalmente criado por Scott Helme, e atualmente mantido pela Snyk, é uma ferramenta de análise voltada à avaliação de configurações de segurança em aplicações *web*, com foco específico na verificação de cabeçalhos HTTP relacionados à proteção do navegador (SecurityHeaders, 2025).

Segundo o OWASP, o cabeçalho HTTP `Strict-Transport-Security` (HSTS)

é um mecanismo que obriga o navegador a utilizar exclusivamente conexões HTTPS ao acessar uma aplicação, prevenindo ataques de *downgrade* de protocolo e interceptação de tráfego ao impor automaticamente a comunicação segura (Foundation, 2025).

O `Content-Security-Policy` (CSP) é um cabeçalho HTTP que permite definir, de forma explícita, quais origens de conteúdo o navegador está autorizado a carregar e executar, restringindo *scripts*, estilos, imagens e outros recursos. Segundo a OWASP, esse mecanismo atua na prevenção de injeções de conteúdo e execução de código malicioso ao impor políticas de confiança no carregamento de recursos (OWASP Foundation, 2025c).

Segundo a OWASP, o `X-Frame-Options` (XFO) é um cabeçalho HTTP que impede que uma página seja carregada em *frames* ou *iframes* de domínios não autorizados, atuando diretamente na prevenção de ataques de *clickjacking* (OWASP Foundation, 2025b).

O `X-Content-Type-Options` é um cabeçalho HTTP que impede o navegador de realizar *MIME sniffing*, forçando a interpretação dos arquivos exclusivamente de acordo com o tipo de conteúdo declarado pelo servidor, por meio do valor *nosniff* (OWASP Foundation, 2025d).

A `Referrer-Policy` é um cabeçalho HTTP que controla quais informações de origem (*referrer*) o navegador envia ao realizar requisições, permitindo limitar ou omitir dados sensíveis da URL em acessos a recursos externos (OWASP Foundation, 2025g).

A `Permissions-Policy` é um cabeçalho HTTP que permite ao servidor controlar, de forma explícita, quais funcionalidades e APIs do navegador podem ser utilizadas por uma aplicação, como câmera, microfone e geolocalização, restringindo seu uso a origens autorizadas ou bloqueando-o quando necessário (OWASP Foundation, 2025f).

O *clickjacking* é um ataque em que o invasor manipula a interface de uma página para induzir o usuário a clicar em elementos invisíveis ou disfarçados, levando-o a executar ações não intencionais em outra página ou domínio (OWASP Foundation, 2025a). Essa técnica consiste na sobreposição de camadas sobre a interface legítima, caracterizando um tipo de *UI redress* (OWASP Foundation, 2025a).

A injeção de conteúdo ocorre quando uma aplicação processa dados externos sem validação adequada, permitindo a inserção de código arbitrário no contexto do usuário. Sua forma mais comum é o *Cross-Site Scripting* (XSS), no qual *scripts* maliciosos são inseridos em páginas legítimas e executados pelo navegador da vítima (OWASP Foundation, 2021a).

3 METODOLOGIA

Considerando a classificação metodológica apresentada por Wazlawick (2020), este trabalho caracteriza-se como uma pesquisa aplicada com enfoque de *design*, uma vez que busca propor e avaliar soluções práticas para um problema técnico específico, por meio do desenvolvimento de um protótipo funcional e da implementação de mecanismos de segurança.

Diferentemente da pesquisa descritiva ou exploratória, cujo objetivo recai sobre o levantamento de fenômenos ou a observação de comportamentos existentes, a pesquisa de *design* concentra-se na construção de artefatos que apontem como as coisas poderiam ser, atuando sobre o ambiente investigado para propor aperfeiçoamentos concretos. Nesse sentido, este trabalho enquadra-se como pesquisa primária, pois envolve experimentação direta sobre a aplicação desenvolvida, constituindo um processo de intervenção controlada em conformidade com o entendimento de pesquisa experimental aplicado à área da Computação.

Considerando que este estudo tem como foco aplicações *web* de pequeno porte, optou-se pelo desenvolvimento de um protótipo funcional que viabilizasse a experimentação prática. Nesse contexto, fez-se necessária a construção de um ambiente no qual fosse possível observar lacunas de segurança e avaliar, de forma controlada, os efeitos das intervenções planejadas. Dessa forma, o protótipo assumiu o papel de artefato experimental, permitindo a realização dos testes, a identificação de deficiências e a análise do impacto das medidas de segurança aplicadas.

Na etapa de construção do artefato, definiu-se que o protótipo deveria representar uma aplicação *web*, com funcionalidades mínimas suficientes para estabelecer o fluxo de requisições e respostas HTTP. Para o desenvolvimento, utilizou-se a linguagem Python e o *framework* Django, escolhidos por sua ampla adoção e por oferecerem uma estrutura consolidada para aplicações baseadas em servidor. A interface foi desenvolvida com HTML, CSS e JavaScript em sua forma básica, uma vez que o foco do estudo recai sobre os cabeçalhos de segurança nas respostas HTTP. O protótipo resultante constitui uma aplicação simples e funcional, adequada ao contexto experimental da pesquisa.

Para viabilizar o diagnóstico, foi necessário publicar o protótipo em um ambiente acessível publicamente, uma vez que a ferramenta de avaliação opera por meio da análise de respostas HTTP a partir de uma URL. O *deploy* foi realizado na plataforma Railway, escolhida por oferecer um processo de implantação simplificado e automatizado. Após a publicação, a aplicação foi configurada em modo de produção e obteve-se uma URL pública estável, condição necessária para a execução das análises externas. Dessa forma, o *deploy* integrou o procedimento metodológico ao es-

tabelecer o ambiente operacional no qual os *HTTP Security Headers* puderam ser observados e avaliados.

Após a implantação do protótipo em ambiente de produção, realizou-se o diagnóstico inicial com o objetivo de identificar os *HTTP Security Headers* presentes na aplicação antes de qualquer intervenção. Para isso, utilizou-se a ferramenta Security Headers da Snyk, que executa requisições diretamente à URL informada e avalia os cabeçalhos retornados na resposta HTTP. Essa abordagem permite a observação do comportamento da aplicação em condições reais de operação, sem a necessidade de inspeção do código-fonte ou do ambiente interno.

O procedimento para realização do diagnóstico consistiu na submissão da URL pública do protótipo à ferramenta de avaliação, que analisa as respostas HTTP e verifica a presença, ausência ou configuração inadequada dos principais cabeçalhos de segurança. O Security Headers apresenta os resultados de forma objetiva, o que facilita a interpretação e a priorização de ajustes. A avaliação preliminar permitiu identificar lacunas e possibilitar melhorias na segurança da aplicação, estabelecendo o ponto de referência inicial para o processo experimental. Essa etapa registrou o comportamento original da aplicação antes das intervenções e serviu de base para orientar as modificações seguintes.

Com base no diagnóstico inicial, iniciou-se a etapa de intervenção no protótipo, voltada à implementação dos *HTTP Security Headers* ausentes. Embora a aplicação já apresentasse cabeçalhos configurados automaticamente pelo *framework*, esses mecanismos não eram suficientes para mitigar riscos relacionados à interceptação de tráfego, execução não autorizada de *scripts* e uso indevido de recursos do navegador. Assim, a intervenção concentrou-se na implementação do `Strict-Transport-Security`, do `Content-Security-Policy` e do `Permissions-Policy`.

O `Strict-Transport-Security` foi implementado para forçar o uso exclusivo de HTTPS e prevenir ataques de *downgrade*. A `Content-Security-Policy` foi adotada para restringir o carregamento e a execução de *scripts*, mitigando riscos de injeção de conteúdo. Já a `Permissions-Policy` foi aplicada para limitar o acesso a recursos sensíveis do navegador. A adoção desses cabeçalhos visou reforçar o comportamento de segurança da aplicação, sem alterar sua funcionalidade. A aplicação dos cabeçalhos ocorreu no mesmo ambiente de produção e sob as mesmas condições do diagnóstico inicial. Essa estratégia permitiu atribuir as variações observadas exclusivamente às políticas implementadas, sem interferência de alterações de infraestrutura ou funcionalidade. Dessa forma, a implementação dos HTTP *Security Headers* constituiu a etapa central do experimento e estabeleceu a base para a avaliação posterior de sua eficácia.

Após a implementação dos HTTP *Security Headers*, a aplicação foi reavaliada

com o objetivo de verificar o impacto direto das intervenções. Para assegurar a comparabilidade, mantiveram-se o mesmo ambiente de produção, a mesma URL pública e a mesma ferramenta utilizada no diagnóstico inicial. Essa nova análise permitiu comparar o estado inicial com o estado posterior à intervenção, evidenciando as alterações decorrentes das configurações aplicadas. Dessa forma, a reaplicação da ferramenta constituiu a etapa final do procedimento experimental, destinada ao registro sistemático do comportamento da aplicação após as modificações.

4 DESENVOLVIMENTO

4.1 Estrutura da interface: HTML, CSS e JavaScript

A aplicação *web* foi desenvolvida no formato de *blog*, atendendo aos requisitos de simplicidade estrutural e possibilidade de avaliação de cabeçalhos de segurança. A implementação incluiu páginas estáticas e dinâmicas suficientes para criar um fluxo real de requisições HTTP.

Como mostra a Figura 1, a estrutura de navegação da aplicação foi planejada para permitir acesso direto às principais seções, facilitando a experiência do usuário.

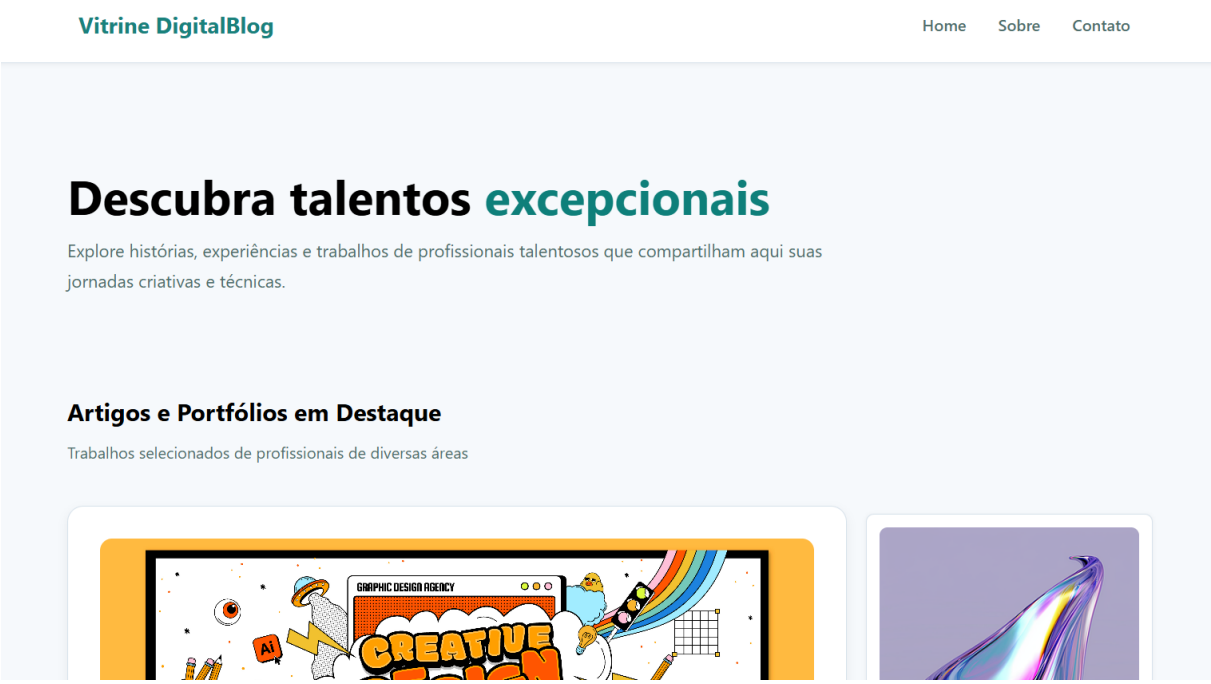
Figura 1 – Estrutura simplificada de navegação da aplicação

Fonte: Autoria própria, 2025.

Cada página foi implementada com HTML, CSS e JavaScript, garantindo funcionalidade mínima para simular o comportamento de um site real. A interface foi projetada com natureza minimalista, priorizando clareza e funcionalidade.

A página “*Home*” atua como porta de entrada da aplicação e apresenta seu propósito de conectar profissionais e destacar portfólios, conforme ilustrado na Figura 2. Possui uma seção inicial com título, subtítulo e chamada para ação, seguida por *cards* com exemplos de perfis e artigos. O conteúdo é organizado de forma vertical e renderizado de maneira estática, simulando um *feed* editorial que concentra toda a navegação da aplicação.

Figura 2 – Página inicial (*Home*) do protótipo



Fonte: Autoria própria, 2025.

A página “Sobre” apresenta o manifesto, os valores e a missão do projeto, conforme ilustrado na Figura 3. O conteúdo é organizado em blocos textuais curtos, que descrevem o processo de curadoria e a equipe de forma hierárquica. A interface mantém caráter minimalista, com tipografia clara e poucos elementos visuais, priorizando a comunicação de transparência e confiança.

Figura 3 – Página “Sobre” com seções de missão e funcionamento

Fonte: Autoria própria, 2025.

A página “Contato” concentra a funcionalidade do protótipo ao disponibilizar um formulário para submissão de portfólios, conforme ilustrado na Figura 4. O usuário informa nome, e-mail, descrição do trabalho e pode anexar arquivos. A interface apresenta campos bem delimitados e rótulos claros, assegurando a correta inserção dos dados.

Figura 4 – Página “Contato” com formulário de submissão

Fonte: Autoria própria, 2025.

4.2 Integração com o framework Django

Com a base visual definida, a aplicação foi integrada ao *framework* Django com o propósito de fornecer *backend* funcional, gerenciamento de rotas e estrutura MVT estável. A utilização do Django possibilitou a criação de templates dinâmicos a partir dos arquivos HTML previamente desenvolvidos, permitindo a reutilização de componentes e a organização consistente da aplicação. Foram configuradas *views* simples para renderização das páginas, além de rotas no arquivo `urls.py` que direcionavam o fluxo de navegação.

A adoção do *framework* não teve como objetivo a implantação de funcionalidades complexas, como autenticação ou armazenamento de dados, mas sim fornecer um ambiente realista de aplicação *web* que produzisse respostas HTTP completas.

Essa característica foi fundamental para o estudo, pois os cabeçalhos de segurança avaliados pela pesquisa são atribuídos à camada de resposta gerada pelo servidor *web*.

4.3 Implantação da aplicação em ambiente de produção

Na etapa seguinte, a aplicação foi implantada em ambiente de produção na plataforma Railway, escolhida pela facilidade de configuração e integração com repositórios do GitHub. O processo envolveu a criação do arquivo `requirements.txt`, a definição do comando de execução via Gunicorn e a ativação do modo de produção, conforme descrito no Quadro 1. A disponibilização pública foi necessária para viabilizar a análise por ferramentas externas de avaliação de segurança, que dependem de acesso real à aplicação.

Quadro 1 – Configuração mínima para execução da aplicação Django em produção.

Item	**Descrição**	**Exemplo / Conteúdo**
SECRET_KEY	Chave criptográfica usada pelo Django para validação de sessões e segurança interna	*django-insecure-xxxxxxxx...*
DEBUG	Define o modo de execução da aplicação; sempre False em produção	*False*
ALLOWED_HOSTS	Lista de domínios autorizados a acessar a aplicação	*
PORT	Porta na qual o servidor Gunicorn deve escutar	*8000* (ou fornecida pelo Railway)
requirements.txt	Arquivo que lista todas as dependências Python necessárias	*Django==5.2.6, gunicorn==23.0.0*
Procfile	Arquivo que indica ao Railway como executar a aplicação	*web: gunicorn meuapp.wsgi*
Gunicorn	Servidor WSGI utilizado para executar a aplicação Django em produção	*gunicorn meuapp.wsgi:application*

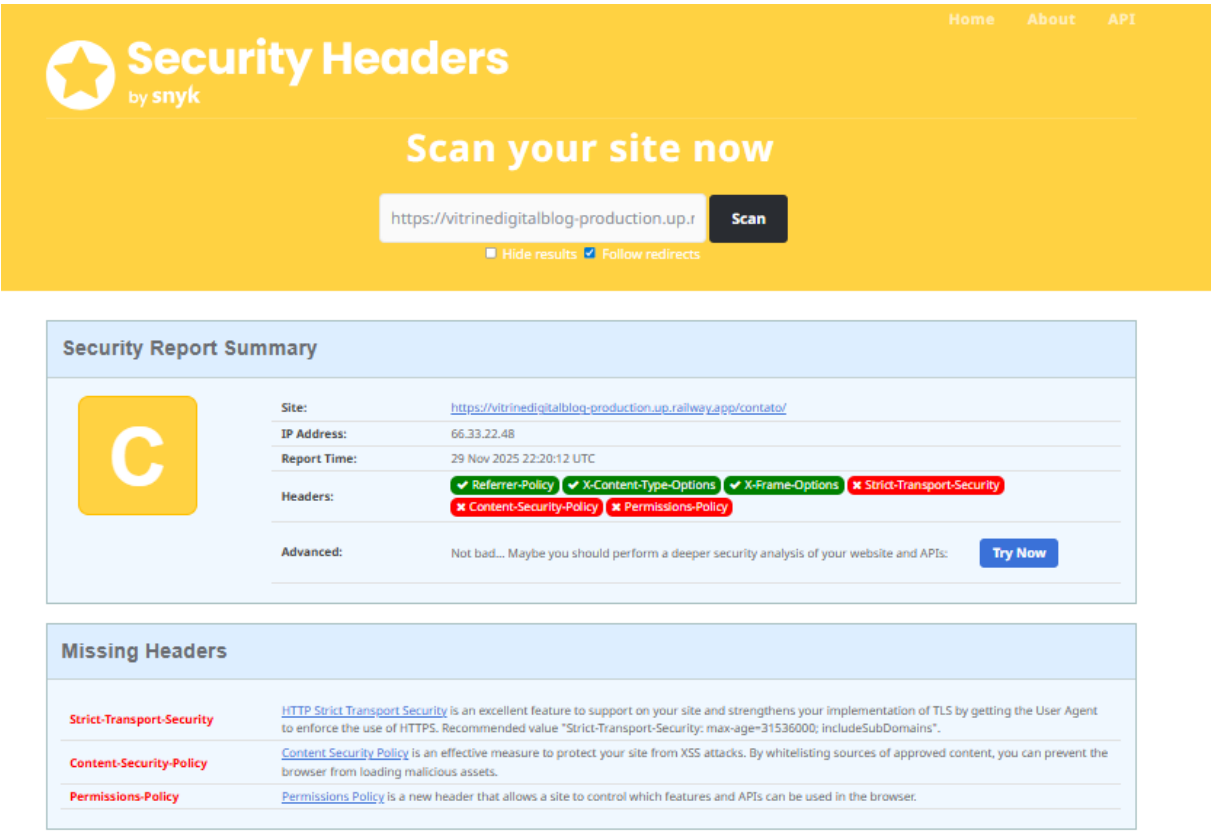
Fonte: Autoria própria, 2025.

4.4 Diagnóstico inicial de segurança

Com a aplicação publicada, realizou-se um diagnóstico inicial dos HTTP *Security Headers* por meio do serviço “Security Headers” do Snyk, conforme apresentado na Figura 5. No primeiro teste, a aplicação recebeu nota C, indicando a presença de

algumas proteções básicas fornecidas nativamente pelo *framework*, mas ainda com ausência de diretrizes essenciais para um ambiente seguro em produção.

Figura 5 – Relatório inicial gerado pelo “Security Headers”



Fonte: Autoria própria, 2025.

O relatório inicial indicou a presença dos cabeçalhos de segurança `X-Frame-Options`, `X-Content-Type-Options` e `Referrer-Policy`, conforme descrito na Tabela 1. Esses mecanismos atuam na prevenção de *MIME sniffing*, no bloqueio da incorporação da interface por *iframes* externos e no controle do envio de informações de origem das requisições. Tais políticas são inseridas automaticamente pelo *framework* e oferecem um nível inicial de segurança, ainda que limitado frente às ameaças atuais.

Tabela 1 – Estado inicial dos cabeçalhos de segurança.

Header	Situação
X-Frame-Options	Presente
X-Content-Type-Options	Presente
Referrer-Policy	Presente
Strict-Transport-Security	Ausente
Content-Security-Policy	Ausente
Permissions-Policy	Ausente

Fonte: Autoria própria, 2025.

O diagnóstico também indicou a ausência dos cabeçalhos `Strict-Transport-Security`, `Content-Security-Policy` e `Permissions-Policy`. Esses mecanismos são essenciais para o uso exclusivo de HTTPS, a restrição de fontes de conteúdo e o controle de recursos sensíveis do navegador, respectivamente. Sua inexistência decorre da configuração padrão do Django, que não os aplica automaticamente, evidenciando a necessidade de configurações adicionais, os quais motivaram a etapa subsequente de implementação.

4.5 Intervenções de segurança

Com base no diagnóstico inicial, foram realizadas três intervenções direcionadas à proteção da aplicação conforme o diagnóstico inicial. A primeira consistiu na configuração do `HTTP Strict-Transport-Security` (HSTS), conforme ilustrado na Figura 6, utilizando o `SecurityMiddleware` do Django, responsável por aplicar mecanismos de segurança no ciclo de requisição e resposta. Essa intervenção teve como objetivo forçar o redirecionamento automático das conexões para HTTPS.

A primeira configuração aplicada foi por meio da diretiva `SECURE_HSTS_SECONDS`. Inicialmente, optou-se por definir um valor reduzido para esta diretiva (uma hora), com o objetivo de realizar testes e evitar problemas de bloqueio na aplicação. Isso se faz necessário porque o HSTS força os navegadores a acessarem o site exclusivamente via HTTPS durante o período especificado. Valores muito elevados logo de início podem causar dificuldades de acesso caso o ambiente não esteja completamente preparado. Após os testes e validações, o valor foi ampliado para $15.552.000$ segundos (cerca de 180 dias).

Além disso, foram ativadas as diretivas `SECURE_HSTS_INCLUDE_SUBDOMAINS` e `SECURE_HSTS_PRELOAD`. A primeira estende a política HSTS a todos os subdomínios da aplicação, reforçando a consistência da proteção. A segunda habilita a aplicação ao processo de *preload*, permitindo que o domínio seja inserido na lista oficial de navegadores que já acessam o site diretamente via HTTPS, eliminando a possibilidade de conexões iniciais inseguras.

Figura 6 – Implementação do HSTS no código

```
132
133  #HTTP Strict Transport Security (HSTS)
134  SECURE_HSTS_SECONDS = 15552000
135  SECURE_HSTS_INCLUDE_SUBDOMAINS = True
136  SECURE_HSTS_PRELOAD = True
137
138  #SSL redirect
139  SECURE_SSL_REDIRECT = True
140
```

Fonte: Autoria própria, 2025.

Em seguida, a configuração do `SECURE_SSL_REDIRECT` foi utilizada para garantir que qualquer acesso HTTP fosse automaticamente redirecionado para HTTPS. Essa implementação evita que usuários acessem a aplicação de forma insegura ou tentem estabelecer conexões não criptografadas.

É importante destacar que a configuração adequada desses parâmetros depende diretamente do ambiente de hospedagem e do comportamento do servidor. Cada aplicação *web* possui particularidades e, portanto, é essencial realizar testes incrementais e estudar cuidadosamente o contexto em que a solução será implantada antes de aplicar políticas permanentes, especialmente no caso do HSTS, que pode impactar o acesso à aplicação de forma contínua após sua ativação.

A segunda intervenção consistiu na aplicação do cabeçalho `Content-Security-Policy` (CSP), voltado à mitigação de ataques por injeção de conteúdo, como XSS e *Clickjacking*, por meio da restrição das fontes de *scripts*, estilos e demais recursos. Para a implementação, utilizou-se o pacote `django-csp`, que integra a política ao fluxo de resposta HTTP, adotando-se inicialmente uma configuração mínima e segura antes de ajustes mais específicos.

O processo de implementação do `Content-Security-Policy` iniciou-se com a instalação do pacote `django-csp`, responsável por integrar o controle das diretivas de segurança ao ciclo de resposta do Django. Em seguida, o módulo foi registrado no `INSTALLED_APPS` e o `CSPMiddleware` foi incluído na pilha de *middlewares* — componentes que atuam entre o navegador e a aplicação, interceptando e processando requisições e respostas — assegurando que todas as respostas HTTP passassem a incorporar automaticamente as políticas definidas, conforme apresentado no Quadro 2. Com essa estrutura habilitada, as diretivas passaram a ser configuradas diretamente no arquivo `settings.py`, permitindo o controle centralizado das regras de carregamento de recursos da aplicação.

Quadro 2 – Configuração do django-csp

Passos	**Comando/configuração**
Instalação do pacote	pip install django-csp
Adição no INSTALLED_APPS	INSTALLED_APPS = [... ’csp’, ...]
Middleware habilitado	MIDDLEWARE = [... ’csp.middleware.CSPMiddleware’, ...]

Fonte: Autoria própria, 2025.

A partir desse ponto, o trabalho deixou de ser apenas técnico e passou a envolver análise cuidadosa das necessidades reais do sistema. Optou-se por uma configuração conservadora, utilizando apenas fontes internas (“*self*”), pois o projeto não dependia de bibliotecas externas, *scripts* remotos ou provedores de *Content Delivery Network*

(CDN). A definição das políticas CSP passou a ocorrer diretamente no `settings.py`, priorizando uma configuração centrada em fontes confiáveis e alinhada à arquitetura da aplicação, conforme Figura 7. Para isso, foram definidas diretivas específicas que controlam *scripts*, estilos, imagens, impedindo que o navegador aceite conteúdos externos sem autorização explícita.

Figura 7 – Implementação do CSP no código

Fonte: Autoria própria, 2025.

A política `Content-Security-Policy` foi estruturada com base no princípio do mínimo privilégio, priorizando o carregamento de recursos exclusivamente a partir do próprio domínio da aplicação *self*. As diretivas `default-src`, `script-src` e `style-src` foram configuradas para permitir apenas conteúdos internos, impedindo a execução de código remoto e a importação de estilos externos, o que reduz significativamente a superfície de ataque e mitiga riscos de injeção de *scripts*.

Para os recursos visuais, a diretiva `img-src` incluiu *self* e `"data:"`, viabilizando o uso de imagens embutidas sem liberar origens externas. Além disso, a diretiva `frame-ancestors` foi definida como *none*, bloqueando a incorporação da aplicação em *iframes* e prevenindo ataques de *clickjacking*. Essas configurações garantiram a preservação da funcionalidade visual do sistema, mantendo controle rígido sobre a origem dos conteúdos carregados.

A configuração do *Permissions-Policy* foi a terceira intervenção aplicada e teve como objetivo restringir explicitamente o acesso a APIs sensíveis do navegador. Embora a aplicação não utilize recursos como câmera, microfone ou geolocalização, navegadores modernos disponibilizam essas interfaces a qualquer página, o que aumenta a exposição de ataques caso *scripts* maliciosos sejam executados no cliente.

O processo iniciou-se com a instalação do pacote `django-permissions-policy` responsável por integrar esse cabeçalho ao Django. Após a instalação, o pacote será incorporado por meio do *middleware* próprio, garantindo que o cabeçalho seja aplicado de forma consistente a cada resposta HTTP. Conforme as recomendações oficiais, o middleware foi adicionado imediatamente após o `SecurityMiddleware` do Django, evi-

tando interferência no pipeline e mantendo a lógica de segurança centralizada, como mostra a Figura 8.

Figura 8 – Middlewares aplicados

Fonte: Autoria própria, 2025.

A etapa seguinte consistiu na definição da política de permissões no arquivo `settings.py`. Optou-se por um modelo de restrição total para recursos não utilizados pela aplicação, utilizando listas vazias (`[ ]`) como forma de indicar que nenhum domínio, incluindo o próprio, está autorizado a acessar essas APIs, conforme Figura 9.

Figura 9 – Implementação do Permissions-Policy no código

Fonte: Autoria própria, 2025.

Cada diretiva foi aplicada com base no princípio do mínimo privilégio, segundo o qual apenas recursos estritamente essenciais à funcionalidade devem ser liberados. O acesso à câmera foi bloqueado, pois a aplicação não realiza captura de imagem ou videoconferência, eliminando potenciais vetores de abuso do dispositivo local. Da mesma forma, a geolocalização foi desabilitada para impedir a exposição da posição física do usuário e prevenir práticas de rastreamento. Por fim, a API de microfone também foi restrita, evitando gravações não autorizadas e reduzindo riscos relacionados à ativação indevida do recurso por *scripts* maliciosos.

4.6 Nova avaliação no “Security Headers”

Após a implementação dos cabeçalhos ausentes, a aplicação foi novamente analisada pela ferramenta “Security Headers”, mantendo-se as mesmas condições experimentais do diagnóstico inicial. No segundo teste, a classificação da aplicação passou de C para A+, demonstrando melhora significativa no nível de proteção, com o reconhecimento da presença dos cabeçalhos `Strict-Transport-Security`, `Content-Security-Policy` e `Permissions-Policy`, anteriormente ausentes, passando a integrar de forma consistente a resposta HTTP, conforme Figura 10.

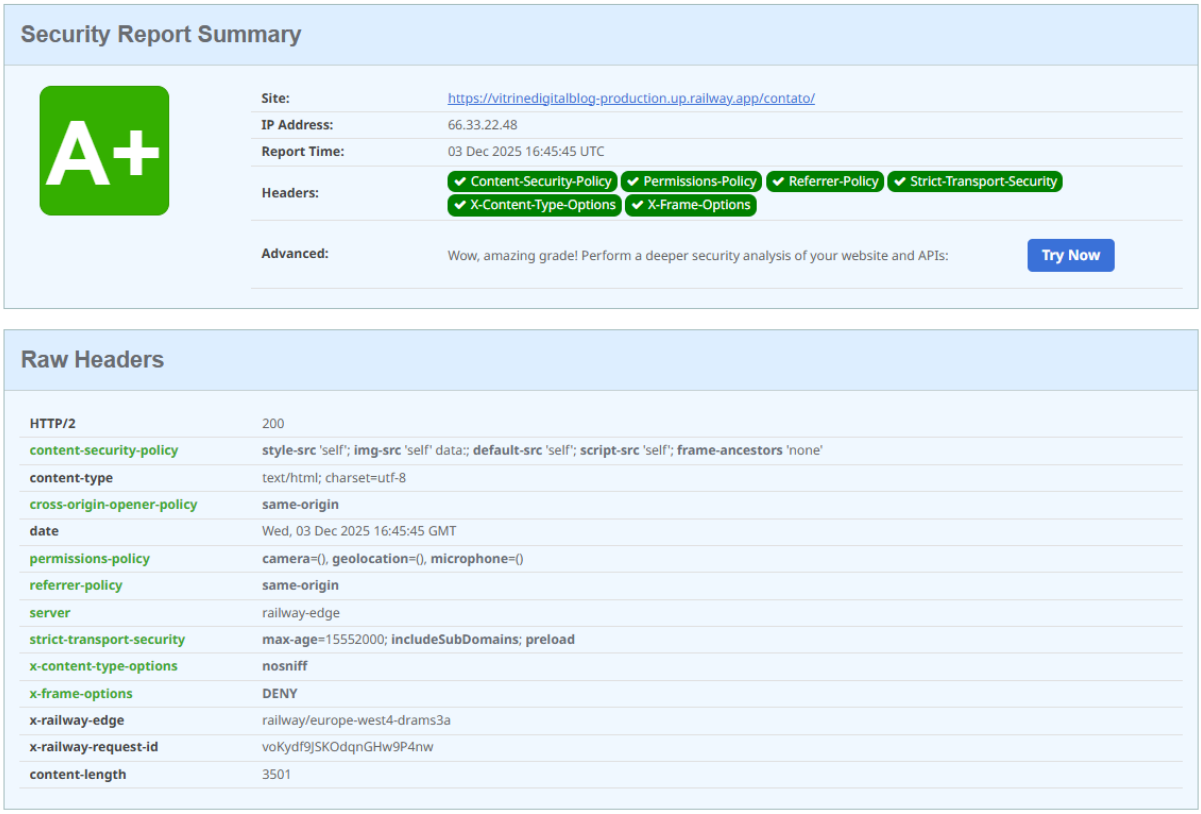
Esse resultado indica que a aplicação passou a operar sob condições mais adequadas do ponto de vista de segurança, com redução de riscos relacionados ao transporte inseguro de dados, à execução de conteúdo não autorizado e ao uso indiscriminado de permissões do navegador. Essa segunda análise indica que a ferramenta passou a reconhecer a presença de todos os seis cabeçalhos relevantes, conforme Tabela 2, validando a eficácia das configurações adicionadas no contexto experimental estabelecido. No contexto da pesquisa, essa evolução não representa a eliminação total de vulnerabilidades, mas demonstra um resultado comparável e verificável da intervenção metodológica, validando a fase experimental e confirmando que medidas simples e bem direcionadas são capazes de elevar substancialmente o nível de proteção em aplicações *web* de pequeno porte.

Tabela 2 – Comparação entre estado inicial e estado pós-intervenção.

Header	**Antes**	**Depois**
X-Frame-Options	Presente	Presente
X-Content-Type-Options	Presente	Presente
Referrer-Policy	Presente	Presente
Strict-Transport-Security	Ausente	Presente
Content-Security-Policy	Ausente	Presente
Permissions-Policy	Ausente	Presente

Fonte: Autoria própria, 2025.

Figura 10 – Segundo relatório gerado pelo Security Headers



Fonte: Autoria própria, 2025.

5 CONCLUSÃO

O presente trabalho teve como objetivo analisar a aplicação de HTTP *Security Headers* em uma aplicação *web* de pequeno porte, buscando compreender em que medida essas políticas contribuem para a mitigação de riscos associados à configuração inadequada de segurança. Para isso, desenvolveu-se um protótipo funcional baseado no *framework* Django, que foi submetido inicialmente a um diagnóstico por ferramenta especializada, revelando um cenário típico de aplicações simples: a presença de alguns cabeçalhos padrões do *framework* e a ausência de políticas de segurança adicionais necessárias ao fortalecimento da proteção da aplicação.

A partir desse ponto inicial, foram aplicadas intervenções controladas, resultando na implementação dos cabeçalhos HTTP `Strict-Transport-Security`, `Content-Security-Policy` e `Permissions-Policy`, seguida de nova avaliação após as modificações. Os resultados obtidos demonstraram que a adoção desses mecanismos elevou de forma expressiva o nível de proteção da aplicação. A mudança da classificação de C para A+ evidenciou que políticas de segurança relativamente simples, mas corretamente configuradas, são capazes de reduzir superfícies de ataque e tornar o ambiente mais resiliente a exploração de vulnerabilidades.

Dessa forma, pode-se afirmar que os objetivos propostos no início deste trabalho foram atingidos: os *headers* selecionados foram identificados e compreendidos, um protótipo foi construído para fins de experimentação, as políticas foram aplicadas de acordo com recomendações técnicas reconhecidas e a comparação entre estados pré e pós-implementação permitiu avaliar o impacto das intervenções realizadas. O estudo confirmou, portanto, que a utilização de HTTP *Security Headers* não apenas fortalece a aplicação, mas pode fazê-lo de maneira acessível para desenvolvedores menos experientes e para projetos que não dispõem de equipes de segurança especializadas.

Ainda que os resultados sejam satisfatórios dentro do escopo definido, é importante destacar que a implementação de *headers* não representa uma solução completa ou definitiva para o problema da segurança em aplicações *web*. Cada sistema apresenta características próprias, de modo que a escolha e a calibragem das políticas devem ser adequadas ao perfil da aplicação, considerando fatores como natureza do conteúdo, contexto de uso e exposição a recursos externos. A pesquisa mostrou que existem ferramentas simples e de fácil acesso que facilitam a identificação de lacunas e auxiliam o processo de correção. Também evidenciou que diferentes cabeçalhos atuam de forma complementar e que seu efeito, embora significativo, limita-se ao controle do comportamento na camada HTTP.

5.1 Trabalhos Futuros

Por esse motivo, recomenda-se que futuros trabalhos explorem a efetividade prática dessas configurações por meio de abordagens mais profundas, como testes de invasão (*pentesting*) direcionados aos vetores de ataque já mencionados no trabalho. Avaliar como cada *header* reage diante de cenários reais de exploração — incluindo simulações de ataques ou análise por *scanners* automatizados de vulnerabilidade — permitiria ampliar a compreensão sobre o papel desses mecanismos na defesa de aplicações *web* e contribuir para o desenvolvimento de estratégias integradas de proteção em ambientes de escala reduzida. Dessa forma, o estudo aqui apresentado cumpre sua finalidade exploratória e abre caminhos para investigações que agreguem camadas adicionais de segurança e validação prática ao contexto analisado.

REFERÊNCIAS

AWATI, Rahul; GILLIS, Alexander S. What is a Web Server and How Does It Work? **TechTarget**, 2024. Online article. Disponível em: `https://www.techtarget.com/whatis/definition/Web-server`.

BAKER, Matthew. **Secure Web Application Development: A Hands-On Guide with Python and Django**. No Starch Press, 2024. ISBN 978-1718502665.

BRYANT, Antony. What the Web Has Wrought. **Informatics**, v. 7, n. 2, p. 15, 2020. DOI: `10.3390/informatics7020015`.

DUCKETT, Jon. **HTML and CSS: Design and Build Websites**. Wiley, 2011. ISBN 978-1118008188.

FIELDING, Roy T.; RESCHKE, Julian. **RFC 7231: Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content**. 2014. Disponível em: `https://datatracker.ietf.org/doc/html/rfc7231`.

FLANAGAN, David. **JavaScript: The Definitive Guide**. 7. ed.: O'Reilly Media, 2020. ISBN 978-1491952023.

FOUNDATION, OWASP. **HTTP Strict Transport Security Cheat Sheet**. 2025. `https://cheatsheetseries.owasp.org/cheatsheets/HTTP_Strict_Transport_Security_Cheat_Sheet.html`.

GOURLEY, David *et al.* **HTTP: The Definitive Guide**. O'Reilly Media, 2002. ISBN 978-1565925090.

KHAIRE, Prajkta *et al.* Web Server Analysis. **IJRASET – International Journal for Research in Applied Science & Engineering Technology**, v. 10, p. 1–8, 9 2022. DOI: `10.22214/ijraset.2022.41472`.

KISHNANI, Urvashi; DAS, Sanchari. Securing the Web: Analysis of HTTP Security Headers in Popular Global Websites. **arXiv**, abs/2410.14924, 2024. DOI: `10.48550/arXiv.2410.14924`.

LI, Tao *et al.* A Survey on Web Application Testing: A Decade of Evolution. **arXiv**, abs/2412.10476, 2024. DOI: `10.48550/arXiv.2412.10476`.

LUTZ, Mark. **Learning Python: Powerful Object-Oriented Programming**. 6. ed.: O'Reilly Media, 2023. ISBN 978-1492051367.

MEYER, Eric A.; WEYL, Estelle. **CSS: The Definitive Guide**. 5. ed.: O'Reilly Media, 2023. ISBN 978-1098117610.

MOZILLA FOUNDATION. **Security/HTTP Headers**. 2023. Disponível em: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers.

NAWROCKI, M.; KOŁODZIEJ, J. Vulnerabilities of Web Applications: Good Practices and New Trends. **ACIG Journal of Web Security**, v. 3, n. 2, 2024. DOI: 10.60097/ACIG/199521.

OWASP FOUNDATION. **Clickjacking**. 2025. Disponível em: https://owasp.org/www-community/attacks/Clickjacking.

OWASP FOUNDATION. **Clickjacking Defense Cheat Sheet**. 2025. https://cheatsheetseries.owasp.org/cheatsheets/Clickjacking_Defense_Cheat_Sheet.html.

OWASP FOUNDATION. **Content Security Policy Cheat Sheet**. 2025. https://cheatsheetseries.owasp.org/cheatsheets/Content_Security_Policy_Cheat_Sheet.html.

OWASP FOUNDATION. **Cross Site Scripting (XSS)**. 2021. Disponível em: https://owasp.org/www-community/attacks/xss/.

OWASP FOUNDATION. **HTTP Headers Cheat Sheet — X-Content-Type-Options**. 2025. https://cheatsheetseries.owasp.org/cheatsheets/HTTP_Headers_Cheat_Sheet.html.

OWASP FOUNDATION. **OWASP Secure Headers Project**. 2025. https://owasp.org/www-project-secure-headers/.

OWASP FOUNDATION. **OWASP Top 10 – 2021: The Ten Most Critical Web Application Security Risks**. 2021. Disponível em: https://owasp.org/Top10/.

OWASP FOUNDATION. **Permissions Policy Cheat Sheet**. 2025. https://cheatsheetseries.owasp.org/cheatsheets/Permissions_Policy_Cheat_Sheet.html.

OWASP FOUNDATION. **Referrer Policy — OWASP Cheat Sheet**. 2025. https://cheatsheetseries.owasp.org/cheatsheets/Referrer_Policy_Cheat_Sheet.html.

PROJECT, OWASP Secure Headers. **HTTP Security Response Headers Cheat Sheet**. 2025. https://cheatsheetseries.owasp.org/cheatsheets/HTTP_Headers_Cheat_Sheet.html.

SECURITYHEADERS. **SecurityHeaders.com — HTTP Security Header Scanner**. 2025. https://securityheaders.com/.

SILVA, Juliano José da. **Segurança em aplicações web: principais vulnerabilidades e estratégias de prevenção**. 2011. Monografia – Universidade Federal do Paraná, Curitiba, PR, Brasil.

TARIQ, Zain; ZAINAB, Bint e; HUSSAIN, Muhammad Zunnurain. Evaluating the Effectiveness and Resilience of SSL/TLS, HTTPS, IPSec, SSH, and WPA/WPA2 in Safeguarding Data Transmission. **UCP Journal of Engineering & Information Technology**, v. 1, n. 2, p. 01–07, 2024. DOI: 10.24312/ucp-jeit.01.02.144.

TECHNOLOGIES, Akamai. **State of the Internet 2025 – Web Application & API Attack Report**. 2025. Relatório analítico sobre o aumento de ataques a aplicações web e APIs.

WAZLAWICK, Raul Sidnei. **Metodologia de pesquisa para ciência da computação**. 3. ed.: Elsevier, 2020. ISBN 978-85-504-0000-0.