UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

MATHEUS SYDOR

OWNEDBOX: UMA PLATAFORMA DE ENSINO PARA SEGURANÇA WEB OFENSIVA E DEFENSIVA

GUARAPUAVA

2026

MATHEUS SYDOR

OWNEDBOX: UMA PLATAFORMA DE ENSINO PARA SEGURANÇA WEB OFENSIVA E DEFENSIVA

OwnedBox: A Teaching Platform for Offensive and Defensive Web Security

Trabalho de Conclusão de Curso apresentado como requisito para obtenção do título de Técnologo em Tecnologia em Sistemas para Internet do Curso Superior de Tecnologia em Sistemas para Internet da Universidade Tecnológica Federal do Paraná.

Orientador: Prof. Dr. Roni Fabio Banaszewski

GUARAPUAVA

2026

RESUMO

A crescente complexidade dos sistemas digitais e o aumento de ataques cibernéticos tornam essencial o desenvolvimento de ferramentas educacionais que auxiliem estudantes e profissionais a compreender vulnerabilidades e práticas de segurança. Este trabalho apresenta o projeto e desenvolvimento da plataforma **OwnedBox**, uma aplicação web de código aberto voltada ao ensino de segurança ofensiva e defensiva por meio de laboratórios controlados. A solução organiza o conteúdo em módulos práticos que abordam vulnerabilidades recorrentes no OWASP Top 10, como *SQL Injection*, *Cross-Site Scripting* (XSS) e upload de arquivos maliciosos, permitindo ao usuário explorar cenários reais de pentest de forma segura. O sistema implementa autenticação própria, instanciação sob demanda de laboratórios e validação de tokens, além de um painel de progresso do usuário. Cada laboratório é executado em um contêiner Docker isolado, no qual uma flag exclusiva é injetada por variável de ambiente; ao explorar a vulnerabilidade e capturar a flag, o usuário submete o token correspondente, que é validado pelo sistema e registra automaticamente a conclusão do módulo. O desenvolvimento seguiu metodologias ágeis (Scrum, Kanban e MoSCoW) e utilizou tecnologias como Laravel, Docker e SQLite. Os resultados demonstram que a plataforma oferece um ambiente didático eficaz para aprendizado prático em cibersegurança, contribuindo para a formação de profissionais qualificados na área.

Palavras-chave: cibersegurança; pentest; aplicações web vulneráveis; educação em segurança; owasp.

ABSTRACT

The increasing complexity of digital systems and the rise of cyberattacks highlight the need for educational tools that help students and professionals understand vulnerabilities and security practices. This work presents the design and development of **OwnedBox**, an open-source web platform aimed at teaching offensive and defensive web security through controlled laboratories. The solution organizes content into practical modules covering recurring OWASP Top 10 vulnerabilities, such as *SQL Injection*, *Cross-Site Scripting* (XSS), and malicious file uploads, enabling users to safely explore realistic pentesting scenarios. The system implements custom authentication, on-demand laboratory provisioning, and token validation, along with a user progress panel. Each laboratory runs in an isolated Docker container into which a unique flag is injected through an environment variable; after exploiting the vulnerability and capturing the flag, the user submits the corresponding token, which is validated by the system and automatically records the module completion. The development followed agile methodologies (Scrum, Kanban, and MoSCoW) and employed technologies such as Laravel, Docker, and SQLite. Results indicate that the platform provides an effective educational environment for practical cybersecurity training, contributing to the qualification of professionals in the field.

Keywords: cybersecurity; pentesting; vulnerable web applications; security education; owasp.

LISTA DE FIGURAS

LISTA DE CÓDIGOS-FONTE

LISTA DE ABREVIATURAS E SIGLAS

Abreviaturas

pentest	Teste de Intrusão (*penetration test*)

Siglas

CDN	*Content Delivery Network* (Rede de Distribuição de Conteúdo)
CSRF	*Cross-Site Request Forgery* (Falsificação de Requisição entre Sites)
CTF	*Capture The Flag* (Capture a Bandeira)
DVWA	*Damn Vulnerable Web Application* (Aplicação Web Extremamente Vulnerável)
GDPR	*General Data Protection Regulation* (Regulamento Geral de Proteção de Dados da União Europeia)
HTB	*Hack The Box*
HU	História de Usuário
LGPD	Lei Geral de Proteção de Dados
MVP	Produto Mínimo Viável, do inglês *Minimum Viable Product*
ORM	*Object-Relational Mapping* (Mapeamento Objeto-Relacional)
OWASP	*Open Web Application Security Project* (Projeto Aberto de Segurança em Aplicações Web)
PHP	*PHP: Hypertext Preprocessor*
TCC	Trabalho de Conclusão de Curso
THM	*TryHackMe* (Tente me Hackear)
URL	*Uniform Resource Locator* (Localizador Uniforme de Recursos)
WWW	*World Wide Web* (Rede Mundial de Computadores)
XSS	*Cross-Site Scripting* (Script entre Sites)

SUMÁRIO

1 INTRODUÇÃO

Este trabalho tem como objetivo principal apresentar o projeto de Trabalho de Conclusão de Curso (TCC) que pretende abordar as principais vulnerabilidades web e o desenvolvimento de um sistema web de aprendizado na área de cibersegurança ofensiva e defensiva. O sistema proposto tem como objetivo criar um ambiente virtual para que estudantes e entusiastas de área da segurança digital possam aprender e testar suas habilidades.

1.1 Considerações iniciais

A segurança da informação tem se consolidado, nas últimas décadas, como um dos pilares das empresas, governos e sociedade em um mundo cada vez mais conectado com o meio digital. Quando surgiu a ARPANET, no final da década de 1960, inicialmente desenvolvida para fins militares, até a sua popularidade comercial na década de 1990, o meio digital trouxe inúmeros benefícios para a população, mas também abriu espaço para a proliferação de ameaças cibernéticas (CASTELLS, 2003). A popularização da *World Wide Web* (Rede Mundial de Computadores) (WWW), junto ao crescimento do comércio eletrônico e das redes sociais, ampliou consideravelmente os vetores de ataque, fazendo com que a cibersegurança tenha se tornado uma área estratégica para organizações e indivíduos (STALLINGS, 2019)

Por volta de 1980 já existiam estudos e a difusão dos primeiros vírus e softwares maliciosos, impulsionando o surgimento dos antivírus comerciais. Nos anos de 1990 e 2000, as ameaças começaram a evoluir, ataques direcionados a sistemas corporativos, bancos de dados sensíveis e infraestruturas críticas passaram a ser visados por cibercriminosos (KASPERSKY, 2020). Com isso, surgiram formas de prevenir e remediar essas vulnerabilidades, surgiram ferramentas como *firewalls*, sistemas de detecção e prevenção de intrusão, além de leis cada vez mais rígidas, como o *General Data Protection Regulation* (Regulamento Geral de Proteção de Dados da União Europeia) (GDPR) e, no Brasil, a Lei Geral de Proteção de Dados (LGPD) que reforçam a necessidade de proteger o sistema e seus usuários (União Europeia, 2016; BRASIL, 2018).

Nos últimos anos, a segurança da informação consolidou-se como um pilar crítico para a sociedade moderna. Impulsionada por eventos recentes como a pandemia de COVID-19, a rápida transição para o trabalho remoto e o crescimento exponencial do comércio eletrônico aumentaram drasticamente a dependência global de aplicações web. Consequentemente, a proteção de dados sensíveis passou a ser uma prioridade inegociável para indivíduos, empresas e governos. Nesse cenário, relatórios como *Open Web Application Security Project* (Projeto Aberto de Segurança em Aplicações Web) (OWASP) Top Ten (OWASP Foundation, 2021) evidenciam que vulnerabilidades como SQL Injection, *Cross-Site Scripting* (Script entre Sites) (XSS) e upload de arquivos maliciosos permanecem entre as mais exploradas em ataques reais, reforçando a necessidade de mecanismos eficazes de prevenção e mitigação.

Empresas recorrem ao Teste de Intrusão (*penetration test*) (pentest), um teste de segurança cibernética que tem como principal objetivo simular um ataque para encontrar vulnerabilidades em um sistema. Este teste é feito de maneira legal e com contrato, garantindo que nenhuma vulnerabilidade explorada causará danos para os clientes da empresa e garantindo que o sistema está seguro (IBM, 2025). Além disso, a prática possui caráter educacional, sendo replicada em laboratórios controlados e plataformas específicas que oferecem ambientes seguros e éticos para estudantes e profissionais desenvolverem suas habilidades. Dessa forma, é possível praticar técnicas ofensivas sem riscos jurídicos ou danos a sistemas de terceiros, em conformidade com o Marco Civil da Internet e a Lei Carolina Dieckmann (BRASIL, 2014; BRASIL, 2012).

Existem diversas iniciativas internacionais, como as plataformas *Hack The Box* (HTB) e *TryHackMe* (Tente me Hackear) (THM), que têm desempenhado um papel importante no ensino de segurança ofensiva e defensiva. No entanto, essas ferramentas apresentam barreiras relacionadas ao idioma, custos de acesso e falta de contextualização com a realidade brasileira. Além disso, são direcionadas prioritariamente ao mercado global, o que reforça a carência, no Brasil, de soluções didáticas acessíveis, em português, capazes de facilitar o aprendizado individual e de atender tanto a ambientes acadêmicos quanto corporativos.

Devido a isso, é importante investir em metodologias inovadoras de ensino em cibersegurança. A segurança digital não é somente um diferencial competitivo entre as empresas, mas uma necessidade. Laboratórios controlados podem ajudar estudantes, entusiastas e profissionais a entenderem as técnicas ofensivas de exploração de vulnerabilidades, e ao mesmo tempo permiti-los a desenvolver habilidades defensivas para proteger sistemas. Tendo isso em vista, a OwnedBox foi desenvolvida como uma solução gratuita e de código aberto, voltada ao aprendizado prático e à introdução de estudantes e entusiastas aos conceitos fundamentais de cibersegurança ofensiva e defensiva. A plataforma permite que instituições de ensino e organizações adaptem os laboratórios às suas necessidades específicas e colaborem com a evolução da aplicação web.

1.2 Objetivos

Nesta seção é apresentado o objetivo geral deste trabalho.

1.2.1 Objetivo geral

Desenvolver uma aplicação web denominada **OwnedBox**, que funcione como um laboratório de ensino em cibersegurança ofensiva e defensiva, possibilitando a exploração prática de vulnerabilidades web comuns em um ambiente controlado, seguro e didático.

1.2.2 Objetivos específicos

Os objetivos específicos do projeto *OwnedBox* foram definidos de forma a orientar o desenvolvimento das principais funcionalidades do sistema e garantir que o protótipo atenda aos propósitos educacionais e técnicos propostos. São eles:

- Implementar o sistema de autenticação e cadastro de usuários.
- Desenvolver módulos de aprendizado ofensivo e defensivo.
- Criar laboratórios interativos com vulnerabilidades simuladas.
- Gerar *tokens* de validação para comprovar a conclusão dos desafios.
- Construir uma página de perfil com estatísticas e progresso.
- Implementar contramedidas defensivas para cada vulnerabilidade.
- Disponibilizar um ambiente seguro, isolado e instrutivo para prática.

2 TRABALHOS RELACIONADOS

O ensino de segurança da informação tem avançado significativamente nos últimos anos, impulsionado pela disponibilidade de plataformas que simulam ambientes vulneráveis e permitem a prática de técnicas ofensivas e defensivas. Essas soluções têm contribuído para o aprendizado prático de estudantes, profissionais e entusiastas, mas muitas delas apresentam barreiras relacionadas ao idioma, aos custos de acesso ou à falta de contextualização com a realidade acadêmica brasileira. Nesse cenário, destaca-se a importância de analisar ferramentas amplamente utilizadas na área, especialmente aquelas que influenciaram a concepção da **OwnedBox**. Entre as principais iniciativas encontram-se o HTB, o THM e o *Damn Vulnerable Web Application* (Aplicação Web Extremamente Vulnerável) (DVWA), cada uma com características próprias e relevância significativa no ecossistema de cibersegurança.

2.1 Hack The Box

O *Hack The Box* é uma das plataformas mais reconhecidas internacionalmente para prática de *pentest* e *red teaming*. Lançado em 2017, o HTB oferece máquinas virtuais com diferentes níveis de vulnerabilidades, oferecendo aos alunos a oportunidade de praticar técnicas de exploração em um ambiente controlado e constantemente atualizado (Hack The Box Ltd., 2025)

Pontos Positivos:

- Grande quantidade de laboratórios e desafios, com atualização frequente.
- Comunidade ativa e fóruns de suporte.
- Reconhecimento no mercado, sendo referência em treinamentos de cibersegurança.

Pontos Negativos:

- Interface totalmente em inglês, dificultando o uso por iniciantes brasileiros.
- Curva de aprendizado elevada, exigindo conhecimentos prévios.
- Muitas funcionalidades só estão disponíveis mediante assinatura paga.

2.2 TryHackMe

O *TryHackMe* é uma plataforma de ensino de segurança cibernética com foco em *gamificação* e aprendizado gradual. Ela disponibiliza laboratórios com tutoriais guiados e desafios práticos. Ela se destaca pelo foco em atividades do tipo *Capture The Flag* (Capture a Bandeira)

(CTF), modalidade competitiva em que os participantes exploram falhas em laboratórios virtuais para conquistar pontos e melhorar sua posição em rankings globais (TRYHACKME, 2025)

Pontos Positivos:

- Abordagem didática com tutoriais passo a passo.
- Estrutura de aprendizado organizada em trilhas temáticas.
- Requer apenas um navegador para começar a usar (sem necessidade de configuração avançada).

Pontos Negativos:

- Versão gratuita limitada, com acesso restrito a laboratórios.
- Conteúdo em inglês, o que pode ser um obstáculo para estudantes brasileiros.

2.3 Damn Vulnerable Web Application

O *DVWA* é uma aplicação web propositalmente vulnerável, desenvolvida para fins educacionais. É uma ferramenta bastante utilizada em cursos de segurança, permitindo a exploração de falhas em diferentes níveis de dificuldade (RandomStorm, 2025).

Pontos Positivos:

- Gratuito e de código aberto.
- Permite testar vulnerabilidades clássicas da web (SQL Injection, XSS, *File Upload*, *Cross-Site Request Forgery* (Falsificação de Requisição entre Sites) (CSRF), etc.).
- Flexibilidade para instalação em ambiente local.

Pontos Negativos:

- Interface simples e pouco amigável.
- Ausência de trilhas pedagógicas organizadas, dificultando o uso em contexto acadêmico formal.

3 MATERIAIS E MÉTODOS

3.1 Materiais

O projeto **OwnedBox** foi desenvolvido utilizando o **framework Laravel**, que oferece uma estrutura robusta e segura para a criação de aplicações web modernas. O **Blade**, motor de *templates* nativo do Laravel, foi empregado para a construção das interfaces dinâmicas e responsivas, permitindo a integração entre o *front-end* e o *back-end* de forma eficiente. Para a estilização das interfaces, foi utilizado o *framework* CSS **Bootstrap**, incorporado às páginas por meio de *Content Delivery Network* (Rede de Distribuição de Conteúdo) (CDN), o que dispensou a instalação local da biblioteca e agilizou o uso de seus componentes prontos, como formulários, botões e *cards*.

O banco de dados **SQLite** foi adotado para o armazenamento das informações, devido à sua confiabilidade, desempenho e ampla compatibilidade com o Laravel. O acesso aos dados foi realizado por meio do **Eloquent**, o *Object-Relational Mapping* (*Object-Relational Mapping* (Mapeamento Objeto-Relacional) (ORM)) nativo do Laravel, que mapeia as tabelas do banco em classes de modelo e permite manipular os registros como objetos, dispensando a escrita manual de consultas SQL para as operações comuns da aplicação.

O gerenciamento das dependências do *back-end* foi feito com o **Composer**, gerenciador de pacotes do PHP, responsável por instalar e manter atualizadas as bibliotecas utilizadas pelo projeto, incluindo o próprio *framework* Laravel. Para o controle de versão do código, foi utilizado o **Git**, com o repositório hospedado na plataforma **GitHub**, possibilitando o acompanhamento das alterações e a colaboração entre desenvolvedores.

O **Docker** foi implementado para a criação de ambientes de desenvolvimento padronizados, assegurando que a aplicação funcionasse de maneira consistente em diferentes sistemas operacionais. Além disso, o método **Kanban** no **Github Projects** foi utilizado para o gerenciamento das tarefas e acompanhamento do progresso do projeto, promovendo uma organização visual e clara das etapas de desenvolvimento.

3.2 Métodos

O desenvolvimento do sistema **OwnedBox** foi conduzido de forma iterativa e incremental, seguindo princípios das metodologias ágeis.

Inicialmente, foi realizada uma etapa de levantamento e refinamento dos requisitos, permitindo identificar as funcionalidades essenciais do sistema e compreender as necessidades dos usuários. Em seguida, foi feita a modelagem preliminar do banco de dados, assim como o planejamento das telas, rotas, e componentes que foram implementados no *framework* Lara-

vel. A camada de apresentação foi estruturada com Blade, priorizando critérios de usabilidade, clareza e experiência do usuário.

A gestão das tarefas foi realizada por meio do método Kanban, utilizando quadros no *GitHub Projects*. As atividades foram distribuídas em colunas representando seu estado durante cada *sprint*, como **To Do**, **In Progress**, **Review** e **Done**. Esse modelo de acompanhamento permitiu visualizar o progresso e priorizar demandas conforme o avanço do desenvolvimento.

A condução do projeto seguiu uma adaptação da metodologia *Scrum*, adequada ao contexto de desenvolvimento individual. As *sprints* tiveram duração de duas semanas, e o orientador desempenhou os papéis de *Product Owner* e *Scrum Master*, enquanto o aluno atuou como *Developer*, responsável pela implementação das funcionalidades e acompanhamento do andamento das atividades. Com essa combinação de métodos, foi possível conduzir o desenvolvimento do sistema de forma organizada e incremental.

4 ANÁLISE DO SISTEMA

Este trabalho apresenta o desenvolvimento da aplicação web **OwnedBox**, um ambiente de ensino gratuito de código aberto voltado para a prática de segurança web ofensiva e defensiva. O sistema foi disponibilizado como um **protótipo funcional Produto Mínimo Viável, do inglês *Minimum Viable Product* (MVP)**, contemplando três vulnerabilidades web entre as mais recorrentes segundo o OWASP Top 10 (OWASP Foundation, 2021): *SQL Injection*, *Cross-Site Scripting* e *Upload de Arquivos Maliciosos*. Cada vulnerabilidade foi apresentada em um módulo específico, permitindo que o estudante explore a falha ofensivamente, receba um **token de validação** ao concluir o desafio e, em seguida, visualize contramedidas defensivas que demonstrem como corrigir o problema.

A aplicação foi distribuída como **código aberto no GitHub**, acompanhada de *snapshots* das máquinas de laboratório, possibilitando que qualquer usuário instale o sistema em seu próprio ambiente local ou servidor. Para viabilizar essa instalação, foi incluída uma **página de instruções com os requisitos necessários**, de modo a facilitar a configuração mesmo por usuários com pouca experiência em implantação de aplicações web. Essa escolha elimina a necessidade de manter a plataforma publicada em um domínio público, reduz riscos de segurança e reforça o caráter acadêmico e didático do projeto.

Por se tratar de um projeto de **código aberto**, os utilizadores podem contribuir ativamente com a evolução da aplicação, propondo melhorias, corrigindo falhas e desenvolvendo **laboratórios personalizados** que atendam a demandas específicas. Essa abertura à colaboração amplia o potencial educacional da plataforma, tornando-a um recurso dinâmico e em constante aperfeiçoamento pela comunidade.

Além disso, a aplicação conta com uma **página de estatísticas do usuário**, na qual é possível acompanhar o progresso individual, visualizar os *tokens* coletados e obter uma visão clara do desempenho durante os estudos.

4.1 Perfis de usuários

A aplicação web contempla, nesta etapa de desenvolvimento, apenas um perfil de usuário: o **Estudante** (ou Usuário Comum). Esse perfil é responsável por realizar o cadastro e autenticação na plataforma, acessar os módulos de aprendizado, explorar as vulnerabilidades simuladas e visualizar os *tokens* de validação obtidos ao concluir os desafios propostos.

4.2 Funcionalidades essenciais

Nesta seção são apresentadas as histórias de usuário classificadas de acordo com o método MoSCoW. Para o escopo deste projeto, foram priorizados apenas os requisitos *Must*

Have, considerados essenciais para o funcionamento mínimo viável do sistema. Os demais requisitos — *Should Have*, *Could Have* e *Won't Have* — encontram-se detalhados no Apêndice A.

Cada funcionalidade está estruturada no formato padrão de História de Usuário (HU) — Como [perfil], quero [funcionalidade] para [objetivo] — e identificada por um código no padrão HUxx (como HU01, HU02, HU03, e assim por diante), o que possibilita uma melhor organização, rastreabilidade e referência ao longo do trabalho.

- **HU01**: **Como** um estudante, **quero** realizar meu cadastro informando dados básicos (nome, e-mail e senha), **para** que eu possa acessar a aplicação web e participar dos módulos de aprendizado.
- **HU02**: **Como** estudante cadastrado, **quero** efetuar *login* com minhas credenciais, **para** acessar meu perfil e os módulos de aprendizado.
- **HU03**: **Como** estudante autenticado, **quero** acessar meu perfil com informações de desempenho, **para** acompanhar meu progresso na aplicação web.
- **HU04**: **Como** estudante de cibersegurança, **quero** acessar laboratórios com vulnerabilidades simuladas, **para** praticar técnicas ofensivas em ambiente controlado.
- **HU05**: **Como** estudante da aplicação web, **quero** visualizar contramedidas para cada vulnerabilidade, **para** aprender como corrigir falhas e aplicar boas práticas defensivas.
- **HU06**: **Como** estudante que conclui um desafio, **quero** receber um *token* de validação, **para** comprovar que atingi o objetivo proposto.
- **HU07**: **Como** estudante da plataforma, **quero** acessar relatórios de desempenho, **para** acompanhar minha evolução ao longo do tempo.
- **HU08**: **Como** estudante da OwnedBox, **quero** ter garantia de que meus dados estão protegidos, **para** utilizar a aplicação web sem riscos.
- **HU09**: **Como** estudante, **quero** ter acesso a uma página com instruções de instalação e requisitos do sistema, **para** configurar a aplicação web corretamente em meu ambiente local.

5 PROJETO DO SISTEMA

Este capítulo apresenta o projeto do **OwnedBox**, descrevendo como o sistema foi organizado e preparado para sua implementação. São detalhadas as decisões iniciais de arquitetura, o planejamento das primeiras sprints, os protótipos de interface e a modelagem do banco de dados, estabelecendo a estrutura necessária para o desenvolvimento da aplicação. O objetivo é definir, de forma clara e objetiva, os principais componentes do sistema e o fluxo de trabalho que orientou sua construção.

5.1 Identidade visual

A identidade visual do *OwnedBox* foi concebida para refletir o propósito central da plataforma: oferecer um ambiente seguro, intuitivo e imersivo para o estudo de vulnerabilidades e práticas de pentest. Assim como apresentado na análise anterior, o sistema prioriza uma estética moderna alinhada a plataformas de cibersegurança, garantindo clareza na navegação, coerência visual e foco no conteúdo técnico. O *design* adotado também aproxima o usuário da experiência típica de laboratórios profissionais, como os utilizados em treinamentos de *red team* e ambientes práticos de segurança ofensiva.

5.1.1 Paleta de cores

A paleta de cores do *OwnedBox* foi definida para transmitir profissionalismo, tecnologia e concentração. Inspirada em ambientes comparáveis ao *Hack The Box*, a interface utiliza predominantemente tons escuros, permitindo ao usuário explorar os laboratórios por longos períodos sem desconforto visual. As principais escolhas são:

- **Background:** tons de cinza-escuro e preto suave, fornecendo contraste adequado e criando um ambiente imersivo semelhante a terminais e interfaces técnicas.
- **Texto e ícones:** branco e cinza claro, garantindo legibilidade em ambiente dark.
- **botões e barras de progresso:** azul claro e verde limão, destacando visualmente o avanço do usuário nos módulos.

A Tabela 1 apresenta a paleta de cores utilizada, com seus respectivos códigos e representações visuais.

Tabela 1 – Paleta de cores do OwnedBox

Elemento	Cor	Hexadecimal
Background principal		#121c21
Cinza secundário		#192b33
Texto branco		#FFFFFF
Texto cinza claro		#D1D5DB
Azul (progresso e botões)		#1294D4
verde limão		#56D412

Essa paleta equilibra estética e funcionalidade. Os tons escuros favorecem o foco e reduzem distrações, enquanto as cores destacadas reforçam a percepção de ação, progresso e segurança dentro da plataforma.

5.1.2 Tipografia

A tipografia utilizada no *OwnedBox* prioriza legibilidade e modernidade, características essenciais em sistemas de estudo técnico. A fonte adotada foi a **Space Grotesk**, disponibilizada pelo *Google Fonts*, empregada nos pesos *regular* (400), *medium* (500) e *bold* (700) ao longo das telas do sistema. Trata-se de uma fonte *sans-serif* de padrão minimalista, amplamente utilizada em *dashboards* e plataformas educacionais de segurança da informação, cuja escolha reforça a clareza e a precisão esperada em aplicações voltadas para profissionais e estudantes da área. A Figura 1 apresenta a fonte Space Grotesk utilizada no projeto.

Figura 1 – Fonte tipográfica Space Grotesk utilizada no OwnedBox.

5.1.3 Logotipo

O logotipo do *OwnedBox* é apresentado na Figura 2. Ele foi desenvolvido com o objetivo de refletir simultaneamente segurança, investigação e exploração, sendo esses os pilares fundamentais do sistema. O cubo presente no símbolo contém elementos como um cadeado, um escudo e uma lupa, representando análise e busca por vulnerabilidades, respectivamente. Sua composição geométrica e as cores utilizadas alinham-se ao tema geral do sistema, reforçando a sensação de ambiente técnico e confiável.

Figura 2 – Logotipo do sistema OwnedBox.

5.2 Prototipação de telas

Com base nos requisitos levantados e priorizados pela técnica *MoSCoW*, foram desenvolvidos os protótipos de interface do sistema utilizando a ferramenta *Figma*.

A prototipação teve como objetivo representar, de forma visual e interativa, os principais fluxos de navegação da aplicação, dando uma visão da experiência do usuário antes da implementação definitiva.

Nesta seção, são apresentados os protótipos das telas fundamentais do sistema, incluindo: criação de conta, autenticação, módulos disponíveis, perfil do usuário e sessões de vulnerabilidade. Todas seguem o padrão visual definido para o projeto, priorizando simplicidade, coerência visual e clareza nos fluxos de interação.

5.2.1 Tela de criação de conta

Na Figura 3 é apresentado o protótipo da tela de criação de conta, projetada para permitir o registro de novos usuários, solicitando informações essenciais como nome, e-mail e senha. O *layout* prioriza a clareza dos campos e o fluxo natural de preenchimento, além de incluir mensagens de validação para garantir a segurança das informações inseridas.

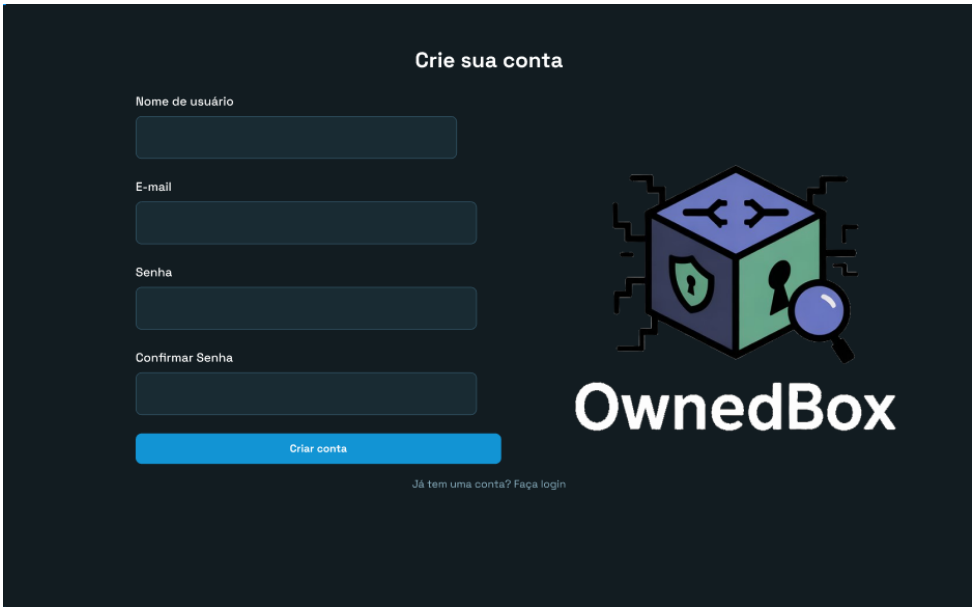


Figura 3 – Protótipo da tela de criação de conta desenvolvido no Figma.

5.2.2 Tela de *login*

Conforme a Figura 4, a tela de *login* foi desenhada para oferecer uma autenticação simples e direta, mantendo consistência visual com a tela de registro. Além dos campos de e-mail e senha, foi previsto um sistema de aviso para tentativas inválidas, conforme o requisito de segurança definido na etapa de levantamento.

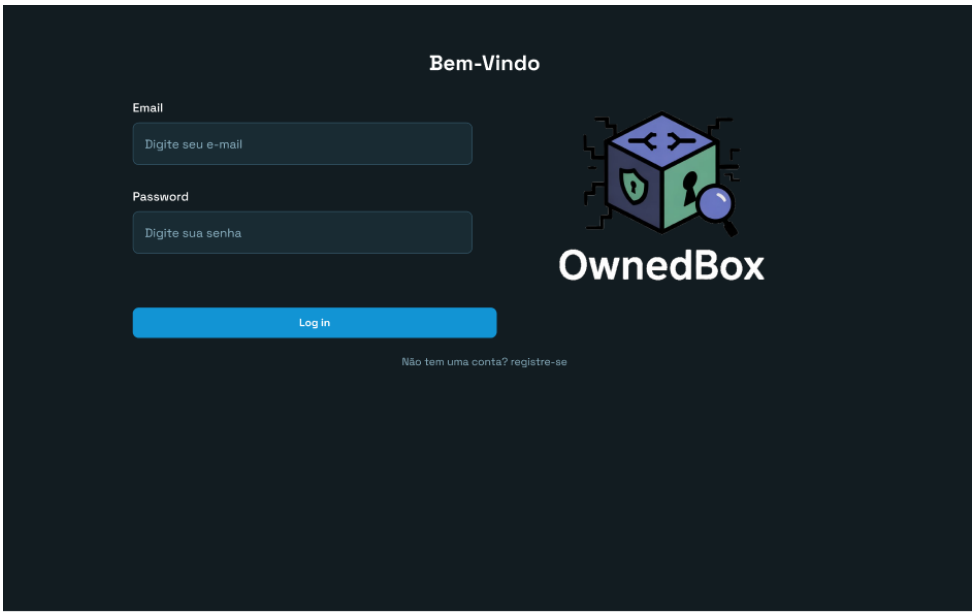


Figura 4 – Protótipo da tela de *login* desenvolvido no Figma.

5.2.3 Tela de módulos

A tela de módulos, apresentada na Figura 5, exibe os desafios de aprendizado disponíveis para o usuário, relacionados a diferentes tipos de vulnerabilidades de segurança. Cada módulo inclui uma breve descrição da vulnerabilidade e uma indicação visual de progresso, permitindo que o usuário acompanhe seu desempenho ao longo do curso. O *design* segue o padrão de cores, tipografia e navegação já estabelecido nas telas de cadastro e *login*, mantendo a consistência visual do sistema.

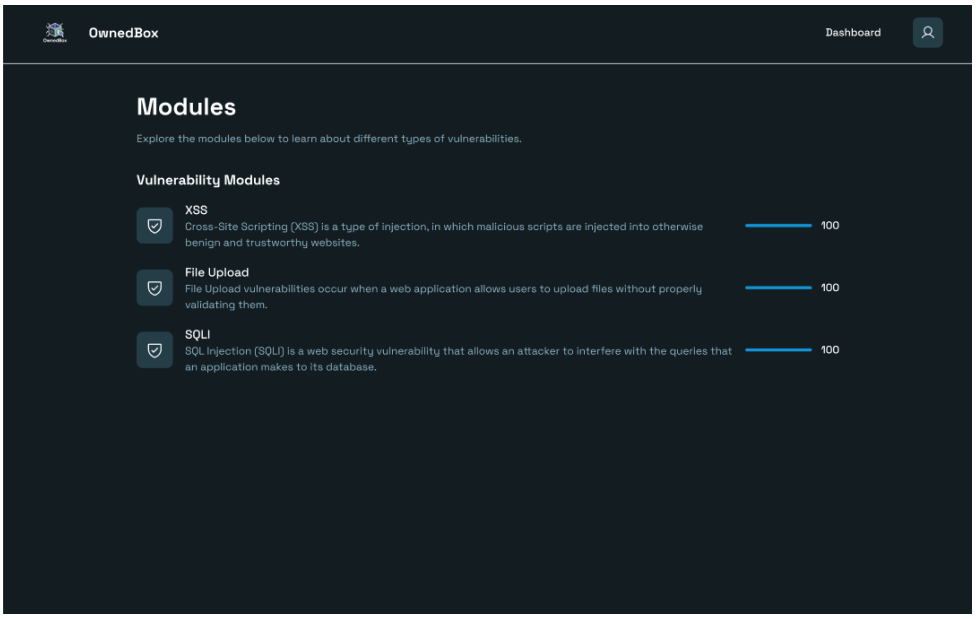


Figura 5 – Protótipo da tela de módulos desenvolvido no Figma.

5.2.4 Tela de perfil

Na Figura 6 é apresentada a tela de perfil, que permite ao usuário visualizar e editar suas informações pessoais, como nome, e-mail e senha. Além disso, a interface inclui uma barra de progresso que indica a porcentagem total de módulos concluídos pelo usuário, fornecendo uma visão rápida do seu desempenho no curso. O *design* mantém a consistência visual com as telas anteriores, utilizando o mesmo esquema de cores, tipografia e elementos de navegação.

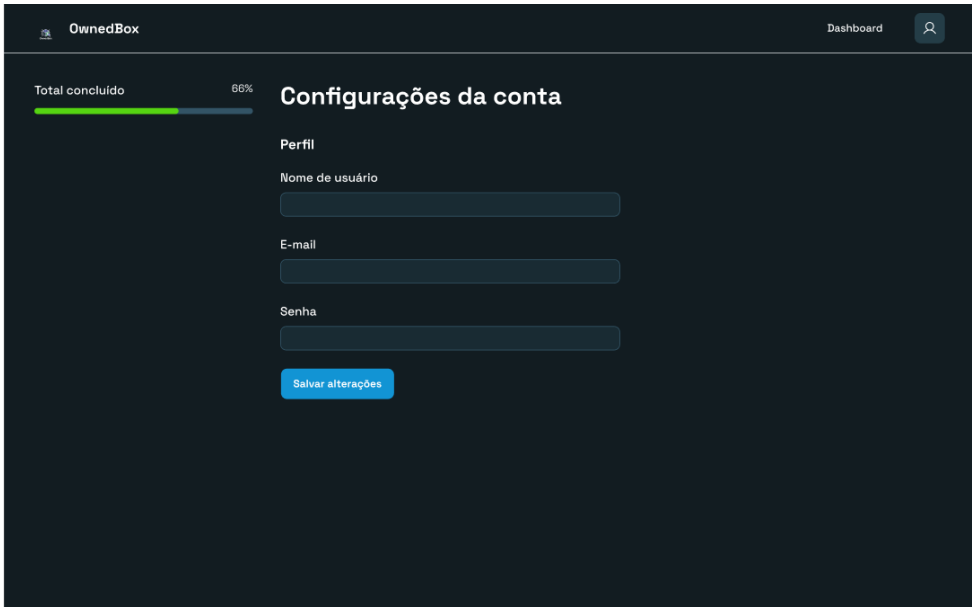


Figura 6 – Protótipo da tela de perfil desenvolvido no Figma.

5.2.5 Tela de sessão de vulnerabilidade

Conforme a Figura 7, a tela de sessão de vulnerabilidade apresenta ao usuário informações detalhadas sobre a vulnerabilidade abordada, incluindo uma descrição teórica e instruções práticas para o laboratório. O *layout* inclui campos interativos, como o botão para gerar instâncias de teste e o campo para submissão de *tokens*, permitindo que o aluno pratique os conceitos aprendidos de forma segura. O *design* mantém consistência visual com as outras telas do sistema, utilizando o mesmo esquema de cores, tipografia e elementos de navegação.

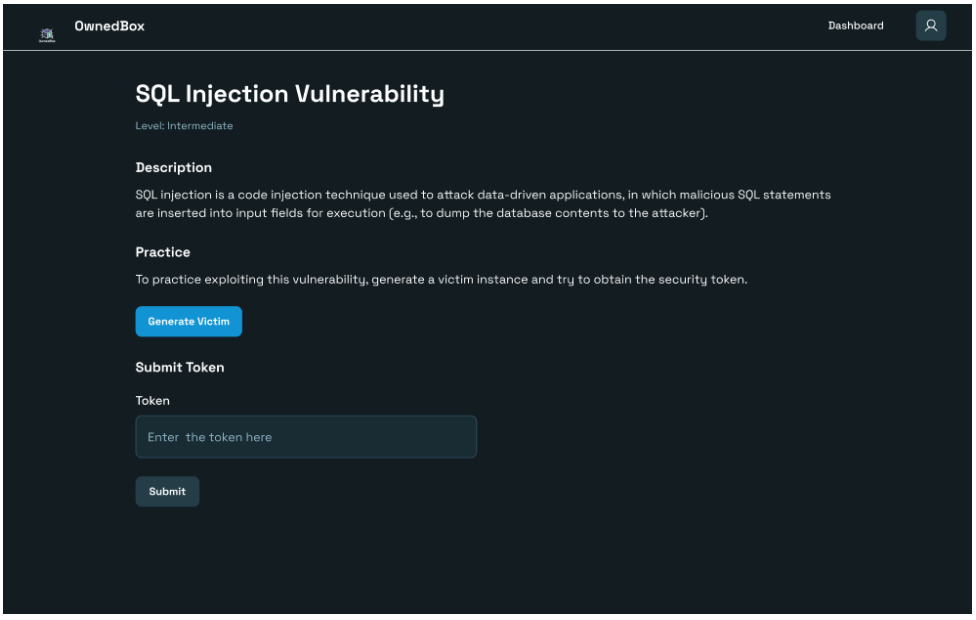


Figura 7 – Protótipo da tela de sessão de vulnerabilidade desenvolvido no Figma.

5.2.6 Tela de laboratório de SQL Injection

A tela de laboratório, ilustrada na Figura 8, compõe a parte prática do sistema e tem como objetivo permitir que o usuário aplique, em um ambiente controlado, os conceitos teóricos apresentados na tela de sessão de vulnerabilidade. O laboratório simula um sistema de autenticação intencionalmente vulnerável, no qual o aluno é desafiado a explorar uma falha de SQL Injection para burlar o sistema de *login* e, assim, obter a *flag* correspondente. Para apoiar o aprendizado, a interface disponibiliza uma seção de dicas, que inclui as credenciais válidas de referência e um exemplo de *payload* de injeção, orientando o aluno de forma progressiva durante a resolução do desafio. O *design* mantém consistência visual com as demais telas do sistema, utilizando o mesmo esquema de cores, tipografia e elementos de navegação.

Figura 8 – Protótipo da tela de laboratório de SQL Injection desenvolvido no Figma.

5.2.7 Tela de laboratório de XSS

O laboratório de XSS, apresentado na Figura 9, mantém a consistência visual com as demais telas do sistema e explora uma vulnerabilidade de *Cross-Site Scripting* refletido. A interface simula uma aba de comentários, na qual o conteúdo digitado pelo usuário é renderizado diretamente na página, sem o devido tratamento. Dessa forma, o aluno é desafiado a injetar código JavaScript por meio do campo de mensagem e, ao explorar a falha, obter a *flag* armazenada nos *cookies* da aplicação. Assim como na tela anterior, o laboratório disponibiliza uma seção de dicas com um exemplo de uso legítimo e um *payload* de injeção, conduzindo o aluno de forma progressiva durante a resolução do desafio.

Figura 9 – Protótipo da tela de laboratório de XSS desenvolvido no Figma.

5.2.8 Tela de laboratório de *File Upload*

O laboratório de *File Upload*, exibido na Figura 10, mantém a consistência visual com as demais telas do sistema e explora uma falha de envio inseguro de arquivos. A interface disponibiliza um campo de *upload* que aceita arquivos em linguagem *PHP: Hypertext Preprocessor* (PHP), sem qualquer validação de tipo ou conteúdo. Dessa forma, o aluno é desafiado a enviar um arquivo malicioso capaz de executar comandos no servidor vulnerável, caracterizando uma execução remota de código (*RCE*), e, a partir disso, navegar pelo sistema de arquivos até localizar a *flag* armazenada em um arquivo de texto. Assim como nos laboratórios anteriores, a tela apresenta uma seção de dicas com um exemplo de *payload*, conduzindo o aluno de forma progressiva durante a resolução do desafio.

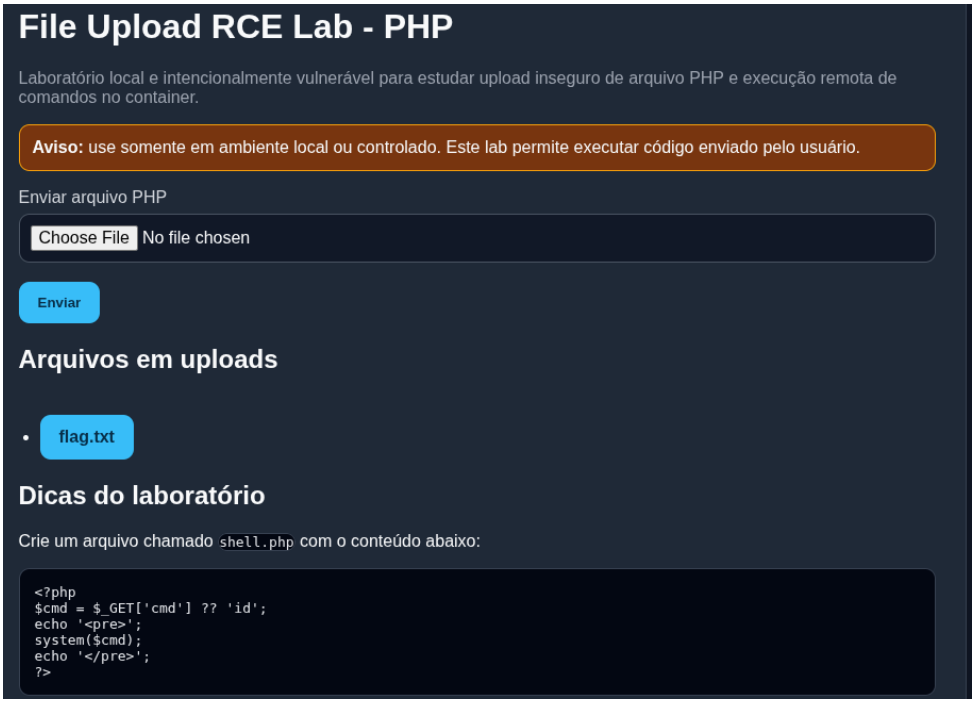


Figura 10 – Protótipo da tela de laboratório de *File Upload* desenvolvido no Figma.

5.3 Modelagem do banco de dados

Para o desenvolvimento do sistema, foi elaborada a modelagem do banco de dados responsável por armazenar as informações de usuários. A estrutura inicialmente proposta, apresentada na Figura 11, era composta por três entidades principais: **users**, **module** e **owned_modules**.

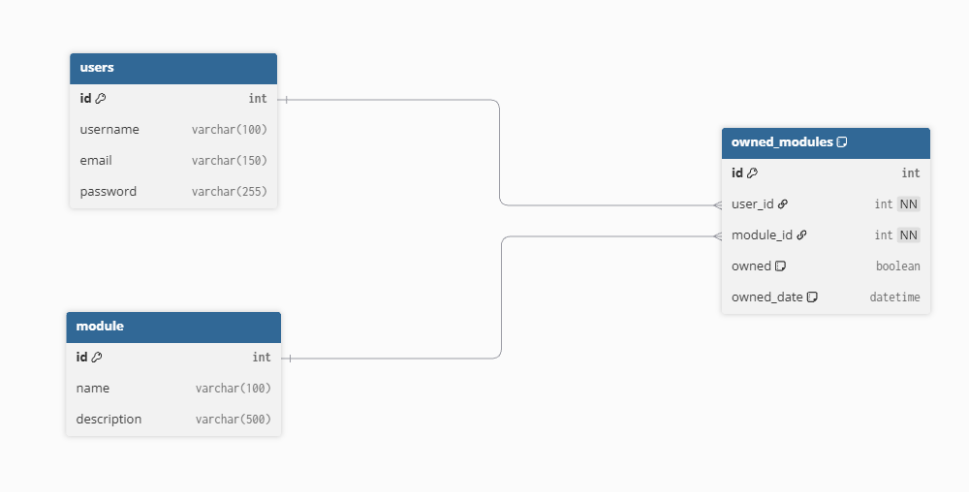


Figura 11 – Modelagem inicial do banco de dados: entidades e relacionamentos

A tabela **users** foi aproveitada diretamente do Breeze, sendo responsável por gerenciar o cadastro e a autenticação dos usuários. Essa decisão visava reduzir a duplicidade de informações e aproveitar a infraestrutura nativa de autenticação segura já existente no *framework*.

A tabela **module** armazenaria os módulos disponíveis no sistema, contendo um identificador, o nome e uma descrição para cada módulo. Essa entidade tinha como objetivo centralizar as informações sobre os conteúdos ou atividades que o usuário poderá acessar.

A tabela **owned_modules** tinha o papel intermediário, representando o relacionamento entre as entidades **users** e **module**. Cada registro nessa tabela indicaria que um usuário possui determinado módulo, armazenando a data de aquisição e um valor booleano que confirma a posse do módulo. Essa estrutura permite identificar quais módulos cada usuário possui e serviria como base para o controle de progresso e funcionalidades futuras.

Ao longo do desenvolvimento, constatou-se que a modelagem inicial apresentava algumas inconsistências que precisaram ser corrigidas. Em reuniões com o orientador, optou-se por reduzir o escopo do projeto e trabalhar de forma mais consistente com as tabelas e seus dados. Em decorrência dessa decisão, surgiram modificações na estrutura do banco, conforme apresentado na Figura 12.

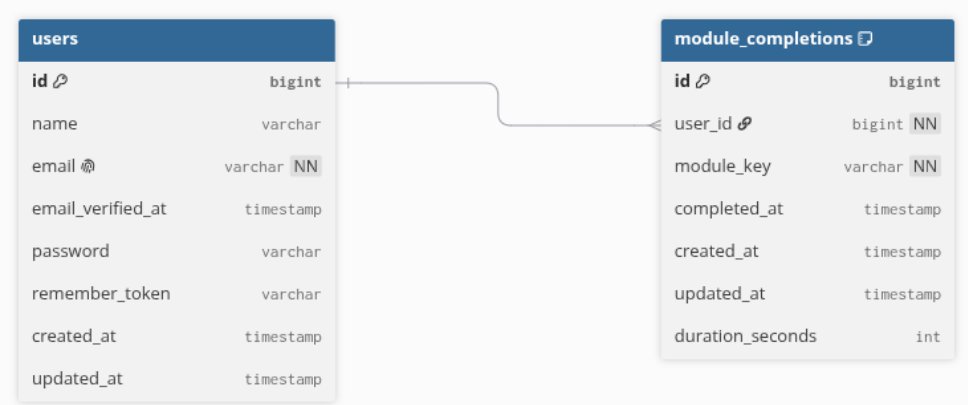


Figura 12 – Modelagem final do banco de dados: entidades e relacionamentos

A principal modificação consistiu na remoção da tabela **module** e da tabela **owned_modules**, que foram substituídas por uma única tabela denominada **module_completions**. Com essa mudança, deixou-se de manter uma entidade dedicada ao cadastro de módulos: cada módulo passou a ser identificado por uma chave textual (`module_key`), como `xss`, `sql` e `fileupload`, armazenada diretamente na nova tabela. Essa simplificação eliminou a necessidade de um relacionamento muitos-para-muitos e reduziu a complexidade do modelo, alinhando-o ao escopo revisado do projeto.

Além disso, o conceito de *posse* de módulo, antes representado por um valor booleano na tabela **owned_modules**, foi substituído pelo conceito de *conclusão* de módulo. Na nova estrutura, a própria existência de um registro em **module_completions** indica que o usuário concluiu determinado módulo, tornando dispensável o campo booleano anterior.

A tabela **module_completions** passou a registrar, além do identificador do usuário (`user_id`) e da chave do módulo (`module_key`), a data de conclusão (`completed_at`), o tempo gasto na conclusão em segundos (`duration_seconds`) e os campos de controle `created_at` e `updated_at`. Foi também adicionada uma restrição de unicidade sobre o

par (`user_id`, `module_key`), garantindo que não existam registros duplicados de conclusão para o mesmo usuário e módulo.

Por fim, o relacionamento entre as entidades foi simplificado para uma única relação um-para-muitos (1:N) entre **users** e **module_completions**, na qual um usuário pode possuir diversos registros de conclusão. Essa nova modelagem manteve o equilíbrio entre simplicidade e escalabilidade, preservando a possibilidade de expansões futuras, como o registro de notas ou de novas métricas de desempenho, com impacto mínimo sobre a estrutura existente.

5.3.1 Arquitetura do sistema

A arquitetura do OwnedBox foi concebida segundo um modelo em camadas, no qual a interface de usuário, a lógica de negócio e a camada de persistência permanecem desacopladas, e no qual os ambientes propositalmente vulneráveis são executados de forma isolada por meio de contêineres. A Figura 13 apresenta a visão macro do sistema, evidenciando os principais componentes e os fluxos de comunicação estabelecidos entre eles.

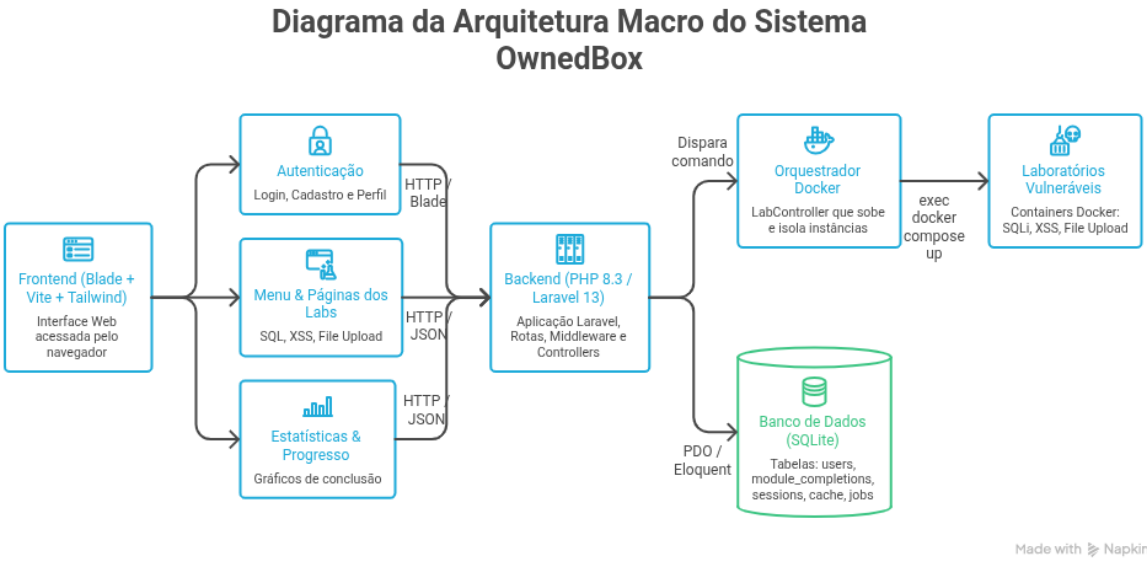


Figura 13 – Diagrama da arquitetura macro do sistema OwnedBox.
Fonte: Elaborado por Napkin.

A camada de *frontend* é responsável pela interface acessada diretamente pelo navegador do usuário. Ela é construída com *templates* Blade, estilizados pelo *framework* CSS Bootstrap, incorporado às páginas por meio de CDN. Essa camada concentra três conjuntos de funcionalidades: o módulo de **autenticação**, que provê o *login*, o cadastro e a edição de perfil; o **menu e as páginas dos laboratórios**, que dão acesso aos desafios de SQL Injection, *Cross-Site Scripting* (XSS) e *File Upload*; e a área de **estatísticas e progresso**, na qual o usuário acompanha graficamente os módulos concluídos. A comunicação dessa camada com o *backend* ocorre por meio do protocolo HTTP, seja para a renderização das páginas Blade,

seja para as requisições no formato JSON utilizadas na geração de vítimas e na validação de *tokens*.

O *backend* constitui o núcleo da aplicação e foi desenvolvido em PHP 8.3 sobre o *framework* Laravel 13. Nessa camada concentram-se as regras de negócio e os mecanismos de segurança, organizados em rotas, *middleware* de autenticação e controladores, com destaque para o `AuthController`, o `UserController` e o `LabController`. Cabe ao *backend* mediar todas as interações entre a interface, o banco de dados e os ambientes de laboratório, garantindo que apenas usuários autenticados tenham acesso aos módulos protegidos.

Um aspecto central da arquitetura é o **orquestrador de contêineres**, implementado no `LabController`. Quando o usuário solicita a geração de uma vítima, o controlador gera um *token* aleatório, denominado *flag* do desafio, injeta esse valor como variável de ambiente e dispara o comando `docker compose up`, provisionando sob demanda uma instância isolada do laboratório correspondente em uma porta dinâmica, escolhida no intervalo de 5100 a 5999. Dessa forma, cada laboratório é executado em um **contêiner Docker** independente, baseado em PHP e Apache, contemplando os três cenários vulneráveis disponíveis: SQL Injection, XSS e *File Upload*. Após o provisionamento, o navegador do usuário passa a acessar diretamente o contêiner pela porta gerada, explorando a vulnerabilidade até capturar a *flag*, que é então submetida ao *backend* para validação. Caso o *token* informado coincida com o valor armazenado em sessão, o sistema registra a conclusão do módulo.

6 DESENVOLVIMENTO DO SISTEMA

Neste capítulo são detalhadas as etapas de desenvolvimento do **OwnedBox**, conduzido de forma iterativa e incremental. As atividades foram organizadas em *sprints* e gerenciadas por meio do método **Kanban**, utilizando quadros no *GitHub Projects*, nos quais as tarefas eram distribuídas em colunas que representavam seu estado, como **Backlog**, **To Do**, **In Progress**, **Review** e **Done**. Esse modelo de acompanhamento permitiu visualizar o progresso e priorizar as demandas conforme o avanço do desenvolvimento.

Cada sprint é apresentada a seguir descrevendo as funcionalidades implementadas, as decisões técnicas tomadas, as principais dificuldades enfrentadas e as adaptações de escopo necessárias ao longo do processo. As funcionalidades são associadas às Histórias de Usuário (HU) definidas na etapa de análise (Capítulo 4), referenciadas em notas de rodapé no momento em que são tratadas. O versionamento do código foi conduzido com Git, mantendo o repositório hospedado no GitHub, de modo que cada incremento descrito nesta seção corresponde a entregas integradas por meio de *Pull Requests* revisados antes da incorporação à *branch* principal.

6.1 Sprint 0

A *Sprint 0*, com duração estimada de cinco dias, foi dedicada à preparação e padronização do ambiente de desenvolvimento. Embora não tenha gerado uma entrega funcional diretamente perceptível ao usuário final, essa etapa foi essencial para assegurar a estabilidade e a produtividade das sprints seguintes.

Nessa fase, foi instalado e configurado o *framework* **Laravel**, juntamente com suas dependências e bibliotecas essenciais, bem como o ambiente de execução **PHP** e o gerenciador de pacotes *Composer*. Foi também definido o **SQLite** como sistema de gerenciamento de banco de dados, escolha alinhada à natureza local e portável da aplicação. Para garantir a consistência entre diferentes ambientes de execução, configurou-se o **Docker**, recurso que seria posteriormente fundamental para a containerização dos laboratórios de vulnerabilidade.

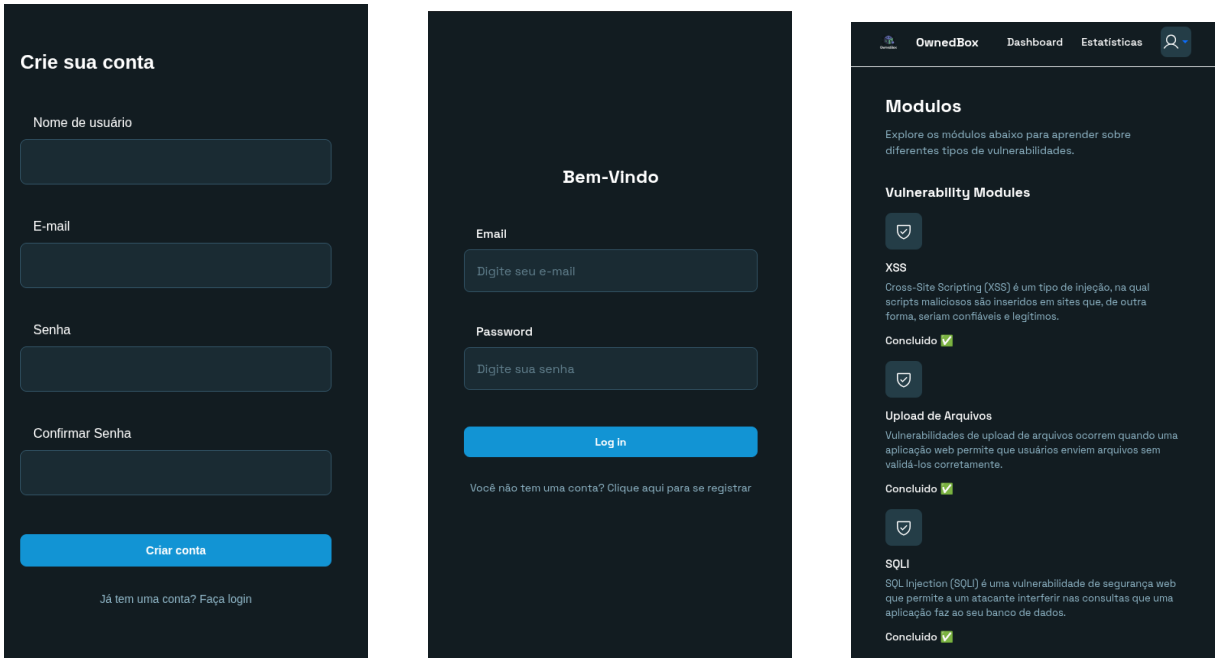
Foi criado o repositório do código-fonte no **GitHub** e realizada a configuração inicial do **Git**, definindo o fluxo de trabalho baseado em *branches* e *Pull Requests*. Para a gestão das tarefas, organizou-se um quadro seguindo os princípios do **Kanban**, com colunas representando os estados das atividades ao longo de cada sprint. Por fim, foram realizadas atualizações das bibliotecas e dependências do sistema, de modo a deixar o projeto devidamente ajustado para o início do desenvolvimento. Durante esse processo, ocorreram alguns imprevistos relacionados a incompatibilidades entre versões de bibliotecas, que exigiram reinstalações e atualizações sucessivas até que o ambiente se estabilizasse e ficasse adequado ao desenvolvimento.

Ao final da *Sprint 0*, o ambiente encontrava-se preparado para suportar o desenvolvimento ágil e incremental do projeto, com o repositório versionado, o quadro de tarefas estruturado e as ferramentas essenciais devidamente integradas.

6.2 Sprint 1

A *Sprint 1*, com duração de duas semanas, foi responsável pela primeira entrega funcional ao usuário e contemplou as histórias **HU01**[1], **HU02**[2] e **HU03**[3], referentes ao fluxo de cadastro, autenticação e perfil dos usuários.

Para melhor organizar o trabalho, a sprint foi dividida em duas etapas principais. Na **primeira semana**, o foco esteve na construção das telas de fluxo de forma unicamente visual, sem lógica de programação. Foram desenvolvidas, com o motor de *templates* **Blade**, as interfaces de **login**, **cadastro** e a tela de **módulos de aprendizado**, estabelecendo o padrão visual definido na etapa de projeto, com a paleta de cores escura e a tipografia previamente especificadas. As telas de cadastro, de *login* e de módulos de aprendizado implementadas são apresentadas na Figura 14, respectivamente em (a), (b) e (c). Essa abordagem permitiu validar a navegação e a experiência do usuário antes da implementação da lógica de negócio.



(a) Cadastro de usuário **(b) Login** **(c) Módulos de aprendizado**

Figura 14 – Telas de cadastro (a), *login* (b) e módulos de aprendizado (c) do OwnedBox.

[1] **HU01**: Como um usuário, quero realizar meu cadastro informando dados básicos (nome, e-mail e senha), para que eu possa acessar a aplicação web e participar dos módulos de aprendizado.

[2] **HU02**: Como usuário cadastrado, quero efetuar *login* com minhas credenciais, para acessar meu perfil e os módulos de aprendizado.

[3] **HU03**: Como usuário autenticado, quero acessar meu perfil com informações de desempenho, para acompanhar meu progresso na aplicação web.

Na **segunda semana**, foi implementada a lógica de negócio responsável pela autenticação e pelo gerenciamento dos usuários. O processo de **login** foi construído sobre o mecanismo nativo de autenticação do Laravel, validando as credenciais e regenerando a sessão a cada acesso bem-sucedido, como ilustrado na Listagem 1. O `AuthController` concentra as ações de exibição do formulário, autenticação e encerramento de sessão, retornando mensagens de erro adequadas em caso de credenciais inválidas.

```php
<?php
public function login(Request $request)
{
    $credentials = $request->validate([
        'email'    => ['required', 'email'],
        'password' => ['required'],
    ]);

    if (Auth::attempt($credentials)) {
        $request->session()->regenerate();
        return redirect()->intended('/menu');
    }

    return back()->withErrors([
        'email' => 'As credenciais fornecidas estao incorretas.',
    ])->withInput();
}
?>
```

Listagem 1 – Autenticação de usuário no `AuthController`.

O gerenciamento dos usuários foi implementado no `UserController`, responsável pelo cadastro, edição e remoção de contas. No cadastro, os dados informados são validados, garantindo, por exemplo, a unicidade do e-mail e a confirmação da senha, e a senha é armazenada de forma segura por meio de *hash*, aproveitando os mecanismos de segurança já consolidados no *framework*. A tela de **perfil** foi integrada à lógica de negócio, permitindo ao usuário visualizar seus dados e editá-los, além de acompanhar seu progresso, atendendo à **HU03**, conforme apresentado na Figura 15.

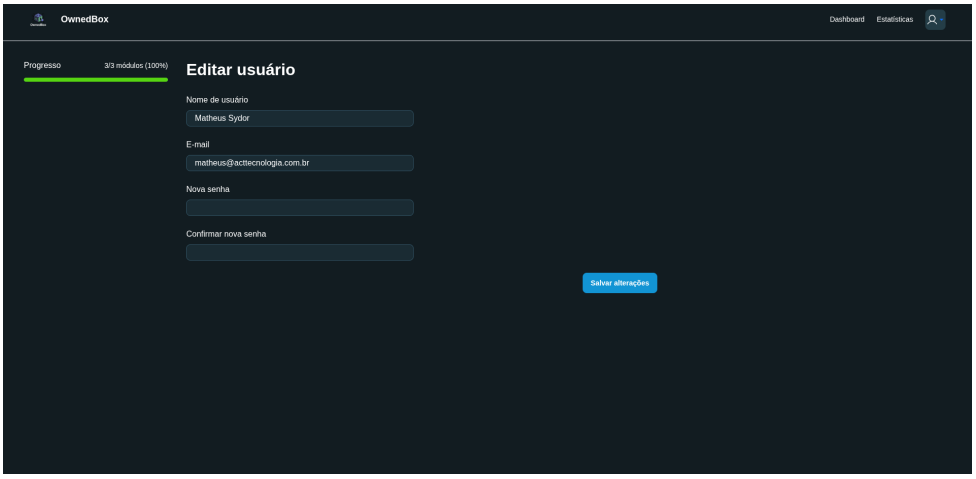


Figura 15 – Tela de perfil do usuário do OwnedBox.

Ao término da *Sprint 1*, a aplicação dispunha de um fluxo completo de cadastro, autenticação e gerenciamento de perfil, estabelecendo a base necessária para o desenvolvimento dos módulos práticos nas sprints seguintes.

6.3 Sprint 2

A *Sprint 2*, com duração de três semanas, foi dedicada à criação dos **laboratórios vulneráveis**, que constituem o mecanismo principal do sistema, atendendo às histórias **HU04**[4] e **HU05**[5]. Nesta etapa, foram desenvolvidos os primeiros laboratórios, sendo eles *SQL Injection*, *Cross-Site Scripting* (XSS) e *Upload de Arquivos Maliciosos*, cada um implementado como uma pequena aplicação *web* em PHP, propositalmente vulnerável, empacotada em uma imagem Docker e descrita por um arquivo *docker-compose*.

Neste momento do desenvolvimento, optou-se por trabalhar com uma **flag fixa**, definida apenas para fins de desenvolvimento, de modo a permitir a validação do funcionamento dos laboratórios sem a complexidade adicional da geração dinâmica de valores. Essa decisão foi importante para estabelecer um **padrão de desenvolvimento de laboratório**, definindo como cada ambiente seria estruturado, parametrizado e integrado à plataforma antes de avançar para a lógica completa de instanciação.

O principal resultado desta sprint foi a definição da estrutura do **controlador** responsável por orquestrar os laboratórios, o `LabController`. Optou-se por organizar os laboratórios em uma **lista centralizada de configurações**, abordagem que tornou a organização mais clara e estabeleceu um padrão consistente de desenvolvimento. Cada laboratório é descrito por uma entrada nessa estrutura, contendo o caminho da pasta que possui o arquivo *docker-compose*, o prefixo utilizado para isolar instâncias simultâneas, o nome da variável de ambiente que recebe

[4] **HU04**: Como estudante de cibersegurança, quero acessar laboratórios com vulnerabilidades simuladas, para praticar técnicas ofensivas em ambiente controlado.

[5] **HU05**: Como usuário da aplicação web, quero visualizar contramedidas para cada vulnerabilidade, para aprender como corrigir falhas e aplicar boas práticas defensivas.

a *flag*, o formato da *flag* e a chave de sessão usada para armazená-la. A Listagem 2 apresenta essa configuração centralizada.

```php
<?php
private const LABS = [
    'sql' => [
        'path'           => 'sqli-login-lab-php',
'project_prefix' => 'sqli_victim_',
'flag_env'       => 'FLAG_ID',
'flag_format'    => '{token}',
'session_key'    => 'sql_lab_flag',
    ],
'xss' => [
    'path'           => 'xss-cookie-lab-php',
'project_prefix' => 'xss_victim_',
'flag_env'       => 'FLAG_COOKIE',
'flag_format'    => '{token}',
'session_key'    => 'xss_lab_flag',
],
'fileupload' => [
    'path'           => 'file-upload-lab-php',
'project_prefix' => 'fileupload_victim_',
'flag_env'       => 'FLAG_VALUE',
'flag_format'    => '{token}',
'session_key'    => 'fileupload_lab_flag',
],
];
?>
```

Listagem 2 – Configuração centralizada dos laboratórios no `LabController`.

A principal vantagem dessa abordagem é a **extensibilidade**: para adicionar um novo laboratório, basta acrescentar uma nova entrada à lista `LABS` e disponibilizar a respectiva pasta com o *docker-compose*, sem necessidade de alterar a lógica de instanciação ou de validação. O controlador ainda expõe um método auxiliar, `moduleKeys()`, que retorna as chaves dos laboratórios disponíveis, utilizado pelas *views* de menu e perfil para determinar o total de módulos existentes, como mostra a Listagem 3.

```php
<?php
public static function moduleKeys(): array
{
    return array_keys(self::LABS);
}

private function labConfig(string $lab): ?array
{
    return self::LABS[$lab] ?? null;
}
?>
```

Listagem 3 – Recuperação centralizada das chaves dos módulos.

Cada laboratório foi construído para expor a *flag* de uma forma coerente com a falha estudada. No laboratório de *SQL Injection*, por exemplo, a autenticação é realizada concatenando diretamente a entrada do usuário na consulta SQL, de modo intencionalmente vulnerável, conforme ilustrado na Listagem 4. Ao aplicar um *payload* de *bypass* lógico de *login*, o usuário autentica-se como administrador e recupera o identificador da conta, que corresponde à *flag* da instância.

```php
<?php
// Intencionalmente vulneravel para laboratorio.
// NAO use concatenacao de SQL com entrada do usuario em aplicacao
↪  real.
$sql = "SELECT id, username, role FROM users
WHERE username = '$username' AND password = '$password' LIMIT 1";
?>
```

Listagem 4 – Consulta intencionalmente vulnerável do laboratório de *SQL Injection*.

A abordagem didática do sistema separa a dimensão defensiva da ofensiva. A **parte defensiva** é descrita no próprio **módulo** da plataforma, que apresenta, além da explicação teórica da vulnerabilidade, as respectivas **contramedidas** orientam como prevenir a falha e aplicar boas práticas de proteção, atendendo à **HU05**, conforme apresentado na Figura 16. Já a **parte ofensiva**, em que o *payload* malicioso é efetivamente aplicado para explorar a falha, ocorre na **instância vulnerável (vítima)** executada pelo laboratório, ilustrada na Figura 17. Dessa forma, o sistema reúne tanto a forma de explorar a falha quanto a forma correta de preveni-la.

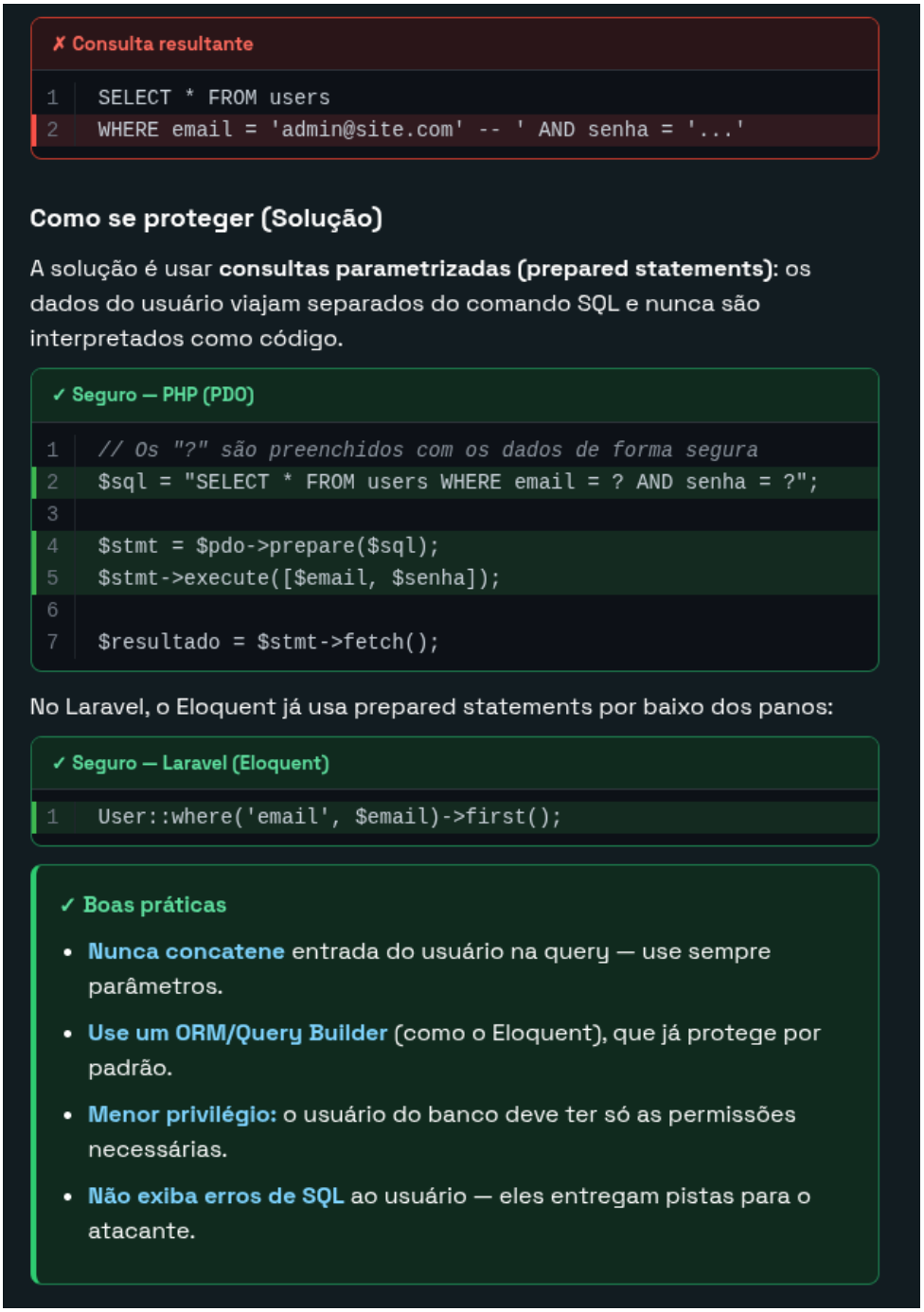


Figura 16 – Contramedidas defensivas apresentadas no módulo do OwnedBox.

Figura 17 – Instância vulnerável do laboratório de *SQL Injection*, alvo da exploração ofensiva.

Ao término da *Sprint 2*, os laboratórios encontravam-se funcionalmente estruturados e integrados ao padrão definido, ainda que operando com a *flag* fixa de desenvolvimento, estabelecendo a base prática para a etapa seguinte.

6.4 Sprint 3

A *Sprint 3* foi dedicada à integração completa dos laboratórios com as funcionalidades dos módulos, contemplando a história **HU06**[6] e consolidando o acompanhamento de progresso previsto na **HU03**. Inicialmente planejada para duas semanas, esta sprint precisou ser estendida por mais duas semanas, totalizando **quatro semanas** de desenvolvimento. A ampliação de prazo foi necessária porque algumas funcionalidades não operavam corretamente sobre a estrutura de laboratório definida na etapa anterior, exigindo adaptações no código para padronizar o *design* e harmonizar o comportamento dos diferentes laboratórios.

Após essas adaptações, foi implementada a lógica de integração entre os laboratórios e os módulos, composta por dois mecanismos centrais: a **geração de uma flag aleatória** para cada instância e a **pontuação do usuário** ao concluir o módulo. Substituiu-se, assim, a *flag* fixa utilizada durante o desenvolvimento por um valor exclusivo, gerado dinamicamente a cada instanciação.

O fluxo de prática inicia quando o usuário, na tela do módulo, aciona a opção de **gerar a vítima**, conforme ilustrado na Figura 18. Essa ação dispara uma requisição assíncrona ao *back-end*, tratada pelo método `generateVictim`, que gera um *token* aleatório a partir de bytes

[6] **HU06**: Como usuário que conclui um desafio, quero receber um *token* de validação, para comprovar que atingi o objetivo proposto.

criptograficamente seguros, sorteia uma porta dentro de uma faixa reservada, define um nome de projeto único para o contêiner e inicia a instância da vítima com o `docker compose up`, injetando a *flag* por variável de ambiente. A *flag* esperada é armazenada na sessão do usuário, e a *Uniform Resource Locator* (Localizador Uniforme de Recursos) (URL) da instância recém-criada é retornada para acesso. A Listagem 5 apresenta o trecho central desse método.

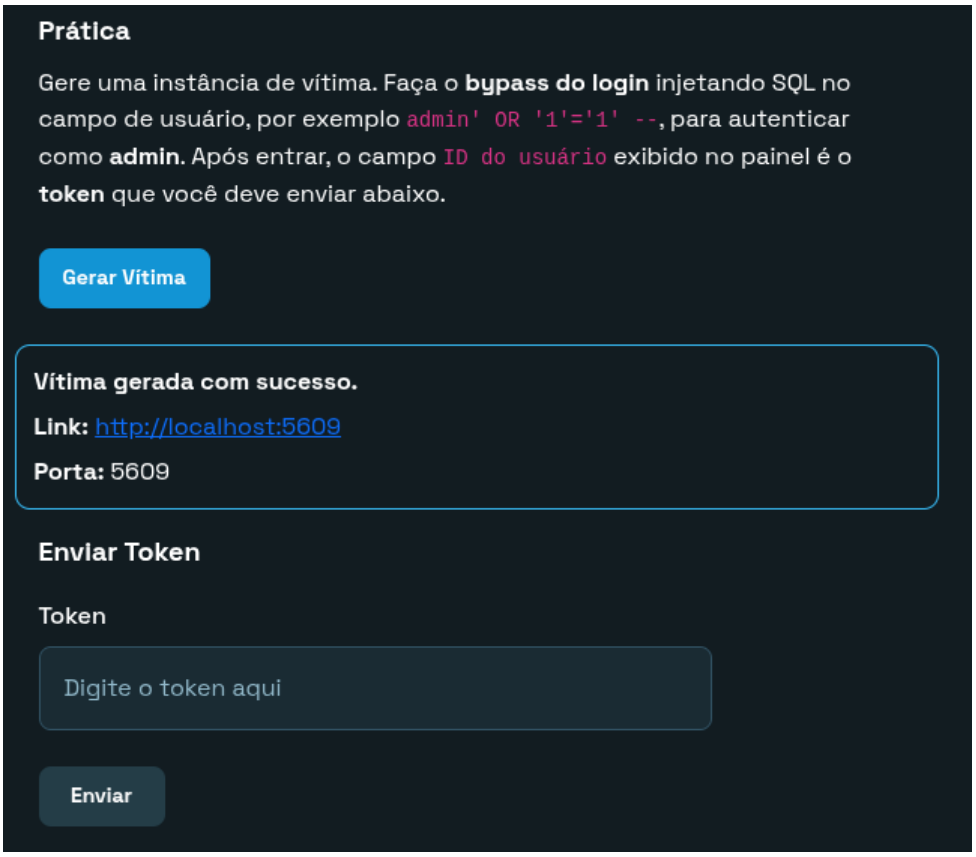


Figura 18 – Tela de prática do módulo, com a geração da instância vítima e o envio do *token*.

```php
<?php
$token       = bin2hex(random_bytes(8));
$flag        = str_replace('{token}', $token,
    $config['flag_format']);
$port        = random_int(5100, 5999);
$projectName = $config['project_prefix'] .
    Str::lower(Str::random(10));

$command = 'cd ' . escapeshellarg($labPath)
. ' && ' . $config['flag_env'] . '=' . escapeshellarg($flag)
. ' LAB_PORT=' . escapeshellarg((string) $port)
. ' COMPOSE_PROJECT_NAME=' . escapeshellarg($projectName)
. ' docker compose up --build -d --force-recreate 2>&1';

exec($command, $output, $return);

session([$config['session_key'] => $flag]);
?>
```

Listagem 5 – Geração da instância vítima com *flag* aleatória.

Como a *flag* é gerada dinamicamente e injetada apenas naquela instância, cada tentativa de exploração ocorre em um ambiente efêmero, único e isolado, evitando o compartilhamento de estado entre usuários e reforçando a segurança e a reprodutibilidade do exercício.

De posse da *flag*, obtida por meio da exploração da vulnerabilidade, o usuário a submete no campo de **envio de token**. O método `validateToken` compara o valor enviado com a *flag* armazenada na sessão, utilizando uma comparação resistente a ataques de temporização (`hash_equals`). Em caso de sucesso, o sistema registra a conclusão do módulo, vinculando o usuário ao laboratório concluído com a respectiva data e hora, conforme a Listagem 6.

```php
<?php
if (hash_equals($expected, $submitted)) {
    $user = $request->user();
    if ($user) {
        ModuleCompletion::updateOrCreate(
            ['user_id' => $user->id, 'module_key' => $lab],
            ['completed_at' => now()],
        );
    }

    return Response::json([
        'success' => true,
        'message' => 'Token correto! Desafio concluido.',
    ]);
}
?>
```

Listagem 6 – Validação do *token* e registro de progresso do usuário.

O registro de conclusão é persistido na tabela `module_completions`, por meio do modelo `ModuleCompletion`, que mantém o vínculo entre o usuário e cada módulo concluído. O uso do método `updateOrCreate`, em conjunto com a restrição de unicidade sobre o par (`user_id`, `module_key`) definida na modelagem, garante que a conclusão de um mesmo módulo seja registrada uma única vez. Esse registro é refletido imediatamente na tela de módulos, que passa a exibir o desafio como concluído, e no painel de progresso do perfil, que recalcula o percentual de módulos finalizados, materializando a **pontuação** do usuário. Todas as requisições são protegidas por *token* CSRF, e a validação só é aceita caso exista uma instância previamente gerada para o usuário.

Ao término da *Sprint 3*, a plataforma passou a acompanhar automaticamente o desempenho dos usuários por meio de *tokens* validados, concretizando o ciclo de aprendizado proposto: o usuário pratica a exploração ofensiva em um ambiente isolado, comprova a conquista por meio do *token* e tem seu progresso registrado de forma consistente e segura.

6.5 Sprint 4

A *Sprint 4*, com duração de duas semanas, foi dedicada ao **refinamento do sistema** e ao desenvolvimento dos elementos de apoio à utilização da plataforma. Diferentemente das sprints anteriores, voltadas à construção das funcionalidades centrais, esta etapa concentrou-se em ajustes avaliados em conjunto com o **orientador**, bem como na entrega das histórias **HU07**[7] e **HU09**[8].

A primeira frente de trabalho consistiu em **ajustes de consistência** identificados ao longo das revisões com o orientador. Foram revisados os textos descritivos dos módulos, padronizando a redação das instruções e das contramedidas apresentadas em cada laboratório, de modo a tornar a linguagem mais clara e uniforme. Também foi padronizado o **formato das flags** entre os diferentes laboratórios, consolidando a estrutura baseada em um *token* aleatório definida na *Sprint 3* como padrão único para todas as instâncias. Adicionalmente, o código-fonte foi uniformizado, com a tradução de identificadores e a inserção de comentários de documentação, melhorando sua legibilidade e manutenibilidade.

A segunda frente foi a criação de uma **página de estatísticas**, atendendo à **HU07**. Essa tela oferece ao usuário uma visão consolidada de seu desempenho, apresentando o percentual de módulos concluídos, a quantidade de laboratórios finalizados na semana corrente e um gráfico com o **tempo de conclusão** de cada módulo, como mostra a Figura 19. Para viabilizar essa funcionalidade, a tabela `module_completions` foi estendida com o campo `duration_-`

[7] **HU07**: Como usuário da plataforma, quero acessar relatórios de desempenho, para acompanhar minha evolução ao longo do tempo.

[8] **HU09**: Como usuário, quero ter acesso a uma página com instruções de instalação e requisitos do sistema, para configurar a aplicação web corretamente em meu ambiente local.

`seconds`, por meio de uma nova *migration*, passando a registrar o tempo decorrido em cada desafio. A Listagem 7 apresenta a agregação dos dados que alimentam a página.

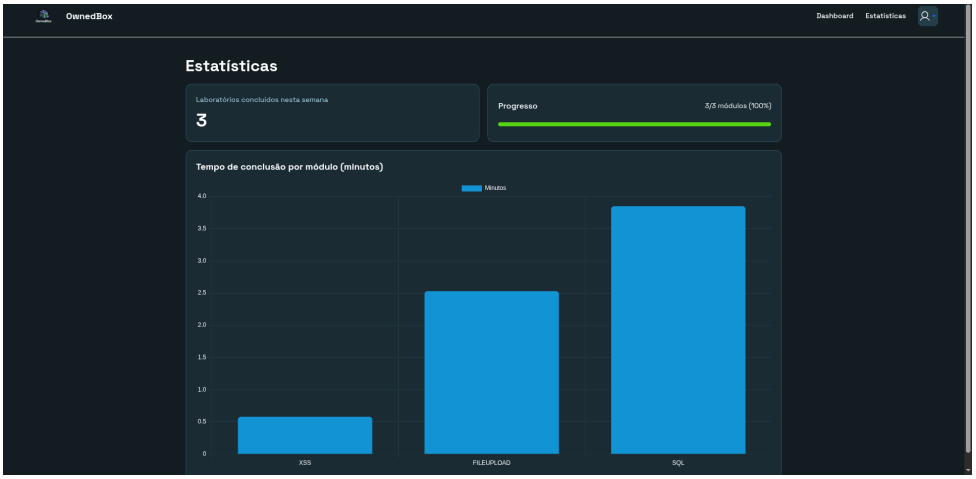


Figura 19 – Página de estatísticas de desempenho do usuário no OwnedBox.

```php
<?php
$completions =
    ↪  $user->moduleCompletions()->orderBy('completed_at')->get();

// Tempo de conclusao (em minutos) por modulo, para o grafico.
$completionTimes = $completions
->whereNotNull('duration_seconds')
->map(fn ($c) => [
    'module'  => strtoupper($c->module_key),
      'minutes' => round($c->duration_seconds / 60, 2),
])
->values();

// Quantos laboratorios foram concluidos nesta semana.
$labsThisWeek = $completions
->where('completed_at', '>=', Carbon::now()->startOfWeek())
->count();
?>
```

Listagem 7 – Agregação dos dados de desempenho da página de estatísticas.

A terceira frente atendeu à **HU09**, com a elaboração de uma **documentação de instalação** acessível ao usuário. O arquivo `README.md` do repositório foi reestruturado para descrever o propósito do projeto, os laboratórios disponíveis, a *stack* utilizada, os pré-requisitos e o passo a passo de instalação e execução do ambiente. Em complemento, foi publicada uma *landing page* por meio do **GitHub Pages**, hospedada a partir de uma página estática no próprio repositório, que apresenta a plataforma e reúne, com exemplos, as instruções de instalação e configuração necessárias para colocar a aplicação em funcionamento em um ambiente local. A Figura 20 apresenta essa página de divulgação e instalação.

Figura 20 – *Landing page* do OwnedBox publicada no GitHub Pages.

Por fim, esta sprint contemplou a revisão das medidas relacionadas à **HU08**[9]. Por se tratar de um requisito de caráter **não funcional**, voltado à proteção dos dados do usuário, sua implementação não correspondeu a uma única entrega, mas foi atendida de forma incremental ao longo do desenvolvimento: o armazenamento de senhas por meio de *hash* foi introduzido na *Sprint 1*, enquanto a proteção das requisições por *token* CSRF, a comparação de *tokens* resistente a ataques de temporização e o isolamento das instâncias vulneráveis foram consolidados nas *Sprints 2* e *3*. Nesta etapa, tais mecanismos foram revisados e validados em conjunto com o orientador, confirmando a adequação das práticas de segurança adotadas.

Ao término da *Sprint 4*, o **OwnedBox** encontrava-se não apenas funcional, mas também documentado e refinado, oferecendo ao usuário o acompanhamento detalhado de seu desempenho e o material necessário para a instalação autônoma da plataforma.

6.6 Considerações sobre o desenvolvimento

Ao longo das sprints apresentadas, o **OwnedBox** evoluiu de um ambiente de desenvolvimento recém-configurado até um protótipo funcional capaz de cadastrar e autenticar usuários, instanciar laboratórios vulneráveis isolados sob demanda, registrar o progresso de aprendizado por meio de *tokens* validados e apresentar relatórios de desempenho ao usuário. Nas etapas finais, o sistema foi ainda refinado e documentado, com a padronização das telas e das *flags* e a disponibilização de uma página de instalação, conforme avaliado em conjunto com o orientador. O desenvolvimento incremental, apoiado no versionamento com Git, permitiu identificar e corrigir limitações ao longo do percurso, como a necessidade de adaptar a estrutura dos laboratórios na *Sprint 3*, sem comprometer as entregas já consolidadas.

A decisão de centralizar a configuração dos laboratórios no `LabController` mostrou-se acertada, conferindo ao sistema um padrão claro e extensível para a inclusão de novos

[9] **HU08**: Como usuário da OwnedBox, quero ter garantia de que meus dados estão protegidos, para utilizar a aplicação web sem riscos.

módulos. Da mesma forma, a separação entre a etapa de prototipação visual e a implementação da lógica de negócio, adotada na *Sprint 1*, contribuiu para a coerência da interface e para a validação antecipada da experiência do usuário. As funcionalidades essenciais previstas no escopo mínimo viável foram, assim, implementadas de forma integrada, estabelecendo uma base sólida para futuras evoluções da plataforma.

7 CONSIDERAÇÕES FINAIS

A crescente complexidade dos sistemas digitais e a intensificação dos ataques cibernéticos tornam indispensável a adoção de práticas avançadas de segurança da informação. Nesse contexto, o pentest consolida-se como uma estratégia essencial para identificar vulnerabilidades e fortalecer defesas, desde que conduzido de forma ética, legal e controlada. A disponibilidade de laboratórios virtuais e de aplicações intencionalmente vulneráveis possibilita que estudantes, pesquisadores e profissionais desenvolvam suas habilidades sem infringir a legislação vigente, de modo que a formação prática em cibersegurança não apenas contribui para a capacitação técnica, mas também promove uma cultura de responsabilidade e conformidade legal.

Diante desse cenário, este trabalho teve como objetivo geral desenvolver a aplicação web **OwnedBox**, concebida para funcionar como um laboratório de ensino em cibersegurança ofensiva e defensiva, possibilitando a exploração prática de vulnerabilidades web comuns em um ambiente controlado, seguro e didático. A partir da fundamentação teórica, da análise de ferramentas correlatas, como o HTB, o TryHackMe e o DVWA, e da definição de um escopo mínimo viável, o sistema foi projetado e implementado de forma iterativa e incremental, conduzindo a proposta inicial até um protótipo funcional efetivamente operante.

Conclui-se que o objetivo geral foi atingido. Ao longo das *sprints* descritas no Capítulo 6, a plataforma evoluiu de um ambiente de desenvolvimento recém-configurado para um sistema capaz de cadastrar e autenticar usuários, instanciar laboratórios vulneráveis isolados sob demanda, validar a conclusão dos desafios por meio de *tokens* e apresentar relatórios de desempenho. Os objetivos específicos, definidos na seção 1.2, foram igualmente contemplados:

- o **sistema de autenticação e cadastro de usuários** foi implementado sobre os mecanismos nativos do *framework* Laravel, com armazenamento de senhas por meio de *hash*;
- os **módulos de aprendizado ofensivo e defensivo** foram estruturados de modo a reunir, em um mesmo fluxo, a exploração da falha e a apresentação de suas contramedidas;
- os **laboratórios interativos com vulnerabilidades simuladas** foram desenvolvidos para os três cenários priorizados, a saber, *SQL Injection*, *Cross-Site Scripting* (XSS) e Upload de Arquivos Maliciosos, cada um empacotado em contêineres Docker;
- os **tokens de validação** passaram a ser gerados dinamicamente a cada instância e verificados por comparação resistente a ataques de temporização, comprovando a conclusão dos desafios;
- a **página de perfil e a de estatísticas** consolidaram o acompanhamento do progresso, exibindo o percentual de módulos concluídos e o tempo de resolução de cada desafio;

- as **contramedidas defensivas** foram associadas a cada vulnerabilidade, orientando a correção das falhas e a adoção de boas práticas;
- por fim, o **ambiente seguro, isolado e instrutivo** foi assegurado pela execução de cada laboratório em uma instância efêmera e exclusiva, complementada por mecanismos como a proteção das requisições por *token* CSRF.

Como contribuição, o OwnedBox apresenta-se como uma alternativa didática gratuita e de código aberto, disponibilizada em português e voltada à introdução de estudantes e entusiastas aos conceitos fundamentais de cibersegurança. Ao reunir, em um único exercício, a dimensão ofensiva, isto é, a exploração da vulnerabilidade na instância vítima, e a dimensão defensiva, representada pela compreensão das contramedidas correspondentes, a plataforma procura suprir lacunas observadas em soluções consagradas, frequentemente restritas pelo idioma, por custos de acesso ou pela ausência de trilhas pedagógicas adequadas ao contexto acadêmico brasileiro. Adicionalmente, a decisão de centralizar a configuração dos laboratórios e de distribuir o sistema como software de código aberto confere à solução um caráter extensível e colaborativo, permitindo que instituições de ensino e a própria comunidade proponham melhorias e desenvolvam novos laboratórios.

Reconhecem-se, contudo, as limitações inerentes ao caráter de protótipo funcional do projeto. O escopo concentrou-se nos requisitos classificados como *Must Have* pelo método MoSCoW, contemplando um único perfil de usuário e três vulnerabilidades, distribuídas por meio de instalação local. Tais delimitações, embora coerentes com os objetivos propostos, abrem caminho para a continuidade do trabalho.

Como perspectivas para trabalhos futuros, destacam-se a ampliação do catálogo de laboratórios para abranger outras vulnerabilidades recorrentes no OWASP Top Ten, a inclusão de novos perfis de usuário, como o de instrutor ou administrador, a incorporação de elementos de gamificação e *rankings*, bem como a realização de estudos de validação com usuários reais, capazes de aferir empiricamente a eficácia pedagógica da plataforma. Essas e outras possibilidades de evolução encontram-se detalhadas no Apêndice A, no qual as funcionalidades não contempladas no escopo mínimo viável são organizadas segundo o método MoSCoW, nas categorias *Should Have*, *Could Have* e *Won't Have*. Dessa forma, o OwnedBox consolida-se não apenas como um protótipo funcional alinhado aos objetivos estabelecidos, mas também como uma base sólida para evoluções futuras.

REFERÊNCIAS

BRASIL. **Lei n.º 12.737, de 30 de novembro de 2012 - Lei Carolina Dieckmann**. 2012. Disponível em: http://www.planalto.gov.br/ccivil_03/_ato2011-2014/2012/lei/l12737.htm. Acesso em: 08 set. 2025.

BRASIL. **Lei n.º 12.965, de 23 de abril de 2014 - Marco Civil da Internet**. 2014. Disponível em: http://www.planalto.gov.br/ccivil_03/_ato2011-2014/2014/lei/l12965.htm. Acesso em: 08 set. 2025.

BRASIL. **Lei n.º 13.709, de 14 de agosto de 2018 - Lei Geral de Proteção de Dados (LGPD)**. 2018. http://www.planalto.gov.br/ccivil_03/_ato2015-2018/2018/lei/l13709.htm. Acesso em: 08 set. 2025.

CASTELLS, M. **A Galáxia da Internet: Reflexões sobre a Internet, os Negócios e a Sociedade**. Rio de Janeiro: Jorge Zahar Editor, 2003. Tradução de *La Galaxia Internet* (2001). ISBN 9788571107403.

Hack The Box Ltd. **Hack The Box**. 2025. https://www.hackthebox.com/. [Acesso em: 2025-09-09].

IBM. **O que é um teste de penetração (pentest)?** 2025. Acesso em: 08 set. 2025. Disponível em: https://www.ibm.com/br-pt/think/topics/penetration-testing.

KASPERSKY. **A Brief History of Computer Viruses What the Future Holds**. 2020. Acesso em: 08 set. 2025. Disponível em: https://www.kaspersky.com/resource-center/threats/a-brief-history-of-computer-viruses-and-what-the-future-holds.

OWASP Foundation. **OWASP Top 10: The Ten Most Critical Web Application Security Risks**. 2021. https://owasp.org/Top10/. Acesso em: 11 set. 2025.

RandomStorm. **Damn Vulnerable Web Application (DVWA)**. 2025. https://github.com/digininja/DVWA. [Acesso em: 2025-09-09].

STALLINGS, W. **Cryptography and Network Security: Principles and Practice, 8th edition**. [*S.l.*]: Pearson, 2019. 832 p.

TRYHACKME. **TryHackMe - Plataforma de Aprendizado em Segurança Cibernética**. 2025. Acesso em: 08 set. 2025. Disponível em: https://tryhackme.com/.

União Europeia. **Regulamento (UE) 2016/679 do Parlamento Europeu e do Conselho, de 27 de abril de 2016 - Regulamento Geral de Proteção de Dados (GDPR)**. 2016. https://eur-lex.europa.eu/legal-content/PT/TXT/?uri=CELEX%3A32016R0679. Acesso em: 08 set. 2025.

APÊNDICE A – Funcionalidades Classificadas como Should Have, Could Have e Won't Have (MoSCoW)

Este apêndice apresenta as histórias de usuário classificadas como *Should Have*, *Could Have* e *Won't Have*, que representam funcionalidades desejáveis ou futuras expansões do sistema.

A.1 Should Have (Desejável, mas não obrigatório)

Estes requisitos representam melhorias importantes para a experiência do usuário e a análise de desempenho, porém não são imprescindíveis para a entrega mínima do projeto.

- Relatórios visuais de desempenho com gráficos interativos.
- Estatísticas detalhadas, incluindo tempo gasto, número de tentativas e nível de dificuldade dos módulos.

A.2 Could Have (Poderia ter em versões futuras)

Os itens abaixo são considerados opcionais e podem ser incorporados em versões futuras do sistema, ampliando suas funcionalidades e escopo de aplicação.

- Painel administrativo para gestão de usuários e módulos.
- Dashboard para professores acompanharem o desempenho de turmas.
- Ranking gamificado de usuários com base em tokens e desafios concluídos.

A.3 Won't Have

Estes requisitos foram identificados como fora do escopo atual devido à limitação de tempo e recursos disponíveis, podendo ser considerados apenas em futuras evoluções do projeto.

- Aplicativo mobile nativo.
- Integração com outras plataformas de ensino (por exemplo, Moodle).
- Suporte multilíngue.
- Implementação de todas as vulnerabilidades do OWASP Top 10.