UNIVERSIDADE FEDERAL DA PARAÍBA
CENTRO DE INFORMÁTICA
CURSO DE GRADUAÇÃO EM CIÊNCIA DE COMPUTAÇÃO

MATHEUS GAMA DOS SANTOS

SEGURANÇA EM APLICAÇÕES WEB:
UMA ABORDAGEM PRÁTICA PARA IDENTIFICAR VULNERABILIDADES

JOÃO PESSOA - PB
2023

Matheus Gama dos Santos

SEGURANÇA EM APLICAÇÕES WEB:
UMA ABORDAGEM PRÁTICA PARA IDENTIFICAR VULNERABILIDADES

Trabalho de Conclusão de Curso apresentado à Coordenação do Curso de Ciência de Computação da Universidade Federal da Paraíba como requisito complementar para obtenção do título de Bacharel em Ciência de Computação, sob orientação do professor Sr. Derzu Omaia.

João Pessoa
2023

Catalogação na publicação
Seção de Catalogação e Classificação

S237s Santos, Matheus Gama dos.
Segurança em aplicações web: uma abordagem prática para identificar vulnerabilidades / Matheus Gama dos Santos. - João Pessoa, 2023.
48 f. : il.

Orientação: Derzu Omaia.
TCC (Graduação) - UFPB/CI.

1. Aplicação web. 2. Segurança. 3. Zed attack proxy. I. Omaia, Derzu. II. Título.

UFPB/CI CDU 004.738.5

Elaborado por Michelle de Kássia Fonseca Barbosa - CRB-738

Matheus Gama dos Santos

SEGURANÇA EM APLICAÇÕES WEB:
UMA ABORDAGEM PRÁTICA PARA IDENTIFICAR VULNERABILIDADES

Trabalho de conclusão de curso submetido à Banca Examinadora designada pelo Curso de Graduação em Ciência de Computação da Universidade Federal da Paraíba como requisito para obtenção do grau de Bacharel em Ciência de Computação.

BANCA EXAMINADORA

Assinatura: _____
Prof. Derzu Omaia
(Orientador)

Assinatura: _____
Prof. Alisson Brito

Assinatura: _____
Caio Victor Do Amaral Cunha Sarmento

João Pessoa, 17 de Dezembro de 2023.

Dedico este trabalho à minha evolução profissional, acrescentando o conhecimento adquirido de conceitos, ferramentas e todo resto aqui descrito.

AGRADECIMENTOS

Gostaria de expressar minha gratidão a meu professor e orientador Derzu Omaia, por ser sempre solícito e paciente durante o desenvolvimento deste trabalho, sendo essencial para a realização do mesmo.

Agradeço também aos meus colegas e amigos que me ajudaram em momentos mais técnicos com conhecimentos que tornaram o estudo de caso, apresentado neste documento, mais robusto e próximo de uma aplicação comercial, alinhando assim com o objetivo deste trabalho.

Sou muito grato a todos os meus familiares pelo apoio que me deram durante todo esse processo, sempre me incentivando a seguir em frente e me animando nos momentos difíceis.

"Quando o homem compreende a sua realidade, pode levantar hipóteses sobre o desafio dessa realidade e procurar soluções. Assim, pode transformá-la e o seu trabalho pode criar um mundo próprio, seu eu e as suas circunstâncias".

Paulo Freire
Educação e mudança

RESUMO

Segurança na web é um conceito de grande importância e bastante discutido na atualidade, porém precariamente aplicado em alguns ambientes acadêmicos e profissionais, sendo necessário sua conscientização nos mesmos. Tendo em vista o exposto, este projeto visa, através da análise de uma aplicação web construída com frameworks populares, identificar pontos vulneráveis em termos de segurança, com base nos resultados de um teste de penetração utilizando a ferramenta Zed Attack Proxy (ZAP).

Palavras-chave: Aplicação web, segurança, Zed Attack Proxy.

ABSTRACT

Web security is a concept of great importance and much discussed today, but it is poorly applied in some academic and professional environments, requiring awareness in them. In view of the above, this project aims, through the analysis of a web application built with popular frameworks, to identify vulnerable points in terms of security, based on the results of a penetration test using the Zed Attack Proxy (ZAP) tool.

Keywords: Web application, security, Zed Attack Proxy

LISTA DE FIGURAS

LISTA DE ABREVIATURAS

API	Application Programing Interface
ASVS	Application Security Verification Standard
AWS	Amazon Web Services
CNH	Carteira Nacional de Habilitação
CORS	Cross-Origin Resource Sharing
CSP	Content Security Policy
DAST	Dynamic Application Security Testing
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IAST	Interactive Application Security Testing
JSON	JavaScript Object Notation
JWT	Json Web Token
LDAP	Lightweight directory access protocol
NoSQL	Not Only SQL
ORM	Object-Relational Mapping
OWASP	Open Worldwide Application Security Project
REST	REpresentational State Transfer
SAST	Static Application Security Testing
SQL	Structured Query Language
SPA	Single Page Application
UML	Unified Modeling Language
URL	Uniform Resource Locator
UUID	Universally Unique IDentifier
XSS	Cross Site Scripting
ZAP	Zed Attack Proxy

SUMÁRIO

1. INTRODUÇÃO

Segurança na web é um tema que vem ganhando cada vez mais atenção. Frequentemente, vemos notícias de dados confidenciais e sensíveis sendo vazados, serviços indisponíveis, crimes virtuais, entre outros problemas relacionados. Porém, estar preparado para mitigar esses problemas não é uma tarefa simples, já que requer conhecimento técnico, muitas vezes não disponível, e boas práticas de desenvolvimento que, por vezes, são negligenciadas por algumas instituições e empresas.

As empresas de pequeno e médio porte têm a tendência de possuir times de desenvolvedores em menor quantidade, além de precisarem lidar com um ritmo mais acelerado de produção, tornando-se o grupo mais afetado por esse problema. Há um foco maior nas funcionalidades e na experiência do usuário, dando pouca atenção aos requisitos de segurança. Isso faz com que seus produtos sejam vulneráveis, abrindo brechas para hackers mal-intencionados, podendo causar prejuízos financeiros, reputacionais e legais.

Além disso, a segurança na web também não é abordada de forma suficiente nos cursos de tecnologia de ensino superior, que muitas vezes não oferecem disciplinas específicas sobre o assunto ou não atualizam os conteúdos conforme as novas ameaças e soluções surgem. Isso faz com que os profissionais formados nessa área não tenham a competência necessária para desenvolver sistemas web seguros ou para realizar análises de segurança adequadas.

Diante desse cenário, é fundamental que se busque aprimorar os conhecimentos sobre a segurança na web, tanto no âmbito acadêmico quanto no profissional. Nesse sentido, este trabalho tem como propósito demonstrar o processo de análise de segurança de um sistema web, de forma básica, para que com base nos resultados, as vulnerabilidades apontadas sejam mitigadas e o sistema tenha um mínimo de segurança. Espera-se contribuir para a melhoria da qualidade e da confiabilidade dos produtos oferecidos aos usuários e conscientização e a capacitação dos desenvolvedores e gestores de sistemas web.

1.1 OBJETIVO GERAL

Este trabalho tem como objetivo principal demonstrar o processo de análise de segurança de um sistema web, de forma básica, para que com base nos resultados, as vulnerabilidades apontadas sejam mitigadas e o sistema tenha um mínimo de segurança.

Como objetivo secundário iremos descrever o funcionamento e configurações de um sistema web, nesse caso o RentX, e, com base no relatório gerado a partir de um *pentest*,

realizar uma análise básica de segurança. Espera-se que o processo descrito sirva como exemplo, para empresas e profissionais de tecnologia que não aplicam ou não dão a devida atenção à segurança em suas aplicações web, de como implementá-la em seu ambiente de forma inicial.

1.2 OBJETIVOS ESPECÍFICOS

1) Descrever e implementar o sistema RentX.
2) Definir uma abordagem de análise de segurança, dado o contexto onde o sistema está inserido.
3) Definir o processo de configuração e execução de um teste de penetração.
4) Interpretar os resultados, identificando as vulnerabilidades e meios de mitigá-las.

1.3 TRABALHOS RELACIONADOS

Para a realização deste trabalho, podem ser citadas duas fontes que se relacionam ou inspiraram sua ideia e objetivos, sendo essas apresentadas a seguir.

Em Vulnerabilidades em Sistemas Web, de OLIVEIRA, B. F (2019), são abordados os tipos de vulnerabilidade, mecanismos de segurança para mitigá-las e o uso de ferramentas para realizar o levantamento das mesmas em alguns sites alvo determinados pelo autor. Pode-se afirmar que o estudo realizado por ele tem grande semelhança com este trabalho, porém com objetivos distintos, o que ainda o torna uma grande inspiração para o tema e metodologia abordados.

O projeto RentX, que foi utilizado para o estudo de caso, não foi uma proposta de software originada pelo autor. Inicialmente, RentX era uma sugestão de aplicativo móvel, incluída em um desafio de programação pertencente a um módulo do bootcamp Ignite, da empresa Rocketseat (ROCKETSEAT, 2023), fornecendo arquivos de design no Figma, modelagem UML e até mesmo a API da aplicação. O diagrama UML e API foram refeitos para se adequar ao contexto, retirando, adicionando e reescrevendo componentes e regras de negócio, para transformar a ideia de um aplicativo, em uma aplicação web SPA com estrutura comercial. Pode-se dizer que a ideia e recursos providenciados do RentX foram essenciais para o desenvolvimento do trabalho.

2. REFERENCIAL TEÓRICO

Esta seção tem por finalidade estabelecer conceitos teóricos que são utilizados no trabalho. É dividida em 2 partes, sendo a primeira sobre conceitos de segurança, fontes de referência em segurança e definições ou termos que fazem parte do universo de engenharia de software e a segunda sobre ferramentas utilizadas no trabalho.

2.1 CONCEITOS

Esta subseção irá conceituar fontes que fazem parte do embasamento teórico do trabalho em quesitos de segurança, bem como termos que são utilizados no estudo de caso, como padrões arquiteturais e de projeto.

2.1.1 OWASP

O Open Web Application Security Project, ou OWASP, é uma organização internacional sem fins lucrativos dedicada a conceber, adquirir, operar e manter aplicações confiáveis. Um dos princípios fundamentais do OWASP é que todos os seus materiais estejam disponíveis gratuitamente e sejam de fácil acesso em seu site, tornando possível para qualquer pessoa melhorar a segurança de suas próprias aplicações web. Os materiais que eles oferecem incluem documentação, ferramentas, vídeos e fóruns. (OWASP, 2020)

2.1.2 OWASP TOP TEN

OWASP Top Ten List of the Most Critical Web Application Security Risks, ou, simplesmente, OWASP Top Ten, é um relatório, atualizado regularmente, a partir de dados objetivos e subjetivos fornecidos por organizações patrocinadoras da OWASP, que descreve as dez principais questões de segurança para aplicações web. Esse relatório é elaborado por uma equipe de especialistas em segurança do mundo todo e se concentra nos dez riscos mais críticos para a segurança de aplicativos da web. (SULLIVAN; LIU, 2012)

2.1.2 CONTROLE DE ACESSO FALHO (OWASP TOP10 A01 2021)

O controle de acesso é uma medida de segurança que garante que os usuários só possam executar ações dentro das permissões fornecidas. Quando falha, pode levar à divulgação, modificação ou destruição não autorizada de dados ou executar ações não autorizadas àquele tipo de usuário. Expandindo o conceito, temos alguns exemplos, como:

- Violação do Princípio do Privilégio Mínimo ou Negar por Padrão, que diz que qualquer recurso deve estar disponível para qualquer tipo de usuário, mas regras devem ser criadas para o acesso dos mesmos.
- Modificar componentes da aplicação, URL ou interceptar requisições à API, mudando seu conteúdo de alguma forma (ataque man-in-the-middle), para burlar o controle de acesso.
- Acessar informações de outros usuários, ou realizar ações de privilégio elevado indevidamente.
- Configuração falha de Cross-origin Resource Sharing (CORS), permitindo acesso de fontes desconhecidas, potencialmente perigosas, ou sem autorização. (OWASP, 2021)

2.1.2 ATAQUES DE INJEÇÃO (OWASP TOP10 A02 2021)

Ataques de injeção acontecem quando os dados que o usuário fornece não sofrem validação, filtragem ou sanitização por parte da aplicação, abrindo espaço para consultas ou execução de ações maliciosas através de parâmetros dinâmicos usados na construção de queries ou durante a execução de uma ação que não leva em consideração o contexto em que está sendo executada.

Alguns dos ataques mais comuns desta categoria são injeções SQL, NoSQL, comandos de sistema, ORMs, LDAP, entre outros. Geralmente a identificação de vulnerabilidades desse tipo podem acontecer através de testes automatizados, sejam eles estáticos (SAST), dinâmicos (DAST) ou interativos (IAST), usando ferramentas no pipeline de CI/CD, impedindo que essa falha de estrutura chegue ao ambiente de produção. (OWASP, 2021)

2.1.3 FALHA DE CONFIGURAÇÃO (OWASP TOP10 A05 2021)

Seguindo a definição da OWASP, falhas de configuração remetem à falta de configurações de segurança ou existência delas de forma inapropriada ou incompleta. Alguns exemplos seriam:

- Ausência de segurança apropriada em qualquer parte da aplicação produto ou configurações impropriamente configuradas em serviços da nuvem.
- Funcionalidades desnecessárias habilitadas ou instaladas, como portas desnecessariamente abertas, serviços, páginas, contas ou privilégios.

- Contas criadas por padrão, como, por exemplo, de administrador para alguns bancos de dados, ainda habilitadas ou com credenciais não modificadas.
- Mensagens de erro que não sofrem tratamento e retornam mais informação do que deveriam para o usuário.
- Configuração, desde variáveis de sistema, até variáveis usadas por frameworks, não contendo valores seguros. (OWASP, 2021)

2.1.4 COMPONENTES VULNERÁVEIS E DESATUALIZADOS (OWASP TOP10 A06 2021)

Considerando componentes como qualquer componente da aplicação, seja ele do lado do cliente ou servidor, temos que o sistema se encontra vulnerável quando:

- Desconhece-se as versões de todos os componentes usados diretamente e suas dependências.
- O software se encontra vulnerável, sem suporte, ou vencido. Isso inclui o sistema operacional, servidor, sistema de gerenciamento de banco de dados, aplicações, APIs e todos os componentes, ambientes de execução e bibliotecas.
- Não ocorrem análises de vulnerabilidade, nem verificação de informações de segurança, sobre os componentes utilizados, regularmente.
- Não são corrigidos problemas ou realizadas atualizações na plataforma, frameworks e dependências de forma regular, mesmo após soluções e versões já terem sido publicadas abertamente, deixando o ambiente exposto e desatualizado.
- Há ausência de testes de compatibilidade em bibliotecas recém atualizadas. (OWASP, 2021)

2.1.5 APPLICATION SECURITY VERIFICATION STANDARD (ASVS)

O Application Security Verification Standard (ASVS) é um padrão de segurança de aplicativos aberto, desenvolvido e mantido pelo Open Web Application Security Project (OWASP). É um esforço da comunidade para criar um conjunto de requisitos e controles de segurança focados na definição dos recursos funcionais e não funcionais necessários para projetar, desenvolver e testar sites modernos, com o objetivo de ajudar as organizações a criar confiança na integridade das suas aplicações e a identificar e mitigar riscos.

O principal objetivo do ASVS é padronizar a cobertura e o rigor disponíveis no mercado para auditorias de segurança de dados de aplicações web utilizando um padrão

aberto comercialmente viável. O padrão fornece uma base para testar controles de segurança técnica, ambiental e de aplicativos, que devem proteger contra vulnerabilidades como Cross-Site Scripting (XSS) e injeção de SQL.

O ASVS define 3 níveis de verificação de segurança, com cada nível aumentando em profundidade. O nível um é o nível básico de requisitos de segurança de um aplicativo, apresentando garantia baixa e sendo completamente testável quanto à penetração. O nível dois é para aplicações que contém dados confidenciais que requerem proteção e é recomendado para a maioria das aplicações. O nível três é para aplicações mais críticas, ou seja, que executam transações de alto valor, contém dados médicos ou que exijam o mais alto nível de confiança.

O formato do documento se define em um conjunto de requisitos de segurança que fornecem proteção adequada contra ataques comuns a aplicativos da web, cada um tendo sua indicação do nível em que deve estar incluído. (OWASP, 2021)

2.1.6 ARQUITETURA EM CAMADAS

Arquitetura em Camadas, como o nome infere, divide o sistema em camadas, de maior ao menor nível, cada parte exercendo uma função no sistema/aplicação. As camadas mais superficiais são de alto nível, com interfaces de interação com usuário, como front-ends de sites e interfaces de aplicações de desktop, enquanto as camadas mais inferiores, se aproximam mais do conjunto de instruções de máquina. As camadas intermediárias têm papel de execução de serviços e/ou funções. (KHAN; A AGRAWAL, 2014)

2.1.7 TESTES DE PENETRAÇÃO (PENTEST)

O teste de penetração, ou *pentest*, pode ser definido como uma tentativa legal e autorizada de localizar e se aproveitar de sistemas, aplicações ou redes de computador com o intuito de deixá-los mais seguros. Pode-se resumir o processo em apontar as vulnerabilidades de um sistema, através de ataques que não tem a intenção de danificá-lo, e fornecer recomendações específicas para os problemas encontrados durante o teste. (ENGEBRETSON, 2013)

O processo de *pentest* pode ser dividido em 5 estágios:

- Planejamento e Reconhecimento: definir escopo e objetivos do teste, incluindo quais sistemas ou partes do sistema deverão ser analisadas e os métodos a serem utilizados.

- Escaneamento: identificar meios de entrada e serviços existentes em um sistema, como portas, IPs, entre outros.
- Obter Acesso: nesse caso, “obter acesso” se refere a tomar o controle sobre o sistema, explorando suas vulnerabilidades, no sentido de burlar falhas de segurança e controlá-lo, podendo comprometê-lo parcial ou totalmente.
- Manter o Acesso: verificar se a vulnerabilidade cria uma brecha para acesso persistente, ou longo o suficiente para que se torne cada vez mais profundo.
- Análise: gerar um relatório com o resultado detalhando vulnerabilidades específicas, conteúdo acessado e tempo até o ataque ser detectado. (IMPERVA, 2019)

2.1.8 BCRYPT

Bcrypt é um algoritmo de *hash*, propositalmente lento, que introduz valores aleatórios, chamados de *salt*, múltiplas vezes no momento do *hashing* dos dados de entrada. Ideal para ser utilizado em senhas, dado que dificulta exponencialmente os ataques de força bruta, realizados a partir de *rainbow tables*, dicionários contendo as senhas mais frequentemente utilizadas, protegendo assim o banco de dados mesmo que ele tenha sido comprometido. (BATUBARA; EFENDI; NABABAN, 2021)

2.1.9 JSON WEB TOKEN (JWT)

JSON Web Token, ou JWT de forma sucinta, é um padrão aberto (RFC 7519) que define uma maneira compacta e *URL-safe*, ou seja, ausente de caracteres que não podem ser representados em uma URI, de representar informações que vão ser transmitidas entre duas partes. O conteúdo é definido em um JSON que será codificado e introduzido no payload do token, sendo assinado digitalmente utilizando algum segredo, usando algoritmo HMAC. (JONES. M.; BRADLEY. J.; SAKIMURA. N., 2015)

2.2 FERRAMENTAS

Está subseção irá conceituar ferramentas utilizadas no projeto do estudo de caso, além de softwares de análise de vulnerabilidades.

2.2.1 REACT

React é uma biblioteca do javascript criada pelo antigo Facebook, agora Meta, que permite que você construa interfaces de usuário a partir de peças individuais chamadas

componentes, facilitando o reaproveitamento e manutenção de código. (META OPEN SOURCE, 2023)

O React surgiu com o objetivo de otimizar a atualização e sincronização de atividades simultâneas no feed do Facebook, como chat, status, entre outros, que na época eram consideradas atividades de descrição complexa. Por ter mostrado grande eficiência, foi incorporado em outras redes sociais e posteriormente se tornou código aberto, o que fez com que se tornasse a biblioteca mais popular do mercado para desenvolvimento de interfaces de usuário. (ROVEDA. U, 2023)

2.2.2 NODE

Segundo o site Mozilla e documentação do próprio Node, ele é uma ferramenta/ambiente de desenvolvimento assíncrono baseado em eventos, de código aberto, que executa javascript fora do browser, no lado do servidor (backend), em tempo real. (MOZILLA, 2023)(TSC, 2023)

Hoje, é usado principalmente para desenvolvimento de sistemas web, com performance e escalabilidade excelentes, em conjunto com vários frameworks, como o Express.js, Nest.js, entre outros. Além disso, possui uma comunidade muito ativa que alimenta o gerenciador de pacotes node (NPM) com diversas bibliotecas e ferramentas para desenvolvimento de aplicações em geral. (MOZILLA, 2023)

2.2.3 EXPRESS

Express.js é um framework de aplicações web construído com Node.js que dispõe de diversas funcionalidades para aplicações tanto da web quanto mobile. Essa ferramenta tem o intuito de facilitar a construção de APIs, trazendo vários métodos utilitários HTTP e middlewares com implementação e utilização acessíveis. Atualmente, diversos outros frameworks utilizam o Express na base de sua construção, como o Sails, Blueprint, Nestjs, entre outros. (OPENJS FOUNDATION, 2023)

Em uma pesquisa feita em 2023 pela Enlyft, uma empresa que fornece soluções de inteligência de negócio e análise de dados para empresas, com 88.087 empresas que usam o framework, Express é bastante utilizado nos Estados Unidos, com 36%, na maioria dos casos em empresas de pequeno e médio porte.

2.2.4 POSTGRESQL

PostgreSQL é um banco relacional open source, com mais de 35 anos de desenvolvimento ativo, que ganhou bastante reconhecimento por ser robusto, performático e confiável. Possui diversas funcionalidades para ajudar desenvolvedores e administradores a desenvolver ambientes robustos para seus sistemas, manter a integridade e facilitar a manipulação e manutenção dos dados. (POSTGRESQL GLOBAL DEVELOPMENT GROUP, 2023)

2.2.5 TYPESCRIPT

Typescript é uma linguagem de programação open source desenvolvida pela Microsoft, que veio como uma extensão do javascript, trazendo tipagem forte a uma das linguagens mais usadas no mercado atualmente, além de ferramentas para facilitar a manipulação e definição de tipos.

A proposta é aumentar a integração com os editores de texto e ferramentas de lint, melhorando o controle sobre o código da aplicação javascript com a adição de sintaxe de tipos. No momento do build da aplicação, o código é transpilado para javascript puro, tornando o typescript altamente aplicável para ambientes que já utilizam o javascript. (MICROSOFT, 2023)

2.2.6 PRISMA

Prisma é uma ferramenta de mapeamento objeto-relacional (ORM) de código aberto que simplifica a comunicação e interação de um banco relacional e um aplicativo. O ORM é uma técnica que mapeia objetos de uma linguagem de programação para tabelas de um banco de dados relacional.

Dentre as vantagens do uso do prisma estão: suporte para integração com frameworks populares (Express, GraphQL, Next.js, entre outros), migrações de banco geradas automaticamente, fácil instalação e implementação em projetos, linguagem de modelagem simples, cliente type-safe e fácil de usar (para realizar operações no banco de dados via código), além de um visualizador de banco integrado que pode ser executado no navegador, sem a necessidade de instalações adicionais. (PRISMA, 2023)

2.2.7 DOCKER

Docker é uma plataforma para criar, implantar e gerenciar aplicações de forma rápida e padronizada. A aplicação é empacotada e executada em um ambiente isolado, chamado de *container*. Vários *containers* podem rodar concomitantemente na mesma máquina *host*. Como são autocontidos e leves, não é necessário se preocupar com nenhuma dependência estar presente no sistema onde vai ser executado, garantindo portabilidade e facilidade, independente se a aplicação está em ambiente de desenvolvimento ou produção. (DOCKER, 2020)

2.2.8 ZED ATTACK PROXY (ZAP)

Zed Attack Proxy (ZAP) é uma ferramenta gratuita e open-source para realizar testes de penetração, desenvolvido por membros da OWASP. Foi construído para testar aplicações web, sendo bastante configurável e extensível, através da permissão de criação de scripts e uso de *Add-ons* de seu próprio *marketplace*.

A ferramenta faz parte de um grupo chamado “man-in-the-middle proxy”, ou seja, intercepta as requisições entre o browser e a aplicação web, inspecionando as mensagens enviadas, modificando seu conteúdo, caso necessário, e direcionando esses pacotes para o destino. (ZAP, 2023)

3. METODOLOGIA E DESENVOLVIMENTO

Nesta seção, será apresentado o processo de análise de um sistema web, abrangendo também informações sobre o estudo de caso. Serão abordados: o procedimento de levantamento dos pontos de análise, ferramentas utilizadas e detalhamento arquitetural e funcional do estudo de caso. Os resultados do que será exposto será apresentado na seção de resultados.

3.1 ESCOLHA DA ESTRATÉGIA DE ANÁLISE

Para realizar uma análise de segurança é preciso definir o escopo e o nível de confiança que a aplicação deve ter. Para isso, foi utilizado, como auxílio o documento ASVS definido pela OWASP. Dado o objetivo deste trabalho, foi escolhido nível um, que seria o nível mais básico, definindo assim se uma aplicação teria o mínimo para não ser considerada uma aplicação insegura.

Como dito anteriormente, o nível um pode ser validado através de testes de penetração, e por isso se deu a necessidade de procurar uma ferramenta para tal validação. Com o resultado dos testes gerados por essa ferramenta, seria feita uma análise com base no relatório, realizando um levantamento dos problemas do sistema e possivelmente propondo uma linha de ação para mitigar o problema.

Testes de penetração são bastante intrusivos, simulando ataques externos à uma aplicação. Logo, seria necessário obter um ambiente alvo em que pudessem ser feitos os testes, sem riscos de punição ou complicações judiciais. Dada a situação, optou-se por desenvolver um sistema e providenciar o ambiente adequado, evitando completamente os problemas citados.

3.2 ESTUDO DE CASO

Para o sistema alvo, como uma forma de tornar o exemplo relevante, foram escolhidas linguagens, frameworks, banco de dados e soluções de hospedagem populares, além de produzir um sistema com funcionalidades úteis para que pudesse ser considerado um produto de fato.

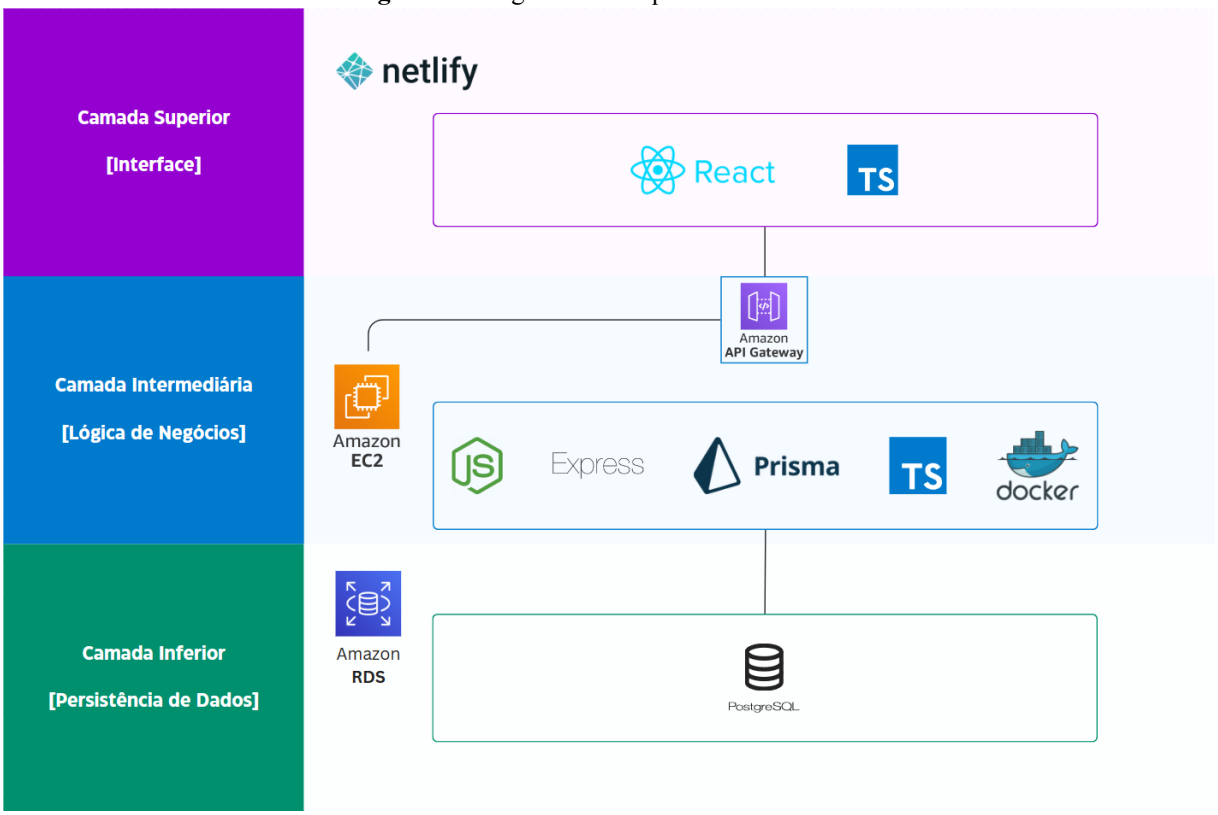
O produto é um sistema de aluguel de carros, chamado RentX, inspirado em um projeto do curso Ignite, da Rocketseat, onde os usuários poderiam criar contas e agendar o aluguel dos carros disponíveis, selecionando o período desejado antes de realizar o agendamento. Toda parte de pagamento, entrega e devolução seria feita presencialmente na agência da companhia responsável por gerenciar o sistema, portanto a utilidade do produto seria facilitar o processo de aluguel de carros, tanto para o usuário, quanto para a agência.

Nas próximas subseções, será feito o detalhamento de cada componente do sistema e suas possíveis interações e fluxos de funcionamento.

3.2.1 VISÃO GERAL

Para a construção do sistema, optou-se pela arquitetura em três camadas, tendo o frontend em React.js como a camada superior, responsável pela interface do usuário, uma API REST, responsável pela lógica de negócios, como a camada intermediária que se comunica com a camada inferior que é responsável pela persistência dos dados. Abaixo temos um diagrama representando a arquitetura do sistema:

Figura 1 - Diagrama de Arquitetura do RentX.



Fonte: Autoria própria.

Temos a camada superior, com uma aplicação React e Typescript, hospedada em um servidor https do Netlify. Adentrando a camada intermediária, temos o API Gateway da AWS, que recebe todas as requisições do front end e, a partir de um mapeamento de rotas, repassa a requisição para o servidor da instância EC2, onde uma API Node, com o framework Express.js, Prisma e Typescript, é executada em um *container* docker. Por fim, a camada inferior contém uma instância de um banco de dados PostgreSQL na AWS RDS, recebendo as queries realizadas pela camada intermediária.

3.2.2 FRONT END

O frontend da aplicação foi feito usando o Vite, que nada mais é que uma ferramenta moderna de geração de *build*, que em ambiente de desenvolvimento disponibiliza ES *modules* nativos para o browser, tornando inicialização e atualização do app expressivamente rápidas, e, em produção, o *bundle* gerado é altamente otimizado. Tornou-se padrão para templates Vue e React e tem suporte para Typescript e JSX. (VITE, 2019)

Com exceção de componentes como alguns *inputs* que possuem interações um tanto mais complexas, o restante foi desenvolvido com React e Typescript puros, ou realizando

pequenas customizações em componentes já prontos de bibliotecas conhecidas, como Radix-UI.

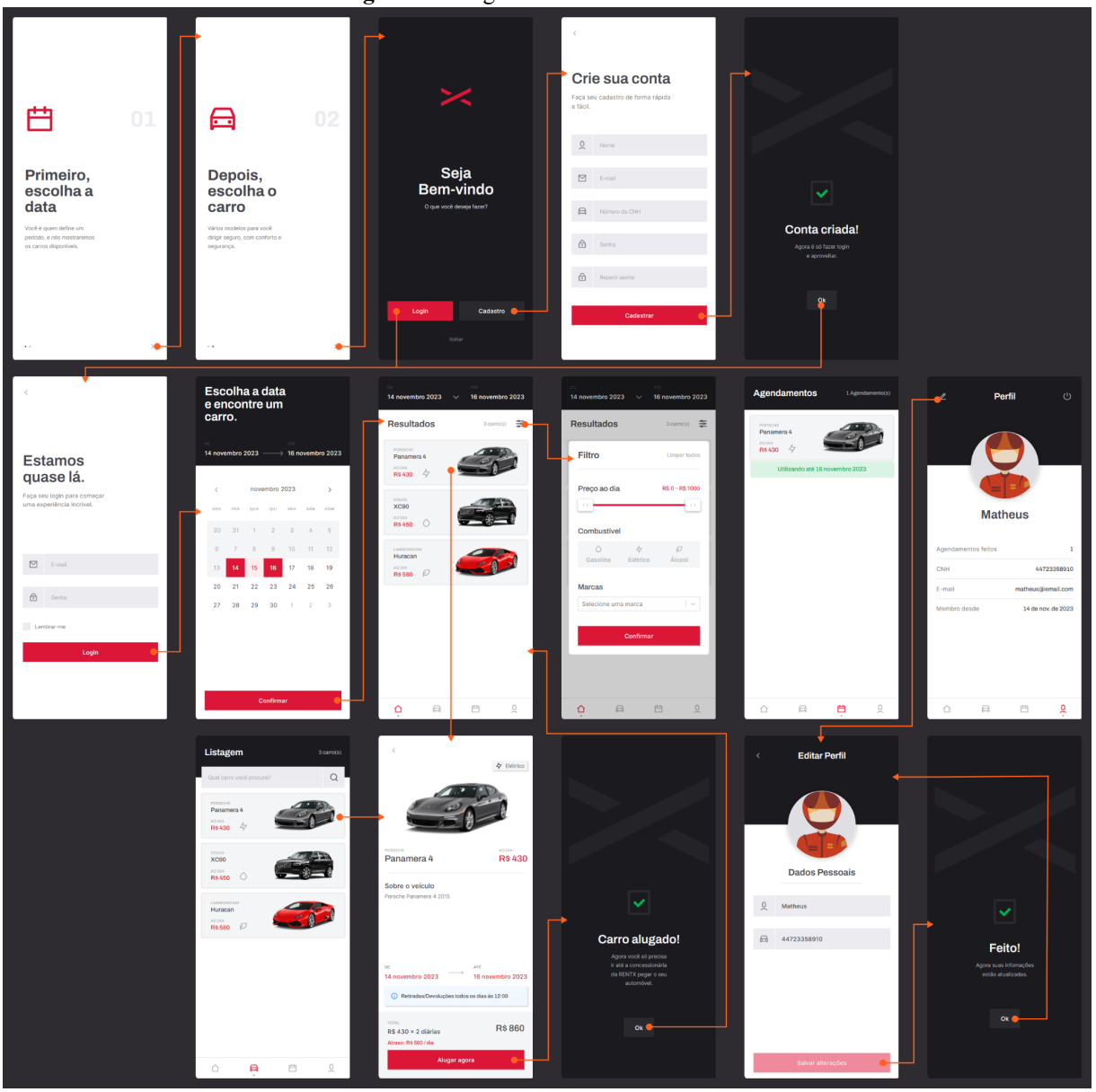
Devido ao escopo do projeto ser pequeno, optou-se por não utilizar uma biblioteca de gerenciamento de estados, como Redux ou MobX, utilizando *hooks* e a Context API do próprio React para tornar informações e funções disponíveis para toda a aplicação.

Para a realização de requisições, foi adotada a biblioteca Axios, conhecida por sua simplicidade e eficiência. Em relação a configurações realizadas, apenas foi definida a URL base do servidor, para o qual serão direcionadas as requisições HTTP para buscar, modificar e persistir dados no sistema. Importante ressaltar que é feita a importação dessa URL das variáveis de ambiente, que, no momento de geração da build de produção, tem as instâncias de importação substituídas pelos valores reais das variáveis, já que em aplicações client-side, o servidor serve a aplicação para o browser do cliente, não tendo acesso às variáveis de ambiente do servidor de produção onde está hospedado pelo javascript. Consequentemente, o endereço da API fica exposto para qualquer indivíduo que consiga utilizar as ferramentas de desenvolvedor do navegador. Porém, é dever da própria API controlar os acessos, por meio de mecanismos de segurança, para impedir interações indesejadas.

Para a autenticação, foi criado um *hook* que expõe para a aplicação dados do usuário autenticado, além de fornecer funções de *login* e *signout*. A função de *login* recebe os dados do formulário de *login* e realiza uma requisição para a API REST, na tentativa de autenticar o usuário. Caso o processo de autenticação seja bem sucedido, a API retornará um token JWT, que servirá como token de sessão, sendo enviado no cabeçalho de todas as requisições daí em diante. Há a opção de o usuário persistir o login, salvando o token no localstorage, podendo ser recuperado posteriormente caso a aplicação seja aberta em outra aba ou instância do navegador, sem a necessidade de realizar o processo de login novamente. Mais detalhes sobre o token de sessão gerado serão fornecidos na próxima seção do trabalho.

Muitas aplicações hoje em dia adotam o método de design *mobile first*, que significa criar o design de um site para desktop começando com a versão mobile, para depois adaptar para telas maiores, sendo este o caso para a aplicação desse estudo de caso. Na Figura 2, podemos visualizar o diagrama de fluxo do usuário para o RentX, que ilustra o caminho que o usuário percorre para completar tarefas e navegar pelo site.

Figura 2 - Diagrama de Fluxo do Usuário



Fonte: Autoria própria.

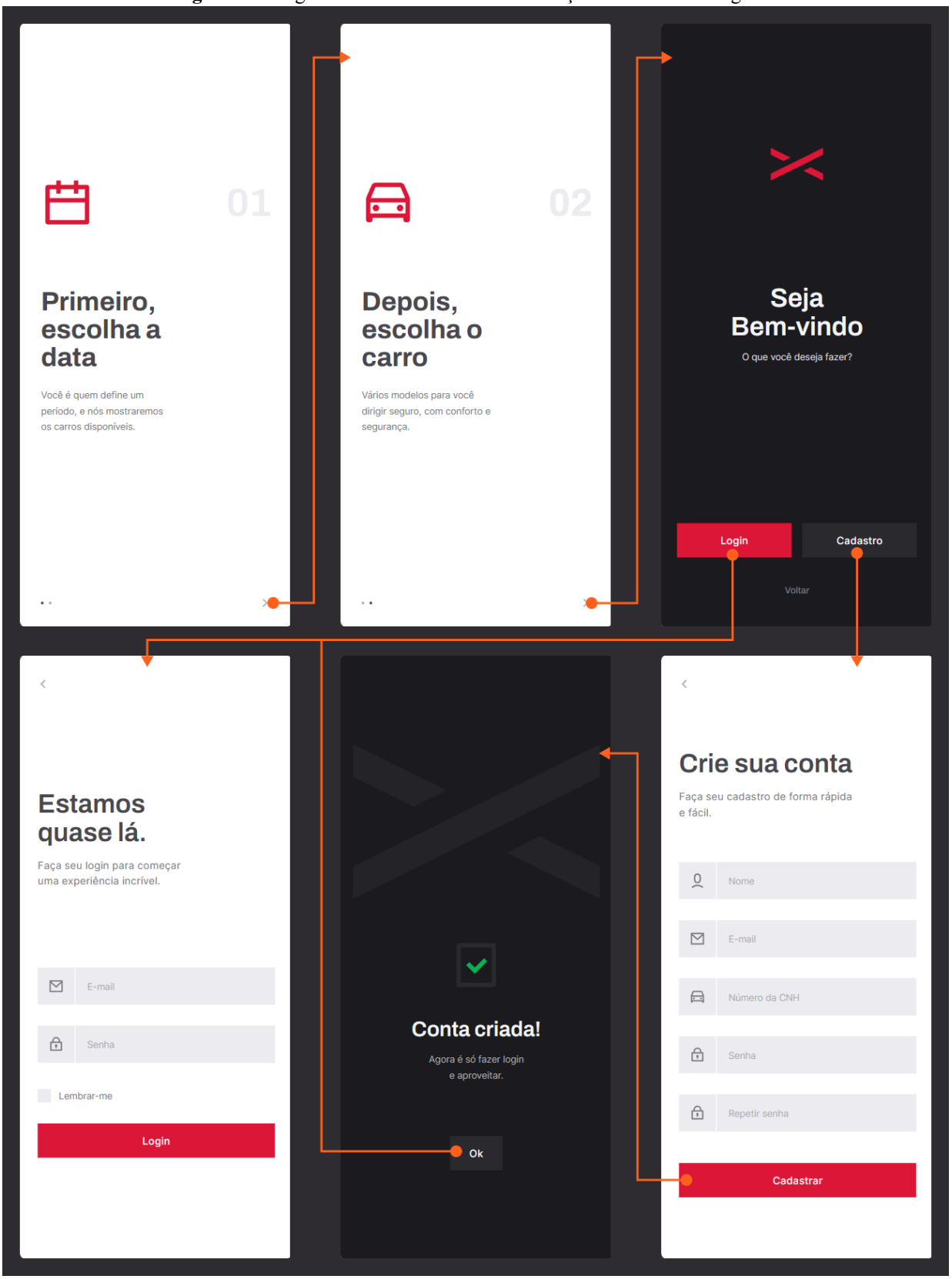
Podemos dividir o uso da aplicação em suas tarefas em 3 partes: criação de uma conta e login do usuário, processo de realizar agendamento de aluguel de carros e visualizar os agendamentos e dados do usuário, com possibilidade de edição desses dados. A Figura 3 ilustra o fluxo 1, começando com uma introdução breve, descrevendo por alto os passos para executar a tarefa principal da aplicação. Após as telas de introdução, o usuário se depara com uma tela que possui botões que levam para a tela de login ou tela de cadastro. Se escolhido fazer o cadastro, será levado para a tela relacionada, tendo que preencher um formulário contendo nome, endereço de e-mail, número da CNH e senha.

Para a validação de formulários no frontend, assim como para a definição de tipagens dos modelos de negócio dentro da API, foi utilizado o Zod, que é uma biblioteca de validação

de esquemas integrando Typescript. Esquemas são a representação de um tipo de dado, seja esse um valor primitivo ou estruturas mais complexas. Definindo o tipo de cada campo, podendo adicionar regras como tamanho do conteúdo, verificação de padrões por meio de regex, entre outros métodos fornecidos pelo próprio Zod, é possível criar uma validação para cada input, com mensagens de erro personalizadas, para garantir que dados passados por esse formulário estejam no formato do valor que é esperado. Após clicar no botão “Cadastrar”, é feita uma requisição POST para o *endpoint* “/users”, utilizando a biblioteca Axios, criando o usuário, a partir dos dados fornecidos no corpo da requisição, e direcionando para uma tela que exibe o feedback positivo do processo e um botão “Ok”, que se clicado leva para a tela de login.

A tela de login possui um formulário que solicita e-mail e senha, também com validação semelhante à da tela de cadastro, além de ter uma checkbox que dá opção de persistir a sessão para os próximos acessos. Ao submeter este formulário, é realizada uma requisição POST para o *endpoint* “/sessions”, com os valores desses campos no corpo da requisição, retornando informações do usuário e um token de autenticação JWT, que será imbuído no cabeçalho de todas as requisições enquanto a aba do navegador não for fechada, além de ser salvo no localStorage, caso o usuário opte pela persistência da sessão. Caso uma nova aba ou instância do navegador seja aberta, na inicialização da aplicação, o contexto de autenticação, criado com a Context API do React, tenta recuperar esses dados do localStorage e atualizar os headers das requisições subsequentes. Se a autenticação for bem sucedida, o usuário é direcionado para a página principal.

Figura 3 - Diagrama de Fluxo do Usuário: Criação de Usuário e Login



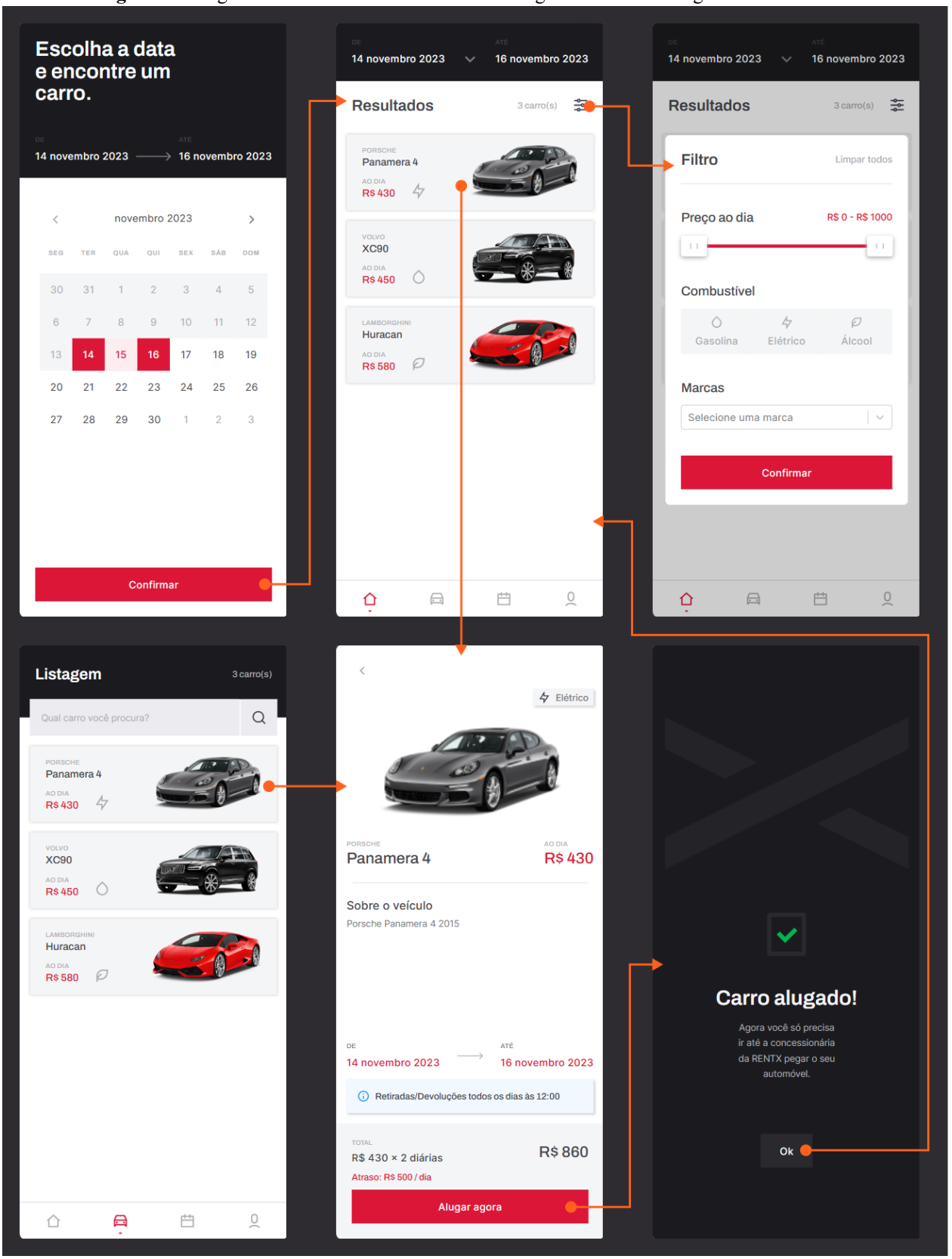
Fonte: Autoria própria

A Figura 4 ilustra o fluxo 2, processo de realizar um agendamento. Ao acessar a aplicação na página inicial, o usuário se depara com uma modal de tela inteira, contendo um calendário para selecionar o período de aluguel desejado e após a confirmação das datas, é exibida uma listagem de carros disponíveis. No canto superior direito, há um botão de filtro, que se clicado, abre uma modal com filtros disponíveis para a filtragem da lista por faixa de preço da diária, tipo de combustível e marca do veículo. Essa tela faz parte do conjunto de 4 telas acessíveis pela barra de navegação, que lista as páginas: principal, de listagem por pesquisa, lista de agendamentos e lista de informações do usuário.

Na página de listagem por pesquisa há uma barra onde pode ser inserido o nome ou parte do nome do carro desejado, sendo feita uma pesquisa inclusiva do valor. As telas inicial e de listagem realizam a mesma requisição GET para o endpoint "/cars/available", passando por *query params* os atributos respectivos em cada caso e recebendo uma lista de carros disponíveis para aluguel, sendo exibida através de cards com a foto e informações básicas de cada carro.

Ao clicar no card, o usuário é levado para a página de detalhes do carro, contendo nome, marca, preço da diária, descrição e custo sobre atraso por diária excedida. Além disso, é exibido o período escolhido para o agendamento, um aviso sobre o horário de início e término das diárias e um cálculo do total a pagar. Ao clicar no botão "Alugar agora", é feita uma requisição POST para o endpoint "/rentals", fornecendo o id do carro, e as datas de início e fim do agendamento, pelo corpo da requisição. Caso o usuário não esteja autenticado ou o token esteja inválido, será levado para a tela de login. Sendo a requisição bem sucedida, há o direcionamento para a tela de feedback positivo, com um botão "Ok", que quando clicado leva para a página principal novamente.

Figura 4 - Diagrama de Fluxo do Usuário: Realizar Agendamento de Aluguel de Carros.



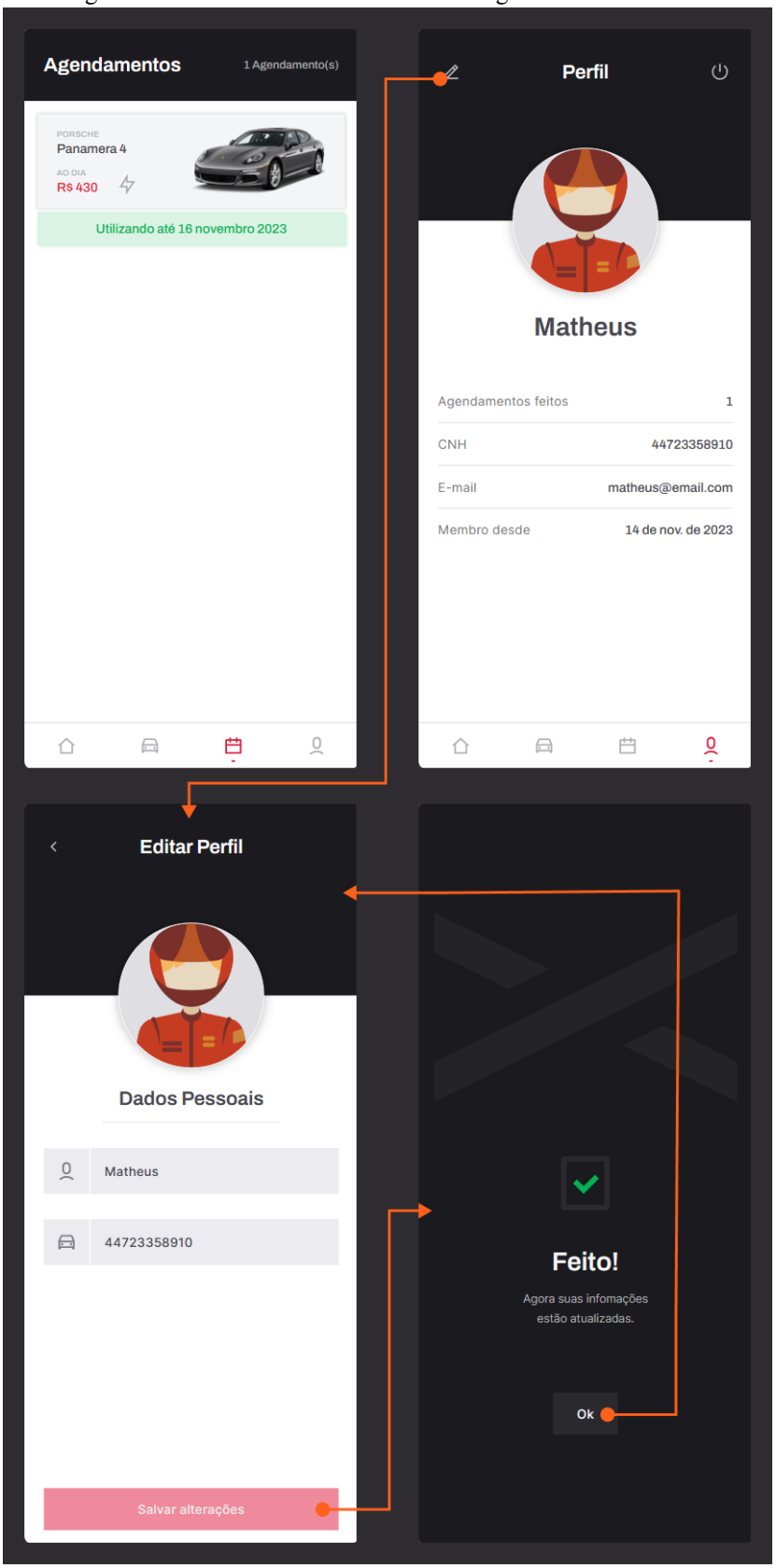
Fonte: Autoria própria

Por fim, a Figura 5 ilustra o fluxo 3, visualizar agendamentos e dados do usuário. A tela de agendamentos do usuário é acessada pela barra de navegação, por uma das outras 3 telas, ou diretamente pela barra de endereços do navegador. O usuário deve estar autenticado para ter acesso a essa página, sendo redirecionado para a página de login caso não esteja. É realizada uma requisição GET para o endpoint "/rentals/user", retornando uma lista com todos os agendamentos do usuário e suas informações, sendo exibidos na tela como cards sem interação, seguidos do status do agendamento.

A página de perfil do usuário é acessada da mesma forma que a tela da lista de agendamentos e também restringe o acesso a usuários autenticados, redirecionando para a tela de login seja esse o caso. Ao entrar na página, é feita uma requisição GET para o endpoint "/users/profile", trazendo informações de nome, número de agendamentos já realizados, número da CNH, e-mail e data de criação da conta.

No canto superior esquerdo se encontra um botão que, ao ser clicado, leva para a tela de edição, do perfil, que requisita novamente da API os dados do usuário, exibindo um formulário preenchido com nome e número da CNH. Ao realizar alguma alteração nos valores dos campos, o botão "Salvar alterações" é liberado, e, quando clicado, realiza uma validação, semelhante a outros formulários da aplicação, e envia os dados por meio de uma requisição PATCH para o endpoint "users/profile" no corpo da requisição. Caso esse processo seja bem sucedido, o usuário é direcionado à página de feedback positivo e o botão de "Ok", quando clicado, retorna para a página de edição com os dados atualizados.

Figura 5 - Diagrama de Fluxo do Usuário: Visualizar Agendamentos e Dados do Usuário.



Fonte: Autoria própria.

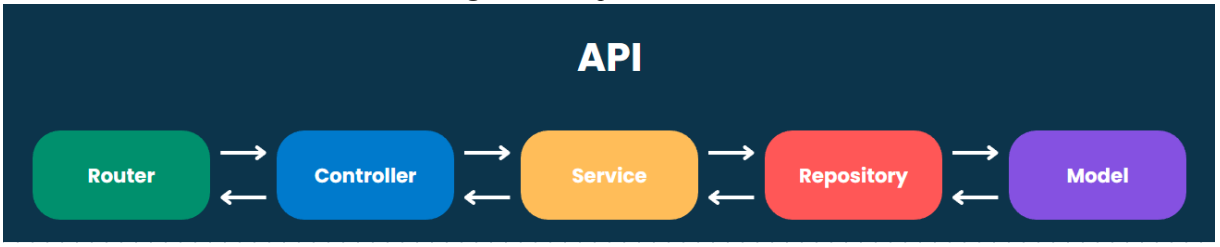
Foi feito o deploy da aplicação no Netlify, um serviço de hospedagem com foco em tecnologias front end, que fornece um plano gratuito que permite integrar com repositórios do github, lida com variáveis de ambiente, disponibiliza a aplicação em HTTPS, entre outros recursos, com mínimas limitações de customização, de uma forma simples e direta. Somente foi necessário adicionar a variável de ambiente com o endereço do servidor que executa a API.

3.2.3 API

A API do sistema foi desenvolvida com Node.js e Typescript, utilizando Express.js, um framework bastante popular e poderoso para construção de APIs. Assim como o conjunto de componentes do sistema, no backend da aplicação também foi implementada a arquitetura em camadas. Podemos dividir a API do RentX nas seguintes camadas:

- *routers*, ou roteadores, responsáveis pelo roteamento das requisições que são recebidas por cada endpoint;
- *controllers*, ou controladores, que são atribuídos a rotas ou sub rotas de um roteador e são responsáveis por receber e processar as requisições, além de prover uma resposta para as mesmas;
- *services*, ou serviços, responsáveis pelas regras de negócio do sistema, sendo usados nos controladores;
- *repositories,* ou repositórios, que fazem o papel de interface da camada de persistência, abstraindo e delimitando os métodos de acesso a essa camada, utilizados pelos serviços;
- *models*, ou modelos, que são abstrações dos modelos de negócio, sendo associados a um repositório.

Figura 6 - Arquitetura da API



Fonte: Autoria própria

Assim como no front end da aplicação, Typescript em conjunto com a biblioteca Zod são uma combinação eficiente para explicitar tipos de variáveis simples e complexas. Seu principal uso foi definir os modelos de negócio, aplicando regras como: seguir certos padrões

(endereços de e-mail, número da CNH, UUIDs, entre outros), tamanhos de string, valores padrão, etc.

Os modelos de negócio da aplicação são sessão, usuário, categoria, carro e aluguel, sendo estes descritos na próxima subseção. Cada modelo possui uma classe utilitária que provê métodos para validar um objeto, total ou parcialmente, em relação à definição, e também sanitizar objetos, para que contenham apenas atributos contidos no modelo em questão. A lista de modelos de negócio e suas inter relações serão descritas na subseção seguinte.

O roteador de autenticação possui apenas um endpoint, sem restrições de acesso, com método POST para a rota "/sessions", sendo essa responsável pelo login do usuário. O controlador de autenticação, utilizado pela rota supracitada, realiza a tentativa de recuperar e-mail e senha do corpo da requisição, retornando um erro caso um deles não esteja presente. Ocorre então a validação desses dados, usando a classe utilitária, anteriormente citada, do modelo *User*. Em seguida, o serviço de autenticação confirma a existência do usuário a partir do e-mail fornecido e compara a senha que foi salva no banco de dados para esse usuário com o valor que veio no corpo da requisição. Após essa etapa de validação, ainda dentro do serviço, é gerado um token JWT, que expira em 1 dia, e é retornado um objeto contendo o token e informações do usuário, com exceção da senha.

O fluxo citado segue o padrão descrito no início desta subseção, ou seja, roteador, controlador, serviço e repositório. Para evitar repetição e simplificar a explicação, as próximas rotas farão uma abstração da passagem dessas etapas dentro do fluxo.

O roteador que engloba rotas do usuário possui 2 endpoints: "/users", que cria um usuário quando uma requisição POST é realizada, e a sub rota "users/profile", que retorna as informações do usuário quando o método utilizado for GET e atualiza o mesmo, parcialmente, caso seja um PATCH.

Importante pontuar que as rotas do *endpoint* de perfil exigem que o usuário esteja autenticado. Para verificar se o usuário já fez o login, tem-se um middleware que procura nos cabeçalhos da requisição, pelo token JWT de autenticação, anteriormente citado. É feita a validação do mesmo, garantindo a autenticidade da origem e que ainda não foi expirado, para então recuperar o id do usuário que fez a requisição. Similar à autenticação, o modelo *User* também possui um controlador, que define 3 métodos, sendo um para cada uma das situações listadas:

- Ao criar o usuário, deverão ser fornecidos nome, e-mail, senha, número da CNH e o valor do campo “repetir senha". Ocorre a validação desses dados em relação ao modelo do usuário, além da igualdade da senha em relação ao valor do campo “repetir senha”. Em seguida, verifica-se a existência do usuário, a partir do e-mail, com o método fornecido pelo repositório, com a finalidade de evitar duplicatas. Continuando, encripta-se a senha usando a função de hash do bcrypt, com *salt* de tamanho 8 e por fim um novo usuário é criado no banco com esses dados.
- Ao solicitar informações de perfil do usuário, o valor do id, recuperado na etapa de garantir a autenticação, é validado em relação ao modelo *User*, impedindo que o valor tenha formato diferente do esperado caso o conteúdo do token tenha sido manipulado. Com a ajuda do repositório do *User*, verifica-se a existência de um usuário atribuído a esse id, além de incluir o número de aluguéis realizados por ele, usando o repositório do *Rental*. Por fim, é retornado um objeto com todas essas informações, com exceção da senha do usuário.
- Semelhante ao processo acima, é recuperado o id do usuário da sessão, além do nome e/ou número da CNH, caso um ou ambos sejam alterados, garantido que ao menos um dos dois esteja presente no corpo da requisição. O id e o conjunto de informações a modificar passam pela validação em relação ao modelo de referência e atualiza-se o usuário de acordo, usando o método disponibilizado pelo repositório.

O roteador de categorias possui apenas um endpoint, “/categories”, que disponibiliza os métodos GET, para listar as categorias, e POST, para criar uma categoria. A rota de listagem é pública e a sua resposta deve ser sempre o resultado da chamada do método de retornar todas as categorias, do repositório associado ao modelo *Category*. Criar uma categoria requer que o usuário esteja autenticado e seja administrador, sendo verificado por um outro middleware que verifica a flag “isAdmin” do mesmo. Para realizar o processo, é necessário fornecer o nome, ícone, descrição e nome que deve ser usado na hora de exibir essa categoria, no corpo da requisição, ocorrendo a validação desses dados e criando a categoria, caso ela não exista.

O roteador de categorias é composto por 3 *endpoints*, cada um permitindo apenas 1 método. São esses:

- GET /cars/available: Responsável pela listagem de carros. Caso não seja passado nenhum *query param*, todos os carros com a flag available de valor verdadeiro serão retornados. Essa rota é pública e aceita cinco parâmetros opcionais, sendo esses id,

nome, categorias incluídas, faixa de preço e marca do conjunto de carros a serem listados. Quando um ou mais parâmetros são recebidos, ocorre a validação dos mesmos e, em caso de sucesso, serão usados como filtro para o método do repositório de carros.

- GET /cars/brands: Responsável pela listagem de marcas. Essa rota é pública e retorna todas as marcas de carros presentes na tabela de carros.
- POST /cars: Responsável por cadastrar novos carros. Exige autenticação e privilégios de administrador para ser utilizada e recebe todos os atributos do modelo *Car* através do corpo da requisição, com exceção do id, indicação de disponibilidade e data de criação, que são automáticos. Após validação desses dados, é criado o novo registro no banco.

Por fim, o roteador de aluguéis é composto por 3 *endpoints*, cada um permitindo apenas 1 método. São esses:

- POST /rentals: Responsável por criar um agendamento de aluguel. Exige que o usuário esteja autenticado e forneça o id do carro, data de início e data final esperada do aluguel no corpo da requisição. Após a validação desses dados, além do próprio id do usuário, são feitas várias verificações. A partir do id do carro, é garantido que o mesmo exista e esteja disponível. Também comparam-se as datas fornecidas, não permitindo períodos de aluguel de menos de 24 horas, já que uma diária seria o mínimo a se pagar, e períodos de aluguel inválidos, começando no passado. Por fim, é criado o registro na tabela de aluguéis e atualiza-se o status de disponibilidade do carro.
- GET /rentals/user: Responsável por listar os agendamentos, ativos ou não, do usuário autenticado.
- POST rentals/{rentalId}/end: Responsável por encerrar um aluguel. Exige que o usuário esteja autenticado e forneça o id do aluguel que deve ser encerrado como parâmetro de rota. Após validar os dados, verifica-se a existência do registro na tabela de aluguéis, bem como se o usuário autenticado está associado ao mesmo ou se este já foi finalizado. É calculado o valor total, levando em conta possíveis multas por atraso ou situações onde o carro é devolvido antes da data esperada, pagando no mínimo uma diária. Por fim, o registro tem seus dados atualizados e o status de disponibilidade do carro é alterado de acordo.

A hospedagem da API foi feita na AWS EC2, serviço que permite criar servidores virtuais em nuvem, a partir de uma imagem do Docker, ligado ao AWS API Gateway, que funciona como um interceptador de requisições, mapeando cada rota de uma API a uma rota desse serviço. Essa estratégia, além de trazer mais uma camada de segurança no acesso, exime a necessidade de criação de um certificado SSL para ter um servidor HTTPS, já que o serviço do API Gateway já provê um endereço HTTPS quando é utilizado.

A comunicação com o banco de dados é feita a partir do framework Prisma, que define os modelos, seus relacionamentos, além de configurações de conexão, em um arquivo *schema* com sintaxe e extensão próprias do Prisma. A API irá se conectar à instância postgres na AWS RDS e, na primeira conexão, aplicar um *database seed*, que é nada mais que popular o banco com valores pré definidos, iniciando o banco de dados com 3 carros e 3 categorias.

3.2.4 BANCO DE DADOS

Para o banco de dados, foi escolhido o PostgreSQL, por ser amplamente utilizado e bastante seguro, implantado em uma instância do AWS RDS. Na Figura 7, temos o diagrama UML da aplicação, com seus 4 modelos e relacionamentos entre eles. Por conta da utilização de tokens JWT para persistir a sessão, optou-se por não armazená-las em uma tabela, já que sua assinatura é sempre verificada no momento da autenticação, garantindo a origem e autenticidade.

O modelo *User* possui id, nome, senha, e-mail, número da CNH, *flag* booleana que identifica se o usuário é administrador, com valor falso por padrão, e campos automáticos que identificam quando o registro foi criado e atualizado pela última vez. O id, assim como em todos os outros modelos, é um UUID aleatório auto gerado quando o registro é criado. O e-mail foi definido como *unique*, garantindo que não haja mais de um usuário associado ao mesmo e-mail.

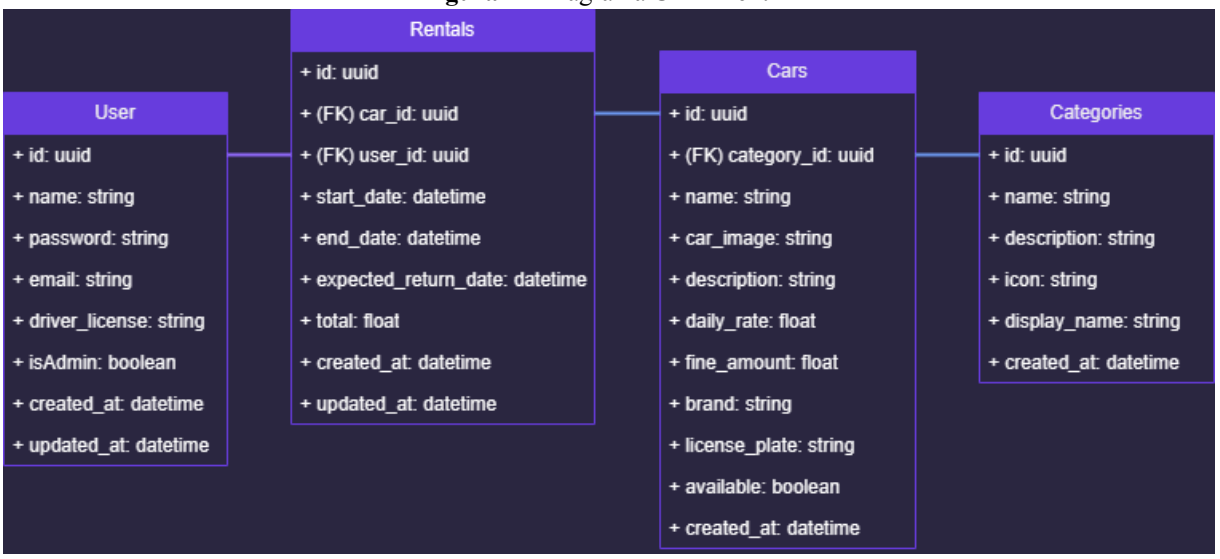
O modelo *Rentals*, representando os aluguéis, possui: id, datas de devolução esperada, início, finalização do aluguel, valor total pago, incluindo possíveis multas por atraso. Além dos mesmos campos automáticos, temos relacionamentos *Many-to-One* tanto para *User* quanto para *Cars*, através dos IDs do usuário e carro associados ao aluguel. Como “end_date” e “total” são preenchidos após a finalização de um aluguel, seus valores podem ser nulos.

O modelo *Cars* possui id, *flag* booleana que indica se o carro está disponível para aluguel, nome, foto do carro, descrição, valor da diária, valor da multa, marca, placa, que deve

ser única em toda a tabela, campo automático representando quando o registro foi criado e, com relação *Many-to-One* com a tabela de categorias, id da categoria ao qual o carro pertence.

Por fim, o modelo *Categories* possui id, nome, que deve ser único em toda a tabela, descrição, ícone, nome representativo e campo automático registrando quando foi criada.

Figura 7 - Diagrama UML RentX



Fonte: Autoria própria

3.2.5 AMBIENTE E CONFIGURAÇÃO DOS TESTES

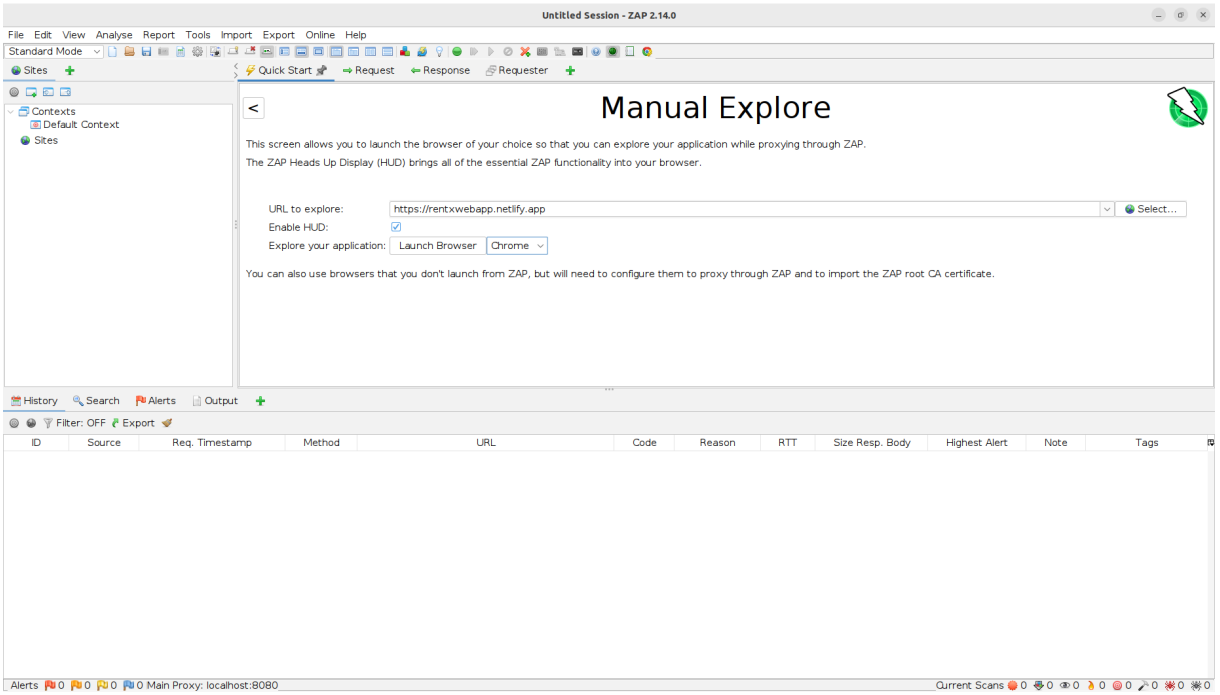
Para a realização dos testes de segurança, foi utilizado o programa VirtualBox, que permite criar máquinas virtuais, com uma instância do Ubuntu 22.04. Nele, com a ferramenta Zed Attack Proxy (ZAP), foi realizado o teste de penetração no endereço https://rentxwebapp.netlify.app, onde estava hospedado o frontend da aplicação.

Segundo a documentação do ZAP e dado o contexto da aplicação, foi escolhida a Exploração Manual, que consiste em, através de um *browser* controlado pela ferramenta, interagir e explorar o sistema manualmente, realizando a etapa de reconhecimento com mais precisão, já que por vezes a automação completa do processo geralmente sofre de algumas limitações, sendo essas: acessar todos os caminhos, já que rotas autenticadas não podem ser testadas, a não ser que seja configurado antes da análise, e definir a sequência da exploração em escaneamentos passivos ou tipos de ataque em um ataque automatizado.

Após interagir com toda a aplicação e autenticar o usuário, é realizado um *active scan* a partir do contexto criado, que seria o processo de procurar vulnerabilidades na aplicação. Ao fim do escaneamento, é possível encontrar os alertas, separados por nível de risco, com informações como nível de confiança, origem, descrição e possíveis soluções. Os resultados

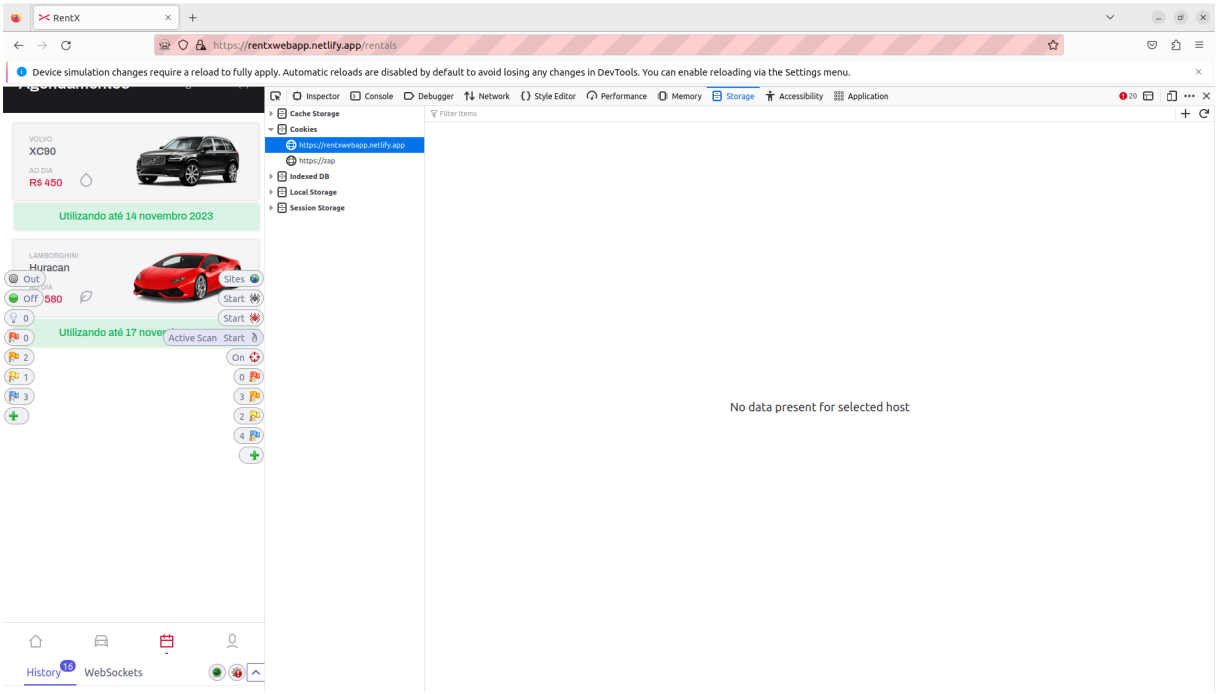
do teste podem ser exportados como relatório em formato HTML. As Figuras 8 e 9 mostram a tela inicial da exploração manual e sua visão dentro do *browser* controlado.

Figura 8 - Tela do Explorador Manual da Ferramenta ZAP



Fonte: Ferramenta ZAP

Figura 9 - Tela do Navegador Controlado da Exploração Manual



Fonte: Ferramenta ZAP

4. RESULTADOS

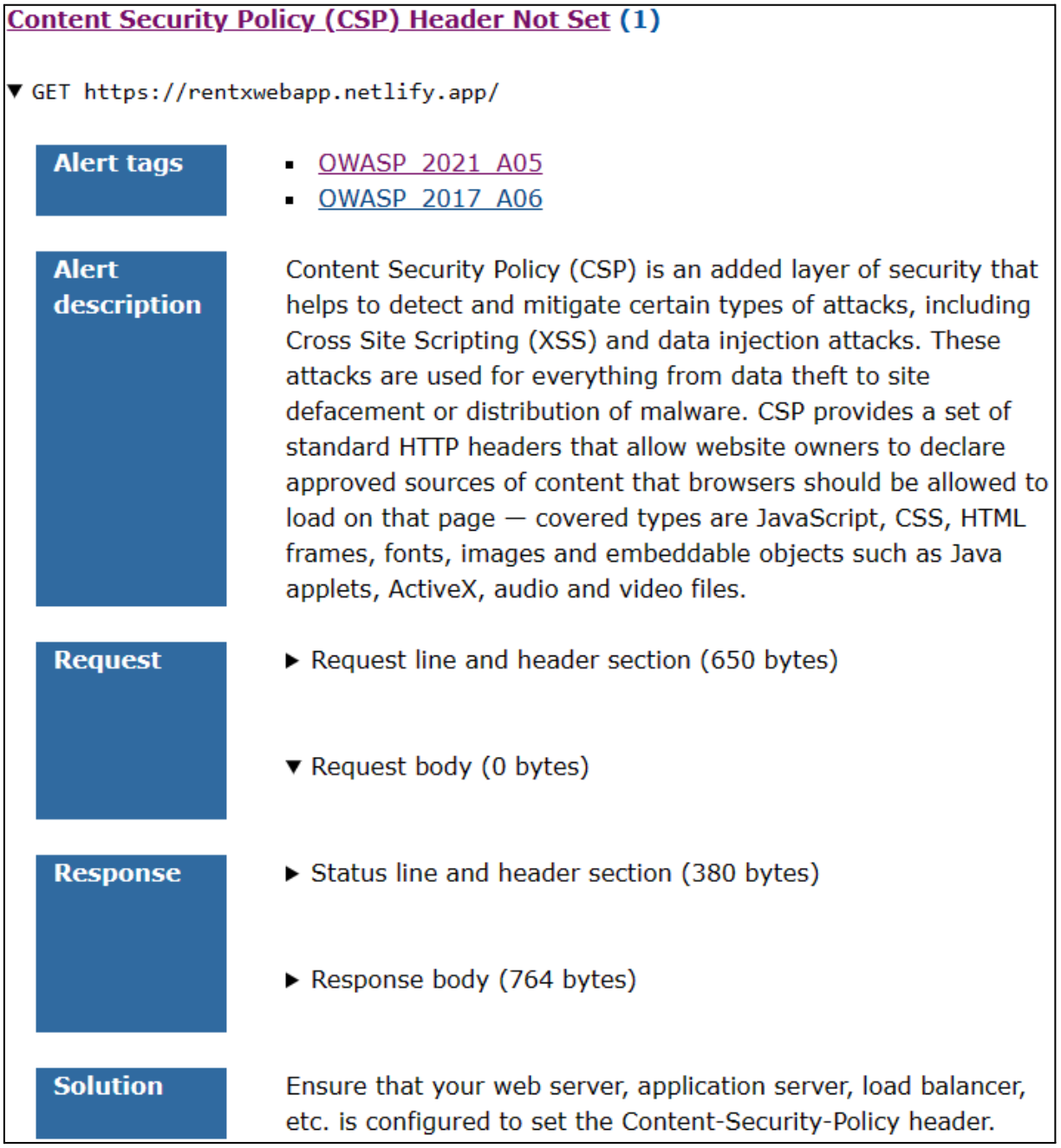
Esta seção visa analisar o relatório gerado pela ferramenta ZAP em cima do sistema Rentx, identificando as principais vulnerabilidades encontradas durante o teste, e interpretando os resultados. O relatório exibiu quatro alertas de risco médio e dois de risco baixo.

Na Figura 10, temos um alerta de risco médio, avisando que o CSP não está presente nos cabeçalhos da requisição, estando de acordo com o item A06 do OWASP Top 10 de 2021. Como pode ser visto na descrição, Content Security Policy, ou CSP, é uma camada adicional de segurança que facilita a detecção e mitigação de certos tipos de ataques, como Cross Site Scripting e ataques de injeção, que tem o intuito de roubar dados do usuário ou até desconfigurar o site para distribuição de malware.

Esse protocolo permite configurar a origem permitida do conteúdo que pode ser exibido e/ou executado em uma página, como por exemplo: scripts, mídias em geral, imagens, entre outros. Segundo o MDN, a partir da versão 23 do Firefox, lançada em 2013, Content-Security-Policy se tornou o novo padrão, substituindo o X-Content-Security-Policy. Para realizar essa configuração, pode-se configurar o servidor web para retornar esse cabeçalho, ou adicionar a tag “<meta>” no html, como por exemplo:

```
<meta http-equiv="Content-Security-Policy" content="default-src 'self'; img-src https://*; child-src 'none';">
```

Figura 10 - Relatório ZAP: CSP Header Not Set

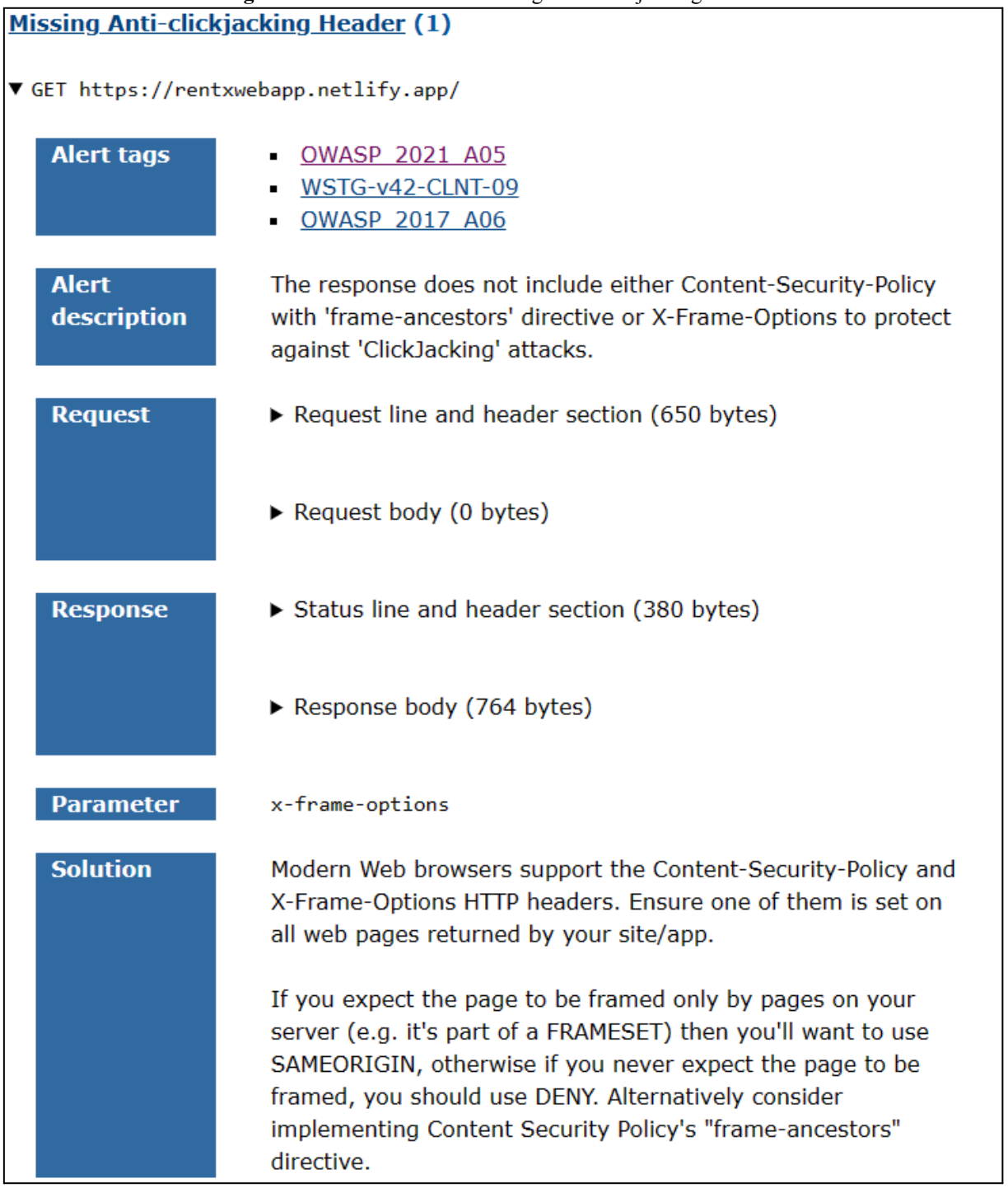


Fonte: Relatório gerado automaticamente pelo ZAP

Na figura 11, temos um alerta de risco médio, avisando que o cabeçalho Anti-clickjacking não está presente, estando de acordo com o item A05 do OWASP Top 10 de 2021. Como pode ser visto na descrição, a resposta da requisição não inclui a diretiva "frame-ancestors" no CSP, o que faz sentido, já que o protocolo não está presente, ou declaração do X-Frame-Options, que tem o mesmo papel e tornou-se obsoleto para navegadores com suporte ao CSP. Essa propriedade define a origem de páginas ou conteúdos que podem imbuir a página. Caso não seja definida, a aplicação fica vulnerável ao *clickjacking*, que nada mais é do que enganar o usuário a clicar em um elemento de uma

página que está invisível ou mascarado como outro. Por exemplo, ao usar uma tag iframe, é possível renderizar o site inteiro do RentX sobrepondo um site malicioso, sendo o conteúdo que está sendo renderizado completamente visual e o verdadeiro site malicioso tem seus elementos que possuem interação posicionados por baixo, fazendo o usuário clicar em algo que não esperava.

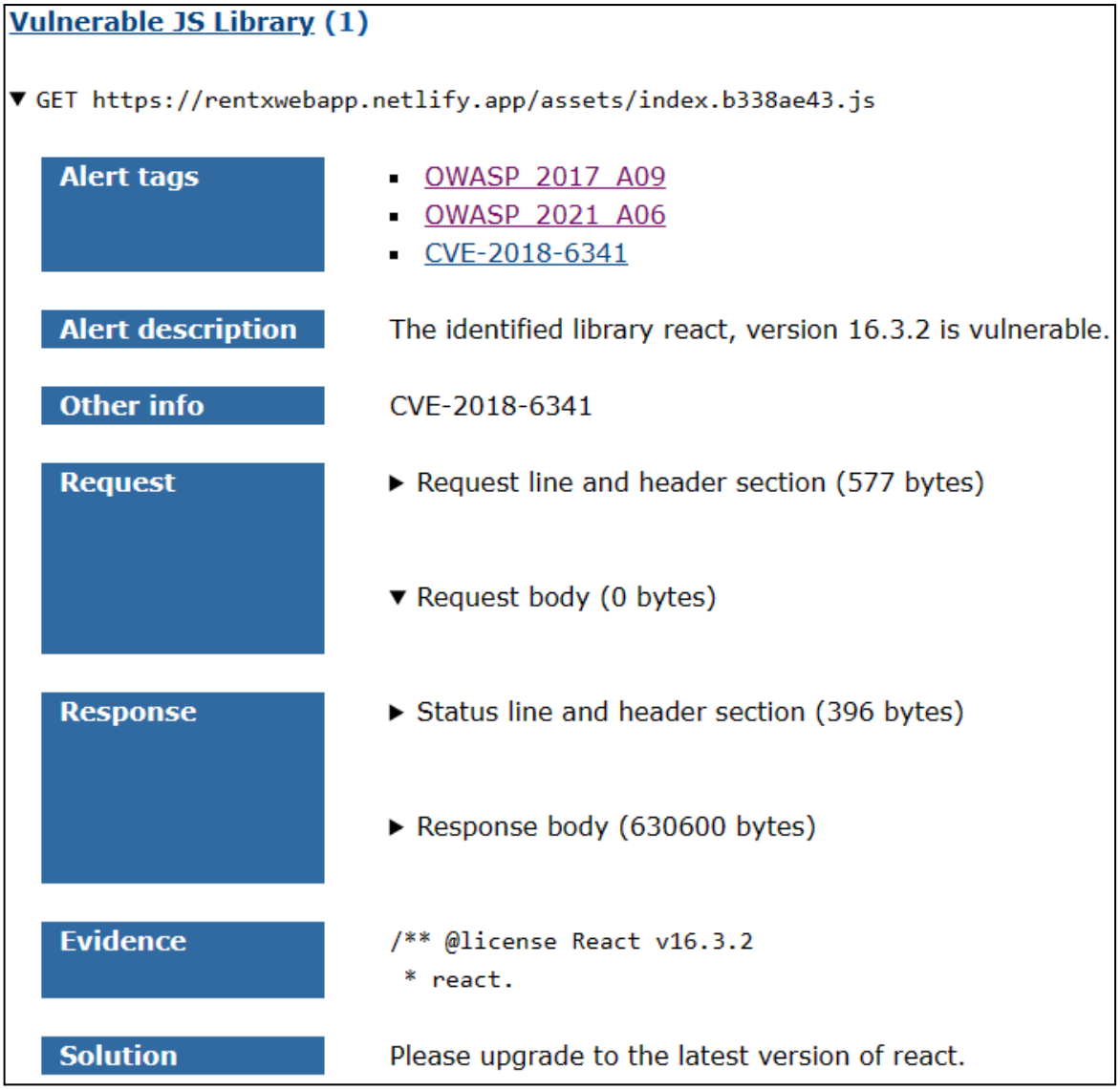
Figura 11 - Relatório ZAP: Missing Anti-clickjacking Header



Fonte: Relatório gerado automaticamente pelo ZAP

Na figura 12, temos um alerta de risco médio, avisando sobre Vulnerable JS Library, estando de acordo com o item A06 do OWASP Top 10 de 2021. Foi identificado o comentário "@license React v16.3.2", assumindo que a aplicação usa essa versão do React em componentes ou dependências. Analisando a pasta "node_modules", responsável por guardar os arquivos de todas as bibliotecas do projeto, pode-se de fato encontrar alguns arquivos javascript contendo esse comentário, e, considerando que o React 16 perdeu suporte ativo em Outubro de 2020, mas ainda recebe suporte de segurança, não há preocupação imediata, apenas algo a se considerar com o passar do tempo, principalmente se a versão perder suporte total.

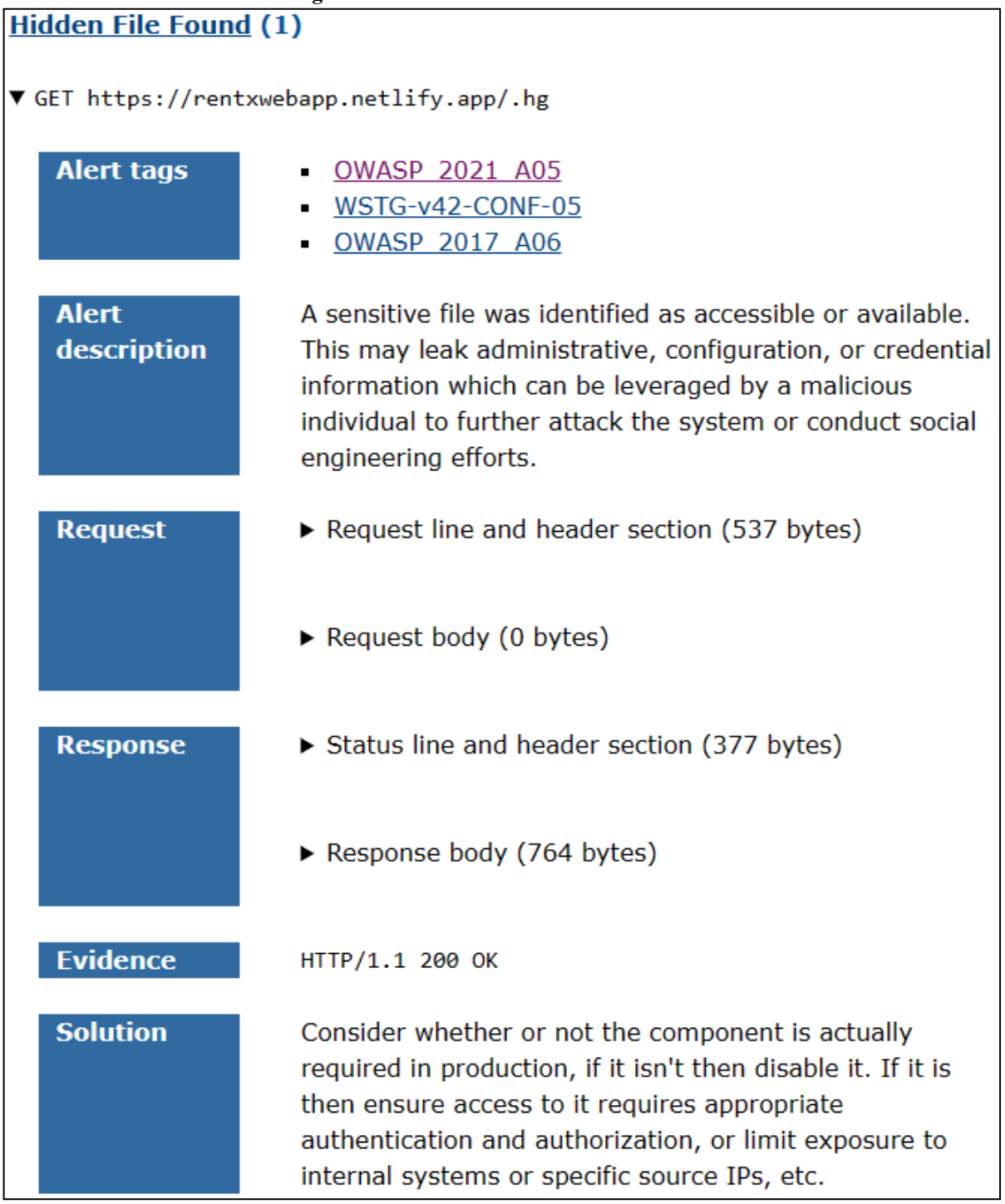
Figura 12 - Relatório ZAP: Vulnerable JS Library



Fonte: Relatório gerado automaticamente pelo ZAP

Na figura 13, temos um alerta de risco médio, avisando sobre Hidden File Found, estando de acordo com o item A05 do OWASP Top 10 de 2021. Foi identificado o arquivo ".hg", que após investigação, descobriu-se ser de alguma biblioteca base, de difícil acesso, algo como um arquivo de cabeçalho. Dado a natureza deste, não é considerado nada muito alarmante.

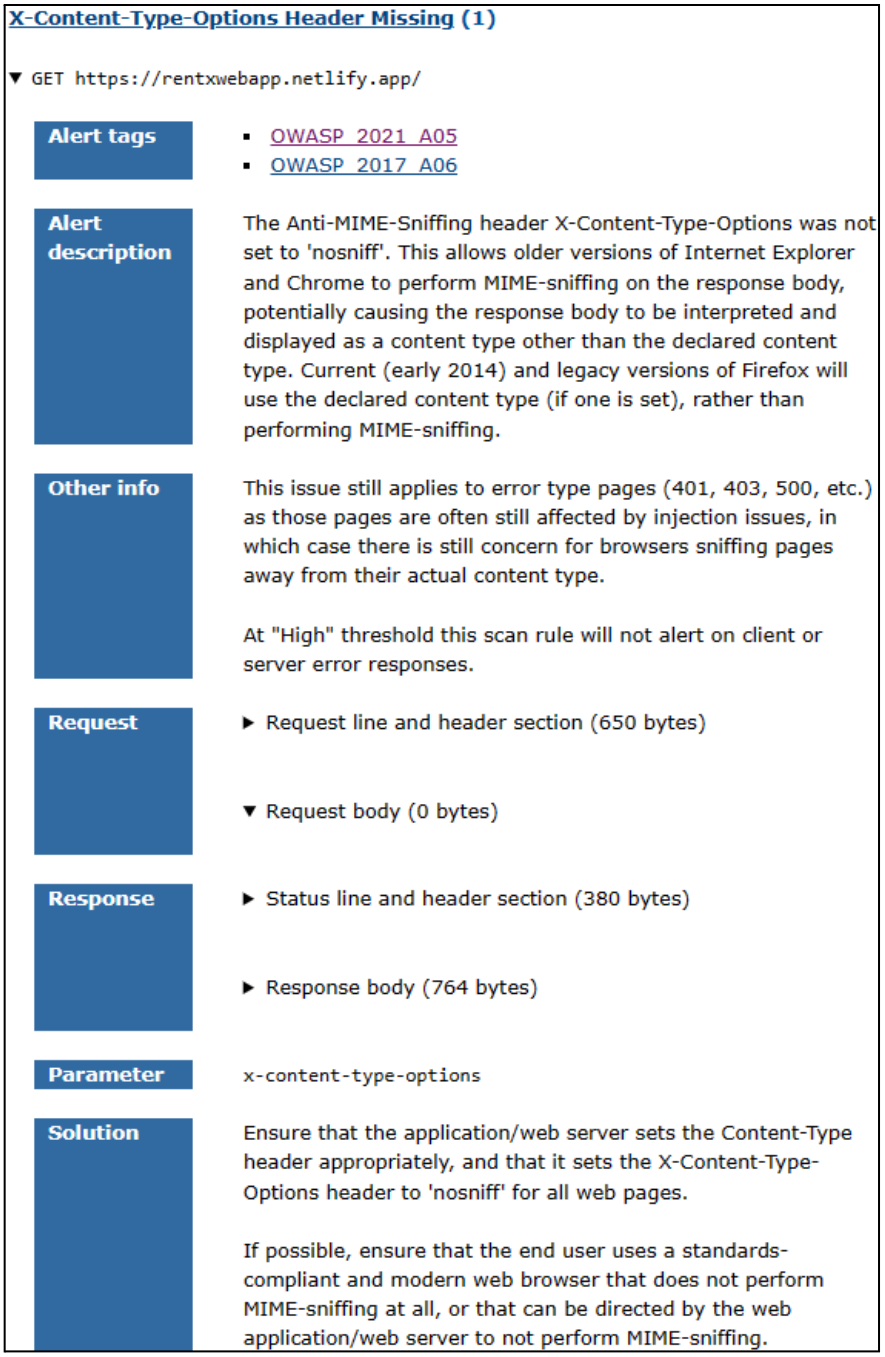
Figura 13 - Relatório ZAP: Hidden File Found



Fonte: Relatório gerado automaticamente pelo ZAP

Na figura 14, temos o primeiro alerta de baixo risco, de ausência do X-Content-Type-Options nos cabeçalhos, estando de acordo com o item A05 do OWASP Top 10 de 2021. Segundo o MDN, este cabeçalho é responsável por indicar que os *MIME types* declarados no cabeçalho Content-Type devem ser respeitados e não modificados. Na ausência do mesmo, o sistema se encontra vulnerável ao MIME-Sniffing, onde o browser tenta inferir o tipo de conteúdo, sendo preocupante em casos onde este seja executável.

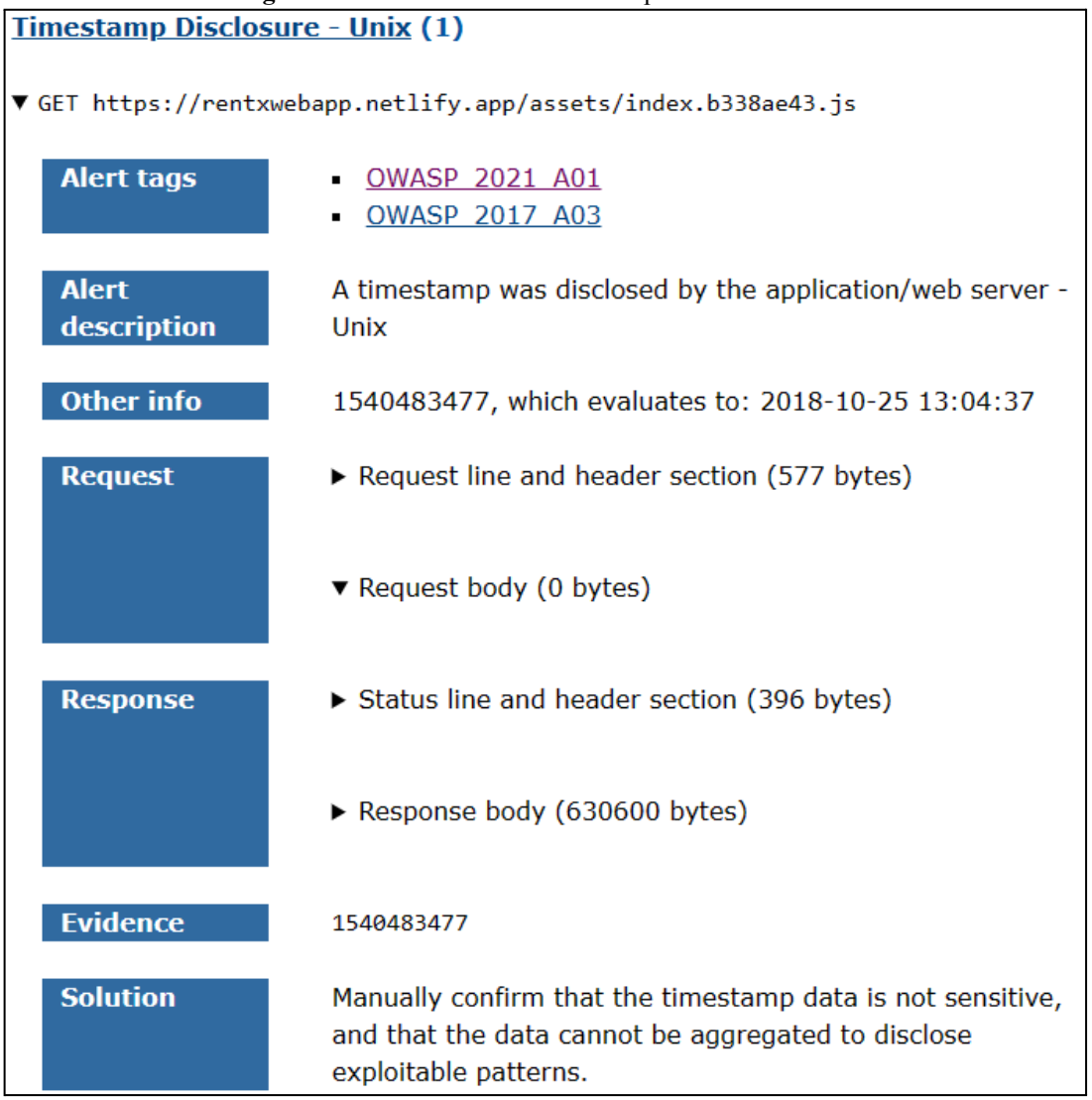
Figura 14 - Relatório ZAP: X-Content-Type-Options Header Missing



Fonte: Relatório gerado automaticamente pelo ZAP

Por fim, na figura 15 temos o alerta de baixo risco, da presença de Timestamp Disclosure - Unix, estando de acordo com o item A01 do OWASP Top 10 de 2021. A descrição aponta a presença de um formato *Timestamp* pelo lado do servidor, que para a ferramenta poderia ser algo não intencional. Porém, é garantido que, pela construção do sistema, nenhum *timestamp* possui conteúdo sensível, sendo assim pouco relevante.

Figura 15 - Relatório ZAP: Timestamp Disclosure - Unix



Fonte: Relatório gerado automaticamente pelo ZAP

4.1 CONCLUSÃO DOS RESULTADOS

Conforme apresentado nos resultados, a aplicação possui poucas vulnerabilidades de baixo risco, e um número pequeno de riscos médios, sem nenhum risco alto ou crítico. Como a maioria dos pontos abordam configurações a serem feitas, solucionar os problemas apontados parecem ser de fácil correção.

É possível visualizar que alguns alertas foram mais severos do que o necessário. Isso se deve à configuração adotada para realizar os testes, da mesma forma que poderia influenciar o nível de risco de alguns deles ou introduzir alertas novos.

5. CONCLUSÃO

O propósito desse trabalho é demonstrar o processo de análise de segurança de um sistema web, através da visão de um contexto onde a aplicação é conhecida, construída com ferramentas modernas e tem estrutura de sistema próximo ao que encontramos em aplicações profissionais.

Ao longo do documento, foi explorada a arquitetura e construção do RentX, abordando configurações e procedimentos (autenticação, validação, entre outros). Em seguida, introduziu-se a ideia de analisar a segurança básica do programa. Desse modo, sendo necessário realizar testes de penetração, a fim de descobrir as falhas do mesmo e formas de mitigá-las.

A partir do relatório gerado pela ferramenta conseguiu-se ter um levantamento dos riscos ao qual o sistema estava exposto, também fornecendo informações e referências úteis para entender e corrigir os problemas apresentados. Portanto, pode-se afirmar que os objetivos propostos pelo trabalho foram contemplados, sendo possível usar-se desse exemplo para introduzir a segurança em ambientes profissionais que carecem da conscientização da mesma.

5.1 TRABALHOS FUTUROS

Analisando os resultados do teste de penetração e também considerando todo o referencial teórico associado, podemos identificar que uma análise mais avançada, com configurações mais específicas, pode trazer resultados mais interessantes. Como dito anteriormente, a ferramenta ZAP traz opções vastas de configuração, inclusive para incluir scripts feitos pelo próprio testador, aumentando a especificidade e talvez até realizando testes que não conseguem ser realizados inteiramente por *scanners* automáticos. Com mais tempo e estudo, pode ser realizada uma análise mais profunda e profissional, que simule mais fielmente um ataque externo de um usuário mal intencionado experiente.

A ferramenta abordada neste trabalho não é a única ferramenta de *pentest*, existindo várias outras no mundo de softwares de segurança. Poderia ser feita uma comparação entre ferramentas, ou até com técnicas mais manuais e analisar os resultados para medir sua eficácia.

6. REFERÊNCIAS

DOCKER DOCS. Docker Overview. In: Guides. **Docker**, [s. I.], 2023. Disponível em: <https://docs.docker.com/get-started/overview/> . Acesso em: 09. nov. 2023.

ENGEBRETSON, P. **The basics of hacking and penetration testing:** Ethical hacking and penetration testing made easy. 2. ed. Boston: Syngress, 2013.

BATUBARA, T. P.; EFENDI, S.; NABABAN, E. B. Analysis Performance BCRYPT Algorithm to Improve Password Security from Brute Force. v. 1811, n. 1. **Journal of Physics: Conference Series**, Indonesia. 2021.

EISENTRAUT. et al. About PostgreSQL. **The PostgreSQL Global Development Group**. [s. I.], 2023. Disponível em: <https://www.postgresql.org/about/>. Acesso em: 17. jun. 2023.

EXPRESS.JS - Companies using Express.js. In: Software Frameworks - **Enlyft**, Estados Unidos, 2017. Disponível em: <https://enlyft.com/tech/products/express-js>. Acesso em: 17. jun. 2023.

EXPRESS. **Open JS Foundation**. [s. I.], 2017. Disponível em: <https://expressjs.com/>. Acesso em: 17. jun. 2023.

IMPERVA. What is Penetration Testing. In: Learning Center - **Imperva**, California, 2023. Disponível em: <https://www.imperva.com/learn/application-security/penetration-testing/>. Acesso em: 09/11/2023

JONES. M.; BRADLEY. J.; SAKIMURA. N. JSON Web Token. **Internet Engineering Task Force (IETF)**. [s. I.], 2015. Disponível em: <https://datatracker.ietf.org/doc/html/rfc7519>. Acesso em: 09. nov. 2023.

KHAN, R. A.; AGRAWAL. A. **Software Engineering:** a practitioner's approach. [s.l.]: Alpha Science International Ltd, 2014.

MDN CONTRIBUTORS. Introdução: Express/Node. In: Aprendendo desenvolvimento web - **MDN Web Docs**, [s. I.], 03 ago. 2023. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Learn/Server-side/Express_Nodejs/Introduction>. Acesso em: 31. mar. 2023.

NODE.JS. About Nodejs. **Open JS Foundation**. [s. I.], 2023. Disponível em: <https://nodejs.org/en/about>. Acesso em: 31. mar. 2023.

OLIVEIRA, B. F. **Vulnerabilidades em Sistemas Web**. Trabalho de conclusão de curso (Bacharelado em Sistemas de Informação) - Universidade Federal de Uberlândia, Uberlândia, 2019. Disponível em: <https://repositorio.ufu.br/bitstream/123456789/24092/1/VulnerabilidadesSistemasWeb.pdf>. Acesso em: 31. out. 2023.

OWASP. About the OWASP Foundation. **OWASP Foundation**, California, 2023. Disponível em: <https://owasp.org/about/>. Acesso em: 26. out. 2023.

OWASP. OWASP Application Security Verification Standard. **OWASP Foundation**, California, 2023. Disponível em: <https://owasp.org/www-project-application-security-verification-standard/>. Acesso em: 28. out. 2023.

PRISMA. What is Prisma?. In: Concepts - **Prisma Data**, Berlin, 2023. Disponível em: <https://www.prisma.io/docs/concepts/overview/what-is-prisma>. Acesso em: 02. jul. 2023.

REACT, **Meta Open Source**. [s.I.], 2023. Disponível em: <https://react.dev>. Acesso em: 21. mar. 2023.

ROVEDA. U. O QUE É REACT: PARA QUE SERVE, COMO FUNCIONA E CARACTERÍSTICAS. In: Blog da Kenzie Academia - **Tecnologia**. Manaus, 07 fev. 2023. Disponível em: <https://kenzie.com.br/blog/react/>. Acesso em 21. mar. 2023.

SULLIVAN, B.; LIU, V. **Web application security**: a beginner's guide. 1. ed. [s.l.]: New York Mcgraw Hill Professional, 2011.

VITE. Why Vite. **Vite Team**. [s. I.], 2019. Disponível em: <https://vitejs.dev/guide/why.html>. Acesso em: 31. out. 2023.

What is TypeScript?. **Microsoft**. Estados Unidos, 2023. Disponível em: <https://www.typescriptlang.org/>. Acesso em: 17. jun. 2023.

OWASP. A01:2021 – Broken Access Control. **OWASP Foundation**, California, 2021. Disponível em: <https://owasp.org/Top10/A01_2021-Broken_Access_Control/>. Acesso em: 13. nov. 2023

OWASP. A05:2021 – Security Misconfiguration. **OWASP Foundation**, California, 2021. Disponível em: <https://owasp.org/Top10/A05_2021-Security_Misconfiguration/>. Acesso em: 13. nov. 2023

OWASP. A06:2021 – Vulnerable and Outdated Components. **OWASP Foundation**, California, 2021. Disponível em: <https://owasp.org/Top10/A06_2021-Vulnerable_and_Outdated_Components/>. Acesso em: 13. nov. 2023

ROCKETSEAT. **Rocketseat**, [s. I.], 2023. Disponível em: <https://www.rocketseat.com.br/>. Acesso em: 13. nov. 2023.

ZAP. Getting Started. The Software Security Project - **Linux Foundation**, [s. I.]. Disponível em: <https://www.zaproxy.org/getting-started/>. Acesso em: 13. nov. 2023.