Nome: Gabriel Brito Bitencourt

Nome: Mateus Brito Bitencourt

Nome: Cecília Weselowsky da Cunha

Nome: Gabriel Marques Costa

Relatório Técnico: Projeto DockShield

Disciplina: Ferramentas de Desenvolvimento Web (FDW) **Semestre:** 2025/2

1. Introdução e Objetivo

Este documento tem como objetivo detalhar as decisões técnicas, arquiteturais e de implementação adotadas no desenvolvimento do projeto **DockShield** (Sistema de Análise de Vulnerabilidades em Containers). O projeto constitui o Produto Mínimo Viável (MVP) exigido para a avaliação N2, integrando conceitos de desenvolvimento web *full-stack*, segurança da informação e orquestração de infraestrutura.

O sistema foi projetado para resolver o problema de visualização e gerenciamento de relatórios de segurança (CVEs) em imagens Docker, oferecendo uma interface web segura e segregada para administração e consulta.

2. Arquitetura do Sistema

A solução foi construída sobre uma arquitetura de **microsserviços containerizados**, onde cada componente principal do sistema reside em seu próprio ambiente isolado, comunicando-se através de uma rede interna gerenciada pelo Docker. Esta abordagem foi escolhida para garantir a escalabilidade, a facilidade de manutenção e a simulação de um ambiente de produção real.

A arquitetura é composta por três serviços fundamentais:

1. **Serviço de Autenticação (Frontend/Auth):** Desenvolvido em Node.js.
2. **Serviço de Aplicação (Backend/Core):** Desenvolvido em Python/Flask.
3. **Serviço de Persistência (Database):** Banco de dados NoSQL MongoDB.

A comunicação entre o usuário final e os serviços ocorre via protocolo HTTP, com redirecionamentos controlados e compartilhamento de credenciais via *Cookies* seguros.

3. Detalhamento das Tecnologias Implementadas

Conforme requisitado na especificação do projeto, abaixo detalhamos a implementação dos conceitos abordados no curso.

3.1. HTML e CSS (Interface do Usuário)

A camada de apresentação foi desenvolvida utilizando HTML5 semântico para estruturação do conteúdo e CSS3 para estilização.

- **Design Responsivo:** A interface adapta-se a diferentes tamanhos de tela através do uso de *media queries* e frameworks de estilização (Bootstrap), garantindo usabilidade em dispositivos móveis e desktops.
- **Renderização:** O projeto utiliza uma abordagem híbrida. No serviço Node.js, arquivos estáticos são servidos diretamente. No serviço Flask, utiliza-se o motor de templates **Jinja2** para renderização dinâmica de dados (Server-Side Rendering - SSR), permitindo a injeção de dados do banco diretamente no HTML antes do envio ao cliente.

3.2. JavaScript (Lógica do Cliente e Servidor)

O JavaScript desempenha um papel duplo neste projeto:

1. **Client-Side:** Scripts auxiliares para manipulação do DOM e validação básica de formulários no navegador.
2. **Server-Side (Node.js):** O núcleo do serviço de autenticação foi escrito inteiramente em JavaScript, utilizando o ambiente de execução Node.js. Isso permitiu o uso de I/O não bloqueante, ideal para lidar com múltiplas requisições de login simultâneas.

3.3. Frontend e Serviço de Autenticação (Node.js)

O componente denominado "Frontend" atua, na verdade, como um **Gateway de Autenticação**.

- **Framework:** Foi utilizado o **Express.js** pela sua robustez e minimalismo na criação de rotas e middlewares.
- **Gerenciamento de Identidade:** O sistema permite o cadastro de novos usuários, login, alteração de senha e exclusão de conta.
- **Segurança da Senha:** As senhas nunca são armazenadas em texto plano. Utilizou-se a biblioteca **bcrypt** para a geração de *hashes* com *salt*, garantindo que, mesmo em caso de vazamento do banco de dados, as credenciais originais permaneçam protegidas.
- **Mecanismo de Sessão:** A autenticação é *stateless* (sem estado no servidor), baseada em **JWT (JSON Web Tokens)**. Ao realizar o login com sucesso, o servidor assina um token contendo o ID do usuário e o envia ao navegador dentro de um *Cookie* com a flag HttpOnly. Isso impede que scripts maliciosos no cliente (XSS) acessem o token.

3.4. Backend e API (Python/Flask)

O núcleo de negócios da aplicação foi desenvolvido em Python utilizando o framework **Flask**.

- **Integração com Apache:** Para atender aos requisitos de um ambiente de produção robusto, a aplicação Flask não roda isoladamente. Ela é servida através do servidor web **Apache HTTP Server**, utilizando o módulo **mod_wsgi**. Isso simula um cenário real de *deploy* corporativo, onde o servidor de aplicação é gerenciado por um servidor web de alta performance.
- **Middleware de Autorização:** Foi implementado um decorador personalizado (@login_required) em Python. Este componente intercepta todas as requisições, extrai

o *Cookie* de autenticação gerado pelo serviço Node.js e valida a assinatura digital do JWT usando uma chave secreta compartilhada.

- **Processamento de Dados:** O Backend é responsável por ler os relatórios de vulnerabilidade (armazenados em formato JSON complexo e Markdown), processá-los e convertê-los para HTML visualizável utilizando a biblioteca markdown.

3.5. Persistência de Dados (MongoDB)

Optou-se por um banco de dados NoSQL (**MongoDB**) devido à natureza não estruturada e hierárquica dos relatórios de CVE (*Common Vulnerabilities and Exposures*).

- **Estrutura:** O banco foi dividido logicamente em duas bases:
    1. login: Armazena a coleção de credenciais de usuários.
    2. DockShield: Armazena as coleções referentes a cada imagem Docker analisada.
- **Conectividade:** Ambos os microsserviços (Node.js e Flask) conectam-se à mesma instância do banco, mas operam em *databases* ou coleções distintas, mantendo a separação de responsabilidades.

4. Infraestrutura e Containerização (Docker)

O projeto utiliza o **Docker** para garantir a portabilidade e a reprodutibilidade do ambiente de desenvolvimento e produção.

4.1. Docker Compose

A orquestração dos serviços é definida no arquivo docker-compose.yml, que estabelece:

- **Serviços:** Definição dos containers frontend, backend e db.
- **Rede (Networking):** Criação de uma *bridge network* interna (fatec-network), permitindo que os containers se comuniquem utilizando nomes de host (DNS interno) como db ou backend, isolando o tráfego do ambiente externo.
- **Volumes:** Implementação de volumes persistentes (mongo_data) para o container do MongoDB. Isso assegura que os dados dos usuários e relatórios não sejam perdidos quando os containers são reiniciados ou destruídos.

4.2. Variáveis de Ambiente

Para seguir as boas práticas de segurança (12-Factor App), todas as configurações sensíveis (strings de conexão com o banco, chaves secretas JWT e URLs de redirecionamento) foram abstraídas em arquivos de configuração (.env e web_config.ini). Isso permite que o sistema seja implantado em diferentes ambientes (IPs diferentes) sem necessidade de alteração no código-fonte.

5. Conclusão

O MVP entregue demonstra uma aplicação web funcional, segura e arquiteturalmente madura. A integração entre tecnologias distintas (JavaScript e Python) foi realizada com sucesso através de padrões de mercado (JWT/Cookies), e a infraestrutura Docker garante que o sistema possa ser executado em qualquer servidor Linux com o mínimo de configuração manual.