Product Requirements Document (PRD)

# 1. Title & Version

* **Product Name: Container Image Vulnerability Scanner**
* **Version: 1.0**
* **Date: 21/3/25**
* **Author: Mohamed Sugail**
* **Notion:** [**Link**](https://www.notion.so/Product-Requirement-Docx-1bd796400ea880af8e3ccfc143cae487?pvs=4) **(**Recommended**)**

# CONTENTS:

|  |
| --- |
| * [**Overview**](#_Overview:) * [**Objective & Goals**](#_Objectives_&_Goals:) * [**Problem**](#_Problem_Statement:) * [**Backgrounds**](#_Backgrounds:) * [**Target Audience**](#_Target_Audience:) * [**Use cases**](#_Use_Cases:) * [**Features**](#_Features:) * [**Technical Requirements**](#_Technical_Requirements:) * [**Low-Fidelity Wireframe**](#_Low-Fidelity_Wireframe_(UI):) * [**System Architecture Design**](#_System_Architecture_Overview:) * [**UI & UX Design**](#_UI_&_UX) * [**User Stories**](#_User_Stories:) * [**Release Criteria**](#_Release_Criteria:) * [**Timeline**](#_Timeline:) * [**Development Actions**](#_Development_Action:) * [**Success Certain**](#_Success_Certain:) * [**Conclusion**](#_Conclusion:) |

# **Overview:**

As part of the compliance assessment, this task requires designing a security product that scans container images for vulnerabilities. The product should help users manage and prioritize vulnerabilities in their container images, ensuring compliance with security standards. The deliverables include a Product Requirements Document (PRD) and low-fidelity wireframes, with an optional list of development action items.

## Purpose:

* Clearly state the objective of the document and the product.
* The outlines the requirements for a Container Image Vulnerability Scanner (CIVS). It describes thepurpose, features, functionality, and expected behavior of the product."

# Objectives & Goals:

## Objective:

* To create a Product Requirements Document (PRD) that outlines the features and technical needs of the security product.
* To design low-fidelity wireframes that illustrate the user interface of the product.
* Optionally, to list development action items for implementing the product.

## Goals:

* Develop a user-friendly tool that scans container images for vulnerabilities.
* Provide detailed vulnerability reports, categorizing them by severity.
* Enable quick remediation with actionable insights.
* Ensure seamless integration into existing DevOps workflows.
* Allow users to quickly identify container images that contain known vulnerabilities.
* Provide clear insights into the severity of each vulnerability.
* Facilitate the prioritization of remediation efforts.
* Reduce the risk of deploying vulnerable container images.

# Problem Statement:

Organizations depend on containerized applications, but these container images may have vulnerabilities. Security teams require an efficient way to scan these images, identify vulnerabilities, and take corrective actions.

# Backgrounds:

In modern application development, container images are commonly used to package and deploy applications. These images often include third-party libraries and dependencies that may have known vulnerabilities. This product seeks to tackle the challenge of identifying and managing vulnerabilities in container images, with the goal of enhancing overall security.

# Target Audience:

* **DevOps Engineers and Security Teams:**
* Need efficient tools for quick container image scans. Require prioritization of vulnerabilities by risk.
* Seek integration of scanning in CI/CD pipelines.
* **Developers:**
* Want to ensure container images are free from vulnerabilities before deployment.
* Prefer user-friendly tools for scanning and fixing issues.
* Appreciate guidelines on secure coding practices.
* **Compliance Officers:**
* Require security reports for audits and compliance.
* Need to document vulnerability management efforts. Seek assurance of meeting regulatory standards.
* **Security Engineers:**
* Need advanced threat detection for container applications.
* Require insights into security gaps in images. Aim to establish best practices for container security.

# Use Cases:

## Use Case 1: DevOps Engineer Integrating Security Scans in CI/CD

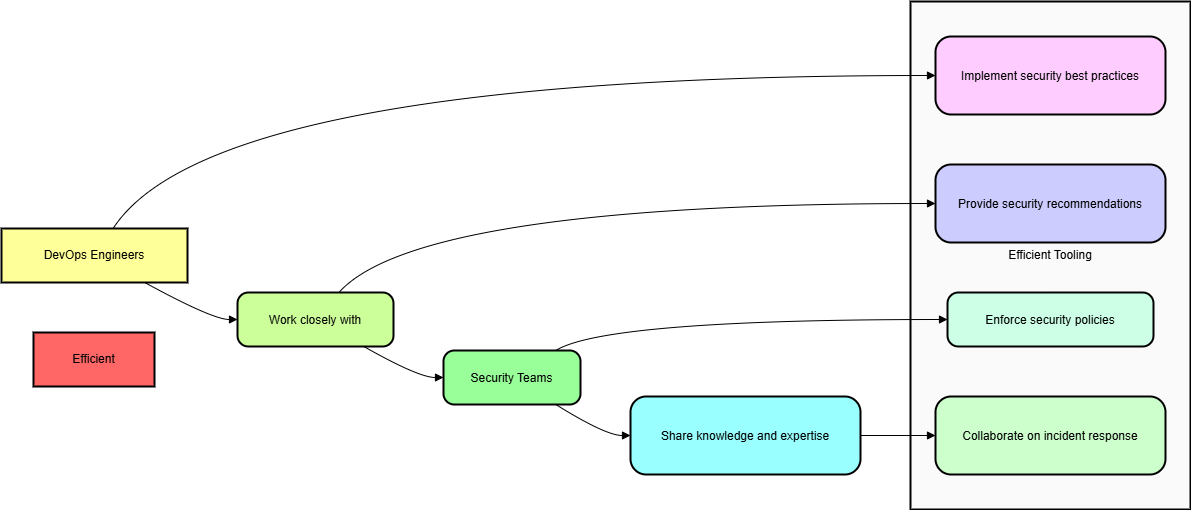
* A DevOps engineer configures the vulnerability scanner in their Jenkins pipeline.
* The scanner automatically scans container images before deployment.
* If vulnerabilities are detected, the pipeline fails, and the engineer is notified.
* The engineer reviews the scan results and applies the recommended fixes.

## Use Case 2: Security Analyst Monitoring Container Security

* A security analyst logs into the vulnerability scanner dashboard.
* They check the latest scan results and analyse high-risk vulnerabilities.
* They generate a security compliance report for an upcoming audit.
* The analyst shares remediation actions with the development team.

## Use Case 3: Developer Fixing Vulnerabilities in a Container Image

* A developer receives an alert about a critical vulnerability in an image.
* They use the scanner’s fix recommended.



# Features:

## Core Features

**✅ Container Image Scanning**:

* Scan container images using tools like Trivy or Clair.
* Fetch vulnerability data from CVE databases.
* Support scanning images from both public and private container registries.
* images stored in repositories (e.g., Docker Hub, AWS ECR, Google Container Registry) for known vulnerabilities.
* Automated scheduling of scans (e.g., daily, weekly)

**✅ Severity-Based Filtering:**

* Categorize vulnerabilities as Critical, High, Medium, or Low.
* Allow users to filter vulnerabilities based on severity.
* Highlight fixable vulnerabilities separately.
* Customizable severity thresholds

**✅ Search & Filtering:**

* Filter scan results by image name, CVE ID, package name, and severity.
* Implement a search feature for quick access to results.

**✅ Reporting & Visualization:**

* Display vulnerabilities in a tabular format with sorting options.
* Provide export options in JSON, CSV, and PDF formats.
* Generate detailed audit logs for compliance purposes.
* Dashboard with an overview of the number of images scanned, total vulnerabilities found, and severity distribution.
* Detailed reports for individual images with vulnerability information, including CVE ID, description, severity, and remediation steps.
* Filtering and sorting options based on severity, image, repository, and other criteria

**✅ Vulnerability Detection:**

* Support for multiple vulnerability types (e.g., OS packages, application dependencies.
* Regular updates to the vulnerability database to ensure accuracy.

**✅ Fix Suggestions:**

* Recommend updated versions for vulnerable dependencies.
* Provide direct links to patched versions of affected packages.

**✅ CI/CD Integration:**

* Integrate with Jenkins, GitHub Actions, and GitLab CI/CD to enable automated security scans.
* Provide a REST API to facilitate integration with third-party tools.
* API for integration with CI/CD tools (e.g., Jenkins, GitLab CI).
* Integrate with notification systems (e.g., Slack, email).

**✅ User Roles & Permissions:**

* Support role-based access control (RBAC) for different users.
* Allow administrators to manage permissions for team members.

Non-Functional Requirements:

* Scalability: The system must efficiently support scanning over 10,000 container images.
* Performance: Scans should be completed within 5 minutes per image.
* Security: Ensure protection against the exposure of private container registries.
* Reliability: Aim for 99.9% uptime to ensure uninterrupted scanning.

# **Technical Requirements:**

* Backend server (e.g., Node.js, Python) for handling API requests.
* Database for storing scan results and history (e.g., PostgreSQL, MongoDB).
* Integration with vulnerability scanning libraries (e.g., Trivy, Clair) and container registries via APIs.

## User Workflow:

1. Users select or upload container images for scanning.

2. The system scans the images using Trivy or Clair.

3. A detailed vulnerability report is generated.

4. Users can filter results, search for specific vulnerabilities, and download reports.

5. Fix suggestions are provided for affected dependencies.

6. Users integrate the scanner into CI/CD pipelines for continuous monitoring.

# **Low-Fidelity Wireframe (UI):**

┌─────────────────────────────────────────────────────────┐

│ 🔒 Container Image Security Scanner 👤 User ▼ │

├─────────────────────────────────────────────────────────┤

│ 🔍 [Search Images...] [Repository▼] [Severity▼] [Date▼] │

├─────────────────────────────────────────────────────────┤

│ Dashboard Overview │

│ ┌─────────┐ ┌─────────┐ ┌─────────┐ ┌─────────┐ │

│ │ Critical │ │ High │ │ Medium │ │ Low │ │

│ │ 42 │ │ 156 │ │ 298 │ │ 523 │ │

│ └─────────┘ └─────────┘ └─────────┘ └─────────┘ │

│ │

│ 📊 Vulnerability Distribution │

│ ╔════════════════════════════════╗ │

│ ║ 🔴 4% 🟠 15% 🟡 29% 🟢 52% ║ │

│ ╚════════════════════════════════╝ │

│ │

│ Image List [+ Scan New Image] │

│ ┌───────────────────────────────────────────────┐ │

│ │ □ Name Repository Critical High Med Low │ │

│ │ □ img1 DockerHub 2 5 12 25 │ │

│ │ □ img2 ECR 5 8 15 31 │ │

│ │ □ img3 ACR 0 2 8 12 │ │

│ └───────────────────────────────────────────────┘ │

└─────────────────────────────────────────────────────────┘

┌─────────────────────────────────────────────────────────┐

│ Image Details: img1 [← Back] │

├─────────────────────────────────────────────────────────┤

│ 📋 Metadata │

│ Repository: DockerHub │

│ Last Scan: 2025-03-20 09:15 UTC │

│ │

│ 🔍 Vulnerabilities │

│ ┌───────────────────────────────────────────────┐ │

│ │ 🔴 CVE-2025-1234 openssl Critical [Fix →] │ │

│ │ 🔴 CVE-2025-5678 libcurl Critical [Fix →] │ │

│ │ 🟠 CVE-2025-9012 nginx High [Fix →] │ │

│ └───────────────────────────────────────────────┘ │

└─────────────────────────────────────────────────────────┘

┌─────────────────────────────────────────────────────────┐

│ ⚙️ Integration Settings │

├─────────────────────────────────────────────────────────┤

│ CI/CD Pipeline Integration │

│ [✓] GitHub Actions [ ] Jenkins [ ] GitLab CI │

│ │

│ 🔔 Notification Settings │

│ [✓] Slack [ ] Email [ ] MS Teams │

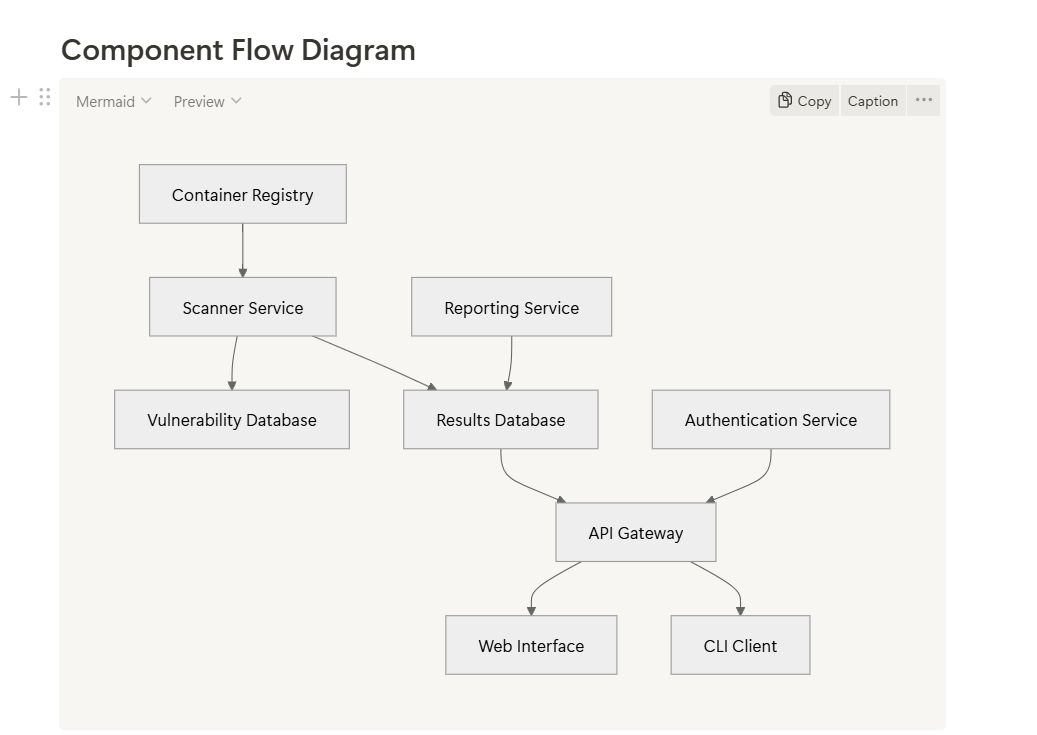
│ │

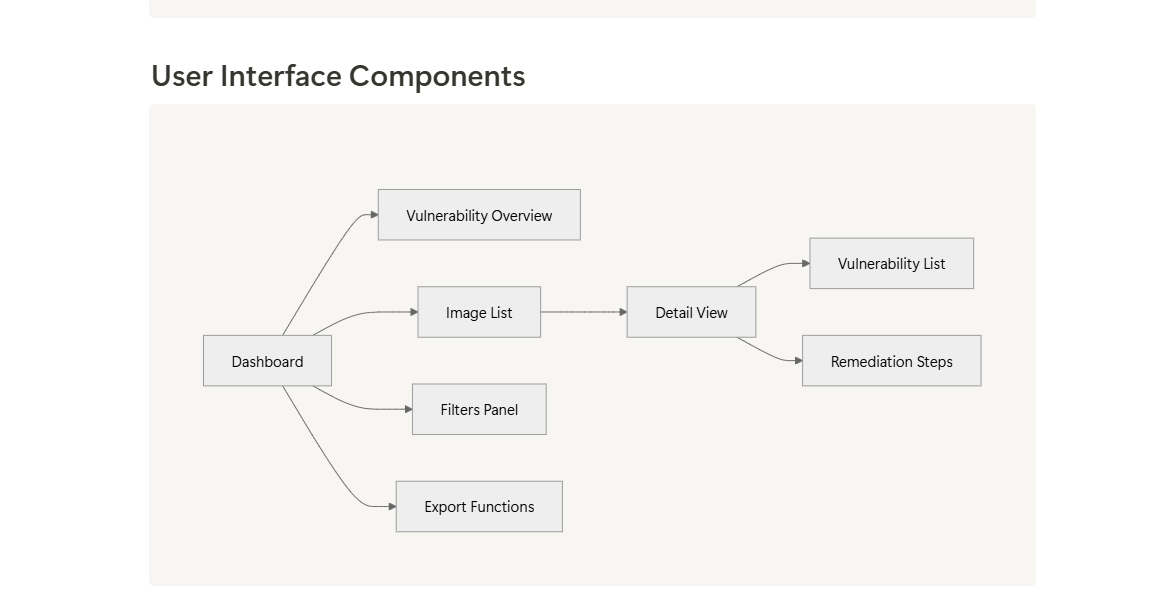
│ ⚠️ Alert Conditions │

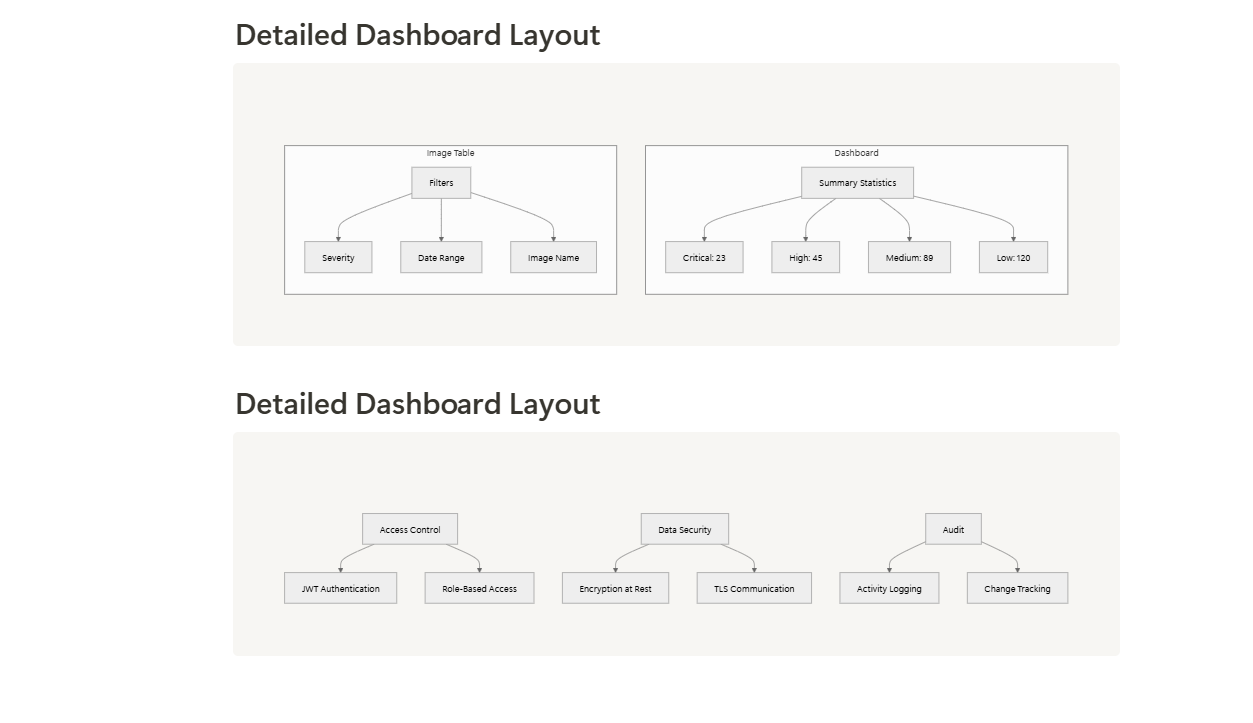
│ [✓] Critical [✓] High [ ] Medium [ ] Low │

└─────────────────────────────────────────────────────────┘

# **System Architecture Overview:**

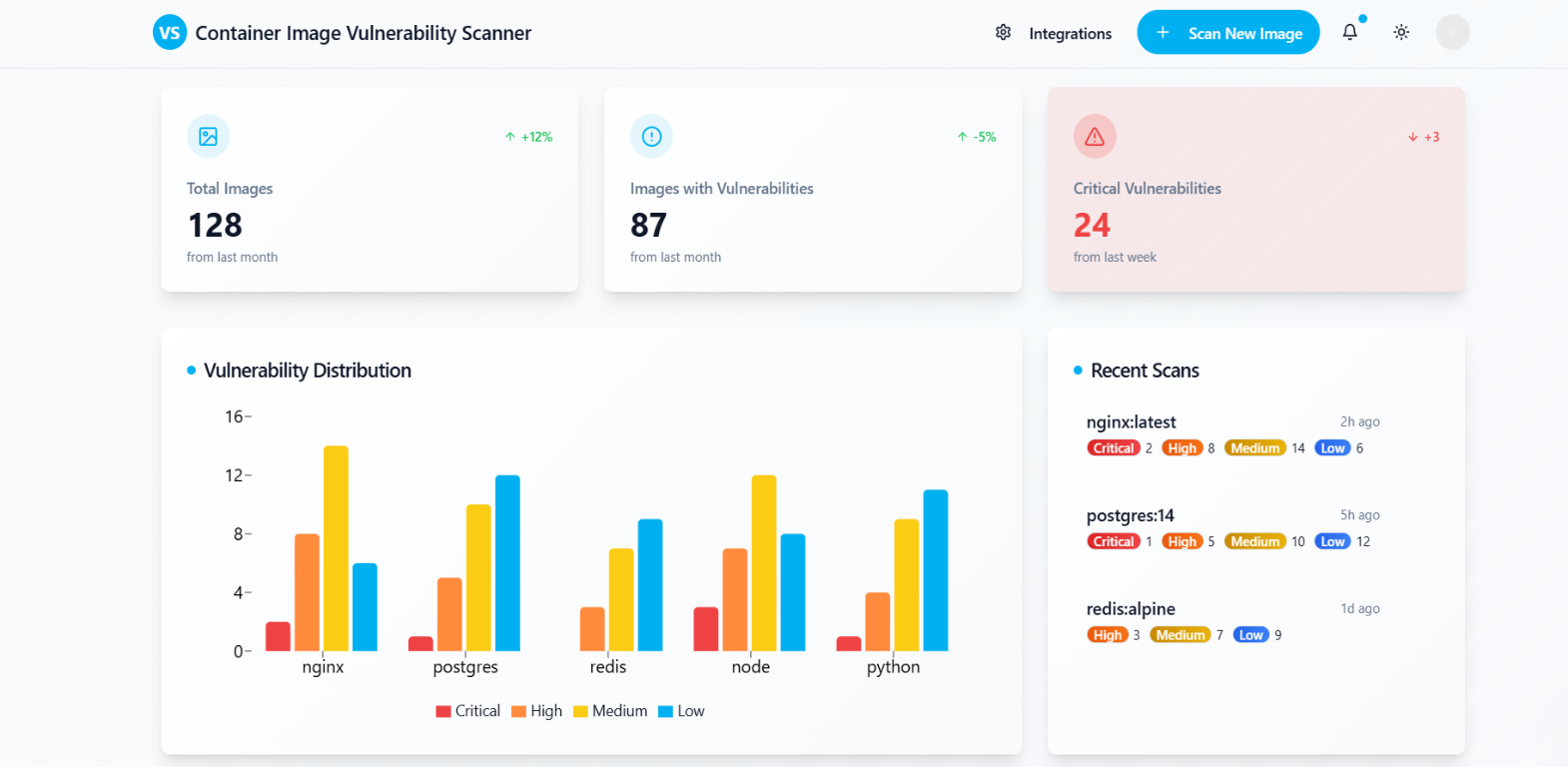




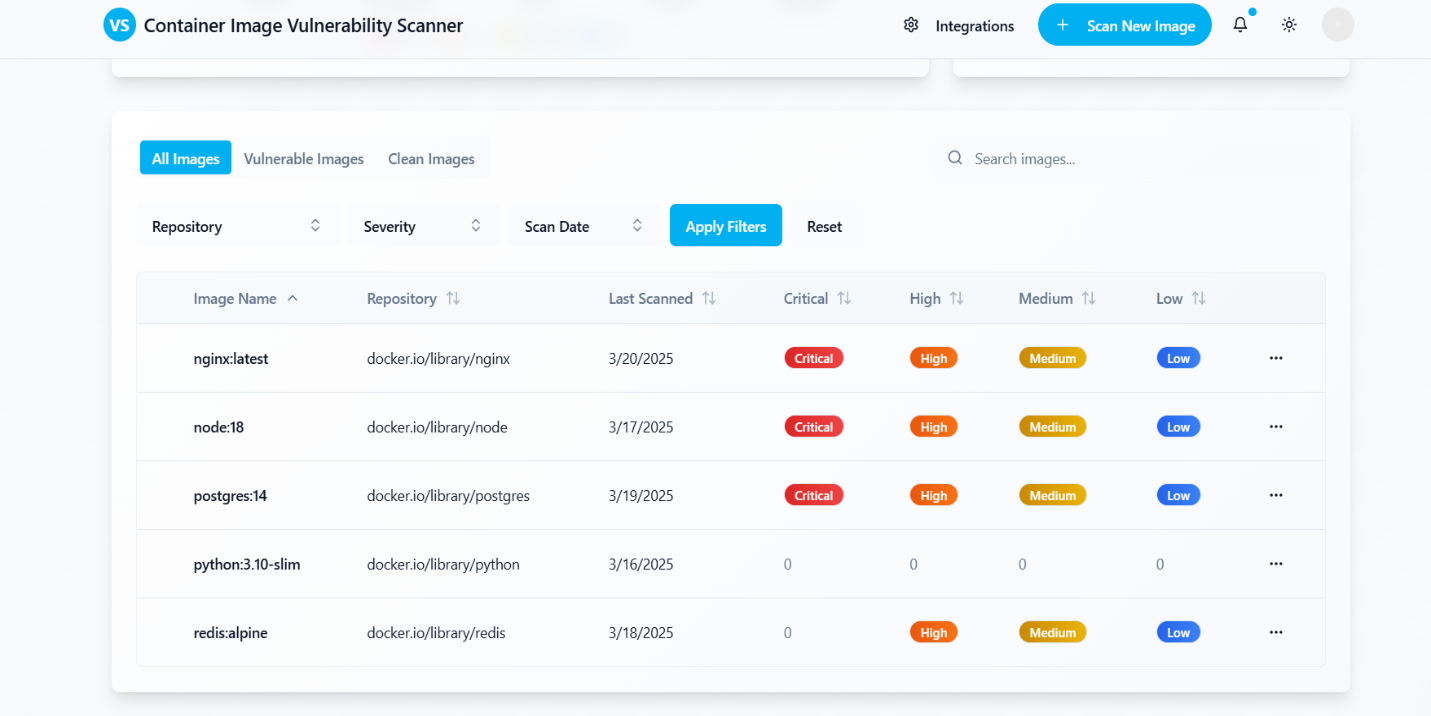


# **UI & UX Design:**

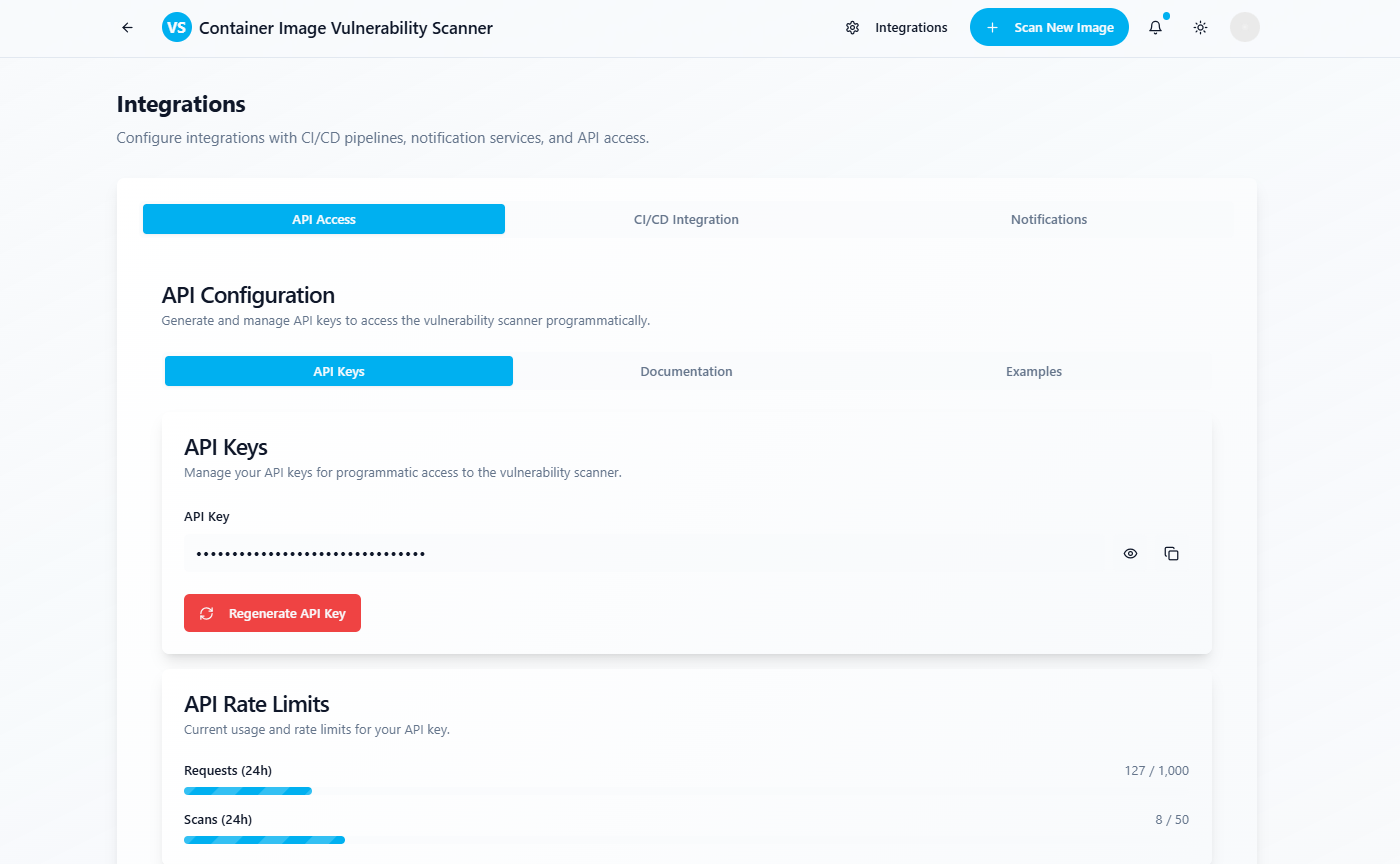
**DASHBOARD**



**IMAGE VULERABILITY DETAILS**

****

**INTEGERATION**

****

# **User Stories:**

As a DevOps engineer, I need to quickly identify and remediate Critical and High vulnerabilities in our container images. Security engineers require a dashboard summarizing container image vulnerability status for security monitoring. Developers need detailed vulnerability reports for their container images to facilitate remediation during development.

# **Release Criteria:**

**1. Functionality:**

* The tool must scan container images and accurately identify vulnerabilities.
* It should provide severity scoring along with actionable remediation steps.

**2.Usability:**

* The user interface must be intuitive, allowing for clear navigation.
* An easy-to-read dashboard and detailed reports should be included.

**3.Performance:**

* Scans must be completed within acceptable time limits (less than 5 minutes per image).
* The tool must be capable of handling large-scale repositories containing thousands of images.

**4.Security:**

* Sensitive data, such as repository credentials, must be handled securely.
* Compliance with security standards, such as SOC 2 or ISO 27001, is required.

# **Timeline:**

|  |  |  |
| --- | --- | --- |
| Phase | Duration | Deliverables |
| Phase 1: Scanning Engine | 3 weeks | Basic scanning functionality with vulnerability detection |
| Phase 2: Reporting & Dashboard | 2 weeks | Dashboard summarizing scan results; detailed reports |
| Phase 3: Integration | 2 weeks | API for CI/CD tools; notification system |
| Phase 4: Testing & Release | 1 week | Final testing and bug fixes |

# **Development Action:**

**✅ Backend Development:**

* Integrate Trivy/Clair API for scanning.
* Develop a backend using Python (Flask/Django) or Go.
* Store scan results in a PostgreSQL/SQLite database.
* Implement authentication and authorization mechanisms.

**✅ Frontend Development:**

* Use React or Vue.js to build the user interface.
* Implement interactive data tables for scan results.
* Ensure a responsive design for mobile and desktop users.

**✅ CI/CD Integration:**

* Set up Jenkins/GitHub Actions for automated scans.
* Provide API endpoints for continuous security monitoring.
* Implement webhooks for real-time notifications.

# **Success Certain:**

**✅ Accuracy & Effectiveness**

**Detection Rate:** Identify at least 99% of known vulnerabilities.

**False Positive Rate:** Keep below 5% to ensure reliable results.

**✅ Performance & Scalability**

**Scan Speed:** Complete scanning within 5 minutes per image.

**Concurrency:** Support scanning of 100 or more images simultaneously.

**✅ User Adoption & Experience**

**User Satisfaction Score:** Maintain an average score of 4.5 out of 5 or higher.

**Onboarding Time:** Users should be able to set up and run a scan within 5 minutes.

**✅ Integration & Automation**

**CI/CD Adoption Rate:** At least 80% of users should integrate scans into their CI/CD pipelines.

**API Usage:** 70% of users should utilize the API for automated scans.

**✅ Compliance & Security**

**Compliance Coverage:** Meet PCI-DSS, SOC2, and GDPR standards.

**Incident Response Time**: Address security issues within 24 hours.

# **Conclusion:**

The Container Image Vulnerability Scanner is a crucial security tool designed to identify and mitigate risks associated with containerized applications. By providing comprehensive vulnerability reports, actionable remediation steps, and seamless CI/CD integration, it empowers DevOps and security teams to maintain a robust security posture. With well-defined success criteria, continuous monitoring, and planned enhancements, this tool will evolve to meet the ever-changing landscape of cloud security