Product Requirements Document

Product Name: Container Image Vulnerability Scanner
Prepared By: Venukumar Thumati
Date: April 10, 2025.

1. Problem Statement

Organizations using containers often manage thousands of container images. These images may have known vulnerabilities due to outdated or insecure dependencies. It is difficult for users to identify:

- Which images are vulnerable
- The severity of the vulnerabilities
- What action needs to be taken

2. Goal

Provide users with a dashboard that:

- Displays all container images and their vulnerability status
- Highlights critical/high severity issues
- Allows filtering, sorting, and prioritizing remediation actions

3. User Personas

Primary User:

- DevSecOps Engineers / Security Engineers

Secondary User:

- DevOps / SRE / Engineering Managers

4. User Stories

- As a user, I want to view a list of all container images along with their vulnerability count.
- As a user, I want to filter images based on vulnerability severity (Critical, High, Medium, Low).
- As a user, I want to sort images by the number of vulnerabilities.
- As a user, I want to click on an image to view detailed vulnerabilities.
- As a user, I want to export or mark images that need urgent fixing.

5. Features & Functional Requirements

Feature	Description
Dashboard View	List all container images with name, tags, and scan summary
Severity Summary	Show a color-coded bar/indicator for each severity level
Image Detail View	On click, show vulnerabilities, CVE ID, description, fix available
Filtering	Filter by severity, scan date, image name
Sorting	Sort by severity, image name, or date

Search	Search by image name
Actions	Mark image as “Fix Needed”, “Ignored”, or “Fixed”
Auto Scan Scheduler	Optional: Schedule regular scanning of images

6. UX Expectations

- Intuitive dashboard with clean layout
- Use red/orange/yellow/green to visually represent severity
- Ensure it works with large datasets (thousands of images)
- Responsive UI for different screen sizes

7. Prioritization (MVP vs Future)

MVP:

- Dashboard view
- Severity filter and sorting
- Detail view of image
- Mark image for action

Future Enhancements:

- Export reports
- Integration with CI/CD tools (Jenkins, GitHub Actions)

- Notification system (email/slack alerts)

8. KPIs / Success Metrics

- Time to identify critical vulnerabilities
- Reduction in unpatched critical images
- Number of images scanned per day/week
- User engagement with dashboard

Low-Fidelity Wireframes

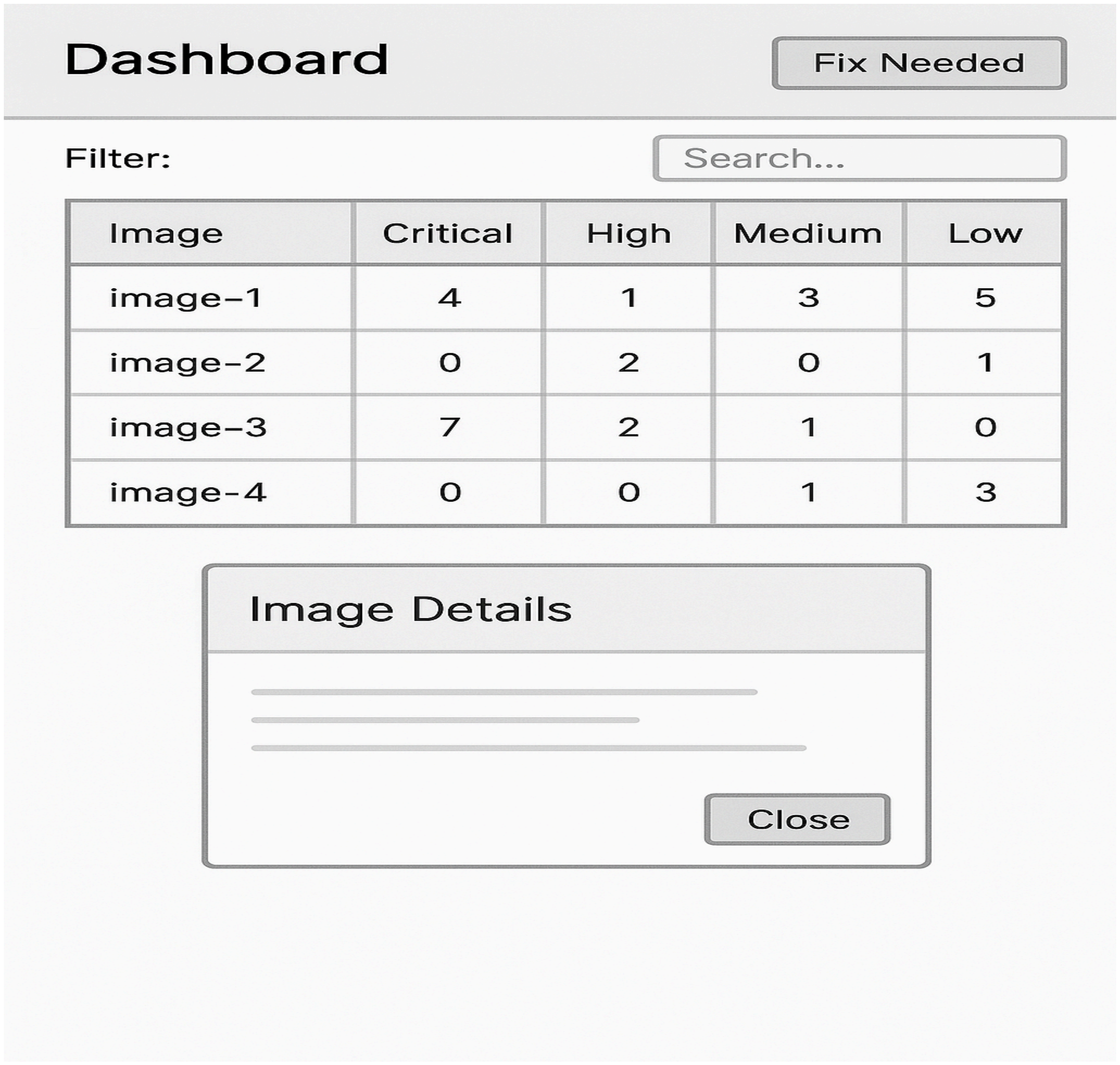

Image	Critical	High	Medium	Low
image-1	4	1	3	5
image-2	0	2	0	1
image-3	7	2	1	0
image-4	0	0	1	3

Development Action Items

Backend Development:

- Container Scanning Engine Integration
  - Integrate tools like Trivy, Clair, or Anchore for vulnerability scanning of container images.
  - Schedule periodic scans (cron jobs or event-based triggers) to detect new vulnerabilities in stored images.

Database Design

- Tables/collections to store:
  - Image Metadata (Name, Tag, Digest, Date Scanned)
  - Vulnerability Details (CVE ID, Severity, Description, Fix Available)
  - Scan Results Mapping (Image ↔ Vulnerability List)
  - User actions like Marked as Fixed, Ignored

Security & Performance

- Secure API endpoints (JWT, OAuth2)
- Handle high-volume scan data with pagination and indexing

Frontend Development:

- ### Dashboard UI Components
  - Image listing table with columns: Image Name, Scan Date, Severity Summary
  - Severity shown with color indicators or badges (Critical, High, Medium, Low)
- ### Filter/Search Controls
  - Severity filter (dropdown or checkbox)
  - Search bar for image name/tag
  - Sort functionality (by date, severity, name)
- ### Modal or Detail Page View
  - On row click, show modal or route to detail page with:
    - Full vulnerability list

        - CVE ID, severity, description, and remediation steps
        - “Fix Needed”, “Ignore”, “Fixed” buttons

- **Responsiveness & Performance**
    - Use lazy loading, pagination for large image sets
    - Mobile/tablet responsiveness (optional for future)

API Design (Sample Endpoints)

Method	Endpoint	Description
`GET`	`/api/images`	List all container images with scan summary
`GET`	`/api/images/{id}`	Get details and vulnerabilities for a single image
`POST`	`/api/images/{id}/mark`	Mark an image as “Fix Needed”, “Ignored”, or “Fixed”
`GET`	`/api/vulnerabilities`	List all known vulnerabilities

`POST`	`/api/scan`	Trigger scan manually for an image

CI/CD & Automation

- Set up **CI pipeline** to trigger scanning after new image builds like GitHub Actions, Jenkins.
- Integrate scanner into container registry workflow.
- Alerting mechanism for **critical/high vulnerabilities** via email or Slack.
- Version control: Use Git for managing frontend/backend codebase.