Product Requirements Document (PRD)

Introduction & Vision

The product, "ImageGuard," is a security tool designed to scan container images for known vulnerabilities. With thousands of images in a typical repository, DevOps and Security teams struggle to identify and prioritize threats. ImageGuard will provide a clear, actionable dashboard that highlights vulnerabilities by severity, enabling teams to quickly identify which images need immediate attention and how to fix them.

Our vision is to empower teams to ship secure containerized applications by making vulnerability management simple and efficient.

User Personas

- **Priya, the DevOps Engineer:** Responsible for building and deploying applications. Her primary goal is to ensure the CI/CD pipeline is efficient and secure. She needs to quickly see if a new build introduces critical vulnerabilities before it goes to production.
- **Sam, the Security Engineer:** Responsible for the overall security posture of the organization. He needs a high-level overview of vulnerabilities across all repositories and wants to track the mean time to remediation (MTTR) for critical issues.

Features & User Stories

Epic 1: Vulnerability Dashboard

- **As Priya/Sam, I want** a dashboard overview of all my scanned images, **so that** I can quickly understand the overall security risk.
- **As Sam, I want** to see a breakdown of vulnerabilities by severity (Critical, High, Medium, Low), **so that** I can prioritize our team's efforts on the most significant threats.
- **As Priya, I want** to see a list of the "Most Vulnerable Images," **so that** I can immediately focus on the biggest problem areas.

Epic 2: Image Repository View

- **As Priya, I want** to view a list of all my scanned images in a sortable and filterable table, **so that** I can easily navigate through thousands of images.
- **As Priya/Sam, I want** to filter images based on their vulnerability severity (e.g., show me all images with at least one "Critical" vulnerability), **so that** I can create a targeted remediation plan.
- **As Priya, I want** to search for a specific image by its name or tag, **so that** I can check the status of a particular application.

Epic 3: Detailed Vulnerability Report

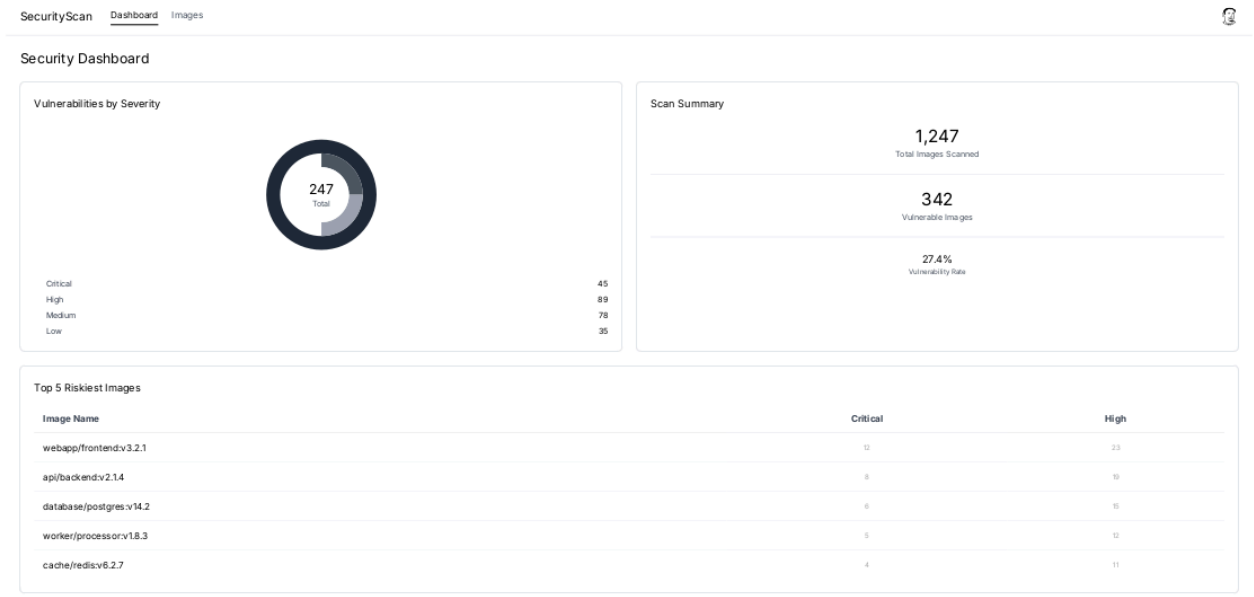
- **As Priya, I want** to click on an image and see a detailed list of all its vulnerabilities, **so that** I can understand the specific risks.
- **As Priya, I want** the vulnerability report to show me the vulnerable package, its current

version, and the version that includes a fix, **so that** I know exactly how to remediate the vulnerability.

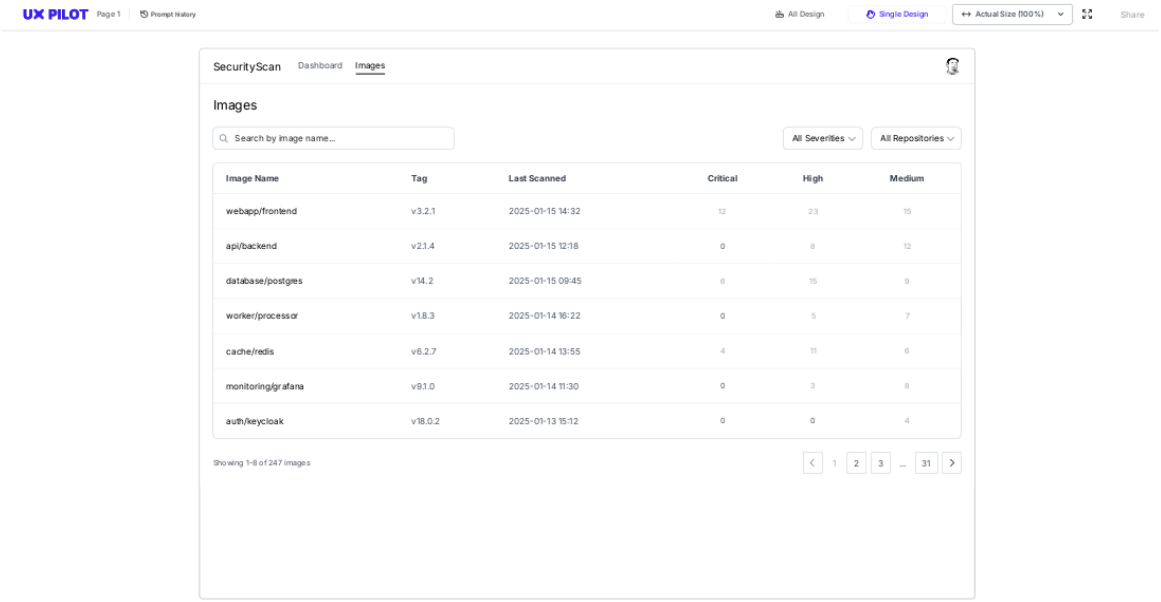
- **As Sam, I want** each vulnerability to link to its CVE (Common Vulnerabilities and Exposures) entry (e.g., CVE-2021-44228), **so that** I can get more context about the threat.

Low-Fidelity Wireframe

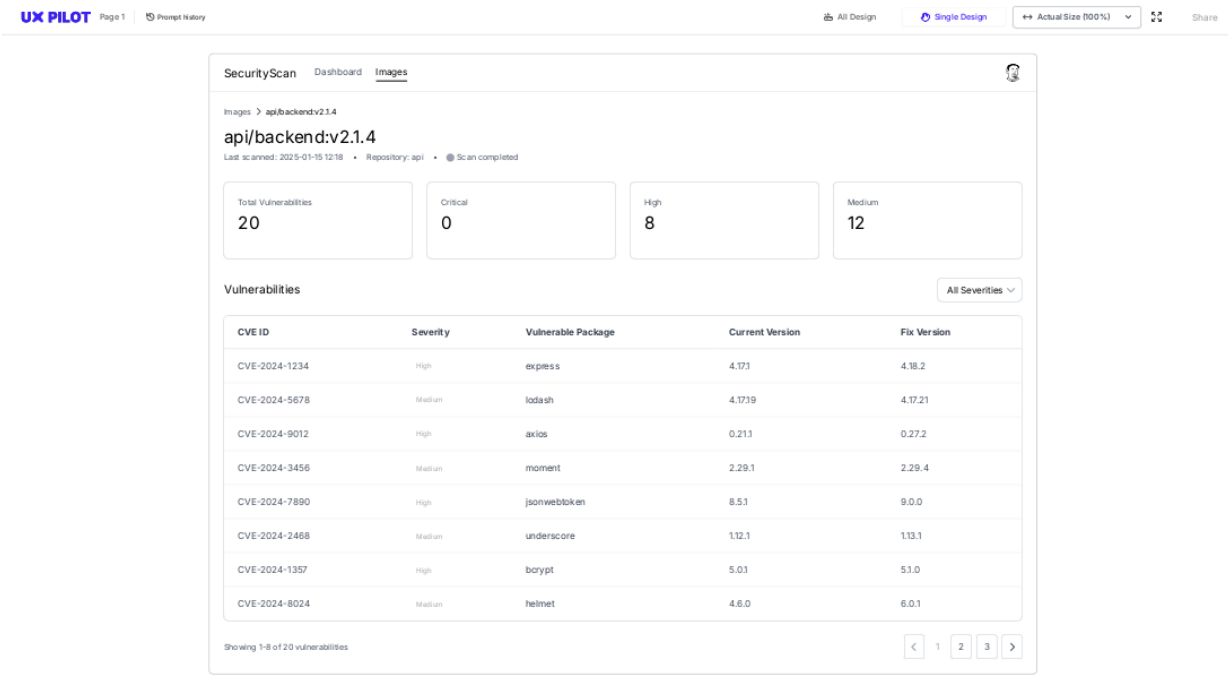
Screen 1: Dashboard



Screen 2: Container Images



Screen 3: An Image Scan Results



Development action items to discuss with Development Team

Epic: Backend & API (`~4 Sprints`)

- **Task 1 (Spike):** Research and select a container scanning engine (e.g., Trivy, Grype, Clair).
- **Task 2:** Set up the chosen scanner and create a service that can accept an image name and trigger a scan.
- **Task 3:** Design a database schema to store image metadata, scan results, and vulnerability details (CVE, severity, package, etc.).
- **Task 4:** Develop API endpoints:
    - `POST /api/scan`: To trigger a new scan.
    - `GET /api/images`: To list all images with their vulnerability counts (for Screen 2).
    - `GET /api/images/{id}`: To get detailed vulnerability info for a single image (for Screen 3).
    - `GET /api/dashboard/summary`: To provide aggregated data for the dashboard (for Screen 1).
- **Task 5:** Implement authentication to secure the API endpoints.

Epic: Frontend (`~3 Sprints`)

- **Task 1:** Set up the frontend framework (e.g., React, Vue, Angular).

- **Task 2:** Build the main application shell (navigation, layout).
- **Task 3:** Develop the Dashboard page, including data visualization components (charts, lists) that consume the dashboard API.
- **Task 4:** Build the Images List page, implementing the table with client-side or server-side sorting, filtering, and search functionality.
- **Task 5:** Create the Image Detail page to display the granular vulnerability report.
- **Task 6:** Integrate frontend with all backend APIs and handle loading/error states.

Epic: Infrastructure & CI/CD (`Ongoing`)

- **Task 1:** Create a CI/CD pipeline to automatically build, test, and deploy the backend and frontend services.
- **Task 2:** Containerize the application (backend API, frontend server, scanning service).
- **Task 3:** Plan for scaling the scanner service, as scanning thousands of images can be resource-intensive. Consider a job queue system (e.g., RabbitMQ, SQS) to manage scan requests.