Product requirements Container Image Vulnerability Scanner

**1. Introduction**

This document outlines the requirements for a security product designed to scan container images for known vulnerabilities and present the findings to users in a clear and actionable manner. The primary goal is to enable users to identify and remediate vulnerabilities in their container images, thereby improving the security posture of their applications.

**2. Goals**

* Provide users with a comprehensive view of vulnerabilities found in their container image repositories.
* Enable users to easily identify images with critical and high severity vulnerabilities.
* Facilitate the process of identifying which images require immediate attention and fixing.
* Support users with a large number of container images.

**3. Target Users**

* Security Engineers
* DevOps Engineers
* Software Developers
* Platform Engineers

**4. Functional Requirements**

* **Image Scanning:**
  + The product must be able to scan container images from various registries (e.g., Docker Registry, Amazon ECR, Google Container Registry).
  + The product should support scanning images based on tags and digests.
  + Scanning should be performed on-demand or through scheduled scans.
* **Vulnerability Detection:**
  + The product must identify known vulnerabilities in the operating system packages and application dependencies within the container images.
  + It should leverage reputable vulnerability databases (e.g., CVE, NVD).
  + The product should categorize vulnerabilities based on severity levels (e.g., Critical, High, Medium, Low, Informational).
* **Reporting and Visualization:**
  + The product should provide a dashboard or list view to display scanned images and their vulnerability status.
  + Users should be able to see the total number of vulnerabilities and the breakdown by severity for each image.
  + The UI should clearly highlight images with critical and high severity vulnerabilities.
  + Users should be able to drill down into individual images to view the details of each vulnerability.
  + Vulnerability details should include:
    - CVE ID
    - Severity level
    - Description of the vulnerability
    - Affected package and version
    - Suggested remediation (if available)
* **Filtering and Searching:**
  + Users should be able to filter images based on registry, repository, tag, and vulnerability severity.
  + Users should be able to search for specific images or vulnerabilities.
* **Notifications (Optional - Could be a later enhancement):**
  + The product could provide notifications (e.g., email, Slack) when new critical or high severity vulnerabilities are discovered in monitored images.
* **Integration (Optional - Could be a later enhancement):**
  + The product could integrate with CI/CD pipelines to automatically scan images during the build process.
  + Integration with ticketing systems (e.g., Jira) could allow for automated issue creation for high-risk vulnerabilities.
* **User Authentication and Authorization:**
  + The product should provide secure user authentication and authorization mechanisms.
  + Role-based access control could be implemented to manage user permissions.

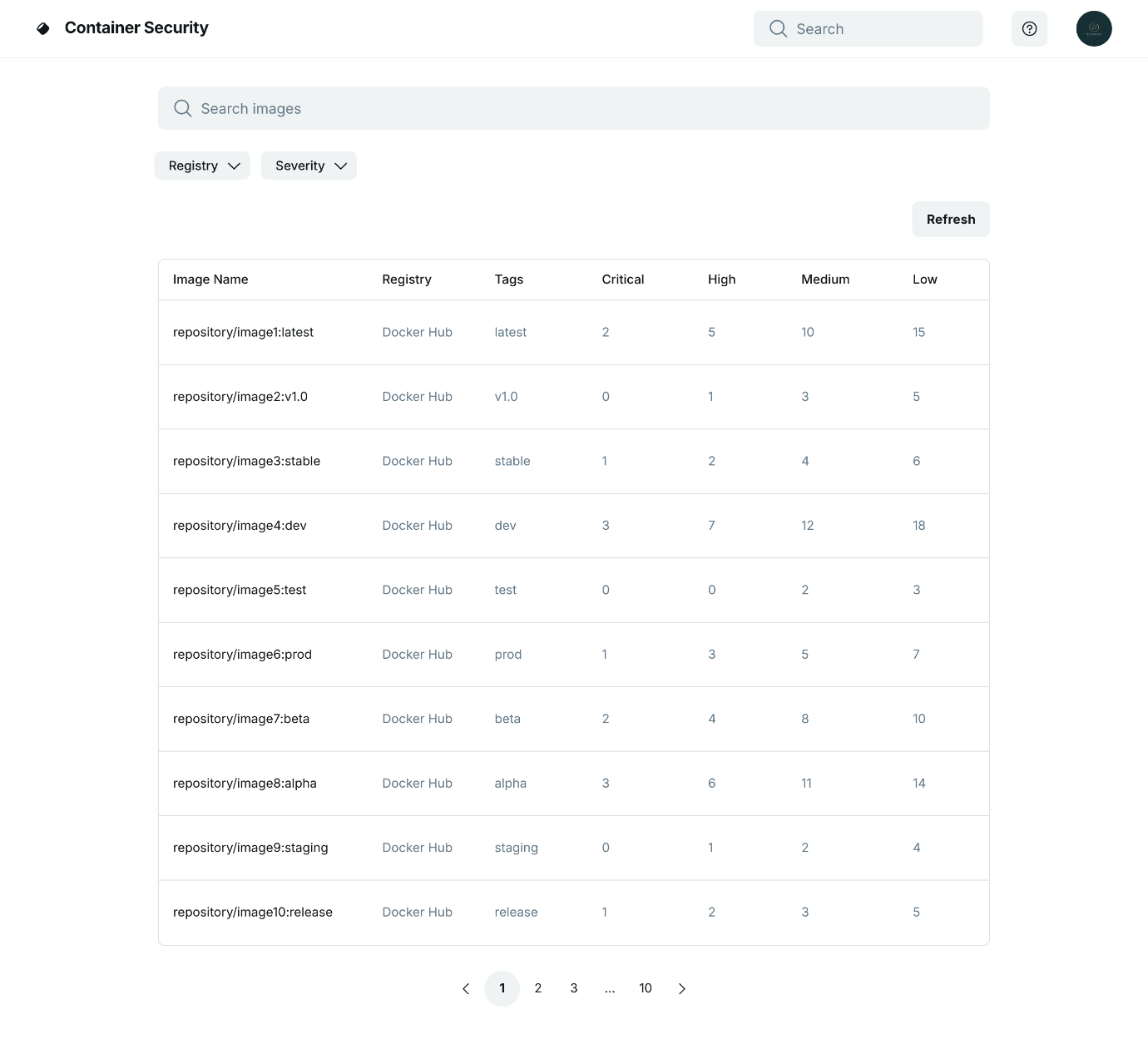
**5. Non-Functional Requirements**

* **Performance:** The product should be able to scan images efficiently and provide results in a timely manner, even with a large number of images.
* **Scalability:** The system architecture should be scalable to handle a growing number of images and scan requests.
* **Reliability:** The vulnerability detection process should be accurate and reliable. The system should be available and stable.
* **Security:** The product itself should be secure and not introduce new vulnerabilities. Access to vulnerability data should be protected.
* **Usability:** The user interface should be intuitive and easy to navigate, allowing users to quickly understand the vulnerability status of their images.

**Low-Fidelity Wireframes**

Here are some basic wireframes to illustrate the user interface:

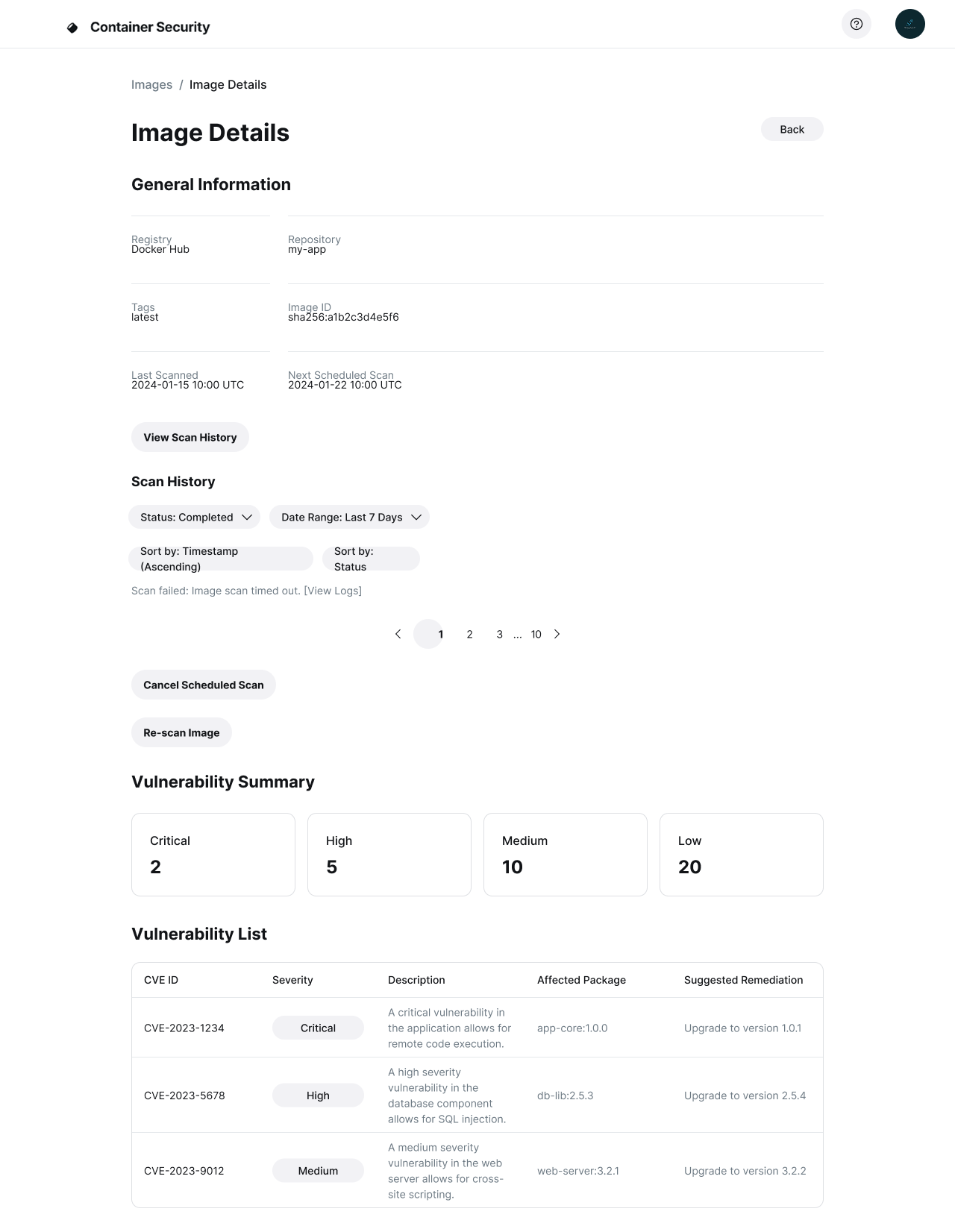
1. **Dashboard/Image List View:**



**Key elements:**

* Shows a list of scanned container images.
* Displays the image name (including repository and potentially the image ID), registry, and tags.
* Clearly indicates the count of vulnerabilities by severity level for each image.
* Highlights images with critical and high vulnerabilities (e.g., using red or orange indicators).
* Provides the timestamp of the last scan.
* Includes search and filtering options to help users find specific images.

**2. Image Details View (Accessed by clicking on an image in the list):**



**Key elements:**

* Provides detailed information about the selected container image.
* Shows a summary of vulnerabilities by severity.
* Lists each detected vulnerability with its key details, including a description and suggested remediation.

**(Bonus/Optional) Development Action Items**

Here are some initial development action items for discussion with the development team:

1. **Choose a Vulnerability Scanning Engine:** Research and select an appropriate open-source or commercial vulnerability scanning engine (e.g., Trivy, Anchore Engine, Clair).
2. **Implement Image Registry Integration:** Develop the functionality to connect and authenticate with various container image registries (Docker Hub, ECR, GCR, etc.). Consider different authentication methods.
3. **Develop Image Scanning Logic:** Implement the core logic to pull container images from the registries and trigger the vulnerability scans using the chosen engine.
4. **Design and Develop the Data Model:** Define the database schema to store information about images, scan results, and vulnerabilities.
5. **Build the Backend API:** Create APIs to manage image scans, retrieve scan results, and provide data to the frontend.
6. **Develop the Frontend User Interface:** Implement the UI based on the wireframes, focusing on clear presentation of vulnerability data and ease of navigation.
7. **Implement Filtering and Search Functionality:** Add features to allow users to filter and search through their images and vulnerabilities.
8. **Implement User Authentication and Authorization:** Set up secure user management with appropriate roles and permissions.
9. **(Optional) Implement Notification System:** Integrate a notification mechanism to alert users about new critical or high severity vulnerabilities.
10. **(Optional) Explore CI/CD Integration:** Investigate how to integrate the scanner into existing CI/CD pipelines.

This is a starting point. The requirements and wireframes can be further refined based on user feedback and more detailed discussions with the stakeholders and development team.