Problem Statement 1:

Title: Product Requirement and Low-Fidelity Wireframes

Product Requirements Document (PRD)

1. Product Overview

Name: Container Image Vulnerability Scanner
Purpose: Enable users to monitor, prioritize, and remediate vulnerabilities across thousands of container images in their repositories.
Target Users: DevOps Engineers, Security Engineers, SREs.

---

2. Key Problems to Solve

- Users need visibility into vulnerabilities across thousands of images.
- Users need to quickly identify and prioritize images with **critical/high** vulnerabilities.
- Users need actionable insights to **remediate** issues.

---

3. User Stories

- As a user, I want to **view a list of all container images** with vulnerability summaries.
- As a user, I want to **filter images** by severity level (e.g., critical, high).
- As a user, I want to **sort images** by the number of vulnerabilities.
- As a user, I want to **drill down** into a single image to see vulnerability details.
- As a user, I want to see **fix availability** and **recommendations** for each vulnerability.
- As a user, I want to be notified about **new critical vulnerabilities** in my images.

---

4. Core Features

Feature	Description
Dashboard Summary	Overview of scanned images and their vulnerability distribution.
Image List View	Tabular view of all container images with summary stats (e.g., # of vulnerabilities, last scanned, risk score).

Feature	Description
Filters & Sorting	Filter by severity (Critical/High/Medium), fixable status, repository, etc.
Image Detail View	Deep dive into vulnerabilities in a single image, grouped by severity and package.
Remediation Guidance	Suggested actions like "Upgrade to x version", links to CVEs, fix status.
Search Functionality	Search by image name or tag.
Notification Setup	Alerting users when critical vulnerabilities are detected.

---

5. Non-Functional Requirements

- **Scalability**: Handle thousands of images efficiently.
- **Performance**: Load summary dashboard within 3 seconds.
- **Security**: Role-based access control (RBAC).
- **Integrations**: GitHub, DockerHub, private container registries.

---

6. KPIs (Success Metrics)

- % of high/critical vulnerabilities remediated after detection.
- Time to detect and display vulnerabilities after image scan.
- User adoption and engagement rates.

Low-Fidelity Wireframes

---

1. Dashboard Overview Page

```
mathematica
CopyEdit
+-------------------------------------------------------------+
|  [Logo]  | Dashboard | Images | Alerts | Settings           |
+-------------------------------------------------------------+
|   Summary Cards:                                            |
|   [Total Images Scanned]   [Critical Vuln Count]           |
|   [High Vuln Count]        [Fixable Vuln Count]            |
+-------------------------------------------------------------+
|   Vulnerability Trend Graph (Over time)                     |
|   Top 10 Images by Vulnerability Count (List/Bar Chart)     |
+-------------------------------------------------------------+
```

---

2. Image List View

```
+-------------------------------------------------------------+
|  [Search bar] [Filter: Severity | Fixable | Repo]           |
+-------------------------------------------------------------+
| Image Name | Tag | Critical | High | Medium | Last Scanned |
|-------------------------------------------------------------|
| nginx      | 1.19| 5        | 12   | 20     | 2025-06-21    |
| myapp-api  | 2.3 | 1        | 4    | 0      | 2025-06-20    |
+-------------------------------------------------------------+
| Pagination: < 1 2 3 ... >                                   |
```

---

3. Image Detail View

```
sql
CopyEdit
+-------------------------------------------------------------+
| Image: nginx:1.19                                           |
| Repo: dockerhub/nginx   Last Scanned: 2025-06-21            |
+-------------------------------------------------------------+
| Vulnerability Table:                                        |
| Severity | Package | CVE ID     | Fix Available | Notes     |
|----------|---------|------------|---------------|-----------|
| Critical | openssl | CVE-2023...| Yes (1.1.1w)   | Upgrade   |
| High     | zlib    | CVE-2022...| No             | -         |
+-------------------------------------------------------------+
| [Download Report]  [Fix Suggestions]                        |
```

---

4. (Optional) Notification Settings Page

```
pgsql
CopyEdit
+-------------------------------------------------------------+
| [Enable Email Alerts] [ x ]                                 |
| Alert me when:                                              |
| [x] Critical vulnerability is detected                      |
| [x] New image is scanned                                    |
| Email: [__________]  [Save Settings]                        |
+-------------------------------------------------------------+
```

Development Action Item

Item	Description
Backend API	Develop APIs to fetch image lists, vulnerabilities, scan times.

Item	Description
Vulnerability Scanner Integration	Integrate with Clair, Trivy, or custom scanners.
Data Aggregation Layer	Summarize and group vulnerabilities efficiently.
Frontend Table Components	Reusable components for lists, filters, and sorting.
RBAC and Auth	Secure access to data based on user roles.
Notifications Service	Email or webhook-based alert system.
CI/CD Integration	Auto-trigger scan jobs on image upload or tag update.

Problem Statement 2:

Title: Kubernetes Security Scan

Step 1: Install a Local K8s Cluster (using Minikube as an example)

```
# Install Minikube (if not already installed)
curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-
linux-amd64
sudo install minikube-linux-amd64 /usr/local/bin/minikube

# Start Minikube
minikube start
```

Step 2: Install Kubescape

```
# Install Kubescape (Linux/macOS)
curl -s
https://raw.githubusercontent.com/kubescape/kubescape/master/install.sh |
/bin/bash
```

Step 3: Scan the Cluster and Output to JSON

```
# Run the scan and save output in JSON
kubescape scan framework nsa --format json --output path/to/findings.json
```

Output: Sample JSON Snippet

```
{
  "formatVersion": "2.0",
  "customerGUID": "12345678-90ab-cdef-1234-567890abcdef",
  "frameworkReports": [
    {
      "name": "nsa",
      "controls": [
        {
          "controlID": "C-0015",
          "name": "Minimize access to secrets",
          "score": 80,
          "status": "failed",
          "ruleReports": [
            {
              "ruleName": "Ensure secrets are only mounted where needed",
              "status": "failed",
              "resource": {
                "kind": "Pod",
                "name": "nginx-pod",
                "namespace": "default"
              }
            }
          ]
        }
      ]
    }
  ]
}
```


Final Deliverable

Once you run the scan, submit the file

`findings.json`

This file will contain all the Kubernetes misconfigurations or security risks found by the tool.

Problem Statement 3 (Technical):

Step #1: Create a Go Web App for Date & Time

main.go

package main

```
import (
	"fmt"
	"net/http"
	"time"
)

func handler(w http.ResponseWriter, r *http.Request) {
	currentTime := time.Now().Format("Mon Jan 2 15:04:05 MST 2006")
	fmt.Fprintf(w, "Current Date & Time: %s", currentTime)
}

func main() {
	http.HandleFunc("/", handler)
	fmt.Println("Server started at :8080")
	http.ListenAndServe(":8080", nil)
}
```

Dockerfile

```
# Use a minimal base image
FROM golang:1.21 as builder
```

```
WORKDIR /app

COPY main.go .

RUN go build -o datetime-app


FROM gcr.io/distroless/base-debian10

COPY --from=builder /app/datetime-app /datetime-app


ENTRYPOINT ["/datetime-app"]
```

Build and Push Docker Image

```
# Build the Docker image

docker build -t <your-dockerhub-username>/datetime-app:latest .


# Login to DockerHub

docker login


# Push the image

docker push <your-dockerhub-username>/datetime-app:latest
```

Step #2: Deploy to Kubernetes (Declarative, 2 Replicas)

deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: datetime-app
spec:
  replicas: 2
  selector:
    matchLabels:
      app: datetime
  template:
    metadata:
      labels:
        app: datetime
    spec:
      containers:
        - name: datetime
          image: johndoe/datetime-app:latest
          ports:
            - containerPort: 8080
```

service.yaml

```
apiVersion: v1
```

```
kind: Service
metadata:
  name: datetime-service
spec:
  type: LoadBalancer
  selector:
    app: datetime
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080
```

Deploy to Kubernetes

```
kubectl apply -f deployment.yaml
kubectl apply -f service.yaml
```

Step #3: Expose App to Internet

If using a platform like **Qwiklabs** or **GCP**, the `LoadBalancer` service will provision an **external IP**:

Expected output:

```
NAME               TYPE           CLUSTER-IP     EXTERNAL-IP      PORT(S)          AGE
datetime-service   LoadBalancer   10.0.123.45    35.202.XXX.XXX   80:32456/TCP     2m
```

Go to:

http://<EXTERNAL-IP>

And you should see:

Current Date & Time: Mon Jun 23 13:47:22 UTC 2025