Chouaib Doukkali University
National School of Applied Sciences of El Jadida
Department of Telecommunications, Networks, and Computer Science

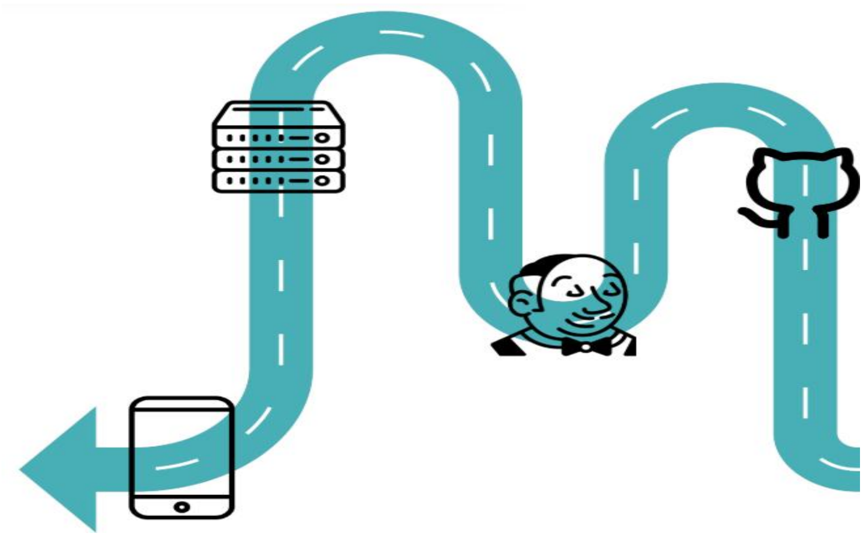
DEVSECOPS PROJECT

Major: **Cybersecurity and Digital Trust**

Year: **2nd Year**

Subject:

A DevSecOps Approach to Secure Mobile App Development



Prepared by :
Anejjar Walid
Bahajoub Nizar
Nadir Ziad
Naoumi Abdelmoughite

Supervised by:
Prof. Assad

Academic Year: 2024/2025

Abstract

This report outlines the development of a mobile application designed for absence management, following DevSecOps principles to ensure security, efficiency, and reliability throughout the software development lifecycle. The project encompasses the entire process, from initial design to deployment, with a strong emphasis on automation, continuous integration, and robust testing.

Key phases include setting up a secure architecture, implementing Continuous Integration (CI) using Jenkins, and conducting advanced tests (integration, end-to-end, and performance) to validate the application's functionality and resilience. The deployment leverages Docker and Kubernetes for containerization and orchestration, ensuring scalability and consistency across environments. A comprehensive CI/CD pipeline is demonstrated, highlighting seamless integration between development, security, and operations.

The report concludes with insights into the project's outcomes, lessons learned, and future enhancements to further optimize the absence management system. By adhering to DevSecOps practices, the project not only delivers a high-quality mobile application but also establishes a framework for secure and agile development in future initiatives.

Table of Contents

1 Introduction

This project presents a modern absence management solution designed for academic institutions, combining a React frontend with a Python/Flask backend to deliver a secure and user-friendly mobile application. The system simplifies attendance tracking by allowing professors to generate unique QR codes for each class session, which students can scan to automatically record their presence. All components are containerized using Docker for portability and deployed via Kubernetes for scalability.

The application features intuitive role-based interfaces: professors can create sessions, view attendance analytics, and manage classes, while students can track their attendance history and receive real-time status updates. The backend, built with Python/Flask, handles QR generation, session management, and data processing, ensuring reliable performance even during peak usage periods.

Security and automation are central to the project's design. We implemented a comprehensive DevSecOps pipeline using Jenkins for continuous integration, SonarQube for code quality analysis, and Trivy for vulnerability scanning. Every code change triggers automated tests, security checks, and deployment to staging environments. Runtime protection includes ModSecurity for attack prevention and TLS encryption for secure data transmission.

By leveraging modern technologies and DevSecOps practices, this solution not only addresses the immediate need for efficient absence management but also establishes a foundation for future enhancements. The modular architecture allows for easy expansion, whether adding new features, integrating with existing school systems, or scaling to support larger institutions. The result is a robust, maintainable system that demonstrates how thoughtful application design can transform administrative processes in educational settings.

.

2 Context and Architecture

Introduction:

This section provides a comprehensive overview of the application, detailing its technical foundation, and the specific functionalities designed for professors and students. The application is a web and mobile solution for managing class attendance, leveraging a modern tech stack and a robust CI/CD pipeline to ensure scalability and reliability.

2.1 Application overview:

The technical foundation of the application is built on a modern, scalable stack. The frontend is developed using React, a JavaScript library for building user interfaces, leveraging JSX for component-based development and Tailwind CSS for styling. React's component-based architecture allows for reusable UI elements, ensuring maintainability and consistency across the web and mobile interfaces

The backend is powered by Python using Flask, a lightweight and flexible web framework. Flask is well-suited for building RESTful APIs because it provides simplicity and fine-grained control over request handling. This allows the backend to efficiently manage API requests, route endpoints, and process data, all while maintaining minimal overhead. Thanks to Flask's modular design, it's easy to extend the backend with additional features or integrate with other services as needed.

The Flask backend communicates with a PostgreSQL database, which serves as the central repository for storing data such as user information, attendance records, and course details. PostgreSQL is a powerful, open-source relational database known for its robustness and scalability, making it an excellent choice for handling complex queries and ensuring data integrity.

2.2 Professor Features:

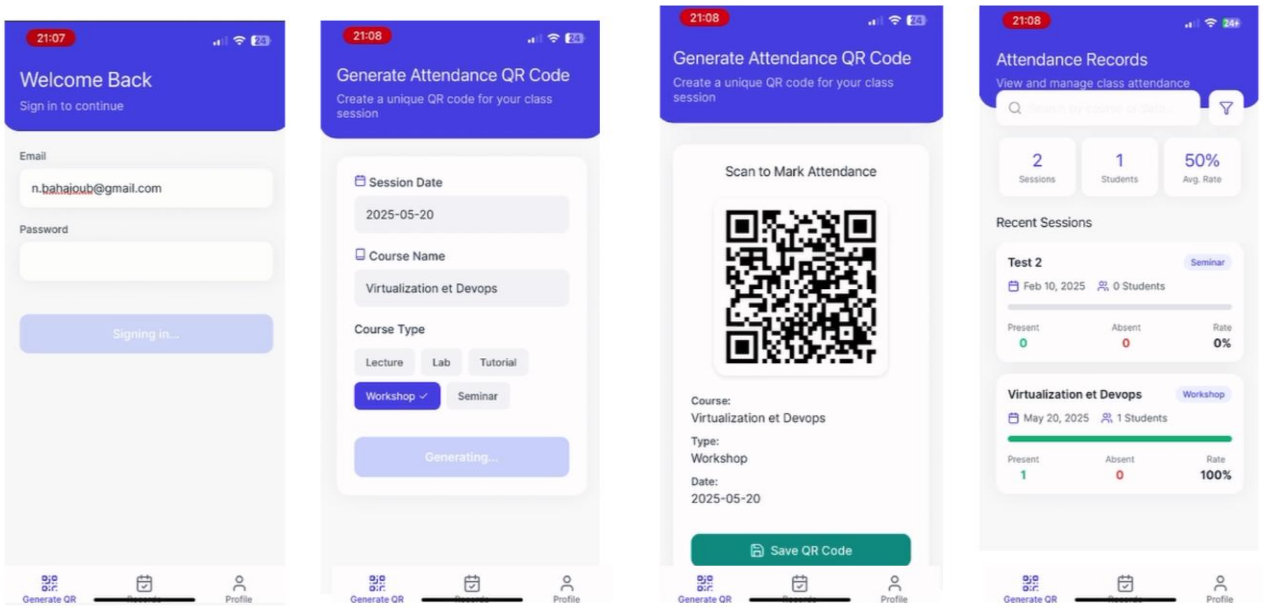
The application provides professors with a comprehensive dashboard to manage attendance and course-related activities. Key functionalities include:

• **Login:** Professors can sign in using their email and password, with secure authentication handled by the Flask backend and stored credentials in the PostgreSQL database. The login interface ensures a seamless entry point to the dashboard.

• **Generating QR Code Dashboard:** Professors can generate dynamic QR codes for each class session, which students scan to mark attendance. The QR code generation is handled by the backend, ensuring unique codes tied to specific courses and timestamps.

• **Attendance Records:** This feature allows professors to view and manage attendance records for their courses. The dashboard displays attendance data with filtering options by course and date, presenting metrics such as attendance percentages (e.g., 50% as shown in the demo). Professors can export or update records as needed.

• **Profile:** The profile section enables professors to view and update their personal information, including contact details and course assignments, ensuring accurate and up-to-date data.


Welcome Back
Sign in to continue
Email
n.bahajoub@gmail.com
Password
Signing in...

Generate Attendance QR Code
Create a unique QR code for your class session
Session Date
2025-05-20
Course Name
Virtualization et Devops
Course Type
Lecture | Lab | Tutorial | Workshop ✓ | Seminar
Generating...

Generate Attendance QR Code
Create a unique QR code for your class session
Scan to Mark Attendance
Course:
Virtualization et Devops
Type:
Workshop
Date:
2025-05-20
Save QR Code

Attendance Records
View and manage class attendance
2 Sessions | 1 Students | 50% Avg. Rate
Recent Sessions
Test 2 — Seminar — Feb 10, 2025 — 0 Students — Present 0 — Absent 0 — Rate 0%
Virtualization et Devops — Workshop — May 20, 2025 — 1 Students — Present 1 — Absent 0 — Rate 100%

Figure 1: *Professor's pages.*

2.3 Student Features:

The student interface is designed to be user-friendly, enabling students to engage with the attendance system efficiently. Key functionalities include:

• **Login:** Students access the application via a login page using their email and password, with authentication managed securely by the Flask backend. The interface is optimized for both web and mobile access.

• **Profile:** The profile page allows students to view and update their personal information, such as contact details and enrolled courses, ensuring their data remains current.

• Attendance: Students can view their attendance history for each course, with details such as attendance status and dates, providing transparency and accountability.

• **Scan QR Pages:** The QR code scanning feature enables students to mark their attendance by scanning professor-generated QR codes during class sessions. The mobile application uses the device's camera to scan codes, with real-time feedback (e.g., Try Again if scanning fails, as shown in the demo).

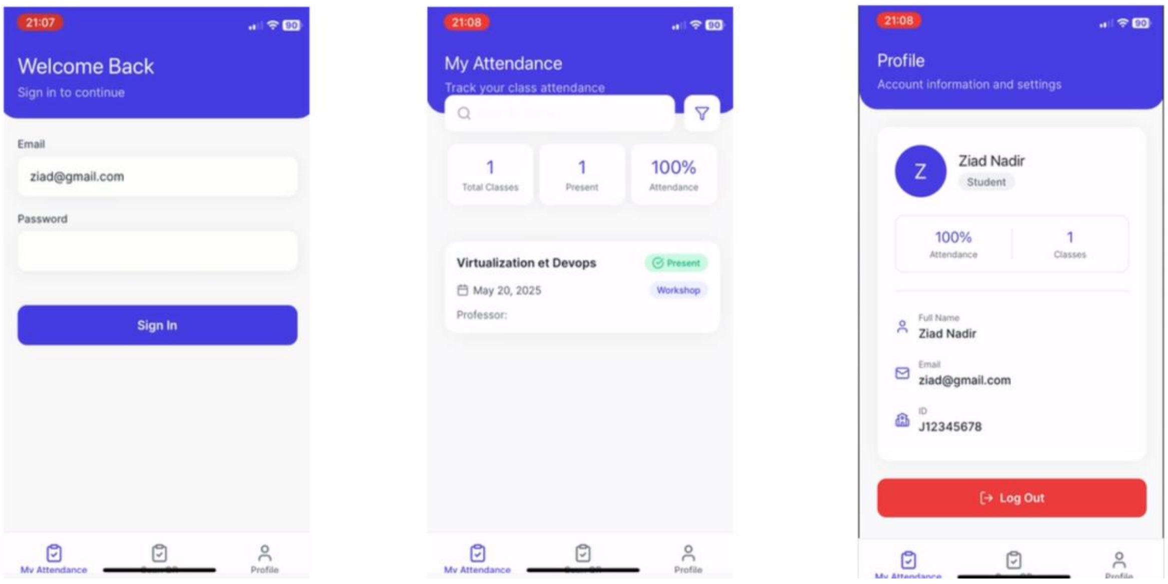


Figure 2: *Student's login, profile and attendance pages.*

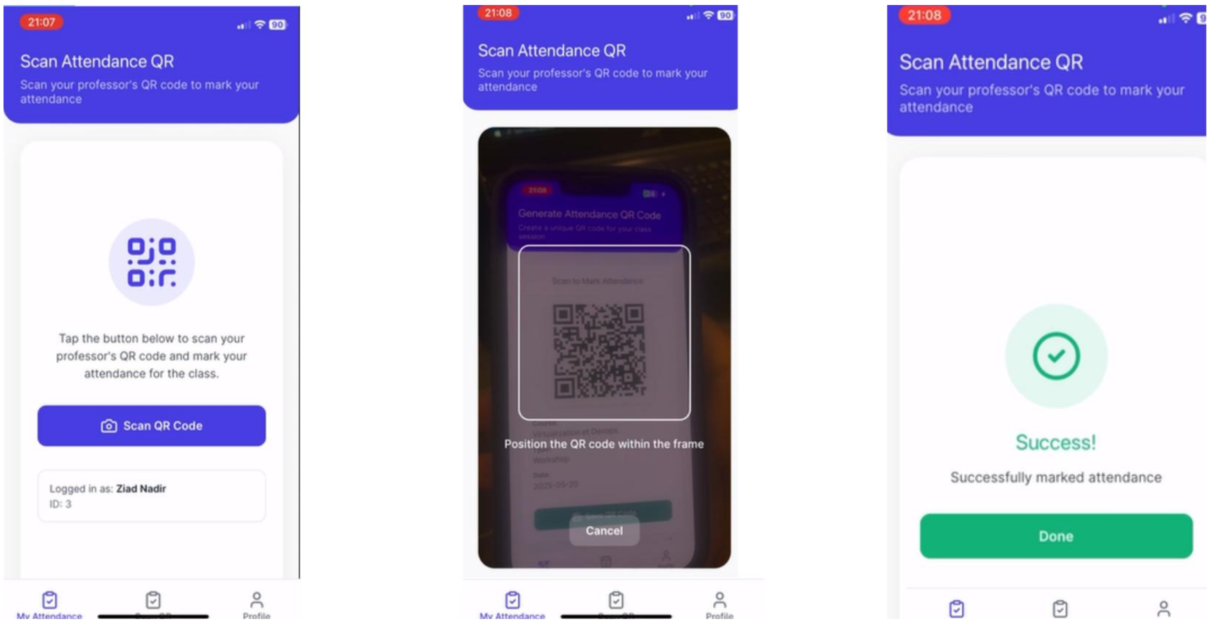

21:07
Scan Attendance QR
Scan your professor's QR code to mark your attendance
Tap the button below to scan your professor's QR code and mark your attendance for the class.
Scan QR Code
Logged in as: Ziad Nadir
ID: 3
My Attendance
Profile
21:08
Scan Attendance QR
Scan your professor's QR code to mark your attendance
Generate Attendance QR Code
Position the QR code within the frame
Cancel
21:08
Scan Attendance QR
Scan your professor's QR code to mark your attendance
Success!
Successfully marked attendance
Done

Figure 3: *Scan attendance QR page.*

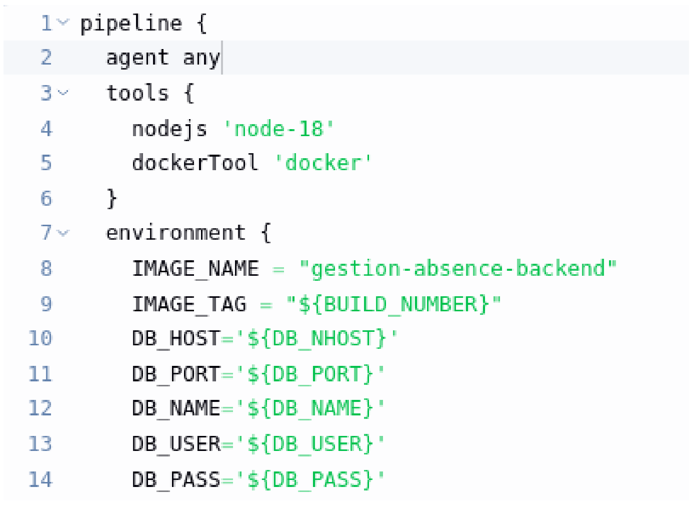
3 Continuous Integration with Jenkins

Introduction :

This section details the Continuous Integration (CI) process implemented for the attendance management application using Jenkins, an open-source automation server. The CI pipeline automates code validation, testing, building, and deployment, ensuring high quality software delivery. The pipeline, defined in a Jenkinsfile, integrates with GitHub, Docker, Nexus, and Kubernetes to support a robust CI/CD workflow. The following subsections describe the pipeline setup, the automated build process, and the integration with GitHub

3.1 Jenkins Pipeline Setup

The Jenkins pipeline is configured as a multibranch pipeline, enabling automatic detection and processing of multiple repository branches. The pipeline is defined in a declarative Jenkinsfile, which specifies stages for code checkout, building, testing, and deployment. It runs on any available Jenkins agent, with tools such as Node.js (version 18) and Docker explicitly configured. Environment variables, including database credentials (DB_HOST, DB_PORT, DB_NAME, DB_USER, DB_PASS) and Nexus registry details (NEXUS_REGISTRY), are defined to ensure consistent configuration across stages. The pipeline includes a timestamps() option to log execution times for each step, aiding in debugging and performance monitoring. The pipeline is designed to handle the backend image (gestion-absence-backend) with a build-specific tag (BUILD_NUMBER) for versioning.



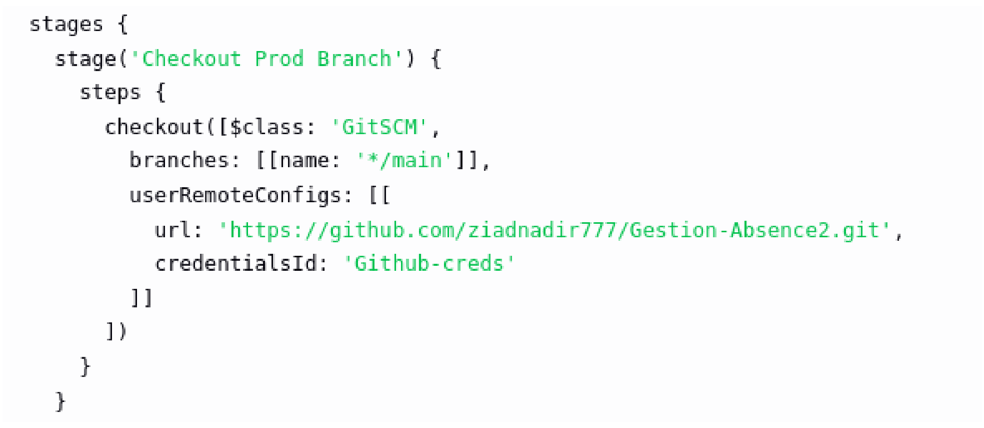
```
pipeline {
  agent any
  tools {
    nodejs 'node-18'
    dockerTool 'docker'
  }
  environment {
    IMAGE_NAME = "gestion-absence-backend"
    IMAGE_TAG = "${BUILD_NUMBER}"
    DB_HOST='${DB_NHOST}'
    DB_PORT='${DB_PORT}'
    DB_NAME='${DB_NAME}'
    DB_USER='${DB_USER}'
    DB_PASS='${DB_PASS}'
```

Figure 4: *Jenkins file basic setup.*

3.2 Integration with Github

The Jenkins pipeline integrates tightly with GitHub to enable continuous integration triggered by code changes. The Checkout Prod Branch stage uses the GitSCM plugin to clone the main branch of the repository located at https://github.com/ziadnadir777/Gestion-Absence2 authenticated via credentials stored in Jenkins (Github-creds).

Webhooks configured in GitHub trigger the pipeline automatically upon code pushes to the main branch, ensuring immediate validation of new commits. The multibranch pipeline dynamically detects branches and pull requests, creating corresponding Jenkins jobs to validate feature development. This integration provides real-time feedback to developers through GitHub status checks, ensuring that all code changes pass the CI pipeline before merging. The use of secure credentials and automated triggers supports a collaborative development environment while maintaining code quality

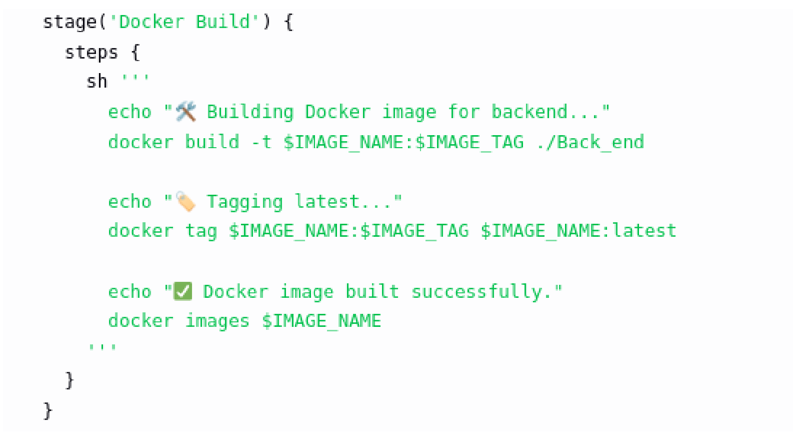


```
stages {
  stage('Checkout Prod Branch') {
    steps {
      checkout([$class: 'GitSCM',
        branches: [[name: '*/main']],
        userRemoteConfigs: [[
          url: 'https://github.com/ziadnadir777/Gestion-Absence2.git',
          credentialsId: 'Github-creds'
        ]]
      ])
    }
  }
```

Figure 5: *Github Integration.*

3.3 Automated build process

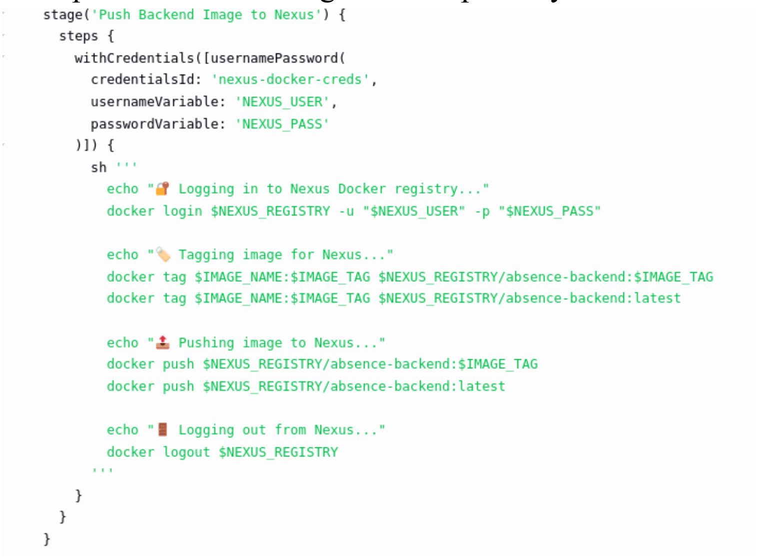
The automated build process is a critical component of the Jenkins pipeline, ensuring consistent and reproducible builds. The process begins with the **Docker Build** stage, where the backend application, located in the Back_end directory, is built into a Docker image tagged as **gestion-absence-backend:$BUILD_NUMBER** and **gestion-absence-backend:latest**.



```
stage('Docker Build') {
  steps {
    sh '''
      echo "🛠️ Building Docker image for backend..."
      docker build -t $IMAGE_NAME:$IMAGE_TAG ./Back_end

      echo "🏷️ Tagging latest..."
      docker tag $IMAGE_NAME:$IMAGE_TAG $IMAGE_NAME:latest

      echo "✅ Docker image built successfully."
      docker images $IMAGE_NAME
    '''
  }
}
```

Figure 6: *Docker build stage.*

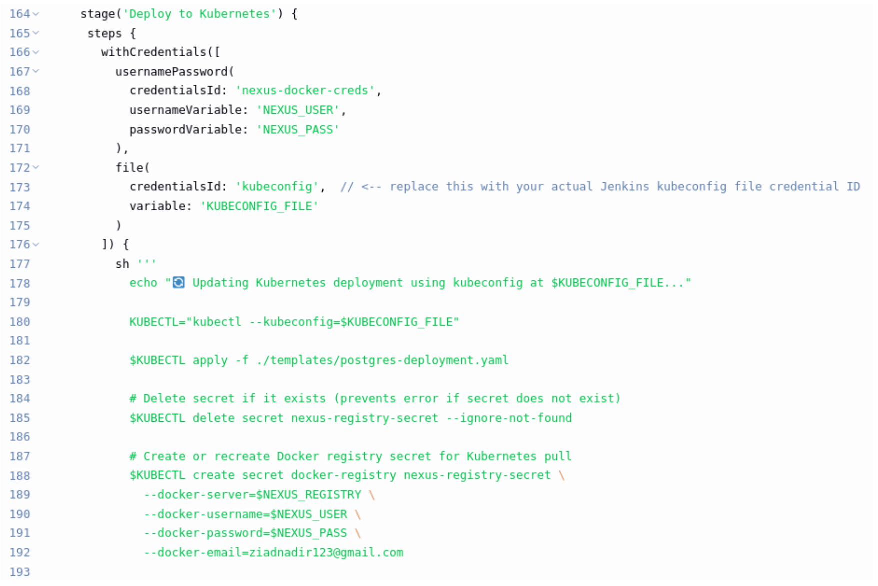
The docker build command compiles the Python/Flask backend into a containerized image, ensuring portability. The Push Backend Image to Nexus stage logs into the Nexus repository using credentials stored in Jenkins **(nexus-docker-creds)**, tags the image for Nexus, and pushes both the build-specific and latest tags to the repository.



```
stage('Push Backend Image to Nexus') {
  steps {
    withCredentials([usernamePassword(
      credentialsId: 'nexus-docker-creds',
      usernameVariable: 'NEXUS_USER',
      passwordVariable: 'NEXUS_PASS'
    )]) {
      sh '''
        echo "🔐 Logging in to Nexus Docker registry..."
        docker login $NEXUS_REGISTRY -u "$NEXUS_USER" -p "$NEXUS_PASS"

        echo "🏷️ Tagging image for Nexus..."
        docker tag $IMAGE_NAME:$IMAGE_TAG $NEXUS_REGISTRY/absence-backend:$IMAGE_TAG
        docker tag $IMAGE_NAME:$IMAGE_TAG $NEXUS_REGISTRY/absence-backend:latest

        echo "📤 Pushing image to Nexus..."
        docker push $NEXUS_REGISTRY/absence-backend:$IMAGE_TAG
        docker push $NEXUS_REGISTRY/absence-backend:latest

        echo "🚪 Logging out from Nexus..."
        docker logout $NEXUS_REGISTRY
      '''
    }
  }
}
```

Figure 7: *Push backend image to nexus stage .*

The Deploy to Kubernetes stage applies a PostgreSQL deployment configuration, creates a Docker registry secret (nexus-registry-secret), updates the backend deployment image, and ensures the rollout completes successfully using kubectl commands with a provided kubeconfig file. This process ensures that validated builds are seamlessly deployed to a Kubernetes cluster.



```
    stage('Deploy to Kubernetes') {
     steps {
       withCredentials([
         usernamePassword(
           credentialsId: 'nexus-docker-creds',
           usernameVariable: 'NEXUS_USER',
           passwordVariable: 'NEXUS_PASS'
         ),
         file(
           credentialsId: 'kubeconfig',  // <-- replace this with your actual Jenkins kubeconfig file credential ID
           variable: 'KUBECONFIG_FILE'
         )
       ]) {
         sh '''
           echo "🔄 Updating Kubernetes deployment using kubeconfig at $KUBECONFIG_FILE..."

           KUBECTL="kubectl --kubeconfig=$KUBECONFIG_FILE"

           $KUBECTL apply -f ./templates/postgres-deployment.yaml

           # Delete secret if it exists (prevents error if secret does not exist)
           $KUBECTL delete secret nexus-registry-secret --ignore-not-found

           # Create or recreate Docker registry secret for Kubernetes pull
           $KUBECTL create secret docker-registry nexus-registry-secret \
             --docker-server=$NEXUS_REGISTRY \
             --docker-username=$NEXUS_USER \
             --docker-password=$NEXUS_PASS \
             --docker-email=ziadnadir123@gmail.com

```

Figure 8: *Deploy to kubernetes stage.*

4 Advanced Testing Strategies

Introduction :

This section details the advanced testing strategies implemented to ensure the reliability, security, and quality of the attendance management application. The testing suite encompasses unit and integration tests, GitLeaks for secret detection, SonarQube for static code analysis, Dependency-Check for dependency vulnerabilities, Trivy for container image scanning, and OWASP ZAP for dynamic security testing. These tests are integrated into the Jenkins CI/CD pipeline, as defined in the Jenkinsfile, to automate validation and maintain high standards throughout the development lifecycle.

4.1 Unit and Integration Tests

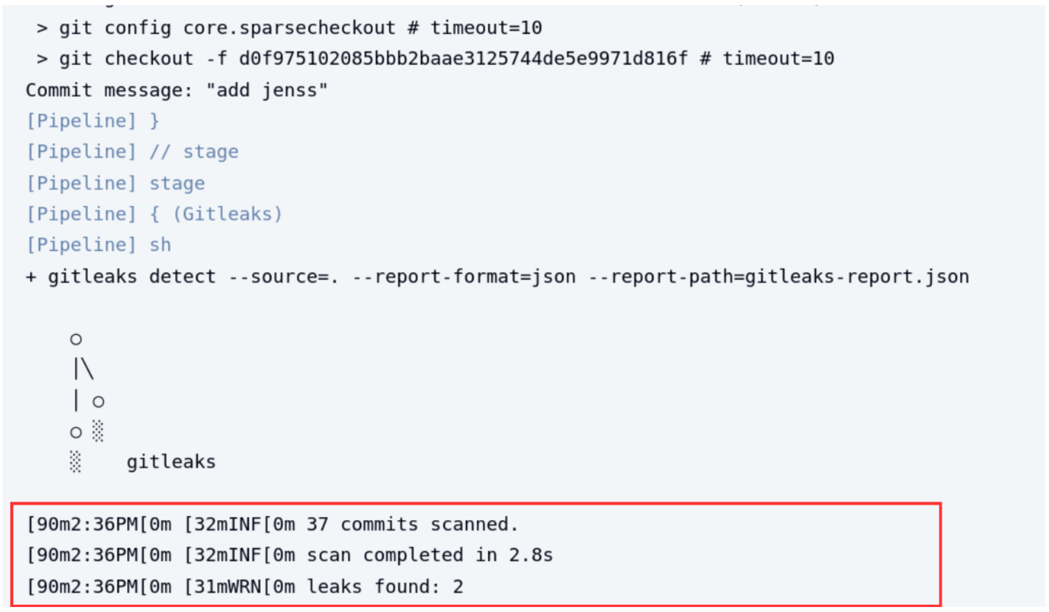
Unit and integration tests form the cornerstone of the application's testing strategy, ensuring that individual components and their interactions function correctly.
Unit tests, implemented using the pytest framework for the Python/Flask backend, validate discrete functions and modules, such as user authentication, QR code generation, and database operations.

These tests are executed within the Jenkins pipeline to identify issues early. Integration tests verify the interoperability between the React frontend, Flask backend, and PostgreSQL database, ensuring that API endpoints (e.g., attendance record retrieval) function seamlessly. For instance, integration tests confirm that the backend correctly processes requests from the frontend and updates the database accordingly. These tests run in isolated Docker containers, ensuring consistency across development and production environments.

4.2 GitLeaks Testing

GitLeaks testing is employed to identify and prevent the accidental inclusion of sensitive information, such as **API keys, credentials, or passwords**, in the GitHub repository. Integrated into the Jenkins pipeline, GitLeaks scans the source code in the Front_end and Back_end directories for patterns matching common secret formats. This ensures that sensitive data are not exposed in the repository.

GitLeaks generates reports highlighting any detected secrets, allowing developers to remediate issues before code is merged into the main branch. This step enhances the security of the application by preventing credential leaks.

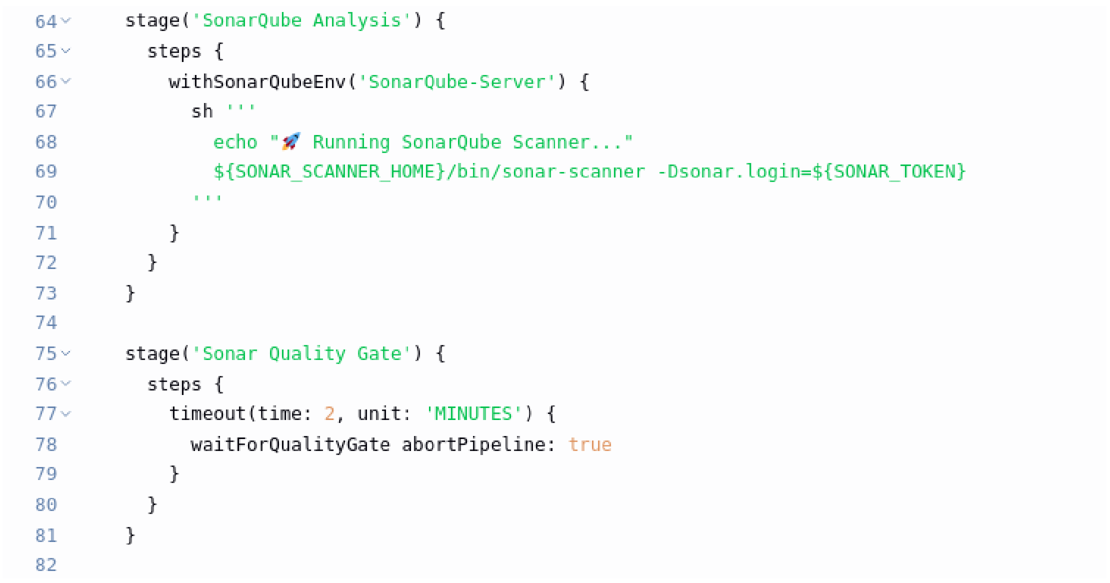


```
 > git config core.sparsecheckout # timeout=10
 > git checkout -f d0f975102085bbb2baae3125744de5e9971d816f # timeout=10
Commit message: "add jenss"
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Gitleaks)
[Pipeline] sh
+ gitleaks detect --source=. --report-format=json --report-path=gitleaks-report.json

    ○
    │╲
    │ ○
    ○ ░
    ░    gitleaks

[90m2:36PM[0m [32mINF[0m 37 commits scanned.
[90m2:36PM[0m [32mINF[0m scan completed in 2.8s
[90m2:36PM[0m [31mWRN[0m leaks found: 2
```

Figure 9: *GitLeak Test.*

4.3 SonarQube Testing

SonarQube is utilized for static code analysis to ensure code quality and identify security vulnerabilities in both the React frontend and Python/Flask backend. Integrated into the Jenkins pipeline(as indicated by the commented-out **SonarQube Analysis stage** in the Jenkinsfile.

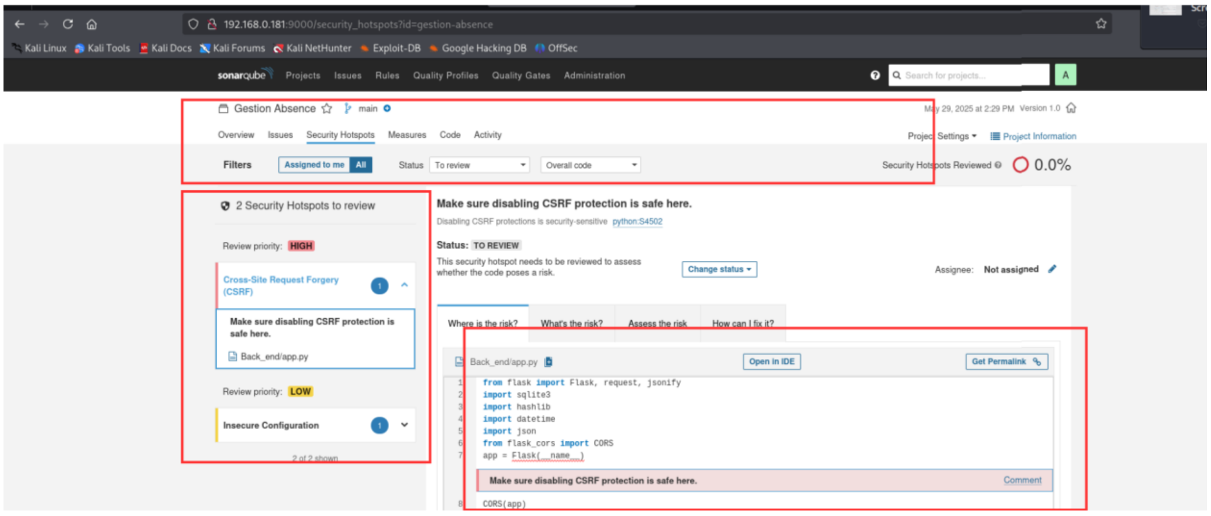


```
    stage('SonarQube Analysis') {
      steps {
        withSonarQubeEnv('SonarQube-Server') {
          sh '''
            echo "🚀 Running SonarQube Scanner..."
            ${SONAR_SCANNER_HOME}/bin/sonar-scanner -Dsonar.login=${SONAR_TOKEN}
          '''
        }
      }
    }

    stage('Sonar Quality Gate') {
      steps {
        timeout(time: 2, unit: 'MINUTES') {
          waitForQualityGate abortPipeline: true
        }
      }
    }

```

Figure 10: *SonarQube Analysis stage.*

Then SonarQube scans the codebase for bugs, code smells, and security hotspots, such as those reported at 192.168.0.18:9000/security.hotspots2=gestion-absence. The analysis evaluates metrics like code coverage, duplication, and adherence to coding standards.

A quality gate, configured with a two-minute timeout, ensures that only code meeting predefined quality thresholds proceeds to further stages. SonarQube's integration provides actionable insights, enabling developers to address issues early and maintain a robust codebase.



Make sure disabling CSRF protection is safe here.

Disabling CSRF protections is security-sensitive python:S4502

Status: TO REVIEW

This security hotspot needs to be reviewed to assess whether the code poses a risk.

```
from flask import Flask, request, jsonify
import sqlite3
import hashlib
import datetime
import json
from flask_cors import CORS
app = Flask(__name__)
```

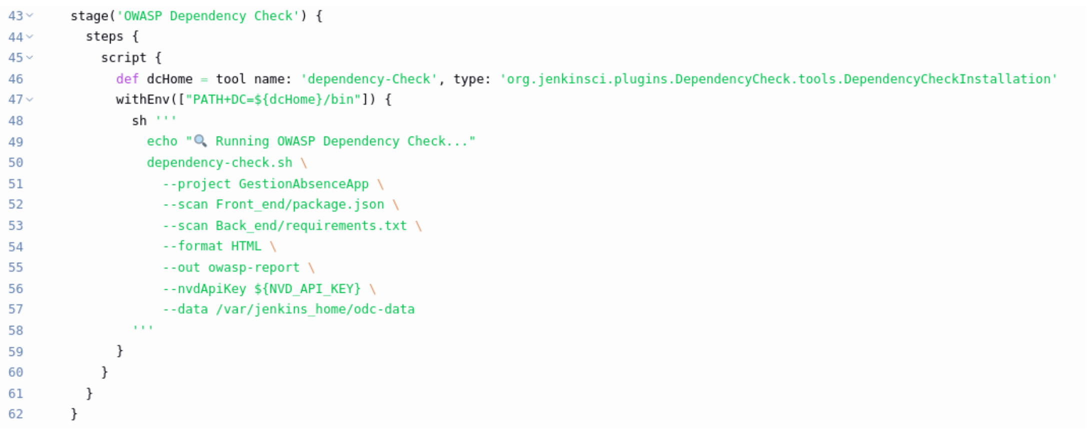
Make sure disabling CSRF protection is safe here.

```
CORS(app)
```

Figure 11: *SonarQube Test.*

4.3 Dependency-check Tests

Testing with Dependency-Check Dependency-Check, integrated into the Jenkins pipeline, scans the application's dependencies for known vulnerabilities.
For the frontend, it analyzes the package.json file in the Front_end directory, while for the backend, it examines the requirements.txt file in the Back_end directory.
Using the **NVD API key** (stored as nvd-api-key in Jenkins), Dependency-Check cross-references dependencies against the National Vulnerability Database, generating an HTML report stored in the owasp-report directory. This process ensures that libraries used in the React and Flask components are free from known exploits, enhancing the application's security posture.

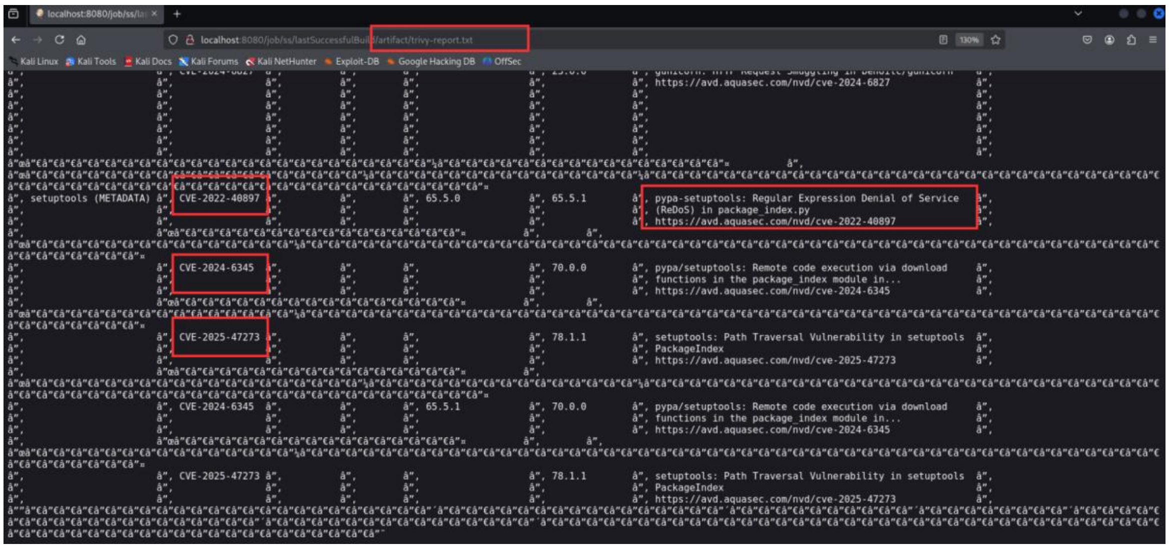


```
stage('OWASP Dependency Check') {
  steps {
    script {
      def dcHome = tool name: 'dependency-Check', type: 'org.jenkinsci.plugins.DependencyCheck.tools.DependencyCheckInstallation'
      withEnv(["PATH+DC=${dcHome}/bin"]) {
        sh '''
          echo "🔍 Running OWASP Dependency Check..."
          dependency-check.sh \
            --project GestionAbsenceApp \
            --scan Front_end/package.json \
            --scan Back_end/requirements.txt \
            --format HTML \
            --out owasp-report \
            --nvdApiKey ${NVD_API_KEY} \
            --data /var/jenkins_home/odc-data
        '''
      }
    }
  }
}
```

Figure 12: *Dependency-check.*

4.4 Trivy tests

Trivy is employed to scan **Docker images for vulnerabilities**, ensuring the security of the gestion-absence-backend image built in the Jenkins pipeline. Integrated as part of the CI/CD process, Trivy examines the Docker image for issues in its operating system and application dependencies before it is pushed to the Nexus repository . Trivy identifies vulnerabilities such as outdated packages or known CVEs, providing detailed reports to guide remediation. This step ensures that only secure images are deployed to the Kubernetes cluster, mitigating risks in the production environment.



```
localhost:8080/job/ss/lastSuccessfulBuild/artifact/trivy-report.txt

â”,                          â”, CVE-2024-6827 â”,         â”,         â”,                 â”, 23.0.0      â”, gunicorn: HTTP Request Smuggling in benoitc/gunicorn   â”,
â”,                          â”,               â”,         â”,         â”,                 â”,             â”, https://avd.aquasec.com/nvd/cve-2024-6827              â”,
â”, setuptools (METADATA) â”, CVE-2022-40897 â”,        â”,         â”, 65.5.0          â”, 65.5.1      â”, pypa-setuptools: Regular Expression Denial of Service  â”,
â”,                          â”,               â”,         â”,         â”,                 â”,             â”, (ReDoS) in package_index.py                            â”,
â”,                          â”,               â”,         â”,         â”,                 â”,             â”, https://avd.aquasec.com/nvd/cve-2022-40897             â”,
â”,                          â”, CVE-2024-6345 â”,         â”,         â”,                 â”, 70.0.0      â”, pypa/setuptools: Remote code execution via download    â”,
â”,                          â”,               â”,         â”,         â”,                 â”,             â”, functions in the package_index module in...            â”,
â”,                          â”,               â”,         â”,         â”,                 â”,             â”, https://avd.aquasec.com/nvd/cve-2024-6345              â”,
â”,                          â”, CVE-2025-47273 â”,        â”,         â”,                 â”, 78.1.1      â”, setuptools: Path Traversal Vulnerability in setuptools â”,
â”,                          â”,               â”,         â”,         â”,                 â”,             â”, PackageIndex                                           â”,
â”,                          â”,               â”,         â”,         â”,                 â”,             â”, https://avd.aquasec.com/nvd/cve-2025-47273             â”,
â”,                          â”, CVE-2024-6345 â”,         â”,         â”, 65.5.1          â”, 70.0.0      â”, pypa/setuptools: Remote code execution via download    â”,
â”,                          â”,               â”,         â”,         â”,                 â”,             â”, functions in the package_index module in...            â”,
â”,                          â”,               â”,         â”,         â”,                 â”,             â”, https://avd.aquasec.com/nvd/cve-2024-6345              â”,
â”,                          â”, CVE-2025-47273 â”,        â”,         â”,                 â”, 78.1.1      â”, setuptools: Path Traversal Vulnerability in setuptools â”,
â”,                          â”,               â”,         â”,         â”,                 â”,             â”, PackageIndex                                           â”,
â”,                          â”,               â”,         â”,         â”,                 â”,             â”, https://avd.aquasec.com/nvd/cve-2025-47273             â”,
```

Figure 13 : *Trivy report.*

Conclusion

Advanced testing and security tools like pytest, GitLeaks, SonarQube, Dependency-Check, and Trivy ensure the attendance app is robust, secure, and high-quality. Integrated into the Jenkins CI/CD pipeline, they validate functionality, protect data, and secure deployments throughout development.

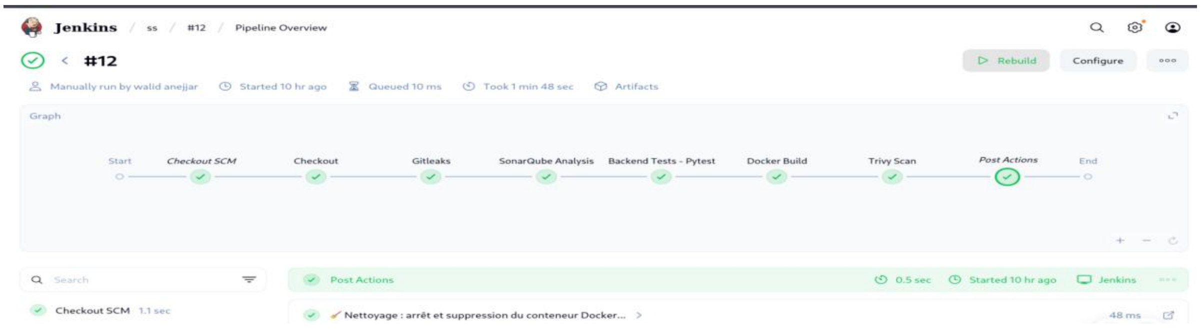


Figure 14 : *Final result after tests.*

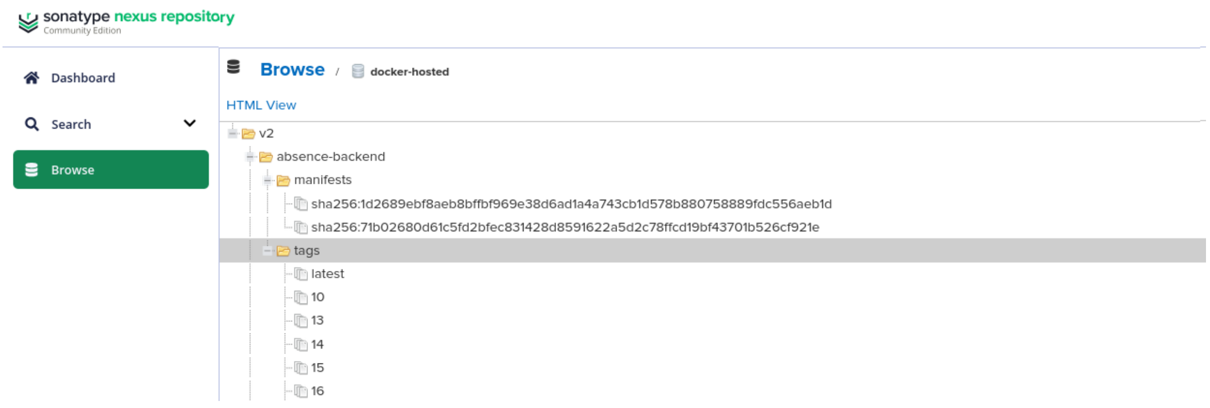
5 Deployment with Kubernetes

Introduction :

This section details the deployment phase of the attendance management application, focusing on the automated process of retrieving artifacts from the Nexus repository and deploying them using Kubernetes. The deployment strategy, integrated into the Jenkins CI/CD pipeline as defined in the Jenkinsfile, ensures scalability, reliability, and high availability. The application leverages Docker for containerization, Kubernetes for orchestration, and NGINX for load balancing, with configurations designed to support multiple replicas for enhanced availability.

5.1 Artifact Retrieval from Nexus

The deployment phase begins with the retrieval of containerized artifacts from the Nexus repository. The Jenkins pipeline, specifically the Push Backend Image to Nexus stage, builds the backend Docker image **(gestion-absence-backend)** and tags it with the build number ($BUILD_NUMBER) and a latest tag. Using secure credentials (nexus-docker-creds), the pipeline logs into the Nexus registry, tags the image as 10.0.2.15:8082/absence-backend:$BUILD_NUMBER, and pushes it to the repository.
This process ensures that validated, versioned artifacts are stored securely and are readily accessible for deployment. The Nexus repository acts as a centralized artifact management system, enabling traceability and version control for all deployed images.

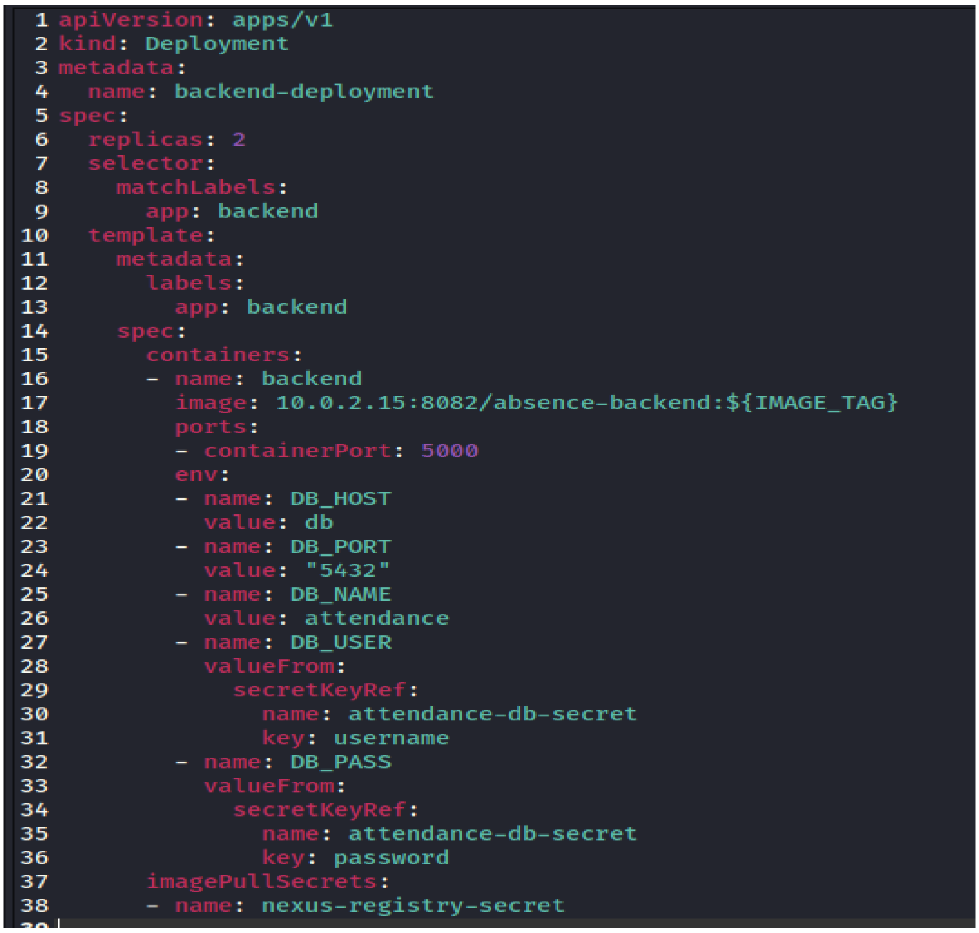


```
v2
  absence-backend
    manifests
      sha256:1d2689ebf8aeb8bffbf969e38d6ad1a4a743cb1d578b880758889fdc556aeb1d
      sha256:71b02680d61c5fd2bfec831428d8591622a5d2c78ffcd19bf43701b526cf921e
    tags
      latest
      10
      13
      14
      15
      16
```

Figure 15 : *Artifacts in Nexus.*

5.2 Kubernetes Deployment Configuration

Kubernetes orchestrates the deployment of the application using the configuration defined in the **backend-deployment.yaml** file. The deployment, named backend-deployment, specifies two replicas to ensure high availability and load distribution. The configuration defines a pod template with a single container (backend) running the 10.0.2.15:8082/absence-backenimage, exposing port 5000 for the Flask backend. Environment variables (DB_HOST, DB_PORT, DB_NAME) are set to connect to the PostgreSQL database (db:5432, attendance), with sensitive credentials (DB_USER, DB_PASS) sourced from a Kubernetes secret **(attendance-db-secret)**The imagePullSecrets field references nexus-registry-secret, created in the Jenkins pipeline, to authenticate image pulls from Nexus. The Deploy to Kubernetes stage in the Jenkinsfile applies this configuration, updates the deployment image, and verifies the rollout status using kubectl commands.



```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: backend-deployment
spec:
  replicas: 2
  selector:
    matchLabels:
      app: backend
  template:
    metadata:
      labels:
        app: backend
    spec:
      containers:
      - name: backend
        image: 10.0.2.15:8082/absence-backend:${IMAGE_TAG}
        ports:
        - containerPort: 5000
        env:
        - name: DB_HOST
          value: db
        - name: DB_PORT
          value: "5432"
        - name: DB_NAME
          value: attendance
        - name: DB_USER
          valueFrom:
            secretKeyRef:
              name: attendance-db-secret
              key: username
        - name: DB_PASS
          valueFrom:
            secretKeyRef:
              name: attendance-db-secret
              key: password
      imagePullSecrets:
      - name: nexus-registry-secret
```

Figure 16 : *Kubernetes Deployment Configuration.*

5.3 High Availability and Scalability

The application is designed for high availability and scalability, leveraging Kubernetes' orchestration capabilities and **NGINX** as a load balancer.

The backend-deployment configuration specifies two replicas, ensuring that at least two instances of the backend container are running across the Kubernetes cluster. This setup provides fault tolerance, as the failure of one pod does not disrupt service, with Kubernetes automatically rescheduling failed pods. The NGINX load balancer, configured with

SSL/TLS, distributes incoming traffic across the replicas, optimizing performance under high load.
 Additionally, **ModSecurity** is integrated as a Web Application Firewall (WAF) within the NGINX configuration, providing an extra layer of security by filtering and monitoring HTTP traffic to detect and block potential threats, such as SQL injection or cross-site scripting (XSS) attacks.

Kubernetes horizontal pod autoscaling can further adjust the number of replicas based on resource usage, ensuring scalability for increased user demand, such as during peak attendance tracking periods. This architecture guarantees minimal downtime, robust performance, and enhanced security for both professor and student interfaces.

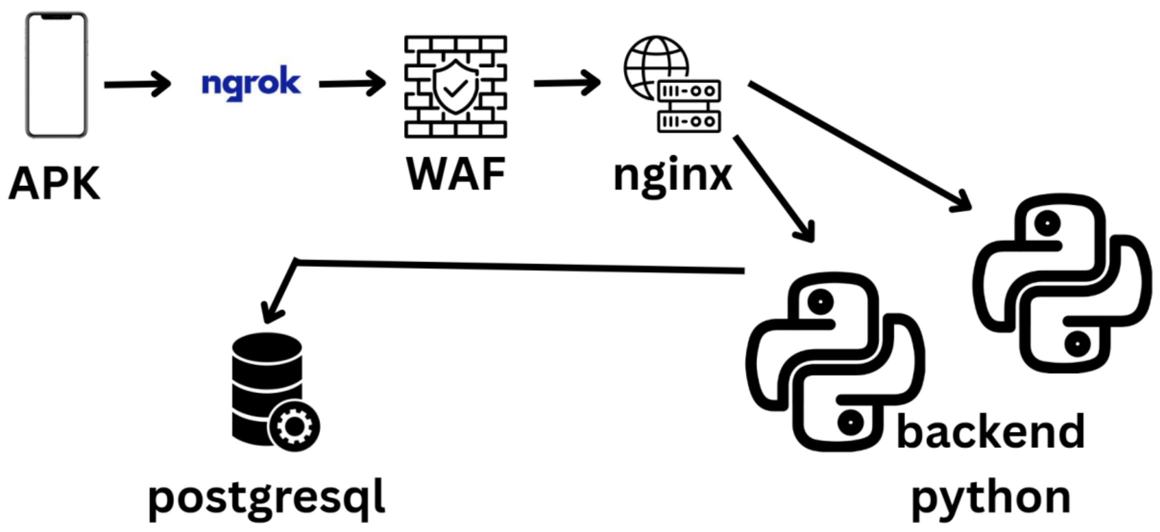


Figure 17 : *High Availability and Scalability Architecture.*

Conclusion

The development of the absence management application demonstrates a successful implementation of DevSecOps principles, delivering a secure, scalable, and efficient solution for academic institutions.

The application, built with a React frontend and Python/Flask backend, provides intuitive interfaces for professors and students to manage attendance through QR code-based tracking, supported by a robust PostgreSQL database.

The Jenkins CI/CD pipeline automates code validation, testing, and deployment, integrating tools like pytest, GitLeaks, SonarQube, Dependency-Check, and Trivy to ensure code quality and security. Docker containerization and Kubernetes orchestration, combined with NGINX load balancing and ModSecurity as a WAF, enable high availability and scalability, with multiple replicas ensuring fault tolerance.

This project not only meets the immediate needs of attendance management but also establishes a modular, extensible framework for future enhancements, such as integration with broader academic systems or additional features. By prioritizing security, automation, and collaboration, this DevSecOps approach sets a strong foundation for agile and resilient software development.

The End