Project Report: *Open-Source Incident Management System (IMS)*

1. Abstract

This project focuses on developing an Open-Source Incident Management System (IMS) that helps organizations efficiently log, track, and resolve application or infrastructure related issues. The system is built using Python and the Flask framework, offering both a user-friendly web interface and powerful RESTful APIs for backend operations.

Key features include role-based access control (Manager, Technician, Reporter), persistent data storage using SQLite, and asynchronous email notifications for critical updates. The entire application is containerized with Docker, making it easy to deploy and ensuring consistency across different environments.

2. Introduction

The main goal of this project is to build a centralized and efficient platform for managing the entire lifecycle of an incident from the time it is reported to its final resolution.

It helps teams collaborate effectively by assigning roles and sending real-time email updates, which in turn reduces response time, improves transparency, and enhances accountability during technical outages or service disruptions.

3. Tools and Technologies Used

Category	Tools
Backend Framework	Python (Flask)
Database	SQLite (via Flask-SQLAlchemy)
Authentication & Persistence	Flask-SQLAlchemy (ORM), Flask-Login, Flask-Mail
Frontend	HTML, Bootstrap 5, Tailwind CSS
Containerization	Docker
Version Control	Git & GitHub

4. Steps involved

The development followed an agile, phase-wise approach to ensure that each part of the system was built, tested, and integrated efficiently.

Phase 1: Environment and Setup

- Created the project structure and virtual environment.
- Installed all required Python dependencies using a requirements.txt file.
- Verified the Flask environment and basic app routing.

Phase 2: Application Logic and Data Modeling

- Designed database models for User and Incident using Flask-SQLAlchemy.
- Implemented Flask routes for user authentication (login/logout) and CRUD operations on incidents.
- Introduced role-based authorization, allowing Managers and Technicians to assign, update, or resolve incidents, while Reporters could only raise them.

Phase 3: Database Initialization and Email Integration

- Added a custom Flask CLI command flask initdb to initialize the database and insert sample users (admin, tech_alice, reporter_bob).
- Configured Flask-Mail with SMTP credentials for email notifications.
- Implemented asynchronous threading to send emails in the background, ensuring the app remains responsive.
- Set up notifications for both new incidents (to alert managers- admin) and status updates (to inform the reporter or technician).

Phase 4: Frontend and Containerization

- Designed a responsive, mobile-friendly web interface using Bootstrap 5 and Tailwind CSS.
- Developed a unified template for both the login page and the incident dashboard.
- Created a Dockerfile to containerize the application, bundling the environment, dependencies, and pre-initialized database.
- Pushed the complete source code and configuration to GitHub for version control and open-source collaboration.

Phase 5: Testing and Demonstration

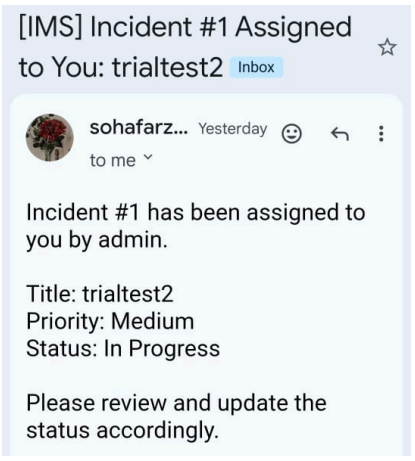
- Built and ran the Docker image successfully, verifying it worked independently of the host environment.
- Simulated a full incident workflow from creation to closure using different roles.
- Confirmed that email alerts were sent correctly at every stage.

5. Conclusion

The Open-Source Incident Management System (IMS) achieved its objective of providing a complete, secure, and easily deployable solution for handling incidents. Its features including role-based access, persistent data storage, real-time notifications, and Docker containerization make it suitable for production environments and team-based Devops operations.

Admin's inbox:

Technician Alice inbox:



Reporter Bob's inbox: