Advanced Offshore Oil Platform Management System

Table of Contents

Introduction

Offshore oil platforms are a fundamental part of the energy industry, used to extract oil and gas from beneath seas and oceans. These operations require precise monitoring, quick response, and intelligent control to prevent risks and ensure operational efficiency. This project aims to develop an integrated software system using Artificial Intelligence and the Internet of Things (IoT) to manage platform operations efficiently, enhance safety, and intelligently control equipment.

1. Software Process Activities

1.1 Planning

- Feasibility Analysis: Technical feasibility to define hardware/software requirements, and economic feasibility to assess cost-benefit.
- Risk Management: Covers communication risks, power outages, sensor malfunctions.
- Resource Allocation: Defines developers, devices, and required servers.

1.2 Requirements Analysis

- Conducting workshops with marine operation engineers to understand real needs.
- Analyzing previous systems to identify strengths and weaknesses.

1.3 Design

- Architectural Design: Using Microservices for system segmentation.
- Database Design: Choosing PostgreSQL for structured data and MongoDB for unstructured data.

1.4 Implementation

- Backend: Python with Django for advanced management and security.
- Frontend: ReactJS to offer a smooth and interactive user experience.

1.5 Testing

- Unit Testing with PyTest.
- Integration Testing simulating data flow from sensors.

1.6 Deployment

- Using Docker containers and Kubernetes for cloud orchestration.

1.7 Maintenance

- Security patches, feature updates based on user feedback.

2. Application Types

Application Type	Description	Technologies Used
Web Application	Dashboard & monitoring	Django, ReactJS
Mobile App	Real-time data monitoring	Flutter, Firebase
Embedded System	Direct pump control	Raspberry Pi, C++
Real-Time System	Sensor and pressure monitoring	Python, MQTT Protocol

3. Architecture

3.1 Multi-Tier Architecture

1. UI Layer: React - notifications and charts.
2. Application Layer: Data analysis, device control.
3. Database Layer: PostgreSQL for operations, MongoDB for unstructured logs.
4. Communication: WebSocket and MQTT for IoT.

3.2 Microservices Architecture

1. User Management Service
2. Pressure/Temperature Monitoring Service
3. Risk Management Service
4. Each service operates independently and is scalable.

4. Activity Model

4.1 Data Management

- Data collection from IoT sensors.

4.2 Risk Management

- Automatic leak detection using AI.
- Real-time alert generation.
- Auto shut-off valves.

5. High-Level System Requirements

Requirement	Description
Security	Emergency auto shutdown system
Reliability	99.99% uptime
Integration	SCADA and Open APIs support

6. Objectives

1. Reduce system failures by 40%.
2. Cut operational costs by 15%.
3. Improve energy efficiency by 25%.
4. Speed up emergency response time to under 3 seconds.

7. Plan-Driven vs Agile Development Plan

Criterion	Plan-Driven	Agile (Scrum)
Flexibility	Inflexible	Flexible and adaptive
Documentation	Detailed	As needed
Best for	Fixed-scope projects	Changing requirements

8. Software Reuse

1. TensorFlow libraries for fault prediction.
2. Use of PostgreSQL and MongoDB as ready-made components.
3. Open communication protocols (MQTT, WebSocket).

9. Software Specifications

9.1 Functional Requirements

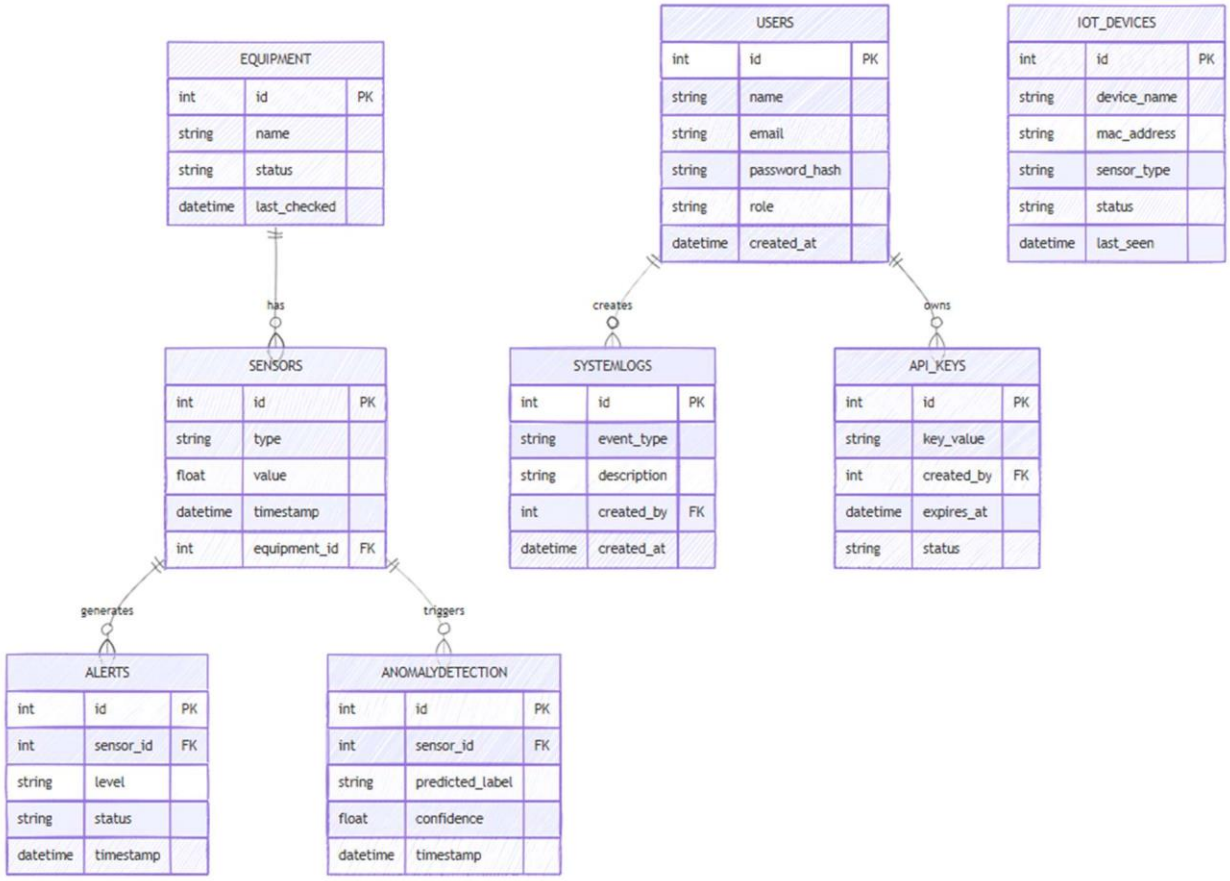
1. Pressure and temperature monitoring.
2. Event management.
3. Real-time alerts.
4. Valve shutdown in emergencies.

9.2 Non-Functional Requirements

1. Response time < 0.5 seconds.
2. AES-256 encrypted communication.
3. Temporary offline operation support.

10. Design Activities

10.1 Database Design



10.2 Algorithm Design

- Leak Detection Algorithm: Random Forest Algorithm for emergency classification.

11. Prototyping

- Dashboard design using Figma.
- Use of mock APIs for UI testing.

12. Technical, Human, and Organizational Issues

Type	Problem	Proposed Solution
Technical	Signal delay	High-speed satellite networks
Human	Resistance to new system	Gradual training for operators
Organizational	Standards compatibility	Use of international protocols/specs

13. User Requirements

- User-friendly interface for operators.
- Periodic reports for management.
- Visual and audible emergency notifications.

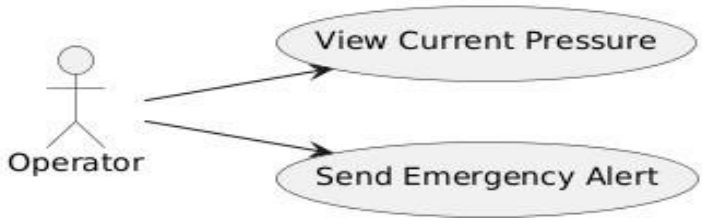
14. System Requirements

Component	Minimum Requirement
Server	16GB RAM, 8-Core CPU
Operating System	Linux Ubuntu 20.04+
Database	PostgreSQL 12+, MongoDB 5+

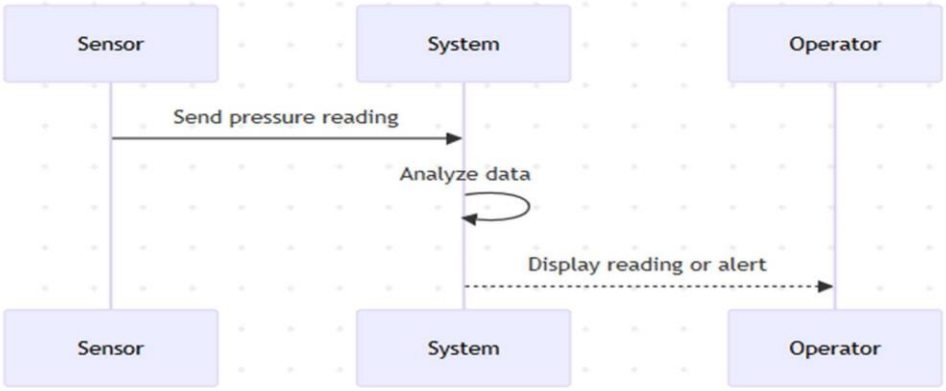
15. Functional & Non-Functional Requirements

Function	Non-Functional Requirement
Pressure Monitoring	Response within 0.5 seconds
Leak Detection	99.7% accuracy
User Management	High-level encryption (AES-256)

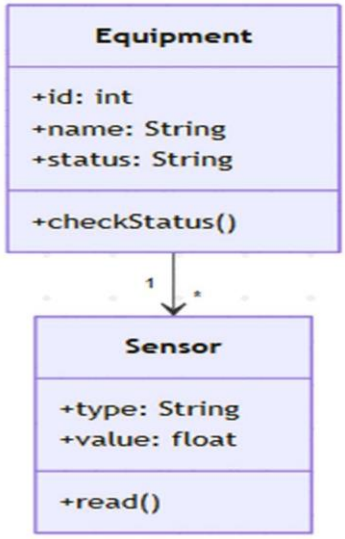
16. Use Case Diagram – Pressure Monitoring



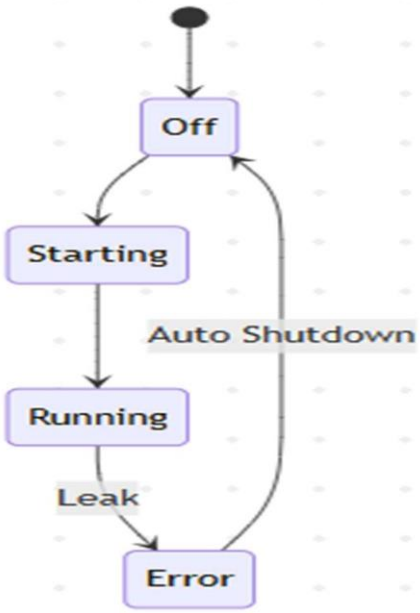
17. Sequence Diagram – Pressure Reading



18. Class Diagram – Equipment Management



19. State Diagram – Pump State



References

- Sommerville, Ian. *Software Engineering*, 10th Edition.
- IEEE Standards for Offshore Software Systems.
- OPEC Offshore Reports (2023).
- Documentation: PostgreSQL, MongoDB, MQTT, TensorFlow.

- Very Thanks for other students :
  - Khalid Waleed Althomali
  - Ahmad Saad Alharthi
  - Fares Omar Manshawi
  - Bilal Yousef Alsinni
  - Mohammad Anas Alkhotani
  - Abdulrahman Hamad Alharthi
  - Anas Sami Alharthi
  - Muqbil Muslat Alsubaie
  - Jawad Bilal Alqurashi
  - Hasan Majed Alqurashi
  - Nawaf Faisal Alotaibi

Coding by :

- Saad Saeed Almalki