TED UNIVERSITY

CMPE492/ SENG492

<safeSCOPE>

Low Level Design Report

Team Members:

Arda BARAN 19172802022 Computer Engineering

Yakup Mert AKAN 15574008550 Software Engineering

Baran KUZUCANLI 36529547832 Computer Engineering

Sena ÖZTÜRK 19750960502 Computer Engineering

Supervisor: Ali BERKOL

Jury Members: Tolga Kurtuluş ÇAPIN, Emin KUĞU

Table Of Contents

1. Introduction 4
    1.0.1 Key Objectives of the Low-Level Design 4
    1.0.2 Importance of Low-Level Design in Software Development....4
    1.0.3 Relationship Between High-Level and Low-Level Design 4
  1.1 Object Design Trade-offs 5
    1.1.1 Extensibility vs. Simplicity 5
    1.1.2 Performance vs. Maintainability 5
    1.1.3 Coupling vs. Cohesion 6
    1.1.4 Object-Oriented vs. Data-Oriented Design 6
  1.2 Interface Documentation Guidelines 6
    1.2.1 REST API Standards 6
    1.2.2 WebSocket Communication 7
    1.2.3 Subsystem Interaction 7
    1.2.4 Frontend API Integration 7
    1.2.5 Database Interface Standards 7
  1.3 Engineering Standards (e.g., UML and IEEE) 8
    1.3.1 UML Standards 8
    1.3.2 IEEE Software Engineering Standards 8
    1.3.3 Security Standards 8
  1.4 Definitions, Acronyms, and Abbreviations 9
2. Packages 9
  2.1 OpenCV (cv2) 10
  2.2 NumPy (numpy) 10
  2.3 Flask 10
  2.4 Firebase Admin SDK (firebase_admin) 11
  2.5 Threading (threading) 11
  2.6 Time (time) 11
3. Class Interfaces 12
  3.1. Main Class 12
  3.2. YoloInference Class 12
  3.3. FirebaseConfig Class 12
  3.4. Routes Class 12
  3.5. PPEDetection Class 13
  3.6. ImageProcessing Class 13

3.7 Attributes 14
4. Glossary 15
4.1 General Software Engineering Terms 15
4.2 Artificial Intelligence and Machine Learning Terms 16
4.3 System Components and Technologies 17
4.4 Safety and Workplace Compliance Terms 17
4.5 Security and Access Control Terms 18
4.6 Software Development and Deployment Terms 19
5- References 19

1. Introduction

The Low-Level Design (LLD) of safeScope refines the system's high-level architecture into a detailed blueprint for implementation. This document provides a comprehensive view of the internal structures, including class definitions, interface specifications, trade-offs in object design, and adherence to engineering standards.

The purpose of this document is to ensure that all stakeholders, including developers, system architects, and quality assurance teams, have a unified and comprehensive understanding of the software components, their interactions, and the rationale behind design decisions. The LLD serves as a roadmap that bridges the conceptual framework outlined in the high-level design (HLD) with the actual code implementation, ensuring that development follows a well-structured and scalable approach.

1.0.1 Key Objectives of the Low-Level Design

The key objectives of this document include:

- Defining the architecture of safeScope at a granular level, including subsystem interactions and dependencies.
- Providing a structured breakdown of class hierarchies, APIs, and data flow.
- Ensuring modularity and reusability by implementing best practices in object-oriented programming.
- Optimizing performance by making strategic design trade-offs between efficiency and maintainability.
- Enhancing security and reliability by adhering to industry standards such as IEEE and OWASP best practices.

1.0.2 Importance of Low-Level Design in Software Development

A well-defined low-level design is critical for achieving the following software development goals:

- **Code Consistency**: Ensuring uniform design patterns across all modules, reducing redundancy, and improving maintainability.
- **Scalability**: Designing the system in a way that accommodates future enhancements without requiring extensive rework.
- **Performance Optimization**: Addressing potential bottlenecks at the design stage to improve efficiency and responsiveness.
- **Security and Compliance**: Incorporating security protocols, encryption methods, and compliance guidelines from the outset to safeguard data and system integrity.
- **Traceability and Documentation**: Establishing clear links between system requirements, architectural decisions, and code implementation.

1.0.3 Relationship Between High-Level and Low-Level Design

The HLD focuses on the macro-level architecture, defining system modules, major functionalities, and technology choices. In contrast, the LLD delves deeper into:

- **Detailed Class and Object Descriptions**: Specifying attributes, methods, and their relationships.
- **Database Schema and Data Flow**: Defining how data is stored, retrieved, and processed.
- **Component-Level Interfaces**: Documenting APIs, data formats, and inter-component communication protocols.
- **Algorithmic Implementation**: Providing step-by-step execution logic for core system functionalities.

Through object-oriented design principles, the LLD ensures that safeScope maintains its modularity, maintainability, and extensibility while leveraging modern software engineering standards such as UML, IEEE guidelines, and object-oriented best practices.

This document serves as a bridge between the high-level system design and actual implementation by refining application objects, incorporating solution objects, and defining precise subsystem interfaces and components. A structured approach in LLD minimizes development risks, improves collaboration, and ensures a smooth transition from design to deployment.

1.1 Object Design Trade-offs

Object design trade-offs arise when making critical decisions about how the system's components interact, how flexible and reusable the design should be, and how to optimize performance while maintaining code quality. The design of safeScope reflects careful consideration of various software engineering principles, ensuring that maintainability, performance, modularity, and extensibility are balanced effectively.

Choosing the right object-oriented design (OOD) principles requires trade-offs in different aspects of system design. While some principles enhance system flexibility, they may also introduce complexity and performance overhead. Others prioritize efficiency and speed, but may limit future extensibility. The following sections discuss these trade-offs in detail, explaining how safeScope navigates these challenges.

The design decisions made in safeScope reflect careful consideration of these factors:

1.1.1 Extensibility vs. Simplicity

- The system is designed to support additional PPE types or safety rules without requiring major changes to existing modules.
- A highly flexible and modular structure ensures new computer vision models or additional proximity monitoring sensors can be easily integrated.
- A microservices-based approach is used to facilitate independent module updates without breaking the entire system.
- However, introducing excessive flexibility may lead to increased code complexity and performance overhead.

1.1.2 Performance vs. Maintainability

- The YOLO object detection model is computationally efficient, allowing real-time PPE detection.
- Lightweight pre-processing techniques (e.g., OpenCV for bounding box visualization) are used to ensure fast responses.
- However, real-time processing may require GPU acceleration, which can add hardware dependency constraints.
- Code modularization ensures maintainability by breaking down functionalities into well-defined service layers.

1.1.3 Coupling vs. Cohesion

- Low coupling between detection services (PPE detection and proximity monitoring) ensures changes in one module do not affect others.
- High cohesion within each module means that each class and function has a clear, single responsibility, making debugging easier.
- However, excessive separation of components can increase data exchange latency and complexity in coordinating subsystems.
- Well-defined APIs between subsystems help mitigate interdependency issues.

1.1.4 Object-Oriented vs. Data-Oriented Design

- The system follows OOP principles by using well-defined classes for detection models, database interactions, and API services.
- However, certain parts (e.g., real-time inference with YOLO) benefit from a data-oriented approach where raw image processing is prioritized over object encapsulation for speed.

1.2 Interface Documentation Guidelines

Interfaces define how components interact within the system, ensuring clear communication protocols, input/output specifications, and modular interactions. The following guidelines outline best practices for safeScope’s API and subsystem interfaces:

1.2.1 REST API Standards

- **Backend-to-Frontend Communication:** Uses RESTful APIs and WebSockets for real-time updates.
- **HTTP Methods:**
  - **GET** → Retrieve PPE compliance data, logs, and alerts.
  - **POST** → Submit real-time detection results from YOLO processing modules.
  - **PUT** → Update safety policies and PPE compliance rules.
  - **DELETE** → Remove outdated compliance logs.
- **JSON format** is used for request/response payloads.

1.2.2 WebSocket Communication

- WebSockets handle real-time PPE detection results and worker proximity alerts.
- The frontend listens to alert events and updates the dashboard UI instantly.

1.2.3 Subsystem Interaction

Each **service (PPE detection, proximity monitoring, alert management, and frontend UI)** has clearly defined interfaces:

Request:

```json
{
  "image": "base64-encoded-image",
  "model_version": "v3.0"
}
```

5. **Response:**

```json
{
  "detections": [
    {"label": "Helmet", "confidence": 0.98, "bbox": [x1, y1, x2, y2]},
    {"label": "Vest", "confidence": 0.95, "bbox": [x1, y1, x2, y2]}
  ]
}
```

Request:

```json
{
  "worker_positions": [
    {"id": "worker_1", "x": 10, "y": 20},
    {"id": "worker_2", "x": 15, "y": 25}
  ],
  "machine_positions": [
    {"id": "excavator_1", "x": 12, "y": 22}
  ]
}
```

10. **Response:**

```json
{
  "alerts": [
    {"worker_id": "worker_1", "risk_level": "HIGH", "distance": "2.5m"}
  ]
}
```

1.2.4 Frontend API Integration

- The frontend dynamically fetches compliance data using AJAX calls to REST APIs.
- WebSocket events trigger UI updates, ensuring real-time visibility.

1.2.5 Database Interface Standards

- The NoSQL Firebase database follows a structured document-oriented model.
- Data is indexed for fast retrieval, and role-based access ensures data security.

1.3 Engineering Standards (e.g., UML and IEEE)

safeScope follows established engineering standards to ensure scalability, maintainability, and clarity in design:

1.3.1 UML Standards

The Unified Modeling Language (UML) is used to visualize system interactions and class relationships:

- **Use Case Diagrams** → Define system functionalities and user interactions.
- **Class Diagrams** → Represent the object hierarchy and relationships.
- **Sequence Diagrams** → Describe request-response interactions between subsystems.

1.3.2 IEEE Software Engineering Standards

safeScope aligns with IEEE Std 1016-2009 (Software Design Description) and IEEE Std 830-1998 (Software Requirements Specification):

- IEEE 1016-2009: Defines structured modular design, component relationships, and interface definitions.
- IEEE 830-1998: Ensures clear requirements traceability from high-level design to low-level implementation.

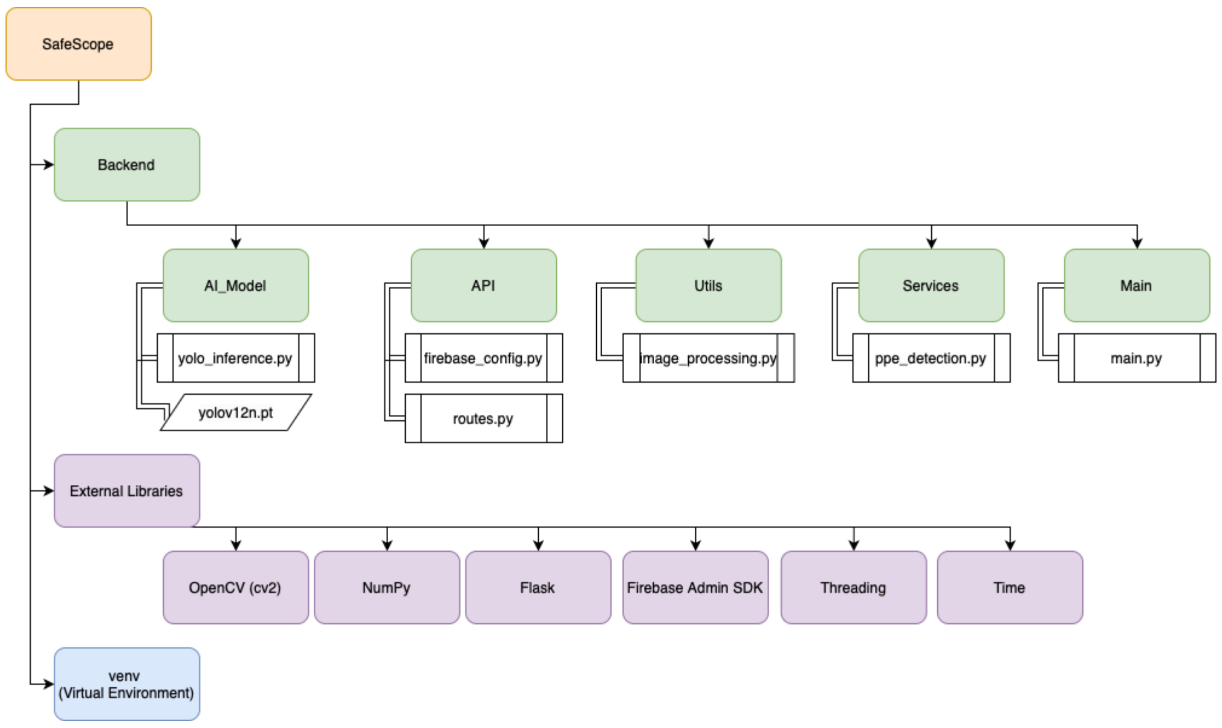
1.3.3 Security Standards

- OWASP Best Practices for securing APIs and preventing unauthorized access.
- TLS 1.3 Encryption for secure data transmission between components.
- Role-Based Access Control (RBAC) to restrict sensitive operations.

1.4 Definitions, Acronyms, and Abbreviations

Term	Definition
AI	Artificial Intelligence
PPE	Personal Protective Equipment
YOLO	You Only Look Once (Object Detection Algorithm)
REST API	Representational State Transfer Application Programming Interface
WebSocket	A communication protocol for real-time data exchange
RBAC	Role-Based Access Control
TLS	Transport Layer Security
UML	Unified Modeling Language
IEEE	Institute of Electrical and Electronics Engineers
JSON	JavaScript Object Notation (Data exchange format)

2. Packages



2.1 OpenCV (cv2)

• **Description**: OpenCV is a popular open-source library for computer vision and image processing tasks.

• Functions:

Capturing and processing video frames

Image manipulation (resizing, filtering, etc.)

Feature extraction

Object detection using AI models

• **Usage**:

Used in PPE detection and Proximity detection for processing frames.

Helps in drawing bounding boxes on detected objects in the video_feed endpoint.

2.2 NumPy (numpy)

• **Description**: NumPy is a library for numerical computing and array processing in Python.

• Functions:

Creating and managing arrays and matrices

Performing mathematical operations efficiently

Handling large datasets for AI models

• **Usage**:

Used in YOLO inference processing for handling detection outputs.

Converts image data into NumPy arrays for efficient processing.

2.3 Flask

• **Description**: Flask is a lightweight web framework used for creating REST APIs in Python.

• **Functions**:

Handles HTTP requests and responses

Provides REST API endpoints for PPE detection and video streaming

Manages communication between AI model and frontend

• **Usage**:

Used to create REST API endpoints such as /detect_ppe, /send_to_firebase, and /video_feed.

Manages the server-side logic for SafeScope.

2.4 Firebase Admin SDK (firebase_admin)

• **Description**: Firebase Admin SDK allows secure communication with Firebase Realtime Database.

• **Functions**:

Sending and retrieving detection data from Firebase

Authenticating and managing database access

Storing and logging PPE detection results

• **Usage**:

Used in send_data_to_firebase function to store detected PPE compliance data.

Ensures that logs are maintained for compliance tracking.

2.5 Threading (threading)

• **Description**: The Threading module in Python allows for parallel execution of tasks.

• **Functions**:

Running background tasks while the Flask server is active

Handling multiple simultaneous detections

Improves system efficiency

• **Usage**:

Used to send detection data to Firebase while simultaneously running the API.

Runs the send_data_to_firebase function in a separate thread.

2.6 Time (time)

•**Description**: Time module is used for handling timestamps and delays in Python.

• **Functions**:

Recording timestamps for detected PPE data

Adding delays between detections to optimize performance

• **Usage**:

Used in send_data_to_firebase to store timestamps for each detection.

Used for adding a delay before each new detection cycle.

3. Class Interfaces

3.1. Main Class

The Main class is responsible for starting the Flask API server and running PPE detection as a background process.It initializes and registers API routes, ensuring that real-time detection and REST API functionality work simultaneously.

Methods:

register_routes(): Registers the Flask API routes using routes.py.

start_ppe_detection(): Starts the **PPE detection service** as a background thread.

start_api(): Starts the **Flask API server** to handle HTTP requests.

3.2. YoloInference Class

YoloInference is responsible for loading and executing YOLO inference for PPE detection.

Detects helmets, vests, gloves, and other PPE in video frames.

Methods:

detect_ppe(frame): Runs the YOLO model on a given frame to detect PPE objects.

load_model(): Loads the pre-trained YOLO model into memory.

3.3. FirebaseConfig Class

Handles communication with Firebase Realtime Database.

Methods:

send_to_firebase(data): Sends PPE detection results to Firebase.

get_data_from_firebase(): Retrieves stored detection logs from Firebase.

3.4. Routes Class

Manages Flask API endpoints for SafeScope.

Methods:

detect_ppe_api(): API route to perform PPE detection (POST /detect_ppe).

send_to_firebase_api(): API route to send detection results to Firebase (POST/send_to_firebase).

video_feed(): Streams real-time video feed with detections (GET /video_feed).

3.5. PPEDetection Class

Continuously processes video frames, detects PPE, and stores results in Firebase.

Methods:

run_ppe_detection(): Captures frames, runs inference, and logs results in a loop.

capture_frame(): Captures a new frame from the camera.

process_results(results): Filters and formats YOLO model output for database storage.

3.6. ImageProcessing Class

Handles image processing for detected objects.

Methods:

process_video_frame(frame): Processes frames and overlays detection bounding boxes.

cornerRect(frame, bbox): Draws rounded rectangles around detected PPE objects.

3.7 Attributes

Attribute	**Description**	**Usage**	**Data Type**
camera	OpenCV VideoCapture object used to capture frames	ppe_detection.py, video_feed.py	cv2.VideoCapture
model	YOLO model object used for PPE detection	yolo_inference.py	YOLO
ALL_PPE_ITEMS	List of PPE class names that can be detected	yolo_inference.py	list
results	List of detection results from YOLO model	detect_ppe(frame)	list
boxes	List of bounding boxes for detected objects	process_video_frame(frame)	list
confidence	Confidence score of detected objects	detect_ppe(frame)	float
timestamp	Stores the time when a detection occurs	ppe_detection.py	String
detected_ppe	List of detected PPE items	detect_ppe(frame)	list
missing_ppe	List of PPE items that are not present	detect_ppe(frame)	list
firebase_url	Firebase database URL for storing detections	firebase_config.py	String
thread	Background thread for PPE detection	main.py	Thread
x1, y1, x2, y2	Bounding box coordinates for detected objects	process_video_frame(frame)	int
api_routes	Dictionary storing all API routes	routes.py	dict
frame_width, frame_height	Dimensions of the video frame	process_video_frame(frame)	int
image_format	Format of the processed image	image_processing.py	String
video_source	Source of the video feed (camera or file)	ppe_detection.py	String

4. Glossary

This glossary provides detailed explanations of key terms, abbreviations, and concepts used throughout the safeScope Low-Level Design (LLD). These terms are essential for understanding the architecture, design principles, and technical implementation of the system.

4.1 General Software Engineering Terms

API (Application Programming Interface)

A set of functions and protocols that allow different software components to communicate. In safeScope, REST APIs and WebSocket APIs are used to exchange detection data, alerts, and user interactions.

Asynchronous Processing

A programming approach where tasks are executed independently, allowing other operations to continue without waiting for completion. safeScope uses asynchronous API calls and background processing to ensure real-time performance.

Backend

The server-side part of an application that handles database operations, business logic, and API communication. In safeScope, the backend is built using Flask (Python) and Firebase to manage detection results, user authentication, and alert notifications.

Frontend

The client-side part of an application that users interact with, typically a web-based interface. In safeScope, the frontend is built using React.js to display PPE compliance data, alerts, and real-time monitoring dashboards.

CRUD (Create, Read, Update, Delete)

The fundamental operations performed on databases:

- Create → Insert new data
- Read → Retrieve data
- Update → Modify existing data
- Delete → Remove data
  These operations are implemented in safeScope’s database system (Firebase) to store PPE compliance logs and safety alerts.

Microservices Architecture

A design pattern where an application is divided into smaller, independent services that communicate through APIs. In safeScope, services like PPE detection, proximity monitoring, and alert management operate as loosely coupled modules, ensuring scalability and maintainability.

4.2 Artificial Intelligence and Machine Learning Terms

AI (Artificial Intelligence)

The simulation of human intelligence in computers to perform tasks like image recognition, decision-making, and automation. In safeScope, AI is used to detect PPE compliance and monitor worker proximity to machines.

Computer Vision

A field of AI that enables systems to analyze and interpret images or videos. safeScope relies on computer vision models for real-time detection of PPE (helmets, gloves, vests, etc.) using image processing techniques.

YOLO (You Only Look Once)

A real-time object detection algorithm used in safeScope to identify PPE compliance. It processes images in a single pass, making it significantly faster than traditional detection methods.

OpenCV (Open-Source Computer Vision Library)

A popular computer vision library used for image processing, object tracking, and feature detection. safeScope uses OpenCV for:

- Pre-processing video frames before running AI models
- Drawing bounding boxes around detected PPE items
- Calculating object positions for proximity detection

Deep Learning

A subset of machine learning that uses neural networks with multiple layers to analyze data. The YOLO model in safeScope is a deep learning model trained on PPE detection datasets to recognize helmets, gloves, and safety vests.

Neural Network

A machine learning model inspired by the human brain, consisting of layers of interconnected nodes (neurons). The YOLO model in safeScope is a type of convolutional neural network (CNN) specialized for image detection.

4.3 System Components and Technologies

Firebase

A cloud-based NoSQL database used for real-time data storage and synchronization. safeScope stores detection logs, user settings, and alerts in Firebase for quick access and real-time updates.

Flask

A lightweight Python web framework used to build safeScope's REST APIs. It manages:

- API endpoints for PPE detection requests
- Database interactions for storing compliance logs
- Real-time notifications for safety violations

React.js

A JavaScript library used to develop safeScope's user-friendly web interface. It enables:

- Dynamic dashboards displaying PPE compliance statistics
- Real-time updates using WebSockets
- User interaction with system settings and reports

WebSocket

A communication protocol that allows real-time, bidirectional data exchange between a server and a client. safeScope uses WebSockets for:

- Sending real-time PPE detection results to the frontend
- Alerting workers when they are too close to machinery
- Instant UI updates without refreshing the page

4.4 Safety and Workplace Compliance Terms

PPE (Personal Protective Equipment)

Safety equipment that workers wear to minimize workplace hazards. In safeScope, AI models detect the following PPE items:

- Helmet → Protects against head injuries
- Gloves → Prevents hand injuries
- Safety Vest → Enhances worker visibility
- Safety Glasses → Protects eyes from debris

OSHA (Occupational Safety and Health Administration)

A U.S. regulatory agency that sets workplace safety standards. safeScope helps organizations comply with OSHA regulations by ensuring workers wear PPE and follow safety protocols.

ILO (International Labour Organization)

A UN agency that establishes global labor safety standards. safeScope aligns with ILO recommendations for workplace hazard monitoring and PPE compliance.

Proximity Detection

A feature in safeScope that monitors the distance between workers and hazardous machinery using computer vision and AI algorithms. It triggers alerts when workers enter danger zones.

4.5 Security and Access Control Terms

RBAC (Role-Based Access Control)

A security model that restricts system access based on user roles. safeScope assigns different privileges to:

- Workers → Can view their PPE compliance status
- Safety Officers → Can monitor workplace-wide compliance and receive alerts
- Administrators → Can configure system settings and generate reports

TLS (Transport Layer Security)

A cryptographic protocol that ensures secure communication between system components. safeScope encrypts all data transmissions using TLS 1.3 to prevent unauthorized access.

Encryption

A method of encoding data so that only authorized users can read it. safeScope applies AES-256 encryption to protect worker safety data and compliance logs.

Audit Logs

A detailed record of system activities for tracking compliance and security events. safeScope maintains audit logs for:

- PPE detection results
- Proximity alerts
- System access and modifications

4.6 Software Development and Deployment Terms

CI/CD (Continuous Integration/Continuous Deployment)

A software development practice that automates code testing and deployment. safeScope integrates CI/CD pipelines to:

- Automatically test AI models before deployment
- Ensure backend services remain stable
- Deliver real-time updates to the web application

Docker

A containerization tool that packages safeScope's services (backend, AI models, and frontend) into isolated environments. This ensures:

- Easy deployment across different machines
- Consistency between development and production environments

Load Balancer

A system that distributes incoming network traffic across multiple servers to improve performance and reliability. safeScope uses cloud-based load balancing to handle high workloads in large industrial deployments.

5- References

- Smith, J., & Jones, A. (2020). "The Role of AI in Enhancing Workplace Safety." Journal of Occupational and Environmental Medicine, 62(5), 345-352
- National Institute for Occupational Safety and Health (NIOSH). "Workplace Safety and Health Topics." (2023). Retrieved from www.cdc.gov/niosh
- Brown, T., & Green, P. (2019). "Proximity Monitoring in High-Risk Work Environments." Safety Science, 118, 42-50.
- World Economic Forum. "The Future of Work: AI and Workplace Safety." (2022). Retrieved from www.weforum.org
- IEEE. "IEEE Standard for Software Design Descriptions (IEEE Std 1016-2009)." Retrieved from standards.ieee.org
- IEEE. "IEEE Recommended Practice for Software Requirements Specifications (IEEE Std 830-1998)." Retrieved from standards.ieee.org
- OWASP Foundation. "OWASP Security Best Practices for APIs." Retrieved from owasp.org
- Django Software Foundation. "Django Web Framework." Retrieved from www.djangoproject.com
- Google. "Firebase Documentation." Retrieved from firebase.google.co
- React. "React: A JavaScript Library for Building User Interfaces." Retrieved from reactjs.org
- MDN Web Docs. "WebSocket API." Retrieved from developer.mozilla.org

- Redmon, J., & Farhadi, A. (2018). "YOLOv3: An Incremental Improvement." Retrieved from pjreddie.com
- Plotly Technologies Inc. "Plotly Python Graphing Library." Retrieved from plotly.com
- NumPy Developers. "NumPy: The Fundamental Package for Numerical Computing." Retrieved from numpy.org
- OpenCV Developers. "OpenCV: Open Source Computer Vision Library." Retrieved from opencv.org
- Docker Inc. "Docker: Secure and Scalable Application Deployment." Retrieved from docker.com
- FastAPI Developers. "FastAPI: High-Performance APIs with Python." Retrieved from fastapi.tiangolo.com