BILKENT UNIVERSITY

CS492 - DESIGN PROJECT FINAL REPORT

Berkin Kağan Ateş	22102519
Burak Baştuğ	22102266
Umut Başar Demir	22102376
Berin Su İyici	22102342
Arda Öztürk	22103616

Table of Contents

1. Introduction

The logistics industry is a dynamic, continually expanding sector that encompasses warehouses, trucks, loads, routes, time constraints, and cost considerations. Due to the inherent complexity of this business, technology may be needed to help manage all of these components. Based on this concept, TırGöz offers a user-friendly system for companies in the logistics sector, leveraging computer vision technology and route-optimization algorithms to enhance the efficiency of logistical operations. TırGöz proposes a system that provides a holistic view of logistics operations by merging perception and planning layers. TırGöz provides operational efficiency by reducing manual inspection time and optimizing workforce usage. TırGöz offers cost reduction by minimizing late deliveries and unnecessary travel through adaptive route planning. TırGöz ensures work quality by preventing shipment errors through real-time alerts. TırGöz also considers the scalability by ensuring low latency and dynamic route optimization even with a large number of trucks and warehouses.

2. Requirement Details

2.1 Non-functional Requirements

In order to create the TırGöz system as an advanced experience for logistics operations some of the non-functional requirements are followed through the development process.

2.1.1 Usability

TırGöz focuses on high-efficiency usage for logistics operations. The system can be used with minimal effort due to its simple and user-friendly interface. The core functionalities are intuitive and easy to learn. For instance, optimized path, and real-time alert displays do not require the user to search or interpret complex data. Additionally, to enhance usability, anomaly detection alerts are accompanied by short event clips for easy visual verification, thereby minimizing the time required for manual inspection.

2.1.2 Reliability

TırGöz ensures high operational reliability. Since the Computer Vision and Route Optimization Services are isolated, a failure in one independent module does not affect the other independent

modules. Additionally, TırGöz prioritizes rapid recovery from any failure to maintain the integrity of real-time events and data, particularly during the critical hours of the day. Additionally, the anomaly detection function aims to achieve a high accuracy rate through deterministic validation checks.

2.1.3 Performance

The primary goal of TırGöz is to ensure low-latency processing due to its real-time functional nature. To maintain real-time alerts, TırGöz processes live video streams and detects anomalies with an end-to-end latency of no more than **1000** milliseconds. Also, the Route Optimization Module aims to recalculate the optimal route based on warehouse and truck occupancies within a desired time constraint for the standard operational load.

2.1.4 Supportability

TırGöz is designed to be straightforward, making it easier to understand bugs and problems, thereby reducing the time required to fix errors and install new features. This is why logs are used to record detailed notes in a consistent and easy-to-read format. Additionally, TırGöz aims to minimize complete system restarts, thereby keeping the system operational during minor system changes. To further increase supportability, the system's technical documentation is maintained in a complete and accurate manner.

2.1.5 Scalability

TırGöz must be able to handle an increased number of cameras, trucks, and deliveries. For instance, the Computer Vision Service and Event Processing API should not exceed the expected real-time constraints, even as the number of cameras and streams increases. The Route Optimization Service must perform calculations for both smaller and larger numbers of trucks and deliveries. Additionally, TırGöz must efficiently manage the rapidly growing volume of stored data, including short video clips of detected anomalies, events, and metrics.

2.2 Functional Requirements

2.2.1 Streaming

- TırGöz must demonstrate the live stream of the truck loading/unloading, workers' situation in the warehouse during the work hours with minimal possible delay.
- TırGöz must allow users to record desired parts of the live streams.
- TırGöz must record the short clips of detected events to provide visual evidence for alerts.

2.2.2 Detection

- TırGöz must detect the location of the items in the warehouse.
- TırGöz must detect the physically damaged items.
- TırGöz must send alerts for the detection of physically damaged items.
- TırGöz must detect the count of the item loaded/unloaded.
- TırGöz must detect if the item is loaded into the wrong truck.
- TırGöz must send alerts for the detection of wrongly loaded items.
- Based on the detections, TırGöz must demonstrate the
    - Truck occupancy percentage for every truck.
    - Item counts in the trucks and the warehouse by validation checks after loading.

2.2.3 Route Optimization

- TırGöz must calculate and demonstrate the optimal routes for the trucks based on the desired distance and time constraints.
- TırGöz must dynamically recalculate the optimal routes based on the changes in the truck loads and orders.
- TırGöz must let stores request items from the warehouse, then use these requests as input for optimal route calculation.

3. Final Architecture and Design Details

3.1 TırGöz Final Architecture

TırGöz follows a microservices based system design. This design system enables modularity, scalability and independence for ease of deployment of different system modules. The system includes a frontend dashboard interface, and API gateway, backend microservices, data layer, streaming and analytics components. This bundle's goal is to support logistics monitoring and optimisation in real time. The architecture achieves this goal by combining CV supported warehouse monitoring with logistics decision support [1]. This combination enables detection of operational events and their integration into route planning and warehouse analytics in near real time.

The frontend dashboard is a web interface that provides an operational dashboard to logistics operators. This provides the operator with insight into warehouse activity, access to detected anomalies and ability to manage logistics workflows. Communication of frontend and backend services are done through REST APIs with API gateway. This means of communication ensures security and structure for data exchange between subsystems.

The backend services are responsible for core functionalities such as video stream processing, CV inference, anomaly detection, event generation, warehouse state analysis and route optimisation. These services process camera streams incoming from warehouse environments and transform the images into structured operational events such as movement of cargo, occupancy changes and potential anomalies.

The data layer consists of multiple storage systems optimised for different types of data. PostgreSQL is used for structured operational data such as warehouse metadata and logistics events observed with their associated data [2]. Redis supports caching and event messaging, this approach enables low latency communication between components. Object storage systems are used for storing the video clips and other media artifacts generated during anomaly detection.

The architecture also integrates stream processing pipelines for near real time video analysis. In this analysis video are processed by CV models to detect objects, track movements and identify anomalies in warehouse operations. These events are then published to backend services, updating the system state.

This modular architecture allows our system to integrate visual perception, logistics analytics and operational decision support within a unified platform, without the need for cloud storage support. The modular approach also keeps the system flexible for future extensions such as additional vision models, integration with enterprise logistics systems and optimisation algorithms.

3.1 Overview

3.1.1 Client Layer

3.1.1.1 Role Views

The system has four client-side roles. They are Admin, Manager, Operator, and Driver views. While Admin can access all system functionalities, other roles will be limited to functionalities like stream display and anomaly detection.

3.1.1.2 Shared Views

In addition to role-based views, some pages will be common across all roles. The authentication view is a fully shared view. Map view is also a shared view, but it may present different functionalities and a user interface to different roles.

3.1.2 API Gateway Layer (Gateway Subsystem)

3.1.2.1 Request Router

The system has multiple microservices, each running on a different port on the host machine. The request router is responsible for redirecting client-layer requests to the correct services in other layers.

3.1.2.2 Auth Guard

The auth guard is responsible for validating JWT tokens on every incoming request and enforcing the permissions of the system's roles accordingly.

3.1.2.3 WebRTC Signal Handler

A peer-to-peer structure is used for RTSP streaming on the web application side. The signal handler manages WebRTC signalling between the client layer and MediaMTX streaming server for real-time viewing.

3.1.2.4 WebSocket Handler

The WebSocket Handler maintains persistent real-time communication between the client and backend services. It sends YOLO detection metadata, such as object classes, bounding boxes, confidence scores, timestamps, and stream identifiers, to the frontend so that detection overlays can be drawn on live streams. It also pushes anomaly alerts, sensor warnings, and system notifications to users instantly without repeated polling. The handler works with the Auth Guard to ensure that only authorized users can access relevant streams and notification channels.

3.1.3 Server Layer

3.1.3.1 Streaming Service

The streaming service contains a recording service. It had contained a media stream server as well; however, the MediaMTX module has been moved to the data acquisition layer. In that way, it can get the stream without relaying it to the server layer, thereby increasing efficiency. The Recording Service will be responsible for tracking stream chunks in the event of anomalies and for general recording functionality. It will store the recording data as video chunks.

3.1.3.2 Computer Vision Services

This subsystem is the main one that contains the core computer vision logic, including anomaly detection and item counting. It triggers the notification service after finding an anomaly and finishing its other logical processes. The YOLO sends the metadata it provides to the client layer via WebSocket. This communication is also used for the notification service.

3.1.3.3 Route Planning Services

The route planning layer retrieves the relevant data from the data acquisition layer to generate optimal delivery and pickup routes. It combines the raw data with parameters from the map service and outputs the final result to the route optimization service. It then

assigns the generated routes to drivers and serves these information to the computer vision services.

3.1.3.4 Core Backend Services

The core backend contains three main services: the auth service, the analysis service, and the event bus. The auth service manages responses from endpoints across different services based on user roles and credentials. The analysis service applies statistical formulations to raw data and generates meaningful statistical observations. The event bus has been added recently. It logs all system events to PostgreSQL for historical analysis and reporting.

3.1.4 Data Acquisition Layer

3.1.4.1 Camera Subsystem

The main point is to retrieve the streaming data and configure the system for the incoming real-time stream, including its connection details (IP address, port, and credentials). Additionally, it tracks the FPS and ping latency for each stream and allows the user to observe bandwidth usage.

3.1.4.2 Sensor Subsystem

It has two services: a sensor polling service and an alert publisher. The sensor polling sends requests to sensors every 2 seconds to read current values and check the critical boundary values. The alert publisher communicates to the event bus if the boundary value is exceeded and notifies the client layer.

3.1.5 Data Layer (Persistence Subsystem)

3.1.2.1 Redis Cache

It provides the system with high-speed data retrieval for frequently accessed data across different parts of the system, such as authentication credentials, sensor health monitoring, and anomaly notifications. It is also used to simulate stream-based hardware, such as RFID scanners. It provides an arbitrary scanning stream via Redis Events.

3.1.2.2 PostgreSQL

It is the system's main storage structure. It stores all structured data, including credentials, roles, events, goods, and warehouse attributes.

3.1.2.3 Minio Object Storage

It stores the recorded video chunks generated by the recording service in case of anomalies or user-intended recording requests.

3.2 Subsystem decomposition

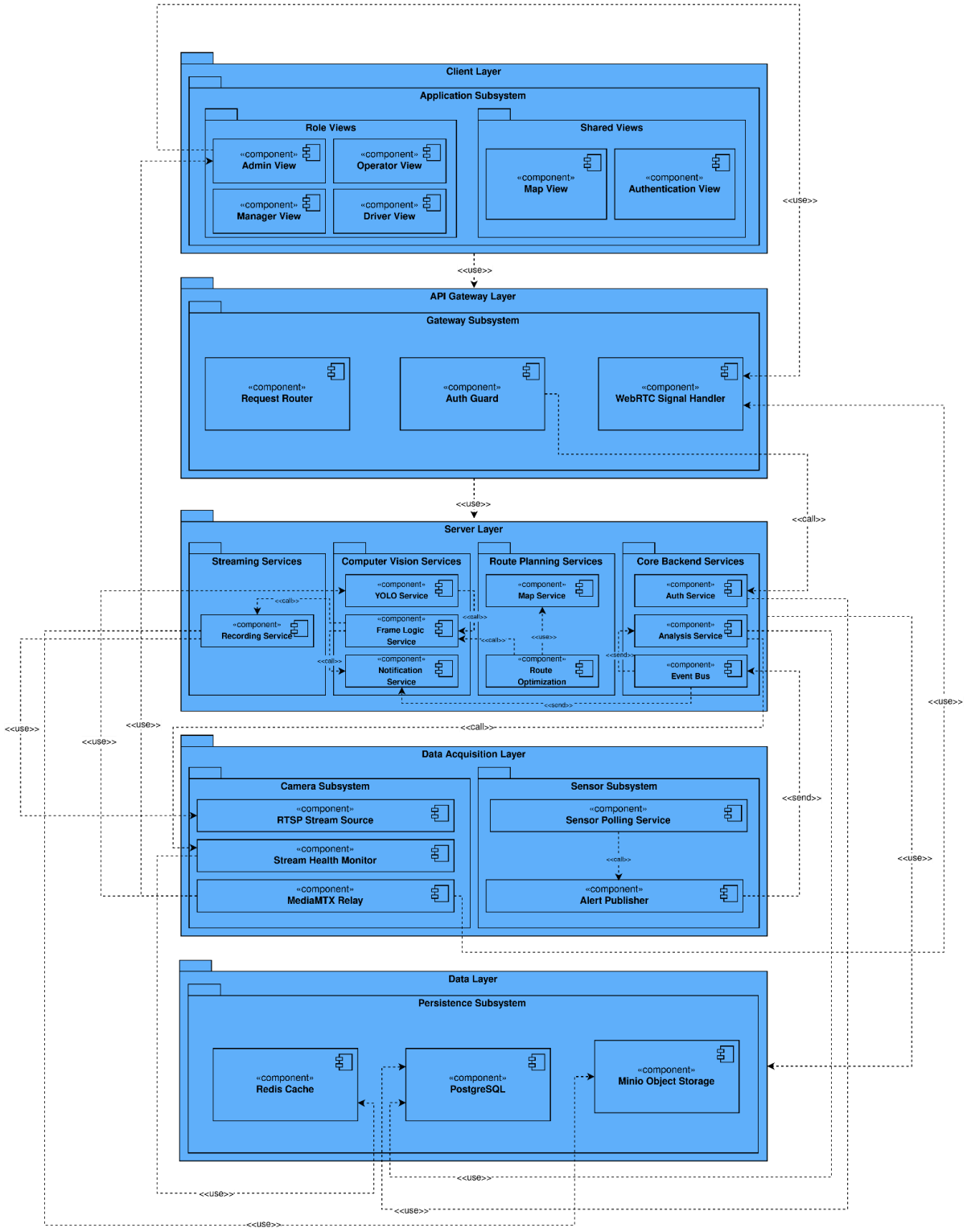


Figure 1: Subsystem Decomposition

3.3 Hardware/Software Mapping

The whole subsystem is deployed on a local host server in warehouses, without any cloud service connection. In the project, there are three hardware components: a local host server, RTSP Cameras, and environmental sensors.

3.3.1 Local Host Server

All backend services, the API gateway, and MediaMTX run on a single local host server.

Hardware	**Software Components**
Local Host Server	API Gateway Layer (Request Router, Auth Guard, WebRTC Signal Handler)
Local Host Server	Server Layer (Recording Service, YOLO Service, Frame Logic Service, Notification Service, Map Service, Route Optimization Auth Service, Analysis Service, Event Bus)
Local Host Server	Data Acquisition Layer (RTSP Stream Source, MediaMTX Relay, Sensor Polling Service, Alert Publisher, Stream Health Monitor)
Local Host Server	Data Layer (Redis Cache, PostgreSQL, Minio Object Storage)

3.3.2 IoT Devices

RTSP cameras are on the same subnet as the host server and are connected to it via a router or switch. Environmental sensors can be connected either directly to the host server using the deployment configuration.

Hardware	Software Components
RTSP Camera	Camera Subsystem (RTSP Stream Source, Stream Health Monitor, MediaMTX Relay)
Environmental Sensors	Sensor Subsystem (Sensor Polling Service, Alert Publisher)

3.3.2 Deployment Diagram

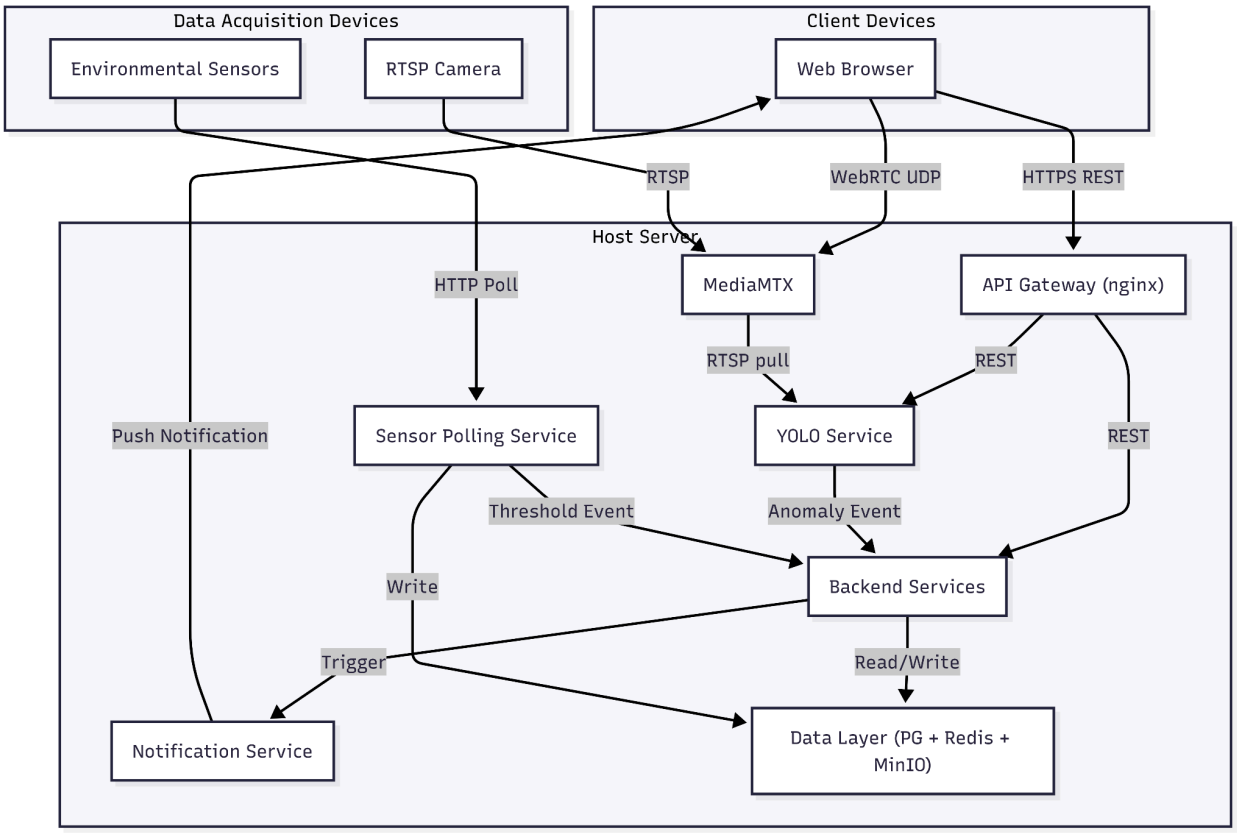


Figure 2: Deployment Diagram

3.4 Persistent Data Management

3.4.1 Relational Data

PostgreSQL is used for the main structured data storage. It stores account credentials, roles, camera-sensor configurations, detected anomaly entries, inventory logs, products, routes, analytic metrics, and warehouse attributes. It provides ACID compliance and a reliable storage unit. Anomaly event entries can grow rapidly during usage. Therefore, indices are applied to timestamps and camera IDs to provide efficient queries.

3.4.2 Cache Management

Redis is used for cache management. It provides high-speed retrieval for frequently accessed data such as authentication session tokens, aggregated statistical calculations, sensor status pings, and route data. Besides its caching benefits, Redis serves as the messaging backbone for the Event Bus through its built-in Pub/Sub mechanism. This dual use of Redis reduces PostgreSQL load and improves overall system performance.

3.4.3 File Storage

The system provides a client layer for recording real-time streams and saving clips of anomalous scenes. Since storing records can be costly for the server's local storage units and PostgreSQL, Minio provides a scalable solution for this type of data.

3.5 Access Control and Security

3.4.1 Authentication

The system uses four steps for the authentication process. These are JSON Web Token, the auth guard in the gateway, and the auth service in the server layer. The client layer submits its credentials to the auth guard in the gateway, which redirects to the auth service, which authenticates the client and generates a token. This is returned to the browser and stored in its local storage. In addition to credential submissions, the auth guard also checks every incoming request to forward it to the server later. Unauthenticated requests are rejected at the gateway level without reaching the server layer.

3.4.2 Role-Based Structure

The system has a role-based structure, with four roles for the client: Admin, Operator, Manager, and Driver.

- **Admin:** It has full system access. It can view the live stream, perform statistical analysis, handle products/orders and administer accounts.
- **Operator:** It has the same functionalities as the admin role except for user administration and settings.
- **Manager:** It is the role assigned to each branch that warehouses and transports goods. They can give and view orders for their own branches. They have very limited access to many features compared to the operator and admin roles.
- **Driver:** The driver role has access only to their assigned routes on the map.

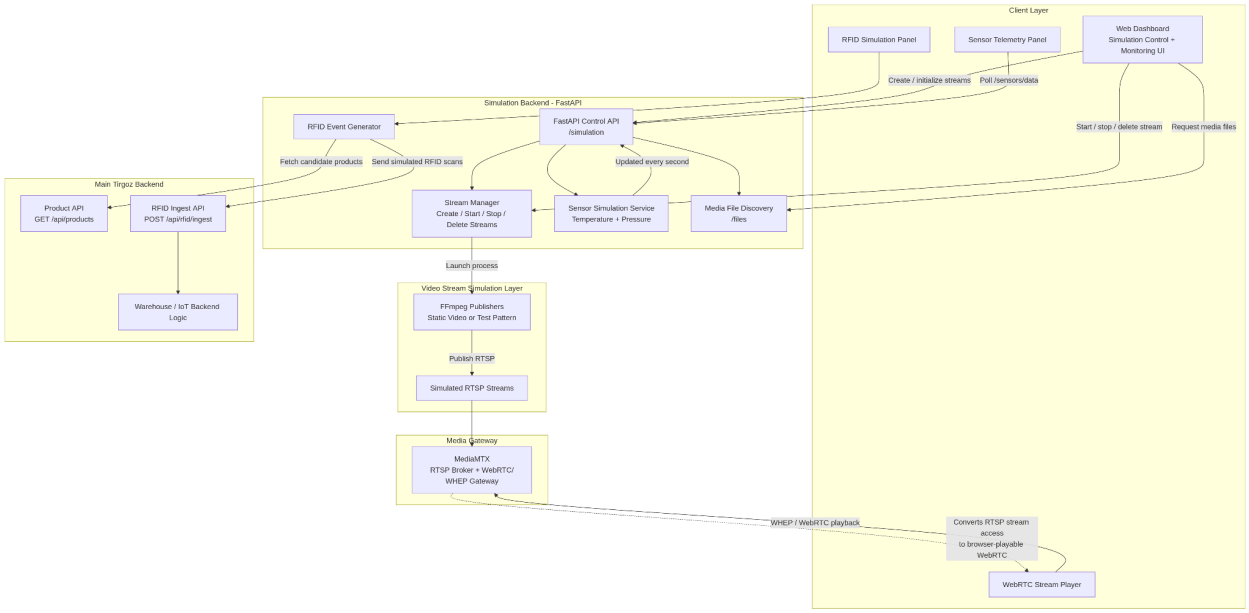
3.4.3 API Security

All communications between the client and the server layers use HTTPS. It provides the system payloads coming from the client to the server, encrypted. Every endpoint call triggers the auth guard in the gateway, which redirects the request to the corresponding service. Requests with missing, expired, or invalid tokens are rejected at the gateway level and never reach the backend services. This security check also verifies the request's role level and prevents lower-level roles from accessing endpoints that require higher-level roles.

In addition, the API Gateway performs security checks for WebRTC signalling, ensuring that only peers with the permitted user role can connect to the MediaMTX source peer.

3.6 Simulator

3.6.1 High-Level Architecture



3.6.2 Backend

It is the backbone of the simulation structure. It primarily uses FastAPI to generate mock sensor data, security camera streams, and RFID events. It communicates with the main app through its API endpoints.

3.6.3 Client Layer

It is a basic dashboard that enables developers to start and stop simulating modules for sensors, streams, and an RFID scanner. It also provides tabs for viewing streams and logs for the RFID scanner.

3.6.4 Video Stream Simulation Layer and Media Gateway

The video stream simulation layer retrieves a static video from the server’s storage and generates an infinite-loop RTSP stream using FFmpeg. Then, these streams are sent to MediaMTX, the media gateway in both the main app and the simulator. MediaMTX relays the stream to the client in a format that is playable in browsers via WebRTC. This enables streams

to be viewed in the simulation dashboard before implementing WebRTC signaling in the main application's client layer.

3.6.5 How does the Main TırGöz Backend use the Simulator?

The main app uses the simulator as the main data source. It allows developers to continue implementing features without external cameras, sensors, or scanners. This creates a small sandbox for developers before the real deployment in a warehouse.

4. Development/Implementation Details

4.1 Client Layer Services

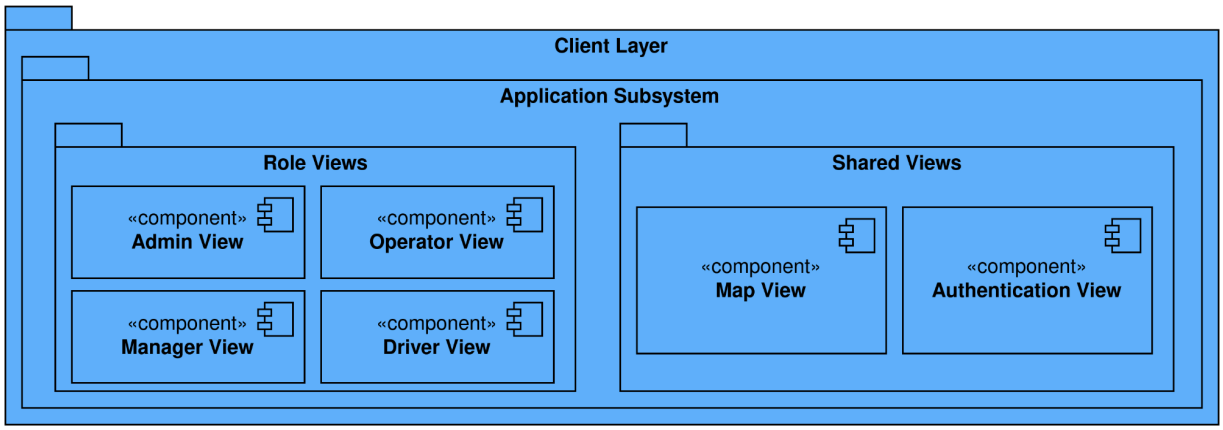


Figure 3: Client Layer

4.1.1 Application Subsystem

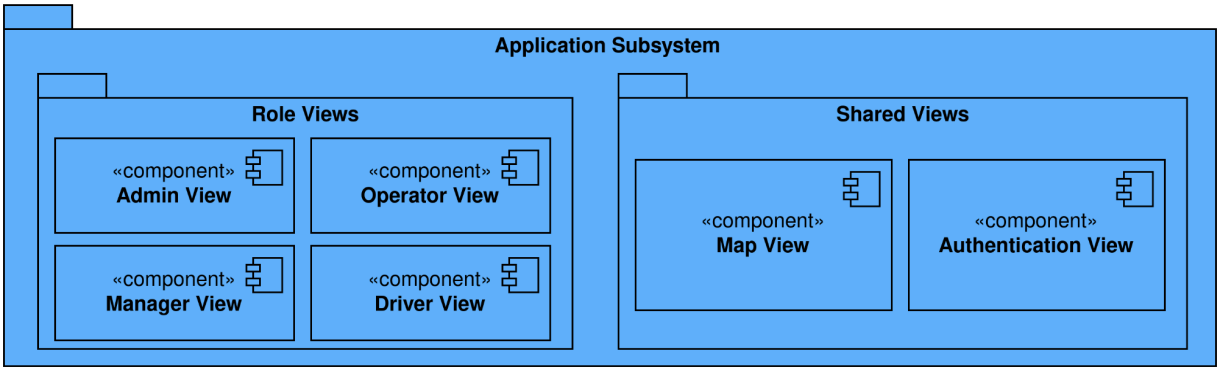


Figure 4: Application Subsystem

4.1.1.1 Role Views

In the TırGöz system there are four roles with different functionalities. This difference in use requires four different Views based on the role.

Admin View: Admin can reach every page, such as warehouse, orders, products, devices and settings pages, and additionally can see user handling areas.

Operator View: Operator can see the pages related to warehouse, orders, products, and devices. This view is the second largest view after the admin, since the operator needs to manage and monitor every step that our system reaches. .

Manager View: Manager has two main functionalities, so the manager can see requesting a product (creating an order) area, and monitor the order approval status.

Driver View: The driver has the most limited view only consisting of the route on a map view. This is because the only function of the driver is to complete physical delivery of the products.

4.1.1.2 Shared Views

Although there are different roles, some functionalities are in common use for all user types.

Notification View: Every account can view its own notifications using the sidebar navigation.

Authentication View: Every user type uses the same authentication views such as common login page to authenticate. Also success, fail and unauthorized access messages and views are common for every user. Also every user can update their password using the same view from their main page after logging in the system.

4.2 API Gateway Layer Services

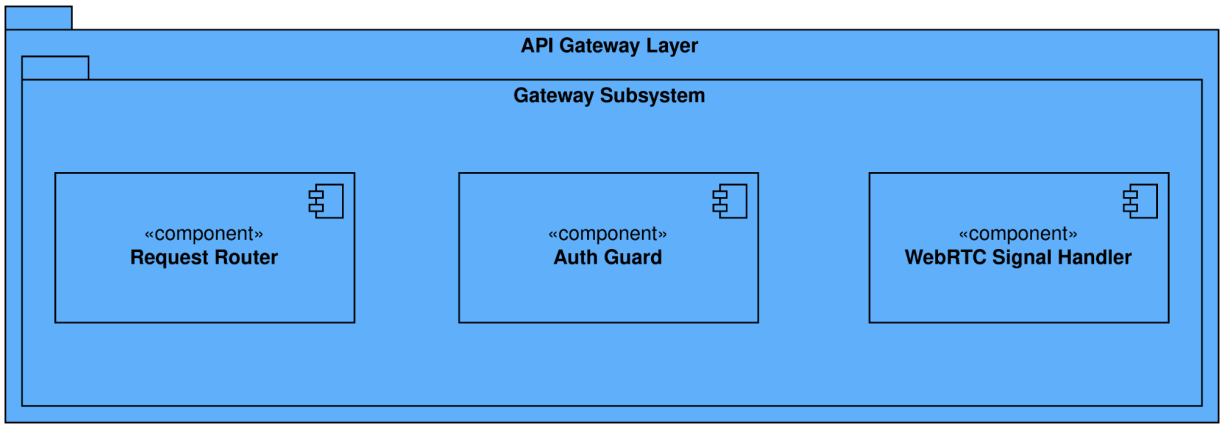


Figure 5: API Gateway Layer

4.2.1 Gateway Subsystem

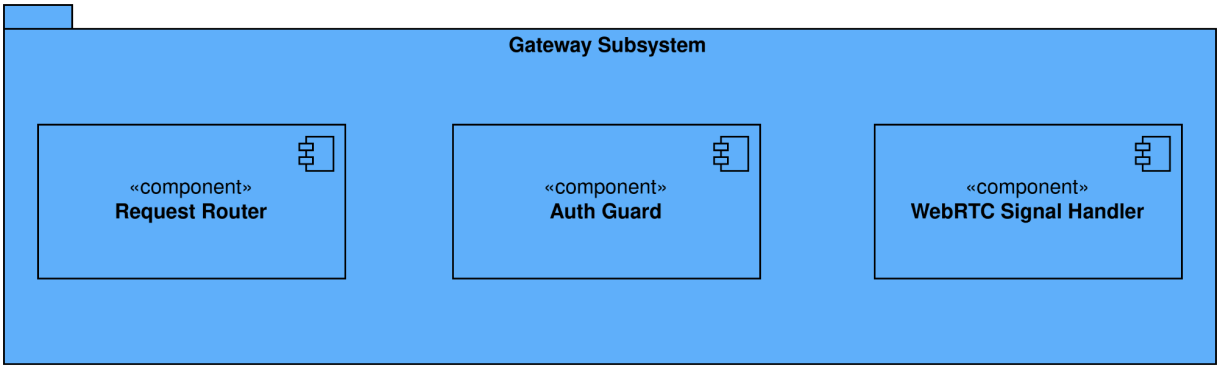


Figure 6: Gateway Subsystem

4.2.1.1 Request Router

Tırgöz project is a multiservice architecture project. It has multiple subsystems and components as microservices, and all of these services are in active communication. To handle the many requests going to different services, an Nginx-based gateway subsystem has been implemented. One of this gateway's functionalities is routing incoming requests.

It receives requests from the client layer, inspects them, and redirects them to the corresponding services. It gets requests on port 80. Requests to '/api/' are redirected to the backbone API service. '/simulation' forwards their requests to the simulation microservice. Besides these gates, '/minio/' are forwarded to the MinIO object storage. It has a 100 MB request size limit for HTTP messaging.

4.2.1.2 Auth Guard

Auth Guard is the system's first security check. It aims to handle security check loads without using auth elements in the server layer. It checks the credentials and role permissions before forwarding the requests.

For credentials, it checks the token validity. If the token is invalid, it returns a 401 Unauthorized response. Additionally, it returns a 403 Forbidden response if there is an inconsistency in role permissions.

4.2.1.3 WebRTC Signal Handler

The system provides real-time streaming functionality via the WebRTC protocol, which uses a peer-to-peer connection and UDP. In a peer-to-peer structure, two peers should find each other and engage in a signalling process to establish a constant stream pipeline between them. The signal handler handles this signalling process by acting as the intermediary between the client browser and the MediaMTX source peer. Both peers share an SDP (Session Description Protocol) that contains each peer's media capabilities and connection parameters. In that way, WebRTC determines the best candidate network path to relay the stream. Since both the host machine and cameras are in the same subnet, no TURN or STUN server is needed for the current implementation.

In addition, it has a peer-to-peer security architecture. It enforces a JWT token validation before beginning the signalling of the process. Once all validations are passed, the video stream flows between MediaMTX and the browser, directly bypassing the server layer to reduce the latency and server load.

4.3 Server Layer Services

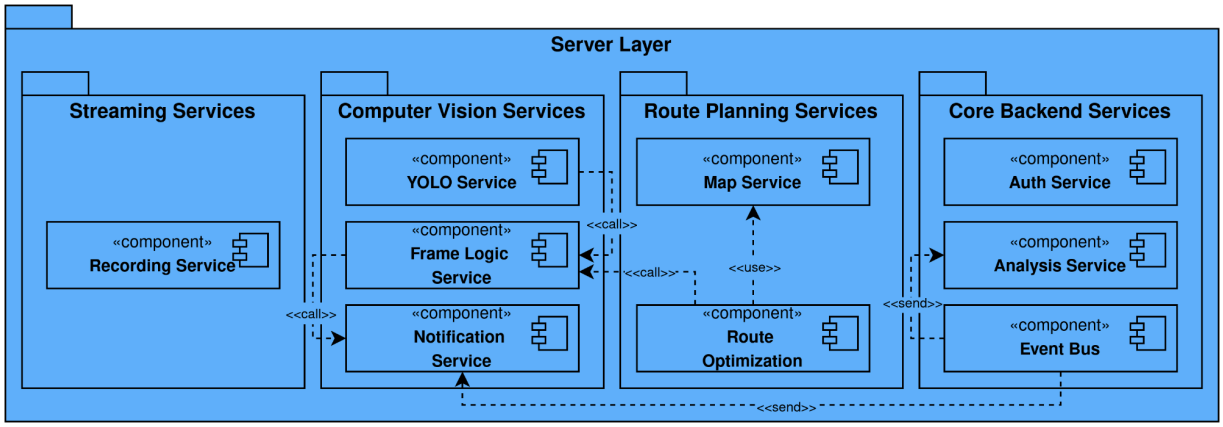


Figure 7: Server Layer

4.3.1 Streaming Services

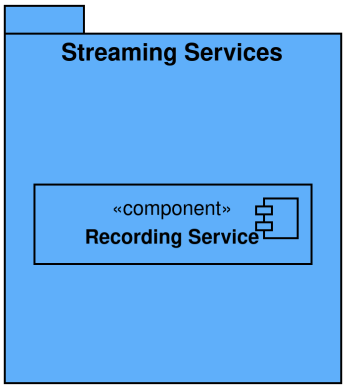


Figure 8: Streaming Services

4.3.1.1 Recording Service

Recording real-time streams has been implemented as a separate service because it uses an OS pipeline to invoke the FFMPEG recording feature. Since the building FFMPEG pipeline can be costly for the server, it is implemented as an atomic structure. It is called in user-intended requests and the Frame Logic Service in the computer vision services. It basically records stream chunks in response to triggers from other services in the system and stores them in the object storage unit.

4.3.2 Computer Vision Services

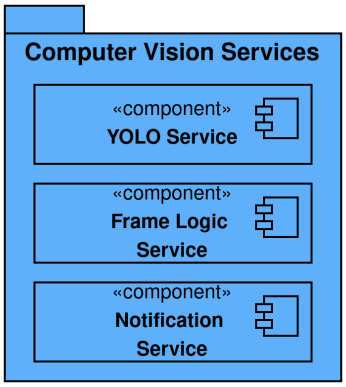


Figure 9: Computer Vision Services

4.3.2.1 YOLO Service

The YOLO Service serves as the primary object detection engine within the Computer Vision Services subsystem. It receives real-time video frames that are relayed directly by the MediaMTX module from the Data Acquisition Layer. By using YOLO deep learning models, this service performs low-latency inference to identify warehouse items, estimate capacity, and detect physical anomalies. Upon processing the frames and identifying any irregular events or anomalies, it generates structured metadata and triggers the Notification Service to alert the relevant users

4.3.2.2 Frame Logic Service

The Frame Logic Service acts as the main decision-making part of the computer vision system. It takes the basic information found by the YOLO Service and uses it to do things like count items or decide if a real problem has happened. When it spots an important event or a broken item, it tells the Recording Service to save a video clip of that exact moment. After finishing these checks, it passes the information along to trigger the Notification Service to alert the users.

4.3.2.3 Notification Service

The Notification Service acts as the messenger for the computer vision system. After the other services finish checking the video and spot a problem, they trigger this service to raise the alarm. It then takes these important alerts and passes them along to the core backend system. This makes sure that warehouse operators and managers get the warnings on their screens immediately, and also serves the action buttons depending on the notification content.

4.3.3 Route Planning Services

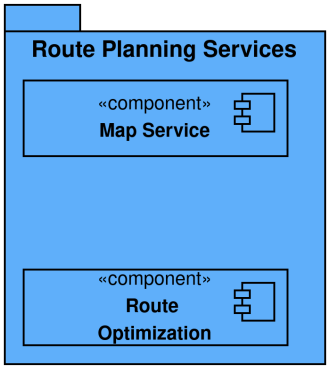


Figure 10: Route Planning Services

4.3.3.1 Map Service

The Map Service acts as the geographic map for the route planning system. It provides the essential map parameters, such as location data and distances, needed to figure out how to get from one place to another. By supplying this information, it helps the Route Optimization service accurately calculate the best and fastest delivery paths for the trucks. OpenStreetMaps and MapBox apis are used for displaying the map and road information.

4.3.3.2 Route Optimization

The Route Optimization service acts as the main brain for delivery planning. It takes geographic details from the Map Service and combines them with live data from the warehouse, such as changing truck states and new orders to generate the best delivery and pickup routes. This service relies on rule-based math to update and calculate optimal paths for the drivers.

4.3.4 Core Backend Services

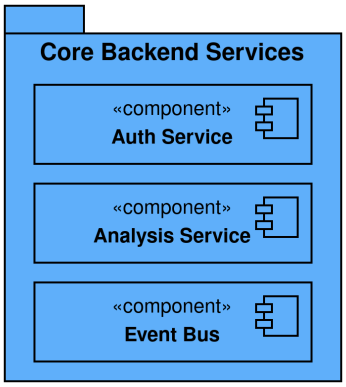


Figure 11: Core Backend Services

4.3.4.1 Auth Service

The Auth Service acts as the main security checker for the backend system. When a user logs in, it takes their credentials from the gateway, verifies their identity, and creates a secure session token. It also manages what each person is allowed to do, making sure that responses from different parts of the system match the permissions of the user's role.

4.3.4.2 Analysis Service

The Analysis Service acts as the main data parser for the platform. It takes the raw data collected from the system and applies statistical formulations to it. By doing this, it generates meaningful statistical observations, making it easy for administrators and operators to quickly understand the current state and efficiency of their logistics operations on their dashboards.

4.3.4.3 Event Bus

The Event Bus acts as the main message center for the system. It uses Redis and its Pub/Sub feature as the backbone for handling messages. When other parts of the system, like the alert publisher, notice that a specific limit has been crossed, they send a message directly to the Event Bus. Recently added to the core backend, its main job is to safely log all of these system events into the PostgreSQL database so they can be saved for historical analysis and reporting late.

4.4 Data Acquisition Layer Services

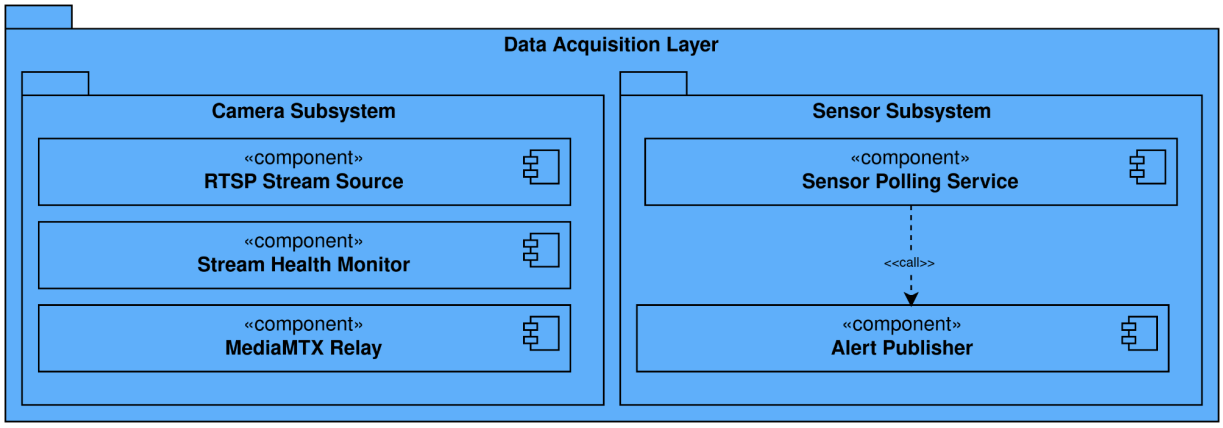


Figure 12: Data Acquisition Layer

4.4.1 Camera Subsystem

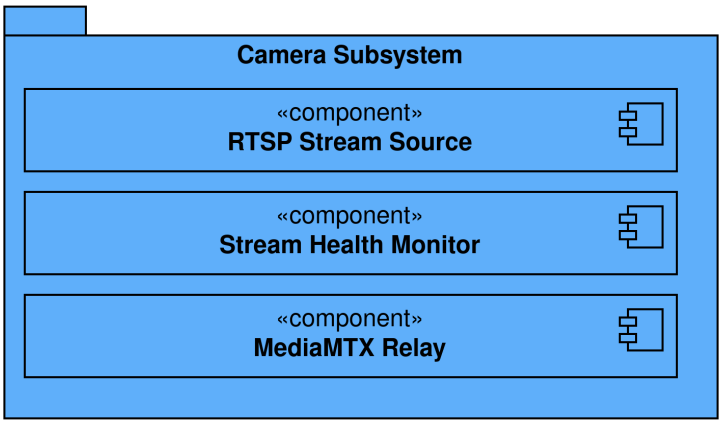


Figure 13: Camera Subsystem

4.4.1.1 RTSP Stream Source

It behaves like a connection point before the MediaMTX Relay. It contains the configurations that identify the physical cameras, which are IP addresses, port numbers, and RTSP URLs. These configurations can be fully editable, and new ones can be added or old ones removed. It provides a flexible deployment environment for different physical implementations.

4.4.1.2 Stream Health Monitor

The health monitor is a diagnostic service that continuously tracks all cameras by retrieving the health and connection status of each registered camera. It informs the system of the cameras' key metrics, including frame rate and ping latency.

In addition to the monitoring feature, it also detects connection losses for each camera and triggers its event listener to communicate with the Event Bus, which then transfers the warning to the client layer and displays the corresponding message to the final user.

4.4.1.3 MediaMTX Relay

Video and audio streams can be published, read, proxyed, recorded, and played back using MediaMTX, a ready-to-use, zero-dependency real-time media server and media proxy. It is considered a "media router" that directs media streams from one end to the other. The main reason MediaMTX is used is that web browsers cannot display the RTPS stream sources

directly using HTML DOM elements. Therefore, the RTSP stream should be converted to a protocol that can be viewed in the browser with low latency.

After the conversion, it relays the stream to the client browser using the signalling process handled by the WebRTC Signal Handler in the Gateway Subsystem. At the same time, it relays the output stream to the YOLO service to process the frames. MediaMTX handles these dual or multiple streams efficiently. Because it relays the same RTSP stream to multiple consumers without duplicating the stream flow between cameras and itself [3]. It is integrated with the gateway subsystem using the ‘stream/’ proxy endpoint.

4.4.2 Sensor Subsystem

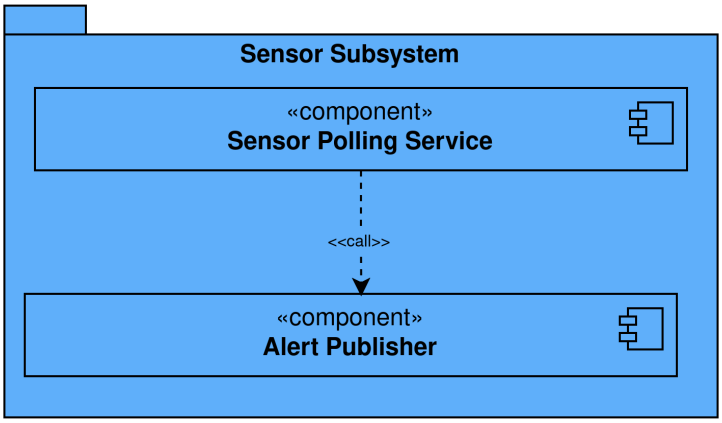


Figure 14: Sensor Subsystem

4.4.2.1 Sensor Polling Service

The Sensor Polling Service acts as the constant checker for the physical environment of the warehouse. It actively sends requests to the connected environmental sensors to read their current values. By constantly collecting this data, it checks to see if any critical boundary values have been crossed, and then works with the Alert Publisher to raise the alarm if a problem is found.

4.4.2.2 Alert Publisher

The Alert Publisher acts as the alarm bell for the sensor system. If the sensor polling service finds that a critical boundary value has been exceeded, this service immediately takes action. It communicates the problem directly to the Event Bus and notifies the client layer so that users can see the warning on their screens right away.

4.5 Data Layer Services

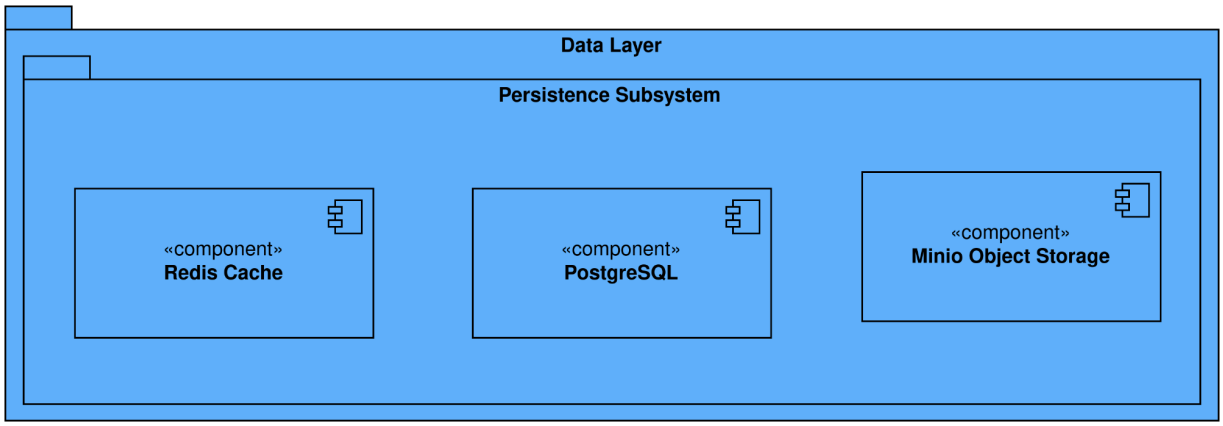


Figure 15: Data Layer

4.5.1 Persistence Subsystem

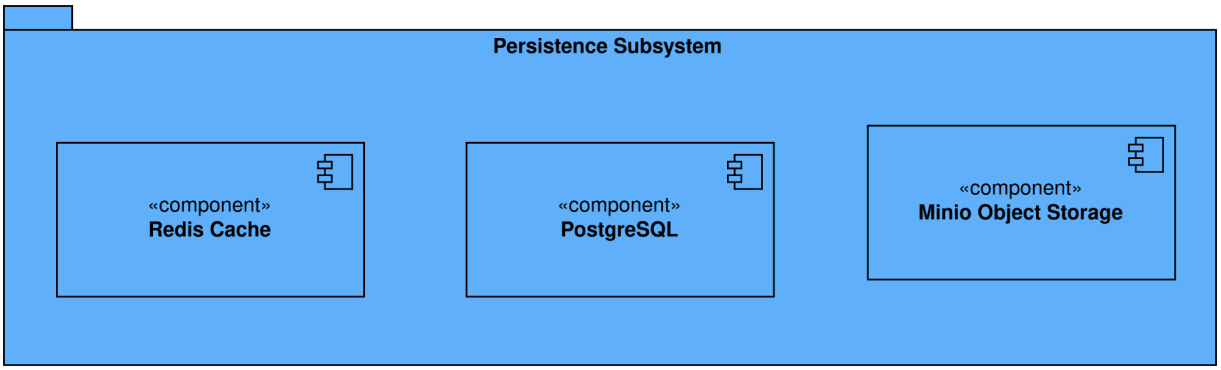


Figure 16: Persistence Subsystem

4.5.1.1 Redis Cache

The Redis Cache acts as the high speed memory for the system. It stores and retrieves data that is used all the time, like login details, anomaly notifications, and sensor status updates [4]. Because it is so fast, it also works as the main messaging system for the Event Bus. By handling these quick tasks, it takes the heavy lifting off the main database and helps the whole platform run much smoother.

4.5.1.2 PostgreSQL

PostgreSQL acts as the main database for the entire system. It stores all the organized information, such as user accounts, camera settings, warehouse items, and logs of any problems found. Because the list of recorded events can grow very quickly, it uses special tracking methods on things like time and camera IDs so that the system can always search and find information fast.

4.5.1.3 MinIO Object Storage

MinIO Object Storage acts as the main video archive for the system. It stores all the recorded video chunks that are created when an anomaly is detected or when a user specifically asks to record a live stream. Because keeping large video files would take up too much space on the local server or the main database, MinIO provides a scalable solution to handle and organize this data.

5. Test Cases and Results

5.1 Test Cases for Functional Requirements

5.1.1 Computer Vision Services Test Cases

Test ID	TC-01
Category	Functional
Objective	Verify YOLO object detection for standard warehouse items.
Procedure	1. Initialize the CV subsystem container. 2. Feed a pre-recorded RTSP test stream of the loading dock containing distinct boxes and pallets. 3. Intercept the JSON output from the YOLO inference engine.

	4. Validate the bounding box coordinates against the ground truth. 5. Check the confidence scores for the 'box' and 'pallet' classes.
Expected Results	The object detection module correctly identifies boxes and pallets, drawing accurate bounding boxes with confidence scores exceeding the predefined minimum threshold.
Priority	High
Date Tested	20.02.2026
Test Result	**Passed.** The YOLO model successfully detected standard boxes and pallets with an average confidence score of 0.7, consistently exceeding the 0.6 minimum threshold. Bounding boxes aligned with ground truth coordinates.

Test ID	TC-02
Category	Functional
Objective	Verify the synthetic data generation pipeline outputs.
Procedure	1. Define a JSON/YAML configuration for a specific environment template (e.g., truck interior, warehouse shelf layouts). 2. Execute the synthetic data generation script. 3. Inspect the designated output directory. 4. Cross-reference the generated images with their corresponding exported annotation files.
Expected Results	The pipeline successfully outputs generated images alongside correctly

	formatted annotation files (containing bounding boxes, tracking IDs, and anomaly labels).
Priority	Medium
Date Tested	14.01.2026
Test Result	**Deprecated.** The synthetic data generation pipeline has been formally deprecated and removed from the system architecture. The project now relies entirely on gathered and augmented real world datasets to ensure higher accuracy in edge cases.

Test ID	TC-03
Category	Functional
Objective	Validate data augmentation for model training.
Procedure	1. Input a verified clean image of a truck interior with its annotations. 2. Trigger the real-life scenario augmentation module. 3. Apply sequential transformations: gaussian noise, motion blur, and localized dimming (lighting changes). 4. Programmatically verify that the bounding box coordinates remain correctly aligned with the transformed objects.
Expected Results	The system outputs augmented images simulating scenarios without breaking the bounding box annotations.
Priority	Medium
Date Tested	05.03.2026

Test Result	**Passed.** Sequential transformations were successfully applied to the real-world dataset images. Bounding box coordinates remained correctly anchored to the target objects.

Test ID	TC-04
Category	Functional
Objective	Test persistent object tracking across frames.
Procedure	1. Feed a 30-second video clip where an item moves from the warehouse staging area into the truck region. 2. Introduce a temporary visual occlusion. 3. Log the unique tracking IDs assigned by the tracking module frame by frame.
Expected Results	The tracking module assigns an ID to the item and successfully maintains that exact same ID throughout the movement.
Priority	High
Date Tested	18.03.2026
Test Result	**Passed.** The tracking algorithm successfully maintained the same unique ID for the item across the entire clip, and successfully recovered/re-assigned the correct ID after visual occlusions.

Test ID	TC-05
Category	Functional
Objective	Verify anomaly detection for damaged goods.
Procedure	1. Stream a specific test video clip showing a visibly crushed box being loaded. 2. Monitor the Anomaly Detection module's classifier output probabilities. 3. Confirm the 'damage' class probability exceeds the predefined condition threshold. 4. Verify that an event payload (containing severity, confidence score, and a short video clip URI) is generated and published to the Alert Service.
Expected Results	The module successfully flags the damage, generates the correct event schema, and extracts the short video clip with timestamps for visual evidence
Priority	High
Date Tested	02.04.2026
Test Result	**Passed.** The system successfully triggered the Recording Service and the Alert Service received the complete JSON payload containing the short clip URI.

Test ID	TC-06

Category	Functional
Objective	Evaluate model robustness under stress conditions.
Procedure	1. Load a validation dataset specifically curated with different scenarios. 2. Execute the evaluation script to run inference on the entire dataset. 3. Parse the resulting evaluation logs to calculate the mean Average Precision. 4. Compare the results against the baseline accuracy threshold.
Expected Results	The model completes the evaluation pipeline, and the parsed log files confirm that the accuracy remains within acceptable limits.
Priority	Medium
Date Tested	15.04.2026
Test Result	**Passed.** The evaluation script processed 5,000+ validation frames under heavy system load. The mean Average Precision (mAP) remained stable at 0.84.

Test ID	TC-07
Category	Functional
Objective	Verify worker safety equipment detection.
Procedure	1. Input a video frame showing a warehouse worker walking through a designated zone without a required safety vest/helmet.

	2. Ensure the CV subsystem is actively tracking humans. 3. Check the anomaly event generation stream results. 4. Verify the event metadata sent to the Alert Service.
Expected Results	The CV subsystem identifies the worker, successfully flags the missing safety equipment, and pushes an event containing the event details, confidence score, and a replay option.
Priority	High
Date Tested	22.04.2026
Test Result	**Passed.** The human tracking model successfully identified the worker. The secondary safety classifier flagged the absence of a helmet with 0.94 confidence, publishing the event payload to the Alert Service.

Test ID	TC-08
Category	Functional
Objective	Validate barcode detection and decoding.
Procedure	1. Present an item with a clear barcode moving through the defined scanning zone. 2. Trigger the Object Detection module to locate the barcode's surface region. 3. Pass the cropped barcode region to the decoding library.

	4. Cross-reference the decoded string with the test Inventory Database.
Expected Results	The module locates the barcode, successfully decodes the metadata, validates it against the database, and updates the item's location and "count" state
Priority	Low
Date Tested	28.04.2026
Test Result	**Passed.** The bounding box accurately cropped the barcode region. The decoding library successfully extracted the string, verified it against the PostgreSQL Inventory Database, and updated the "count" state.

Test ID	TC-9
Category	Component
Objective	Check that the YOLO service loads the trained model weights successfully at start.
Procedure	1. Start the YOLO Service container. 2. Observe the startup logs of the service. 3. Verify that the configured model weights file is found and loaded. 4. Confirm that the service enters ready state without runtime errors.
Expected Results	The YOLO service loads the model weights successfully and in ready state for inference.
Priority	High
Date Tested	03.02.2026

Test Result	**Passed.** The container initialized, successfully loaded the custom .pt weights file from the persistent volume, and entered the ready inference state in 4.2 seconds without memory allocation errors.

Test ID	TC-10
Category	Training
Objective	Check that the dataset loader rejects annotation files with missing or malformed labels before training starts.
Procedure	1. Prepare a training dataset containing at least one malformed annotation file. 2. Start the training or fine-tuning pipeline. 3. Observe the dataset validation step. 4. Check whether the problematic file is reported correctly.
Expected Results	The training pipeline stops or skips invalid data according to configuration, reports the malformed annotation clearly, and does not proceed silently with broken labels.
Priority	High
Date Tested	18.03.2026
Test Result	**Passed**. The updated validation script now correctly identifies malformed .txt annotation files, safely skips them to prevent pipeline crashes, and outputs a detailed,readable error trace to the console.

Test ID	TC-11

Category	Performance
Objective	Check that inference service keeps the memory usage within acceptable limits during long running stream processing.
Procedure	1. Start the CV subsystem with a continuous RTSP test stream. 2. Let the service run for an extended period (more than 30 minutes) 3. Record the memory usage at regular intervals. 4. Compare the memory usage trend over time. 5. If memory usage increases significantly, test with longer time intervals to stress test the system. 6. Repeat 3-5
Expected Results	Memory usage remains stable after without abnormal growth or memory leak symptoms during continuous inference.
Priority	High
Date Tested	30.04.2026 (Retested after final changes)
Test Result	Passed. After 2+ hours of continuous streaming with 3 streams in parallel memory usage trend did not show signs of abnormal growth. The YOLO inference service stayed around approximately 2.15–2.27 GiB out of 15.5 GiB. The backbone API stayed around approximately 270–310 MiB, the simulation service stayed around approximately 430 MiB in the later part of the test, and MediaMTX stayed around approximately 42–45 MiB. Process counts remained stable, with no crash, restart loop, or visible memory leak symptoms.

5.1.2 Live Stream Acquisition, Relay, and Delivery Tests

Test ID	TC-12
Category	Functional
Objective	Test that the new camera configuration can be added to the system as an IoT device.
Procedure	1. Log in as an admin. 2. Click on the ‘Devices’ in the sidebar. 3. Add the ‘+’ button to insert a new configuration. 4. Submit device type, name, IP address, and RTSP URL. 5. See the success message in the user interface.
Expected Results	The new configuration is successfully added to the PostgreSQL.
Priority	High
Date Tested	02.12.2025
Test Result	Pass

Test ID	TC-13
Category	Functional
Objective	Check that MediaMTX can receive the stream coming from the RTSP camera.
Procedure	1. After adding camera configuration via the UI, set the RTSP stream URL in the MediaMTX yaml file. 2. Starting the connection between the camera device and the MediaMTX server.

	3. Check the logs to confirm the active stream.
Expected Results	The stream server receives the stream without errors.
Priority	High
Date Tested	03.12.2025
Test Result	Pass

Test ID	TC-14
Category	Functional
Objective	Check that the stream server can relay the stream to the frontend view.
Procedure	1. Log in as an admin. 2. Click on the ‘Devices’ in the sidebar. 3. Click on the ‘View Stream’ button. 4. Display the real-time stream on the videoplayer component.
Expected Results	The stream can be played in the video player.
Priority	High
Date Tested	03.12.2025
Test Result	Pass

Test ID	TC-15

Category	Functional
Objective	Check that Stream Health Monitor can detect a lost camera connection.
Procedure	1. Log in as an admin. 2. Click on the ‘Devices’ in the sidebar. 3. Click on the ‘View Stream’ button. 4. Display a real-time line chart showing the current ping and latency values. 5. The line chart can indicate when the connection is lost and notify the user.
Expected Results	Stream Health Monitor reports the camera as disconnected.
Priority	Medium
Date Tested	12.04.2026
Test Result	Pass

Test ID	TC-16
Category	Security
Objective	Check that WebRTC signalling is prevented when no valid session token is present.
Procedure	1. Call the ‘start stream’ endpoint via Swagger UI. 2. Observe whether the WebRTC Signal Handler blocks the signalling request at the gateway layer.
Expected Results	The signalling request is rejected by the WebRTC Signal Handler.

Priority	High
Date Tested	12.04.2026
Test Result	Pass

Test ID	TC-17
Category	Functional
Objective	Check whether MediaMTX can relay the stream to the YOLO service.
Procedure	1. Check that MediaMTX receives the stream and the YOLO service is running. 2. Check the YOLO microservice logs that it can generate metadata using the MediaMTX stream relay. 3. Display the stream with YOLO-drawn boundary boxes on the simulation page.
Expected Results	YOLO service can receive the stream and process it.
Priority	High
Date Tested	03.03.2026
Test Result	Pass

5.1.3 Sensor Polling, Threshold Evaluation, and Alert Publishing Tests

Test ID	TC-18
Category	Functional

Objective	Test that the new sensor configuration can be added to the system as an IoT device.
Procedure	1. Log in as an admin. 2. Click on the ‘Devices’ in the sidebar. 3. Click on the ‘+’ icon under the sensors title. 4. Submit device type, name, and IP address. 5. Seeing the success message box in the user interface.
Expected Results	The new sensor configurations are registered to the PostgreSQL database.
Priority	High
Date Tested	12.04.2026
Test Result	Pass

Test ID	TC-19
Category	Functional
Objective	Test that the system can read the current value of the registered sensors every 2 seconds.
Procedure	1. After registering a sensor in the system, start the Sensor Polling Service. 2. Observe the incoming data through the simulation page. 3. Verify that reading can be displayed in the UI.
Expected Results	The values can be read successfully from the sensors and displayed on the screen.

Priority	High
Date Tested	12.12.2025
Test Result	Pass

Test ID	TC-20
Category	Functional
Objective	Check that the Sensor Polling Service can detect when a registered sensor loses its connection.
Procedure	1. Start the Sensor Polling Service. 2. Create a case where one of the sensors loses its connection. 3. Observe the Sensor Polling Service response and wait for the warning message in the UI.
Expected Results	The service can detect a lost connection immediately.
Priority	Medium
Date Tested	23.04.2026
Test Result	Pass

5.1.4 Authentication and Authorization Tests

Test ID	TC-21
Category	Functional
Objective	Check that a user can be added successfully to the system by admin.
Procedure	1. Open the application and login as admin.

	2. Click the register user button. 3. Fill the required information. 4. Click the register button
Expected Results	The system successfully registers the new user.
Priority	High
Date Tested	21.02.2026
Test Result	Pass

Test ID	TC-22
Category	Functional
Objective	Check that a registered user can log in.
Procedure	1. Open the application login page. 2. Enter the required information. 3. Click the login button.
Expected Results	The system successfully allows the user to log in.
Priority	High
Date Tested	21.02.2026
Test Result	Pass

Test ID	TC-23

Category	Functional
Objective	Check that an unregistered user cannot log in.
Procedure	1. Open the application login page. 2. Enter the required information. 3. Click the login button.
Expected Results	The system does not allow the user to log in. Shows the login failed error.
Priority	High
Date Tested	21.02.2026
Test Result	Pass

Test ID	TC-24
Category	Functional
Objective	Check that users can only use the authorized endpoints.
Procedure	1. Login the system with different roles (for instance admin). 2. Try to send a POST request to register a user as an admin. 3. Observe that the request is successful. 4. Login the system with different roles (for instance driver). 5. Try to send a POST request to register a user as a driver.
Expected Results	The driver gets unauthorized access as a response and the attempt fails.
Priority	Medium
Date Tested	21.02.2026

Test Result	Pass

Test ID	TC-25
Category	Functional
Objective	Check that the admin can delete users.
Procedure	1. Login the system as admin. 2. Find the user management area. 3. Select the user which is wanted to be deleted. 4. Click the delete button.
Expected Results	The user is removed from the system.
Priority	Critical
Date Tested	21.02.2026
Test Result	Pass

Test ID	TC-26
Category	Functional
Objective	Check that users can update their passwords.
Procedure	1. Login the system with any role. 2. Find the change password area and click on it. 3. Fill the desired fields. 4. Click the confirmation button.

Expected Results	The user password is updated with the new information.
Priority	Medium
Date Tested	21.02.2026
Test Result	Pass

Test ID	TC-27
Category	Functional
Objective	Check that the system logs out the user.
Procedure	1. Login the system with any role. 2. Find the logout button and click.
Expected Results	The system revokes all refresh tokens and access tokens in backend and frontend are revoked, then, the user is led to the login page.
Priority	Critical
Date Tested	21.02.2026
Test Result	Pass

5.1.5 Product/Order Tests

Test ID	TC-28
Category	Functional
Objective	Check the system can manage handling (single/bulk upload and modification) of the products.

Procedure	1. Open the application. 2. Login as admin or warehouse operator. 3. Open the Products page from the sidebar. 4. Use the bulk upload or add buttons (by filling the add form) to add products. 5. After adding a product, use the edit button (by filling the edit form) to edit the selected product.
Expected Results	Products are added successfully and edited successfully, then the new products inventory is displayed after the handling operations.
Priority	Medium
Date Tested	17.04.2026
Test Result	Pass

Test ID	TC-29
Category	Functional
Objective	Check the system allows a branch manager to create/cancel an order.
Procedure	1. Open the application. 2. Login as branch manager. 3. Open the Ordering page from the sidebar. 4. Use the product list that is displayed to select the desired amount of the available product stock to create an order. 5. After completing the desired order, use the checkout button to complete the order. 6. Open the Orders page and use the cancel button to cancel the

	desired order.
Expected Results	Order is successfully created and can be monitored at the Orders page. At Orders page the selected order is cancelled successfully and reserved stock is calculated according to the status of the order.
Priority	Medium
Date Tested	30.04.2026
Test Result	Pass

Test ID	TC-30
Category	Functional
Objective	Check the system allows an admin or warehouse operator to approve, disapprove and reject an order.
Procedure	1. Open the application. 2. Login as admin or warehouse operator. 3. Open the Orders page from the sidebar. 4. Use the approve reject and disapprove buttons to change the status of an order.
Expected Results	Orders are created at the pending status. If an order is pending and there exists enough stock for the entire order, the approve button is active and sends the order to approved status and calculates the reserved stock accordingly. If an order is pending, the reject button is always active and sends the order to rejected status. If an order is approved, the disapprove button is active and sends the order to pending status, also calculates the reserved stock accordingly.

Priority	Medium
Date Tested	30.04.2026
Test Result	Pass

5.1.6 Cross Subsystem Tests

Test ID	TC-31
Category	Integration
Objective	Check that duplicate anomaly events are not sent to the dashboard repeatedly for the identical instance of the anomaly within the short time window.
Procedure	1. Start the CV subsystem and Notification Service 2. Feed a test stream containing the same instance of an anomaly for some determined period. 3. Monitor the event output generated by the CV subsystem. 4. Count how many anomaly notifications are published for the same event window.
Expected Results	The system suppresses repeated duplicate anomaly events and posts to notification service only the intended number of notifications.
Priority	High
Date Tested	01.05.2026
Test Result	PASS

Test ID	TC-32
Category	Integration
Objective	Verify that during an active loading detection, product detections are converted into loading events, validated against the assigned order, and shown in the loading monitor.
Procedure	1. Start all required containers. 2. Make sure there are approved orders, and active dock-camera setup. 3. Create or select an active loading context in the Loading Screen. 4. Feed a controlled stream containing a known product barcode. 5. Move or simulate the detected item from the warehouse side to the truck side of the dock. 6. Observe backend logs or frontend network activity to verify that events are created. 7. Check the Loading Monitor to confirm that the product movement and order progress are shown.
Expected Results	The detected product movement is recorded as a loading event, checked against the assigned order, and displayed in the Loading Monitor with updated loading progress/status.
Priority	High
Date Tested	04.05.2025
Test Result	PASS

Test ID	TC-33

Category	Integration
Objective	Check that a model produced by the fine-tuning pipeline can be loaded directly by the deployed YOLO service without manual modification.
Procedure	1. Complete a fine-tuning run and obtain the exported model weights. 2. Replace the previous deployment weights with the newly trained weights. 3. Restart the YOLO service. 4. Run inference on a known validation sample. 5. Verify that the service uses the new model successfully.
Expected Results	The deployed YOLO service loads the newly fine-tuned model directly and performs inference successfully without additional conversion or manual patching.
Priority	High
Date Tested	20.03.2026
Test Result	PASS. A fine-tuned model fitting the YOLO service model is mounted at the models folder and inference resumed upon rebuild of yolo service successfully. Note: This test requires the fine-tuned model to fit into the contract deployed model. This contract includes specifications sucha as Ultralytics-compatible format, expected class order, expected class names, and compatible detection/classification output behavior.

Test ID	TC-34

Category	Compatibility
Objective	Check that inference results remain valid for input streams with different supported resolutions.
Procedure	1. Prepare test streams of the same scene in different supported resolutions. 2. Run the CV subsystem on each stream separately. 3. Record the detected objects and anomaly decisions. 4. Compare the output labels across different test resolutions
Expected Results	The CV subsystem processes all supported resolutions successfully and consistently detects needed data points. Reports with the confidence score if resolution is below a certain threshold.
Priority	Medium
Date Tested	23.02.2026
Test Result	Passed

Test ID	TC-35
Category	Usability
Objective	Check that anomaly outputs shown on the interface are understandable and visually verifiable by the user
Procedure	1. Log in as an admin or operator. 2. Open a page where anomaly detections are listed. 3. Trigger or load a known anomaly case. 4. Inspect whether the interface shows ;

	a. the anomaly type, b. Timestamp, c. and related clip or frame evidence. 5. Ask the tester to identify the anomaly only from the displayed information.
Expected Results	The user interface presents anomaly results clearly enough for a non-technical user to understand what happened and verify the detection visually.
Priority	Medium
Date Tested	27.04.2026
Test Result	Pass

Test ID	TC-36
Category	Integration
Objective	Verify that a loading related event generated from the detection/backend flow is processed by the backbone event decision and persisted in PostgreSQL
Procedure	1. Start the required services to generate an event. 2. Create or select an active loading context with a known dock, truck and order. 3. Trigger a known event by sending a controlled detection payload to trigger product detection. 4. Query the PostgreSQL event table. 5. Verify event entry is stored with correct metadata such as time,

	event type etc.
Expected Results	The detection triggered event is successfully processed by the backend, stored with correct metadata. If applicable, the event is visible through notification service
Priority	High
Date Tested	02.05.2025
Test Result	Pass

Test ID	TC-37
Category	Functional
Objective	Check the notification system processing its parameters and displaying the right form of the notification successfully.
Procedure	1. Open the application. 2. Login as any role. 3. Open the notifications tab from the sidebar. 4. View and interact with the notification using buttons on it. 5. Observe that the notifications can come from different subsystems and with different purposes.
Expected Results	Based on the scope, notification can be sent as role based or username based. Notifications have different levels of severity. Different notifications have the same structure but can contain different contents and icons. Notifications can only be seen by the intended recipient, not by every account.
Priority	Medium

Date Tested	30.04.2026
Test Result	Pass

5.1.7 Route Planning Tests

Test ID	TC-38
Category	Functional
Objective	Verify that the route planning service recalculates the route when a new delivery order is added during an operation.
Procedure	1. Start with an active route 2. Approve a new delivery order 3. Observe updated route assignments for trucks.
Expected Results	The routing service recalculates routes and integrates the new order into the most optimal route without violating capacity or time constraints.
Priority	High
Date Tested	27.04.2026
Test Result	Pass

Test ID	TC-39
Category	Functional
Objective	Verify that the routing algorithm respects truck capacity constraints.

Procedure	1. Define truck capacity limits. 2. Create delivery orders whose total load exceeds the truck capacity. 3. Run the route planning service. 4. Observe generated routes and assigned deliveries.
Expected Results	The route planning system assigns deliveries without exceeding the truck capacity. Excess deliveries are assigned to another vehicle or marked as pending.
Priority	Medium
Date Tested	27.04.2026
Test Result	Pass

Test ID	TC-40
Category	Integration
Objective	Verify that damaged item detection from the CV subsystem affects routing decisions.
Procedure	1. Load items into a truck. 2. Simulate the CV subsystem detecting damaged cargo. 3. Check routing decision.
Expected Results	The shipment is flagged and excluded from delivery routes until resolved.
Priority	Low
Date Tested	01.05.2026

Test Result	Pass. We added a human verification step after the CV detections to avoid false results.

Test ID	TC-41
Category	Functional
Objective	Verify that deliveries are distributed across multiple trucks.
Procedure	1. Add orders exceeding one truck's capacity. 2. Define multiple trucks with different capacities. 3. Run route planning. 4. Observe route assignments.
Expected Results	Deliveries are distributed among trucks while minimizing distance and respecting capacity constraints.
Priority	Medium
Date Tested	29.04.2026
Test Result	Pass

Test ID	TC-42
Category	Integration
Objective	Verify that the routing system updates routes when a delivery order is cancelled.
Procedure	1. Generate a route with multiple stops 2. Cancel an order 3. Check the route plan

Expected Results	The cancelled order is removed and routes are recalculated to maintain optimal delivery sequences.
Priority	Medium
Date Tested	29.04.2026
Test Result	Pass

Test ID	TC-43
Category	Performance
Objective	Verify that the system performs efficiently under heavy workload conditions.
Procedure	1. Simulate high numbers of route planning requests. 2. Generate multiple CV events simultaneously. 3. Measure response time and system latency.
Expected Results	System maintains acceptable response times and route planning completes within the defined performance limits.
Priority	Medium
Date Tested	02.05.2026
Test Result	Pass

Test ID	TC-44
Category	Functional
Objective	Verify that route planning respects delivery time window constraints.

Procedure	1. Create delivery orders with specific time windows. 2. Run the route planning service. 3. Check assigned delivery order sequence.
Expected Results	Routes satisfy all time window constraints. If a route cannot meet the time window, the order is flagged or scheduled for another route.
Priority	Medium
Date Tested	30.04.2026
Test Result	Pass

Test ID	TC-45
Category	Functional
Objective	Verify system behavior when the routing algorithm cannot find a feasible solution.
Procedure	1. Define delivery orders with strict time windows and large load requirements. 2. Configure trucks with insufficient capacity or conflicting schedules. 3. Run the route planning service. 4. Check results.
Expected Results	The routing service identifies that no feasible solution exists and returns a clear error message. Some orders may also be skipped due to time constraints, which is also sent as a message.
Priority	Medium
Date Tested	01.05.2026

Test Result	Pass

Test ID	TC-46
Category	Integration
Objective	Verify that generated routes are correctly displayed on the frontend interface.
Procedure	1. Generate a route plan through the backend routing service. 2. Open the web interface. 3. Navigate to the route visualization page. 4. Load the generated route plan. 5. Inspect the displayed route information.
Expected Results	The interface correctly displays route paths, assigned trucks, delivery order sequence, and estimated arrival times without visual or data inconsistencies.
Priority	Medium
Date Tested	28.04.2026
Test Result	Pass

Test ID	TC-47
Category	Functional
Objective	Verify that routes are recalculated when truck capacity is changed
Procedure	1. Generate a route plan with deliveries assigned to a truck. 2. In the driver page update the trucks' capacity.

	3. Observe new delivery assignments.
Expected Results	The routing system removes excess deliveries from the truck and reassigns them to other trucks or future routes.
Priority	Medium
Date Tested	01.05.2026
Test Result	Pass

5.2 Test Cases for Non Functional Requirements

Test ID	TC-48
Category	Non-functional
Objective	Validate real-time inference latency constraints.
Procedure	1. Inject a precise timestamp payload into a continuous 60 FPS video frame at the streaming gateway. 2. Stream the video to the CV Subsystem. 3. Capture the resulting event payload at the backend integration layer. 4. Calculate the delta between the initial frame injection timestamp and the event generation timestamp.
Expected Results	The latency from video frame ingestion to event detection is strictly <= 1000 milliseconds.
Priority	High

Date Tested	28.04.2026
Test Result	PASS

Test ID	TC-49
Category	Non-functional
Objective	Verify that the YOLO manager can recover from a terminated per-stream inference worker
Procedure	1. Start the CV subsystem and simulation. 2. Confirm that YOLO workers are actively processing an RTSP stream. 3. Manually terminate the inference worker process. 4. Monitor the YOLO manager's logs and verify the manager detects the missing worker. 5. Verify that the manager detects the dead worker and starts a replacement worker.
Expected Results	The YOLO manager detects that worker has died, logs the failure and starts a new worker for the same running stream.
Priority	High
Date Tested	24.04.2026
Test Result	PASS

Test ID	TC-50

Category	Check that MediaMTX can handle multiple stream connections.
Objective	Non-functional
Procedure	1. Add at least 5 camera configurations as admin. 2. Open the live streams in Warehouse>Streams tab. 3. Display the streams switching among the camera checkboxes.
Expected Results	All streams can be displayed without bandwidth and delay problems.
Priority	High
Date Tested	03.12.2025
Test Result	Pass

Test ID	TC-51
Category	Non-functional
Objective	Check that the Sensor Polling Service can handle consecutive HTTP requests at 2-second intervals under system load.
Procedure	1. Start the Sensor Polling Service. 2. Preparing a high-load environment for the running services. 3. Observe the latency and interval timestamps.
Expected Results	Polling shows no latency even under high load.
Priority	Medium
Date Tested	07.04.2026
Test Result	Pass

Test ID	TC-52

Category	Non-Functional
Objective	Check that users can only see the determined features on their main page.
Procedure	1. Login the system with different roles. 2. Check the features provided on the page.
Expected Results	Users can only see the features related to their roles, for instance, the driver can only see the map, notifications and password change but the admin can see warehouse, map, streams and every other feature.
Priority	Medium
Date Tested	08.04.2026
Test Result	Pass

Test ID	TC-53
Category	Non-Functional
Objective	Check that users can only reach their authorized pages.
Procedure	1. Login the system with manager or operator role. 2. They can access the order handling page. 3. Then logout and login as driver. 4. The driver tries to route to the order handling or any monitoring page other than route monitoring.
Expected Results	The driver is led to the 403 no permission page.
Priority	Critical
Date Tested	12.03.2026

Test Result	Pass

6. Maintenance Plan and Details

TırGöz must be continuously monitored after deployment to ensure all services including the backend, frontend, database, object storage, video streaming, and computer vision modules to operate as intended. TırGöz requires periodic maintenance in order to prevent data loss, storage depletion since the system depends on real time video processing, data intensive database operations and containerised services. A detailed logging system must be maintained and checked for a year after deployment especially in early stage clients.

6.1 Monitoring and Health Status

Layer	Tool / Mechanism	What We Watch	Frequency
API Gateway	Nginx logs / health checks	Failed requests, routing errors, latency	Real-time
Backend API	Health endpoint and Docker logs	API availability, database connection errors	Once per 30s
YOLO Service	Worker logs and inference statistics	Model loading, frame latency, frontend frame age, crash loops	Real-time
MediaMTX	Stream logs	RTSP/WebRTC stream status, disconnected streams	Real-time
Cameras	Stream Health Monitor	FPS, ping latency, camera connection	Once per 10s
Sensor Service	Polling logs	Missing readings, delayed sensor responses	Once per 5s
PostgreSQL	Database logs	Storage growth, slow queries, event table size	Daily

MinIO	Object storage checks	Failed uploads, stored clip size	Daily
Host Server	Docker stats / disk usage	CPU/GPU, RAM, disk usage, container restarts	Once per 5 min

6.2 Scheduled Maintenance Tasks

Task	**Frequency**	**Automated**?	**Responsible Team**
Backup PostgreSQL database	Daily	Yes	Backend / DBA
Verify backup restoration	Weekly	Manual	Backend / DBA
Clean old anomaly clips from MinIO	Weekly	Yes	DevOps
Check container health and restarts	Daily	Semi-automated	DevOps
Review YOLO latency and memory logs	Weekly	Manual	CV Team
Validate camera and sensor connections	Weekly	Manual	System Admin
Rotate JWT signing keys	Monthly	Yes / Semi-automated	Backend
Re-index frequently used database tables	Monthly	Yes	DBA
Test new YOLO model compatibility	Before model update	Manual	CV Team

Run regression tests before release	Before each release	Manual	Whole Team

6.3 Model and Service Updates

Updates to models and related services must be handled carefully. The YOLO service depend on compatible model format, predefined class name and output contracts. These contracts must be upheld for inference service and the cooperation between the services. Event schemas rely on these contracts to be upheld.

Preceding the deployment of a new model, the currently working models must be backed up and the new model should be tested on a known validation stream and output events must be checked for its correctness. This is currently done manually but planned to be automated for future deployments of the project.

Service updates should be released in a controlled way. Changes should first be implemented on a separate branch, tested with Docker Compose, and verified with the related functional and integration tests. Special attention should be given to changes in event schemas, because the YOLO service, backend, database, notification system, and frontend all depend on consistent event payloads.

6.4 Scaling and Capacity Planning

TırGöz is designed to be able to work on a local warehouse server. For larger deployments, the first option is improving the local server with more RAM, better CPU or GPU support. As YOLO services a resource heavy component further GPU acceleration can significantly improve inference time.

The most resource demanding service is the YOLO inference service because it proceses live camera frames and runs multiple specialiser models. If the number of the streams increase the system might need GPU scaling, lower frame sampling frequencies or increased number of workers for inference.

Vertical scaling can be used and the local host server can be upgraded with more RAM, a stronger CPU or CUDA enabled GPU with higher VRAM. This is especially needed for increasing the number of RTSP streams without decreasing inference throughput. For larger deployments, GPU acceleration with NVIDIA GPU using compatible PyTorch, ONNX runtime or TensorRT backend can be run on the system. Horizontal scaling is another option as the services are containerised, independent services such as the backbone API, frontent, simulation service and YOLO workers can be spread across different machines if more fit to the client's needs and resources. Per stream scaling can be used to assign different camera groups so the cameras with typically higher inference demands do not bottleneck the system. This approach is also possible as an extension of horizontal scaling.

6.5 Backup Strategies

Automated daily backups are scheduled for the PostgreSQL database. These backup dumps are securely transferred to a separate physical storage drive or an off-site local network attached storage to prevent data loss if the main server's disk fails. Redis data is treated as volatile and is intentionally excluded from backups to optimize resource usage. Also, the recovery protocol involves replacing a local server and deploying the system via its containerized Docker setup. Administrators will pull the latest Docker images, mount the most recent PostgreSQL backup volume, and remap the static IP configurations to reestablish connections with the existing cameras and sensors.

7. Other Project Elements

7.1 Consideration of Various Factors in Engineering Design

This section analyzes the impact of critical engineering factors on the design and development of the TırGöz platform. It provides a comprehensive evaluation of how these considerations shaped our technical decisions to ensure the system is not only functional, but also sustainable and socially responsible.

7.1.1 Constraints

Technology and Platform Constraints: The system's architecture and development workflow are shaped by its reliance on RTSP security cameras, limiting input sources to devices capable of delivering real-time video streams. This requirement influences hardware selection and ensures compliance with streaming standards. The Computer Vision pipeline must also rely on real-time inference frameworks, such as YOLO variants, that support video decoding and low-latency execution [5]. On the backend, a containerized Docker deployment must be used, and the frontend must rely on cross-platform web technologies, thereby preventing the use of platform-specific UI toolkits.

Real Time Processing Constraints: The Computer Vision module, including decoding, detection, and event generation, must operate within a latency of 1,000 ms. Any computationally heavy approaches that exceed this limit cannot be used. Continuous real time streaming is required; paused or delayed frames are unacceptable because operators must react instantly to operational issues [6]. The Route Optimization module must also compute updated routes within a tight time budget as warehouse and truck states change. Since vehicle routing is NP-hard, exact solvers (e.g., full linear or integer programming) are not feasible for real-time use, making heuristic or approximation algorithms mandatory [7].

Integration Constraints: All modules must communicate through standardized REST/HTTP and web sockets to retain consistency across components. Data must be stored in the designated storage layers: a relational database for structured truck, warehouse, and configuration data; Redis for cached real-time states and temporary results; and MinIO object storage for videos and images [8]. This separation prevents ad hoc storage patterns and ensures predictable access. The Computer Vision module must also adhere to a predefined event logging schema to maintain traceability and ensure that event data is consumable by other services.

Resource Constraints: The entire pipeline must be optimized for efficient memory and compute use. Video parameters, including clip duration, resolution, compression, bitrate, and FPS, must be controlled to avoid unnecessary storage consumption. The Route Optimization module must also utilize algorithms that remain efficient as the problem size increases, since memory usage in distance matrices, cost tables, and constraint sets grows rapidly. As a result, industry-standard optimization techniques are required.

Security and Compliance Constraints: All communication, both video streams and API traffic, must use secure protocols such as HTTPS/TLS. GDPR and KVKK regulations impose strict retention limits on video clips, event logs, and operational records, mandating the implementation of automatic deletion mechanisms. Role-based authentication and authorization ensure that each user type (operators, security staff, administrators) only accesses data and functions appropriate to their responsibilities.

Testing and Deployment Constraints: Because live camera access is not always available during development, the system must support testing with recorded videos to enable repeatable evaluations. Testing primarily focuses on modular validation, controlled simulations, and small-scale experiments, rather than full industrial deployment scenarios.

Economic Constraints: Since the project must operate without enterprise-grade resources, such as GPU servers, specialized networking systems, or high resolution cameras. All components must run on standard workstations and low-cost development servers. This limits model complexity and prevents the use of computationally expensive solvers. To control long-term operational costs, the system cannot rely on paid cloud services, commercial APIs, or licensed optimization software; instead, it must use open source libraries and cost efficient storage solutions. Video storage must be managed through clip duration limits, compression, and automated retention policies. Camera acquisition is also constrained to affordable RTSP compatible devices already deployed in warehouses.

7.1.2 Standards

UML 2.5.1: Used for all modeling artifacts (use case, class, sequence, state, activity diagrams). Ensures a consistent and standardized representation of system structure and behavior [9].

IEEE 29148: Defines structure and quality criteria for functional and non-functional requirements, ensuring clarity, correctness, and traceability [10].

IEEE 1058: Provides guidelines for project management documentation such as work packages, schedules, team roles, and deliverables.

IEEE 1016: Specifies how software design descriptions should be structured, supporting architectural clarity, interface definitions, and design documentation.

ISO/IEC/IEEE 12207: Defines the software lifecycle processes across planning, development, testing, deployment, and maintenance.

ISO/IEC 27001: Guides information security practices, including access control, secure storage, authentication, authorization, and logging, ensuring confidentiality, integrity, and availability.

ISO/IEC/IEEE 29119: Provides a structured approach to testing, including test planning, test design, execution, and reporting.

IEEE 829: Used conceptually to structure test cases and test documentation.

REST Principles & JSON Schema Standards: Define how microservices communicate, specifying consistent request/response structures and data validation rules.

7.2 Ethics and Professional Responsibilities

During the development of the TırGöz project, the team recognized several ethical and professional responsibilities. We prioritized user privacy by ensuring that video surveillance is limited to logistics-related tasks, intentionally avoiding the collection of unnecessary personal data and ensuring the system is not repurposed for employee monitoring beyond the project scope. To uphold data protection standards, the system architecture incorporates automated deletion mechanisms and strict retention limits for video clips and operational records. Professionally, we emphasized transparency and accountability by designing automated outputs and alerts that are clear and interpretable, ensuring the system supports rather than replaces human decision-making. Furthermore, we maintained technical integrity by utilizing small-scale experiments and modular validation to ensure the reliability and safety of both the perception and planning layers.

7.3 Teamwork Details

7.3.1 Contributing and functioning effectively on the team to establish goals, plan tasks, and meet objectives

During development, each member contributed effectively to different technical areas of the system to meet the project's core objectives. Burak Başţuğ primarily focused on authorization, authentication, and testing processes to ensure system requirements were validated. Arda Öztürk and Berin Su İyici worked on the computer vision inference pipeline, successfully developing mechanisms for item counting, anomaly detection, and training related models. Umut Başar Demir focused on the route planning component to dynamically generate and update delivery routes based on operational constraints. Berkin Kağan Ateş implemented the streaming pipeline, core backend services, and the web interface. Furthermore, the team functioned effectively as a unit by continuously reviewing each other’s work and assisting with cross-component integration tasks to ensure the overall system operated smoothly.

7.3.2 Helping creating a collaborative and inclusive environment

The team emphasized open communication throughout the development lifecycle. Collaboration was supported by using GitHub for version control, ensuring that multiple members could contribute simultaneously through development branches and that all code was peer-reviewed prior to merging. We held regular meetings to maintain a shared understanding of the evolving architecture and current project status. To foster a highly inclusive environment, team members actively supported each other in learning unfamiliar technologies, such as computer vision pipelines and Websocket usage. Spontaneous technical discussions allowed experienced members to guide others, ensuring that everyone could actively participate and improve their technical skills throughout the project.

7.3.3 Taking lead role and sharing leadership on the team

Leadership within the TırGöz project was distributed among team members along the boundaries of the microservice architecture. This decentralized approach ensured that decision-making responsibilities were shared equitably. Each subsystem had a designated

member responsible for taking the lead role in its development tasks and ensuring that both implementation and report deliverables were completed on time. All members held authoritative decision-making capabilities within their designated areas, while carrying the professional responsibility to notify others and solicit feedback when necessary. Despite these specific leadership roles, major architectural choices were discussed and decided upon collaboratively as a team.

7.3.4 Meeting objectives

Throughout the project, the team monitored the initial milestones and objectives established in the project plan during the Analysis phase. The primary objective of developing a functional, real-time computer vision pipeline was successfully met, with the YOLO inference service operating well within the targeted constraint. The route optimization milestone was also fully realized, successfully integrating live warehouse occupancy metrics with vehicle routing algorithms. Infrastructure objectives, including the deployment of a containerized microservices architecture, secure WebRTC streaming, and role-based access control, were completed at a high level of proficiency. While minor timeline and scope adjustments were made the overall system integration and final deliverables were completed on schedule. Ultimately, all critical functional and non-functional objectives defined in the initial project plan were met at a production-ready level.

7.4 New Knowledge Acquired and Applied

During this phase, our team gained practical experience working across the full TirGoz ecosystem and learned how a multi-repository system can be organized and evolved in a coordinated way. In our backbone repository, we used a layered FastAPI architecture built around api, crud, model, and schema modules, and we learned how to combine database initialization, default data seeding, RFID stream handling, and real-time WebSocket communication for detection and anomaly processing. Our recent commits in this repository focused on anomaly recording, CRUD-based data flow, and backend event publishing, which strengthened our understanding of how to keep business logic in the correct layer. In the frontend repository, we worked with a React + TypeScript application structured through pages, components, containers, contexts, constants, and theme folders, and we improved route protection, localization, and live UI synchronization. Also, such as for recording deletion behavior, WebSocket-driven updates, and user interface refinements, which helped us

understand how frontend state must stay aligned with backend events. In our test workspace, we learned how simulation streams, MediaMTX-based RTSP transport, and YOLO-based anomaly detection are connected, including the TCP-based streaming configuration and the forwarding pipeline used to reliably deliver detection results. We applied service orchestration knowledge through Nginx gateway routing, environment-based configuration, and coordinated startup of the frontend, backend, simulation, Redis, MinIO, and streaming services.

8. Conclusion and Future Work

8.1 Conclusion

In conclusion, the logistics industry counters issues such as loading/unloading goods, security management, and route optimization. TırGöz aims to solve these problems by detecting related events during truck docking sessions, ensuring good tracking, conducting security checks, and optimizing truck route selection after leaving the warehouses by leveraging upcoming orders to the warehouse. TırGöz improves efficiency by reducing manual operations, eliminating incorrect loading, optimizing travel routes, and detecting anomalies in warehouses through a real-time notification system.

TırGöz presents these valuable features using microservices with different architectures for the modules that address different types of problems in warehouses. It uses web streaming technologies such as WebRTC to relay the security camera streams to the YOLO services for processing the frames to detect anomalies, perform inventory counting validations, conduct security checks, etc. It works as an integrated part of the different sensors and cameras and handles async/sync requests using Postgres, Redis, and Minio. It aims to provide an isolated hardware-software service to the warehouses without an external cloud service.

8.2 Future Work

TırGöz can provide the core capabilities for solving problems such as route optimization, anomaly detection, inventory control, and security checks. However, it still has some minor constraints for warehouse managers, such as the need to use ArUco barcodes on containers and a dock camera setup to improve the detection of goods during loading and unloading.

Warehouse managers might want to continue using their own barcode systems rather than ArUco barcodes. Even though the current barcode system is better suited to computer vision detection algorithms, a better YOLO model can be fine-tuned to adapt to each warehouse's barcode alphabet.

Dataset availability for warehouse components such as boxes, forklifts, and worker equipment is limited for training models for every warehouse environment. Therefore, a synthetic dataset can be created with very specific, detailed information for specialized warehouses.

Recording anomalies and notifying the managers might be enough for the first release, but indicating the precise location of this anomaly in a 3D digital twin of the warehouse’s blueprint can be more effective and user-friendly.

In the current version of the system, the model used in the specific RTSP stream is determined manually by the admin user. However, this decision can also be automated, building an agentic decision pipeline. Thus, the rigid constraints imposed by the camera-model matching can be eliminated, and multiple stream sources can be used more efficiently.

9. Glossary

This section provides definitions of commonly used concepts, terms, and abbreviations.

Term	**Definition**
RTSP	A networking protocol used for streaming video from security cameras.
CCTV	Also called Closed Circuit Television. A type of surveillance camera.
YOLO	A deep learning model for object detection

Ad hoc	A temporary solution that does not meet the standards of the system
Anomaly Detection	Detecting unusual events and abnormal patterns.
Occupancy Estimation	Process of approximating how full a space is, in our case, a warehouse and a truck load.
Heuristic Algorithms	Fast and rule-based solutions that determine good solutions. Used in our case because route optimisation is computationally expensive.
Meta-Heuristic Algorithms	Problem-agnostic frameworks that guide and improve heuristic rules.
Vehicle Routing Problem	An NP-hard optimisation in the field of logistics. The goal is to find optimal route sets for a fleet.
Dynamic Routing	Real-time adjustments of delivery routes based on new information such as load delays, traffic.
OCR	Abbreviation for Optical Character Recognition, a technology for converting some sort of image into editable and/or machine readable data.

10. References

[1] A. D. Patel and A. R. Chowdhury, “Vision-based object classification using deep learning for inventory tracking in automated warehouse environment,” in *Proc. 2020 20th Int. Conf. Control, Automation and Systems (ICCAS)*, Busan, Korea, Oct. 2020, pp. 1–6, doi: 10.23919/ICCAS50221.2020.9268394.

[2] “Develop with Redis,” Redis Documentation, 2025. [Online]. Available: https://redis.io/docs/latest/develop/. Accessed: Nov. 21, 2025.

[3] *SQLAlchemy Project*, “*SQLAlchemy Documentation*.” [Online]. Available: https://www.sqlalchemy.org/. Accessed: Apr. 21, 2026.

[4] J. Liu, X. Wang, and Y. Zhang, "Anomaly Detection in Logistics Warehouses Based on YOLOv8," in *2024 IEEE International Conference on Logistics and Supply Chain Management (LSCM)*, Shanghai, China, 2024, pp. 112-118.

[5] H. Wang et al., "Dynamic Clustering for Multi-Depot Capacitated Vehicle Routing with Time Windows: A CW Heuristic Approach," in *2024 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM)*, Singapore, 2024, pp. 98-103.

[6] Z. Zhang, "Heuristics for Vehicle Routing Problem: A Survey and Recent Advances," *IEEE Access*, vol. 11, pp. 12345-12360, 2023.

[7] “Develop with Redis,” Redis Documentation, 2025. [Online]. Available: https://redis.io/docs/latest/develop/. Accessed: Nov. 21, 2025.

[8] “About the Unified Modeling Language Specification Version 2.5.1.” Www.omg.org, www.omg.org/spec/UML/2.5.1/About-UML.

[9] “IEEE Standards Association.” IEEE Standards Association, standards.ieee.org/ieee/29148/6937/.

[10] “ISO/IEC 27001 Standard – Information Security Management Systems.” ISO, 2022, www.iso.org/standard/27001.