ONE4ALL - Agile and modular cyber-physical technologies supported by data-driven digital tools to reinforce manufacturing resilience

Project nr: **101091877**

D3.4 – IOP infrastructure, modules and performance

Version: 1.0

Funded by the European Union

ONE4ALL project is funded by the European Union under the Grant Agreement No 101091877. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or HADEA. Neither the European Union nor the granting authority can be held responsible for them.

Characteristics of deliverable	
Title	ONE4ALL
Partner	IDE
Contributors	SDU, INNO, KIT, IBT, HOLO
Short description of deliverable	Intelligent Orchestration Platform infrastructure and technologies implemented in the development. First report
Submission date	30/06/2024
Type	Other
Audience	Public
Keywords	Data Platform, Smart Manufacturing, Services Orchestration, DevOps, Microservice Architecture

ONE4ALL Key Facts

Acronym	ONE4ALL
Project title	Agile and modular cyber-physical technologies supported by data-driven digital tools to reinforce manufacturing resilience
GA n°	101091877
Starting date	01/01/2023
Duration-months	48
Call (part) identifier	CLIMATE NEUTRAL, CIRCULAR AND DIGITISED PRODUCTION 2022 (HORIZON-CL4-2022-TWIN-TRANSITION-01)
Type of Action	HORIZON-RIA HORIZON Research and Innovation Actions
Topic identifier	HORIZON-CL4-2021-TWIN-TRANSITION-01-03
Consortium	11 organisations, all EU Member States
Model GA type	HORIZON Action Grant Budget-Based

ONE4ALL Consortium Partners

N.	Partner	Acronym	Country
1	IDENER RESEARCH & DEVELOPMENT AGRUPACION DE INTERES ECONOMICO	IDE	ES
2	INNOPHARMA RESEARCH LIMITED	INNO	IE
3	CRIT CENTRO DI RICERCA E INNOVAZIONE TECNOLOGICA SRL	CRIT	IT
4	EXELISIS IKE	EXE	EL
5	SYDDANSK UNIVERSITET	SDU	DK
6	INNOBOTICS	IBT	IT
7	MADAMA OLIVA SRL	MAD	IT
8	HOLOSS - HOLISTIC AND ONTOLOGICALSOLUTIONS FOR SUSTAINABILITY, LDA.	HOLO	PT
9	TECHNISCHE UNIVERSITAT DORTMUND	TUDO	DE
10	Orifarm Supply S.R.O.	ORI	CZ
11	Karlsruher Institut Fuer Technologie	KIT	DE

Disclaimer

Funded by the European Union. Views and opinions expressed are, however, those of the author(s) only and do not necessarily reflect those of the European Union or HADEA. Neither the European Union nor the granting authority can be held responsible for them.

Executive Summary

Deliverable 3.4 IOP infrastructure, modules, and performance provides an update of WP3 advances, mainly T3.3. IOP Infrastructure tools—development and validation. The main goal of T3.3 is to develop a solid backend infrastructure for ONE4ALL IOP. During this period, the focus has been on constructing the architecture's structure and the technologies that underpin it. Furthermore, data sources and ONE4ALL modules, such as the RCPM, should be integrated.

This deliverable is the first report of the Intelligent Orchestration Platform (IOP) of ONE4ALL projects. It aims to focus mainly on how the architecture of the IOP and its modules have been developed based on the technological scouting and following work presented in D3.1. Additionally, the submission of this deliverable concurs with the end of stage 1 of the ONE4ALL project; the stage focused on developing the innovative solutions proposed. The advances performed within the following validation stage, installation and demonstration will be presented in the upcoming versions of this deliverable. Two more versions of this deliverable are expected by M36 and M48, which will deepen the implementation, deployment, and performance in line with the upcoming stages of the project.

The deliverable starts with an overview of the architecture baseline for the IOP, *Kappa Architecture*, which has already been detailed in depth in D3.1. Further on, the next chapter of the deliverable, *Chapter 2 – IOP Architecture and Foundation Technologies*, details how the IOP architecture is structured in practice and especially how the foundation technologies make it work. On the one hand, the structure is divided into layers: *Ingestion Layer*, *Storage Layer*, *Processing Layer* and *Serving Layer*, each detailed in its chapter, from 3 to 6 accordingly. On the other hand, foundation technologies enable and sustain modularity, interoperability, reconfigurability and smart orchestration, each a key aspect of the IOP. Within these technologies are presented Docker and Kubernetes, the first solving the modularity and interoperability issues, and the second ensuring a smart orchestration of the services while ensuring their accurate reconfigurability when needed.

Additionally, in Chapter 2, there are details on how the development workflow of the IOP works, integrating concepts such as continuous integration and continuous deployment, facilitating the scale-up and modular work-frame. Chapters 3 to 6 go deeper within the IOP structure and detail how each layer aims, how it has been constructed, and how each works in practice. Practical examples are presented with technical information and template codes to ease understanding. An update on the security aspects is presented to conclude the IOP infrastructure definition. A more detailed view of the security aspects can be found in deliverable 3.2. Within this deliverable, the focus has been on only updating the more transversal points and those intrinsic to the IOP architecture and technologies.

Now that the foundation has been settled, the next step focuses on completing the integration of ONE4ALL technologies and external modules (i.e. demonstration sites MES systems); secondly, scale-up, test and validation; and lastly, advance in developing the IOP frontend.

Table of contents

List of acronyms

API	Application Programming Interface
CD	Continuous Deployment
CI	Continuous Integration
DDoS	Distributed Denial of Service
DMP	Data Management Plan
DT	Digital Twin
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
IOP	Intelligent Orchestration Platform
IoT	Internet of Things
k3s	Kubernetes
KPI	Key Performance Indicator
MES	Manufacturing Execution System
MQTT	Message Queuing Telemetry Transport
ORM	Object-Relational Mapping
RCPM	Reconfigurable Cyberphysical Production Module
ROS	Robot Operating System
UI	User Interface
YOLO	You Only Look Once

List of figures

List of tables

No table of figures entries found.

1. Introduction

The IOP is a cloud-based framework accessible remotely through any computer or mobile phone. However, the mobile phone connection will mainly be implemented for monitoring and user authentication. In regards to the foundation of the IOP it is constructed using a microservices-based architecture. This software architecture pattern structures the IOP as a collection of small, independent, and loosely coupled services, each responsible for a specific business capability. Each microservice is designed to be independently deployable, scalable, and replaceable and communicates with other microservices through lightweight protocols, typically HTTP or message queues.

In a microservices architecture, each service is developed, tested, and deployed separately, allowing for greater flexibility and faster development cycles. This approach also allows for better fault isolation, as a failure in one microservice does not necessarily affect the entire application. Additionally, microservices architecture allows for more efficient resource use by scaling individual services based on their specific usage patterns rather than scaling the whole application.

1.1. Kappa Architecture

The Kappa Architecture (Figure 1) is a software architecture used for processing streaming data. The main premise behind the Kappa Architecture is that you can perform real-time and batch processing, especially for analytics, with a single technology stack. It is based on a streaming architecture in which an incoming data series is first stored in a messaging engine like Apache Kafka. Subsequently, a stream processing engine will retrieve the data, convert it into a format suitable for analysis, and then deposit it into an analytics database for easy querying by end users.

.

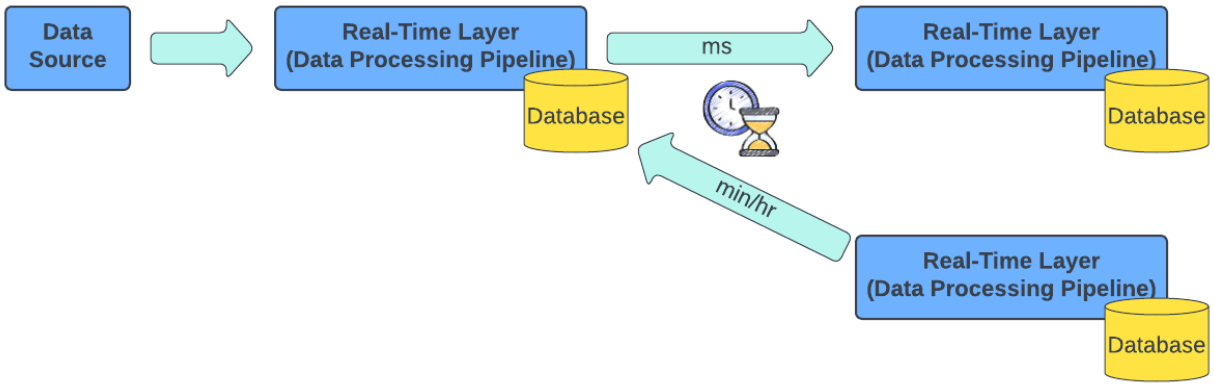

Figure 1: Kappa Architecture

The Kappa Architecture supports (near) real-time analytics when the data is read and transformed immediately after it is inserted into the messaging engine. This makes recent data quickly available for end-user queries. It also supports historical analytics by reading the stored streaming data from the messaging engine later in a batch manner to create additional analysable outputs for more types of analysis.

2. IOP Architecture and foundation technologies

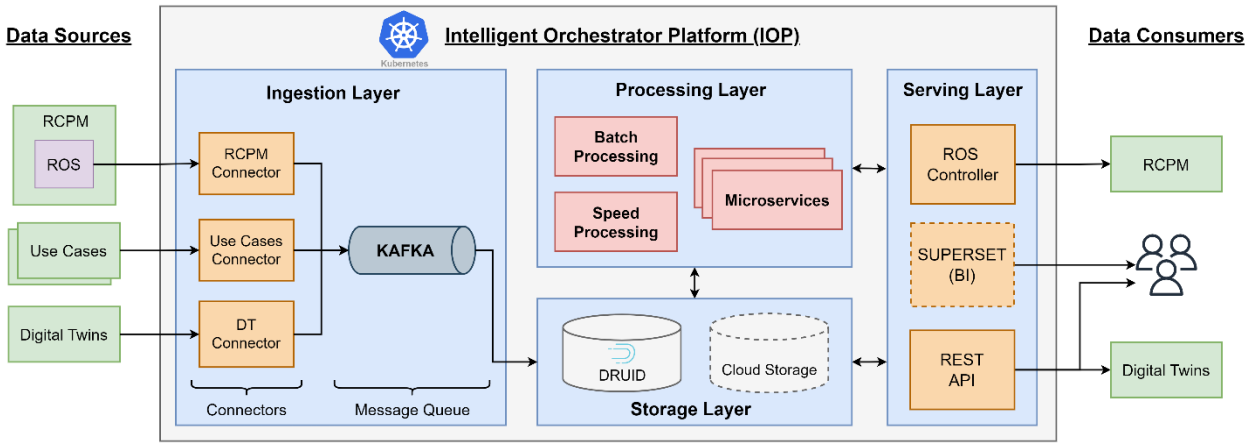

Figure 2: IOP architecture simplified overview

There are 3 components in the architecture, as reflected in Figure 2:

- **Data Sources:** all the different data producers external to the IOP.
- **IOP:** the central orchestrator and processing platform.
- **Data Consumers:** client applications that receive processed data and control signals. Also includes the user interface.

The IOP is, in turn, composed of 4 elements: the ingestion layer, the storage layer, the processing layer, and the serving layer. Modules in the IOP will be deployed as Docker containers orchestrated by Kubernetes.

2.1. Docker

Docker provides a standardised way to package applications and their dependencies into containers. Containers are lightweight, provide a consistent environment for the application across different stages of development, testing, and production, and can start up quickly. Docker also allows for easy version control and distribution of containers, which can be especially beneficial in a microservices architecture where many different services need to be managed and updated independently. By using Docker, the benefits of isolation (each service runs in its container), portability (containers can run on any system that supports Docker), and resource efficiency (containers share the host system's kernel and start faster than virtual machines) are achieved.

2.1.1. How does it work?

As follows is presented a practical step-by-step guide from creating a Dockerfile to pushing the image to a registry:

1. **Write a Dockerfile:** A Dockerfile is a text document that contains all the commands a user could call on the command line to assemble an image. It defines the environment inside the container, including the operating system, software versions, and application code. Here's an example Dockerfile for a simple Python application:

```
# Use the official Python image as base image
FROM python:3.11-slim

# Set the timezone
ENV TZ="Europe/Madrid"

# Set the working directory in the container
WORKDIR /app

# Copy the dependencies file to the working directory
COPY requirements.txt.

# Install any needed packages specified in requirements.txt
RUN pip3 install --upgrade --no-cache-dir -r requirements.txt

# Copy the content of the local src directory to the working directory
COPY src.

# Run the application
CMD ["main.py"]
ENTRYPOINT ["python3"]
```

2. **Build the Image:** After creating the Dockerfile, use the Docker build command to create a Docker image. This image is a snapshot of the application and its environment.

```
docker build -t registry.idener.es/one4all/my-app.
```

In this command, -t tags the image with a name (registry.idener.es/one4all/my-app) and tells Docker to look for the Dockerfile in the current directory.

3. **Push the Image to the Private Registry:** After building the image, it can be pushed to the private registry using the docker push command.

```
docker login registry.idener.es

docker push registry.idener.es/one4all/my-app
```

The docker login command authenticates you with the private registry (registry.idener.es). After successful authentication, the docker push command uploads the image to the registry.

The Docker image is now stored in the private registry and is available to be pulled and deployed onto any environment, including Kubernetes clusters. Pushing the image to a private registry is secured and accessible only to authorised users within your organisation or deployment pipeline, ensuring that it is ready for deployment with Kubernetes.

2.2. Kubernetes

Kubernetes is being used because it is a powerful container orchestration tool that automates containerised applications' deployment, scaling, and management. In a microservices architecture, where there could be a large number of microservices to manage, Kubernetes provides features such as service discovery, load balancing, self-healing (automatically restarting failed containers), automated rollouts and rollbacks, and secret and configuration management. By using Kubernetes, we can ensure that the IOP can handle dynamic scaling based on load, maintain high availability, and enable smooth updates and maintenance of services without downtime. Kubernetes also supports a wide range of environments, including on-premises, public cloud, and hybrid cloud, which adds to its flexibility. The integration with Docker ensures that containers can be efficiently managed across different environments, providing a seamless workflow for continuous integration and continuous delivery (CI/CD) practices.

2.2.1. k3s: Lightweight Kubernetes

K3s is a highly lightweight distribution of Kubernetes designed for ease of installation and operation. It is particularly suitable for edge computing, IoT, CI/CD environments, and development purposes where full-scale Kubernetes might be overkill due to its operational complexity and resource requirements. Here's a breakdown of what k3s is, why it's chosen and how it's installed:

2.2.1.1. What is k3s?

K3s is a fully compliant Kubernetes distribution with the following features:

- **Lightweight:** It has a significantly reduced binary size compared to upstream Kubernetes, which makes it fast to install and easy to run, even in resource-constrained environments.
- **Simplified operation:** K3s removes less common features and legacy, alpha, and non-default features that can make Kubernetes challenging to operate.
- **Single binary:** All necessary Kubernetes cluster components are included in a single executable, simplifying the setup process.
- **Reduced dependencies:** K3s includes a lightweight storage backend based on SQLite3 as the default storage mechanism, eliminating the need for etcd in certain use cases. Nevertheless, both techniques will be implemented, exploiting their advantages on the need.
    - ***etcd*** *is a distributed key-value store designed to securely store configuration data and metadata for distributed systems. It is a critical component in many K3s installations, where it acts as the primary data store for all cluster states and configurations.*
    - ***SQLite3*** *is a self-contained, high-reliability, embedded, full-featured, public-domain SQL database engine. Among its advantages:* ***simpler*** *to manage and set up in comparison to traditional distributed storing systems like etcd;* ***lightweight****, reduced resource consumption (e.g. CPU, memory);* ***embedded****, can be embedded directly into applications, simplifying the deployment and operational complexity.*

2.2.1.2. Why Choose k3s?

K3s is chosen in the project for its simplicity and minimal resource requirements, which are ideal for managing microservices efficiently. K3s makes it possible to have a fully functional Kubernetes cluster that is less demanding on system resources, which is essential in environments with limited computing resources. Additionally, k3s' streamlined installation and setup process enables quick deployments and iterations on the applications, speeding up development and testing workflows.

K3s also supports most of the features of a full Kubernetes distribution, ensuring that essential capabilities like scaling, self-healing, and service management are not missed. These features allow

the developers to focus on developing and deploying their applications without worrying about complex cluster management.

2.2.1.3. How is K3s installed?

Installing k3s is straightforward and can be done with a single command, which downloads and executes the installation script. Bellow there is presented a step-by-step basic overview of the installation process followed:

1. **Prepare the Nodes:** Ensure that each machine meets the minimal requirements for k3s, such as a supported operating system and architecture, and has network connectivity.
2. **Install k3s on the Server Node:** Run the installation script on your server node with the following command:

```
curl -sfL https://get.k3s.io | sh -
```

   This command will download the k3s installation script and execute it, which sets up the Kubernetes master node.

3. **Get the Node Token:** Once the master is set up, you need to retrieve the node token, which is required to join worker nodes to the cluster. The token can be found in */var/lib/rancher/k3s/server/node-token* on the server node.

4. **Install k3s on the Worker Nodes**: On each worker node, run the installation command using the node token and the server's IP address:

```
curl -sfL https://get.k3s.io | K3S_URL=https://<server-node-ip>:6443
K3S_TOKEN=<node-  token> sh -
```

5. **Verify the Cluster:** After installation, verify the cluster by running `kubectl get nodes` on the server node, which should show the server and all worker nodes in a Ready state.

A working k3s cluster is achieved by following these steps, ready to deploy any applications.

2.3. Development workflow

Before diving into the development workflow, it's crucial to understand the concepts of Continuous Integration (CI) and Continuous Deployment (CD) and the role of GitHub Actions in automating these processes.

2.3.1. Continuous Integration (CI)

Continuous Integration is a development practice where developers frequently merge their code changes into a central repository, preferably several times daily. Each merge triggers an automated build and testing process, which provides immediate feedback on the integration's success. This practice aims to identify issues early in the development cycle, making them more manageable and less costly to resolve.

2.3.2. Continuous Deployment (CD)

Continuous Deployment is the next step after Continuous Integration. Every change that passes the automated tests in CD is automatically deployed to the production environment without a developer's explicit approval. Therefore, software updates are delivered to users as soon as they are ready, ensuring that the product evolves quickly and efficiently.

2.3.3. GitHub Actions

GitHub Actions is a CI/CD platform that automates all software workflows, with CI/CD at its heart. The chosen CI/CD pipeline builds the container image, pushes it to a private registry and deploys the new version on the server (via Kubernetes rollout). All this process is triggered when a new release (tag) is created on the repository. The pipeline is directly integrated into GitHub; Actions allows you to automate your build, test, and deployment pipeline without leaving the GitHub ecosystem. You can write individual tasks, called actions, and combine them into workflows to run on certain events within your repository, such as a push, a pull request, or a release.

The development workflow is significantly enhanced through the implementation of CI/CD, with GitHub Actions serving as the automation engine for these processes. Here's a detailed look at the steps involved in this automated workflow:

1. **Code Changes and Commit:** Developers write code in their local development environment. Once they've implemented a new feature or fixed a bug, they commit their changes to a feature branch in the GitHub repository.
2. **Code Review and Merge:** The changes are then submitted for review through a pull request. Team members analyse the code for quality, performance, and alignment with project standards. Upon approval, the changes are merged into the main branch.
3. **Creating a Release:** When the team is ready to deploy, they create a new Release via the GitHub UI. This release is tagged with a version number that follows semantic versioning (e.g., v1.0.2). The tag signifies an immutable snapshot of the codebase at a specific time and marks a version ready for deployment.
4. **Building the Docker Image:** Triggered by creating a new release, a GitHub Action workflow starts the automated process of building the Docker image. It uses the Dockerfile to package the application and its dependencies into a container image.
5. **Pushing the Image to the Registry:** Once the image is built, another GitHub Action workflow securely pushes it to the private container registry (e.g., registry.idener.es), tagging it with the release version.
6. **Updating the Kubernetes Deployment:** With the new image available in the registry, an additional automated workflow initiates a rolling update to the Kubernetes deployment. This update references the new image tag and gracefully replaces the existing pods with new ones running the updated application.
7. **Monitoring and Verification:** After the deployment, monitoring tools within the Kubernetes environment watch for the health and performance of the application. Any issues trigger alerts so that developers can address them promptly.

By automating these steps, the CI/CD pipeline reduces manual errors, saves time, and ensures that the software delivery process is consistent and repeatable. It enables developers to focus on writing code while the pipeline handles the verification, build, and deployment processes, leading to a more reliable and efficient release cycle.

3. Ingestion – Layer 1

The ingestion layer serves as the entry point for data. It is responsible for collecting, importing, and organising data from various data sources. This data could originate from different places, such as databases, log files, external APIs and IoT devices.

The primary functions of the ingestion layer include:

1. Data Collection: It gathers data from various sources and formats. It might involve batch ingestion (where data is collected at periodic intervals) or real-time ingestion (where data is collected as soon as it is produced).
2. Data Transformation: The ingested data may not be in a suitable format for processing or analysis. The ingestion layer might need to perform necessary transformations, like converting data formats normalising or denormalising data.
3. Data Validation: This stage ensures the quality and accuracy of data. It may involve checks for the completeness of data, detecting duplicates, or validating the data format.
4. Data Storage: The ingestion layer stores the data in a suitable format for further processing.

The ingestion layer can be broken down into two primary components: connectors and the message queue.

3.1. IOP connectors – Data Pipelines

The role of the connectors is to interface with various data sources and fetch the required data. Furthermore, they work as **data fusion pipelines**. In this sense, the connectors are responsible for understanding these sources' data protocols and formats. For example, a connector for a ROS-based robot would need to understand ROS queries, while a connector for a web API would need to know how to handle HTTP requests and parse responses. Connectors also handle security and authentication measures these data sources require (like TLS encryption).

Connectors also often handle some level of data transformation, converting the data from its source format into a format suitable for the message queue and the rest of the pipeline. In this line, usually a reconfiguration of the connector is required. The reconfiguration might involve changing the data types, encoding/decoding data or handling missing data.

3.1.1. MQTT Connector

MQTT (Message Queuing Telemetry Transport) is a lightweight, publish-subscribe network protocol for efficient, reliable message delivery in resource-constrained environments, such as embedded systems or mobile devices. It is commonly used in Internet of Things (IoT) applications due to its low bandwidth requirements and ability to operate over unreliable networks. MQTT operates on a broker-based architecture, where clients publish messages to specific topics, and the broker forwards these messages to clients subscribed to those topics.

An MQTT broker deployed in the IOP serves as an entry point for MQTT-enabled devices. The broker is implemented with Eclipse Mosquitto, an open-source (EPL/EDL licensed) message broker. In conjunction with the broker, a custom Python script has been developed that reads messages from the broker and publishes them to kafka.

More details on how the IoT devices work and their MQTT connection to the IOP can be found in D4.2.

3.1.2. ROS2 Connector

ROS2, a framework used for modelling and controlling robots, is being used alongside Kafka for communication with the real robot. The ingestion layer contains a specific ROS2 module called "Connector," which receives messages from the robot and transfers them through Kafka Topics. Further information regarding the parameters obtained from the robot and the communication format can be found in D4.7.

3.1.3. OPC-UA Connector

OPC-UA (OPC Unified Architecture) is a machine-to-machine communication protocol for industrial automation. It is designed to provide a platform-independent and secure framework for data exchange between various devices and systems in industrial environments. This kind of connector is required to connect to demonstration sites IT infrastructure and ingest their data into the IOP in real time. The demonstration sites implement Enterprise Resource Planning (ERP) and Manufacturing Execution System (MES) to monitor and manage their resources. As follows a brief introduction of both concepts:

- The ERP sits at the top level and is responsible for the overall management of the business. In the context of manufacturing, ERP handles high-level planning, scheduling, and management tasks. Therefore, integrates various business processes, including finance, human resources, supply chain management, and more.

- MES operates below the ERP and focuses on the execution and monitoring of manufacturing processes. It bridges the gap between ERP and the manufacturing lines by managing and tracking production. Therefore, the MES gathers data from the manufacturing lines (e.g. production modules, IoT devices).

The IOP and all its modules will act as an additional system within their IT infrastructure for the demonstration sites. Furthermore, the communication protocol foreseen is OPC-UA.

The first step is to develop an OPC-UA server within the IOP to achieve the connection. The server connects to the clients (demonstration sites) and reads the data gathered. The OPC-UA server development implements Python and the *opcua-asyncio* open-source library, ensuring compatibility and efficient data handling. This step includes configuring the OPC-UA server, ensuring it can effectively communicate with the MES software and seamlessly ingest data into the Kafka cluster. Once developed and configured the server, the next step is to establish the connection with the client, the demonstration site. Lastly, appropriate data models must be created within the IOP databases to support the incoming data from the demonstration site, ensuring it is structured and stored correctly for further processing and analysis.

3.2. Kafka

Apache Kafka is a high-throughput, fault-tolerant platform for handling real-time data feeds, enabling the ingestion, storage, and processing of streams of records in a reliable and scalable manner. Kafka receives data from the connectors and stores these as immutable logs, making it possible for multiple downstream systems to consume the data. Each piece of data (message) stored in Kafka is appended to a category known as a topic. Also, Kafka does some data validation, ensuring the data is in the correct format and not corrupted.

In short, the connectors are responsible for data collection and transformation, while Kafka serves as the high-throughput conduit that validates the data and ensures that it is reliably stored and available for consumption by subsequent components in the pipeline.

3.2.1. Strimzi

Apache Kafka is implemented using Strimzi on a Kubernetes cluster for the ONE4ALL project. Strimzi is an open-source project that provides a way to run an Apache Kafka cluster on Kubernetes. This section details the implementation process, including setup, configuration, and managing the Kafka clusters using Strimzi.

Strimzi simplifies the deployment and management of Kafka clusters on Kubernetes by providing Kubernetes-native resources and tools. It automates the operational aspects of running Kafka, such as broker management, topic configuration, user security, and disaster recovery, significantly reducing manual overhead.

3.2.1.1. Installation

1. Download the Strimzi Release Artifacts

The first step consists of downloading the Strimzi release artefacts. The latest Strimzi releases are found on the Strimzi GitHub releases page.

2. Create Strimzi's CRDs and Namespace

Strimzi uses CRDs to define custom resources like Kafka, KafkaConnect, and KafkaTopic. Therefore, the next step is to apply the CRD definitions and create a namespace for the Strimzi cluster operator:

```
kubectl create -f strimzi-cluster-operator-*/install/cluster-operator/ -n kafka
```

This command creates the necessary CRDs and a namespace called kafka (or another namespace of your choosing).

3. Install the Strimzi Cluster Operator

The Strimzi Cluster Operator manages Kafka clusters within a Kubernetes cluster. Install the operator by applying the Strimzi installation files:

```
kubectl apply -f strimzi-cluster-operator-*/install/cluster-operator/ -n kafka
```

This command deploys the Strimzi Cluster Operator to the namespace created previously. The operator will now watch for and manage Kafka resources in the namespace indicated.

4. Deploy a Kafka Cluster

With the Strimzi Cluster Operator in place, Kafka cluster deployment is enabled. Strimzi provides a range of example resource files as guidance and gets started:

```
kubectl apply -f strimzi-cluster-operator-*/examples/kafka/kafka-persistent.yaml
-n kafka
```

This command creates a Kafka cluster with persistent storage. The YAML example files can be used as a baseline and customised to suit the specific requirements of the use case before applying them.

5. Verify the Installation

Once deployed the Kafka cluster, it is possible to check the status of the deployment using *kubectl*:

```
kubectl get pods -n kafka
```

This command lists the Kafka broker pods, ZooKeeper pods, and the Strimzi operator pod running in the namespace and can be monitored.

6. Accessing the Kafka Cluster

After the Kafka cluster is up and running, you can configure your applications to produce and consume messages. The necessary connection details and service endpoints can be found by examining the services created in the Kubernetes namespace.

3.2.2. How to ingest data in Kafka

Using any of the many client libraries for the different programming languages is possible, increasing the system's interoperability. For example, using the Python *confluent-kafka* library, the code would look like this:

```
# Import libraries
import json
from confluent_kafka import Producer

# Configure producer
producer_config = {
    'bootstrap. servers': KAFKA_BOOTSTRAP_SERVERS,
    'security.protocol': 'SASL_PLAINTEXT',
    'sasl. mechanisms': 'SCRAM-SHA-512',
    'sasl.username': USERNAME,
    'sasl.password': PASSWORD
}

# Create a Kafka producer instance
producer = Producer(producer_config)

# Define a callback function to handle delivery reports
def delivery_report(err, msg):
    if err is not None:
        print('Message delivery failed: {}'.format(err))
    else:
        print('Message delivered to {} [{}]'.format(msg.topic(), msg.partition()))

# Define a function to send messages
def send_message(producer, topic, message):
    producer.produce(topic, key='test-message', value=json.dumps(message),
callback=delivery_report)
    producer.flush()

from datetime import datetime
dt = datetime.now()

# Send a message
message = {'timestamp': dt.isoformat() , 'message' : 'hello'}
send_message(producer, KAFKA_TOPIC, message)
print("Message sent: ", message)

# Wait for any outstanding messages to be delivered and delivery reports to be
received.
producer.flush()
```

4. Storage – Layer 2

The storage layer is where data resides after it is ingested for the rest of the modules to use.

4.1. Druid

Apache Druid is a high-performance, real-time analytics database designed to handle large-scale, fast-moving data streams. It enables the ingestion, storage, and interactive querying of event data with low latency and high concurrency, making it ideal for powering modern data-driven applications.

Druid ingests data from various sources, processes it in real-time, and stores it in a column-oriented format optimised for fast analytical queries. The tool enables efficient filtering, aggregation, and computation over massive datasets. Druid's architecture includes a robust indexing system that supports time-based partitioning and data sharding, ensuring scalable performance as data volumes grow.

Additionally, Druid provides robust data validation and transformation capabilities, ensuring that incoming data adheres to the required schemas and is free from corruption. As a complement to Druid's native support for complex event processing, the whole process enables real-time insights and alerting on data streams.

Therefore, Druid acts as a high-performance, scalable engine for real-time analytics, capable of transforming and validating large volumes of event data and making it instantly available for interactive querying and downstream analytics.

For example, the Druid database is currently used in ONE4ALL to store RCPM-cobot information such as positions, trajectories, and battery percentage. All this information is being retrieved from Kafka in real time and serves both for retrieving data for the Decision Support System and as a historical of the robot's behaviour.

4.1.1. Kafka-Druid connector

The Kafka-Druid connector is a high-performance integration that enables seamless data ingestion from Apache Kafka into Apache Druid. It allows real-time data streams from Kafka topics to be ingested, transformed, and indexed by Druid, ensuring low-latency access to fresh data for interactive analytics. The connector efficiently handles large-scale data feeds, validating and transforming the data as it moves from Kafka to Druid. By leveraging the strengths of both platforms, the Kafka-Druid connector ensures that high-throughput, real-time event data is reliably ingested into Druid's high-performance analytical database, ready for immediate querying and analysis.

4.1.1.1. Kafka-Druid connector setup

Druid provides a UI for connection with several tools, including Kafka, as reflected in Figure 3.

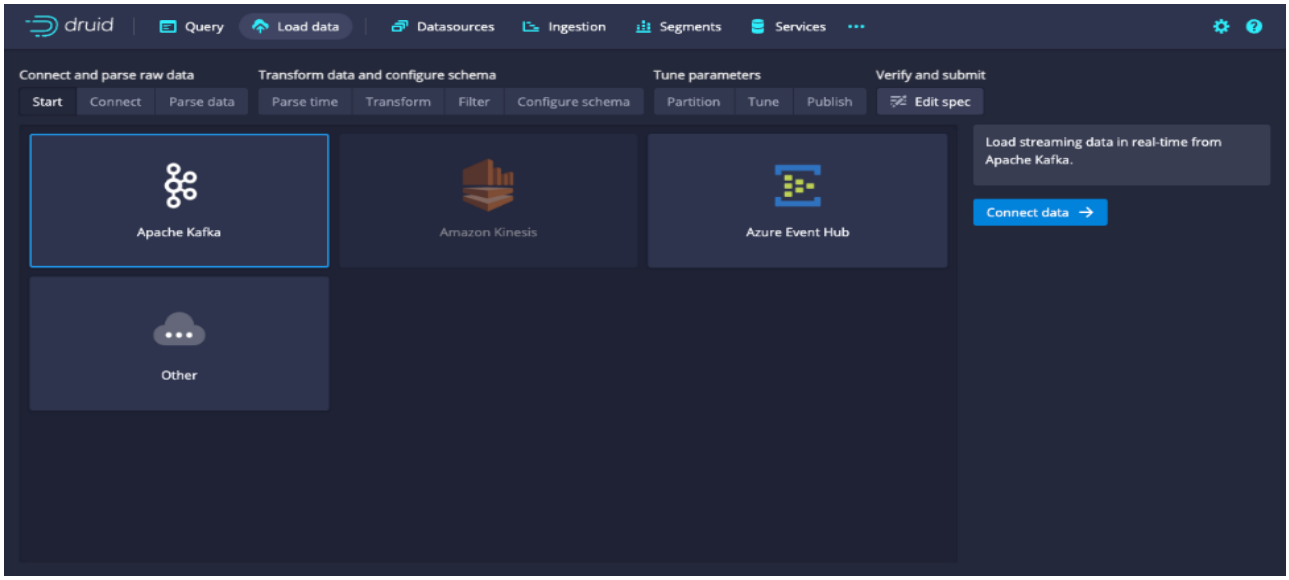

Figure 3: Druid User Interface

This UI allows the configuration of the connection spec as well as other configurations and parameters such as Columns and values filters, the maximum size of a partition, when to delete a partition, and columns format ... More information about the spec creation can be found in Druid's documentation[1].

Once all these parameters are selected, the connection spec is launched, and the data ingestion begins. Figure 4 presents the data sources connected through the pipeline.

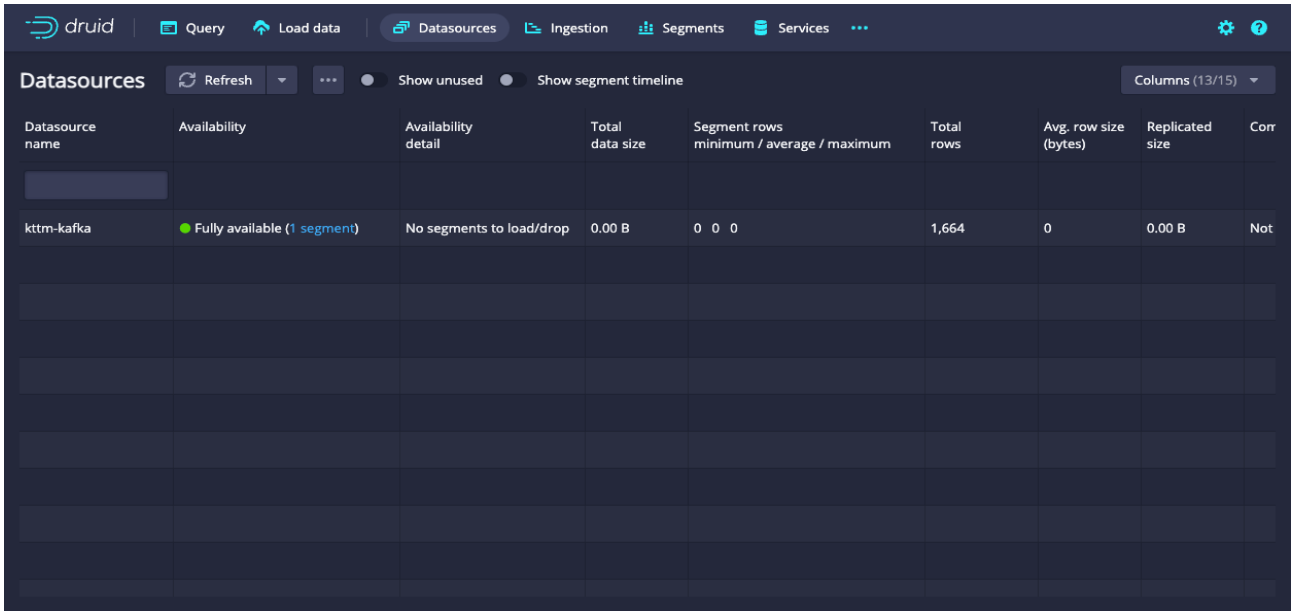

Figure 4: Data sources representation in Druid UI

[1] https://druid.apache.org/docs/latest/tutorials/tutorial-kafka/

4.2. Postgres

PostgreSQL is a powerful, open-source relational database management system (RDBMS) renowned for its robustness, extensibility, and standards compliance. Designed to handle a wide range of workloads, from small single-machine applications to large internet-facing applications with many concurrent users, PostgreSQL is an ideal choice for developers and enterprises seeking a reliable and high-performance database solution.

PostgreSQL supports advanced data types and performance optimisation features that cater to modern application requirements. Its ACID (Atomicity, Consistency, Isolation, Durability) compliance ensures reliable transaction processing, making it suitable for data integrity and fault tolerance applications. The database also supports full-text search, JSON (JavaScript Object Notation) querying, and geospatial data types, providing flexibility for diverse application needs.

The architecture of PostgreSQL includes a sophisticated indexing system that supports multiple indexing methods, such as B-tree, Hash, GiST, SP-GiST, GIN, and BRIN, allowing efficient data retrieval and improved query performance. Additionally, PostgreSQL's support for advanced indexing techniques like partial and expression indexes further enhances its capability to handle complex queries efficiently.

PostgreSQL also offers powerful procedural languages for stored procedures and triggers, including PL/pgSQL, PL/Tcl, PL/Perl, and PL/Python, enabling developers to write custom functions and automate database tasks. This extensibility is complemented by a vibrant ecosystem of extensions, such as PostGIS for geospatial analysis and TimescaleDB for time-series data, which can be easily integrated to extend PostgreSQL's core functionality.

For example, the PostgreSQL database is used for user and robot management. On the one hand, user data such as identification or related company will be stored here. On the other hand, robot data such as Kafka topics, identification or status will be stored here.

4.2.1. SQLAlchemy

SQLAlchemy is a powerful, open-source SQL toolkit and Object-Relational Mapping (ORM) library for Python. It provides developers with a comprehensive suite of tools to interact with relational databases in a Pythonic and efficient manner, making it a popular choice for database access in Python applications.

At its core, SQLAlchemy offers a high-level ORM that abstracts the database interactions into Python classes and objects. The approach allows developers to work with database records as native Python objects, simplifying database operations and enabling seamless integration with the rest of the application code. The ORM supports advanced features such as lazy loading, relationship management, and automatic schema generation, which help build complex data models and ensure data integrity.

In addition to the ORM, SQLAlchemy includes a powerful SQL Expression Language. This lower-level API provides fine-grained control over database interactions and allows developers to construct SQL queries programmatically. The SQL Expression Language is designed to be database-agnostic, enabling the same code to work across multiple database backends without modification.

SQLAlchemy's architecture emphasises flexibility and performance. It supports connection pooling, lazy loading, and transaction management, optimising resource usage and improving application responsiveness. Furthermore, SQLAlchemy is highly extensible, allowing developers to customise its behaviour and integrate it with other libraries and frameworks.

One of SQLAlchemy's key strengths is its support for multiple database dialects, including PostgreSQL, MySQL, SQLite, Oracle, and Microsoft SQL Server. The flexibility provided by the application built with SQLAlchemy can easily switch between different database systems as needed. Additionally, SQLAlchemy's declarative syntax makes defining database schemas and relationships easy, promoting code readability and maintainability.

SQLAlchemy is a robust and versatile SQL toolkit and ORM for Python, offering a comprehensive set of features for efficient and effective database access. Its combination of high-level abstractions and low-level control makes it suitable for a wide range of applications, from simple scripts to complex enterprise systems.

4.2.2. Alembic

Alembic is a lightweight database migration tool with SQLAlchemy, designed to handle the evolution of database schemas in a consistent and version-controlled manner. It allows developers to manage changes to database structures, such as adding or altering tables and columns, through a series of incremental, reversible migration scripts.

At the core of Alembic's functionality is its command-line interface, which provides commands to generate new migration scripts, apply migrations to the database, and roll back changes. These migration scripts are written in Python and leverage SQLAlchemy's metadata to ensure the changes are database-agnostic, supporting various database systems like PostgreSQL, MySQL, SQLite, and others.

Alembic's version control capabilities enable teams to collaborate on database schema changes efficiently, ensuring that all modifications are tracked and can be rolled forward or backwards as needed. It is particularly valuable in agile development environments where database schemas evolve rapidly alongside application code- In conclusion, Alembic is an essential tool for managing database migrations in SQLAlchemy-powered applications, providing a structured and reliable approach to evolving database schemas while maintaining consistency and control over database changes.

4.2.3. SQLModel

SQLModel is a modern library for interacting with SQL databases in Python, combining the best features of SQLAlchemy and Pydantic. Designed to simplify database operations, SQLModel offers a declarative syntax that makes it easy to define data models and interact with databases using Python classes.

At the core of SQLModel is its integration with SQLAlchemy, leveraging its robust ORM capabilities to provide seamless database interactions. SQLModel classes are essentially SQLAlchemy models with additional Pydantic features, allowing for both database schema definition and data validation within the same framework. This dual functionality reduces redundancy and enhances code maintainability.

SQLModel's syntax is straightforward and user-friendly, making it accessible to developers of all skill levels. It supports asynchronous operations, ensuring efficient handling of I/O-bound tasks and improving application performance. Additionally, SQLModel is database-agnostic, supporting multiple database backends such as PostgreSQL, MySQL, SQLite, and others. In conclusion, SQLModel is a powerful tool for managing SQL databases in Python, offering a clean and efficient approach to database operations by integrating SQLAlchemy's ORM and Pydantic's data validation into a single, cohesive library.

5. Processing – Layer 3

The processing layer takes the stored data and performs computations, transformations, or analyses. This could involve a wide range of tasks, from simple data aggregation operations to complex machine learning algorithms. Depending on the system's requirements, processing can occur in real-time or in batches.

5.1. APIs

APIs, or Application Programming Interfaces, are sets of definitions and protocols that enable different software applications to communicate with each other. They act as intermediaries, allowing one application to access the functionality or data of another without needing to understand the internal details of its implementation.

An API defines a series of calls or requests that can be made, along with the required data and the format of the responses. For example, a web service API might allow an application to retrieve current weather information, send a text message, or interact with a social media platform. This standardised communication facilitates integration and functionality sharing between disparate systems.

APIs are essential for enabling interoperability between software applications, allowing them to leverage each other's features and data efficiently and securely.

5.1.1. FastAPI

FastAPI is a modern, high-performance web framework for building APIs with Python, designed to be easy to use and highly efficient. It is based on standard Python-type hints, enabling automatic validation and serialisation, streamlining the development process and improving code quality.

At its core, FastAPI leverages the asyncio capabilities of Python to support asynchronous programming, making it ideal for building fast, concurrent web applications. This asynchronous support allows FastAPI to handle a large number of simultaneous connections efficiently, making it suitable for real-time applications and microservices architectures.

FastAPI's design emphasises developer productivity and code readability. Its automatic generation of interactive API documentation, using Swagger UI and ReDoc, allows developers to test and understand API endpoints quickly. Additionally, FastAPI integrates seamlessly with popular data validation and serialisation libraries like Pydantic, ensuring that data handling is both robust and straightforward. In conclusion, FastAPI is a powerful and user-friendly framework for building high-performance APIs in Python, offering asynchronous capabilities, automatic documentation, and seamless integration with data validation libraries, all designed to boost productivity and ensure code quality.

5.1.1.1. FastAPI Endpoints

As an example, two endpoints have been implemented for the RCPM-cobot control.

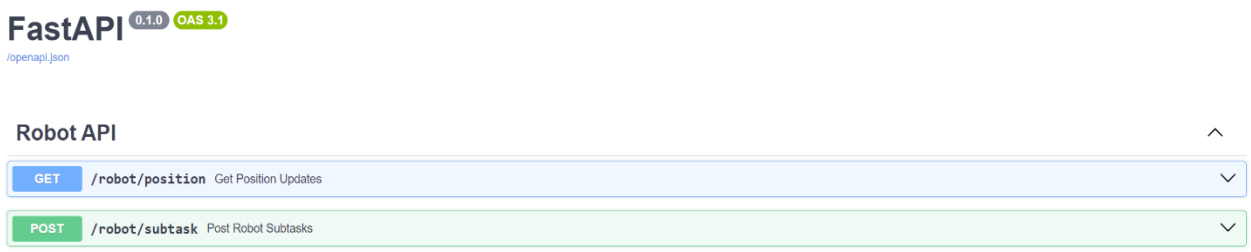

Figure 5: RCPM FastAPI endpoints

- **Robot/position**: This endpoint retrieves data from the Druid database and returns Robot data such as actual position, actual path, and battery level. This endpoint will be used as control and to feed the Decision Support System.
- **Robot/subtask**: This endpoint sends a message to the corresponding Kafka topic. It contains information about the next actions to be performed by the robot. This endpoint does not save data in any database.

5.2. RCPM data processing - ROS2

Similarly to the ingestion layer, the processing layer contains a ROS2 "Controller" module responsible for sending commands to the RCPM. This module divides tasks the DSS sends into lists of low-level movements and sends them to the cobot controller using ROS2 Actions and Services. Further information regarding this communication and a complete list of possible commands can be found in **Deliverable 4.7**.

5.3. AI-based vision system

5.3.1. Objective and scope of the vision system

The primary objective of the vision system in the ONE4ALL project is to enable the robotic arm to accurately identify, locate, and grasp objects within its operational environment. This involves detecting different types of objects, determining their coordinates, and providing this data to the robotic arm in real-time to facilitate effective manipulation.

The AI-based vision system is a critical component of ONE4ALL's solution, directly impacting its effectiveness and efficiency. The ability of the cobot to autonomously pick and place objects relies heavily on the accuracy of the vision system. This enhances the overall automation process and increases productivity in the operations.

The scope of the vision system includes:

I. **Object detection**: identifying objects within the camera's field of view using image processing and machine learning techniques.
II. **Object localisation**: determining the coordinates of detected objects and providing them to the robotic arm.
III. **Communication**: sending the processed data to the robotic arm's control system in real-time.

Regarding the ORI use cases, the vision system should be able to detect medication boxes more specifically, taking into consideration the various sizes and shapes in which these medication boxes may come and the different label types they may have. In Figure 6 are some examples of them:

Figure 6: Orifarm demonstration site product samples

On the other hand, concerning the MOL end-of-line use case, the vision system aims to correctly identify various packaging types for olives and other pickled products, including jars, cans, and pouches. Additionally, these types of packaging, particularly glass jars and metal cans, can have reflective surfaces that may complicate detection. The ones presented in Figure 7 are some objects that the system needs to detect in the MOL demonstration site:

Figure 7: Madama Oliva demonstration site products sample

5.3.2. Description of the vision system

The vision system is a comprehensive solution designed to enable the robotic arm to detect, identify, and locate various objects within its operational environment. The system integrates multiple components to achieve high precision and reliability in object detection and localisation. The main modules are the following:

I. **Image Acquisition**: high-resolution industrial cameras are strategically placed to capture workspace images, ensuring full coverage and high-quality visual data.
II. **Object Detection**: the system employs state-of-the-art machine learning algorithms, specifically YOLOv8 (You Only Look Once), introduced in [1] by J. Redmon et. Al., to detect objects within the images quickly and accurately. Additionally, the detection process outputs bounding boxes around identified objects, along with class labels and confidence scores.
III. **Object Localisation**: the coordinates of the bounding boxes are extracted and converted into real-world coordinates using camera calibration data and stereoscopic depth vision to determine the distance of objects from the camera, providing three-dimensional localisation.
IV. **Data Processing and Communication**: the processed data, including object type and coordinates, is aggregated and formatted for transmission throughout a robust communication protocol, ensuring that the data is transmitted to the robotic arm's control system in real-time, enabling accurate and timely object manipulation.
V. **Integration with Robot**: the vision system integrates with the robot's control system, providing it with the necessary data to execute precise movements for object picking and placing.

5.3.3. Model development and training

5.3.3.1. Data provenance

The development of an effective vision system relies heavily on high-quality data. The data used for training the detection model comes from different sources to ensure robustness and accuracy:

I. **Public datasets**: the specific model used is *YOLOv8*, which is trained with the standard dataset *COCO* (Common Objects in Context), a large-scale, richly annotated dataset which contains over 200.000 labelled images with more than 80 object categories. Furthermore, various datasets used for training were sourced from *Roboflow*, a platform known for its extensive collection of both public and private datasets. Specifically, only publicly available datasets from Roboflow's community contributions were utilised, ensuring that the data is freely accessible and well-annotated for the project's needs.
II. **Internal datasets**: images captured from the use cases' operations and environments, providing relevant and specific examples of the objects the system needs to detect. Moreover, each image is labelled with bounding boxes around the objects of interest, along with class labels. The whole process is done on the *Roboflow* platform, too.

5.3.3.2. Data preprocessing

In machine learning, input data quality and consistency significantly impact model performance. Raw data often presents several issues, such as inconsistent orientations, varying sizes, poor contrast and limited diversity. These challenges can hinder the model's ability to learn effectively and generalise to new data, resulting in poor performance. To address these issues, before training, the data from internal datasets undergoes several preprocessing steps to enhance its quality and suitability for the model:

I. **Auto-orientation**: addressing the issue of mismatched images and labels in a dataset due to varying camera orientations by extracting orientation information from the EXIF metadata, ensuring proper alignment of bounding boxes with images for dataset creation.
II. **Resize**: it involves scaling images to a consistent and predetermined dimension, automatically adjusting labels proportionally, which is essential as varying image sizes and resolutions complicate model training.

III. **Auto-adjust contrast** focuses on contrast adjustment in images to highlight finer details and minimise noise, thus simplifying the analysis process.

IV. **Data augmentation** is employed to expand the dataset used for training models by applying transformations to the original data, creating variations that foster more robust learning. These transformations include flips, rotations, crop, grey scale, saturation, brightness, exposure, blur, noise, cutout, and mosaic. Each one is graphically presented in Figure 8.

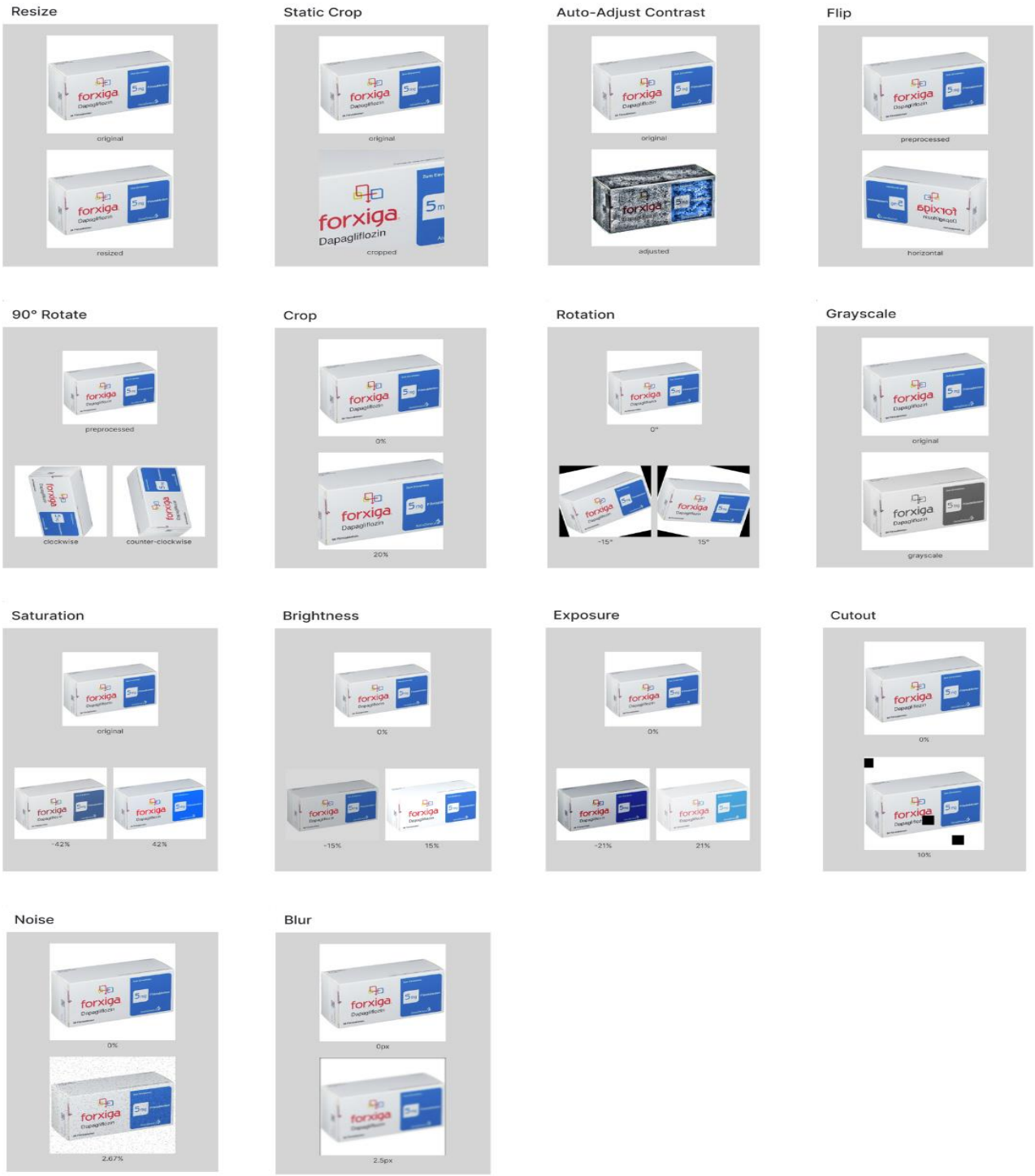

Figure 8: Data Augmentation Techniques

5.3.3.3. Data splitting

The dataset is divided into three subsets: **training**, **validation**, and **test sets**, following a widely adopted standard in machine learning. This standard split involves allocating 70% of the data to the training set, 10% to the validation set, and 20% to the test set (as reflected in Figure 9).

Allocating around 70% of the data to the training set provides the model with a substantial amount of examples to learn the features and patterns of the objects, which is crucial for capturing the underlying relationships within the data. This large portion helps in reducing the risk of underfitting. Underfitting occurs when the model fails to capture the underlying patterns in the data, often because the training set is too small or the model is too simple.

The 10% designated for the validation set is used to tune hyperparameters and monitor the model's performance during the training process. By validating the model on a separate set, necessary adjustments can be made without overfitting the model to training data. This overfitting happens when the model learns the training data too well, including its noise and outliers, which results in poor generalisation of new data. The validation set provides a checkpoint to ensure that the model maintains its ability to generalise well to unseen data, thus helping select the best model architecture and parameters.

The remaining 20% is reserved for the test set, which is used exclusively for the final evaluation of the model. This set acts as a proxy for new, unseen data and is essential for obtaining an unbiased assessment of the model's performance. Using a substantial portion, ~20% for testing, ensures that the performance metrics are reliable and accurately reflect the model's ability to generalise to new data.

This splitting is done randomly to guarantee that each subset is representative of the overall dataset, providing a robust foundation for both training and evaluation. Otherwise, there would be a risk of bias that might lead to skewed results and unreliable performance metrics.

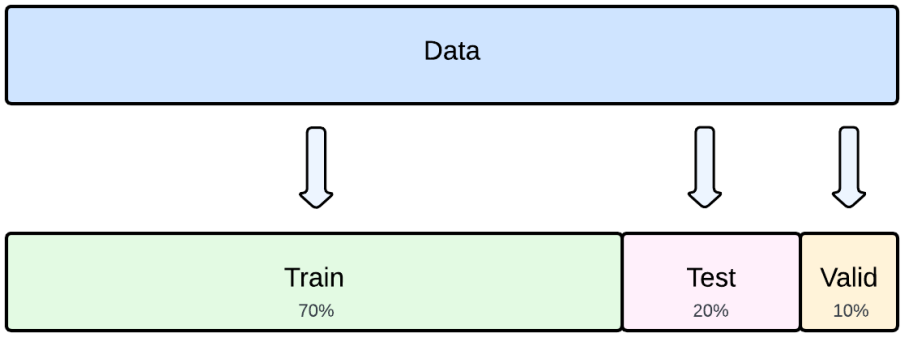

Figure 9. Data splitting graphical representation

5.3.3.4. Training process

The training process involves different key steps to develop a robust object detection model:

I. **Model Selection**: YOLO (You Only Look Once), specifically YOLOv8 provided by *Ultralytics*, is utilised due to its balance between speed and accuracy, making it suitable for real-time object detection. In [1] and [2], comparative studies have been conducted between YOLO and other object detection methods such as R-CNN, SSD or DPM, and YOLO has proven to offer superior detection speed and high accuracy, essential for real-time applications.

II. **Hyperparameter Tuning**: When tuning the hyperparameters for the model, several key parameters are considered to influence the model's performance heavily. These include:

a. **The number of epochs** determines how often the training process will iterate over the entire training dataset. More epochs allow the model to learn more thoroughly but may lead to overfitting if too high. A preliminary number of epochs is set to 100. This value is chosen to ensure sufficient learning while keeping an eye on potential overfitting. The parameter may be adjusted based on the model's performance in future iterations.

b. **Image Size**: it affects both the computational efficiency and the level of detail that the model can capture. Larger images provide more detail but require more computational resources. The images are resized to 640 pixels. The size is selected to balance the previously mentioned trade-off between computational efficiency and level of detail. Adjustments may be made as well in the future if necessary.

c. **Batch size**: it determines the number of training examples utilised in one iteration of training. Larger batch sizes can make the training process faster and provide a more accurate estimate of the gradient, but they require more memory and computational power. The preliminary value is set to the default value, 16, to strike a balance between training speed and resource requirements.

III. **Validation and test metrics**: throughout the training process, the model's performance is continuously monitored using a validation set, with built-in vital metrics such as:

a. **Precision (P)**: it indicates the accuracy of positive predictions. It is the proportion of true positives (correctly identified objects) out of the total predicted positives (true positives plus false positives). High precision means that few irrelevant objects are classified as relevant; when the model predicts an object, it is very likely to be correct.

b. **Recall (R)**: it is the proportion of true positives out of the total actual instances (true positives plus false negatives). It measures the model's ability to detect all relevant instances in the dataset.

c. **Mean Average Precision at 50% IoU threshold (mAP50)**: it is a common metric for assessing the accuracy of object detectors that evaluates the model's performance using a 50% overlap threshold between the predicted bounding boxes and the ground truth boxes. A prediction is considered correct if the overlap (IoU) between the predicted bounding box and the ground truth bounding box is at least 50%. A high mAP50 score indicates that the model's predictions are accurate at this baseline level of overlap.

d. **Mean Average Precision averaged over IoU thresholds from 50% to 95% (mAP50-95)**: it averages the precision scores over multiple IoU thresholds, ranging from 50% to 95% in 5% increments. It evaluates how well the model performs in easy detection tasks (50% overlap) and more challenging ones (up to 95% overlap).

Together, these metrics form a comprehensive evaluation framework that guides fine-tuning by being evaluated after each epoch to ensure effective learning and prevent overfitting. The same ones are also used in the final evaluation with the test set to provide a consistent and unbiased assessment of the model's performance on unseen data.

5.3.4. Initial testing and results

The first step is to start from the base model of YOLOv8 without further training. As previously indicated, this model has been pre-trained with large general-purpose datasets capable of identifying various types of objects. However, the drawback is that it exhibits limited accuracy when applied to specific tasks or specialised domains, which is this case.

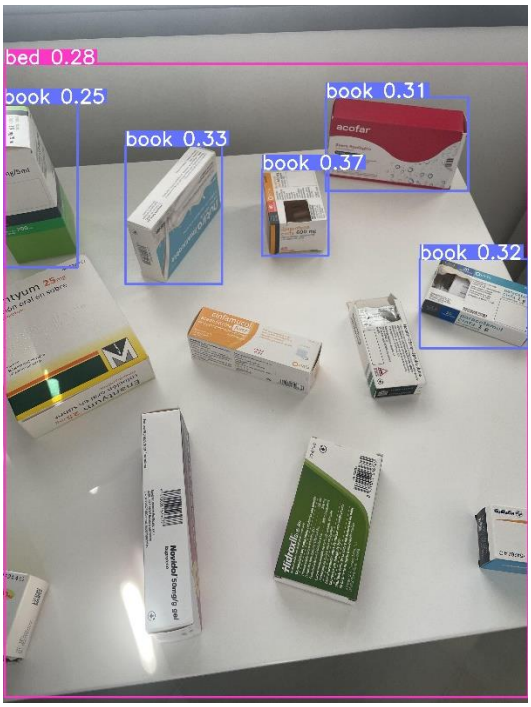

*Figure 10. AI-based vision system **base** model (YOLOv8) results*

As shown in Figure 10, although the base model can detect some objects, it fails to achieve accuracy in both detection and classification, which are fundamental requirements for the project.

After training the entire model with *Roboflow*'s public dataset consisting of 1055 images of various medicinal boxes of different types, shapes, colours, and light conditions, and with only a class named *"box"*, the results are illustrated in Figure 11:

Figure 11. AI-based vision system-trained model results

As shown in Figure 11, the vision system exhibits a slight improvement in performance, particularly regarding confidence and classification accuracy. Nevertheless, it still requires significant enhancement to achieve a level of performance that is sufficiently satisfactory to meet the project's criteria.

The next step consisted of conducting fine-tuning (i.e. adjusting a pre-trained model's parameters on a new, specific dataset to improve its performance for a particular task) on the previous model with an internally created dataset derived from images captured from the production line and applying the respective data augmentation and other data preprocessing techniques. Once completed this step, as reflected in Figure 12, it is evident that the results and confidence improve considerably, and that the detection and classification become significantly more accurate.

Figure 12. AI-based vision system results after fine-tunning

It is worth mentioning that, in the process of fine-tuning, tests have been conducted by freezing various numbers of layers within the model to determine which alternative yields better performance. Layer freezing is a technique used in transfer learning where some layers of a pre-trained neural network are "frozen" without applying any changes to them during training on a new dataset. The technique is shown in Figure 13. This means that the weights of these layers are not updated during the training. Typically, the frozen layers tend to be the initial ones, which capture general features, while the later layers, which are more specific, are fine-tuned to adapt better to the new task. Ultimately, full fine-tuning has proven to be the most accurate option in this case.

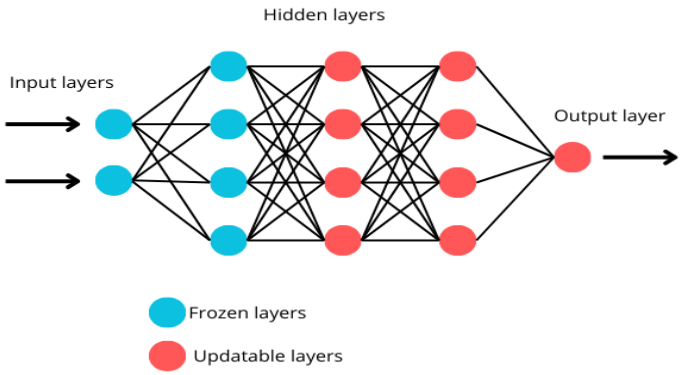

Figure 13: Neuronal Network layer frozen technique visualisation

Moreover, it will also be necessary to specify the coordinates of the bounding boxes so the cobot can grab the object. The first approach is to obtain the two-dimensional pixel coordinates of the bounding boxes from the inference results, as reflected in Figure 14. This can be achieved by accessing the '*xyxy*' attribute of the '*boxes*' object, which contains the coordinates in the format (*xmin*, *ymin*, *xmax*, *ymax*), representing the top-left and bottom-right corners of each box in pixel units. The following image shows the coordinates of each bounding box's centre for more clarity.

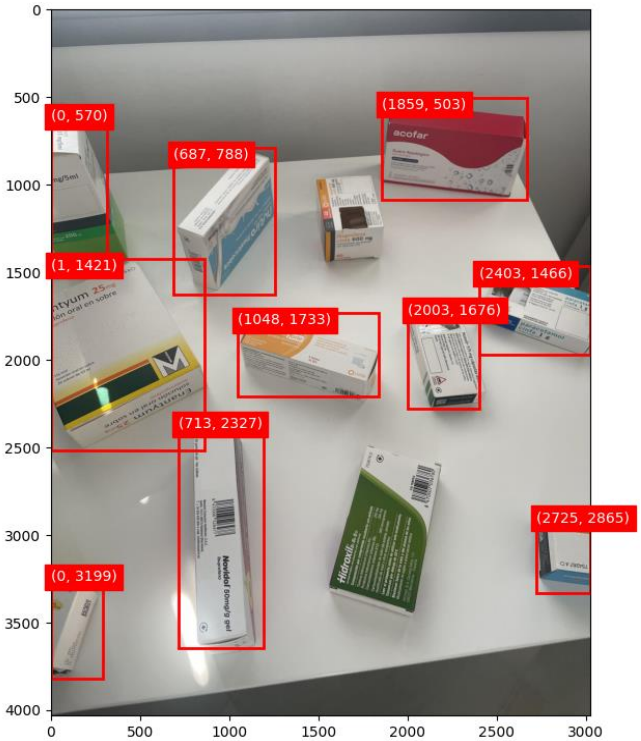

Figure 14: Products coordinates estimation - concept version

5.3.5. AI-based vision system KPIs

The KPIs (Key Performance Indicators) will be progressively defined and adjusted as the vision system development progresses. Certain KPIs will be valid to specific stages of development, and they will also depend on the type of environment that the vision system is working on and its complexity as well.

I. **Training stage**: during this phase, the main objective is the accuracy and efficiency of the model's learning. The applicable KPIs at this stage are the metrics defined in 5.3.3.4, namely:

- Precision (P)
- Recall (R)
- Mean Average Precision at 50% IoU threshold (mAP50)
- Mean Average Precision averaged over IoU thresholds from 50% to 95% (mAP50-95)

II. **Testing stage:** once the model is trained, it enters the testing stage, where the focus shifts to evaluating its performance in more realistic scenarios. The applicable KPIs at this stage include:

- Detection Accuracy: percentage of correctly detected objects in a test set. This KPI evaluates the model's ability to identify objects.
- Classification Accuracy: percentage of correctly classified objects among the detected ones. This KPI ensure that the model detects objects and classifies them correctly.
- False Positive Rate: percentage of incorrect detections made by the model. This KPI assesses how often the model makes a mistake when identifying an object that is not present.

Deployment and Operational Stage: in this stage, the vision system will face real-world environments with varying levels of complexity and challenges. The KPIs applicable to this stage will be defined later once the vision system is ready to transition from the testing phase to real-world deployment.

5.3.6. AI-based vision system's next steps

Moving forward, the following steps to enhance the system involve the application of refinement techniques to improve the existing model. A thorough literature review will be conducted alongside continuous testing to identify and leverage suitable methods for model enhancement. In addition, the model will also be specifically tailored for the *Madama Oliva* use case, involving training with datasets that more accurately reflect the types of objects encountered. This process will incorporate a multi-class classification approach to distinguish between various object types, ensuring detailed recognition and categorisation.

Furthermore, it is pivotal to obtain precise real-world coordinates by considering the camera's specifications and the depth information it provides. Accurately mapping two-dimensional pixel coordinates to real-world measurements is critical to the system's precision.

Besides, the system will also embrace *MLOps* practices to enhance deployment efficiency, monitoring, and supervision. This combines DevOps, data engineering, and machine learning principles to ensure efficient and reliable deployment of machine learning models into production environments. The system will guarantee streamlined upgrades and systematic version control using Continuous Integration and Delivery (CI/CD), scalability, and reproducibility. This will ensure that the operational lifecycle of ML models is optimised for performance, allowing rapid response to new data.

Lastly, integration with the IOP is planned to facilitate real-world production environment testing. This is crucial to validate the system's performance and ensure its reliability under operational conditions.

6. Serving – Layer 4

The serving layer is responsible for delivering the processed data to end users or applications in a useful and accessible manner. This could involve serving data to dashboards, reporting tools, or APIs for consumption by other services. It's also responsible for ensuring fast and reliable data access, potentially requiring optimisation or caching strategies. This layer also handles access control and security, ensuring users can only access the data they are authorised to see.

6.1. Frontend UI

6.1.1. RCPM UI preliminary designs

Figure 15 shows a preliminary design of what the GUI would look like:

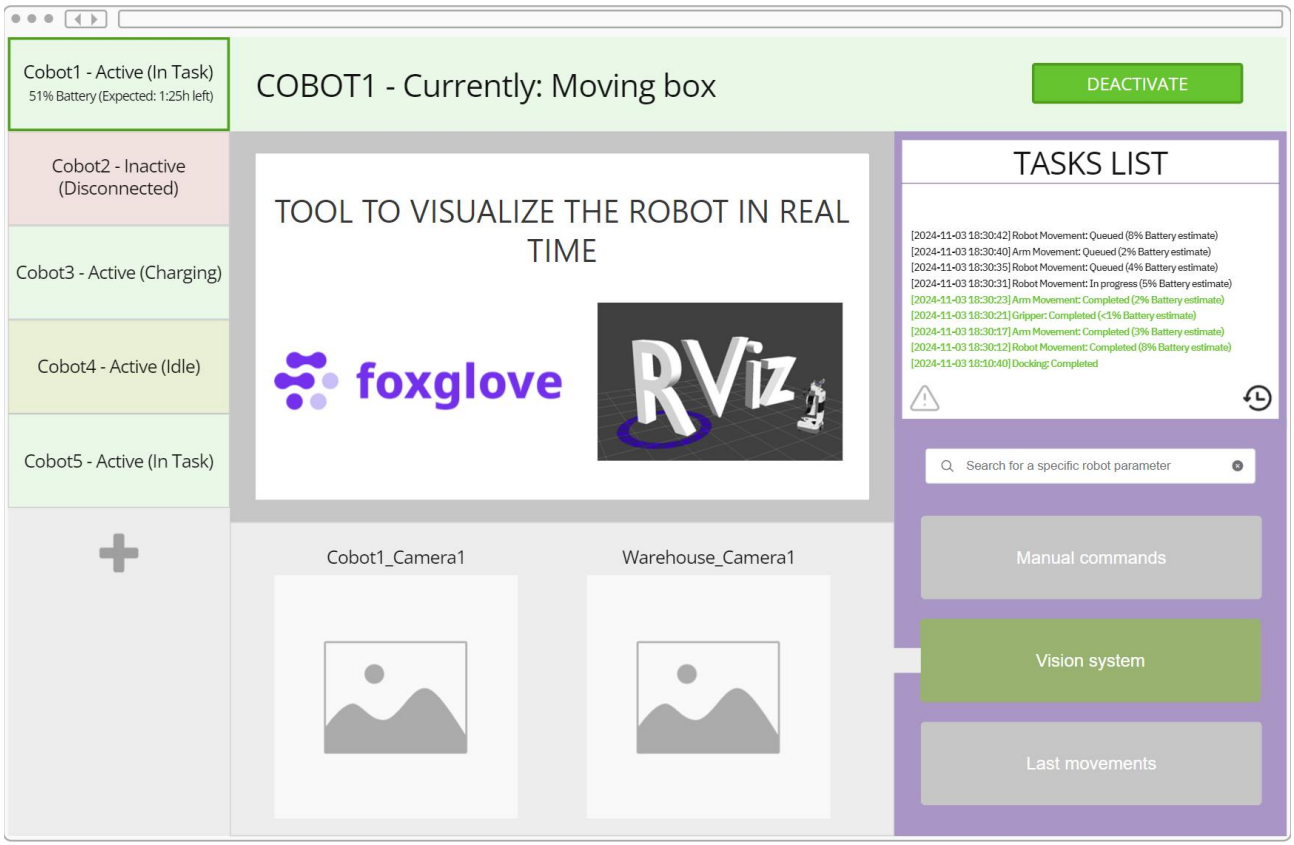

Figure 15: Preliminary design of RCPM remote control User Interface

There will be a certain window inside the interface displaying the robot's position, speed and path. For that purpose, a certain tool (such as Foxglove or RViz) will be used as visualisation software.

Given that this is only a draft, it's currently unable to communicate to the cobot and the DSS. In later stages of development, a functional GUI will be created to be able to see all cobots and tasks in real time.

6.2. Superset

Apache Superset is an open-source tool designed for data exploration, visualisation, and dashboard creation, specifically catering to business intelligence needs. Its role is to convert processed data into meaningful and interactive visual representations. Superset's versatile suite of visualisation options, combined with its capacity for data filtering, slicing, and dicing, allows business intelligence professionals to effectively identify trends, patterns, and insights.

7. Security

The IOP infrastructure serves as the fundamental support mechanism that facilitates the intake, processing, storage, and transmission of data from digital devices to the appropriate tools and applications for various objectives. The IOP oversees the efficient exchange of information and integration across multiple systems and data formats while supporting the functioning of data-driven digital twins. This provides important insights and enhances end-users' decision-making. Moreover, given the IOP's critical role in the ONE4ALL initiative, it is imperative to prioritise its security.

In the absence of adequate security measures, the IOP tools and infrastructure are susceptible to various malicious cyber-attacks, including DDoS, Man-in-the-middle, malware injection, spear phishing, SQL injection, ransomware, and many more. This consequently has a devastating impact on the IOP. Therefore, several protective measures are being implemented to guarantee the security of the IOP infrastructure, its tools and applications, and all associated data.

Furthermore, in order to achieve the necessary level of security, D3.2 in M10 offered a plethora of security measures and recommendations that must be adhered to and executed. These measures also include a collection of robust and resilient protection mechanisms specifically designed for the IOP tools and infrastructure. To achieve the required security, the IOP employs a variety of protection technologies. These technologies are being implemented in different parts of the IOP, along with the network, data, tools, and infrastructure. Some of these essential security measures include role-based access control, user authentication, monitoring and incident response, network security, data security through access control, data usage auditing, data leak prevention, data compliance, and data protection through encryption.

To this end, this chapter first discusses the implementation stages used in IOP development. It then provides an update on the integration of security into the IOP infrastructure and concludes with some parts of the implementation that have either been completed or are nearing completion.

7.1. Security implementation stages

The implementation of Task 3.2 from M10 is divided into 3 facets. This includes the preventive measures, measures that come with the IOP adoption, and measures that introduce some novelty in the IOP security.

7.1.1. Preventive measures

These measures are mainly adopted to ensure the IOP is prevented from cyberthreats even before they happen. This will help avoid any unforeseen challenges in mitigating threats after they occur. Moreover, while not all security measures require implementation, some can be achieved by adhering to certain security rules and recommendations outlined in the guidelines in D3.2. Therefore, this phase entails the adoption of preventative measures, which require adherence to certain

requirements both during and after the IOP's development. The measures can be categorised under the following protection mechanisms.

- **User authentication:** Implementing robust password and biometric authentication policies with regular updates, ensuring devices are not operating with default configurations, often updating device software, and avoiding hard-coding device credentials.

- **Data security:** Implementing user prompts on sensitive data; establishing data categorisation strategies and policy; ensuring thorough evaluation of cloud vendors (if applicable), prior to utilising their services; routine check on data transmission channels, devices, and modules.

- **Tools and devices management:** Ensuring frequent tools and devices updates; ensuring firewalls, anti-malwares, antiviruses, and anti-spywares are installed on all devices (especially the critical ones), utilising encryption capabilities in tools and devices; removal of unused features in applications and devices, etc.

- **Network and cloud security:** Installing and configuring networks and devices, segmenting the network, identifying and categorising asset values connected to the network, assessing and auditing segmented networks, maintaining standardised device configurations, and regularly updating database software.

7.1.2. Measures that come with IOP adoption

The security defence mechanisms outlined in D3.2 highlight several defensive measures already incorporated into the IOP architectural design. Several components of the architecture, structured into three separate levels: bare-metal servers, Kubernetes API, and other services to Kubernetes, guarantee role-based access control. Other defensive mechanisms adopted by the IOP include the following:

- Prometheus and Grafana monitoring stack for real-time monitoring and incidence reporting.

- Leveraging token-based systems with JWT (JSON Web Tokens) for data access control.

- Single Sign-On (SSO), password, and biometric (multi-factor) for authentication.

- Adopting Transport Layer Security (TLS) to encrypt communication channels in order to secure the network.

7.1.3. Measures that introduce some novelty to the IOP security

These measures entail developing and integrating novel techniques to enhance the security and privacy of IOP and its services. The measure also ensures secure authentication. Thereby protecting user biometric data while enhancing its privacy. This novel integration involves using a cancellable biometric template protection scheme rather than just a simple authentication mechanism.

All three phases will be integrated into different elements and components of the IOP. Thus, ensuring a secure and privacy preserving IOP infrastructure. Below, we itemise how these measures are implemented in the IOP.

7.2. Security integration into IOP

As earlier highlighted, IOP security integrates various defense mechanisms to ensure the security of not just the infrastructure but the entire set of tools, services, and data that ensures its seamless operability. This section discusses how these security mechanisms integrate into the IOP. It further highlights some of the mechanisms that are already integrated, such as authentication and access control.

7.2.1. Security integration

Since the IOP ensures the optimum interoperability of both the IT and OT technologies, it is imperative that the adopted security measures and defense mechanisms mitigate all vulnerabilities relating to these technologies. Therefore, we identified some possible threats the IOP infrastructure can be vulnerable to and integrated different defense measures to mitigate them. This is done based on the four integral elements/layers of the IOP: the ingestion layer, the storage layer, the processing layer, and the serving layer.

7.2.1.1. Ingestion layer security

The ingestion layer allows data entry into the IOP from different sources. As such, ensuring the security of this layer and its components is paramount. Security defense mechanisms including authentication, access control, network segmentation, network monitoring and incidence response as well as data encryption are integrated into layer to ensure a secure and hitch-free entry of data, thereby mitigating attacks such as sphere phishing, eavesdropping, SQL injection, Man-in-the-middle, DDoS, section hijacking, and other network-related attacks.

7.2.1.2. Storage layer security

The storage layer retains all collected data for further usage by other modules within the IOP. Therefore, security risks such as DoS, malicious insiders, and unauthorised access poses grievous concern to this layer. To mitigate such threats, the IOP integrates security measures such as access control, user authentication, and data encryption. These mechanisms will ensure only valid and trusted users can access the right data while also guaranteeing that it is protected from unforeseen attacks via encryption.

7.2.1.3. Processing layer security

This layer receives the stored data and perform different processes on it depending on the operational requirements. The layer performs some of the crucial tasks of the IOP. Therefore, it is imperative to ensure that all data processing pipelines within the layer are secured from cyberthreats. Hence, different protection mechanisms from the storage and ingestion layer are also shared in this layer. In addition, defense mechanisms such as authorisation and privacy-preservation are also integrated in this layer. As a result, a plethora of attacks including data tampering, privacy leakage, IOP service tampering, privilege escalation, Man-in-the-middle, software attacks, and brute-force can all be mitigated.

7.2.1.4. Serving layer security

The serving layer interprets all processed information from the processing layer and transmit the outcome to the consumers. Given that this layer directly interacts with the end users, attacks such as phishing, credential stuffing, spoofing, session hijacking, and privacy-related poses a severe security threat to this layer. Therefore, ensuring its security is of paramount importance. Hence, defense mechanisms such as authentication, access control, anti-malwares, anti-spywares, firewalls, encryption, and anti-viruses are used to guarantee the security of this layer.

7.2.2. Security overview

With the proposed security integration strategy at each layer of the IOP, the IOP and its components are guaranteed protection. Furthermore, adding multi-factor authentication, which includes passwords, biometrics, single sign-on, and good practices from the guidelines in D3.2 will provide even more security to the IOP. This guarantees comprehensive protection for the IOP.

Nonetheless, a complete picture of all these security integrations will be covered in D3.3. Therefore, the D3.3 in M30 will focus on providing critical technical details regarding the implementation of all defence mechanisms and how they integrate into each layer of the IOP architecture. Detailed experimental results and security analysis follow this.

References and Resources

[1] J. Redmon, S. Divvala, R. Girshick and A. Farhadi, "You only look once: Unified, real-time object detection," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[2] L. Tan, T. Huangfu, L. Wu and e. al., "Comparison of YOLO v3, Faster R-CNN, and SSD for Real-Time Pill Identification," *Research Square, preprint*, vol. 1, 2021.