POLITECNICO DI TORINO

MASTER’s Degree in COMPUTER ENGINEERING

MASTER’s Degree Thesis

A Web Application For Real-Time Monitoring Of A Fleet Of Intelligent Closure And Filling Devices In An Industrial Environment

Candidate	Supervisors	Company
MARIO DEDA	Prof. STEFANO QUER	AROL GROUP

MARCH 2023

SUMMARY

The Industrial Internet of Things (IIoT) is transforming the manufacturing industry by connecting machineries, devices, and people to create smart factories. By enabling the collection and analysis of large amounts of data from industrial equipment and devices, IIoT allows real-time monitoring and analysis of production and industrial processes, resulting in increased efficiency and cost savings. Continuous monitoring and analysis of the performance of devices can ensure that industrial facilities are operating at their highest level of efficiency and performance. By integrating cutting-edge technologies, the IIoT is revolutionizing the way industries work and is bringing about a new era of Industry 4.0. As more and more industrial devices become connected to the internet and generate vast amounts of data, and as the industry focuses more and more resources on Industry 4.0 strategies, the need for custom IIoT solutions and tools tailored to their specific needs is increasing. In this context, in order to take advantage of the full potential of IIoT, businesses, and industries need new applications, especially web-based ones, that can process, analyze, and present the collected data in a meaningful way. The aim of this thesis is to study the feasibility and implement a proof-of-concept web application that is able to perform real-time and historical monitoring, through the use of a dynamic dashboard, of a large fleet of intelligent industrial closure and filling devices. The main purpose of this application is to provide real-time visibility into the performance and status of these devices, enabling quick identification and resolution of device issues, improving operational efficiency, and reducing downtime. In the end, a real-world use case scenario is described and used as a case study.

CONTENT

LIST OF TABLES

LIST OF FIGURES

ACRONYMS

IIoT - Industrial Internet of Things
IoT - Internet of Things
CPS - Cyber-physical Systems
UI - User Interface
PLC - Programmable Logic Controller
TCP - Transmission Control Protocol
IP - Internet Protocol
OPC-UA - Open Platform Communications Unified Architecture
MQTT - Message Queuing Telemetry Transport
AWS - Amazon Web Services
ID - Identification
JSON - JavaScript Object Notation
PDF - Portable Document Format
JWT - JSON Web Token
IDE – Integrated Development Environment
SQL - Structured Query Language
SVN - Subversion
HTML - Hypertext Markup Language
CSS - Cascading Style Sheets
SCSS - Sassy Cascading Style Sheets
JS - JavaScript
OS - Operating System
YAML - Yet Another Markup Language
XML - Extensible Markup Language
CLI - Command Line Interface
SDK - Software Development Kit
RDS - Radio Data System
SNS - Social Networking Service
SQS - Simple Queue Service
API - Application Program Interface
PHP - Hypertext Preprocessor
RDBMS - Relational Database Management Systems
REST - Representational |State Transfer
MIT - Massachusetts Institute of Technology
URL - Uniform Resource Locators
DOM - Document Object Model
JSX - JavaScript XML
NPM - Node Package Manager
I/O – Input/Output
RBAC - Role-based Access Control
APP - Application
HMAC - Hash-based Message Authentication Code
RSA - Rivest-Shamir-Adleman
ECDSA - Elliptic Curve Digital Signature Algorithm
RBA - Role-based Authorization
CDN - Content Delivery Network
ECS - Amazon Elastic Container Service
ELB - Application Elastic Load Balancer

S3 - Amazon Simple Storage Service
EC2 - Amazon Elastic Cloud Compute
VPC - Virtual Private Cloud
ACM - Amazon Certificate Manager
SSL - Secure Sockets Layer
TLS - Transport Layer Security
UX - User Experience
UI – User Interface

1 Introduction

The advent of the Internet of Things (IoT) has marked a new era of connectivity and innovation in technology. IoT is a technology ecosystem that connects physical devices and sensors to the internet, enabling them to collect and exchange data. This data can be used to gain insights, automate processes, and create new services that improve efficiency and enhance our daily lives.

The concept of IoT has been around for decades, but it wasn't until recent advancements in wireless communication and sensor technology that it became feasible on a large scale. With IoT, everyday objects such as appliances, cars, and even clothing can be transformed into smart devices that gather and transmit data, enabling new levels of automation and efficiency.

IoT initially emerged as a consumer technology, focused on connecting devices such as smart home appliances and wearable devices to the internet. However, as the technology matured and became more widespread and as manufacturers began to see the potential of using IoT to monitor equipment and optimize production processes, it became clear that IoT had the potential to revolutionize the manufacturing industry as well. As a result, IoT started to be integrated into industrial settings, leading to the emergence of the Industrial Internet of Things (IIoT).

IIoT builds on the basic principles of IoT but is specifically designed for use in industrial applications. The evolution from IoT to IIoT, marks a significant shift in the use of connected devices in the industry. IoT and IIoT have already revolutionized many processes, from healthcare and industries to education and defense security [1]. However, IIoT is designed for heavy-duty tasks such as manufacturing, monitoring, and control. So, IIoT uses more precise, durable, and resistant devices, actuators, and sensors. Both IoT and IIoT share a set of core principles such as data management, network, security, and the cloud. The main differences between IoT and IIoT are the scalability and the volume of the generated data and how this data is handled and processed. Since IIoT devices generate massive amounts of data, new technologies and techniques are required such as data streaming, big data, machine learning or artificial intelligence. As IIoT is often employed in sensitive and mission-critical processes, it must be ensured that the data it handles is continuous and more precise than what traditional IoT systems require. For instance, considering a monitoring system in a nuclear power plant or a manufacturing facility should be precise and continuous in order to prevent hazardous events.

Industry 4.0, on the other hand, is a broader concept that refers to the fourth industrial revolution, where advanced technologies such as AI, robotics, and big data analytics are used to create smart factories that are highly automated and optimized to run at the highest efficiency possible. Industry 4.0 and IIoT are closely related because IIoT is one of the key technologies that enable Industry 4.0. IIoT provides the real-time data and connectivity that is necessary for smart factories to operate effectively. Without IIoT, it would be impossible to create the level of connectivity and automation that is required for Industry 4.0.

The advent of the Industrial Internet of Things (IIoT) and Industry 4.0 has revolutionized the way industries operate by enabling the integration of advanced technologies such as real-time sensor data monitoring into traditional manufacturing processes. The ability to collect and analyze large volumes of real-time data from machines, processes, and systems has opened up new possibilities for optimizing manufacturing processes, increasing efficiency, and reducing costs.

Real-time sensor data monitoring plays a crucial role in this process by providing real-time insights into the performance of industrial processes and enabling predictive maintenance. This technology has the potential to transform traditional manufacturing by facilitating proactive decision-making and providing valuable insights into the performance of individual machines, the production line, and the overall production process.

However, implementing real-time sensor data monitoring in the manufacturing industry can also be challenging [2]. Manufacturers must ensure that the data is collected and transmitted correctly and securely, to avoid the risk of data breaches or cyber-attacks. They are also requested to make significant investments in new technologies and implement new systems able to process and extract benefits from the large volumes of collected data.

In this context, in order to take advantage of the full potential of IIoT, manufacturers need new applications, especially web-based ones, that can process, analyze, and present the collected data in a meaningful way.

In order for manufacturers to be able to detect issues and anomalies as they occur, and provide them with the ability to proactively address issues before they become larger problems, new strategies to efficiently manage and analyze the data in real-time are needed. Web applications are essential for IIoT as they provide an efficient and user-friendly interface to manage, analyze and visualize the data collected from sensors and devices.

Web applications are software programs that can be accessed through a web browser. They provide an interface for managers and engineers to efficiently manage and analyze the data collected from sensors and devices. Web applications can display real-time data, set up alerts and notifications, and enable remote monitoring. In addition, web applications enable remote monitoring, which allows managers and engineers to monitor production processes and equipment from anywhere in the world. This is particularly important in today's globalized manufacturing environment, where production may be spread out across multiple locations.

1.1 IoT

The Internet of Things (IoT) refers to a network of physical devices, such as sensors and appliances, that are connected to the internet and can collect and exchange data. These devices can be remotely monitored and controlled, creating a seamless integration between the physical and the digital world.

IoT is a rapidly evolving technology that has garnered significant attention in recent years for its potential to revolutionize a wide range of industries and aspects of daily life. With the widespread deployment of the Internet of Things technology, the number of connected IoT devices has increased in an incredible trend by generating huge amounts of data. The growth of the IoT is being driven by advances in sensing, communication, and computing technologies, and its impact is being felt across a variety of domains, including healthcare, manufacturing, energy, transportation, and more. The real-time analysis of IoT data can give useful information which can aid in decision-making on these systems, leading to an enhancement of both system reliability and efficiency [2]. With the increasing proliferation of connected devices, IoT has the potential to create a more interconnected and efficient world by allowing objects to communicate and exchange data in real-time.

1.2 Industrial IoT

The industrial internet of things (IIoT) term refers to a number of interconnected sensors, instruments, and other devices that are networked together with computers that have specific industrial applications, like manufacturing and energy management [3]. It is a rapidly emerging and evolving technology that is transforming the way industrial systems and processes are monitored and controlled. By integrating different advanced connectivity, communication, and computing technologies into industrial systems, the IIoT is enabling real-time monitoring, control, and optimization of many industrial processes. The growth of the IIoT is being driven by the increasing availability of connected devices, big data analytics, and cloud computing, which are enabling new levels of automation, efficiency, and data-driven decision-making. This enhanced connectivity and data collection, exchange, and analysis, facilitates improvements in productivity and efficiency as well as other economic benefits.

The IIoT is enabled by different technologies that interact together such as Cyber-physical systems (CPS) which integrate the dynamics of physical processes with those of software and communication, cloud computing. Another technology that empowers IIoT is edge computing which is a distributed computing paradigm that brings computer data storage closer to the location where it is needed. In contrast to cloud computing, edge computing refers to decentralized data processing at the edge of the network. Big data analytics is also another key technology that enables the examining large and varied datasets [3].

The Layered Modular Architecture

IIoT systems are usually designed following the Layered Modular Architecture, where each layer of the architecture represents different digital technological stacks, as shown in Table 1.1, Figure 1.1, and Figure 1.2 which list some examples of the possible technological stacks present in each of the layers of the architecture and their hierarchy [3].

The device layer refers to the physical components which generate the data. The network layer consists of communication protocols that handle and transport the data generated by the physical layer, while the service layer consists of applications that manipulate, combine and visualize such data. The top-most layer of the architecture, the content layer, also referred to as the user interface, is in charge of handling the interaction between the user and the application.

Layered Modular Architecture in IIoT	
Content layer	UI devices like computer screens, tablets, smart surfaces
Service layer	Applications that manipulate and visualize data, like dynamic dashboards
Network layer	Network Communications protocols like Ethernet, Wi-Fi, 4G/5G, Bluetooth
Device layer	Hardware: Smart edge devices, cyber-physical systems (CPS), machines, sensors

Table 1.1: The four layers of the IIoT Layered Modular Architecture

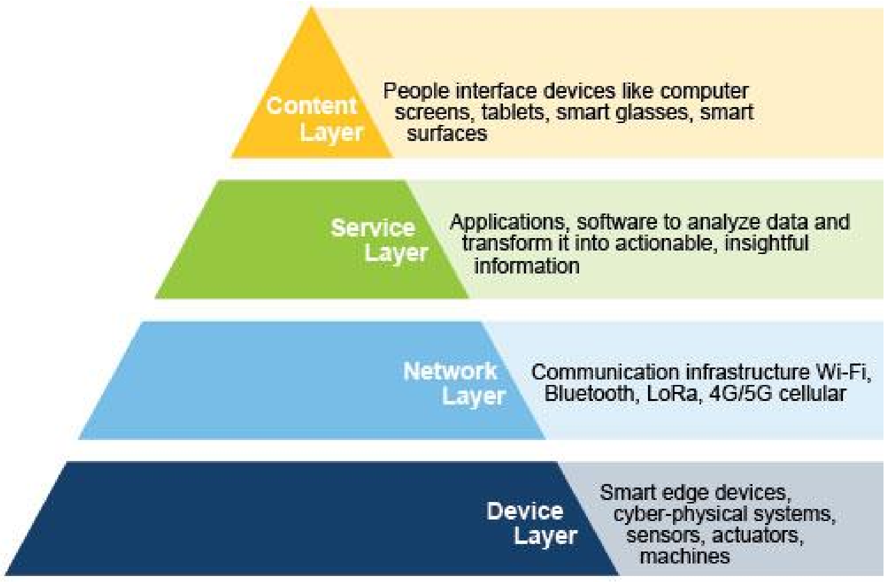

Content Layer
People interface devices like computer screens, tablets, smart glasses, smart surfaces
Service Layer
Applications, software to analyze data and transform it into actionable, insightful information
Network Layer
Communication infrastructure Wi-Fi, Bluetooth, LoRa, 4G/5G cellular
Device Layer
Smart edge devices, cyber-physical systems, sensors, actuators, machines

Figure 1.1: The four layers of the IIoT Layered Modular Architecture
Source: [4]

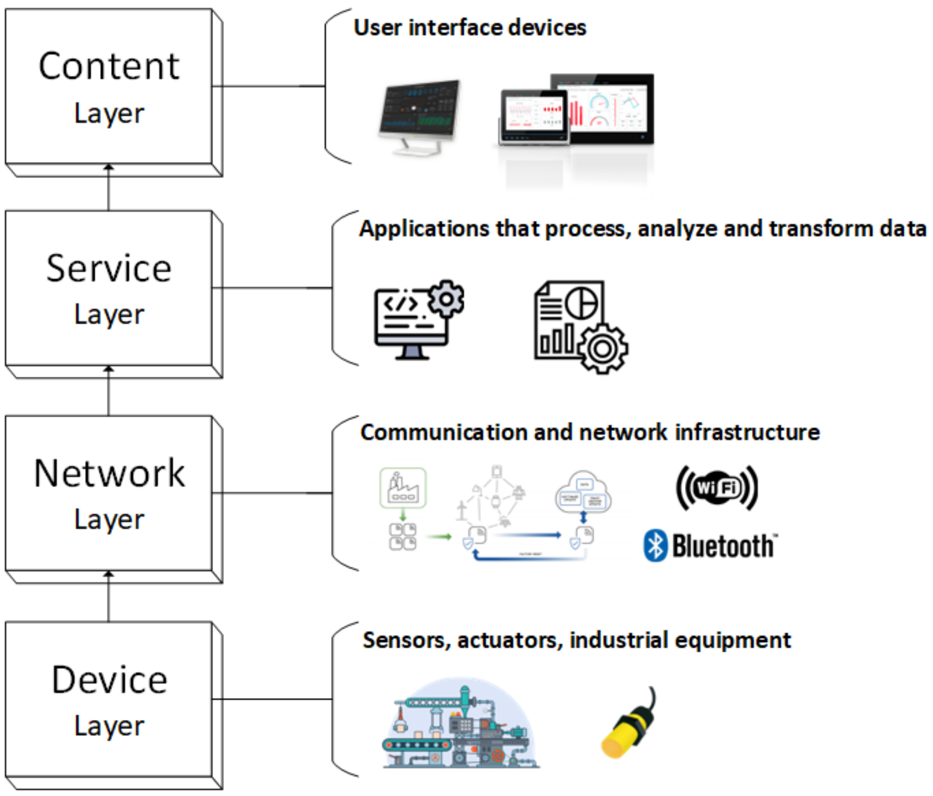

Content Layer
User interface devices
Service Layer
Applications that process, analyze and transform data
Network Layer
Communication and network infrastructure
WiFi
Bluetooth
Device Layer
Sensors, actuators, industrial equipment

Figure 1.2: The four layers of the IIoT Layered Modular Architecture

1.3 Industry 4.0

Industry 4.0, also known as the Fourth Industrial Revolution, refers to the latest step of development in the digitalization of industry, characterized by the convergence of physical and digital technologies. The term "Industry 4.0" was first used at the Hannover Fair in Germany in 2011. Later, the term

Industry4.0 was adopted by the German government in 2013 as a strategic attempt to revolutionize the manufacturing industry and label the strategic industrial policies within the country [5]. It represents a new stage in controlling and organizing the full cycle of the industrial value chain.

At its core, Industry 4.0 is about creating a new level of integration and collaboration between people, machines, and data, with the goal of improving productivity, efficiency, and competitiveness. This integration is made possible by the widespread adoption of connected devices, sensors, and communication technologies, which enable real-time monitoring, control, and optimization of complex industrial processes. For almost an era, it has been subject to active research domain, driving a fast evolution in different industrial processes. This quick evolution has brought different advances and advantages like high-speed and low-cost electronic logic circuits and much quicker signal processing capabilities [6]. Likewise, the number of sensor systems used, and the variety of their applications is increasing constantly. These sensors, together with different devices can communicate and collaborate via the internet to remotely perform process monitoring and control.

The goal of Industry 4.0 is to create smart factories and supply chains that are highly flexible, efficient, and agile, capable of responding quickly to changing market demands and customer needs. Even though this technology is already in use, the fourth industrial revolution is served and is considered the future of manufacturing. The result is a new level of automation and collaboration, where machines and systems are able to work together seamlessly, sharing data and making decisions in real-time.

The widespread adoption of the Internet of Things (IoT) and in turn the growth of cloud computing and edge computing, provide organizations with the ability to store, process, and analyze large amounts of data generated by industrial systems. These advances also allow for advanced real-time monitoring and control of industrial systems and enable other state-of-the-art opportunities like predictive maintenance. Organizations now have the possibility to achieve operational performance levels previously unachievable without the use of the IoT, cloud computing, and cyber-physical systems.

1.4 AROL Group

AROL Group includes AROL Closure System, a global point of reference for 40 years in the design, production, and distribution of closure, feeding and cap orientation systems and UNIMAC-GHERRI, specialist in filling and closing glass containers with twist-off caps. They also include TIRELLI which focused on packaging plants for the cosmetic industry and MACA Engineering, specialized in the design and production of machines and complete lines for the production, assembly, and cutting of aluminum and plastic caps.

The solutions proposed by AROL GROUP can thus be used in the beverage, wine and spirits, food, cosmetic, home care, and chemical sectors. To date, AROL GROUP has installed more than 30,000 machines worldwide [7].

1.4.1 AROL and IIoT

AROL, being a manufacturer of industrial machineries, with a large number of them deployed in the field, has naturally shifted its focus on adapting to the rapid technological changes that the Industrial Internet of Things has introduced. They too have adopted the IIoT Layered Modular Architecture using different robust industrial technological stacks.

Table 1.2, Figure 1.3, and Figure 1.4 show in detail the different technologies that AROL currently uses in its IIoT Layered Modular Architecture technological stack.

AROL Layered Modular Architecture	
Content layer	In-machinery industrial display system (human-machinery interface)
Service layer	In-machinery embedded software and firmware
Network layer	Modbus TCP/IP protocol, PackML, Bluetooth
Device layer	Capping machineries and their built-in sensors

Table 1.2: The four layers of the IIoT Layered Modular Architecture as currently implemented by AROL

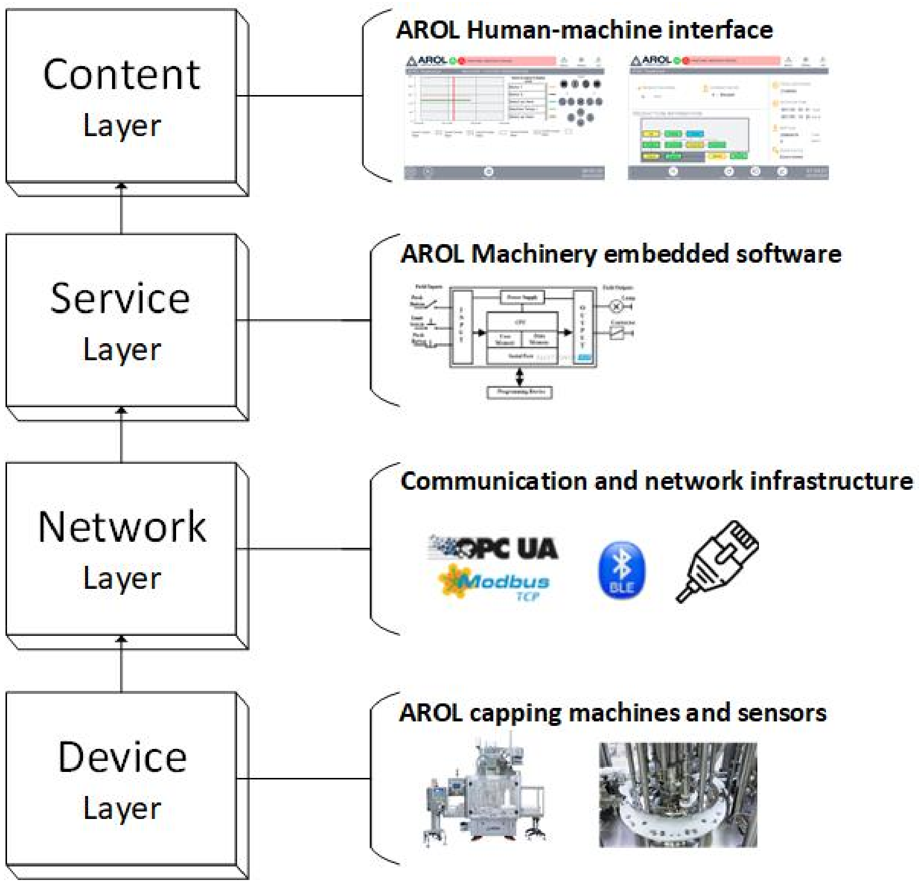

Content Layer
Service Layer
Network Layer
Device Layer
AROL Human-machine interface
AROL Machinery embedded software
Communication and network infrastructure
OPC UA
Modbus TCP
BLE
AROL capping machines and sensors

Figure 1.3: AROL current IIoT architecture and infrastructure

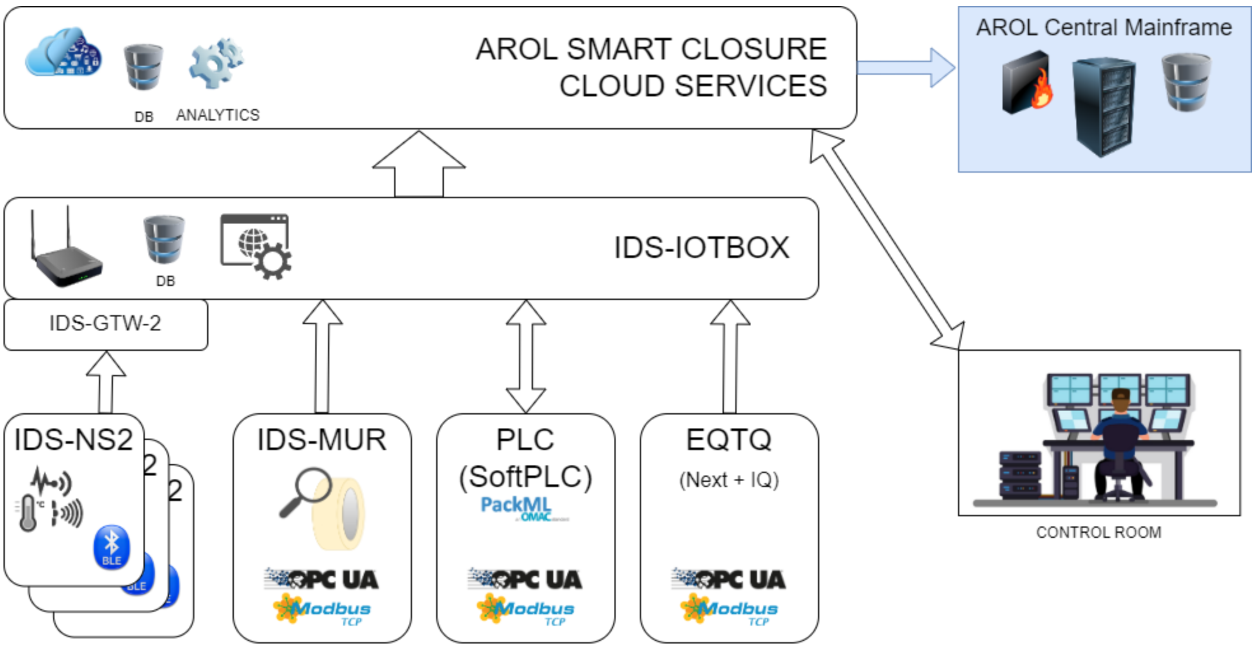


Figure 1.4: AROL current IIoT architecture and infrastructure

Device Layer

The device layer is represented by the capping machineries. Capping machineries are industrial equipment used to apply and secure a cap or closure onto a container, such as a bottle or a jar. Capping machines are widely used in the packaging industry to secure the contents of containers, maintain the freshness and quality of the product and protect it from contamination. A capping machinery can be automated or manual and can vary in size and complexity. Some machines, like most of the machines that AROL produces, are designed for high-speed, high-volume operations, while others are designed for lower-volume or specialized applications. An example of a modern, high-volume AROL capping machine is shown in Figure 1.5 and Figure 1.6.

AROL capping machineries are designed to provide efficient, reliable, and accurate capping of containers, and to ensure that the closures are securely and properly applied. Their capping machineries can improve the speed, accuracy, and consistency of the capping process, and minimize the risk of product contamination or waste.

Network Layer

AROL capping machineries are equipped with different sensors that retrieve real-time information from the different machinery components. Sensors can be mounted in the central body of the machinery or inside the capping heads. Body-mounted sensors are managed and polled by the machinery PLC, which connects to the sensors using the Modbus TCP/IP protocol, which relies on the IEEE Ethernet 802.3 protocol. The machinery PLC also complies to different industry standards, like PackML, which is an industry technical standard for the control of packaging machines, as an aspect of industrial automation. The head-mounted sensors, on the other hand, are polled via Bluetooth or Modbus TCP/IP.

In order to adapt to the rapid changes that Industry 4.0 and the IIoT are introducing in the modern industry, AROL has extended and modernized the aforementioned process of sensor data collection. Sensor data exchange with the cloud infrastructure is supported by the OPC UA standard, which is a cross-platform, open-source standard for data exchange from sensors to cloud applications. By combining edge computing devices and custom Node-RED scripts, sensor data processing is brought closer to the devices where data is generated. These edge computing devices, which are directly mounted on each machinery, have processing and short-term storage capabilities and they help reduce the amount of data that is being stored in the cloud, since only a small subset of data needs to be sent to the cloud server for long term storage or further processing and analyzing, a process also known as miniaturization.

Data ingestion in the cloud is supported by cloud services and protocols like MQTT, a lightweight messaging protocol designed for constrained devices, and AWS IoT Core, a managed cloud service that provides secure, bi-directional communication for Internet-connected devices (such as sensors, actuators, embedded devices, wireless devices, and smart appliances).

Service Layer

Each machinery is equipped with onboard software that controls the operation of the machinery and handles the interactions with the machinery operator. This embedded software acts as the service layer.

The goal of this thesis is to design and implement a web application that, by utilizing machinery sensor data stored in the cloud, will provide a new and more feature-rich service layer to AROL.

Content Layer

Each AROL machinery is equipped with a machinery-mounted industrial-grade display that acts as a human-machinery interface. It is responsible for displaying the embedded UI and handling operator inputs and interactions. Examples of the built-in AROL human-machinery interface are illustrated in Figure 1.7 and Figure 1.8. This display acts as the content layer.

By utilizing the new web application that will be implemented throughout this thesis, the content layer can be extended to computer screens, which can be installed inside the factory or miles away from it.

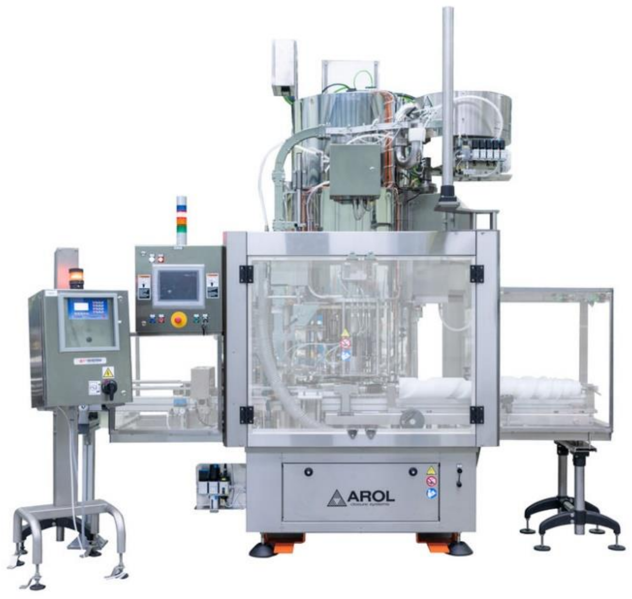


Figure 1.5: Equatorque PK, an example of an AROL capping machinery
Source: [7]

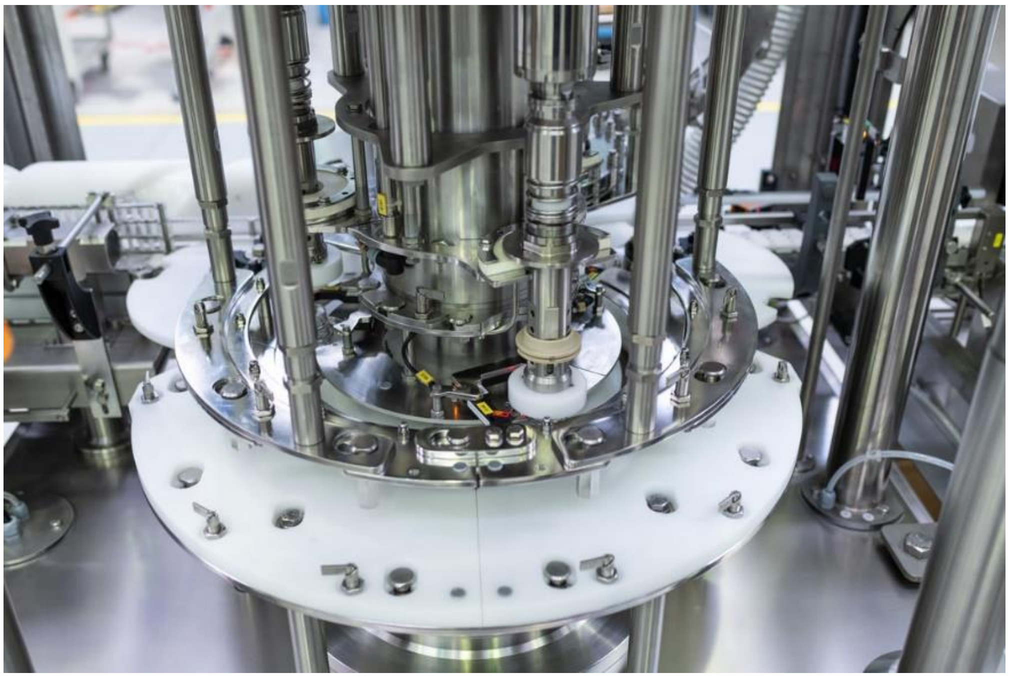
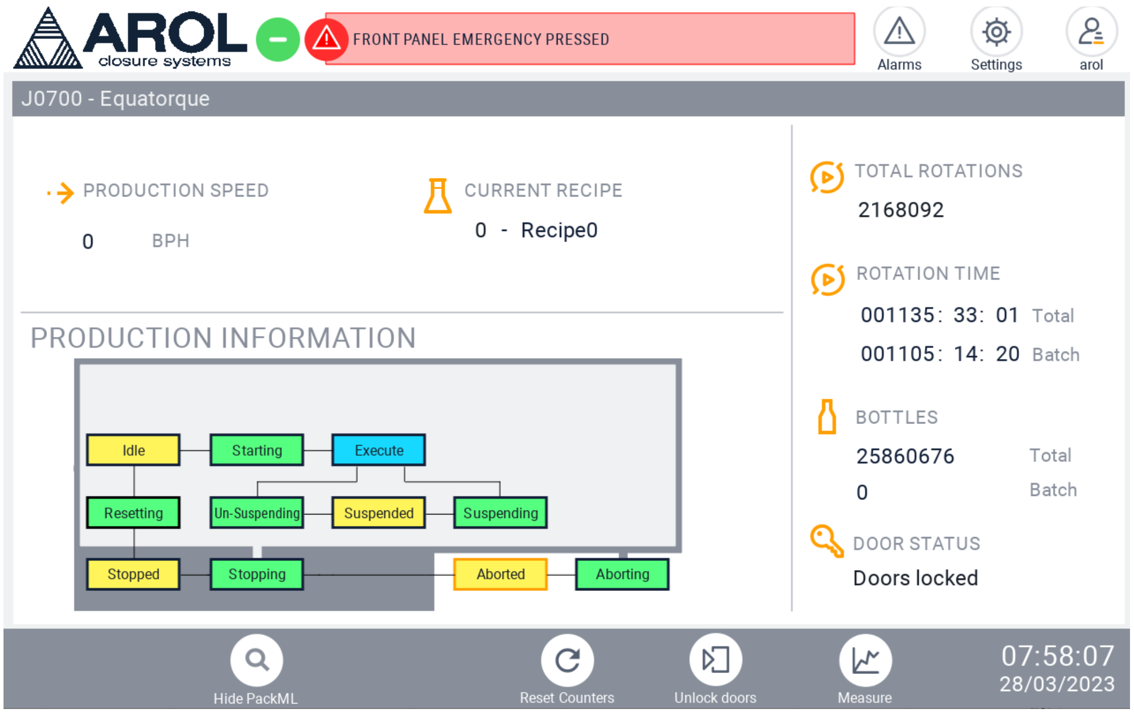


Figure 1.6: Equatorque PK, an example of an AROL capping machinery, close-up view Source: [7]



FRONT PANEL EMERGENCY PRESSED

J0700 - Equatorque

PRODUCTION SPEED
0 BPH

CURRENT RECIPE
0 - Recipe0

PRODUCTION INFORMATION

TOTAL ROTATIONS
2168092

ROTATION TIME
001135: 33: 01 Total
001105: 14: 20 Batch

BOTTLES
25860676 Total
0 Batch

DOOR STATUS
Doors locked

07:58:07
28/03/2023

Figure 1.7: AROL capping machinery built-in human-machinery interface showing the general machinery information

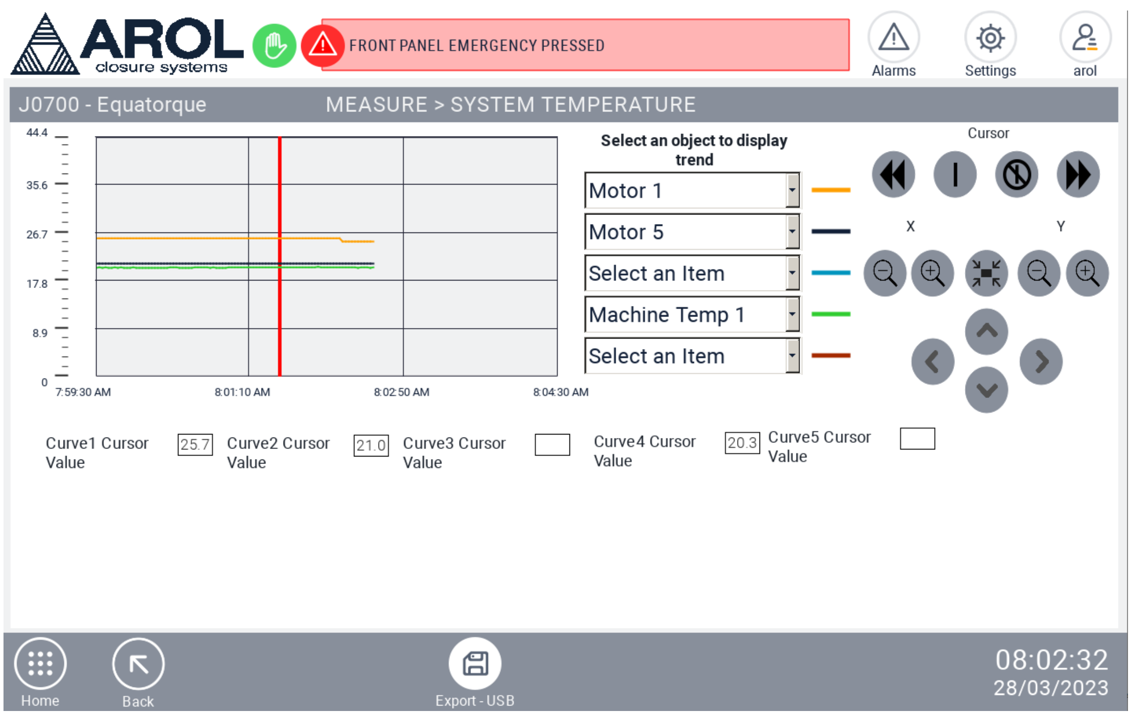


Figure 1.8: AROL capping machinery built-in human-machinery interface showing sensor measurements

1.5 Objectives

The main objective of this thesis is to implement a proof-of-concept custom web application for AROL Group, which must be able to scan, fuse, process, and visualize in a human-friendly manner machinery sensor data that is stored in the cloud. Data visualization must be achieved through a completely customizable and dynamic dashboard solution. Furthermore, the application should provide a per-machinery document storage feature which would help reduce communication overheads between the production plant staff and the machinery operators. All of these functionalities should be protected by a secure authentication mechanism and a Role Based Access Control authorization enforcement scheme.

1.6 Defining requirements

In order to fulfill the aforementioned project objectives, a brainstorming session was organized with AROL stakeholders. During this meeting, some of the main project requirements were defined and finalized.

The proposed web application should implement an AROL client company model which is able to distinguish client companies from each other. The application user accounts must be assigned to a company and the user accounts of a certain company must access only the belonging company resources.

Since the application will target AROL clients, they should be able to browse and list all of their machineries. Customer machineries have a geographic position and they belong to a production plant. AROL client company users must be able to view and brows all company machineries. This interaction should also be implemented using an interactive map navigation system.

In order to support the primary goal of an IIoT real-time monitoring tool, the application should implement a suitable and flexible AROL capping machinery data model. Each machinery has a globally unique ID it can be equipped with multiple capping heads, based on the client's requests. The machinery sensors can be mounted in the machinery body (static position) or in the moving machinery capping heads. Each sensor measurement must have a value and a timestamp.

All the collected machinery sensor data must be visualized through a dynamic dashboard, where clients' users can visualize the collected data from their machineries. The web application must implement a fully dynamic and user-customizable dashboard that can be configured and populated with an unlimited number of different widget types. These widgets should be used to properly visualize and monitor sensor data. The user must be able to insert widgets into the dashboard from an appropriate side panel using a drag-and-drop gesture. Widgets should be resizable, repositionable, and removable, and widget types that are able to monitor many sensors in parallel should be provided to the users. The user must also be able to configure which sensors to monitor in each widget at any point in time. Additionally, sensor data aggregation functions (sample min, sample max, sample average) should also be implemented.

Since a client company may have multiple users that will access the proposed web application, users must be able to independently create multiple dashboards for a certain machinery. The application should also implement a persistence system in order to save and load machinery dashboards. Users should also be able to delete dashboards. As users might be interested to reuse certain dashboard layouts for similar machineries, a dashboard templates feature should be implemented. It should allow importing dashboard layouts from dashboards of machineries of the same model.

In order to support the continuous saving and loading of the dashboards an efficient dashboard data model should be implemented. To aid in implementing the aforementioned requirement, this model should be serializable, preferably in a JSON format.

A shared document storage system where client company staff can share documents also needs to be implemented. The shared document storage should store official machinery documentation which cannot be modified or removed by users. It should also be able to store custom documents uploaded by the staff. For the scope of this thesis, only the storage of PDF files is necessary. The document storage system should emulate a traditional filesystem, with files and folders in order to provide a familiar experience to the users. Users should also be allowed to upload, delete and rename documents and folders present in the shared storage.

Since the application will treat sensitive data, data protection strategies should be properly and correctly implemented. Most importantly, users of a company must not be able to see or modify the data of any other company under any circumstance. A reliable username and password authentication system must also be implemented. The system must also allow, as a security measure, for a user account to be disabled and the password of an account to be reset.

An authorization system based on a Role-Based Access Control (RBAC) authorization strategy must be implemented in order to allow users with different roles and different permissions. The application should provide user accounts that can have 3 roles. The administrator role will be able to access any resource that belongs to his/her company. The manager role will function as the production plant manager which will be able to manage user accounts related to the assigned plant. The worker role, on the other hand, will be able to access resources that are permitted to him. This authorization strategy is also augmented by a custom permissions system that can restrict access to machinery resources, like dashboards and documents. A certain non-administrator user can be assigned read, modify, or write access to both machinery dashboards and documents individually. In order to aid users in quickly identifying which operations are permitted to them, non-accessible actions are visually restricted and disabled on the application frontend. The backend, on the other hand, will strictly authorize every request that it receives in order to allow permissions to be restricted at any time in the lifecycle of the application.

The developed web application must be lightweight and easily deployable. The application should be capable to be deployed locally and it should be capable to be deployed on devices with different operating systems and different hardware resources without any significant changes in the source code. In order to simplify prototype testing and quick in-the-field deployments the application should be capable to be containerized and deployed with Docker.

As the final step of this thesis, the developed web application must be deployable on Amazon Cloud Services (AWS). To support such requirements, all the technologies used in the application must be AWS compatible and a suitable AWS infrastructure needs to be designed in order to host all the components of the application.

2 Local development environment & Tools

This section aims to describe the local development environment and the different tools used during the development of the proposed web application, as shown in Table 2.1. Each tool is then described by providing details on what they offer, which are the processes that they facilitate and arguments are provided to support the importance of each of them.

Local development environment	
JetBrains WebStorm	Development of the frontend
JetBrains IntelliJ IDEA	Development of the backend, manage connections with databases
Docker	Containerization of web application components
LocalStack	Local emulation of cloud service

Table 2.1: The tools used during the development phase of the proposed application and their main function

2.1 IDEs

Integrated Development Environments are software applications that provide a complete and convenient development environment for software developers to write, test, and debug code. They typically include different tools like a code editor, a debugger, build automation tools, syntax highlighting, code completion, and version control integration.

During the development process, IDEs are important because they provide the developer with a range of tools that simplify software development process. By having all the necessary tools in one application, developers can work more efficiently and with fewer distractions.

Another benefit is that IDEs can also help improve the quality of code by providing real-time feedback on issues like syntax errors, incorrect function usage, or even optimization suggestions. By catching these issues early, developers can reduce the likelihood of introducing bugs or performance issues into their code.

2.1.1 IntelliJ IDEA

IntelliJ IDEA is an intelligent integrated development environment (IDE) for developing computer software. It is developed and maintained by JetBrains and is available in both a community edition (free to use) and a commercial edition (with additional features). It was first released in 2001 and has since become one of the most popular IDEs [8]. IntelliJ IDEA is known for its intelligent code completion, on-the-fly code analysis, and refactoring capabilities. It also supports a wide range of languages and frameworks, making it a popular choice among developers [9].

While IntelliJ IDEA is primarily an IDE for the Java programming language, it also understands and supports many other languages like Kotlin, JavaScript, TypeScript, and SQL. IntelliJ IDEA is an all-in-one solution, but if the developer ever needs any extra feature, its rich plugin marketplace is there to help them. It can predict and suggest code completions based on context and can detect and highlight code errors and inconsistencies as code is written, allowing the developer to potential problems before the program is run. IntelliJ IDEA offers also an extremely thorough refactoring functionality. For example, when a class or an interface is renamed, every expression in the codebase impacted by this change will be automatically updated.

IntelliJ IDEA offers also ready-made tools to manage databases. It is able to analyze SQL queries; connect to a large selection of database technologies; run live queries; view and export data; manage database schemas via a dedicated user interface.

Being a modern IDE, IntelliJ IDEA provides a unified user interface for all the most significant version control systems including Git, GitHub, SVN, and many others. By tightly integrating different version control systems, it allows developers to interact with the version control and perform different actions like view the changes history, manage and visualize branches, merge conflicts, and much more without having to leave the IDE.

IntelliJ IDEA supports most of the major application servers like Tomcat, WebSphere, WebLogic, and many others. Code artifacts can be quickly deployed onto application servers and debugged directly from the IDE.

All of these features and more make IntelliJ IDEA a powerful and versatile IDE that is well-suited for a wide range of development tasks and projects.

2.1.2 WebStorm

WebStorm is a JavaScript-based integrated development environment (IDE) optimized for web development, developed and maintained by JetBrains. The first version of WebStorm was released in 2010, and it quickly became one of the most loved IDEs by developers for web development. It is built on top of the IntelliJ IDEA, another platform developed by JetBrains, and is designed specifically for web development requirements using technologies such as HTML, CSS, JavaScript, Typescript, and Node.js [10].

WebStorm provides a wide range of key features for web development. It features a powerful code prediction and completion system that is based on the written code context. WebStorm can detect and highlight errors and problems in the code before the program is run. Furthermore, it includes several tools to help web developers check and improve the quality of code, such as JSLint and JSHint for JavaScript, and CSSLint and SCSS-Lint for CSS, both ubiquitous web programming languages. WebStorm provides a large range of refactoring options, too. It also takes care to automatically fix and refactor any other section of code that may be influenced by the original refactor operation.

As a web programming-focused IDE, WebStorm supports contemporary and modern web development technologies such as ECMAScript, TypeScript, React, Angular, Vue.js, and many others. In order to execute web applications, it includes a built-in terminal, allowing developers to run command-line tools and scripts directly from the IDE.

WebStorm also includes built-in support for the most popular version control systems and has a large ecosystem of plugins and extensions, allowing developers to customize and extend the basic functionalities of the IDE.

In summary, WebStorm is a powerful and versatile IDE made primarily for web developers, providing them with powerful tools and features for writing and debugging web applications, as well as support for popular web development frameworks and technologies.

2.2 Docker

Docker is a platform-as-a-service that uses OS-level virtualization and allows for developing, shipping, and running applications in containers. Docker was first released in 2013 by Dotcloud. The company developed the platform to help with the deployment and scaling of its own platform-as-a-service offering. The technology was open-sourced in March 2013 and the name "Docker" was adopted later in the same year [11].

Docker architecture

As illustrated in Figure 2.1, Docker uses a client-server architecture. The Docker client talks to the Docker daemon, which does the heavy lifting of building, running, and distributing your Docker containers. The Docker client and daemon can run on the same system, or you can connect a Docker client to a remote Docker daemon. The Docker client and daemon communicate using a REST API, over UNIX sockets or a network interface. Another Docker client is Docker Compose, which lets you work with applications consisting of a set of containers [12].

Docker containers

Containers are lightweight software packages that include everything the software needs to run, such as code, runtime, libraries, and settings. This technology is called containerization. It allows developers to pack their applications and their corresponding dependencies into containers that can be easily transferred between different environments. This feature enables the development, testing, and deployment of applications on separate systems, including local development environments and test and production servers. Docker containers share the host's operating system kernel to avoid the overhead associated with running a fully-fledged virtual machine so they are very lightweight and efficient, as described in Figure 2.2 below.

Docker has become one of the most popular containerization platforms and it's being used by developers all over the world. It's supported by major multinational cloud providers such as Amazon, Google, and Microsoft.

Dockerfile

Dockerfile is a simple text file that contains directives and instructions to build a Docker image. A Docker image is purely a read-only file with a collection of instructions. When these instructions are executed by the Docker Engine, a Docker container is created. An example of a working Dockerfile that containerizes a Node.js application is reported below in Figure 2.3.

Docker Compose

Docker Compose is a tool that assists developers in defining and creating multi-container applications. Compose allows to define the required services in a YAML file, as well as start and terminate them with a single command. YAML is an XML-based scripting language that stands for Yet Another Markup Language. Docker Compose is used for running multiple containers as a single service. Each of the containers runs in isolation but is fully able to interact with the other containers when needed. Figure 2.4 demonstrates a working Docker Compose configuration file that will spin un three docker containers, one for the frontend, one for the backend, and one for the application database.

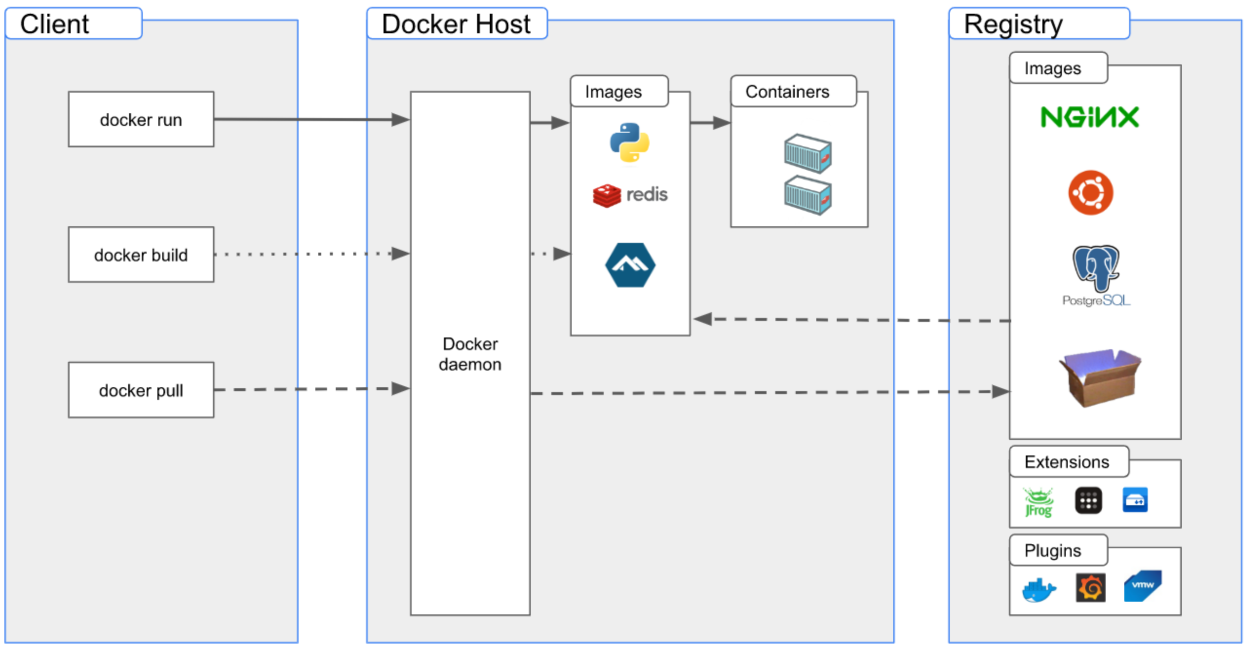


Figure 2.1: Docker architecture [12]

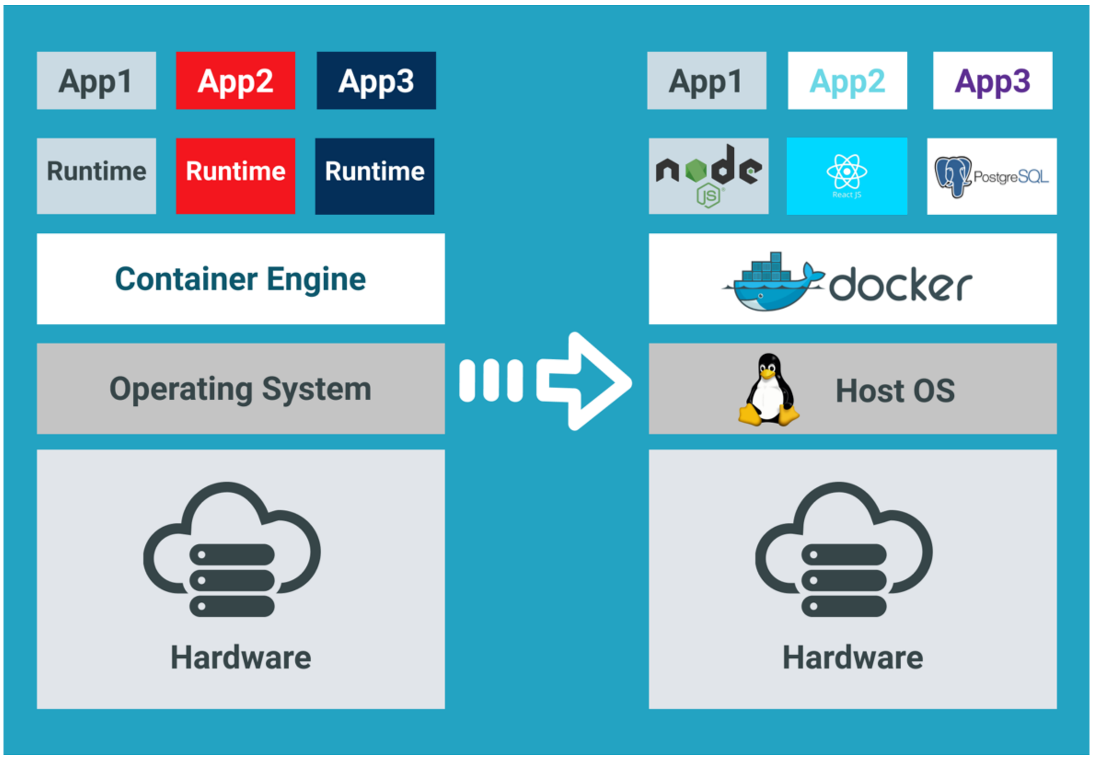
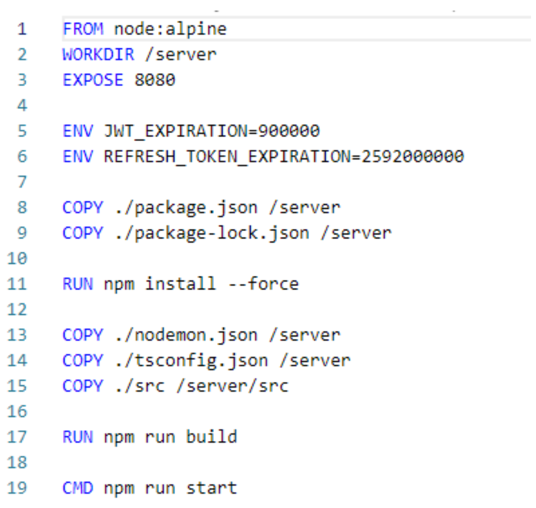
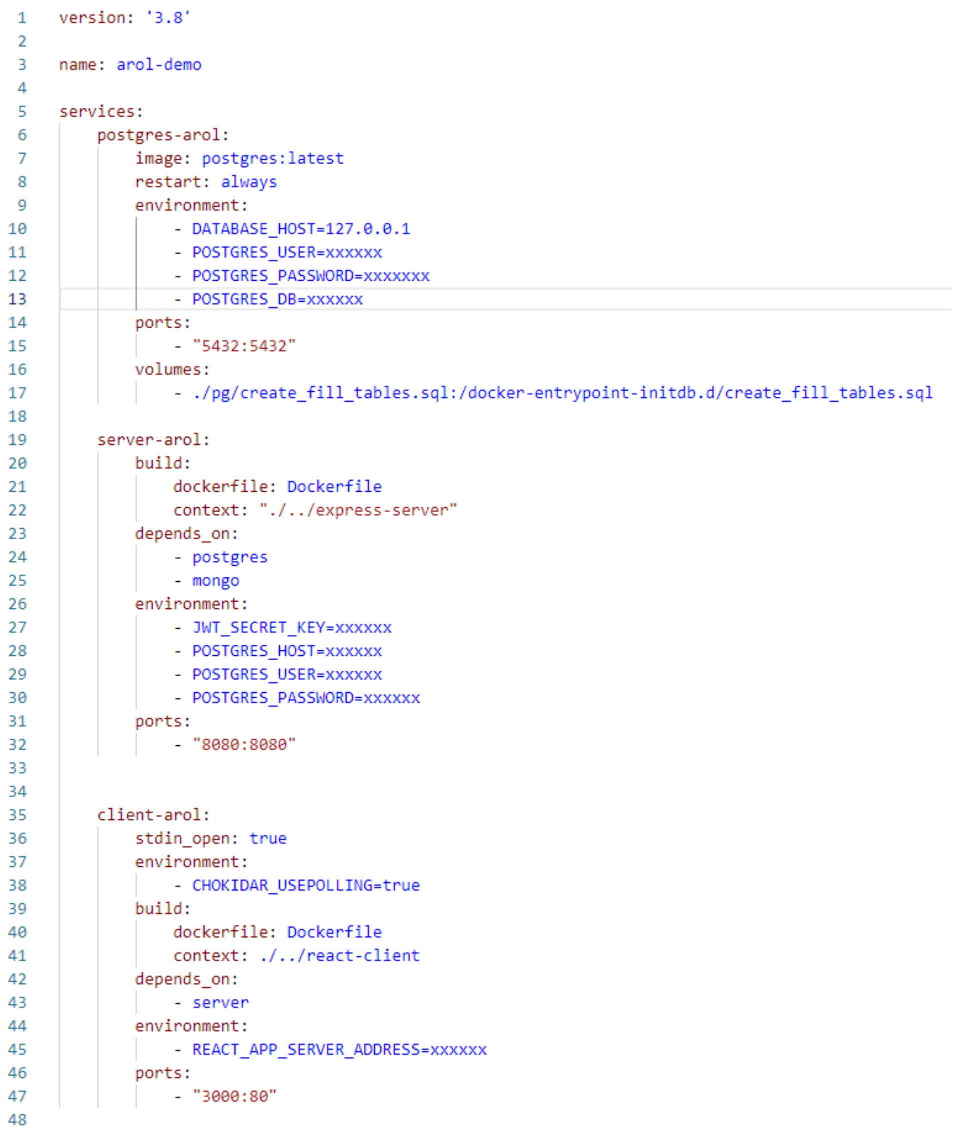


Figure 2.2: Containerization of web application components using Docker



```
FROM node:alpine
WORKDIR /server
EXPOSE 8080

ENV JWT_EXPIRATION=900000
ENV REFRESH_TOKEN_EXPIRATION=2592000000

COPY ./package.json /server
COPY ./package-lock.json /server

RUN npm install --force

COPY ./nodemon.json /server
COPY ./tsconfig.json /server
COPY ./src /server/src

RUN npm run build

CMD npm run start
```

Figure 2.3: Example of a Dockerfile configuration



```yaml
version: '3.8'

name: arol-demo

services:
    postgres-arol:
        image: postgres:latest
        restart: always
        environment:
            - DATABASE_HOST=127.0.0.1
            - POSTGRES_USER=xxxxxx
            - POSTGRES_PASSWORD=xxxxxxx
            - POSTGRES_DB=xxxxxx
        ports:
            - "5432:5432"
        volumes:
            - ./pg/create_fill_tables.sql:/docker-entrypoint-initdb.d/create_fill_tables.sql

    server-arol:
        build:
            dockerfile: Dockerfile
            context: "./../express-server"
        depends_on:
            - postgres
            - mongo
        environment:
            - JWT_SECRET_KEY=xxxxxx
            - POSTGRES_HOST=xxxxxx
            - POSTGRES_USER=xxxxxx
            - POSTGRES_PASSWORD=xxxxxx
        ports:
            - "8080:8080"


    client-arol:
        stdin_open: true
        environment:
            - CHOKIDAR_USEPOLLING=true
        build:
            dockerfile: Dockerfile
            context: ./../react-client
        depends_on:
            - server
        environment:
            - REACT_APP_SERVER_ADDRESS=xxxxxx
        ports:
            - "3000:80"
```

Figure 2.4: Example of a Docker Compose configuration

2.3 LocalStack

LocalStack is an open-source tool that provides an easy-to-use test/mocking framework for developing Cloud applications [13]. It allows developers to create a local version of various cloud services, such as AWS S3, DynamoDB, RDS, SNS, SQS, and many others, on their own development machines. This allows developers to test their applications against a local version of these services, without incurring the cost of using the actual cloud services. LocalStack also provides a simple command-line interface for starting and stopping these services, and for interacting with them via the AWS CLI or SDKs.

LocalStack provides many benefits for developers working on cloud applications as it enables them to run and test their cloud-based applications locally, without having to rely on an internet connection or incur the cost of using actual cloud services. LocalStack allows the developers to work on their applications in an isolated environment, without affecting other applications or services running on the same machine. It also provides a simple command-line interface for starting and stopping services, and for interacting with them via the AWS CLI or SDKs.

To support the local development of cloud applications, LocalStack supports a wide range of AWS cloud services, including AWS S3, DynamoDB, SNS, SQS, Kinesis, Elasticsearch, and many others. This allows developers to test their applications against a variety of services, and to switch between different services as needed. It also allows customizing the behavior of the services it provides, by configuring various environment variables or by providing custom implementations of services. This allows developers to better test their applications in a realistic environment.

In order to use and communicate with LocalStack, a container instance needs to be spun up using Docker. After the instance is up and running, the cloud service emulator can be contacted using the AWS SDK. All the requests need to be sent to the HTTP proxy (by default on port 4567) of the instance, which interprets all the requests and forwards them to the proper service [14]. Figure 2.5 below reports in detail the LocalStack cloud emulator architecture.

Overall, LocalStack is a powerful tool for developing and testing cloud-based applications, by providing a local version of cloud services that developers can use to test their applications in an isolated and controlled environment.

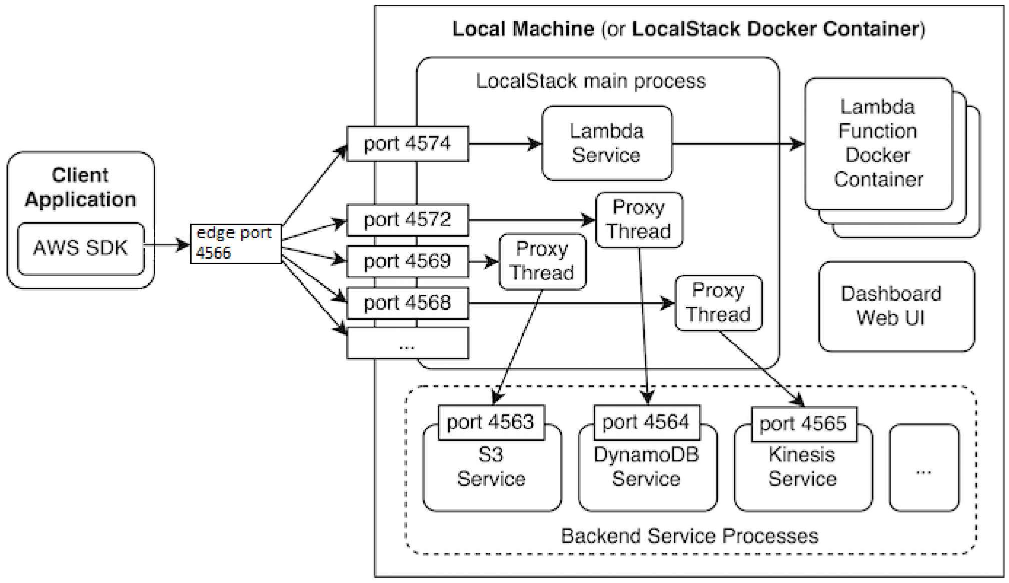


Figure 2.5: LocalStack cloud services emulator architecture high-level overview [14]

2.4 AWS Management Console and CLI

The AWS Management Console is a web-based graphical user interface application that comprises and refers to a broad collection of service consoles for managing AWS resources. It provides access to each service console and offers a single place to access the information developers need to perform their AWS-related cloud tasks. The AWS Management Console also offers access to the individual AWS service consoles, which offer a wide range of tools for cloud computing, service state monitoring, as well as information about costs and billing [15].

The Management console is not the only possibility to interact with the different cloud services that Amazon offers since Amazon also offers the Amazon Web Services (AWS) Command Line Interface (CLI). But the Management Console offers different advantages over the CLI client, such as a user-friendly interface, visual representation of the different services and their functionalities, and more advanced security features such as multi-factor authentication. Another great advantage of the Management Console is that it allows performing multiple tasks simultaneously.

It is important to note that the Management Console is not a silver bullet. The CLI client provides certain features that are not present in its counterpart, such as allowing users to automate different tasks, it allows the integration of AWS resources with other tools, and in enables users to write custom scripts that enable more advanced workflows.

3 Technologies and Methodology

The developed web application was created with the main goal of collecting, processing, and visualizing machinery sensor data in a human-readable format. Sensor data is extracted using NodeRed scripts from a sensor-equipped AROL machinery and is uploaded to an AWS Timestream instance running in the cloud. The backend of the web application, developed using the ExpressJS framework, collects uploaded sensor data from the AWS Timestream persistence layer and app configuration data from a PostgreSQL Operational Database and performs fusion, processing, and manipulation of the collected data. Document storage is implemented using an AWS S3 bucket while the documents and the corresponding folder structure are saved in the PostgreSQL Operational Database. The backend exposes its functionalities over a REST API which is queried by the frontend, developed using the ReactJS library, which in turn is in charge of outputting and visualizing the processed data. Figure 3.1 shows a high-level overview of the application architecture, the overall infrastructure, the data flows, and the role of each component.

This chapter of the thesis aims at explaining in detail all the technologies, the different architecture patterns, and all the methodologies used to implement the proposed web application.

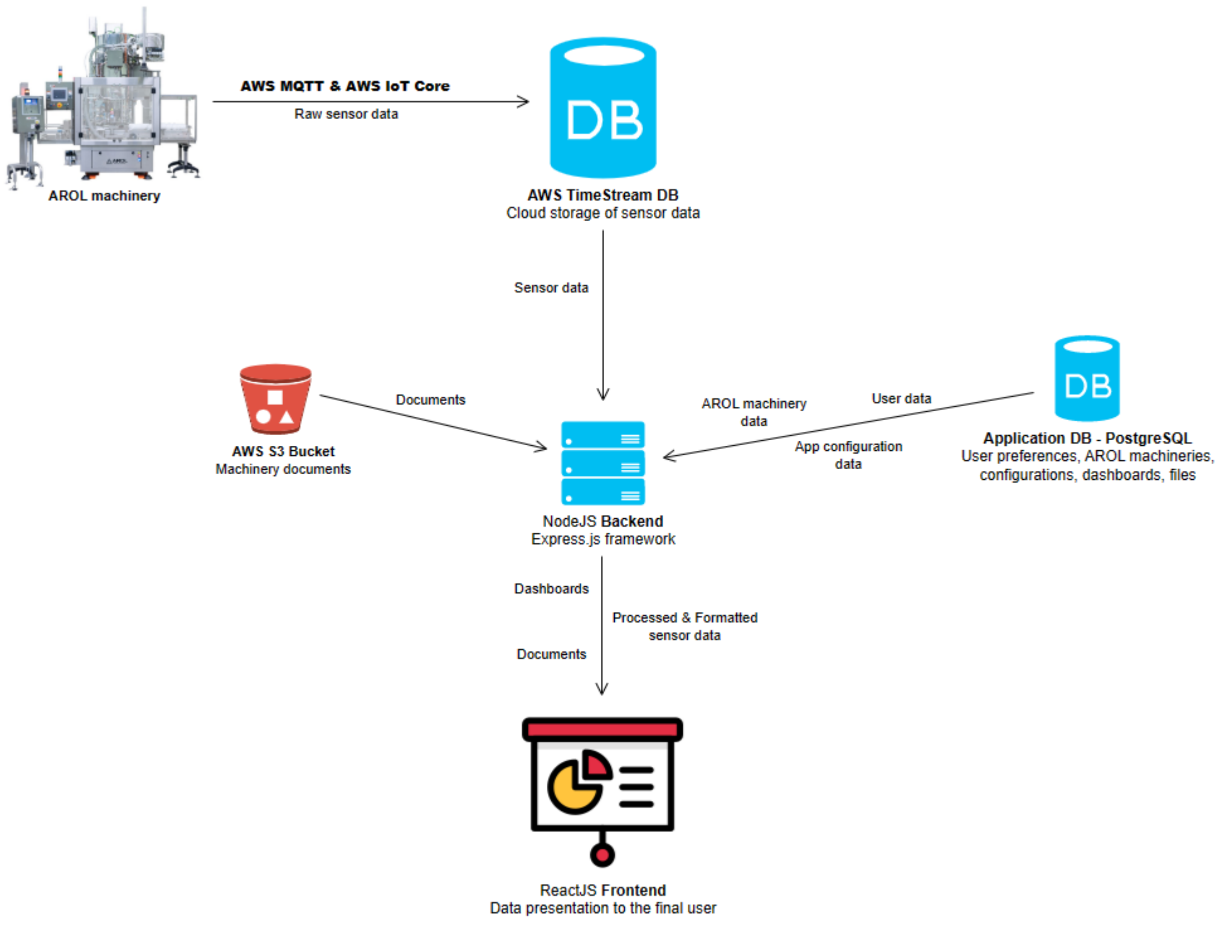


Figure 3.1: Web Application architecture diagram

3.1 Programming languages

In this section of the thesis, the main programming languages used to develop the web application, JavaScript and TypeScript are detailed and their strengths and weaknesses are explained. Since TypeScript is actually a superset of JavaScript that aims to improve the robustness of applications and the developer experience, a comparison is drawn between the two languages.

3.1.1 JavaScript

JavaScript is a widespread lightweight, dynamic, and just-in-time compiled programming language that is one of the core technologies of the World Wide Web, alongside HTML and CSS [16]. JavaScript's dynamic capabilities include runtime object construction, variable parameter lists, function variables, dynamic script creation, and object introspection. It is also a prototype-based language, meaning that objects can inherit properties and methods from other objects, rather than classes. This allows for a more flexible and less structured approach to programming. JavaScript was created as a client-side scripting language, meaning that it is executed on web browsers rather than on a server. This can be easily overcome though, through technologies such as Node.js, which make it possible to run JavaScript on the server side, too.

JavaScript was first developed in 1995 by Brendan Eich at Netscape Communications Corporation, and it was first known as Mocha, then LiveScript. In December 1995, it was officially named JavaScript [17].

JavaScript is widely supported by all modern web browsers, which means that it can be used on all major operating systems and devices. JavaScript can be used to create a wide range of applications, including simple scripts that add interactivity to web pages, complex web applications, and even desktop and mobile apps.

JavaScript is one of the most widely-used web programming languages in the world and has several key features that make it a popular choice for web development. It is the presiding client-side scripting language of the Web, with 98% of all websites using it for this purpose as of February 2022 [18]. Even though JavaScript is a single-threaded language, the JavaScript engine allows asynchronous programming by handling multiple tasks asynchronously at the same time, rather than executing them one after the other in a synchronous manner. This can help prevent the program from freezing when the execution of other code is in progress.

JavaScript allows to build cross-platform applications as it can be used for both front-end and back-end web development, and it can also be used to create desktop and mobile apps. It can also be used and integrated with other web technologies such as HTML, CSS, and various web frameworks and libraries.

One of the strongest points of JavaScript is the fact that a great number of ubiquitous frameworks and libraries such as AngularJS, React, Vue.js, Node.js, and Express.js are developed in JavaScript. These frameworks are of great aid when building large and complex web applications, helping developers by providing a set of tools and pre-written code. These frameworks, provide pre-written code and libraries that help developers to avoid reinventing the wheel every time they write an application.

Overall, JavaScript is a very powerful and versatile programming language that is widely used for web development. It is one of the most popular programming languages for web development and has a large and active community, making it a great choice for web applications of different sizes and complexities.

3.1.2 TypeScript

TypeScript, on the other hand, is an open-source, strongly typed, programming language that is a superset of JavaScript. It was first developed and released by Microsoft in 2012. TypeScript was created by Anders Hejlsberg, who at the time was also the lead architect of C# team at Microsoft [19]. The primary goal of TypeScript was to create a type-safe, object-oriented language that would be a superset of JavaScript. As a superset of JavaScript, TypeScript adds optional static typing to JavaScript. This means that data types of variables, function parameters, and return values can be specified, which can make code more readable and robust, enabling developers to catch code and type errors early in the development process. TypeScript also includes features such as classes, interfaces, and modules, which are not natively supported in JavaScript and can help facilitate the organization and maintenance of large projects. TypeScript also includes a feature called 'type inference' which means that it can automatically infer the type of a variable based on its value, which helps developers reduce the number of explicit type annotations that they need to write.

It's worth noting that TypeScript compiles to JavaScript. TypeScript provides a built-in compiler that can convert TypeScript code into JavaScript code. This means that any code written in TypeScript can run in any JavaScript environment, with the downside of introducing an additional step in the development process, potentially increasing the project's complexity.

TypeScript is becoming increasingly popular among developers, many popular JavaScript libraries and frameworks are written in TypeScript, such as Angular, React, and Vue.js, and many companies are adopting it as their main language for web development.

TypeScript has become increasingly popular over the years, and it's now supported by many popular frameworks and libraries such as Angular, React, and Vue.js. In addition to web development, TypeScript is also being used in the development of desktop, mobile, and server-side applications.

3.1.3 JavaScript vs TypeScript

In this section, a summary of the main differences between JavaScript and TypeScript is reported.
Even though TypeScript is a superset of JavaScript, there are important differences between the two.
To visually aid in understanding the differences, the main differences between the two programming languages are reported in Table 3.1 below.

	JavaScript	**TypeScript**
Language type	Scripting	Object-oriented
File extension	.js and .jsx	.ts and .tsx
Typing	Loosely typed and dynamic typing only	Strongly typed, support both dynamic and static typing
Interface support	No	Yes
Optional parameters support	No	Yes
Generis support	No	Yes
Modules support	No	Yes
Compilation	Not needed	Transpilation into Javascript and compilation

Table 3.1: Summary of differences between JavaScript and TypeScript

3.2 Architecture

In this section, the most important system design and architectural choices and patterns are described and detailed.

The system architecture is crucial for the success of any software project. Without proper design and architecture, software applications quickly become difficult to scale and maintain, leading to costly and time-consuming updates, modifications, and refactoring. A well-designed system architecture ensures that the implemented application not only meets functional requirements but also provides essential features like security, reliability, and overall efficiency.

A well-designed system architecture also helps in minimizing the side effects of errors and bugs and reduces the likelihood of future technical debt. As the project codebase becomes larger in size and as more developers become part of it, a solid system architecture and design allows for better collaboration between developers and provides stable and versatile foundations for the implementation of future features and improvements.

3.2.1 System architecture – multitier architecture

The three-tier architecture is the most popular implementation of the multi-tier architecture [20]. It is a well-established software design pattern [21]. The three-tier architecture is a modular, client-server architecture that consists of a single presentation tier, an application tier, and a data tier [22].

As shown in Figure 3.2 each of the three tiers is a separate logical and physical computing level. The presentation level is the user interface; the application tier is where data processing happens and where the business logic is placed; and the data tier is where application data is stored and managed. Table 3.2 shows a summary of the function and the technology used to implement each of the architecture levels.

Tier	Function	Technology
Presentation/Client Tier	User interface	ReactJS
Application/Server Tier	Business logic and calculations	ExpressJS
Data Tier	Data storage and persistence	PostgreSQL, AWS TimeStream, AWS S3

Table 3.2: Three-tier architecture - function and implementation technology of tiers

Presentation/Client Tier

The presentation tier is the UI and communication layer of the application. It is developed using web technologies and programming languages such as HTML5, CSS, and JavaScript. This level allows the end user to interact with the application. The presentation tier is run in the web browser. It communicated with the application tier through application program interface (API) calls. Its main goal is to display to and collect information from the user.

Application/Server Tier

The application tier also referred to as the server or logic tier, is where the business logic of the application is found. The business logic is a set of business rules that determine how data can be created, stored, and changed. At this level, the information collected from the presentation tier is processed - occasionally against other data in the data tier - using the specified business logic. The application tier is responsible to add, remove or update data present in the data tier.

The application tier is usually developed using programming languages like Python, Java, PHP, or JavaScript using the Node.js runtime and it is in charge of communicating with the data tier using API calls.

Data Tier

The data tier, sometimes referred to as the database tier or data access tier is where the data processed by the application tier is stored, persisted, and managed. The data tier is comprised of a database and a program for handling read and write access to such database. Data tier technologies include relational database management systems (RDBMS) such as MySQL, PostgreSQL, Oracle, or Microsoft SQL Server, or NoSQL Database servers such as CouchDB or MongoDB.

In a three-tier application, communication is only permitted through the application tier. The presentation tier and the data tier cannot communicate directly with one another.

The main advantage of the three-tier architecture is that since each tier is run on its own infrastructure, each tier can be developed separately and can be updated or scaled horizontally on demand without impacting any of the remaining two tiers. Horizontal scalability ensures improved application performance and high availability since many instances of the same tier are active at the same time.

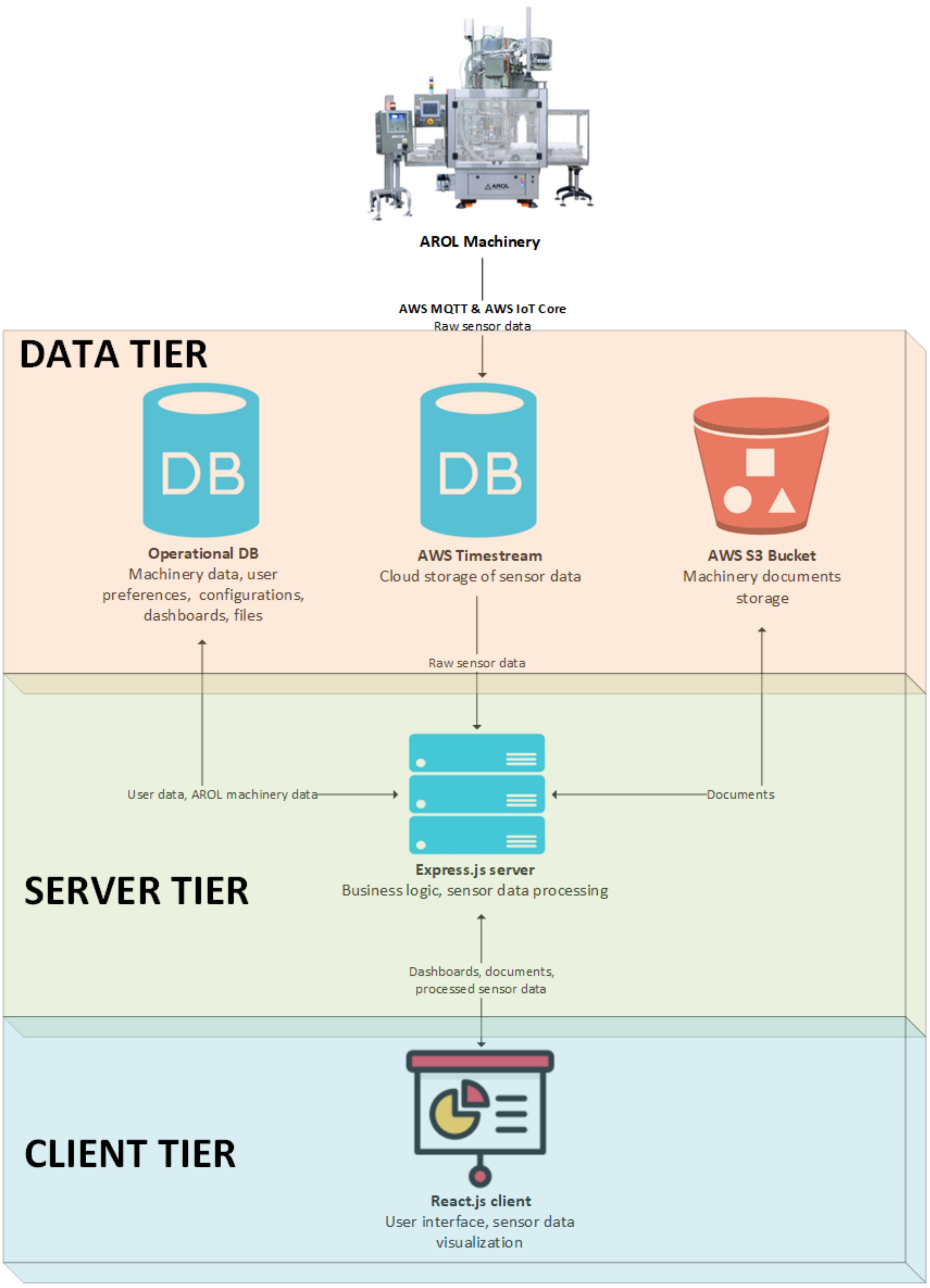

AROL Machinery
AWS MQTT & AWS IoT Core
Raw sensor data
DATA TIER
DB
Operational DB
Machinery data, user preferences, configurations, dashboards, files
DB
AWS Timestream
Cloud storage of sensor data
AWS S3 Bucket
Machinery documents storage
Raw sensor data
User data, AROL machinery data
Documents
SERVER TIER
Express.js server
Business logic, sensor data processing
Dashboards, documents, processed sensor data
CLIENT TIER
React.js client
User interface, sensor data visualization

Figure 3.2: Three-tier architecture [20]

3.2.2 Server architecture – multilayer architecture

The layered architecture pattern also referred to as the n-layer architecture, is one of the most common architectural styles [23]. It is a lightweight architecture that does not suffer from the typical problems of having a large number of abstractions and large amounts of boilerplate code. The reasoning behind the Layered Architecture is that components with similar and homogeneous functionalities are organized into horizontal layers, where each layer is designed to serve a specific purpose within the application, as demonstrated in Table 3.3.

Layer	Function
Routing Layer	Exposing API
Controller Layer	Manage incoming HTTP requests
Service Layer	Business logic and calculations
Repository Layer	Data Tier access methods

Table 3.3: Four-layer architecture - functions of layers

In many resources, the words layer and tier are used interchangeably – and mistakenly. They aren't the same. A layer refers to a functional division of the software, while a tier refers to a functional division of the software that runs on a separate infrastructure from the other tiers. This distinction is important as layers can't offer the same benefits as tiers [21].

In this project, the goal was to design a RESTful API backend service. As depicted in Figure 3.3, the 4-layer architecture pattern was used as the backend architecture for the web application.

Routing layer

The Routing layer is somewhat exclusive to middleware web frameworks, like Express.js. It is in charge of handling the routing functionality and declaring the middleware chain that each request and subsequent response must follow.

Controller layer

This layer is responsible to manage the incoming requests, validate their inputs, and delegate them to the service layer. This layer does not contain any logic nor does it perform any data manipulation.

Service layer

The Service layer is responsible to execute the business logic and rules. Services are in charge of performing data manipulation and coordinating data flow between the other layers of the system. The service layer can communicate with the repository layer to store or retrieve additional data.

Repository layer

This layer is in charge of performing any database-related tasks. This layer abstracts the underlying data storage mechanism and is used as the single point of access to the data storage.

Even though the Layered Architecture is a simple, efficient, and flexible pattern, if not correctly implemented it too has some dangerous pitfalls, as it can result in tightly coupled software components and monolithic applications. Another problem is if any layer is bypassed. If layer bypassing is allowed, security can become a concern. The Service Layer usually acts as an integrity checker for the data that it handles. If the not correctly designed and handled, communication between layers can quickly become complicated and chaotic.

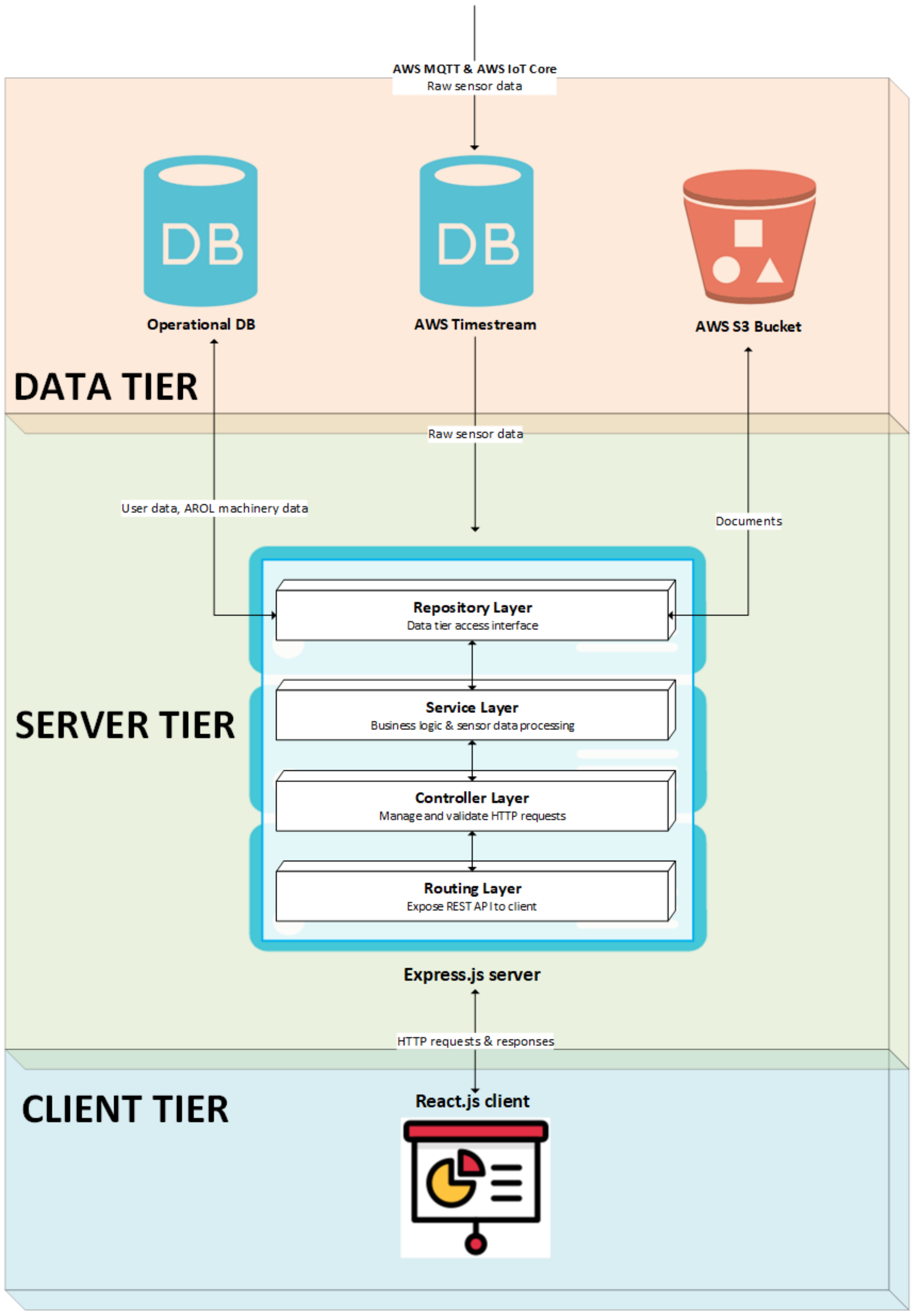


Figure 3.3: Server tier multilayer architecture

3.3 Client Tier

This section of the thesis describes the technologies used to implement the application client, or presentation, tier. The client tier of web applications is usually implemented using web technologies. In order to speed up the development process and provide much richer user interfaces, libraries, like React.js, or dedicated frameworks can also be used.

3.3.1 React.js

React.js, also known as React or ReactJS, is an open-source JavaScript library for building user interfaces or UI components. It was developed and is maintained by Meta, and is now widely used by many companies and developers to build web applications [24].

React.js is a JavaScript library for building user interfaces. React was initially created by Jordan Walke, a software engineer at Facebook, in 2011 [24]. Back in 2011, Facebook had a massive user base and faced a challenging task. It wanted to offer users a richer user experience by building a more dynamic and more responsive user interface that was fast and highly performant. He was inspired by a similar library called XHP, which, at the time, was used to build Facebook's website. Walke's goal was to make it easier to build large web applications with complex user interfaces by breaking them down into smaller, reusable components.

In 2012, React was first used in production at Facebook in the News Feed feature, and it was later used in the mobile version of the site. React was released to the public in 2013 when Facebook open-sourced and released it to the public under the MIT license.

React allows developers to build reusable UI components, which can be easily composed to create complex user interfaces. It also allows developers to manage the state of their applications efficiently. Its popularity has grown rapidly and it is now one of the most widely used JavaScript libraries for building web and mobile applications. React follows a component-based architecture, where the UI is broken down into small, self-contained components that manage their own state and can be easily reused and combined.

Since its initial release, React has grown rapidly in popularity among web developers. It has also had a significant impact on the way web development is done, with many other libraries and frameworks implementing similar component-based architectures, such as Angular and Vue.js.

Typically, you request a webpage by typing its URL into your web browser. Your browser then sends a request for that webpage, which your browser renders. If you click a link on that webpage to go to another page on the website, a new request is sent to the server to get that new page.

This back-and-forth loading pattern between your browser (the client) and the server continues for every new page or resource you try to access on a website. This typical approach to loading websites works just fine but considering a very data-driven website, the back-and-forth loading of the full webpage would be redundant and create a poor user experience.

One of the key features of React is its ability to handle changes to the state of a component and update the UI in a targeted and systematic way, known as the Virtual DOM (Document Object Model). React keeps track of the state of each component, and when the state changes, it only updates the parts of the UI that have changed, rather than re-rendering the entire UI. React figures out the least expensive way to patch the actual DOM with that update without rendering the actual DOM. As a result, React’s components and UIs very quickly reflect the changes since the developer does not have to reload an entire page every time something updates

Another advantage of React is JSX. JSX is a syntax extension, which allows developers to write HTML-like elements in JavaScript code, making it easy to create and manage the structure and layout of the React components.

React follows a one-way data flow architecture, which means that data flows in a single direction, from the parent component to its children components. This allows developers to easily manage the state of a component or the whole application. React also provides the Context API feature that allows data sharing between components without having to use the one-way data flow architecture and without having to pass states through multiple levels of the component tree. This feature helps make the code more manageable and readable.

Recently, React has seen a new addition to its set of features: React Hooks. React Hooks are a new feature in React that allows using state and other features in functional components, rather than having to use class components. This makes code understandable, and maintainable and allows reusing stateful logic across components.

Today, React is one of the most used JS frameworks, with giant companies like Facebook, Instagram, Netflix, Reddit, Uber, Airbnb, Discord, and many others relying on it for their websites.

React is a very powerful and popular library for building user interfaces and web applications, providing developers with a component-based architecture, efficient UI updates via the Virtual DOM, and a wide range of tools and resources to aid them in building efficient and scalable web applications. React has also grown more robust over the years and can now be used to also build native mobile applications using React Native and Desktop apps using Electron.js.

Main libraries

Libraries are collections of pre-written code that can be used by developers to add extra functionalities, simplify and speed up the development process. React libraries are built on top of the core React framework and provide features and tools that are not included in the base framework. Additionally, using well-established and widely-used libraries can often offer higher performance and better security than building custom code since these libraries often have large active communities that support, develop, and check the code for problems or security holes. React has a large and active community and there is a large selection of libraries and tools available.

The most important libraries used in the frontend of the proposed application are listed below.

- **ChakraUI –** a simple, modular, and accessible component library that gives you the building blocks you need to build your React applications. Chakra UI provides a set of accessible, reusable, and composable React components that make it super easy to create websites and apps [25].
- **React Router –** it is a React library that enables managing the URLs of a React web application. It enables defining routes that correspond to specific components or views in the application, making it easy to create complex user interfaces. With React Router, users can navigate between different pages without reloading the entire application, resulting in a faster and smoother user experience [26].
- **React Grid Layout –** a grid layout system, for React. It is responsive and supports breakpoints.
- **React PDF –** a PDF viewer component that allows the rendering of PDF documents.
- **Chonky File Browser –** Chonky is a file browser component for React. It tries to recreate the native file browsing experience in your browser. This means your users can make selections, drag & drop files, toggle between List and Grid file views, use keyboard shortcuts, and much more [27].

- **Leaflet** – an open-source JavaScript library for creating interactive maps and geospatial applications on the web. It provides a simple and lightweight approach to creating maps that can be embedded in web pages or web applications [28].

3.4 Server Tier

In this section, the technologies used to implement the server, or application tier are specified. The application tier is usually implemented using dedicated web application frameworks, which contain best practices, and ready-made code that greatly helps developers to not reinvent the wheel, prevent security problems and reduce the potential of critical bugs.

3.4.1 ExpressJS – a NodeJS framework

Express.js is the de facto server framework for Node.js [29]. Express.js is designed to build web applications and APIs. It provides a minimal and flexible structure for building web applications and is often used in combination with other Node.js modules and libraries to handle tasks such as routing, middleware, and database access. It is open-source and has a large and active community of contributors who have built many useful modules and plugins that can be easily added to Express.js applications.

Node.js is a JavaScript runtime built on Chrome's V8 JavaScript engine. It allows developers to run JavaScript on the server side, creating server-side applications with JavaScript. Node.js was first developed by Ryan Dahl in 2009 [30]. It provides an event-driven, non-blocking I/O model, which makes it lightweight and efficient for data-intensive real-time applications that run across distributed devices. Node.js operates on a single thread, allowing it to handle thousands of simultaneous event loops. It also includes a package manager, called npm, which makes it easy to install and manage libraries and modules for use in Node.js applications.

Express.js was first created by TJ Holowaychuk in 2010, as a simple web framework for Node.js [29]. At that time, Node.js was still a new and relatively unknown platform, and there were very few web frameworks available for it. TJ's goal was to create a minimal and flexible framework that could be used to build web applications and APIs quickly and easily.

Express.js quickly gained popularity among Node.js developers, and its community of contributors grew rapidly. This led to the creation of a large number of additional modules and plugins that can be easily added to Express.js applications.

Express.js has had a significant impact on the Node.js ecosystem and has inspired the creation of many other web frameworks for Node.js, such as Koa and Hapi.

Today, Express.js remains one of the most popular web frameworks for Node.js and is widely used in the development of web and mobile applications. Its simplicity, flexibility, and abundance of modules and plugins make it a great choice for small and medium-sized projects.

Express.js provides a routing mechanism that allows developers to map specific URLs to different functions or controllers in their applications. These functions can handle different HTTP methods (e.g., GET, POST, PUT, DELETE) and respond with appropriate data or views.

Express.js also provides a middleware mechanism that allows developers to specify code that should be executed before or after certain routes. This can be used to perform tasks such as authentication, logging, or data validation.

Express.js is primarily used to build backends for web applications since its simplicity and flexibility make it a great choice for small and medium-sized projects. It is a minimal and unopinionated framework, meaning that it does not come with a lot of built-in functionalities that would force the structure and the architectural organization pattern of the application. This allows developers to structure their applications in the way that works best for them. By being built on top of Node.js, Express.js uses the same non-blocking, event-driven I/O model, which makes it ideal for building performant and scalable web applications. Another benefit of being built on top of Node.js is that Express.js can be easily integrated with other popular Node.js modules and libraries for handling databases, data validation, and authentication.

Express.js has a powerful built-in routing mechanism that allows easily mapping specific URLs to different functions of the application. These functions, usually named controllers, can handle requests with different HTTP methods (like GET, POST, PUT, DELETE, and more) and respond with the appropriate data. This functionality is further extended by a middleware mechanism. A middleware allows developers to specify functions that should be executed before or after certain routes. This pattern perfectly suits the execution of traditional tasks such as authentication, logging, or data validation. Since Express.js has a large and active community, there exist many useful modules and plugins that can be easily added to Express.js to augment its features and functionalities.

In summary, Express.js is a simple, minimal, and flexible web framework that allows developers to quickly and easily build web applications and APIs using JavaScript. It's built on top of Node.js and is well-suited for building high-performance, scalable web applications, and APIs. Its simplicity, flexibility, and abundance of modules and plugins make it a great choice for small and medium-sized projects [31].

Main libraries

Similar to React libraries, Express.js libraries are used to augment the existing feature set and reduce the need to write repetitive code to allow for faster development. Being a very popular framework, Express.js features a very rich selection of compatible libraries.

This list of the most notable libraries used for the development of the backend of the proposed application is as follows:

- **Multer –** a Node.js middleware for handling multipart/form-data, which is primarily used for uploading files.
- **PG-Promise –** a PostgreSQL interface for Node.js.
- **AWS SDK for JS –**The AWS SDK for JavaScript is a software development kit (SDK) that enables building JavaScript applications that use Amazon Web Services (AWS) services. The AWS SDK for JS provides a set of APIs that make it easy to interact with AWS services [32].

3.5 Data Tier

This section contains details regarding the database technologies used by the web application to store and persist data.

3.5.1 PostgreSQL

PostgreSQL is a powerful, enterprise-class, open-source object-relational database management system (ORDBMS). It is known for its advanced features, high performance, and adherence to the SQL standard. It is a very stable database that is supported by more than 20 years of development by the open-source community. The PostgreSQL project started in 1986 at Berkeley Computer Science Department, University of California.

PostgreSQL supports multiple data types such as arrays, key-value stores, and JSON, as well as custom user-defined types as well as custom extension functions and operators. It also supports advanced indexing and searching, including modern features such as full-text search and spatial indexing. To provide performant operations, PostgreSQL supports concurrent access and highly concurrent tasks through an advanced concurrency control feature [33].

PostgreSQL supports data replication strategies and offers high availability through multiple baked-in replication features that can also be extended by a large selection 3rd party tools. As an enterprise-class database management system (DBMS), it offers advanced security features, like role-based access control (RBAC), data encryption, and secure connections.

PostgreSQL, thanks to its feature set, stability, and adherence to industry standards, is often used for mission-critical applications, data warehousing, web and mobile applications, and geospatial systems. It is also a popular database management system as it can be easily installed and run on most platforms.

3.5.2 Amazon Timestream

Amazon Timestream is a fast, scalable, serverless, and fully managed time series database service offered by AWS. It is designed to handle high volume, high velocity, and highly dimensional time series data, making it an optimal choice for IoT applications. A time series is a sequence of records that can be pictured as data points over a certain interval. While time series data can also be persisted in a traditional relational database system, these often experience scaling issues due to the amount of data needed to process in short amounts of time.

Timestream, being a time series database is optimized for storing and querying data that is timestamped, such as sensor data, metrics, and logs. Some typical use cases include any type of data where values or metrics are repeatedly measured at regular time intervals, like IoT data, DevOps data, App analytics, and more.

As Timestream is a type of NoSQL database, it has its own type of data model that is distinct from both traditional SQL models and many other NoSQL models. It is considered to be a schema-less database since there is no enforced schema [34]. However, it still carries over concepts such as databases and tables, together with some specific concepts, defined as follows:

- Database: a collection of tables
- Table: an encrypted repository that stores the time series records
- Record: a combination of a timestamp, at least one dimension, and one measure
- Dimensions: attributes that describe metadata of record always stored as strings
- Measure: the data value that constitutes the measurement

Timestream automatically tiers data based on their age and access frequency. This strategy helps move older, unused data into slower storage devices and move newer, more frequently accessed data into very fast storage devices, resulting in higher performance. Timestream currently has 2 types of storage, a fast memory storage where data is initially stored and deduplicated, and a slower magnetic storage which offers cost-effective and long-term storage.

Amazon Timestream offers multiple features that make it a very attractive option to be used in IIoT scenarios. As it uses a unique architecture that is heavily optimized and tailored for time series data, it supports fast and efficient querying of large data sets. Timestream supports standard SQL querying, allowing developers to easily query and analyze time series data using familiar SQL syntax. It extends the standard SQL syntax with numerous scalar and aggregate functions and additional time series interpolation functions for data points that may be missing. It also offers high scalability and can handle high read and write throughputs.

3.5.3 Amazon Simple Storage Service

Amazon Simple Storage Service (S3) is a service provided by Amazon Web Services (AWS) that allows storing and retrieving large amounts of data, such as files and images, in the cloud.

S3 is organized in a flat structure, meaning that there are no hierarchical relationships between any of the objects stored in a bucket. An S3 bucket is a logical container for storing objects in S3. Each object within the same bucket shares the same permissions and is identified by a unique key. This key is the resulting string composed of the object's name and the name of the bucket in the file was stored [35].

3.6 Project file structure

A web project file structure refers to how files and directories are organized within the web development project. The main goal when choosing a file structure is to show the purpose of each element by separating them into a hierarchy of folders. A suitable file structure makes it easier for anyone with access to the project, developers included, to easily navigate the project files. It also makes locating and accessing files easier. By having the ability to quickly locate files, developers can accomplish more work faster. The choice of the file structure can potentially streamline productivity and increase code consistency and shareability [36].

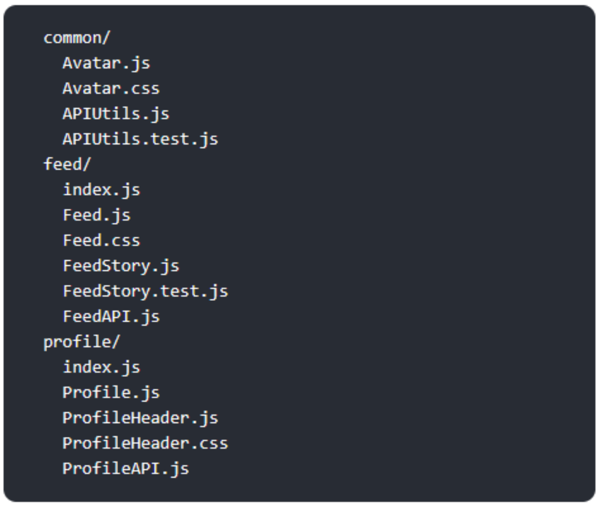
3.6.1 Client file structure

The client file structure chosen for this project is a hybrid between grouping components by their type and grouping them by their corresponding route. Each of the two approaches has its corresponding benefits. The logic behind choosing a hybrid file structure is to have the best of both worlds.

There are benefits and drawbacks to grouping components by route. The main benefit is that the files and folders structure directly communicates the application features, but for some components, a lot of time needs to be spent deciding where to place the feature exactly. This can be highlighted, for example, when a certain component needs to be shared between different routes. Another disadvantage is the risk of ending up with too much of a nested structure. If such a scenario occurs, a refactor might be the only solution. Figure 3.4 illustrates a file structure organized by grouping by component route.

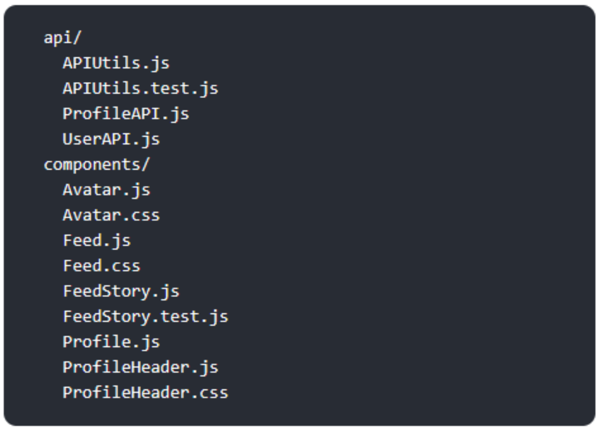
On the other hand, also grouping components by their types has its benefits and drawbacks. This strategy ensures a flatter file and folder structure, but finding out which files are used by which features gets quite tricky too fast as the project size grows. Since each directory contains components that are usually related to each other this can result in a lack of modularity. Another problem is that it might be difficult to integrate new developers into the project as they need to find their way into the codebase that does not communicate the application features. Figure 3.5 illustrates a file structure organized by grouping by component type.

Some examples of grouping by file type are: services (perform API calls), utils (general purpose utility classes), Router (React Router routes), reducers, and contexts [37].



```
common/
  Avatar.js
  Avatar.css
  APIUtils.js
  APIUtils.test.js
feed/
  index.js
  Feed.js
  Feed.css
  FeedStory.js
  FeedStory.test.js
  FeedAPI.js
profile/
  index.js
  Profile.js
  ProfileHeader.js
  ProfileHeader.css
  ProfileAPI.js
```

Figure 3.4: Example of grouping files by route



```
api/
  APIUtils.js
  APIUtils.test.js
  ProfileAPI.js
  UserAPI.js
components/
  Avatar.js
  Avatar.css
  Feed.js
  Feed.css
  FeedStory.js
  FeedStory.test.js
  Profile.js
  ProfileHeader.js
  ProfileHeader.css
```

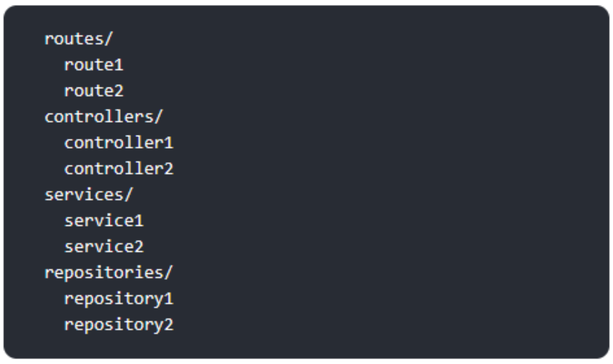
Figure 3.5: Example of grouping files by their type

3.6.2 Server file structure

On the other hand, in the server file structure, components are grouped by their corresponding functional layer, also referred to as Layer-first or Responsibility-based layering [38].

This strategy has different benefits and drawbacks. As long as the project size is not large, it is an easy-to-follow and efficient file and folder structure but problems start to arise when the project codebase becomes large as now layers of the same feature are further and further away from each other and the developer will need a good amount of scrolling to move between the different layers of the same feature.

This approach was chosen for the backend of this project as the number of features that need to be implemented server-side is not too large and the drawbacks of the Layer-first pattern can easily be mitigated. Figure 3.6 illustrates a Layer-first organized file structure.



```
routes/
  route1
  route2
controllers/
  controller1
  controller2
services/
  service1
  service2
repositories/
  repository1
  repository2
```

Figure 3.6: Example of the Layer-first file organization

3.7 Security

Application security is one of the most important aspects of a well-built web application because it helps protect the application itself and the sensitive data it processes. Proper mitigation strategies must be put in place to prevent unauthorized access, data breaches, and malicious attacks targeting the application. It is also important to implement well-tested and best-practice security strategies in order to deliver the proper protection level and mitigate security incidents. This topic is becoming increasingly important every day as modern web applications treat sensitive data and as cyber-attacks are becoming increasingly frequent. This section describes the security measures, including authentication and authorization strategies, put in place during the development of the proposed web application and the motivation behind them.

3.7.1 Authentication strategy

Authentication is the process of verifying the identity of a user before granting them access to resources or allowing them to perform certain actions. It is the procedure of confirming that the user actually is whom they claim to be. Authentication is a very important aspect of application security, especially if the application handles sensitive or confidential information.

Table 3.4 details some user authentication methods that can be used to help secure a system and explains their strengths and weaknesses. This project implements a token-based authentication strategy based on the JSON Web Token.

Authentication strategy	Pros	Cons
Password authentication	Simple	Requires regular renews to keep a high level of security
Multi-factor authentication	Very secure	The infrastructure needed to distribute authentication keys
Certificate-based authentication	Very secure	The very complex infrastructure needed
Biometric authentication	Fast and secure	Complex privacy problems to store sensitive biodata
Token-based authentication	Very scalable, efficient, secure and performant	Sensitive to poor implementations and introduces a data overhead on each request

Table 3.4: Authentication strategies and their pros and cons [39] [40]

JSON Web Token

JSON Web Token (JWT) is a proposed Internet standard that defines a compact and self-contained way for securely transmitting information between parties with optional signature and optional encryption whose payload holds a JSON object that asserts a number of claims. The tokens are signed either using a shared secret (with the HMAC algorithm) or a public/private key pair (RSA or ECDSA).

JWT consists of three parts:

1. Header: identifies the algorithm used to generate the token signature.
2. Payload: contains the token claims. Claims are statements about an entity (for example, the user when the token is used for authentication) and additional data. The claims can be standard claims, which are claims that are commonly included in tokens, and custom claims which can be defined by the developer.
3. Signature: validate in a secure way the token. This part of the token is used to verify that the sender of the JWT is really whom it says it is and to ensure that the message wasn't tampered with along the way.

Figure 3.7 illustrates the structure of a JSON Web Token while Figure 3.8 illustrates an example of a JWT token and shows also a destructured view of the contents of each part of the token.

JWTs are often used to authenticate users in web applications and they can also be used to authenticate and authorize APIs. Once a user is logged in with their credentials, a JSON Web Token is returned by the authenticator. This token must be saved locally in the application so that each subsequent request will also include the JWT in the request header, allowing the user to access API resources that are protected. This strategy replaces the traditional approach in which a session is created in the server and a cookie is returned to the client.

JWT is a stateless authentication mechanism, meaning the server does not need to keep track of the token, it just needs to verify that the token is valid by checking the signature part. This strategy allows for great horizontal scalability and reduces the server load.

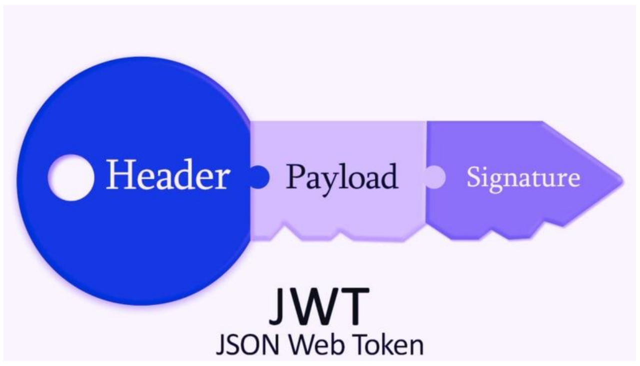
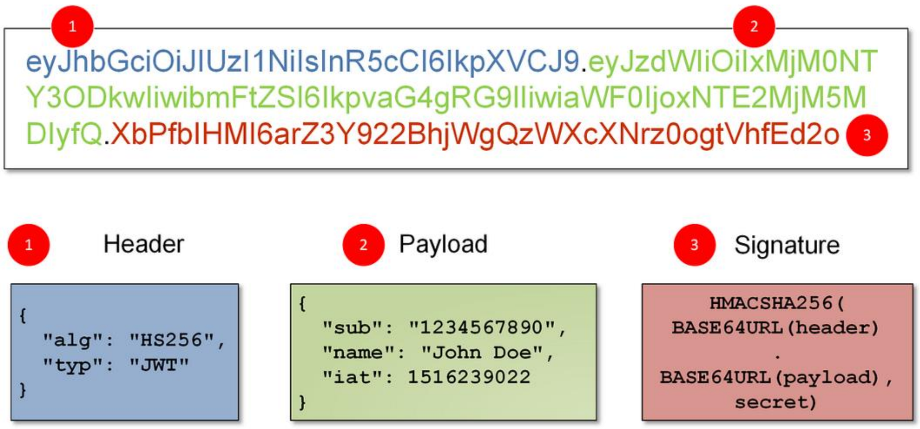


Figure 3.7: JSON Web Token structure
Source: [41]



```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NT
Y3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5M
DIyfQ.XbPfbIHMI6arZ3Y922BhjWgQzWXcXNrz0ogtVhfEd2o
```

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "iat": 1516239022
}
```

```
HMACSHA256(
BASE64URL(header)
.
BASE64URL(payload),
secret)
```

Figure 3.8: JSON Web Token Destructuring
Source: [42]

JWT Authentication

JWT authentication works by having the server issue a token to a client which has provided valid credentials. The client then includes this token in every subsequent request to the server. The server then validates that the token has not been tampered with and that it has not expired. If such a check is successful then the request was received from an authenticated user.

A high-level overview of how this procedure works is as follows:

a. User registration
   1. The client sends its registration details to the server
   2. The server checks if a user with the same details already exists. If not, the new user details are stored in the database and a success message is sent back to the user.
   3. The user receives a confirmation of the successful registration
b. User login and protected resource access
   1. The client sends a login request with their credentials to the server.
   2. The server authenticates the client by validating its credentials and if the validation was successful, it creates a JWT and sends it back to the client.
   3. The client receives the generated JWT and stores it locally (in a browser cookie or the local storage).
   4. In every subsequent request, the client sends the previously received JWT in the HTTP request headers.
   5. The server receives the new client request and extracts the JWT from the HTTP request headers. The signature of the JWT is validated to ensure that the token has not been tampered with since its generation. If the signature is valid, the server decodes the token

to access its claims (also referred to as the payload) and identifies the client who sent the request. If the client is authenticated and authorized to access the requested resource, the server processes the request and replies back to the client with a response.

6. The client receives back from the server the requested resource.

Figure 3.9 represents visually the token authentication flow described above, including both the registration and the protected resource access procedure.

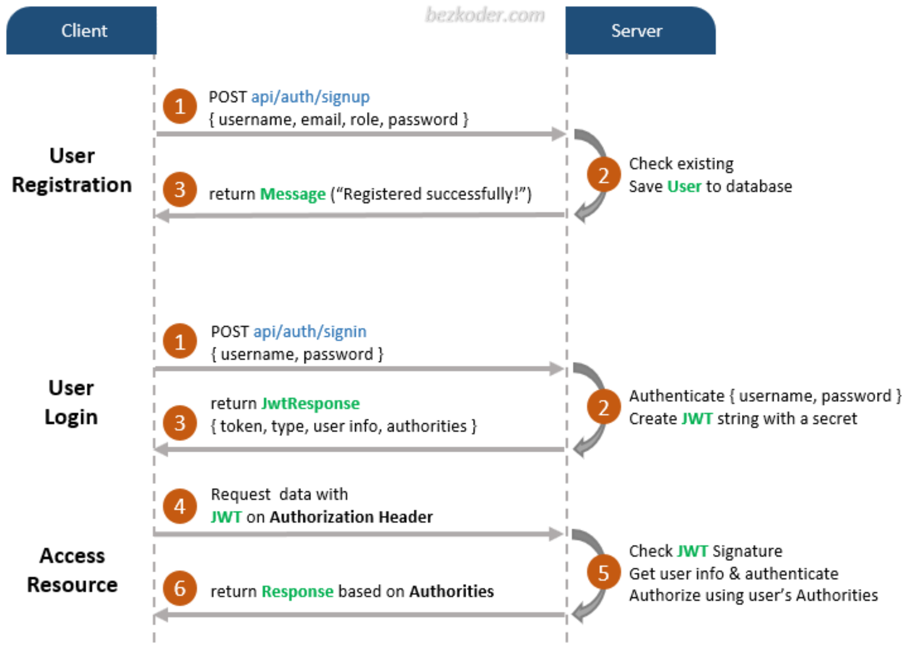

Client
bezkoder.com
Server
User Registration
1 POST api/auth/signup { username, email, role, password }
2 Check existing Save User to database
3 return Message ("Registered successfully!")
User Login
1 POST api/auth/signin { username, password }
2 Authenticate { username, password } Create JWT string with a secret
3 return JwtResponse { token, type, user info, authorities }
Access Resource
4 Request data with JWT on Authorization Header
5 Check JWT Signature Get user info & authenticate Authorize using user's Authorities
6 return Response based on Authorities

Figure 3.9: JWT authentication procedure flowchart
Source: [43]

JWT Token Refresh

JWT tokens are set to expire in order to increase the overall security of the application and prevent unauthorized access. If a certain JWT token is stolen or intercepted during the communication, an attacker can use it to gain unauthorized access to the protected resources.

By setting an expiration time for the tokens, the server is able to ensure that even if a certain token is to be stolen, it will only be valid for as long as the token is not expired, so for a limited period of time. This approach limits the window of opportunity for the attacker to use the token which was stolen.

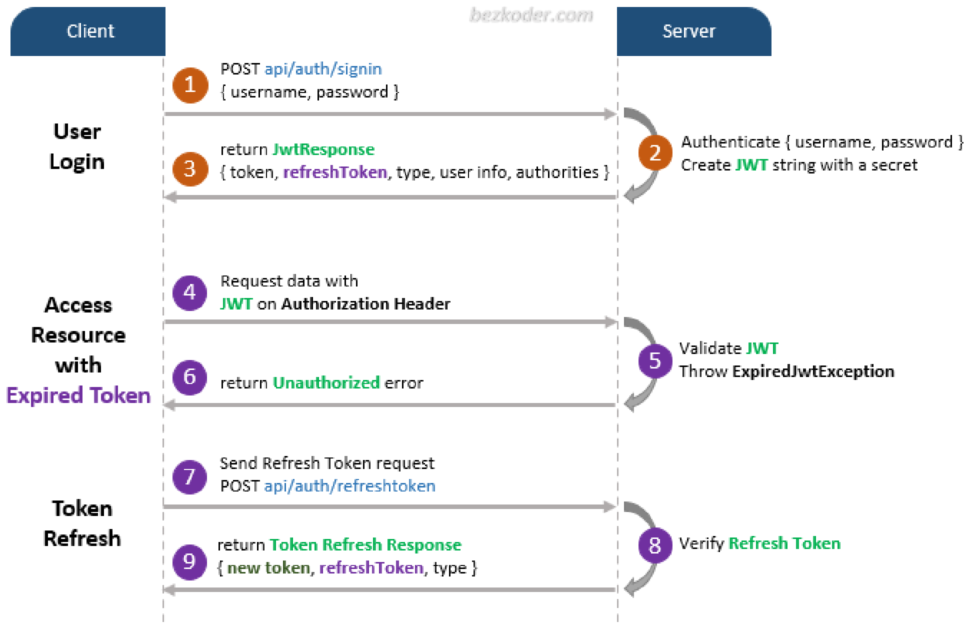
To avoid requesting the user to re-login every time the token expires, which can be a source of frustration, a JWT token refresh strategy can be implemented. JWT token refreshes work by issuing a new JWT to the client after the originally generated one has expired. The client, then, can use this newly generated token to continue sending authenticated requests to the protected resources of the server.

The strategy is implemented as follows:

1. The client sends a login request with their credentials to the server.
2. The server authenticates the client by validating its credentials and if the validation was successful, it creates a JWT and sends it back to the client.
3. The client receives the generated JWT and stores it locally (in a browser cookie or the local storage).

4. After some time has passed, the client sends an HTTP request to the server with the originally generated access JWT.
5. The server receives the request and validates the access token. The server checks the access token expiration time and if the token is found to be expired, the request will be rejected with the appropriate error message.
6. If the client receives a response that indicates that its token is expired.
7. A new request will be sent to the server. This request will include the refresh token that was generated by the server together with the original token. This is the token refresh request.
8. The server will receive the token refresh request and will validate the refresh token, through its JWT signature, and will verify that the token is not expired. If the check is successful, a new access JWT and a new refresh token are generated and sent back to the client.
9. The client receives both the new access token and the new refresh token and stores them locally (in browser cookies or the local storage). The client can now use the newly generated access token to make authenticated requests and access protected resources.

Figure 3.10 illustrates the token refresh procedure described above including both the login and token refresh procedure.



bezkoder.com
Client
Server
User Login
1 POST api/auth/signin { username, password }
2 Authenticate { username, password } Create JWT string with a secret
3 return JwtResponse { token, refreshToken, type, user info, authorities }
Access Resource with Expired Token
4 Request data with JWT on Authorization Header
5 Validate JWT Throw ExpiredJwtException
6 return Unauthorized error
Token Refresh
7 Send Refresh Token request POST api/auth/refreshtoken
8 Verify Refresh Token
9 return Token Refresh Response { new token, refreshToken, type }

Figure 3.10: JWT token refresh procedure flowchart
Source: [44]

3.7.2 Authorization & Access Control

Authorization is the process of determining whether a user has the necessary rights to perform a certain action or to access a certain resource. The authorization process is somewhat related to the authentication process since it is a step of the authentication process, where a user's identity is verified and it is determined which actions or application resources they are actually allowed to access.

Role-based authorization (RBA) is the procedure of controlling user access to application resources or actions based on their role. In the role-based authorization technique, roles are assigned to the users and each role is associated with a set of permissions that determine what users with that role are allowed to access.

On the other hand, access control is the process of determining who is allowed to access a specific resource or perform a specific action. It is a collection of rules and mechanisms that enforce the authorization process decisions.

3.8 Deployment

In this section, the web application deployment procedure on Amazon Web Services (AWS) will be described and the proposed cloud architecture will be detailed. All the relevant Amazon cloud services will also be listed and their functionalities will be described.

3.8.1 Deployment strategy

The web application, together with its infrastructural components, is deployed using Amazon Web Services (AWS). The Frontend is deployed via a public AWS Simple Storage Service (S3) Bucket and the backend is deployed using AWS Elastic Container Service (ECS), a highly scalable, fully managed container orchestration service. The PostgreSQL operational database is deployed via AWS Relational Database Service (RDS), a fully managed, distributed relational database service. Sensor data on the other hand is stored on AWS Timestream, a fast, scalable, and serverless time-series database. Figure 3.11 shows the proposed infrastructure design that can host an ECS cluster, a PostgreSQL relational database, and a web application that is served to clients via a public S3 bucket.

The application can also be deployed locally by containerizing its main components in different containers using Docker. If an internet connection is not possible, local AWS service emulation can also be done using LocalStack, a cloud service emulator.

The entire infrastructure is designed to follow the AWS guidelines, including all the canonical security systems (Security Group, ciphering of data and traffic, etc.) therefore they are to be considered as included in the architecture even though not explicitly specified. For example, when speaking of an S3 bucket we are talking about encrypted buckets that have the right access policies.

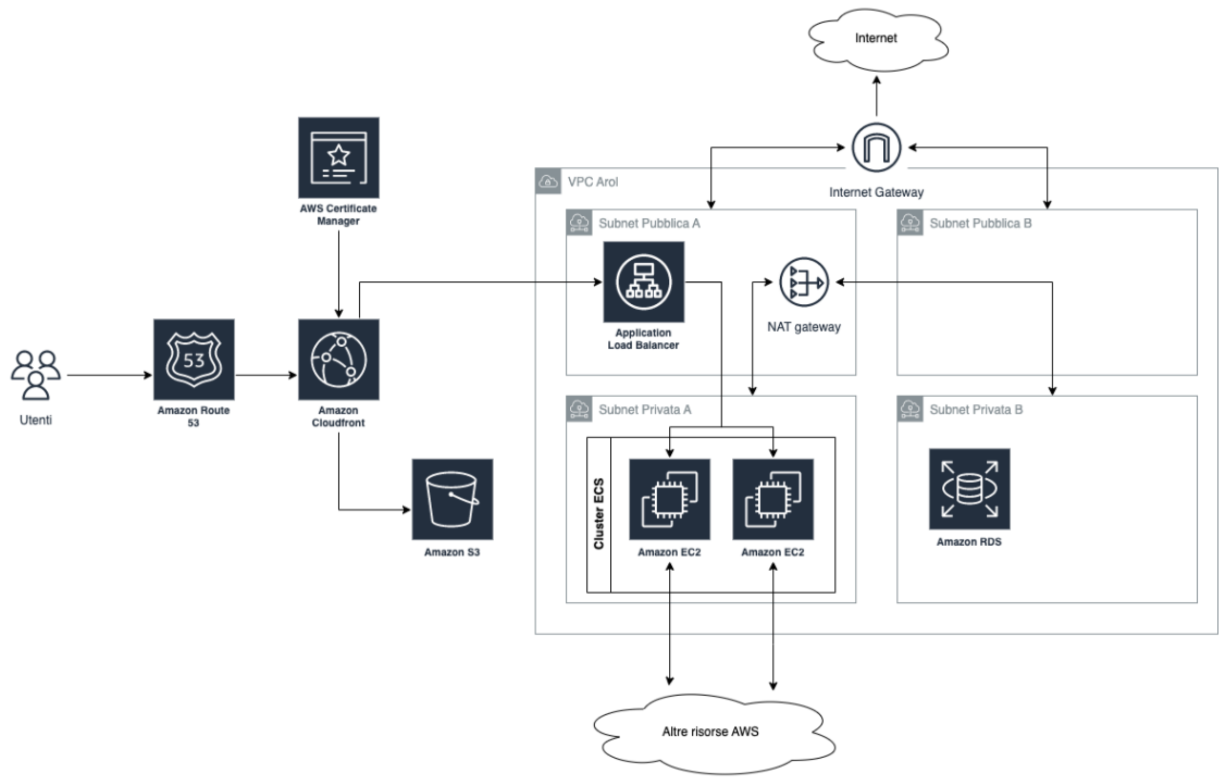


Figure 3.11: Proposed AWS cloud infrastructure to deploy the web application

Description of the proposed cloud architecture

The proposed architecture calls for the construction of all the necessary resources, designed with a consideration to high availability and high scalability. A dedicated VPC network to host all the necessary cloud resources. In order to host the developed Web Application, a combination of the CloudFront (CDN) and S3 construct will be used, while all the backend resources will be deployed containers based on an ECS cluster. Inside the VPC there will also be the relational database of type PostgreSQL.

VPC

To host all the cloud resources, the proposed VPC will include 2 public subnets and 2 private subnets, both instantiated in two different availability zones. The VPC will also contain 1 Internet Gateway for internet access, 1 NAT Gateway positioned in one of the two private subnets to provide internet access to the resources instantiated in the private subnets, a Routing table configured for traffic routing, an Application Load Balancer for routing HTTP/S traffic to ECS cluster services and a Hosted Zone on Route 53 for DNS management.

Client tier deployment

For hosting the Web Application, different resources are expected to be involved. DNS resolution will be enabled by Route 53 while the CloudFront CDN system will serve and cache the web application resources requested via HTTP/S. A Certificate Manager will be used to issue a valid SSL certificate for the website. Finally, a public S3 bucket will be used for storing all the static resources that make up the developed web application.

Server tier deployment

The backend will be built from one or more Docker containers that will be served via an ECS cluster. The ECS cluster will be based on EC2 nodes and an AutoScaling Group will be implemented to provision EC2 machines to join the ECS cluster. A Service will also be created to make the backend available to the Application Load Balancer, while DNS registration will be created to route the traffic to the service.

Data tier deployment

A relational database on AWS RDS with a PostgreSQL engine will be created for this infrastructure. The database will be created within the private subnets; therefore, it will only be reachable from within the VPC (e.g., from the backend).

3.8.2 AWS Services

In this section, the different AWS services used in the cloud architecture described above are detailed. Table 3.5 also shows the function of each cloud service.

AWS Service	**Function**
Amazon Route 53	DNS service
Amazon CloudFront	Content Delivery Network
Amazon Simple Storage Service (S3)	Object storage
Amazon Certificate Manager	Provision, manage, and deploy public and private SSL/TLS certificates
Application Elastic Load Balancer (ELB)	Automatic distribution of traffic across multiple targets, such as EC2 instances
Virtual Private Cloud (VPC)	Provides a virtual private cloud, by provisioning a logically isolated section of Amazon Web Services Cloud
Internet Gateway	Enables resources in public subnets to connect to the internet
NAT Gateway	Enables instances in a private subnet to connect to services outside their VPC and blocks external services from connecting with the private instances
Amazon Elastic Cloud Compute (EC2)	Allows to rent virtual computers on which to run their custom computer applications
Amazon Elastic Container Service (ECS)	Orchestration service that allows to deploy, manage, and scale containerized applications
Amazon Relational Database Service	Allows to set up, operate and scale a relational database for use in applications

Table 3.5: Proposed AWS services and their function

Amazon Route 53

Amazon Route 53 is a highly available and scalable cloud Domain Name System (DNS) service offered by Amazon Web Services (AWS). It is designed to be an extremely reliable and cost-effective way to route end users to web applications by translating domain names into IP addresses.

Route 53 allows for registration of domain names, DNS resolution and traffic routing to web resources, such as web servers or Amazon S3 buckets, health monitoring of web resources and automatic traffic routing to healthy resources, traffic flow management and latency-based routing to the resources that provide the best performance for a given user.

Amazon CloudFront

Amazon CloudFront is a content delivery network (CDN) service provided by Amazon Web Services (AWS). It is designed to distribute content, such as web pages, to users around the world. It works by storing copies of the needed files at multiple locations, called "edge locations", around the world. When a user requests a file, CloudFront automatically routes the request to the edge location that is closest to the user to minimize latency. This operation is also known as "caching" and is one of the main mechanisms that CloudFront uses to speed up the delivery of content around the world

Amazon Elastic Cloud Compute (EC2)

Amazon Elastic Compute Cloud (EC2) is a service provided by Amazon Web Services (AWS). It is used to launch and run virtual machines (VMs), called instances, in the cloud.

EC2 instances are of different types, each optimized for specific workloads and configurations. Instance types include general-purpose instances, compute-optimized instances, memory-optimized instances, storage-optimized instances, GPU instances, and low-cost ARM instances. This architecture will leverage general-purpose EC2 instance types.

Amazon Elastic Container Service (ECS)

Amazon Elastic Container Service (ECS) is a container orchestration service provided by Amazon Web Services (AWS). It allows running and scaling containerized applications on a cluster of EC2 instances.

An Amazon ECS cluster is a logical grouping of container instances. Each cluster is made up of one or more EC2 instances that have the ECS agent present. The ECS agent is a software component that communicates with the ECS service to run and manage containers on the instances.

Amazon Certificate Manager (ACM)

Amazon Certificate Manager (ACM) is a service provided by Amazon Web Services (AWS) that allows deploying and managing Secure Sockets Layer/Transport Layer Security (SSL/TLS) certificates. It provides a convenient way to create, import, and manage certificates for use with other AWS web services. ACM also provides a private certificate authority (CA) service to manage private certificates.

4 The web application

In this section, the proposed web application is presented and the different technical aspects and challenges are discussed in detail. The implemented application features and main pages are also thoroughly described and demoed through the use of multiple screenshots.

4.1 Overview

The proposed web application is mainly intended to be used by AROL customers in order to monitor and analyze their machinery fleet operation. Customer machineries are organized into clusters, where each cluster represents a production plant. The idea behind this project was to create a proof-of-concept application that can be used by AROL customers to monitor the operational state and the sensor data of each machinery. As certain machinery can be equipped with more than 150 different sensors, all collecting data in parallel, a dynamic dashboard solution is proposed. The dashboard can be populated by users with a large selection of different widgets, each monitoring single or multiple sensors in parallel. Each widget can show not only the real-time operational state of the selected sensors but can also be used to visualize historical sensor data.

As the application is intended to be used by the production plant staff to provide real-time monitoring of the machinery operation, a shared document storage feature was also implemented. The document storage can be used to store and view machinery manuals or custom documents uploaded by the staff itself.

AROL customers are usually companies with a relevant number of employees so the application features a role-based access control system. The roles hierarchy is: Administrator, Manager, and Worker. The permissions system regulates access to both machinery dashboards and documents. If a certain user has no access to any of the aforementioned features, the user is considered to have no access at all to that machinery. Access to the machinery dashboards and documents is designed to include three levels of authority: Read, Modify, and Write access – all hierarchically dependent on each other. For example, if a user has no Read permission over machinery documents, it can have neither Modify nor Write access to such a resource.

The Administrator role is granted full access to all the resources of the system, including managing machineries, managing company user accounts as well as assigning the proper permissions to each of them. The Administrator is the only role that can create user accounts of any role, including other administrator accounts.

The Manager role is envisioned as a production plant manager. This role has access to the machineries granted by the Administrator. The Manager can create other Worker or Manager accounts but is limited to assigning permissions only for the machines to which he has access and to the highest permission authority that is granted to him/her.

The Worker role is the lowest level of authority available. This role has no access to any feature of the user and permissions management system. The Worker can only access machinery dashboards and/or documents to which he is granted at least Read access.

Both the Manager and the Worker roles are limited to access only machinery resources (dashboards and/or documents) granted to them. For example, a Worker that has no Read permissions on a certain resource he/she is prohibited from accessing it.

4.2 UX & UI Design

User Interface and User Experience design are two critical components of any successful digital product. A well-designed user interface can make a product more visually appealing and easier to use, while a positive user experience can encourage users to engage with the product more frequently and for longer periods of time [45].

User Interface design refers to the visual design and layout of the user interface, including elements such as color, typography, and layout. The goal of UI design is to create an interface that is visually appealing, intuitive, and easy to use.

User Experience design, on the other hand, refers to the overall experience that a user has while interacting with a digital product. This includes everything from the ease of use of the product to the emotional response that it elicits. The goal of UX design is to create a product that is user-friendly, engaging, and meets the needs of the user.

Both the User Interface and the User Experience design are also important in industrial applications, including those that rely on Industrial Internet of Things technologies. In these applications, UI and UX design is critical to ensuring that users can easily access and make sense of the vast amounts of data generated by connected devices. In an industrial setting, the User Interface design must be intuitive and easy to use, as it will be used by staff workers and technicians who may not be technology experts. The design must also be tailored to the specific needs of the industrial application and the workers who use it. The user experience too should be designed to minimize the cognitive load of users, providing them with clear, concise information and easy-to-use tools. This can help to reduce the risk of errors and improve productivity.

A qualitative User Interface and User Experience design can potentially help improve safety by providing clear and concise information to its users. A well-designed interface allows users to quickly identify potential hazards or problems with different industrial equipment, allowing them to take corrective action before an accident occurs.

4.2.1 Layout and Design patterns

The main goal of the design philosophy behind the implemented web application is to provide a simple, consistent, and robust interface to the users. Even though the primary goal of the application is an industrial one, meaning that the users prioritize getting the job done over flashy graphics, the interface should be modern and comfortable to use.

Layout

One of the most important components in designing an easy-to-use and functional application is its layout. The purpose of the layout is to arrange the content of the application in such a way that the user experience is enhanced.

The implemented layout is comprised of three main components:

- The sidebar: acts as the primary mean of navigation and contains links to all the resources of the application. Special care has been taken to remove the need of using dropdown menus. This greatly simplifies the interaction of the user with the navigation system and makes the application usable on different industrial devices, such as touchscreen ones. The sidebar continuously highlights its selected state to aid users to know where they are in the application. In order to maximize the area available to the app viewport the sidebar can be expanded and collapsed

- The navigation bar: acts as the secondary mean of navigation and contains helpful items for the user.
- The app viewport: this is where the user can interact with the application and the selected pages are displayed. The content inside the app viewport is organized in cards that act as floating islands. This strategy visually groups the page content based on its function inside the resource context. The app viewport features the corresponding title and an optional description for the displayed page. As the user navigates deeper into the page, breadcrumbs will be shown to aid users to know their exact location in the hierarchical structure of the resource.

The position of both the sidebar and the navigation bar is fixed and consistent throughout the user interaction with the application. They also both feature a home link, providing users with a safe start location on the application.

Design Patterns

Design patterns, on the other hand, are reusable solutions to common problems that occur in software design [46]. Design patterns are generally recognized as established best practices and provide a proven solution to a recurring design problem.

- Drag & Drop uploads: allow users to perform operations on one or more objects by moving them from one place to another.
- Modal windows: enable users to take an action or cancel the overlay and can continue interacting with the original page.
- Notifications: informing users about important updates and messages.
- Module tabs: separating content into sections and accessing it via a single content area using a flat navigation structure that does not refresh the page when selected.
- Carousel: allow users to browse through a set of items and possibly select one of them
- Alternating table row colors: visually separating similar-looking rows in a table apart, for the user to quickly distinguish the values of each row.

4.3 Application features

Now, we move to list and describe in detail all the main features implemented during the development of the proposed web application. Every implemented feature is associated with one or multiple images that visually depict the details of the implementation. For each feature, the corresponding context is also described in order to express the reasoning behind the implementation choices and aid in understanding why the target users may be interested in using them.

4.3.1 Login page

The implemented web application enforces a mandatory authentication procedure in order for users to be able to access it. This authentication is necessary since users can only view the machineries, and their related resources, of only the company to which they belong. The login page is the first page that is loaded when the application is started. As shown in Figure 4.1, users need to authenticate using their email and password combination. The email and password strategy works well to authenticate users and limit access only to their company resources, but the problem is that users need to enter the aforementioned credentials every time they access the application. To tackle this shortcoming and limit the number of times users need to authenticate when accessing the application, a “Remember me” functionality has been implemented. If such an option is selected during the login procedure, the user will not need to re-authenticate when he/she reopens the application at a later moment.

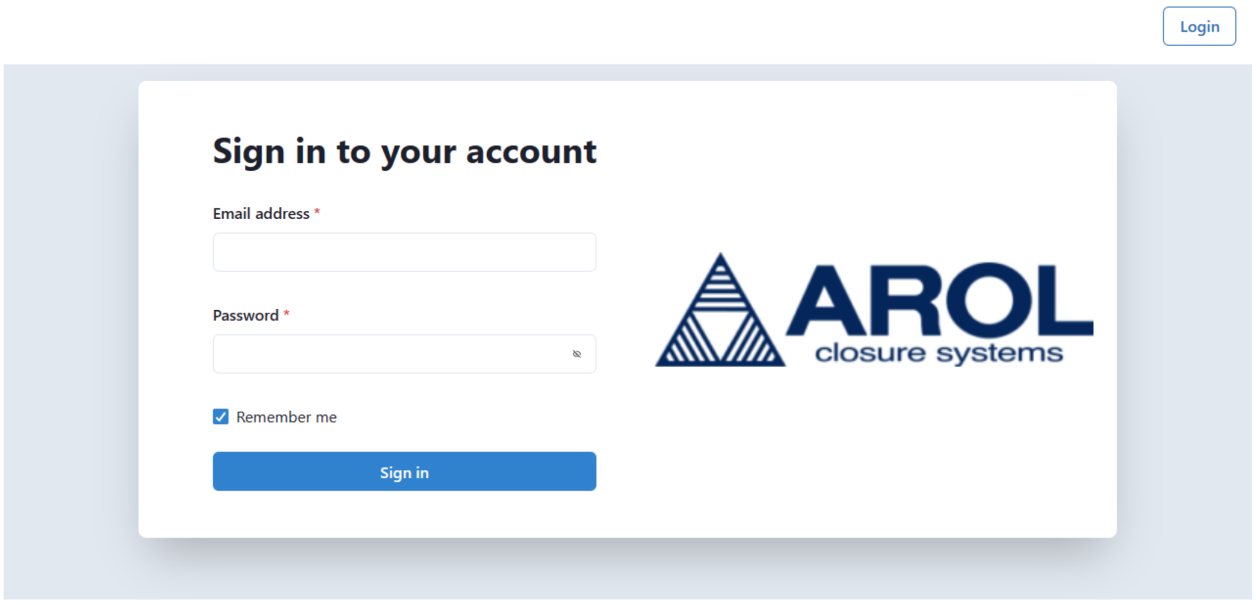


Figure 4.1: Login page - user authentication

4.3.2 Home page

After the user has completed the authentication procedure, they are redirected to the home page of their company. As shown in Figure 4.2, the home page main goal is to show some quick information about the company's machineries. Company machineries are presented through a map component, grouped by the production plant where those are installed, and through a carousel component which provides additional details on each machinery, like the machinery model, its type, and the corresponding number of heads. The machineries shown by the carousel are presented in an automatic slideshow format. By clicking the machinery in the carousel, through a shortcut, users can access the corresponding machinery landing page.

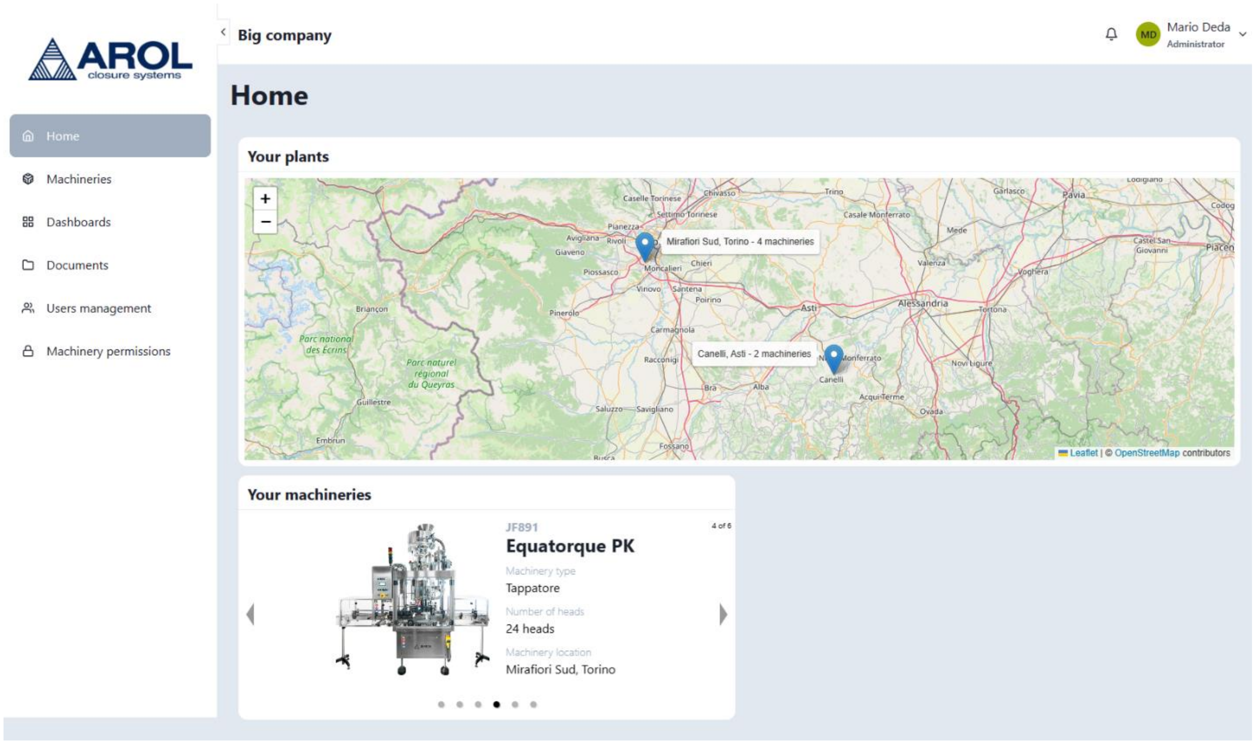


Figure 4.2: Company home page

4.3.3 Machinery browsing

This page of the application was created with the main purpose of aiding users to quickly locate and access any machinery resource they need. As illustrated in Figure 4.3, a hierarchical navigation system was implemented with the intention of reducing the amount of time a user spends searching the required machinery. At each step of the navigation the list of machineries is restricted based on the previous input.

Machineries are clustered by the corresponding production plant at which they were installed. This strategy was deemed as preferable for customers which own a large number of machineries at different locations, but in order to avoid unnecessary navigation hurdles for a small customer which may have just a single production plant, an automatic shortcut is implemented – if the customer company has just a single production plant, the application will automatically navigate to the machineries of that production plant.

This production plants panel also provides quick access to the permissions system through a button that is actionable only by users with an Administrator and/or Manager role.

After a production plant is chosen, the user is presented with a scrollable list of all the machineries installed in that production plant, as shown in Figure 4.4. Each entry in this list provides some quick information that is useful to the user to be able to correctly choose the required machinery.

By clicking the required machinery entry, the user is presented with all the selected machinery details and he/she can find two buttons that navigate them to the selected machinery dashboard or documents, as shown in Figure 4.5. Both the aforementioned buttons are usable only if the user has at least Read access to the denoted resource.

An interactive map functionality was also implemented to aid users to visualize where their machineries are located. The same navigation procedure, from a production plant to a single machinery, can easily be executed using the map by clicking the corresponding map markers.

Both of the described navigation systems are bidirectional, meaning that a change in either of them is reflected in the other and vice-versa.

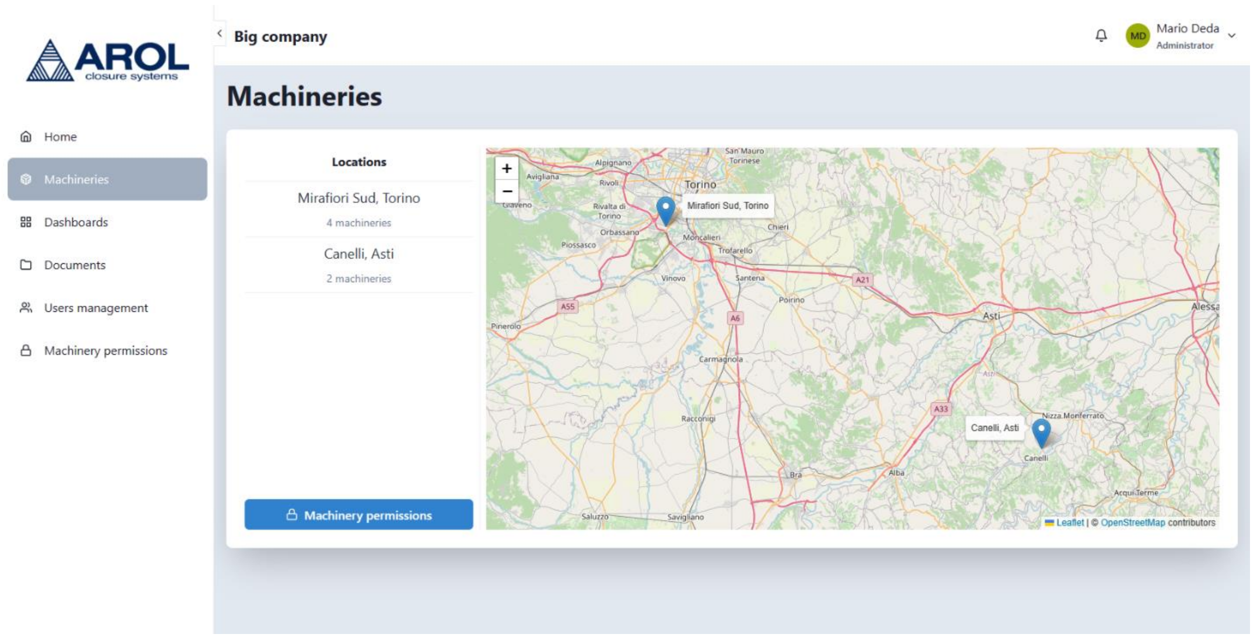


Figure 4.3: Machinery browsing - production plants view

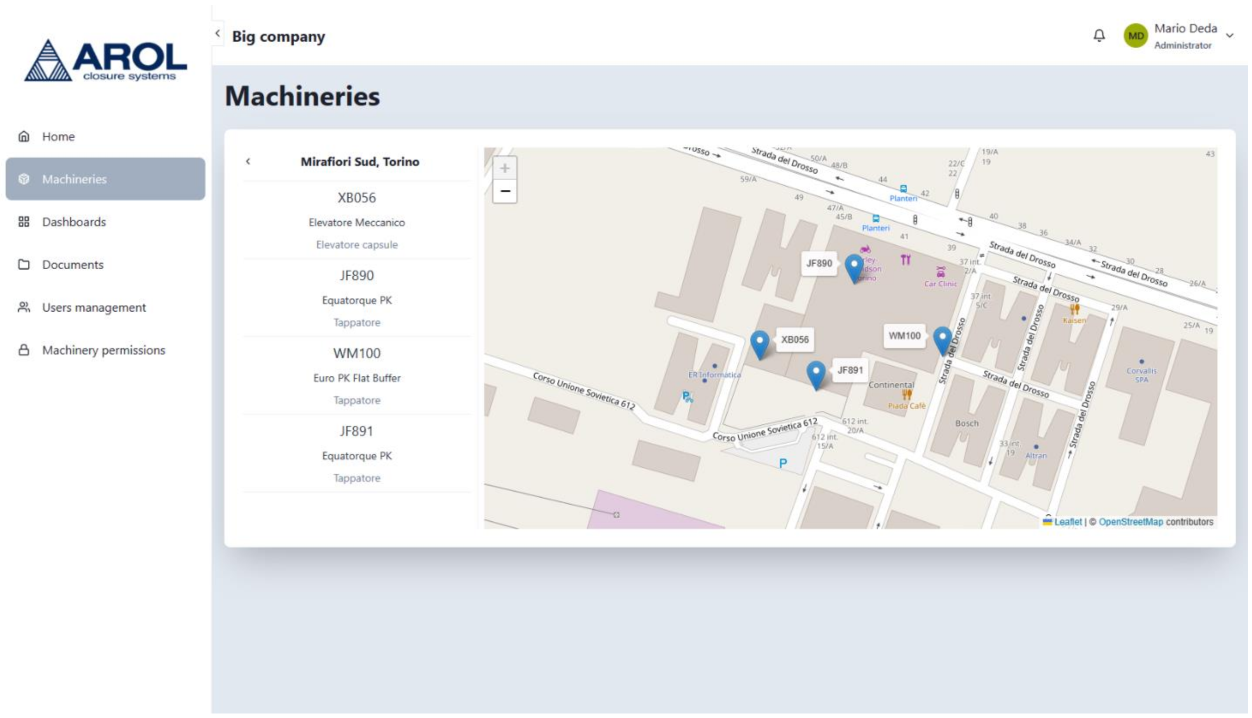


Figure 4.4: Machinery browsing - production plant machineries

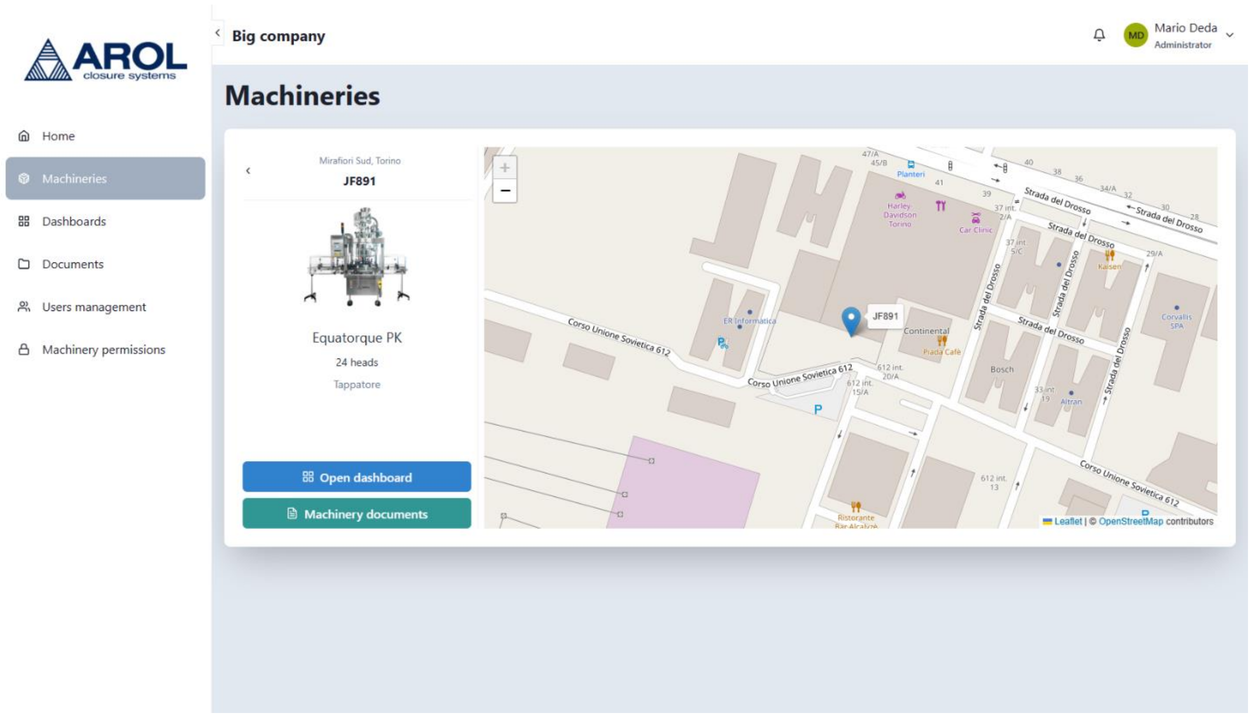


Figure 4.5: Machinery browsing - machinery details

4.3.4 Machinery landing page

As shown in Figure 4.6, this page provides the user a shortcut to navigate from the machinery dashboards to its documents or vice-versa, provided that the user has at least Read access to one of the resources. This page is accessible by using the Breadcrumbs link navigation when the user is on the machinery dashboards or documents page.

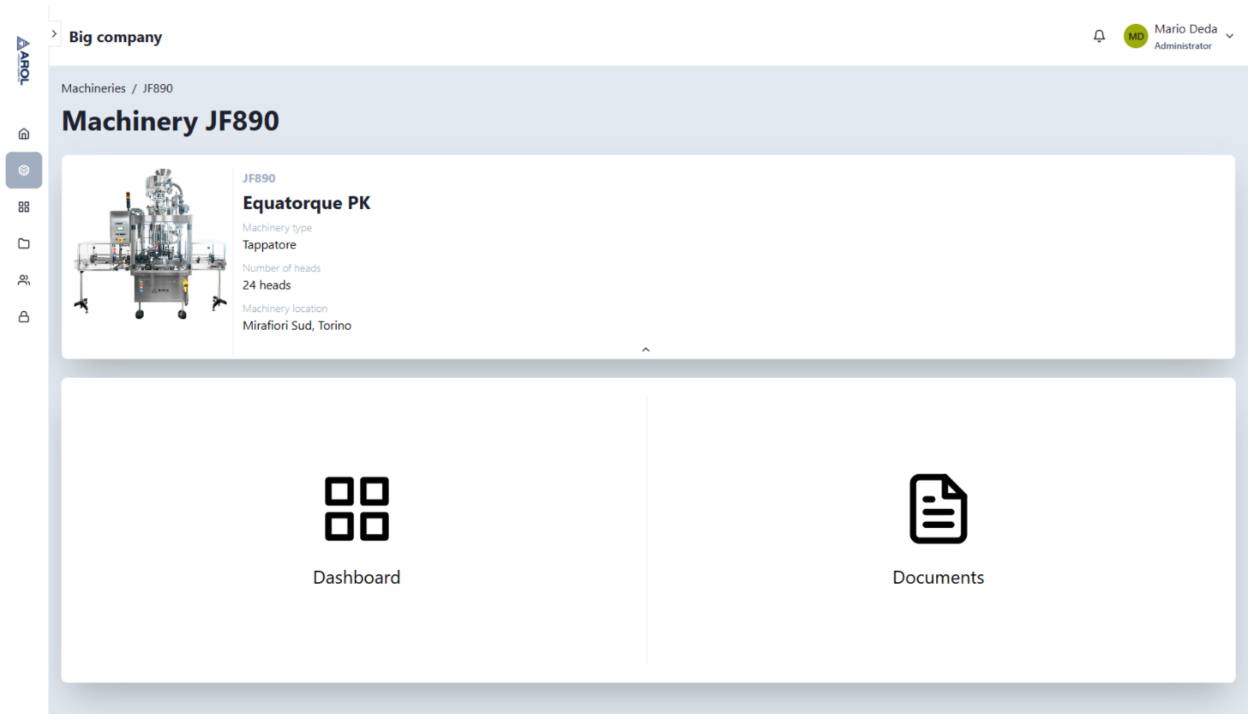
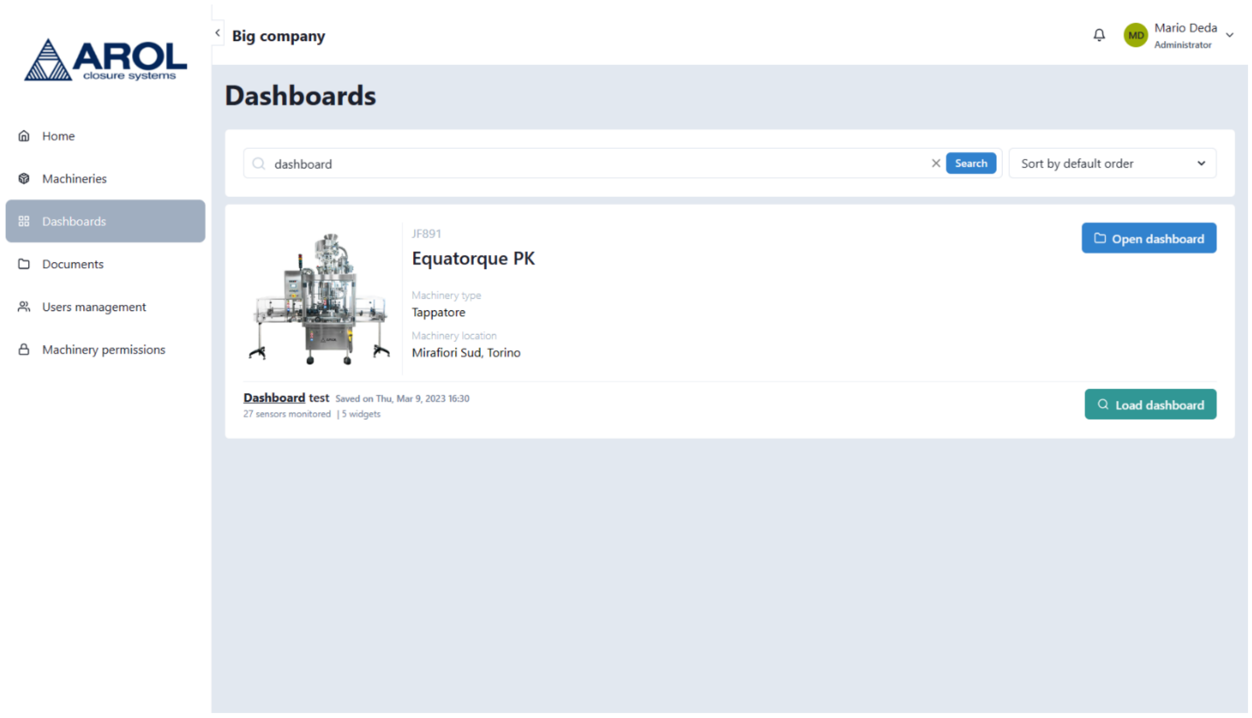


Figure 4.6: Machinery landing page

4.3.5 Machinery dashboards quick-browse

This page, as illustrated in Figure 4.7, provides a quick way to access all the saved dashboards of the machineries to which the user has at least dashboard Read access. Search and sorting functionality is also available, providing the user shortcuts to quickly find the required entry.



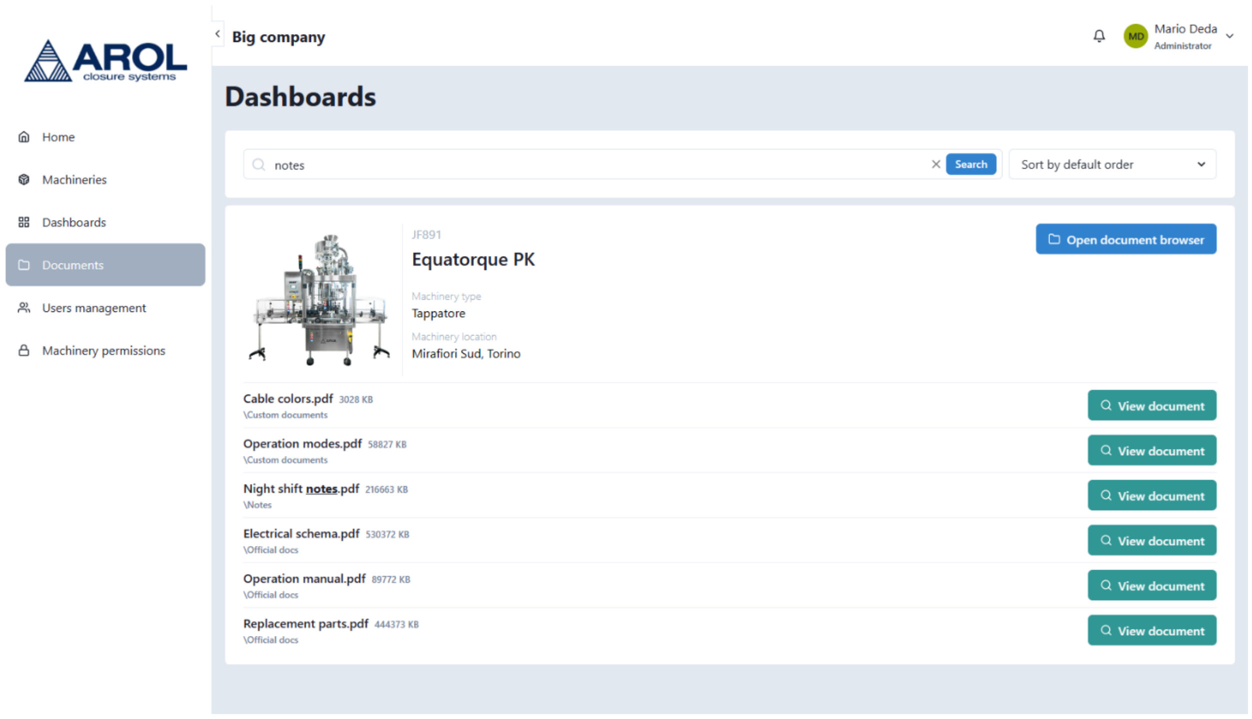
JF891
Equatorque PK
Machinery type
Tappatore
Machinery location
Mirafiori Sud, Torino

Dashboard test Saved on Thu, Mar 9, 2023 16:30
27 sensors monitored | 5 widgets

Figure 4.7: Machinery dashboards quick-browse

4.3.6 Machinery documents quick-browse

As shown in Figure 4.8, this page is similar to the “Machinery dashboards quick-browse” page described above, with the difference being that instead of dashboards, the user can quickly navigate any document that has been uploaded in the shared document storage of each machinery.



JF891
Equatorque PK
Machinery type
Tappatore
Machinery location
Mirafiori Sud, Torino

Cable colors.pdf 3028 KB
\Custom documents
Operation modes.pdf 58827 KB
\Custom documents
Night shift notes.pdf 216663 KB
\Notes
Electrical schema.pdf 530372 KB
\Official docs
Operation manual.pdf 89772 KB
\Official docs
Replacement parts.pdf 444373 KB
\Official docs

Figure 4.8: Machinery documents quick-browse

4.3.7 Machinery documents

This page allows the user to interact with the shared document storage of the selected machinery. This interaction is offered through an integrated file-browsing component. The document storage system was created with the main goal of trying to emulate a traditional filesystem and allow users to store and view PDF files.

Documents are organized in folders, as illustrated in Figure 4.9 and Figure 4.10. Each machinery has a root folder, named after its system-wide unique ID. Table 4.1 lists all the operations that are permitted on folders, together with the corresponding permissions needed to complete the operation. Table 4.2 on the other hand shows all the operations permitted on documents and the permissions needed to perform the action.

As shown in Figure 4.11, an integrated document viewer is also included to view any uploaded document. The document viewer provides some basic functionalities such as navigating the document page by page, zooming in or out the document (from 50% to 125% zoom), rotating the document, and downloading the document

The shared document storage system offers protection from overwrite-on-upload for duplicated filenames and takes special care to notify the user about any irreversible actions that they might be executing, like deleting a document or a folder.

Folder operation	**Permissions needed**
Create a new folder	Write access
Delete an existing folder and all of its content	Write access
Rename an existing folder	At least Modify access
Browse folder contents	At least Read access

Table 4.1: Machinery documents - folder operations and needed permissions

Document operation	**Permissions needed**
Upload one or many new documents simultaneously	Write access
Delete an existing document	Write access
Rename an existing document	At least Modify access
View an existing document	At least Read access
Download an existing document	At least Read access

Table 4.2: Machinery documents - document operations and needed permissions

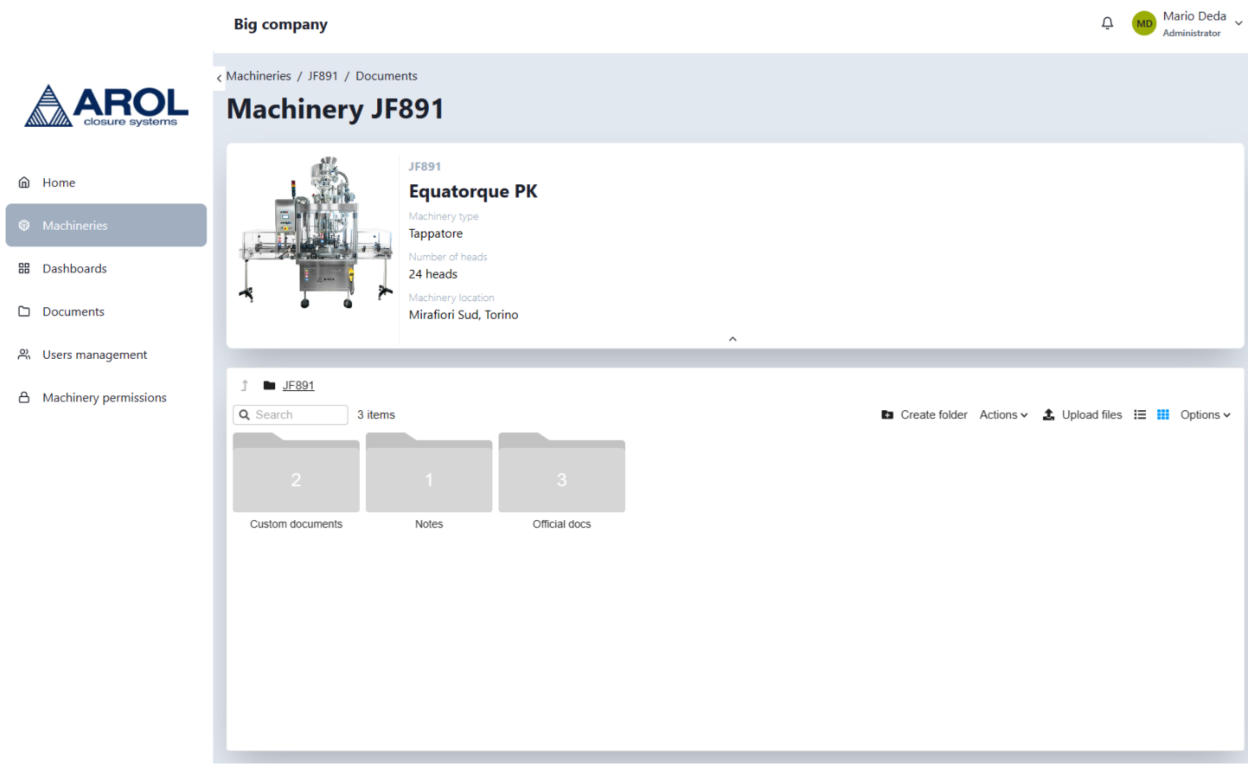
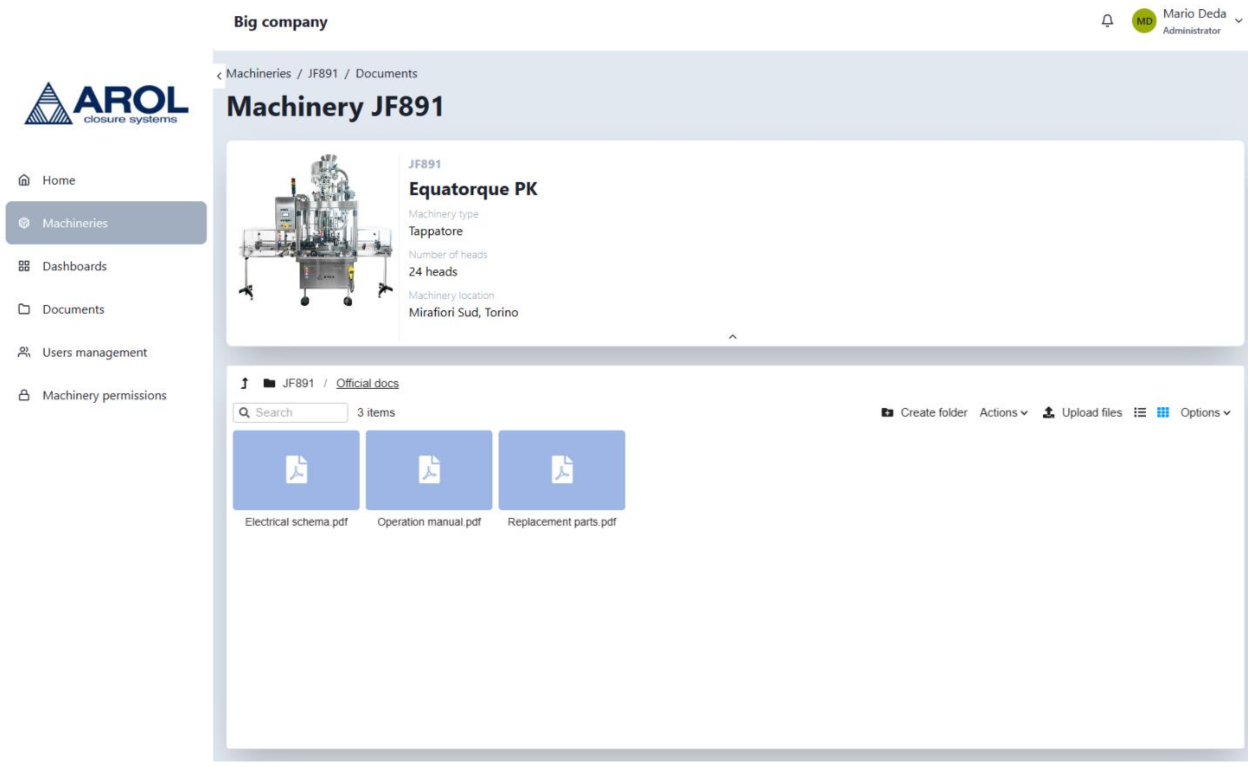


Figure 4.9: Machinery documents - folders view



Machinery JF891

JF891

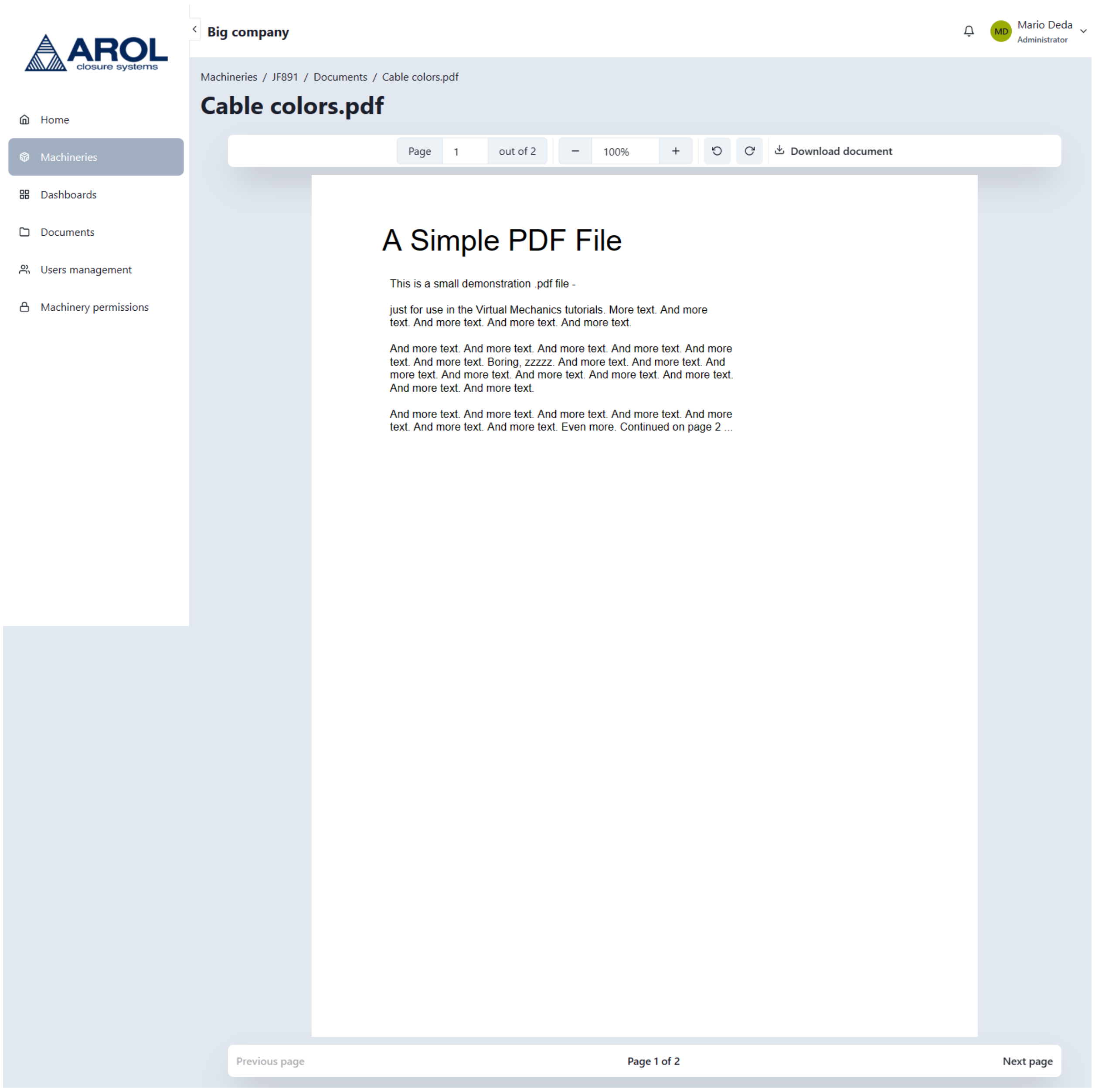
Equatorque PK

Machinery type
Tappatore

Number of heads
24 heads

Machinery location
Mirafiori Sud, Torino

Figure 4.10: Machinery documents - files view



A Simple PDF File

This is a small demonstration .pdf file -

just for use in the Virtual Mechanics tutorials. More text. And more text. And more text. And more text. And more text.

And more text. And more text. And more text. And more text. And more text. And more text. Boring, zzzzz. And more text. And more text. And more text. And more text. And more text. And more text. And more text. And more text. And more text.

And more text. And more text. And more text. And more text. And more text. And more text. And more text. Even more. Continued on page 2 ...

Figure 4.11: Machinery documents - embedded document viewer

4.3.8 Machinery dashboards

As shown in Figure 4.12, Figure 4.13, and Figure 4.14, this page features the dynamic dashboard system which can be used to perform real-time machinery monitoring and view historical machinery data. The dynamic dashboard system's main objective is to provide the user the possibility to build custom dashboards, with the needed widgets and monitor the desired machinery sensors. It provides the possibility to drag and drop widgets from the left panel, reposition and resize them and configure the sensors to monitor.

In order to provide users with the maximum freedom in creating dashboards that cater to their needs, the dashboard system provides a collision management system that allows widgets to be quickly dragged around by automatically reformatting the layout of the dashboard to prevent the widgets overlapping each other. Furthermore, it can recompact the dashboard widgets both vertically and horizontally to remove any unused free space between the widgets. Dashboard compaction can be configured using the appropriate menu present in the dashboard control panel, as shown in Figure 4.15.

The dashboard dimension can be also extended vertically to allow fitting more widgets. The number of widgets that the dashboard can host and its dimensions can be extended virtually infinitely.

As shown in Figure 4.16, in the left panel, there is a large selection of widgets that the user can choose from, each of them providing a unique representation of the sensor data and a different number of sensors that can be monitored contemporarily. Table 4.3 lists all the available widgets present in the dashboard system, together with the monitoring capability that each one of them offers.

Widget name	**Widget category**	**Number of sensors that can be monitored**
Line chart	Chart (multi-value) widget	As many sensors as machinery heads
Bar chart	Chart (multi-value) widget	As many sensors as machinery heads
Area chart	Chart (multi-value) widget	As many sensors as machinery heads
Pie chart	Chart (multi-value) widget	At most 5
Current value	Single-value widget	At most 1 (latest value shown)
Thermostat gauge	Single-value widget	At most 1 (latest value shown)
Tachometer gauge	Single-value widget	At most 1 (latest value shown)

Table 4.3: Machinery dashboard - available widgets

All widgets, including those that allow multiple sensors to be monitored in parallel and those that allow only a single sensor to be monitored, can be extended to use aggregation functions. Aggregation functions are helper functions that can be used to quickly show the minimum, maximum or the average of a group of samples. When aggregation functions are applied to widgets that allow sensors to be monitored in parallel, their effect is applied over the samples at each time sample time, while for widgets that can monitor at most one sensor at a time, their effect is applied over a configurable number of past samples. This strategy allows aggregation functions to effectively summarize the progress in time of the sensor state. Figure 4.17 shows an aggregation function applied over a chart widget, which has been configured to monitor multiple sensors in parallel.

In order to populate the widgets with the desired sensors a custom panel has been implemented, as shown in Figure 4.18. From this panel, users can select the sensors that they want to monitor, select any data aggregation functions, and configure the number of samples that they want to load for widgets that allow parallel monitoring of sensors. For widgets that allow monitoring multiple sensors in parallel, like the chart widgets, a multi-selector shortcut is enabled. This multi-selector allows users to quickly select all the identical sensors that are mounted in different capping heads. As their name suggests, single-value widgets allow to monitor at most one sensor, pie chart widgets allow to monitor at most five sensors in parallel while chart widgets allow to monitor as many sensors in parallel as there are capping heads in the selected machinery.

After the widget has been populated, its heading panel is enabled and displayed. This panel shows a configurable widget title and a list of shortcuts to different widget functionalities like the widget chart, a button to refresh the chart, a sensor history viewer for single value widgets, and the widget menu. The widget menu, as shown in Figure 4.19, contains links that allow users to add more sensors to monitor, configure the widget settings, delete the widget altogether, and the possibility to lock the widget in place by disabling the possibility to move or resize it. As illustrated in Figure 4.20, the widget settings panel allows users to quickly remove sensors that are being monitored, and change the number of samples that are being displayed for widgets that allow parallel monitoring of sensors. The widget settings panel also allows users to modify any configured aggregations, if any, as shown in Figure 4.21.

Chart widgets are designed with the main goal of allowing users to monitor multiple sensors in parallel and for long chunks of time. As shown in Figure 4.13, since the number of samples being displayed in the widget may be large, a tooltip has been implemented to allow to user to explore in detail the values of each sensor at a given time. Another commodity that can quickly help users decode the information presented by the chart is the chart legend. As illustrated in Figure 4.14, the chart legend lists all the sensors being monitored in the widget together with the corresponding color of each sensor. Another particular feature implemented on chart widgets is the difference between real-time values and “filler” values that are inserted in the chart to ensure its continuity in time, even when a certain sensor might not report new data. As shown in the “Average friction” widget in Figure 4.12, in order to distinguish both situations, a colored round dot has been used to signal the presence of a real-time value. These round dots only appear at sample times.

As discussed above, charts that allow multiple sensors to be monitored in parallel are designed to simplify the monitoring of a large amount of data. If users need to monitor or browse sensor data across a large window of time, the dimensions of the chart can result insufficient to display all the required data in a readable way. As illustrated in Figure 4.22, in order to tackle this problem, full-screen charts have been implemented. As the name suggests, this feature allows users to display a certain chart in a full-screen version, considerably increasing the chart dimensions and allowing much more data to be displayed at once.

The last problem faced by charts and the enormous amount of data that they can contain and handle is the problem of how to quickly navigate in the chart to a desired point in time. This problem has been resolved by implementing a quick navigation panel, as illustrated in Figure 4.23. This panel enables users to quickly scroll all the data sample times, without having to navigate inside the chart and select the desired date to navigate to. For each entry in this panel, the number of data samples present in that point in time is also shown.

Single-value widgets, or widgets that can monitor at most one sensor at a time, on the other hand, face the opposite challenge. They are limited to displaying at most one value at a time. This can result problematic as users may also be interested in viewing also past values or they may be interested to check sensor data trends. In order to resolve this shortcoming of single-value widgets, a sensor data history panel has been implemented. This panel, as shown in Figure 4.24, shows past sensor data, sorted in chronological order. It also shows the difference between each data sample with respect to the previous measurement.

As a certain machinery can have multiple dashboards, each depicting a certain monitoring scenario, a dashboard persistence functionality was implemented. Users can save their dashboards from the dashboard control panel, as shown in Figure 4.25. The user can either quick-save the dashboard or save the dashboard as a new dashboard by filling in the details, as shown in Figure 4.26. If a dashboard is marked as the default one, it will be automatically loaded when users access the dashboard page of the corresponding machinery. Only users with dashboard Write access can save a new dashboard or overwrite an existing one. Users with Modify access can only overwrite any existing dashboard while users with Read access are only permitted to load and view any previously saved dashboard. As illustrated in Figure 4.27, dashboards are managed and loaded from a custom menu, where all the machinery dashboards are listed, together with some statistics like the number of widgets and the number of sensors monitored in that dashboard. Any dashboard modifications need to be saved in order for the changes to be persisted and avoid unintentionally losing any unsaved work. To prevent this situation, any time the user tries to navigate away from the unsaved dashboard, the application will notify the user and ask for confirmation that they actually want to discard the changes.

Since a certain dashboard configuration can be suitable for different machineries of the same model, dashboard templates are implemented. Instead of starting from scratch with a new dashboard layout, the user can load template dashboards that have been previously created for machineries of the same model, as shown in Figure 4.28. When a dashboard template is loaded, a new dashboard is created and the layout of the template is automatically applied to it. The dashboard layout includes an exact copy of the original dashboard widgets, together with their type, position, and name. The template, however, does not include the original dashboard monitored sensors. This was done in order to give the user the possibility to quickly select the necessary sensors to monitor without having to go through the process of removing the originally monitored sensors. For example, a dashboard that was saved for the EQUATORQUE machinery with ID JF891 can be loaded as a template for other machineries of the model EQUATORQUE.

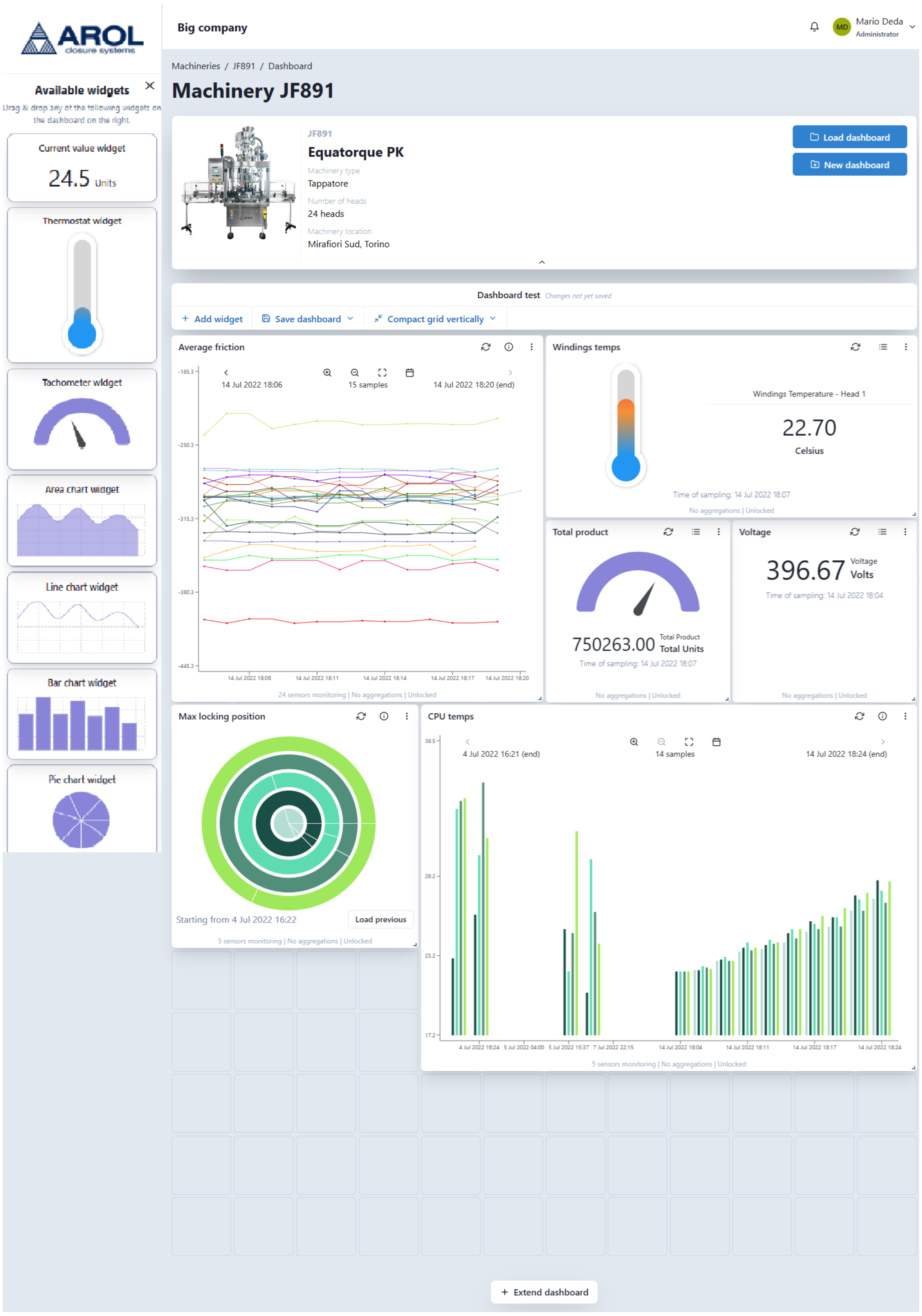
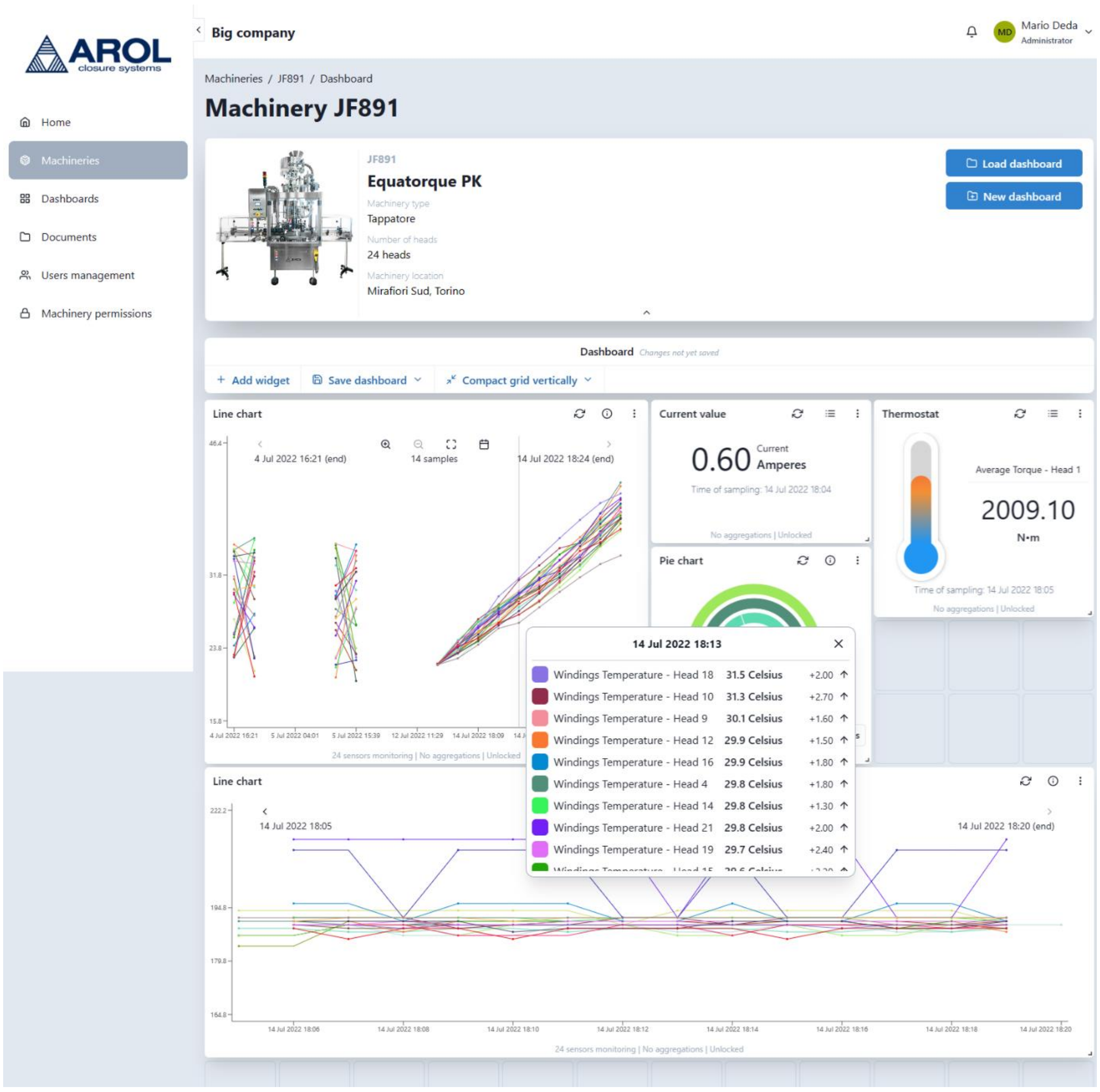


Figure 4.12: Machinery dashboard



Machineries / JF891 / Dashboard

Machinery JF891

JF891

Equatorque PK

Machinery type: Tappatore

Number of heads: 24 heads

Machinery location: Mirafiori Sud, Torino

Dashboard *Changes not yet saved*

Current value: 0.60 Current Amperes — Time of sampling: 14 Jul 2022 18:04 — No aggregations | Unlocked

Thermostat: Average Torque - Head 1 — 2009.10 N·m — Time of sampling: 14 Jul 2022 18:05 — No aggregations | Unlocked

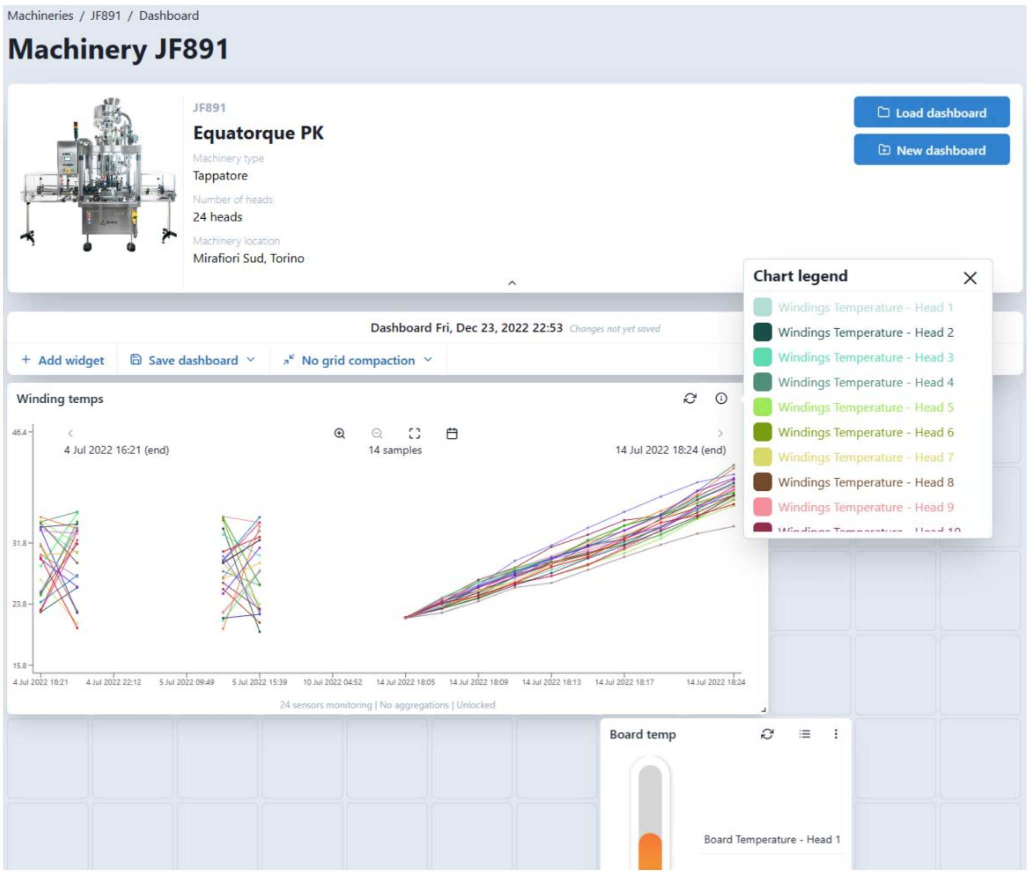
Line chart: 4 Jul 2022 16:21 (end) — 14 samples — 14 Jul 2022 18:24 (end) — 24 sensors monitoring | No aggregations | Unlocked

Line chart: 14 Jul 2022 18:05 — 14 Jul 2022 18:20 (end) — 24 sensors monitoring | No aggregations | Unlocked

14 Jul 2022 18:13

Windings Temperature - Head 18	31.5 Celsius	+2.00	↑
Windings Temperature - Head 10	31.3 Celsius	+2.70	↑
Windings Temperature - Head 9	30.1 Celsius	+1.60	↑
Windings Temperature - Head 12	29.9 Celsius	+1.50	↑
Windings Temperature - Head 16	29.9 Celsius	+1.80	↑
Windings Temperature - Head 4	29.8 Celsius	+1.80	↑
Windings Temperature - Head 14	29.8 Celsius	+1.30	↑
Windings Temperature - Head 21	29.8 Celsius	+2.00	↑
Windings Temperature - Head 19	29.7 Celsius	+2.40	↑

Figure 4.13: Machinery dashboard – chart tooltip


Machineries / JF891 / Dashboard

Machinery JF891

JF891

Equatorque PK

Machinery type
Tappatore

Number of heads
24 heads

Machinery location
Mirafiori Sud, Torino

Dashboard Fri, Dec 23, 2022 22:53 *Changes not yet saved*

Chart legend

- Windings Temperature - Head 1
- Windings Temperature - Head 2
- Windings Temperature - Head 3
- Windings Temperature - Head 4
- Windings Temperature - Head 5
- Windings Temperature - Head 6
- Windings Temperature - Head 7
- Windings Temperature - Head 8
- Windings Temperature - Head 9

Figure 4.14: Machinery dashboard - chart legend

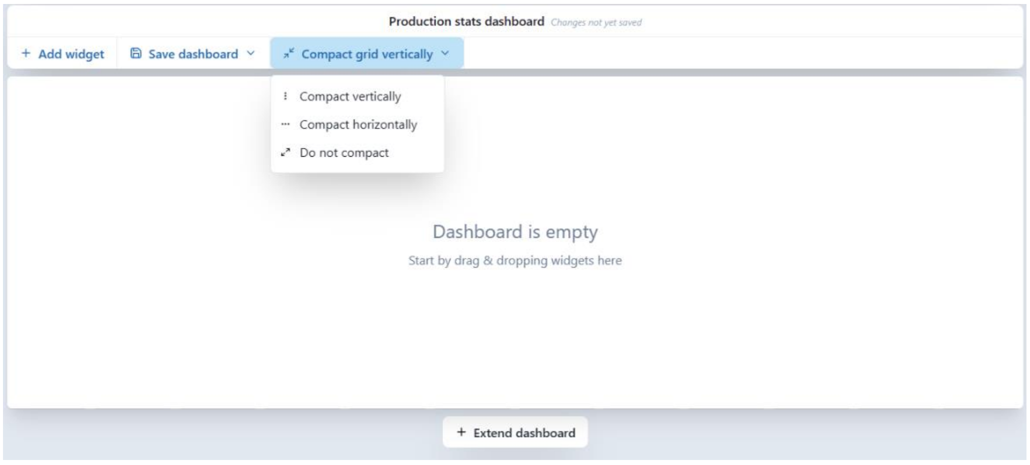

Production stats dashboard *Changes not yet saved*

Figure 4.15: Machinery dashboard - dashboard compaction type menu

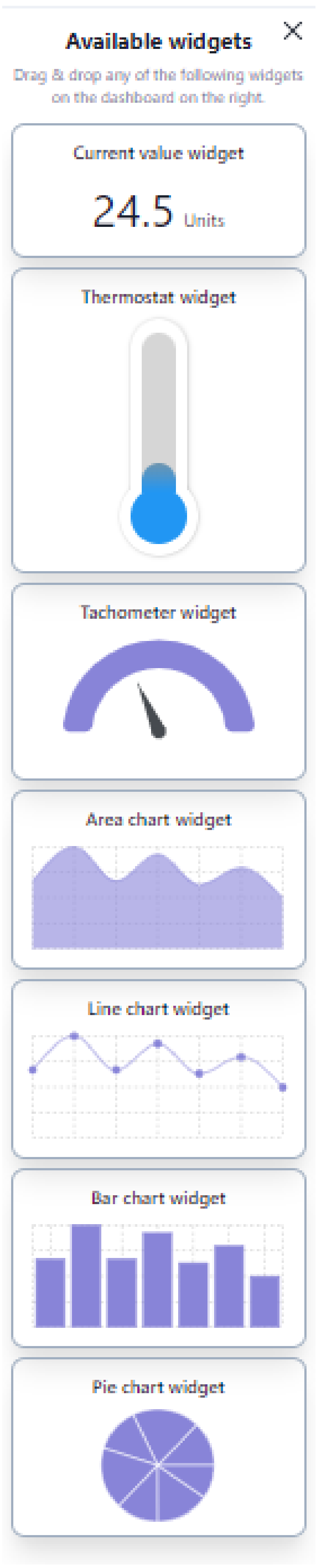
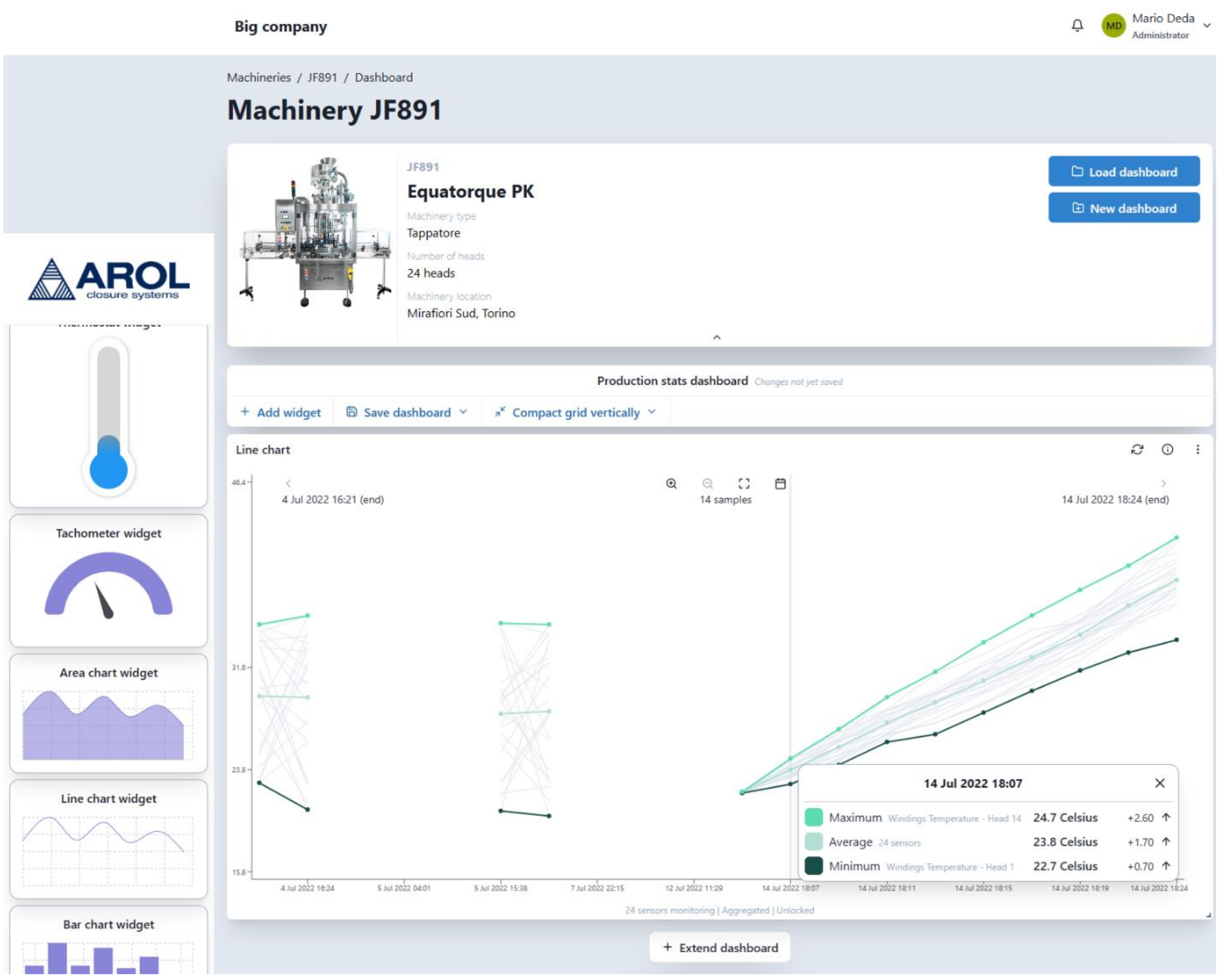


Figure 4.16: Machinery dashboard - widget selector sidebar

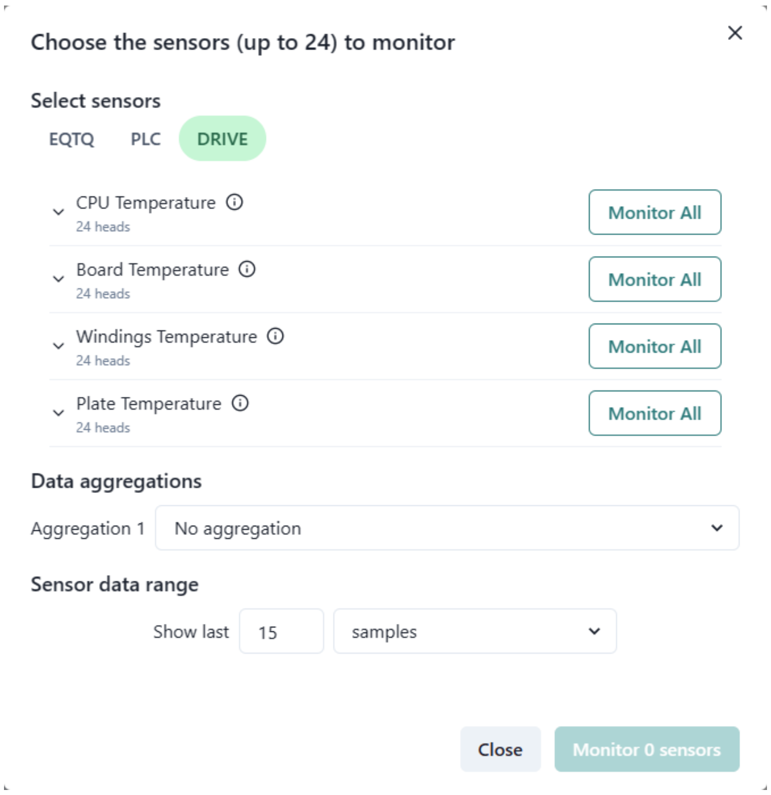


JF891
Equatorque PK
Machinery type
Tappatore
Number of heads
24 heads
Machinery location
Mirafiori Sud, Torino

14 Jul 2022 18:07

Maximum	Windings Temperature - Head 14	24.7 Celsius	+2.60	↑
Average	24 sensors	23.8 Celsius	+1.70	↑
Minimum	Windings Temperature - Head 1	22.7 Celsius	+0.70	↑

Figure 4.17: Machinery dashboard - aggregation function on a chart widget



Choose the sensors (up to 24) to monitor

Select sensors

EQTQ PLC DRIVE

CPU Temperature
24 heads
Monitor All

Board Temperature
24 heads
Monitor All

Windings Temperature
24 heads
Monitor All

Plate Temperature
24 heads
Monitor All

Data aggregations

Aggregation 1: No aggregation

Sensor data range

Show last: 15 samples

Close

Monitor 0 sensors

Figure 4.18: Machinery dashboard - sensors to monitor configuration panel

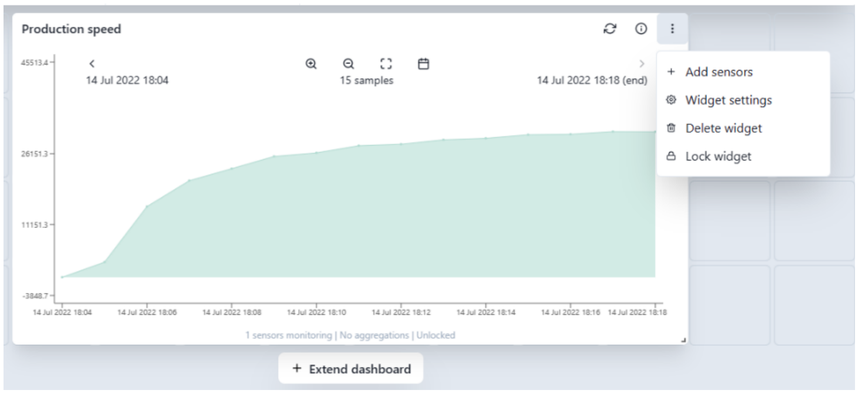
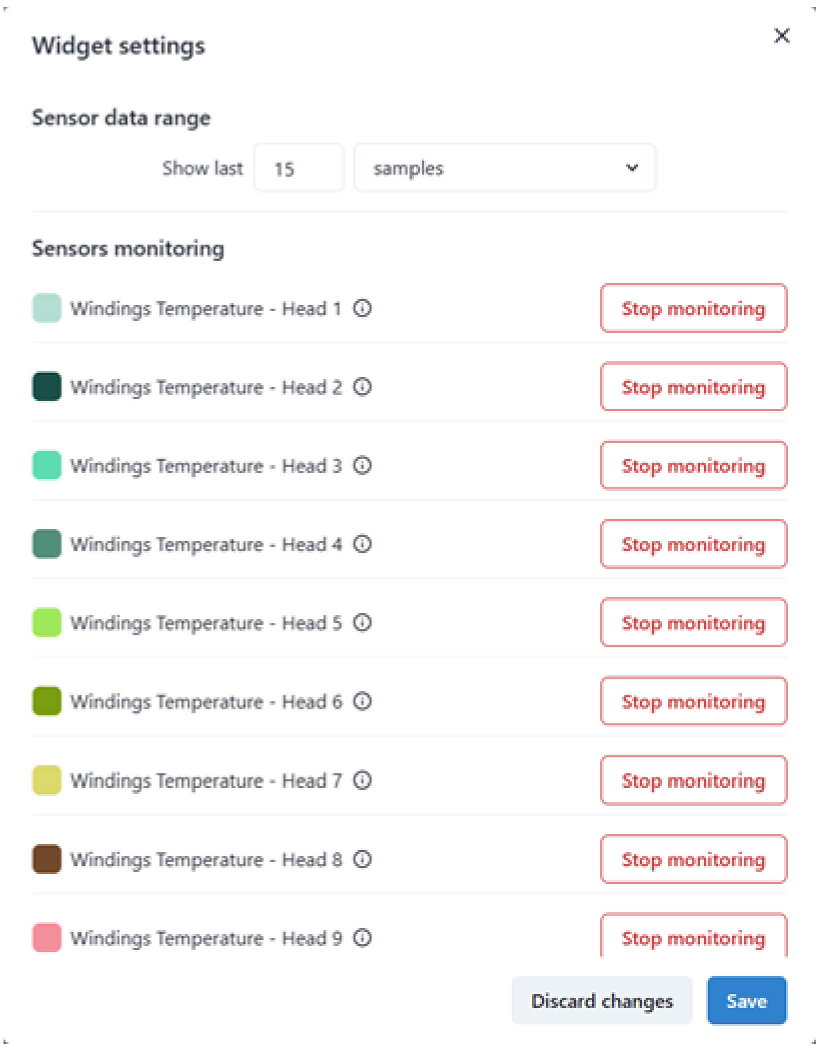


Figure 4.19: Machinery dashboard - widget menu



Widget settings

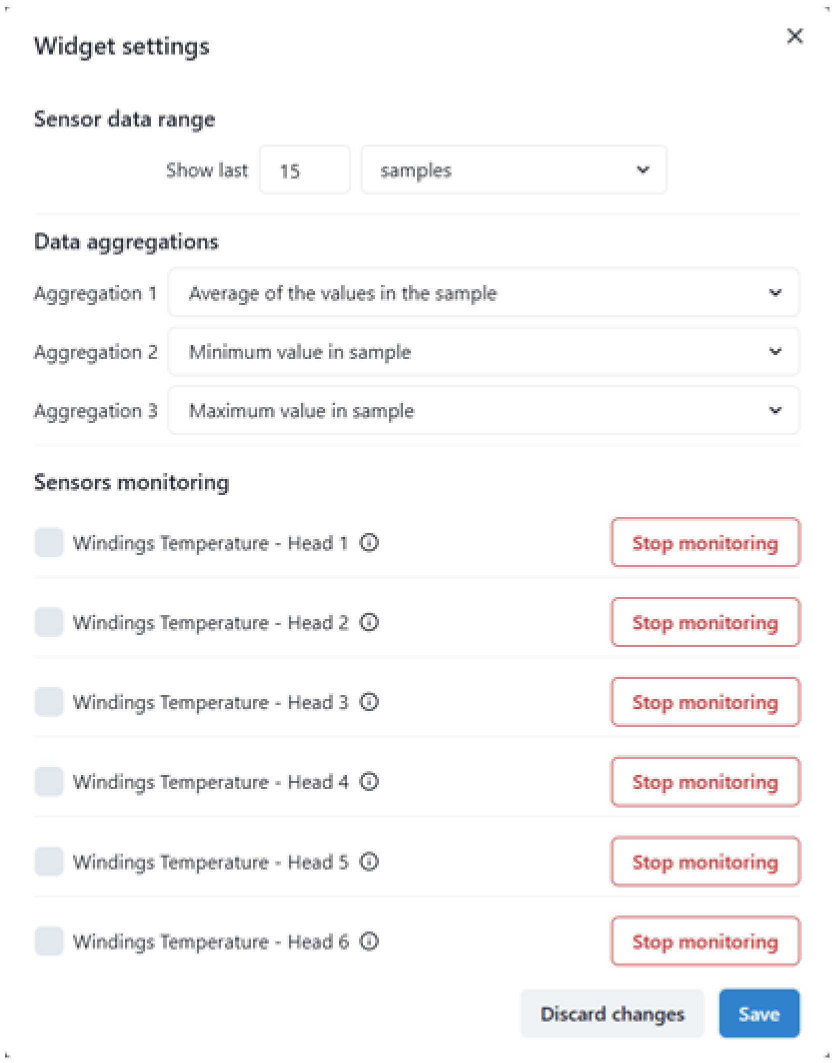
Sensor data range

Show last 15 samples

Sensors monitoring

Windings Temperature - Head 1	Stop monitoring
Windings Temperature - Head 2	Stop monitoring
Windings Temperature - Head 3	Stop monitoring
Windings Temperature - Head 4	Stop monitoring
Windings Temperature - Head 5	Stop monitoring
Windings Temperature - Head 6	Stop monitoring
Windings Temperature - Head 7	Stop monitoring
Windings Temperature - Head 8	Stop monitoring
Windings Temperature - Head 9	Stop monitoring

Figure 4.20: Machinery dashboard - widget settings menu (without aggregations)


Widget settings

Sensor data range

Show last 15 samples

Data aggregations

Aggregation 1 Average of the values in the sample

Aggregation 2 Minimum value in sample

Aggregation 3 Maximum value in sample

Sensors monitoring

Windings Temperature - Head 1 — Stop monitoring

Windings Temperature - Head 2 — Stop monitoring

Windings Temperature - Head 3 — Stop monitoring

Windings Temperature - Head 4 — Stop monitoring

Windings Temperature - Head 5 — Stop monitoring

Windings Temperature - Head 6 — Stop monitoring

Figure 4.21: Machinery dashboard - widget settings (with aggregations)

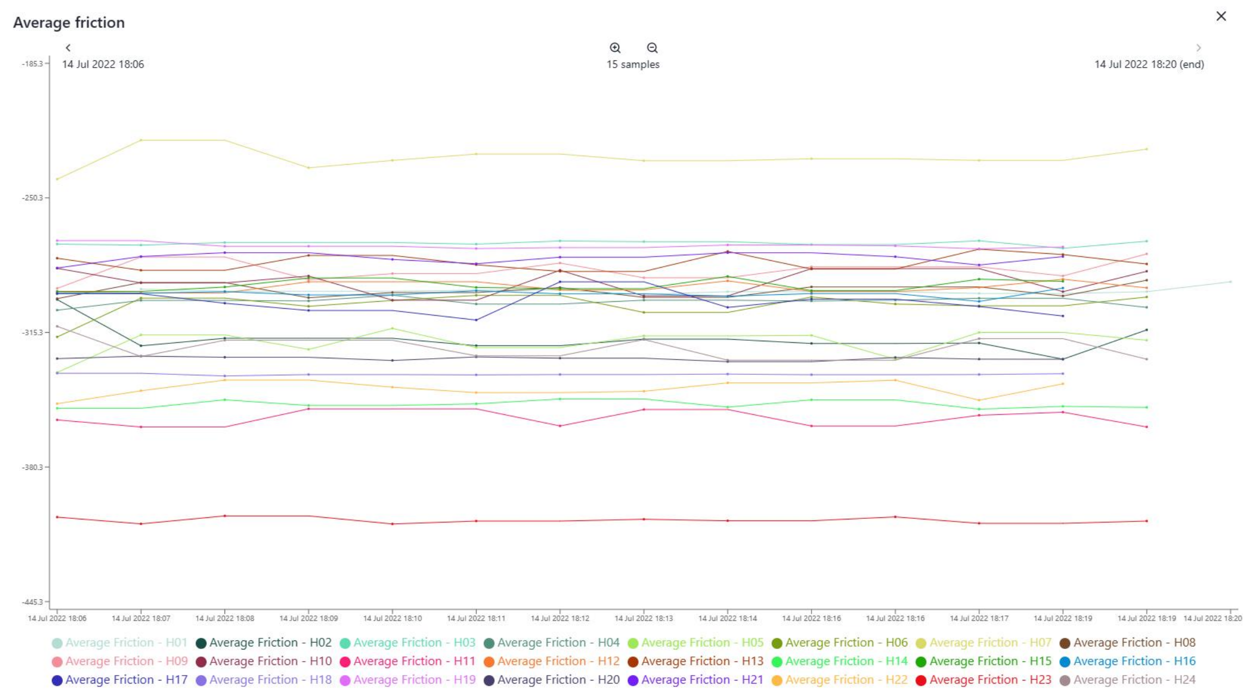
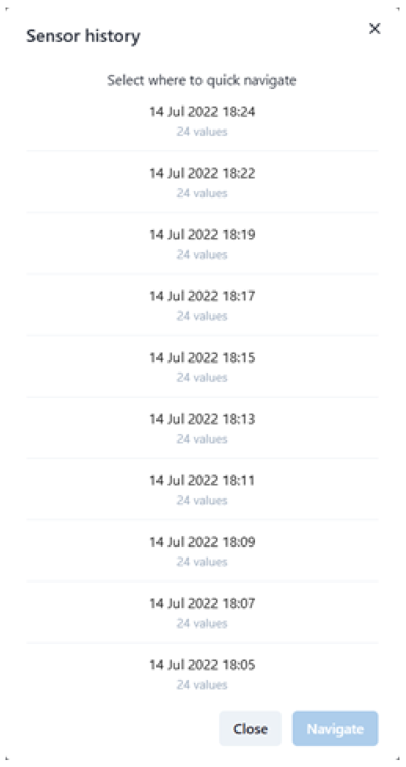


Figure 4.22: Machinery dashboard – full-screen viewer for chart widgets


Sensor history

Select where to quick navigate

14 Jul 2022 18:24
24 values

14 Jul 2022 18:22
24 values

14 Jul 2022 18:19
24 values

14 Jul 2022 18:17
24 values

14 Jul 2022 18:15
24 values

14 Jul 2022 18:13
24 values

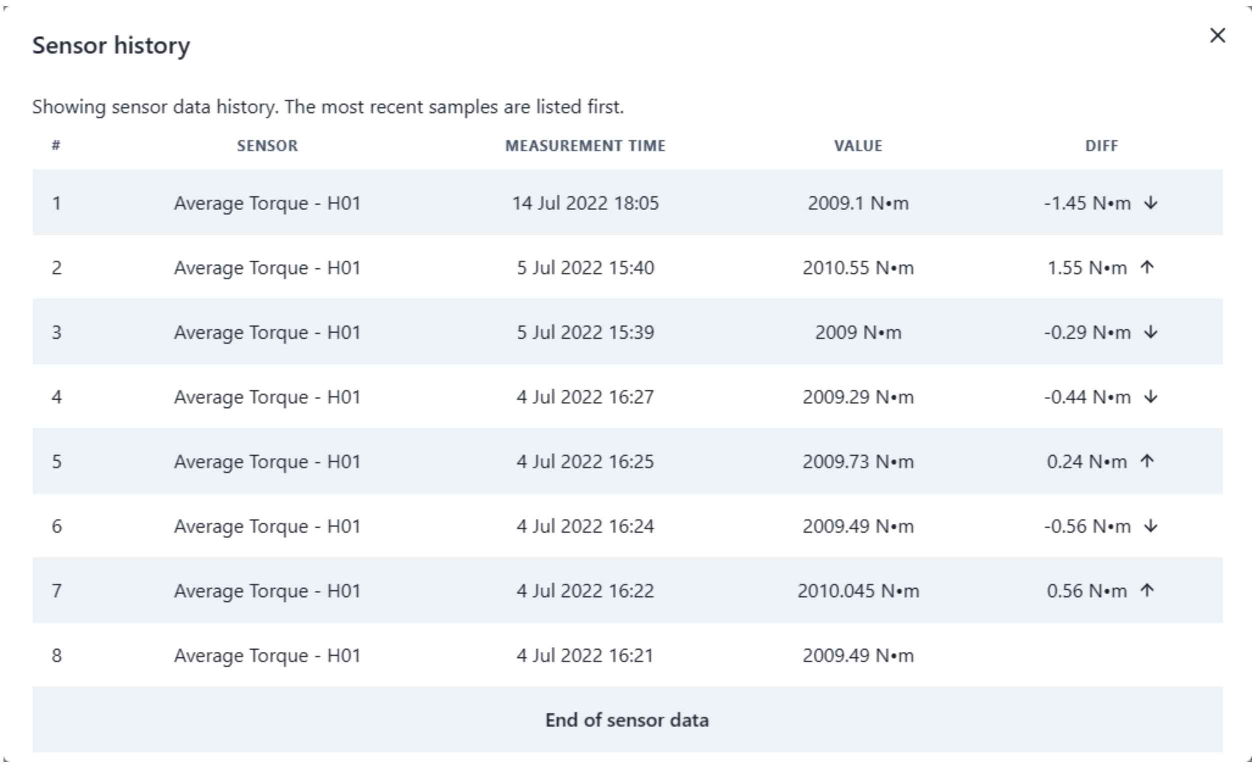
14 Jul 2022 18:11
24 values

14 Jul 2022 18:09
24 values

14 Jul 2022 18:07
24 values

14 Jul 2022 18:05
24 values

Figure 4.23: Machinery dashboard - chart quick navigation panel



Sensor history

Showing sensor data history. The most recent samples are listed first.

#	SENSOR	MEASUREMENT TIME	VALUE	DIFF
1	Average Torque - H01	14 Jul 2022 18:05	2009.1 N•m	-1.45 N•m ↓
2	Average Torque - H01	5 Jul 2022 15:40	2010.55 N•m	1.55 N•m ↑
3	Average Torque - H01	5 Jul 2022 15:39	2009 N•m	-0.29 N•m ↓
4	Average Torque - H01	4 Jul 2022 16:27	2009.29 N•m	-0.44 N•m ↓
5	Average Torque - H01	4 Jul 2022 16:25	2009.73 N•m	0.24 N•m ↑
6	Average Torque - H01	4 Jul 2022 16:24	2009.49 N•m	-0.56 N•m ↓
7	Average Torque - H01	4 Jul 2022 16:22	2010.045 N•m	0.56 N•m ↑
8	Average Torque - H01	4 Jul 2022 16:21	2009.49 N•m	

End of sensor data

Figure 4.24: Machinery dashboard - sensor data history panel

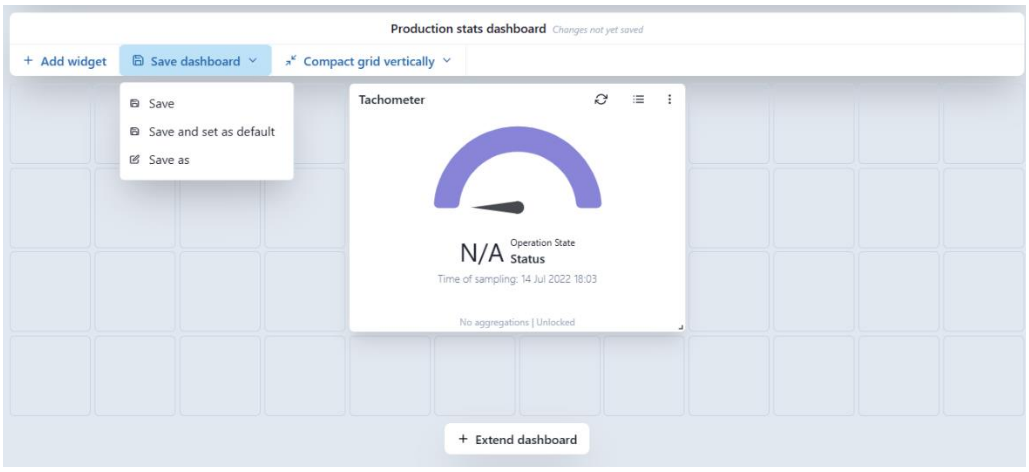
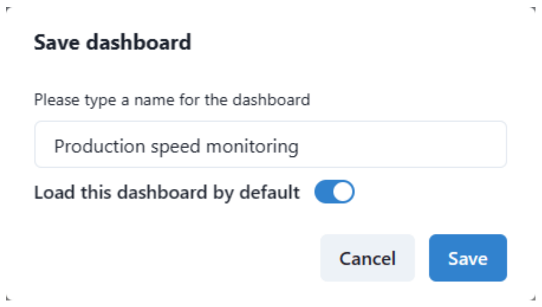


Figure 4.25: Machinery dashboard - save dashboard menu



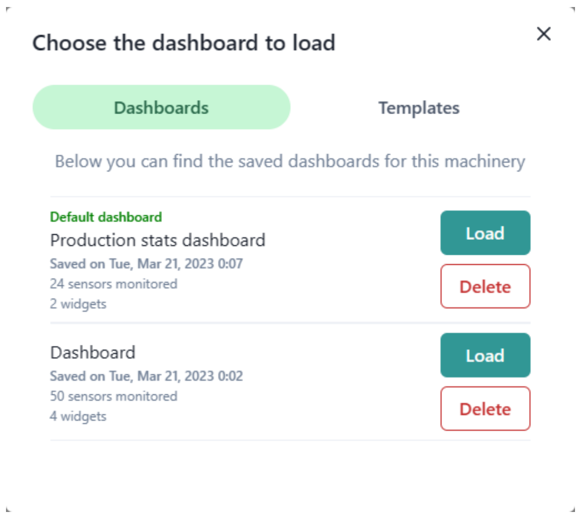
Save dashboard

Please type a name for the dashboard

Production speed monitoring

Load this dashboard by default

Figure 4.26: Machinery dashboard - Save As menu



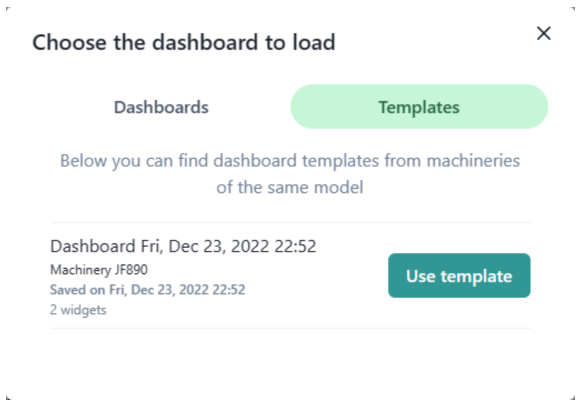
Choose the dashboard to load

Below you can find the saved dashboards for this machinery

Default dashboard
Production stats dashboard
Saved on Tue, Mar 21, 2023 0:07
24 sensors monitored
2 widgets

Dashboard
Saved on Tue, Mar 21, 2023 0:02
50 sensors monitored
4 widgets

Figure 4.27: Machinery dashboard - manage machinery dashboards panel



Choose the dashboard to load

Below you can find dashboard templates from machineries of the same model

Dashboard Fri, Dec 23, 2022 22:52
Machinery JF890
Saved on Fri, Dec 23, 2022 22:52
2 widgets

Figure 4.28: Machinery dashboard - load dashboard as template

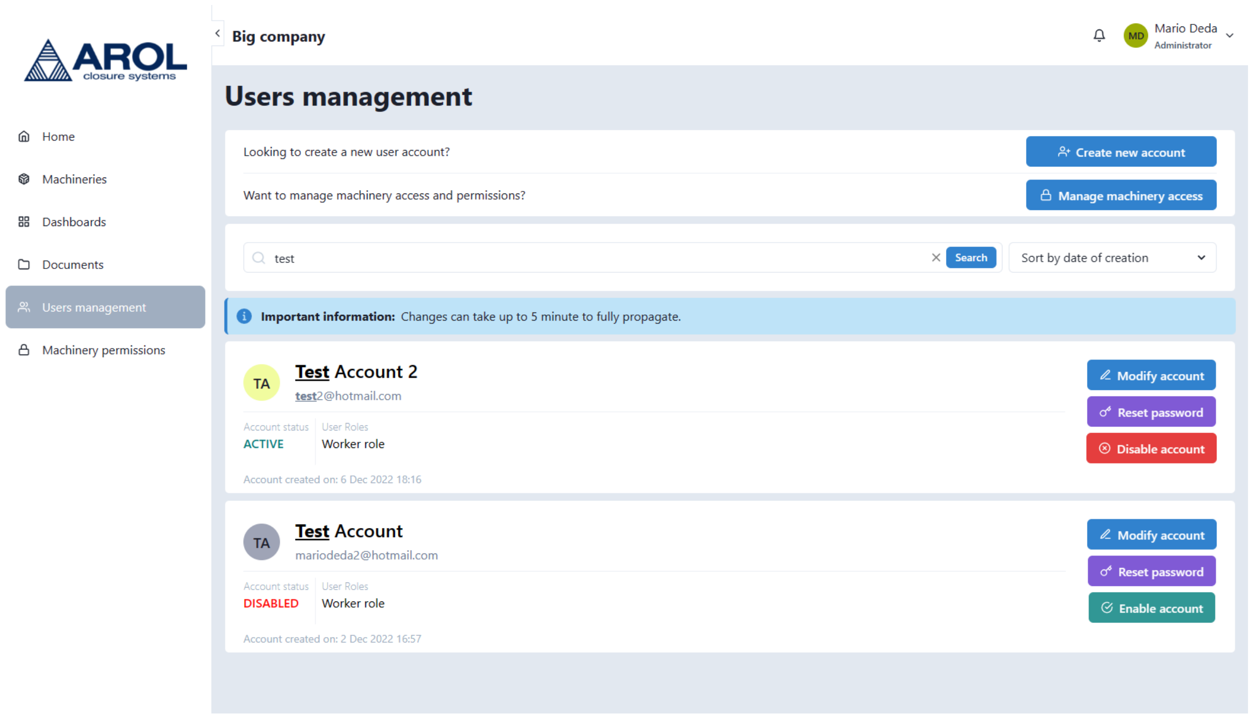
4.3.9 User management

Figure 4.29 and Figure 4.30 show the User Management console or page, which is responsible for interacting with the user accounts management system. This system is fully accessible by the Administrators and partially available to the Managers. It also provides some basic search and sort functionality in order to quickly locate any required user account.

Table 4.4 below lists all the operations that can be performed via the user management system and the corresponding role(s) needed to perform the operation.

Operation	**Role(s) needed**
Create a new user account	Manager and Administrator
Modify user account details	Administrator only
Disable user account	Administrator only
Reset account password	Administrator only

Table 4.4: User management - operations and needed roles



Users management

Looking to create a new user account?

Want to manage machinery access and permissions?

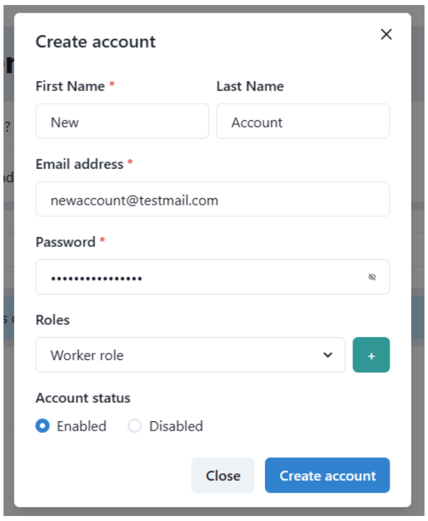
test

Important information: Changes can take up to 5 minute to fully propagate.

TA **Test** Account 2
test2@hotmail.com
Account status: ACTIVE | User Roles: Worker role
Account created on: 6 Dec 2022 18:16

TA **Test** Account
mariodeda2@hotmail.com
Account status: DISABLED | User Roles: Worker role
Account created on: 2 Dec 2022 16:57

Figure 4.29: User management console


Create account

First Name *: New

Last Name: Account

Email address *: newaccount@testmail.com

Password *: ••••••••••••••••

Roles: Worker role

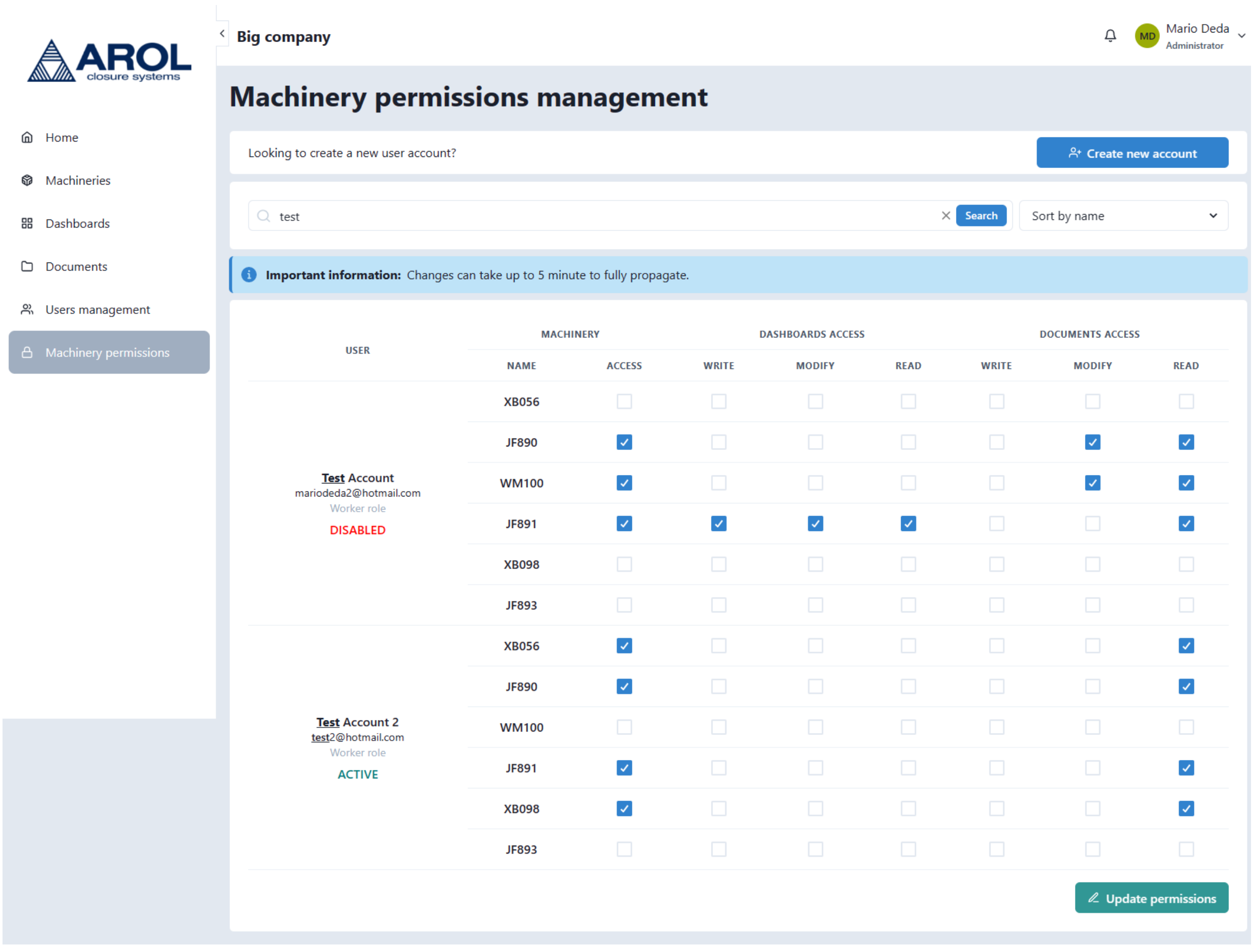
Account status: [✓ Enabled] Disabled

Figure 4.30: User management console - create a new user account modal

4.3.10 Permissions management

Figure 4.31 shows the Permissions Management console or page. This page allows the user to configure the machinery permissions for each user. It is accessible only by the Administrator and the Manager and it also provides a search and sort functionality in order to quickly locate any required user account and machinery. The administrator can manage the permissions of all the users for all the company machineries, while the Manager is restricted to only manage the permissions for the machineries to which he has been given access. The Manager cannot assign permissions that are of higher authority than his.

Machineries, user accounts, and the corresponding permissions are organized in a table. For each machinery, all user accounts are listed, together with their permissions. Administrators and Managers can provide or revoke access to dashboards and document resources by checking and unchecking the corresponding checkboxes.



Machinery permissions management

Looking to create a new user account?

test

Important information: Changes can take up to 5 minute to fully propagate.

USER	MACHINERY		DASHBOARDS ACCESS			DOCUMENTS ACCESS		
	NAME	ACCESS	WRITE	MODIFY	READ	WRITE	MODIFY	READ
Test Account, mariodeda2@hotmail.com, Worker role, DISABLED	XB056	☐	☐	☐	☐	☐	☐	☐
	JF890	✓	☐	☐	☐	☐	✓	✓
	WM100	✓	☐	☐	☐	☐	✓	✓
	JF891	✓	✓	✓	✓	☐	☐	✓
	XB098	☐	☐	☐	☐	☐	☐	☐
	JF893	☐	☐	☐	☐	☐	☐	☐
Test Account 2, test2@hotmail.com, Worker role, ACTIVE	XB056	✓	☐	☐	☐	☐	☐	✓
	JF890	✓	☐	☐	☐	☐	☐	✓
	WM100	☐	☐	☐	☐	☐	☐	☐
	JF891	✓	☐	☐	☐	☐	☐	✓
	XB098	✓	☐	☐	☐	☐	☐	✓
	JF893	☐	☐	☐	☐	☐	☐	☐

Figure 4.31: Machinery permissions management console

4.4 Technical details

This section describes some interesting technical difficulties that needed to be overcome during the development of the proposed web application. Together with a thorough description of each problem, the corresponding solutions together with arguments to support them are provided and detailed.

4.4.1 Dashboards persistence & de/serialization

Dashboards may be subject to multiple changes and edits during their lifetime, as users may need to rearrange, add or remove widgets or configure the set of sensors to monitor. To provide a persistence mechanism for the dashboards, so users can store their desired configurations, the dashboard system should not only be efficient but also take care to store and persist the dashboard configuration in a serializable data structure.

Serialization is the process of converting data structures or objects in a program into a format that can be stored or transmitted. For serializing the dashboards and preparing them to be securely stored and persisted, the JavaScript Object Notation (JSON) serialization strategy has been chosen.

JSON serialization is the process of converting data objects or structures into a JSON format, which is a lightweight and widely-used data interchange format. In JSON serialization, the data objects are transformed into a string of text that can be transmitted or stored. JSON has become a very popular format for serialization thanks to its simplicity, readability, and interoperability with multiple programming languages.

In order for the users to be able to load a previously persisted dashboard configuration, the inverse process of serialization must be executed. Deserialization is the process of converting serialized data back into its original format, which typically is an object or a data structure. It is the opposite of serialization.

Once the previously saved dashboard JSON string has been deserialized into a data object, the dashboard is loaded and displayed to the user.

Figure 4.32 below illustrates in detail the dashboard persistence and de/serialization workflow.

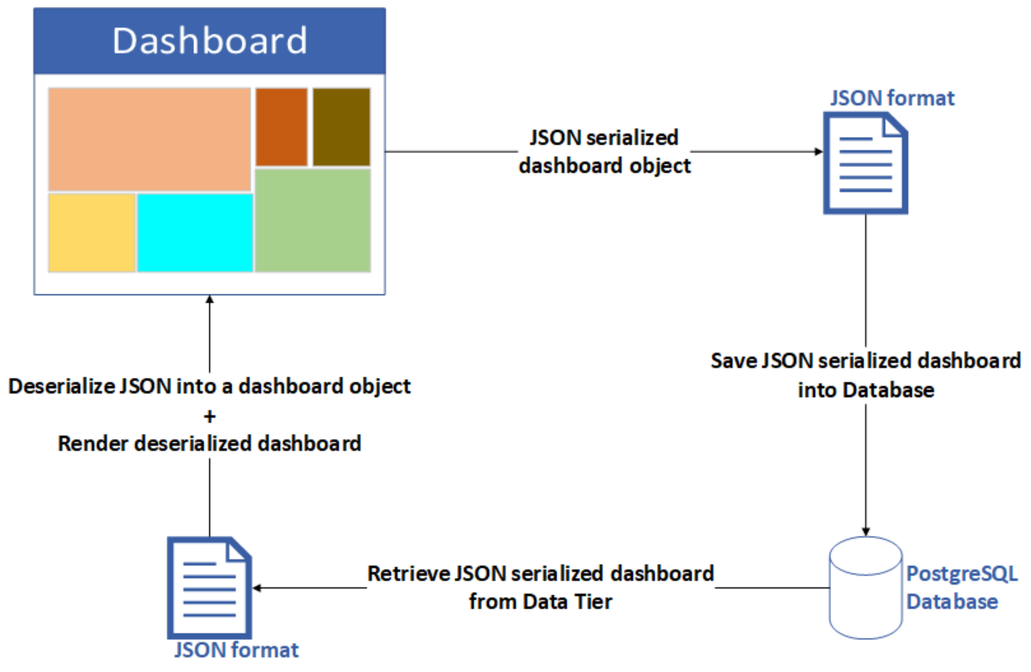

Dashboard
JSON serialized dashboard object
JSON format
Save JSON serialized dashboard into Database
PostgreSQL Database
Retrieve JSON serialized dashboard from Data Tier
JSON format
Deserialize JSON into a dashboard object
+
Render deserialized dashboard

Figure 4.32: Dashboard persistence strategy

4.4.2 Sensor data processing

As summarized in Figure 4.33, the goal of this step is to transform the raw sensor data that is stored in the cloud into useful information that can be presented to the user to enable them to make informed decisions.

Firstly, the necessary sensor data is queried and extracted from the cloud. As the data is stored in AWS Timestream, there are costs associated with the amount of data scanned and retrieved, so in order to limit the costs, special care has been taken to limit as much as possible the amount of data transferred from the cloud by allowing the user to select the exact amount of sensor data they need. The user can specify the number of samples to load or select a time window which can be a custom number of days, weeks, or months.

After the first step, the raw sensor data needs to be processed and a chain of operations needs to be performed. Firstly, a bucketing operation is performed on the data of each sensor. The bucket duration, or interval, can be customized by the user. Sensor data is bucketed based on the bucketing rule of each sensor such as minimum, maximum, average, majority, etc. After this operation, the sensor data is checked for time gaps in the data that indicate that the machinery has been turned off. Marker samples that indicate that the machinery was turned off, and for how much time, are then inserted correspondingly.

For widgets that need to monitor multiple sensors in parallel, subsequent processing operations need to take place. Initially, a vertical bucketing operation is performed. This operation segments sensor data into slices, which are time intervals, or windows, that contain at most one sample per each monitored sensor. The vertical bucketing is necessary as the sampling time resolution is measured in milliseconds and the devices that collect sensor data may not be fast enough to take a reading of all the sensor values within a millisecond. This operation ensures that the data samples which were read virtually at the same time are grouped inside the same slice. After the samples are correctly grouped together, the samples of each sensor in each slice that may have missing values are filled with filler values. The samples which have filler values are specially marked in order to distinguish them from the samples which have real values. This operation is necessary so that widgets that need continuous data can function properly.

In case the user has selected one or more aggregation functions, they are now calculated and inserted correspondingly in the processed data.

Following the processing steps described above, additional checks are performed to determine if there exist any sensor data samples which are older than the set of sensor data that was just processed. If necessary, an additional query is performed against the database that stores the sensor data. This check is required in order to inform the user in time if they have reached the end of the sensor data history.

The processed data is then correctly formatted and bundled together and sent to the client application for it to be displayed to the user.

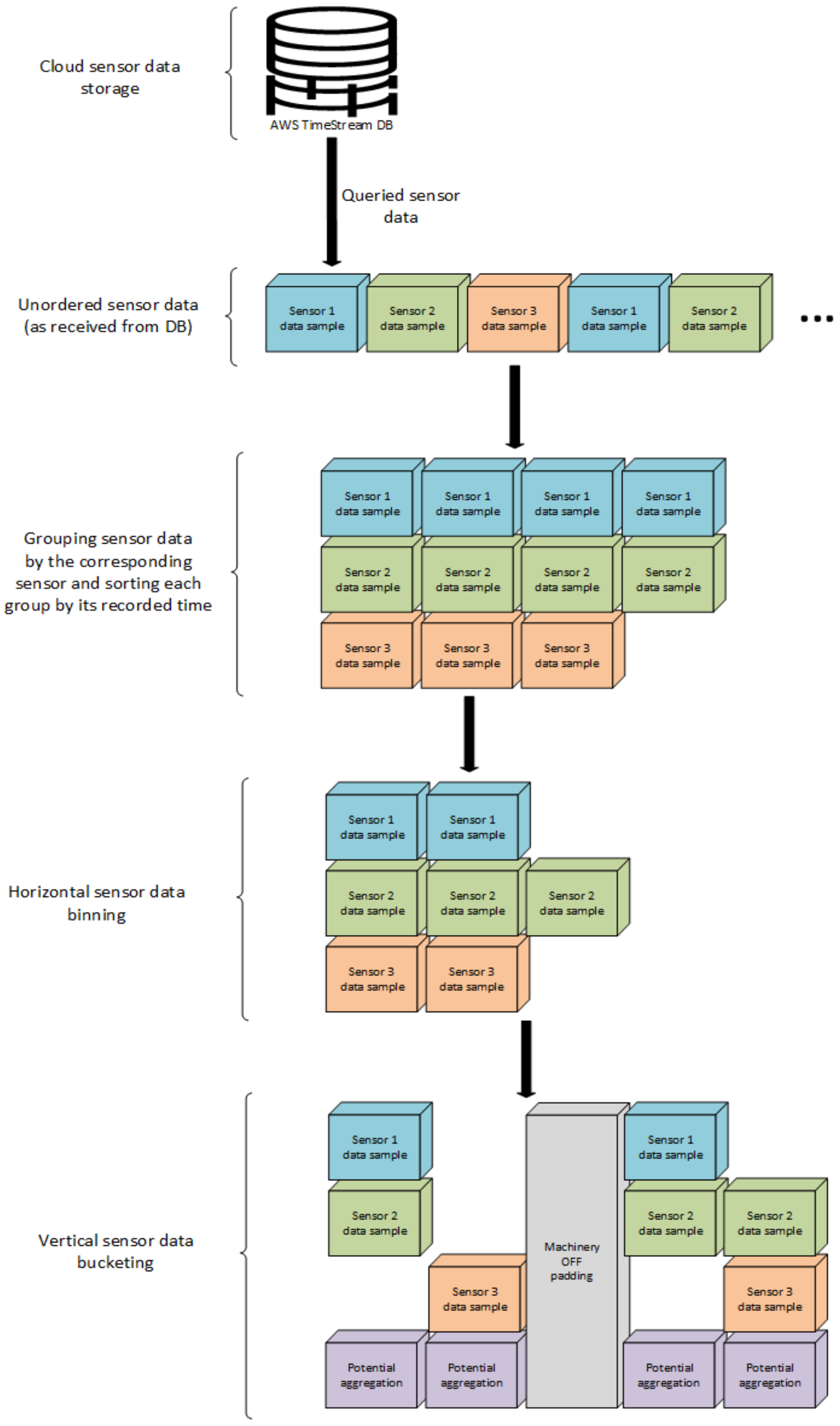

Cloud sensor data storage
AWS TimeStream DB
Queried sensor data
Unordered sensor data (as received from DB)
Sensor 1 data sample
Sensor 2 data sample
Sensor 3 data sample
Grouping sensor data by the corresponding sensor and sorting each group by its recorded time
Horizontal sensor data binning
Vertical sensor data bucketing
Machinery OFF padding
Potential aggregation

Figure 4.33: Sensor data processing diagram

4.4.3 Monitoring multiple sensors in parallel

The ability to monitor sensors in parallel allows enables users to cross-check data, which can increase the accuracy and reliability of how the measurements are interpreted. This is especially true if multiple sensors measuring the same parameter are being monitored parallelly as any significant discrepancies can be quickly identified.

In order to enable users to monitor multiple sensors in parallel, charts have been used. Charts allow to easily visualize data from multiple sensors in a single view, which can help identify trends or patterns that might not be immediately apparent when looking at the data from individual sensors.

With the aim of augmenting, improving, and enriching the user experience while navigating the monitored sensors, a chart control panel has been implemented. This panel houses additional features that have been implemented in the chart viewer such as zooming in and out of the data.

It is important to always keep in mind that the amount of sensor data is always increasing. This problem is only worsened as more and more sensors are monitored in parallel. Not only is it difficult for the user to monitor a high number of samples in a limited amount of space but it can also potentially lead to high usage of system resources and as a consequence a less responsive interface. In order to tackle the aforementioned issues and stress points in the application, a sliding window data navigation strategy has been implemented in chart widgets.

As demonstrated in Figure 4.34, the sliding window data navigation is a technique that involves processing a fixed-size subset of data, which moves through the entire dataset one or more steps at a time. One of the key benefits of the sliding window technique is that it allows for the efficient processing of large datasets. By processing a fixed-size window of data at a time, the computational complexity of the data handling algorithms is reduced and the memory requirements are also minimized. This technique is well suited to many data types, including time-series data, in which the sliding window represents a segment of time.

Users can interact with the sliding window system by controlling and navigating the chart data to the left or to the right by clicking the appropriate buttons in the chart control panel. The number of steps that the chart moves is dynamically adjusted based on the number of samples that are being displayed inside the active window.

Both the zoom-in and zoom-out functionalities work by leveraging the sliding windows system. This ensures high performance even in highly parallel monitoring scenarios.

As discussed in the “Sensor data processing” section, it is important to limit the amount of data that is queried and retrieved to and from the cloud, but under no circumstances this restriction should impact the final user experience while navigating the monitored sensors. To alleviate the problem of having to load older, or historical, sensor data samples by batches to limit the amount of unnecessary data being moved around, a prefetching functionality has been implemented.

The prefetch system will start to load a previous batch of data when the user reaches a prefixed threshold, close to the end of the data available in the oldest batch available locally. This strategy ensures seamless navigation in “normal” network conditions, with the user not noticing that the data is being dynamically loaded as they scroll the chart.

As the amount of data grows it may become increasingly difficult and time-consuming for the user to be able to navigate to previous points in time as they may be a great number of steps away. In order to resolve this problem, a quick navigation functionality has been implemented. The user is allowed to quick-navigate to the desired point in time by just scrolling and selecting the desired entry from a list of chronologically sorted samples and their time of recording.

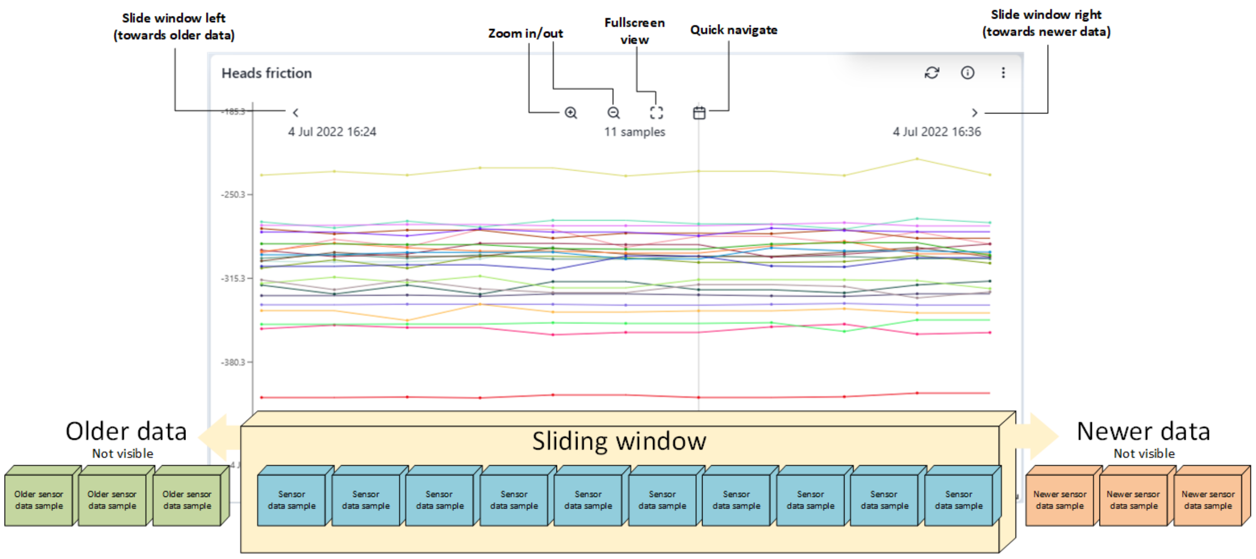


Figure 4.34: Sliding Window data visualization technique in chart widgets

4.5 Performance

Now we move to analyze the implemented application performance in terms of used memory and application loading times. In this section, the JavaScript memory organization is explored and measurements of the application memory usage have been collected. In order to measure the application loading times, the app web vitals have been measured and described.

4.5.1 JavaScript memory organization

The JavaScript engine organized memory into two main areas: the memory stack and the memory heap.

The memory stack is where static data allocation occurs. It is a data structure that is used to store information about the execution of a program. The stack is used to keep track of all the functions that the JavaScript interpreter needs to call. The stack is populated at compilation time, so its size is already known at compile time and it will not change during the lifecycle of the program. This memory frame contains primitive values such as strings, numbers, booleans, and references to objects and functions. The stack is also used to store vital JavaScript structures like the call stack and the event loop.

As discussed above, the stack only contains references to objects and functions. These objects and functions are actually stored in the heap. The heap is the memory section where dynamic memory allocation occurs. Since it is dynamically allocated, the heap does not have a fixed size. Its size is known only at runtime and will change during the execution of the program since objects are created dynamically as the program runs. To prevent the heap size from continuously increasing when the program is in execution, when an object is no longer needed, it is marked for garbage collection and the memory it occupied is released and freed.

The JavaScript Garbage Collector is used to manage the last step of the memory lifecycle: releasing unused memory. The garbage collector periodically checks the heap for objects that are no longer referenced by the program and releases the memory occupied by these objects. It is important to note that this process happens automatically and is transparent to the developer.

To limit the amount of unnecessarily allocated memory in a program, it is important to understand and prevent memory leaks. Memory leaks happen in the form of global variables, forgetting to clear timers and callbacks, or out of Document Object Model (DOM) references [47].

Throughout the development of this web application, special care has been taken to never compromise on performance and efficiency. This results in a very optimized web application that can run even on devices that very limited resources.

4.5.2 Application memory usage

Table 4.5 summarizes the amount of memory that the application uses on each of its main pages. Every measurement during this test was collected using the Google Chrome Developer Tools and was taken after manually triggering the garbage collector.

Application page	**Heap memory size**
Login page	10.8 MB
Home page	13.4 MB
Machinery browser	13.7 MB
Dashboard with 4 widgets, 50 sensors	15.3 MB
Dashboard with 12 widgets, 177 sensors	29.9 MB
Machinery landing page	13.6 MB
Documents	18.4 MB
Documents viewer	20.2 MB
Dashboard quick-browse with 8 machineries	13.2 MB
Documents quick-browse with 8 machineries	13.4 MB
User management console	13.3 MB
Machinery permissions console	16.9 MB

Table 4.5: Application heap memory usage, garbage collection triggered before each measurement

On the other hand, Figure 4.35 shows in the form of a chart the memory usage of the application during a 128 seconds session recorded using the Google Chrome Developer Tools. During the usage session, all the application pages were accessed and interacted with. In the end, a logout was performed. It is important to note that before starting the application, the garbage collector was manually triggered. The application reported very good performance with very contained memory usage. The minimum recorded memory usage was at the start of the session, on the login page, with only 10.1MB of memory used, while the maximum was recorded in the machinery dashboard with 56.7MB of memory used. The maximum memory usage was caused by loading a dashboard with 12 widgets and 177 sensors being monitored in parallel.

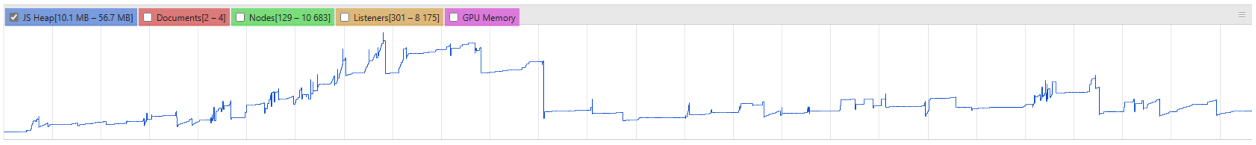


Figure 4.35: Application memory usage measured during a 128 seconds interaction session, with a minimum memory usage of 10.1MB and a maximum of 56.7MB

4.5.3 Web vitals

Web Vitals are a set of standardized metrics introduced by Google that help application developers measure the performance, usability, and overall user experience that a user would perceive when visiting a certain website or web application.

Core Web Vitals help identify user experience problems by generating metrics for the page loading time, the ease of interaction, and the visual stability of the page. Additionally, Google has announced that they will use Web Vitals as a ranking factor in search results, further emphasizing their importance for web applications.

Table 1.1 shows some performance metrics based on the Web Vitals for the implemented web application. A description is also provided for each reported metric. The application loading timeline is also shown in Figure 4.36.

Metric	**Description**	**Result**
Full Time to First Byte	Time for the server to respond to an HTML document request	306ms
First Contentful Paint	Time to render Document Object Model (DOM) elements	2.12s
Time to Interactive	Time for the page content to become fully interactive	2.12s
Largest Contentful Paint	Time to load the page main content	2.33s
Visually Complete	Time to completely finish rendering whole page content	2.37s
Cumulative Layout Shift	How much the page content unexpectedly shifts	0 (no content shift)
CPU Time	Amount of time the browser main thread was busy	571ms

Table 4.6: Web Vitals metrics of the web application (measured using debugbear.com)

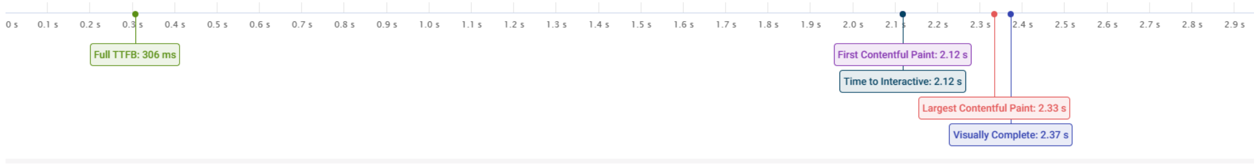


Figure 4.36: Web application loading timeline
Source: DebugBear.com, user console

5 Use case scenario

To test, evaluate and validate the potential of the implemented web application, a real-world use case scenario was staged. This test aims to help validate the proposed web application by demonstrating its practical application, its potential impact, and its effectiveness. This process is also important in determining the possible improvements or shortcomings in the proposed application. Another goal of this process is to test the reliability of the application in real-world conditions and the correctness of the outputted data, two of the most important features for an application used in an industrial environment.

5.1 Introduction

AROL capping machineries operate by applying different types of caps to different types of containers.

The containers are fed into the capping machine, usually via a conveyor belt and the caps or closures are fed into the capping machine, usually from a hopper. The capping machine then sorts the caps and places them onto the containers automatically. The machine applies the required torque to the cap, tightening it onto the container to ensure a secure seal. The caps are then checked to ensure the quality of the cap seal.

Every step of this process is monitored using specialized sensors and the data is collected by the machinery PLC. The IoT Box mounted on the machine periodically extracts the data from the machinery PLC. The collected data is then properly formatted and automatically uploaded to the Cloud.

In order to stage a scenario that is as close as possible to the predicted real-world usage of the application, an AROL machinery with 24 capping heads was used and two AROL technicians were chosen to be part of the test. Firstly, a 30-minute instructional session was held with the technicians where they were instructed on the purpose of the application and how to use it. Then the technicians were provided with a test account that was configured to have a Worker role. Then they were charged with creating a custom machinery dashboard and monitoring the chosen machinery vitals through it. The machinery used in this scenario was left running for 8 hours, simulating a real-world factory usage pattern. During this time, machinery data was continuously collected and uploaded to the cloud. The AROL technicians, after creating the desired dashboard, were tasked with monitoring the selected machinery sensors and vitals using the application. Both technicians were also tasked to periodically check the human-machinery interface of the AROL machinery to validate that the data displayed by the web application was consistent with what the machinery was reading.

During the whole test session, the technicians were accompanied to observe any difficulties they might face while using the application, continuously collect their feedback, and instruct them in case they needed help to interact with the application.

5.2 Sensor Data

During the test, the technicians were instructed to monitor some of the sensors that they deemed important in reporting the machinery operating conditions. The selection of the sensors to monitor was left to the technicians to simulate a real-world usage pattern in the production factory. Table 5.1 lists the sensors that the technicians selected for the test, the sensor measurement unit, and a brief description of each sensor's functionality.

Sensor name (unit)	**Description**
Operation mode (status code)	Indicates the operating mode in which the machine is. **MANUAL (status code 1)** – in manual operation the machine does not follow the production, but the operator acts manually to make it run (for example for debugging or cleaning activities) **AUTOMATIC (status code 2)** – this is the functionality in which the machine is placed when it is in production. **MAINTENANCE (status code 3)** – in some machines there is a specific mode for maintenance operations, in simpler machines the manual mode is used
Operation state (status code)	Indicates the operating state in which the machine is found. This machinery state depends on the surrounding conditions. **STANDBY (status code 1)** – awaiting commands **NO BOTTLES (status code 2)** – awaiting incoming containers before being able to start applying caps **NO CAPS (status code 3)** – awaiting incoming caps **EXITFULL (status code 4)** – at the outlet there is an accumulation of containers so the machine has to wait for this accumulation to disperse **SLOWRUN (status code 5)** – the machinery works in a slowdown condition for various reasons; for example, there are still a few incoming containers or outgoing containers that are starting to accumulate. **RUNNING (status code 6)** – the machinery is working correctly
Alarm status (status code)	This variable indicates one of the possible reasons that led the machine to stop due to an alarm condition. There are many reasons, and they depend heavily on the machine's conditions. Some conditions are listed as follows: **NO_ALARM (status code 0)** – no alarms have been raised, normal operation **SAFETY_ON (status code 1)** – safety circuit tripped (operator safety) **…_DOOR (status code 10 to 20)** – one or machinery doors are open (the machinery cannot operate with the doors open) **LEFT/RIGHT_TUNNEL (status code 5 and 6)** – door on the inlet and outlet part of the bottles is open **IN/MID/OUT_STAR (status code 7, 8, and 9)** – one of the bottle transfer systems inside the machine has tripped and there has been a jamming of the bottles, in order to avoid their destruction, the system stops in alarm.
Production speed (bottles/hour)	Number of bottles capped per hour
Total product (bottles)	Total number of bottles capped by the machinery
Capping heads friction (N)	The friction experienced by each capping head
Capping heads applied torque (N•m)	The torque applied to bottle caps by each capping head
Capping heads bad closures (bottles)	Bottles that could not be properly capped due to an anomaly by each capping head

Table 5.1: Sensor selected for the use case scenario test

5.3 Results

Following the initial instructional session, the AROL technicians were able to navigate, locate and access the dashboard page of the selected machinery. They were also able to create, populate and save a custom dashboard on their own, without needing assistance. Figure 5.1 shows the dashboard created and used for monitoring sensor data by the AROL technicians. As it can be seen by the quality of the created dashboard, the 30-minute instructional session was indeed sufficient for the technicians to understand and learn how to use the dashboard. They were also able to use the shortcuts for collapsing both the side navigation panel and machinery details panel, even though this was not included in the instructional session. When asked about this behavior, one of the technicians explained that the placement and the shape of the collapse and expand buttons were obviously indicating to him their functionality. This trend was repeatedly noticed during the whole test as the technicians got more and more comfortable with the application. They also commented that the application was very forgiving to user errors and provided quick fallback solutions. This functionality encouraged them to be confident in exploring the different application features and interact with almost all of the functionalities that they discovered during the test.

Based on the evaluation carried out by the technicians, the real-time data displayed in the application was found to be accurate and reliable. The machine was allowed to run for 8 hours, during which the technicians closely monitored and compared the data between the dashboard and the machinery to ensure that the visualized data was consistent and valid. In total, more than 25,000 different data samples were collected from the machinery sensors and visualized in the application dashboard.

Most importantly, the technicians who evaluated the monitored data were able to verify that there were no discrepancies between the data shown on the machinery and the dashboard. The proposed application was able to display the data in a timely and reliable manner, providing an accurate representation of the machinery's vitals.

The technicians also pointed out that the advanced data visualization features (chart zoom, chart pan, etc.) resulted very useful to them, especially when they needed to navigate between different data points. They also indicated that they were satisfied with the application's performance and its fluidity in performing the required actions.

It is important to point out that the application did not experience any crashes or abnormal behaviors during its operation, despite the variety and the large amount of the handled data. Table 5.2 and Figure 5.2 depict the memory usage trend of the application during the 8-hour test. These measurements were performed every 30 minutes through the Google Chrome developer console. As it can be seen by the chart, during the first hours the application memory usage continuously increases. This is expected since the application is continuously acquiring new sensor data and storing them, but the data that it has already acquired is not thrown away. This is an intended behavior of the application, as it caches older immutable data to provide a much smoother user experience when the operator may want to navigate to previous data samples. In order to stop the application memory usage from infinitely increasing, since if left running for long enough, it would eventually crash, after a threshold is passed, older cached data will be thrown away in favor of new real-time data. The chart demonstrates this feature starting from the sixth hour of operation. The application memory usage reaches a plateau, indicating that older cached data is being cleared while new real-time data is being acquired.

Overall, the feedback provided by the technicians was positive and motivating considering that the proposed application is just a proof-of-concept one. The most important outcome of the test was that the application's reliability and accuracy in performing its main objective, monitoring machinery parameters, resulted more than satisfactory.

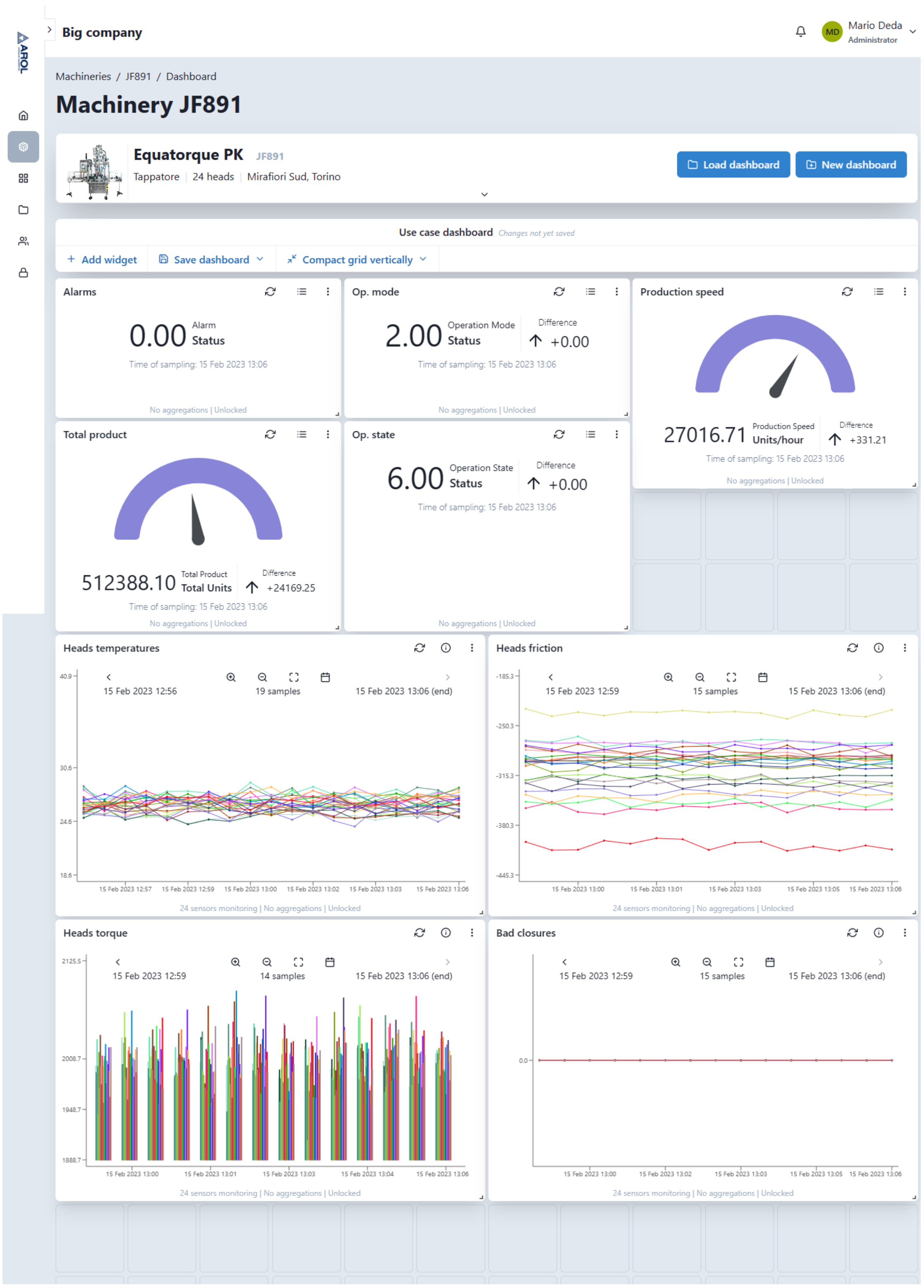

AROL
Big company
Mario Deda
Administrator
Machineries / JF891 / Dashboard
Machinery JF891
Equatorque PK JF891
Tappatore | 24 heads | Mirafiori Sud, Torino
Load dashboard
New dashboard
Use case dashboard Changes not yet saved
+ Add widget
Save dashboard
Compact grid vertically
Alarms
0.00 Alarm Status
Time of sampling: 15 Feb 2023 13:06
No aggregations | Unlocked
Op. mode
2.00 Operation Mode Status
Difference +0.00
Time of sampling: 15 Feb 2023 13:06
No aggregations | Unlocked
Production speed
27016.71 Production Speed Units/hour
Difference +331.21
Time of sampling: 15 Feb 2023 13:06
No aggregations | Unlocked
Total product
512388.10 Total Product Total Units
Difference +24169.25
Time of sampling: 15 Feb 2023 13:06
No aggregations | Unlocked
Op. state
6.00 Operation State Status
Difference +0.00
Time of sampling: 15 Feb 2023 13:06
No aggregations | Unlocked
Heads temperatures
15 Feb 2023 12:56
19 samples
15 Feb 2023 13:06 (end)
24 sensors monitoring | No aggregations | Unlocked
Heads friction
15 Feb 2023 12:59
15 samples
15 Feb 2023 13:06 (end)
24 sensors monitoring | No aggregations | Unlocked
Heads torque
15 Feb 2023 12:59
14 samples
15 Feb 2023 13:06 (end)
24 sensors monitoring | No aggregations | Unlocked
Bad closures
15 Feb 2023 12:59
15 samples
15 Feb 2023 13:06 (end)
24 sensors monitoring | No aggregations | Unlocked

Figure 5.1: Use case scenario test - dashboard created by technicians

Time since test start	Task	Memory usage (interval average)
0h 00m	Login, select machinery and create the dashboard	22.5 MB
0h 30m	Populate the dashboard and start monitoring	27.8 MB
1h 00m	Monitoring via dashboard	26.5 MB
1h 30m	Monitoring via dashboard	29.8 MB
2h 00m	Monitoring via dashboard	30.5 MB
2h 30m	Monitoring via dashboard	33.5 MB
3h 00m	Monitoring via dashboard	35.1 MB
3h 30m	Monitoring via dashboard	40.0 MB
4h 00m	Monitoring via dashboard	41.2 MB
4h 30m	Monitoring via dashboard	45.3 MB
5h 00m	Monitoring via dashboard	45.9 MB
5h 30m	Monitoring via dashboard	50.4 MB
6h 00m	Monitoring via dashboard	56.7 MB
6h 30m	Monitoring via dashboard	56.8 MB
7h 00m	Monitoring via dashboard	57.0 MB
7h 30m	Monitoring via dashboard	56.4 MB
8h 00m	Monitoring via dashboard	58.2 MB

Table 5.2: Application memory usage trend during use case scenario test

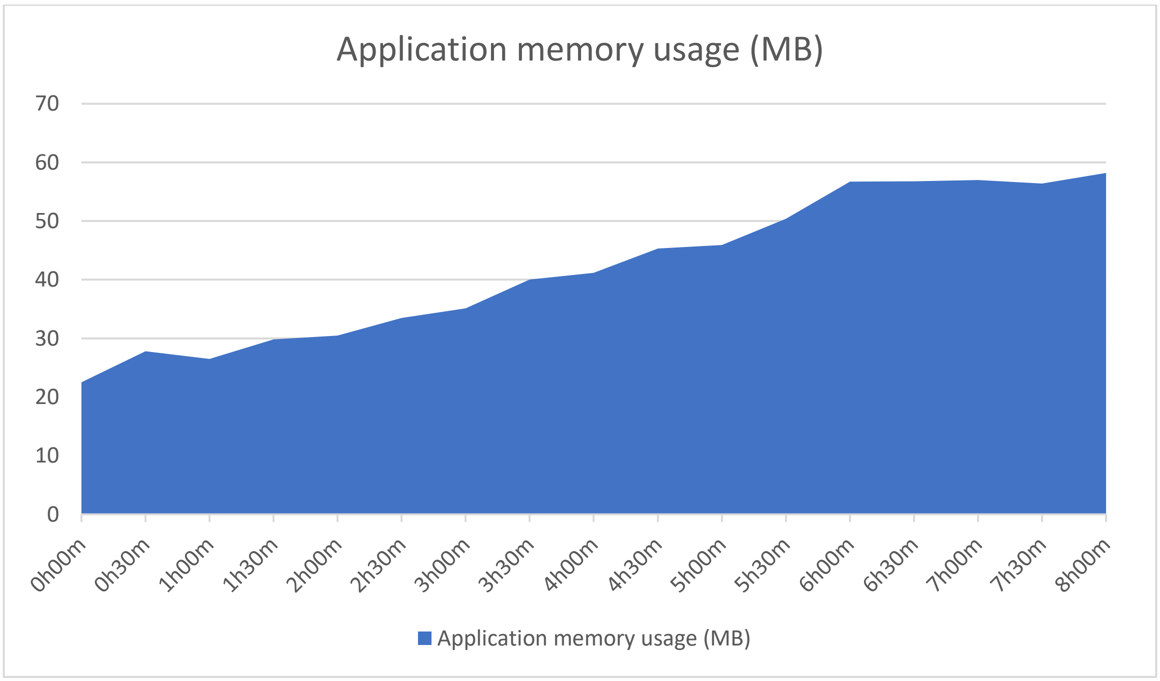


Figure 5.2: Chart of application memory usage trend during use case scenario test

6 Conclusions and future work

The Industrial Internet of Things is a powerful technology that is transforming the manufacturing industry. Through the integration of devices, sensors, and software, IIoT enables companies and manufacturers to collect and monitor data in real time, resulting in increased efficiency and productivity. With numerous devices and sensors generating data at a high frequency, it can be challenging for machinery operators to interpret this data in real time. To provide a solution for this problem, the main goal of this thesis was to study the feasibility and implement a proof-of-concept web application that enables efficient machinery data monitoring by operators through the use of a customizable and dynamic dashboard.

Firstly, the current IoT technological stack of AROL, which will be a starting point for designing the web application, was thoroughly detailed and described. Then, in tandem with AROL, the application requirements were defined and finalized. The tools and technologies that were used to develop the proposed application were then listed and described. To provide a reliable and secure application, architectural, security, and design strategies were carefully scrutinized. Such choices, together with arguments supporting them were then listed and carefully detailed. Then, the developed application features were cataloged and the design decisions were detailed and justified. The application performance was then measured and the results were commented. A deployment strategy was also defined by designing the appropriate cloud architecture and deploying it, together with the application, on Amazon Web Services.

In order to be able to test and verify application effectiveness aiding operators in monitoring a machinery in real-time, a real-world use case scenario was staged. The findings and the results of the test were positive as the AROL technicians appraised the application for its stability, the simplicity of navigating the machinery data, and its user-friendliness. These results were very motivating and the careful design of the architecture and methodical implementation resulted in a fully-working prototype application.

The proposed web application can be furtherly extended and improved with new functionalities, like real-time alarms and event notifications. A real-time sensor data analytics module can also be developed on top of the existing monitoring functionalities to improve the operational efficiency of the machineries and enable more advanced functionalities, like predictive maintenance. Furthermore, given the success that the real-world use case scenario test achieved, further review and development of the current application is necessary in order to transform it into a production-ready product.

7 References

[1] A. Sari, A. Lekidis and I. Butun, "Industrial Networks and IIoT: Now and Future Trends," 2020, pp. 3-55.

[2] T. Yu and X. Wang, "Real-Time Data Analytics in Internet of Things Systems," 2020.

[3] "Industrial Internet of Things," [Online]. Available: https://en.wikipedia.org/wiki/Industrial_internet_of_things. [Accessed 20 3 2023].

[4] "Industrial IIoT: Transforming Operations with Data Communications," International Society of Automation, [Online]. Available: https://www.isa.org/intech-home/2022/august-2022/features/industrial-iiot-transforming-operations-with-data. [Accessed 20 3 2023].

[5] Y. Ersoy, "The Advantages and Barriers in Implementing of Industry 4.0 and Key Features of Industry 4.0," 2022.

[6] T. Kalsoom, S. Ahmed, P. M. Rafi-Ul-Shan, M. Azmat, P. Akhtar, Z. Pervez, M. Imran and M. Ur Rehman, "Impact of IoT on Manufacturing Industry 4.0: A New Triangular Systematic Review," vol. 13, 2021.

[7] "AROL Group," [Online]. Available: https://www.arol-group.com/. [Accessed 20 3 2023].

[8] "IntellJ IDEA," [Online]. Available: https://en.wikipedia.org/wiki/IntelliJ_IDEA. [Accessed 20 3 2023].

[9] "IntelliJ IDEA overview," [Online]. Available: https://www.jetbrains.com/help/idea/discover-intellij-idea.html. [Accessed 20 3 2023].

[10] "Getting started with WebStorm," [Online]. Available: https://www.jetbrains.com/help/webstorm/getting-started-with-webstorm.html. [Accessed 20 3 2023].

[11] "Docker (software)," [Online]. Available: https://en.wikipedia.org/wiki/Docker_(software). [Accessed 20 3 2023].

[12] "Docker overview," [Online]. Available: https://docs.docker.com/get-started/overview/. [Accessed 20 3 2023].

[13] "Localstack: AWS on your laptop," [Online]. Available: https://www.kloia.com/blog/localstack-aws-on-your-laptop. [Accessed 20 3 2023].

[14] "Localstack basics," [Online]. Available: https://docs.localstack.cloud/contributing/basics/. [Accessed 20 3 2023].

[15] "What is the AWS Management Console," [Online]. Available: https://docs.aws.amazon.com/awsconsolehelpdocs/latest/gsg/learn-whats-new.html. [Accessed 20 3 2023].

[16] "About JavaScript," [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript. [Accessed 20 3 2023].

[17] "JavaScript," [Online]. Available: https://en.wikipedia.org/wiki/JavaScript. [Accessed 20 3 2023].

[18] "Usage statistics of JavaScript as client-side programming language on websites," [Online]. Available: https://wayback.archive-it.org/all/20220213043439/https:/w3techs.com/technologies/details/cp-javascript. [Accessed 13 2 2023].

[19] "TypeScript," [Online]. Available: https://en.wikipedia.org/wiki/TypeScript. [Accessed 20 3 2023].

[20] "Three-tier architecture overview," [Online]. Available: https://docs.aws.amazon.com/whitepapers/latest/serverless-multi-tier-architectures-api-gateway-lambda/three-tier-architecture-overview.html. [Accessed 20 3 2023].

[21] "What is three-tier architecture," [Online]. Available: https://www.ibm.com/topics/three-tier-architecture. [Accessed 20 3 2023].

[22] "3-tier application architecture," [Online]. Available: https://www.techtarget.com/searchsoftwarequality/definition/3-tier-application. [Accessed 20 3 2023].

[23] M. Richards, Software architecture patterns, O'Reilly Media, 2015.

[24] "React (JavaScript library)," [Online]. Available: https://en.wikipedia.org/wiki/React_(JavaScript_library). [Accessed 20 3 2023].

[25] "Getting Started with ChakraUI," [Online]. Available: https://chakra-ui.com/getting-started. [Accessed 20 3 2023].

[26] "React router overview," [Online]. Available: https://reactrouter.com/en/main/start/overview. [Accessed 20 3 2023].

[27] "Chonky Description," [Online]. Available: https://chonky.io/docs/2.x/. [Accessed 20 3 2023].

[28] "LeafletJS Overview," [Online]. Available: https://leafletjs.com/index.html. [Accessed 20 3 2023].

[29] "Express.js," [Online]. Available: https://en.wikipedia.org/wiki/Express.js. [Accessed 20 3 2023].

[30] "Node.js," [Online]. Available: https://en.wikipedia.org/wiki/Node.js. [Accessed 20 3 2023].

[31] "Express," [Online]. Available: https://expressjs.com/. [Accessed 20 3 2023].

[32] "AWS SDK for JavaScript," [Online]. Available: https://aws.amazon.com/sdk-for-javascript/. [Accessed 20 3 2023].

[33] "What is PostgreSQL," [Online]. Available: https://www.postgresqltutorial.com/postgresql-getting-started/what-is-postgresql/.

[34] "A Deep Dive into Amazon Timestream," [Online]. Available: https://manta-innovations.co.uk/2020/11/20/A_deep_dive_into_Timestream/. [Accessed 20 3 2023].

[35] "Amazon S3," [Online]. Available: https://aws.amazon.com/s3/. [Accessed 20 3 2023].

[36] "Programming 101: File Structures," [Online]. Available: https://medium.com/@deshayk/programming-101-file-structures-2e4699ac0fc2#:~:text=A%20key%20advantage%20of%20file,code%2C%20and%20locate%20associated%20results.. [Accessed 20 3 2023].

[37] "File Structure," [Online]. Available: https://legacy.reactjs.org/docs/faq-structure.html. [Accessed 20 3 2023].

[38] P. Eeles, "Layering Strategies".

[39] "5 User Authentication Methods that Can Prevent the Next Breach," [Online]. Available: https://www.idrnd.ai/5-authentication-methods-that-can-prevent-the-next-breach/. [Accessed 20 3 2023].

[40] W. Zikai and W. Sun, "Review of Web Authentication," 2020.

[41] "JWT parts image," [Online]. Available: https://dev-to-uploads.s3.amazonaws.com/uploads/articles/2dqiq3itw8z649wxmilt.png. [Accessed 20 3 2023].

[42] "JWT destructured image," [Online]. Available: https://miro.medium.com/v2/resize:fit:1200/1*aAH0mMomx1dLidhoNCVmNw.png . [Accessed 20 3 2023].

[43] "JWT authentication flow image," [Online]. Available: https://bezkoder.com/wp-content/uploads/2019/10/spring-boot-authentication-jwt-spring-security-flow.png. [Accessed 20 3 2023].

[44] "JWT token refresh flow image," [Online]. Available: https://www.bezkoder.com/wp-content/uploads/2021/04/spring-boot-refresh-token-jwt-example-flow.png. [Accessed 20 3 2023].

[45] R. Garett, J. Chiu, L. Zhang and S. D. Young, "A Literature Review: Website Design and User Engagement," 2017.

[46] "User Interface Design patterns," [Online]. Available: https://ui-patterns.com/. [Accessed 20 3 2023].

[47] "JavaScript memory management," [Online]. Available: https://felixgerschau.com/javascript-memory-management. [Accessed 20 3 2023].