POLITECNICO DI TORINO

Master's Degree in Computer Engineering

Master's Degree Thesis

Data management system enabling digital twin for resistance spot welding

Supervisors

Prof. Giulia BRUNO

Prof. Manuela DE MADDIS

Dr. Gabriel ANTAL

Dr. Emiliano TRAINI

Candidate

Andrea CENCIO

October 2024

Abstract

In today's rapidly evolving manufacturing processes, the digital twin is revolutionizing industrial processes through real-time data-driven insights. Resistance Spot Welding (RSW), a commonly used joining technique in the automotive and aerospace industries, is a key area where digital twin technology holds great promises for improving quality, efficiency, and predictive maintenance. However, the integration of digital twin models with RSW requires robust data management systems, capable of handling large volumes of data.

This thesis presents the design and implementation of a data management system (DMS) that enables the development and operation of a digital twin for Resistance Spot Welding. The proposed DMS addresses the critical challenges in data collection, processing, storage, and analysis by incorporating modern data architecture principles with advanced analytics. A scalable and flexible architecture is developed, ensuring that the system can handle and process the data generated by sensors and monitoring equipment during the RSW process.

The system's data collection framework integrates a wide range of personalizations that capture key parameters, such as welding current, electrode force, temperature, and results of the welding. This real-time data are then processed using modern techniques, like edge computing, to minimize latency and ensure the data's availability for nearly immediate feedback in the digital twin environment. The processed data are then stored in a microservices-based infrastructure, allowing for centralized management, long-term storage, and easy access for future analysis.

One of the key point of this thesis is the unlock of the possibility to integrate a predictive model in the framework within the digital twin that uses historical data to forecast potential defects in the welding process. A Machine learning algorithm is actually employed to predict outcomes such as weld quality, joint strength, and the likelihood of equipment failure. These predictions enable proactive adjustments to the welding process, reducing defects, optimizing production, and extending equipment life.

The system also incorporates a visualization tool, providing operators with a modern user-friendly interface to monitor welding performances and historical trends of the welding process. By integrating this tool into the system, it facilitates real-time decision-making and supports predictive maintenance strategies, ultimately leading to improved production efficiency and reduced downtime due to machine failures.

The results of this thesis demonstrate the effectiveness of the proposed data management system in enabling a functional digital twin for Resistance Spot Welding. Furthermore, the scalability of the system is validated by its ability to

handle increasing data loads as production environments become more complex and sensor networks expand.

This thesis contributes to the growing body of knowledge in digital twin technology by offering a comprehensive data management solution tailored for high-frequency, high-volume industrial processes such as RSW. Starting from these points, future works can focus on refining the predictive models, integrating additional processing methods, and exploring the system's application to other welding techniques and manufacturing processes. Ultimately, the data management system developed in this thesis paves the way for the broader adoption of digital twins in advanced manufacturing, contributing to smarter and more efficient production lines.

Summary

This thesis explores the development of a data management system with the purpose of enabling a digital twin of the resistance spot welding (RSW) process. The primary aim is to enhance the efficiency, quality control, and predictive maintenance of RSW operations by integrating advanced data management techniques with digital twin technology.

- Theory Pills: The first chapter discusses about some theory pills, to better help the understanding of the whole development. It expose the challenges managing and retrieving data from the welding process to ensure consistent quality.
- Methodology: The methodology section outlines the development of the data management system. It involves several key components:
  - Data Acquisition: Describes the sensors and data collection methods used to gather real-time data from the welding process.
  - Data Storage: Explains the database architecture designed to store large volumes of heterogeneous data securely and efficiently.
  - Data Processing: Details the algorithms and computational techniques used to process raw data from the machinery into comprehensible charts.
  - Integration with Digital Twin: Discusses how the processed data will be integrated into the digital twin model to enable real-time simulation and analysis.
- System Architecture: The system architecture chapter provides a detailed description of the overall system design. It includes graphics and explanations of the various modules and their interactions. Emphasis is placed on the scalability, flexibility, and robustness of the system to handle diverse data types and volumes.
- Implementation and Testing: This section covers the practical implementation of the data management system. It describes the setup of the experimental

test machine and the data collection process. Various tests, starting from real data acquisition and already validated results, are conducted to validate the accuracy and reliability of the data management system and its integration with the digital twin.

- Results and Conclusion: The results section presents the findings from the implementation and testing phase. It showcases how the whole storage data management system, developed in this way, can be integrated with a digital twin to predict welding outcomes, detect anomalies, and suggest corrective actions. The discussion delves into the implications of these results for the manufacturing industry, particularly in improving product quality and operational efficiency. The conclusion instead summarizes the key contributions of the thesis. It emphasizes the successful development of a data management system that can enable an effective digital twin for resistance spot welding. The potential benefits for industry applications, such as reduced downtime, enhanced quality control, and predictive maintenance, are highlighted. The thesis also outlines areas for future research, including the expansion of the system to other advanced AI model, other welding processes and further refinement of predictive algorithms.

- References: A comprehensive list of references is provided, citing all sources used in the research, including academic papers, industry reports, and technical manuals relevant to digital twins, data management systems and resistance spot welding.

Acknowledgements

To all those who said I wouldn't make it.

Table of Contents

List of Figures

Listings

Chapter 1

Theory pills

1.1 Resistance Spot Welding

1.1.1 Introduction

Resistance Spot Welding (RSW) is a welding process[1] commonly used in the automotive, aerospace and electronics industries. The key advantage of this technique is its ability to join thin metal sheets quickly and efficiently, allowing a faster high-volume production.

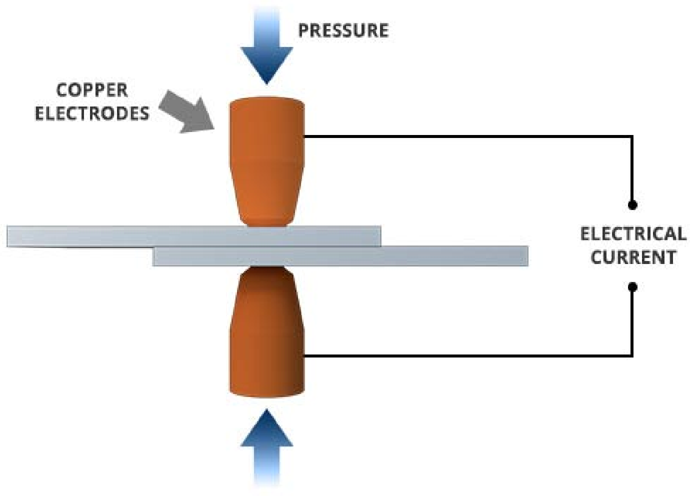


Figure 1.1: Scheme of Resistance Spot Welding process[2]

Resistance Spot Welding is a type of welding through a fusion process where two or more metal sheets are joined by applying pressure and a flow of current through the materials at the joint (see Fig 1.1). Localized melting takes place due to the heat generated from resistance along interface of both materials, and as weld nugget is formed.

1.1.2 RSW in detail

The key components of a Resistance Spot Welding system are:

- Electrodes, that are typically made of copper, the electrodes conduct electricity and apply pressure to the workpieces.
- Power Supply that provides the electrical current required for the welding process.
- Control System, that regulates the timing, current and pressure to ensure consistent weld quality.

The heat generated during the Resistance Spot Welding process can be described by Joule's law:

$$Q = I^2Rt$$

In which:

- I is the current passing through the materials
- R is the electrical resistance at the joint,
- t is the time for which the current is applied.

1.1.3 The Physical Transformations

The electrical resistance, in fact, is a critical factor in determining the amount of heat generated and is influenced by the material properties, surface condition and electrode geometry. The heat must be sufficient to melt the metal at the interface but not too much, continuously controlled by sensor and algorithms to prevent excessive expulsion of molten material.

The Resistance Spot Welding process involves rapid consequential heating and cooling, which significantly impacts on the microstructure of the welded materials.

The key metallurgical phenomena include:

- Nugget Formation: The weld nugget is the small, roughly spherical region of molten metal that solidifies to form the joint.

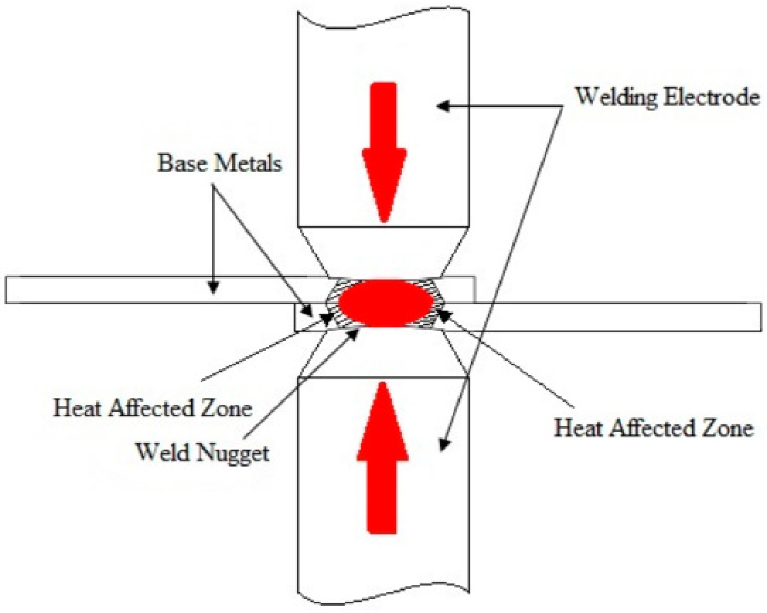


Figure 1.2: Detail of RSW process[3]

- Heat-Affected Zone (HAZ): The area surrounding the weld nugget, where the microstructure has been altered by the heat but has not melted.
- Phase Transformations: Depending on the material, phase transformations may occur during the cooling process, influencing the mechanical properties of the joint.

Quality of weld in RSW is affected by several parameters and they will be discussed now:

- Welding Current, because higher current increases the heat generated, influencing the size and strength of the weld nugget.
- Welding Time, because the duration of current flow must be optimized to ensure proper nugget formation without excessive melting or expulsion.
- Electrode Force, because the pressure applied by the electrodes affects the contact resistance and the heat distribution across the joint.
- Material Properties, because the electrical and thermal conductivity, as well as the thickness and surface condition of the workpieces, play crucial roles in determining the weld quality.

The optimization of these parameters is critical to guarantee consistent weld quality and minimizing defects.

1.1.4 Present and Future of RSW

Recent innovations in sensor technology build material and control algorithms have led to some significant improvements in the monitoring and control of the Resistance Spot Welding process. In fact real-time feedback systems, incorporating sensors for current, voltage and displacement, let be possible the dynamic adjustment of welding parameters to compensate for variations in material properties or external conditions.

With the usage even more common of advanced high-strength steels (AHSS), aluminum alloys and composites in industries like automotive and aerospace, Resistance Spot Welding techniques have evolved to deal the challenges of welding these materials. Innovations like adaptive control systems and new electrode designs have been developed to improve the weldability of materials with varying thermal and electrical conductivities.

Hybrid welding techniques [4] applied by combining Resistance Spot Welding with other welding or bonding methods, have been developed to improve joint strength and durability, especially in multi-material assemblies. For example, Resistance Spot Welding combined with adhesive bonding can provide improved mechanical properties and corrosion resistance in automotive structures.

Advances in computational modeling have enabled more accurate predictions of the RSW process, including heat generation, nugget formation and residual stresses. Finite Element Analysis (FEA) and other simulation tools are increasingly used to optimize welding parameters, reduce defects and design more robust welding systems.

The demand for more sustainable manufacturing processes has led to innovations aimed at reducing the energy consumption of RSW. New power supply designs, optimized welding schedules and the development of more efficient electrode materials are among the approaches being pursued to make RSW more environmentally friendly.

1.1.5 To Sum Up

Resistance Spot Welding remains a cornerstone technology in modern manufacturing, with ongoing research and development focused on enhancing its efficiency, adaptability and applicability to new materials. The integration of advanced monitoring systems, computational modeling and hybrid techniques are driving the evolution of RSW, ensuring its continued relevance in the face of emerging manufacturing challenges.

1.2 Digital Twin

1.2.1 Introduction

The concept of Digital Twin has gained significant interest in recent years, due to the improvements in digital technologies, Internet of Things (IoT) and artificial intelligence (AI). A Digital Twin is a virtual representation of a physical entity or system, continuously fed with real-time data updates and capable of simulating, predicting and optimizing the performance of its real-world counterpart[5].

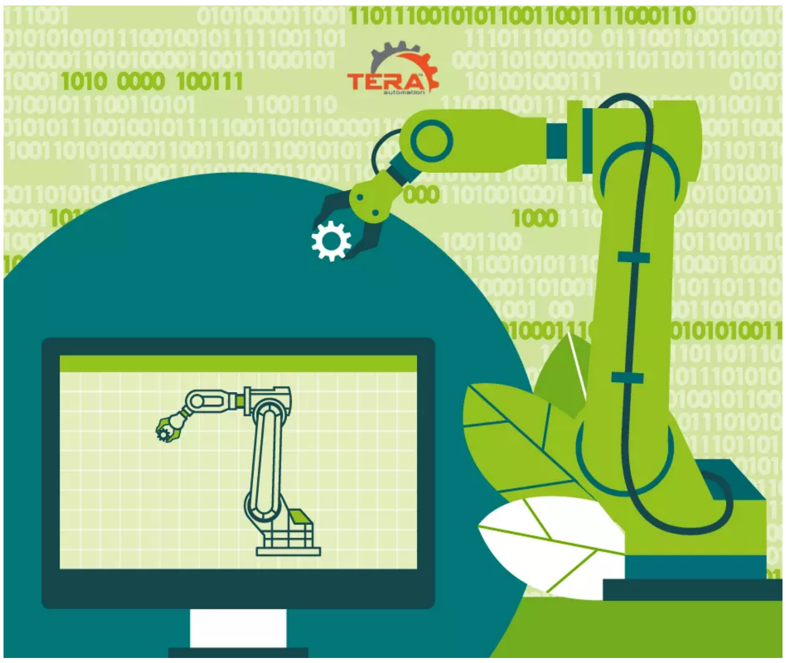


Figure 1.3: High Level example of a Digital Twin[6]

1.2.2 Theoretical Foundations of Digital Twin

The term "Digital Twin" was first coined by Dr. Michael Grieves in 2003 during a presentation on Product Lifecycle Management (PLM). At first the concept was focused on creating virtual models of products to simulate their performance throughout their entire lifecycle. Over the time, the idea has expanded to include entire systems, their manufacturing processes, smart cities, healthcare devices and more.

A Digital Twin can be defined as a dynamic, digital representation of a physical object, system, or process. It integrates data from various sources, including sensors, historical records and predictive models, to provide insights into the current state, predict future behavior and optimize performance.

1.2.3 Digital Twin's components

A fully functional Digital Twin typically consists of the following key components:

- Physical Entity: it is the real-world object, system, or process being represented by the Digital Twin.
- Virtual Model: it is the digital counterpart that mirrors all the physical entity, including its geometry, behavior and functional aspects.
- Data Integration: it is the flow of real-time data from sensors and other sources into the virtual model that reflects the current state of the physical entity.
- Analytics and Simulations: they are tools and algorithms used to analyze the data, run simulations and make predictions about the future behavior of the physical entity.
- Communication Interface: that is a platform for interaction between the physical and digital twins, allowing for real-time feedback, control and optimization.

1.2.4 Digital Twin typologies

Digital Twins can vary in complexity and functionality, often categorized into different levels of maturity:

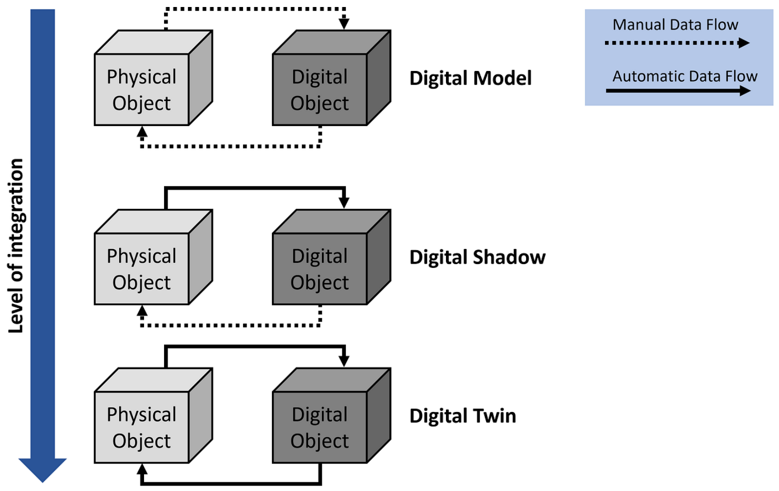


Figure 1.4: Different Typologies of DTs[7]

- The most simplest one is the Digital Modelthat is a static digital representation of a physical entity with no real-time data integration with it.
- Then it comes the Digital Shadow: it is A digital representation updated with real-time data, but with limited interaction or feedback to the physical entity.
- Finally, the Digital Twin: a fully integrated model with bidirectional communication, real-time updates and the ability to influence the physical entity's operations.

1.2.5 Fields of application

In Industry 4.0 Digital Twins are an angular stone for smart manufacturing. They enable real-time monitoring, predictive maintenance and optimization of production processes, letting the possibility to reduce downtime and enhance efficiency. For instance, a Digital Twin of a factory floor can simulate the impact of changes in production schedules, machine configurations, or material flows, allowing managers to make informed decisions about the lifetime cycle of the product.

Digital Twins are increasingly used year by year in urban planning and smart city initiatives. By creating a virtual model of a city, urban planners can simulate the effects of infrastructure projects, traffic management strategies or environmental changes. In healthcare, Digital Twins are used to model individual patients, allowing for personalized treatment plans and predictive healthcare, but also the aerospace industry has been a pioneer in adopting Digital Twins for the design, testing and maintenance of aircraft and spacecraft. Finally, digital Twins are used in the energy sector to model power plants, grids and renewable energy sources.

1.2.6 Way of representation in CE Solutions

The backbone of a Digital Twin is the integration of data from the physical world into its digital counterpart. In computer engineering this is achieved through IoT devices and sensors that continuously collect and transmit data to the digital model. These sensors monitor various parameters such as temperature, pressure, vibration and more, providing a real-time view of the physical entity's state.

IoT platforms and middlewares play a crucial role in aggregating, then processing and finally managing the vast amounts of data generated. These platforms ensure the data is accurately captured, then filtered and finally forwarded to the Digital Twin model for further analysis.

Digital Twins often require significant computational resources for data processing then run simulations and then generate predictions. Cloud computing could offers the scalability and flexibility, allowing Digital Twins to reach powerful computational capabilities without the need for extensive on-premises infrastructure.

For time-sensitive applications (where low latency is critical), edge computing is employed. Edge computing brings computational resources closer to the data source, reducing the delay in data processing and enabling real-time decision-making.

The data ingested by the Digital Twin must be analyzed to generate actionable insights. This is where data analytics, machine learning (ML) and more generally artificial intelligence (AI) come into play. These technologies enable the Digital Twin to learn from historical data or recognize patterns and make predictions about future states or failures.

For example in predictive maintenance, one of the possible field of application of this thesis, machine learning algorithms can analyze sensor data to predict when a machine part is likely to fail, allowing for proactive replacement before a breakdown occurs. AI-driven optimization algorithms can also be used to adjust operations in real-time to improve efficiency and reduce costs.

1.2.7 Simulation

Simulation is a core component of Digital Twin technology[8], enabling the virtual model to mimic the behavior of its physical counterpart. In computer engineering, various simulation tools and platforms are used to create and run these models:

- Finite Element Analysis (FEA): Used for simulating the physical behavior of objects under stress, heat and other forces.

- Computational Fluid Dynamics (CFD): Used to simulate fluid flow, which is essential in industries like aerospace, automotive and energy.

- Discrete Event Simulation (DES): Used for modeling the operation of complex systems, such as manufacturing processes or logistics networks.

These tools are often integrated into the Digital Twin framework, allowing engineers to test scenarios, validate designs and predict outcomes in a virtual environment before applying them in the real world.

The effectiveness of a Digital Twin relies heavily on how its data and insights are presented to users. Graphical interactive dashboards, 3D visualizations and virtual/augmented reality (VR/AR) interfaces are commonly used to represent Digital Twins.

- 3D Visualization: Tools like Unity, Unreal Engine and custom 3D engines are used to create detailed visual representations of physical entities, enabling users to interact with the Digital Twin in an intuitive way.

- Virtual and Augmented Reality: VR/AR technologies are increasingly used in Digital Twin applications, especially for training, remote maintenance and collaborative design. These technologies allow users to immerse themselves in the virtual environment and interact with the Digital Twin as if they were handling the real object.

- Interactive Dashboards: Platforms like Tableau, Power BI and custom-built dashboards (like Streamlit in our case) are employed to visualize real-time data, key performance indicators (KPIs) and simulation results. These dashboards provide a comprehensive view of the Digital Twin's status, helping users make informed decisions quickly.

Effective communication between the physical and digital twins is essential for the system's functionality. Protocols like MQTT, OPC UA and RESTful APIs are commonly used to ensure flawless data exchange. Additionally the security of this data, especially in critical infrastructure or sensitive applications, is predominant.

In computer engineering security measures such as encryption, authentication and access control are implemented to protect the integrity and confidentiality of the data. Blockchain technology is also being explored as a way to ensure secure and immutable data exchange between digital twins, particularly in distributed systems.

1.2.8 Challenges

There are some key challenges in Digital Twin construction, such as:

- Data Quality and Integration: Ensuring the accuracy and consistency of data from diverse sources can be challenging, particularly in complex systems with multiple sensors and devices.
- Scalability: Maintaining their performance and scalability becomes a significant challenge, because Digital Twins grow in complexity and scope, particularly in global applications like smart cities or large industrial systems.
- Interoperability: The integration of Digital Twins across different platforms and technologies requires standardized protocols and interfaces, which are still evolving.
- Security and Privacy: The vast amounts of data collected and processed by Digital Twins raise concerns about security and privacy, particularly in sensitive applications like healthcare and critical infrastructure.

1.2.9 The future of Digital Twins

The development of industry standards for Digital Twin implementation will improve interoperability and accelerate adoption across various sectors. The integration of advanced AI techniques will enhance the predictive and prescriptive capabilities of Digital Twins, enabling more autonomous and intelligent systems. As the technology matures, Digital Twins are expected to expand into new domains such as agriculture, education and entertainment, unlocking new opportunities for innovation. The widespread use of Digital Twins, particularly in areas like healthcare and smart cities, will necessitate a greater focus on ethical considerations, including data privacy, transparency and the potential for AI-driven decisions.

1.3 Microservices

1.3.1 Introduction

In last years microservices architecture has emerged as a new methodology to design and develop complex software applications. Unlike traditional and past monolithic architectures, about which we will talk about in the next section, where all components of an application are strongly developed, integrated and distributed as a single unit, microservices break the whole application into a series of small, independent services, each with its own simple specific task. This subdivision offers numerous advantages in terms of scalability, maintainability and development speed, that we will analyze more in deep in the next paragraphs.

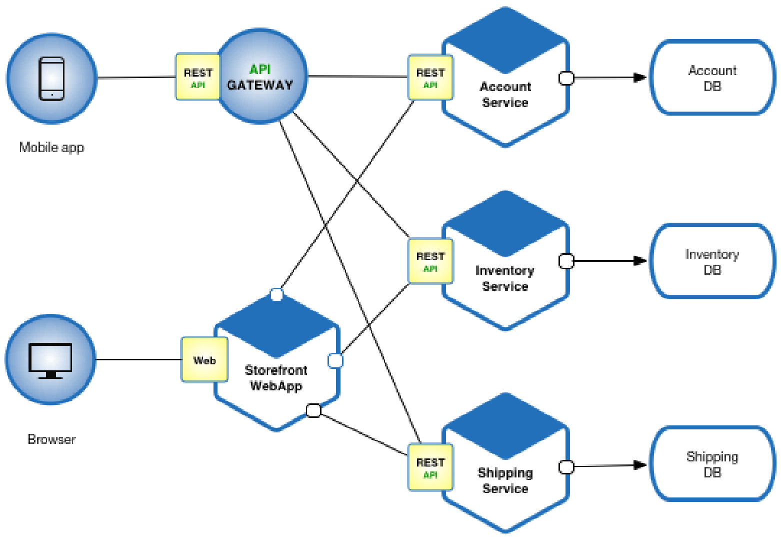


Figure 1.5: Example of Microservices Architecture[9]

1.3.2 Principal Characteristics

Microservices have more than one peculiarity:

- Service Indipendence: every microservice functions as an autonomous application with its own lifecycle[1]. This independence allows for the development, testing, deployment and scaling of services independently from one another.
- Communication via API[2]: microservices communicate with each other through

[1]Every microservice can be developed, tested and released independently

[2]API = Application Programming Interface

well-defined APIs, typically using network protocols like HTTP/HTTPS. This approach facilitates integration and cooperation between different services. The use of RESTful[3] APIs or gRPC (remote procedure calls) for synchronous communication and message brokers for asynchronous communication ensures that services can interact seamlessly, regardless of their underlying technologies. This API-driven communication also makes it easier to expose functionalities to external clients or third-party applications, enhancing the system's extensibility in the future steps of development.

- Heterogeneous Technologies: microservices are not limited to a single technology stack or programming language and so it allows them to be developed using the most appropriate tools for each specific requirement[4]. For instance, one microservice might be written in Python due to its simplicity and extensive library support, while another might be in Java for its robustness and performance in handling concurrent transactions. This heterogeneity enables organizations to leverage the best features of different technologies, optimizing each microservice for its intended purpose. Furthermore, this flexibility allows teams to adopt new technologies and methodologies without needing to overhaul the entire system, promoting innovation and technological advancement.
- Failure Isolation: failures in microservices are mostly isolated to individual services, reducing the impact on the overall system. This isolation improves the system's resilience and reliability. For example, if a payment processing microservice encounters an issue, it won't affect the user authentication or product catalog services. This segregation of services ensures that the failure of one does not cascade into a systemic failure, enhancing the stability and robustness of the application. Techniques like circuit breakers and bulkheads can be implemented to manage and contain failures effectively within microservices.
- Scalability: microservices enable scaling of only those parts of the application that require more resources, improving operational efficiency and optimizing costs. This granular scalability is achieved by deploying instances of specific microservices on demand, based on their load. For example, if an e-commerce application might need to scale its order processing microservice during peak shopping seasons, it can do it without scaling the user management microservice. This targeted scaling ensures that resources are allocated precisely where

[3]REST = REpresentational State Transfer

[4]For example, you can develop a backend using NoSQL db that stores files and a RDBMS for data handling, like in this application

needed, leading to significant cost savings and better performance. Cloud platforms, ad exemple, offer auto-scaling capabilities that make it easier to dynamically adjust the number of instances of a microservice based on real-time demand.

1.3.3 Microservices Advantages

- Deployment Flexibility: Microservices architecture consent for more flexible deployment strategies. Continuous Integration and Continuous Deployment (CI/CD) pipelines can be specifically tailored to the specific needs of each service, enabling more frequent and reliable updates. This flexibility supports rapid iteration and experimentation which are crucial for modern software development practices[5].

- Enhanced Security: By breaking down applications into smaller services, microservices can improve security through isolation. Each service can be secured independently, applying the principle of least privilege[6]. Access controls and security policies can be fine-tuned for each service, avoiding that a security breach can impact on whole system.

- Improved Fault Tolerance: Microservices can improve fault tolerance through redundancy and failover mechanisms. Critical services can be duplicated across multiple nodes or data centers, to guarantee redundancy, ensuring availability even in the event of a hardware failure or network outage. This redundancy is essential for maintaining high availability and ensuring business continuity.

- Organizational Alignment: Microservices are naturally used in agile development practices and organizational structures. Teams can be organized around specific services, fostering a sense of ownership and accountability. This alignment promotes cross-functional collaboration, as each team includes all the necessary skills to develop, deploy and maintain their respective services.

- Cost Management: Enabling more precise resource allocation, microservices help in managing costs more effectively. Resources can be provisioned based on the actual needs of each service, avoiding over-provisioning and reducing wastage. This cost efficiency is particularly beneficial in cloud environments, where resource usage directly impacts billing.

[5]For example, you can work on a microservice and test new improvements without interrupting the whole application

[6]If a payment system have an issue, the customer can still navigate through the site, even though he cannot purchase anything

- Faster Time-to-Market: The independence and parallel development capabilities of microservices shorten the time-to-market for new features and improvements[7]. Organizations can respond more swiftly to market demands and user feedback, maintaining a competitive edge in the fast-paced digital landscape.

1.3.4 Microservices Disadvantages

- Service Coordination: With a large number of microservices, the need for coordination increases. DevOps and developers must manage interactions, dependencies and updates across various services, which can become very complicated.
- Monitoring: Supervising a microservices-based architecture requires advanced tools to monitor the status and performance of each service. Performance issues or failures can be difficult to isolate, there are needed employees with specifical background and knowledge.
- Configuration Management: Each microservice may have its own configuration making then centralized configuration management more complex.
- Latency: Each call between microservices introduces latency. While a single call might not be significant, in a highly interconnected system, cumulative latency can become problematic.
- Points of Failure: The network itself becomes a potential point of failure. Connectivity issues can disrupt the functioning of services. Security: Communication between microservices must be secure. This includes implementing authentication and authorization, encrypting data in transit and managing security keys.
- Consistency Models: In a distributed system maintaining data consistency is complex. Many microservices-based systems opt for eventual consistency, where data may not be immediately consistent across all services.
- Distributed Transactions: The capability to split transactions on multiple services can be very complicated. Developers must implement mechanisms like sagas or two-phase commit (2PC), which can be difficult to manage and can introduce further complexity and latency.

[7]Very often companies allocate more than oone developers to work on the same application, with each developer that have in charge one microservice

- Orchestration Tools: Deploying and managing a large number of microservices requires tools like Kubernetes[8] to orchestrate containers. These tools add another layer of complexity and require specialized skills.
- Continuous Deployment: With many independent services, continuous deployment can become a nightmare if not managed properly. Each service needs to be tested and deployed independently, increasing the risk of errors and inconsistencies[9].
- Operational Overhead: The need to manage multiple code repositories, `CI/CD` pipelines and deployment configurations increases operational overhead.
- Staff Expertise: Implementing and maintaining a microservices architecture requires staff with advanced skills in DevOps, container management, distributed application security and monitoring.
- Debugging and Troubleshooting: Tracing and resolving bugs in a distributed system can be extremely difficult. Distributed logs, call tracing and monitoring metrics become crucial for diagnosing issues.

1.3.5 To Sum Up

Microservices offer many advantages, such as scalability and independent development teams, but they also come with significant challenges. It is essential to carefully evaluate these disadvantages and prepare to address them with the right skills and tools before adopting a microservices architecture.

[8]Very often people interrogate about the differences between Docker and Kubernetes: the fact is that they're not in competition against each other, but they're complementary!

[9]Very often non-structured developers releases microservice code without testing it. In case of fault, this led to a very difficult problem solving, because it is more difficult to track the error between microservices

1.4 Docker

Docker is an open-source platform that automates the deployment, scaling and management of applications within lightweight and portable containers. Containers package an application and its dependencies, providing a consistent environment across various stages of development and deployment[10].

Docker was first released by Docker Inc. in 2013. It pulls out containerization technology, which has been a part of the Linux kernel through *namespaces* and *control groups*. Docker's ease of use and comprehensive tooling made it popular, leading to widespread adoption in both development and production environments.

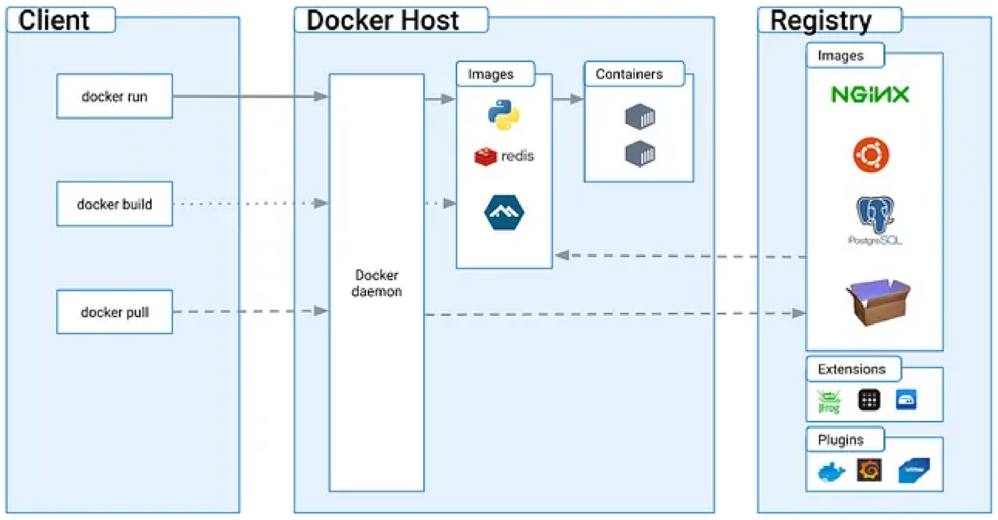


Figure 1.6: Example of Docker Architecture[11]

1.4.1 Docker Principal Characteristics

Docker uses a client-server architecture:

- Docker Client: A command-line interface (CLI) that users interact with.
- Docker Daemon: A background service running on the host that builds, runs and manages Docker containers.
- Docker Images: Pre-built templates used to create different containers.

- Docker Containers: Standalone, lightweight and executable software packages that include everything needed to run a software, including also the code, runtime, system tools, libraries and settings.

1.4.2 Docker Advantages

Docker has some advantages, that will be discussed now[12]:

- Portability: Containers can run on any system that is supported by Docker, ensuring consistent behavior across different environments. It is easy to develop on a platform (for example Windows PCs) and deploy on another (for example Linux servers)
- Efficiency: Containers use the same host system's kernel and resources, which results in lower overhead compared to virtual machines, that have to replicate all layers.
- Isolation: Containers have been developed to isolate applications and their dependencies from the other containers and the host machine, ensuring that they do not interfere with each other and that the communication between them is only possible through user-defined bridges.
- Scalability: Docker integrates with orchestration tools like Kubernetes, making it easier to scale applications horizontally.
- Rapid Deployment: Containers can be started and stopped quickly, which facilitates rapid development, testing and deployment cycles[10].
- Version Control: Docker images can be versioned, providing a clear history of changes and the ability to roll back to previous versions if it is wanted to.
- Simplified Dependency Management: Containers include all necessary dependencies, reducing issues related to environment configuration.

1.4.3 Docker Disadvantages

On the contrary, it has also some disadvantages[13]:

- Security Concerns: Since containers share the host OS kernel, a security vulnerability in the kernel can potentially affect all containers.

[10]It is also possible to switch off some containers while problem debugging, letting others running flawless

- Complexity in Orchestration: Managing multiple containers in production requires orchestration tools like Kubernetes, which can add complexity. The simple use of Docker could not be fully enough.
- Performance Overhead: Although lighter than virtual machines, containers still introduce some overhead compared to running applications directly on the host.
- Persistent Storage: The configuration to have the possibility of managing persistent data in containers has to be carefully tied up, especially in a distributed environment.
- Networking Overhead: Container networking introduces additional layers of abstraction that can impact network performance.
- Limited Compatibility with Non-Linux Systems: Docker is primarily designed for Linux and while it supports Windows and macOS, some features may not be fully compatible or sharing the same performances.
- Learning Curve: For teams new to containerization, there is a significant learning curve associated with Docker and its ecosystem[11].

1.4.4 To Sum Up

Docker has revolutionized the way applications are developed by providing a consistent, portable and efficient environment[11]. Its advantages have made it a vital tool in modern software development. However, challenges such as security concerns, orchestration complexity and performance overhead must be carefully managed to fully have a positive return on Docker's capabilities. Understanding these trade-offs is crucial for effectively integrating Docker into a computer engineering workflow.

[11]Expecially while connecting containers between each other

1.5 Monolithic Architecture

1.5.1 Introduction

The monolithic architecture, in contrast with the newer and modern microservices approach, represent the most traditional approach in software development. This model is based on the theory about the fact that all application components (User UI, business logic, data access and so on) are integrated in a unique block, not separable. This typo of software architecture development is widely used in many sectors, due to his simplicity and his easy development. it has to be remember that, despite being and old conception of architecture, it is not necessarily bad, compared to microservices one.

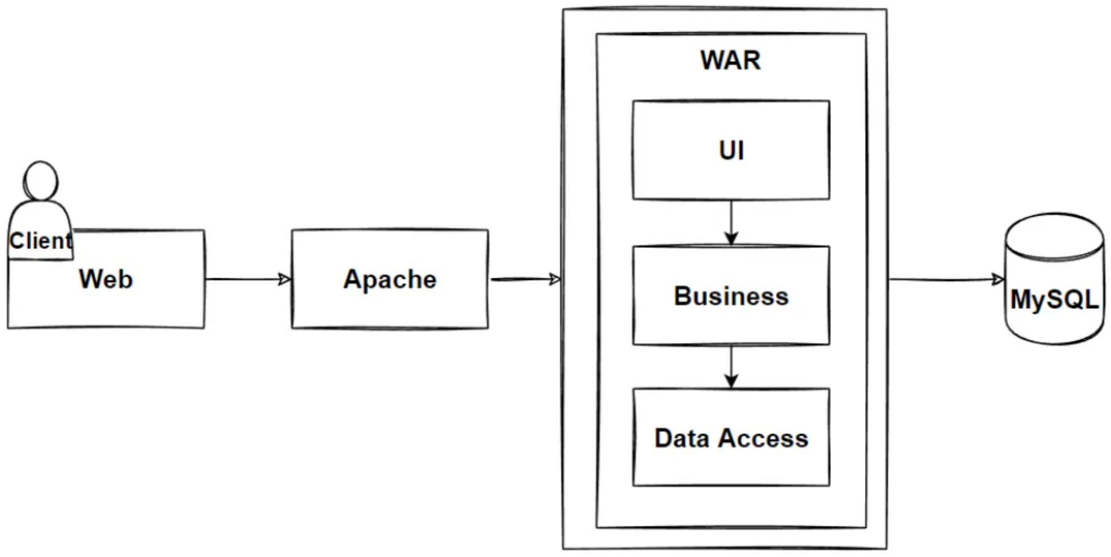


Figure 1.7: Example of Monolithic Architecture[14]

1.5.2 Monolithic Advantages

- Simplicity of Development and Debugging: one of the main advantages of monolithic architecture is its simplicity. All the application code resides in a single project, making it easier for developers to understand the entire system. Debugging and testing a monolithic application can be simpler because all components are present in the same memory space and can be tested together.

- Improved Performance: monolithic applications can offer better performance compared to distributed architectures (like microservices) as they eliminate the network latency associated with calls between separate services. This is particularly beneficial for applications that require high response speed.

- Easy Deployment: with a monolithic architecture, the deployment process is generally simpler. The entire application is developed and then mantained as a single unit, reducing the complexity of the deployment process compared to microservice-based applications, which require the coordination and deployment of multiple independent services.
- Centralized Management: having a single code base centralizes dependency and configuration management. This makes it easier to maintain and update third-party libraries used in the project reducing the risk of incompatibilities.
- Lower Initial Costs: for startups and small businesses, a monolithic architecture can be a more economical choice. The time and resources needed to develop and manage a monolithic application are often less than those required for a microservices-based architecture.
- Compatibility with Existing Frameworks: many development frameworks are designed with a monolithic architecture in mind. This means developers can leverage a wide range of mature tools and libraries, accelerating the development process.
- Ease of Vertical Scalability: scaling a monolithic application can be simpler in some scenarios, as it merely requires increasing the server resources (CPU, RAM). This approach is straightforward and can be quickly implemented to handle a temporary increase in load.

1.5.3 Monolithic Disadvantages

- Vertical Scalability Limits: Monolithic applications primarily rely on vertical scaling, which involves adding more resources (CPU, RAM) to a single server. This approach has physical and cost limitations, making it difficult to scale beyond a certain point.
- Inefficiency in Horizontal Scaling: Horizontal scaling, which involves adding more servers, is challenging with monolithic architecture because it requires the entire application to be duplicated across servers, leading to inefficiencies. Development and Deployment Challenges
- Slower Development Cycles: As the application grows, the codebase can become large and complex, making it harder for developers to work on and understand. This can slow down development and lead to longer release cycles.

- Deployment Bottlenecks: Any change in the code, even a minor one, requires the entire application to be redeployed. This can cause downtime and increase the risk of deployment errors, affecting the entire system.
- Codebase Complexity: Over time, a monolithic application can become cumbersome and difficult to manage due to tightly coupled components. Changes in one part of the application can inadvertently affect other parts, leading to increased maintenance efforts.
- Technology Lock-In: Monolithic architectures can be difficult to evolve with new technologies. Introducing new frameworks or languages may require significant rewrites of the entire application, which is both time-consuming and risky.
- Single Point of Failure[12]: A failure in any part of the monolithic application can potentially bring down the entire system. This lack of fault isolation can lead to significant downtime and impact overall reliability.
- Difficulties in Fault Isolation: Identifying and isolating issues can be challenging because all components are interdependent. This can complicate troubleshooting and prolong the time needed to resolve issues.
- Team Scaling Challenges: As the team grows, coordination becomes more complex. Multiple developers working on the same codebase can lead to merge conflicts, duplicated efforts and difficulties in maintaining code quality.
- Limited Parallel Development: Monolithic architecture limits the ability of teams to work on different features or components independently because changes in one area often require coordination and testing across the entire application.
- Uniform Technology Stack: Monolithic applications often use a single technology stack, which can limit innovation. Different parts of the application may have different requirements and being restricted to a single technology stack can be suboptimal.
- Comprehensive Testing Requirements: Testing a monolithic application requires comprehensive testing of the entire system, even for small changes. This can be time-consuming and resource-intensive.
- Difficulty in Automated Testing: Automation of tests becomes more complex as the application grows, requiring extensive setup and maintenance of test environments.

[12]A breach in a service can broke down the entire application

1.5.4 To Sum Up

Despite the growing popularity of microservice architectures, monolithic architecture remains a valid choice for many applications, in particular for small and medium-sized projects. Its simplicity, superior performance in certain contexts and ease of management and deployment make it a pragmatic and effective solution. By the way it is important to carefully consider the specific needs of the project and potential future challenges related to scalability and maintenance before adopting a monolithic architecture.

1.6 Relational Databases

1.6.1 Introduction

A relational database is a type of database that organizes data into interrelated tables. This model, introduced by Edgar F. Codd in the 1970s, uses a column and row structure to facilitate data management and manipulation. Each table, also known as a relation, represents an entity and each row (or record) represents an instance of that entity. The columns (or fields) define the attributes of the entities.

The key concept of relational databases is the usage of primary keys and foreign keys to establish and maintain relationships between tables[15]. A primary key is a unique identifier for each record in a table, while a foreign key is a field that creates a link between two tables.

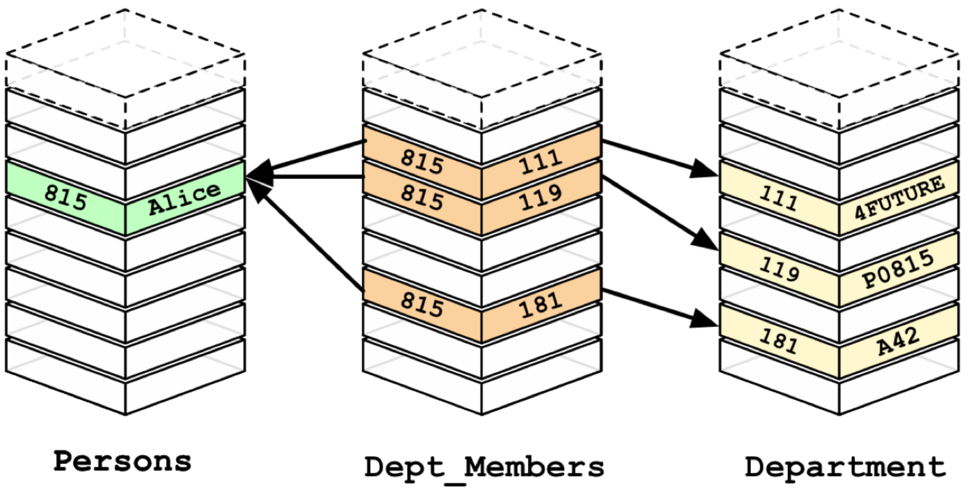


Figure 1.8: Example of Relational Database JOIN with Foreign Keys[16]

1.6.2 Relational Databases Advantages

- Data Integrity: The use of primary keys and foreign keys ensures data consistency and integrity.
- Usability: Insert, update and delete operations are simple thanks to the SQL[13] language.

[13]SQL = Structured Query Language

- Scalability: Relational databases can handle large volumes of data and users.
- Transactionality: Support ACID (Atomicity, Consistency, Isolation, Durability) transactions which ensure reliable operations.

1.6.3 PostgreSQL

PostgreSQL is an open-source relational database management system (RDBMS[14]), known for its stability, extensibility and compliance with SQL standards. Initially developed as part of the POSTGRES project at the University of California, Berkeley, it has become one of the most advanced RDBMS systems available.

1.6.4 Principal Characteristics of PostgreSQL

- Standards Compliance: PostgreSQL complies with ANSI SQL[15] standards, making it compatible with a wide range of applications and tools.
- Extensibility: Supports the addition of new functions, data types, operators and procedural languages.
- Support for ACID Transactions: Ensures that all database operations are atomic, consistent, isolated and durable.
- Advanced Index Management: Supports various types of indexes such as B-tree, hash, GiST, SP-GiST, GIN and BRIN, improving query performance.
- Security: Offers robust authentication, authorization and data encryption functions.

1.6.5 PostgreSQL Architecture

The PostgreSQL architecture is made up of several key components:

- Server Process (Postmaster): The main process that manages client connections, query execution and transaction control.
- Shared Buffers: Shared memory area used to store temporary data and table pages.
- Write-Ahead Logging (WAL): Log system that guarantees the durability of transactions[16].
- Background Processes: Auxiliary processes that perform maintenance tasks such as checkpointing, garbage collection and replication.

[14]RDBMS = Relational DataBase Management System

[15]SQL = Structured Query Language

[16]The changes are first written in logs. If the changing movement has success, it will be propagated through the real database

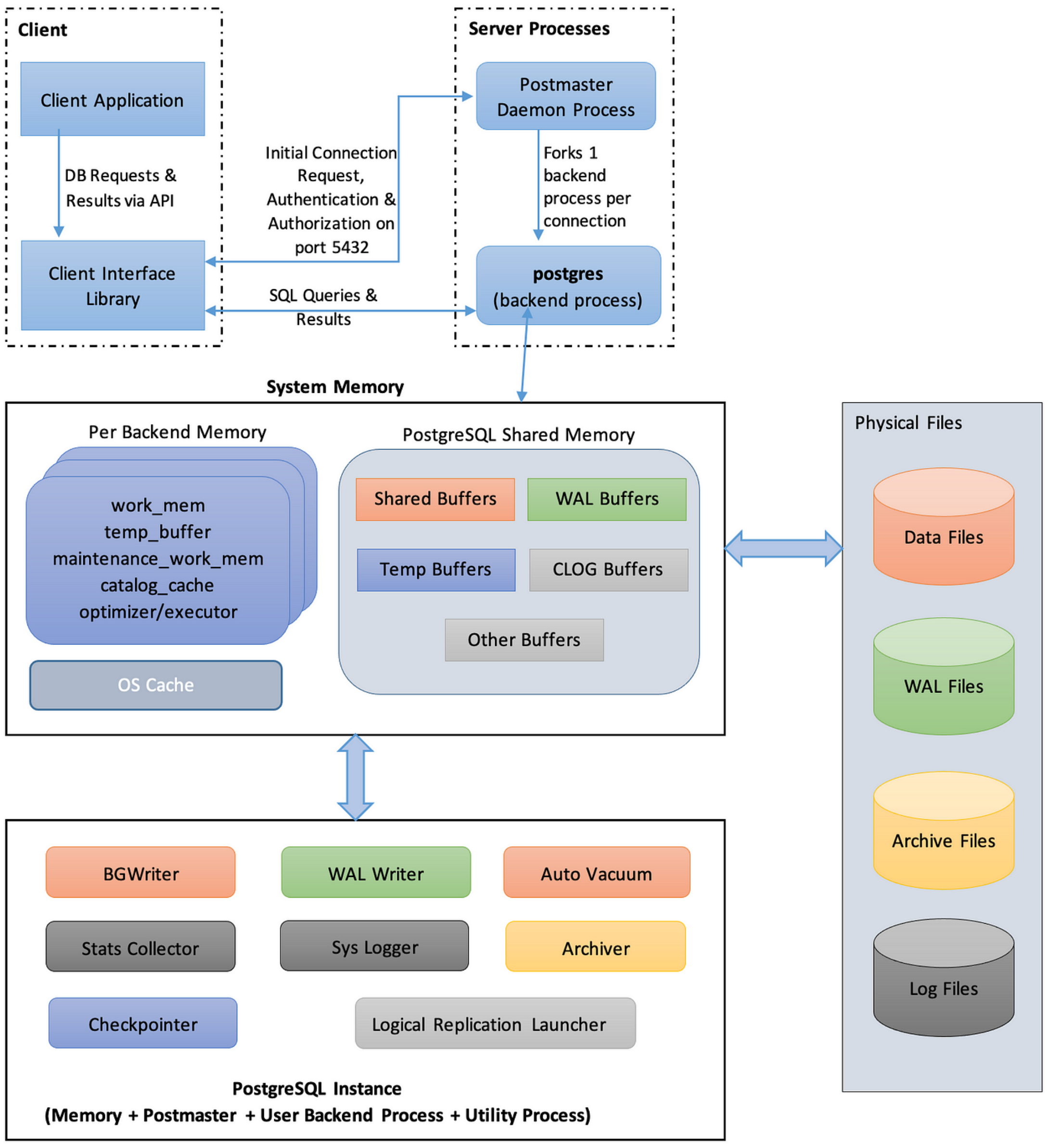


Figure 1.9: Example of Postgres Architecture[17]

PostgreSQL supports a wide range of data types, including strings, numbers, dates, arrays and custom types. Data management is based on CRUD (Create, Read, Update, Delete) operations performed via SQL.

1.6.6 PostgreSQL Advantage Functionalities

- Functions and Procedures: PostgreSQL supports writing functions and procedures in various languages such as pgSQL, Tcl, Perl and Python.
- Advanced Indexes: In addition to B-tree indexes, it supports GIN indexes for full-text search and GiST indexes for spatial data.
- Table Partitioning: Allows you to split large tables into smaller sub-tables to improve query performance.
- Replication and Clustering: Offers synchronous and asynchronous replication mechanisms for fault tolerance and load balancing.

1.6.7 To Sum Up

PostgreSQL is a robust and versatile choice for data management, combining advanced features with a strong emphasis on standards compliance and extensibility. Its active community and extensive documentation make PostgreSQL one of the most reliable and sustainable solutions for modern database needs.

1.7 Non Relational Databases

1.7.1 Introduction

NoSQL[17] databases represent a category of database management systems that deviate from the traditional relational model. Designed to handle large volumes of unstructured or semi-structured data, NoSQL databases offer flexibility, scalability and performance optimized for specific use cases. They emerge as a response to the limitations of relational databases, especially in the context of web applications, big data and cloud computing.

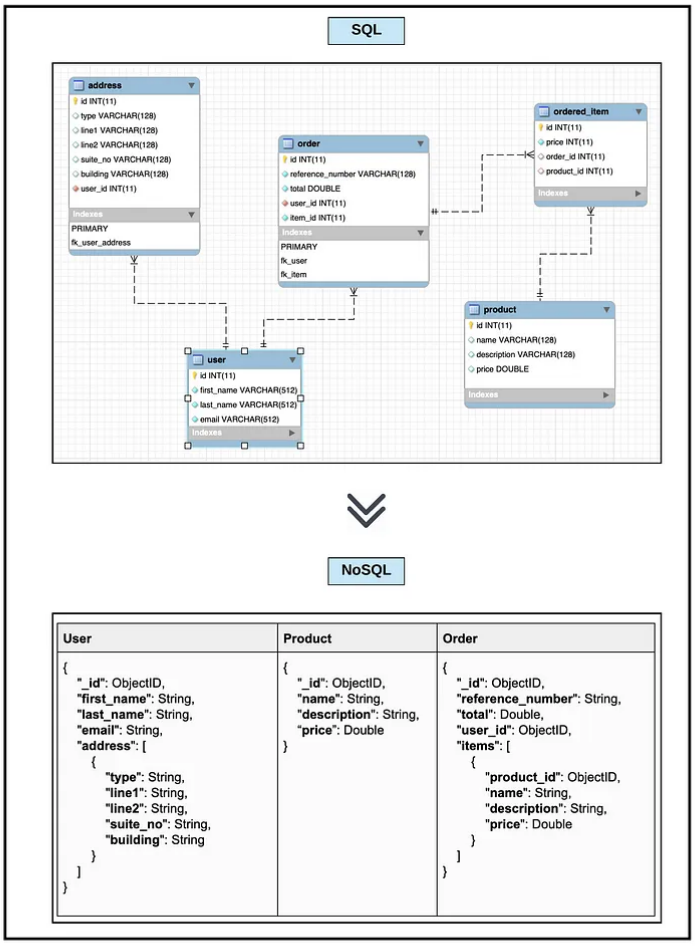
1.7.2 Non Relational Databases Advantages

- Schema-less: Data can be stored freely, without a predefined tabular schematic, allowing for greater flexibility in managing data that can vary.
- Horizontal scalability: It has the ability to distribute the load across multiple servers.
- High availability and fault tolerance: It is designed to guarantee service continuity even in the presence of hardware failures.
- Variety of data models: Support different data models such as documents, columns, graphs and key-values.

NoSQL data models type and some examples

- Document Databases: Use semi-structured documents, often in JSON or BSON format. Examples are MongoDB and CouchDB.
- Columnar Databases: Organize data into tables, rows and columns, but with greater flexibility than relational databases. Examples include Cassandra and HBase.
- Graph Databases: Manage highly connected data, represented as nodes, edges and properties. Examples include Neo4j and ArangoDB.
- Key-Value Databases: Store key-value pairs, ideal for applications that require quick access to data. Examples include Redis and DynamoDB.

[17]NoSQL = Not Only SQL



User	Product	Order
`{ "_id": ObjectID, "first_name": String, "last_name": String, "email": String, "address": [ { "type": String, "line1": String, "line2": String, "suite_no": String, "building": String } ] }`	`{ "_id": ObjectID, "name": String, "description": String, "price": Double }`	`{ "_id": ObjectID, "reference_number": String, "total": Double, "user_id": ObjectID, "items": [ { "product_id": ObjectID, "name": String, "description": String, "price": Double } ] }`

Figure 1.10: Example of a NoSQL JSON-based DB, compared to a traditional RDBMS[18]

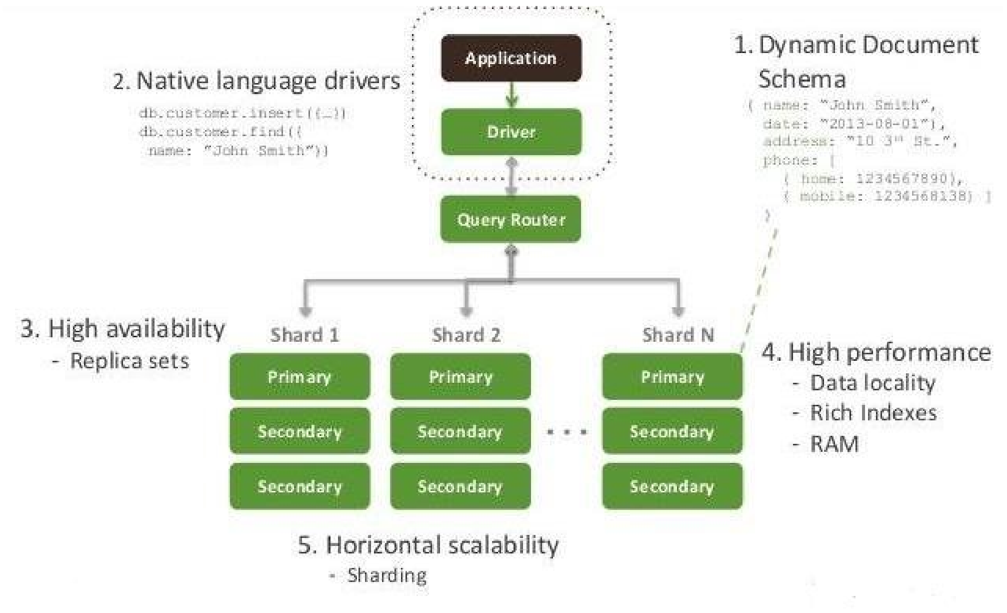
1.7.3 MongoDB

MongoDB is one of the most popular NoSQL databases, known for its ability to store data in document format using BSON[18]. MongoDB was designed to handle modern applications that require agility, scalability and high performance.

[18]BSON = Binary JSON

1.7.4 Principal Characteristics of MongoDB

- Document-Oriented: Data is stored in JSON-like documents[19], making them easy for developers to read and write.
- Schema-less: It does not require a predefined data model, allowing dynamic changes to the structure of documents.
- Scalability: Supports horizontal scalability through sharding, a technique that distributes data across multiple machines.
- High Availability: Uses data replication to ensure fault tolerance and high availability.
- Indexes: Supports various types of indexes to improve query performance.



```
db.customer.insert({…})
db.customer.find({
 name: "John Smith"})
```

```
{ name: "John Smith",
  date: "2013-08-01"},
  address: "10 3rd St.",
  phone: [
    { home: 1234567890},
    { mobile: 1234568138} ]
}
```

Figure 1.11: MongoDB Architecture[19]

1.7.5 MongoDB Architecture

MongoDB's architecture is based on three main components: databases, collections and documents.

- Database: A container for collections of documents.

[19]BSON format

- Collections: Groups of documents stored within a database. it is similar to tables in relational databases.
- Documents: Units of data stored in BSON format, containing fields and values.

It have also some advanced features, such as[20]:

- Indexes: MongoDB supports the creation of indexes to improve query performance. Indexes can be created on one or more fields in a document.
- Sharding: The sharding technique allows you to distribute data across multiple nodes, improving scalability and performance.
- Replica Set: A group of MongoDB instances that maintain the same data set, providing redundancy and high availability.
- Safety: MongoDB offers various security mechanisms, including authentication, authorization, data encryption at rest and in transit and audit logging.

1.7.6 To Sum Up

MongoDB, as a representative of NoSQL databases, offers a flexible and scalable alternative to traditional relational databases. Its ability to handle semi-structured data, combined with advanced features like replication and sharding, makes it an ideal choice for many modern applications. Understanding its features, architecture and basic operations is critical to unlocking the full potential of MongoDB in software development projects.

1.8 Web App Frameworks

1.8.1 Introduction

In the modern digital era, web applications have become essential for all the people. These applications, which run on web browsers and provide interactive services to users, range from simple websites to complex systems handling millions of transactions daily. Building such applications from scratch can be a daunting task, requiring extensive knowledge of various programming languages, databases and web technologies. This is where web application frameworks are the most common used choice.

Web application frameworks are software libraries designed to simplify the development of web applications. They provide a structured foundation, including pre-written code, tools and libraries, which developers can use to build robust and scalable applications more efficiently. By abstracting the common tasks involved in web development, such as routing, authentication and data handling, frameworks allow code developers to spend time in developing more functions of their application, instead of thinking about redoing the same things already developed by others.

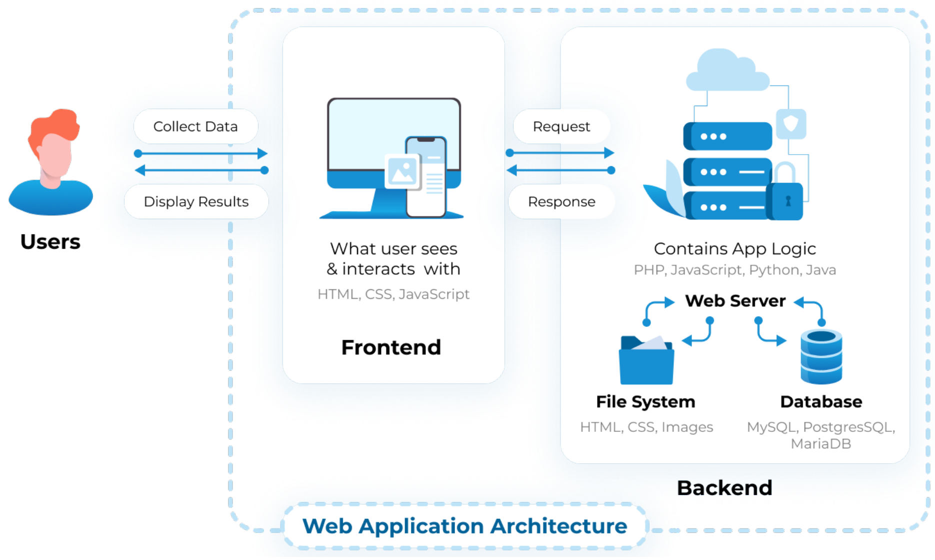


Figure 1.12: Example of a Web App Framework Structure[21]

Key Features of Web Application Frameworks:

- Modular Design: Frameworks are typically modular, offering components that can be reused across different projects. This modularity promotes code reuse and helps maintain consistency across applications.
- Scalability: Good frameworks are designed to support the growth of applications, ensuring they can handle increasing loads and complexities as user demands grow.
- Security: Frameworks often include built-in security features, to protect against common web vulnerabilities like SQL injection and cross-site scripting. This helps developers build secure applications without needing to implement these protections from scratch.
- Efficient Development: By providing ready-to-use components and automating repetitive tasks, frameworks significantly reduce development time and effort. This allows for quicker iterations and faster time-to-market.
- Community and Support: The most popular frameworks have large communities and extensive documentation over the internet. This support network can be fundamental for troubleshooting issues, sharing best practices and staying updated with the latest developments.

Different types of Web Application Frameworks:

- Front-end Frameworks: These focus on the user interface and experience. Examples include React, Angular and Vue.js. They help create dynamic and responsive web interfaces.
- Back-end Frameworks: These handle the server-side logic, database interactions and application workflows. Examples include Django, Flask, Express and Ruby on Rails. They provide tools for managing data, user authentication and API development.
- Full-stack Frameworks: These provide both front-end and back-end capabilities, allowing developers to build complete applications within a single framework. Examples include Meteor and Next.js.

1.8.2 Web App Framework Advantages

- Consistency and Best Practices: Frameworks enforce best practices and coding standards, which lead to more maintainable and reliable code.

- Rapid Development: Frameworks accelerate development processes through reusable components and automated tasks.
- Reduced Errors: Pre-built components and standardized structures reduce the likelihood of coding errors and security flaws.
- Scalability and Performance: Frameworks are optimized for performance and scalability, ensuring applications can grow and handle increasing traffic.
- Community and Resources: Extensive community support and resources make problem-solving easier and provide continuous learning opportunities.

1.8.3 Streamlit

Streamlit is an awesome web app framework, open-source, specifically designed for Machine Learning and Data Science projects. It allows developers and data scientists to create and share custom web applications for machine learning and data science with minimal effort. Streamlit's straightforward interface and powerful features enable quick prototyping and deployment of interactive web apps, making it a popular choice among professionals to visualize data and models.

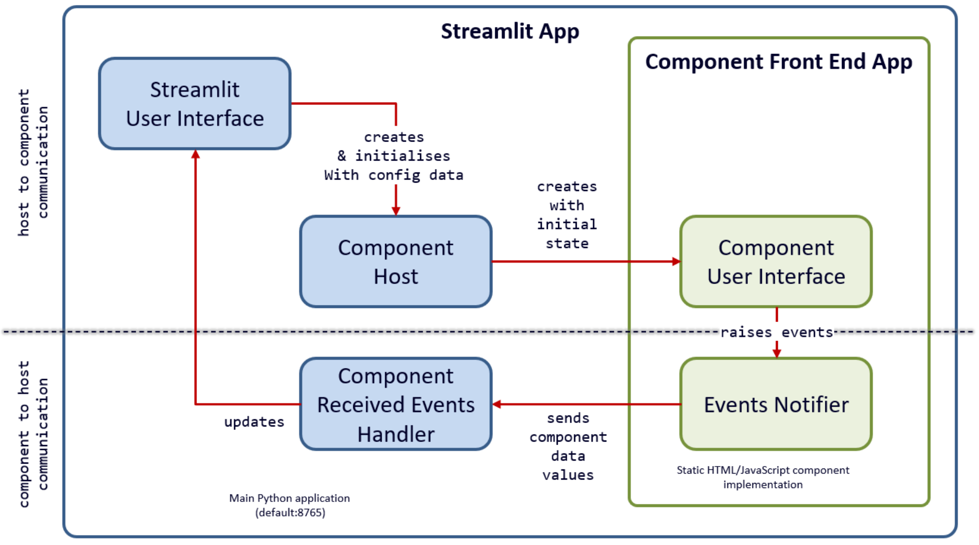

Streamlit App
Component Front End App
host to component communication
component to host communication
Streamlit User Interface
creates & initialises With config data
creates with initial state
Component Host
Component User Interface
raises events
Component Received Events Handler
Events Notifier
updates
sends component data values
Static HTML/JavaScript component implementation
Main Python application (default:8765)

Figure 1.13: Details of a Streamlit App Structure.[22]

1.8.4 Principal Characteristics of Streamlit

- Ease of Use: Streamlit applications are built with pure Python and the framework allows for the transformation of Python scripts into interactive apps effortlessly.
- Widgets: Streamlit provides a variety of widgets, most of them interactive, such as sliders, buttons and text inputs, which can be used to create dynamic applications.
- Real-time Updates: Apps built with Streamlit can be updated in real-time as the code data changes, providing an up-to-date view of results.
- Flawless Integration: Streamlit integrates ideally with popular data science libraries like *NumPy*, *Pandas*, *Matplotlib* and more.

1.8.5 Streamlit Advantages

- Rapid Prototyping: Streamlit allows for quick development and iteration of prototypes. This is particularly useful for academic projects where iterative testing and visualization of models and results are crucial.
- Simplified Development Process: Streamlit simplifies the process of creating interactive web applications, because developers do not need to have extensive knowledge of web development frameworks like HTML, CSS, or JavaScript, as Streamlit handles the web interface automatically.
- Interactive Data Visualization: The framework supports a wide range of interactive widgets that can be used to create rich and dynamic visualizations. This is beneficial for exploring data and presenting findings in a more engaging way.
- Open-Source and Community Support: Being open-source, Streamlit has a robust community that contributes to its development and provides support. This community aspect ensures that users can find help and share solutions to common problems.
- Real-time Data Processing: Streamlit applications can be updated in real-time, allowing users to interact with data and see immediate results. This feature is advantageous for monitoring live data streams and conducting real-time analyses.

1.8.6 Streamlit Disadvantages

- Performance Limitations: Streamlit can face performance issues with very large datasets or highly complex applications, as it runs in a single-threaded process and contents need to be refreshed at every change in input. This may limit its use for applications requiring high computational performance.
- Limited Customization: While Streamlit is easy to use, it offers limited customization options compared to more advanced web frameworks. Users with specific design or functionality requirements may find it restrictive.
- Dependency on Python: Streamlit is built for Python, which means it cannot be used with projects that are based on other programming languages. This can be a limitation if the project involves a multi-language stack.
- Deployment Complexity: Deploying Streamlit apps can be straightforward for simple use cases, but as the application grows in complexity, deployment can become more challenging, especially when integrating with other systems or when scaling.

- Learning Curve for Complex Widgets: While basic widgets are easy to implement, more complex interactions may require a deeper understanding of both Streamlit and the underlying Python libraries. This could increase the learning curve for users new to these tools.

1.8.7 To Sum Up

Streamlit offers a robust and user-friendly platform to develop interactive applications for machine learning and data science. Its advantages make it an excellent choice for rapid prototyping and interactive visualization of data. However, potential limitations such as performance issues and limited customization should be considered based on the specific requirements of the project. For projects heavily relying on Python and requiring quick development cycles, like this thesis project, Streamlit can be an ideal tool, while more complex applications may require complementary solutions.

1.9 Reverse Proxy

1.9.1 Introduction

A reverse proxy server is an intermediary server that is posed between client devices and a web server. Unlike a forward proxy, which simply manages traffic from clients to the internet, a reverse proxy handles traffic coming from the internet and directs it to the appropriate server in backend. This mechanism is widely used in modern web architectures due to its numerous benefits.

1.9.2 Reverse Proxy Advantages

[23]

- Load Balancing: One of the primary functions of a reverse proxy is to distribute incoming client requests across multiple backend endpoint. This ensures no single server becomes overwhelmed with too many requests, improving overall system performance and reliability.

- Security Enhancement: Reverse proxies can act as a shield between the client and the backend servers, for example not directly exporting ports of the services and filtering traffic using a set of rules, offering an additional layer of security. They can hide the existence and characteristics of the backend servers, preventing direct attacks on them. Moreover, they can perform tasks like SSL termination, where the reverse proxy handles SSL encryption and decryption, reducing the burden on backend servers and acting as a security layer for web frameworks.

- Web Acceleration: By caching content, reverse proxies can serve repeated requests for the same content quickly, reducing the load on backend servers and speeding up response times. This is particularly useful for static content like images, stylesheets and JavaScript files.

- Compression and Decompression: Reverse proxies can compress outgoing data and decompress incoming data, reducing the amount of data transferred over the network and improving load times.

- Application Firewall: Reverse proxies can inspect incoming requests and filter out malicious traffic, providing an additional security layer through web application firewalls.

- SSL[20] Offloading: Handling SSL encryption and decryption can be resource-intensive for backend servers. Reverse proxies can manage this process, freeing up backend to handle more requests.

1.9.3 Principal Usages

- Content Delivery Networks (CDNs): CDNs often use reverse proxies to cache content at various locations worldwide, reducing latency by serving content from a location closer to the user.
- Microservices Architecture: In microservices, a reverse proxy can route requests to different services, helping manage and scale services independently.
- High-Availability Systems: Reverse proxies help in creating redundant and failover systems, ensuring high availability of services.

1.9.4 Example of Reverse Proxy Servers

Several reverse proxy solutions are widely used, each with its strengths and suited to different scenarios:

- Nginx: Known for its high performance and flexibility, Nginx is often used as a reverse proxy and load balancer.
- HAProxy: A robust and reliable solution for high availability and load balancing.
- Apache HTTP Server: Through the modproxy module, Apache can serve as a reverse proxy.
- Traefik: A modern reverse proxy and load balancer designed for microservices and containerized environments.

[20]SSL = Secure Socket Layer

1.9.5 NGINX

NGINX is an open-source web server used also as a reverse proxy server, load balancer and HTTP cache. NGINX was designed originally to handle up to 10,000 simultaneous connections on a single server. Over the years, NGINX has grown in popularity due to its high performance, stability, JSON-based configuration and low resource consumption. NGINX is widely used by many high-traffic websites such as Netflix, GitHub, WordPress.com and many others.

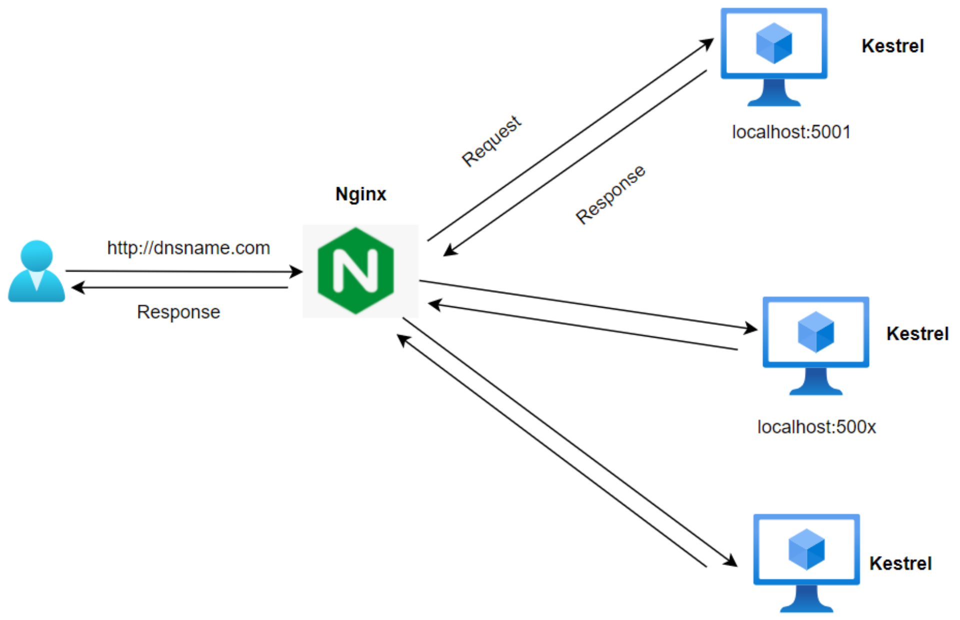


Figure 1.14: NGINX Reverse Proxy configuration[24]

1.9.6 NGINX Advantages

[25]

- High Performance and Scalability: NGINX is known for its ability to handle a large number of simultaneous connections keeping low memory usage. This makes it ideal for high-traffic websites and applications. It uses an event-driven, asynchronous architecture that allows it to efficiently manage multiple connections within a single thread.
- Reverse Proxy and Load Balancing: NGINX can be used as a reverse proxy to distribute incoming traffic across multiple servers. This helps in achieving high availability and reliability by balancing the load and ensuring that no single

server becomes a bottleneck. It supports various load balancing algorithms like round-robin, least connections and IP hash.

- Security[26]: NGINX offers various security features such as SSL/TLS termination, access control and request rate limiting. This helps in protecting web applications from common threats and vulnerabilities.
- HTTP/2 and gRPC Support[27]: NGINX supports modern web protocols like HTTP/2, which improves web performance by allowing multiple requests and responses to be multiplexed over a single connection. It also supports gRPC, which is a high-performance RPC[21] framework.
- Static Content Serving and Caching: NGINX excels at serving static content such as images, videos and other static files. It can also cache responses from upstream servers, reducing the load on backend servers and improving response times for clients.
- Extensibility: NGINX is highly extensible through modules. There are many third-party modules available that extend its functionality, including modules for WebSocket proxying, image resizing and more.

1.9.7 NGINX Disadvantages

- Complex Configuration for Beginners: NGINX's configuration syntax can be complex and difficult to understand for beginners. This might lead to misconfigurations and potential security issues if not handled properly.
- Limited Support for Dynamic Content: NGINX is optimized for serving static content and acting as a proxy. While it can handle dynamic content through FastCGI, SCGI and uWSGI interfaces, it is not as efficient as servers like Apache in running dynamic web applications directly.
- Less Mature Ecosystem for Some Applications: Although NGINX has a strong ecosystem, some specific applications and modules might have less community support compared to more established web servers like Apache.
- Learning Curve for Advanced Features: Utilizing advanced features such as complex load balancing configurations, SSL/TLS settings and custom modules can have a steep learning curve and require in-depth knowledge of NGINX's inner workings.

[21]RPC = Remote Procedure Control

- Documentation and Community Support: While there is extensive documentation available for NGINX, some users might find it less comprehensive compared to other web servers. Community support, while strong, might not be as vast as more established platforms like Apache.

1.9.8 To Sum Up

NGINX has become a fundamental component of modern web architecture due to its advantages. While it offers numerous advantages it also comes with challenges like a steep learning curve and complex configuration. NGINX presents a rich topic for exploration, from its architectural design and implementation to its practical applications and performance optimizations in various scenarios.

1.10 From Industry 4.0 to Industry 5.0

1.10.1 Introduction

The so named *Industry 4.0* represents the fourth industrial revolution, characterized by the fusion of digital, physical and biological systems. It takes advantage of advanced technologies such as the Internet of Things (IoT), Artificial Intelligence (AI), robotics, big data and cloud computing to create smart factories and enhance manufacturing processes.

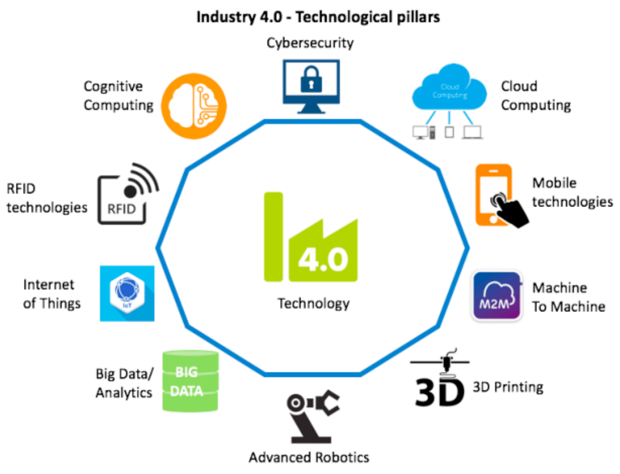


Figure 1.15: Various technologies for Industry 4.0[28]

1.10.2 Main Principles

Its development is based on these principles:

- Interconnectivity: it enables seamless communication between devices, machines and humans through IoT and industrial internet systems.
- Data-Driven Decision Making: it utilizes big data analytics to gain insights, optimize processes and predict trends, leading to smarter and faster decisions.
- Automation and Autonomy: it integrates AI and robotics to automate complex tasks, reduce human error and increase productivity.
- Customization and Flexibility: it supports mass customization and flexible production lines that can quickly adapt to changing demands.
- Sustainability: it promotes energy efficiency and reduce waste through optimized processes and circular economy practices.

- Cybersecurity: it ensures the security and integrity of interconnected systems to protect against cyber threats.

1.10.3 Main Goals

These are the main objectives of Industry 4.0:

- Enhance Operational Efficiency: the goal is to improve manufacturing processes through automation, reducing downtime and increasing output.
- Foster Innovation: the goal is to drive technological advancements and product development through data insights and new capabilities.
- Create Smart Factories: the goal is to develop interconnected, autonomous factories that can operate with minimal human intervention.
- Improve Quality: the goal is to utilize real-time data and machine learning to detect defects early, ensuring higher product quality.
- Boost Competitiveness: the goal is to equip industries with the tools needed to compete in the global market through superior technology and innovation.

1.10.4 Main Strategies

These are the main strategies:

- Adoption of Advanced Technologies: through the implementation of IoT, AI, robotics and other Industry 4.0 technologies across production lines.
- Skill Development: training the workforce in the new skills required to operate and manage Industry 4.0 technologies.
- Collaborative Ecosystems: promote the collaboration between companies, research institutions and governments to accelerate Industry 4.0 adoption.
- Infrastructure Investment: investing in digital infrastructure, such as high-speed internet and secure cloud computing, to support Industry 4.0 initiatives.
- Continuous Improvement: regularly assess and update processes to leverage the latest technological advancements and maintain competitiveness.

1.10.5 To Sum Up

Industry 4.0 has marked a significant evolution in manufacturing and industrial processes. Through embracing its principles and strategies, organizations can achieve greater efficiency, innovation and sustainability, positioning themselves as leaders in the new industrial era.

1.11 Industry 5.0

1.11.1 The new concept behind Industry 5.0

While Industry 4.0 has revolutionized manufacturing through automation, interconnectivity and data-driven processes, Industry 5.0 takes this evolution a step further by emphasizing the collaboration between humans and machines. It seeks to harness the strengths of both to create more personalized, sustainable and socially responsible manufacturing practices.

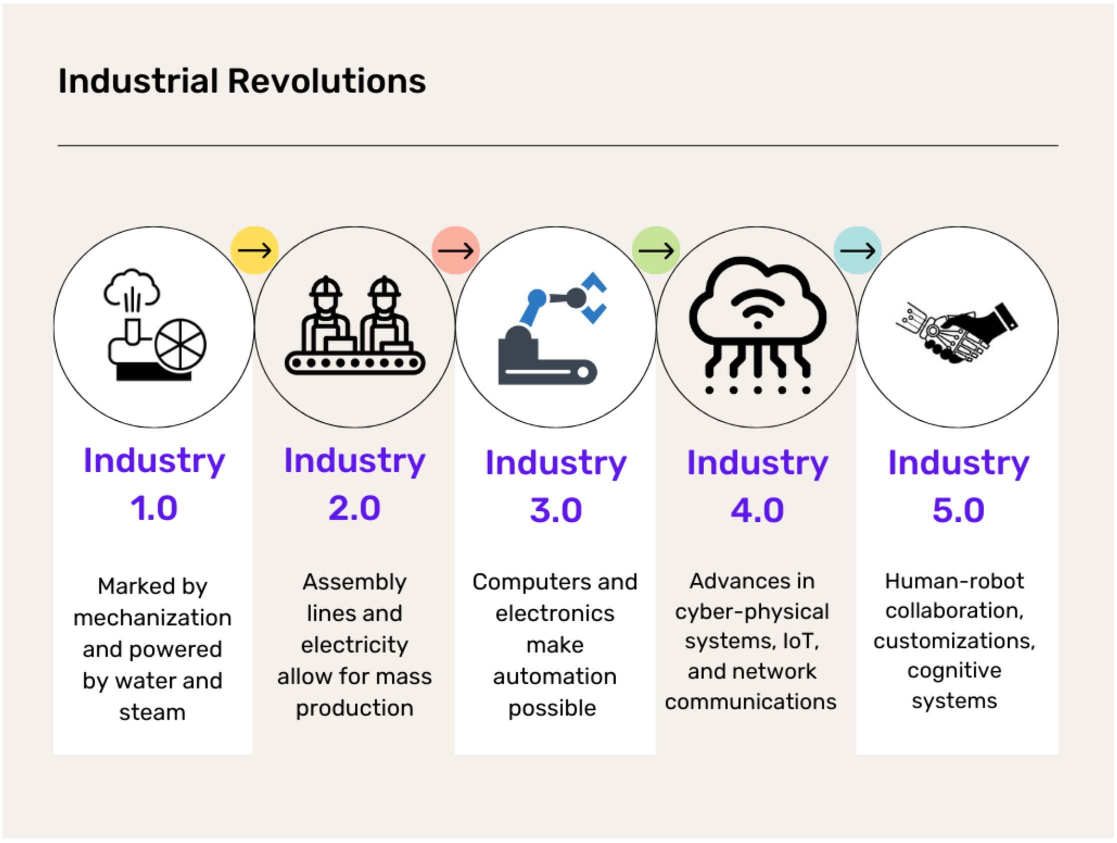

Industrial Revolutions
Industry 1.0
Marked by mechanization and powered by water and steam
Industry 2.0
Assembly lines and electricity allow for mass production
Industry 3.0
Computers and electronics make automation possible
Industry 4.0
Advances in cyber-physical systems, IoT, and network communications
Industry 5.0
Human-robot collaboration, customizations, cognitive systems

Figure 1.16: Progresses to Industry 5.0[29]

1.11.2 What's new in Industry 5.0

Industry 5.0 is based on these key features:

- Human-Centric Approach: the brand new Industry 5.0 places humans back at the center of the production process, focusing on enhancing human capabilities rather than replacing them. It emphasizes collaboration between humans and robots *cobots*, where robots handle repetitive tasks, while humans focus on creativity, problem-solving and decision-making.

- Personalization: through Industry 5.0 mass customization becomes possible at an unprecedented scale. The integration of advanced technologies allows manufacturers to produce highly customized products tailored to individual customer preferences, without sacrificing efficiency.
- Sustainability: Industry 5.0 addresses the growing need for sustainable practices in manufacturing. It promotes the use of green technologies, circular economy principles and energy-efficient processes, aiming to reduce the environmental impact of industrial activities.
- Resilience and Flexibility: Industry 5.0 aims to create more resilient and flexible manufacturing systems that can adapt to changes in demand, supply chain disruptions and unforeseen events. This resilience is achieved through advanced technologies like AI, which can predict and respond to disruptions in real-time.
- Ethics and Social Responsibility: Industry 5.0 emphasizes the ethical use of technology and the social responsibilities of companies. It supports fair labor practices, data privacy and the ethical use of AI and automation.

1.11.3 To Sum Up

Industry 5.0 builds on the technological advancements of Industry 4.0 but with a greater focus on human collaboration, personalization, sustainability and social responsibility. It represents a shift towards a more balanced, ethical and sustainable approach to industrialization, where technology and humanity coexist harmoniously to drive innovation and create value.

1.12 Machine Learning

1.12.1 Introduction

Machine learning (ML) is a subset of artificial intelligence (AI) focused on the development of algorithms that enable computers to learn from and make decisions based on data[30]. Rather than being explicitly programmed to perform a specific task, machine learning systems use statistical techniques to improve their performance over time with the usage of more new data. This adaptive learning process is particularly useful in solving complex problems where traditional algorithms may fall short, such as image recognition, natural language processing and predictive analytics.

In engineering, machine learning has become increasingly relevant due to its ability to optimize processes, improve decision-making and automate tasks in various domains. From predictive maintenance in manufacturing to optimizing energy consumption in smart grids, ML applications are driving innovation and efficiency.

1.12.2 Types of Machine Learning

There are three primary categories of machine learning:

- Supervised Learning: In supervised learning, the model is trained on a labeled dataset, meaning the input data comes with corresponding correct outputs. The goal is for the algorithm to learn the mapping from inputs to outputs and be able to generalize this relationship to new, unseen data. Common applications include regression and classification problems.

- Unsupervised Learning: In this type, the algorithm is provided with data that is not labeled and the system tries to identify hidden patterns or structures in the data. Clustering and dimensionality reduction are typical examples of unsupervised learning tasks.

- Reinforcement Learning[31]: This approach involves an agent that learns to make decisions by interacting with an environment. The agent receives feedback in the form of rewards or penalties and uses this to improve its decision-making policy over time. Reinforcement learning is particularly applicable in robotics and control systems.

1.12.3 Key Concepts

- Training and Testing: The process of developing a machine learning model involves splitting the data into two sets: the training set and the test set.

The training set is used to "teach" the model, while the test set evaluates its performance on unseen data[32]. Cross-validation techniques are also employed to prevent overfitting.

- Overfitting and Underfitting: Overfitting occurs when a model becomes too complex and captures noise in the data rather than the underlying pattern, leading to poor generalization. Underfitting, on the other hand, occurs when the model is too simple to capture the complexities of the data, resulting in low accuracy.
- Model Evaluation: Various metrics are used to assess the performance of a machine learning model, such as accuracy, precision, recall, F1-score and mean squared error (MSE), depending on the task. Proper evaluation ensures that the model performs well not only on the training data but also on new data.

1.12.4 Engineering Applications

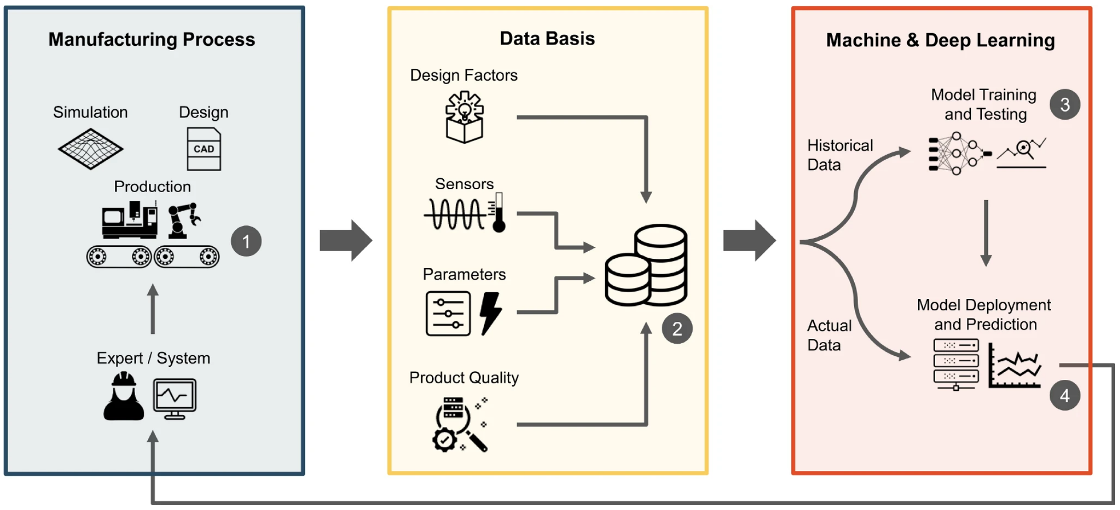


Figure 1.17: Machine Learning Flow[33]

In experimental engineering research, machine learning can be applied in several ways:

Predictive Modeling: Machine learning models can predict outcomes based on historical data, enabling engineers to forecast trends, optimize processes and reduce uncertainties in design and production.

Automation and Control: Algorithms in reinforcement learning are used for automating control systems in fields such as robotics, where the goal is to optimize performance with minimal human intervention.

Fault Detection and Maintenance: Predictive maintenance leverages machine learning models to anticipate equipment failures before they occur, minimizing downtime and reducing operational costs in industries such as manufacturing and energy.

1.12.5 To Sum Up

Machine learning offers powerful tools for solving complex problems in engineering. Its ability to analyze large datasets, detect patterns and make accurate predictions makes it an invaluable asset in experimental and applied research. As more data becomes available and computational power increases, the role of machine learning in engineering is expected to grow, enabling the development of more efficient, intelligent and autonomous systems.

Chapter 2

Methodology

2.1 Introduction to System Data Management Methodologies

In this chapter the methodology for Data Acquisition, Data Storage and Data Processing will be analyzed. In today's data-driven world, effective management of data is critical for ensuring operational efficiency, compliance with regulations, and driving informed decision-making.

Through the various phases of Data Acquisition, Data Storage and Data Processing, it will enhanced operational efficiency, data accessibility, advanced analytics, enabling organizations to unlock valuable insights and maintain a competitive edge.

2.2 Data Acquisition

2.2.1 Introduction

This section details the data acquisition process in the designed system, which includes both a frontend interface and a backend architecture. Data acquisition refers to the collection, transmission and validation of data that flows between the system's frontend (where user interactions occur, in this case Streamlit) and the backend (where the data is processed and stored, in this case PostgreSQL and MongoDB). Now the mechanisms, protocols, and methodologies used to acquire data efficiently and securely will be explored.

There are several possible ways to acquire data and load the system:

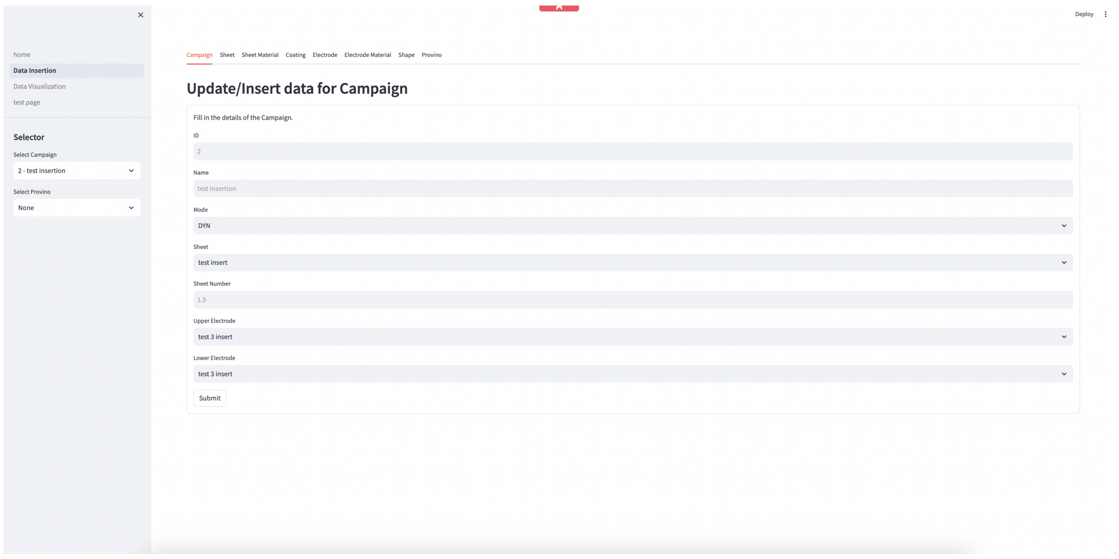
- Through user input;
- Through external API, in case of a system that is connected to external entities;

- Through already stored data, loaded from DBs or predetermined files.

Within this system, the first and the last method has been chosen, and will be analyzed in the next paragraphs.

2.2.2 Acquisition through user input

The acquisition of data through user input is handled via a frontend framework called Streamlit. On the frontend side the user is guided step by step to fill in tables of inputs for the various *provini* (tests). This process ensures that the database is populated with relevant and structured information about how the machine was configured during each test. The intuitive interface simplifies the input process, making it easier for users to provide accurate data and ensuring that all necessary parameters are captured for future analysis or reference.



Selector

Select Campaign: 2 - test insertion

Select Provino: None

Update/Insert data for Campaign

Fill in the details of the Campaign.

ID: 2

Name: test insertion

Mode: DYN

Sheet: test insert

Sheet Number: 1.5

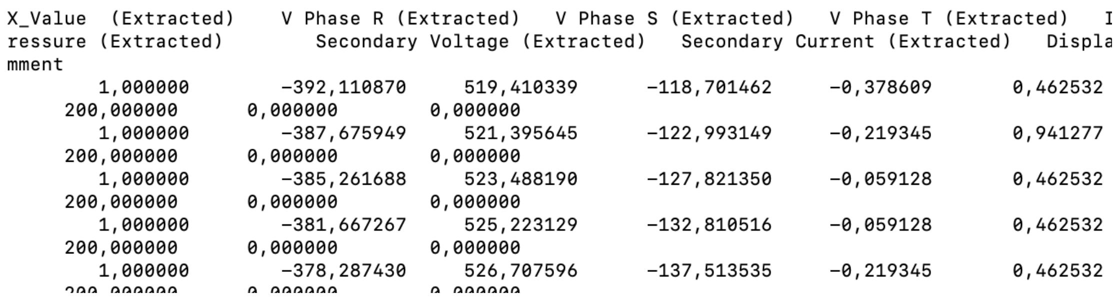
Upper Electrode: test 3 insert

Lower Electrode: test 3 insert

Figure 2.1: An example of data insertion through GUI

2.2.3 Acquisition through files

The acquisition of data through files is carried out using the output file generated by the machine. This file has a highly complex and detailed structure, making it necessary to apply a parsing process to extract relevant information. The raw data, as presented in the output file, is not immediately usable in its original format due to its intricate CSV structure. Therefore, specific parsing techniques are employed to interpret the data, breaking it down into manageable and meaningful components. These extracted details are then systematically organized and stored in the database, ensuring that all critical information about the machine's performance and settings is accurately recorded and easily accessible for future analysis.



```
X_Value  (Extracted)    V Phase R (Extracted)   V Phase S (Extracted)   V Phase T (Extracted)   I
ressure (Extracted)        Secondary Voltage (Extracted)   Secondary Current (Extracted)   Displa
mment
        1,000000        -392,110870     519,410339      -118,701462     -0,378609       0,462532
     200,000000      0,000000        0,000000
        1,000000        -387,675949     521,395645      -122,993149     -0,219345       0,941277
     200,000000      0,000000        0,000000
        1,000000        -385,261688     523,488190      -127,821350     -0,059128       0,462532
     200,000000      0,000000        0,000000
        1,000000        -381,667267     525,223129      -132,810516     -0,059128       0,462532
     200,000000      0,000000        0,000000
        1,000000        -378,287430     526,707596      -137,513535     -0,219345       0,462532
     200,000000      0,000000        0,000000
```

Figure 2.2: An example of data file structure

2.3 Data Storage

2.3.1 Introduction

This section details the data storage process in the designed system, which includes both a frontend interface and a backend architecture. Data storage refers to the collection of data that flows through the system's frontend (where user interactions occur, in this case Streamlit) and will be stored in the the backend (where the data is processed and stored, in this case PostgreSQL and MongoDB). Now the mechanisms, protocols, and methodologies used to store data efficiently and securely will be explored.

Within this system, two storage DataBases are used:

- PostgreSQL for all data details of *provini*;
- MongoDB, to keep raw data files of various *provini*.

Now the two storage options will be analyzed in detail.

2.3.2 PostgreSQL

PostgreSQL is a Relational SQL database management system particularly adapt to store linked data. Here data are organized into tables, linked together by the usage of indexes. Data are inserted directly from front end or file and the information are stored in relative tables. As shown in photo, they are useful to enhance query performance.

In fact, indexes make data retrieval much faster by reducing the amount of data that needs to be scanned. Instead of going through all the rows in a table, PostgreSQL can use the index to directly locate the relevant rows. Also, it is useful, when joining large tables, to have indexes on the join columns because they can significantly improve the performance of the join operation. The usage of incremental indexes is what mantains the correlations between *provini* and different campaigns.

2.3.3 MongoDB

MongoDB is a Non Relational document-oriented database management system particularly conveninent for storing large data. It's main usage in this project is the file storage capability. MongoDB can store and manage large files through a specification called GridFS.

GridFS is MongoDB's specification for storing and retrieving large files, such as images, audio files, videos, or any type of binary data. MongoDB doesn't store

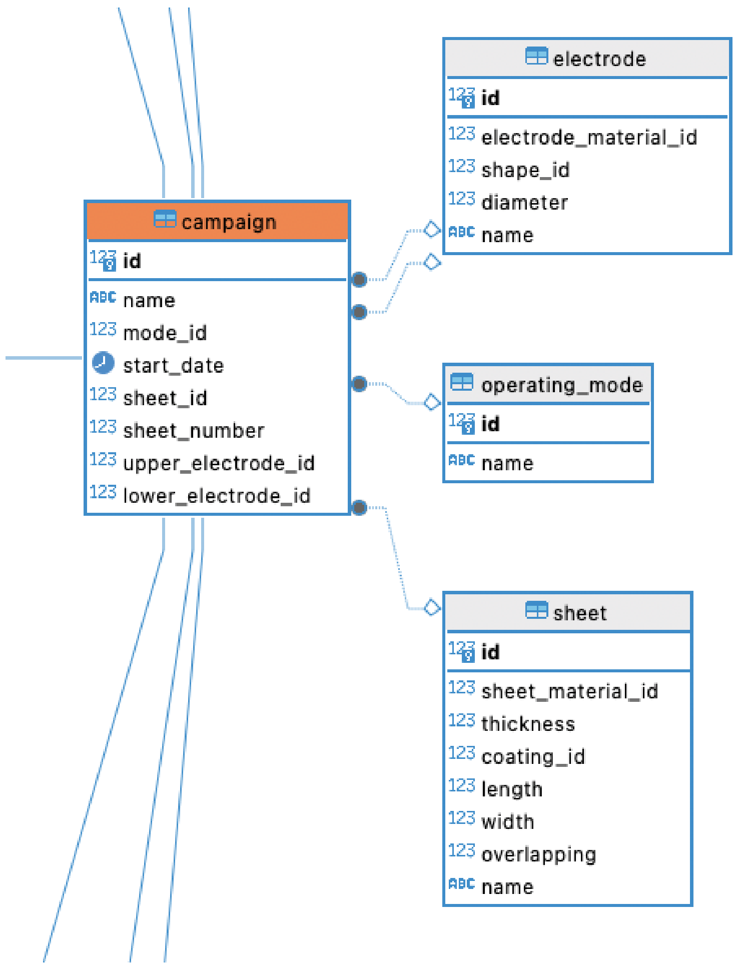


Figure 2.3: An Example of usage of indexes

these files directly in a single document but divides them into smaller parts (chunks) and stores the metadata separately. The reason for this is that MongoDB has a size limit of 16 MB per document, so GridFS splits the file into manageable parts.

When a file is uploaded using GridFS, MongoDB:

- Splits the file into chunks (default chunk size is 255 KB).
- Each chunk is stored in a separate document within a collection called fs.chunks.
- Metadata about the file (like filename, upload date, and file length) is stored in a collection called fs.files.

GridFS uses two main collections:

- fs.files: Contains metadata about each file, such as Filename, File size, Upload date, md5 hash and Content type (if provided);
- fs.chunks: Contains the actual chunks of the file, each chunk having a reference to the file (files id), the binary data (data) and the chunk's order in the file (n).

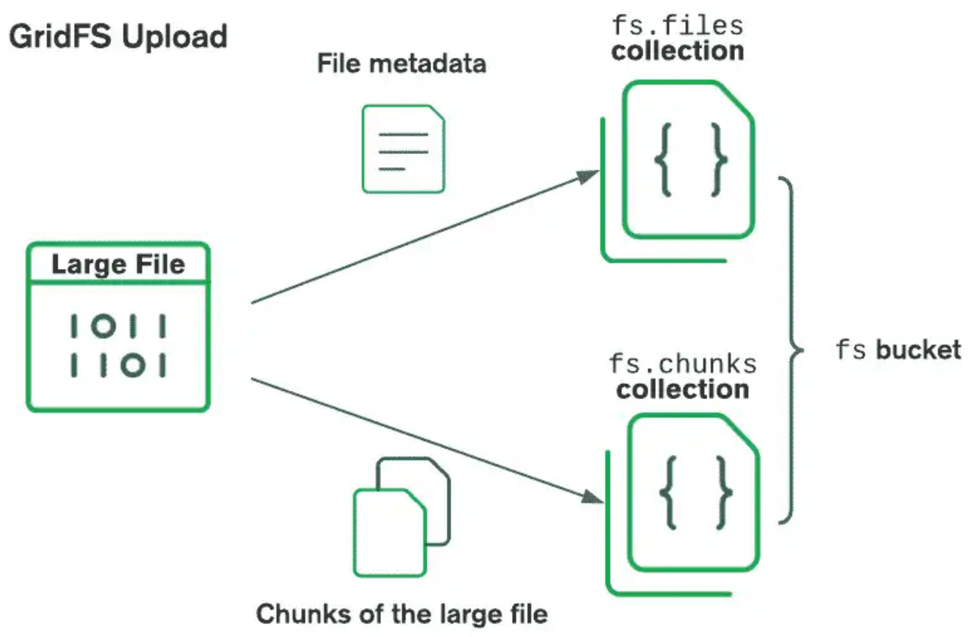


Figure 2.4: MongoDB GridFS structure[34]

To retrieve a file using GridFS, MongoDB reads all the chunks, reassembles them in the correct order based on the chunk number (n), and streams the file back as a unique sequence of bytes.
It is possible to store files larger than 16 MB, and since the chunks are stored as separate documents, GridFS enables efficient querying and management of files that would otherwise exceed BSON's size limit.

- Uploading a File: When a file is uploaded to MongoDB, when specified the file, GridFS takes care of chunking it and saving the chunks into the database.
- Downloading a File: When downloading, GridFS reassembles the file and streams it back as a whole.

- Partial File Retrieval: GridFS allows retrieving only specific portions of a file. For example, it is possible to stream a video file in chunks, which is useful for applications like media streaming.

GridFS has several advantages in this context of use:

- No Size Limit: Handles files larger than MongoDB's 16 MB document size limit.
- Efficient File Streaming: Ideal for applications like media streaming where files are read in chunks, or in which it is needed to retrieve data from files.
- Metadata Support: Can be associated metadata with your files (e.g., file type, custom attributes).

Although his advantages already analyzed, GridFS has also some disadvantages: for example, even though is is ideal for large files that streamed or work with in pieces, for smaller files or simpler use cases, it is better to store the file data directly in MongoDB in binary format, using BSON without GridFS.

2.4 Data Processing

2.4.1 Introduction

This section details the data processing process in the designed system, which includes both a frontend interface and a backend architecture. Data processing refers to the manipulation of data that has been taken from system's backend (where the data is processed and stored, in this case PostgreSQL and MongoDB) and will be shown in system's frontend (where user interactions occur, in this case Streamlit). Now the mechanisms, protocols, and methodologies used to process data efficiently and securely will be explored.

Within this system, two types of data management has been done:

- Data management with data retrieved from PostgreSQL;
- Data management with files retrieved from MongoDB.

Now the two management procedures will be analyzed in detail.

2.4.2 PostgreSQL data management

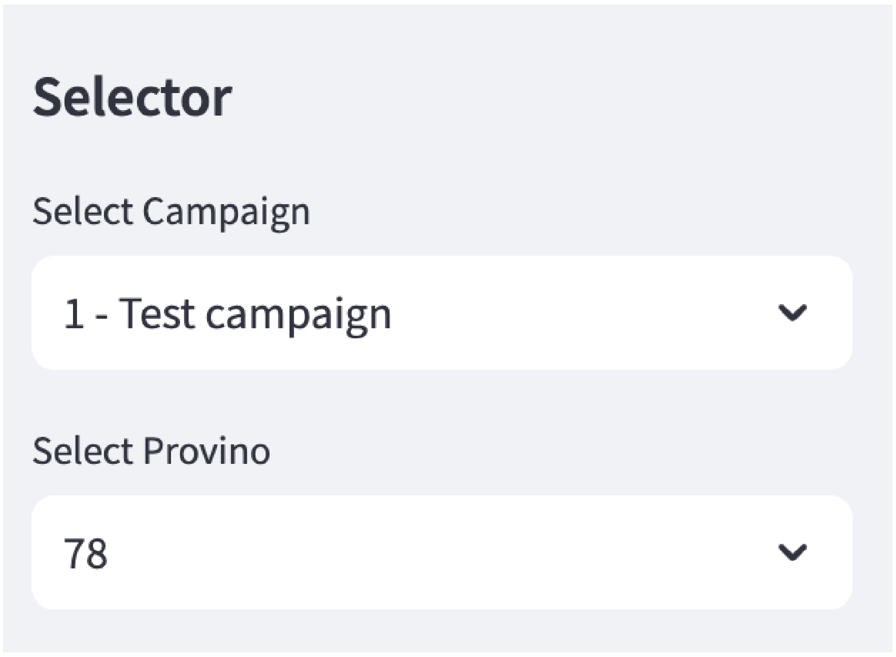
PostgreSQL stores data in interlinked tables, as already shown before, so to be able to process and manage data retrieved from the DB it is necessary to retrieve data through query that can combine multiple tables. A simple and intuitive usage of data retrieved from PostgreSQL tables is the filtering of possible selections in the GUI

In this case, the index of campaign and *provino* is used to show the correct values set in the database for the relative *provino.*

2.4.3 MongoDB files management

MongoDB is used to store raw data files outputted from the welding machine. To analyze these files it is necessary to first download the file locally, then a process for removing header lines from file is applied, and at last a parsing algorithm is applied. Data are stored internally in CSV format, so it is necessary to load them first in script and then serialize them in various collections. After that it is possible to regroup them and filter out spike values.

Here above, an example of how raw data outputted from the welding machine of this experiment have been converted and resumed in an xlsx file, to be more human readable.



Selector

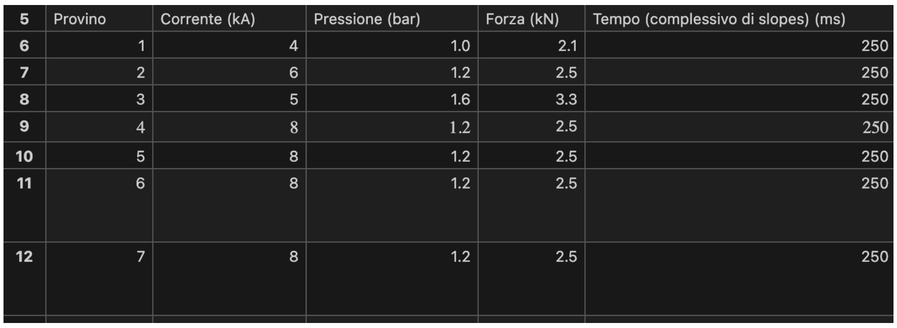
Select Campaign

1 - Test campaign

Select Provino

78

Figure 2.5: GUI Filtering box



5	Provino	Corrente (kA)	Pressione (bar)	Forza (kN)	Tempo (complessivo di slopes) (ms)
6	1	4	1.0	2.1	250
7	2	6	1.2	2.5	250
8	3	5	1.6	3.3	250
9	4	8	1.2	2.5	250
10	5	8	1.2	2.5	250
11	6	8	1.2	2.5	250
12	7	8	1.2	2.5	250

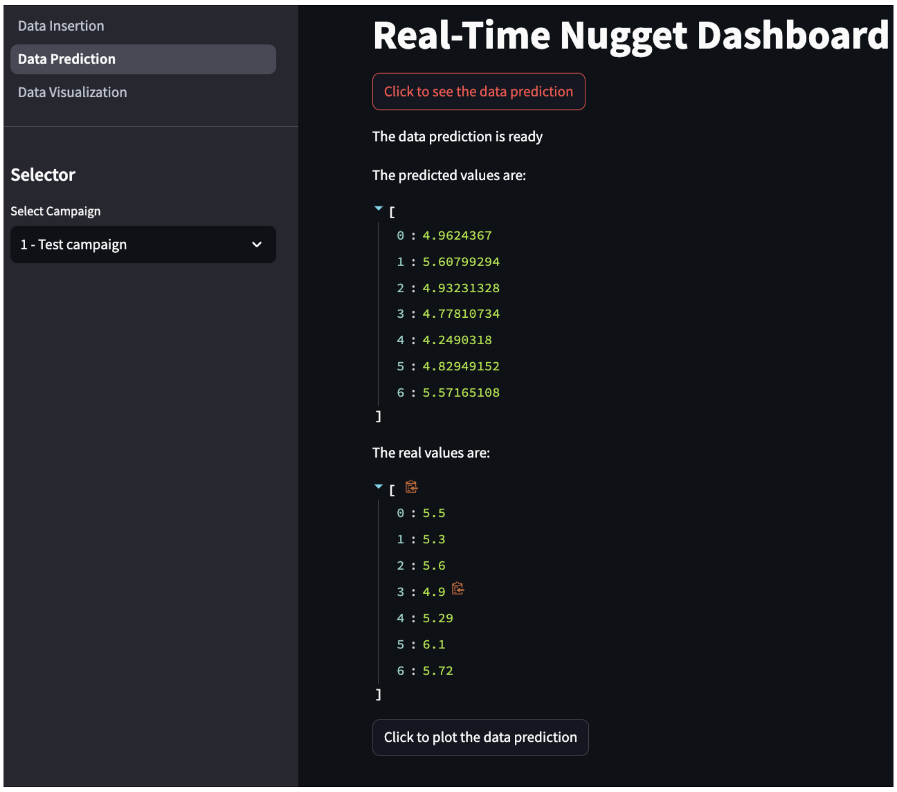
Figure 2.6: An Example of how data are organized in human readable files

2.4.4 Integration with Machine Learning

For testing purposes, a small Machine Learning algorithm has been integrated into the front-end.

This algorithm simply takes all the strength data from the samples that achieved a nugget and uses 80% of the samples for training, then uses the remaining 20% to estimate a prediction, which is then compared with the real result, and the error is estimated.

Since this is not the main topic of the thesis, it has not been fully explored, but it serves as a demonstration to show how this framework can integrate with potentially advanced models.



Real-Time Nugget Dashboard

The data prediction is ready

The predicted values are:

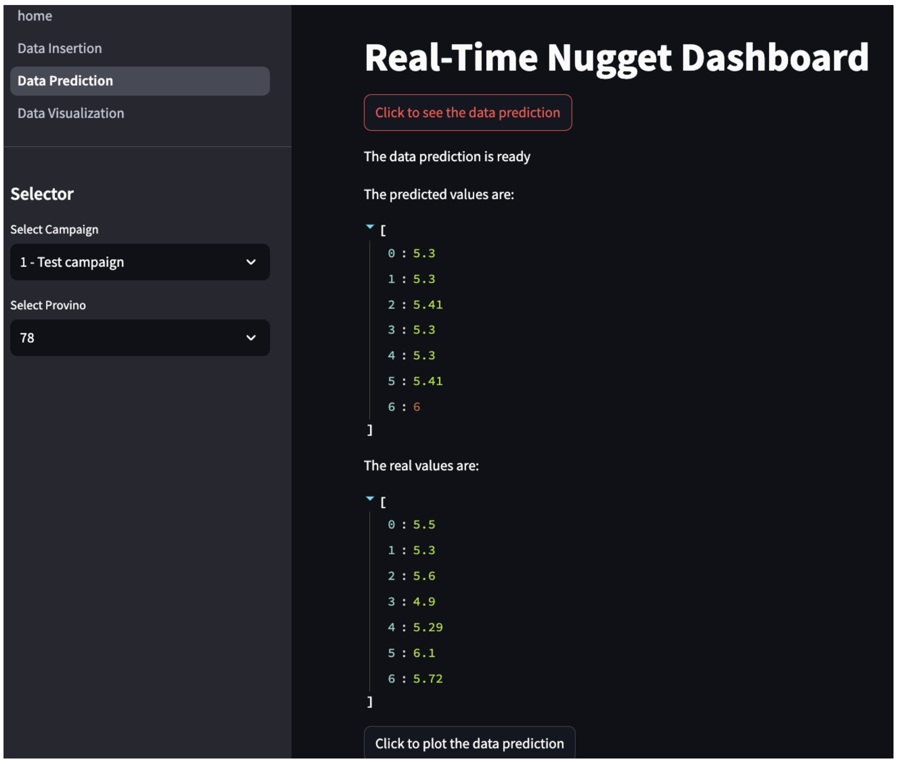
```
[
  0 : 4.9624367
  1 : 5.60799294
  2 : 4.93231328
  3 : 4.77810734
  4 : 4.2490318
  5 : 4.82949152
  6 : 5.57165108
]
```

The real values are:

```
[
  0 : 5.5
  1 : 5.3
  2 : 5.6
  3 : 4.9
  4 : 5.29
  5 : 6.1
  6 : 5.72
]
```

Figure 2.7: A simple button that predict nugget diameter trained on Test Campaign through a Linear Regression Model

To be able to compare between different models, also another simple algorithm based on Decision Tree regression has been inserted, with more accurate results.



Real-Time Nugget Dashboard

The data prediction is ready

The predicted values are:

```
[
  0 : 5.3
  1 : 5.3
  2 : 5.41
  3 : 5.3
  4 : 5.3
  5 : 5.41
  6 : 6
]
```

The real values are:

```
[
  0 : 5.5
  1 : 5.3
  2 : 5.6
  3 : 4.9
  4 : 5.29
  5 : 6.1
  6 : 5.72
]
```

Figure 2.8: A second test with a Decision Tree Regression algorithm that predict nugget diameter trained on Test Campaign

Chapter 3

System Architecture

3.1 Introduction

Here presented the four main actors that compose the whole system.

This system represents an advanced technological solution that combines the advantages of two complementary databases, PostgreSQL and MongoDB, to efficiently manage and store data. PostgreSQL, a relational database, is used for operations requiring ACID[1] transactions and the management of structured data, while MongoDB, a NoSQL database, handles unstructured or semi-structured data, like files in our specific case, offering flexibility in data modeling.

The frontend of the system is developed using Streamlit, an open-source framework that allows for the creation of interactive web applications for data visualization and analysis quickly and easily. Streamlit enables users to interact with the data in real-time through an intuitive and responsive interface.

To ensure that web traffic is managed securely and efficiently, the system utilizes NGINX as a reverse proxy. NGINX handles incoming requests, distributing them across various system components, thereby enhancing scalability and the overall security of the architecture.

This combination of technologies provides a robust and flexible platform, suitable for a wide range of applications—from enterprise data management to advanced analytics—while ensuring high performance and ease of development.

[1]Atomicity, Consistency, Isolation and Durability

3.2 Experimental Campaign

3.2.1 The goal

The goal of the experimental campaign conducted at the J-Tech laboratory of Politecnico di Torino was to produce a series of welds using the available resistance spot welding (RSW) machine. By adjusting welding current and electrode force, 3-4 spot welds were made per configuration to analyze both weld diameter and load strength using tensile-shear tests. Various process signals, such as electrode force and displacement, were monitored, and the data collected were essential for subsequent analysis using data-driven models and finite element simulations. These analyses aimed to identify the optimal range of parameters for model performance.

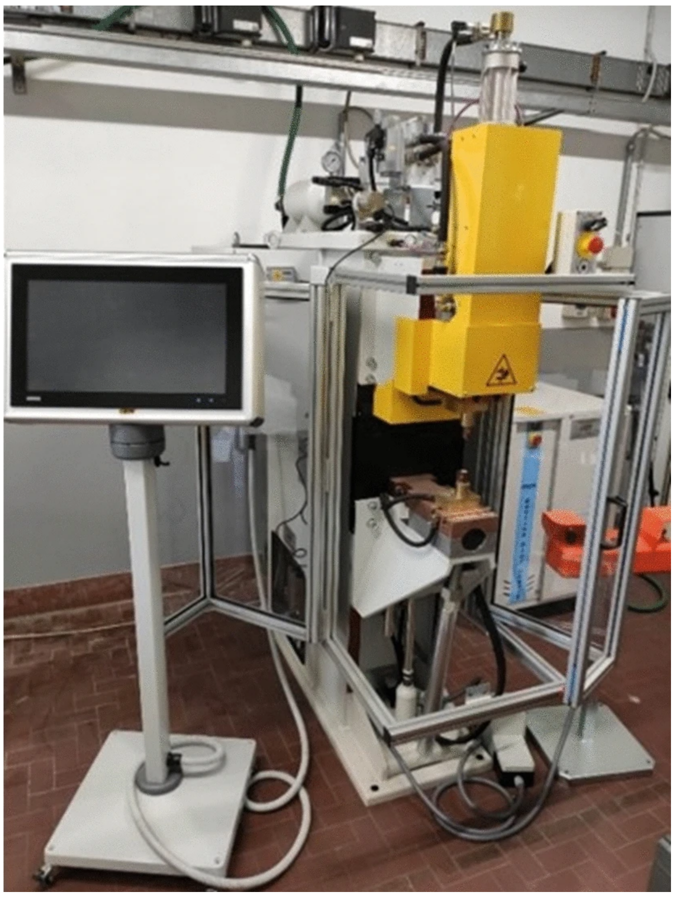


Figure 3.1: The Resistance Spot Welding Machine used for the experimental campaign[35]

3.2.2 Technical details and experimental tests

The experimental setup utilized a medium-frequency direct current RSW machine equipped with a TE700 control unit (TECNA®), with electrode displacement and force monitored using high-precision sensors. The welding materials consisted of DP590 steel strips (45 mm x 105 mm x 1 mm) following ISO 14273 standards, and the electrodes were made from a copper-chrome-zirconium alloy (ISO 5182 A2-2 Cu-Cr-Zr electrodes with a contact diameter of 5 mm and a truncated cone shape).

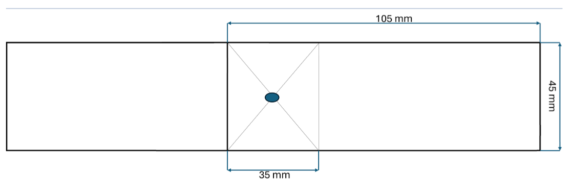


Figure 3.2: Specimen geometry according to ISO 14273

The weld time is 250 ms (upslope=25 ms, current time=150 ms, downslope=25 ms). Multiple sets of welding current $I = \{8, 9, 10, 11, 12, 12.5, 13, 14\}$ kA and welding pressure $P = \{1, 1.2, 1.4, 1.6, 1.8, 2, 2.2\}$ bar). Before testing, approximately 50 preliminary welds were conducted to condition the electrodes. The welding cycle parameters were established in accordance with ISO 14373:2015 standards, and each weld cycle included phases like squeeze time, weld time and hold time.

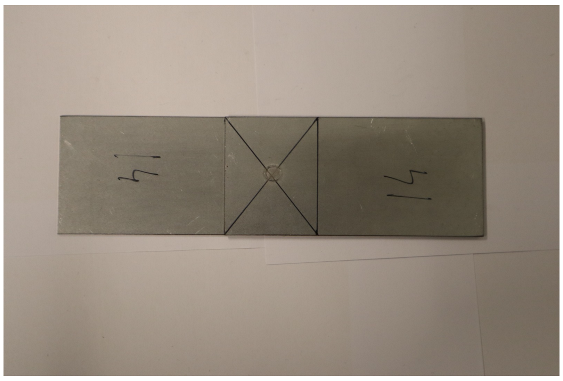


Figure 3.3: The Front Part of a weld

78 welds were conducted in total. On various tests, a shear tension test (STT) has been applied with a crosshead speed of $10\,\mathrm{mm\,min^{-1}}$, with peak load values recorded to evaluate weld performance. After the STT the weld has been cutted and the diameter of the nugget has been measured through macrographic examination. For a few specimens the weld nugget diameter has been measured without previously perform the STT, in order to have a more precise measurement.

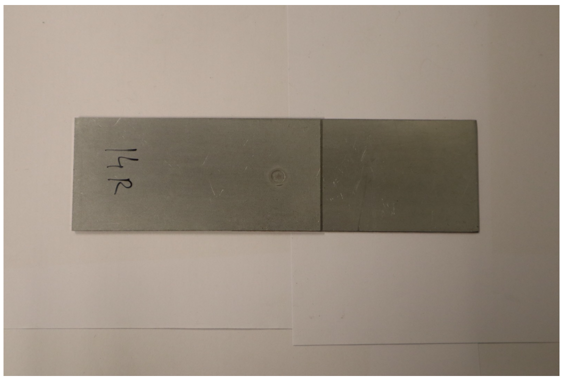


Figure 3.4: The Back Part of a weld

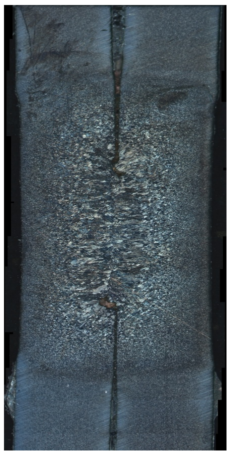


Figure 3.5: The Detail of a weld

In this figure we can observe a detail of a weld, specifically the detail of the weld number 14. As we can see, this weld have some problems of porosity caused by the expulsion[2] phenomenon. Specimen 14 is affected by expulsion because of excessive welding parameters[3].

[2]the ejection of molten material

[3]combination of too high current and too low pressure

3.3 The Start

The first draw scheme of the system has been made upon a list of prerequisites drawn up after a series of interviews to correctly identify the needs of the architecture. The system has been firstly designed as a combination of three actors: a frontend and two different databases, one specifically tailored to store data while the other to store file outputted from the RSW machinery.

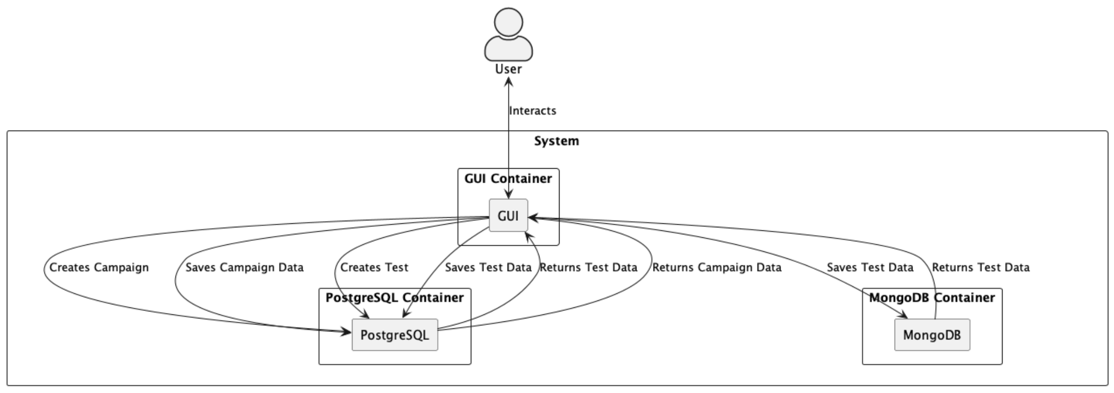


Figure 3.6: The first schema designed for the implementation

- Streamlit has been chosen to build the entire user interface of the framework. It communicates with the backend with a *direct* connection, as the system resides entirely on a docker cluster, to fetch data from PostgreSQL and retrieve files from MongoDB, using Python libraries like psycopg2 and SQLAlchemy for PostgreSQL connection, and pymongo for MongoDB connection. This has been designed as the only point of interaction between the user and the whole framework *All HTTP requests sent by the Streamlit app are directly sent to the caller.*

- PostgreSQL has been chosen to serve as the primary database for structured data, like campaigns and tests data and summarised results of the various tests, to speed up the data retrieve for the future Machine Learning integration. As already said before, it provides ACID-compliant transactions to ensure data reliability.

- MongoDB has been chosen instead for his capability of storing binary files and unstructured data. In this implementation, as the file is larger than 16MB *the maximum capability of BSON*, his native GridFS feature has been widely adopted.

3.4 The End

In the second (and for now last) version several changes has been made in the internal and external structure of the system. The most important one, at this level, is the adoption of an NGINX proxy server to filter and redirect requests to/from the frontend.

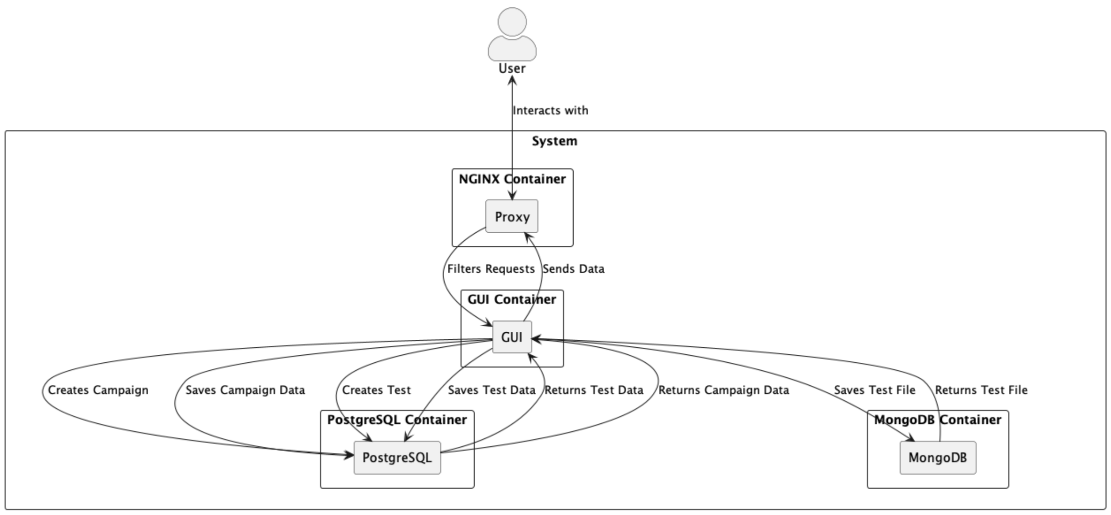


Figure 3.7: The last schema designed for the implementation

The adoption of an NGINX Proxy Server has been decided because of his capabilites, such as reverse proxy and load balancer, primarily ensuring security, scalability, and performance. It also manages HTTPS traffic, forwarding requests to the appropriate frontend container, and provides TLS/SSL termination: in fact, all incoming HTTP/HTTPS traffic passes through NGINX. It can provide also caching, compression, and other optimizations to improve performance.

It is also important to point out the fact that not only NGINX handles HTTP-S/SSL encryption and forwards secure requests to the appropriate services, but also authentication and authorization layers can be added (such as JWT tokens, OAuth, etc.) to manage secure access.

This final architecture has been designed also with an eye for future improvements, having a special consideration for *scalability* (NGINX can handle a large number of incoming requests and distribute the load across multiple servers), *security* (NGINX ensures secure connections and can manage firewalls, rate-limiting, and SSL certificates) and *separation of concerns* (PostgreSQL handles structured, transactional data while MongoDB efficiently manages unstructured file storage).

This system would be ideal for web applications that need secure data handling, robust file storage and a user-friendly interface for rapid deployment.

Chapter 4

Implementation and Testing

4.1 Introduction

In this chapter, the implementation of the entire system architecture discussed in the previous chapter will be explored in detail, as well as the code that processes and manages data from the experimental campaign.

The various components will be examined in depth, including the backend, the frontend, the reverse proxy, and the glue of the whole system, namely the Docker microservices architecture.

In the final part of the chapter, the integration with a simple Machine Learning algorithm will be introduced, which, while basic, is useful for testing the system integration.

4.2 PostgreSQL

Here is presented the first scheme for the relational database of PostgreSQL.

The central table of the whole architecture is the campaign. It is composed by:

- an incremental identification integer;
- a name that can be specified by the user;
- the mode of operation that the machine is set;
- the start date of the campaign
- the details about the sheets and the two electrodes, the upper and lower one.

The electrode is differentiated among material and shape, while the sheet can be recognized by the material and the coating. Various tables store also useful information about sheets and electrodes, thickness, length, width, and shape.

As various modes of control are present in the machine, but fixed to 6, separated tables for each control mode has been detailed, because every control mode of the machine requires some different settings.

The size of each column has been specifically tailored from prerequisites: some columns are in common between all control modes, some others are in common only with limited set of them, and other ones are unique. This choice has been made because PostgreSQL isn't able to handle flowlessy data with variable columns in the same table, and the idea of creating a unique larger table for all modes with the usage of nullable fields has been rejected to increase further scalabilities.

A test of a campaign is a single record of one of six tables designed in the Uppermost part of the image. Every table have an incremental identification integer, that it is used to identify various tests among the same control mode and campaing. The uniqueness of the identification of the test is guaranteed by the couple of campaign ID and test ID.

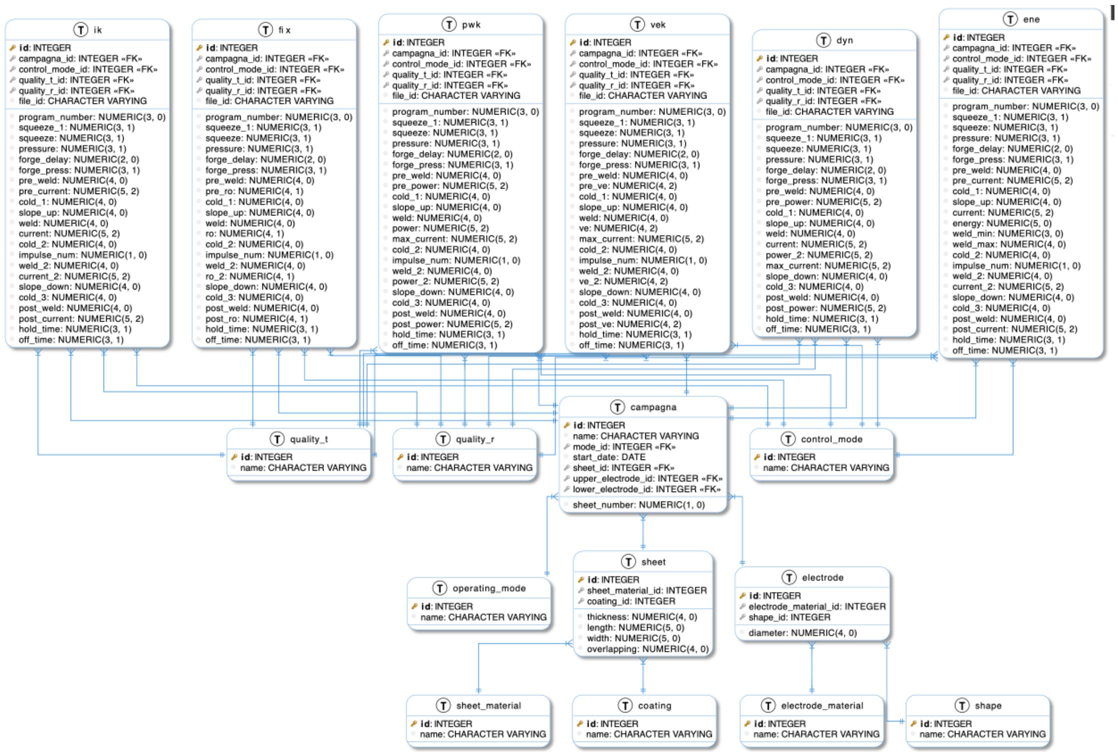


Figure 4.1: The first relational schema designed for PostgreSQL

In the final relational scheme implemented for PostgreSQL some things have been varied, comparing to original ones.

For example, a table has been inserted: to enable the possibility to integrate machine learning algorithms in the framework the table *provino results* has been added.

It contains not only *provino id* and *campaign id* but also it highlights some results of the test, such as breaking force, nugget diameter, and various notes that can be compiled by the operator at the machine, reporting user experience or some other useful comments on the result of the test.

It has been added also view named *all test*, and it is used by the frontend streamlit to regroup all tests in a single query, to enabling the possibility of quickly filtering out non-relevant information for the user selection.

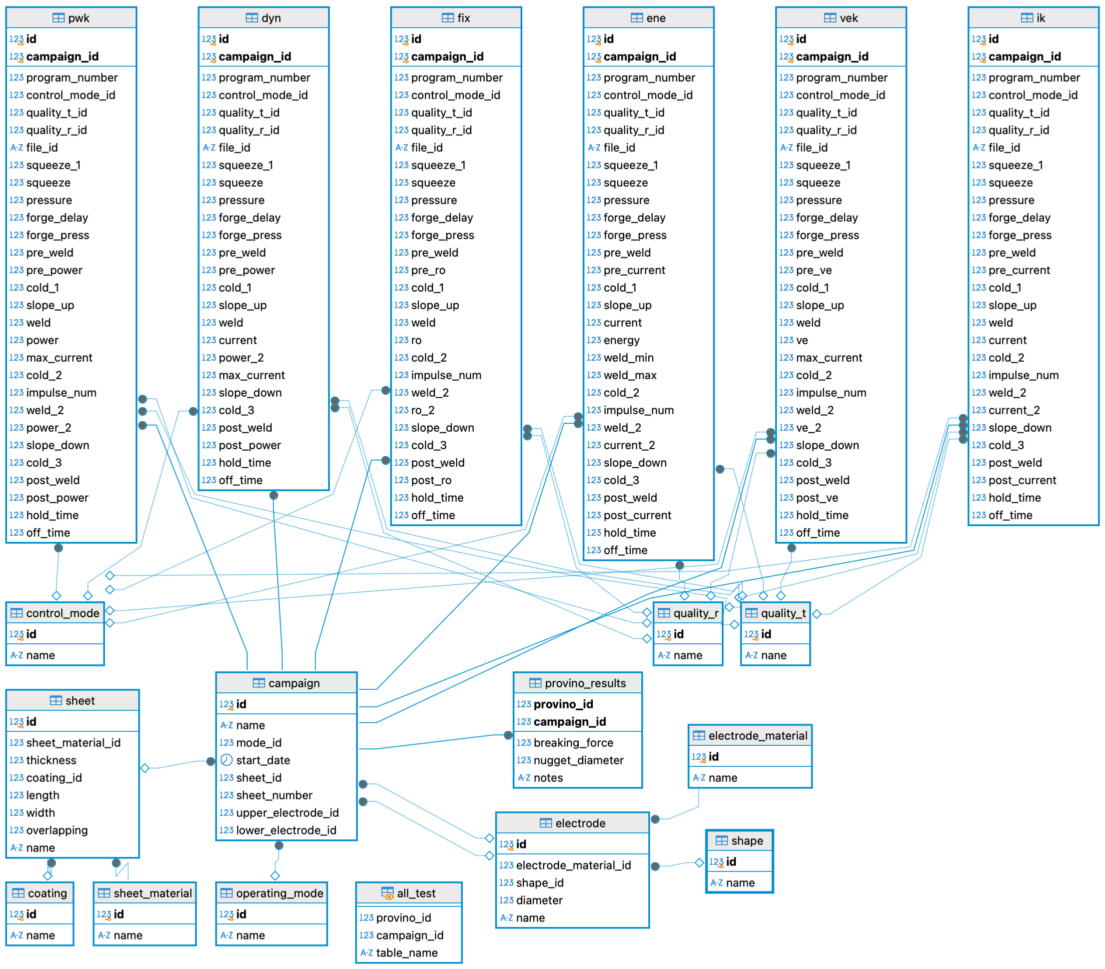


Figure 4.2: The final relational schema implemented for PostgreSQL

4.3 MongoDB

Here is detailed the simple internal structure of MongoDB, the NoSQL database inserted in the Architecture to store files outputted from RWS machine.

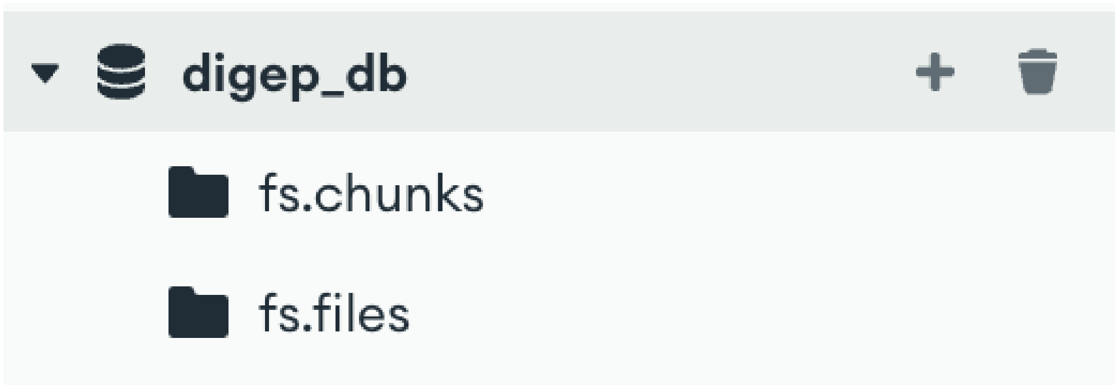
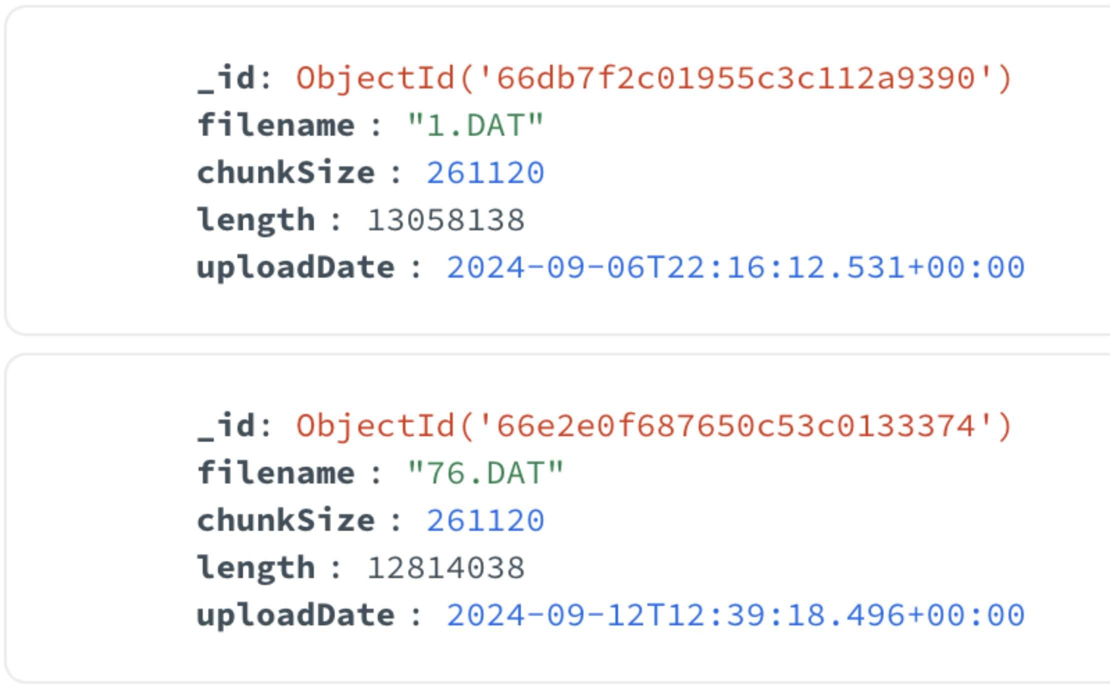


Figure 4.3: The simple GridFS schema implemented for MongoDB

This is the implementation of GridFS, the file storage system to store large files and binary data.

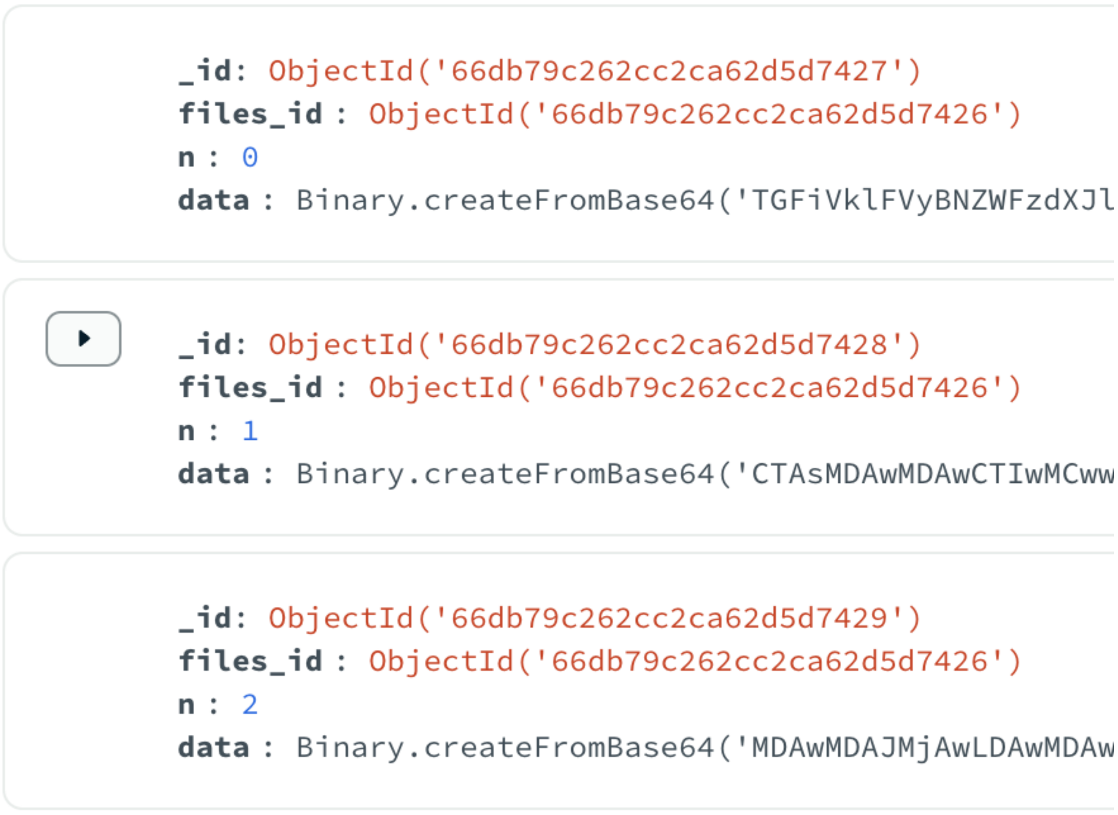


```
_id: ObjectId('66db7f2c01955c3c112a9390')
filename : "1.DAT"
chunkSize : 261120
length : 13058138
uploadDate : 2024-09-06T22:16:12.531+00:00
```

```
_id: ObjectId('66e2e0f687650c53c0133374')
filename : "76.DAT"
chunkSize : 261120
length : 12814038
uploadDate : 2024-09-12T12:39:18.496+00:00
```

Figure 4.4: An example of files loaded in MongoDB

This is a detail of how files are stored in MongoDB's GridFS storage.
The metadata file, such as file id, file name, size, length and date of upload, are inserted in the collection *fs.files*.



```
_id: ObjectId('66db79c262cc2ca62d5d7427')
files_id : ObjectId('66db79c262cc2ca62d5d7426')
n : 0
data : Binary.createFromBase64('TGFiVklFVyBNZWFzdXJl

_id: ObjectId('66db79c262cc2ca62d5d7428')
files_id : ObjectId('66db79c262cc2ca62d5d7426')
n : 1
data : Binary.createFromBase64('CTAsMDAwMDAwCTIwMCww

_id: ObjectId('66db79c262cc2ca62d5d7429')
files_id : ObjectId('66db79c262cc2ca62d5d7426')
n : 2
data : Binary.createFromBase64('MDAwMDAJMjAwLDAwMDAw
```

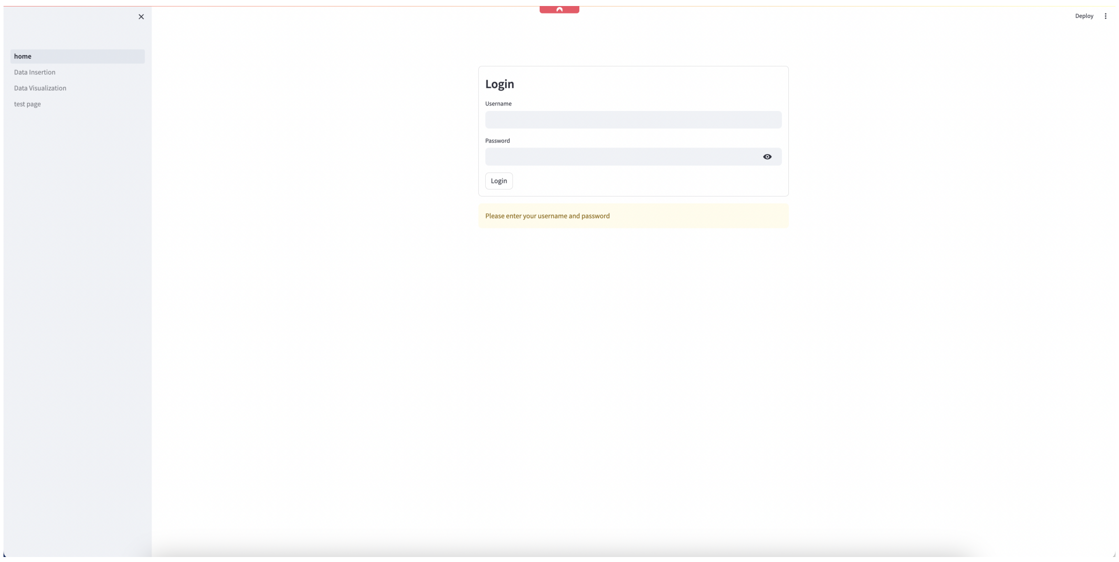
Figure 4.5: Some chunks of a file

The chunks of each file, instead, are stored in the collection *fs.chunks* and are ordered in ascending order.

4.4 Streamlit

Streamlit has been chosen as the frontend of the framework and has been designed to have a login screen, a page in which data can be inserted, a page in which data can be retrieved and visualized based on user inputs, and finally a page call Mark currently under the development yet, can be integrated a machine learning algorithm and an interface with a digital twin.

4.4.1 Login Page



Please enter your username and password

Figure 4.6: The Login Page

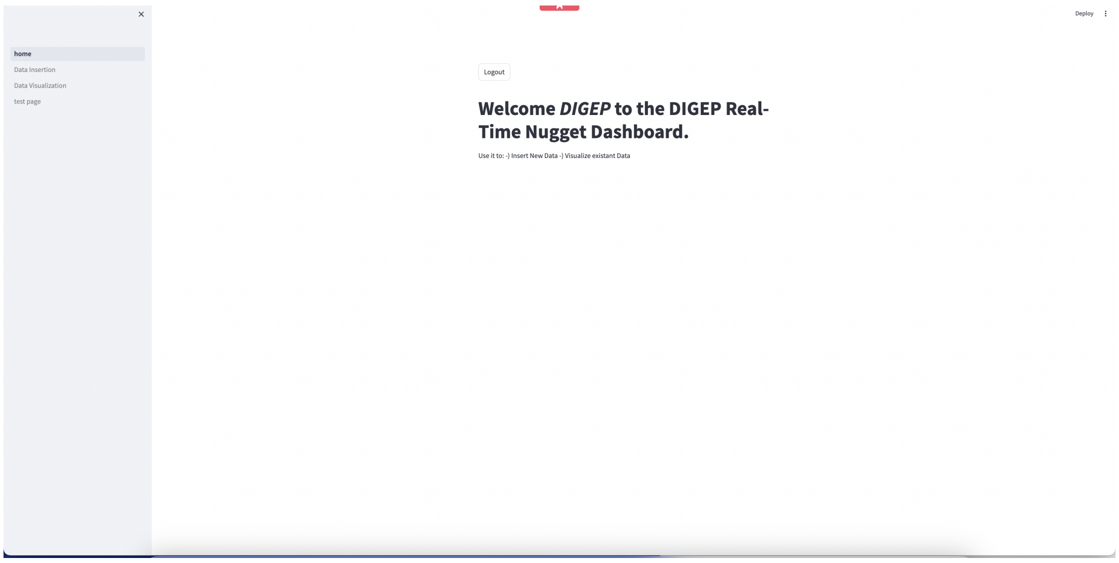
Using a login page on Streamlit provides several advantages, especially for applications that need to control access to certain features, data, or personalized content. Here are some key reasons why a login page is useful:

- Access control: as the framework should be accessible through internet, only authorized users should have the possibility to view or interact with sensitive data.
- Role-based access: There is also the possiblity to configure different roles (e.g., admin, user, machine operator) and a login system can help manage the permissions for each role, limiting the reachability of config pages or deep or premium functionality.
- Customization: different users can have tailored experiences, such as customized dashboards, settings, or data based on their preferences.

- Data Security: For apps dealing with private or sensitive information (e.g., in our case, with experimental studies), a login system helps ensure that this information is not exposed to unauthorized users.
- Profile management: Users can manage their own accounts, settings, and preferences. This can be useful for apps that require users to input personal or usage data.
- Session tracking: Login functionality can help manage user sessions, ensuring that users don't have to re-enter data multiple times during a session.
- Multi-user applications: If it is intended for the future to enable the possibility of creating multiple users and different roles, a login page helps manage multiple accounts and user data efficiently.
- Collaboration: You can create a collaborative environment where users can share or contribute data based on their login credentials.

To sum up, it has been chosen to insert a login page to have an additional layer of security, personalization and scalability to our Streamlit framework.

Here is the *real* Welcome Page of the frontend, if all has been successful:



Welcome *DIGEP* to the DIGEP Real-Time Nugget Dashboard.

Use it to: -) Insert New Data -) Visualize existant Data

Figure 4.7: The Login Window, if successful

4.4.2 Insertion Page

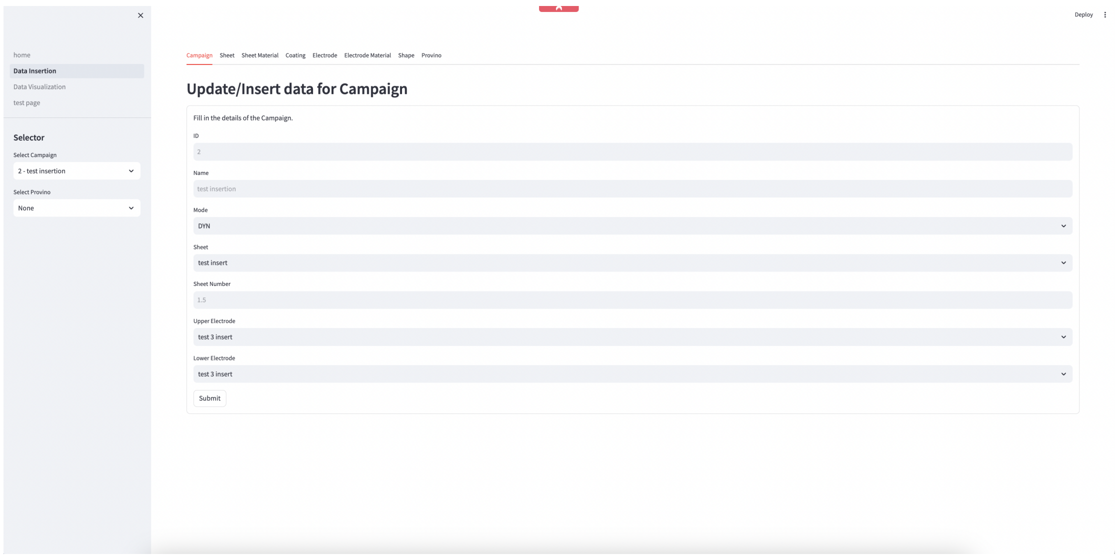
The frontend framework includes a data entry page that is accessible only after a successful login.

This page is divided into a sidebar on the left and a main content area on the right. Streamlit allows tables to either stretch across the entire page or be centered; in the image below, the table is fully stretched.

The main content is organized into several tabs, each enabling users to insert or update specific data related to the experimental campaign. Users must first select the test they want to modify or use as a base for data insertion from the sidebar.

The first tab allows users to input or update campaign-level data, while the other tabs, except the last one, enable them to modify or add new information related to the components of the campaign.

The final tab is dedicated to entering data for individual test samples (provini). Based on the campaign selected in the sidebar, a table with all relevant fields for that control mode is presented, allowing users to modify a single parameter and submit a new test to the database without re-entering all the data. In the provino tab, users can also upload the output file generated by the RSW machine, either during or after the initial test data entry.



Selector

Select Campaign: 2 - test insertion

Select Provino: None

Update/Insert data for Campaign

Fill in the details of the Campaign.

ID: 2

Name: test insertion

Mode: DYN

Sheet: test insert

Sheet Number: 1.5

Upper Electrode: test 3 insert

Lower Electrode: test 3 insert

Figure 4.8: The first draft for data insertion page

4.4.3 Visualization Page

The framework obviously also has the ability to display the data of individual test samples, if the user requests it.

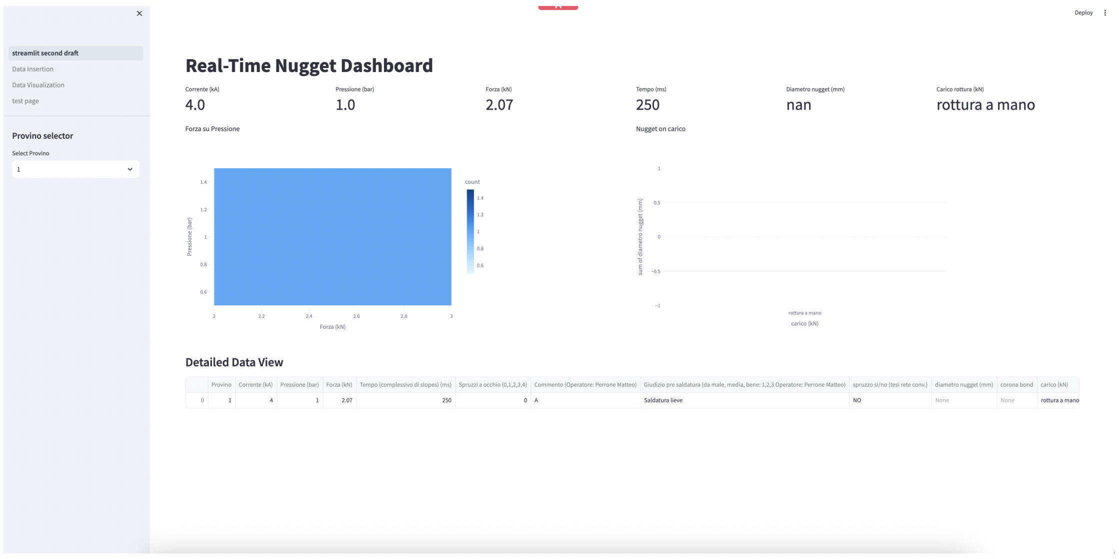
For the first version, we started with an Excel file that contained a summary of the test results, which we then converted into a CSV structure to make it easier to handle in the Python code. We then added the ability in the sidebar to select the sample for which the user wanted to view the data.

At the top section, all the important data from the test results is summarized, along with the relevant units of measurement (note that the system handles "not a number" values generated by the machine and any fields that contain comments from the user who analyzed the welding result instead of a numerical value).

In the middle section, there are two sample charts: one showing the relationship between force and pressure, and another showing the relationship between load and nugget diameter. In this case, the second chart is empty because the nugget broke with just hand pressure.

At the bottom of the page, as a debug feature, the corresponding row from the original Excel file is displayed to help better analyze the correlation between the displayed data and the original data.

This was also useful in the initial phase for validating the conversion of tabular data into visual data, but more importantly, for handling special cases already mentioned above.



Real-Time Nugget Dashboard

Corrente (kA)	Pressione (bar)	Forza (kN)	Tempo (ms)	Diametro nugget (mm)	Carico rottura (kN)
4.0	1.0	2.07	250	nan	rottura a mano

Detailed Data View

	Provino	Corrente (kA)	Pressione (bar)	Forza (kN)	Tempo (complessivo di slopes) (ms)	Spruzzi a occhio (0,1,2,3,4)	Commento (Operatore: Perrone Matteo)	Giudizio pre saldatura (da male, media, bene: 1,2,3 Operatore: Perrone Matteo)	spruzzo si/no (tesi rete conv.)	diametro nugget (mm)	corona bond	carico (kN)
0	1	4	1	2.07	250	0	A	Saldatura lieve	NO	None	None	rottura a mano

Figure 4.9: The first draft for data visualization page

A second version of the data visualization page was developed once the file upload to MongoDB was validated, featuring corrected graphs displaying the five key parameters of our case study (Force, Voltage, Current, Displacement, and Resistance) over time. Below is an example.

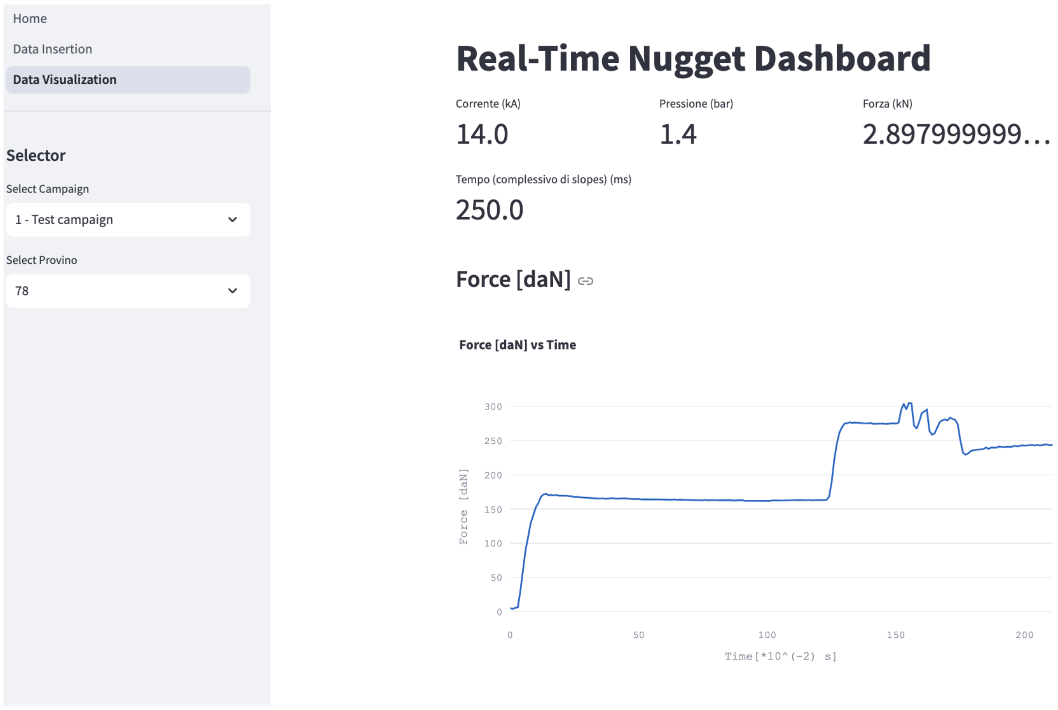


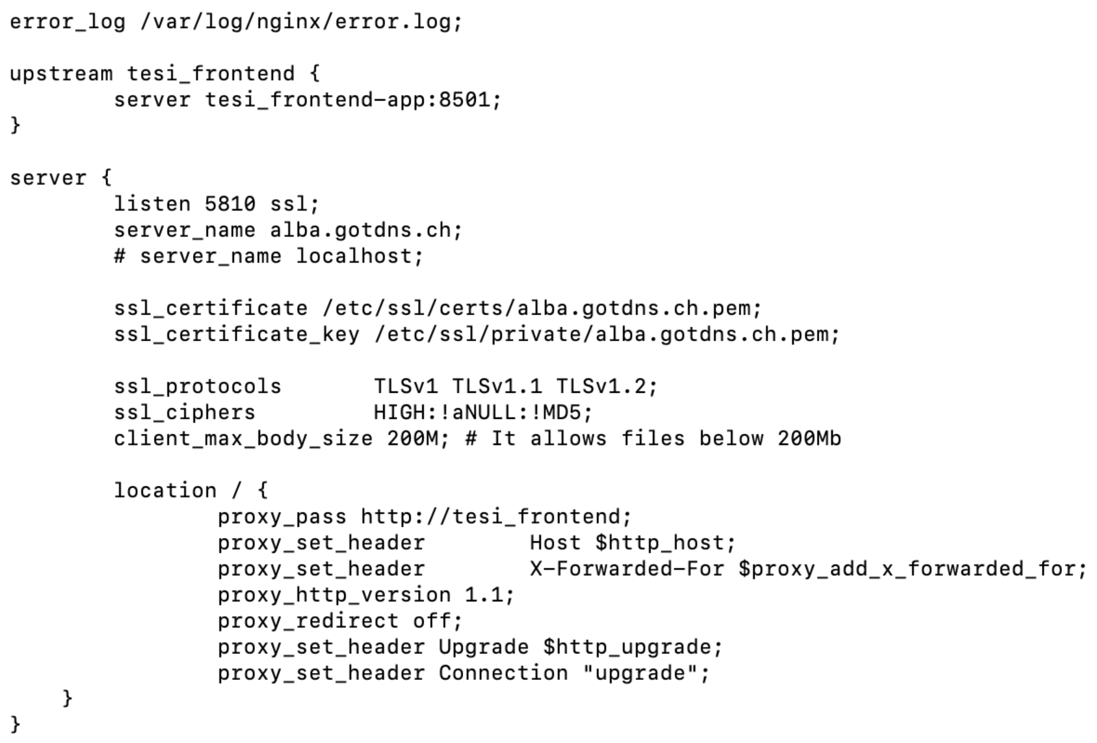
Figure 4.10: The final data visualization page

4.5 NGINX

NGINX is a web server that functions as a reverse proxy within this framework.

Its role is to filter connections directed to the frontend, thereby adding an additional layer of security for the only internet-accessible endpoint of the framework. This setup also enables the use of an HTTPS SSL connection with the frontend.

By default, Streamlit is launched using the HTTP protocol. While it can be configured to run directly over HTTPS, the developers advise against this in production environments. They recommend using a reverse proxy instead, as the direct HTTPS functionality has not yet been thoroughly tested or certified for security.



```
error_log /var/log/nginx/error.log;

upstream tesi_frontend {
        server tesi_frontend-app:8501;
}

server {
        listen 5810 ssl;
        server_name alba.gotdns.ch;
        # server_name localhost;

        ssl_certificate /etc/ssl/certs/alba.gotdns.ch.pem;
        ssl_certificate_key /etc/ssl/private/alba.gotdns.ch.pem;

        ssl_protocols       TLSv1 TLSv1.1 TLSv1.2;
        ssl_ciphers         HIGH:!aNULL:!MD5;
        client_max_body_size 200M; # It allows files below 200Mb

        location / {
                proxy_pass http://tesi_frontend;
                proxy_set_header        Host $http_host;
                proxy_set_header        X-Forwarded-For $proxy_add_x_forwarded_for;
                proxy_http_version 1.1;
                proxy_redirect off;
                proxy_set_header Upgrade $http_upgrade;
                proxy_set_header Connection "upgrade";
    }
}
```

Figure 4.11: NGINX config file

Here is the NGINX configuration: it listens externally on port 8501, while internally, the frontend is exposed on port 5810. The configuration includes the SSL certificates and sets up the reverse proxy functionality for the container hosting the frontend, allowing the transmission of files up to 200 MB.

4.6 Microservices

The entire framework was built as an aggregate of containers, utilizing the microservices paradigm to fully leverage the benefits outlined in previous chapters.

It currently consists of four containers:

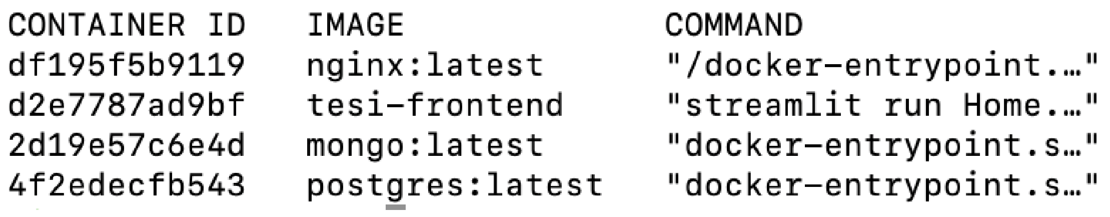
- PostgreSQL;
- MongoDB;
- NGINX;
- Streamlit - frontend.

In the future, thanks to the container-based architecture, it will be easy to add, for example, a container dedicated to advanced artificial intelligence predictions, to integrate or expand Machine Learning technique already tested, or a back-end system for a digital twin.

Additionally, an extra API layer between backend and frontend or Internet could be introduced if there is a need to open database access to external systems or integrate multiple front-end containers within the framework. At present, the NGINX container is solely used as a reverse proxy for the front-end.

The microservices architecture is completed by a subnet, isolated from the host machine's network, through which the various containers interact with each other.

With a potential expansion of the container network, Kubernetes can be employed for the orchestration, management, and monitoring of the framework.

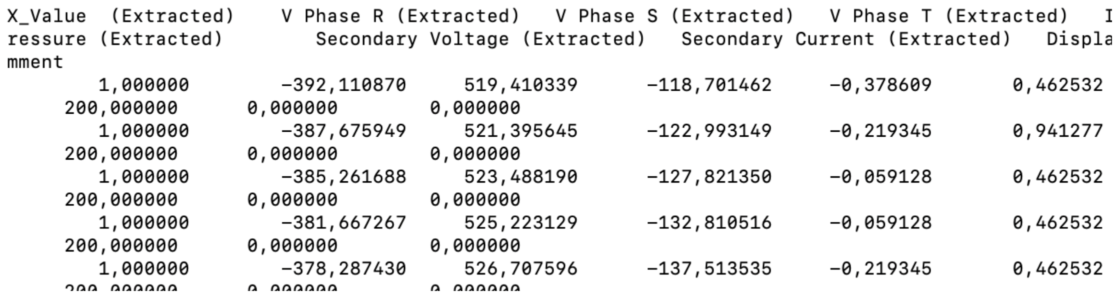


```
CONTAINER ID   IMAGE             COMMAND
df195f5b9119   nginx:latest      "/docker-entrypoint.…"
d2e7787ad9bf   tesi-frontend     "streamlit run Home.…"
2d19e57c6e4d   mongo:latest      "docker-entrypoint.s…"
4f2edecfb543   postgres:latest   "docker-entrypoint.s…"
```

Figure 4.12: A screenshot of the running containers

4.7 Data Acquisition

The data acquisition in RSW involves monitoring and recording various parameters to ensure the quality and consistency of the welds. Here's an overview of how data acquisition works in RSW and what kind of data is typically outputted in a file:



```
X_Value  (Extracted)    V Phase R (Extracted)   V Phase S (Extracted)   V Phase T (Extracted)   I
ressure (Extracted)        Secondary Voltage (Extracted)   Secondary Current (Extracted)   Displa
mment
     1,000000       -392,110870     519,410339      -118,701462     -0,378609       0,462532
  200,000000      0,000000        0,000000
     1,000000       -387,675949     521,395645      -122,993149     -0,219345       0,941277
  200,000000      0,000000        0,000000
     1,000000       -385,261688     523,488190      -127,821350     -0,059128       0,462532
  200,000000      0,000000        0,000000
     1,000000       -381,667267     525,223129      -132,810516     -0,059128       0,462532
  200,000000      0,000000        0,000000
     1,000000       -378,287430     526,707596      -137,513535     -0,219345       0,462532
  200,000000      0,000000        0,000000
```

Figure 4.13: A crop of an example file from TECNA TE700

Some Key Parameters monitored in RSW could be:

- Current: The electrical current passing through the electrodes. This is a critical parameter as it directly affects the heat generation.
- Voltage: The voltage across the welding electrodes.
- Force/Pressure: The pressure applied by the welding electrodes to hold the sheets together.
- Time/Duration: The duration for which the current is applied (also known as weld time).
- Electrode Displacement: The movement of the electrodes during the welding process.
- Temperature: The temperature at the weld spot.

Here is descripted the data acquisition process for a single weld:

- Sensors and Transducers: Various sensors are placed on the welding machine to measure current, voltage, force, and temperature. These sensors convert physical parameters into electrical signals.
- Data Logger/Acquisition System: The signals from the sensors are sent to a data acquisition system or data logger. This system collects the data at high frequencies to capture the details of the welding process.

- Signal Conditioning: Before recording, the raw signals may be conditioned to filter noise, amplify signals, and convert them into a usable format.
- Data Storage: The conditioned data is then stored in a file, which can be in formats like CSV, TXT, or proprietary formats specific to the data acquisition system used.

4.7.1 Typical Data Output Format

The output file from the data acquisition system typically contains time-stamped records of all the monitored parameters. Here is an example of what the data might look like in a CSV file:

Timestamp	**Current (A)**	**Voltage (V)**	**Force (N)**	**Weld Time (ms)**	**Electrode Displacement (mm)**	**Temperature (°C)**
2024-06-24 10:00:01.001	12000	2.5	3000	500	0.02	150
2024-06-24 10:00:01.002	11980	2.5	3000	500	0.02	152
2024-06-24 10:00:01.003	12010	2.5	3000	500	0.02	151

4.7.2 Analysis of Data

Once the data is collected, it can be analyzed to:

- Ensure Quality: Verify that the welds meet the required specifications.
- Detect Anomalies: Identify any deviations from the expected process parameters that could indicate potential issues.
- Optimize Processes: Adjust welding parameters for better performance and efficiency.
- Traceability: Maintain records for quality control and traceability purposes.

Various tools and software can be used for the data acquisition and analysis in RSW, such as:

- National Instruments LabVIEW
- MATLAB
- Custom software provided by welding equipment manufacturers
- Statistical process control (SPC) software for monitoring and controlling the welding process.

But it will presented later on, in Data Processing section
In summary, the data acquisition process in Resistance Spot Welding involves monitoring critical parameters, collecting and conditioning data, and storing it in a structured format for further analysis to ensure the quality and efficiency of the welding process.

4.8 Data Storage

In the data storage section, I want to analyze how the code handles saving data to the databases and how various queries for upserting and fetching data are performed.

Starting with the insertion process, the page features a select box in the sidebar.

Listing 4.1: Sidebar select box

```python
# Perform query for compaign.
campaign = sql_conn.query(
    'SELECT c.id, c.name, om.id as "mode_id", LOWER(om.name) as "mode_name", c.sheet_number FROM campaign c INNER JOIN operating_mode om on om.id = c.mode_id ORDER BY c.id DESC;', ttl=Timedelta("3s"))

# Combine id and name of campaigns
campaign_selection = campaign["id"].astype(str) + " - " + campaign["name"]

# top-level filters
campaign_filter = st.sidebar.selectbox(
    "Select Campaign", options=campaign_selection)
```

With this query, the framework filters the data requested by the user and retrieves information about the selected test sample (provino) and/or to be used as a placeholder.

Listing 4.2: Query to retrieve data for tabs

```python
# Perform query for mode
mode = sql_conn.query(
    'SELECT om.id, om.name FROM operating_mode om LEFT JOIN campaign c on om.id = c.mode_id ORDER BY om.id DESC;')
# Perform query for sheet
sheet = sql_conn.query(
    'SELECT * FROM sheet s ORDER BY s.id DESC;')
# Perform query for sheet_material
sheet_material = sql_conn.query(
    'SELECT * FROM sheet_material sm ORDER BY sm.id DESC;')
# Perform query for coating
coating = sql_conn.query(
    'SELECT * FROM coating c ORDER BY c.id DESC;')
# Perform query for shape
shape = sql_conn.query(
    'SELECT * FROM shape s ORDER BY s.id DESC;')
# Perform query for electrode
electrode = sql_conn.query(
    'SELECT * FROM electrode e ORDER BY e.id DESC;')
```

```
# Perform query for electrode_material
electrode_material = sql_conn.query(
    'SELECT * FROM electrode_material em ORDER BY em.id DESC;')

# Perform query for provini
all_provini = '''SELECT * FROM all_test at WHERE at.campaign_id = :
    c_id ORDER BY at.provino_id DESC;'''
with sql_conn.session as session:
    provini = session.execute(
        text(all_provini), {"c_id": int(campaign_filter.split(" - ")
    [0])}).fetchall()
```

Subsequently, using this data, the various tabs that make up the data entry page are populated, and the retrieved data is inserted as a placeholder for potential modification. Below, only the functionality of one tab is explained to avoid redundant analysis of very similar code.

Listing 4.3: Handling data of a tab

```
with campaign_tab:
    #write header
    st.header("Update/Insert data for Campaign")
    #prepare form data
    campaign_form = st.form("campaign_form")
    campaign_form.write("Fill in the details of the Campaign.")
    id = campaign_form.text_input("ID", placeholder=campaign["id"].
    values[0])
    name = campaign_form.text_input(
        "Name", placeholder=campaign["name"].values[0])
    mode_name = campaign_form.selectbox(
        "Mode", options=mode["name"].values.tolist())
    sheet_name = campaign_form.selectbox(
        "Sheet", options=sheet["name"].values.tolist())
    sheet_number = campaign_form.text_input(
        "Sheet Number", placeholder=campaign["sheet_number"].values
    [0])
    upper_electrode_name = campaign_form.selectbox(
        "Upper Electrode", options=electrode["name"].values.tolist())
    lower_electrode_name = campaign_form.selectbox(
        "Lower Electrode", options=electrode["name"].values.tolist())

    # Add a button to upload data
    if campaign_form.form_submit_button("Submit"):
        try:
            insert_query = '''INSERT INTO campaign (id, name, mode_id
    , sheet_id, sheet_number, upper_electrode_id, lower_electrode_id)
    VALUES (:id, :name, :mode_id, :sheet_id, :sheet_number, :
    upper_electrode_id, :lower_electrode_id)
```

```
            ON CONFLICT (id) DO UPDATE SET name = EXCLUDED.name,
    mode_id = EXCLUDED.mode_id, sheet_id = EXCLUDED.sheet_id,
    sheet_number = EXCLUDED.sheet_number, upper_electrode_id =
    EXCLUDED.upper_electrode_id, lower_electrode_id = EXCLUDED.
    lower_electrode_id;'''
            with sql_conn.session as session:
                session.execute(text(insert_query), {
                                "id": int(id),
                                "name": name,
                                "mode_id": mode[mode['name'] ==
    mode_name].values[0][0],
                                "sheet_id": sheet[sheet['name'] ==
    sheet_name].values[0][0],
                                "sheet_number": sheet_number,
                                "upper_electrode_id": electrode[
    electrode['name'] == upper_electrode_name].values[0][0],
                                "lower_electrode_id": electrode[
    electrode['name'] == lower_electrode_name].values[0][0]})
                session.commit()
            st.write("Campaign inserted successfully.")
        except Exception as e:
            st.write("An error occurred while inserting the Campaign.
    ")
            st.write(e)
```

Here, I add a step to explain how the file upload is managed in the "provino" tab, which differs from the other tabs.

Listing 4.4: Managing file upload

```
file_form = st.form("file upload", clear_on_submit=True)
    testfile = file_form.file_uploader(
        "Upload your file here", type=["csv", "xlsx", "dat"], key=1)

    if file_form.form_submit_button("Upload") and testfile is not
    None:
        try:
            # Insert on MongoDB
            database = mongo_conn["digep_db"]
            fs = GridFS(database)
            id = fs.put(testfile, filename=testfile.name).binary.hex
    ()
            testfile_bytes = testfile.getvalue()

            file_values = {
                "id": provino_id,
                "campaign_id": campaign_id,
                "file_id": id,
            }

            columns = ', '.join(file_values.keys())

            # Generate placeholder string for values
            keys_list = [f":{key}" for key in file_values.keys()]
            placeholders = ', '.join(keys_list)

            # Generate conflict update string
            conflict_update = ', '.join(
                [f"{key} = excluded.{key}" for key in file_values.keys() if key != 'id'])

            # Insert on PostgreSQL at the given provino_id and campaign_id
            insert_query = (
                f"""INSERT INTO {table_name} ({columns})
                VALUES ({placeholders})
                ON CONFLICT (id, campaign_id) DO UPDATE SET {conflict_update}"""
            )
            with sql_conn.session as session:
                session.execute(text(insert_query), file_values)
                session.commit()
            st.write(f"File uploaded with id: {id}")
        except Exception as e:
            st.write("An error occurred while uploading the file.")
            st.write(e)
        finally:
            st.rerun()
```

In the data visualization section, we see how data is retrieved from MongoDB once the corresponding sample is selected in the sidebar. The management of charts will be discussed in the data processing section.

4.9 Data Processing

In this section, I want to further analyze in details the process of data retrieving from MongoDB and data processing in the dashboard.

Listing 4.5: Import Python libraries and page config

```python
import streamlit as st
from pymongo import MongoClient
from bson import ObjectId
from sqlalchemy import text
import pandas as pd
from pandas import Timedelta
from gridfs import GridFS
import plotly.express as px  # interactive charts
import plotly.graph_objects as go
from streamlit_autorefresh import st_autorefresh
import os
import statistics
import matplotlib.pyplot as plt

st.set_page_config(
    page_title="Data Visualization Nugget Dashboard",
    page_icon="",
    layout="wide",
)
```

As every python file (as I have decided to include the data processing directly in Streamlit Web Interface) I have imported the necessary libraries to retrieve data and do the necessary calculations, then I have configured the page with Title and Icon.

Listing 4.6: Permission Checking

```python
if 'role' not in st.session_state or st.session_state["role"] != "
    DIGEP":
    st.error("You are not authorized to view this page.")
    st.stop()

# Setting streamlit autorefresh
st_autorefresh(interval=10000)

# dashboard title
st.title("Real-Time Nugget Dashboard Analyzer")
```

Then, just to ensure the privacy of data, I have checked the permission of the viewer to view correctly data, otherwise I will block him from further seeing values. After that I have set an automatic refresh for the page, ensuring that the data remains up-to-date, and configured the title for the page.

Listing 4.7: DB Connections

```python
# Initialize connection.
# Uses st.cache_resource to only run once.
@st.cache_resource
def init_connection():
    return MongoClient(**st.secrets["mongo"])

client = init_connection()

# Pull data from the collection.
# Uses st.cache_data to only rerun when the query changes or after 10
    min (600 secs).

@st.cache_data(ttl=600)
def get_data():
    db = client.mydb
    items = db.mycollection.find()
    items = list(items)  # make hashable for st.cache_data
    return items

# Initialize postgres connection.
sql_conn = st.connection("postgresql", type="sql")
```

After this last Streamlit configuration, I have initialized connections for MongoDB and PostgreSQL, to use the to further retrieve data from databases.

Notice that the MongoDB connection is cached. This choice has been done to prevent the system from being overloaded by a continuous request of init a connection from this page.

Listing 4.8: Data retrieve and sidebar config

```python
# Sidebar title
st.sidebar.header("Selector")

# Perform query for compaign.
# , ttl="10m")
campaign = sql_conn.query(
    'SELECT c.id, c.name FROM campaign c ORDER BY c.id ASC;')

# Combine id and name of campaigns
campaign_selection = campaign["id"].astype(str) + " - " + campaign["
   name"]

# top-level filters
campaign_filter = st.sidebar.selectbox(
    "Select Campaign", options=campaign_selection)

# Perform query for provini
```

```
all_provini = '''SELECT * FROM all_test at WHERE at.campaign_id = :
    c_id ORDER BY at.provino_id DESC;'''
with sql_conn.session as session:
    provini = session.execute(
        text(all_provini), {"c_id": int(campaign_filter.split(" - ")
    [0])}).fetchall()

# top-level filters
provino_filter = st.sidebar.selectbox(
    "Select Provino", options=list(map(lambda x: f"{x[0]}", provini))
    )

provino_data = [provino for provino in provini if provino[0] == (None
     if provino_filter == 'None' else int(
     provino_filter)) and provino[1] == int(campaign_filter.split(" -
    ")[0])][0]
provino_id = provino_data[0]
campaign_id = provino_data[1]
table_name = provino_data[2]

# Perform query for provino selected
provino_query = f'''SELECT * FROM {table_name}
    WHERE id = {provino_id} and campaign_id = {campaign_id};'''
df = sql_conn.query(provino_query, ttl=Timedelta("3s"))
```

Then it starts the page configuration, as the other pages, with a Sidebar on the left and a general container for the principal window of the page.

I have chosen not to cache the query for the campaign, as it will be executed only when the page is reloaded.

I have also reused the campaign number as a foreign key for a PostgreSQL view, to match only the index of that campaign selected for the Streamlit Selectbox. Then data for the selected Provino are retrieved.

Listing 4.9: Page config

```
# creating a single-element container
placeholder = st.empty()

# dataframe filter

# test to check df
with placeholder.container():

    # create six columns for values
    kpi1, kpi2, kpi3 = st.columns(3)

    # fill in those six columns with respective metrics or KPIs
    kpi1.metric(
        label="Current (kA)",
        value=df["current"].values[0],
    )

    kpi2.metric(
        label="Pressure (bar)",
        value=df["pressure"].values[0],
    )

    kpi3.metric(
        label="Force (kN)",
        value=df["pressure"].values[0]*2.07,
    )

    kpi4, kpi5, kpi6 = st.columns(3)

    kpi4.metric(
        label="Time (complessivo di slopes) (ms)",
        value=df["slope_up"].values[0] +
        df["slope_down"].values[0] + df["weld"].values[0],
    )
    '''
    kpi5.metric(
        label="Nugget Diameter (mm)",
        value=df["diametro nugget (mm)"].values[0],
    )

    kpi6.metric(
        label="Break force (kN)",
        value=df["carico (kN)"].values[0],
    )
    '''
```

Here I created the container for the main page, and started to configure the container with the requested rows and columns, with graphs and values according to specification retrieved

Listing 4.10: Load Data from files

```
        # parametrized to have the possibility to use and additional
        input to let the user choose
        n = 400

        if df['file_id'].values[0] is not None:
            fs = GridFS(mongo_conn['digep_db'])
            file_id = df['file_id'].values[0]
            with fs.get(file_id=ObjectId(file_id)) as dataset_file:
                with open(f'./tmp/{str(provino_id)}.DAT', 'wb') as f:
                    f.write(dataset_file.read())
            dataset = pd.read_csv(
                f'./tmp/{str(provino_id)}.DAT', sep='    ', engine='python', skiprows=22, encoding='cp1252')
            dataset = dataset.drop(['X_Value', 'Air Pressure (Extracted)',
                                     'V Phase S (Extracted)', 'V Phase T (Extracted)',
                                     'I Phase T (Extracted)', 'V Phase R (Extracted)', ' (Extracted)', 'V Phase R (Extracted)',
                                     'V Phase S (Extracted)', 'Temperature (Extracted)', 'Energy (Extracted)',
                                     'V Phase T (Extracted)', 'I Phase R (Extracted)', 'I Phase S (Extracted)', 'Comment'], axis=1)
            for j in list(dataset.columns):
                dataset[j] = dataset[j].str.replace(',', '.')
            for j in list(dataset.columns):
                dataset[j] = dataset[j].astype('float')
```

Here the calculation with the appropriate algorithm take place.

First, the parametrization valued is configured (ready to be trasformed in an input value, chosen by user, in this first run fixed to 400 samples, but easily changeable.

The file retrieved from MongoDB is now read, and the not necessary columns of values are dropped, to ensure the maximum speed achievable.

Listing 4.11: Data Calculation

```
    # take the variable i'm interested in - force (indice 0) - voltage (indice 1) - current (2) - displacement (3) - resistance (4)

    # create 5 columns for charts
    graphics = dict()

    graphics['force'] = pd.DataFrame()
    graphics['voltage'] = pd.DataFrame()
    graphics['current'] = pd.DataFrame()
    graphics['displacement'] = pd.DataFrame()
    graphics['resistance'] = pd.DataFrame()

    for index in range(0, 5):
        rows_signal = []
        rows_signal = dataset.iloc[:, index]

        # even though the process is at t=cost, file length will vary from test to test
        # make a list with the file length, I'll use to plot the signal

        length = len(rows_signal)
        # mean on N data to have a value every  per avere un
    valore ogni 2,5*10^-5 * N s (instead of one every 2,5*10^-5 s with
     FC 40 kHz)
        # to have a clearer signal
        signal_mean = []
        for k in range(1, length, n):
            end = k+min(n, length-k)
            start = k
            mean = statistics.mean(rows_signal[start:end])
            signal_mean.append(mean)
```

Then the mean of the value are calculated (according to the value configured before) and results are stored in Python lists.

Listing 4.12: Data Plot

```
        # graphic
        # force (0) - voltage (1) - current (2) - displacement
    (3) - resistance (4)

        string = ""

        if index == 0:
            string = "Force [daN]"
            index_dict = 'force'
        elif index == 1:
            string = "Voltage [V]"
            index_dict = 'voltage'
        elif index == 2:
            string = "Current [A]"
            index_dict = 'current'
        elif index == 3:
            string = "Displacement [um]"
            index_dict = 'displacement'
        elif index == 4:
            string = "Resistance [Ohm]"
            index_dict = 'resistance'
        else:
            string = ""

        fig = st.columns(1)
        st.markdown(f"### {string}")
        fig = go.Figure()
        length_list = [x for x in range(0, len(signal_media))]
        meas_df = pd.DataFrame(
            {string: signal_media, 'data': length_list})
        fig.add_trace(go.Scatter(
            x=meas_df['data'], y=meas_df[string], mode='lines',
    name='1 weld - media'))
        fig.update_layout(
            title=f'{string} vs Time',
            xaxis_title='Time[*10^(-2) s]',
            yaxis_title=string,
            font=dict(
                family="Courier New, monospace",
                size=18,
                color="RebeccaPurple"
            )
        )
        st.write(fig)
```

Here the mean values are drawn into graphs and showed to the user, using figures from a Python library called *plotly*

Listing 4.13: Data print test

```
        #Just a test, to print out the columns to doble check values
        st.markdown("### Detailed Data View")
        st.dataframe(df)
```

As a final confirmation in this first version of the Data Visualizer, raw values from the python *DataFrame* are shown to the user.

4.10 Integration with Machine Learning

Since the code used to pass files and prepare results for the application of a Machine Learning algorithm is virtually identical to the code developed for data visualization—except for the extraction of only the force, rather than the five parameters monitored according to the RSW process—the code will not be included at this stage to avoid redundancy and streamline the document. In the Machine Learning section, the input file containing the necessary data for processing is simply taken, and the data is split: 80% is used to train the model, while the remaining 20% is used to test the training. At this point, a linear regression algorithm is applied, and its performance is tested by comparing it with the actual results. An example of the graphical output can be found in Chapter 2.

Chapter 5

Conclusion

5.1 Thesis Project

In this thesis, the development of a data management system framework for interfacing a digital twin with an RSW welding machine was analyzed, with the potential to integrate Machine Learning algorithms for predicting welding outcomes by using the historical data from tests conducted and recorded in the system.

Starting from the study of the theoretical principles behind the Digital Twin in industry, modern backend and frontend techniques, and microservices architecture as opposed to monolithic architecture, an experimental testing campaign was conducted to collect data for this study. Once sufficient data was gathered for processing the results, the requirements for the framework development were analyzed in agreement with the client.

Data storage is managed using two databases with different data handling paradigms: for test-related data, the use of a traditional SQL database like PostgreSQL ensures a certain level of relational correlation between various campaigns, individual test samples (provini), and their results. On the other hand, for file storage, MongoDB, a NoSQL database, allows the raw files output by the welding machine to be stored and made available for further computation.

Data processing is performed in real-time, utilizing an algorithm that has been refined to filter out impurities typically found in the raw sensor data output. The use of averaging and the concurrent parameterization of the algorithm's application allow for rapid adjustments in the calculation based on specific needs.

The frontend has been kept simple to ensure resilience and to leave open the possibility for future developments and integrations. Nevertheless, a series of features, such as user login for a personalized experience, the retrieval of the selected test sample data as a placeholder, and the graphical representation of the algorithm's computed results, allow even users less familiar with technology to quickly record data and evaluate the machine's computed test results from the raw data.

5.2 Future Developments

My work in this thesis aimed at the groundwork for the development of a fully functional Digital Twin to support an industrial RSW welding machine. This framework will form the backbone of the future Digital Twin, and the development has been specifically tailored to be easily integrable and expandable based on evolving needs.

One simple enhancement, once it is fully implemented on a remotely accessible server, could be the development of a user registration and login system through a form, allowing users to interact autonomously with the machine. Of course, this must be carefully orchestrated with role-based access management, providing different levels of privileges based on the type of account created.

Once this is done the user experience can be further customized with personalized style and graphical settings.

Future developments could focus on integrating advanced Deep Learning algorithms, as well as fully-fledged AI predictive model techniques for predicting future welding results, potentially using multiple methods simultaneously, weighted as needed.

Another possible future development, outside the scope of this study, could involve strengthening cybersecurity techniques for communication between the internet and the frontend, once the system is deployed in production.

Bibliography

[1] Hongyan Zhang and Jacek Senkara. *Resistance Welding - Fundamentals and Applications*. 2nd Edition. CRC Press, 2011. ISBN: 978-0-429-09726-3. DOI: `10.1201/b11752` (cit. on p. 1).

[2] TWI Global. *What is Spot Welding? (A Complete Welding Process Guide)*. 2024. URL: `https://www.twi-global.com/technical-knowledge/faqs/what-is-spot-welding` (visited on 05/06/2024) (cit. on p. 1).

[3] Ichwan Fatmahardi, Mazli Mustapha, Azlan Ahmad, Mohd Nazree Derman, Turnad Lenggo Ginta, and Iqbal Taufiqurrahman. «An Exploratory Study on Resistance Spot Welding of Titanium Alloy Ti-6Al-4V». In: *Materials* 14.9 (2021). ISSN: 1996-1944. DOI: `10.3390/ma14092336`. URL: `https://www.mdpi.com/1996-1944/14/9/2336` (cit. on p. 3).

[4] World Auto Steel. *Hybrid Welding Procedures*. 2024. URL: `https://ahssinsights.org/joining/adhesive-joining/hybrid-welding-procedures/` (visited on 08/12/2024) (cit. on p. 4).

[5] Diego M. Botín-Sanabria, Adriana-Simona Mihaita, Rodrigo E. Peimbert-García, Mauricio A. Ramírez-Moreno, Ricardo A. Ramírez-Mendoza, and Jorge de J. Lozoya-Santos. «Digital Twin Technology Challenges and Applications: A Comprehensive Review». In: *Remote Sensing* 14.6 (2022). ISSN: 2072-4292. DOI: `10.3390/rs14061335`. URL: `https://www.mdpi.com/2072-4292/14/6/1335` (cit. on p. 5).

[6] Tera Automation. *Digital Twin e transizione verde: la questione dei dati*. 2022. URL: `https://www.tera-automation.com/blog/333-digital-twin-e-transizione-verde-la-questione-dei-dati` (visited on 04/12/2024) (cit. on p. 5).

[7] Leon Eversberg. *What Is a Digital Twin?* 2023. URL: `https://medium.com/geekculture/what-is-a-digital-twin-46ad1f549cce` (visited on 04/13/2024) (cit. on p. 7).

[8] Zeeshan Ali, Raheleh Biglari, Joachim Denil, Joost Mertens, Milad Poursoltan, and Mamadou Traoré. «From modeling and simulation to Digital Twin: evolution or revolution?» In: *SIMULATION* 100 (Mar. 2024). DOI: `10.1177/00375497241234680` (cit. on p. 8).

[9] Chris Richardson. *Microservice Architecture pattern.* 2023. URL: `https://microservices.io/patterns/microservices` (visited on 06/24/2024) (cit. on p. 11).

[10] Alex M. *Docker vs Kubernetes: The Ultimate Guide to Containerization and Orchestration.* 2024. URL: `https://1gbits.com/blog/docker-vs-kubernetes/` (visited on 10/02/2024) (cit. on p. 16).

[11] Docker Inc. *What Is Docker?* 2024. URL: `https://docs.docker.com/get-started/docker-overview/` (visited on 05/29/2024) (cit. on pp. 16, 18).

[12] Daniel Adetunji. *How Docker Containers Work – Explained for Beginners.* 2023. URL: `https://www.freecodecamp.org/news/how-docker-containers-work/#:~:text=A%20container%20is%20a%20lightweight,%20standalone,%20and%20executable,that%20they%20can%20run%20consistently%20across%20different%20environments.` (visited on 06/07/2024) (cit. on p. 17).

[13] Artkai. *Comparing Docker, Containerd, CRI-O, and Runc.* 2023. URL: `https://artkai.io/blog/best-containerization-tools` (visited on 07/16/2024) (cit. on p. 17).

[14] Mehmet Ozkaya. *When to use Monolithic Architecture.* 2023. URL: `https://medium.com/design-microservices-architecture-with-patterns/when-to-use-monolithic-architecture-57c0653e245e` (visited on 04/27/2024) (cit. on p. 19).

[15] Altexoft. *Comparing Database Management Systems: MySQL, PostgreSQL, MSSQL Server, MongoDB, Elasticsearch, and others.* 2023. URL: `https://www.altexsoft.com/blog/comparing-database-management-systems-mysql-postgresql-mssql-server-mongodb-elasticsearch-and-others/` (visited on 05/25/2024) (cit. on p. 23).

[16] Ryan Boyd & William Lyon Michael Hunger. *RDBMS & Graphs: Why Relational Databases Aren't Always Enough.* 2016. URL: `https://neo4j.com/blog/rdbms-graphs-why-relational-databases-arent-enough/` (visited on 04/07/2024) (cit. on p. 23).

[17] Sumeet Shukla. *PostgreSQL Architecture.* 2023. URL: `https://medium.com/@sumeet.k.shukla/postgresql-architecture-6df259dc1145` (visited on 06/02/2024) (cit. on p. 26).

[18] Danish Siddiq. *Learning NoSQL — NoSQL Database Designing.* 2019. URL: `https://medium.com/tech-tajawal/nosql-modeling-database-structuring-part-ii-4c364c4bc17a` (visited on 05/30/2024) (cit. on p. 29).

[19] Akshay Mungekar. *Data Storage and Management Project.* Feb. 2019. DOI: `10.13140/RG.2.2.27354.18880` (cit. on p. 30).

[20] Noel. *How To Optimize MongoDB Performance & Security.* 2013. URL: `https://medium.com/@noel.B/how-to-optimize-mongodb-performance-security-6fd3ba1304c1` (visited on 07/02/2024) (cit. on p. 31).

[21] Viacheslav Petrenko. *Understanding Contemporary Web Application Architecture: Key Components, Best Practices, and Beyond.* 2024. URL: `https://litslink.com/blog/web-application-architecture` (visited on 07/05/2024) (cit. on p. 32).

[22] Arvindra Sehmi. *Introduction to Streamlit and Streamlit Components.* 2022. URL: `https://auth0.com/blog/introduction-to-streamlit-and-streamlit-components/` (visited on 04/25/2024) (cit. on p. 35).

[23] Jan Wiśniewski. *Comprehensive Guide to Reverse Proxy Servers: Benefits, Use Cases, and Key Configurations.* 2024. URL: `https://infatica.io/blog/reverse-proxy-servers/` (visited on 08/22/2024) (cit. on p. 38).

[24] Amit Kumar Shinde. *Understanding Nginx As A Reverse Proxy.* 2021. URL: `https://medium.com/globant/understanding-nginx-as-a-reverse-proxy-564f76e856b2` (visited on 07/21/2024) (cit. on p. 40).

[25] Ankit Jain. *Caddy vs Nginx vs Apache - Comparison of Web Servers (2024).* 2024. URL: `https://json-server.dev/web-servers-compared-caddy-nginx/` (visited on 08/30/2024) (cit. on p. 40).

[26] Fahri Yesil. *Network Protocols: A Comprehensive Guide.* 2023. URL: `https://medium.com/@fahriiyesill/demystifying-networking-protocols-a-comprehensive-guide-171c81ac07ae` (visited on 09/07/2024) (cit. on p. 41).

[27] Rishikesh roy. *Load Balancer vs Reverse Proxy vs API Gateway.* 2023. URL: `https://www.linkedin.com/pulse/load-balancer-vs-reverse-proxy-api-gateway-rishikesh-roy/` (visited on 07/31/2024) (cit. on p. 41).

[28] Maicon Saturno, Vinícius Pertel, and Fernando Deschamps. «Proposal of an automation solutions architecture for Industry 4.0». In: (July 2017) (cit. on p. 43).

[29] Kathleen Siddell. *Computer Vision Will be Essential to the Industry 5.0 Era.* 2023. URL: `https://alwaysai.co/blog/industry5.0` (visited on 06/16/2024) (cit. on p. 45).

[30] Mohsen Soori, Behrooz Arezoo, and Roza Dastres. «Artificial intelligence, machine learning and deep learning in advanced robotics, a review». In: *Cognitive Robotics* 3 (2023), pp. 54–70. ISSN: 2667-2413. DOI: https://doi.org/10.1016/j.cogr.2023.04.001. URL: https://www.sciencedirect.com/science/article/pii/S2667241323000113 (cit. on p. 47).

[31] Yulia Gavrilova. *Reinforcement Learning: How It Works*. 2024. URL: https://serokell.io/blog/reinforcement-learning-guide (visited on 09/26/2024) (cit. on p. 47).

[32] Open Text. *What is Machine Learning*. 2024. URL: https://www.opentext.com/what-is/machine-learning (visited on 05/24/2024) (cit. on p. 48).

[33] Hasan Tercan and Tobias Meisen. «Machine learning and deep learning based predictive quality in manufacturing: a systematic review». In: *Journal of Intelligent Manufacturing* 33.7 (2022), pp. 1879–1905. ISSN: 1572-8145. DOI: 10.1007/s10845-022-01963-8. URL: https://doi.org/10.1007/s10845-022-01963-8 (cit. on p. 48).

[34] mongoDB. *GridFS*. 2024. URL: https://www.mongodb.com/docs/drivers/node/current/fundamentals/gridfs/ (visited on 08/30/2024) (cit. on p. 55).

[35] Luigi Panza, Giulia Bruno, Gabriel Antal, Manuela De Maddis, and Pasquale Russo Spena. «Machine learning tool for the prediction of electrode wear effect on the quality of resistance spot welds». In: *International Journal on Interactive Design and Manufacturing (IJIDeM)* 18.7 (2024), pp. 4629–4646. ISSN: 1955-2505. DOI: 10.1007/s12008-023-01733-7. URL: https://doi.org/10.1007/s12008-023-01733-7 (cit. on p. 62).