ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CESENA CAMPUS

Department of Computer Science and Engineering - DISI
Second-cycle Degree in Digital Transformation Management
Class: LM-91

Retail Inventory Management in the Digital Era: A Web-Based Solution for In-Store Retail Operations

Graduation thesis in
Software Engineering

Supervisor
Prof. Giovanni Ciatto

Candidate
Riccardo Leonelli

2nd Session
Academic Year 2024-2025

Abstract

This thesis presents the design, implementation, and evaluation of *Stock Uniters*, a web-based stock management system developed for Poltronesofà. The project was motivated by the limitations of the previous Excel-based workflow, which relied on manual updates, suffered from delays in communication, and lacked accountability and traceability.

The developed solution replaces fragmented spreadsheets with a centralized, full-stack application. The system features a Vue.js Single Page Application frontend and a FastAPI backend integrated with the company's Oracle infrastructure. Core functionalities include real-time stock registration, document scanning and uploads, role-based dashboards, and secure authentication through JWT. By embedding business rules directly into application logic, the system not only supports daily operations but also enforces organizational consistency.

A pilot deployment in five stores demonstrated significant improvements in efficiency and data reliability. Key performance indicators (KPIs), such as stock registration delay, document completeness, and inventory accuracy, showed measurable progress compared to the Excel-based process. Moreover, the system contributed to improved managerial visibility and enhanced accountability at the store level.

Beyond technical contributions, this work highlights the broader role of digital tools in enabling organizational change. The project illustrates how the introduction of *Stock Uniters* served as a concrete step in Poltronesofà's digital transformation, demonstrating that even focused solutions can yield meaningful benefits in efficiency, traceability, and cultural alignment towards data-driven decision-making.

Acknowledgements

I would like to express my deepest gratitude to my family, whose constant support, patience, and encouragement have given me the strength to face this journey. Their example and trust have been a precious guide throughout both my academic and personal path.

Thanks also go to my friends, who have stood by my side with understanding and lightheartedness, offering support during challenging times and sharing with me the satisfaction of every milestone achieved. Their presence has made these years not only a path of study, but also a rich and unforgettable human experience.

Disclaimer on the Use of Generative AI

This thesis may include content that was generated or refined using generative artificial intelligence (GenAI) tools such as ChatGPT by OpenAI. These tools were used solely to assist with tasks such as language correction, summarization, idea generation, or code suggestions. All outputs were critically reviewed and edited by the author to ensure academic integrity and originality. The author remains fully responsible for the content, analysis, and conclusions presented in this work.

Contents

List of Figures

Chapter 1

Introduction

In recent years, digital transformation has become a central theme in the evolution of retail organizations. Companies increasingly recognize that competitiveness does not depend solely on the quality of products, but also on the efficiency, transparency, and responsiveness of the processes that support daily operations. Inventory management, in particular, represents a critical domain where digital solutions can replace manual and fragmented practices, transforming them into structured, real-time workflows that benefit both local staff and upper management.

Poltronesofà, as a leading furniture retailer with a widespread store network, faces the challenges typical of distributed organizations: heterogeneous operational practices, delays in communication, and the risk of incomplete or inconsistent data. Before this project, stock management in stores was handled mainly through Excel spreadsheets, manually updated and periodically shared with area managers. While functional at a basic level, this approach was prone to errors, lacked traceability, and delayed decision-making. Such inefficiencies demonstrated the need for a digital tool capable of standardizing workflows, improving data reliability, and providing immediate visibility across the network.

The thesis addresses this challenge through the design, implementation, and evaluation of *Stock Uniters*, a web-based stock management application tailored to the operational needs of Poltronesofà. The project is not limited to software development in a narrow sense: it represents an active contribution to the company's digital transformation, showing how technological choices can be aligned with organizational goals. By embedding business rules (such as warranty validation, conditional acceptance of deliveries, and mandatory justifications for adjustments) directly into the application logic, the system acts not only as a support tool but also as an instrument of governance and cultural change.

From a technical perspective, the project demonstrates the use of a modern full-stack architecture that combines a Vue.js Single Page Application for the frontend

with a FastAPI backend and Oracle integration. The solution leverages role-based authentication, document scanning, and responsive dashboards to support daily operations while ensuring accountability and security. From an organizational perspective, the system has enabled real-time stock registration, reduced reporting delays, and improved transparency between store staff and area managers.

A central managerial motivation behind the project was the improvement of specific Key Performance Indicators (KPIs) that directly reflect the efficiency and reliability of stock management. In particular, the thesis focuses on reducing stock registration delays, increasing document completeness, minimizing mismatches between physical and digital stock, and improving overall inventory accuracy. These KPIs translate the company's strategic goals—such as transparency, accountability, and timely decision-making—into measurable outcomes, providing a concrete benchmark for evaluating the impact of the digital solution.

The structure of the thesis reflects the evolution of the project itself: from the analysis of requirements, through the design and implementation phases, to the evaluation of its impact in a pilot deployment. The final discussion highlights both the contributions achieved and the limitations encountered, pointing to possible directions for future improvement.

In general, this work illustrates how a targeted digital solution, developed with a clear understanding of business processes, can act as a catalyst for efficiency, accountability, and data-driven management. At the same time, it underlines the role of the developer not only as a programmer, but also as a participant in organizational innovation and change.

Chapter 2

Background

In today's retail landscape, efficient inventory management is essential for delivering consistent customer experiences, especially in organizations with a widespread and distributed store network. As product offerings diversify and operational complexity increases, the ability to monitor, control, and respond to stock movements in real time has become a strategic priority [RDT01]. This is particularly true for companies like Poltronesofà, where customer service excellence and operational consistency across multiple locations are key differentiators in a highly competitive market.

This chapter provides the necessary context for understanding the motivation and scope of the system developed in this thesis. It begins by introducing Poltronesofà, a leading Italian furniture brand recognized for its handcrafted products and expansive retail presence. The overview highlights the company's emphasis on in-store customer experience and the logistical challenges that arise from operating across multiple countries and store formats. Within this context, particular attention is given to the internal handling of post-sales support materials, such as maintenance kits, which—although secondary to the primary product offering—play a crucial role in supporting brand quality and service expectations.

The chapter then explores the role of **web applications** in enabling modern business workflows. It explains how browser-based systems provide a practical and scalable solution for managing operations across distributed environments, allowing for real-time access to centralized data without requiring local installations or complex hardware infrastructure. This is especially relevant for retail operations, where store staff and managers must be able to interact with systems quickly and intuitively while serving customers.

Following this technological overview, the chapter outlines the **specific use case** that motivated the development of the solution presented in this thesis. It identifies the limitations of existing manual and semi-digital methods used to manage stock, including spreadsheets, email exchanges, and verbal communication,

and describes how these methods led to inefficiencies, inconsistencies, and a lack of visibility. By analyzing the workflows and responsibilities of different user roles within the organization, the chapter defines the operational pain points that the proposed system aims to resolve.

To situate the project within a broader technological context, the chapter also examines **existing inventory management solutions** and related digital tools. It compares enterprise-grade platforms, lightweight apps, and mid-tier solutions, evaluating their suitability for the specific needs of Poltronesofà's store network. This comparison illustrates the functional gap that the custom-developed platform seeks to address, namely, the need for a tailored, scalable, and user-friendly solution that aligns closely with real operational workflows.

Finally, the chapter presents an overview of the **architecture and technologies** used in the implementation of the system. It discusses the rationale behind the selection of FastAPI and Vue.js, and explains how their modularity, performance, and ease of integration support the platform's goals. Particular emphasis is placed on the system's Progressive Web App (PWA) capabilities, its multi-role access model, and its integration with existing infrastructure such as the Oracle Database used by the company.

By the end of this chapter, the reader will have a comprehensive understanding of the organizational, technological, and functional background that shaped the development of the Stock Uniters platform. This foundation is essential for appreciating the design decisions and implementation strategies described in the chapters that follow.

2.1 About Poltronesofà

Poltronesofà is a leading Italian furniture company renowned for its expertise in the design, production, and sale of handcrafted sofas and armchairs. Established in 1995 in Forlì, the company has experienced steady and continuous growth over the years, evolving from a national brand with local recognition into a prominent player in the European furniture market. Today, Poltronesofà operates an extensive retail network composed of hundreds of stores, not only throughout Italy but also in countries such as France, Belgium, and Switzerland. This geographic expansion reflects a business strategy centered on internationalization, brand consistency, and strong logistical capabilities.

The company's core identity is deeply rooted in traditional Italian craftsmanship. All products are designed and produced with attention to artisanal quality, aiming to combine aesthetics with long-lasting comfort. Each sofa and armchair can be customized to suit individual customer needs, with an extensive catalog of upholstery materials, colors, finishes, and modular configurations. This high level

of personalization allows Poltronesofà to serve a wide and diverse customer base, while reinforcing its reputation for elegance, comfort, and durability.

What distinguishes Poltronesofà from many of its competitors is its distinctive retail approach. Rather than focusing solely on online sales or self-service experiences, the company emphasizes an in-store, high-touch customer journey. Every store is staffed by trained sales consultants who guide customers through the selection and configuration process, ensuring that the final product reflects both aesthetic preferences and functional requirements. This model requires not only excellent interpersonal skills from staff, but also efficient and reliable operational support to manage quotations, orders, deliveries, and service follow-up.

Behind the scenes, this high level of customer service is enabled by tight coordination across several critical business domains, particularly sales, logistics, and customer service. These areas must interact seamlessly to support day-to-day operations and respond quickly to customer requests or logistical changes. Given the distributed nature of the retail network and the volume of daily transactions, the ability to maintain consistency and efficiency becomes a strategic priority.

To address these needs, Poltronesofà has progressively integrated various internal digital tools into its workflows. These tools are used to facilitate communication between stores and headquarters, standardize internal procedures, manage stock and order information, and ensure a common operational framework across all retail locations. By digitizing and automating parts of its internal processes, the company is able to monitor performance, reduce human error, and maintain a high standard of service quality, regardless of the geographic location of a store.

The continued success of Poltronesofà demonstrates the effectiveness of this approach, combining the tradition of Italian craftsmanship with modern organizational practices and digital innovation. This balance has allowed the company to strengthen its brand identity while remaining agile and responsive to changes in market demand.

The company's ongoing digital transformation is also supported by a dedicated internal **IT department**, which plays a strategic role in the development, integration, and maintenance of digital systems across the organization. This team is responsible for ensuring system availability, data security, and the seamless operation of business-critical applications used daily by staff in stores and at headquarters. In addition to infrastructure and software support, the department also manages **Business Intelligence (BI)** tools that provide essential insights for decision-making. Through centralized data collection and analysis, the BI systems enable managers to monitor KPIs, identify operational trends, and optimize resource allocation. These capabilities are crucial for maintaining efficiency across a distributed retail network and for aligning local activities with the company's broader strategic goals. The collaboration between IT specialists, business stake-

holders, and store personnel ensures that technological solutions are practical, scalable, and closely aligned with real-world workflows.

2.2 What is a Web Application?

A web application is a software solution that operates within a web browser, offering a dynamic and interactive user experience without requiring traditional software installation on the client device. Unlike desktop applications that must be manually installed and maintained on individual machines, web applications run on centralized servers and can be accessed through standard web protocols using a URL. This architectural choice significantly reduces the complexity of deployment and maintenance, particularly in large or distributed organizational contexts.

By leveraging the Internet and modern web technologies, web applications provide universal access across various platforms and devices. This includes desktops, laptops, tablets, and smartphones, regardless of the underlying operating system. Such cross-platform compatibility is especially beneficial in environments like retail, where employees may use different types of hardware and need consistent access to the same system. Moreover, updates to the application can be deployed centrally on the server, ensuring that all users always interact with the latest version without requiring manual intervention.

At a technical level, web applications adopt a client-server architecture [RR10]. The **client-side** (or frontend) handles the presentation layer and user interactions. It is responsible for rendering the interface elements that users see and interact with, such as forms, tables, buttons, and charts. The **server-side** (or backend), on the other hand, manages the application's core functionality. It handles tasks such as business logic, data validation, communication with the database, and interfacing with third-party services. The communication between client and server typically occurs through HTTP requests, with data exchanged in standardized formats such as JSON.

Modern frontend development often utilizes JavaScript-based frameworks such as React, Angular, or Vue.js. These frameworks facilitate the development of responsive, modular applications by introducing concepts like component-based architecture, state management, and two-way data binding. Vue.js, in particular, offers a lightweight yet powerful approach to building responsive applications. Its ease of integration, low learning curve, and rich ecosystem make it especially suitable for medium-sized projects that require flexibility and speed of development. For these reasons, Vue.js was selected for this project.

On the backend, frameworks like FastAPI, Django, or Express.js provide the foundation for defining API routes, authenticating users, processing logic, and interacting with databases. FastAPI, the backend framework used in this project,

is known for its high performance and its ability to automatically generate documentation from type annotations. It leverages modern features of the Python programming language, such as asynchronous support and strong typing, to simplify development and improve reliability.

For the *Stock Uniters* application, the architectural roles of the frontend and backend are clearly defined and tailored to meet the operational needs of the company:

- The **frontend** is designed to be lightweight, intuitive, and fully responsive. It allows employees to interact with the system using a variety of devices—including desktop PCs, tablets, and mobile phones—which is essential in a store environment where hardware configurations vary. The interface prioritizes usability and speed, ensuring that tasks such as stock registration and withdrawal can be completed quickly and with minimal training.

- The **backend** exposes RESTful endpoints secured by JWT-based authentication. This ensures that only authorized users can perform critical operations. The backend handles all stock-related activities, such as registering new product entries, processing withdrawals for customer use, logging returns, and enabling manual adjustments during inventory reconciliation. All data is persistently stored in a relational database, which ensures consistency and traceability of operations.

This clear separation of concerns ensures not only technical modularity but also facilitates maintainability and scalability. Each layer of the application can be developed, tested, and deployed independently, which streamlines development and enables faster iteration cycles. Furthermore, this architecture supports continuous improvement, a critical requirement in a fast-paced retail context where operational processes evolve regularly and systems must adapt accordingly.

2.3 Company and Use Case Context

The digital solution was conceived in response to specific challenges faced by store-level staff across the Poltronesofà retail network. These challenges stemmed from fragmented and often manual workflows used to manage Uniters kit stock. Prior to the implementation of the new system, stores frequently relied on spreadsheets, emails, and phone calls to track kit availability, request replenishments, or report discrepancies—a process prone to errors, delays, and lack of transparency.

This decentralized and inconsistent approach made it difficult to obtain an accurate view of stock movements at the local and regional levels. Store employees had no standardized interface to verify current availability, document transactions,

or view historical data. Replenishment procedures were often reactive rather than planned, with stores initiating contact with central offices only when shortages became urgent. Moreover, the lack of structured return handling and audit trails reduced accountability and introduced a margin for error that was difficult to quantify or correct.

The newly developed web application addresses these gaps by centralizing all stock-related operations into a single, unified interface. The system is accessible from any device with internet access and is designed to accommodate the diverse operational environments of Poltronesofà stores, from high-traffic urban locations to smaller regional branches. Through a role-based access system, different categories of users interact with the application according to their responsibilities and information needs. Key user roles and their interactions with the system include:

Sales staff who are responsible for issuing kits to customers or returning unused or defective kits. They require a fast, intuitive interface that minimizes time spent on administrative tasks and ensures that stock movements are recorded accurately and promptly.

Store managers who oversee stock levels and perform periodic inventory checks. Their responsibilities include validating transactions, resolving discrepancies, and maintaining alignment between physical inventory and the digital system.

Area managers who oversee multiple stores within a region. They rely on aggregated data to evaluate store performance, identify anomalies in stock behavior, and coordinate actions such as bulk replenishments or policy changes across their assigned territories.

By unifying these roles under a shared system, the application enables real-time tracking of stock movements, reduces reliance on manual processes, and facilitates performance monitoring at various organizational levels. In doing so, it helps close the gap between daily operational needs and strategic oversight. This digital transformation not only enhances the efficiency of routine tasks but also supports broader organizational objectives related to standardization, transparency, and customer satisfaction.

2.4 Problem Domain

The domain addressed by this project centers on the internal distribution and lifecycle management of Uniters kits, a logistical process that, although seemingly

simple, involves multiple steps, stakeholders, and potential points of failure. The problem is not one of scale alone, but of coordination, consistency, and clarity in the way stock is handled at the store level and beyond.

The operational focus includes several interconnected areas:

Tracking incoming and outgoing kits When kits are delivered to the store or issued to customers, the system records the transaction in real time. This ensures accurate and up-to-date stock levels at all times, reducing the risk of stockouts or redundant orders. Each movement is timestamped and linked to a specific user, improving traceability and responsibility attribution.

Managing returns Kits may be returned for various reasons, including product defects, incorrect shipments, or excess stock. This structured process reduces ambiguity and facilitates future data analysis on return trends.

Inventory reconciliation Regular manual stock checks are supported by features that enable users to correct discrepancies between physical and digital inventories. Each correction is logged with timestamps and user credentials, creating an auditable trail for future review. This is especially important for compliance with internal standards and for resolving disputes or anomalies.

Dashboard visualization The application integrates interactive dashboards that display key performance indicators (KPIs), such as stock availability per store. These visualizations help managers at different levels make informed decisions and respond quickly to operational issues. Data can be filtered by store, time period, or transaction type to suit different analytical needs.

Downloadable user guide To enhance usability and reduce the need for external training sessions, the system provides a downloadable PDF guide. This document outlines the main workflows, role-specific instructions, and troubleshooting tips. It is accessible directly from the application's interface, ensuring that help is always within reach, even in decentralized store environments with limited IT support.

PWA (Progressive Web App) support The application is designed with PWA standards, allowing users to install it on their desktop or mobile home screens as if it were a native app. This enhances accessibility and convenience, particularly for sales staff who operate in fast-paced, customer-facing settings. Offline capabilities and automatic updates further support usability across diverse network conditions.

Each of these areas is directly linked to operational efficiency. A delay in recording a stock movement, for example, may result in shortages that affect customer service. Similarly, unresolved discrepancies can lead to inaccurate reporting and complicate supply planning. The system's design therefore prioritizes clarity, speed, and reliability. By aligning the digital tools with real-world workflows, the application aims to reduce friction, enhance visibility, and ensure that each stakeholder has the information they need, when they need it.

2.5 Existing Solutions and Related Work

A wide range of inventory management solutions exist, ranging from enterprise-level ERP systems to lightweight, store-level tools. However, few align well with the specific operational needs of Poltronesofà's post-sales kit tracking. The company's requirements are distinct in that they involve specialized workflows, multiple user roles across a distributed network, and the need for a lightweight yet reliable interface suited to fast-paced retail environments.

Enterprise platforms such as **SAP**, **Oracle NetSuite**, or **Microsoft Dynamics 365** provide powerful, integrated solutions that cover a broad spectrum of business functions, including finance, procurement, human resources, and inventory management [LL20]. These platforms are highly customizable and scalable, making them suitable for large multinational corporations with complex operational requirements. However, their use often comes with considerable overhead. Implementation can span several months, involving configuration, testing, data migration, and user training. These systems typically require dedicated IT teams for ongoing support and updates. For a focused use case like the internal tracking of Uniters kits—where agility, ease of use, and rapid deployment are essential—such platforms are often too rigid, expensive, and resource-intensive.

On the opposite end of the spectrum, many stores resort to ad hoc tools such as spreadsheets, shared documents, or simple inventory applications. These tools are familiar, easy to implement, and require minimal onboarding. However, their informality also introduces several limitations. Manual data entry increases the risk of errors, while the lack of automation makes real-time synchronization between stores and central management impractical. Moreover, data fragmentation often arises when multiple, unsynchronized versions of the same spreadsheet circulate among employees. This leads to inconsistencies, reduced visibility, and difficulties in generating aggregated reports or conducting audits.

Some mid-tier inventory tools—such as *Zoho Inventory*, *inFlow*, or *Sortly*—offer more robust features and mobile accessibility, addressing some of the shortcom-

ings of basic solutions. These platforms often include barcode scanning, user role management, and basic reporting dashboards. However, they are still designed with generic use cases in mind. Customizing them to fit Poltronesofà's internal processes, such as handling stock returns with classification, reconciling physical inventory, and aggregating data across multiple stores, would likely require extensive configuration, which can increase complexity and reduce user adoption. Additionally, integrating them with existing internal systems could require additional middleware or manual intervention.

The *Stock Uniters* platform was developed to fill this gap: offering a balance between simplicity and structure. Unlike heavy ERP solutions, it does not require external consultants or long deployment times. At the same time, it overcomes the fragmentation and limitations of ad hoc tools by introducing a centralized, web-based interface with real-time synchronization and standardized workflows. The platform is specifically tailored to Poltronesofà's operational environment, supporting key roles such as sales staff, store managers, and area managers. It allows for consistent stock registration, discrepancy resolution, and performance monitoring, all within an interface designed to be intuitive and accessible—even in stores with limited technical infrastructure.

By targeting the specific operational pain points identified through direct observation and stakeholder input, the solution positions itself not as a generic inventory system, but as a domain-specific tool that aligns with real business needs. Its development illustrates the value of custom applications in niche but critical areas where neither off-the-shelf nor enterprise solutions offer the right fit.

2.6 Technology Stack Overview

The architecture of the *Stock Uniters* platform reflects modern best practices in web development, with a clean separation between frontend and backend components. This modular design supports maintainability, performance optimization, and future scalability across a distributed retail network. The components were selected with careful consideration of the company's existing infrastructure, the skill sets of internal teams, and the practical needs of end users operating in dynamic store environments.

2.6.1 Backend (FastAPI)

The backend is built using FastAPI, a modern, asynchronous web framework in Python known for its high performance, automatic generation of OpenAPI-compliant documentation, and strong support for type validation [Ram22]. FastAPI provides a structured and efficient way to define RESTful endpoints responsible

for handling core operations such as registering new stock entries, recording withdrawals, processing kit returns, and applying inventory corrections.

The backend communicates with an **Oracle Database** to ensure consistent and persistent data storage. Data integrity and access control are central to the backend's architecture, with all operations protected by **JWT-based authentication**. This token-based system allows for secure identification of users and enables differentiated access depending on role (e.g., sales staff, store managers, area managers). The backend also includes mechanisms for input validation, error handling, and structured logging, all essential for operational reliability and auditability in a business-critical environment.

2.6.2 Frontend (Vue.js)

The frontend is developed using Vue.js, a progressive JavaScript framework that supports reactive data binding and component-based UI development [Vue23]. Its lightweight nature and gradual learning curve make it especially suitable for retail applications that require responsive design and fast interactions. The user interface is fully responsive and adapts to a variety of devices, including desktops, tablets, and smartphones, to support staff operating in different roles and physical contexts.

The interface includes intuitive dashboards, dynamic forms, and real-time feedback for user actions. It is designed around the needs of each role, presenting only the relevant features and minimizing complexity. Features such as camera-based barcode scanning are directly integrated into the interface, allowing for fast item identification and stock registration without the need for external scanners. Data visualization is achieved through interactive charts and summaries that provide actionable insights at a glance.

Additionally, the Vue.js frontend is enhanced with **Progressive Web App (PWA)** capabilities [Dev22]. This allows users to install the application directly on their device's home screen, whether mobile or desktop, and interact with it as if it were a native app. Key benefits include:

- Offline mode: Enables continued functionality during network interruptions.
- Automatic updates: Ensures the latest version is always available without user action.
- Faster loading: Service workers cache assets for improved performance.
- Accessibility: Facilitates daily use in store environments with varying network conditions.

These features are particularly valuable for sales staff working in high-traffic environments, where quick access and stability are critical.

2.6.3 User Documentation (PDF Guide)

In order to support decentralized teams and reduce reliance on formal training, the application includes a downloadable **PDF user guide**. This document is embedded within the platform and provides clear, role-specific instructions on performing everyday operations. It covers workflows such as issuing and returning kits, checking inventory, and resolving discrepancies. By being directly accessible within the user interface, the guide ensures that help is always within reach, even in locations with limited IT support.

The chosen technology stack ensures flexibility, security, and scalability. It supports real-time operations in dynamic retail environments, simplifies user onboarding, and allows for incremental updates without disrupting service. Furthermore, the clear separation between frontend and backend allows developers to iterate quickly and deploy improvements independently, reinforcing the platform's ability to adapt to evolving business needs and standards across the Poltronesofà retail network.

Chapter 3

Analysis

This chapter presents a structured analysis of the requirements that guided the development of the *Stock Uniters* web application. Following the identification of specific operational challenges during the preliminary investigation phase, the goal of this analysis is to translate those challenges into a well-defined set of technical and functional specifications [Som16]. These specifications serve as the foundation for the system's architecture, user interface, and interaction flows described in subsequent chapters.

The analysis is rooted in real business workflows observed across multiple Poltronesofà retail stores. By engaging directly with end users including sellers, store managers, and area managers, the project was able to capture a range of needs, constraints, and expectations. These insights were translated into concrete system requirements, ensuring that the resulting platform would align closely with daily store operations and broader organizational goals. The chapter is organized into several sections.

Functional Requirements These define the system's core capabilities, such as recording kit deliveries, issuing kits tied to Uniters warranty sales, registering returns, adjusting inventory, and generating reports.

User Roles and Access Levels A clear role-based access model is essential for both usability and data security. This section outlines the responsibilities and permissions of sellers, store managers, and area managers within the system.

Non-Functional Requirements These capture the qualities the system must exhibit to be usable and reliable in a retail context. Key concerns include responsiveness across devices, security, stability, and traceability.

Use Cases A set of structured scenarios illustrating how users interact with the system to accomplish operational goals. Each use case describes the flow of actions for a specific function, from inbound stock registration to brochure requests.

User Stories Realistic narratives that encapsulate user needs in a concise format, helping to bridge the gap between business goals and technical implementation. These stories also serve as a reference for evaluating feature completeness.

By providing a comprehensive view of what the system must do and how it will be used, this chapter ensures that all subsequent design and development work remains grounded in actual operational requirements. It also facilitates early detection of gaps, ambiguities, or mismatches between the software and its intended context of use.

The result is a requirements model that not only supports efficient implementation but also promotes long-term adoption and scalability across the Poltronesofà store network.

3.1 Functional Requirements

This section outlines the core functional requirements that the system must satisfy in order to support operational needs across different user roles within the Poltronesofà retail network. These functionalities were defined based on a detailed analysis of existing workflows, with the aim of standardizing processes, improving traceability [Bis02], and enabling real-time visibility into stock movements.

3.1.1 Authentication and User Access

Secure login with credentials The system must provide a secure login mechanism that authenticates users based on unique credentials. This ensures that only authorized personnel can access sensitive stock data and perform operations.

Role-based access (seller, store manager, area manager) Each user must be assigned a specific role that determines the features and data accessible within the application. Sellers are limited to basic operations such as issuing kits, while store managers and area managers have extended permissions for stock control, reporting, and oversight [SCFY96].

3.1.2 Inbound Stock Registration

Upload of delivery document images Users must be able to attach scanned images or photographs of delivery notes to new stock entries, providing visual proof of incoming shipments.

Declaration of received quantities Upon receiving a shipment, users must declare the number of kits received. This information is recorded and cross-checked with expected delivery quantities.

Conditional acceptance in case of anomalies If discrepancies are found between the declared and expected quantities, such as missing or damaged items, the system must allow conditional acceptance of the delivery and flag the record for later resolution.

Physical count and confirmation The system must require a physical stock count to confirm the actual number of items received. Once validated, the entry is finalized and stock levels are updated accordingly.

3.1.3 Stock Exit Registration

Associate kits with specific orders with warranty Each stock withdrawal must be linked to a specific order that must include the Uniters warranty. This association improves traceability and provides context for downstream reporting.

Confirm and finalize delivery Users must confirm each withdrawal before it is finalized. Once confirmed, the system updates stock levels and records the transaction with a timestamp and user ID.

3.1.4 Returns Management

Register return of kits The system must support the registration of returned items. Users can document the return through a dedicated interface that captures relevant metadata.

3.1.5 Inventory Adjustments

Manually correct stock levels Store managers must be able to manually update stock quantities during reconciliation or in response to errors.

Provide justification and ensure traceability Every manual correction must include a written justification and be recorded with full traceability, including the responsible user and timestamp.

3.1.6 Movement History and Reporting

Display historical records for all stock operations The system must maintain a complete audit trail of all stock movements, including entries, withdrawals, returns, and adjustments.

Allow filtering by date and type Users must be able to filter transaction history based on time range, operation type, and store location to facilitate targeted reviews.

Enable export to Excel For external analysis and reporting, the system must support the export of filtered data sets to Excel-compatible formats.

3.1.7 Brochure Management

Allow store users to request brochures to include in kits Users must be able to place internal requests for brochures directly through the application when assembling complete kits.

Track available quantities in each store The system must track brochure stock levels independently of kits, displaying current quantities per store in real time.

Automatically trigger reorder requests when below threshold When brochure quantities fall below a predefined minimum threshold, the system must automatically notify relevant users or generate a reorder request to ensure continued availability.

3.2 User Roles and Access Levels

The system defines three primary user roles, each with distinct access levels and operational responsibilities. Role-based access control (RBAC) ensures that users can only interact with the parts of the system relevant to their position, thereby improving security, accountability, and usability. The following roles have been defined to reflect the real-world organizational structure within the Poltronesofà retail network:

Seller This role is assigned to store-level staff responsible for day-to-day handling of Uniters kits. Sellers are authorized to register the receipt of new kits delivered to the store, record stock exits when kits are issued to customers, and process returns. Sellers operate exclusively within their assigned store context and do not have visibility into other stores or administrative functions. The interface available to them is streamlined for speed and simplicity, reducing the learning curve and supporting fast-paced customer interactions.

Store Manager / Vice Store managers (or their appointed vice managers) have broader access and greater responsibility than sellers. In addition to all seller permissions, they are authorized to perform inventory adjustments when physical stock and system records do not match, enter justifications for manual corrections (ensuring transparency and traceability), and complete and validate monthly stock counts as part of the reconciliation process. Managers serve as the first line of accountability for data accuracy and process compliance within their store. They also have access to additional reporting tools and dashboards specific to their store's activity.

Area Manager This role is intended for regional supervisors who oversee multiple stores. Area managers have read-only access to view historical movement data for all stores in their territory, monitor stock levels, usage patterns, and discrepancies through a centralized dashboard, and analyze trends and performance metrics across stores to support regional decision-making. The interface for area managers emphasizes data visibility and comparative analysis rather than operational input. Their role is to supervise, evaluate, and support store-level operations without directly performing stock transactions.

This clearly defined role hierarchy ensures that each user interacts with the system according to their responsibilities, minimizes the risk of unauthorized actions, and promotes a secure and scalable multi-store environment.

3.3 Non-Functional Requirements

In addition to functional capabilities, the system must adhere to a set of non-functional requirements that ensure usability, security, and operational reliability. These characteristics are essential for successful adoption in a distributed retail environment and directly influence user satisfaction and long-term maintainability.

Responsiveness The user interface must be fully responsive, enabling seamless operation across a range of devices, particularly tablets and desktop computers [iso19]. Since store staff may use different hardware depending on local setup and workflows, the application must adapt its layout and components to various screen sizes without compromising functionality or clarity.

Ease of Use The system must be intuitive and require minimal training for new users. Guided workflows, clear navigation, and contextual help features must be incorporated to reduce onboarding time and ensure efficient daily use. This is especially important in retail settings where staff turnover may be high and training resources limited [Nie93].

Security All access to the system must be authenticated, with user identity verified through secure login mechanisms. Role-based access control must prevent unauthorized users from accessing or modifying sensitive data [Bis02]. Additionally, the system must enforce secure data transmission protocols and protect against common web vulnerabilities.

Stability The platform must demonstrate high availability and consistent performance during regular usage. Given that stock operations occur daily in multiple stores, the system must reliably handle concurrent transactions, provide real-time feedback, and recover gracefully from potential errors or network interruptions.

Traceability All user actions must be recorded with full auditability. This includes capturing the user ID, timestamp, and type of action for each operation affecting stock records. Such traceability supports accountability, facilitates error resolution, and enables internal audits or historical reviews when discrepancies occur.

3.4 Use Cases

This section describes the main use cases supported by the system. Each use case represents a specific interaction between a user and the system, focused on achieving a concrete business goal. The use cases were derived from real operational workflows observed in Poltronesofà stores and validated through discussions with end users and stakeholders [Coc01].

Figure 3.1 illustrates the overall structure of the system's use cases and their relationships with the primary user roles: sellers, store managers, and area managers. The diagram provides a high-level visual summary of the system's functionality,

making it easier to understand how different actors interact with specific features of the platform.

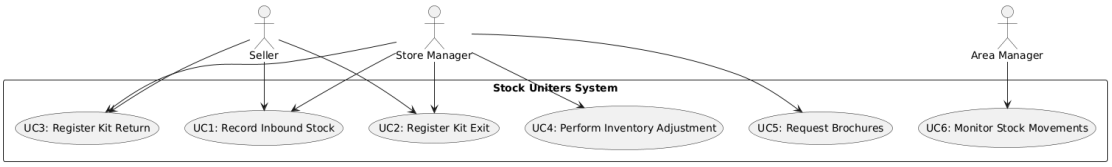

Figure 3.1: Use Case Diagram of the Stock Uniters System

3.4.1 UC1 – Record Inbound Stock

Actor Seller

Goal Register the arrival of new Uniters kits at the store location

Preconditions The user must be authenticated and associated with a store

Main Flow

1. The seller logs into the application using their credentials
2. Navigates to the "Inbound Stock" section from the main dashboard
3. Uploads an image or file of the delivery document (e.g., packing slip)
4. Enters the number of kits received, comparing with the document
5. Confirms the registration to finalize the operation

Postconditions The inbound stock movement is stored in the system, linked to the user and timestamp, and becomes visible in the movement history

3.4.2 UC2 – Register Kit Exit

Actor Seller

Goal Confirm the issuance of a Uniters kit associated with a customer warranty request

Preconditions The user must be logged into the system with an active session

Main Flow

1. The seller selects the warranty associated with the customer
2. Confirms the issuance of the corresponding kit
3. The system automatically updates the inventory and logs the transaction

Postconditions The kit exit is recorded and associated with the order, enabling traceability

3.4.3 UC3 – Register Kit Return

Actor Seller

Goal Register the return of a Uniters kit to the system and ensure traceability

Preconditions The user must be authenticated and associated with a store. A valid kit exit must already exist for the order.

Main Flow

1. The user scans or enters the barcode of the returned kit
2. The system checks whether the returned item matches the original kit delivered for that order
3. If the match is correct, the return is recorded in the database
4. The action is timestamped and linked to the user

Postconditions The kit return is stored in the movement history, restoring the item to inventory and enabling traceability

Notes This feature improves accountability and enables accurate stock reconciliation during audits

3.4.4 UC4 – Perform Inventory Adjustment

Actor Store Manager

Goal Correct discrepancies identified between physical and system stock counts

Preconditions The user is authenticated with manager privileges

Main Flow

1. The store manager logs into the system
2. Navigates to the inventory adjustment module
3. Inputs the corrected quantity for a specific item and provides a justification
4. Confirms the adjustment to apply the change

Postconditions The stock level is updated, and the adjustment is logged with timestamp, justification, and user identity for auditing purposes

3.4.5 UC5 – Request Brochures

Actor Store Manager

Goal Submit a request for additional brochures when local stock falls below the predefined threshold

Preconditions The user must be authenticated and associated with a specific store

Main Flow

1. The manager accesses the brochure request section from the main interface
2. Confirms the request using a predefined action (no manual quantity input required)
3. The system registers the request and automatically notifies the relevant central team responsible for dispatch

Postconditions A brochure request is recorded and linked to the requesting store

Notes The quantity to be sent is managed centrally based on policy; stores simply express the need through the system

3.4.6 UC6 – Monitor Stock Movements

Actor Area Manager

Goal View and analyze historical stock movement data across multiple stores within their assigned territory

Preconditions The area manager must be authenticated with read-only access rights

Main Flow

1. The area manager logs into the system
2. Selects one or more stores to monitor
3. Applies filters by date, movement type (inbound, exit, return, adjustment), or other relevant criteria
4. Optionally exports the filtered dataset to Excel for external analysis

Postconditions The selected data is retrieved and displayed in the dashboard; exports are saved locally by the user if needed

3.5 User Stories

The following user stories describe realistic interactions with the system from the perspective of its main user roles. Each story illustrates a specific operational need and highlights how the platform supports daily activities within the Poltronesofà retail network [Coh04]. These narratives were designed to reflect real workflows and guide the definition of system features and user interface elements.

3.5.1 Seller – Register Incoming Stock

User Story *As a seller, I want to upload a delivery document and record the number of kits received, so that stock entries are immediately reflected in the system.*

Description This story captures the need for timely and accurate registration of inbound stock at the store level. Sellers must be able to process incoming deliveries quickly and ensure that the recorded quantities align with physical inventory, while attaching visual proof of the shipment.

3.5.2 Seller – Register Kit Exit

User Story *As a seller, I want to link a kit to a specific sale with warranty and mark it as delivered, so that the inventory is accurately updated.*

Description This story reflects the core business rule that kits are only distributed to customers when they have purchased a Uniters warranty. The system must enforce this constraint by requiring the selection of a verified warranty order before allowing the stock exit to be recorded. This ensures consistency and full traceability of all kit deliveries.

3.5.3 Seller – Return a Kit

User Story *As a seller, I want to register a returned kit and explain the reason, so that the system tracks all movements.*

Description Returns may occur for various reasons, such as customer refusals, incorrect issues, or product damage. This story emphasizes the importance that the system maintains a complete and auditable stock history.

3.5.4 Store Manager – Adjust Inventory

User Story *As a store manager, I want to correct stock quantities manually when discrepancies are found, so that inventory reflects the real situation after monthly checks.*

Description Inventory reconciliation is a key responsibility of store managers. This story highlights the need for authorized users to perform manual adjustments when mismatches arise between physical counts and system data, while ensuring that every change is logged and justified.

3.5.5 Store Manager – Request Brochures

User Story *As a store manager, I want to request new brochures when the store stock is low, so that every outgoing kit can be completed properly.*

Description Since brochures are a required component of the Uniters kits, store managers must be able to initiate replenishment requests as soon as stock runs low. This story ensures that stores never lack the materials necessary to fulfill their warranty-related obligations.

3.5.6 Area Manager – Monitor Movements

User Story *As an area manager, I want to see a full history of stock movements across my stores, so that I can monitor operations and follow up when needed.*

Description Area managers are responsible for supervising multiple stores within a region. This story reflects their need for visibility into store-level activity, enabling them to detect anomalies, compare performance, and support local teams based on data-driven insights.

Chapter 4

Design

The design phase translates the requirements identified in Chapter 3 into a concrete technical blueprint for the *Stock Uniters* system. While the analysis stage focused on what the system must achieve from a functional and non-functional perspective, the design stage addresses how these objectives are realized through architectural choices and component organization.

The chapter adopts a top-down approach: it begins with the overall **system architecture**, showing the separation of responsibilities between frontend, backend, and database layers. It then introduces a **high-level architecture diagram** that provides a visual representation of these components and their relationships. Finally, the **component overview** describes the main building blocks of both the frontend and the backend, highlighting their responsibilities and the design rationale behind their organization.

By providing both structural and behavioral views of the system, this chapter ensures that the implementation in the following sections can remain aligned with the business goals of usability, traceability, and scalability. The resulting design balances modularity and integration, allowing the platform to evolve while maintaining consistency with the operational requirements of Poltronesofà stores.

4.1 System Architecture

The architecture of the *Stock Uniters* system was designed to ensure robustness, scalability, and ease of use across the entire Poltronesofà retail network. Given the distributed nature of the organization, with hundreds of stores operating under heterogeneous conditions, the system needed to guarantee consistent access to centralized data, provide a user-friendly interface, and maintain strong security standards. To address these requirements, a **modular client-server architecture** was adopted, separating responsibilities into three distinct layers: presenta-

tion, business logic, and data management.

4.1.1 Architectural Paradigm

The Stock Uniters platform follows a **three-tier architecture**, a well-established paradigm in enterprise system design that promotes separation of concerns and long-term maintainability. This choice was made after considering alternative approaches such as monolithic architectures and microservices:

Monolithic approach Initially considered for its simplicity and reduced initial development effort, this approach was discarded due to its lack of flexibility. A single codebase handling both frontend and backend logic would have made scaling and independent updates more difficult.

Microservices architecture This alternative offers strong scalability and independence of services, but it also introduces significant operational overhead, particularly in terms of deployment, monitoring, and inter-service communication. Given the specific scope of Stock Uniters, the additional complexity was not justified.

Three-tier client-server model Selected as the most balanced option, this architecture enables clear modularization, strong maintainability, and easier future integration with external systems (such as ERP or BI platforms), without introducing unnecessary complexity.

This layered approach ensures that each part of the system can evolve independently: the frontend can improve its usability without modifying backend logic, the backend can extend its APIs without altering the database, and the database can be optimized or migrated while preserving business logic and presentation layers.

4.1.2 Frontend Layer (Vue.js)

The **frontend** of Stock Uniters is a **Single Page Application (SPA)** built using Vue.js. This framework was chosen for its balance between flexibility and simplicity. Unlike traditional multi-page applications, a SPA loads the application once and dynamically updates the user interface based on API calls, resulting in faster interactions and an improved user experience.

From a design perspective, the frontend focuses on:

- **Responsiveness**: The system must operate seamlessly across desktops, tablets, and mobile devices, reflecting the diverse hardware available in stores.

- **Role-based dashboards**: The interface adapts to the user's role (seller, store manager, area manager), reducing cognitive load by showing only the relevant features.

- **Integrated features**: File uploads (delivery notes, images), QR and barcode scanning through device cameras, and interactive dashboards that visualize KPIs in real time.

- **PWA capabilities**: Installation on devices, offline caching, and automatic updates via service workers, ensuring uninterrupted operations even in areas with unstable network connections.

The SPA model was selected over alternatives such as server-side rendering (SSR), which—although faster for initial loads—would have complicated offline support and PWA integration. Given that in-store usage prioritizes responsiveness and independence from constant connectivity, SPA was deemed the optimal solution.

4.1.3 Backend Layer (FastAPI)

The **backend** provides the business logic and exposes functionality through a **RESTful API** built with FastAPI. The framework was selected based on three design considerations: high performance, modern asynchronous programming support, and automatic documentation generation through OpenAPI.

Key responsibilities of the backend include:

- **Authentication and authorization**: User login is handled via credentials, with JWT tokens providing secure session management and role-based access enforcement.

- **Business logic enforcement**: The backend ensures that operational rules are respected: each kit exit must be linked to an order with a warranty included, deliveries with anomalies must be flagged for conditional acceptance, and manual adjustments must always include a justification.

- **Error handling and logging**: Structured responses are returned for all operations, while logs are stored for both debugging and audit purposes.

- **Data serialization and transport**: Communication with the frontend occurs over HTTPS using JSON, ensuring interoperability and lightweight payloads.

The choice of FastAPI over alternatives such as Django or Express.js was guided by its speed, simplicity, and strong typing. While Django offers an extensive ecosystem, it would have introduced unnecessary weight given the relatively limited scope of the project. Express.js, though lightweight, lacks built-in support for type validation and automatic documentation—both critical for a maintainable enterprise-grade system.

4.1.4 Database Layer (Oracle)

The **Oracle Database** forms the persistent storage layer of Stock Uniters. Its adoption aligns with Poltronesofà's existing IT infrastructure, ensuring compatibility with corporate standards of reliability and data governance. Oracle was chosen over open-source alternatives such as PostgreSQL or MySQL primarily for its enterprise-grade features, strong transaction management, and support by the company's internal IT department. Data persistence is organized as follows.

User management Tables storing user identities, roles, and store affiliations.

Stock tracking Real-time quantities of Uniters kits and brochures, linked to store IDs.

Transactions Full history of entries, exits, returns, and adjustments, each tied to a user and timestamp.

Attachments Metadata and references to external files (e.g., delivery document scans). The actual images are stored as PNG files in a dedicated virtual machine (WSL), while the database keeps only the paths or identifiers for retrieval.

4.1.5 Cross-Cutting Concerns

Beyond the three primary layers, the architecture incorporates several design decisions that address cross-cutting concerns:

- **Security**: End-to-end encryption (HTTPS), RBAC, audit logging, and database-level permissions safeguard sensitive operational data.
- **Scalability**: The modular design allows horizontal scaling of the frontend and backend services, and replication strategies for the database.

- **Maintainability**: Modularization ensures that updates to the frontend, backend, or database schema can be deployed independently with minimal disruption.

- **Resilience**: Error handling strategies and monitoring ensure that transient failures do not disrupt operations.

4.2 High-Level Architecture Diagram

Figure 4.1 provides a high-level overview of the *Stock Uniters* platform. The diagram highlights the separation of responsibilities between the frontend, backend, and database layers, as well as the modular decomposition of backend services into specialized components. This representation is intended to make explicit the major design choices and architectural principles that guided the system's development.

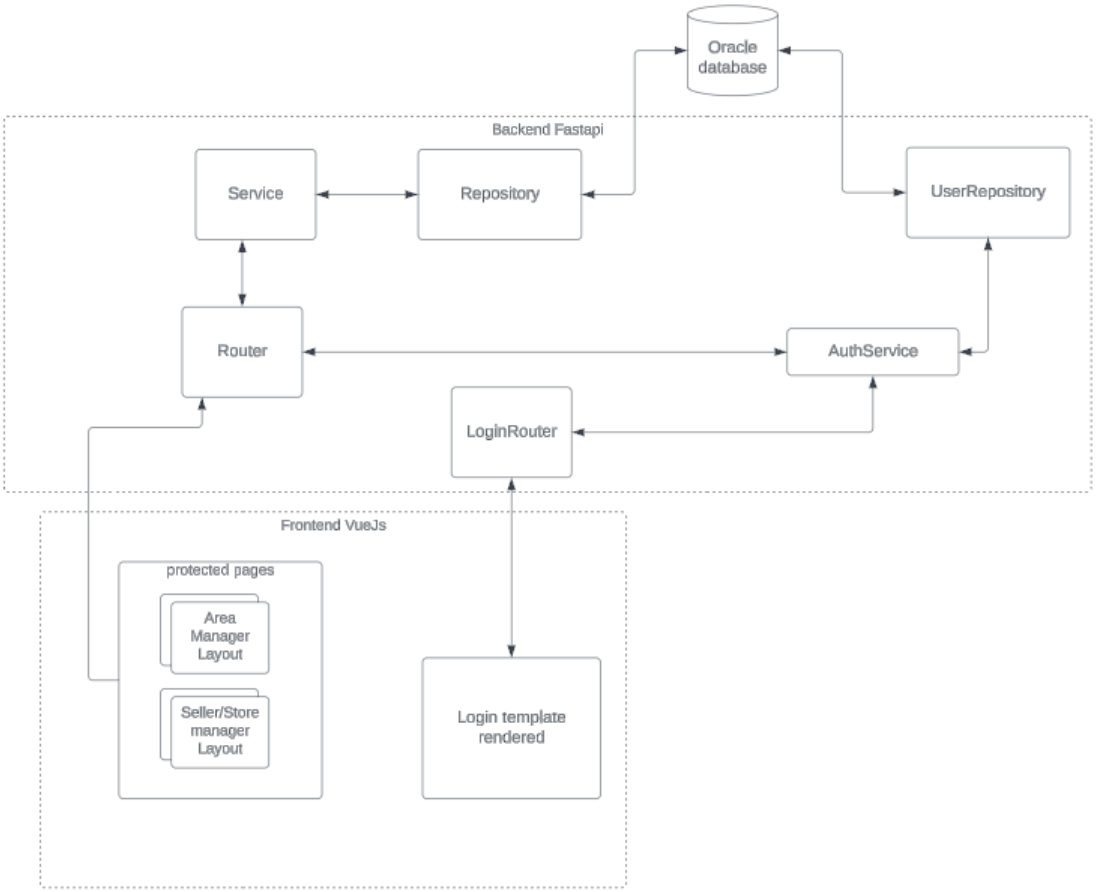

Figure 4.1: High-level architecture of the Stock Uniters platform

4.2.1 Purpose of the Diagram

The primary objective of the diagram is to offer a simplified yet comprehensive view of the system's internal organization. By abstracting away low-level implementation details, it allows both technical and non-technical stakeholders to understand how the platform is structured, how responsibilities are distributed among components, and how modularity is achieved.

This kind of representation is especially relevant in enterprise contexts, where multiple actors with different backgrounds (IT staff, managers, developers, end users) are involved in system evaluation. While developers may be concerned with the internal consistency of APIs and database schemas, managers may focus on maintainability and scalability. A high-level diagram provides a common language for aligning these perspectives.

4.2.2 Frontend Layer

The **Frontend Vue.js** block encapsulates all user-facing functionality. The diagram distinguishes between two key elements:

- The **Login template (public)**, representing the entry point of the application, accessible to all users prior to authentication.
- The **Protected pages**, accessible only to authenticated users. These are further specialized into:
  - **Seller/Store Manager Layout**, providing features for stock handling, returns, and adjustments.
  - **Area Manager Layout**, presenting aggregated data and supervisory dashboards.

This separation mirrors the logical division between public and private areas of a web application, a best practice in secure system design. It also reflects the system's emphasis on role-based customization: each user role is provided with a tailored interface that simplifies navigation and emphasizes only the most relevant tasks.

4.2.3 Backend Layer

The backend, implemented with FastAPI, is organized into several distinct modules. Each module is represented in the diagram as an independent block, connected by explicit dependencies:

Router Exposes RESTful endpoints to the frontend and directs requests to the appropriate service layer. This design decouples request handling from business logic, simplifying both maintenance and testing.

Service Encapsulates the application's core business rules. By concentrating operational logic here, the system enforces consistency across all interactions, regardless of which frontend component initiates the request.

Repository Provides an abstraction layer for data persistence. Following the repository pattern, it shields the service layer from direct database queries, ensuring that business logic remains independent of the underlying persistence technology.

LoginRouter A specialized router responsible for authentication-related endpoints. Its clear isolation from other routers enhances security and simplifies the enforcement of authentication flows.

AuthService Dedicated to authentication and authorization logic. It validates user credentials, manages JSON Web Tokens (JWT), and enforces role-based access control (RBAC).

UserRepository A specialized repository that handles all user-related queries, including retrieval of roles and permissions. Its separation from the general-purpose repository ensures clearer responsibility boundaries and facilitates auditing of user management operations.

This modularization reflects deliberate design choices aimed at enhancing extensibility. For example, new services or repositories can be introduced without altering existing modules, provided they conform to the same interaction contracts. Similarly, authentication mechanisms can be updated or replaced without disrupting unrelated business logic.

4.2.4 Database Layer

At the top of the diagram, the **Oracle Database** serves as the persistent data store. Its role is abstracted through repository interfaces, ensuring that backend services never interact with the database directly. This design provides two significant advantages:

Technology independence Although Oracle is currently used due to corporate standards, the abstraction layer would allow a future migration to a different relational database with limited impact on business logic.

Consistency and traceability All operations are funneled through repositories, which enforce data integrity, log operations, and provide standardized access methods.

The database is designed to manage operational entities (users, stock levels, transactions) and supporting information (audit logs, attachments). Its central position in the diagram reflects its role as the "single source of truth" for the system.

4.3 Component Overview

The component overview describes how the *Stock Uniters* platform is decomposed into concrete building blocks. While the architectural view illustrates the separation between frontend, backend, and database, this section focuses on the functional organization of the system's components.

Frontend Pages

The frontend is implemented as a Vue.js Single Page Application (SPA). Each page corresponds to a distinct business workflow, providing a task-oriented user experience that minimizes complexity and accelerates adoption.

Login Page Collects user credentials and initiates the authentication flow. Based on the role returned by the backend, the user is redirected to the appropriate workspace.

Home Page Serves as the navigation hub, offering shortcuts to the most frequently used functions and a snapshot of current stock status.

Inbound Stock Supports the registration of new deliveries through document uploads, barcode or QR scanning, and quantity entry. The page reduces manual data entry, improving accuracy.

Outbound Stock Allows sellers to associate kit exits with validated warranty sales. The page enforces organizational rules by requiring a verified order before confirming withdrawal.

Returns Guides users in associating a return with the original sale.

Inventory Provides store managers with tools for manual adjustment and monthly stock reconciliation. Justifications are mandatory, reinforcing accountability.

Dashboard Displays aggregated metrics and KPIs to managers and area managers, enabling quick evaluation of operational performance.

Backend Modules

The backend is structured into two primary modules, reflecting the dual nature of the system: security enforcement and operational execution. This modularization reduces coupling, improves maintainability, and ensures that core responsibilities are clearly separated.

Authentication Module Responsible for login, token issuance, and role-based access control. This module ensures that only authenticated users can access the platform and that each request is executed within the correct scope (seller, store manager, area manager). It is isolated from business logic so that authentication policies can evolve independently.

Operational Module Encapsulates all business functionalities, including inbound and outbound stock registration, returns management, inventory adjustments, brochure requests, and reporting. Within this module, sub-packages organize the logic by process domain, but they all share a common foundation of transactional consistency and auditability. By consolidating operational responsibilities in a single module, the system simplifies coordination across workflows and maintains uniform enforcement of traceability rules.

Chapter 5

Implementation

5.1 Key Technologies Used

The implementation of the *Stock Uniters* platform was guided by the technological choices introduced in Chapter 2 and driven by the functional and non-functional requirements analyzed in Chapter 3. The selected technologies were chosen not only for their technical capabilities but also for their suitability within the organizational context of Poltronesofà, where usability, reliability, and scalability are critical.

FastAPI

FastAPI was adopted as the backend framework for its combination of high performance, modern Python support, and ease of integration. Its asynchronous nature ensures that the system can handle concurrent requests efficiently, an essential feature in a distributed retail environment where multiple stores interact with the platform simultaneously. The automatic generation of OpenAPI documentation also simplifies testing and long-term maintainability. Importantly, FastAPI's strong typing and validation mechanisms directly support the system's need for accuracy and traceability, as required in functional requirements such as *Inbound Stock Registration* and *Inventory Adjustments*.

Vue.js

The frontend of the application was developed using Vue.js, a progressive JavaScript framework selected for its responsiveness, component-based structure, and relatively low learning curve. Vue.js enables the creation of dynamic, role-specific interfaces that reflect the access model defined in Chapter 3. By leveraging reactive data binding, the frontend ensures that stock levels, movement history, and

dashboards are updated in real time, meeting non-functional requirements of responsiveness and ease of use. In addition, its support for Progressive Web App (PWA) features guarantees stability even under fluctuating network conditions.

Oracle Database

Persistent storage of operational data is managed through an Oracle Database, which was already part of Poltronesofà's IT infrastructure. Integrating the application with this system reduces overhead for deployment and ensures consistency with existing Business Intelligence workflows. The relational model of Oracle supports the project's need for structured data and strong referential integrity, which is vital for maintaining audit trails of all stock movements.

JSON Web Tokens (JWT)

For authentication and access control, the system uses JWT-based mechanisms. This approach allows secure, stateless sessions between the client and server, minimizing overhead while ensuring that each request is tied to a verified user identity. JWTs also encode role-specific information, directly enabling the role-based access control (RBAC) model defined in Chapter 3. By doing so, they enforce functional requirements such as differentiating seller, manager, and area manager permissions.

Progressive Web App (PWA) Capabilities

PWA standards were incorporated into the frontend implementation to enhance usability in diverse store environments. Features such as offline mode, home screen installation, and automatic updates contribute to the system's non-functional requirements of stability and accessibility. In particular, offline availability ensures that sales staff can continue performing core operations such as kit issuance even when temporary connectivity issues occur.

Supplementary Tools and Libraries

Additional tools and libraries were integrated to support development and improve user experience. These include:

- **Axios**, for efficient HTTP communication between frontend and backend.
- **Vuetify**, a Vue.js UI library, to ensure a consistent and professional interface aligned with usability requirements.

- **Pandas and SQLAlchemy**, used during development for data handling and ORM (Object Relational Mapping) tasks.

- **Docker**, employed for containerization to simplify deployment and ensure consistency across environments.

Together, these technologies form a coherent stack that directly addresses the operational needs identified during the analysis phase. The combination of FastAPI, Vue.js, Oracle, and JWT provides the foundation for secure, responsive, and scalable stock management across Poltronesofà's distributed retail network.

5.2 Backend Implementation (FastAPI)

The backend of the *Stock Uniters* platform was developed using the FastAPI framework, as introduced in Chapter 2. Its design follows a modular architecture aimed at separating concerns, ensuring maintainability, and facilitating future scalability. The backend exposes a set of RESTful endpoints that handle the core business logic of stock management, including inbound and outbound registrations, returns, inventory adjustments, and reporting functionalities defined in Chapter 3.

Each endpoint is protected by JWT-based authentication, ensuring that only authorized users can interact with the system, and role-based access control (RBAC) is enforced at the service layer to guarantee that operations correspond to the permissions of sellers, store managers, and area managers. The communication between client and server occurs through JSON payloads.

To maintain code clarity and long-term sustainability, the backend was organized into distinct modules representing different application domains (authentication, stock operations, and reporting). This separation not only simplifies development but also allows each component to evolve independently. The following subsections describe the most important backend components, starting with the routing strategy and API versioning.

Routing and API Versioning

This section documents each HTTP endpoint exposed by the `stock` router. All endpoints are protected by JWT and receive the authenticated user via the `TokenData` dependency (`get_current_user`). Business logic is delegated to the `Service` abstraction through dependency injection (`get_service`), which encapsulates database access, validation, and side effects (email notifications, file storage). Unless otherwise stated, responses are JSON and errors are raised as `HTTPException` with appropriate status codes.

Conventions

- **Roles**: Seller (S), Store Manager (M), Area Manager (A). Authorization is enforced in the `Service`.
- **Dates**: Strings in `dd/MM/YYYY` format.
- **Files**: Uploaded via `multipart/form-data`.
- **Status codes**: `200 OK` on success, `4xx/5xx` on errors.

`POST /stock/carica_bolla` **(UC1 – Record Inbound Stock)**

Purpose. Registers the *draft* of an inbound delivery ("bolla") for Uniters kits received from the supplier, attaching the delivery document and optional anomaly photos.

Inputs (form).

- `descr` [optional]: textual notes or anomaly description.
- `file` [required]: delivery document (image/PDF).
- `negozio` [required]: store code (string).
- `nro_bolla` [required]: delivery note identifier.
- `nro_pelle`, `nro_tessuto` [required]: declared quantities by kit type.
- `anomaly_photos` [optional]: list of images documenting discrepancies.

Process. The router forwards to `Service.carica_bolla`, which:

1. Validates absence of duplicate delivery number for the store.
2. Stores files to the configured path and persists a "bozza" row.
3. Inserts expected lines (pelle/tessuto) with declared quantities.
4. Optionally triggers an email notification with attachments when `descr` is provided.

Response. `200 OK` with {`"message": "Dati inseriti correttamente"`}.
Roles. S, M.
Error cases. Duplicate bolla (`400`), storage/DB error (`500`).

`POST /stock/conferma_bolla` **(UC1 – Physical Count Confirmation)**

Purpose. Confirms the *physical* quantities counted in store and finalizes the inbound movement.

Inputs (form).

- `nro_bolla` [required]
- `nro_sparate_pelle`, `nro_sparate_tessuto` [required]: effective counted quantities.

Process. Delegated to `Service.conferma_bolla`:

1. Compares effective vs declared quantities, flagging mismatches.
2. Sends discrepancy email when needed.
3. Inserts inventory movements (ING) into the historical log.
4. Closes the draft.

Response. `200 OK` with confirmation message.
Roles. S, M.
Error cases. Missing draft, DB failure (`500`).

`POST /stock/conferma_bolla_mancante` **(UC1 – Inbound Without Supplier Draft)**

Purpose. Same as above, used when the supplier bolla is missing (manual inbound confirmation).

Inputs. `nro_bolla`, `nro_sparate_pelle`, `nro_sparate_tessuto`.

Process. `Service.conferma_bolla_mancante` records ING movements directly.

Roles. S, M.

`GET /stock/get_all_barcode`

Purpose. Returns the list of supported barcodes for kits and accessories (reference data for UI and validation).

Inputs. None.

Process. `Service.get_all_barcode` fetches catalog values.

Roles. S, M, A (read-only).

`POST /stock/carica_spedizione_corriere` **(UC1 – Inbound via Courier, no Uniters bolla)**

Purpose. Creates a draft inbound based on a courier shipment (`nro_sped_corr`) instead of a Uniters bolla.

Inputs (form). `descr`, `file`, `negozio`, `nro_sped_corr`, optional `anomaly_photos`.

Process. `Service.carica_spedizione_corriere` persists draft and attachments; optional email on anomalies.

Roles. S, M.

`POST /stock/uscita_kit` **(UC2 – Register Kit Exit)**

Purpose. Registers the issuance of a kit (exit) tied to a specific order with Uniters warranty.

Inputs (form). `nro_ord` (order id), `barcode` (kit type).

Process. `Service.uscita_kit`:

1. Verifies the order is not already fulfilled.
2. Checks stock availability of the kit and brochure (B111111).
3. Inserts two USC movements: brochure and kit; may auto-trigger brochure reorder if threshold breached.
4. Clears any "consegna posticipata" flag for the order.

Response. `200 OK` with confirmation.

Roles. S, M.

Errors. No stock available (`400`), DB failure (`500`).

`GET /stock/get_bozze_bolle`

Purpose. Lists existing inbound drafts for the current user's store.

Inputs. `negozio` (ignored in handler; store taken from token for safety).

Process. `Service.get_bozze_bolle`.

Roles. S, M.

`GET /stock/get_ordini_mancanti` **(UC2 – Orders Waiting for Kit)**

Purpose. Lists orders pending kit issuance for a given store.

Inputs. `negozio` (store code).

Process. `Service.get_ordini_mancanti` derives kit type from line description (pelle/tessuto) and maps `CONSEGNA_POST` to boolean.

Roles. S, M.

`GET /stock/get_ordini_mancanti_by_user`

Purpose. Same as above, filtered by the logged-in user (useful for per-operator assignment/queues).

Inputs. `negozio`.

Process. `Service.get_ordini_mancanti_by_user`.

Roles. S, M.

`GET /stock/get_ordini_usciti`

Purpose. Returns orders where kits have already been issued (audit/visibility).

Inputs. `negozio`.

Process. `Service.get_ordini_usciti`.

Roles. S, M, A (read-only).

`POST /stock/reso_kit` **(UC3 – Register Kit Return)**

Purpose. Records a returned kit and restores inventory if it matches the original exit.

Inputs (form). `nro_ord`, `barcode`.

Process. `Service.reso_kit` verifies the returned barcode matches the kit previously issued for the order, then inserts a RES movement.

Roles. S, M.

Errors. Mismatched kit (`400`), DB failure (`500`).

`GET /stock/get_magazzino` **(Store Inventory Snapshot)**

Purpose. Returns current quantities for the authenticated user's store.

Inputs. None.

Process. `Service.get_magazzino`.

Roles. S, M.

`POST /stock/rettifica_magazzino` **(UC4 – Inventory Adjustment)**

Purpose. Allows a manager to perform a manual adjustment (positive or negative) with optional justification.

Inputs (form). `cod_art`, `qta_to_move` (signed int), `descrizione` [optional].

Process. `Service.rettifica_magazzino` inserts a RET movement and logs justification for traceability.

Roles. M.

Errors. Validation/DB failure (`500`).

`GET /stock/get_info_bolla`

Purpose. Retrieves detailed rows of a specific inbound bolla for the user's store.
Inputs. `nro_bolla`.
Process. `Service.get_info_bolla`.
Roles. S, M.

`GET /stock/delete_bozza`

Purpose. Deletes a draft bolla (test + lines) for the user's store if present.
Inputs. `nro_bolla`.
Process. `Service.delete_bozza`, returning `200` on success or `400` if not present.
Roles. M (or S if policy allows).

`GET /stock/get_bolla_image`

Purpose. Returns the stored image/PDF path for a given document and store.
Inputs. `rif_doc`, `cod_negozio`.
Process. `Service.get_bolla_image_path`; router wraps the result in `FileResponse`.
Roles. S, M.

`GET /stock/get_storico` **(UC6 – Movement History)**

Purpose. Returns historical movements and summarized totals for the authenticated store in a given date range.
Inputs. `from_data`, `to_data`.
Process. `Service.get_storico` converts dates, fetches movements (ING/USC/RET/RES), computes totals and on-date stock levels, merges postponed deliveries.
Roles. S, M; A typically uses multi-store variants.

`GET /stock/get_neg_area_manager` **(Multi-Store Summary)**

Purpose. Returns per-store aggregates (movements, postponed counts, current stock) for the area overseen by the authenticated user.
Inputs. `from_data`, `to_data`.
Process. `Service.get_neg_area_manager`, with internal filtering based on username/role.
Roles. A (read-only).

`GET /stock/get_storico_by_neg`

Purpose. Retrieves movements and totals for a specific store and period (useful to area managers and HQ users).

Inputs. `from_data`, `to_data`, `negozio`.

Process. `Service.get_storico_by_neg`.

Roles. A, M (with scope).

`GET /stock/get_excel_rettifiche` **(Export)**

Purpose. Generates an Excel report of adjustments over a time window, tailored to the user's language/territory.

Inputs. `from_data`, `to_data`.

Process. `Service.get_excel_rettifiche` returns a path; router streams it via `FileResponse`.

Roles. M, A.

`GET /stock/get_excel_area_negozi` **(Export)**

Purpose. Produces an Excel summary of stores under the area manager, with per-store aggregates.

Inputs. `from_data`, `to_data`.

Process. `Service.get_excel_area_negozi`.

Roles. A.

`GET /stock/get_excel_movimenti_neg` **(Export)**

Purpose. Exports a flattened dataset of movements per store and period (row per movement).

Inputs. `from_data`, `to_data`.

Process. `Service.get_excel_movimenti_neg`.

Roles. A.

`GET /stock/get_consegne_posticipate_by_neg`

Purpose. Returns counts of postponed deliveries per kit type for a given store (for dashboards and alerts).

Inputs. `cod_negozio`.

Process. `Service.get_consegne_posticipate_by_neg`.

Roles. M, A.

`GET /stock/get_excel_giacenza_negozi_before` **(Export)**

Purpose. Exports "start-of-period" stock levels by store, merged with postponed metrics.

Inputs. `from_data`, `to_data`.

Process. `Service.get_excel_giacenza_negozi_before`.

Roles. A.

`GET /stock/set_consegna_post` **(Flag Postponed Delivery)**

Purpose. Sets or clears the *consegna posticipata* flag for an order and kit type.

Inputs. `consegna_post` (bool), `ord` (order id), `kit_type` (`P`/`T`).

Process. `Service.set_consegna_post` inserts or deletes the flag row.

Roles. S, M.

`GET /stock/reorderBrochures` **(Brochure Reorder)**

Purpose. Requests brochure replenishment for the current store, optionally forced by the caller.

Inputs. `forced` [default: true].

Process. `Service.reorder_brochures` reads current stock and sends an email to the central team with store code and quantities.

Roles. M.

Response. `200 OK` with generic confirmation message.

`GET /stock/checkReorderBrochures` **(Reorder Eligibility)**

Purpose. Returns whether the store is currently eligible for an automatic brochure reorder (policy-driven threshold).

Inputs. None.

Process. `Service.check_reorder_brochures` executes a policy query and returns a boolean.

Roles. M.

5.2.1 JWT Authentication and Request Processing

This subsection details the implementation of authentication and request handling based on JSON Web Tokens (JWT). The goal is to provide a stateless, lightweight mechanism that satisfies the requirements defined in Chapter 3 for secure access (RBAC), traceability, and low operational overhead in a distributed retail context.

Credential Handling and Token Issuance

Authentication relies on two core functions: one for hashing user passwords at rest and another for minting short-lived access tokens after successful login. Sensitive values such as the JWT signing key and the password "pepper" are externalized as environment variables, injected at runtime. This ensures secrets are never committed into the source code and can be rotated without code modifications.

Password hashing (`get_password_hash`). The function derives a fixed-length digest using `RIPEMD-160` over the UTF-8 bytes of the password concatenated with a pepper string provided by the environment variable `PASSWORD_PEPPER`. The resulting hex digest is truncated to 32 characters. In production deployments, stronger password hashing functions such as **Argon2id** or **bcrypt** should be preferred, ideally through a vetted library like `passlib`, in combination with unique per-user salts and a secret pepper.

Access token creation (`create_access_token`). After successful authentication, an access token is created. The function copies the provided claims (`data`), adds an expiry claim (default 30 minutes from the current time), and signs the JWT with the symmetric key provided via `SECRET_KEY_JWT`. The signing algorithm defaults to `HS256` but can be adjusted by setting the environment variable `ALGORITHM_JWT`. Tokens embed only the minimal identity and authorization context required (`username`, `negozio`), reducing exposure if a token is intercepted.

Operational note. Access tokens are deliberately short-lived (30 minutes by default) to limit the blast radius in case of compromise. If longer sessions are required, a refresh-token mechanism can be adopted: a long-lived, httpOnly cookie refresh token combined with a short-lived access token for API interactions.

Token-Based Request Authentication

Every protected endpoint depends on `get_current_user`. FastAPI's `HTTPBearer()` dependency extracts the `Authorization: Bearer <JWT>` header and provides the raw token to the function.

Decoding and validation. The function:

1. Verifies the token signature using the secret key loaded from the environment.
2. Validates the expiry claim automatically via the JWT library.

3. Extracts application-specific claims (`username`, `negozio`) used for downstream business logic.

If validation fails, a `401 Unauthorized` is returned with a `WWW-Authenticate: Bearer` header, consistent with RFC 6750.

RBAC integration. This function ensures authentication, while authorization is enforced within the `Service` layer. This separation ensures routers remain lightweight and business rules remain centralized.

5.3 Frontend Implementation (Vue.js)

The frontend of the *Stock Uniters* application was developed as a **Vue 3 Single Page Application (SPA)**. Its design emphasizes responsiveness, modularity, and role-based adaptation. Thanks to the use of **Vite** as build tool, the system benefits from fast hot-reload during development and optimized bundling in production.

Frameworks and UI Libraries

The interface combines multiple libraries to achieve both usability and maintainability:

- **Bootstrap 5** and **Bootstrap Icons** for layout, grid system, and consistent styling.
- **PrimeVue** for advanced UI components (tables, dialogs, pickers).
- **FontAwesome** for iconography, integrated through the `vue-fontawesome` wrapper.
- **SweetAlert2** and **vue3-toastify** to provide immediate user feedback, confirmation dialogs, and toast notifications.

Routing and Navigation

Vue Router manages client-side routing. Each route corresponds to a functional area (inbound stock, outbound kits, returns, inventory, dashboards). Unauthorized access attempts are intercepted and redirected to the login view.

State Management

Application state is centralized using **Pinia**. Core state includes:

- authenticated user data and role,
- current store context,
- cached stock levels and dashboards.

Reactive data updates allow real-time synchronization with backend responses.

Forms and Validation

Form handling is reinforced through **Vee-Validate**, which provides declarative input validation rules. This ensures accuracy in operations such as delivery declarations, returns, and manual adjustments.

Internationalization

The system supports multilingual interfaces via **vue-i18n**, allowing Poltronesofà to deploy the application in diverse regions without duplicating code.

Camera Integration

For document capture and kit traceability, the frontend integrates:

- Native file upload components for attaching delivery notes or anomaly pictures.

Progressive Web App (PWA)

The system is distributed as a **Progressive Web App**, leveraging:

- `vite-plugin-pwa` and `register-service-worker` for offline caching and automatic updates.
- Installability on desktops and mobile devices, enabling a native-like user experience.

5.4 Authentication Flow

Authentication in *Stock Uniters* follows a **JWT-based stateless model**. The frontend exchanges user credentials with the backend during login, and receives a signed JSON Web Token (JWT) that encodes identity and role information.

Login and Token Acquisition

1. The user accesses the login page and submits their credentials.
2. The backend verifies the credentials and, if valid, issues a **JWT signed with a server-side secret**.
3. The frontend stores the JWT securely (`localStorage` or `sessionStorage`).

Token Usage

- On each API request, the frontend attaches the token in the header:

  ```
  Authorization: Bearer <jwt_token>
  ```

- The backend validates the token's signature and expiration before processing the request.
- If validation fails, the backend responds with `401 Unauthorized`, prompting re-authentication.

Access Control

- **Role-based checks** are performed based on claims inside the JWT (seller, store manager, area manager).
- The frontend uses the same role information to enable or disable navigation routes and UI features.
- Unauthorized requests are blocked early by both the frontend (route guards) and the backend (middleware).

Session Lifecycle

- Tokens have a short expiration time to reduce risk exposure.
- When the token expires, the user must log in again.
- This stateless approach simplifies scaling: no session state is stored on the server.

Chapter 6

Evaluation

The *Evaluation* chapter assesses the outcomes of the *Stock Uniters* project, both from an operational perspective and within the broader framework of Poltronesofà's digital transformation strategy.

Unlike previous chapters, which focused on technical design and implementation details, this section highlights how the introduction of the application reshaped existing processes, created measurable improvements, and generated organizational impact. The evaluation is therefore not limited to verifying functional correctness, but extends to analyzing how technology adoption has influenced workflows, roles, and decision-making practices.

A central aspect of this chapter is my direct involvement not only as a developer, but also as an active participant in strategic choices. By contributing to the definition of evaluation criteria, key performance indicators, and pilot test parameters, I acted as a bridge between the technical implementation and the organizational goals of the company.

6.0.1 Application Pages Overview

This section presents the main user interfaces of the *Stock Uniters* application. These pages represent the daily entry points for store staff and managers, and their design strongly influenced the usability and effectiveness of the system.

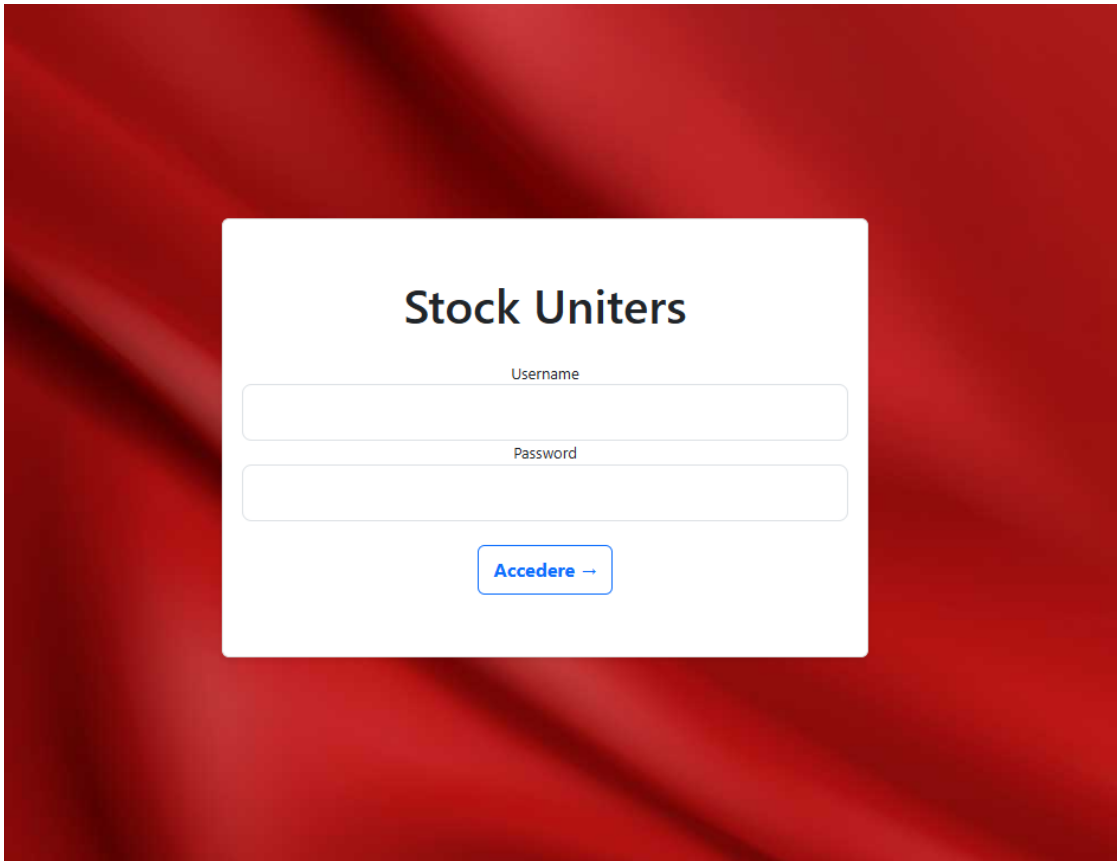

Figure 6.1: Login page of the application

The login page provides secure access through username and password. Once authenticated, users are redirected to their dedicated workspace, with functionality adapted to their role (seller, store manager, or area manager). This ensures that each user sees only the features relevant to their responsibilities.

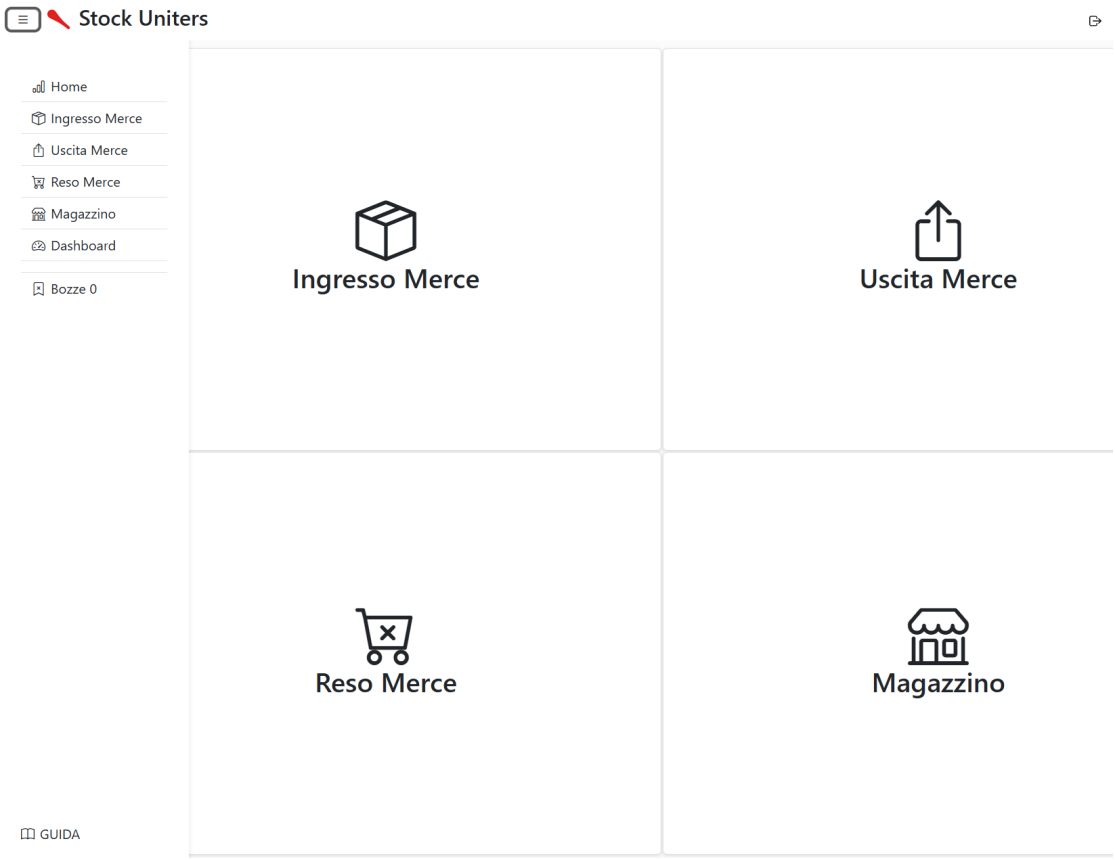

Figure 6.2: Home page and navigation menu

The home page serves as the central navigation hub. From here, users can quickly access the core modules: Ingresso Merce, Uscita Merce, Reso Merce, Magazzino, and Dashboard. Its simplicity reduces complexity and shortens the time required to carry out daily operations. This menu is also available in the side bar.

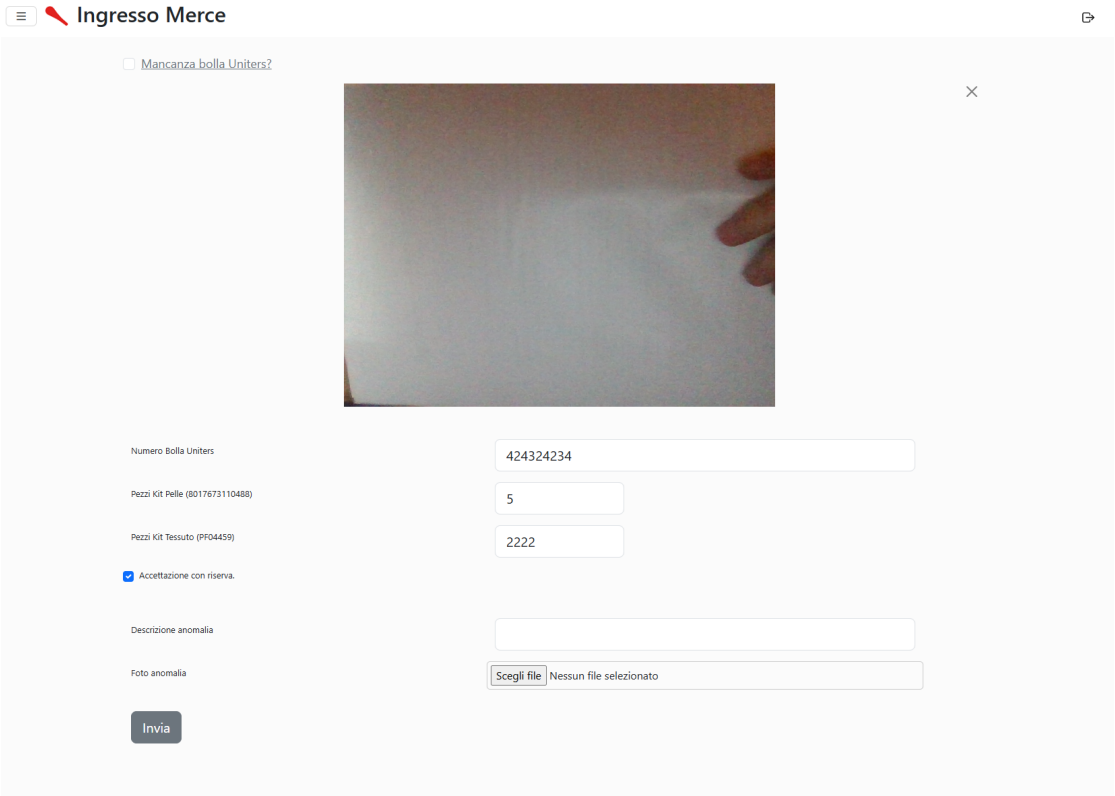

Figure 6.3: Ingresso Merce page

The Ingresso Merce page is dedicated to registering incoming stock. Users record the delivery note, quantities, and attach photographs in case of anomalies. This workflow enforces accountability by requiring conditional acceptance if discrepancies are found.

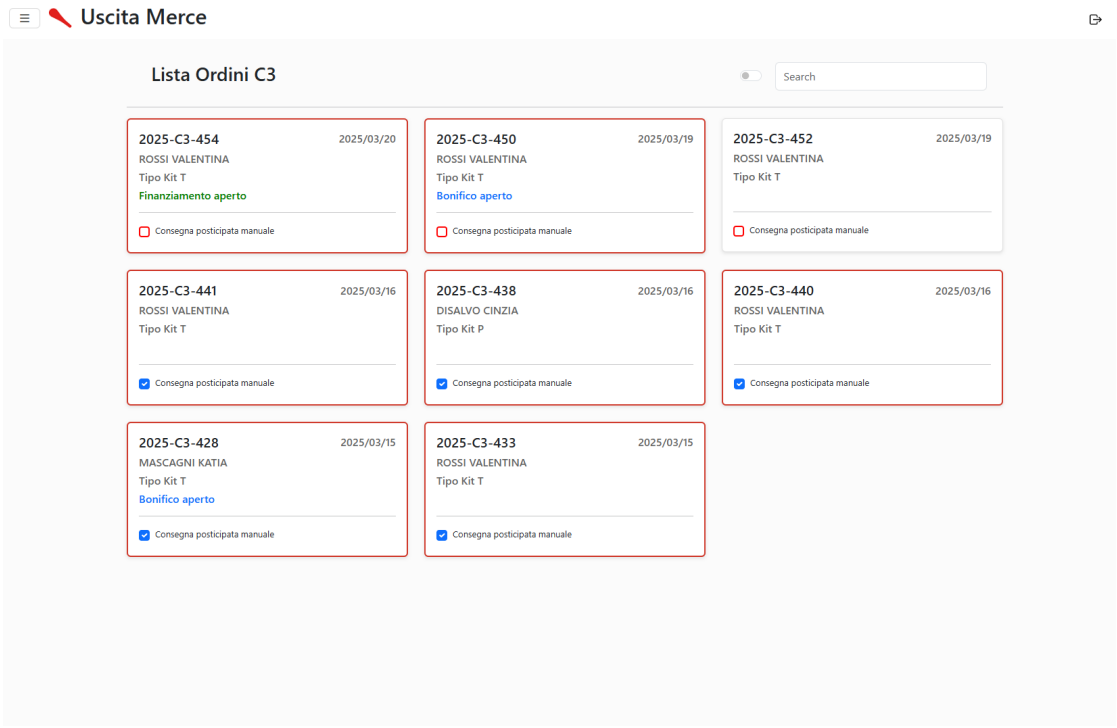

Figure 6.4: Uscita Merce page

The Uscita Merce page allows sellers to link outgoing kits to customer orders covered by warranty. It ensures that only validated orders can be fulfilled, thereby improving traceability and guaranteeing consistency between physical and digital records.

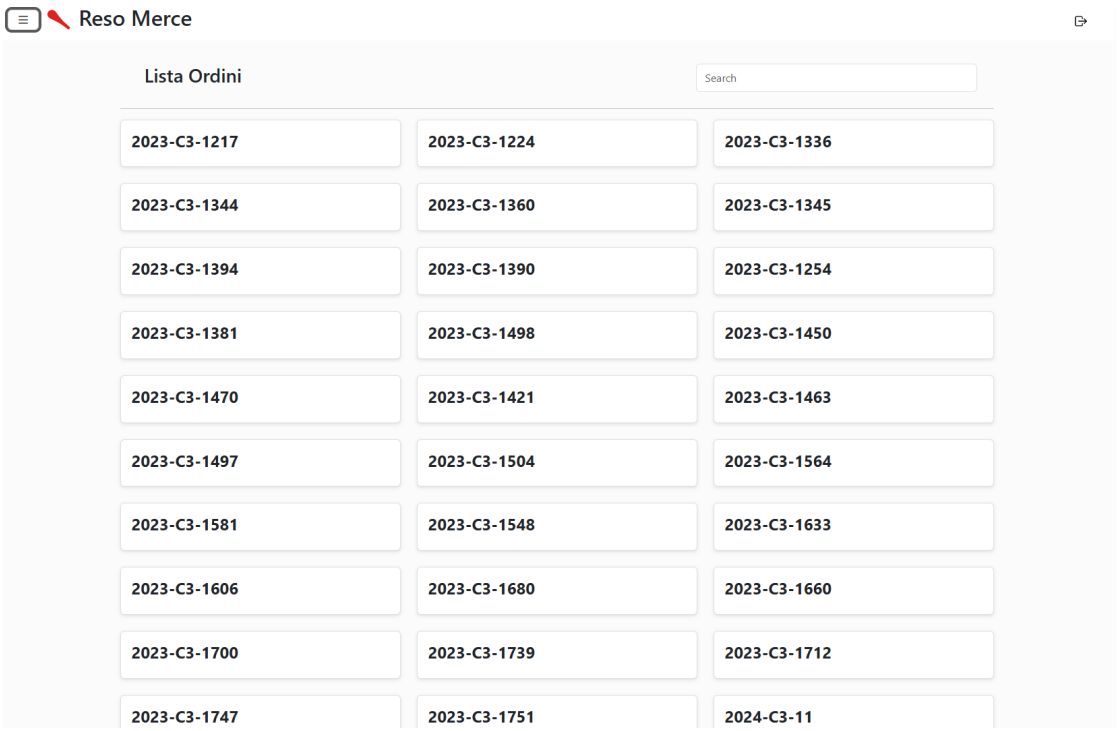

Figure 6.5: Reso Merce page

The Reso Merce page is used to record kit returns. By associating each return with the original order, the system guarantees full traceability of movements and simplifies subsequent reconciliation.

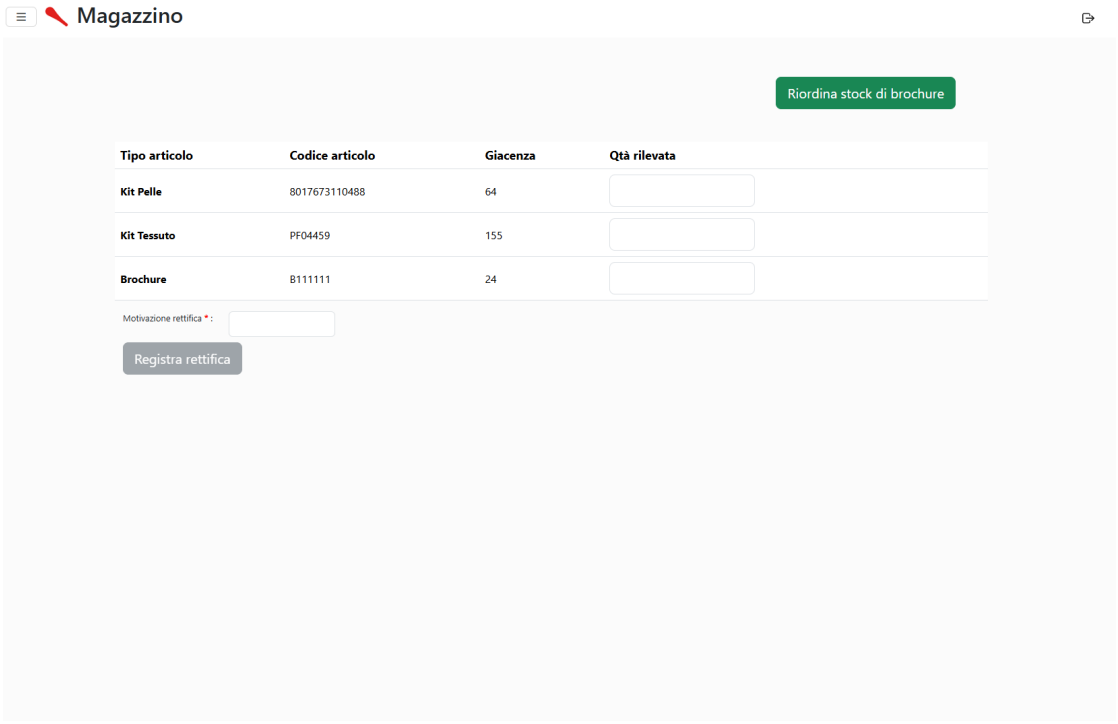

Figure 6.6: Magazzino page

The Magazzino page provides an overview of the current stock, including both kits and brochures. It enables managers to perform adjustments during monthly inventory checks. Every correction requires a justification, reinforcing transparency and governance.

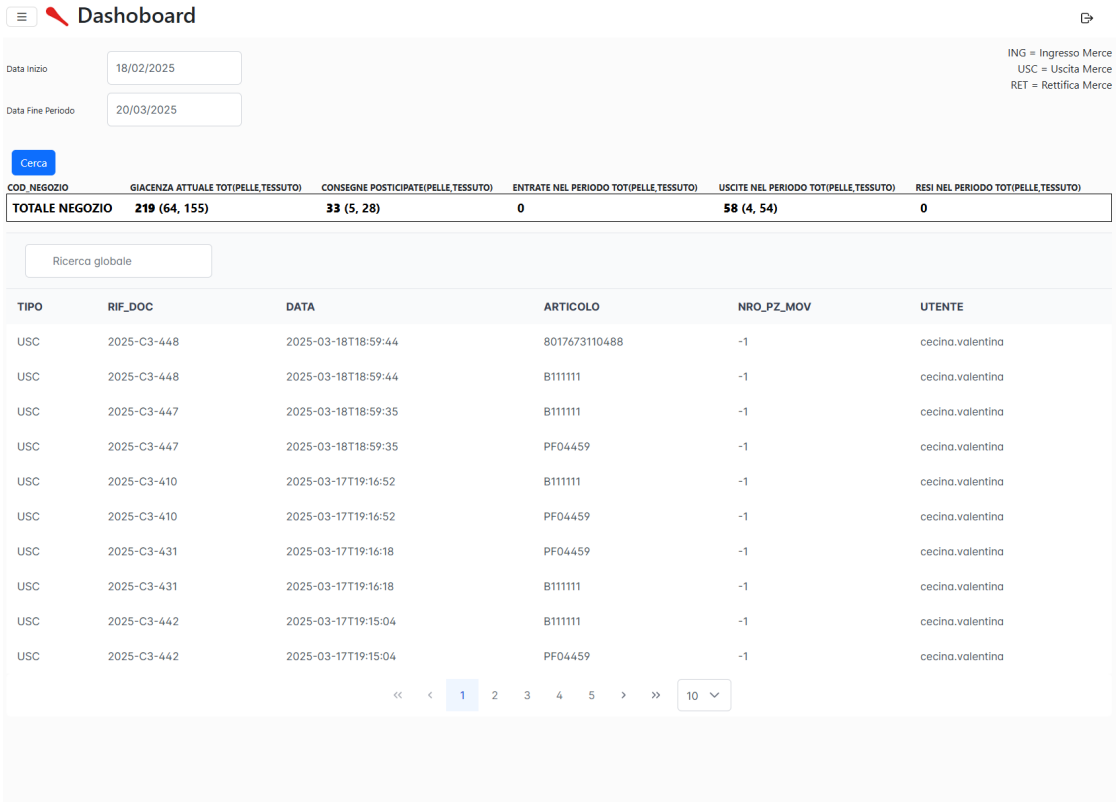

Figure 6.7: Dashboard page

The Dashboard aggregates indicators such as stock entries, exits, returns, and postponed deliveries. This visualization supports managers and area supervisors in monitoring operations and making data-driven decisions in real time.

6.1 Evaluation Setup

Before the introduction of the *Stock Uniters* application, inventory management processes in Poltronesofà stores were handled manually using Excel spreadsheets.

Store staff were responsible for updating stock quantities in local Excel files, which were then periodically shared with Area Managers. These managers had to manually monitor updates, consolidate data, and ensure reporting accuracy across all their assigned stores.

This system was prone to several inefficiencies:

- Delays in communication and consolidation of data.

- Inconsistent or missing entries due to human error.
- Difficulty in tracing who made a change and when.

In addition to these technical shortcomings, the Excel-based workflow represented a broader organizational challenge: it perpetuated a fragmented, non-standardized approach to information sharing, which was increasingly misaligned with the company's ambition to modernize its digital infrastructure. From this perspective, the evaluation of the new system was not limited to verifying functional correctness, but was framed as part of a wider process of **digital transformation**.

To test and evaluate the effectiveness of the new application, a pilot phase was carried out in five strategically selected stores. These stores were chosen to represent different geographical areas and operational contexts, ensuring that the results of the trial would be meaningful and generalizable. Both store staff and Area Managers were instructed to use the app during their day-to-day operations, replacing the traditional Excel-based process.

My involvement in this phase went beyond the technical role of software implementation. I actively contributed to defining the evaluation criteria together with managers, establishing metrics not only for efficiency (speed of data entry, reduction in errors) but also for adoption and usability (ease of training, user satisfaction, and perceived improvement in accountability). This participatory approach positioned me as an enabler of organizational change, bridging the gap between the IT department and operational staff, and ensuring that the pilot could serve as a concrete step in the company's overall digitalization roadmap.

6.2 Automation and Improvements

The introduction of the *Stock Uniters* application brought several immediate benefits that were perceived not only at a technical level but also at an organizational one.

Real-time data entry Stock movements were recorded directly by store employees through the app, ensuring that information was immediately available at the company level without the need for subsequent consolidation.

Elimination of Excel dependency The complete removal of local spreadsheets significantly reduced the risk of fragmented or outdated information. This also marked a cultural shift, replacing a historically manual process with a standardized, centralized system.

Immediate dashboard access Area Managers could monitor stock status and movements across all stores from a single interface. This enabled data-driven decision making, reducing the time spent on manual checks and freeing managers to focus on higher-value activities.

User accountability Each action within the app was tied to an authenticated user, with timestamps and mandatory justification fields for every adjustment. This not only improved traceability but also fostered a sense of responsibility among employees, aligning operational behavior with corporate governance principles.

Beyond these measurable improvements, the project also served as a catalyst for broader organizational change. By participating directly in the design of workflows and the prioritization of features, I contributed to ensuring that automation did not merely replicate existing practices but actively improved them. Decisions such as enforcing role-based dashboards, mandating justifications for adjustments, and providing managers with integrated dashboards were not purely technical: they reflected a conscious effort to embed accountability, transparency, and efficiency into the company's daily operations.

In this sense, the application functioned as both a technological tool and a vehicle for **digital transformation**, reshaping how information was captured, validated, and shared across the Poltronesofà retail network.

6.3 Impact on Operations

During the pilot in the five stores, the following operational improvements were observed:

Stock entries and exits were processed more quickly and accurately The direct input of data into the application minimized transcription errors and removed delays caused by manual updates.

Inventory reconciliation was simpler and faster Managers could rely on automatically generated records with complete traceability, significantly reducing the effort required to perform monthly checks.

Area Managers reduced the time spent requesting or aggregating data manually Dashboards and exports provided consolidated visibility across all assigned stores, eliminating the need for repeated communication with store staff.

Store managers also reported that the system improved internal control and reduced ambiguity during monthly stock checks. The ability to verify who carried out each action, and when, added a new layer of transparency that strengthened accountability and trust within local teams.

From an organizational perspective, these operational changes highlight how the adoption of the *Stock Uniters* application went beyond simple efficiency gains. By participating directly in the definition of validation rules, data flows, and reporting mechanisms, I helped shape a solution that integrated seamlessly into daily retail operations while also reinforcing corporate governance practices.

In this way, the project not only delivered measurable time savings but also fostered a cultural shift towards **data-driven and standardized operations**, a key milestone in Poltronesofà's broader digital transformation journey.

6.4 Relevant KPI: Stock Registration Time

One particularly relevant Key Performance Indicator (KPI) for evaluating the efficiency of the new system is the **stock registration time**, defined as the average delay between the physical reception of goods in-store and their digital registration into the system. This KPI was selected because it directly reflects one of the most critical pain points of the previous process: the gap between operational reality and the availability of updated information for decision-making.

In the Excel-based workflow, this delay could span several hours or even days, depending on staff availability and whether the spreadsheet was correctly updated and forwarded to Area Managers. This meant that managers often operated with incomplete or outdated data, reducing their ability to intervene promptly in case of stock shortages, discrepancies, or anomalies. Moreover, the absence of a standardized procedure increased variability between stores, making it difficult to compare performance or enforce consistent practices.

Thanks to the new application, the registration workflow was streamlined and partially automated:

- The stock registration process begins as soon as the goods are received, removing the possibility of postponing the update.

- Store staff use tablets or smartphones to photograph the delivery document and record quantities immediately, ensuring that both quantitative and documentary evidence are captured at the same time.

- Data is saved and made available to managers in real time, allowing for immediate visibility across the retail network without waiting for manual consolidation.

The introduction of this workflow produced tangible improvements. As shown in Table 6.1, the average delay in registration dropped from **3.2 days** to **0.8 days**, a reduction that not only accelerates operational reporting but also minimizes the risk of discrepancies between physical and digital stock. Similarly, the **document completeness rate** increased significantly, reflecting how the app encouraged staff to attach delivery notes systematically. The reduction in **mismatch incidence** confirms that faster and more structured registration leads to fewer inconsistencies, while the overall **inventory accuracy** rose to 92%, demonstrating that the information used by managers is now more reliable.

KPI	**Before (Excel)**	**After (App)**
Stock registration delay	3.2 days	**0.8 days**
Document completeness rate	68%	**96%**
Mismatch incidence	22%	**7%**
Inventory accuracy	79%	**92%**

Table 6.1: Example KPI comparison before and after introducing the system

Beyond the numerical results, this KPI illustrates how the application contributed to creating a more disciplined and transparent workflow. By integrating photo capture, mandatory data entry, and immediate synchronization, the system eliminated delays that were previously accepted as unavoidable. As a result, Area Managers gained earlier access to accurate data, enabling them to focus on analysis and decision-making rather than chasing updates.

This demonstrates how the *Stock Uniters* application was not only a technological upgrade, but also an enabler of process standardization, aligning store-level activities with the company's strategic goal of real-time visibility across its retail network.

Chapter 7

Conclusion and Future Work

This final chapter reflects on the results achieved throughout the thesis, positioning the *Stock Uniters* application within the broader context of Poltronesofà's digital transformation. It summarizes the main contributions of the project, highlights the limitations encountered during the pilot phase, and outlines possible directions for future development. The intention is not only to close the technical narrative of design and implementation, but also to emphasize the role of this work as an enabler of organizational change, where digital tools support greater efficiency, accountability, and data-driven decision-making across the retail network.

7.1 Summary of Contributions

This thesis presented the design and implementation of a web-based stock management system tailored to the operational needs of Poltronesofà, addressing inefficiencies of the previous Excel-based workflow and aligning daily operations with the company's digital transformation strategy.

The developed solution delivered multiple contributions, both operational and technological:

Simplification and standardization of workflows Inventory management at the store level was unified into a single platform, reducing variability between locations and ensuring consistent practices.

Traceability and accountability Every stock movement (entry, exit, return, adjustment) is logged with user credentials, timestamps, and justification fields, creating a transparent and auditable process.

Managerial visibility Dashboards and reporting tools provide store managers and area managers with real-time access to stock levels and historical movements, enabling faster and better-informed decisions.

Integration of supporting features Document scanning, image uploads, and inventory corrections are embedded in the workflow, reducing manual steps and ensuring data completeness.

From a technical standpoint, the system demonstrates the successful application of a modern, full-stack architecture:

Backend FastAPI provides a performant, type-safe, and maintainable framework, integrated with Oracle to ensure compatibility with enterprise infrastructure.

Frontend Vue.js enables the development of a responsive, mobile-friendly Single Page Application, ensuring usability across devices and in varied store contexts.

Security JWT-based authentication and role-based access control safeguard sensitive data and enforce user accountability.

Beyond these tangible results, the project also represents a contribution in terms of organizational change. By embedding operational rules into the application logic (e.g., warranty association with kit exits, mandatory justifications for adjustments), the system became not only a technical tool but also a mechanism for enforcing corporate governance principles.

My active involvement went beyond programming: I participated in the definition of requirements, evaluation criteria, and pilot testing, acting as a link between the IT department and store operations. This role highlights one of the central contributions of the thesis: demonstrating how a technical solution can serve as a catalyst for **process innovation and cultural change**, supporting Poltronesofà in its transition towards data-driven decision making and digital standardization.

7.2 Limitations

While the pilot deployment has shown positive results, a few limitations were identified that are important to acknowledge both from a technical and organizational

perspective. Recognizing these limitations is a fundamental step in ensuring that the system can evolve in alignment with the broader goals of Poltronesofà's digital transformation journey.

Connectivity dependency Upload operations can be delayed in stores with unstable Wi-Fi. Although this issue did not prevent the overall success of the pilot, it occasionally caused frustration among employees who expected immediate confirmation of their actions. This limitation emphasizes how technological innovation must be supported by adequate infrastructure: without stable connectivity, even the most advanced digital tools risk being perceived as unreliable. It also highlights the strategic need to coordinate IT solutions with investments in store-level infrastructure, ensuring that digital adoption is not hindered by uneven technical readiness.

User learning curve Some users initially confused "conditional acceptance" with "forced closure." This limitation underlines a recurring challenge in digital transformation projects: introducing new systems often requires redefining not only tools but also mental models and organizational vocabulary. Even a well-designed interface cannot fully prevent misunderstandings if the underlying processes are unfamiliar to staff. The lesson learned is that training, user support, and continuous feedback loops are essential components of adoption. Technology alone is not sufficient; it must be accompanied by a cultural change that helps employees internalize new ways of working.

No offline mode The system currently does not support offline operations or synchronization. While acceptable in the pilot phase, this limitation could become a barrier to large-scale deployment, especially in areas where connectivity is weak or inconsistent. Offline support is particularly relevant for mobile use cases, such as when employees register stock movements directly from the warehouse floor. The absence of this functionality limits the robustness of the solution and can undermine trust in the tool if users perceive it as dependent on external conditions beyond their control.

7.3 Future Work

To further improve the system and consolidate its role within Poltronesofà's digital ecosystem, the following developments are recommended. These enhancements do not only address technical limitations, but also aim to reinforce adoption, scalability, and long-term sustainability.

Offline-first capabilities An offline-first approach would allow store staff to continue operating even in the absence of a stable internet connection, with local data storage and automatic synchronization once connectivity is restored. This functionality would mitigate one of the main limitations identified during the pilot, ensuring reliability and uninterrupted operations. From a strategic point of view, it would also strengthen user trust in the system, making digital workflows resilient to external conditions.

Improved onboarding and training While the application is designed to be intuitive, initial confusion with certain terms and processes revealed the importance of guided onboarding. Future iterations could integrate interactive tooltips, contextual in-app guides, and clearer terminology. These improvements would lower the learning curve, standardize knowledge across stores, and reduce the dependence on external training sessions. Effective onboarding is not only a usability enhancement but also a key factor in ensuring that the digital transformation becomes embedded in daily practice.

Expanded reporting Current dashboards provide essential visibility, but future versions could include time-based trends, error heatmaps, and automatic anomaly detection. Such features would elevate the application from an operational tool to a decision-support system, enabling managers to anticipate issues rather than simply react to them. This evolution would align with the broader goal of fostering a culture of data-driven management across the company.

Scalability testing As the pilot demonstrated the system's potential, preparing for large-scale deployment across all Poltronesofà stores becomes a priority. Stress testing and scalability validation are required to guarantee performance under higher transaction volumes and concurrent users. Beyond technical robustness, this step represents a strategic milestone: scaling the solution means extending the benefits of efficiency, traceability, and standardization to the entire retail network.

Integration with other systems Connecting the application with existing ERP platforms or sales tools (e.g., FIP) would further increase its value. Integration would reduce data duplication, align stock management with sales processes, and allow end-to-end traceability from product delivery to final sale. This step would consolidate the application's role as part of a broader digital ecosystem, strengthening its contribution to the company's digital transformation roadmap.

Taken together, these developments illustrate that the Stock Uniters system is not a finished product but an evolving platform. Each improvement contributes

not only to resolving technical constraints but also to embedding digital practices more deeply into organizational culture.

Final Remarks

The *Stock Uniters* system demonstrates the value of targeted digital transformation in retail operations. It shows how even relatively simple tools—when tailored and well-integrated—can yield measurable efficiency, traceability, and user satisfaction benefits.

From a technical perspective, the project confirmed that modern frameworks such as FastAPI and Vue.js can be successfully combined with enterprise infrastructure like Oracle to deliver agile yet robust solutions. From an organizational perspective, it highlighted how embedding operational rules directly into the application enforces consistency, accountability, and transparency across stores.

A key takeaway from this experience is that digital transformation is not only about deploying new technologies, but about rethinking processes, roles, and decision-making practices. Through my active participation in both the technical development and the strategic design of workflows, this thesis exemplifies how the figure of the developer can also act as an agent of change, bridging technical expertise with organizational objectives.

Ultimately, the system represents more than a software solution: it is a concrete step towards a culture of data-driven decision-making within Poltronesofà. By transforming a previously fragmented and manual workflow into a standardized, traceable, and real-time process, the project laid the foundations for further innovations and paved the way for the company's ongoing digital evolution.

Bibliography

[Bis02] Matt Bishop. *Computer Security: Art and Science*. Addison-Wesley, 2002.

[Coc01] Alistair Cockburn. *Writing Effective Use Cases*. Addison-Wesley, 2001.

[Coh04] Mike Cohn. *User Stories Applied: For Agile Software Development*. Addison-Wesley, 2004.

[Dev22] Mozilla Developers. Progressive web apps (pwas), 2022. `https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps`.

[iso19] ISO 9241-210:2019 ergonomics of human-system interaction — human-centred design for interactive systems, 2019. International Organization for Standardization, `https://www.iso.org/standard/77520.html`.

[LL20] Kenneth C. Laudon and Jane P. Laudon. *Management Information Systems: Managing the Digital Firm*. Pearson, 16 edition, 2020.

[Nie93] Jakob Nielsen. *Usability Engineering*. Academic Press, 1993.

[Ram22] Sebastián Ramírez. Fastapi documentation, 2022. `https://fastapi.tiangolo.com`.

[RDT01] Ananth Raman, Nicole DeHoratius, and Zeynep Ton. Execution: The missing link in retail operations. 2001.

[RR10] Leon Rosen and Roger Resnick. *Web Application Architecture: Principles, Protocols and Practices*. Wiley, 2010.

[SCFY96] Ravi Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-based access control models. 1996.

[Som16] Ian Sommerville. *Software Engineering*. Pearson, 10 edition, 2016.

[Vue23] Vue.js Team. Vue.js documentation, 2023. `https://vuejs.org`.