INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




Automatização de Processos para Previsão e Análise de Inventário

DIOGO FILIPE MAGALHÃES SALVADOR
Setembro de 2024

POLITÉCNICO
DO PORTO

INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




Automating Processes for Inventory Forecasting and Analysis

DIOGO FILIPE MAGALHÃES SALVADOR
Setembro de 2024

POLITÉCNICO
DO PORTO

ISEP INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO

Automating Processes for Inventory Forecasting and Analysis

Diogo Filipe Magalhães Salvador

A dissertation submitted in partial fulfillment of the requirements for the degree of Master of Science, Specialisation Area of Software Engineering

Supervisor: Dra. Fátima Rodrigues
Co-Supervisor: Dra. Dulce Mota

Porto, September 14, 2024

Statement of Integrity

I hereby declare having conducted this academic work with integrity.

I have not plagiarised or applied any form of undue use of information or falsification of results along the process leading to its elaboration.

Therefore the work presented in this document is original and authored by me, having not previously been used for any other end.

I further declare that I have fully acknowledged the Code of Ethical Conduct of P.PORTO.

ISEP, Porto, September 14, 2024

Abstract

In the fast-evolving domain of inventory management, businesses must navigate complex challenges involving data management, forecasting, and operational efficiency.

This thesis focuses on the development of an automated inventory forecasting platform primarily for Kontrolsat, a technology and e-commerce company, but also extendable to other businesses.

The platform was designed to analyze historical sales data, sales trends, and product attributes to provide accurate inventory predictions.

Machine learning models and ensemble learning techniques were employed to enhance the precision of forecasts, thus enabling better decision-making and reducing the risk of stockouts or overstocking.

In order to guarantee optimal performance, a large portion of this focusses on the system's scalability and maintainability, utilising software engineering principles like modularity, domain-driven design, and multi-processing.

The system was also built to be easily customizable and user-friendly, allowing seamless integration with Kontrolsat's existing infrastructure while accommodating future growth.

The results are promising, with 40 products presenting acceptable predictions, which corresponds to a success rate of 90% in the predictions. This demonstrates the platform's ability to enhance inventory management efficiency by providing valuable insights and accurate forecasts.

There are, however, some prediction entries that require further analysis, as their error percentage is extremely high. This mostly happens with products that have a very low number of historical sales data, or none at all.

While the system is still in active development, it has already shown promising results and has the potential to be a valuable tool for businesses.

Keywords: Forecasting, Inventory Management, Automation, Data Analysis, Predictive Analytics, Process Optimization

Resumo

Tendo em conta a rápida evolução da gestão de inventário, as empresas enfrentam desafios complexos relacionados à gestão de dados, previsões e eficiência operacional.

Esta dissertação foca-se no desenvolvimento de uma plataforma automatizada de previsão de inventário, principalmente para a Kontrolsat, uma empresa de tecnologia e e-commerce, mas também extensível a outros negócios.

A plataforma foi desenvolvida para analisar dados históricos de vendas, tendências de vendas e atributos dos produtos para fornecer previsões de inventário precisas.

Modelos de Machine Learning e técnicas de ensemble learning foram utilizados para melhorar a precisão das previsões, permitindo uma melhor tomada de decisões e reduzindo o risco de ruturas de stock ou excesso de stock.

Para garantir um desempenho ideal, grande parte do trabalho foca-se na escalabilidade e manutenção do sistema, utilizando princípios de engenharia de software como modularidade, design orientado ao domínio e multiprocessamento.

O sistema também foi construído para ser facilmente personalizável e intuitivo, permitindo uma integração perfeita com a infraestrutura existente da Kontrolsat, ao mesmo tempo que acomoda o crescimento futuro.

Os resultados são promissores, com 40 produtos a apresentarem previsões aceitáveis, o que corresponde a uma taxa de sucesso de 90% nas previsões. Isto demonstra a capacidade do sistema melhorar a eficiência na gestão de inventário, fornecendo insights valiosos e previsões precisas.

No entanto, há algumas previsões que requerem uma análise mais aprofundada, porque a sua percentagem de erro é extremamente elevada. Isto ocorre principalmente com produtos que têm um número muito reduzido, ou inexistente, de dados históricos de vendas.

Embora o sistema ainda esteja em desenvolvimento, já demonstrou resultados promissores e tem o potencial de se tornar uma ferramenta valiosa para as empresas.

Acknowledgement

I would like to express my gratitude to my colleague Hugo Carrulo for his help and constant presence throughout my entire Bachelor's and Master's degree. Many of the projects created shifted my perspective of software development, and, without him, it would have been a completely different experience. I also want to thank the head of the company Kontrolsat, Bruno Carrulo, for his encouragement and for always being available to help and guide me in the development of my professional career. I would also like to thank my supervisor, Dr. Fátima Rodrigues, and my co-supervisor, Dr. Dulce Mota, for the support provided throughout the development of this document and the project.

Contents

List of Figures

List of Tables

List of Acronyms

AI	Artificial Intelligence.
ANN	Artificial Neural Network.
ARIMA	AutoRegressive Integrated Moving Average.
BNN	Bayesian Neural Network.
CORS	Cross-Origin Resource Sharing.
CPU	Central Processing Unit.
CSV	Comma Separated Values.
DB-MPDSS	Data-based model predictive decision support system.
DDD	Domain-Driven Design.
DI	Dependency Injection.
DRL	Deep-Reinforcement Learning.
DSS	Decision Support System.
EOQ	Economic Order Quantity.
ERP	Enterprise Resource Planning.
ETL	Extract, Transform, Load.
FAC	Invoice.
FMCG	Fast-Moving Consumer Goods.
FR	Invoice Receipt.
GradientBoosting	Gradient Boosting Model.
GUI	Graphical User Interface.
JSON	JavaScript Object Notation.
JWT	JSON Web Token.
KNN	K-Nearest Neighbors.
MAE	Mean Absolute Error.
MAML	Model-Agnostic Meta-Learning.
MAPE	Mean Absolute Percentage Error.
MCDM	Multicriteria Decision Making.
ML	Machine Learning.
NFR	Non-functional Requirement.
NoSQL	Not Only Structured Query Language.

RandomForest	Random Forest Model.
RL	Reinforcement Learning.
RMSE	Root Mean Square Error.
SaaS	Software as a Service.
Sage 50cc	Sage 50 Cloud - Accounting Software.
SARIMA	Seasonal AutoRegressive Integrated Moving Average.
SCRUM	SCRUM Methodology.
SHA-256	Secure Hash Algorithm 256-bit.
SVM	Support Vector Machines.
XGBoost	eXtreme Gradient Boosting Model.
XLSX	Microsoft Excel Spreadsheet Format.
XSS	Cross-Site Scripting.

Chapter 1

Introduction

The present chapter pretends to provide the main theme of this dissertation, highlighting the analysis and development of an automated inventory forecasting platform.

1.1 Contextualization

Inventory management is undergoing an extraordinary shift in the fast-paced corporate environment of today, which is fuelled by data-centric decision-making and rapid technological innovation.

Due to the instability of market needs and the growing complexity of supply chains, traditional techniques of inventory management are no longer sufficient. These traditional methods frequently have trouble adapting swiftly to shifts in the market, which can result in problems like overstocking or understocking as well as higher holding costs.

In order to meet client needs, maintain inventory levels, and save operating expenses, businesses now need to implement more dynamic systems. This is especially pertinent to Kontrolsat, the dissertation's central company, as inventory management is currently dominated by manual decision-making processes.

For context, Kontrolsat is a technology and e-commerce company that specializes in the sale of electronic products. It has both a physical store and an online platform, which are a significant part of its business. The company has the goal to digitalize most of its processes, and the inventory management system is one of the most important ones.

Kontrolsat can improve inventory levels and forecast demand more accurately by putting in place an automated forecasting system that makes use of data-driven insights.

A method like this will cut down on excess inventory and related expenses while enabling the business to concentrate on stocking the most well-liked and in-demand products. The switch to computerised inventory forecasting is not without its difficulties, though.

In order to guarantee that the system delivers the desired benefits without interfering with the business's activities, several obstacles must be overcome.

1.2 Objectives

The following objectives outline the system's vision and its expected outcomes:

Development of a Comprehensive Forecasting Platform

The platform should support data import from various types of sources such as Comma Separated Values (CSV) and Microsoft Excel Spreadsheet Format (XLSX) files or even databases, and be capable of processing this data for Machine Learning (ML) models.

Enhanced Accuracy and Reliability of Forecasts

The forecasting models must deliver high accuracy to prevent overstocking or stockouts. The system should provide a reliable and consistent forecast for the company.

User Customization and Flexibility

The system must allow users to customize their experience by configuring settings for how the system operates. This customization should be user-friendly and intuitive. Users should be able to easily adjust settings to meet their specific needs, especially features for the models and algorithms used. Also, users should be able to filter responses based on the product ID, category, price range, and more.

Data Retrieval and Anomaly Detection

Anomaly detection should be integrated to highlight irregular patterns in historical sales data. This will aid the system's predictions and also provide the user with valuable insights into past sales.

Users should be able to retrieve historical sales data in a user-friendly format, that can be easily interpreted and analyzed, and even usable for other purposes.

Multi-Processing for Performance Optimization

The system should be able to handle large datasets and process them efficiently. This will require the use of multi-processing techniques to optimize performance and speed up the processing time.

Developer-Focused Enhancements

The system should be designed with developers in mind as well. For now, only myself will be working on the system, but in the future, it should be easy to onboard new developers.

The system should support distinct environments, that are easy to switch between and have their own set of configurations.

Scalability and Sustainability

The system must have a modular architecture that ensures that components can be added or updated without significant disruptions. This will allow the system to scale as the company grows and its needs change.

1.3 Methodology

The methodology was structured to effectively meet the objectives set out for the development of the automated inventory forecasting platform. The project was divided into three major development phases, referred to as trains.

Each train normally lasted for 2 months and focused on a specific aspect of the system.

- Train 1 (February - March 2024): Focused on laying the groundwork for the project. This was considered the investigation and planning phase.

- Train 2 (April - May 2024): Concentrated on the implementation of forecasting algorithms and machine learning models in a simple project.
- Train 3 (June - July 2024): Migration to a more complex architecture, that would be used in the final project. This phase focused on transforming the simple project from the previous train into a more scalable and professional looking solution.
- Train 4 (August - September 2024): Finalization of the project, and documentation of its architecture, design and implementation.

Each train was divided into 4 sprints, each lasting two weeks. At the start of each sprint, a sprint planning session took place to define the specific tasks and prioritize which features to develop. Given that there was only one developer, daily SCRUM meetings did not occur. Instead, more flexible task management was used.

At the end of each sprint, a retrospective was conducted to evaluate how the sprint had gone.

This entire methodology was defined at the beginning to properly organise the development process, and to ensure that the project would be delivered on time and no features would be left behind.

Having that defined, the project could then start with the comprehensive literature review that focused on key domains, including inventory management, automation, machine learning, and software engineering.

After that, a thorough requirement analysis was carried out to determine exactly what the system needed to do and how it need to be created.

When the requirements were established, the architectural design of the platform was reviewed, selecting technologies and tools that best fit the project's needs. Special attention was given to ensuring the system's modular structure.

This methodology helped a lot in the development process. It allowed to continuously build and improve the system.

As previously stated, both the simple and complex project were developed. By the end, and thanks to the methodology, the system was able to handle large datasets and provide accurate forecasts.

In the end, this methodology was comprehensive, iterative and allowed for development of a robust and scalable system.

1.4 Ethical Considerations

The development and implementation of an automated inventory management platform at Kontrolsat necessitate rigorous attention to data privacy and security. Given that the platform will handle sensitive data, including sales records and potentially customer information, it is imperative to ensure that all data is safely handled. The platform should incorporate robust encryption and access control mechanisms to safeguard against unauthorized data access.

It is also essential to maintain a level of transparency that allows for human oversight, since all of the decisions made by the platform can be changed and should be confirmed by a

stakeholder at the company. Errors and miscalculations should be properly reported, so that the platform can be fixed and improved, and so that any orders should be handled manually during the maintenance period. The platform's algorithms should be reviewed on a regular basis for biases that could lead to unfair outcomes, such as unfairly favouring specific products.

The introduction of automation in inventory management may also raise concerns regarding its impact on the workforce. However, this platform is being developed as a way to help the people at the company, so that they can focus on more important tasks. The decision-making process can be automated, but the order confirmation should always be handled manually.

Lastly, since Artificial Intelligence (AI) and ML has such a big role in inventory management algorithms, it is critical to consider the moral, societal, and legal implications of their behaviour. (Dignum 2018) Some of the ethics, are shown in Figure 1.1.

In practice, this entails the technical incorporation of ethical reasoning capabilities, regulatory and engineering methods for assessing the ethical implications of AI systems, and adherence to codes of conduct and standards that ensure the integrity of developers and users. (Ethics for Design) (Dignum 2018)

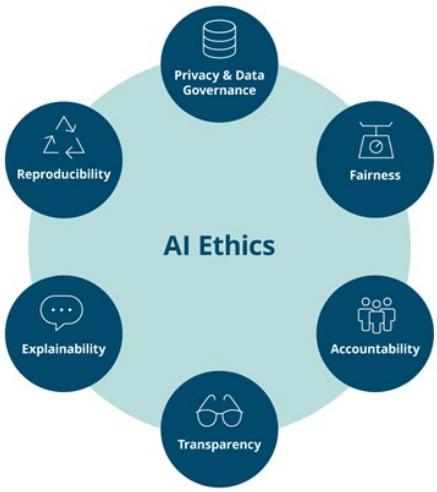

Figure 1.1: Ethics in Artificial Intelligence

These ethical considerations mentioned in Figure 1.1, are integral to the responsible development and implementation of the automated inventory management platform at Kontrolsat.

1.5 Structure

Seven chapters, each concentrating on a distinct facet of creating the automated inventory forecasting platform, make up this thesis.

This current chapter introduces the context of the project, outlining the motivation behind this research. The objectives and methodology of the project are also detailed, along with the ethical considerations required for the development of an automated system.

Chapter 2 provides an in-depth review of the theoretical foundations relevant to the research. It covers key concepts in inventory management, predictive analytics, machine learning, and software engineering principles. The review also addresses the historical evolution of inventory systems, current trends, and case studies of real-world applications in inventory forecasting.

Chapter 3 provides a detailed analysis of the system's requirements, including functional and non-functional requirements. It also discusses the value proposition of the system and the architectural approach that will be employed.

Chapter 4 goes into detail on the system's architecture, highlighting the platform's modular structure and domain-driven design methodology. It goes on how many elements, including the app, domain, infrastructure, and machine learning modules, interact to provide scalability and adaptability.

Chapter 5 focuses on the practical implementation of the system. It explains how the system is initialized, how predictions are generated, and how the system handles data imports. Additionally, various use cases are detailed, illustrating how both users and developers interact with the system.

Chapter 6 presents an analysis of the prediction data and compares forecasted results with actual sales data. Metrics are used to assess the forecasting systems' performance, and discussions on error distribution and prediction quality are included.

The thesis's conclusions are summed up in the last chapter, which also addresses Kontrolsat's practical ramifications.

Chapter 2

Theoretical Concepts and Literature Review

A thorough understanding of the underlying theories and relevant literature is essential before and while embarking on the development of an automated inventory forecasting platform.

This chapter establishes the theoretical foundation necessary for constructing such a system, drawing from various fields including inventory management, automation, machine learning, and software engineering.

The aim is to amalgamate these theoretical frameworks to bolster the conception and execution of the forecasting platform, especially for Kontrolsat. This will guarantee that the platform is not only firmly based on established techniques but also customised to address the pragmatic obstacles encountered by the company.

These theoretical insights are integral in shaping the platform's development, guaranteeing that it aligns with industry standards while offering innovative solutions for more efficient inventory management and inventory forecasting.

2.1 Historical Overview of Inventory Management

Inventory management, an essential component of commerce and industry, has evolved dramatically throughout time, reflecting broader changes in trade, technology, and economic theory. This section follows inventory management's historical development, providing insights into its evolution from ancient practices to modern systems.

One popular story regarding inventory management can be traced back to ancient times, with the biblical story of Joseph. In this narrative, Joseph foresees seven years of bountiful harvests followed by seven years of famine. He then advises the pharaoh to stockpile grain during the years of plenty to ensure that there is enough food supply during the famine years. This is a good example of the fundamental essence of inventory management - forecasting demand and managing resources to meet future and current needs (Sheffi 2017).

Ford W. Harris's development of the Economic Order Quantity (EOQ) model was one of the earliest and most significant contributions. The EOQ model formalised inventory management principles by giving a mathematical framework for determining the optimal order quantity that minimises overall inventory expenses (Senthilnathan n.d.).

This was one of the early models that contributed to the first forms of demand prediction, which is essential for forecasting. Thus, making it an important to be analysed properly for

the development of the forecasting system and its implementation in Kontrolsat's business operations.

Nowadays, people go to the supermarket to buy a variety of things for their everyday needs. These items are then stored and used in accordance with their daily requirements. Companies, all across the world, manage inventory goods relevant to their trade on a daily basis (Fernández 2000).

In the 20th century, Enterprise Resource Planning (ERP) systems revolutionized inventory management. They emerged as comprehensive platforms that integrate various aspects of business operations, such as inventory management (Zhao and Tu 2021).

The system takes advantage of information technology and advanced management concepts to enhance decision making processes, as well as planning and operational efficiency processes (Zhao and Tu 2021).

The introduction of ERP systems into the market signified a shift from the traditional inventory management models to a more integrated, modern and technology-focused approach, as seen in Figure 2.1. It provides a holistic view of the supply chain by enabling efficient management of inventory levels and contributing to reducing inventory costs (Zhao and Tu 2021).

Figure 2.1: ERP - Enterprise Resource Planning

The shift towards ERP-based inventory management highlighted the inadequacy of traditional models in the face of rapidly evolving market demands and the information age (Zhao and Tu 2021).

The historical evolution of inventory management reflects a transition from rudimentary methods to sophisticated models and systems.

This sophistication is evident in ERP systems nowadays. While these allow businesses to track inventory, predictive models, like the ones planned to be implemented, take it a step further by forecasting demand based on historical data.

The shift from antiquated techniques to ERP systems and beyond highlights the need of incorporating forecasting into contemporary inventory management procedures, allowing companies to more precisely predict demand. The forecasting model in this thesis is constructed on the basis provided by this evolution.

2.2 Current Trends in Inventory Management

Inventory management is a critical component for business success in the continually changing landscape of e-commerce. The combination of technology and innovative tactics is reshaping how companies manage inventory in order to improve productivity, profitability, and customer pleasure.

In Figure 2.2, we can see multiple methods of retailling, and their evolution over the years. Single channel approaches focus on one mode of customer interaction, either physical or digital. Multi-channel introduces diversity in platforms but lacks integration. While Omnichannel is characterized by its cohesive and integrated customer experience across various platforms.

Omni-channel retailing marks a significant shift in inventory management operations. This approach blends physical store presence with the online operations, which require careful coordination in inventory strategies. Basically, as the line between physical and online retailing grows thinner, this new approach aims to deliver a seamless customer experience regardless of the channel (Marco Melacini and Marchet n.d.).

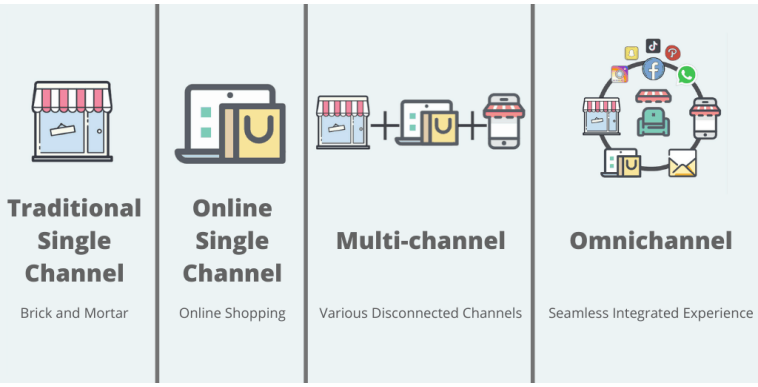

Figure 2.2: Evolution from traditional methods to omni-channel retailing (IntellStyle 2023)

As inventory is now dispersed across multiple channels, forecasting becomes more complex, requiring systems that can predict demand across both online and offline platforms.

With the COVID-19 pandemic, this shift was accelerated in a much faster pace, which compelled retailers to quickly adapt to changing consumer behaviours and market demands (Liu et al. 2024).

These traditional inventory management strategies are quickly becoming a relic of the past, and with that, a new age of technology based solutions are being created and implemented in the industry (Marco Melacini and Marchet n.d.) (Liu et al. 2024) (Namir, Labriji, and Lahmar 2021).

Nowadays, a new, revolutionary transformation has been showing up in the world of inventory management and prediction algorithms. AI and ML have transcended traditional boundaries, bringing unprecedented levels of efficiency and accuracy to these systems (Paula Vidal et al. 2022) (Singh 2023) (Osman, Alinkeel, and Bhavshar n.d.).

The transition is being facilitated by advanced forecasting technologies, technologies that are extremely important in the development of the system that will be used by Kontrolsat, such as AutoRegressive Integrated Moving Average (ARIMA) and eXtreme Gradient Boosting Model (XGBoost) (Namir, Labriji, and Lahmar 2021).

In the context of Kontrolsat's platform, XGBoost for example is particularly important because it offers flexibility in handling different types of demand patterns, helping to predict inventory levels more accurately and with less computational overhead.

These technologies are prepared to take vast amounts of both structured or unstructured data, with multiple different formats to generate a conclusive and intelligent response/output, as exemplified in Figure 2.3. Complementing these technological strides, cloud-based solutions, like Amazon Web Services, Azure or Google Cloud, have also emerged as pivotal tools that streamline the management of inventory data, ensuring real-time visibility across diverse locations.

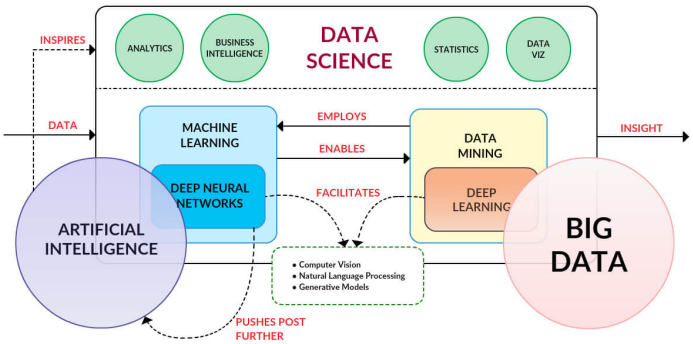

Figure 2.3: Artificial Intelligence and Machine Learning integration in popular Big Data systems

The combination of AI, ML, and cloud technology ushers in a new era in inventory management, marked by enhanced decision-making speed, accuracy, and overall responsiveness to changing market conditions (Singh 2023).

These recent advances not only demonstrate how quickly inventory management is developing, but they also form the basis for Kontrolsat's forecasting platform's conception and execution.

Through the integration of AI/ML models and the utilisation of cloud technologies, the platform seeks to deliver precise and timely forecasts that correspond with the intricate requirements of contemporary retail settings.

2.3 Machine Learning and Predictive Analytics

Inventory management must address stochastic demand and diverse factors like seasonality and product characteristics.

Inventory forecasting has historically placed a great deal of reliance on statistical models like ARIMA, which are effective for linear data with obvious patterns and seasonality. When it comes to handling intricate, non-linear correlations found in demand data, these models are inadequate.

The main benefit of ML for forecasting is its capacity to learn from big datasets and spot linkages and hidden patterns that more conventional models might overlook.

A study regarding the improvement of inventory management within the Industry 4.0 framework was reviewed and analysed to verify if the platform could be easily integrated into Kontrolsat's business operations and to check what were the best methods, strategies and models in this environment.

The study introduces a decision support framework for inventory management, integrating Multicriteria Decision Making (MCDM) and ML. It emphasises the value of ML in inventory management, particularly in dynamic environments and how it may transform data into meaningful insights, optimising inventory strategies and forecasting demand (Paula Vidal et al. 2022).

A major challenge when integrating ML into inventory management lies in choosing between different ML models. Supervised learning approaches, which include models like XGBoost, Gradient Boosting Model (GradientBoosting), and Random Forest Model (RandomForest), are commonly used for forecasting tasks where labeled historical data is available.

The ensemble learning paradigm, which includes techniques like bagging and boosting, has proven particularly effective for inventory forecasting. Incorporating bagging and boosting approaches allows for greater forecasting accuracy through model diversity.

Also, models such as Support Vector Machines (SVM), Artificial Neural Network (ANN), K-Nearest Neighbors (KNN) are particularly effective for modeling nonlinear relationships in irregular demand patterns (Paula Vidal et al. 2022).

However, for new products, predicting demand accurately from the launch is challenging due to the lack of historical data.

According to a study by Tsukasa Demizu, Yusuke Fukazawa and Hiroshi Morita regarding inventory management of new products in retailers, current Reinforcement Learning (RL) approaches in inventory management focus mostly on products with lengthy sales histories, restricting their application to new products (Demizu, Fukazawa, and Morita 2023).

One of the solution that was analysed in the study was the use of a model-based Deep-Reinforcement Learning (DRL) approach for new products, which would use an inventory forecasting model trained on historical data of similar past products (Demizu, Fukazawa, and Morita 2023). A structural diagram of DRL can be seen in Figure 2.4.

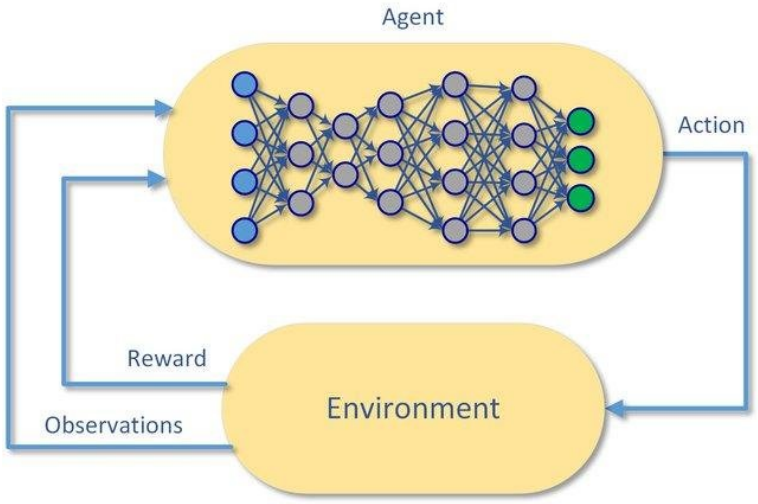

Figure 2.4: Structural diagram of Deep Reinforcement Learning

Choosing the appropriate ML model is vital for balancing computational efficiency with prediction accuracy in inventory management.

The model would have to be trained first with a vast amount of data regarding similar products to the new one, and followed by a testing phase to see how good the inventory forecasting platform is operating with the given dataset (Osman, Alinkeel, and Bhavshar n.d.). For the demand models, it would be beneficial to use something similar to Model-Agnostic Meta-Learning (MAML), and for probabilistic predictions something like Bayesian Neural Network (BNN). The effectiveness of the proposed method was tested using real-world historical sales data of smartphones and outperformed other approaches, showing significant improvements in total reward, inventory turnover, and stock-out rate. These models were effective in capturing demand trends and variations for different products (Demizu, Fukazawa, and Morita 2023).

Based on the finding from the previous studies found, the system will need an efficient Data-based model predictive decision support system (DB-MPDSS), and this model should analyse multiple techniques to determine which would be more effective in the catalog used currently at the company (Fernandez et al. 2021).

AI can improve inventory management in four ways, three of which really important for the platforms implementation:

1. Planning Predictions - AI can help with inventory management models, handling predictions, and operations management (Osman, Alinkeel, and Bhavshar n.d.);
2. Data Mining - AI displays proficiency in data processing and quick action (Osman, Alinkeel, and Bhavshar n.d.);
3. Stocking Management and Fulfillment - AI evaluates customer behavior patterns and assists in inventory planning (Osman, Alinkeel, and Bhavshar n.d.);
4. AI-Based Robotics - Robots, combined with AI, automate internal warehouse activities, however, this is an improvement that won't be explored in this thesis (Osman, Alinkeel, and Bhavshar n.d.).

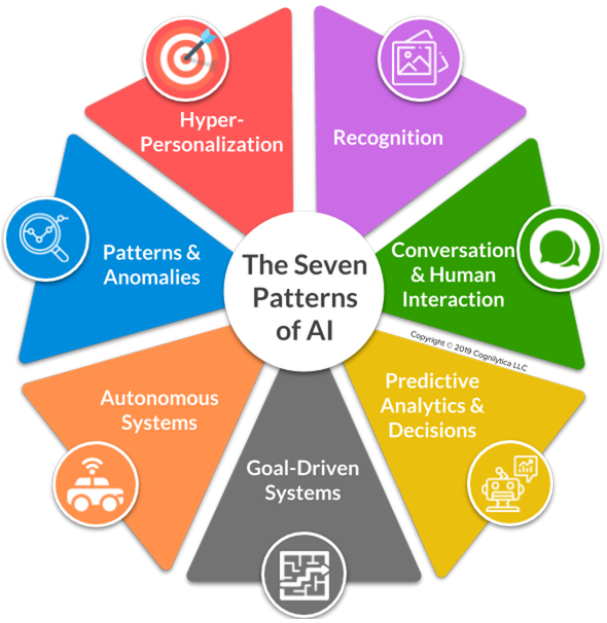

Figure 2.5: Patterns of Artificial Intelligence

Based on the patterns found in figure 2.5, this thesis will analyse the following: Goal-Driven Systems, Recognition and Predictive Analytics and Decisions.

Even though Kontrolsat concentrates on electronics and technology e-commerce, conclusions can be drawn from deployment in other sectors. Inventory management in a hospital pharmacy is one of them.

Managing drug inventory in hospitals is a complex task due to urgent and variable clinical demands. Kontrolsat has a more linear, more predictable consumer demands, however, it is beneficial to look up one of the more variable and complex use cases for this inventory management platform. In both cases, balancing demand satisfaction and cost control is crucial (Fernandez et al. 2021).

The objective of a study on the implementation of a DB-MPDSS at a hospital pharmacy is quite similar to that of this thesis, because the system should use past pharmacy inventory records to enhance the size and timeliness of new orders. The hospital pharmacy handles a range of medicines with fluctuating inventory levels. A discrete-time linear model is used to represent stock evolution, considering order quantities and delivery delays which are also important in the implementaton of the platform in Kontrolsat business operations (Fernandez et al. 2021).

The DB-MPDSS uses historical drug records to inform ordering decisions. The system improves decision accuracy and reduces reliance on pharmacists' experience and intuition (Fernandez et al. 2021).

The DB-MPDSS's performance was assessed over four months and decreased the number of orders and improved stock management without any stockouts, although increasing the average amount of money immobilised due to higher stock levels of expensive drugs. This is something to keep in mind in the implementation of the solution, since electronics can be expensive and average amount of money immobilised should not increase as much, since it can pose a financial challenge for the company (Fernandez et al. 2021).

Another study regarding inventory forecasting in small/medium sized businesses, like Kontrolsat, proposes using the XGBoost regression model for inventory forecasting. Implementing this ML algorithm could enhance the accuracy of demand predictions (Javaregowda et al. 2020). Also, the study proposes that, in the future, the accuracy of inventory forecasting models can be enhanced by incorporating categorical embeddings in neural networks. This is something that should be analysed in Kontrolsat's platform implementation (Javaregowda et al. 2020).

A new study regarding End-to-End Inventory Management Models highlights the complexities faced by e-commerce companies, like for example Amazon, in managing inventories with a diverse range of products and fluctuating demand patterns (Qi et al. n.d.).

The training part of the proposed model in the previously mentioned study leverages historical data, including past demand, vendor lead times, item specifications, and temporal information. This approach enables the model to learn and adapt to various inventory management patterns. In the context of Kontrolsat, the implementation of such an E2E deep learning model could streamline the inventory replenishment process (Qi et al. n.d.).

After looking at many advantages to the use of ML and AI in inventory management, it is also important to understand that the cost of implementing AI, ethical and privacy concerns are significant hurdles in its implementation (Singh 2023). Since this technology processes vast ammounts of data, some of this can contain confidential or sensitive information. In Kontrolsat's case, the users information needs to be protected, even if it is used by the platform to calculate demand predictions (Chen and Biswas 2021).

In the context of the proposed platform, the following ML models are considered for implementation:

1. **Random Forest Regressor**: This model is like a team of decision-makers working together. It builds multiple decision trees and then averages their predictions to come up with a final result. This improves accuracy and reduces the chance of making big mistakes by relying on just one tree.
2. **Gradient Boosting Regressor**: This is a model that builds one decision tree at a time, each one trying to correct the mistakes made by the previous one. This method is like learning from your mistakes and continuously getting better with each new attempt.
3. **XGBoost Regressor**: This is an advanced version of Gradient Boosting. It's known for being very fast and efficient, making it a popular choice in real-world applications. XGBoost not only tries to correct mistakes, like Gradient Boosting Regressor, but also does it more efficiently, using smart techniques to handle large datasets and reduce the time it takes to get accurate results.
4. **Bagging Regressor**: The Bagging Regressor is like running several different versions of a model all at once and then averaging their results. This model helps reduce variability in the predictions, and makes the final output more stable and less prone to errors.

Supervised learning models like these are particularly effective for inventory forecasting tasks, where historical data is available. The models will be trained and tested using historical sales data from Kontrolsat's e-commerce platform.

2.4 Good Practices in Software Engineering

Following good practices of software engineering is necessary for creating scalable and reliable inventory management systems. These practices serve as a roadmap for developing effective, maintainable, and flexible systems that satisfy organisational objectives and permit expansion and change throughout time.

2.4.1 Modularity and Separation of Concerns

In order to provide flexibility and easier maintenance, inventory systems must be developed with modularity in mind.

Specific functions, including data collection, preprocessing, demand forecasting, and reporting, are executed by each module. Because the concerns are separated, modifications to one module (like the forecasting model) won't affect the operation of other modules, guaranteeing the system's continued adaptability.

As a result, this will speed up development and allow for the addition of new features in the future (Hossain, Babar, and Paik 2009).

2.4.2 Scalability and Performance

Large datasets must be handled by inventory systems, which also need to react to variations in supply and demand.

Scalability is therefore an important factor in software engineering. The capacity of the system to sustain performance levels as data volume or user count rises is important.

Performance optimisation strategies including multi-threading, multi-processing, or the utilisation of scalable cloud infrastructure are necessary to achieve this.

The inventory system must be scalable in order to expand with the company and support additional goods, markets, or even business models.

2.4.3 Agile Development

Agile places a strong emphasis on iterative improvement, continuous feedback, and short development cycles, or sprints, all of which are crucial for ensuring that the system is in line with changing business needs.

The system can be constructed gradually thanks to iterative development, which makes sure that essential functions like data integration and demand forecasting are implemented and improved upon before moving on to other features.

This way, it is possible to smoothly incorporate new machine learning models or enhanced demand forecast algorithms into the system's iterative cycles.

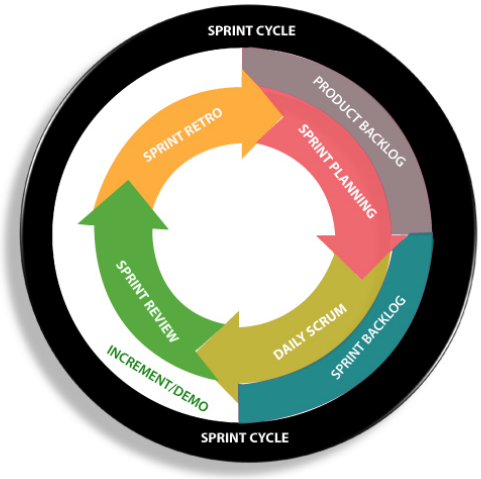

Figure 2.6: Agile Scrum Methodology

Typically, SCRUM Methodology (SCRUM) emphasizes collaboration and transparency among cross-functional teams, however, the development of this platform will be done primarily by one person, so the principles and practices will be adapted to a single-person project (Hossain, Babar, and Paik 2009). The Sprint cycle can be seen in Figure 2.6.

This approach was chosen, with sprints lasting two weeks, to ensure that the system is continuously improved and that new features are added in a timely manner. It was also beneficial to prioritize the most important features and deliver them first. And, of course, for organizing the development process.

2.4.4 Data Flow

Any inventory management system must be able to effectively store and retrieve enormous amounts of data in real-time, which makes effective data management essential. It is imperative to adhere to principles like normalisation, indexing, and transaction management to guarantee redundant-free data storage and prompt retrieval.

Data flow architectures like Extract, Transform, Load (ETL) pipelines will be employed to handle the flow of data from various sources into the system, which will then be used for analysis and machine learning model predictions.

2.5 Research Methodology and Sources

The process of selecting and analyzing studies was guided by a set of specific questions, ensuring that the findings are relevant.This section details the sources used, the criteria for selection, and the overall methodology adopted in the literature review.

The research for this thesis was predominantly sourced from reputable academic and scientific databases. The following are some of the databases used:

- IEEE Xplore;
- ScienceDirect;
- Web of Science;

- JSTOR

These databases were chosen for their authoritative content, wide-ranging coverage of topics, and the presence of peer-reviewed, high-quality research papers.

The selection of studies was driven by specific research questions, intended to guide the focus of the dissertation towards the most pertinent information. Here are some of the essential questions:

- How has inventory management evolved over time, and what are the current trends in this field?
- What role does ML play in modern inventory management practices?
- How are software engineering principles integrated into the development of inventory systems?
- What real-world applications and case studies can provide practical insights into automated inventory forecasting?
- What are the future directions and emerging technologies in inventory prediction?
- What tecnologies are being used for the implementation of these systems?
- What are the ethical and privacy concerns regarding the implementation of these systems?

The studies were also selected based on their relevance to inventory forecasting, recent technological advancements, and practical applications in e-commerce. Priority was given to peer-reviewed research published recently, focusing on machine learning, automation, and real-world case studies in inventory management.

The studies were chosen in part because of their applicability to e-commerce, new technology advancements, and inventory forecasting. Peer-reviewed papers that were recently published and that concentrated on automation, machine learning, and real-world inventory management case studies received priority.

A methodical strategy was used to make sure the review was organised and objective.

The following were the steps taken to conduct such a review:

1. Filtering the studies based on their recency, relevance, and quality;
2. Analyse if the concept are applicable to the development of the platform;
3. Discard any irrelevant or outdated studies, if the information found ends up not being useful;
4. Extracting the most relevant information from the selected studies;
5. Synthesising the findings to draw meaningful conclusions.

Primarily, this method was defined to directly address the research questions previously specified.

This way, it was possible to ensure that the literature review was comprehensive, accurate, and relevant to the development of the automated inventory forecasting platform.

In summary, the research methodology made sure that the literature review was targeted, methodical, and informed by specific research questions. Relevant, high-quality research was obtained using reliable databases and rigorous selection criteria. These insights provided important information on the trends and technologies that shaped the creation of the automated inventory forecasting platform.

2.6 Case Studies and Real-World Applications

The findings from the study "Forecast-Driven Inventory Management for the Fast-Moving Consumer Goods (FMCG) Industry" give useful real-world applications in the context of Kontrolsat's development of an automated platform for inventory forecasting and analysis (Mesfer n.d.).

This study investigates various inventory forecasting models in the FMCG industry using real-world data to develop an inventory control policy for a third-party logistics provider. This approach aligns well with Kontrolsat's focus on automating inventory forecasting and analysis (Mesfer n.d.).

Several models and ML techniques, such as ARIMA, Seasonal AutoRegressive Integrated Moving Average (SARIMA), Random Forests, XGBoost, and Prophet, were employed in the study. These models were evaluated using cross-validation techniques and accuracy measures like Root Mean Square Error (RMSE), Mean Absolute Percentage Error (MAPE), and Mean Absolute Error (MAE) (Mesfer n.d.).

In this study, XGBoost was proven to be the most successful model, demonstrating its capacity to generate accurate FMCG inventory forecasts. This conclusion is especially important for Kontrolsat since it suggests that implementing ML techniques such as XGBoost into the inventory prediction platform could improve forecasting accuracy.

The study's results led to the formulation of a continuous review policy to improve inventory management for the third-party logistics provider.

Looking at another study, Linde Bangladesh Limited's inventory system is designed to properly manage and regulate goods. This includes precise order placing timing as well as rigorous tracking of orders, amounts, and suppliers (Alam and Ahmed n.d.).

Incorporating Linde Bangladesh Limited's real-world inventory management methods into the creation of Kontrolsat's automation platform for inventory forecasting and analysis can also bring useful practical insights. Linde's approach, which successfully integrates inventory management policies and EOQ strategies, can be used to optimise Kontrolsat's platform (Alam and Ahmed n.d.).

Linde Bangladesh Limited's use of inventory management strategies provides significant insights for Kontrolsat. Kontrolsat can dramatically improve operational efficiency, cut costs, and boost customer satisfaction by combining efficient inventory control, EOQ models, and advanced forecasting methodologies into its inventory prediction platform (Alam and Ahmed n.d.).

2.7 Future Directions and Emerging Technologies

A study published by Abdulelah S. Al Mesfer underlines the evolving landscape of inventory prediction and highlights the need for continuous advancement in this field. This suggests

a future where inventory prediction technologies will likely move beyond traditional models, incorporating more sophisticated and possibly hybrid methodologies that blend statistical analysis with advanced ML techniques. ML and AI are the technologies expected to revolutionize inventory prediction technologies, as it as been seen in some recent implementations in multiple business environments (Mesfer n.d.).

Additionally, this study points towards the integration of external factors into inventory prediction models. Furthermore, the idea to apply these approaches to various retail scenarios is noteworthy. It foreshadows a future in which demand prediction technologies are not only advanced in their capabilities, but also versatile and adaptable across multiple industries (Mesfer n.d.).

Another research is dedicated to developing a sophisticated Decision Support System (DSS) for inventory management, integrating advanced forecasting and classification models. The proposed DSS is a testament to the evolving landscape of inventory management, particularly in demand prediction (Cadavid and Zuluaga n.d.).

Figure 2.7: DSS - Decision Support Systems

The DSS will also encompass various comprehensive inventory management models, as seen in Figure 2.7, indicating a shift towards more integrated and holistic systems in inventory management. The precise specification of all important criteria inside the DSS is a vital part of this future direction. This meticulous approach foreshadows a future in which inventory prediction tools are not only sophisticated in their analytical powers, but also intricately tuned to represent the intricacies of real-world events (Cadavid and Zuluaga n.d.).

For the system, these insights highlight the need to design a system that is not just functional with current technologies but is also adaptable and capable of evolving alongside future advancements in inventory prediction technologies.

Chapter 3

Requirements Analysis

As previously stated, a proper analysis of the system needs to be performed, to understand the value that the system will provide.

With that in mind, the following analysis was performed, and outlined in this chapter.

3.1 Value Analysis

Firstly, before even moving on to defining technical requirements for the system, it is important to understand the value that the system will provide.

This value will first be analysed from a business perspective. After the business value analysis is performed, a proposition will be made to guide the development process.

3.1.1 Business Value

The system provides significant business value by enabling companies to make data-driven decisions regarding inventory management, sales forecasting, and overall operational efficiency. This approach reduces operational costs and enhances revenue by allowing businesses to optimize their decision-making processes through precise forecasting and inventory management.

For example, Kontrolsat, an e-commerce company with a large inventory sold both online and in physical stores, can leverage the system to handle their extensive historical sales data. The system would enable them to accurately forecast sales and dynamically manage inventory, which is crucial for businesses dealing with large volumes of sales data over extended periods.

3.1.2 Value Proposition

The system offers a value proposition that can cater to small and medium-sized businesses by providing a cost-effective, intuitive, and easy-to-use platform for sales forecasting. Currently, only larger companies can typically afford such advanced systems. However, by making this system accessible, smaller companies can gain a competitive edge, leveling the playing field. The system helps smaller businesses make decisions based on data rather than intuition, offering a significant advantage in today's competitive market.

In addition, the system's intuitive design makes it particularly appealing to businesses looking for a solution that can seamlessly integrate into their existing processes without the need for extensive technical expertise or large financial investments. By improving operational

efficiency through accurate sales forecasting and inventory management, businesses can achieve better results with fewer resources.

3.2 Functional Requirements

Now that the system's value has been established, it is possible to properly define the functional requirements that the system must meet. These requirements aim to provide the necessary functionality to deliver the system's value proposition.

For the purposes of this analysis, the use cases specified will be considered as functional requirements. This is because the use cases describe the system's behaviour and how it interacts with its actors.

These describe what the system should do, how it should behave, and what it should accomplish. As well as presenting acceptance criteria for each one, so that it is clear when a requirement is met.

In the beggining, only some of the use cases were specified, mainly focusing on the essential features of the system (forecasting). But, as the system grew, more use cases were added to improve the quality of life for developers and users alike.

These enhancements aimed to streamline system interaction, improve the user and developer experience, and ensure that the system remained scalable.

3.2.1 Use Cases for the User

The following use cases describe the features and functionalities that are essential for users to interact with the system.

Figure 3.1 shows the user use case diagram.

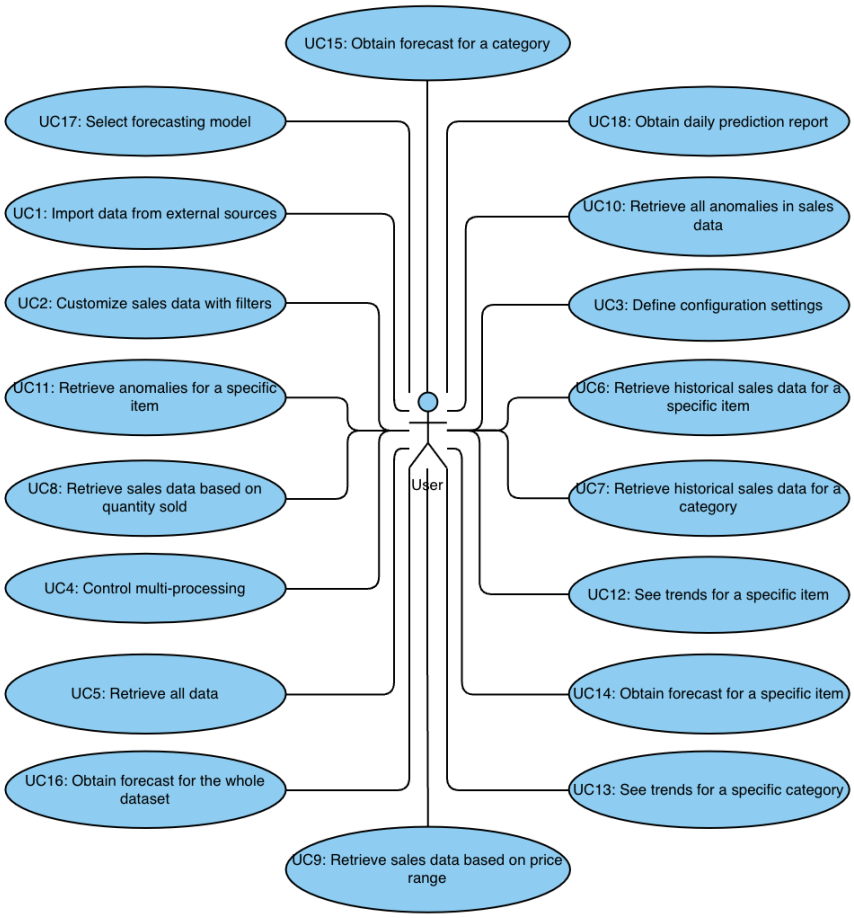

Figure 3.1: User Use Case Diagram

Use Case 1: User can import data from external sources into the system

The user should be able to upload data from databases, CSV or XLSX files and import it into the system.

Acceptance Criteria:

- User can select data source type
- System validates the data and imports it accordingly
- System provides feedback on the import process including timing and success/failure

Use Case 2: User can customize the sales data retrieval based on various filters

Users can apply filters such as product ID, category, price range, date range, and more.

Acceptance Criteria:

- User can select multiple filters at once through query parameters in the endpoint called
- System validates the filters and returns the data accordingly

Use Case 3: User can define its specific configuration settings for the system

User can define settings such as features to be used, models to be used, and more.

Acceptance Criteria:

- Have a configuration file that allows the user to define the settings
- System reads the configuration file and applies the settings accordingly

Use Case 4: User has control over the multi-processing done in the system

User can select how well and how fast the system should perform, how many resources it should use.

Acceptance Criteria:

- Have a variable in the Global Settings file that allows the user to select the performance level
- Variable value defines the number of resources the system uses (e.g number of threads, number of processes)

Use Case 5: User can retrieve all data available in the system

User can request all the data in a structured format that was imported into the system.

Acceptance Criteria:

- User can request all the data through an endpoint (e.g. historical sales data)
- System returns the data in a structured format
- Pagination is applied to the structure data
- Each request has a log entry in the system specifying the request and the response for potential debugging

Use Case 6: User can retrieve historical sales data for a specific item

User can request the historical sales data for a specific item.

Acceptance Criteria:

- User can request the data for a specific item through an endpoint
- System returns the data in a structured format

Use Case 7: User can retrieve historical sales data for a specific category

Similarly to the previous use case, instead of a specific item, user can request the historical sales data for a specific category.

Acceptance Criteria:

- User can request the data for a specific category through an endpoint
- System returns the data in a structured format

Use Case 8: User can retrieve historical sales data based on quantity sold

Similarly to the previous use case, instead of a specific category, user can request the historical sales data based on the quantity sold.

Acceptance Criteria:

- User can request the data based on if the quantity sold is above, below or equal to a certain threshold
- System returns the data in a structured format

Use Case 9: User can retrieve historical sales data based on price range

User can request the historical sales data based on the price range.

Acceptance Criteria:

- User can request the data for a specific item through an endpoint
- System returns the data in a structured format

Use Case 10: User can retrieve all of the anomalies found in the historical sales data

The user can check the anomalies found in the historical sales data.

Acceptance Criteria:

- User requests the information in all historical sales data, and there he can see entries that are marked as anomalies

Use Case 11: User can retrieve the anomalies of a specific item

Similarly to the previous use case, instead of all the data, user can request the anomalies for a specific item.

Acceptance Criteria:

- User requests the information for a specific item sales data, and there he can see entries that are marked as anomalies

Use Case 12: User can see trends for a specific item

When the user requests the data for a specific item, he can see the trends of the sales data.

Acceptance Criteria:

- Trends are properly calculated and displayed in a structured format as a parameter in the response

Use Case 13: User can see trends for a specific category

Similar to the previous use case, but instead of a specific item, the user can see the trends for a specific category.

Acceptance Criteria:

- Trends are properly calculated and displayed in a structured format as a parameter in the response

Use Case 14: User can obtain forecast preditions for a specific item

Users can request forecast predictions for specific items.

Acceptance Criteria:

- User selects a valid item ID and a forecast date range.
- The system provides future sales predictions for that item, including a confidence interval (RMSE)

- The user can select between daily or aggregated predictions.

Use Case 15: User can obtain forecast preditions for a specific category

Similar to the previous use case, but instead of a specific item, the user can request forecast predictions for a specific category.

Acceptance Criteria:

- User selects a valid category ID and a forecast date range.
- The system provides future sales predictions for that category.
- The user can select between daily or aggregated predictions.

Use Case 16: User can obtain forecast predictions for all of the products in the dataset with one click

Users can request sales predictions for all products.

Acceptance Criteria:

- User selects a date range.
- The system provides the future sales predictions for all products available in the system.

Use Case 17: User can select the model used for forecasting

Users can choose which machine learning model is used for sales forecasting in the global settings.

Acceptance Criteria:

- A variable is available in the Global Settings file that allows the user to select which models can be used by the system for forecasting.

Use Case 18: User can obtain a report for the daily preditions

By adding a query parameter named 'daily', the user can obtain a report for the daily predictions.

Acceptance Criteria:

- User can add this extra parameter to the request
- System will skip the aggregation process and provide the daily predictions

3.2.2 Use Cases for the Developer

The following use cases describe the features and functionalities that are essential for developers to work on the system.

Figure 3.2 shows the developer use case diagram.

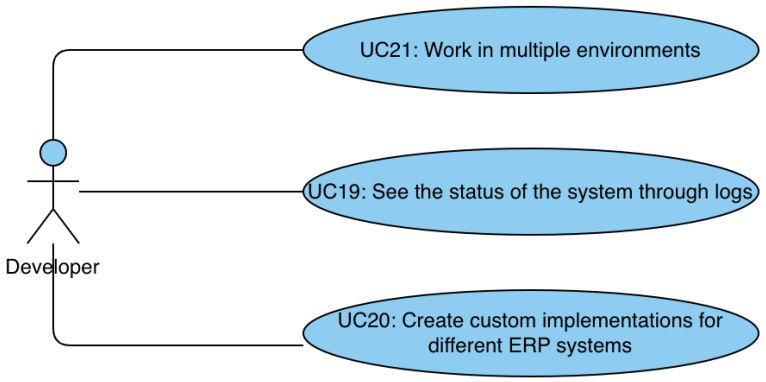

Figure 3.2: Developer Use Case Diagram

Use Case 19: Developer can see the status of the system at all times through the logs

The system provides a robust logging mechanism that records system operations, errors, and events.

Acceptance Criteria:

- The system logs key events such as data processing, prediction generation, and errors.
- Logs include timestamps, severity levels, and relevant information for debugging.
- The developer can filter logs by severity (INFO, DEBUG, WARNING, ERROR, CRITICAL).
- The logs can be exported for further analysis.

Use Case 20: Developer can create custom implementations for different ERP systems

Developers have the ability to extend the system by creating custom service implementations for different ERP systems.

Acceptance Criteria:

- Developer can create new custom implementation for different ERP systems.
- The system can load and use these custom services without needing changes to core components.
- Based on the user configuration, the system can switch between different custom implementations.

Use Case 21: Developer has multiple environments to test the system

The developer can work in multiple environments such as CLEAN_TEST, TEST, and PROD.

Acceptance Criteria:

- The developer can configure different environments. These can be selected in a .env file or through environment variables.

- Each environment has its own set of configurations in the system.
- The system can switch between environments without manual code changes.

3.3 Non-Functional Requirements

Non-functional requirements are constraints that the system must adhere to. These are not directly related to the system's functionality, but rather to its performance, security, and other quality attributes.

And in the context of the proposed system, some of these areas are extremely important, especially in terms of performance and scalability.

3.3.1 Usability

The following non-functional requirements are related to the system's usability and user experience.

They are essential to ensure that the system is user-friendly and intuitive. Not only for the user, but also for developers and maintainers.

These are the non-functional requirements related to usability:

Non-functional Requirement (NFR) 1: System processes imported data for machine learning models

The system should be ready to process the imported data for machine learning models.

Acceptance Criteria:

- Data is imported, cleaned, and prepared for model training.
- System performs data validation checks and rejects invalid data.

NFR 2: System caches previously imported data

The caching mechanism should store imported data and reuse it when needed.

Acceptance Criteria:

- Data is cached and reused without re-importing.
- Cache is invalidated when source data changes (based on the recorded hash values).

NFR 3: System caches trained machine learning models

Similar to the previous use case, the system should cache trained machine learning models.

Acceptance Criteria:

- Models are trained on new data and cached.
- Cached models are reused if no data changes are detected.

NFR 4: System implements a proper logging mechanism

The system should have a robust and intuitive logging mechanism.

Acceptance Criteria:

- Logs key events and errors.
- Logs are categorized by severity and stored in a file.

3.3.2 Maintainability and Scalability

Having already established the system's usability, the following non-functional requirements are related to maintainability and scalability.

These are essential to ensure that the system is easy to maintain and extend over time. If the system is not maintainable, it will become obsolete and difficult to scale.

With that in mind, the following non-functional requirements were identified:

NFR 5: System follows Domain-Driven Design (DDD)

The system should follow the Domain-Driven Design principles to separate concerns and improve maintainability.

Acceptance Criteria:

- Clear separation of domains (app, domain, infrastructure, ml).

NFR 6: System has a clear and organized file structure

The system should have a clear and organized file structure to improve readability and maintainability.

Acceptance Criteria:

- Files are logically grouped and named.
- Structure is easy to navigate and extend.

3.3.3 Performance

Performance is a crucial aspect of the system, especially when dealing with large datasets and complex machine learning models.

Without a good performance, the system will not be able to process data in a reasonable time frame.

After looking into industry standards and best practices, the following non-functional requirement was identified:

NFR 7: System uses multi-processing to improve performance

The system should use multiple Central Processing Unit (CPU) cores to process data and improve performance and speed.

Acceptance Criteria:

- System uses multiple CPU cores for data processing.
- Tasks are distributed across threads or processes to optimize speed.

3.4 Domain-Driven Design Architecture

To meet all of the proposed requirements, a system's architecture must be designed to be modular, scalable, and maintainable.

For that, the architectural approach that should be utilized in this system is DDD.

DDD places a strong emphasis on organising the system around domain models and fundamental business logic, which closely mirrors the business processes and rules found in the real world.

The following principles illustrate how DDD should be applied in this system:

- **Separation of Concerns**:

  The system should be divided into distinct modules where each is responsible for a specific part of the application. The business logic remains at the core of each component's development thanks to its modularity.

- **Domain Model Centrality**:

  The Domain should capture the core business logic of the system.

- **Layered Architecture**:

  The system must have a layered architecture, where the Domain Module is at the core, interacting with other modules in the system. The system is easier to maintain because of this layering, which makes it possible to distinguish clearly between technical concerns and business rules.

This design allows the system to remain flexible and scalable as it evolves over time.

Chapter 4

System Architecture and Design

With the requirements already defined, the next step is to properly outline how the system should be built. That will be the focus of this chapter.

This chapter is divided into three main sections: Analysis of the System, Design Decisions and Challenges.

The first section provides an in-depth look at the system's architecture, emphasising how each part works together in a modular framework to accomplish the application's main objectives. The second section delves into the design decisions that were made during the development process. The third section outlines the challenges that were encountered during the development process.

It is organised this way to provide a clear understanding of the system, followed by an explanation of the design choices that were made to achieve the system's goals.

4.1 Architecture and Design Analysis

This section delves into the architectural and design principles that form the backbone of the system.

After a careful analysis of the system's requirements, the architecture was properly designed to meet the system's goals.

The system's architecture was designed to be modular, scalable, and maintainable, with a strong emphasis on DDD.

The system's architecture is divided into four main modules:

- **App Module**: Controls how API endpoints are exposed and configured, allowing outside users to communicate with the system.;
- **Domain Module**: Encapsulates the business logic. Ensures that the application accurately reflects the underlying business processes;
- **Infrastructure Module**: Provides the necessary tools for parallel processing, data storage, and retrieval, ensuring that the system can handle large amounts of data efficiently;
- **ML Module**: Focuses on the machine learning aspects, from model training to prediction generation.

The API architecture was designed to manage forecasts, sales-related operations, and data processing in an efficient and scalable way.

The modular architecture that underpins the system's API prioritises maintainability, scalability, and separation of concerns. During development, one of the main goals was to make the system fully modular so that it could be customised for various customers and businesses. The API's modules are each in charge of a certain set of responsibilities, which keeps the system's components organised.

The API's architecture is designed following the principles of Domain-Driven Design. This approach ensures that the system is modular, maintainable, and closely reflects the underlying business logic.

The app, domain, infrastructure, and ml modules are the core elements of this architecture, each of which has a particular role inside the system.

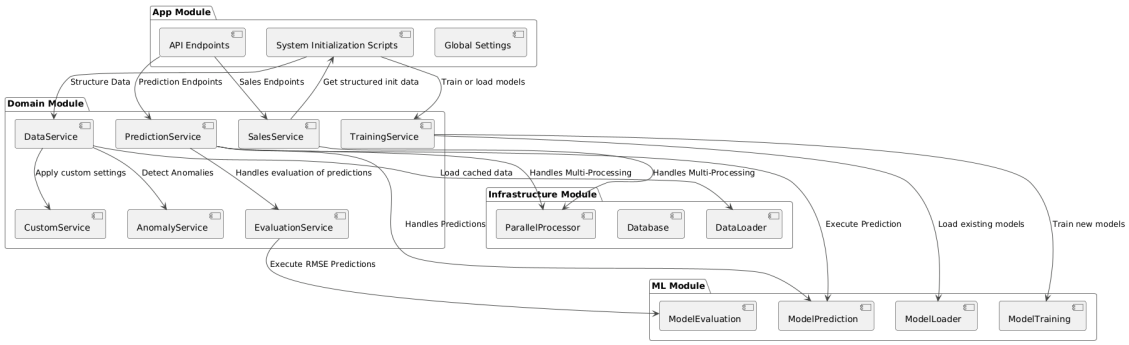

Figure 4.1: Component Diagram of the API Architecture

4.1.1 App Module

This module is responsible for configuring the application, setting up dependency injection, and managing the API endpoints that interact with the outside world.

1. API Endpoints

The API endpoints are the primary interface for users to interact with the system. It is in the app module that these endpoints are defined and configured.

They are designed to be user-friendly and intuitive. A focus on URL simplicity and consistency ensures that users can quickly grasp how to interact with the system.

2. System Initialization Scripts

The System Initialization Scripts are integral to the setup and configuration of the system during its startup phase.

This process involves three primary tasks:

1. Loading of the historical sales, product and category data;
2. Preparation of the imported data;
3. Training of the forecasting models based on the data imported into the system.

These scripts ensure that the data is up-to-date and correctly formatted, allowing the system to operate efficiently. The initialization scripts also handle the detection of changes in the data, applying necessary updates only when required.

3. Global Settings

The Global Settings are crucial for the proper setup of the imported data and configuration of runtime settings for the program. There is a wide range of parameters that influence the application's behavior.

These settings are defined using a configuration class, which centralizes the key constants, paths, and model parameters required throughout the system. By loading these configurations from specified defaults and environment variables, the system guarantees adaptability and flexibility in various circumstances.

Key settings include:

- **Application Environment**: Specifies the environment in which the application is running. The three available environments are CLEANTEST (Will always reload the data, and perform the TEST logging for debugging), TEST (Adds specific logging for debugging purposes), and PROD (Release version with clean and straight forward logging settings), each with specific configurations and logging patterns.
- **Processing Multiplier**: Determines the percentage of available CPU resources allocated to processing tasks, allowing the user to scale how quickly the system processes data (e.g., 0.5 for 50% of CPU resources).
- **Custom Implementation Path**: Defines the path to the custom service implementation, which allows the system to apply specific business rules, such as the Sage 50cc accounting software integratio used in Kontrolsat.
- **Constant Definitions**: Several constants, such as product ID, category ID, and sales-related fields need to be defined here. When users first start using the system, a prompt will ask them for the names of these fields, so that the system can properly process the data. In the future, these prompt will be removed, and the system will be able to automatically detect these fields in the imported data.
- **Anomaly Detection Features**: Specifies the features used for anomaly detection within the sales data. This includes encoded identifiers and specific sales-related metrics. At the moment, these are manually configured by the user.
- **Model Features and Selection**: Defines the features used in model training, as well as the machine learning models selected for use in the system. Only the models that are selected here will be used to perform the predictions.

4. Custom Exceptions and Error Handling

Error handling ensures that exceptions are managed gracefully and that users receive meaningful feedback when issues arise. Since this proved to be a complex application, custom exceptions were created to handle specific scenarios and aid in debugging.

The system utilizes both custom exceptions and default FastAPI exceptions to handle various error scenarios.

Here are some of the exceptions used in the system:

- **NoDailyForAllItemsException**: This exception is raised when the users tries to get the daily forecasts for all products that were imported in the system. This proved to be a very heavy operation, and ultimately, it was decided to only allow users to see the daily forecasts for individual products. This was done to provide the user with a better experience, and as such is considered as a quality of life feature in the system and this exception is raised to inform the user that this feature is not available.
- **NotFoundException**: This exception is raised when the user tries to get information regarding a specific product Id or category Id that does not exist in the system. It provides the user with a clear message that informs the user that the provided input is not in the system. In the future, the system will add a feature for type checking that provides the user with a list of available product Ids or category Ids that are the closest to the input provided.
- **HTTPException**: This is the most commonly used exception in FastAPI and is used to return specific HTTP status codes and messages for a wide range of errors, such as bad requests (400) or internal server errors (500).
- **RequestValidationError**: This exception is automatically raised by FastAPI when the input data validation fails. It makes use of the validators defined to confirm that the input data is correct. If the data is not correct, this exception is raised and the user is informed of the error and how to correct the request body or query parameters to get the desired output.
- **RuntimeError**: This exception is handled through a custom runtime handler, which captures unexpected runtime errors and returns a standardized response, helping maintain the system's stability and providing useful debugging information.

5. Logging Configuration

The logging configuration in this system is designed not just as a tool for development, but as a crucial component for maintaining visibility and understanding across the entire lifecycle of the application.

What I mean by this is that during development, the logging system is configured to provide insight into the system's operations. By setting the log level to 'DEBUG', every detail of the application's behavior is captured, allowing potential future developers to trace the execution flow and identify issues easily.

However, the true power of this logging system, and the core reason why it was implemented was for when the application is deployed into production.

Potentially, in the future, the system will be used by many users, and maintaining high visibility into its operations becomes essential. The main idea was for in scenarios where users have problems/errors with the system, and try to contact the developers. With this logging system, it is possible to look for the specific user id, filter the logs to see only the ones with 'ERROR' level, and quickly identify the problem that the user had and quickly provide a timestamp for a fix.

The logging setup is carefully structured to balance detail with clarity, using different log levels:

- **INFO**: For general information about the system's operations, such as when a historical sales imports or when a prediction is generated.

- **DEBUG**: For detailed information about the system's behavior, such as when a specific function is called or a variable is assigned.
- **WARNING**: For potential issues that do not impact the system's operation but may require attention, such as computational resources nearing capacity (which can happen frequently when Processing Multiplier in Global Settings is set to a high value).
- **ERROR**: For errors that impact the user's operation, such as when a user enters an invalid product ID or category ID, or when an exception occurs.
- **CRITICAL**: For errors that cause the system to crash or become unusable. These are more severe than 'ERROR' logs and require immediate attention. Most of these logs are raised based on the infrastructure where the system is running. However, if the system crashes and can't restart itself, a CRITICAL log will be raised.

For CRITICAL errors, eventually, platforms such as Opsgenie will be used to notify developers that a CRITICAL error has occurred. This will be done so that the developers can quickly jump on the problem and fix it.

The format of the logs was also designed with this use case in mind.

Each log entry begins with a timestamp, followed by the severity level, and the logger's name, which indicates the part of the system where the log was generated. The log message then provides a detailed description of the event. This structured approach ensures that logs are not only readable but also easily searchable, by leveraging platforms such as Elasticsearch or Kibana.

Moreover, the system's logging configuration includes both console output and file-based logging with rotation.

For each user that users the system, a specific log file will be created. This log file will contain all the logs that are related to the user. This is done so that the developers can quickly identify the problem that a particular user.

In summary, the logging configuration is a cornerstone of the system's architecture, designed to provide maximum visibility and traceability.

6. Pagination and Filtering Configuration

The pagination configuration was built to ensure smooth data retrieval and an enhanced user experience, even when dealing with extensive datasets.

At the heart of this configuration is the `PaginationUtils` class, which divides large datasets into manageable pages.

The main problem that this feature solves are the absurd response times that the system would have if it tried to return all the data at once. This considerably reduces the time it takes to retrieve data, as only a subset of the data is processed at a time.

In each response, the user will be able to check which page he is currently on, how many pages there are in total, and how many records are in each page.

Note that, by adding a specific query parameter $page_size$ in the request, the user can specify how many records he wants to see in each page.

But the power of this system doesn't stop at simply breaking data into pages.

A significant part of the development process was allocated to designing a filtering system that provides the most flexibility and customization possible to the user, without making the use of the system too complex.

It goes a step further by offering highly customizable filters, allowing users to fine-tune the data they retrieve.

Here's how you can make the most of these filters:

- **Filter by Item ID**
- **Filter by Category ID**
- **Filter by Price Range**
- **Filter by Sales Quantity**
- **Filter by Date Range**

The combination of pagination and these advanced filtering options ensures that users can extract precisely the data they need or want, no matter the complexity of the dataset.

7. API Security Configuration

Ensuring the security of an API is paramount, particularly as the system evolves from development into production.

Key security measures include:

- **Cross-Origin Resource Sharing Configuration**: The system is configured to manage Cross-Origin Resource Sharing (CORS) policies, which control how resources on the server are accessed by external domains.
- **Basic Security Headers**: The API is designed to include security headers in its responses, such as 'Content-Security-Policy', 'X-Content-Type-Options', and 'Strict-Transport-Security'. These headers help protect against common vulnerabilities like Cross-Site Scripting (XSS) or even man-in-the-middle attacks.
- **Input Validation and Sanitization**: The system ensures that all input data is validated and sanitized before processing.
- **Prepared for JSON Web Token (JWT) Authentication**: The system architecture is designed to support JWT for user authentication and authorization. However, during the active development phase, JWT is still to be implemented to avoid introducing additional complexity in the debugging process.

These security measures lay the groundwork for a secure API as the system moves towards production. While the primary focus during development is on functionality, these security features ensure that the system is ready to handle the demands of a production-ready environment.

8. Mappers Configuration

Mappers are essential to the design of the API in the context of Domain-Driven Design (DDD) because they make sure that data is correctly transformed and transferred between the application's layers. Mappers are particularly useful in scenarios where data structures differ significantly.

They provide a clear mechanism for handling these differences, making the system more robust to evolve as the domain model or external requirements change.

4.1.2 Domain Module

This module is the core of the system's business logic.

1. Interfaces

Interfaces play a critical role in the architecture of the system by defining the contracts that various components must adhere to.

In this system, interfaces are used extensively to decouple the implementation of services from their usage within different parts of the application.

This way, the system enforces a separation of concerns that allows each part of the application to evolve independently.

Here are some of the reasons why interfaces were implemented in this API:

- Decoupling Implementation from Usage:

  For example, an interface for a sales service (‘ISalesService‘) allows the rest of the system to interact with sales-related operations without needing to know how those operations are implemented. This means that if the implementation changes, the rest of the system does not need to be updated, as long as the new implementation adheres to the same interface.

- Flexibility and Extensibility:

  Interfaces provide a flexible foundation for extending the system's capabilities. Because interfaces define what a service should do, without dictating how it should be done.

- Clear Contract Definition:

  Interfaces provide a clear contract that specifies what methods a service must implement. This clarity ensures that all implementations of a service adhere to the same expectations.

- Organizational Clarity:

  While the folders within the interfaces section of the domain module are used primarily for organizational purposes, grouping interfaces by their domain of use (e.g., app, domain, infrastructure, ml) helps keep the codebase clean and navigable.

2. Services

The Services within the Domain Module are where the core business logic is implemented.

These services interact with the repositories, data loader, and other parts of the system to execute the specific tasks required by the system.

Here's an overview of the key services and how they fit into the architecture:

- **Anomaly Service** (‘anomaly_service.py‘): The Anomaly Service is tasked with detecting and managing anomalies in historical sales data. The accuracy and dependability of the data that the predictive models rely on are guaranteed by this service and other structure preparation services. The service operates as part of the data preparation

pipeline, and makes sure that the information supplied to the prediction models is accurate and consistent.

- **Custom Service** (‘custom_service.py‘, ‘sage_impl.py‘): The Custom Service handles software-specific business logic. The "SageImpl" class is an example of a custom implementation tailored to a particular client (in this case, using Sage 50cc software used by Kontrolsat). This service applies custom rules to the data, such as filtering transaction documents based on specific criteria. This layer of customization allows the system to different clients.
- **Data Service** (‘data_service.py‘): The Data Service is responsible for the loading and preparing the input data in the system. This service checks cached information in the system, and provides a clear flow between the necessary steps to prepare the data for the machine learning models. The true implementation for all of these stages is in the Sales Preparation Service, which is called by the Data Service. This service can be thought of as the orchestrator of the data preparation process, ensuring that the data is correctly loaded, filtered, and transformed for use in the system, both at the first and last preparation stage of the data.
- **Prediction Service** (‘prediction_service.py‘): This one is responsible for implementing the logic necessary for generating sales forecasts by using the trained machine learning models. Besides handling the predictions, it also incorporates trends and evaluates the model accuracy through metrics like RMSE. It plays an important role in ensuring that predictions are accurate. Like previously stated, the service integrates with the ‘TrendsService‘ to generate trend reports that enhance the predictive accuracy by accounting for underlying historical sales patterns.
- **Sales Preparation Service** (‘sales_preparation_service.py‘): The Data Service is responsible for the logic regarding preparation and transformation of the data in the system. It contains three main stages: filling missing dates, filling average sales, and filling trends information. The business logic for each of these stages is implemented in this service.
- **Training Service** (‘training_service.py‘): This service is responsible for managing the life cycle of machine learning models within the system, specifically focusing on the training and loading of these models. This service ensures that the models are trained on the latest available data and stored for future use. When data changes are detected, the service initiates the training process for each selected model, utilizing multiprocessing to optimize performance. The trained models are then saved, tagged with a unique hash based on the data they were trained on. If no data changes are detected, the service loads pre-existing models from the cache.
- **Evaluation Service** (‘evaluation_service.py‘): This service will handle all necessary methods for evaluating the trained models. It handles separation of the data into chunks for better processing using multi-processing.
- **RMSE Service** (‘rmse_service.py‘): This one specializes in computing the RMSE for different models, helping to determine which model offers the most accurate predictions. It operates by processing chunks of data, executing predictions using the provided models, and then comparing the predicted values against the actual historical sales data. The service uses these comparisons to calculate the error.

- **Trends Service** (‘trends_service.py‘): The Trends Service is a key component within the system’s architecture designed to analyze historical sales data over time and identify trends.

  To ascertain if a product’s sales are increasing upward, downward, or staying stable, it uses statistical modelling to forecast future sales and compares these forecasts with existing sales data.

  Within the system’s architecture, it provides an additional layer of analysis of the historical sales data. By incorporating trend analysis into the forecasting process, the service allows the system to adjust predictions based on observed sales patterns. It also provides users with insights into the underlying trends driving sales for specific products or categories.

  The service is designed to be modular, enabling it to be easily extended or adapted to different contexts or data requirements. It is implemented with scalability in mind.

3. Repositories

The Repositories within the Domain Module serve as the key interface between the business logic and the data sources.

Repositories provide an abstraction layer over the data storage mechanisms. This needed to be implemented to properly connect with the databases of businesses that use the system, in particular the Kontrolsat businesses that uses a Sage 50cc accounting software database to store their sales data.

Whether the data is stored in a relational database or a Not Only Structured Query Language (NoSQL), the repositories shield the rest of the system from these details.

By centralizing data access logic within repositories, the system ensures that all interactions with data are consistent and follow the same rules. This was a key point in the development of the platform, as different businesses have different ways of storing their data, and the system needs to be able to adapt to these different ways.

These are some of the specific repositories file configured in the system:

- **Data Repository** (‘data_repository.py‘): This one is responsible for fetching the necessary information from the business database, such as historical sales data, product information, and category data. Since the system shouldn’t write any new information to these kinds of databases, thus protecting the integrity of the data, the repository is read-only.
- **Prediction Repository** (‘prediction_repository.py‘): This repository is responsible for managing the storage and retrieval of prediction results. This will be done in a NoSQL database, as the data will be stored in a JavaScript Object Notation (JSON) format and each user will have a specific document in the database. These entries will be fully encrypted, so that even me and, in the future, potential developers, don’t have access to confidential data from the users or companies that use the system.

The design of the repositories in your system adheres to the principles of separation of concerns and single responsibility. By keeping data access logic isolated within repositories, the system remains modular and easier to maintain.

4.1.3 Infrastructure Module

In the Infrastructure Module, two critical components are implemented to manage data handling and processing: the Data Loader and the Parallel Processor.

These components are essential to the system's capacity to handle and analyse massive amounts of data effectively, especially in situations when scalability and performance are critical.

Data Loader

The Data Loader is responsible for the seamless ingestion of data into the system

To optimize performance, the Data Loader uses a hashing mechanism to detect changes in the data.

In the case the user injects files into the system, the system generates a Secure Hash Algorithm 256-bit (SHA-256) for all files. In the case the user uses a database, the system will generate a hash for the data in the database (that is translated into a file in memory).

The system may rapidly ascertain whether the data has changed by comparing this hash with a cached hash from an earlier run. The system can save time and computational resources by avoiding redundant data loading operations if no changes are identified.

In a testing environment, particularly when running in a CLEAN_TEST mode, the Data Loader provides functionality to simulate a first-time run by removing existing hash files.

Again in the file injection scenario, the Data Loader is designed to handle large datasets by leveraging parallel processing. It utilizes the Joblib library's parallel backend to load multiple files simultaneously.

Parallel Processor

Using all of the power available from modern multi-core processors, the system can tackle demanding data processing tasks in parallel thanks to the Parallel Processor. This part makes sure the system can process big amounts of data effectively by dividing up various processing jobs among several CPU cores.

There are some of the processes that leverage the use of multi-processing in the system, as seen

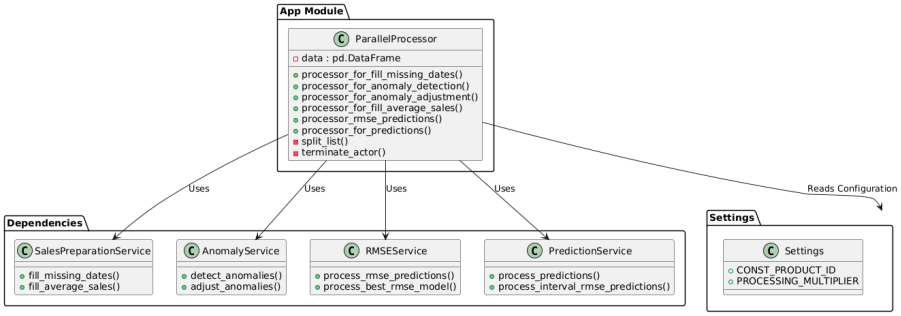

Figure 4.2: Component diagram for Parallel Processor (Multi-Processing Approach).

- **1. Filling Missing Dates**:

One of the primary tasks handled by the 'ParallelProcessor' is filling in missing dates in the data.

First, the data is divided into chunks according to the number of CPU cores that are accessible. A subset of distinct items is represented by each chunk, which are subsequently divided across several parallel workers.

The method utilizes a pool of 'sale_preparation_services', which are distributed among the chunks. Each service processes its assigned chunk to fill in any missing dates.

Once all parallel tasks are completed, the results are combined into a single DataFrame, ensuring that the entire dataset is now complete with no missing dates.

- **2. Anomaly Detection and Adjustment**:

  The 'ParallelProcessor' is also responsible for detecting and adjusting anomalies in the dataset.

  After assigning distinct identifiers to each group, the data is divided into smaller groups. Based on the number of cores the system is running on, a number of instances for the 'anomaly_service' is created. Each chunk is then processed in each instance of 'anomaly_service', and, in it, the anomalies will be detected and marked in the corresponding data subset.

  After anomalies are detected, they are adjusted using a similar parallel process. This involves modifying the dataset to correct outliers, ensuring that the data used for further analysis is reliable.

- **3. Calculating Average Sales**:

  The parallel services work on their assigned chunks simultaneously, and inside of each, a loop through the data will be done and the average sales will be calculated for every record found of a specific product.

  This will provide the models with an indepth look at how the average sales are developing over time, and thus provide a good baseline for the forecasting models.

  At the end, much like in the missing dates process, all the chunks are combined into a single DataFrame, which will be used to train the models.

- **4. RMSE Predictions**:

  This is one of the most computationally expensive processes in the system, as it involves training the models and predicting the sales for each product for each specific timestamp recorded in the provided data.

  For example, if there is data for 100 products and 365 days, and the user wants to predict the sales for the next 30 days, the system use the models generated in the initialization stage and predict the sales for each of the 100 products for each of the 30 days and compare the predicted values with the actual values.

  Then, at the end of this process the system will calculate the RMSE for each of the models and select the one that has the lowest RMSE to provide the user with the best predictions for each specific product.

With this method of evaluation, the system becomes more computationally expensive and this will be the process that takes the longest to finish, since it needs to loop through all the data and calculate the predictions for all the products.

However, it will also provide a very good insight into how the models are performing and if they are providing the best predictions possible.

- **5. Prediction Generation**:

  After all data is processed, and the errors are estimated, the system will generate the predictions for inputs provided by the user.

  This method will define a map between the product and model to be used, based on the errors obtained for each one.

  With this map created, the processor generates predictions in parallel, separating the data into chunks and processing them simultaneously again.

  After all the predictions are generated, they are combined into a single DataFrame and returned to the user.

4.1.4 Machine Learning Module

Model loading, training, prediction, and evaluation are just a few of the phases of the machine learning lifecycle that this module is designed to effectively handle.

The ML module's file structure is organized into distinct directories and files, each serving a specific purpose:

- **model_loader**: Handles loading and saving machine learning models;
- **model_trainer**: Manages the training of different machine learning models;
- **model_predict**: Executes the predictions using trained models.

Model Loader

The ModelLoader class is responsible for loading and saving machine learning models using the joblib library.

There are two methods that are implemented in this class:

- **load_model**: This method loads a pre-trained machine learning model from a specified file path and logs the operation.
- **dump_model**: This method saves a trained machine learning model to a specified file path and logs the operation.

Model Trainer

The ModelTrainer class encapsulates the logic required to train machine learning models. It supports XGBRegressor, RandomForestRegressor, BaggingRegressor, and GradientBoostingRegressor each with pre-configured hyperparameters seen below:

- **XGBRegressor**: 'n_estimators': 100, 'max_depth': 3, 'learning_rate': 0.2, 'objective': 'reg:squarederror'
- **RandomForestRegressor**: 'n_estimators': 100, 'max_depth': 3, 'random_state'=42

- **BaggingRegressor**: 'n_estimators': 100, 'random_state'=42
- **GradientBoostingRegressor**: 'n_estimators': 100, 'max_depth': 3, 'learning_rate': 0.2, 'random_state'=42, 'max_depth': 3

All of these hyperparameters can be changed by the user in the Global Settings of the system, but for now, these are the default ones that are used in the system.

Model Predict

The ModelPredict class is designed to facilitate the prediction phase of the ML pipeline.

It manages feature engineering, and executes predictions using the trained models.

It contains three methods:

- **execute_prediction**: Executes predictions over a specified date range, returning the predicted values alongside relevant metadata.
- **prepare_structured_arrays**: This method prepares the structured arrays required for the prediction process.
- **calculate_prediction**: This method calculates the prediction for each model and returns the results.

The architecture of the ML module is driven by its modularity, reusability, and scalability. By separating the model loading, training, and prediction phases into distinct components, the system is more easily maintained and extended.

4.2 Design Decisions

4.2.1 App Module Decisions and Trade-offs

A number of architectural choices and trade-offs were taken during the App Module's development to strike a balance between maintainability, performance, and flexibility.

Some of the decisions that influenced this module's architecture are listed below:

Feature: Sales Data API routes

One significant decision involved the separation of the Sales and Predictions controllers into distinct classes rather than unifying them under a single controller.

Initially, the system was only going to focus on forecast predictions, but, since the system needed to retrieve sales data as well, the decision was made to create routes that simplify the retrieval of historical sales data and adds some flexibility to the search for information in that data.

Trends and anomalies detection mechanisms are run on the historical sales data, and so, the specific Sales Controller was created to manage these operations that are fundamentally different from those of the Prediction Controller, which focuses on the main goal for this project, which is to generate forecasts.

Feature: Pagination and Filtering

When this API was being developed and tested, the requests took significantly longer to process and be displayed on the user's screen. This happened because the system was trying

to return everything all at once, but the request body would be extremely large, which would take a long time to process.

To solve this problem, the system was designed to return only a small portion of the data at a time, and the user can request more data as needed.

The trade-off here was between simplicity and performance.

When it comes to performance, the system was designed to be as fast as possible, and so, the pagination and filtering system was implemented to build the response of each request in a way that is as fast as possible.

When it comes to simplicity, the system was designed to be as easy to use as possible, and so, the pagination and filtering information was also added to the response, so that the user can easily understand what is being displayed and where the data is coming from.

However, if the system doesn't have a lot of data to process, and based on that building the response is easy, the performance may take a hit, as the pagination and filtering methods will need to be executed, even if they are not needed.

Feature: Flexible Data Source Selection

The ability to allow users to select the data source, choosing between CSV/glsXLSX files or connecting to relational and NoSQL databases was a key feature of the system.

Since the system was built to be as flexible as possible, the decision was made to allow the user to choose the data source that best fits their needs.

Especially since there may be businesses that don't want or can't provide historical sales data directly from their databases, the system was designed to allow the user to upload this data in a file format.

Basically, the decision to implement this feature was based on the need to accommodate diverse user requirements.

Configuration: Dependency Injection (DI)

The system was designed to utilize dependency injection to manage the instantiation and lifecycle of objects.

Implementing dependency injection offers clear benefits in terms of modularity and maintainability.

It allows for better separation of concerns and makes the system more adaptable to change.

However, this method adds more complexity to the design of the system and necessitates careful handling of dependencies and lifecycle events.

While this adds to the initial complexity, it greatly benefits the system' adaptability, particularly as it grows in size and complexity.

Quality of Life: System Initialization

The System Initialization Scripts were designed to handle the complex process of loading, preparing data, and training models.

One trade-off here involved balancing efficiency with flexibility.

The scripts were built to be flexible enough to handle different data sources and configurations, but this flexibility introduced complexity into the system's initialization.

A more streamlined, less flexible initialization process could have been implemented, which would have been easier to develop.

However, this would limit the system's ability to adapt to different environments.

The decision to prioritize flexibility over efficiency was made to ensure that the system could be customized for various users and companies.

After analysing people's needs, it is clear that less inputs and less configurations are better for the user experience, as it makes the system easier to use and understand.

So, to that end, the system was designed to automatically understand and automatically perform the necessary operations, and request the minimum amount of information from the user.

Quality of Life: Custom Exceptions and Error Handling

The system includes both custom exceptions and FastAPI's built-in exceptions to handle a wide range of error scenarios.

One trade-off was between creating highly specific custom exceptions for each possible error versus using more generalized exceptions that could cover multiple scenarios.

Custom exceptions offer better clarity and are easier to debug, as they provide precise information about the error.

However, they increase the complexity of the codebase and require more maintenance.

The decision to use custom exceptions was made to provide more meaningful error messages to developers and users.

Some errors that were implemented contain an error and solution section. The solution section is a brief explanation of what the user can do to solve the problem, offering them a more intuitive experience while using the system.

This decision aligns with the overall goal of making the system as user-friendly as possible.

Quality of Life: Global Settings Management

The configuration of Global Settings was put into place to centralise and streamline the management of system and user parameters.

Centralizing settings makes the system easier to manage and configure but also introduces a single point of failure if misconfigurations occur.

Additionally, it adds a layer of abstraction that might requires documentation.

Despite these potential drawbacks, the centralized approach was chosen to provide a more cohesive and manageable configuration process.

It was implememted with the user in mind. Instead of having a user/company contact the system's administrator to change the features or add a specific custom implementation, among other things, the user can now change these settings themselves, without needing to contact anyone.

The user can now prepare the models as they see fit, and the system will automatically understand and apply the necessary changes.

This was a challenging feature to implement, as it required a lot of work to ensure that the system could understand and apply the necessary changes, but it was worth it, as it greatly improved the user experience.

4.2.2 Domain Module Trade-offs

Just like in the app module, here some decisions had to be made to properly design and properly structure the system.

Feature: Interface-Driven Design

One of the key design decisions in this module was to adopt an interface-driven architecture.

By defining clear interfaces for each service, such as for example IPredictionService, among others, the system gains several advantages:

Since interfaces allow for a clean separation between the definition of what a service does and its implementation, the system becomes more extensible and maintainable.

However, this flexibility comes at the cost of increased complexity. The system needs to manage multiple layers of abstraction and ensure that each service adheres to its interface contract.

This trade-off represent a focus on the overall system build, instead of a more straightforward approach (That sometimes can be more error-prone and better for developers).

Feature: Custom Services for Client-Specific Logic

The decision to implement custom services, such as its Sage 50 Cloud - Accounting Software (Sage 50cc) implementation for Kontrolsat, reflects the need to accommodate client-specific business logic.

The trade-off here involved balancing generalization with specialization. On one hand, a more generalized approach could have been adopted, where the system provides a standard set of services that all clients must conform to.

This would simplify the codebase and reduce the maintenance burden.

However, most people aren't developers. They don't understand how the system works, and if it is hard to use, they'll just stop using it.

So, to that end, the system will add these custom implementations and instead of the user having to provide multiple inputs, the system will fill them in automatically based on the software that the user is using internally at their company for accounting. But of course if they don't want to use these implementations, they can always select none.

Quality of Life: Automated Data Preparation

To enhance the usability of the system, an automated data preparation step was implemented.

The reason why this was done was so each user wouldn't need to update the data structure to fit the system. The system will do it for them. After investigating and getting people' feedback, it was clear that this was a necessary feature to implement, as people tend to be

very busy and don't have time to do these things, and as a consequence, they would just stop using the system.

Of course, this comes with a trade-off. The system will take a bit longer to process the data, and there is a risk that the system might reject valid data that doesn't fit the expected format. This was one of the reason why the robust logging mechanism was implemented as well.

However, with this automation, in a good amount of cases, the system will be able to process the data without any issues, and even fix some of the data that is not in the expected format (e.g. dates in the wrong format).

This decision improves the overall user experience and reduces the need for manual data checks, making the system more user-friendly and reliable.

4.2.3 Infrastructure Module Trade-offs

This module mainly focuses on the parallel processing of data for tasks like anomaly detection and predictions. Several key decisions had to be made during its design and implementation to optimize the performance of the system, which, at first, was a really big challenge for the system.

Feature: Parallel Processing for Data Handling

An important architectural choice was to develop the Parallel Processor class to handle large-scale data processing operations including computing forecasts, detecting and adjusting anomalies, and filling in missing sales values.

Here, the primary trade-off was between complexity and performance.

By allocating work among several CPU cores, parallel processing enables the system to manage enormous datasets effectively.

This leads to much faster processing times, especially for tasks that require a lot of computation, such as calculating and predicting the RMSE.

However, resource management and debugging become much more difficult when parallel processing is used.

It necessitates extremely careful task coordination and may present problems like deadlocks.

But, it was a necessary decision to be made. Without it, the system would have been too slow to be useful for anybody, let alone in a comapny setting.

Feature: Dynamic Resource Allocation With the ability to expand to the hardware it is running on, the system dynamically distributes resources based on the available CPU cores and a customisable Processing Multiplier variable configurable in the Global Settings for the system.

The system's capacity to dynamically assign resources was given top priority in order to increase its adaptability to various deployment environments. Operating on a virtual machine with more limitations or a high-performance server, the system can be tuned to consume the right amount of resources.

This feature was added for two reasons:

1. **Deployment**:

   Let's say the system gets deployed in a Kubernetes cluster. When that happens, per user, the system can be added to a different pod, and the resources can be adjusted accordingly. But the cluster itself can also be scaled up or down, so why would the system need to be able to adjust its own resources? The answer is simple: This variable will allow the system to adjust the resources it uses based on the available resources in the cluster up to a certain limit that is still to be determined. Why go through all this trouble? Well, that bring us to the second reason.

2. **Monetisation**:

   Depending on how the user wants to use the system, they can choose to pay for more resources or save money by using fewer resources. This will be recorded in the system's logs and can be used to generate reports for each specific user. Besides the monthly subscription fee, which will give users a certain threshold of resources, they can choose to pay for more resources as needed. If a historical sales search or prediction needs to be made quickly, the user can quickly increase the resources used by the system to get the results faster, with a corresponding increase in cost.

Of course, in the Kontrolsat case, this feature will not be added, as this project will not be used so much as a Software as a Service (SaaS) but more as a standalone system. Basically, as a one time purchase, that happens to be free.

4.2.4 ML Module Trade-offs

Feature: Cached Model Evaluation

To maximise performance during repeated evaluations, caching techniques were implemented in the model_evaluation.py component to save and reuse model evaluation results, such as RMSE scores.

Basically, if the models were already trained and cached with the data the system currently has, why shouldn't the RMSE values be cached as well?

The computational cost associated with repeating model evaluations is greatly reduced when evaluation results are cached, particularly when large-scale evaluations or iterative development are being conducted. But if the underlying data or models change, there's a chance that cached findings could become outdated if something were not implemented correctly, which could result in erroneous evaluations of the model's performance.

This problem occured during the development phase, where the cached RMSE values were not updated when the underlying data changed, and even with the logging mechanism in place, it was difficult to identify the root cause of the issue.

For future developments of the system, more logging information was added at this stage to help identify the root cause of the problem.

Nevertheless, the decision to implement caching was made to optimize performance of the system to provide the user with a faster experience, even if the future development or update to this feature is a bit more complex.

4.3 Challenges

After the system's architecture was designed, and a proper design was established, some challenges started to arise.

Making the system fully modular and scalable, as well as trying to make as user-friendly as possible, was not an easy task.

Thus, some challenges were faced during the development process.

Below, the key difficulties faced during the development process are outlined:

4.3.1 Data Handling and Integration

One of the primary challenges during development was the seamless integration of various data sources, such as CSV, XLSX files, and relational or NoSQL databases.

Each data format required different processing pipelines and imposed unique requirements on how data was loaded.

CSV and XLSX files had a similar way of loading data, but in the case of databases, the schema and structure were different, requiring custom adapters to handle the data.

The were two ways the system could have been developed. The first would be to only allow a specific data format, such as CSV, which would simplify the data loading process. The second would be to allow multiple data formats, which would require more complex data integration.

Ultimately, the decision was made to allow multiple data formats to provide users with more flexibility in loading their data.

The workflow for this was challenging to implement, as the system needed to automatically detect the type of data source and handle differences in schema, structure, and format.

4.3.2 Data Transformation and Cleaning

Ensuring the data was properly cleaned and transformed for machine learning models was another development challenge.

At first, this analysis was done manually, and so, since the training data was extensive, dating back to 2016, it was complicated to handle some questions, such as:

- How to handle days where a specific product was not sold?
- How can the system identify entries that are outliers?
- How can the system identify that the product is not longer popular?
- How can it handle products that were not sold for a long time?
- How can it handle products that were sold in the past but are no longer sold?
- How can the system distinguish between products that are similar?

These were just some of the questions that needed to be addressed during the data transformation and cleaning process.

In the initialization process of the system, the data is prepared and well structured so that the system can answer these questions for every specific product.

4.3.3 Performance Bottlenecks

A huge portion of the development effort was dedicated to optimizing the system's performance, particularly in data processing and machine learning model training.

At first, the data needed to be limited to be tested. So instead of using the entire dataset, only a portion of it was used. More specifically, only the data since 2022 was used.

After implementing the product-specific RMSE calculation, the system became extremely slow, as it needed to calculate the RMSE for each product and each model.

Ultimately, while this provides a more accurate prediction, it made the system slow and inefficient.

So multi-processing and multi-threading were used to speed up the process.

Implementing this was easier said than done, as it required a complete restructuring of the data processing pipeline and machine learning model training workflows, as well as the core architecture of the system.

This was done sequentially, from the initialization of the system to the predictions generation.

Once everything was implemented, the system was able to process the data and generate predictions much faster.

One downside was that the system was now more complex, and debugging and maintaining it became more challenging. The other downside was that the system was now resource-intensive, and this feature will only be useful in more powerful machines.

4.3.4 Cache Management

As a way to speed up the overall system performance, caching was implemented.

Since the beggining of the project that caching was planned to be implemented, but only after the system was functional.

Once the system was functional, and multi-processing and multi-threading were implemented, was the caching mechanisms implemented.

Since the system had become more complex due to multi-processing and multi-threading, implementing caching was a challenge.

First, this was only going to be implemented in the loading of the data, but then it was implemented in the predictions generation as well since each prediction was taking a long time to generate.

The idea was that if the system had already generated a prediction for a specific product, it would not need to generate it again, thus saving time.

Right now, the system takes a while to generate a prediction, but once that is concluded, that prediction is saved in the cache, and the next time the user wants to check it out, it will respond almost instantly.

4.3.5 Logging Mechanism

Another challenge was implementing a professional level logging mechanism.

Multiple conditions had to be implemented in the system, depending on which environment the system was running on.

The main idea behind this was to have a system that is easy to develop, and easy to maintain. Both in production and development environments.

The logging mechanism was implemented in a way that it would log everything that was happening in the system, from the initialization to the predictions generation.

In every main process, the system also has a timer, and the system logs the amount of time it took to complete that process.

Chapter 5

Implementation

With the requirements, architecture and design already defined, everything is ready to move on to the implementation phase.

This chapter will focus on the system's implementation. It will detail the system's initialization, predictions, user interface, and how to use the system.

5.1 System Initialization

The initialization configuration is split into three distinct parts.

1. Loading of the historical sales, product and category data;
2. Preparation of the imported data;
3. Training of the forecasting models based on the data imported into the system.

All of the business logic for these tasks is contained within the domain module. The app module calls the domain module to perform these tasks, and then stores the prepared results in the memory.

The first of these tasks involves loading the historical sales, product, and category data from the database or from CSV or XLSX files. The user can select which one to use in the settings for the application.

The system checks whether the data has been loaded before, and if it was, gets the data from the cache. If the data has not been loaded before, the system loads the data from the selected source.

This is done by comparing the hash of the current data with the hash of the data stored in the cache. If the hashes are different, the system marks the data for reloading and logs this decision.

This change detection mechanism is crucial for optimizing performance, as it prevents the system from reloading unchanged data.

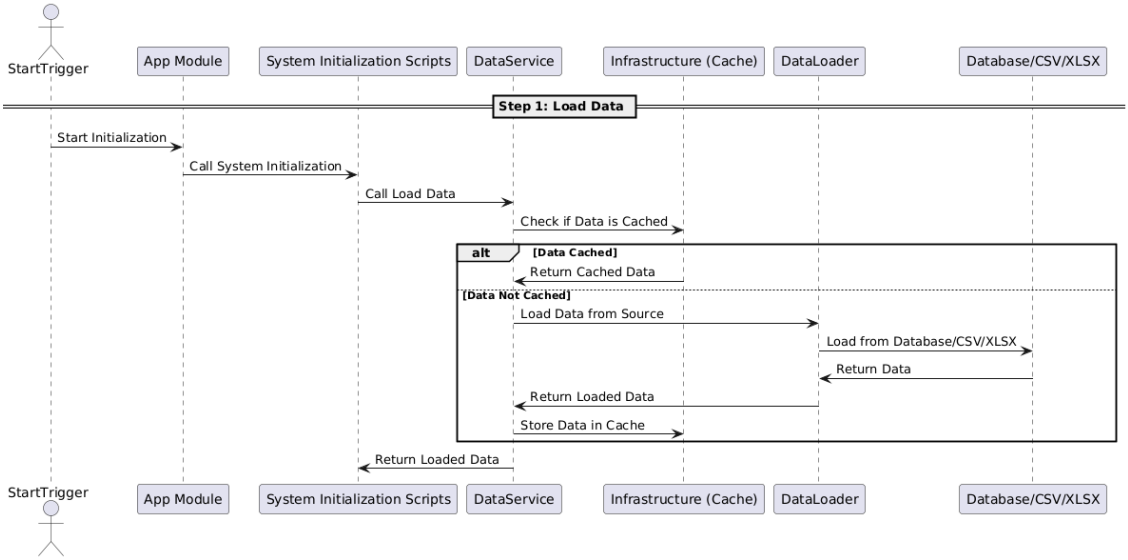

Figure 5.1: Sequence diagram for loading the historical sales data.

After the historical sales data is loaded, the method applies a custom filtering process according to where the data is being loaded from.

This is a critical step as it allows the system to properly understand which data is being provided and how it should be processed.

In the case of Kontrolsat, the historical sales data comes from Sage 50cc, a popular accounting software.

In this program, only entries marked with an Invoice Receipt (FR) and Invoice (FAC) status in the transaction documents are considered as sales. The system filters out all other entries, such as purchases, returns, and other transactions.

This was done in a way that ensures flexibility between other accounting softwares, such as Primavera, SAP, and others. As the system is used by other companies, the more custom filters are added to the system. These custom implementations are added at this stage of the initialization process and their logic is stored in the domain module.

After this process, the system can now move on to the next initialization stage and prepare the structure of the data provided.

In the preparation of the imported data, the system applies a series of transformations to the data.

It begins by standardizing the data types of key identifiers. Specifically, the item and category ids are converted to strings to ensure consistency across the system and, once again, to ensure flexibility between different types of data feeded into the system.

Next, the system reduces the item and category datasets to only the columns that are relevant for further processing. In the case of Kontrolsat, since the system gets the data from the Sage 50cc database, it will take the historical sales data, category and product data from different tables, and the dataframes returned by the loading process will be merged into a single dataframe with the necessary information.

These transformations however are only applied after saving the actual sales records in the system, so that the Sales API gets the actual information present in the accounting software.

To prepare the data for the machine learning models, categorical variables such as product IDs and category names are encoded into numerical formats. This encoding is essential because many of the used machine learning algorithms require numerical input rather than categorical data.

After this, the system goes through a four step process to properly structure the data for the machine learning models.

Filling Missing Sales Values: The system fills in missing dates in the historical sales data. If a product has not been sold on a particular date, the system will create a new entry for that product with a zero value for the sales.

Anomaly Detection: The system applies anomaly detection to the data to identify any outliers. If an outlier is detected, the record is marked for adjustment.

Anomaly Adjustment: The system will adjust all outliers marked for adjustment by replacing its entries's sales quantity with the average of the product's sales up until that records date.

Average Sales: Now that all outliers and missing dates have been adjusted, the system calculates the average sales for each product. This average is used as a baseline for the forecasting models.

To further enhance the dataset, the system adds several time-based features, including the day of the week, month, year, and day of the month for each sales record.

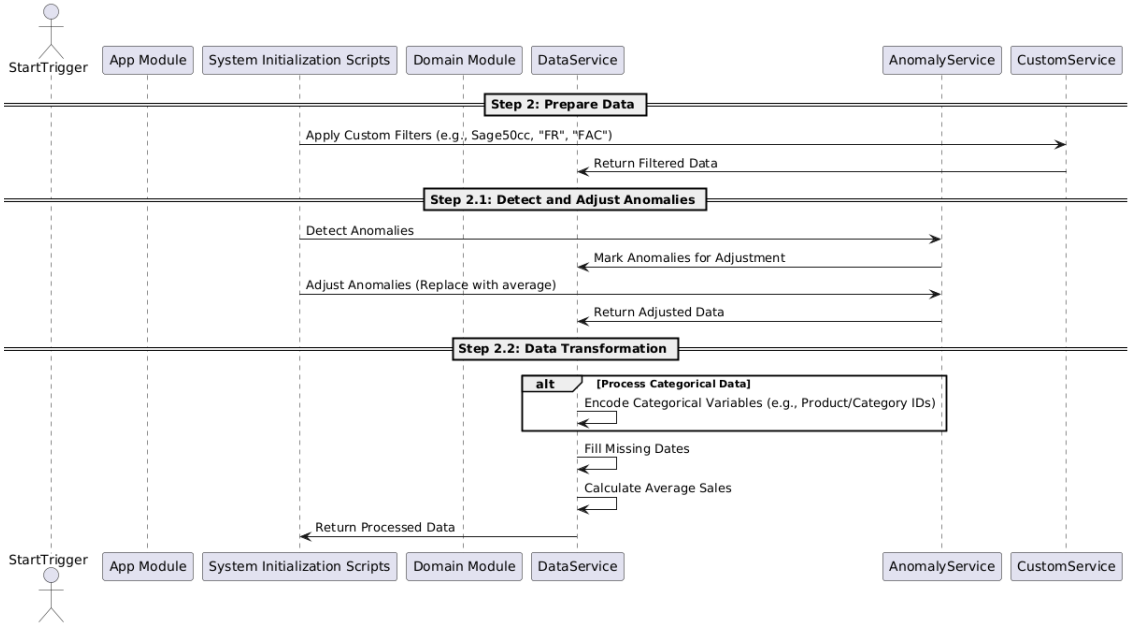

Figure 5.2: Sequence diagram for preparing the imported data.

After all of these transformations are complete, the system feeds both the prepared data and the raw data back to the initialization stage method, which will then call the training process for the forecasting models with the prepared data.

Finally, the final step of the initialization stage is to train the forecasting models based on the prepared data.

Before training the models, the system checks if the data had to be reloaded from the first stage. If it did, then the system will have to train these models with the new imported data. But if it didn't, then the system will look for the model files that are cached in the system and load them into the memory. The file naming pattern is the model name followed by the combination of the hashes of each data source that was imported into the system in the first step of the initialization process.

The system will use multi-threading to train the models in parallel. This is done to speed up the training process, as training these models can be computationally expensive.

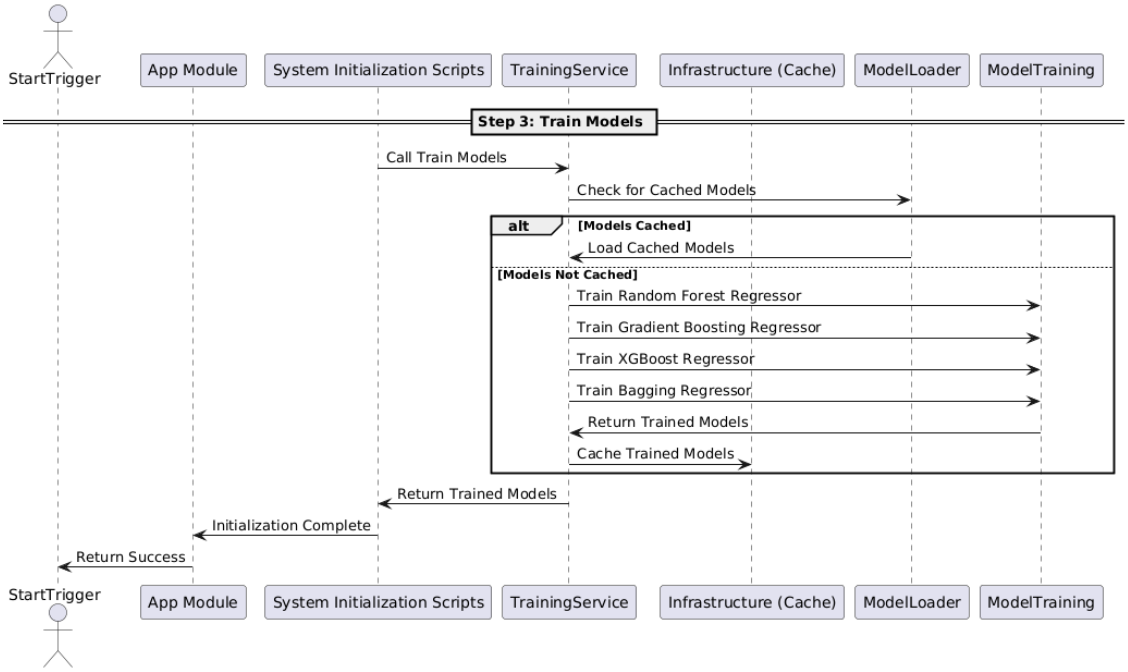

Figure 5.3: Sequence diagram for training the forecasting models.

After the models are trained, the system will save them to the disk, so that it can be loaded into memory next time the system starts up and the data hasn't changed.

With these steps completed, the system is now ready to handle API requests.

5.2 Available API Endpoints

These endpoints are organized within two main controllers for now: the Sales Controller and the Predictions Controller. Each controller is responsible for handling specific types of requests, related to sales data and prediction generation, respectively.

The Sales Controller ('/api/sales') manages endpoints that allow users to query historical sales data. This includes retrieving all sales data, filtering sales by specific items or categories, and applying various criteria such as quantity, price, or date range. The controller is designed to provide flexible access to sales information.

- **GET /api/sales/**: This endpoint retrieves all available historical sales data. Users can apply various filters and pagination will narrow down the results. This endpoint is the most general-purpose, offering access to the entire dataset.
- **GET /api/sales/items/{item_id}**: Retrieves historical sales data for a specific item identified by its unique 'item_id'.
- **GET /api/sales/categories/{family_path}**: Returns historical sales data categorized by category paths. This endpoint is designed for scenarios where users need to view historical sales data grouped by product categories.
- **GET /api/sales/quantities/{comparison}/{quantity}**: Allows users to retrieve historical sales data based on the quantity sold. The 'comparison' parameter can specify conditions like greater than, less than, or equal to a certain quantity.
- **GET /api/sales/prices/{min_price}**: Fetches historical sales data filtered by price, starting from a minimum price and optionally up to a maximum price specified as a query parameter.
- **GET /api/sales/anomalies**: Returns historical sales records identified as anomalies. This endpoint is essential for identifying outliers in the historical sales data, such as unexpected sales volumes at specific dates.
- **GET /api/sales/date/{start_date}**: Retrieves sales data within a specified date range. The 'start_date' is mandatory, while the 'end_date' is optional.

The **Predictions Controller** ('/api/predictions'), on the other hand, is focused on generating and retrieving sales predictions. Users can request predictions for specific items or categories over defined date ranges. The requests in this controller support both daily and aggregate predictions, making it a versatile tool for forecasting future sales.

- **GET /api/predictions/{start_date}/{end_date}**: This endpoint retrieves sales predictions for all items within the specified date range. The user can specify whether they want daily predictions or aggregated predictions by using the 'daily' query parameter, however, a specific 'item_id' as to be provided as well. This endpoint is ideal for users who need to forecast sales across the entire product line over a specific period.
- **GET /api/predictions/items/{item_id}**: Retrieves sales predictions for a specific item, identified by its unique 'item_id', over a defined date range. This is particularly useful for analyzing future sales trends of individual products.
- **GET /api/predictions/categories/{category_id}**: Returns sales predictions grouped by product categories.

Both controllers use FastAPI's routing capabilities to define and expose these endpoints. Exception handling is also integrated into these endpoints, ensuring that any errors during request processing are managed gracefully and that meaningful error messages with solutions are returned to the user.

5.3 System Predictions

The prediction process in the system involves multiple steps.

Below is a detailed breakdown of how predictions are performed, from data preparation to result generation.

5.3.1 Prediction Workflow

Whenever a prediction is requested through the API, the system follows a structured workflow to ensure predictions are generated efficiently. This process can be seen in Figure 5.4.

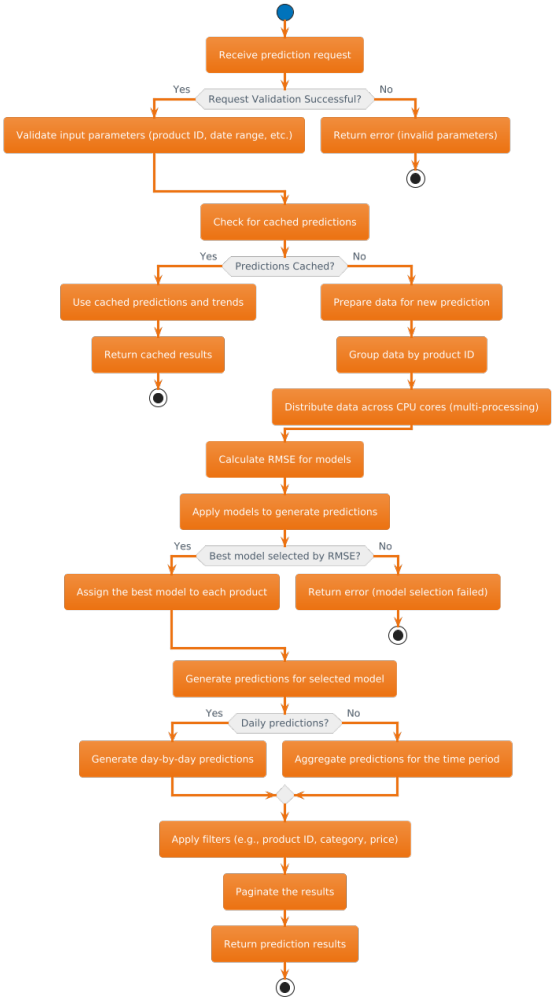

Figure 5.4: Flowchart of the prediction process.

1. **Request Validation and Preparation:**

   Upon receiving a prediction request, the system validates the input parameters, such as the product ID or date range.

2. **Checking for Cached Predictions:**

   Initially, the system determines if predictions for the specified item and time range are already cached. In order to save processing time, the system utilises stored predictions directly if they are available. The system searches the cache for the associated trends as well as the expected sales data.

3. **New Prediction Processing:**

   If no cached predictions are found, or if the data has changed since the last prediction, the system proceeds to generate new predictions. This is done by first preparing the

necessary data for the models. The data is grouped by product ID and processed using multi-processing to improve performance. Chunks of data are distributed across multiple CPU cores for parallel processing.

4. **RMSE Calculation Process**:

   This is the most computational-intensive part of the prediction process. The system evaluates multiple machine learning models to determine the best model for each product.

   The process involves the following:

   - For each chunk, the system applies the models to generate predictions and compares them with the actual historical sales values to compute the RMSE value.
   - Once the RMSE values are calculated for each model, the system selects the model with the lowest RMSE to ensure accurate predictions.

   At the end, each product has a model assigned to it based on the RMSE value. This model will be the one used to generate predictions for that product.

5. **Generating Predictions**:

   After the best-performing model is selected, the system proceeds to generate predictions. The system applies the selected model to the preprocessed data and generates predictions on a daily level.

6. **Aggregating Results**:

   Once predictions are generated for each chunk, the system aggregates the results into a final prediction dataframe. If daily predictions were requested, the system returns the sales forecasts as they are. If aggregate predictions were requested, the system combines the results into a summarized forecast for the entire period.

7. **Returning the Forecast**:

   The final step in the prediction process is formatting and returning the forecast to the user. The system applies any requested filters, such as product ID, category, or price, and paginates the results to ensure they are easily digestible. The predictions are then presented to the user.

   The output of the predictions is seen in Appendix C.

5.3.2 Trend Analysis

The system incorporates trend analysis to evaluate sales data and predict future trends.

This feature is critical for understanding product performance and supporting decision-making, especially based on the quantity predicted to be sold.

The trend analysis process is depicted in Figure 5.5.

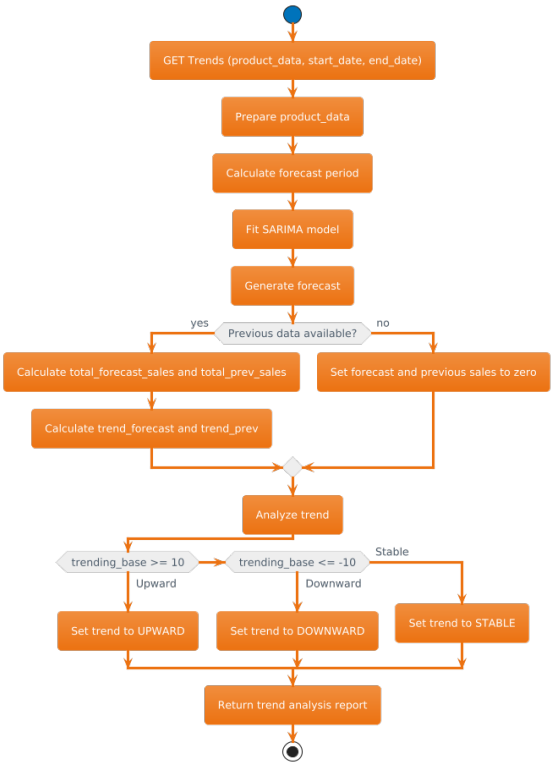

Figure 5.5: Data flow diagram for trend analysis.

Trend Analysis Steps

- **Applying SARIMA Model:** The SARIMA model is used to capture trends and seasonality in the data, making it ideal for forecasting future sales based on past performance.
- **Forecasting and Comparison:** The system predicts future sales for a given period using SARIMA and compares the forecast to the previous equivalent period to determine the trend.

Trend Interpretation

- **Upward Trend:** When forecasted sales exceed past sales by more than 10%.
- **Downward Trend:** When forecasted sales fall below past sales by more than 10%.
- **Stable Trend:** When its in between the above two conditions.

Business Value of Trend Analysis

When a model forecasts future sales, it is crucial for users to trust the predictions, as inaccurate forecasts can lead to significant business consequences.

A business runs the danger of overstocking and suffering financial loss if it orders more inventory than is anticipated. On the other hand, if demand is underestimated, there may be insufficient supply and missed sales opportunities.

Since the forecasted values will never be perfect, and there will always be an error associated with it, trend analysis is used to reduce uncertainty.

Users are better able to weigh the lower and upper bounds of the projection and make more balanced decisions based on which scenario seems more likely when sales are forecasted to surpass or fall short of the expected figure.

Businesses may better navigate the inherent uncertainty in sales forecasting with the help of this feature, which gives them a clearer grasp of prospective outcomes.

5.3.3 Returning the Prediction Results

The predictions are provided either as detailed daily forecasts or as aggregated interval-based results.

- **Daily Predictions**: Detailed forecasts for each day within a specified range. Day-to-Day Sales Analysis.
- **Interval-Based Predictions**: Aggregated sales data for broader timeframes, providing an overall view of sales performance.

For each item, the system provides key details including:

- **Item Information**: Name, category, sales price, and so on...
- **Predicted Quantities**: Forecasted sales for each day or aggregated over a period.
- **Sales Trends**: Indicates if trend is upward, downward, or stable.
- **Model Accuracy (RMSE)**: Upper and lower bounds of the forecasted sales.

In order to handle big datasets, the system also uses pagination. Users may further filter their results by entering parameters like product ID, price range, or quantity sold.

Finally, the system packages the prediction data into a structured format, including metadata, pagination details, and prediction results.

5.4 User Interface

The system lacks a distinct Graphical User Interface (GUI) at this point in its development. Rather, external tools and interfaces are used for all system interactions.

At this development stage, Postman is being used to handle all API requests.

Through Postman, users can easily interact with various endpoints such as retrieving sales data, or getting the predictions.

A Postman collection has been created containing all available API endpoints and their respective request structures.

Also, to assist users in understanding how to interact with the API, Swagger documentation has been integrated into the system.

The Swagger UI provides a clear, interactive way for users to see available API endpoints, the request structure, and the expected responses.

Swagger can be accessed by navigating to '/documentation' in a web browser and is separated into the Sales and Predictions sections.

Finally, all system logs, events, and operations are output to the terminal, giving users real-time visibility into the system's internal processes.

5.5 System Use

The system is designed to be easy to use, with several key tools and configurations available to meet users needs.

Before starting the system, and if you want to import CSV and XLSX files, make sure to follow these steps:

- **Historical Sales Data**: Place the corresponding CSV or XLSX files in the 'data/raw/historical_sales_info' directory.
- **Category Data**: Place the corresponding CSV or XLSX files in the 'data/raw/family_info' directory.
- **Product Data**: Place the corresponding CSV or XLSX files in the 'data/raw/product_info' directory.

If you want to import data from a database, make sure to configure the database connection settings in the '.env' file:

```
DB_HOST=localhost
DB_PORT=5432
DB_USER=diogo
DB_PASSWORD=diogo
DB_NAME=test_db

HISTORICAL_SALES_TABLE=historical_sales_info
PRODUCT_INFO_TABLE=product_info
FAMILY_INFO_TABLE=family_info
```

Before starting the system, make sure to check out the previous sections on installation to ensure that all dependencies are installed and the system is configured correctly.

To start the system, navigate to the project directory in the terminal. Run the following command to initialize the API and other components:

```
python main.py
```

With this command, the system is started, and the initialization process is shown in the logs that are output to the terminal.

When the system is ready for usage and has been successfully initialised, you will receive a notification from it.

You will also be informed by the logs of any failures or configuration problems that arise during launch.

Users can then interact with the API using tools such as Postman, built-in Swagger documentation, or even directly from a web browser.

Chapter 6

Evaluation and Results

Now that the system has been properly structured and implemented, it is time to evaluate its performance and analyze the results.

As previously stated, the system predicts values and gives upper and lower confidence ranges for each prediction using machine learning models. These confidence intervals offer a range of possible outcomes.

Additionally, the system evaluates the trend for each product, categorizing it as "Upward," "Downward," or "Stable."

Based on these trends, and the upper and lower confidence intervals, the user can make informed decisions about inventory management.

The following analysis represents the sales from February 2024 in Kontrolsat.

Predictions were run for 44 products, and key insights from those predictions are included in this evaluation. However, since it is impractical to include all information for the 44 products, only a subset of the data is shown in this analysis. This was done to provide a more concise and focused evaluation.

The following analysis represents the comparison between the predicted values and the actual sales, the error metrics, and the prediction quality.

6.1 Sucess Criteria

Since it is impossible to predict the exact sales for each product, the system aims to provide predictions that fall within an acceptable range.

If the predictions are not within this range, then the prediction quality needs to be registered as Understock to still be considered acceptable.

This is considered acceptable because it is better to have a product out of stock than to have it overstocked, which can lead to financial losses.

For the purposes of this analysis, a 20% error margin is considered acceptable for the predictions, however, users can adjust this value based on their business needs.

A prediction failure is considered when the error margin is above 20%, and the prediction quality is registered as Overstock, since this may lead to an overstocked warehouse and therefore financial losses.

6.2 Prediction Data and Confidence Intervals

An important thing to understand is how the Upper and Lower Confidence Bounds can be used.

Basically, if the trend is "Upward," the user should expect the actual sales to be closer to the upper confidence bound, rather than the lower confidence bound.

On the other hand, if the trend is "Downward," the user should expect the actual sales to be closer to the lower confidence bound.

ItemID	Quantity	Error	Upper C	Lower C	Trend
Formuler z10 SE	869	127	996	742	Downward
Amiko A9 Green	187	11	198	176	Stable
A6	118	24	142	94	Upward
Formuler Z11 Pro Max BT1	96	24	120	72	Upward
Lnb single Amiko	56	5	61	51	Downward
AMIKO MINI HD265	40	4	44	36	Downward
LEAP-S3	75	16	91	59	Downward
ZGemma H8.2H	74	9	83	65	Downward
Amiko F-connector RG6	42	12	54	30	Downward
KHS014K	4	7	11	0	Upward
BHR4037GL	104	53	157	51	Downward

Table 6.1: Predictions with Upper and Lower Confidence Bounds, Errors, and Trends

Table 6.1 shows a small subset of the predictions made by the system, including the RMSE error, quantity, upper and lower confidence intervals, and trend.

The table represents a similar output from the one provided by the system, but the system provides more detailed information in JSON format.

Based on the trend values, the predictions should be adjusted to better represent the actual predicted sales.

ItemID	Adjusted Prediction	Actual Sales in Feb 2024
Formuler z10 SE	742	722
Amiko A9 Green	187	171
A6	142	96
Formuler Z11 Pro Max BT1	120	119
Lnb single Amiko	51	56
AMIKO MINI HD265	36	53
LEAP-S3	59	52
ZGemma H8.2H	65	49
Amiko F-connector RG6	30	80
KHS014K	11	31
BHR4037GL	51	30

Table 6.2: Comparison of Adjusted Predictions and Actual Sales in February 2024

These adjusted predictions represent either the default prediction value, the upper bound prediction, or the lower bound prediction, based on the trend.

After obtaining and adding the values for the actual sales to table 6.2, it is possible to calculate the error metrics to understand how well the system performed.

6.3 Analysis of Prediction Performance

After having the adjusted predictions, it is possible to move forward and understand if the predictions have a good quality and can be trusted.

ItemID	Absolute Error	Error (%)	Prediction Quality
Formuler z10 SE	20	2.77	Within Interval
Amiko A9 Green	16	9.36	Within Interval
A6	46	47.92	Overstock
Formuler Z11 Pro Max BT1	1	0.084	Within Interval
Lnb single Amiko	5	8.93	Within Interval
AMIKO MINI HD265	17	32.08	Understock
LEAP-S3	7	13.46	Within Interval
ZGemma H8.2H	16	32.65	Overstock
Amiko F-connector RG6	50	62.50	Understock
KHS014K	20	64.52	Understock
BHR4037GL	21	70.00	Overstock

Table 6.3: Error Metrics and Prediction Quality

Again, table 6.3 shows a small subset of the error metrics and prediction quality for the predictions made by the system. The overall evaluation was done for all 44 products.

The following insights can be drawn from the analysis:

- **19 products** were accurately forecasted within the confidence interval.
- **21 products** were understocked, which are acceptable given the dynamic nature of sales.
- **4 products** were overstocked, which are unacceptable due to the financial implications.

These results shows very interesting results regarding the performance of the system and how it can be used to aid inventory management decisions.

However, there are still some products that presented some concerning results.

The following products were all predicted to be understocked, but the actual sales were much higher than predicted.

- **32LQ570B6LA and U6-LR:**

  Both of these had a low sales prediction, but the actual sales were much higher. After analysing the training data for these, it was possible to identify the reasoning for this forecast. Both of these products only had one entry in the training data, and both had a really low sales quantity. This led the system to predict low values for these products, as there was no indication that these products would have a higher demand.

Even with anomaly detection, the day-to-day sales were not considered to be outliers, and the outlier here was more the actual sales for February 2024.

- **EXTREMEBOX AIR MOUSE, IBELF and U-Cable-Patch-1M-RJ45-BK:**

  These were completely new products that had no sales data in the training data. These were added to verify the system's ability to predict sales for new products. The system would look more to the category that these were being added to, and try to predict the sales based on the category. While for other products this worked well, for these it did not.

These products present some concerning results, they are still considered to be acceptable, since they tell the user that the trend is downward, and that, normally, these shouldn't have an irregular increase in sales. Like it is the case in February 2024, they had that irregular increase, but this is not the norm. And at least, the company did not order unnecessary stock for these products and avoided financial losses.

However, there is one product in particular that had an irregular pattern in sales, but in the opposite direction.

The product K11 produced the highest error percentage, with a value of 162%. The system predicted that this product would sell 118 units in the month of February, but the actual sales were 45 units. The actual sales of this product were strange, as the sales for the previous months were much higher and closer to the predicted value. An average of the previous months was 121 units, and there was no indication that the sales would drop so much. This was a clear example of how the system fails in predicting sudden changes in sales, and how important it is to have a human touch in the process.

Overall, the system performed satisfactorily with predictions falling within an acceptable range for many products. The following represents the overall performance of the system for the 44 products:

- **Lowest Error:** Product: Formuler Z11 Pro Max BT1, Error: 1, Error Percentage: 0.084%
- **Highest Error:** Product: K11, Error: 73, Error Percentage: 162%
- **Average Error:** 33.8%
- **Success Rate:** 90.91%
- **Failure Rate:** 9.09%

Appendix B shows the prediction analysis process for all 44 products.

Chapter 7

Conclusions

This thesis focuses on the development and application of an automated platform for inventory management. Forecasting and analysis were developed for Kontrolsat, an electronics e-commerce company. However, other users or businesses can easily adapt it and use it.

The system's main goals were to guarantee precise demand forecasts, lower the possibility of overstocking and understocking, and enable a smooth integration with Kontrolsat's current infrastructure.

All of the project's objectives and requirements were met, and the system was successfully implemented, and evaluated.

The evaluation covered in Chapter 6 also showed that the predictive models worked well, yielding forecasts within a given confidence interval that were comparatively accurate.

Additionally, the system recognised trends in sales that were rising, falling, and stable, enabling real-time modifications to inventory control.

Given these results, it is possible to say that the platform has the potential to significantly enhance Kontrolsat's inventory management and operational efficiency.

Now that data-driven insights are available through the forecasting platform, ordering and restocking inventory decisions can be made with greater knowledge.

As Kontrolsat continues to grow, the system's design will also allow it to scale easily, without compromising performance.

And, since it uses ML algorithms, the system will improve its predictions over time, as it learns from new sales data.

There are, however, still some areas that need further development, to make the system more user-friendly and secure.

The introduction of a user-friendly graphical interface will enhance its accessibility, and cloud deployment would enable the system to handle increasing loads as more users adopt it.

There is also significant potential to adapt the platform for other industries, beyond e-commerce.

Also, to try to combat some of the irregularities found in the prediction analysis, the accuracy could be further increased by utilising more sophisticated strategies, such as neural networks, which are particularly good with intricate sales patterns.

7.1 Limitations

Despite the system's successful implementation, several limitations remain that could impact its scalability, performance, and usability in certain environments.

7.1.1 Local Run Only

Currently, the system is designed to run only in local environments.

For businesses that require multi-user or multi-location access, this limitation prevents wider adoption.

It has not yet been optimized for deployment on cloud platforms.

It was built however with that in mind, so it should be easy to deploy it on cloud platforms. But as it stands, it can only be run on a local machine.

7.1.2 No UI Implementation

At present, the system lacks a dedicated graphical user interface (GUI).

Users must interact with the system through API tools like Postman or Swagger.

While these tools are useful for developers, they are not user-friendly for non-technical users, making it challenging for those without programming experience to navigate the system.

7.1.3 No User Authentication and Authorization

The system does not yet include user authentication and authorization mechanisms.

Since the system is still in development, this was not a priority, but it is a crucial feature for ensuring data security and user privacy. Especially, if the system is used in other companies, besides Kontrolsat.

These users would each be related to a specific company, and each of these could also have their specific roles. That would allow multiple users from the same company to access the system, and generate predictions for products.

7.2 Final Remarks

In summary, the developed platform for inventory forecasting and analysis offers significant advantages to Kontrolsat, providing accurate predictions that will help optimize the company's inventory management.

Better decisions about stock levels and product orders are made possible by the system's use of modern machine learning algorithms, which also yield actionable insights through trend analysis and confidence intervals.

Even with its remaining problems, the platform offers a strong base for further development. There are still opportunities for the system to grow into other business sectors and improve Kontrolsat's operations even further.

This project emphasises the need of continual development and adaptation in a rapidly changing e-commerce market, as well as the significance of data-driven decision-making in inventory management.

Bibliography

Alam, Md Shariful and Md Saib Ahmed (n.d.). *Analysis on Inventory Management and Economic Order Quantity of Linde Bangladesh Limited*.

Cadavid, Diana Cecilia Uribe and Carlos Castro Zuluaga (n.d.). *A framework for decision support system in inventory management area*.

Chen, Yasheng and Mohammad Islam Biswas (Nov. 2021). "Turning crisis into opportunities: How a firm can enrich its business operations using artificial intelligence and big data during covid-19". In: *Sustainability (Switzerland)* 13 (22). issn: 20711050. doi: 10.3390/su132212656.

Demizu, Tsukasa, Yusuke Fukazawa, and Hiroshi Morita (Nov. 2023). "Inventory management of new products in retailers using model-based deep reinforcement learning". In: *Expert Systems with Applications* 229. issn: 09574174. doi: 10.1016/j.eswa.2023.120256.

Dignum, Virginia (Mar. 2018). *Ethics in artificial intelligence: introduction to the special issue*. doi: 10.1007/s10676-018-9450-z.

Fernandez, Maria Isabel et al. (June 2021). "A Data-Based Model Predictive Decision Support System for Inventory Management in Hospitals". In: *IEEE Journal of Biomedical and Health Informatics* 25 (6), pp. 2227–2236. issn: 21682208. doi: 10.1109/JBHI.2020.3039692.

Fernández, Gerardo Molinary (2000). "The Evolution of Inventory Management in Manufacturing and Services Companies". In.

Hossain, Emam, Muhammad Ali Babar, and Hye Young Paik (2009). "Using scrum in global software development: A systematic literature review". In: pp. 175–184. isbn: 9780769537108. doi: 10.1109/ICGSE.2009.25.

IntellStyle (2023). *What Is An Omnichannel Retail Strategy... Truly*. url: https://www.intelistyle.com/omnichannel-retail-strategy-simplified/.

Javaregowda, Madhuri et al. (2020). *Inventory management using Machine Learning*. url: www.ijert.org.

Liu, Shiyu et al. (June 2024). "Data-driven dynamic pricing and inventory management of an omni-channel retailer in an uncertain demand environment". In: *Expert Systems with Applications* 244, p. 122948. issn: 09574174. doi: 10.1016/j.eswa.2023.122948. url: https://linkinghub.elsevier.com/retrieve/pii/S0957417423034504.

Marco Melacini Sara Perotti, Monica Rasini and Elena Tappia Gino Marchet (n.d.). *Logistics in omni-channel retailing: modelling and analysis of three distribution configurations*. isbn: 9781509058471.

Mesfer, Abdulelah S. Al (n.d.). *Forecast-Driven Inventory Management for the Fast-Moving Consumer Goods Industry*.

Namir, Khalil, Hassan Labriji, and El Habib Ben Lahmar (2021). "Decision Support Tool for Dynamic Inventory Management using Machine Learning, Time Series and Combinatorial Optimization". In: vol. 198. Elsevier B.V., pp. 423–428. doi: 10.1016/j.procs.2021.12.264.

Osman, Bashir Muhummed, Srirag Alinkeel, and Dhwani Bhavshar (n.d.). "A STUDY ON ROLE OF ARTIFICIAL INTELLIGENCE TO IMPROVE INVENTORY MANAGEMENT SYSTEM". In: *www.irjmets.com @International Research Journal of Modernization in Engineering* 226 (). issn: 2582-5208. url: www.irjmets.com.

Paula Vidal, Guilherme Henrique de et al. (Dec. 2022). "Decision support framework for inventory management combining fuzzy multicriteria methods, genetic algorithm, and artificial neural network". In: *Computers and Industrial Engineering* 174. issn: 03608352. doi: 10.1016/j.cie.2022.108777.

Qi, Meng et al. (n.d.). *A Practical End-to-End Inventory Management Model with Deep Learning*. url: https://ssrn.com/abstract=3921105.

Senthilnathan, Samithambe (n.d.). *ECONOMIC ORDER QUANTITY (EOQ)*. url: http://ssrn.com/abstract=3475239Electroniccopyavailableat:https://ssrn.com/abstract=34752392http://ssrn.com/abstract=3475239.

Sheffi, Yossi (Apr. 2017). "Joseph the Logistician: A Biblical Tale of Our Time". In.

Singh, Navdeep (Nov. 2023). "AI in Inventory Management: Applications, Challenges, and Opportunities". In: *International Journal for Research in Applied Science and Engineering Technology* 11 (11), pp. 2049–2053. issn: 23219653. doi: 10.22214/ijraset.2023.57010. url: https://www.ijraset.com/best-journal/ai-in-inventory-management-applications-challenges-and-opportunities.

Zhao, Bo and Chunlei Tu (2021). "Research and Development of Inventory Management and Human Resource Management in ERP". In: *Wireless Communications and Mobile Computing* 2021. issn: 15308677. doi: 10.1155/2021/3132062.

Appendix A

System Initialization Logs

```
# Application settings
APP_ENV=PROD  # Can be set to CLEAN_TEST or TEST

# Settings to connect to an external database:

DB_HOST=localhost # Optional
DB_PORT=5432 # Optional
DB_USER=diogo # Optional
DB_PASSWORD=diogo # Optional
DB_NAME=test_db # Optional

HISTORICAL_SALES_TABLE=historical_sales_info # Optional
PRODUCT_INFO_TABLE=product_info # Optional
FAMILY_INFO_TABLE=family_info # Optional

# Settings to get the data from imported files:

HISTORICAL_SALES_INFO_PATH=data/raw/historical_sales_info # Optional
HISTORICAL_SALES_CACHE=data/cache/historical_sales_info/cached \\
# Optional

PRODUCT_INFO_PATH=data/raw/product_info # Optional
PRODUCT_INFO_CACHE=data/cache/product_info/cached # Optional

FAMILY_INFO_PATH=data/raw/family_info # Optional
FAMILY_INFO_CACHE=data/cache/family_info/cached # Optional

MODELS_PATH=data/models
MODELS_PATH_CACHE=data/cache/models

# Logging settings:

LOG_LEVEL=INFO # Can also be set to DEBUG for more detailed logs

# Performance settings:

PROCESSING_MULTIPLIER=1 # Can go between 0 and 1
```

Appendix B

Prediction Analysis for 44 products

ItemID	Quantity	Upper Confidence	Lower Confidence	Trend	Error		Actual Quantity	Interval		Absolute Error	Percentage	Quality	REASON
Formuler z10 SE	869	996	742	DOWNWARD	127		722	20.00%		20	0.03	WITHIN	
Amiko A9 Green	187	198	176	STABLE	11		171	20.00%		16	0.09	WITHIN	
A6	118	142	94	UPWARD	24		96	20.00%		46	0.48	OVER	
Formuler Z11 Pro Max BT1	96	120	72	UPWARD	24		119	20.00%		1	0.01	WITHIN	
Lnb single Amiko	56	61	51	DOWNWARD	5		56	20.00%		5	0.09	WITHIN	
AMIKO MINI HD265	40	42	38	DOWNWARD	2		53	20.00%		15	0.28	UNDER	
LEAP-S3	75	91	59	DOWNWARD	16		52	20.00%		7	0.13	WITHIN	
ZGemma H8.2H	74	83	65	DOWNWARD	9		49	20.00%		16	0.33	OVER	
Amiko F-connector RG6	42	54	30	DOWNWARD	12		80	20.00%		50	0.63	UNDER	
KHS014K	4	26	-18	UPWARD	22		31	20.00%		5	0.16	WITHIN	
BHR4037GL	104	157	51	DOWNWARD	53		30	20.00%		21	0.70	OVER	
32LQ570B6LA	2	9	-5	STABLE	7		57	20.00%		55	0.96	UNDER	There was only 1 entry. Low volume of sales
U6-LR	3	4	2	STABLE	1		53	20.00%		50	0.94	UNDER	There was only 1 entry. Low volume of sales
Formuler Z11 Pro BT1	41	45	37	STABLE	4		49	20.00%		8	0.16	WITHIN	
Amiko HD 8165	27	37	17	UPWARD	10		46	20.00%		9	0.20	WITHIN	
DVB4231GL	7	11	3	STABLE	4		46	20.00%		39	0.85	UNDER	There was only 2 entries. Low volume of sales
K11	118	119	117	STABLE	1		45	20.00%		73	1.62	OVER	The average sales from previous months are 121
T30	29	39	19	DOWNWARD	10		43	20.00%		24	0.56	UNDER	
Amiko Mira X 1100	22	34	10	UPWARD	12		41	20.00%		7	0.17	WITHIN	
KT-320 HT PRO	4	5	3	DOWNWARD	1		41	20.00%		38	0.93	UNDER	
MIDE32STV	12	31	-7	UPWARD	19		37	20.00%		6	0.16	WITHIN	
Amiko T765	1	1	1	STABLE	0		33	20.00%		32	0.97	UNDER	
AT0052	1	1	1	STABLE	0		31	20.00%		30	0.97	UNDER	
ED0005	20	24	16	UPWARD	4		30	20.00%		6	0.20	WITHIN	
Xsarius Sniper V	31	36	26	DOWNWARD	5		30	20.00%		4	0.13	WITHIN	
ELA4740EU	21	25	17	DOWNWARD	4		28	20.00%		11	0.39	UNDER	
Extremebox Air Mouse Pro	2	2	2	STABLE	0		27	20.00%		25	0.93	UNDER	
EXTREMEBOX AIR MOUSE	1	2	0	UPWARD	1		27	20.00%		25	0.93	UNDER	Completely new product
Remot control Amiko	27	34	20	STABLE	7		26	20.00%		1	0.04	WITHIN	
161016	43	56	30	DOWNWARD	13		25	20.00%		5	0.20	WITHIN	
VR7544	1	3	-1	STABLE	2		24	20.00%		23	0.96	UNDER	
Formuler Z10	23	30	16	STABLE	7		24	20.00%		1	0.04	WITHIN	
MIDE24ESMART	9	20	-2	UPWARD	11		24	20.00%		4	0.17	WITHIN	
SX887 SE WL	9	14	4	UPWARD	5		24	20.00%		10	0.42	UNDER	
cabohdmi	10	14	6	UPWARD	4		23	20.00%		9	0.39	UNDER	
Amiko TSC-1270	5	7	3	STABLE	2		23	20.00%		18	0.78	UNDER	
RP0444	7	8	6	UPWARD	1		22	20.00%		14	0.64	UNDER	
Amiko WLN-861	18	31	5	STABLE	13		22	20.00%		4	0.18	WITHIN	
GTV-IR1	20	24	16	DOWNWARD	4		20	20.00%		4	0.20	WITHIN	
ANTENA60FERROK	18	23	13	DOWNWARD	5		20	20.00%		7	0.35	UNDER	
Extremebox tv	17	20	14	DOWNWARD	3		20	20.00%		6	0.30	UNDER	
IBELF	0	0	0	STABLE	0		20	20.00%		20	1.00	UNDER	Completely new product
U-Cable-Patch-1M-RJ45-BK	0	0	0	STABLE	0		20	20.00%		20	1.00	UNDER	Completely new product
Formuler Z10 Pro	18	20	16	UPWARD	2		19	20.00%		1	0.05	WITHIN	
											0.3382653		

			ItemID	Error	Percentage
WITHIN	19	Highest Error	K11	73	1.62
UNDER	21	Lowest Error	Formuler Z11 Pro Max BT1	1	0.01
OVER	4				

SUCESS RATE	90.91%
FAIL RATE	9.09%

Figure B.1: Prediction Analysis for 44 products

Appendix C

API Prediction JSON Format


```json
{
    "metadata": {
        "timestamp": "2024-09-11T01:45:03.023011",
        "message": "Predictions retrieved successfully"
    },
    "pagination": {
        "page": 1,
        "page_size": 100,
        "total_items": 1,
        "total_pages": 1
    },
    "details": {
        "start_date": "2024-02-01",
        "end_date": "2024-02-29",
        "item": "Formuler Z10 Pro",
        "family": "IPTV",
        "price": "115.58",
        "predictions": {
            "model_used": "BaggingRegressor",
            "quantities": {
                "total_quantity": 18,
                "bounds": {
                    "min": 16,
                    "max": 20
                }
            }
        },
        "trends": {
            "estimated_interval_forecast": 26,
            "units_per_day": 0,
            "trend": "UPWARD"
        },
        "errors": {
            "RMSE": 2
```


Figure C.1: API Prediction JSON Format seen in Postman

Instituto Superior de Engenharia do Porto ⊙ Rua Dr. António Bernardino de Almeida, 431 ⊙ 4249-015 Porto
Tel. +351 228 340 500 ⊙ Fax +351 228 321 159 ⊙ mail@isep.ipp.pt ⊙ www.isep.ipp.pt