POLITECNICO DI TORINO

Dipartimento di Ingegneria Gestionale e della Produzione

Corso di Laurea Triennale:

Ingegneria Gestionale

Classe L-8 Ingegneria dell'Informazione

A.A. 2024/2025

Sessione di Laurea Dicembre 2025

Design and Development of a Software Application for Demand Forecasting and Inventory Optimization in Bicycle Retail

Relatore:
Prof. Averta Giuseppe Bruno

Candidato:
S310589 Magaldi Davide

1. PROJECT PROPOSAL

This section outlines the project proposal submitted to the supervisor for evaluation prior to the commencement of the work. Subsequently, the modifications made to the initial proposal, which were deemed necessary during the development of the project, will be presented.

1.1 Student author of the proposal

S310589 Magaldi Davide

1.2 Title of the proposal

Design and Development of a Software Application for Demand Forecasting and Inventory Optimization in Bicycle Retail

1.3 Description of the proposed problem

The application aims to assist the management of a bicycle retailer in making decisions regarding the quantities to order for the upcoming year. It does so by forecasting the demand for a particular product based on sales data from the previous two years, allowing the user to choose which forecasting algorithm to apply. Furthermore, it enables the identification of an optimal sub-inventory, that means the one that minimizes storage costs while simultaneously storing the bicycles that are most frequently purchased together in a single order.

1.4 Description of the managerial relevance of the problem

In a managerial perspective, the application facilitates business management by providing decision support based on demand forecasting algorithms. It allows management to determine the optimal quantities of products to order, reducing the risk of overstock and understock, while helping to minimize storage costs. In addition, it enables the identification of which products should be stocked together based on how frequently

they are purchased simultaneously, thereby improving stock management efficiency.

1.5 Description of the datasets for analysis

The database that will be used has been extracted from the data published on the website https://www.kaggle.com/datasets/dillonmyrick/bike-store-sample-database and was also employed in the examinations of Programming Techniques course. Only sales records from years 2016 and 2017 will be considered, excluding those from 2018, as they are significantly fewer compared to the ones from the previous years.

1.6 Preliminary description of the algorithms involved

The main algorithms involved will be:

• Algorithms studied during the Industrial Production Management and Scheduling course, used to forecast demand for year 2018 based on sales data from the previous two years.

• An algorithm that, considering the previous forecasts and the stock situation at the end of 2017, calculates the quantities of a product to be ordered for the new year, ensuring a certain safety stock level in the warehouse.

• A recursive algorithm operating on the nodes of the previously created graph, which, given a maximum range of bicycles that can be stored, identifies the types of bicycles that should be stocked in the sub-inventory in order to minimize storage costs while simultaneously maximizing a defined score, measured as the number of times these types of bicycles are purchased together in the same order.

1.7 Preliminary Description of the Features Planned for the Software Application

Through the graphical interface, the user begins by selecting one of the three available stores for which demand forecasting and stock optimization will be performed. Then, he can choose the forecasting algorithm to apply and, if needed, provide additional parameters. Afterward, the user selects one of the products from a dropdown menu.

For the optimization of the sub-inventory, the user specifies the maximum number of bicycles that can be stored, with a 10% tolerance defining the allowed range. Once all inputs are provided, the user can trigger the calculations through dedicated buttons and immediately view the resulting forecast and optimization outcomes.

1.8 Modifications of the initial proposal

During the execution of the project, the following modifications were made:

• It was considered preferable to select the quarter as the temporal reference period rather than the full year, since the database contains complete data only for the previous two years, which is insufficient for producing reliable annual forecasts. On the other hand, the option of using a single month as the temporal period was discarded, as monthly sales for individual products are often very low or even null, potentially causing significant distortions in demand forecasting. Consequently, demand forecasts and calculation of order quantities refer to the first quarter of the upcoming year.

• As a result of the above, it was necessary to implement an algorithm to aggregate sales on a quarterly basis, starting from the monthly sales data obtained through a dedicated query aggregating by month.

• It was observed that several products in the database had recorded no

sales in previous years. For this reason, when constructing the graph representing products eligible for the application's functionalities, only those that were sold at least once in each year within the relevant store were considered.

• The demand forecasting algorithms studied during the Industrial Production Management and Scheduling that have been implemented in the application are Moving Average, Exponential Smoothing, and Exponential Smoothing with Trend.

• It was deemed appropriate to separate the two main functionalities of the application (forecasting/reordering and optimal sub-inventory generation) into two distinct pages, in order to enhance the intuitiveness, usability, and clarity of the graphical interface. Navigation between the two pages is facilitated through the Back/Next buttons located at the bottom of the interface.

2. DETAILED DESCRIPTION OF THE PROBLEM TACKLED

2.1 Problem identification

The problem addressed in this work was not directly assigned by the bicycle retailer in question, whose name is in fact unknown, instead it was entirely conceived, formulated, and developed by me, based on existing data.

The aim is to demonstrate how Information Technology tools, particularly software development, can serve as powerful instruments to extract valuable insights from raw data and provide significant support to the Management in making strategic decisions.

This project is therefore intended as a practical demonstration of the growing importance of integrating IT solutions into managerial and business processes.

The initial idea was to design and implement a small-scale software application capable of generating meaningful information to assist one of the key stages of Supply Chain Management: the planning phase, which is at the beginning of the entire process.

The application has thus been conceived as an effective demand forecasting tool, implementing some of the most widely used algorithms for this purpose; in addition, it allows the definition of inventory levels and suggests an efficient and cost-optimal organization of the inventory itself.

2.2 Supply Chain Management: the importance of the Planning phase

Supply Chain Management includes all the activities aimed at transforming raw materials into finished products and making them available to end customers in the most efficient way possible. It covers the processes of

planning, sourcing, production, storage, shipping, and distribution.
The primary goal of Supply Chain Management is to enhance overall efficiency, quality, productivity, and customer satisfaction, while simultaneously promoting flexibility and sustainability throughout the entire value chain.
The planning phase plays a crucial role, as it enables the integrated coordination of demand, production, and distribution, optimizing resource utilization and ensuring the continuity of operational flow.
In particular, it requires, as the starting point, demand forecasting: a predictive analysis process based on historical data, through which a company estimates how many products its current or future customers will require.
This allows production to be adjusted accordingly, in order to avoid both the overstocking of unsold products and, conversely, the shortage of items needed to meet customer demand, known as stockout.
The forecasting process represents the conceptual foundation of the software application developed within this work, which, as said, aims to demonstrate how IT-based tools can effectively support decision-making in the early stages of supply chain management.

2.3 Problem Inputs

As already mentioned in the previous paragraph, an effective planning phase requires a solid historical dataset. This means having access to a dataset that contains all relevant information regarding products, sales, and inventory in a clear, well-structured, and consistent format. Therefore, the dataset must be sufficiently large, providing an adequate amount of historical data; complete, meaning it consistently includes all necessary information without gaps or temporal discontinuities; and as clean as possible, i.e., containing data that can be directly used to extract meaningful insights without extensive preprocessing. Consequently, the

quality of the data on which demand forecasts are based plays a crucial role. High-quality and reliable forecasts can only be obtained if the underlying data meet the above requirements.

Another important input for forecasting is the predictive model itself, that is, the algorithm that will process the input data to produce meaningful results. The choice of algorithm is equally significant, as different algorithms applied to the same dataset may lead to highly different outcomes. Thus, it is essential to deeply understand these algorithms in order to interpret the results correctly and determine, depending on the context, which algorithm is most appropriate for the forecast. There are many possible approaches to forecasting. In this project, three time-series-based algorithms have been implemented, giving the end user the flexibility to select the most appropriate method depending on the context and, if desired, to compare the produced outputs.

These algorithms are presented and analyzed in the following subsection.

2.3.1 Implemented Predictive Algorithms

The three demand forecasting algorithms implemented in the application are:

• **Moving Average**: It calculates the average of the last m time periods. When used for forecasting, m is generally chosen to be not too large, so that the average is computed from the most recent periods while ignoring results from periods that are chronologically too distant. In the case of the application, only 8 previous time periods were available, so $m=8$ was set as a fixed value, considering all the data at hand. This algorithm is suitable for contexts with minor fluctuations that do not follow a specific trend or pattern. The related formula is:

$$f_{t+1} = \frac{1}{m} \sum_{i=0}^{m-1} f_{t-i}$$

• **Exponential Smoothing**: This method assigns exponentially decreasing weights to historical data, giving more importance to the most recent periods. The forecast for the following period is obtained as a linear combination of the most recent observed value and the previous forecast, using a smoothing coefficient α ($0 \leq \alpha \leq 1$). This method allows the forecast to adapt rapidly to variations in demand, making it particularly useful for time series without a marked trend but with significant recent fluctuations. In this project, the coefficient is an input parameter provided by the user, making the predictive algorithm more flexible and adaptable. The forecast is given by the following formula:

$$f_{t+1} = \alpha y_t + (1 - \alpha) f_t$$

where y_t is the observed value at time t and f_t is the forecast for time t.

• **Exponential Smoothing with Trend**: This algorithm extends Exponential Smoothing by adding a trend component, allowing it to forecast time series that exhibit a consistent increase or decrease over time. In addition to the smoothing coefficient α, a trend coefficient β is introduced to control the influence of the trend on forecast. This method is particularly suitable when the data show a clear trend (increasing or decreasing) and enables more accurate predictions compared to simple exponential smoothing. In this project, the coefficients α and β are input parameters requested by the application. The forecast f_{t+h} is calculated as follows:

$$\ell_t = \alpha y_t + (1 - \alpha)(\ell_{t-1} + b_{t-1})$$
$$b_t = \beta(\ell_t - \ell_{t-1}) + (1 - \beta) b_{t-1}$$
$$f_{t+h} = \ell_t + h b_t$$

Where ℓ_t = estimated level at time t, b_t = estimated trend at time t and h = forecast horizon.

2.3.2 Input-Related Challenges

As previously mentioned, demand forecasting requires high-quality input data. In the case under analysis, the available database is complete and well-structured; however, it contains a rather limited historical record. This means that, while forecasts can still be produced, their accuracy is likely to be lower than it would be if longer time series were available. Nevertheless, the developed application still represents a valuable tool for the future, as it is expected that the retailer will gradually accumulate more data, making the forecasting system increasingly effective over time.

Another data-related challenge that was addressed, that will be better detailed in the dataset description section, concerns the sales quantities recorded for each individual product, which are generally quite low and sometimes even zero for several consecutive months or years, with notable fluctuations. If not properly managed, this characteristic could lead to unreliable forecasts or distorted results due to the recurring presence of months with no recorded sales.

Despite these limitations, the available inputs provide a solid foundation for addressing and solving the problem, as will be demonstrated throughout this technical report.

2.4 Problem Outputs

The results obtainable from the input data and through the use of the application provide highly valuable information for the company's management. In particular, the software is able to calculate the demand forecast for the following quarter based on the predictive algorithm selected by the user. Moreover, it can determine the exact quantity of each product to purchase in order to meet the predicted demand while simultaneously maintaining a specified safety stock level, provided as input by the user.

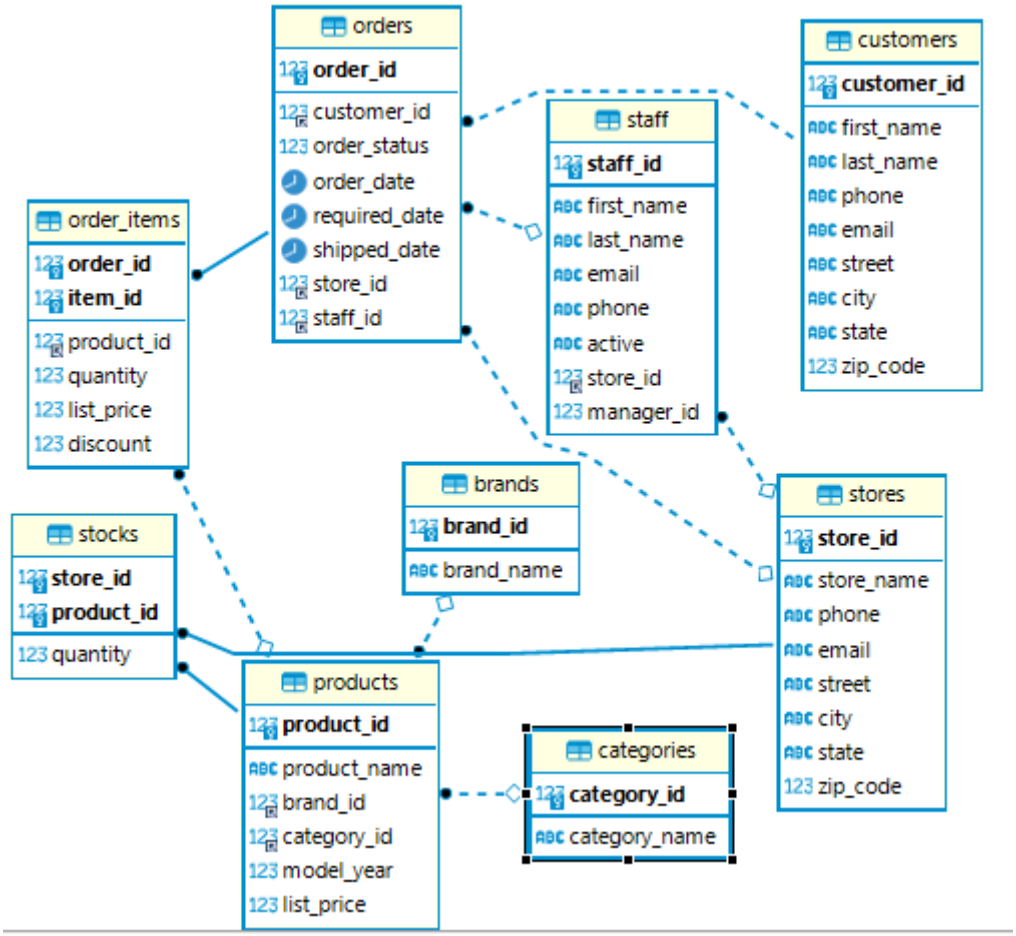
Finally, a section of the application allows the definition of an optimal sub-inventory, with a size within the range specified by the user, indicating which products (with their respective quantities currently held in the main warehouse) should be stocked together as they are most frequently sold in same orders. This approach minimizes storage costs while facilitating the efficient picking of bicycles that are sold simultaneously.

2.4.1 Output-Related Challenges

The main challenge in presenting the outputs lies primarily in providing the results in near real-time, ensuring good performance in the computation of results. This difficulty is particularly evident in the search for the optimal sub-inventory, as the computational complexity of the recursive algorithm grows exponentially with respect to the number of nodes in the graph, i.e., the number of products considered. For this reason, different techniques and strategies have been adopted, which will be described later, to make the algorithm as efficient as possible. Nevertheless, for particularly large input values, the algorithm still requires a significant amount of time to find a solution. However, when operating with realistic numbers for the sub-inventory of the retailer considered, the result is generated within a reasonably short time.

3. DESCRIPTION OF THE DATASET USED FOR THE ANALYSIS

The dataset used has been extracted from the data published on the website https://www.kaggle.com/datasets/dillonmyrick/bike-store-sample-database. It is a relational database containing information about the stores, products, stock, orders, and staff of a bicycle retail company. The database is structured as illustrated in the figure below.



3.1 Key Tables of the Dataset

The main tables from which the data have been extracted are:

• **STORES**: Contains three different stores, which can be interpreted as three branches of the same retailer. Information about the store is relevant because forecasts are generated for the specific store being analyzed at that

time.

• **PRODUCTS**: Contains information on 321 distinct products, including the list price.

• **STOCKS**: Indicates, for each store and product, the quantity currently available in stock.

• **ORDERS**: Records general information about orders placed at a store by a customer on a specific date. In particular, it includes orders received in 2016, 2017, and 2018. Upon analyzing the data, it was observed that orders in 2018 were significantly lower than in the previous two years. Therefore, the analysis only considered data from 2016 and 2017, projecting the forecast onto the first quarter of 2018.

• **ORDER_ITEMS**: Shows the details of each order, including information about the sold products, quantities, and sales prices. It was noted that most of the products listed in the products table did not appear in this order details table, entailing the lack of necessary data to generate forecasts. For this reason, only products that recorded sales in both 2016 and 2017 at the relevant store have been considered.

4. OVERVIEW OF DATA STRUCTURES AND ALGORITHMS

In this section, the focus is on the practical development of the application, presenting the main data structures and algorithms implemented throughout the project.

4.1 Data Structures

The main data structures used are contained within the application model. Specifically, they store the most important information required for the program to function properly. The data structures used in the application include:

• **product_idMap**: An idMap, which allows access to a product using its unique product ID.

• **sales_for_quarters**: A dictionary containing the total sales of the selected product for each past quarter.

• **stocks**: A dictionary containing information about the current stock of a specific product at a given store. Essentially, this is equivalent to the stocks table.

• **graph**: A simple, weighted, undirected graph whose nodes represent all products sold in the selected store in both 2016 and 2017. An edge connects two products if they were sold together at least once in the same order, with the edge weight indicating the number of times the two products were sold together in a single order. This data structure allows efficient search for an optimal solution in generating the ideal sub-inventory.

• A dataclass **Store** which maps exactly to the contents of Stores table.

• A dataclass **Product** which maps exactly to the contents of Products table.

4.2 Main Algorithms

The main algorithms developed within the application are described below. It should be noted that all algorithms include, within the Controller, a validation phase for the user-provided inputs. If the inputs are invalid or missing, the system displays an alert message indicating the issue related to the entered input value.

• **Graph Construction Algorithm**: This algorithm, receiving as input the store selected by the user, creates and populates the graph with the corresponding data. It performs parameterized queries on the database (depending on the selected store) to retrieve the nodes and the edges, along with their respective weights. The algorithm is implemented across the buildGraph() methods of both the Controller and the Model classes.

• **Demand Forecasting Algorithm**: Depending on the predictive method selected by the user and receiving as input the store, the product, and the corresponding algorithm parameters, this algorithm triggers the demand forecasting process. It is implemented within the controller's doForecast() method, which calls the model methods forecastByMovingAverage(), forecastByExponentialSmoothing(), and forecastByExponentialSmoothingwithTrend(). These methods implement the predictive algorithms described in Section 2.3.1 and return the forecasted value for the upcoming quarter.

• **Quarterly Sales Aggregation Algorithm**: This algorithm aggregates past sales data on a quarterly basis, making them suitable for demand forecasting. It retrieves monthly sales data through a dedicated database query and, after performing the quarterly aggregation, populates the sales_for_quarters data structure. It is implemented within the model method getSalesForQuarters().

• **Order Quantity Calculation Algorithm**: Based on the forecasted demand and the safety stock level provided by the user, this algorithm

calculates the exact quantity of each product that needs to be ordered for the upcoming quarter. If the quantity currently available in stock is sufficient to meet the forecasted demand, the algorithm notifies the user that no new orders are required. It is implemented within the model method calculateOrderQuantities().

• **Optimal Sub-Inventory Search Algorithm**: This recursive algorithm identifies the optimal solution on the graph by minimizing storage costs, computed quarterly as 2% of the unit price of each product stored, and simultaneously maximizing the score, calculated as the sum of the edge weights between all nodes included in the solution; in other words, it maximizes the number of times the products in the sub-warehouse have been sold together.

The number of units to be stored in the sub-warehouse is provided as input by the user. Based on this input, the algorithm defines an admissible range of storable units by applying a ±10% tolerance (for instance, if the user input is 100, the sub-inventory will contain between 90 and 110 bicycles). The final output shows which products should be included in the sub-inventory, along with their respective quantities currently available in the main warehouse.

To ensure efficiency, the algorithm adopts a pruning technique, which removes from the search tree all branches that cannot lead to a better solution than the best one currently available. During recursion, if the storage cost of a partial solution exceeds the cost of the current best solution, that recursion stops, significantly reducing the number of operations. Furthermore, the recursion iterates only over nodes following the current one, avoiding duplicate combinations. Since the algorithm aims to identify a set of nodes where the order is irrelevant, this strategy drastically reduces redundant computations and improves efficiency.

The algorithm is implemented within the model method getOptimalSubInventory(), which triggers the recursion by invoking the findNext() method.

5. MAIN CLASSES OF THE APPLICATION

The application has been developed following the MVC (Model-View-Controller) and DAO (Data Access Object) design patterns, that will be described in the next subsection.

5.1 Design Patterns: MVC and DAO

The MVC (Model-View-Controller) pattern is employed to organize the application into three distinct layers. The Model contains the core business logic and manages the data and rules of the application, while the View handles the graphical user interface and presents information to the user. The Controller acts as an intermediary between the Model and the View, processing user inputs and coordinating the flow of data. This clear separation of concerns enhances modularity, facilitates code reusability, and improves maintainability, as modifications in one component, such as the user interface, do not directly affect the others.

In addition, the DAO (Data Access Object) pattern is used to abstract and encapsulate all interactions with the database. It provides a clean interface for performing operations, only queries in this case, without exposing the underlying database implementation. By using the DAO pattern, the application benefits from improved data management, easier maintenance, and greater flexibility, since any changes to the database or the persistence mechanism can be implemented independently of the rest of the application logic.

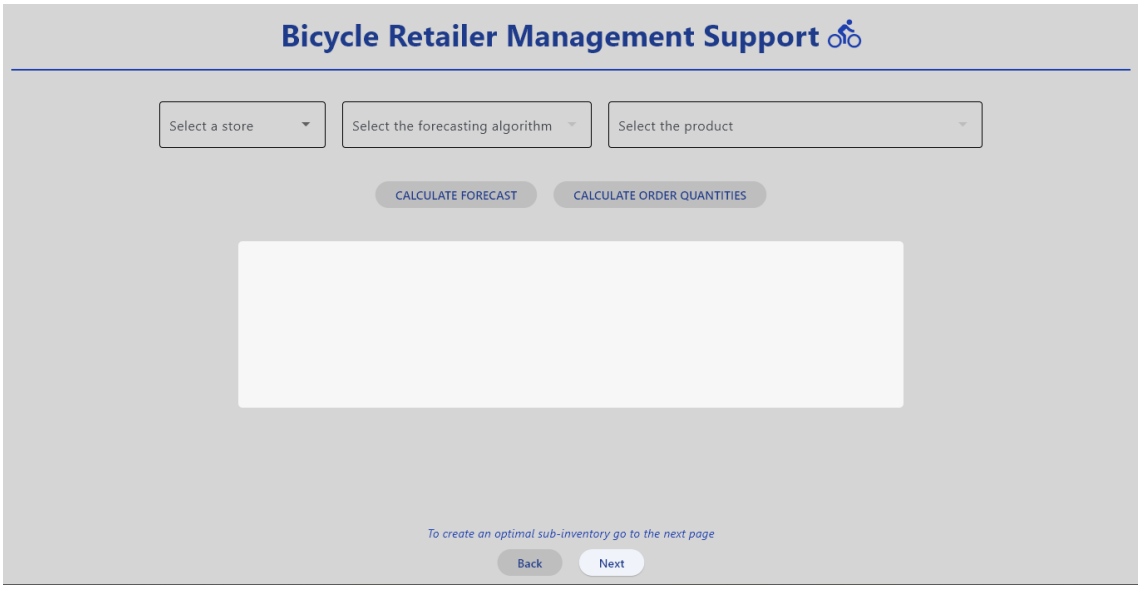
Overall, the combined use of the MVC and DAO design patterns contributes to creating a well-structured, scalable, and easily maintainable application architecture.

The following section presents the diagram of the two classes that contain the algorithmic core of the application: the Controller and the Model.

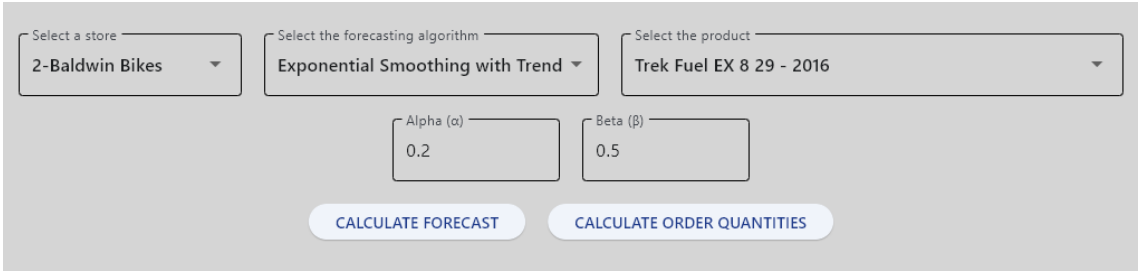
5.2 UML Class Diagrams for Controller and Model

6. APPLICATION USAGE GUIDE

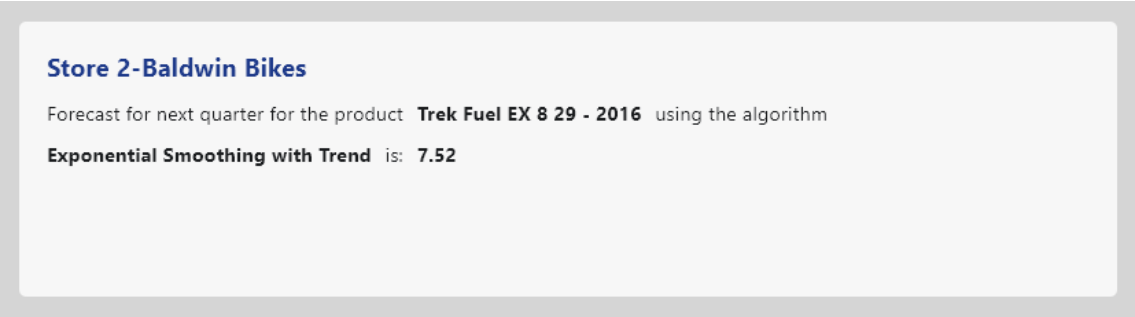
When launching the application, the following main screen will be displayed.



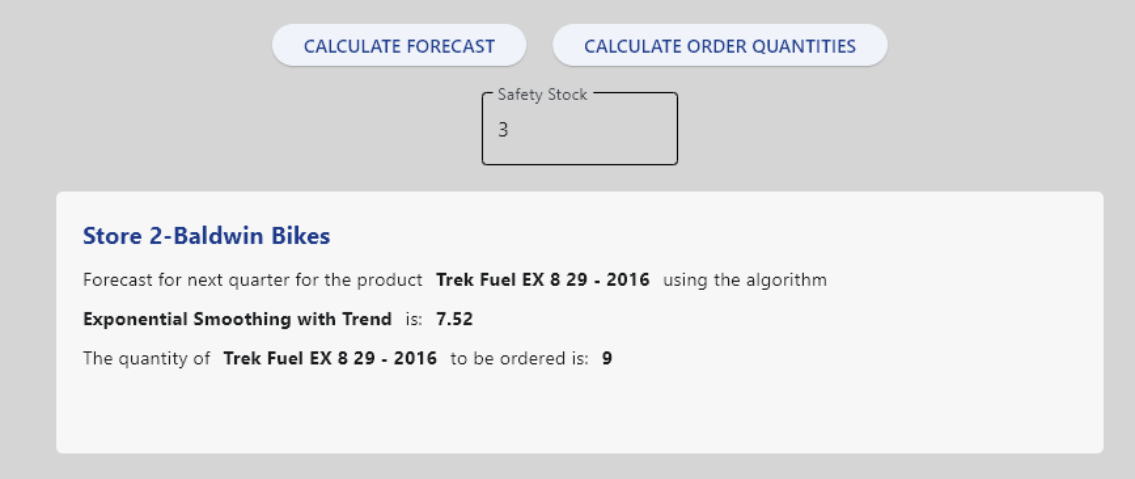
The main page is dedicated to demand forecasting and the subsequent calculation of order quantities. On this screen, the user can initially only select the store for which they want to perform operations using the appropriate dropdown menu. At this point, if the user wishes to proceed with forecasting or order calculations on this page, they must choose one of the three implemented algorithms and, if necessary, enter the parameters required by the application. They must then select one of the available products for which to obtain the forecast. Once these steps are completed, the buttons below will become active, as shown in the image.



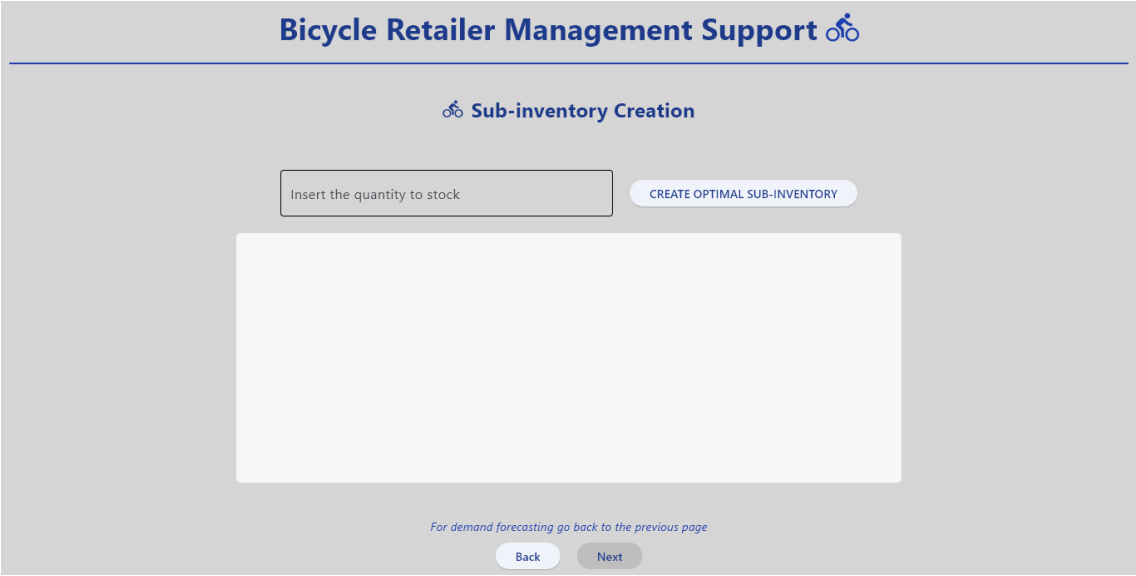
Now, if the user clicks the Calculate Forecast button, the white panel will display the corresponding output, as shown.



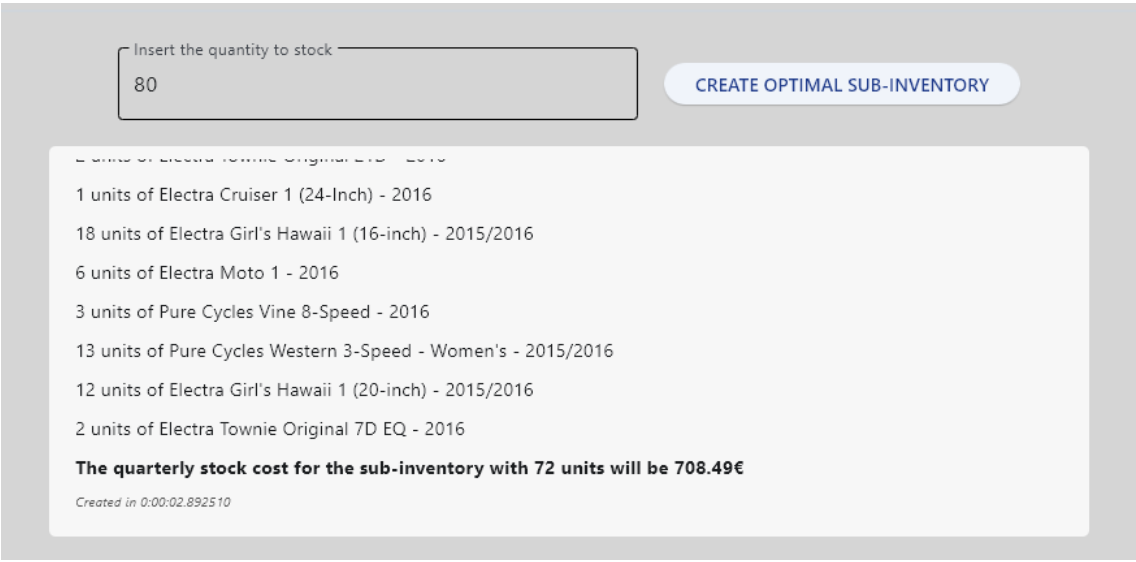
When the user clicks the Calculate Order Quantities button, they are prompted to enter the desired safety stock value. Once the value is entered, they must click the button again. The white panel will then also display the information about the exact quantity to order.



The second page of the application, accessible by clicking the "Next" button on the main page, is dedicated to creating the ideal sub-inventory and appears as shown in the following image.



At this point, the Create Optimal Sub-Inventory button is enabled because the user has already selected the reference store on the previous page; otherwise, it would be inactive. The user can then enter the quantity that will determine the range of bikes storable in the sub-inventory and click the button. This action will display the result in the designated area, as shown in the image.



For a better overview of the application's functionality and to see it in real-time, please refer to the demonstration video available at this link: https://youtu.be/zJbqexVxxAQ

7. EXPERIMENTAL RESULTS OBTAINED

This section presents some results obtained from the use of the developed application, which will later be discussed to assess their relevance, critical aspects, and performance. In particular, the forecasts obtained (and the corresponding quantities to be ordered) for two distinct products,Trek Fuel EX 8 29-2016 and Surly Ice Cream Truck Frameset-2016, will be analyzed as case studies. Subsequently, the results obtained from the creation of the sub-inventory will be reported by varying the quantity entered as input, in order to evaluate outputs and execution times, which are particularly important to analyze due to the rather high computational complexity of the recursive algorithm.

7.1 Forecasting and order quantities results

For each of the two analyzed products, a table will be provided showing the forecast values obtained for the three different stores using the three available algorithms, as well as a table with the corresponding quantities to be ordered for each product, based on the obtained forecasts. It should be noted that the forecasts have been kept with two decimal places for precision purposes, whereas the quantities to be ordered have been rounded up to the next integer, as they represent units of bicycles. Furthermore, the following abbreviations have been used to indicate the algorithms: MA for Moving Average, ES for Exponential Smoothing, and ESwT for Exponential Smoothing with Trend.

For ES, the example uses a value of $\alpha = 0.5$, in order to assign equal weight to the recorded value and the most recent forecast, while for ESwT, the parameters $\alpha = 0.2$ and $\beta = 0.5$ have been used. Finally, as a safety stock value (ss), value 3 has been set, meaning that at least three units of the given product are to be maintained in stock.

PRODUCT Trek Fuel EX 8 29-2016

forecast **Trek Fuel EX 8 29-2016**	**Store 1-Santa Cruz**	**Store 2-Baldwin**	**Store 3-Rowlett**
MA	3.88	11.0	5.12
ES(α=0.5)	1.82	6.24	5.45
ESwT(α=0.2, β=0.5)	0.0	7.52	5.71

Order Quantity **Trek Fuel EX 8 29-2016** (ss=3)	**Store 1-Santa Cruz**	**Store 2-Baldwin**	**Store 3-Rowlett**
MA	0	12	0
ES(α=0.5)	0	8	0
ESwT(α=0.2, β=0.5)	0	9	0

PRODUCT Surly Ice Cream Truck Frameset-2016

forecast **Surly Ice Cream Truck Frameset-2016**	**Store 1-Santa Cruz**	**Store 2-Baldwin**	**Store 3-Rowlett**
MA	4.25	14.12	2.12
ES(α=0.5)	2.81	8.82	1.84
ESwT(α=0.2, β=0.5)	2.37	8.67	2.24

Order Quantity **Surly Ice Cream Truck Frameset-2016** (ss=3)	**Store 1-Santa Cruz**	**Store 2-Baldwin**	**Store 3-Rowlett**
MA	8	7	0
ES(α=0.5)	6	1	0
ESwT(α=0.2, β=0.5)	6	1	0

7.2 Sub-inventory creation results

Regarding the sub-inventory generation, three different input quantities were tested, 50, 75, and 100; representing reasonable capacities for a sub-warehouse. These values define the range of bicycles that can be stored, considering a tolerance of ±10% according to the input quantity.
For each store, the algorithm's output includes the exact number of bicycles selected for storage (a detailed breakdown of the products can be found in the application) and the corresponding total quarterly stock cost. To assess the algorithm's performance, the execution times required to obtain the results are reported, followed by the size of the graph generated for each store. This comparison provides useful insights into the computational behavior and scalability of the recursive optimization algorithm.

Stocked quantity-Total stock cost	Store 1-Santa Cruz	Store 2-Baldwin	Store 3-Rowlett
q1=50	55-758.11€	55-464.95€	47-339.55€
q2=75	82-1561.79€	77-691.49€	78-1085.55€
q3=100	110-2096.68€	110-1268.73€	102-1301.07€

Execution Time	Store 1-Santa Cruz	Store 2-Baldwin	Store 3-Rowlett
q1=50	0.19s	0.66s	0.045s
q2=75	1.48s	3.66s	0.35s
q3=100	7.48s	33.86s	1.97s

Graph Dimensions	Store 1-Santa Cruz	Store 2-Baldwin	Store 3-Rowlett
Number of Nodes	22	25	24
Number of Edges	193	298	152

8. RESULTS EVALUATION AND CONCLUSIONS

8.1 Considerations on Previously Presented Results

Evaluating the results previously presented, data from Section 7 allow the following considerations:

• The predictive algorithms produce forecasts that are generally close to each other, although it can be observed that the results obtained with the Moving Average algorithm tend to deviate more from the other two, which, in contrast, usually provide very similar values. Overall, the forecasts tend to be rather low, particularly for algorithms that assign greater weight to the most recent periods. This indicates that the quantities sold for example products are following a downward trend, which is not captured by the Moving Average algorithm, making it less reliable in this example context.

• The forecasts obtained for the three stores are not similar, with Store 2 consistently showing higher values. This result is easily explained by the fact that Store 2 recorded more sales in previous years compared to the other two stores. This also translates into a greater number of edges in the graph, as shown in the tables in Section 7.2.

• The quantities to be ordered for the two products, while maintaining a low but realistic safety stock, are often zero. This indicates that the company already has more units in stock than needed, generating inventory costs that could have been easily avoided if the system developed in this project had been available beforehand.

• The time required to identify the optimal sub-inventory for reasonable input quantities is relatively low and is influenced not only by the number of nodes in the graph but also by the number of edges. As shown in Section 7.2, the execution times for Store 2 are significantly higher,

corresponding to the greater number of connections between nodes in the graph, while the number of nodes is similar across all three stores. The dependency on the number of edges arises from the calculation of the score for each solution, which requires checking the existence of an edge for all possible combinations of nodes in the graph.

8.2 Conclusions

The results previously described are consistent with the initial objective of the project and represent the outcome of a concrete and functioning solution to the problem outlined in Section 3. Indeed, the information obtained provides a highly useful tool for the company's management to strategically plan the supply chain, as it delivers precise and coherent data within a generally short timeframe.

However, it must be considered that the forecasts, as mentioned, are based on a relatively limited historical dataset. This can constitute a limitation on the accuracy of predictions, which could ideally be overcome over time as the company accumulates a larger historical dataset.

Regarding the recursive algorithm, it should be noted that, given its rather high computational cost, generating particularly large sub-inventories may not yield an immediate response, potentially requiring several minutes to produce a result. Furthermore, the available data express that the company sells only around twenty different products per year. If, instead, all products listed in the database Products table (321 items) were to be considered, the algorithm's execution time would increase dramatically, making this functionality practically unusable within an acceptable timeframe. In such a case, it would be reasonable to operate only on a subset of products, for example by considering for optimization only the 20 products with the highest sales in the previous year, thus reducing the problem to a scenario manageable by the application in near real-time.

Finally, it should be noted that, as the problem has been formulated, the recursive algorithm attempts to simultaneously optimize two objective functions, which are inherently in conflict with each other. Therefore, the solution obtained represents a trade-off between the two objectives, balancing the conflicting requirements to achieve a practically feasible and efficient outcome.

Despite these considerations, the developed software can be regarded as a valid and relevant tool, capable of providing meaningful support to strategic decision-making processes within the company. Its modular and data-driven structure allows for future improvements, making it adaptable to larger datasets and scalable to more complex operational contexts.

8.2.1 License