University of Applied Sciences

HOCHSCHULE EMDEN·LEER

Warehouse operations: Implementing efficient receiving and put-away processes

Mubariz Hajimuradov – 7020764

14.02.2024

Project Supervisor: Jeffrey Wermann

Abstract

This study focuses on optimizing warehouse operations by implementing efficient receiving and put-away processes. Foundational concepts cover barcode systems, including EAN-13. Using Laravel and React, the research details the technology stack selection, installation, and front-end development. Last section goes through what has been achieved in the project, limitations, and future directions of the project. This research offers practical insights into enhancing warehouse efficiency through technology implementation.

Table of Contents

1. Introduction

Efficient warehouse operations are crucial for businesses to meet customer demands and maintain a competitive edge in the market. One of the key aspects of successful warehouse management is the effective design of receiving and put-away processes. These two processes lay the foundation for streamlined operations, accurate inventory management, and overall productivity. In this project, we will explore and implement some essential elements of designing receiving and put-away processes in warehouse operations. Especially focusing on identification of goods and storage location allocation. We will not go into the details of other components for the sake of simplicity.

While we will focus on Receiving and Put-away processes, we are aware of Storage, Order Picking, Packing and other warehouse processes which play important role. [1] The chart in Figure 1 shows Warehouse activities as a percentage of total cost.

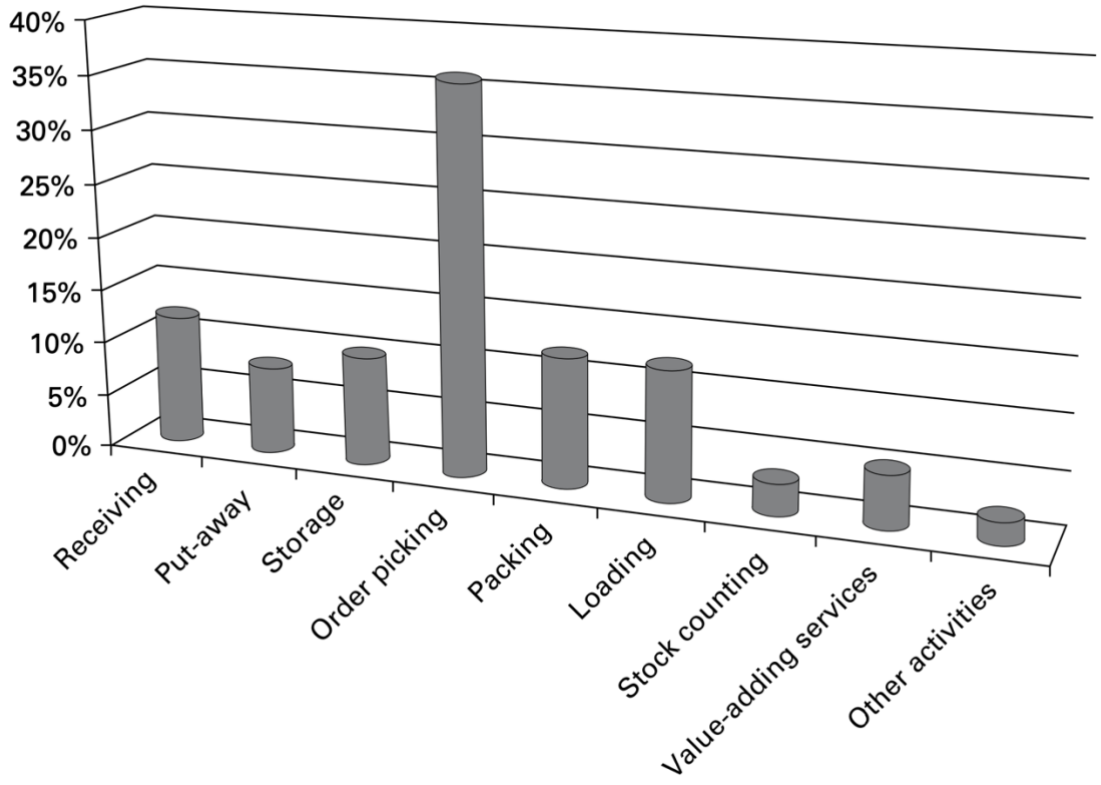


Figure 1. Warehouse activities as a percentage of total cost [1]

It is a common practice for companies to use existing Warehouse Management System solutions or building it for their own custom needs to manage these processes. In any case, there are some challenges need to be addressed:

- **Integration Issues**: Implementing a Warehouse Management System (WMS) may face integration challenges with existing enterprise systems like Enterprise Resource

Planning (ERP) systems, supply chain management tools, and other business applications. [2]

- **Data Accuracy and Integrity**: Maintaining accurate and reliable data within the WMS is crucial. Inaccurate inventory data can lead to errors in order fulfillment, stockouts, or overstock situations.
- **Customization and Scalability**: WMS implementations should be adaptable to the unique needs of the business. Lack of customization options or difficulties in scaling the system to accommodate business growth can be problematic.
- **High Implementation Costs**: The initial cost of implementing a WMS can be substantial, including software, hardware, and training expenses. This financial burden can be a challenge for small and medium-sized enterprises (SMEs). [2]
- **Employee Training and Adoption**: Training staff to effectively use the WMS is essential. Resistance to change or inadequate training can result in errors, inefficiencies, and a failure to fully utilize the system's capabilities. [3]
- **System Downtime and Technical Issues**: Technical glitches, system downtime, and maintenance issues can disrupt warehouse operations. Ensuring a robust IT infrastructure and prompt resolution of technical problems is crucial.
- **Supply Chain Complexity**: WMS needs to align with the complexity of the supply chain. In cases where the supply chain involves multiple partners, global operations, or diverse product types, the WMS needs to handle these complexities effectively.
- **Vendor Selection and Support**: Choosing the right WMS vendor is critical. Issues can arise if the selected vendor lacks adequate support, updates, or if the chosen system becomes obsolete over time.
- **Performance Metrics and Reporting**: The WMS should provide accurate and comprehensive performance metrics and reporting capabilities. Inadequate reporting can hinder decision-making processes.

Many big companies invested lots of resources on research and development to deal with these challenges. Some of them develop these solutions for their own needs while some are Software as a Service (SaaS) providers. It’s a common practice to consider WMS as a part of Supply Chain Management (SCM) Solution and implement them in interoperable way. Currently (in 2023) SAP is considered a Warehouse Management Systems Software market leader with ~38% market share (Figure 2).

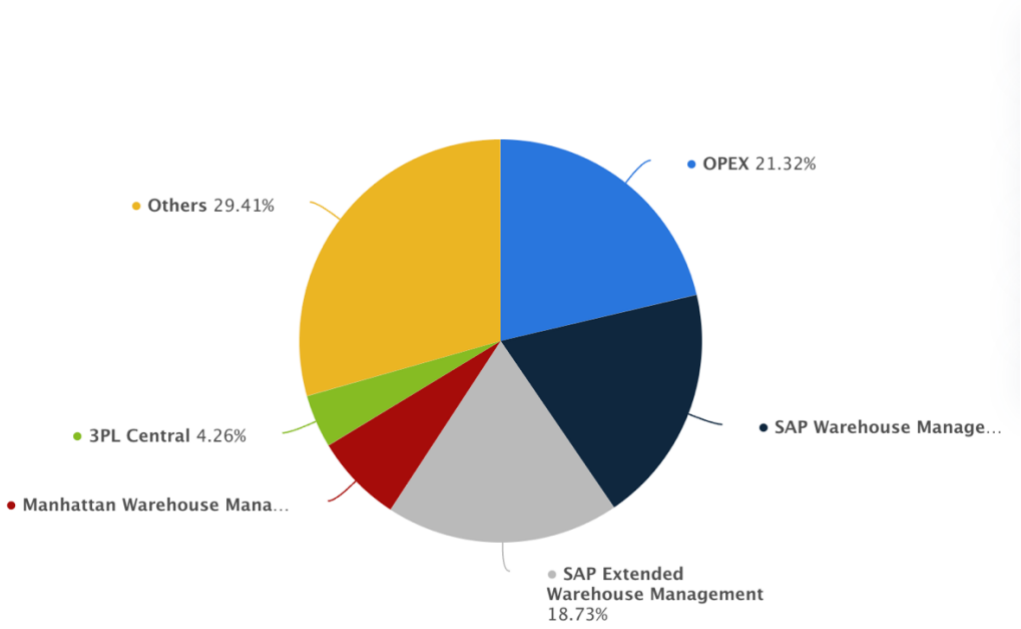


Figure 2. Share of warehouse management software market worldwide in 2023, by vendor [4]

But as companies operating in different industries may have some specific needs, Integration, Adoption processes requires some effort. There are 3rd party companies which provide service, and some develop custom add-ons to extend the functionalities of WMS. Usually, it's also possible for companies to develop only custom extensions and add-ons in-house while using off-the-shelf SCM & WMS solutions (hybrid approach).

While there are so many off-the-shelf SCM & WMS solutions, many companies still tend to develop custom SCM & WMS solutions internally and it has some pros and cons. (Figure 3).

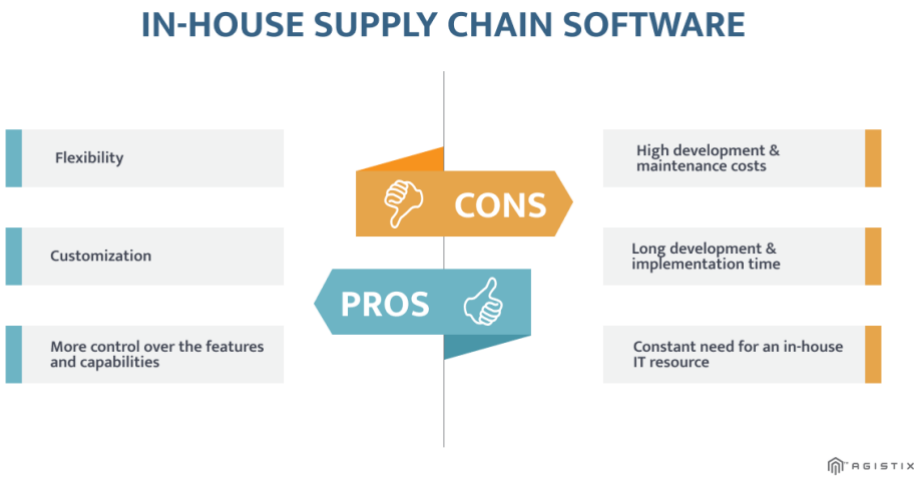


Figure 3. In-House Supply Chain Software: Pros and Cons [5]

In this project, we will try to analyze, understand the complexity of Receiving and Put-away processes of Warehouse and implement them by keeping Small and Medium Enterprises (SMEs) on focus.

2. Foundational Concepts

2.1. Warehouse Operations

Warehouse operations form the cornerstone of efficient supply chain management. The essential operations of warehouses are the same, despite variations in size, type, purpose, ownership, and location. These processes include pre-receipt, receiving, put-away, storage, picking, replenishment, value-adding services, and despatch (Figure 4). [6]

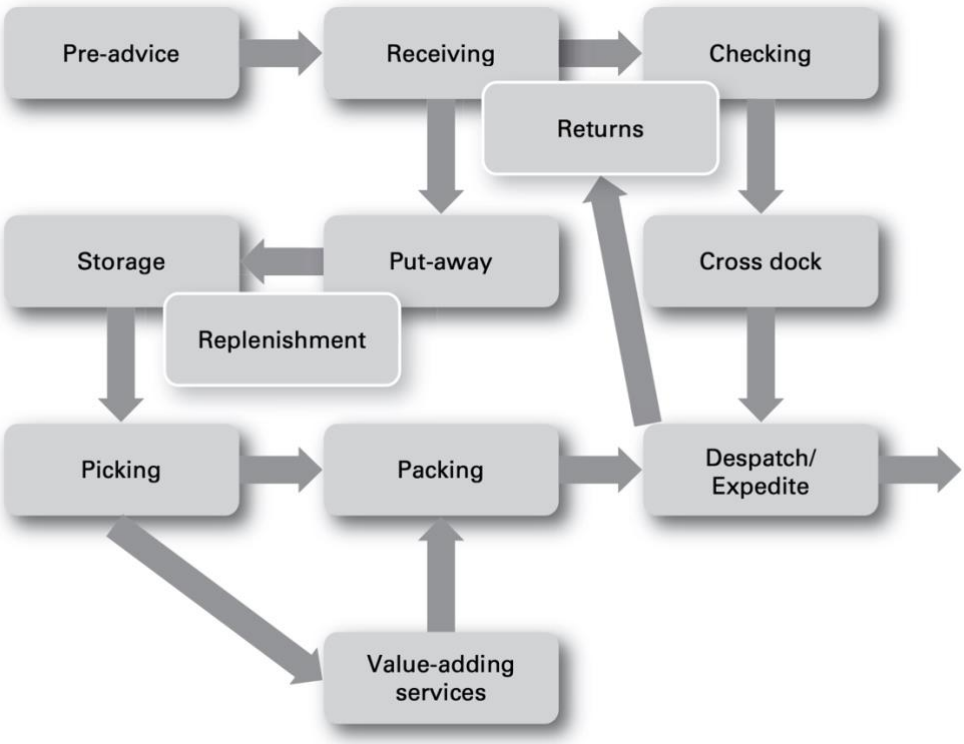


Figure 4. Warehouse Processes [6]

Even though we will not go into details of them in this project, it needs to be mentioned that there are many steps that need to be taken before the actual act of receiving takes place which also may affect efficiency of the warehouse operations.

2.1.1. Receiving

Ensuring the accurate and timely acceptance of incoming items into the warehouse is the main goal of the receiving process. This stage requires precision since it establishes the standard for procedures that follow. Receiving items process:

- Item Arrival and Unloading
- Inspection and Verification
- Documentation and Recordkeeping (Recording)

Efficient receiving is an essential part of the larger supply chain context, not just a logistical phase.

2.1.2. Put-away

Modern Warehouse Management Systems (WMSs) often pre-allocate specific locations for products, guiding operators on where to place goods efficiently. Depending on the warehouse strategy, this could be directly to the dispatch area for cross-docking, to the pick face for replenishment, or to a reserve or bulk-storage location.

To ensure the effectiveness of this system, extensive information must be programmed into the WMS, encompassing various factors such as:

- Size, weight, and height of palletized goods.
- ABC analysis or slotting results, prioritizing fast-moving goods closer to the dispatch area.
- Current order data.
- Family product groups.
- Actual sales combinations.
- Current status of pick face for each product.
- Size of pallet locations.
- Weight capacity of racking.

Determining the optimal placement of goods is crucial, especially if a sophisticated WMS is not in place. In such cases, warehouse managers need to calculate the best location and provide instructions to operators accordingly.

A key decision in this process is whether to use fixed or random locations. Fixed positions assign a specific location for a particular product, aiding pickers in memorizing locations and speeding up the picking process. However, the drawback is that if there's no stock for that product, the slot remains empty, significantly reducing pallet storage utilization.

Factors influencing product location include specific characteristics such as hazardous items requiring special storage and high-value items needing secure conditions like a lockable cage or a secure carousel.

When dealing with cartons, efficient placement is crucial. Fast-moving items should be in the middle row of shelving to reduce picker bending and stretching time, while slower-moving items occupy the lowest and highest shelves.

Grouping items by similarity is another consideration for the warehouse manager. For instance, automotive gearbox parts should be stored together, and products frequently appearing together on a pick list should be located side by side.

Some advanced warehouse systems employ task interleaving, combining put-away with pallet retrieval. In this scenario, the system guides the operator to put away a pallet en route to collecting a picked full pallet or one required for replenishment.

2.2. Barcode Systems & EAN Code

Barcode systems simplify tasks like inventory management, sales transactions, and asset tracking by encoding information in scannable patterns. They reduce errors, automate data entry, and enhance efficiency across various industries, making processes faster and more accurate.

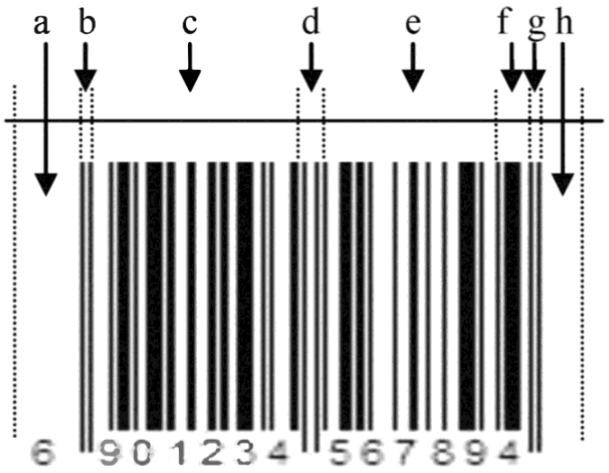
Again, to focus on the main goal of this project, we are going to focus only the most common barcode system EAN-13 during implementation. We must mention not all products have EAN-13 codes, many consumer goods and retail products do. The use of EAN-13 codes is common for items sold in retail stores, especially in supermarkets, department stores, and other places where products are scanned at the point of sale.

Certain types of products, such as small or custom-made items, might not have standardized barcodes. Additionally, products sold in certain regions or through specific channels might use different barcode systems or no barcodes at all.

It's also important to note that different regions may have variations of barcoding systems. For example, in the United States and Canada, the UPC (Universal Product Code) is more common, but it is a 12-digit system that is structurally like the EAN-13 used in many other parts of the world.

2.2.1. EAN-13 Barcode System

The EAN-13 barcode serves as a global standard for labeling retail products. Encoded within the EAN-13 barcode are product identification numbers. Structured as a (7,2) code, each character encompasses a total width of 7 modules, comprising two bars and two empty spaces alternately. The spacing between each bar and its corresponding empty space does not exceed 4 modules. The composition of the EAN-13 barcode includes the left blank area, start character, left data character, intermediate separator, right data character, check character, terminator character, and right blank area, as shown below figure.



a: left blank area
b: start character
c: left data character
d: intermediate separator
e: right data character
f: check character
g: terminator character
h: right blank area

Figure 5. EAN-13 barcode [7]

Consisting of 13 numeric characters, commonly referred to as digits, the EAN-13 barcode incorporates a country code as its initial two or three characters. Depending on the length of the country code, the subsequent sequence consists of either nine or ten characters representing the manufacturer code and product code. Notably, the 13th character of the EAN-13 barcode serves as the checksum digit.

Represented graphically, the EAN-13 barcode manifests as a line pattern corresponding to its 13-digit code. This pattern is constructed with alternating dark bars and light spaces of varying thickness. For clarity, this line pattern is simplistically regarded as alternating black and white bars with diverse widths. [7]

3. Conceptual Framework

If we simplify the definition of warehouse, it can be viewed as a temporary place to store inventory and as a buffer in supply chains. It functions primarily as a static unit, matching product availability to consumer demand. As such, its major goal is to make it easier for suppliers to get items to customers, satisfying demand in a timely and economical manner.

Based on that view we can simplify it as receiving items and shipping items (preparing for shipping), where put-away, picking and packing are in-between processes.

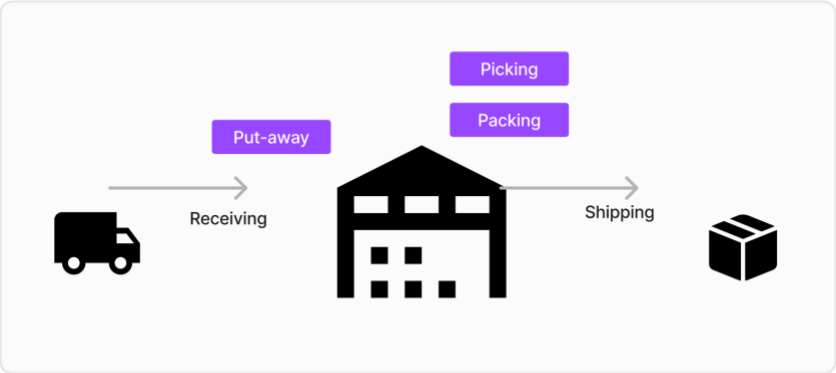


Figure 6. Simplified Warehouse model

If we zoom into the Warehouse itself, usually every Warehouse has Receiving, Packing and Storage Area in some form.

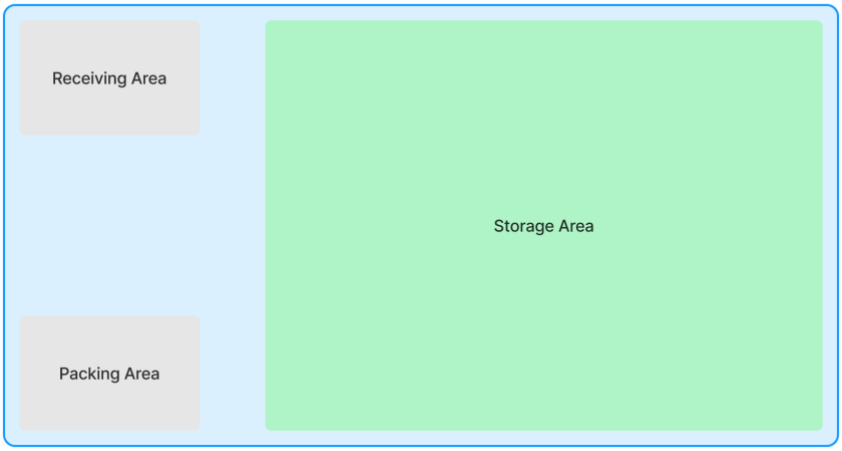


Figure 7. Warehouse Areas

For this project, we will use a specific common Storage Area structure, which contains Aisles (corridors), where each Aisle contains Rows, each Row has Levels, and each Level has Places. We are going to design a Warehouse model which contains 6 Aisles, each containing

6 Rows, each Row containing 4 Levels and each Level containing 5 Places as shown below Figures:

Figure 8. Warehouse, Structure of Storage Area

Figure 9. Warehouse Storage Area, Structure of a Row

In this implementation, the location of an item will look like [Storage Area]-[Aisle]-[Row]-[Level]-[Place], example: SA1-3-2-2-4. Receiving Area and Packing Area can also have their own structure if needed.

We will approach each component as a model which has its own properties. We can simplify the modeling by focusing on key components and their relations to each other and eliminating irrelevant models.

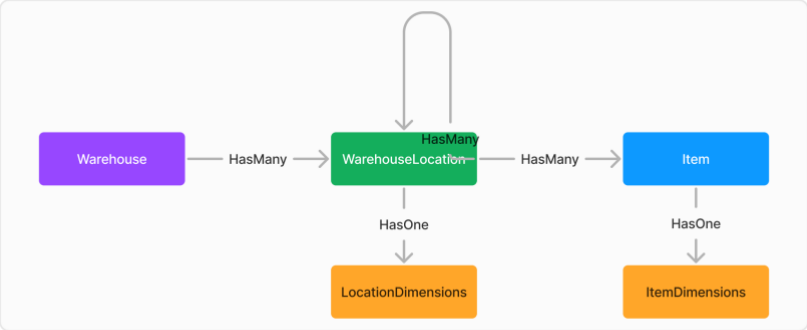


Figure 10. Modelling of Warehouse components and their relations

Every Aisle, Row, Place etc. is modelled as WarehouseLocation, it can even be a moving Card which can store Items. This way, we can implement above specific structure of Warehouse while keeping the system extremely flexible for any kind of implementation of structure.

If we divide the first part: receiving and put-away into simplified procedural steps, it will consist below steps:

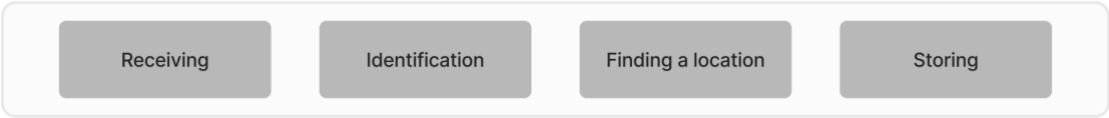


Figure 11. Procedural steps of Receiving & Put-away

This project also aims to develop a mobile UI/UX to allow users receive items with minimum effort. Another component is Administration Panel which will work with back-end, which will allow users to manage Warehouse, Item, WarehouseLocation and other models. It will handle complex logic and processes user requests.

Another component, the database serves as the persistent data storage layer within the web application architecture, adhering to established data governance and security protocols. It facilitates efficient data retrieval based on application logic defined within the backend API, ensuring consistency and integrity of information presented to users. Furthermore, the database exhibits scalability, adapting to increasing user volumes and data demands while

maintaining optimal performance, ultimately acting as the backbone for a seamless user experience. Overall architecture of the system shown in below Figure:

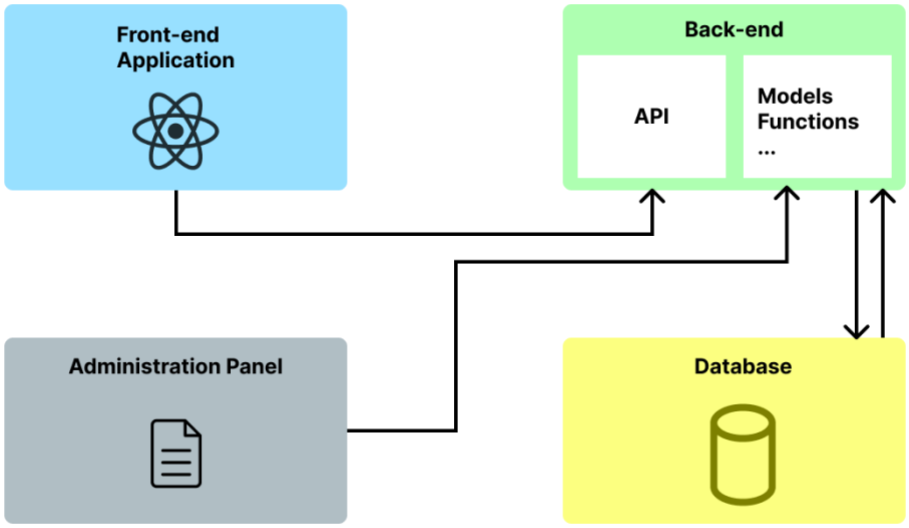


Figure 12. System Architecture

4. Implementation

In this section, we will go into details of Development process and key ideas will be described and explained. Full source code is available at GitHub repository.

4.1. Choosing Technology Stack

We are going to implement the project using Web technologies. Developing a web application provides seamless cross-platform compatibility, accessibility across devices, streamlined updates, and fewer installation headaches. It scales well, is affordable with a single codebase, and facilitates real-time collaboration. Further benefits include increased analytics, ease of setup, and global accessibility.

For storing data, we are going to use **MySQL** database. MySQL is an open-source relational database management system (RDBMS) that is widely used for managing and organizing data. Developed by Oracle, it uses a structured query language (SQL) to interact with databases. MySQL is known for its reliability, scalability, and ease of integration with various programming languages. It supports multiple storage engines, allowing users to choose the most suitable one for their specific requirements. MySQL is commonly utilized in web development for storing and retrieving data, making it a popular choice for dynamic websites and applications.

For back-end we are going to use a PHP framework, **Laravel**. Laravel is an open-source PHP web framework designed for building robust and scalable web applications. It follows the Model-View-Controller (MVC) architectural pattern, providing a structured and modular approach to development. Laravel offers features like Eloquent ORM for database interactions, Blade templating engine for efficient view management, and a robust set of tools for tasks like routing, authentication, and caching. With a focus on developer-friendly syntax and conventions, Laravel simplifies common tasks and accelerates the development process, making it a popular choice for building modern, maintainable PHP applications. Also, we are going to build an API using Laravel, which will be called from front-end.

To be able to create database records fast, we are going to use **Filament**. Filament is a collection of full-stack components specifically designed to enhance and accelerate Laravel development. These components are not only aesthetically pleasing but also designed to be user-friendly and easily customizable. By offering a comprehensive set of tools, Filament aims to serve as a robust foundation for Laravel applications, streamlining the development process and providing developers with a starting point for building feature-rich and visually appealing Laravel apps.

Finally, for building User interface, we are going to use a front-end framework, **React.js**. React.js, often referred to as React, is an open-source JavaScript library developed and maintained by Facebook. It is widely used for building user interfaces in single-page applications where data can change over time without requiring a page reload. React follows a component-based architecture, allowing developers to create reusable UI components that efficiently update in response to changes in data. It employs a virtual DOM (Document Object Model) to optimize rendering performance by selectively updating only the components affected by data changes. React can be seamlessly integrated with other libraries or frameworks, and it is commonly used in conjunction with tools like Redux for state management. Its declarative syntax and efficient rendering make it popular for building interactive and dynamic web applications.

4.2. Installing Laravel and Filament

Running below code installs Laravel and creates a new project:

```
composer create-project laravel/laravel mii-project2
```

Then we are going to install the Laravel package Filament:

```
composer require filament/filament:"^3.0-stable" -W
php artisan filament:install –panels
```

Below command will create a user to login to Admin panel:

```
php artisan make:filament-user
```

And update Laravel model App\Models\User.php to support Filament:

```
use Filament\Models\Contracts\FilamentUser;
use Filament\Panel;

class User extends Authenticatable implements FilamentUser {

	public function canAccessPanel(Panel $panel): bool
	{
	    return true;
	}

	…
```

Now we will be able to access Filament Admin panel at /admin route.

4.3. Creating Laravel models and migrations

The `php artisan make:model` command in Laravel is used to generate a new model file. Models in Laravel are representations of database tables, and this command streamlines the creation of these models. When you run `php artisan make:model Example`, it creates a new `Example` model file in the designated models directory, following Laravel's naming conventions. Below command will create a Laravel model **Item** and migration for it:

```
php artisan make:model Item -m
```

This command also generates a migration file for creating the corresponding database table, allowing us to define the structure and relationships within its application. Next, we can run the same command to generate models and migrations for: *Warehouse, WarehouseLocation, LocationDimensions and ItemDimensions.*

We also will add relationships between models as described in modelling of warehouse components and their relations. Please check repository for the details of models and migrations.

We also need to create Filament Resources for all models which will not be covered in this document. Creating and customizing resources is well documented on official Filament documentation.

4.4. Creating Laravel API routes

In Laravel, creating API routes is a fundamental step in building robust and scalable applications that communicate with external services or client-side applications. Laravel provides a clean and expressive syntax for defining API routes. To create API routes, we typically leverage the routes/web.php or routes/api.php files in the application.

In the routes/api.php file, we can define API routes using the `Route` facade, specifying the HTTP methods (GET, POST, PUT, DELETE, etc.) and the corresponding URI. Laravel's resourceful routing simplifies the creation of RESTful APIs by allowing us to define routes for common CRUD operations with a single `Route::resource()` method call.

Middleware plays a crucial role in API routes to handle tasks such as authentication, authorization, and request validation. Laravel's middleware system allows you to attach middleware to specific routes or groups of routes, ensuring that our API is secure and efficient.

By following Laravel's conventions for creating API routes, we can ensure a well-organized and maintainable codebase while taking advantage of the framework's powerful features for

building modern and scalable APIs. We are going to add below routes to our routes/api.php file:

```
Route::middleware('auth:sanctum')->get('/user', function (Request $request) {
    return $request->user();
});

// Item routes
Route::get('/items', [ItemController::class, 'index']);
Route::get('/items/{id}', [ItemController::class, 'show']);
Route::post('/items', [ItemController::class, 'store']);
Route::put('/items/{id}', [ItemController::class, 'update']);
Route::delete('/items/{id}', [ItemController::class, 'destroy']);

// WarehouseLocation routes
Route::get('/warehouse-locations', [WarehouseLocationController::class, 'index']);
Route::get('/warehouse-locations/{id}', [WarehouseLocationController::class, 'show']);
Route::get('/warehouselocation-by-code/{id}', [WarehouseLocationController::class,
'showByCode']);
Route::post('/warehouse-locations', [WarehouseLocationController::class, 'store']);
Route::put('/warehouse-locations/{id}', [WarehouseLocationController::class, 'update']);
Route::delete('/warehouse-locations/{id}', [WarehouseLocationController::class,
'destroy']);

// Transaction routes
Route::get('/transactions', [TransactionController::class, 'index']);
Route::get('/transactions/{id}', [TransactionController::class, 'show']);
Route::post('/transactions', [TransactionController::class, 'store']);
Route::put('/transactions/{id}', [TransactionController::class, 'update']);
Route::delete('/transactions/{id}', [TransactionController::class, 'destroy']);
Route::post('/receive-item', [TransactionController::class, 'receiveItem']);
```

The initial route, protected by Sanctum middleware, retrieves the authenticated user's information. This is Laravel's default built-in authentication system, ensuring secure access to sensitive user data. We will skip user authentication part and it’s details to focus on main operations.

The subsequent sections define CRUD operations for different resource types, such as items, warehouse locations, and transactions. The routes leverage RESTful conventions, employing the appropriate HTTP methods (GET, POST, PUT, DELETE) for each operation.

The `WarehouseLocationController` includes a method named `showByCode` designed to retrieve a warehouse location based on its unique code. This method expects a single parameter, `$code`, representing the unique identifier associated with the warehouse location.

Inside the method, there is an attempt to find the location using a custom static method, `findByCode`, within the `WarehouseLocation` model. If successful, the method responds with a JSON representation of the located warehouse location, indicating a successful retrieval. In case an exception occurs during the search, likely triggered by the absence of a matching location, the method gracefully catches the exception. It then responds with a JSON object containing the exception message and a 404-status code, signalling that the requested warehouse location based on the provided code was not found. This design ensures robust error handling and provides a specialized API endpoint for efficiently querying warehouse locations using their unique codes, offering a tailored solution for our React front-end app.

The `TransactionController` includes a custom action named `receiveItem`, designed to handle the receipt of items in the system. This method is specifically tailored to the business logic of receiving items into the warehouse and is invoked through the `/receive-item` API route.

Upon receiving a request, the method extracts the `item_id` and `warehouse_location_id` from the request payload, representing the item being received and the target warehouse location, respectively. It then creates a new `Transaction` record using the `Transaction::create` method, initializing key attributes such as the `type` (set to 'received'), `quantity` (set to 1), and a comment indicating that the transaction was created via the API.

The method responds with a JSON representation of the created transaction and an HTTP status code of 201, indicating the successful creation of a resource. This custom action streamlines the process of recording item receipts through a dedicated API endpoint.

4.5. Developing front-end React application

React JS is a JavaScript library for building user interfaces. It lets us create reusable components that manage their own state, making it easier to develop complex and dynamic web applications. Key features of React JS include:

- Component-based architecture: Applications are built from reusable components, which improves code maintainability.
- Virtual DOM: React uses a virtual DOM to efficiently update the real DOM, improving performance.
- JSX: JSX is a syntax extension that allows you to write HTML-like structures within JavaScript code, making it easier to write and understand React components.

- Unidirectional data flow: Data flows in one direction in React applications, from parent to child components, making it easier to reason about how data changes.

Overall, React JS is a powerful library for building user interfaces. It is known for its ease of use, performance, and flexibility which makes it good option for us to implement the front-end UI/UX.

To use React, we need to install it into our Laravel project:

```
npm install react react-dom
```

Full list of React components and their source code are located at /resources/js folder. For navigation purposes, we are using React router:

```
npm install react-router
```

React Router handles navigation in our React app, letting us seamlessly transition between different "pages" (components) based on URL changes. It manages the URL state and renders the appropriate component for each route, providing a smooth user experience and easy navigation structure.

For calling Laravel API endpoints, JavaScript Fetch API is used. The JavaScript fetch function simplifies fetching data from servers. It returns a promise that resolves with the response object. We can then extract the JSON data and handle it based on your needs. It offers a cleaner and more modern approach compared to older methods like XMLHttpRequest.

Here's the final user interface:

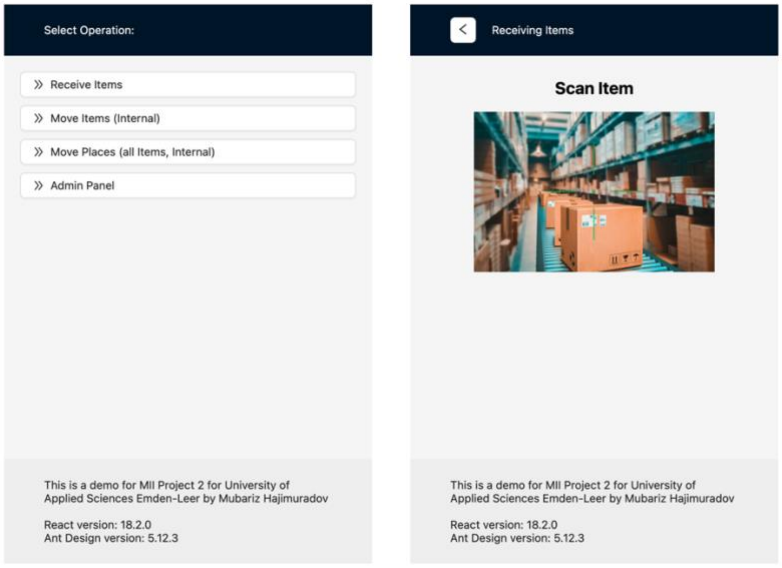


Figure 13. Developed React User Interface

5. Evaluation, Limitations and Summary

This evaluation assessed the basic functionalities and performance of the proposed warehouse management system within a controlled local environment. A Laravel application with a locally installed MySQL database served as the foundation, seeded with simplified warehouse structure data, items, and users. User interactions were simulated through the React front-end application, where scanning of item and location QR codes mimicked the receiving and put-away processes.

Encouragingly, all tested functionalities operated as expected, indicating successful integration between the React front-end and the Laravel back-end through the API. Additionally, API response times ranged between 0.4 and 0.7 seconds, suggesting efficient communication and potential performance benefits.

5.1. What's Done in This Project

This project focused on optimizing warehouse operations by implementing efficient receiving and put-away processes through a web application. Here's a breakdown of the key achievements:

- Utilized Laravel and Filament for a robust and efficient back-end infrastructure with functionalities like data management and API development.
- Implemented React.js for a user-friendly front-end interface, enabling seamless interaction with receiving and put-away processes.
- Leveraged MySQL for data storage, ensuring reliable and scalable data management.
- Developed a user-friendly interface for scanning item barcodes, likely through a mobile app or web platform built with React.js.
- Integrated scanning data with the back-end through the /receive-item API endpoint, facilitating item identification and location selection for efficient receiving.
- Implemented data validation and verification steps to ensure data accuracy during receiving.
- Enabled location identification, potentially through code scanning using the /warehouselocation-by-code/{id} API endpoint.
- Connected put-away actions with inventory updates and transaction creation via the /transactions API, ensuring real-time inventory management.
- Achieved API-based data flow, enabling potential scalability and integration with existing warehouse systems.
- Provided a fast and responsive user experience with server response times ranging from 0.4 to 0.7 seconds.

- Gained positive user feedback on the intuitive "scan-scan-done" approach for streamlined receiving and put-away.

5.2. Limitations and Future Directions

5.2.1. Limitations

The initial evaluation of the proposed solution yielded encouraging results, but several limitations necessitate further investigation prior to real-world implementation. Firstly, the testing environment was entirely simulated, and the user base was restricted, limiting the generalizability of the findings. Validation in an actual operational setting with a more representative user group is essential for robust assessment. Secondly, the absence of specific metrics such as error rates and time improvements hampers a quantitative analysis of potential efficiency gains. Collecting such data in a real-world scenario is crucial to accurately evaluate the solution's impact on operational productivity. Furthermore, the implementation's focus solely on the Receiving and Put-away processes restricts its scope. This may not adequately address advanced functionalities like multi-item receiving or complex location management, which are frequently encountered in intricate warehouse operations. Expanding the implementation to encompass these facets is necessary for broader applicability and to cater to diverse warehouse needs. Addressing these limitations through real-world testing, comprehensive data collection, and scope expansion will provide a more accurate picture of the solution's effectiveness and its readiness for wider adoption within the logistics domain.

5.2.2. Future Directions

- Conduct a better evaluation in a real-world warehouse environment, collecting data on metrics like receiving/put-away time, error rates, and inventory accuracy.
- Integrate the system with existing warehouse infrastructure for seamless data exchange and automated processes.
- Expand functionalities to include features like multi-item receiving, batch processing, and advanced location management.
- Develop reporting and analytics dashboards to provide insights into warehouse performance and optimize resource allocation.

By addressing these limitations and exploring future directions, this project has the potential to significantly improve warehouse efficiency, accuracy, and overall logistics management.

Bibliography

[1] G. Richards, Warehouse Management A Complete Guide to Improving Efficiency and Minimizing Costs in the Modern Warehouse, Kogan Page Limited, 2011, p. 59.

[2] D. &. R. A. D. V. Sai Kiran, Critical success factors of ERP implementation in SMEs, Journal of Project Management 4, 2019.

[3] G. Richards, Warehouse Management A Complete Guide to Improving Efficiency and Minimizing Costs in the Modern Warehouse, Kogan Page Limited, 2011, pp. 55-56.

[4] Statista, "Share of warehouse management software market worldwide in 2023, by vendor," [Online]. Available: https://www.statista.com/statistics/503241/worldwide-data-warehouse-management-software-market-share. [Accessed 23 12 2023].

[5] Agistix, "Supply Chain Visibility Software: In-House vs. Outsourced," [Online]. Available: https://www.agistix.com/blog/supply-chain-visibility-software-in-house-vs-outsourced-2/. [Accessed 23 12 2023].

[6] G. Richards, Warehouse Management A Complete Guide to Improving Efficiency and Minimizing Costs in the Modern Warehouse, Kogan Page Limited, 2011, p. 58.

[7] Y. L. a. L. Zeng, Research and application of the EAN-13 barcode recognition on iphone, 2010.

[8] G. Richards, Warehouse Management A Complete Guide to Improving Efficiency and Minimizing Costs in the Modern Warehouse, Kogan Page Limited, 2011, p. 60.