Universidade do Minho
Escola de Engenharia

Software Development Systems

Project Report

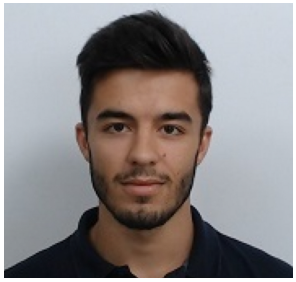
Group 21

University of Minho, Department of Informatics, 4710-057 Braga, Portugal

FILIPE FELÍCIO (A85983)

HENRIQUE RIBEIRO (A89582)

LUÍS ARAÚJO (A86772)



PAULO BARROS (A67639)

RUBEN ADÃO (A89545)

OCTOBER 2020

Table of Contents

List of Figures

List of Tables

1 Introduction

The following pages serve the purpose of in-depth reviewing the sophistication of our rigorous and methodical development process of a intelligent warehouse software system capable of dealing with all the inherent automated processes, whilst, through an elegant graphical user interface, giving functionalities to human users of the system, ranging from different types of **Administrators**, with different levels of permissions, to the **Truck Drivers** or **Floor Workers**.

With the key word on the last paragraph being *sophistication*, this report shall sustain the thesis of our Analyses, Models and Reformulations being the most appropriate and accurate, leading to valid and reliable software prototypes through the models' correlations to the object oriented programming paradigm, in a iterative process cycling through all this phases, culminating in a proper software system design, taking full potential of Class Hierarchy, Abstraction and Interface Implementation in a DRY(*Don't repeat yourself*) approach, in a MVC architecture.

In order to achieve peak performance in a multi-variable set of goals, such as maintainability and scalability, this study will reflect the potential of our own approach to solve the given task.

2 Proposed Project Description

The purpose of this project is to model and develop a warehouse management system of a factory warehouse.

First of all, it is expected that a **Truck** enters the warehouse with the authorization of a manager (**Admin**). When that authorization is confirmed, the **Truck** enters the warehouse and deposits all **Pallets** on a **Receiving Area** to a **Floor Worker**.

After that, the **Floor Worker** scans all **Pallets** and sends a signal that there are **Pallets** to be stored to the system. The system then calculates the best algorithm for the **Robots** to pick them up and store them, **Perishable Pallets** and **non Perishable Pallets** goes to the **Refrigerated shelves** and **non Refrigerated shelves**, respectively.

Simultaneously, a request for a delivery can be made, and for that the system has to locate and check the availability of the **Products**, once more the **Robots** will receive a signal to pick and deliver the **Products** to a **Receiving Area**. When there job is finished a notification is sent, so the package can be picked up.

Receiving an in depth overview of the warehouse, the **Admin** will be capable to oversee all the warehouse information, to properly make decisions and manage the system, such as the location of all **Pallets**.

3 Warehouse Terminology

The following are some of the most common warehouse terms. Learning what these terms mean will make it easier to understand and communicate with the client and likewise easier to develop with the correct denominations about the warehouse and the supply chain.

3.1 3PL

Stands for third party logistics, which is a business's use of an outside company to manage a warehouse or group of warehouses.

3.2 Advanced Shipping Notification

A document that is sent to a warehouse management system from a supplier that provides information about a pending shipment.

3.3 Barcode

A marking made up of a series of bars and spaces used for identification of products in which a scanner is used to read the encoded information.

3.4 Batch picking

A process of order picking in which all the items for multiple orders is picked by a single picker.

3.5 Bill of Lading

A document that details items in a shipment that acts as a receipt given by the carrier of the shipment to the recipient of the shipment.

3.6 Cross Docking

Refers to the practice of moving products directly from the receiving area to the shipping area for distribution rather than being put away and stored for a period of time.

3.7 Discrete picking

A process of order picking in which the picker pick all the items for one particular order.

3.8 Line

The products in an order that share the same SKU or UPC number.

3.9 Order

All the products that are including on one transaction from a customer.

3.10 Purchase Order

A purchase order is a document that is sent from a buyer to a supplier requesting an order for merchandise. The purchase order usually lists the type of item, quantity, and agreed-upon price.

3.11 Putaway

This refers to removing incoming orders from the location where it is received to the final storage area and recording the movement and identification of the storage location where it has been placed.

3.12 Receiving

The process involving the physical receipt of merchandise, its inspection for accuracy and to identify any damage, the determination of where the stock will be stored, delivery to that location, and the completion of receiving reporting.

3.13 Replenishment

The process that involves moving stock from a secondary storage area to a fixed storage location. This could also refer to the process of moving stock between distribution centers or from suppliers to meet customer demand.

3.14 QR Code

A Quick Response (QR) code is a scannable code, made up of various black and white squares, that allows cameras or smartphones to read it and take the user to a stored URL or other information.

3.15 Unit

One particular physical item or product.

3.16 WMS (warehouse management system)

The software solution that keeps track of all warehouse operations including receiving, putaway, picking, shipping, and inventory.

3.17 Zone picking

A process of order picking in which different pickers pick items of an order from specific assigned storage areas to be assembled for shipping later.

4 Domain Model

4.1 First Version

The Business Model describes the entities of the context (habitat) in which the system should be implemented. A model with such focus is important to establish some ground rules about the system's objects, how they will work, and the way the users see the world of the entities that will populate the system, as well.[1].

After a thorough analysis, exploring the requirements of the system, we started by identifying its main entities, trying to understand the way they interact between them and their place in the overall structure of the program, so we can start to draw the model.

With that intent, we made the first draft of our Domain Model (**Fig. 1**) where the main entities of the system are: **Robot, Pallet, Perishable Pallet, Shelf, Refrigerated Shelf, QR Code, Floor Worker, Receiving Area, Shipping Area, Trucks and Admin**.

The **Admin** controls the flow of incomes and outcomes of **Trucks** into the Warehouse, allowing the **Truck**, who ask entry permission through system, to unload pallets in the **Receiving Area**. A **Pallet** is then processed into the system by a, **Floor Worker**, there by reading its **QR Code**. Then, automatically the system allocates one **Robot** of the Warehouse's fleet to transport the **Pallet** according to the system's calculated route. This can mean Cross Docking, transporting to a free Storage Unit for later shipment or to another **Robot** (which is a intermediate step to achieve one of the previous two). In the second option, the system given route leads the Robot to an available **Shelf** of the Warehouse (that can store multiple **Pallets**), this can be a normal one or a **Refrigerated Shelf** used to store **Perishable Pallets**. In the first option or in later stage, a **Truck** at the Shipping Area request a set of **Pallets** and if **Admin** approves it a inverse process occurs and a the Shipment is fulfilled.

The multiplicities were added according to this description of the system.

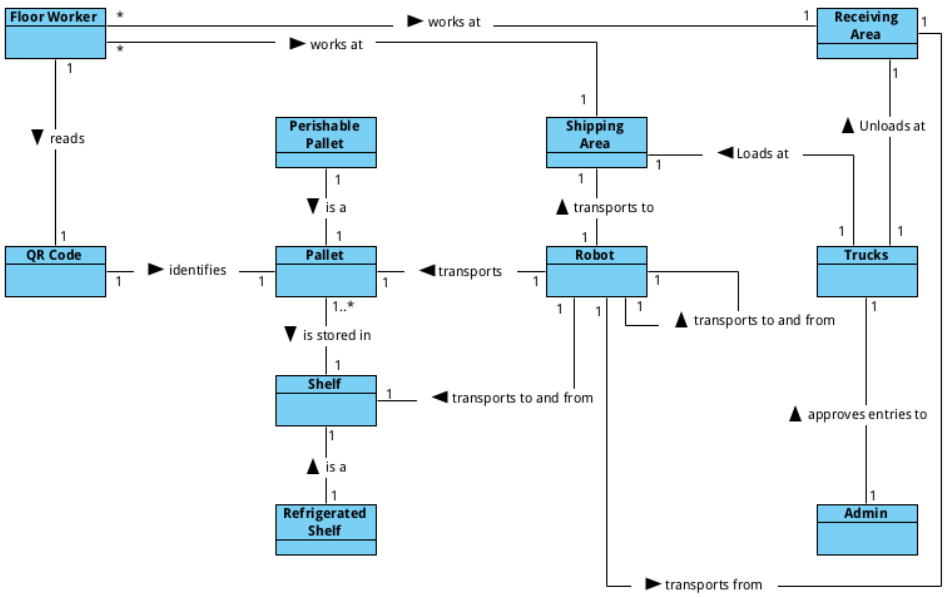


Fig. 1: First Draft of the system's Domain Model.

4.2 Second Version

A further analysis revealed that, even though the model was a good base to understand the system, it was lacking in terms of representing all the entities the system has and their relationships. The major addition was an **Order Server** that issues **Orders** that are processed by the system.

Other important update on was on the **Pallets**, now they are composed of several units of the same **Product**, the information on the overall quantities of one product in the Warehouse (**Stock**) are also a new entity, extremely important to the **Order** fulfilment side of the system.

To keep within the logic of **Pallet** extension in the system, we were inclined to add a **Perishable Product** to the model. Continuing with this approach of interactive and rigorous analyses/modeling we developed a new draft of the business model **(Fig. 2)**.

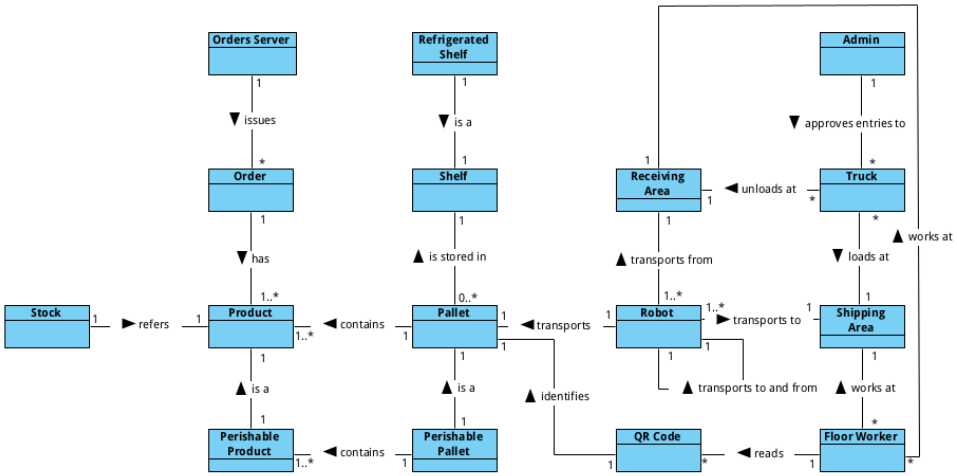


Fig. 2: Second Draft of the system's Domain Model.

4.3 Third Version

In a third iteration, we felt the need to add even more complexity and accuracy to the model, firstly, by integrating a **Status** entity that refers to a **Pallet** information, essential so a **Routing Algorithm** can calculate a **Route** for a **Robot** (we implemented its **Location** as well do to being the main parameter of this function).

In addition, a **QR Code Reader** was a necessary update to better describe the adjacent behaviour.

Finally, all the process of attributing a **Job** to the **Robot** is now modeled through including a **Robot Queue** that essentially is a pool from where Jobs are attributed to **Robots** by the **Routing Algorithm**.

We implemented this changes in the following model draft (**Fig. 3**).

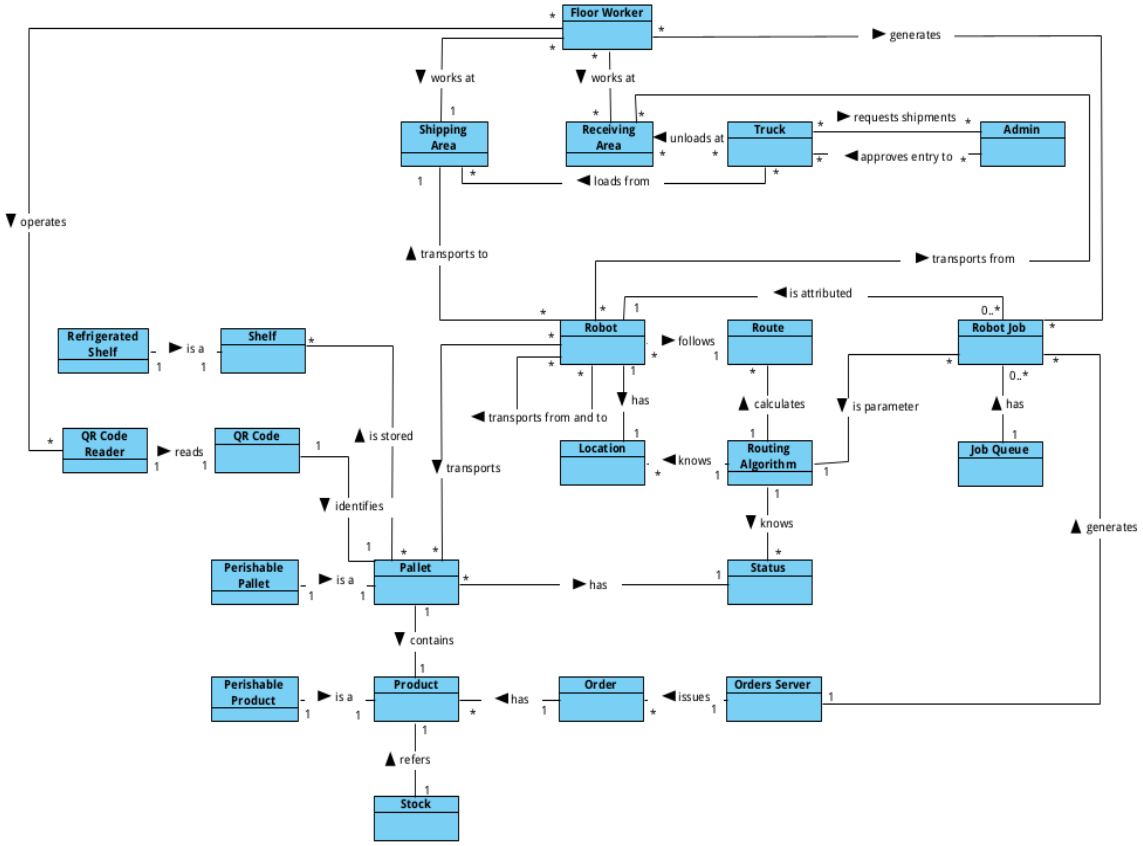


Fig. 3: Third Draft of the system's Domain Model.

5 Use Cases Model

5.1 First Version

Use cases represent distinct pieces of functionality for a system, a component, or even a class. [2]

The team identified 6 actors, from which 4 are human actors. This distinction is necessary due to the need of a **Use Case Authentication** that must be ran before any other.

The **Truck Driver** can only interact with the system for the single case of asking for a discharge of material. The Administrator can then choose to accept or deny the order, the use case being **Authorize discharge order**.

The Administrator must also have a quick way to identify and look through the different pallets stored in the warehouse, they can then ask the system to **List the pallets' locations**.

The **Production Server** has the ability to **Request material from system**, storing an order in the system, which will take the necessary steps to deliver the request as quickly as possible.

Finally the **Floor Worker's** main functionality is to **Notify request satisfaction** once the material requested is ready to be delivered to a **Truck** and removed consequently from the system.

As for the non-human actors, the **QR Code Readers** and the **Robots** will have their own association with the warehouse system. The **QR Code Reader** sends to the system the information carried by every pallet that enters in the warehouse, that being what **Product** is being stored and in what amount.

When it comes to the **Robot**, responsible for moving **Pallets** around the building, it is necessary for the system to know its availability.

Therefore a robot has two important use cases: **Notify pallet picking** and **Notify successful pallet delivery**. This way the system gets to know every time a **Robot** picks up or drops off a **Pallet**.

There is also an unidirectional Use Case that acts upon the **Robot**, which is the ability of the system to **"Notify Robot to transport pallet"**.

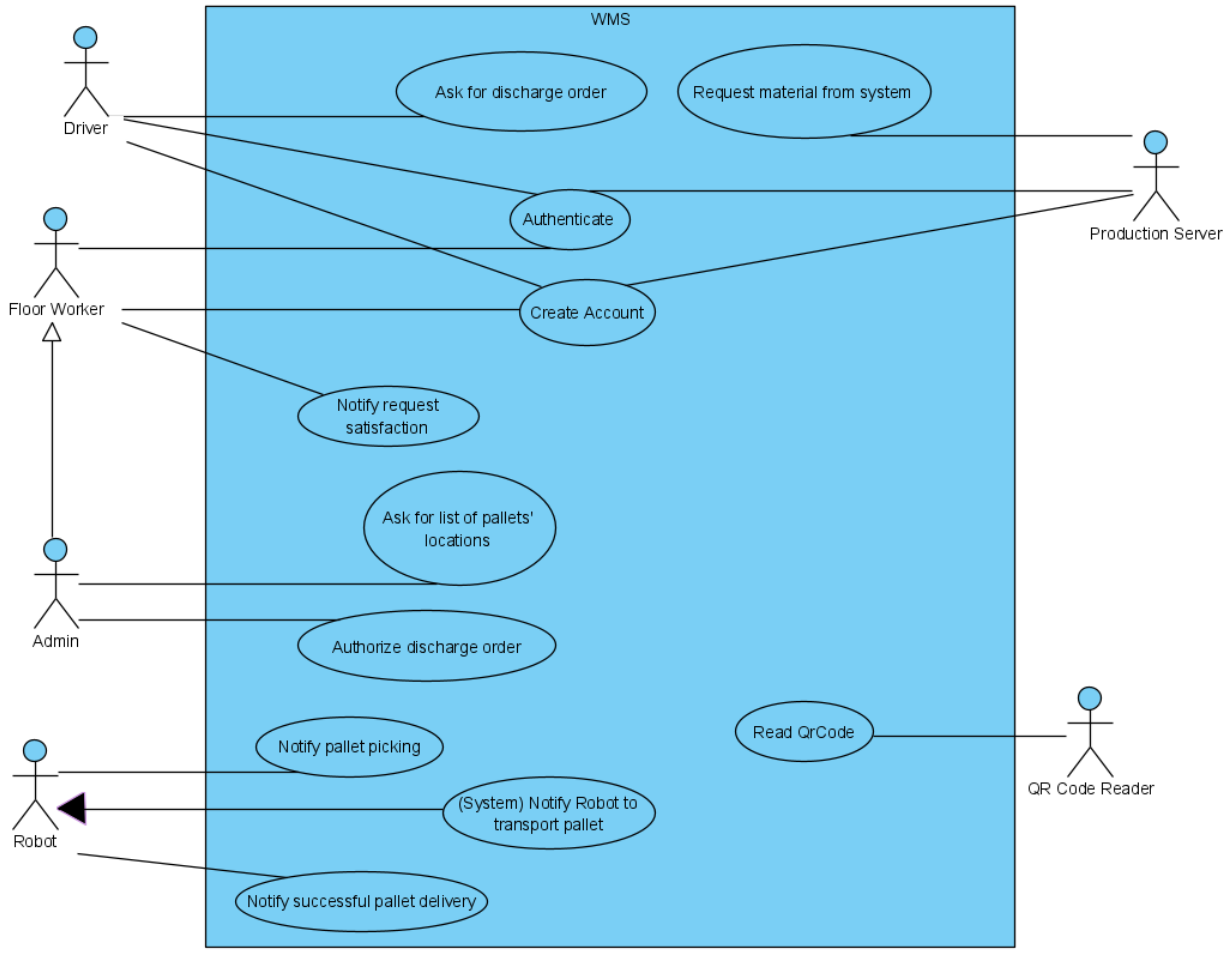


Fig. 4: First Draft of the system's Use Case Model.

5.2 Second Version

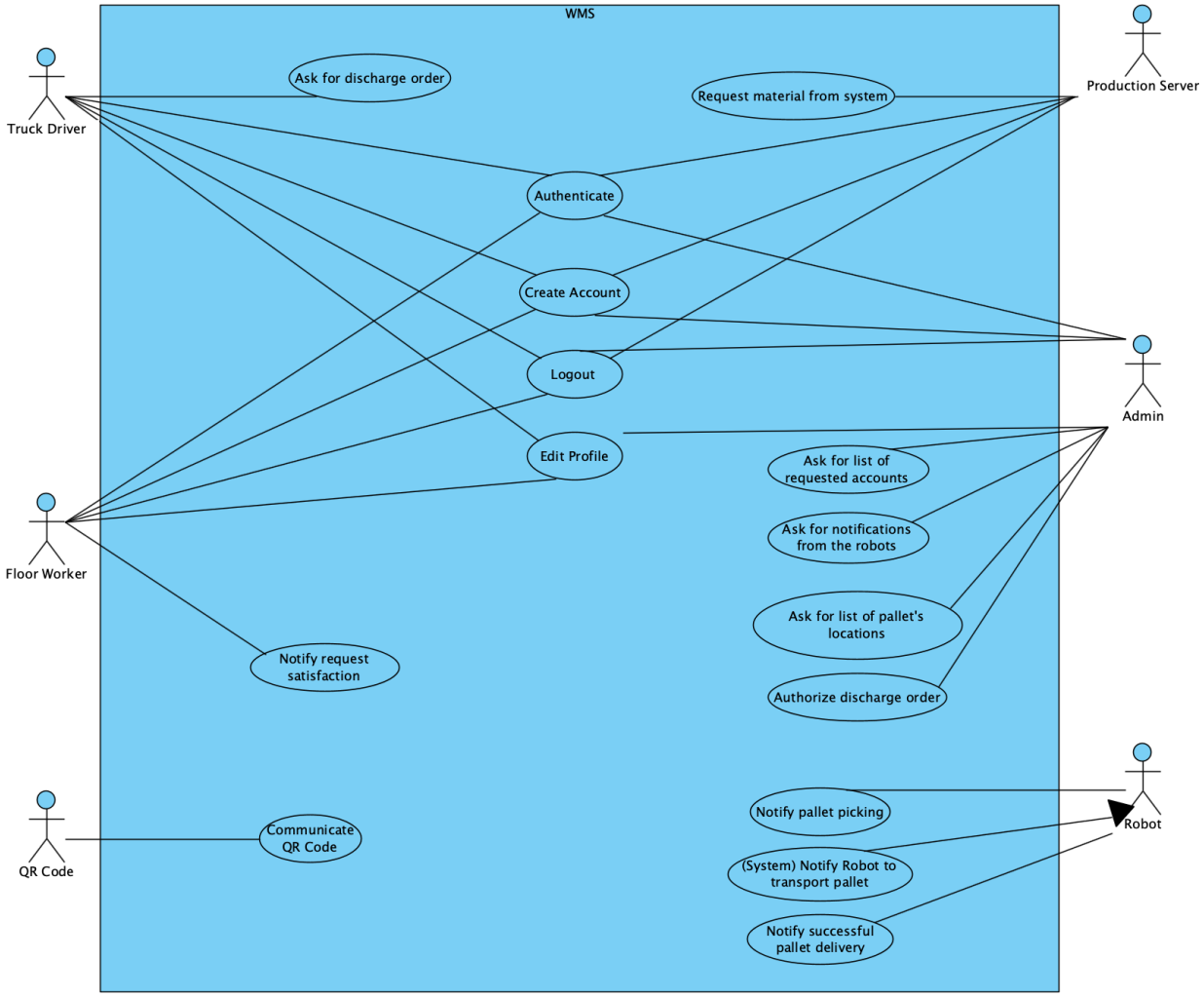


Fig. 5: Second Draft of the system's Use Case Model.

6 Use Cases Specifications

6.1 Create an account

The warehouse worker creates an account, granting him the appropriate permissions to work at the warehouse.

Use Case		
Use Case	Create account	
Actor	Driver, Admin, Floor Worker and Production Server	
Precondition	True	
Post-condition	Actor has an account created.	
	Actor	System
Regular flow	1. Actor inserts credentials through which he can access his account	
		2. Account is created.

Table 1: Use Case: Create an account Specification

6.2 Authenticate

The workers of the warehouse, namely the **Admin** and the **Floor Workers** and the **Truck Drivers** must log in the system in order to use its functionalities. The worker may cancel this process, in which case he will not have access to his account and the warehouse.

Table 2: Use Case: Authentication Specification

Use Case		
Use Case	Authenticate	
Actor	Driver, Admin, Floor Worker and Production Server	
Precondition	True	
Post-condition	Actor is authenticated.	
	Actor	system
Regular flow	Actor requests authentication.	
	Actor inserts his credentials.	System grants log out request.
	.	System verifies actors' credentials
Exception flow [Credentials do not exi st] (step 2)		(2.1) Login is failed
Exception flow [Credentials do not match] (step 3)		(3.1) Login is failed

6.3 Ask for discharge order

A **Truck Driver** can request a discharge order of material to be executed at a given date and time. Consequently, the system creates and keeps a record of this new request.

Use Case		
Use Case	Ask for discharge order	
Actor	Driver	
Precondition	True	
Post-condition	The discharge request gets registered in the system.	
	Actor	System
Regular Flow	1.The Driver request a material discharge at a particular date and time	
		2. System creates a request order record

Table 3: Use Case: Ask for discharge order Specification

6.4 Authorize discharge order

The warehouse's **Administrator**, as he confirms there is a **Truck** at the entrance, will have to verify the existence of a discharge request in the system. If the request does exist, the **Truck's Pallets** are taken to the **Receiving Area**, where they will wait for their **QR Codes** to be read. Otherwise if the system can't find such request, the **Administrator** cannot authorize the discharge and the driver must evacuate.

Use Case		
Use Case:	Authorize discharge order	
Actor	Administrator	
Precondition	There is a truck at the entrance.	
Post-condition	The discharged pallets get the status äwaiting to be read"	
	Actor	System
Regular Flow	1. Administrator verifies if the discharge order is in the system	
	2. Administrator authorizes the discharge	
		3. Pallets get the status "waiting to be read"
Alternative Flow 1 [Order not present in system] (step 2):	2.1. The Administrator doesn't authorize the discharge	

Table 4: Use Case: Authorize discharge order Specification

6.5 Notify pallet picking

When the system asks for a **Robot** to pick a pallet, first it identifies if a pallet is indeed in the targeted location. If so, the **Robot** picks it up and notifies the system of it. Otherwise, the **Robot** will also notify that a **Pallet** is missing.

Use Case		
Use Case:	Notify pallet picking	
Actor:	Robot	
Precondition	Robot has a "pick pallet" request from system and is currently at the right location	
Post-condition	System obtains the information that Robot is carrying a pallet, whose status change to "in transportation"	
	Actor	System
Regular Flow	1. Robot verifies if the pallet exists	
	2. Robot picks the pallet and notifies the system	
Exception Flow [Pallet not found] (step 1)	1.1 Robot notifies the system of the missing pallet	

Table 5: Use Case: Notify pallet picking Specification

6.6 Notify successful pallet delivery

In case a **Robot** is moving a **Pallet** and has arrived at the drop off destination, first it will verify if that area is empty. If so, the **Pallet** is dropped and the system is notified of the successful delivery. If the drop off area is not empty, the **Robot** notifies the system of that problem.

Use Case		
Use Case:	Notify successful pallet delivery	
Actor:	Robot	
Precondition	Robot is transporting a pallet	
Post-condition	Moved pallet is stored at the destination	
	Actor	System
Regular Flow	1. Robot verifies if destination area is free	
	2. Robot drops the pallet at the destination	
Exception Flow [Area not free] (step 1)	1.1 Robot notifies system that the destination area isn't empty	

Table 6: Use Case: Notify successful pallet delivery Specification

6.7 Notify request satisfaction

When the **Floor Worker** confirms that all the requested material is loaded, the **Order** leaves the warehouse and the **Floor Work** notifies the system that the order has been completed.

Use Case		
Use Case:	Notify request satisfaction	
Actor:	Floor Worker	
Precondition:	All the requested material is in the shipping area.	
Postcondition	The material leaves the warehouse	
	Actor	System
Regular Flow	1. The Floor Worker verifies that all material are loaded.	
	2. The order leaves the warehouse.	
	3. The supervisor notifies the system that the request is completed.	

Table 7: Use Case: Notify request satisfaction

6.8 Notify Robot to transport pallet

As soon as the system receives the information that a **Pallet** needs to be distributed, it chooses a **Robot** and calculates the best **Route** for him. After that, the system notifies the chosen **Robot** to transport that the **Pallet**.

Use Case		
Use Case:	Notify Robot to transport pallet	
Actor:	System	
Precondition:	There is a Robot available for transportation.	
Post-condition	Robot is notified to transport pallet.	
	Actor	System
Regular Flow		1. System receives the information that there is a pallet to be distributed.
		2. System chooses a Robot and calculates the best route.
		3. System notifies the Robot to transport the pallet.

Table 8: Use Case: Notify Robot to transport pallet

6.9 Register pallet request

The warehouse may receive a request from the **Order Server**. In this case, the system will register the request in the **Job Queue**, in which it will await completion by one of the warehouses' **Robots**. Should the warehouse be unable to complete this request due to a lack of resources, it will make no delivery and communicate this to the **Server**.

Use Case		
Use Case:	Register pallet request	
Actor:	Production Server	
Precondition:	True	
Postcondition	Pallets are registered in the delivery queue	
	Actor	System
Regular Flow	1.A list of pallets is requested by the Production Server.	
		2. System checks the availability of the requested pallets.
		3. Order is placed in the delivery queue.
Exception Flow [warehouse can't fulfil the request] (step 2)		2.1. System communicates impossibility of delivering requested resources

Table 9: Use Case: Register pallet request

6.10 Request listing of pallets.

The **admin** requests the location and information of all the **Pallets** in the system. Requires authentication beforehand.

Use Case		
Use Case	Request listing of pallets	
Actor	Admin	
Precondition	True	
Post-condition	List of pallets made available to the manager.	
	Actor	System
Regular flow	1. Admin requests listing of the pallets in the system	
		2. Transmit the required information (pallets and their location) to the Admin.

Table 10: Use Case: Request listing of pallets

6.11 Communicate QR Code

Pallet is identified and the **Job** of storing it in the warehouse is added to the **Job Queue**.

Use Case		
Use Case:	Read QR Code	
Actor:	QR Code Reader	
Precondition:	True	
Postcondition	Pallet is identified and added to the job queue.	
	Actor	System
Regular Flow	1. Pallet's QR Code is read.	
		2. Pallet is identified and registered in the system.

Table 11: Use Case: Read QR Code

7 Activity Diagram

7.1 Order

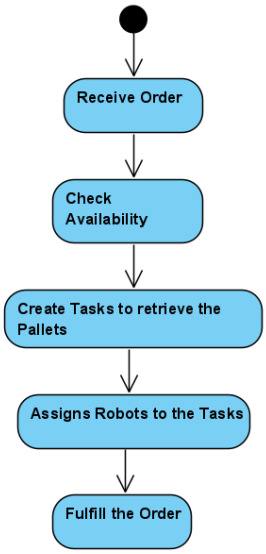


Fig. 6: Order Activity Diagram

8 Behavioral State Machines

In UML, state diagrams are important in that they allow to do a description of the objects' dynamic behavior. This information is not present in other diagrams direct and intuitively.[1]

8.1 Login State

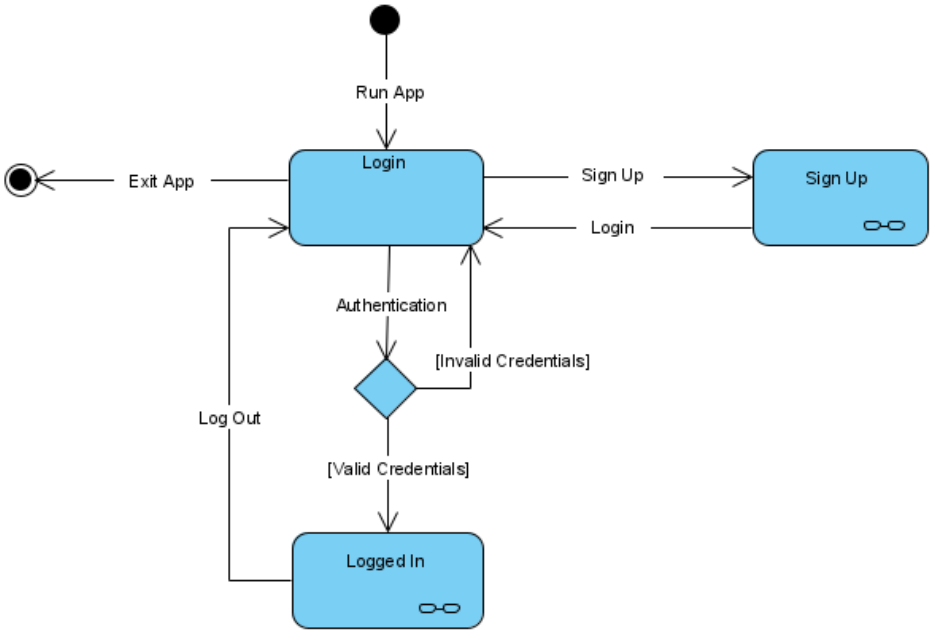


Fig. 7: State Machine of the System

The main states of the user interface has, as we can see in **Fig. 5**, are: **Log In**, **Sign Up** and **Logged In**. The transition that changes the interface from **Log In** to **Logged In** is **Authentication**, this transition is only valid if the credentials inserted are valid, otherwise, it stays in the first state. From **Logged In** we can return to **Log In** through the **Log Out** transition. We can interchange from and to the **Log In** and **Sign Up** states using the **Log In** and **Sign Up** transitions.

8.2 Sign Up User SubState

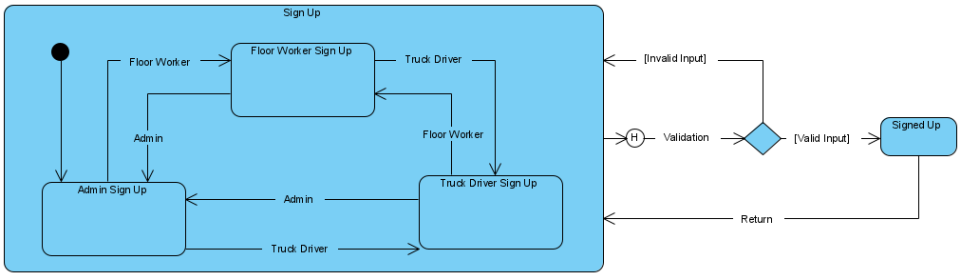


Fig. 8: Expansion of the Sign Up Substate

As we can see in **Fig. 7**, the **Sign Up** substate starts by default in the **Admin Sign Up** state. From there the user can alternate between two more states: **Truck Driver Sign Up** state and **Floor Worker Sign Up** state. From these three the user can create a new account request that an **Admin** later will have to approve. Accordingly to the state it will create a different type of account request. Before submitting the request, the system will check the validity of the input (whether the email is valid and does not exit in the system yet). If the input is we move forward to the **Signed In** state, if it not we return to the previous state. From the **Signed In** state we can return to the **Sign Up** substate.

8.3 Logged In Substate

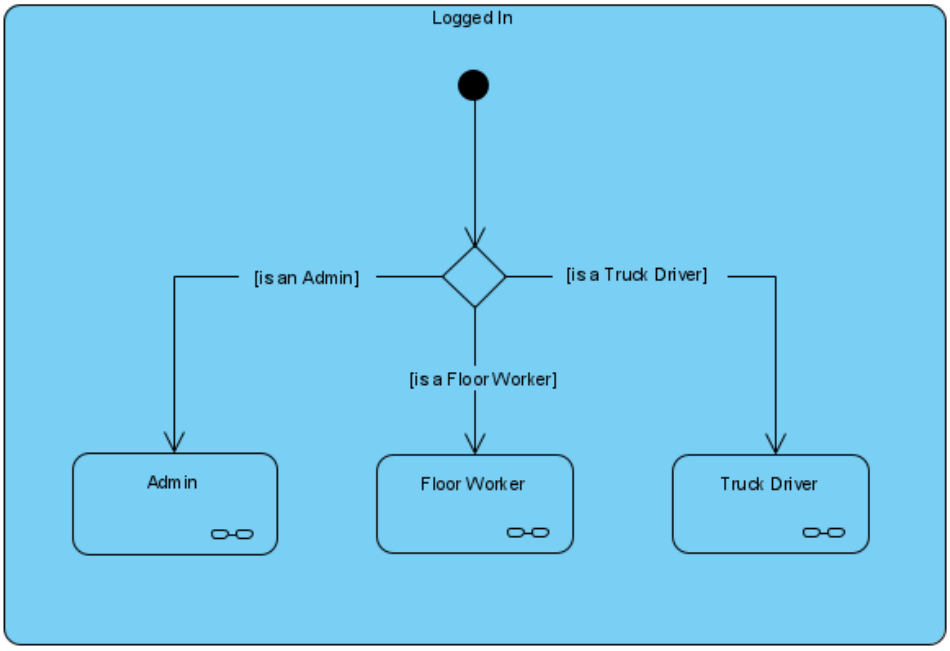


Fig. 9: Expansion of the Logged In Substate

Expanding the **Logged In** substate, that the user encounters once a user successfully logs in into the system, he then goes to one of three substates according to his type of user retrieved in the authentication process. This are: the **Admin Dashboard** substate, **Floor Worker** substate and **Truck Driver** substate.

8.4 Admin Substate

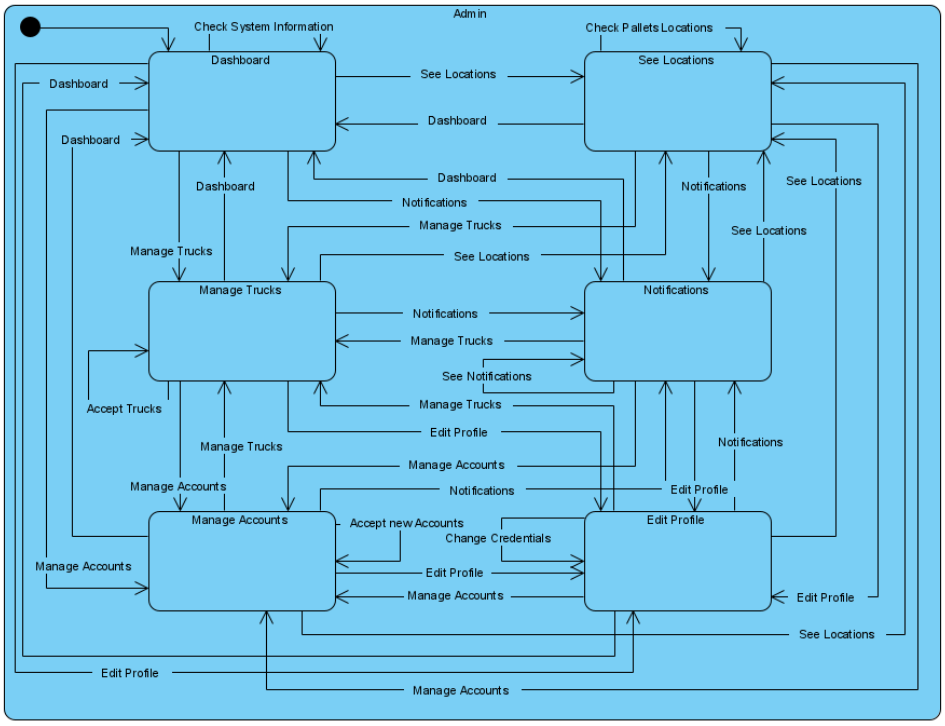


Fig. 10: Expansion of the Admin Substate

In the substate **Admin** (Fig. 12) we delve into the views where the **Admin** oversee all system by getting information and making decisions. By default he enters the **Dashboard** state, from there we can interchange between all the states in this substate, except the **Notifications** state that shows all the Robot's notifications on taking/dropping Pallets, the others are: **Trucks Manager** state (where he consult all the incoming request by **Trucks** and approve or deny them), **Locations** state (where he can check the locations of all the entities of the system), **Accounts Manager** state (where we can manage all the different users accounts in the system).

8.5 Floor Worker Substate

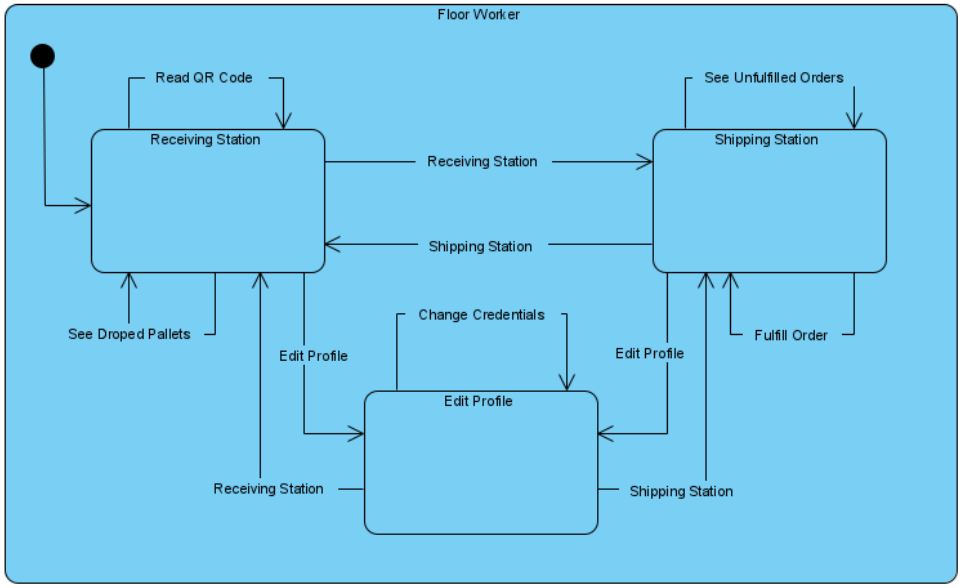


Fig. 11: Expansion of the Floor Worker Substate

In the **Floor Worker** substate we explore all the UI's where the **Floor Worker**, after successfully logged into the system, can interact and perform its duties in the system. By default he enters the **Receiving Station** State where he can see a List of all the **Pallets** that were dropped in the Warehouse and read them into the System. From there interchange places with two more States. He can move to the **Shipping Station** state through the **Shipping Station** transition and to the **Edit Profile** state through the **Edit Profile** transition. In the first, he can see all the **Orders** that were not yet shipped and once he the requested Pallets are in the **Shipping Station** ready for shipment he can assert their fulfillment. In the second, like the **Admin** he can update his system's credentials.

8.6 Truck Driver Substate

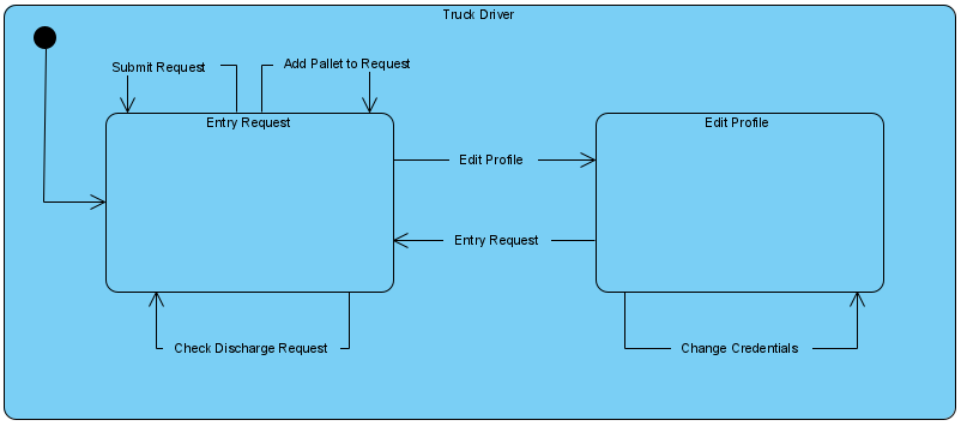


Fig. 12: Expansion of the Truck Driver Substate

Once the **Truck Driver** authenticates into the system, by default, he enters the **Entry Request** State, there he can add Pallets to the Warehouse Entry Request, he can check the request and submit it. From there we can interchange with a **Edit Profile** State (using two transitions with the same names). In the **Edit Profile** state the **Truck Driver** can update all his credentials in the system if their are valid.

9 Component Diagram

When modeling large software systems it is common to break the software into manageable subsystems. UML provides the component classifier for exactly this purpose. A component is a replaceable, executable piece of a larger system whose implementation details are hidden. [2]

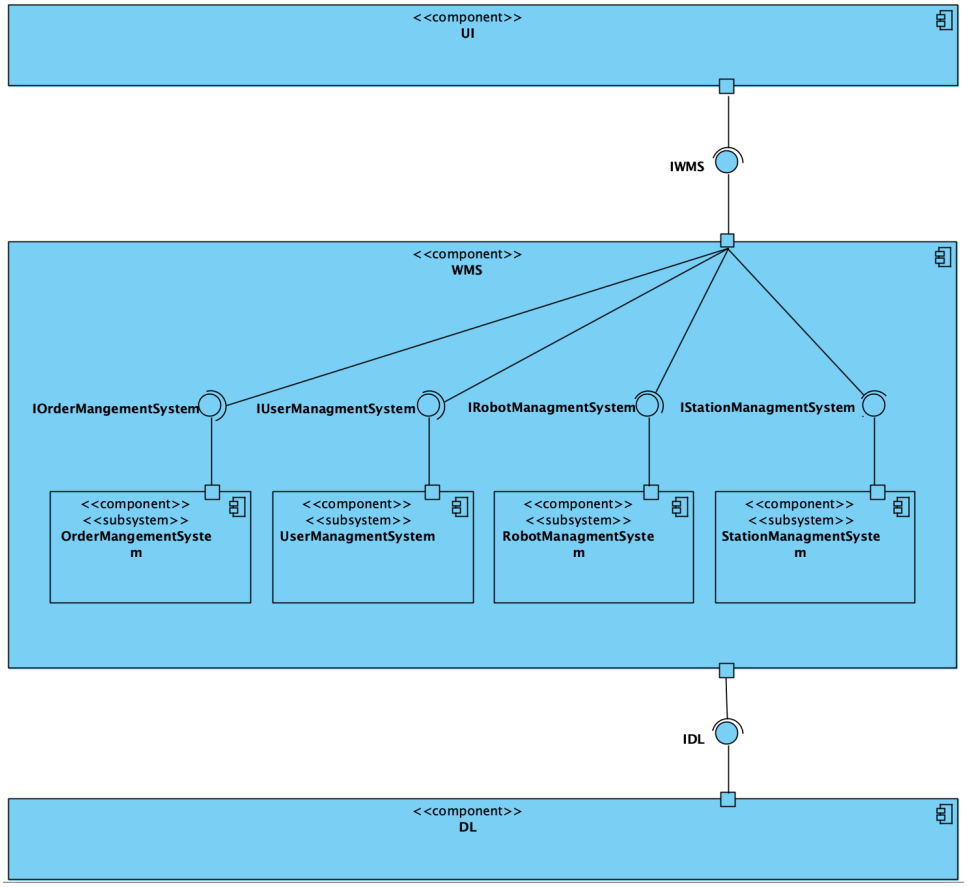


Fig. 13: Component Diagram

10 Sequence Diagram

10.1 Log In

Log In	Responsibilities	API	Subsystems
REGULAR FLOW:			
1. Actor requests authentication.			
2. Actor inserts his credentials.	Register actors' credentials.	verifyPassword(clazz : Class, id : String, password: String) : bool	UserManagementSystem
3. System verifies actors' credentials.	Verifies actors' credentials.	verifyPassword(id : String, password: String) : bool	UserManagementSystem
4. Actor is authenticated.			
Exception flow [Credentials do not exist] (step 2)			
2.1 Login is failed			
Exception flow [Credentials do not match] (step 3)			
3.1 Login is failed			

Fig. 14: Log In Sequence Diagram

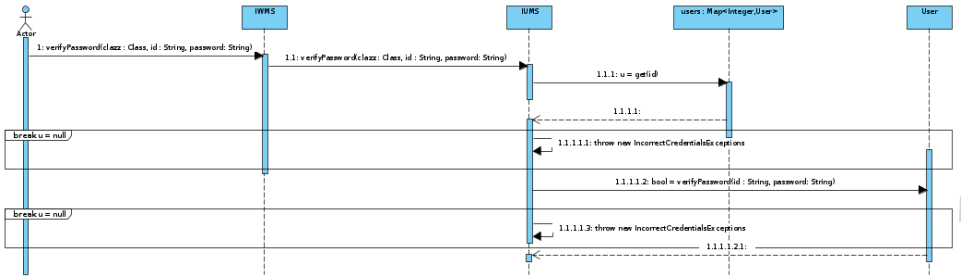


Fig. 15: Log In Sequence Diagram

10.2 Log out

Log Out	Responsibilities	API	Subsystems
REGULAR FLOW:			
1. Actor requests log out.			
2. System grants log out request.	Register actors' log out request.	loginHandler() ; void	Controller
3. System redirects actor to log in screen.	Redirect actor to different state.	redirectTo(state : String) : void	Controller

Fig. 16: Check Availability Sequence Diagram

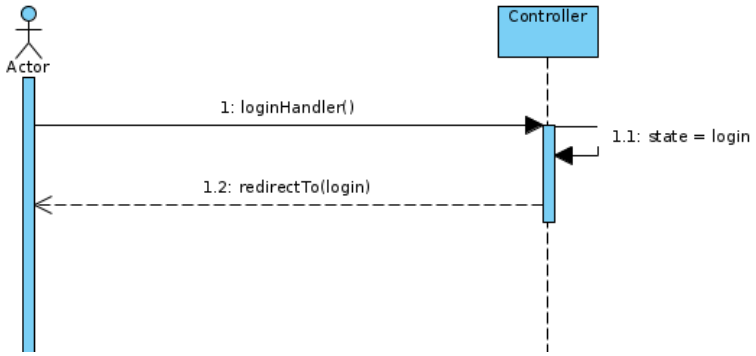


Fig. 17: Check Availability Sequence Diagram

10.3 Provide Pallet Information

Request listing of pallets	Responsibilities	API	Subsystems
REGULAR FLOW:			
1: Receive request by the manager.	Request list of pallets	RequestPallets() : Map <String,Station>	StationManagementSystem
2: Trasmit the required information (pallets and their location) to the admin.	Deliver pallet list	getPallestLocation() : Map <String,Station>	StationManagementSystem

Fig. 18: Provide Pallet Calculation Table

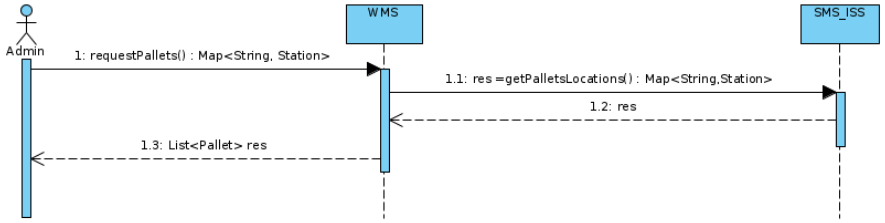


Fig. 19: Provide Pallet Information Sequence Diagram

10.4 Notify pallet picking

Notify pallet picking	Responsabilities	API	Subsystems
Regular Flow			
1. Robot verifies if the pallet exists.	Verifies that the pallet exists	verifyPallet(idPallet : String, idStation : String) : boolean	StationManagmentSystem
2. Robot picks the pallet and notifies the system.	Notify the system of pallet picking	notifyPalletPicking(idPallet) : void	StationManagmentSystem
Exception Flow [missing pallet] (step 1)			
1.1. Robot notifies the system of missing pallet.	Notify the system of missing pallet	cancelTransport(idPallet) : void	StationManagmentSystem

Fig. 20: Notify Pallet Picking Calculation Table

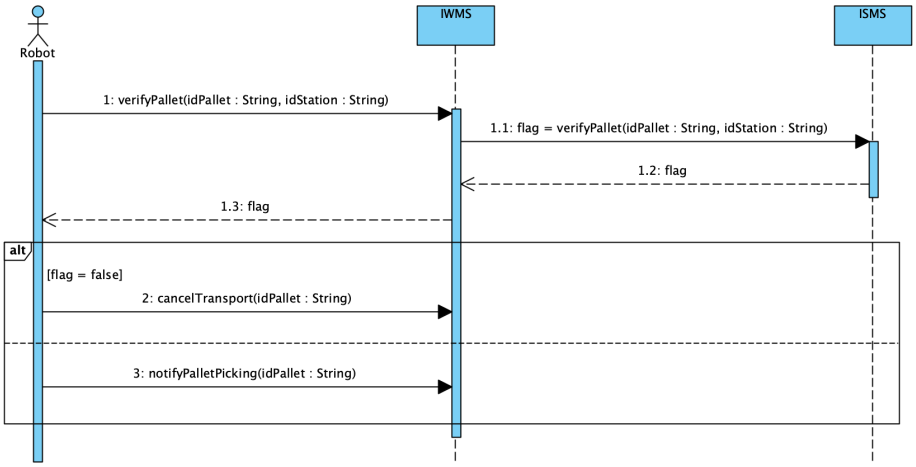


Fig. 21: Notify Pallet Picking Sequence Diagram

10.5 Notify Successful Pallet Delivery

Notify successful pallet delivery	Responsabilities	API	Subsystems
Regular Flow			
1. Robot verifies if destination area is free.	Verifies destination area	destinationAreaClear(idStation : String) : boolean	StationManagmentSystem
2. Robot drops the pallet and notifies the system.	Notifies the system of pallet delivery	notifyDelivery(idPallet : String) : void	StationManagmentSystem
Exception Flow [destination area isn't empty] (step 1)			
1.1. Robot doesn't deliver the pallet.			

Fig. 22: Notify Successful Pallet Delivery Calculation Table

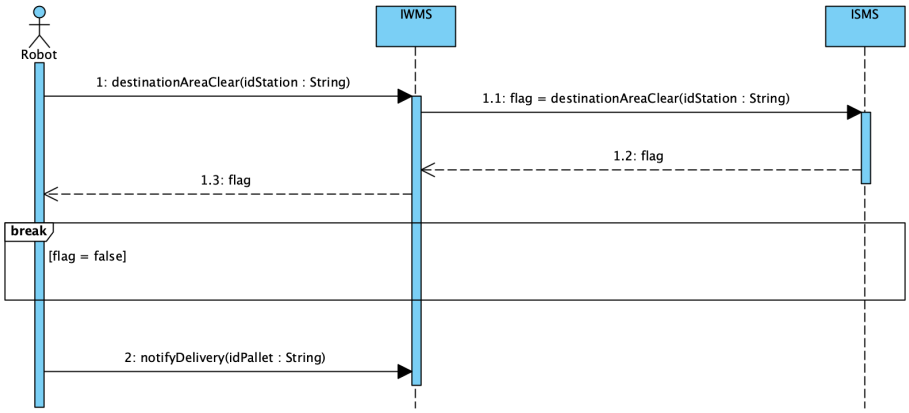


Fig. 23: Notify Successful Pallet Delivery Sequence Diagram

10.6 Register Pallet Request

Register pallet request	Responsabilities	API	Subsystem
Regular Flow:			
1. A list of pallets is requested by the Production Server.	Check pallet availabilty	checkAvailability(pallets : List<Pallet>) : boolean	SMS
2. System checks the availability of the requested pallets.			
3. Order is placed in the delivery queue.	Register Order in Queue	registerOrder(pallets : List<Pallet>) : void	OMS
Exception Flow [warehouse can't fulfil the request] (step 2)			
2.1. System communicates impossibility of delivering requested resource.	Cancel Request	cancelRequest() : void	OMS

Fig. 24: Register Pallet Calculation Table

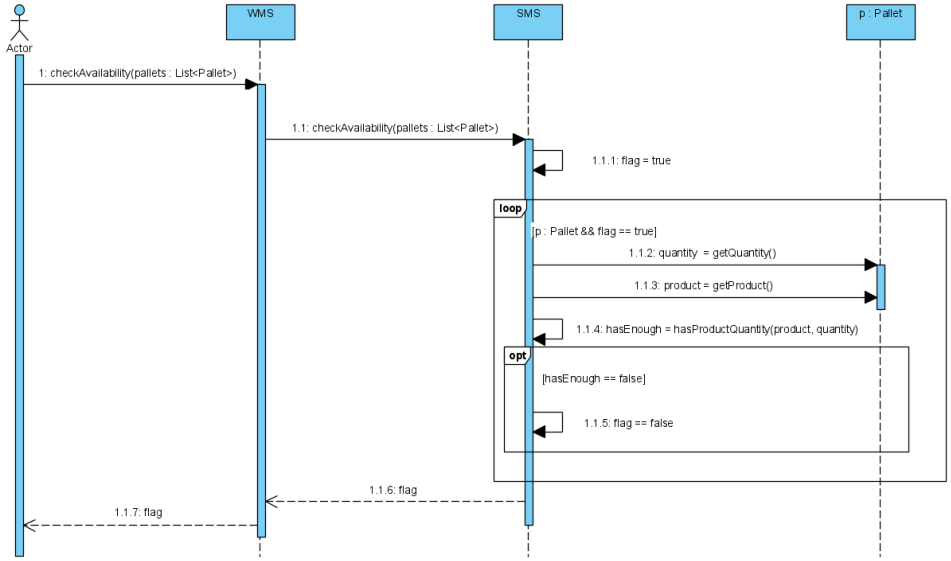


Fig. 25: Check Availability Calculation Table

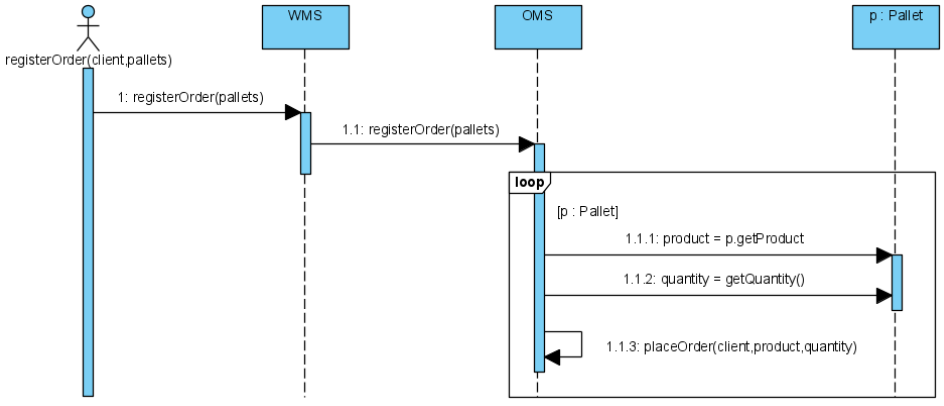


Fig. 26: Register Order Sequence Diagram

10.7 Notify Robot to transport pallet

Notify Robot to transport pallet	Responsabilities	API	Subsystem
Regular Flow:			
1. System receives the information that there is a pallet to be distributed.	Notify Robot	notifyRobot(idPallet : Pallet)	RMS
2. System chooses a Robot and calculates the best route.			
3. System notifies the Robot to transport the pallet			

Fig. 27: Notify Robot Calculation Table

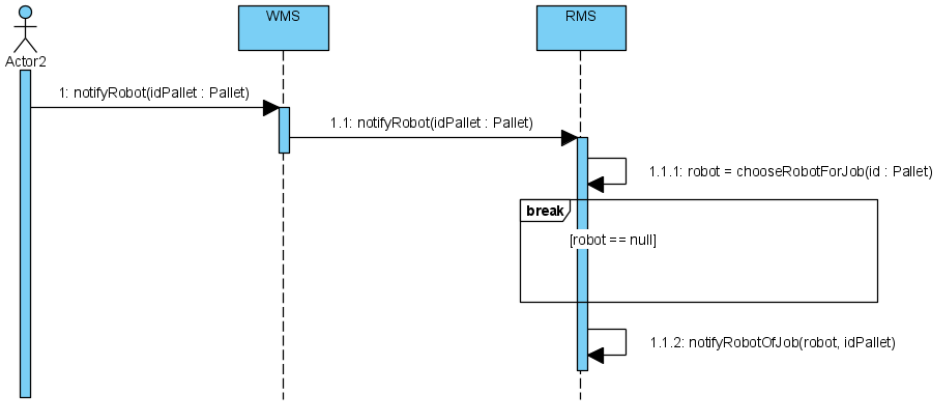


Fig. 28: Notify Robot Sequence Diagram

10.8 Communicate QR Code

Communicate QR Code	Responsabilities	API	Subsystem
Regular Flow:			
1. Pallet's QR Code is read	Register Pallet	registerPallet(product : Product, quantity : float)	SMS
2. Pallet is identified and registered in the system.			

Fig. 29: Notify Robot Calculation Table

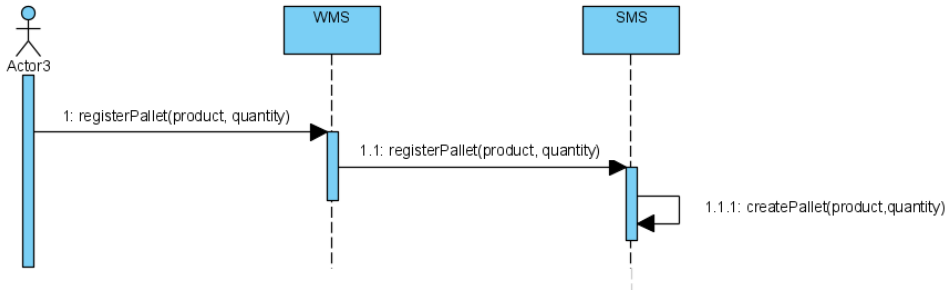


Fig. 30: Notify Robot Sequence Diagram

11 Class Diagram

When writing software you are constantly making design decisions: what classes hold references to other classes, which class "owns" some other class, and so on. Class diagrams provide a way to capture this "physical" structure of a system. [2]

11.1 Source Code

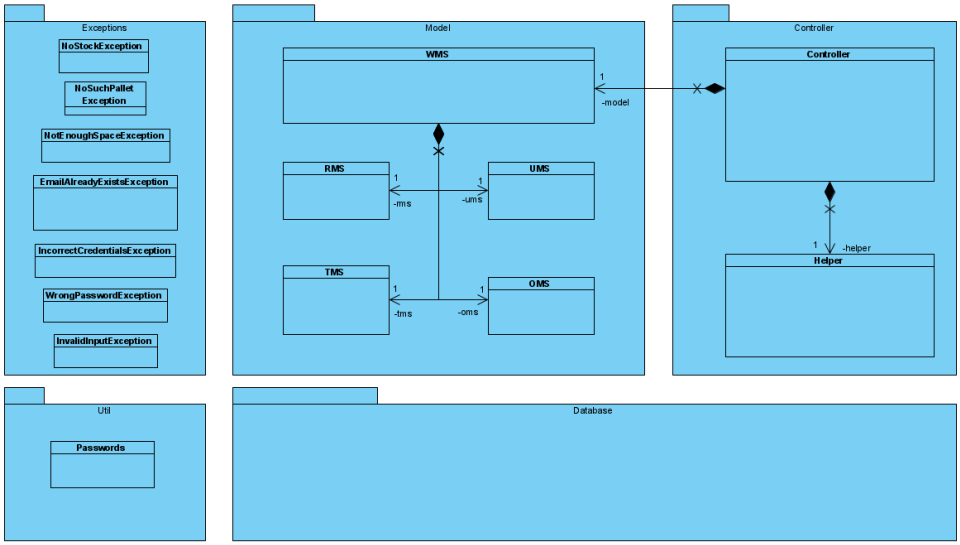


Fig. 31: Source Code Architecture

11.2 Model Package

Designing a sophisticated architecture of the system was a main goal from the start. Following a **MVC** (Model View Controller) one, we start with the **Model**.
First, we realized was that many elements from the initial **Domain Model** that we drawn from physical warehouse such as the **Shelves**, the **Robots**, the **Trucks**, the **Shipping and Receiving Stations** had many common features: they hold **Pallets**, they have a maximum capacity of **Pallets** and they have a location within the warehouse.
They also share a common functionality: The ability to take in **Pallets**. So, in accordance with a **DRY** approach, we propose an abstract **Superclass** called **Station** that all the elements refereed (that we coded into **Classes** as well) will extend, adding useful abstraction to the system. Second, we noted that from the **Station Subclasses** the **Truck** and the **Robot** had an additional functionality that they both performed: the capability to drop a **Pallet** in another **Station**. So we made them implement a **Transporter Interface** that has such **Method**.
Thirdly, we coded the systems' users (the ones we identified previously in other **Models/Diagrams** - the **Admin**, the **Truck Driver** and the **Floor Worker**), we abstracted their common traits into a **User SuperClass**.
Lastly, we had the need to add a unique **ID** to all these classes, in order to identify and organize (for in example fast search) all their individual instances, so we drawn a **SystemEntity SuperClass** extended by the previous two **SuperClasses**.
This strategic decisions enable us to construct intelligent and abstracts **Methods** throughout the system that leave us with **Scalable Software - one of the main goals of our Software Development Process, and accomplish the same set of functionalities with less and more readable code with a clean and intuitive hierarchy logic**.
With all these classes properly designed, we were then able to make a draft of **WMS** (Warehouse Management System) **Class** that supports all the proposed **Functionalities(Methods)** coded in the **Behavioural Modeling of the System**.
We made the first draft of the **Class Diagram** accordingly with what was enunciated.

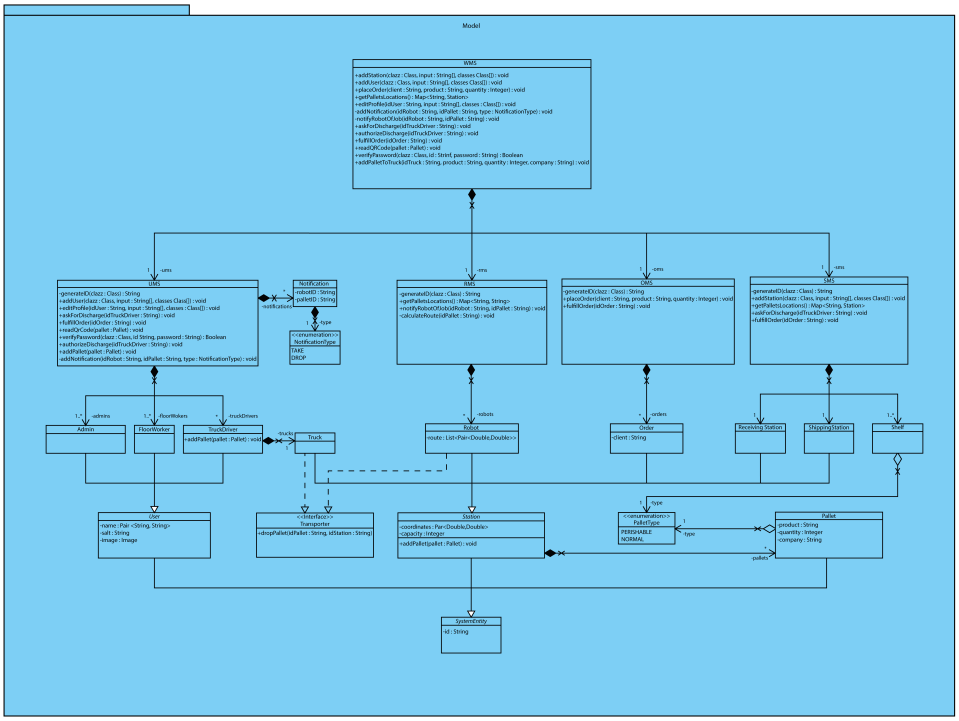


Fig. 32: Model's Class Diagram First Draft

11.3 Controller Package

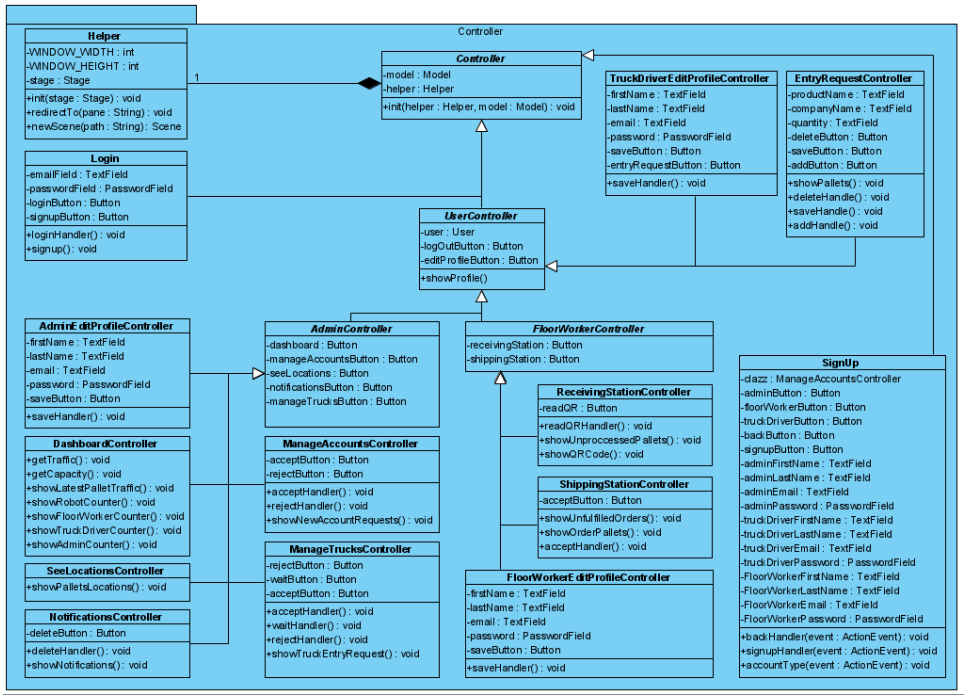


Fig. 33: Controller's Class Diagram First Draft

12 Revised Sequence Diagrams

As the project advanced to a new phase and was stating to grow, some chances had to be made in regards to some of our Sequence Diagrams. Most

12.1 Notify Pallet Picking

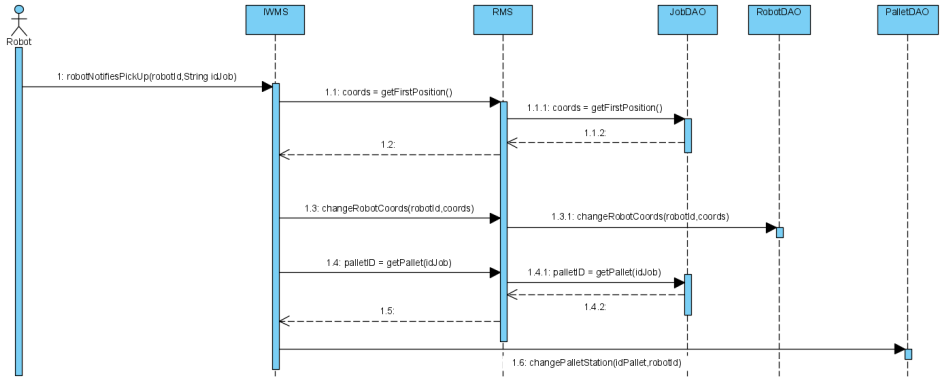


Fig. 34: Notify Pallet Picking

12.2 Notify Pallet Delivery

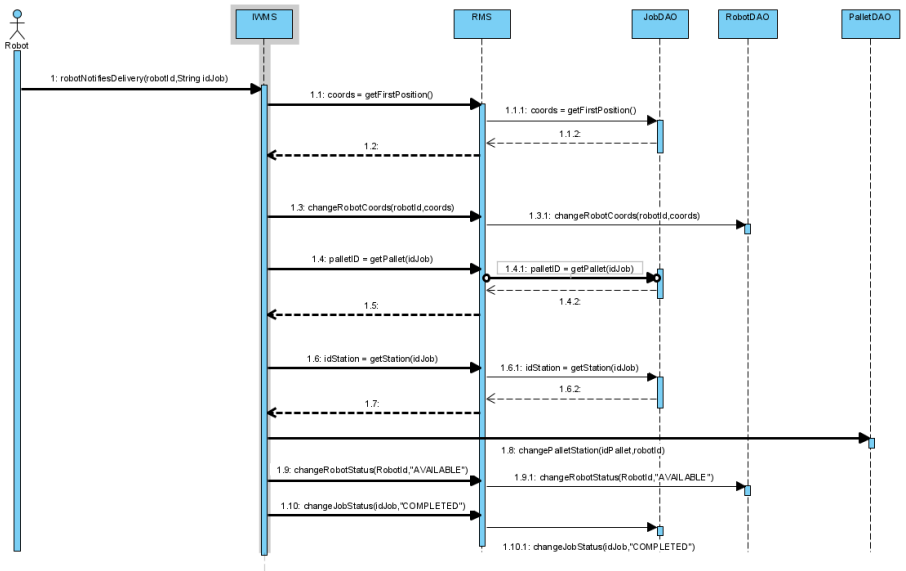


Fig. 35: Notify Pallet Delivery

12.3 Notify Robot to Transport Pallet

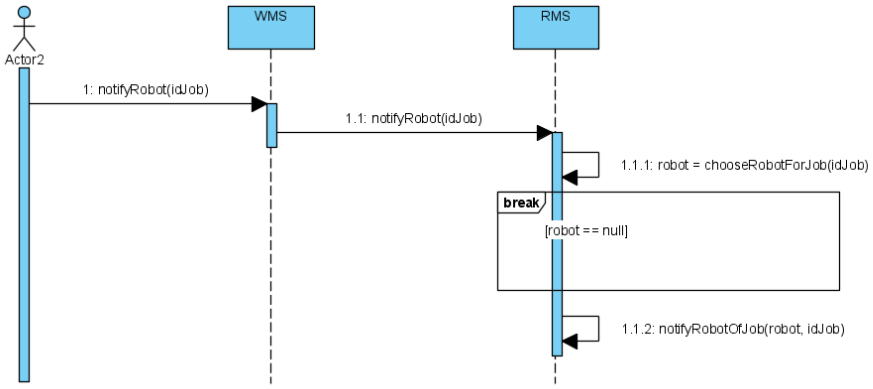


Fig. 36: Notify Robot to Transport Pallet

12.4 Provide Pallet Information

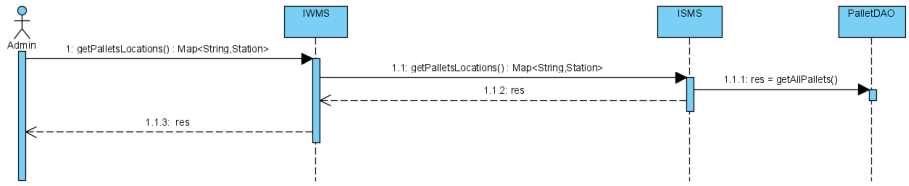


Fig. 37: Provide Pallet Information

13 Database Approach Model

To store all the information necessary for the application to run, the group decided to adopt a more dynamic approach into data management and went with an online relational database. Making use of a free hosting service, our database would allow remote access from various clients, which empowers an easier development and progress checking since anyone was always in tune with the progress and structure of the data used by the program.

13.1 Database Model

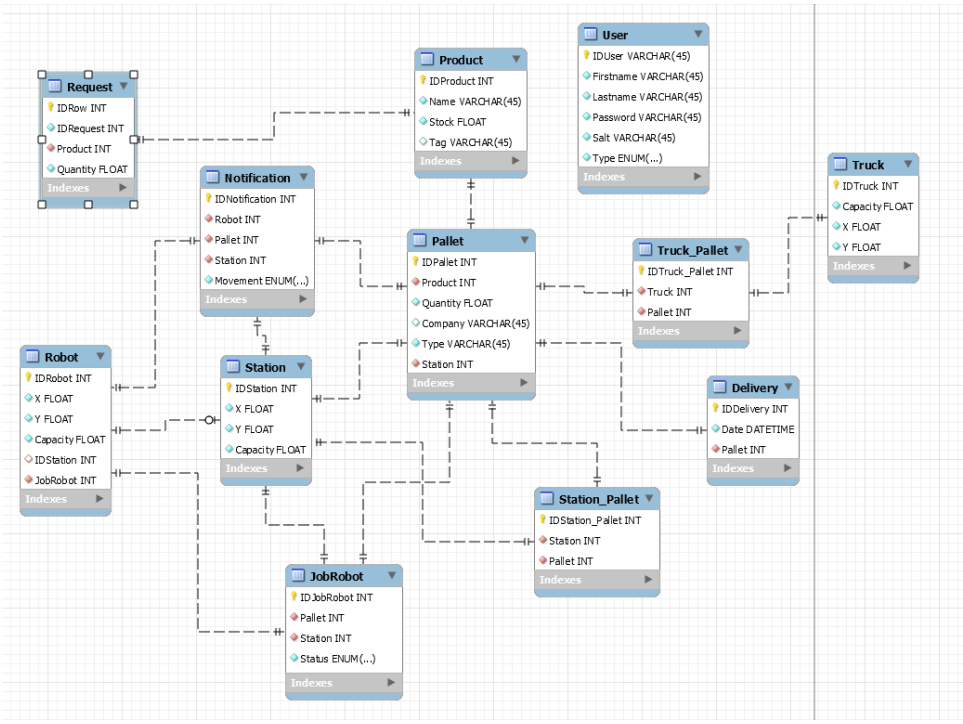


Fig. 38: Database Model created using MySQL Workbench

13.2 Online Hosting

To manage the online database, the hosting service provided a tool name *phpMyAdmin* which eases the process of managing a database. It allows to run SQL code directly form the browser and to manage the different tables with an easy-to-use user interface.

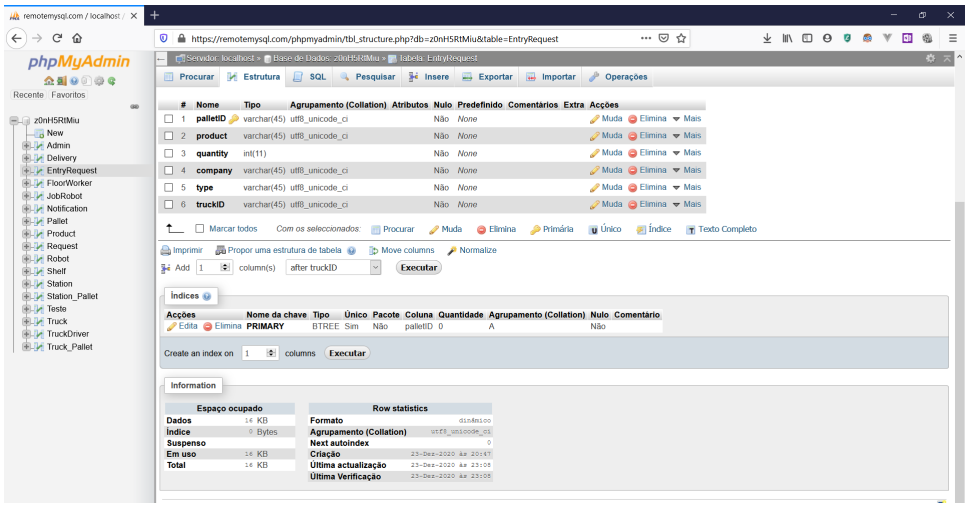


Fig. 39: Our PhpMyAdmin main page

14 Request Server

In order to simulate the process of requesting material from the warehouse, the team decided to build a prototype of a web server who is capable to connect to the application's database and insert a new request for products through user selection.

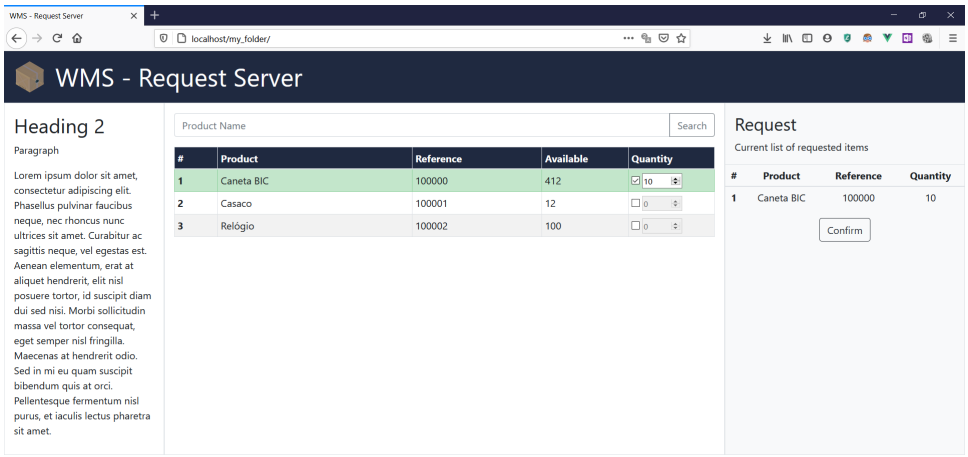


Fig. 40: Web Server Prototype

The user is able to sort through Products, choose which ones to add into the request, and additionally the amount of it. The products' references and stock quantity are both shown on the main table.

15 Critical Analyses

With the development of the conceptual system architecture and behavioral models, we now believe that the project in development is finally able to support all of its expected functionalities. Therefore, the extension of this work to its final stage, using object-oriented programming will be more swift, comprehensive and concise. In addition, with the development of this new phase, some corrections and modifications were made to the previous stage, alongside the construction of the diagrams.

16 User Manual

16.1 Lets get started

(Todo : Instalation of the program and getting the system up and running)

16.2 Log in

After the success of the first step and with the launch of the system, the first window of the program will be shown to the user. In this scene, as that title suggested, the user can choose to Log in to the program or **Create an account**. To **Log in**, the user must provide a combination of authorized credentials. Succeeding the action of the user, the system will authenticate this credentials, and if they are correct the application will redirect the user to a calculated destination.

(TODO: Insert picture of the log in screen)

However, if the system detected an invalidation of this combination, a **Pop-up screen** will appear, warning the user of the invalidation.

(TODO: Insert picture of the invalid authentication pop up)

16.3 Sign Up

From the **Log in** window, and by pressing the **Sign up** button, the user can request an entry to the system by creating an account. Here the use can choose between three types of accounts:

(TODO: Insert picture of the sign up screen)

16.3.1 Sign up: Floor Worker

With this button selected the user will create a **Floor Worker** account.

(TODO: Insert picture of the sign up : floor worker screen)

16.3.2 Sign up: Admin

With this button selected the user will create an **Admin** account.

(TODO: Insert picture of the sign up : admin screen)

16.3.3 Sign up: Truck Driver

With this button selected the user will create a **Truck Driver** account.

(TODO: Insert picture of the sign up : truck driver screen)

After choosing the type of account, the user must fill in the following fields, **First Name**, **Last Name**, **Email**, **Password**. Knowing that the two main information are the **Email** and the **Password**, the application doesn't let a new user have the same **Email** as a user that is already in the system. In addition, and thinking of the user and his security, the system determines how strong is the password introduced by the user and attributes it a score. This score is then converting in a colors system, and are displayed just bellow the password field, where the user can be the judge and choose to modify his password:

- **Red** - Unsafe: Does not meet the minimum standards;
  (TODO: Inset picture of a bad password)
- **Orange** - Good enough: Advisory against employing bad practices;
  (TODO: Insert picture of a good enough password)
- **Green** - Sufficient: Meets minimum standards;
  (TODO: Insert picture of a sufficient password)

– **Blue** - Exceptional: Exceeds minimum standards.
(TODO: Insert picture of a exceptional password)

Right after the submission of the request of the new account, the application will examine the system looking for an email that matches the one introduced by the user. If this match verifies, an **Pop-up screen** will be shown, alarming the user that the credentials are invalid.

(TODO: Insert picture of invalid sign up pop up)

On the other hand, if this match doesn't verifies, an **Pop-up screen** will appear informing the user that his request has been sent and he will need to wait for an approval from an **Admin**.

(TODO: Insert picture of sign up requested pop up)

16.4 Admin

16.4.1 Dashboard

Once a user successfully authenticate himself as an **Admin**, his **Dashboard** will appear. In this scene, a diverse set of information are provided to the **Admin** for him to analyse and make decisions.

(TODO: Insert picture of dashboard)

More in depth of this information the **Admin** is able to inspect:

– **Number of Robots in the system**
(TODO: Insert picture of number of robots)
– **Number of Floor Workers in the system**
(TODO: Insert picture of number of floor workers)
– **Number of Truck Drivers in the system**
(TODO: Insert picture of number of truck drivers)
– **Number of Admins in the system**
(TODO: Insert picture of number of admins)
– **The Weekly Shipments of the warehouse**
(TODO: Insert picture of the weekly shipments)
– **The Warehouse Capacity**
(TODO: Insert picture of the warehouse capacity)
– **The last six received or shipped products**
(TODO: Insert picture of the warehouse capacity)

Additionally, the **Admin** has, at his left, a sidebar where it gives him various possibilities of interacting with the system. In this sidebar the **Admin** has six different buttons, he can find the **Edit Profile** button by hovering his profile picture, manage the requests of **Incoming Trucks**, confirm the requests of **Sign Up Accounts**, the **Locations Of All Pallets**, the **Notifications Of All Robots**, and finally, the **Log Out** button.

(TODO: Inset picture of the sidebar)

16.4.2 Edit Profile

Just as in every application, we also gave our users, in this case the **Admin**, the possibility of changing his credentials. For that, he must fill in all text fields with the following information, **First Name**, **Last Name**, **Email**, **Password**. Keeping that same logic as the **Sign up**, the **Admin** must introduce an **Email** that is available.

(TODO: Insert picture of admin edit profile)

In addition, the system asks for a confirmation of the **Admin**, and for this a **Pop up** screen shows up.

(TODO: Insert confirmation of edit profile)

Once the **Admin** confirms the changes, two things can happen, one of them is the validation of the credentials, where everything went correctly and the changes were made. The other one is the invalidation of the credentials, if this happen a **Pop up** screen will appear warning the **Admin** of the situation.

(TODO: Insert invalidation os credentials screen)

16.4.3 Manage Trucks

For a more in depth control of the warehouse, the **Admin** has the possibility, in this scene, to accept, wait or reject the entries of **Truck Drivers**. If the **Admin** clicks on the accept button the entry of this **Driver** will be permitted and all **Pallets** will be dropped on the **Receiving Station** where a **Floor Worker** can scan each **Pallet**. However, if the **Admin** clicks the reject button, this entry will be completely deleted from the system. Finally, if he doesn't have a decision or wants to freeze this entry, it is also possibly for the **Admin** to put this request in the back of the queue, and make decisions on others requests.

(TODO: Insert picture of the manage truck screen)

16.4.4 Manage Accounts

To have a controlled and logical system, it's not possibly for a new user to create an account without an approval of an **Admin**. Considering this, an **Admin** has the power to accept or reject an account. In this view, he is ably to see the **First Name**, the **Last Name**, the **Email**, and the **Type** of account that an user is trying to create.

(TODO: Insert picture of the manage account screen)

16.4.5 See Locations

As has been mentioned, the **Admin** is the main character in the application, and he is the one who makes the must decisions. Having this in mind, it seems to be clear that every **Admin** needs to overview every **Pallet** and its **Location**. In this scene, he is capably of seeing the **ID**, the **Product**, its **Location**, and the **Type** of all **Pallets**.

(TODO: Insert picture of the locations of the pallets)

16.4.6 Notifications

Lastly, but on the same topic of control and management, the **Admin** is able to see every interaction of a **Robot** with the system. Here he can find all **Notifications** communicated by every **Robot**, delete each **Notification** or delete everyone.

(TODO: Insert picture of notifications)

16.5 Floor Worker

To a assist the **Admin** a new actor is needed. This actor will help the **Admin** control and manage the **Receiving Station** and the **Shipping Station**. This stations are physical points of communication with the **Truck Driver**, and this is where the **Floor Worker** enters in action.

Just as the **Admin**, the **Floor Worker** also has a side bar were he can find, the **Edit Profile** button by once more hovering his profile picture, the **Receiving Station**, the **Shipping Station**, and the **Log out** button.

(TODO: Insert picture of the floor worker side bar)

16.5.1 Edit Profile

Similarly to the **Admin Edit Profile**, the **Floor Worker** has the same system of editing his profile.

(TODO: Insert Floor worker Edit profile screen)

He also has to accept the changes.

(TODO: Insert Floor worker Edit profile confirmation screen)

And he also have the same error message if the credentials are invalid.

(TODO: Insert Floor worker Edit profile invalid credentials screen)

16.5.2 Receiving Station

From the **Log in** window, if a user authenticates himself in the system as a **Floor Worker**, the first scene that will be shown is the **Receiving Station**. Here the **Floor Worker** can see all **Pallets** and there information (**Product, Company, Quantity, Type, QR Code**) in this station. The **Floor Worker** is also able to select a **Pallet** and read its QR Code. After the read of the QR Code, its **Pallet** is ready to be pick up by a robot and store it in the warehouse.

(TODO: Insert receiving station picture)

16.5.3 Shipping Station

After a order is made and all **Pallets** are dropped in the **Shipping Station** the **Floor Worker** must confirm its exit. For this, the **Floor Worker** can select a order and confirm that all **Pallets** are already in the station.

(TODO: Insert Shipping Station)

16.6 Truck Driver

To have a more dynamic application, and to have **Pallets** being introduced to the system, we decided that it would be perfect if the **Truck Driver**, when trying to entry the warehouse, introduce the **Pallets** that he wants to deliver.

Keeping the same logic as the **Admin** and the **Floor Worker**, the **Truck Driver** also has a side bar where he can find an **Edit Profile** button by hovering his profile picture, and the **Log out** button.

(TODO: Insert Sidebar of the truck driver)

16.6.1 Edit Profile

Likewise then **Admin** and the **Floor Worker** edit profile screen, the **Truck Driver** also has the same system of edit profile.

(TODO: Insert Truck driver Edit profile screen)

He has to accept the changes.

(TODO: Insert Truck driver Edit profile confirmation screen)

And he also have the same error message if the credentials are invalid.

(TODO: Insert Truck driver Edit profile invalid credentials screen)

16.6.2 Entry Request

Once more, after the **Log in** window, if a user authenticates himself in the system as a **Truck Driver**, the **Entry Request** will be the first screen to appear. In this scene, the **Truck Driver** can make a request to enter the warehouse.

(TODO: Insert the picture of the entry request screen)

As mentioned, it is the **Truck Driver** that introduces the **Pallets** to the system, so for this reason he will have, at his right, a box where he will be able to introduce the information

needed to populate a **Pallet**. For this he will need to fill in the name of the **Product**, the **Companies** name, the **Quantity** of the **Product** on the **Pallet**, and finally choose between on of the buttons just bellow that indicates the **Type** of the **Pallet**, and this can be a **Normal** or a **Perishable**.

(TODO: Insert the picture of the add pallet side bar)

If the **Truck Driver** doesn't insert all information needed, this insertion will be denied and a **Pop up** screen will appear, warning him that the **Pallet** introduced is invalid.

(TODO: Insert invalid pallet pop up screen)

When correctly introduced, this information will then be listed and displayed for **Truck Driver**, as we can see in the main box. In this box he can also delete each **Pallet** and just below this box he can delete all **Pallets** introduced or send the request to the system.

(TODO: Inset the picture of the main box)

Finally, when the **Truck Driver** is satisfied with his request, he can click the save button. When this event happens an **Pop up** screen will be shown informing the **Truck Driver** that the entry request has been sent and he will need to wait for the approval from an admin.

(TODO: Insert entry request sent pop up)

17 Critical analysis

In this project we were able to create a realistic simulation for a warehouse management system, in which we included a app with modern graphical design and a server database implmentation with PHP and MySQL. The use of models such as the domain and use cases models have allowed us to root out potential logical inconsistencies and provide a clear-view of the structure we needed for our work. This in turn, made all of our work more efficient. Meanwhile, in order to properly implement the

References

1. Nestor, A.: Um Processo de Modelação de Sistemas Software com Integração de Especificações Rigorosas (2008)
2. Pilone, D., Pitamn, Neil: UML 2.0 in a Nutshell: A Desktop Quick Reference, 2nd Edition (2005)