E.T.S. de Ingeniería Industrial,
Informática y de Telecomunicación

Design and development of a mobile application for warehouse logistics management

Grado en Ingeniería Informática

Programa Internacional

Trabajo Fin de Grado

Iker Zabalegui Murugarren

José Javier Astrain Escola

Pamplona, 06/09/2024

Summary

I have developed an Android application that manages the logistics of a warehouse. With this tool, the aim is to reduce the misplacement and loss of stock and improve the control a manager has over its own warehouse.

The development has been carried out following Agile and continuous integration principles. The result is an easy to use, useful and very adaptable application that can be introduced in different types of warehouses. The project's programming phase was divided in seven sprints, each one of them resolved the problems presented in the user stories. The screen layout designs were built to match the patterns of the modern user interfaces that are the standard of the Android application designs nowadays.

The application files are stored in the following GitHub link:
https://github.com/ikerZaba/TFG

Key Words

- Android Studio
- Android
- Java
- Digitalization
- Warehouse
- Logistics
- Containers
- Orders
- Sales
- Inventory
- Stock
- Design
- Layout
- Agile
- Sprint
- Activity
- Form
- SQLite (database)
- Fragment
- QR code
- Flowchart
- List View
- Card View

Index of contents

1.- Introduction

1.1.- Introduction: project background and description of the problem to solve and its context.

In the era of digitalization that we are living in, every business should be taking advantage of the digital tools that can be developed to help manage their company better. These tools vary in shape and form, and they help automate processes and visualize data that can't be managed by hand. Using these applications to your advantage is the basis to grow your business.

One of the most common issues in companies around the world is stock management. This is, because you have to take into account big amounts of product, and you have to know **where** everything is stored for you to sell it afterwards. Not only that, but you also must know **how much** of each product you have stored, **how much** of that product you are selling, **when** did that product arrive, **when** is the shipment leaving… On top of that you must have every data on the stock **up to date**, so you don't sell goods that don't exist or that are already gone.

As you can see, working through all of that is not trivial and the use of digital tools seems obvious. Turns out, that it is not that easy.

Numerous companies nowadays that face these problems, still don't use these types of solutions to help their business and instead opt for more conventional ways. The reasons are simple: money and familiarity with technology. These tools are not cheap to implement, and they require an adaptation period for both the workers and the managers at the enterprise.

But is it more expensive not to run a business with the help of these? The accidental losses of stock and the lack of organization of shipments can amount to a point that the losses are bigger than the money you save from not having the digital solutions.

If there are still businesses that are run without this type of tools, it means that the existing ones either do not adapt to their warehouse or that they are not accessible, either because they are too expensive or because they are difficult to use.

When I look at the topic of managing a warehouse, I think I can offer a new perspective on how this type of apps work and develop my own one. My target is to develop the one application that convinces the small and medium-sized businesses that they can manage their business better using my tool. I think that there is a market opportunity in this field right now.

One of the benefits of working with those type of businesses is that, because of their size, I imagine I can work closely with the manager while I am adapting the tool I am going to develop to their business and create an app that fits perfectly each warehouse. If

the company is bigger, they maybe have more than one location and each one of them has different set ups and it is tougher to make a single app adapted to those.

Finally, I decided to involve myself in this project because the skills that I'm going to improve on and learn about in this project will be very useful in my future in the working environment. I also like the challenge of solving the organization of a warehouse.

1.2.- Objectives

The aim of the project is to develop a tool that can solve the logistical issues that appear when managing a warehouse. The app must cover every aspect of the business that is accountable (stock, transport of the goods, money balance…).

The programming should be such that the resulting application has to be adaptable to different types of warehouses and business fields, as they require different information to be exploited and certain features to be added.

These are the usual topics that an application in this field attacks. But as I stated before, the app is being developed with the small and medium-sized businesses in mind. This means that apart from the objectives above, I need to think about what this type of business would need in a day-to-day basis.

I must think about the adaptation costs, the product has to be cheap to implement in any business situation. Other solutions require the purchase of chips and their respective scanners, and our target may not be able to afford that. They also require the purchase of licenses that range from 500€ to 25000€ per year that the client must pay to use those applications.

The tool must be easy to use because I must assume that the workers have not worked with an application like this before. Moreover, their knowledge about technology can be limited. Programming a tool that is easy to use is fundamental in this case because one of the main reasons to use an app like this is to reduce the human error and the misunderstandings between workers and managers that leads to the loss or misplacement of stock. I essential that I demonstrate to the client that the application will do that in order to convince them to buy it.

As the final objective, the application must be adaptable to the workflow that is already used in the warehouses where it will be used. A tool that requires the client to noticeably change they way their warehouse works will not be appealing for them, because they will think that they will need to teach the workers the new tasks and this means losing time and money.

1.3.- State of the art

There are lots of solutions to the warehouse managing problem. The simplest ones involve writing down every piece of data like transactions and product stock in a spreadsheet. This is very widely spread, as every computer has either Open Office's Calc or Microsoft's Excel programs. In the case of Calc it is a free to use application. You do need your workers to know how to use these tools or to report the information needing to be updated to the person that takes care of the spreadsheets and it is widely known that in human interactions some information might be lost or wrongly transmitted. These spreadsheets can also end up being very complex and long and, although you can create sheets that show lots of very useful information, it can very easily become a soup of data without any direction if you are not able enough with the use of the program.

In the case of the big warehouses around the world like Amazon, they use robots that go around the corridors with sensors that allow them to know what is inside every box to check the stock. There is no information about what application or tools they use to manage the information those robots give them. But the point is, that these types of solutions don't fit the smaller businesses that haven't updated their way of managing their warehouses yet, because the cost of implementing these robots is too much.

Finally, the type of warehouse managing software that is most used is the application. Usually, a computer application where all the information is stored. They also have different features that allow the managers to visualize the information they want easier than a spreadsheet and giving the users the option to, for example, log in. They can add as much complexity and features as the company requires them to. But the problem persists in the way the information is gathered and introduced into the application. The other problem is that some of the tools look very old and outdated, and they are far from easy to use. Let's see some examples:

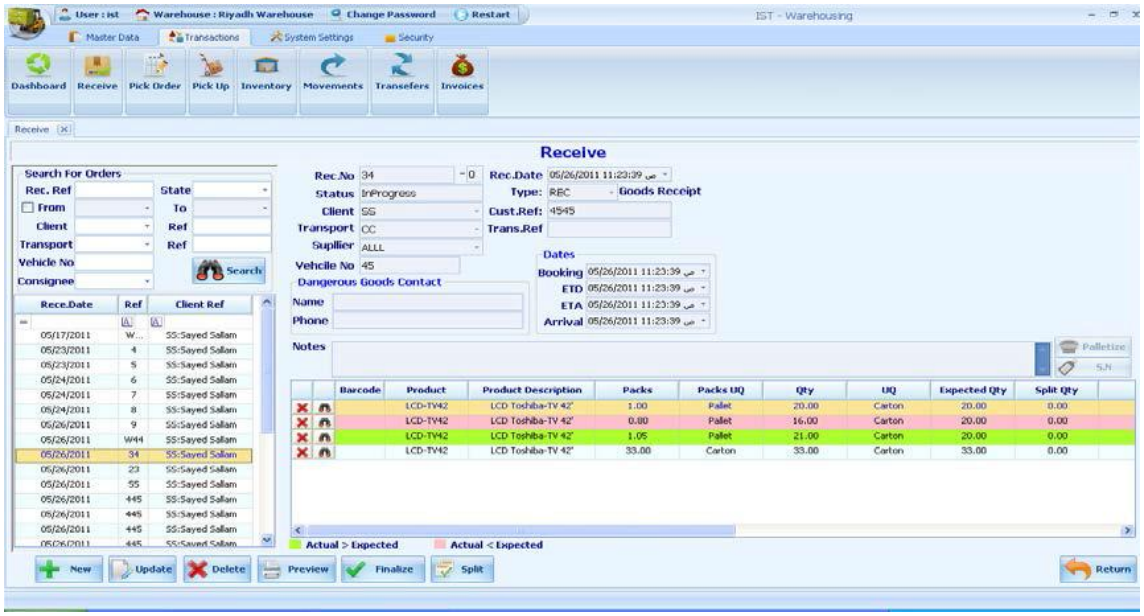

Figure 1: IST Wareouse Management screenshot

This is IST Warehouse, one of the first warehouse management application that pops up when searching on google. This is an example of what I must avoid, an old looking design with too much information on-screen, small sized text and colors that do not match each other.

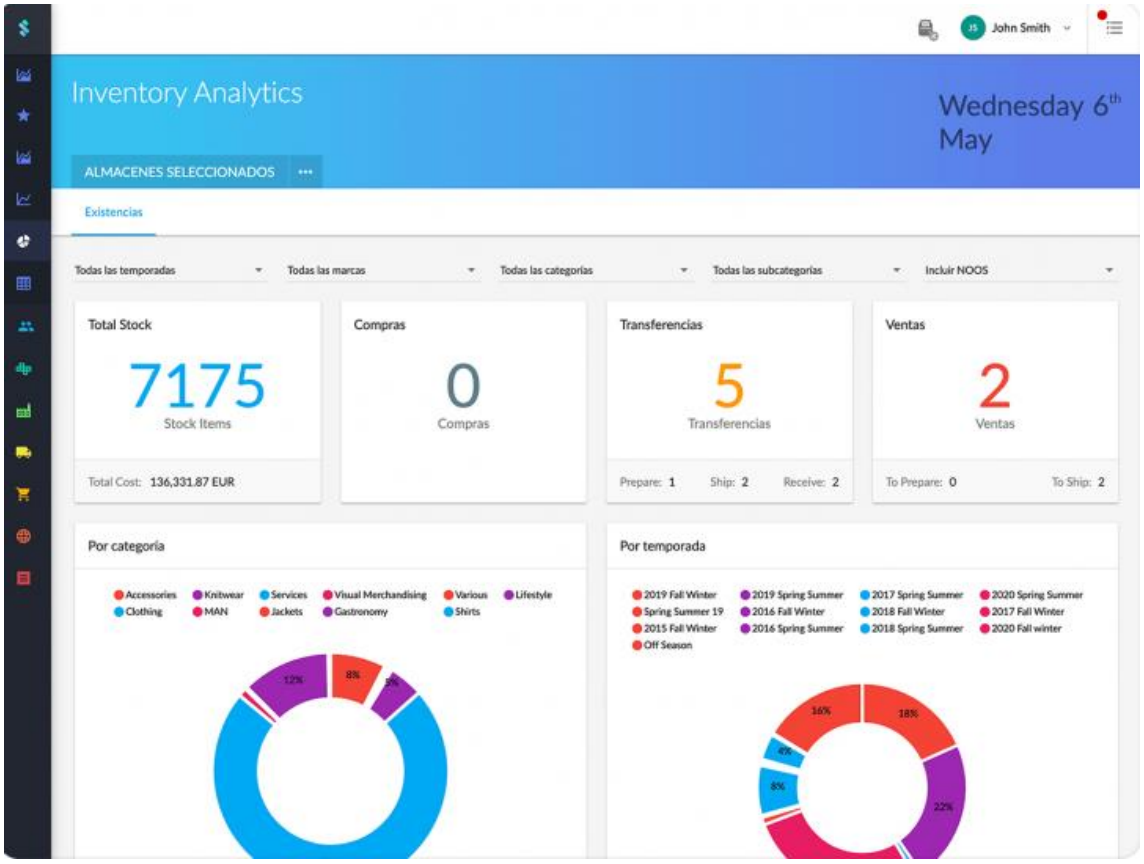

Figure 2: Stock Agile app screenshot

This is Stock Agile, one of the leaders, if not the leader, in warehouse management applications nowadays. This is the type of modern UI design that I should look at when looking to create an app about this topic. This type of design is the standard in the industry. The difference between Stock Agile and my project is that to upload the data about the orders of product and the sales to the app, you need to upload an excel with the data, for example. My idea is to enclose all this process into one app without the need of external information. The workers will notify the arrival of an order, for example, and it will update the database and all the info about the inventory.

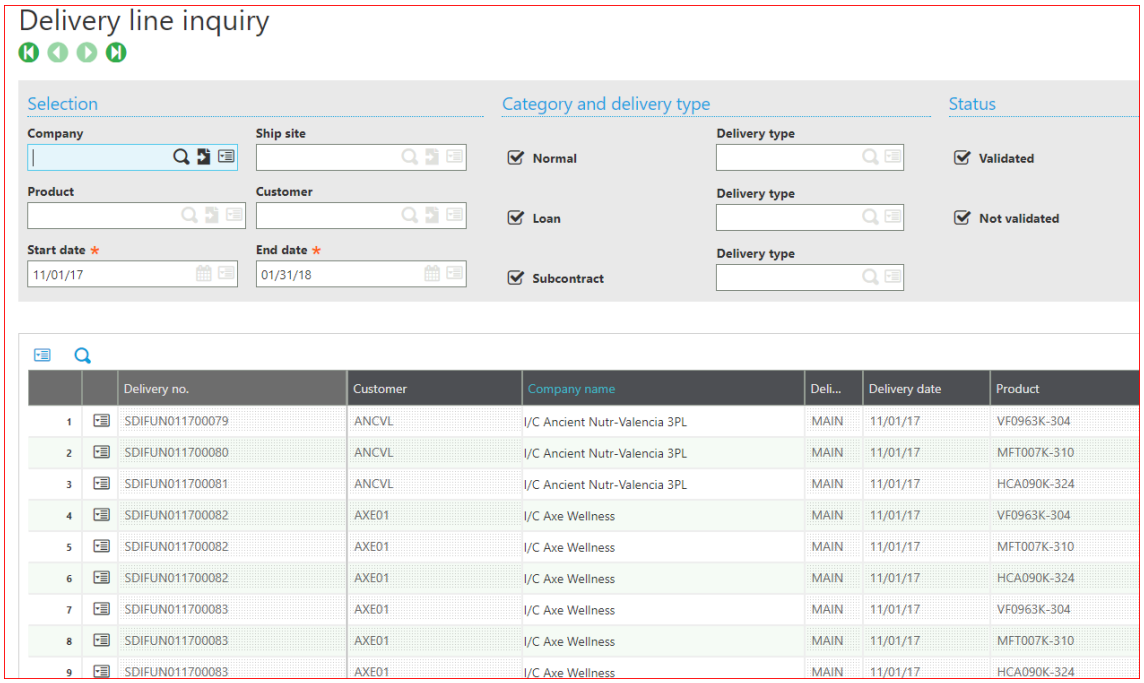

Figure 3: SageX3 Warehouse Managment Application

Finally, this is the SageX3 warehouse management application, also one of the leaders in this business. The problem with this is the prize. It costs 25,000€ per year. For a big company that might not be but the companies I want to program my tool for cannot afford that type of prize. The Stock Agile option is more affordable (65€ a month), but it is not adapted to your business. Other options like Sloteyes cost 300€ a month.

1.4.- Description of the proposal

The project will be about developing an Android application and a methodology of work that suits the use of the tool. As previously stated, the app must be easy to use, with an appealing design and the methodology should be as simple as possible so the workers and the management take as little time as possible to learn and implement it.

I am choosing to develop an app in Android because everyone nowadays owns a phone, so when implementing the app in the business will not need to purchase any new devices. This is because the workers can install the application in their own phones and work with it. This also means that the app can run in tablets and other devices that run the Android operative system if the company already owns them or if they choose to buy more.

I am going to use the Android Studio IDE to develop my application, as it is the IDE to develop Android applications, and it is also free to use. Moreover, I decided that the language I am going to use for the application is Java, because I have used it in many projects before. The app will work from Android 8.0 and up, so it will work in 95% of the devices worldwide (*Source: Android Studio IDE).* The layout for each of the screens in Android Studio are programed in XML language.

A database will be needed to save the information about everything happening in the warehouse. I will use SQLite to create that database because it is free to use, and I am familiar with the SQL language. It is true that SQLite only offers basic types of variables and does only support less than 64 tables. But I do not need those types nor that many tables.

In exchange, SQLite offers reading and writing operations that are very fast. It is almost 35% faster than File system. It only loads the data which is needed, rather than reading the entire file and hold it in memory. If you edit small parts, it only overwrites the parts of the file which was changed. It is also very lightweight, so it is easy to use embedded software with devices like televisions, Mobile phones, cameras, home electronic devices, etc. (Source: JavaPoint).

The name of the application will be "IZInventory". It is a mixture of my initials 'I' and 'Z' with the words 'inventory' and 'easy' which is what the application wants to be, a tool that makes it easy to organize the inventory of a warehouse.

One of the ideas that I have in mind is that each of the containers has its own QR code attached to it, so to run the application in the warehouse a printer with internet access may be needed.

1.5.- Methodology to follow

To ensure the successful development and implementation of the warehouse management application, the methodology adopted will be the AGILE methodology. This approach emphasizes flexibility, collaboration, and customer feedback, which are essential for creating a tool that meets the diverse needs of different businesses. Below are the key steps will be followed in the AGILE methodology:

1. I will start by analyzing the problems that I want the application to resolve and create **user stories** from the perspective of the end-users (workers and managers).

2. These user stories will be divided in **sprints**. These sprints will tend to gather the user stories that have correlation between them (for example: "as a manager I want to access the list of containers" and "as a worker I want to access the content of each container"). The sprints are going to be of similar length.

3. At the end of each sprint, the aim is to have a fully usable application, following the Iterative Development Principle, and to rethink if the process followed in the implementation of the new features can be improved.

4. As this is a project that I am doing alone, daily team meetings and continuous integration will not be part of this project.

I think the Agile methodology is the correct one to use in this case because it encourages continuous delivery, focuses on the customers value and it works very well with small

groups of developers (in this case, only me). I want my development to be focused on adding features that work to an already working application at the end of each sprint.

Ultimately, if the agile development is correctly implemented, the result will be high quality software. Agile guarantees that the software satisfies the required standards and lowers the likelihood of faults by integrating quality assurance throughout the development process. For this to be true, I will need to make automated tests and frequently revise and modify the code.

2.- Analysis and design

To improve my knowledge of the problem I am dealing with, I made a visit to the warehouse of the company my uncle works in. They, as many other companies, do not have a tool to manage the input and output of cargo in the warehouse. In this case, they use an Excel spreadsheet where everything is imputed by hand and the communication between the workers and the managers is done by paper. When a new shipment comes, the workers write down the info of the content on a paper and they do that until the end of the day and then the information is put the next day on the spreadsheet. Naturally, I was told by the manager that a lot of information is lost and that doing an inventory check is almost impossible.

2.1.- The setting

As you do when you are trying to solve a physics problem, one must know the setting of it. In this case, the setting is a warehouse, obviously, but there are different types of warehouses. To solve the problem knowing where the shipments arrive, how the containers are stored and where the sales are done is imperative.

Figure 4: Photo of a warehouse (Lily, n.d.)

I have gathered information to try to assess how the average warehouse looks like. This is because the tool I want to develop has to be adaptable to any type of setting. I have determined that the model of warehouse I am going to be working with is the following:

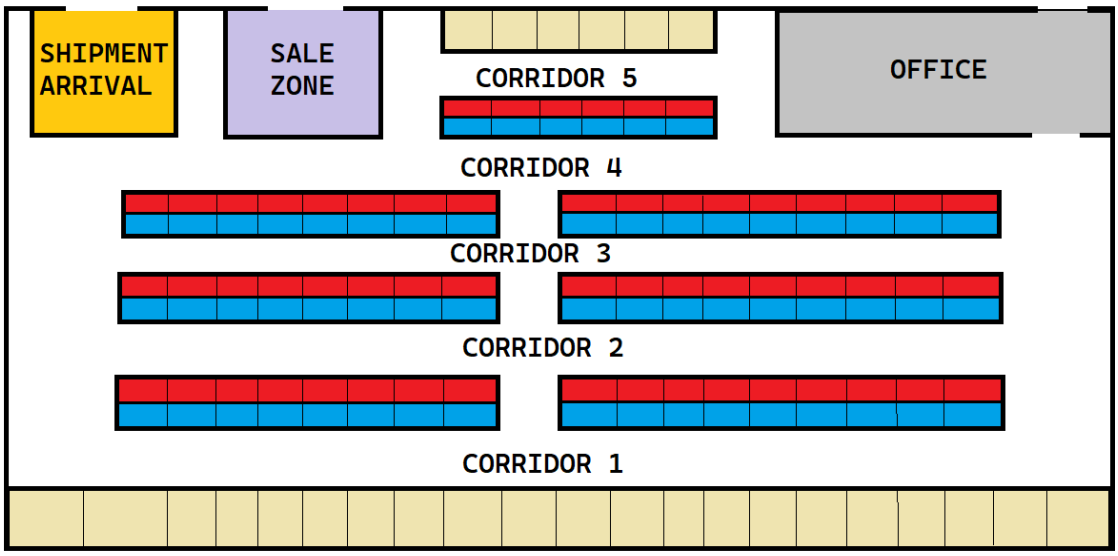

Figure 5: Map of a warehouse

The squares in red blue and khaki represent the boxes in the warehouse. As you can see the warehouse has multiple corridors with boxes to the sides and several boxes on each corridor. One of the tasks that I have to solve is figuring out how to set up a name for each of the containers, so the workers can rapidly know where its place only knowing that name.

My idea is assigning a code to each container. This code will be composed by three parameters: the corridor number, the side and a box number. To implement this each of the container slots will be numbered. I considered also adding a fourth parameter called height and renaming box number to slot number, but I think a three-part code is simpler and it is understandable enough. So, for example, is the code is “1-R-12”, the location will be in corridor 1 to the right and the worker should search for the box number 12.

2.2.- User Stories

“A user story is an informal, general explanation of a software feature written from the perspective of the end user or customer.” (Rehkopf, n.d.)

After learning about the problems that my uncle’s warehouse suffers from not having a warehouse management system and researching what functionalities do the

market leaders offer, I deduced a list of user stories that will lead to my application having the features that I consider indispensable:

- As a manager I want a tool that my workers can use to register what is happening in the warehouse (a new order arrived, a sale is completed, a container is filled…), so that every event can be registered.
- As a manager I want to access the information of the contents of every container in the warehouse.
- As a manager I want a tool that allows me to reduce the stock that is misplaced, so that I can stop losing money because of this issue.
- As a manager I want the tool to be easy to use and cheap to implement in the current workflow.
- As a manager I want to access the information of every order of product that comes into the warehouse, so that my workers can know when an order is arriving.
- As a manager I want to save the information of every sale of the product of the warehouse, so that I know which products I am selling.
- As a manager I want information about the capacity of the warehouse currently, so that I know when to buy more containers.
- As a manager I want to know the weekly performance of the sales activity.
- As a manager I want to be able to know what container is low in stock, so that I know which product to order.

- As a worker I want to know when the product was placed into a container.
- As a worker I want to know when a new product order is arriving at the warehouse, so that I prepare the garage for the arrival.
- As a worker I want to know when a sale is being conducted, so that I can prepare the container to be emptied in advance.
- As a worker I want to be able to access the information of a container rapidly, so that I can know if the product that is inside matches what the application says.
- As a worker I want to be able to notify when a sale transaction is completed, so that my manager knows it.
- As a worker I want to be able to notify when a new order has arrived at the warehouse, so that I can start filling the container.

The worker's user stories are also included because they are going to use the app to complete their everyday tasks. So, I need to listen to their needs in order to make an app that they feel that is easy to use and they can work with.

2.3.- Division of the user stories

Following the agile principles, the previously discussed User Stories will be divided in various Sprints. Each of the Sprints will encapsulate a set of User Stories that are related to one another. The aim of this is to add features to the tool each Sprint that don't modify the previously implemented features, so each of the Sprints is independent. So, each of the Sprints will last around a week and will revolve around what I will call a "topic". The topics of each Sprint will be the following:

1. Container features: will add the entity Container and all its functionalities.
2. Product orders features: will add the entity Order and all its functionalities. The aim is to be able to order products.
3. Product sales features: will add the entity Sale and all its functionalities. The aim is to be able to sell the contents of a container.
4. Solution to accessing the information of a container and connection between the orders and the containers.
5. Division of capabilities between workers and managers
6. Additional features for the manager that show information about the warehouse.
7. Application makeover to ensure usability.

2.4 Design: Architecture

The first step to designing any system is defining the architecture of such system. In the case of Android applications, the recommended architecture is composed by an UI layer and a data layer. (Android, n.d.)

The UI layer's job is to display the application data on the screen. In my case, this layer will be composed by the Java activities and the XML coded layouts. The layouts contain all the elements like buttons, text fields and menus. They also set their size, their shape, their color and where those elements are placed on the screen. Activities are Java classes that define the interactions between the user and those objects defined in the layout file. Activities oversee calling the data layer when needed. It is important not to save any data in activities because they can be terminated, all the data should be placed in the data layer.

The data layer's job is to contain the business logic of the app and exposes the application data. It is composed by two parts repositories and data sources. The data

sources can be remote or local and they can be more than one. In my project, I will only have one data source which will be an SQLite database. The repositories are called by the UI layer to expose the data and update changes in the database. In my case, a Java class will act as the database manager and will contain all the logic about creating the tables, filling them, updating them and consulting them.

Along with this it is recommended to follow some good practices. Firstly, reduce dependencies in Android classes. This means that the app components should be the only classes that rely on Android framework SDK APIs such as context or Toast. It is also recommended to create well defined boundaries of responsibility between the modules. For example, having all the methods that access the database in the same class. Along the lines of this, exposing as little as possible of each module is recommended. Making each part of the app testable in isolation should be a priority, so the app's features are easily testable. There are more recommended practices, but these are the ones that apply to my project.

Following these practices along with implementing correctly the architecture will lead to my app maintainable and robust. It allows the app to scale. More people and more teams can contribute to the same codebase with minimal code conflicts. It helps with onboarding. As Architecture brings consistency to your project, new members of the team can quickly get up to speed and be more efficient in less amount of time. It is easier to test. A good Architecture encourages simpler types which are generally easier to test. Bugs can be investigated methodically with well-defined processes. (Android, n.d.)

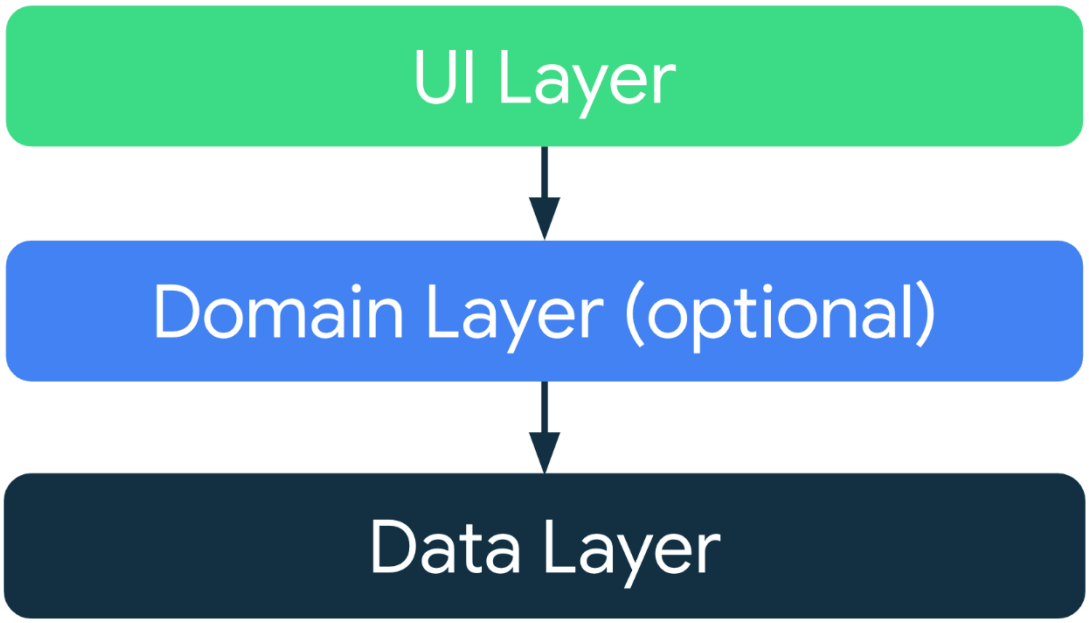

Figure 6: Android architecture diagram (Android, n.d.)

2.5.- Design: Data Layer

Database

As I mentioned before, my app will contain a database. This database needs to be able to contain all the information that I need to run the app.

The Container is the main entity in the application. Everything revolves around filling and emptying these. To fill them, we will create Orders which will bring the product into the warehouse. Creating a sale means that the content of a container is going to be emptied. From this analysis it is clear that I need at least three tables in my database.

The first table will be the table orders. The information that is important about orders is what they bring to the warehouse and when it is arriving. For that, each of the entries in this table will have an ID, a type of product, the company delivering the product, the weight of the order and the date of the arrival. It will also have an attribute called 'status' that will indicate if the order has arrived yet to the warehouse.

The second table is the table containers. Each of the containers must save information about their location in the warehouse and what is their content because the workers need to check it for filling and emptying them. In this case, the primary key will be the container code tat I explained earlier. It functions as such because there cannot be two containers in the same place. Along with this, the product inside the container, the current weight contained inside and the maximum weight that the container can keep will also be attributes of this table. Finally, I also added the date that the product was put into the container because for some products, it can be interesting to know when they arrived, for example for foodstuff.

The last table is the sales table. The sales must register what was sold, from which container, when and to who. Tis information will later be used for the inventory, for the workers to know where the product to be sold it is located and when to be prepared for the sale. To match these needs, the sale will have a foreign key to a container, and it will have the attributes product, weight of the sale, total prize, client, date of the sale and the previously mentioned 'status' attribute. The status in this case will indicate if the sale is completed or not.

There are some decisions I have taken about the dynamic of the application that shape the database how it is. A container has only one type of product inside. I have decided not to save the order id in the container because multiple orders can go into one container. Sales, however, can only link to one container but one container can have its content sold multiple times. The price is total for each sale, not per kilo of product. I save the date of arrival of the product because in some businesses it's important to know when a product arrived.

Anyways, these characteristics can be easily changed if the client requires the database to function in a different way.

Database Manager

The only class that is allowed to directly write or read from the database will be called the DB Manager class. The methods of this class will consist of building queries to either retrieve information from it or to add new entries to them or to modify one specific entry.

The common methods for all three tables will be the ones that add entries and the methods that return the list of all the entries in a table.

Sales and orders will both have methods to confirm the completion of them. For the orders the method will consist of confirming its arrival and for orders the methos will indicate that the sale is done. These two tables also have methods that are used to return only the list of entries that share the same status.

The containers table will be accessed to fill one of the containers in the case of an order arrival or to empty it in the case of a sale. I will also program methods that are used to monitor the status of the warehouse and to do the inventory like getting the number of low stock containers, the total number of containers, get the total free space at the warehouse…

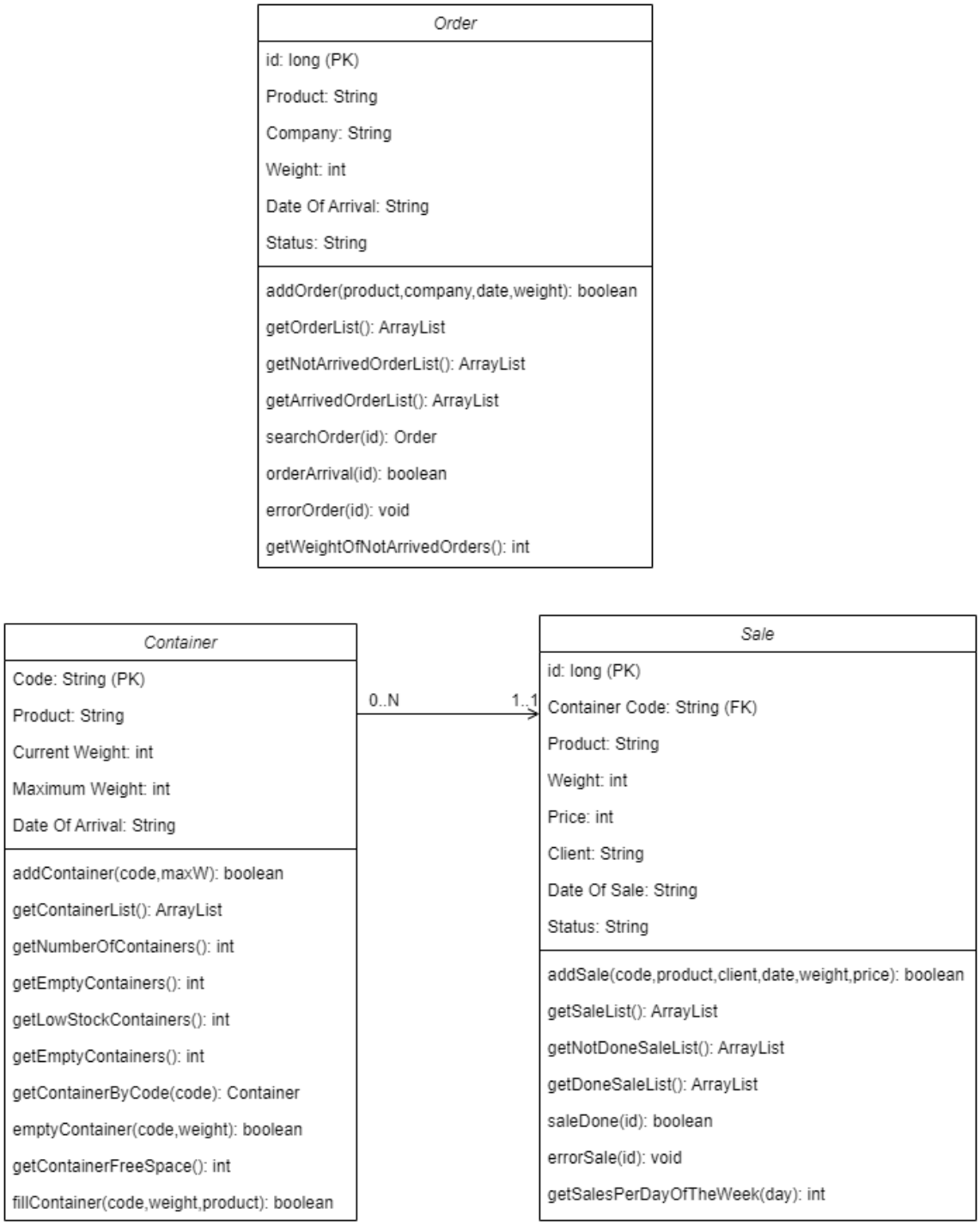

Figure 7: Database table diagram with the names and types of all the attributes and methods.

2.6.- Design: Data Layer, Business Logic

Before starting to develop the application, I must set how the tasks in the warehouse are carried out. There are two processes that are carried out int the warehouse that I must define every step of. As I previously mentioned, I want to solve the problem of miscommunication between the worker and the manager that leads to misplacement or loss of stock. In the domain that my application covers there are two instances where the worker has to register an event in the database and because of that has to communicate with the manager. Thos instances are when an order arrives and when a sale is carried out.

To illustrate how I think the solution of this problem should look like and, therefore, how it will be implemented in my application, I have drawn two flowcharts, one for each process.

2.6.1.- Flowchart for order arrival

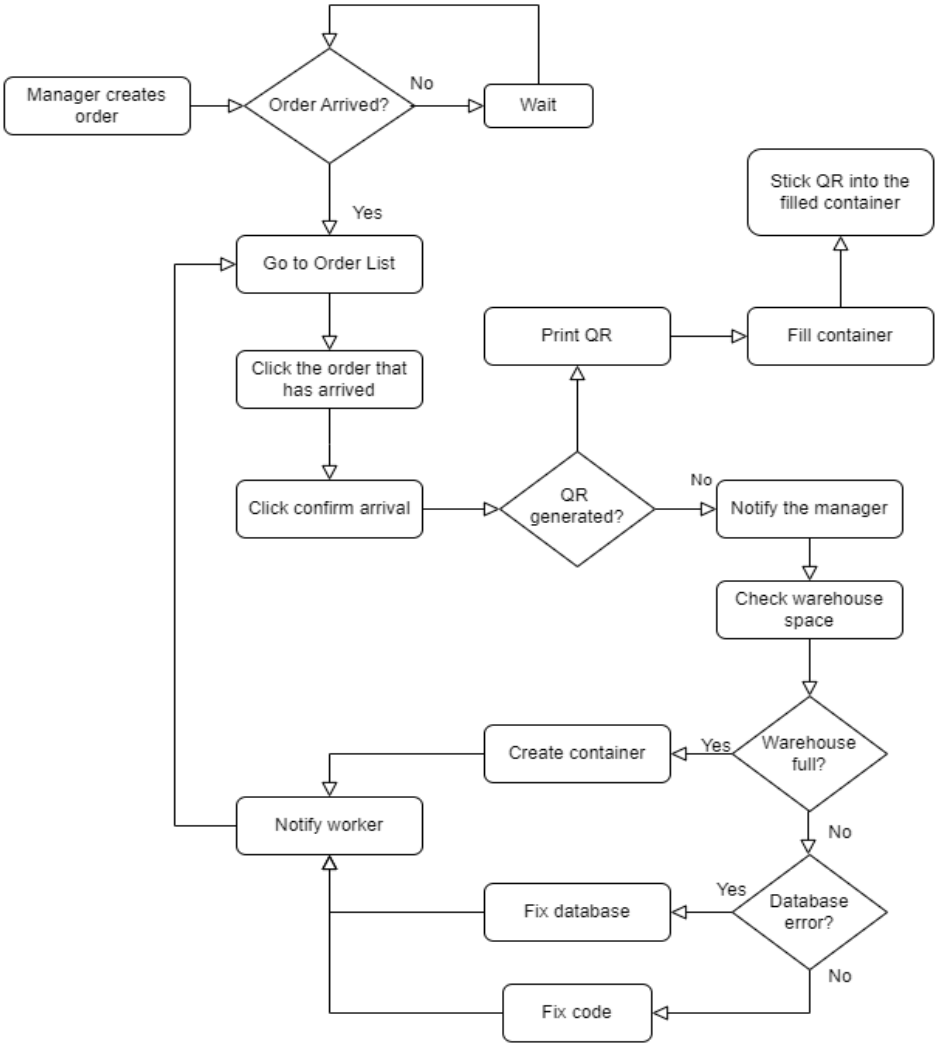

Figure 8: Flowchart diagram of the process to follow in the case of an order arrival

2.6.2.-Flowchart for sale completion

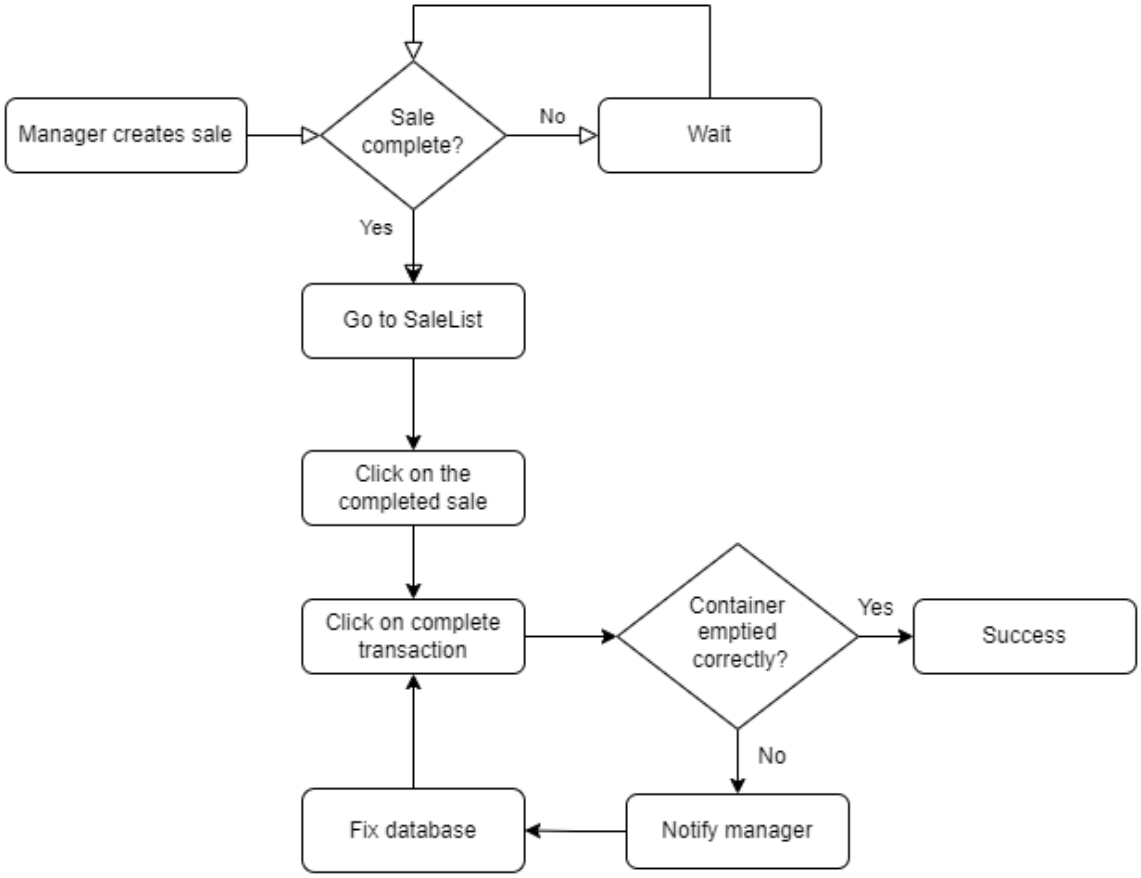

Figure 9: Flowchart diagram of the process to follow when a sale is completed

2.7.- Visual Design: UI Layer

One of, if not the most important ways of transmitting that an application is easy to use is the visual composition of said application. Colors, button placement, the information shown, the size of the text… all need to be in harmony to guide the user through the application the way that the developer intends to. The scope and the target user's familiarity with technology must be understood to build the correct layout for the job.

As I stated before, the users of this application will be the workers and the managers of a warehouse. I assume that the manager will be in contact with me during the development and I will be able to make a tool that shows the information the way they like it. The worker, however, is the most important piece in the jigsaw. I assumed that they have never worked with a tool like this before, so the steps to completing the tasks must be clear and they have to easily reach the important information.

Designing appealing layouts was one of my weaknesses when I first started this project. That is why I researched various websites for some tips and tricks to improve on this. The new norm is minimalistic design, which is helpful in my case because it revolves around leaving a lot of white space for the user to breathe and focusing the attention on small pieces of information clearly highlighted by the contrast in color. Color is also a big factor, but I will talk about it later.

So, the takeaways were the following. To build an application that is user friendly I need it to have a minimalistic design, highlighted icons, shadows, with rounded buttons and a palette of colors that transmits the vibe I want to go for.

2.7.1.- Color

The color palette I selected for the application was the following:

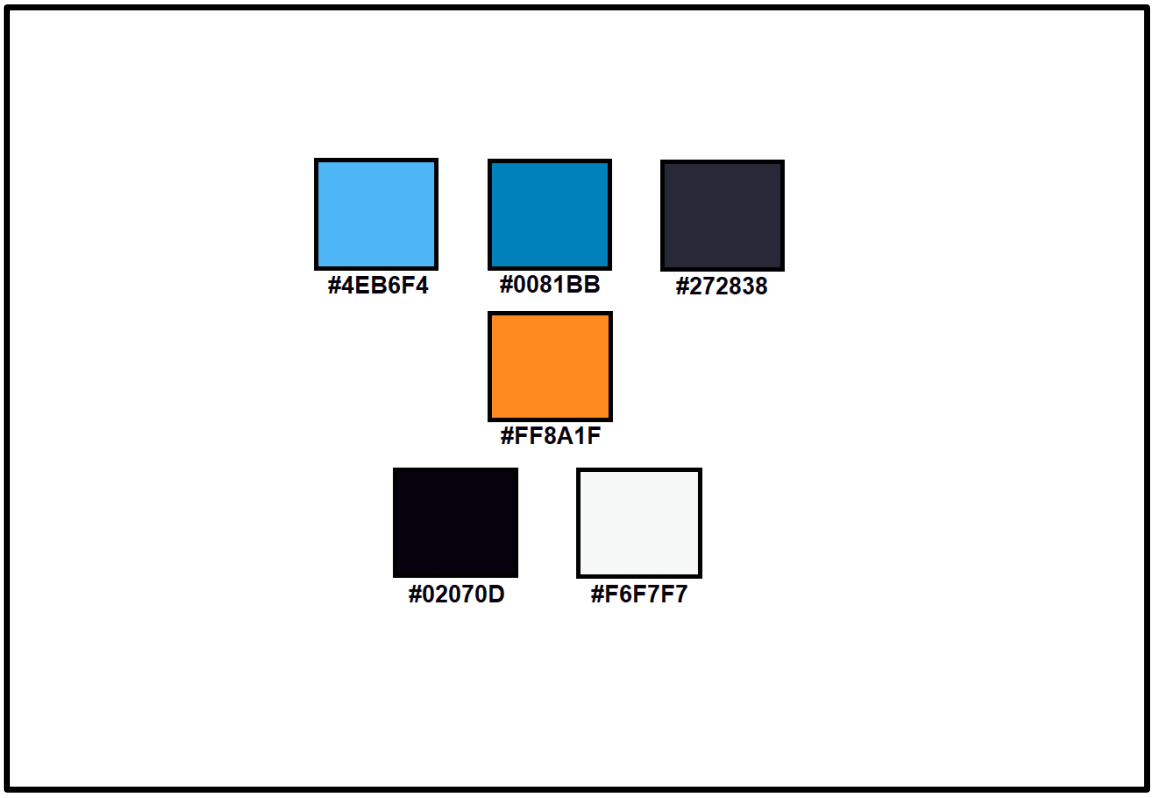

Figure 10: Color palette used in the application design

The color blue as the base of the palette represents security and trust, it transmits calmness and makes the user feel relaxed. Blue is viewed as a cooperative color. The use of shades of blue is widespread in tools that are used in the field of business. So, it makes sense to use it in this application too. Orange, on the other hand, contrasts with blue in a very nice way as blue is a cold color. Orange is used to transmit optimism and

encouragement. I think that uplifting the looks of the application with the use of orange makes it not boring for the user.

Moreover, the white and black used are not total black (#000000) or total white (#FFFFFF) because it is proven that looking at those colors for an extended period of time produces tired eyesight.

2.6 The logo

Finaly, some ideas for a future logo for the application came to my mind and I did some sketches, but I do not possess the skills to make a decent enough logo by myself. The idea is to merge the IZI into a container or into a warehouse.

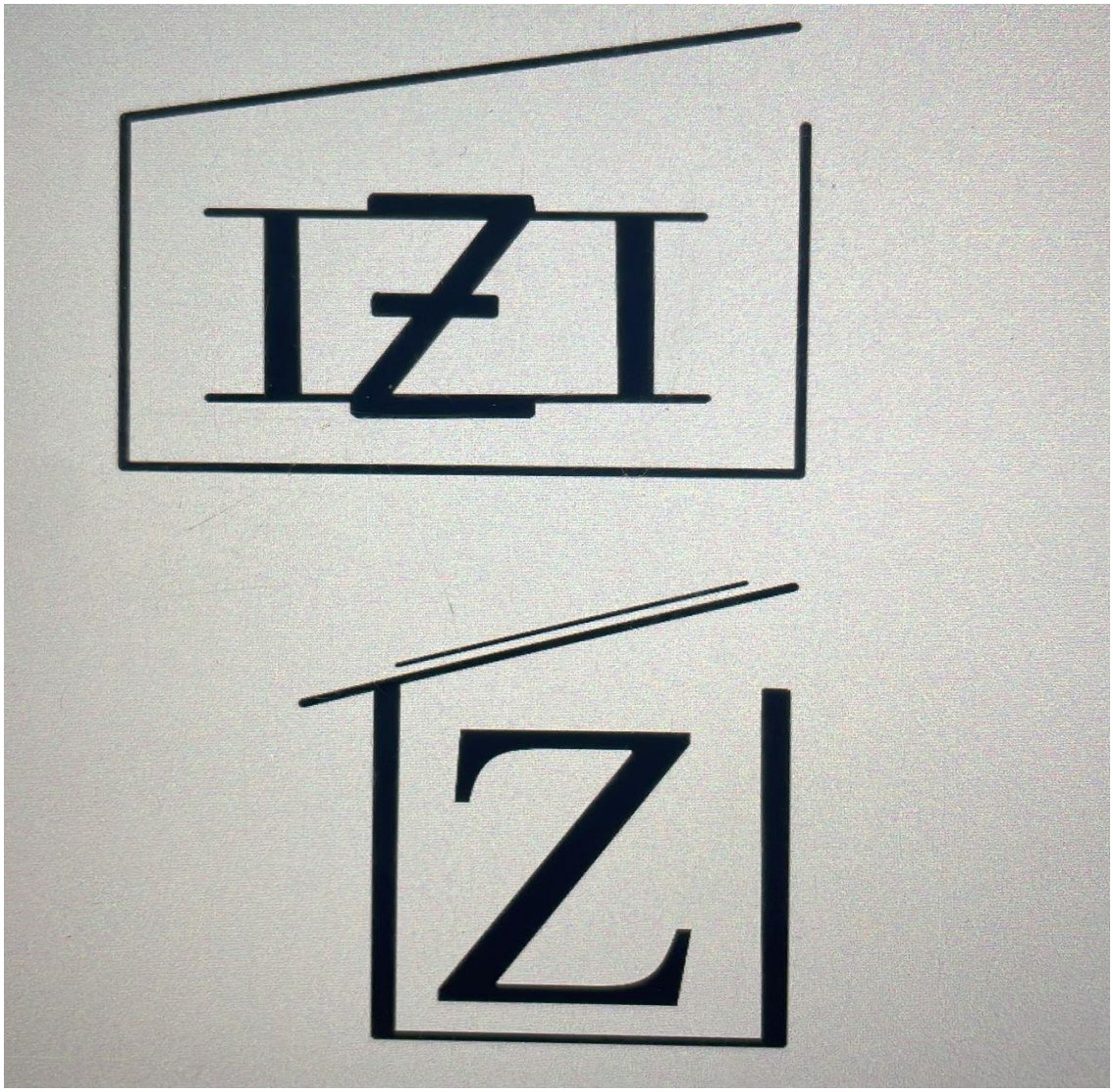

Figure 11: Sketches of the ideas for the logo

3.- Development

3.1.- Sprint 1

User stories

The user stories resolved in this sprint were:

- As a manager I want a tool that my workers can use to register what is happening in the warehouse (a new order arrived, a sale is completed, a container is filled…)
- As a manager I want to access the information of the contents of every container in the warehouse.
- As a worker I want to know when the product was placed into a container.

Development

In this sprint, I created the class Container. Moreover, I programmed the main activity of the application, which contains a list of containers and a menu that will lead you to the other activities in the future. As of right now, the menu only has the option to go to the AddContainer activity which is an activity that contains a form used to add containers to the containers list.

Lastly, I programmed the first table of the database to save the information of the containers.

This is the layout of the main activity. Here you can see the list of containers. When the main activity is created, it calls the DB Manager to obtain the list of containers. This list of containers is passed through a List Adapter. The list adapter's job is to fill each of the List View's items with the information of each of the container list's items. In this case, the information displayed is the code of the container, the name of the product and the date of arrival.

If you want to access the information of one container, you must press one of the items in the list and it will lead you to the ContainerInfoActivity.

As a sidenote I have to make clear that at this time in the development, the color palette being used was a placeholder. In the 7th sprint is where the correct color palette was introduced.

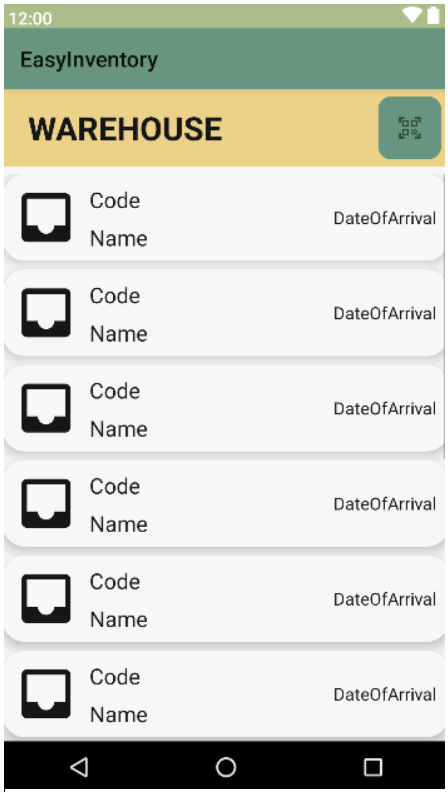

Figure 12: Main Activity layout

To start a new activity, you need a type of object called Intent. This intent can be filled with another object called Bundle. Moreover, this bundle can be filled with values like strings and integers. Those values inside the bundle can be accessed from the newly launched activity. In this case when an item from the container list is clicked in the main activity, the ContainerInfoActivity is launched but before it is all the information of the container is put inside the bundle and into the intent. When the second activity is launched all the information is rescued from the bundle using the method bundle.get() and showed on the screen.

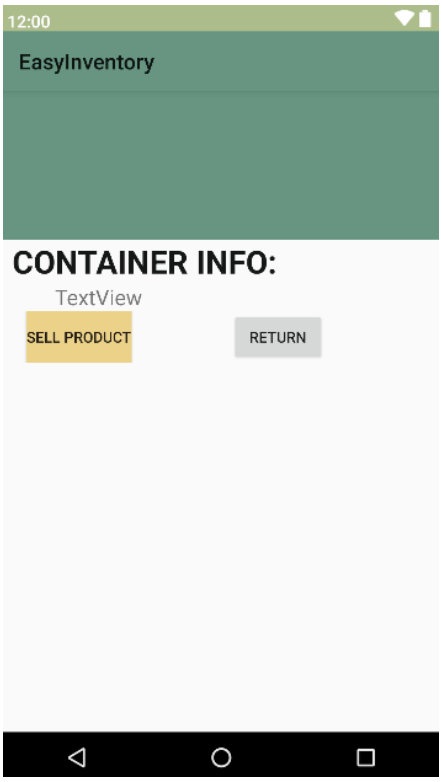

Figure 13: Layout of the container info activity

```
todoItemsAdapter = new ListAdapterContainers( context: this, containerList);
binding.warehouseListView.setAdapter(todoItemsAdapter);
binding.warehouseListView.setOnItemClickListener(new AdapterView.OnItemClickListener() {

    @Override
    public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
        Container container = app.getContainerList().get(position);
        Bundle bundle = new Bundle();
        bundle.putString("code",container.getCode());
        bundle.putInt("maxweight",container.getMaxWeight());
        bundle.putInt("currentweight",container.getCurrentWeight());
        if(container.getProduct()!=null) bundle.putString("content",container.getProduct());
        else bundle.putString("content","empty");

        Intent intent = new Intent(getApplicationContext(), ContainerInfoActivity.class);
        intent.putExtras(bundle);
        startActivity(intent);
    }
});
```

Figure 14: The piece of code that is responsible of launching the container info activity when an item in the list view is clicked

This activity is the AddContainerActivity. The layout consists of a form that the user fills to register the information on the new container to be added. When the button confirm is clicked, the DB Manager's addContainer method is called. If it returns true, the container has been added to the list successfully. If not, the container code already exists.

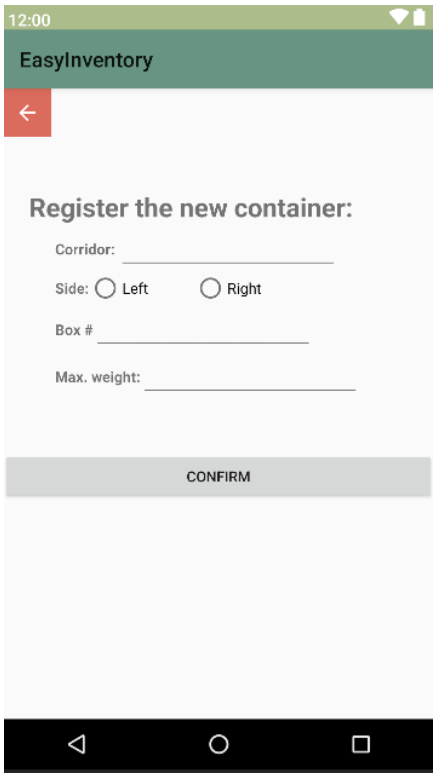

Figure 15: Layout of the add container class

```
//TABLE CONTAINERS
public static final String TABLE_CONTAINERS = "containers";
public static final String CONTAINER_CODE = "code";
public static final String CONTAINER_PRODUCT = "product";
public static final String CONTAINER_CURRENT_WEIGHT = "currentW";
public static final String CONTAINER_MAX_WEIGHT = "maxW";
public static final String CONTAINER_DATE = "dateOfArrival";

public static final String TABLE_CONTAINERS_CREATE =
        "create table containers(code TEXT primary key, product TEXT, currentW INTEGER, maxW INTEGER, dateOfArrival TEXT)";

public static final String TABLE_CONTAINERS_DROP = "drop table if exists "+TABLE_CONTAINERS;

public boolean addContainerDB(String code, int maxWeight){
    this.openWrite();
    ContentValues values = new ContentValues();
    values.put(CONTAINER_CODE, code);
    values.put(CONTAINER_PRODUCT, "empty");
    values.put(CONTAINER_CURRENT_WEIGHT, 0);
    values.put(CONTAINER_MAX_WEIGHT, maxWeight);
    values.put(CONTAINER_DATE, "-");

    if(_myDataBase.insert(TABLE_CONTAINERS, nullColumnHack: null,values) ==-1){
        System.out.println("ERROR IN THE INSERTION");
        this.close();
        return false;
    }
    System.out.println("SUCCESSFUL INSERTION");
    //this.close();
    return true;
}
```

Figure 16: The piece of code from the DB Manager class that contains the method AddContainer

Retrospective

As it was the beginning, I think I worked too slowly, and I can produce more in the same time span. A better sprint plan should be the solution.

3.2.- Sprint 2

User stories

- As a manager I want a tool that my workers can use to register what is happening in the warehouse (a new order arrived, a sale is completed, a container is filled…)
- As a manager I want to access the information of every order of product that comes into the warehouse.
- As a worker I want to know when the product was placed into a container.
- As a worker I want to know when a new product order is arriving at the warehouse.
- As a worker I want to be able to notify when a new order has arrived at the warehouse.

Development

I created the class Order to create instances of the information about the orders of products that come into the warehouse. Then, I programmed the behavior of the class, and I added the table of it to the database. I also programmed a method to generate an order ID so every order has an identifier, which will help in the future if I need to make methods that require searching for orders.

This is the layout of the orders activity. The button on the top right when clicked, changes its texts between “Not Arrived”, “Arrived” and “All”. This refers to the status of the order. If the button shows “Arrived” only the arrived orders will be shown and so on. The behavior of the list view is the same as in the main activity but for the orders list.

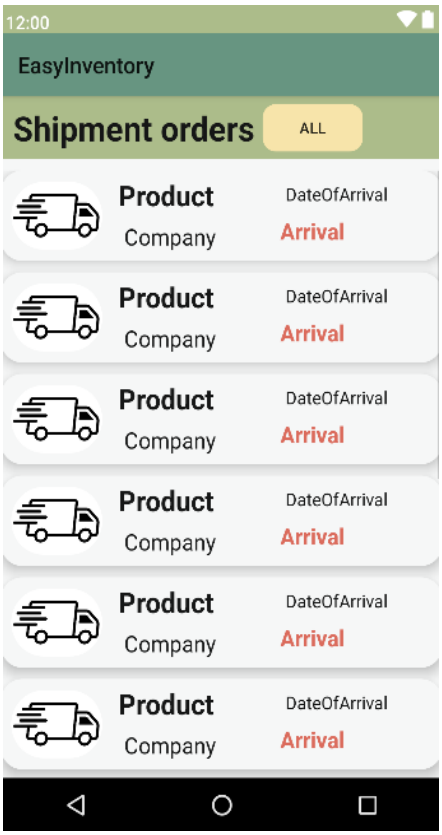

Figure 17: Orders Activity layout

This is the way new orders are put into the database. The buttons to set the time and date do not work yet, right now that information must be inputted manually.

The activity that shows the information of an order is not completed yet. The application can confirm the arrival of an order and fill a container, but the process is not fully programmed yet. I will show the layout when everything is finished.

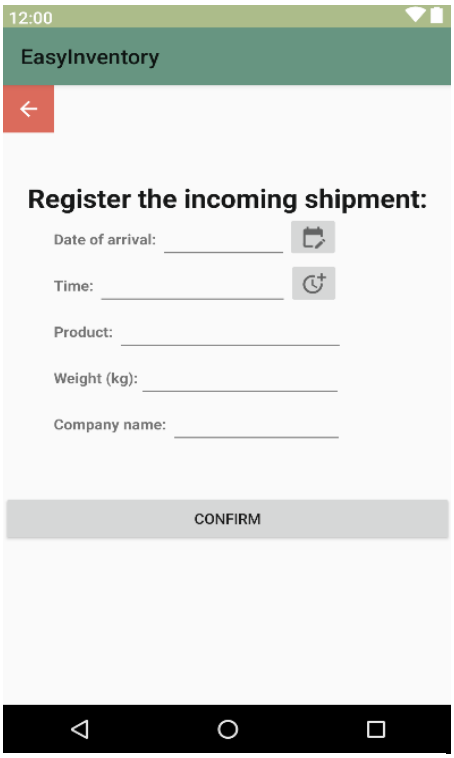

Figure 18: Layout of the addOrder activity

```
public String findContainerToFillDB(Shipment shipment){
    String code = "",containerProduct;
    int currentWeight = 0,maxWeight;
    Boolean success = false;
    Cursor cursor = _myDataBase.query(TABLE_CONTAINERS,
            columns: null, selection: null, selectionArgs: null, groupBy: null, having: null, orderBy: null);

    while(!success && cursor.moveToNext()){
        containerProduct = cursor.getString(cursor.getColumnIndexOrThrow(CONTAINER_PRODUCT));
        if((containerProduct.compareTo("empty")==0) || (containerProduct.compareTo(shipment.getProduct())==0)){
            maxWeight = cursor.getInt(cursor.getColumnIndexOrThrow(CONTAINER_MAX_WEIGHT));
            currentWeight = cursor.getInt(cursor.getColumnIndexOrThrow(CONTAINER_CURRENT_WEIGHT));
            if(shipment.getWeight()<=(maxWeight-currentWeight)){
                code = cursor.getString(cursor.getColumnIndexOrThrow(CONTAINER_CODE));
                success = true;
            }
        }
    }
    if(success){
        ContentValues values = new ContentValues();
        values.put(CONTAINER_CURRENT_WEIGHT, currentWeight+shipment.getWeight());
        values.put(CONTAINER_PRODUCT, shipment.getProduct());
        values.put(CONTAINER_DATE, shipment.getDateOfArrival());

        String selection = CONTAINER_CODE + " = ?";
        String[] selectionArgs = { code };

        int count = _myDataBase.update(TABLE_CONTAINERS,values,selection,selectionArgs);
        if(count==1){
            return code;
        }
    }
    return "";
}
```

Figure 19: method of the class DB Manager that has the task of filling a container with the contents of an order

The method findContainerToFill receives an order and goes throw the list of containers until it encounters one that either is empty or one container that already has the same product and enough space to fit the new order. I have already explained that I have decided to allow multiple orders from the same products to be able to fit into one container, because it allows to save space. If the client decides that this is not useful for them this is the place where I would have to change it.

Retrospective

Right now, the application has the option to order products and fill containers with them. To complete the path of a product through a warehouse I still need to add the ability to sell the product the containers are filled with.

This sprint was more packed with work, and I feel like the amount of work was optimal. I had some troubles with the programming of the database. In the end, I learned how to properly create the tables, search for a specific entry and drop them as I was having problems with those.

3.3.- Sprint 3

User stories

- As a manager I want a tool that my workers can use to register what is happening in the warehouse (a new order arrived, a sale is completed, a container is filled…)
- As a manager I want to save the information of every sale of the product of the warehouse.
- As a worker I want to know when a sale is being conducted.

Figure 20: Sale activity layout

Development

I created the class Sale to manage the information about sales in the warehouse. Following the same procedure as before, I also created the table in the database for sales and all the methods to work with it (add sale, create sale ID…).

The way to create a new sale is different as to how to create a new container or a new order. I have decided that a better process for creating a sale is going to a container's information and placing a button there to sell its contents or part of them (you can see the button in the Figure 13)That way, the information about the container (its code, the product it contains, the weight…) can be directly transmitted to the addSaleActivity instead of having to manually input it and create the possibility of the information not matching the container.

The sales are shown the same as the orders and the containers, an activity with a list view showing the relevant information in each list item and, if you click in one of them, it takes you to saleInfoActivity where you can see the full information about the sale (date of it happening, the product sold, the container code…).

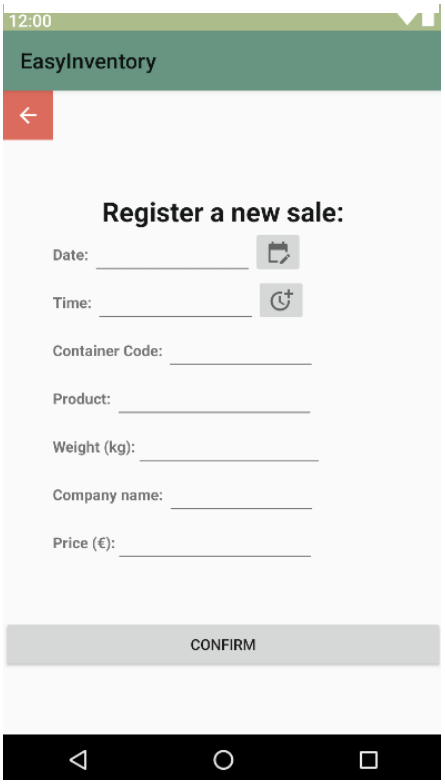

Figure 21: Layout of the activity addSale

```java
public int emptyContainerDB(String codeToSearch, int soldWeight) {
    String code;
    int currentW = 0;
    Boolean found = false;

    Cursor cursor = _myDataBase.query(TABLE_CONTAINERS,
            columns: null, selection: null, selectionArgs: null, groupBy: null, having: null, orderBy: null);

    while(!found && cursor.moveToNext()){
        code = cursor.getString(cursor.getColumnIndexOrThrow(CONTAINER_CODE));
        System.out.println(code);
        if(code.compareTo(codeToSearch) == 0){
            currentW = cursor.getInt(cursor.getColumnIndexOrThrow(CONTAINER_CURRENT_WEIGHT));
            found = true;
        }
    }

    if(found){
        currentW = currentW-soldWeight;
        if(currentW<0){
            return 2;
        }
        ContentValues values = new ContentValues();
        values.put(CONTAINER_CURRENT_WEIGHT, currentW);

        String selection = CONTAINER_CODE + " = ?";
        String[] selectionArgs = { codeToSearch };

        int count = _myDataBase.update(TABLE_CONTAINERS,values,selection,selectionArgs);
        System.out.println("CONTAINER updated");
        //this.close();
        return count-1;
    }
    else return 1;
```

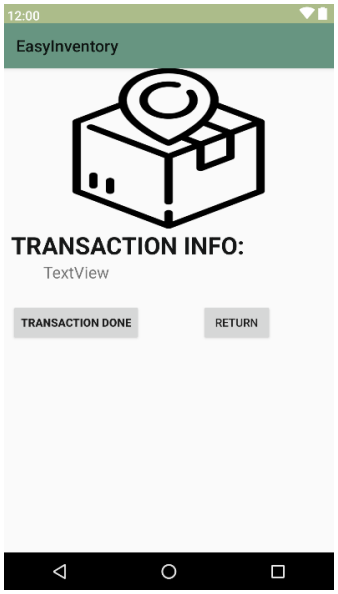

Figure 22: SalesInfo Activity Layout

Figure 23: The function in the DB Manager that has the task to empty the container when a sale is completed

This method is part of the code representation of the process described in Figure 9. The completion of a sale starts by executing this method. If the value returned is 0 the process was correct, and the container is emptied by the amount stated in the sale. If the value is 1 there was no container found for that sale. Else, the value returned is two and that means that the container has not enough product to complete the sale, therefore, more of that product needs to be ordered.

Retrospective

The workload was more than the previous sprint and the duration was a bit longer. The reason for it was adding the status attribute to the sales and the orders and all the methods that go with it.

Now the application fully represents the path of stock through the warehouse, and it is fully functional. But as the amount of data goes up, accessing the information is difficult. So, the aim from now on, is to make it easier for the user to visualize the status of the warehouse. Moreover, there are more features to add regarding the main activity.

3.4.- Sprint 4

User stories

- As a worker I want to be able to access the information of a container rapidly.

- As a worker I want to be able to notify when a new order has arrived at the warehouse.

Development

When a worker needs to access the information of the contents of a specific container, there has to be a physical way to do it. Because, if that is not the case, a worker would need to search manually the container in the list and click it and that is a lot longer than it needs to be. Some warehouses use chips, scanners and other types of solutions that require the purchase of new devices. For the type of business that is the target of my product buying more gadgets is not maybe an option, so I have to find another solution. The solution I propose is the following.

I am going to work with QR codes because every phone and tablet nowadays has the ability to recognize them, so the need to purchase new devices is gone. When a new order's product is placed in a container, the information of that container will update and a QR code with the new information (container code and order ID and information) will be generated. This QR code will be placed into a PDF file that the worker will need to print. The printed QR code will be stuck on the outside of the container. As you see, only a printer is needed and already a lot of warehouses have printers so there is the chance no more tools need to be bought to run my application.

In the application I have programmed an activity that is called QRScannerActivity. It is reached by a button in the main activity with a QR icon. When it recognizes a valid QR code this activity leads to the information of the container whose code was in the QR code.

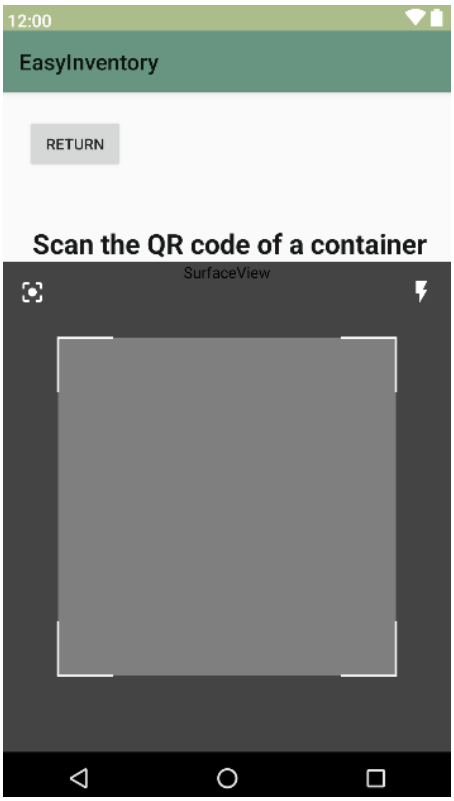

Figure 24: Layout for the QR scanner activity

```
public void fillContainer(View view){
    Shipment shipment = app.searchShipmentDB(_idShipment);
    try {
        String containerCode = app.findContainerAndFill(shipment);
        System.out.println("Code: "+containerCode);
        if(!containerCode.isEmpty()){
            System.out.println("Registering shipment arrival");
            if(app.shipmentArrival(_idShipment)){
                System.out.println("QRGenerator call");
                Bitmap bitmap = qrGenerator.generateQRCode(_idShipment,containerCode);
                try{
                    pdfGenerator.generatePDF(bitmap,shipment,containerCode);

                    Toast.makeText( context: this, text: "PDF successfully downloaded",Toast.LENGTH_LONG).show();

                    Intent intent = new Intent(getApplicationContext(), Warehouse.class);
                    startActivity(intent);
                    finish();
                } catch (Exception e) {
                    Toast.makeText( context: this, text: "PDF creation ERROR",Toast.LENGTH_SHORT).show();
                    System.out.println("PDF creation ERROR");
                    app.errorShipment(_idShipment);
                }
            }
            else{
                Toast.makeText( context: this, text: "Error in the update of the shipment",Toast.LENGTH_SHORT).show();
                System.out.println("Error in the update of the shipment");
                app.emptyContainer(containerCode,shipment.getWeight());
                app.errorShipment(_idShipment);
            }
        }
        else{
            Toast.makeText( context: this, text: "No container available",Toast.LENGTH_SHORT).show();
            System.out.println("No container available");
        }
    }catch (RuntimeException e){
        Toast.makeText( context: this, text: "Data base error",Toast.LENGTH_SHORT).show();
        e.printStackTrace();
        app.errorShipment(_idShipment);
    }
```

Figure 25: The method fillContainer of the activity with the same name. It is called when an order arrives to the warehouse.

This is the code representation of the process described in Figure 8. When an order is confirmed to have arrived, the method finds a valid container to fill and generates the QR code. Then, it creates a PDF file, puts the QR code inside and downloads it to the user's phone in the folder called 'Downloads'. If there is any error in this process the user is notified and the database returns to the state it was before the order had arrived. If that is the case, the order status will not update, and the order will be still deemed "Not Arrived".

```java
public class QRGenerator {

    public Bitmap generateQRCode(int shipmentID, String containerCode){
        MultiFormatWriter multiFormatWriter = new MultiFormatWriter();
        String information = "id: "+shipmentID+", container: "+containerCode;

        try {
            System.out.println("QR creation started...");
            BitMatrix bitMatrix = multiFormatWriter.encode(information, BarcodeFormat.QR_CODE, width: 300, height: 300);

            BarcodeEncoder barcodeEncoder = new BarcodeEncoder();
            Bitmap bitmap = barcodeEncoder.createBitmap(bitMatrix);
            System.out.println("QR successfully created");
            return bitmap;
        }catch (WriterException e){
            throw new RuntimeException(e);
        }
    }
}
```

Figure 26: QR generator class

The information saved in the QR code is the order ID and the container code where the orders contents are to be placed. What the generated QR code contains is no more than a string.

```java
    @Override
    protected void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_qrscanner);
        permissionCheck();
        CodeScannerView codeScannerView = findViewById(R.id.QRscanner);
        codeScanner = new CodeScanner( context: this, codeScannerView);

        codeScanner.setDecodeCallback(new DecodeCallback() {
            @Override
            public void onDecoded(@NonNull Result result) {
                runOnUiThread(new Runnable() {
                    @Override
                    public void run() {
                        app = (Aplication) getApplicationContext();
                        String information = result.getText();

                        //Information format id: #, container: #-X-#
                        String[] parts = information.split( regex: ",");
                        //int shipmentID = Integer.parseInt(parts[0].split(":")[1]);
                        String containerCode = parts[1].split( regex: ":")[1];
                        //because the resulting string contains an space at the beginning
                        containerCode = containerCode.replaceAll( regex: " ", replacement: "");
                        Container container = app.getContainerByCode(containerCode);

                        if(container != null){
                            Bundle bundle = new Bundle();
                            bundle.putString("code",container.getCode());
                            bundle.putInt("maxweight",container.getMaxWeight());
                            bundle.putInt("currentweight",container.getCurrentWeight());
                            if(container.getProduct()!=null) bundle.putString("content",container.getProduct());
                            else bundle.putString("content","empty");

                            Intent intent = new Intent(getApplicationContext(), ContainerInfoActivity.class);
                            intent.putExtras(bundle);
                            startActivity(intent);
                            finish();
                        }
                        else{
```

Figure 27: The code section from the class QR scanner that is responsible of decoding the string inside the QR code. When it decodes one, it obtains the container code and starts the activity containerInfo with the information of that container

Retrospective

This sprint was more about planification and research than about straight up coding. I did not know how to generate nor scan QR codes: I also had to learn how to create, fill and download a PDF file. To have it all work smoothly was not easy, I had to encounter errors like not asking for download and camera permissions, the QR code in the pdf being too small and then not being placed inside the page, the download directory not being correct… All in all, it was easier than I thought because there is a method that allows you to draw a QR code in a PDF by default in the class PDFGenerator, which made things a lot easier.

3.5.- Sprint 5

User stories

- As a manager I want a tool that my workers can use to register what is happening in the warehouse (a new order arrived, a sale is completed, a container is filled…)
- As a manager I want the tool to be easy to use and cheap to implement in the current workflow.

Development

In this sprint I have done the division between the roles of the worker and the manager. Once all the features for the worker have been completed and tested, it is time to divide the application in two and limit what the worker can do to reduce the stock misplacements and the database irregularities.

I designed a login activity and set it up, so it was the first activity shown when the application opens. The login has two options, worker or manager. If the manager option is selected, a password is needed to enter the application.

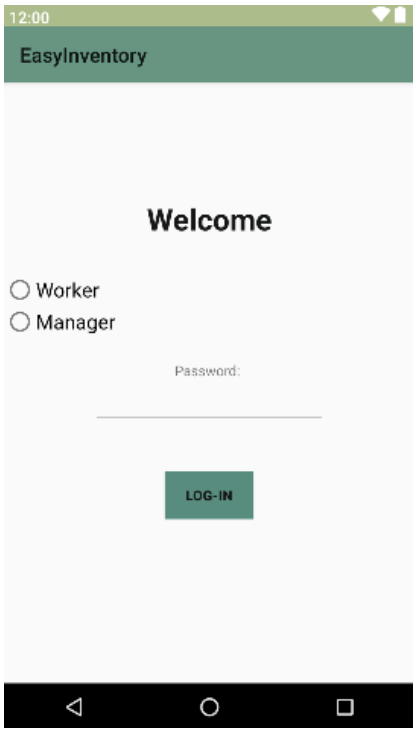

Figure 28: Login activity layout

And so, the division of tasks between worker and manager is done. The worker will have access to the containers, orders and sales lists, the ability to confirm the completion of sales and order arrivals and the ability to scan QR codes. The manager will be able to create containers, orders and sales and will also be able to access all the features that a worker can access.

To know the user of the session, a variable was added to the Application class and every time a new activity opens, that session attribute is checked to know what content needs to be shown.

```
        android:id="@+id/home"
        android:icon="@drawable/ic_baseline_home_24"
        android:title="Warehouse"
        />
    <item
        android:id="@+id/shipments"
        android:icon="@drawable/ic_baseline_arrivals_24"
        android:title="Shipments"
        />
    <item
        android:id="@+id/sales"
        android:icon="@drawable/ic_baseline_shopping_cart_24"
        android:title="Sales"
        />
    <item
    android:id="@+id/inventory"
    android:icon="@drawable/ic_baseline_inventory_24"
    android:title="Inventory"
    />
    <item
        android:id="@+id/addContainer"
        android:icon="@drawable/ic_baseline_add_box_24"
        android:title="Add Container"
        />
    <item
        android:id="@+id/newShipment"
        android:icon="@drawable/ic_baseline_add_business_24"
        android:title="New Shipment"
        />
    <item
        android:id="@+id/newSale"
        android:icon="@drawable/ic_baseline_add_shopping_cart_24"
        android:title="New Sale"
        />
    <item
        android:id="@+id/graphs"
        android:icon="@drawable/ic_baseline_insert_chart_outlined_24"
        android:title="Graphs"
        />
```

Figure 29: XML file that contains the menu items that are shown to the manager

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    if(app.getUser().compareTo("manager")==0){
        getMenuInflater().inflate(R.menu.menu_warehouse_manager, menu);
    }
    else{
        getMenuInflater().inflate(R.menu.menu_warehouse_worker, menu);
    }
    return true;
}
```

Figure 30: The function that checks the user to know which menu to show. It appears in every activity that contains a menu

Retrospective

This sprint was more about testing that the login was well done, and you cannot jump from the worker session to the manager session in case any error occurs. The login layout needs to be improved, but that can also be said to the whole application in general. So far, no bugs regarding session malfunction have been found.

3.6.- Sprint 6

User stories

- As a manager I want information about the capacity of the warehouse currently.
- As a manager I want to know the weekly performance of the sales activity.
- As a manager I want to be able to know what container is low in stock.

Development

As I stated before, when the number of containers, orders and sales goes up, accessing information becomes difficult. To help the manager know the situation of the warehouse, I added two new activities to the application: Inventory and Graphs. The idea of these two activities comes from the research I did at the beginning of the process. After research I decided to use Phil Jay's MP charts to do this.

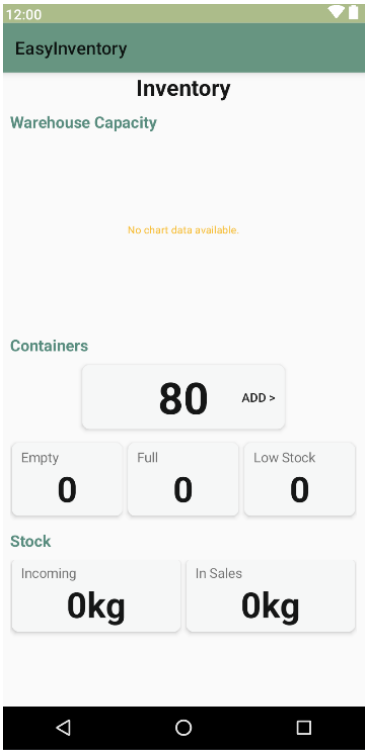

Figure 31: Layout for the inventory activity

The inventory activity's job will be to show the information about the current state of the warehouse containers. The information I have chosen to show is:

- The capacity of the warehouse: calculated by adding all the current weight at each of the containers in the warehouse and dividing it by the sum of the maximum weight every container can contain. All of that multiplied by 100 to show the percentage of the warehouse that is full.
- The number of containers in the warehouse (with a button to add more besides it).
- The containers that are full.
- The containers that are empty.
- The containers that are low in stock (20kg or less).
- The amount of product that is coming to the warehouse.

- The amount of product that is going to be sold.

```
public class InventoryActivity extends AppCompatActivity {
    Aplication app;
    Intent intent;
    PieChart pieChart;

    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_inventory);
        app = (Aplication) getApplicationContext();

        //WAREHOUSE CAPACITY
        pieChart = findViewById(R.id.capacityChart);
        pieChart.setUsePercentValues(false);
        pieChart.setDrawHoleEnabled(true);
        pieChart.getDescription().setEnabled(false);
        pieChart.setDrawEntryLabels(false);

        ArrayList<PieEntry> yValues = new ArrayList<>();

        int totalWarehouseSpace = app.getWarehouseSpace();
        int occupiedWarehouseSpace = app.getOccupiedWarehouseSpace();

        yValues.add(new PieEntry(occupiedWarehouseSpace, label: "Occupied"));
        yValues.add(new PieEntry( value: totalWarehouseSpace-occupiedWarehouseSpace, label: "Free"));

        PieDataSet dataSet = new PieDataSet(yValues,  label: " ");
        dataSet.setColors(ColorTemplate.LIBERTY_COLORS);
        dataSet.setDrawValues(false);

        DecimalFormat df = new DecimalFormat( pattern: "0.00");
        float perOccupied = (float) (occupiedWarehouseSpace*100.0/totalWarehouseSpace);
        pieChart.setCenterText(df.format(perOccupied)+"%");
        pieChart.setCenterTextSize(20);

        PieData pieData = new PieData(dataSet);

        pieChart.setData(pieData);
```

Figure 32: the code for creating the pie chart that shows the capacity of the warehouse in the Inventory activity

The graph activity, nonetheless, shows information about how the sales and the orders are doing. In this activity you can see how much you sell each week and the percentage of sales and orders that are completed.

Only the manager can access these two activities.

```
    //SALES BAR CHART
    salesBarChart = findViewById(R.id.salesBarChart);
    salesBarChart.getAxisRight().setDrawLabels(false);

    ArrayList<BarEntry> barEntries = new ArrayList<>();

    barEntries.add(new BarEntry( x: 0,app.getSalesPerDayOfTheWeek(Calendar.MONDAY)));
    barEntries.add(new BarEntry( x: 1,app.getSalesPerDayOfTheWeek(Calendar.TUESDAY)));
    barEntries.add(new BarEntry( x: 2,app.getSalesPerDayOfTheWeek(Calendar.WEDNESDAY)));
    barEntries.add(new BarEntry( x: 3,app.getSalesPerDayOfTheWeek(Calendar.THURSDAY)));
    barEntries.add(new BarEntry( x: 4,app.getSalesPerDayOfTheWeek(Calendar.FRIDAY)));
    barEntries.add(new BarEntry( x: 5,app.getSalesPerDayOfTheWeek(Calendar.SATURDAY)));

    salesBarChart.getAxisLeft().setAxisMaximum(20f);
    salesBarChart.getAxisLeft().setAxisMinimum(0f);
    salesBarChart.getAxisLeft().setAxisLineWidth(1f);
    salesBarChart.getAxisLeft().setAxisLineColor(R.color.black);
    salesBarChart.getAxisLeft().setLabelCount(4);

    List<String> xValues = Arrays.asList("Mon","Tue","Wed","Thu","Fri","Sat");
    salesBarChart.getXAxis().setValueFormatter(new IndexAxisValueFormatter(xValues));
    salesBarChart.getXAxis().setPosition(XAxis.XAxisPosition.BOTTOM);

    BarDataSet dataSet = new BarDataSet(barEntries, label: "Sales per day");
    dataSet.setColors(ColorTemplate.PASTEL_COLORS);

    salesBarChart.getDescription().setEnabled(false);
    salesBarChart.invalidate();
    salesBarChart.setData(new BarData(dataSet));

}
```

Figure 33: the piece of code in the class Graphs that is used to create the bar chart that shows the sales per day of the week

The dates are saved in the database as string. To know which day of the week each sale took part in, the function getSalesPerDayOfTheWeek goes through every sale in the database that has status “Done” and parses their date into a variable of type LocalDate using the utility of the same name LocalDate.parse(). Using the function date.getDayOfWeek().getValue() and comparing it to the day of the week, which is the entry parameter, you get if the sale was done in the day that the function is looking for.

Retrospective

These are by far the activities that are the most subject to change in the future. Choosing what information is important for the manager depends a lot on the business field you are in. These serve as a testament of what my application can do. It is more important to show that it can be done than what the graphs show.

In terms of workload this was one of the hardest sprints. I had to learn how to do graphs and charts with Phil Jay’s MP charts, then plan what information I wanted to show and make it work. The sales per week was the most difficult one because I had to redo

how the dates are saved in the database. Before the change the dates were saved as YYYY/M/D and now are saved as DD/MM/YYYY.

3.7.- Sprint 7

User stories

- As a manager I want a tool that allows me to reduce the stock that is misplaced.
- As a manager I want the tool to be easy to use and cheap to implement in the current workflow.

Development

The last sprint has consisted of refining the application to be on the level of any of the other applications you can see in the market of warehouse management tools. For this, I have divided the sprint in two parts, one larger one and one small one in the end. Between those sprints, some testing sessions were run with people from outside the project. The idea was to do an application makeover in the first part, get the feedback from the tests and then improve on the parts that the test subjects complained about.

In the first part a complete reimagination of the appearance of the application was made. It started by improving the color palette. The previous color palette had a pastel green base with yellow details, but I think it was not fitting the application vibe I wanted to go with. Therefore, I decided the colors that currently are on the application suit it better (more detailed explanation in the *Analysis and Design* section). Also, I learned that the use of absolute white and absolute black was not recommended.

As I was doing research, I noticed that there is a trend in modern UI design to have a bottom app navigation menu, you can see this in WhatsApp, Instagram, Uber… However, my

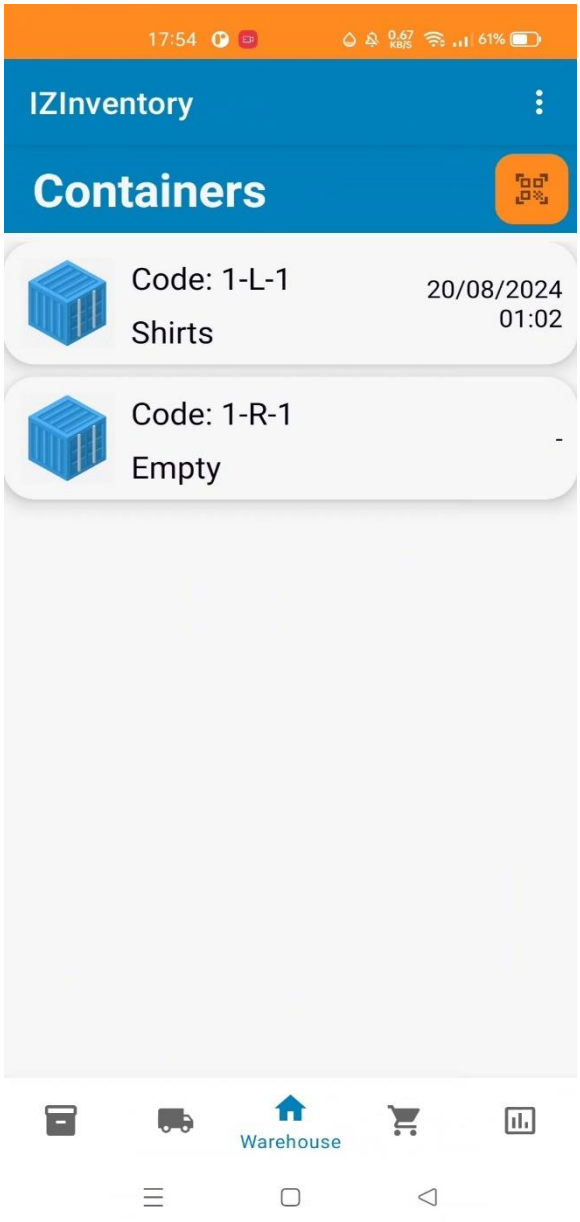

Figure 34: The main activity with the bottom navigation menu implemented

app only had a top right menu with all the activities you can launch from the main. I noticed that there were five activities (warehouse, order list, sale list, inventory and graphs) that were far more important than the others and it made a lot of sense to have those five activities at the bottom so you could move between them with ease. This meant making changes in every activity as, from now on, there was going to be one main activity and five fragments. Tere were a lot of changes to be made to adapt the application to the new philosophy. Accessing the context of the application in each fragment was the main issue. It is needed to access the database (calling the Application class), to print the on-screen messages (with Toast) and when calling the List Adapter (the class used to transform the Array List data into items in the List View) of each of the three fragments with lists. Also, the layout type I was using for each of the fragments (the Constraint Layout) was not working properly, it made the top app bar appear in the middle of the screen, so I changed them to a Frame Layout. For the worker only Warehouse, Orders, and Sales appear in the bottom navigation menu, and they have no top right menu. For the managers, the navigation menu also shows the two remaining options, and they also have the top right menu with the options to launch AddContainer, AddSale and AddOrder.

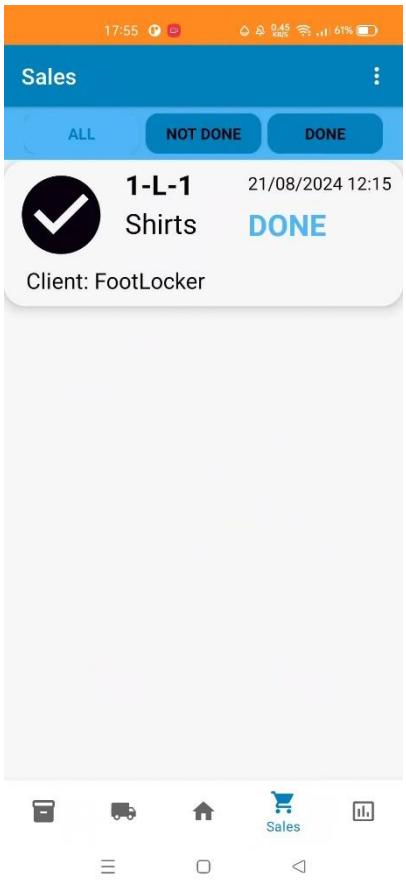

Figure 35: the new layout of the sales activity

However, the changes were not over. I made a rounding style for the buttons, and I changed their colors to match with the app. I did a makeover of the forms in AddContainer, AddOrders and AddSales making them symmetric and more modern. Now you can set the time and date of the order arrivals and sales with a calendar view and a clock. I changed the type of measurement for all the elements in the app from dp to pt, so they adapt better to different screen sizes. The radio button of the login activity where you choose whether you login as a manager or a worker was also made over. Lastly, I improved the icon assets of the application, that speak more to the user and are more related to what the user is seeing.

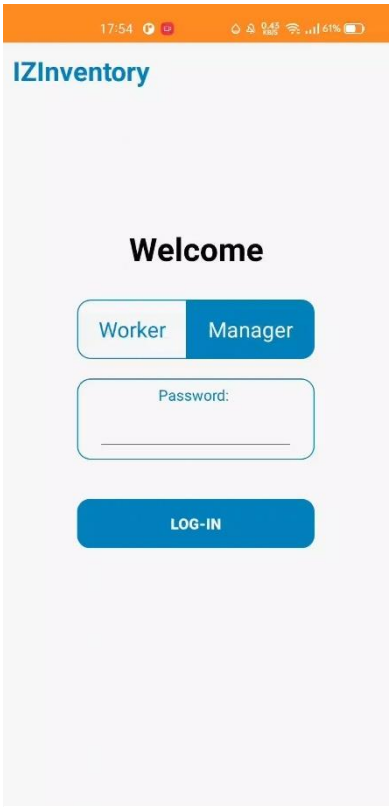

Figure 36: the new layout for the Login activity

After this, I did the test days, and I improved on what the subjects told me. This process is explained in the *Verification and Validation* section later in the document.

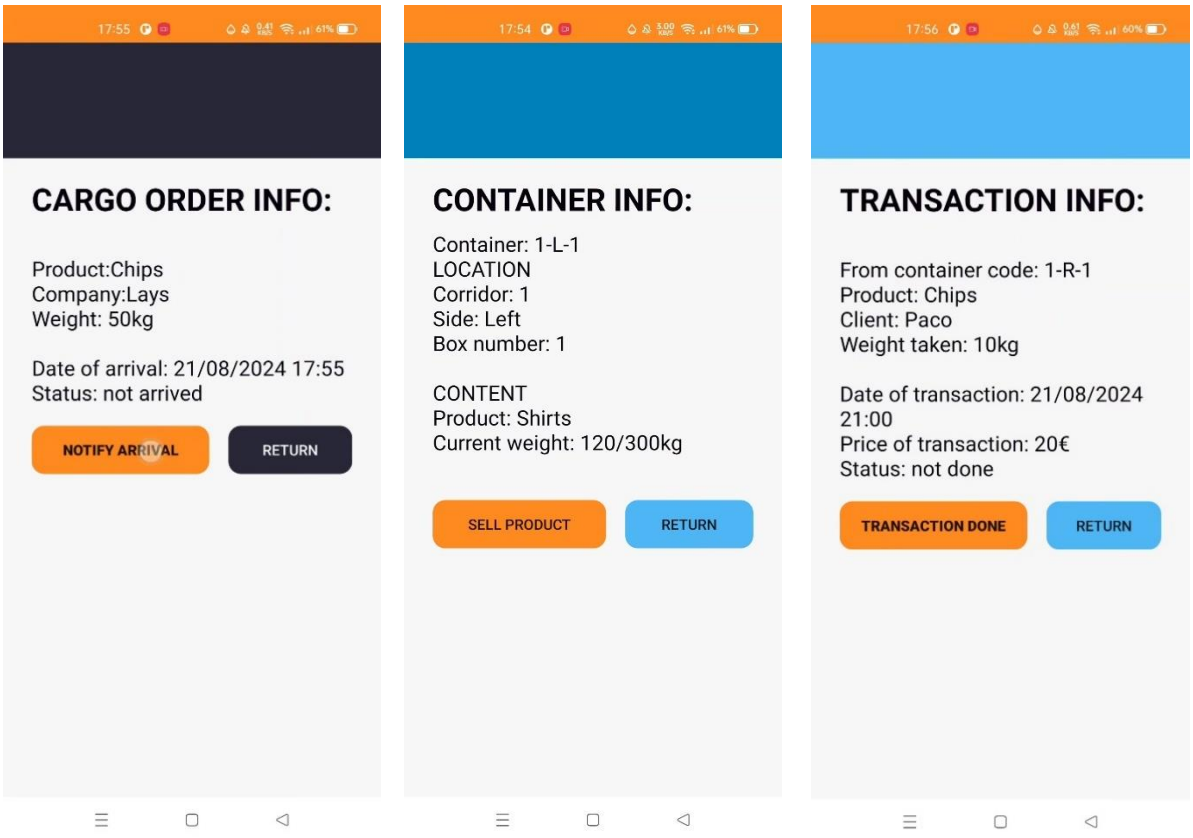

Figure 37: The result of the layout makeover on the info activities

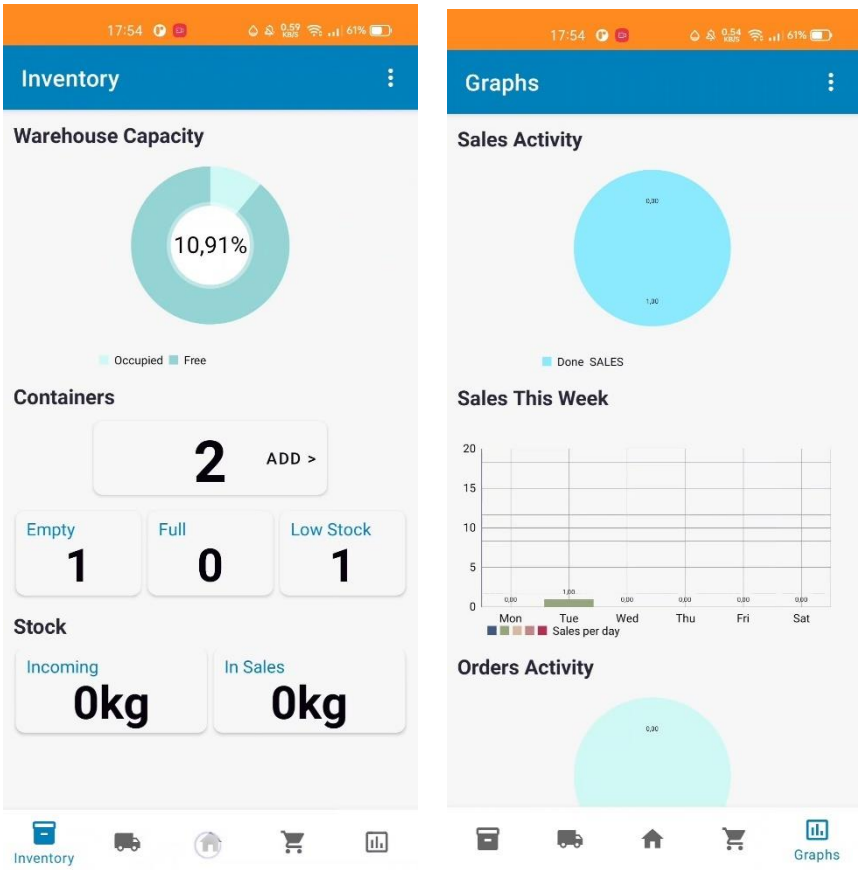

Figure 38: The new look of the Inventory and graphs activities

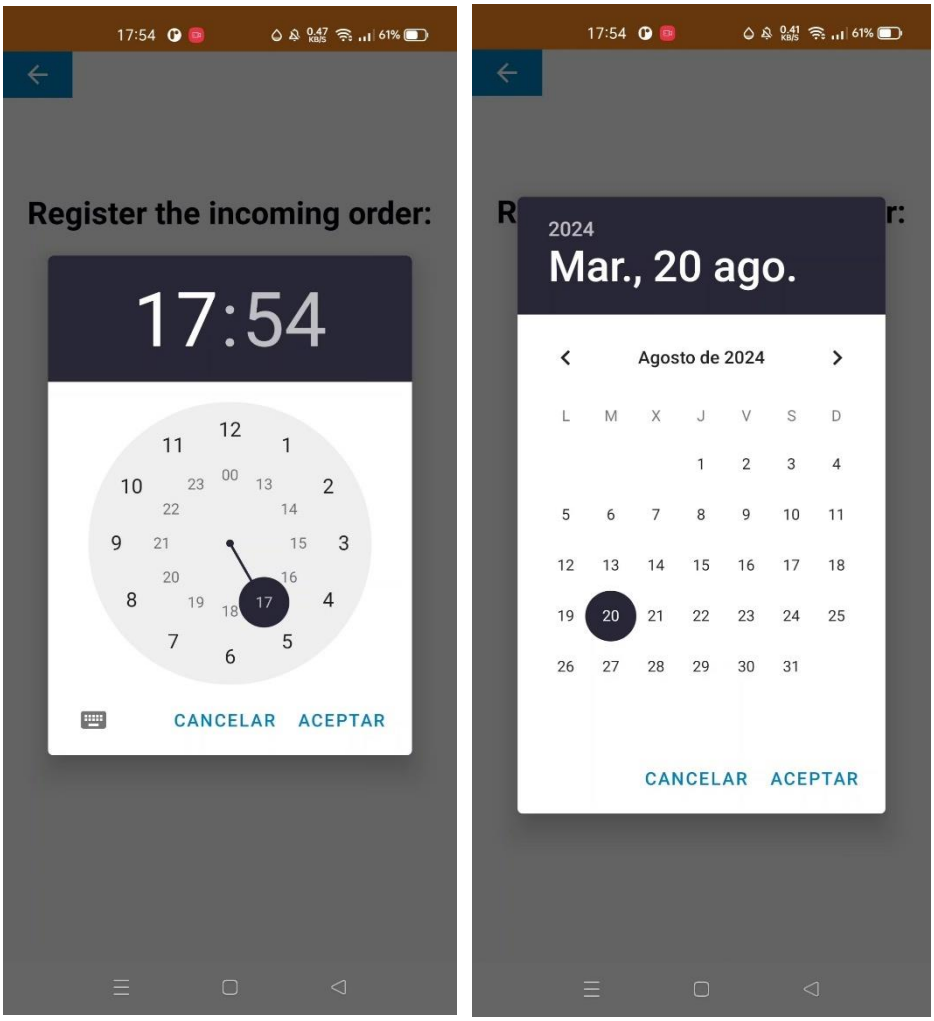

Figure 39: date and time pickers for the addOrder and addSale activities

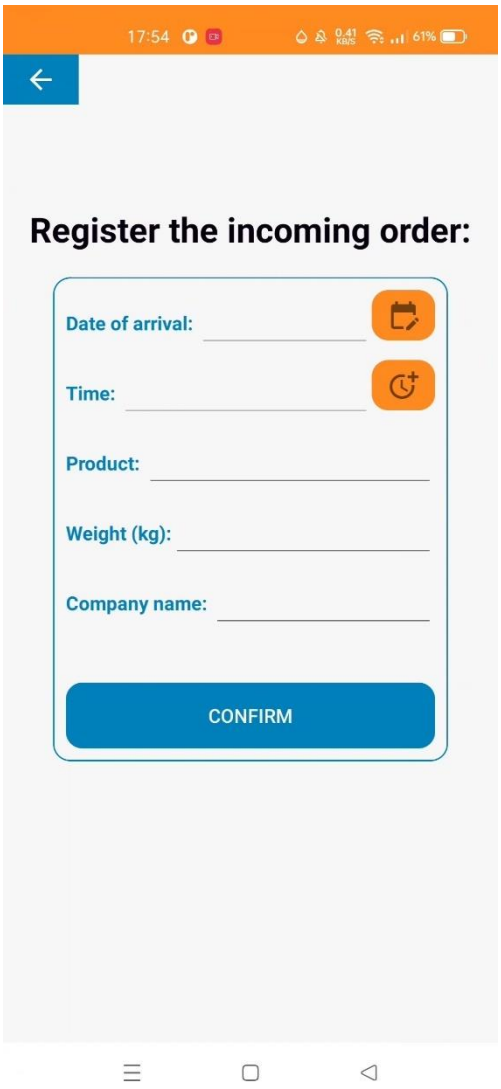

Figure 41: screenshot of the layout of addOrder, showing the new style for the forms in addContainer, addOrder and addSale

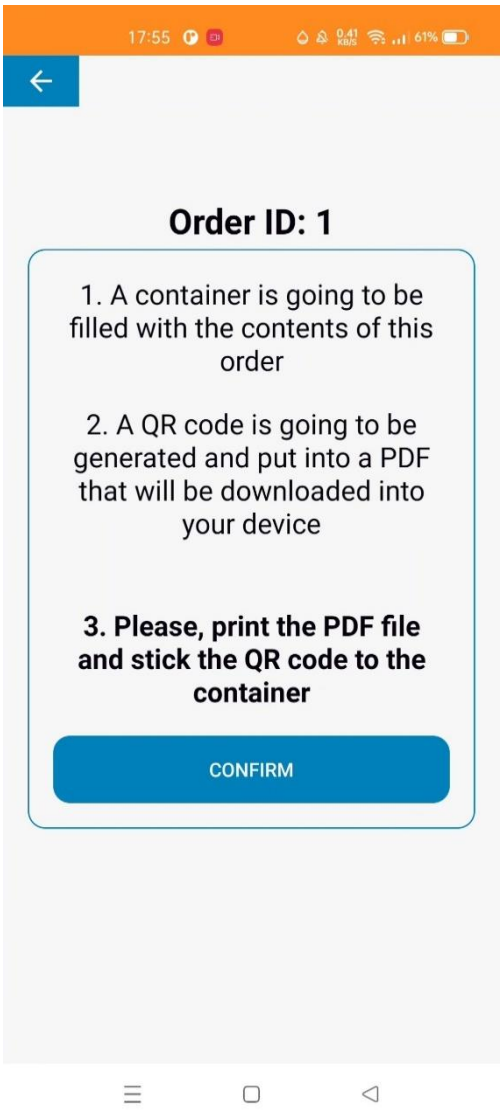

Figure 40: the new layout of the activity fill container

Retrospective

Although the workload was the largest among any of the sprints, I am very satisfied with the results. The application is now looking like it is ready to be out on the market. I improved my designing skills by a wide margin, and I understood the importance of designing a modern and clean UI.

4.- Verification and validation

Validation of the interface design

To validate the interfaces I have designed, I looked at the Material Design Guidelines published in the Android developer's website, and I meet a lot of them.

I use cards to organize content. I also use lists to organize vertical content. I have implemented a bottom navigation bar that lets the user move through my app. I allow users to input information via text and other fields like calendars and clocks. I use Toast to communicate the user important information. I also use shadows in the cards previously mentioned to differentiate them from the background. I use vector drawables, for example for the menu and bottom navigation icons and in the list items for sales. The use of vector drawables is recommended because they are scalable without losing definition. I also have a menu on the top right, indicated with three dots, as per usual.

"With these elements I provide my users with a familiar experience". (Android, n.d.)

However, there are elements that I do not use in my app. I have not implemented any transition animations between activities, and I do not have any floating action buttons, which are very common in apps nowadays.

The Android Developers webpage has also a section dedicated to app quality called "Core app quality" where I can test if my application meets al the minimum requirements. The tests are the following:

Visual and functionality tests

- The app supports standard back button navigation and does not make use of any custom, on-screen "Back button" prompts (**test passed**).
- The app supports gesture navigation (test passed).
- The app correctly preserves and restores user or app state (test not passed, when leaving the app and returning the app returns to the login screen).
- Notifications tests (My app does not require the use of notifications)
- The app supports landscape and portrait orientations and folded and unfolded device states (test passed).
- App fills the app window in both orientations and is not letterboxed because of configuration changes, including device folding and unfolding (test passed).
- The app correctly handles rapid transitions between display orientations and device folding and unfolding without display rendering problems and without losing state (test passed).
- The app displays graphics, text, images, and other UI elements without noticeable distortion, blurring, or pixelation (test passed).
- The app displays text and text blocks in an acceptable manner for each of the app's supported languages (test passed).

- The app's content, and all web contents referred to by the app, support dark theme (test unsuccessful, the app works on dark mode but the color contrast between elements is not acceptable).
- Touch targets should be at least 48dp in size (test passed).
- The app's text and foreground content should maintain a high enough color contrast ratio with its background (test passed).
- The app describes UI elements, except text views, with content descriptions (test not passed).

To end the project, I needed to test every feature in the application. I had tested every feature when it was added, following the sprint structure, but the app had changed so much that there could be some holes in my code and there is no such thing as too much testing. I decided to run some tests with subjects other than myself because I know how the application works, and I am biassed in the sense that I do things in the order I coded them to go. However, how someone uses the application for the first time tells a lot about whether it is designed the optimal way or not. My idea was to test subjects with different knowledge of how the apps are programmed and developed so I did the tests with my mom and some of my friends. None knew how the app worked or looked like before the test, so I told them to tell me everything that went through their minds out loud and I took notes.

4.1.- 1st Test: Adding a new container to the list

The prompt given to the subjects was: Log in as a manager and create a new container.

Everyone completed the assigned task. They said that the process was straightforward, and the appearance of the application was nice. Predictably, my mom was the one with the longest complete time. None added the container by going to inventory and clicking on the button I placed there, but it was expected, as it was more accessible to click on the menu on the top right.

A couple of subjects had trouble understanding what the container code pieces meant (corridor, side and box number), but others understood it perfectly, so I blame it on English not being their first language.

I had to change the color of the text in the confirm button on the form in the class AddContainer, which was in black, and the button had a blue background, so it was not readable. Therefore, I changed the text color to white.

The other thing I had to change which took me more time was that when a container is empty the item in the list shows the text "empty" and instead is more aesthetically pleasing if the first letter is in uppercase. I had to change a couple of methods and reset the database for the change to work.

4.2.- 2nd Test: Create a new order

The prompt given to the subjects was: Log in as a manager and create a new order.

Logging in the second time was faster than the first. Everyone completed the task very fast, as they had seen the option to add order when they opened the menu to add a container.

They complained about two things. The same error as before (black colour text on a blue button) and a more complicated error the format for the time and date was not being shown properly. When a time with single digit minutes or hours was picked the clock showed that the time picked consisted of four digits (HH:MM) but the time shown on the screen after picking was not like that (H:MM, HH:M or H:M depending on which part was single digits). So, I had program that the format was always HH:MM and, if either of the parts was single digits, that a number '0' is added so the format stays consistent.

Something similar happened when picking a date. The format did not stay consistent when the month or the date picked were less than 10. So, I also programmed that a number '0' is added to keep the format always dd/mm/yyyy.

Because of this, I had to make changes on how the dates were parsed when gathering the data for the chart of sales per day of the week in the Graphs activity.

4.3.- 3rd Test: Confirm the order arrival

The prompt given to the subjects was: Log in as a worker and obtain the QR that confirms the arrival of an order.

This test took more time for the participants to finish. Firstly, some of them were confused when logging in as a worker. They were trying to enter a password even though the password field was not enabled, they did not seem to notice the difference. To fix this, instead of using the method setEnabled(false/true) of the objects EditText and TextView, I changed it to setVisibility(INVISIBLE/VISIBLE). The password field is now invisible when the worker option is selected and, only when the manager option is selected, it becomes visible. The response of the participants was very positive, as none seemed to have troubles logging in after the change.

My mom had trouble understanding she had to enter the item in the order list to confirm its arrival, but ultimately, she completed the task.

4.4.- 4th Test: Create a new sale and complete it

The prompt given to the subjects was: Log in as a manager and create a new sale. After that, log in as a worker and confirm the transaction has been completed.

In this test we found a big bug that made the completion of the sales not work as intended. When trying to complete a sale, in the case of an error, a method called errorSale is executed that restores the information of the database as it was before trying to complete the sale. The problem was that an error I made in how the exception in the case of an error triggered, made the error method execute always. This meant that the sale was deemed “not done” but the product was being taken out of the container.

After reprogramming the exception block, everything is working fine.

4.5.- 5th Test: Check if the process registered in the inventory and the graphs

The prompt given to the subjects was: Log in as a manager and check if the statistics have updated correctly.

This test was short but revealed that the change I made in the format of dates broke the table of sales per week. After a change in the format of the parser, the chart is now working as intended.

5.- Conclusions and future lines

The future for this project is adapting this application for a specific business situation. Not all warehouses are the same and not every one of them needs the same features. The tool that I developed at the end of this project is a basis that can rapidly be adapted to the needs of any client. What I am trying to say is that the application that I have now will change a lot depending on what the business we are working for and what this business sells.

If I were to be contracted by an enterprise that sells fruit, for example, I would probably need to put a lot of emphasis on the date that every order has arrived, because if the fruit stays in the warehouse for too long it would rot. Therefore, I would need to develop some type of solution so the manager can clearly know when to sell the product (changing the item color in the list of containers, for example).

Maybe the business that I will be adapting the app for accepts refunds, so I would have to sort that problem out too. Or maybe the client wants to have a web application for the manager that is linked to the mobile application that the workers use. The container's code can also change a lot depending on how the boxes in the warehouse are placed or if it even has boxes.

One big change that would lead me to improve the application a lot would be having the list of all the different products that can be stored in the warehouse beforehand. With this list in mind the number of manual inputs on the application would go down and, hence, the human errors would plummet. I could also set a price (per kilo or per unit) for every product and the prize of every sale could be calculated automatically.

On the other hand, when programming the application, some other ideas came to mind that if the client were to like them could also be implemented. For example, instead of having a list of orders and sales, I could program an activity with a calendar that showed when orders are arriving, and sales are scheduled to be done. Also, I thought about creating notifications that pop up one hour before an order is scheduled to arrive. The client could also opt for another interpretation of how the QR codes work. Instead of having to print them every time an order arrives, I could program the app, so each container has its own container QR code, and you only print them once.

A client could also be interested in identifying every worker with an ID and a profile, to have the information of which worker has confirmed the arrival of an order or the completion of a sale.

In conclusion, after concluding this project, I have acquired the knowledge and design skills to build an application with a modern UI. The application is ready to be adapted to an actual work environment and as projected at the beginning, the app is scalable, easy to use and covers everything a business would need to manage the logistics of a warehouse.

6.- Bibliography

Ahmed, Z. (n.d.). *Integriti*. Retrieved from 20 Easy ways to spice up UI designs: https://www.integriti.io/post/20-easy-ways-to-spice-up-your-ui-designs

Android. (n.d.). *Android Developers*. Retrieved from Guide to App Architecture: https://developer.android.com/topic/architecture

Capterra. (n.d.). Retrieved from Warehouse managment software rankings: https://www.capterra.com/sem-compare/warehouse-management-software/?utm_source=ps-google&utm_medium=ppc&utm_campaign=:1:CAP:2:COM:3:All:4:INTL:5:BAU:6:SOF:7:Desktop:8:BR:9:Warehouse_Management&network=g&gclid=CjwKCAjwuMC2BhA7EiwAmJKRrM17Crmy4seQT-brJ6S8lI

Easy One Coders. (n.d.). *Youtube*. Retrieved from Android Charts | Bar Chart | MP Android Chart | Android Studio 2024: https://www.youtube.com/watch?v=WdsmQ3Zyn84

Free Learning Platform For Better Future. (n.d.). Retrieved from Learning SQLite: https://www.javatpoint.com/sqlite-advantages-and-disadvantages#:~:text=Reading%20and%20writing%20operations%20are,the%20file%20which%20was%20changed.

Gartner Peer Insights. (n.d.). Retrieved from https://www.gartner.com/reviews/market/warehouse-management-systems

GeeksForGeeks. (2024, April 4). Retrieved from https://www.geeksforgeeks.org/agile-methodology-advantages-and-disadvantages/

Jay, P. (n.d.). *GitHub*. Retrieved from MPAndroidChart: https://github.com/PhilJay/MPAndroidChart?tab=readme-ov-file

Laoyan, S. (2024, February 8). *asana*. Retrieved from https://asana.com/es/resources/agile-methodology

Lily, T. (n.d.). *Pexels*. Retrieved from https://www.pexels.com/es-es/foto/hombre-fabrica-almacen-negocio-4483774/

Olsen, J. (2013). *Color Meanings*. Retrieved from https://www.color-meanings.com/#blue