*Workflow:*

The workflow starts with a customer placing an order over the phone. Cashier then starts a new order and types in the customer’s identification. If the customer is found in the system, the sale is created and items are packed and sent to the customer, else a new customer has to be created.

There are two types of payment. The customer can either pay upfront and receive an invoice at the time or pay for the all the things bought at the end of the month. He then would be sent an invoice to be reminded of it and to know how much exactly he has to pay.

Customers who pay for items upfront also receive an invoice at the end of the month but theirs say that they have to pay nothing for this month.

**Functional Requirements:**

*Use Case Diagram:*

The system should handle three actors, who are seen as roles in the system.

The first one is the warehouse workers. They are able to get item locations to find them quickly and efficiently. Then mark when items are packed and loaded into a truck and sent.

The second one is the Cashier. Actors in this role are able to adjust information about items and customers but their main priority lies in making sales and registering payments after they’re done at the end of the month. Send invoice use case is also connected to Cashier, however, they do not send it themselves. The system does it automatically and the actor can only find invoices to check on them or register payment.

The third and topmost one is the managers. They have access to generate statistics to gain information about profits and what is being sold. Also they can make purchases from suppliers when items are out of stock and edit information about suppliers, most often add or update whether they’re active or not. Furthermore managers can edit customer and item information along with cashiers.

*Brief Descriptions:*

There are three main actors in the use case diagram; Manager, Cashier and Warehouse worker. Below are brief descriptions of the use cases as seen in the use case diagram.

Use Case: Generate statistics

The manager would like to generate and view statistics of the business. He/she then checks the system to view information such as number of orders made, how many items were sold, how many items are in stock, customer and supplier information. The system displays this information.

Use Case: Supplier CRU

The manager identifies a supplier and makes an agreement with it (the company). Then he/she adds the information to the system and can view or update the supplier data e.g. name of the supplier.

Use Case: Employee CRU

The manager hires an employee. He/she then creates an employee object in the system and can view or update the employee information e.g. address.

Use Case: Make Purchase

The manager has identified an item he would like to buy from a supplier. He/she checks if the supplier exists in the system and if so orders the desired item.

Use Case: Customer CRU

A customer calls or sends an e-mail to Entafarma. A contract is made between the two and a manager creates customer in the system. Then he can view or update its information e.g. name of the customer.

Use Case: Item CRU

Manager orders new items from a supplier. He/she then creates the item in the system before or as soon as it gets to the warehouse. He can also update information about the item if need be.

The cashier can read information about the item in the system.

Use Case: Create Sale

A customer calls or sends an e-mail to Entafarma to place an order. Cashier checks if customer is in the system and starts a sale by adding items. When all items are added he/she specifies the payment type (upfront or at the end of the month) and finishes the sale.

Use Case: Send Invoice

A cashier checks the system to see if an invoice was sent to a customer. If invoice was somehow not automatically sent, the cashier re-sends the invoice by clicking on “send invoice” button and sends the invoice.

Use Case: Register Payment

A customer pays for the sent invoice. Cashier finds that specific invoice and marks it as paid in the system.

Use Case: Get Location

Warehouse worker enters a sales number in the system and the system returns him item names with their locations in the warehouse.

Use Case: Pack Order:

Warehouse worker packs the items that he received locations for. When it is done he types in the sales number and marks it as packed.

Use Case: Send Order

Warehouse worker carries the packed items to the truck and once it leaves marks that the order is sent in the system

\* As noted in the use case diagrams and brief description, “CRU” has been used instead of “CRUD.” This is because the delete function is not implemented for it would cause issues in the database. For example, we do not want to delete an employee from the system because it is connected to a sale. This means that if an employee is deleted, the sale will have an issue as there would no longer be a reference to the employee who made the sale.

*Domain Model:*

In Employee there is a person\_id which represents a personal identification number which each person in Lithuania has. It is similar to Denmark’s cpr number, however, there it is called Personal Code.

The Customer has no such field because Entafarma deals mostly with other companies. However, there are a few private pharmacies which don’t constitute as such so the class name is Customer. Both of these clients are not people so their names are unique and can serve as identifiers within the system.

In the Sale class there is an attribute called discount. It is used to give customers who have spent a certain amount of money at Entafarma, a discount. The other fields in the Sale class, besides the saleNr, are set to null on the creation and only when the actions attributed to them are made, these fields are updated by a worker.

Multiplicity between Sale and Invoice is zero to one because the sale is created first and only then the invoice.

SaleLine contains the quantity of items and the price of one. This is required to know at what price the products were sold since the amount in the Item class can be changed a couple of times through the year.

Items have a unique identifier barcode. They also have stock to know when there’s a need to resupply. itemType is to show whether the item is liquid or pills while category describes the purpose of the medicine - vitamins, antibiotics, etc.

The company has one huge warehouse so there’s a need for Location class to find everything without wasting time searching every time. To make this as efficient as possible the company made it that on one shelf, only one type of items are stored even if it isn’t filled up. However, if there are more items of the same type than the shelf can hold, they are reserved another one. This is shown in the multiplicity.

PurchaseLine is similar to SaleLine. However, PurchaseLine is for buying items instead of selling them.

Purchase has the dates of when the order was placed and when received to be able to tell how quickly the items are delivered.

In the Suppliers there is a field called active. It can be TRUE or FALSE depending on whether the company still orders from that particular place or not.

*Prioritisation List:*

Out prioritization list consists of two criteria - complexity and business importance. The first one stands for how hard it is to implement a use case while the second its relevance to the company.

Both criteria have been ranked for maximum significance. The highest ranking is six and the lowest is one. The lower the rank, the more difficult or crucial the use case is. After these numbers are multiplied, the score derived decides the use case’s place in the prioritization list. If there are use cases with the same score, they are put in the order that is best for programming.

*createSale FullyDressed*

This diagram shows the process of making a sale. First the customer is found by his id. Then a pop up window appears where the employee types in item barcodes. All the sales are done through phone or by email so that is the information being provided by the customers. After that the cashier chooses the payment type (upfront or monthly) and finishes the sale.

*createSale SSD*

The use case consists of three methods. The first one starts the sale without requiring any input.

Then an item is found by its barcode and its quantity inserted. This is repeated until all the times have been added.

The last method creates a sale object with all the information gained from the previous methods and the employee object and payment type (upfront or at the end of the month). Once this is done the sale object is passed to the database, then saleLines and the createSale method is finished.

*createSale Operation Contracts*

All three methods described above are explained in the operation contracts below. They present what changes after each method and what associations the new sale receive.

It also shows that an ArrayList of saleLines is created in the first method startSale which takes no input.

**Supplementary Specification**

*Relational Model*

The relational model displays the structure of the database. One star means primary key while two stars stand for foreign key. Arrows from them point to locations the keys are taken from.

In this database all tables have an id. This is a precaution in case the system will be updated and many things will have to change. For example if customer names were used as primary keys and then changed their name, the whole database would have to be edited since other tables (sale and address) would point to nothing.

*createSale Communication Diagram*

This diagram presents the inner workings of the system, the use of three layered architecture as detailed in GRASP pattern.

At the top there is graphical use interface (GUI). Its main purpose is to present an easy to use and understand interface while all the information collected from the user input is sent to the controller layer(controller pattern from GRASP).

This layer than processes the information and starts doing the actions requested by the user as it is also the information expert. In this case it starts a sale by creating an empty arrrayList of saleLines. Then a saleLine has to be made but that method is already implemented in saleLine controller (CtrSaleLine). That is why to prevent code duplication the sale controller sends a request to the saleLine controller. This increases coupling but the benefits gained from high cohesion in this case far outweigh the losses.

*Design Class Diagram*

The diagram here shows the way the system is implemented. However, it is too large to fit well within the page so it is only possible to see the program’s outline but not the methods, attributes or their types.

Yet there is one thing that needs mentioning. In the database layer, all of the classes are used with interfaces. This is done so that the method implementation can be changed in future updates without having to adjust the whole program.