Vestbjerg Byggecenter System

Semester 1 project

University College of Northern Denmark

AP Degree - Computer Science

Winter Semester 2021/2022

24.11.2021 - 14.12.2021

Group #3

UCN, University College of Northern Denmark
IT-Programme
AP Degree - Computer Science
CSC-CSD-S211
Group 3

Vestbjerg Byggecenter System

Repository: https://github.com/tomassabol/UCN-semester1-project, version (tag): v0.1

Normal pages: {Exceptions.PageCountError}

Group members:
Daniels Kanepe
Tamás Tóth
Tomáš Sabol
Attila Bakó

14.12.2021

Supervisors: Mogens Holm Iversen, Lars Landberg Toftegaard, Henrik Kristian Ulrik Øllgaard

Abstract/Resume

Our goal was to make a new system to replace the current outdated one for the company Vestbjerg Byggecenter. Before we got into coding the system, we had to analyze the current situation, how the company operates at the moment and how their competitors are doing. Our system will help them in managing the Products, help them with the selling process. We believe that our system will make all the processes faster and more reliable. In our report, you can find out how we designed and implemented the new system.

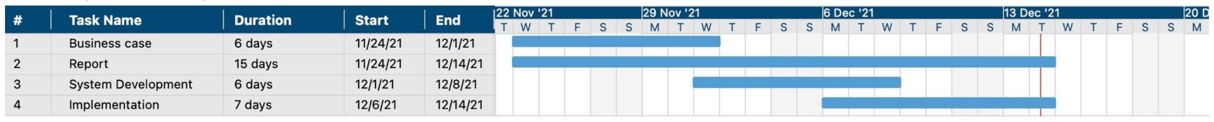
TABLE OF CONTENTS

Introduction

Before starting the project, we needed some structure. A way to prioritize and know what we should be working on, as time was of essence and the deadline is short.

Schedule

And so, as a start, we made a schedule.



#	Task Name	Duration	Start	End
1	Business case	6 days	11/24/21	12/1/21
2	Report	15 days	11/24/21	12/14/21
3	System Development	6 days	12/1/21	12/8/21
4	Implementation	7 days	12/6/21	12/14/21

Group contract

Then, to make sure that each group member lives up to the standards of the rest of the group members, we made a contract that all of us verbally agreed to.

Commitments:
Participate fully (in spirit and actuality). Participate professionally (i.e., civil discourse; abiding by the rules of academic honesty). Meet responsibilities (i.e., completing assigned tasks on time and to the best of your ability). Show up to physical and online group meetings on time. Keep the other team members informed at all times (about progress, attendance etc.) Focus on what is best for the project as a whole See the project through completion.

Meetings:
We will: Be honest and open. Encourage a diversity of opinions on all topics. Give everyone the opportunity for equal participation. Be open to new approaches and listen to new ideas. Avoid placing blame when things go wrong. Instead, we will discuss the process and explore how it can be improved.

UP phase plan

Milestone	Inception	Elaboration	Construction
Goal	Vison Scope Business Modelling	Architecture locked Architectural critical Functionality designed, Implemented and tested	Construct the program (documented) & write unit tests
Number of Iterations	1	2	5
Date	25.11.2021 – 29.11.2021	29.11.2021 - 05.12.2021	05.12.2021 - 13.12.2021
Artifacts	• Brief use cases • Fully dressed use cases • Actor-goal table • System vision	• Domain Model • Class Diagram • Workflow Diagram • System Sequence Diagrams	• Revisions of Use Cases • Implementation • Testing

		• Sequence Diagram • Design Class Diagram	

Inception:

Artifact	Comment
Vision	The general idea for the system and how it will work (e.g., purpose, scope, future ideas...).
Use-Case	A diagram that represents the actions between the actors and the system.
Domain model	A visualisation of the real-world conceptual classes that model the domain.
Mock-up	A prototype of the system.
Workflow diagram	Diagrams that show the step-by-step, linear representation of the main business processes.

Elaboration:

Artifact	Comment
Use cases	Creating the descriptions of the use cases (brief and fully dressed)
SSD	From the fully dressed diagram create the System Sequence Diagrams.
Design Class Diagram	The Diagram that represents the classes used in the system.

Construction:

Artifact	Comment
Revisions of Use Cases	Check over the use cases (Do we need to add some? Remove some?)

Project Case

Introduction

Vestbjerg Byggecenter was founded in 1989 by Anders Olesen and has been in business since then, for over 30 years. In 2007, sons of CEO Anders Olesen - Thomas and Casper, joined the management.

At that time, the company was converted into a limited liability company and changed name to Vestbjerg Byggecenter A/S. The company ís well known in Northern Denmark and is a member of the XL-byg chain, but its system has not been updated for a long time which resulted in an old, insufficient and not user-friendly Windows Vista system they have been using until now. All staff members, including managers, are tired of the current system, therefore we expect them to welcome our new system. We were asked to improve the company's business part as well, therefore we are providing business analysis for the company too.

Problem statement

Currently the company is using an old, outdated, non-user-friendly Windows VIsta system. This is a legacy system, and the support for Windows Vista ended on 11th of April, 2017. This means there are no more bug fixes or security patches, which is a huge issue, as this can leave the company vulnerable to intruders, especially black-hat hackers looking to exploit legacy systems.
Currently, the company does not know where the product is located. It could be in none, one, or even two departments. The company does not have any knowledge on this as the system does not support it.
In an ideal world, the company would also be able to lease out their tools and recall them (which currently cannot be done),, as well as make statistics over the customers, products, contractors, and each co-worker.
And so, this we are looking to address by implementing a new, improved system built with current, up-to-date technology which supports the functionality of generating statistics, tracking products, and leasing items.

Stakeholders

First, we analysed the stakeholders in relation to their influence and interest in our project in order to identify project opponents/supporters, and establish an appropriate level of communication.

Stakeholder	**Contribution**	**Interest**	**Influence / Attitude**	**Importance**
Anders Olsen	Experience, Knowledge, Decision making, Finances, Support	New, more reliable and more user-friendly IT System	High/Positive	High
Thomas Olsen	Knowledge, Decision making, Finances, Feedback		High/Positive	High
Casper Olsen			High/Positive	High
Other Managers	Service, Knowledge, Decision making, Feedback		Medium/Positive	Medium
Administrative Employees	Service, Knowledge, Feedback		Low/Positive	Medium

Warehouse Employees	Service, Knowledge, Feedback		Low/Positive	Medium
XL-byg	Advertisement	More money	Medium/Neutral	Medium
Customers	Money	Product / machine	Low/Neutral	Low

To us, it seems very likely that every stakeholder in the company will have a positive attitude towards the new system; however, some employees may be resistant to change and fear job automation. But, we believe that they have low influence in the company and therefore it will not be an issue.

Risk Analysis

Risk	**Probability**	**Impact**	**Priority**	**Solution**
Data transfer (from old system)	3	2	6	Create system backup. Data transfer will be performed during closing hours
Usability	2	3	6	Employee training
System Downtime	3	1	3	Switch of the systems will be performed during closing hours
System bugs/issues	2	3	6	Employees will have to report every bugs and issues they find and we will fix them in shortest time possible

We figured out that with implementation of the new system, there may arise risks such as transferring the data from the old system to the new one, employees not knowing how to use the new system, system downtime during the transition (from old system to new) and system bugs/reliability issues due to it not having been tested for a prolonged period of time.
To avoid these risks, we have identified ways to respond in order to mitigate and even eliminate some of the risks. On top of that, we will create a backup of the current system for restoration purposes in case something goes wrong.

Preliminary study

And so, as a starting point, we first need to determine if the current system really is deficient / a problem. And so, in this section, we will examine the feasibility aspects of the project such as by checking over the project case, the company's financials and organizational structure, as well as performing some analysis, all of which will enable us to create a business case and make a "go" or "no-go" decision on whether the project is worth pursuing or not.

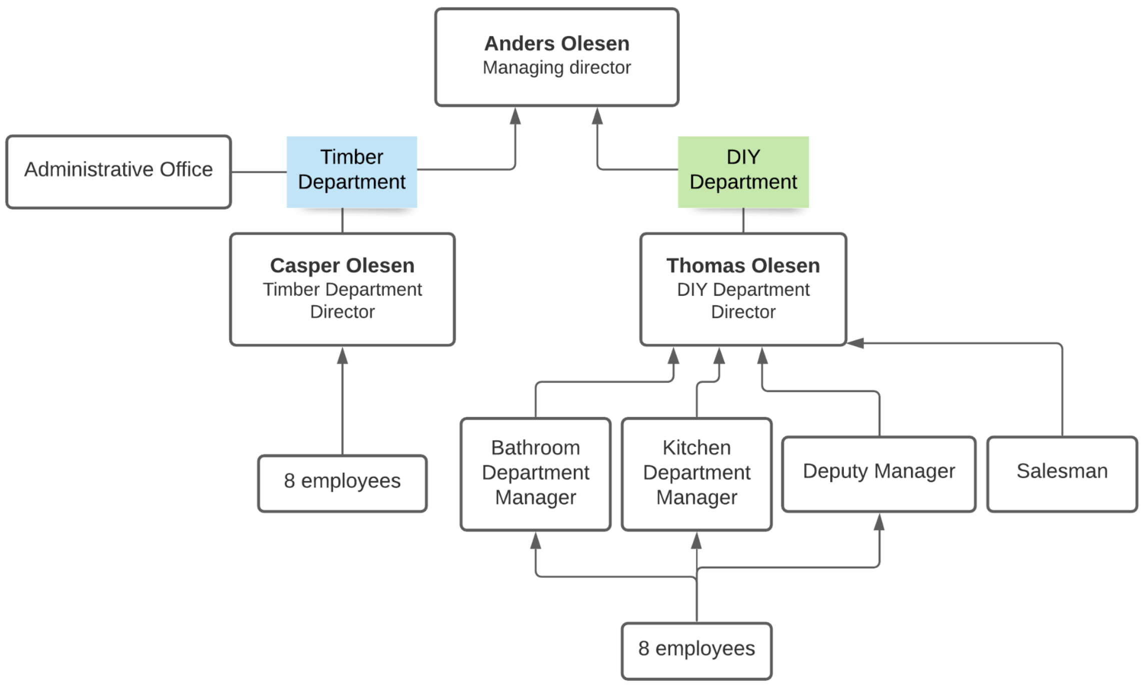
Business analysis

In order to get the necessary information to create the business case, first we need to do a business analysis. First we analysed the company's finance situation and statistics from the last 3 years. Then we focused on organisational structure and management structure, to understand employee relation and scale of the company, find possible flaws and come up with solutions. Next is probably the most important part of our - SWOT analysis and finding strategic objectives. Here we looked at strengths, weaknesses, opportunities and threats of the company related to competition and project planning.

The company

By analysing the company's turnover, we can see that the company grew at a healthy rate of 9.7% from 2018 to 2019 (62.5 mil DKK to 68.6mil DKK); however, the growth rate seems to have slowed down the next year as the turnover growth rate was only 0.53 % (68.6 mil DKK to 69 mil DKK). We fear that in the following years the growth rate might not only decline, but take us into negatives. And thus, with the new system, we would like to bring in not only more revenue, but also prevent the company from falling over from the declining growth rate and threats facing it, as well as try to make use of the opportunities that the company can exploit.

Organisational structure



Visualisation of organisational structure

By looking at the case material, we have identified that the company's structure is multi-divisional. However, we have also identified that each department has its own management style. Namely, the DIY department is divisional, and the timberland department has quite a flat hierarchy.

The hierarchy seems to have been created by trial-and-error, as well as the different managerial styles that the directors prefer. We believe it is a good idea, especially for small companies; however, once the company starts to grow, especially the timberland department, the flat structure might not be the best fit and they might want to follow the DIY department's hierarchical structure.

Management style

Organizational culture here reminds of Power culture, as it consists of informal communication and trust, yet managers still hold the power. Since the 3 of the main managers have family ties, this might cause problems when solving disputes and finding the right direction for the company as it might be hard to outvote them (3 votes).
Anders Olesen:
- o Motivating employees
- o Distributes responsibilities

Thomas Olesen:
- o Cares about employees (yearly interviews)
- o Team Building organiser

Casper Olesen:
- o Detailed and controlling but helpful

SWOT Analysis

Here we have analysed the company's strengths, weaknesses, opportunities and threats related to business competition and the project's planning.

Strengths	**Weaknesses**
● Managers care about employees. ● The managers are very experienced in the industry ● They have a wide variety and lots of products. ● They have discounts and special agreements when placing large orders. ● They are a member of the XL-byg chain.	● Can't generate statistics over the business ● Using an old, unreliable legacy system. ● Stock location is unknown ● They depend on the contractors (item condition, shipping etc.) ● Item leasing is managed manually.

Opportunities:	**Threats**
● Shipping may be faster as stock location is known. ● As they are part of the XL -byg chain, they can integrate the new system with their website and offer products from another established site, thus reaching a larger market share. ● Using big data analysis to adapt to the customers' needs. ● Advertise on social medias(facebook,tiktok,Instagram) ● Have more products	● Competitors may gain an advantage in shipping times as they are tracking the stock location ● Competitor companies that can generate statistics may respond better to trends and such by analysing the data. ● Similar companies, especially larger ones can provide better offers as they can support themselves on lower profit margins. ● The system could break down, and it might be difficult to fix it due to it being based on old technology.

SWOT - Strengths, Weaknesses, Opportunities, Threats

Strategic Objectives

Based on the SWOT Analysis (Strength, Weakness, Opportunities, Threats) we decided to improve on the company's weaknesses and work on eliminating some of the biggest threats facing the company which mainly arise from the current, outdated system.
For example, locating items won't be a problem anymore. The leasing process will be much easier, quicker and automated, thus not dependent on staff member intervention in every step. The new system will be able to generate statistics over the co-workers, customers. With that the company will be able to identify what products are trending at the moment, how frequently the customers come to the shops etc. The managers will also be able to identify how well an employee is performing. On top of all of that, the new system can run on a newer operating system, on up to date technology, so the security, and reliability will be improved.

System vision

The system will be reliable, easy to use, will be stable and secure. The system is able to register, read information, update and delete customers as well as products, and print information about contractors. (Only) Managers of the company will be able to set the price and discount of each product. One feature of the system will produce statistics over the customers, products, contractors and also over each co-worker. The system will be able to make registration from offer until the item is invoiced. There is also an option for making offers, place orders, confirm orders, dispatch notes and invoices. Every new item will come with a barcode to inventory control. The system will check the minimum and maximum stock several times a day. In case that the stock reaches its pre-set minimum, the system will automatically reorder the given amount. Plus it will check if it will not exceed the maximum amount after auto-ordering.
There is also a possibility to integrate the system into the XL-byg webshop, as most shopping happens online nowadays. The management of the company is planning an expansion of products next year,

which will be a much easier task to do with this new system. From this expansion we can also expect more customers and more revenue.

Business case - Vestbjerg Byggecenter

Executive Summary

We came up with a solution for current problems. The best option is to develop a new system for the company – achieving benefits of a reliable system which makes the workflow faster..

Reasons

The company faces several problems which are:

A. Using an old unreliable system.
B. There is no location management for the items.

Business Options

1. Do nothing. With this option still used the old system which can be more unreliable. However, this option saves the expenses of creating the new system.
2. Create a new system with the asked requirements and implement it. For the employees, give training about the new system, so they will be able to use it efficiently.

Expected benefits

- Faster shipping due to item location in the system.
- New system is more user friendly and faster which makes the work flow better.
- Further integration into XL-byg chain

Expected dis-benefits

- It may take more time for the employees to get used to the new system
- Employees may be a bit sceptical about the new statistic function
- The automation of some tasks may cause concerns of downsizing among the employees

Timescale

- Project time: 3 week
- The benefits are to be measured annually (for three years), starting one year after the solution is delivered.

Costs

- Software cost: 540 000 DKK[1]
- Hardware: 150 000 DKK[2]
- Operational costs (support, per year): 20 000 DKK
- Employee training cost: 300 DKK/h/person = ~25000 DKK

[1] 30 days * 6 hours * 750 DKK/hour (salary) * 4 group members
[2] Including tablets, PCs, servers, printers, etc.

Major risks

Employees may feel like their job can be automated and they can lose their job in the future.

Investment appraisal

	Year 2021	Year 2022	Year 2023
Project costs	715 000 DKK	0	0
Operational costs	20 000 DKK	20 000 DKK	20 000 DKK
Benefits	2 170 000 DKK	6 000 000 DKK[3]	9 000 000 DKK
Net benefits	1 435 000 DKK	7 980 000 DKK	8 980 000 DKK

Use Cases & Domain Model

- CRUD Product
- CRUD Product Group
- CRUD Customer
- CRUD Customer Group
- CRUD Employee
- CRUD Contractors
- CRUD Price and Discount
- Generate Statistics
- Sell Product
- Lease Products
- Return Lease

Actor-goal table

In this table, we analyzed the actors - company managers and employees and their goals - tasks.

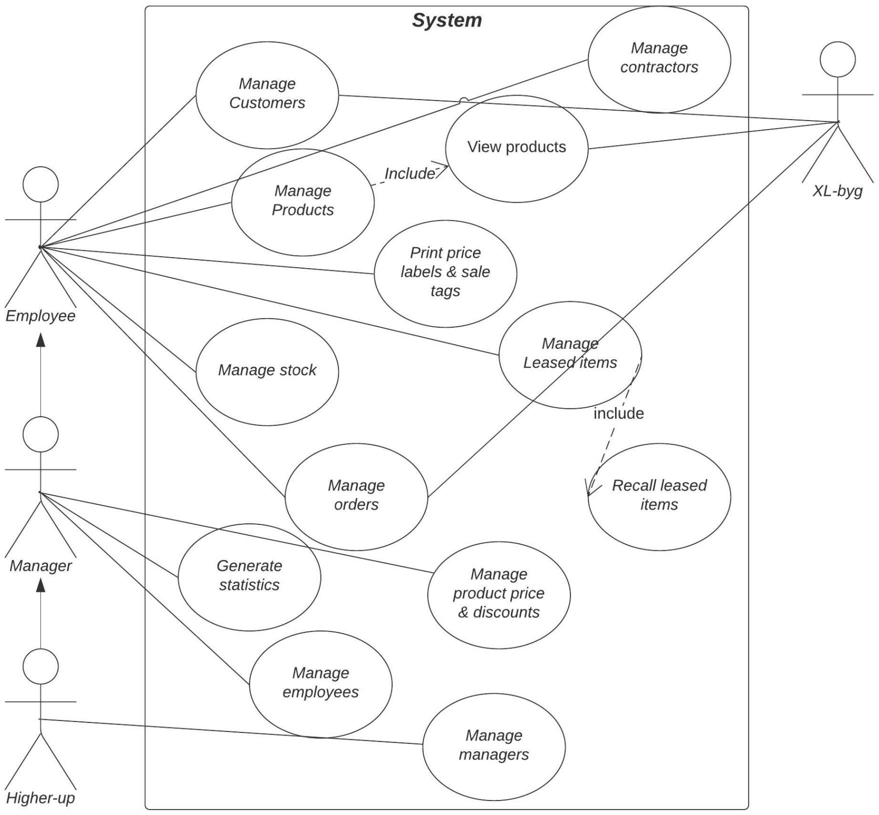
Actor	Goal
Anders Olsen	Manage company
Thomas Olsen	Manage DIY Department
Casper Olsen	Manage Timber Department
Employee	Manage Customers

[3] We expect the company to double, or even triple its profit compared to this year, as management is planning product and therefore customer expansion

	Register new Products Register information about contractors Lease machines Print sale tags Print price labels Create order Inventory control Make statistics
Manager	Everything employee does Manage Employees Manage Products Manage Price Manage Discounts

Use Case Diagram

The system boundary is set to stock/item management, person/company registration and report generation.



Use Case Brief description

Use case: Sell product(s)
The employee uses the system to create a new order. First, the employee adds the items to the order. Then, the employee identifies the customer and saves the order. A receipt is generated.

Use case: Manage order(s)
-A staff member uses the system to identify an order. The employee views the details about the order, and updates its details or deletes it from the system.
-A staff member uses the system to create new orders.

Use case: Invoice
During the sell product(s) use case, once the employee has finished adding items to the order, the employee can alternatively choose to invoice a customer to fix the price and allow the customer to pay later

Use case: Manage product(s)
-A staff member uses the system to identify a product. The employee views the details about the product, and updates its details or deletes it from the system.
-A staff member uses the system to create new products.

Use case: Manage price & discount(s)
The manager uses the system to identify an item. The manager updates a product's price or sets/removes a discount associated with a product. The discount is then checked during the sell product(s) use case.

Use case: Manage stock
-The employee uses the system to find an item and set the minimum and maximum limit.
-If a product's stock reaches the minimum it gives a notification to the Employees.
-Restock manually

Use case:Manage manager(s)
The higher-ups use the system to add a new manager. First, the higher-ups add the manager information. Identifies the manager and adds the system.

Use case:Manage employee(s)
The manager uses the system to add a new employee, get employee information, update employee information, and delete the employee from the system.

Use case:Manage customer(s)
The employee uses the system to add a new customer, get customer information, update customer information, and delete the customer from the system.

Use case:Generate statistic(s)
-The manager uses the system to generate the turn over for an employee, find the employee and choose the time period which he wishes to see.
-The manager is also able to create statistics about customers (frequency of visits, what they usually buy).
-The manager is also able to generate statistics about contractors (most delivered products, items, ?frequency?, how much we pay them)

Use case:Manage contractor(s)
The employees can use the system to update a contractor in the system, update the information and submit it.

Use case:Manage leased item(s)
The employe use

Use case:Recall leased item(s)
The employe use

Use case:Manage Customer group(s)
The employe use
Can also add discount percentages to customer groups.

Use Case Priorities

Here we have prioritized the use cases to determine which use cases are most important, thus, which we shall be working on in each iteration, and to determine which fully dressed use case descriptions we shall write about / focus on.

Use case	**Priority**
Sell Product(s)	1
Manage Stock	2
Lease Products	3
Return Lease	3

Use Case fully dressed

After we agreed on the use case priorities, we moved on and decided to make fully dressed use cases for the most important ones - Sell Product, Manage Stock - set min & max, Manage Stock - restock.

Use case name	Sell Product	
Actors	Employee	
Pre-conditions		
Post-conditions	**An order is created and associated with Person and item(s)**	
Main success scenario	**Actor (Action)**	System (Response)
	1. A customer wants to buy a product(s)	
	2. The employee chooses to create an order.	3. The system asks to identify the items.
	4. The employee identifies the item(s)	5. The item(s) are added to the order
	Step 3-5 repeat until all items are added to order.	
	6. Employee indicates that the order is finished.	7. The system asks to identify the customer.
	8. Employee identifies the customer	9. System finds the customer. System asks for confirmation
	10. Employee confirms that the entered data is	11. System updates the price based on the

	correct	customer category, prints the invoice and asks if the invoice has been paid.
	12. Employee confirms that the invoice has been paid.	13. System finishes the order and associates it with the customer and item(s).
Alternative flows	4. An item was not found in the system. a. System repeats from step 3	
	7.. Customer not in the system a. Register the customer	
	9 Customer was not identified a. System repeats from step 7	

Manage Stock

Use case name	Manage Stock - Set min & max	
Actors	Employee	
Pre-conditions		
Post-conditions	The minimum and maximum stock is set for a Product	
Main success scenario	**Actor (Action)**	**System (Response)**
	1. An employee wants to set the minimum and maximum stock of a product	2. The system asks for the product
	3. The employee identifies the product	4. The system asks for the minimum amount
	5. The employee inserts the amount	6. The system asks for the maximum amount
	7. The employee insert the amount	8. The system asks for a confirmation
	9. The employee confirms the confirmation	
Alternative flows	3.1. The product was not identified a. System repeats step 2 3.2. The product already had a minimum and maximum set a. System ask if the employee wants to update i. Employee confirms 1. System continues from step 4 ii. Employee declines 1. System repeats step 2	
	5. The inserted amount was of wrong attribute a. System repeats from step 4	
	7. The inserted amount was of wrong attribute b. System repeat from step 6	
	9. The employee does not confirm the confirmation	

	a. System repeats from step 1

This (below) is theoretically a use case that can later be programmed to execute automatically when stock level is low.

Use case name	Manage Stock - Restock	
Actors	Employee	
Pre-conditions	Minimum and Maximum already set for a product, Product exists, Contractor exists	
Post-conditions	The product is restocked to the maximum amount	
Main success scenario	**Actor (Action)**	**System (Response)**
	1. The amount of a product reaches the minimum	2. System sends notification
	3. An employee selects restock option in the menu	4. System asks to identify the product
	5. An employee identifies the product that needs to be restocked	6. System finds the product and displays product information and asks which contractor to order from
	7. Employee inputs contractor name	8. System finds contractor and displays information and asks for confirmation
	9. Employee confirms the conformation	
	*ordered items are delivered to the warehouse	
	10. Employee identifies new items	11. Items are added to the system
Alternative flows	1. The amount does not reach minimum, but an employee wants to restock a product anyway a. System continues from step 3	
	6. Product not found in the system a. System repeats from step 4	
	7. Contractor not found in system a. System repeats from step 6	

Identifying classes

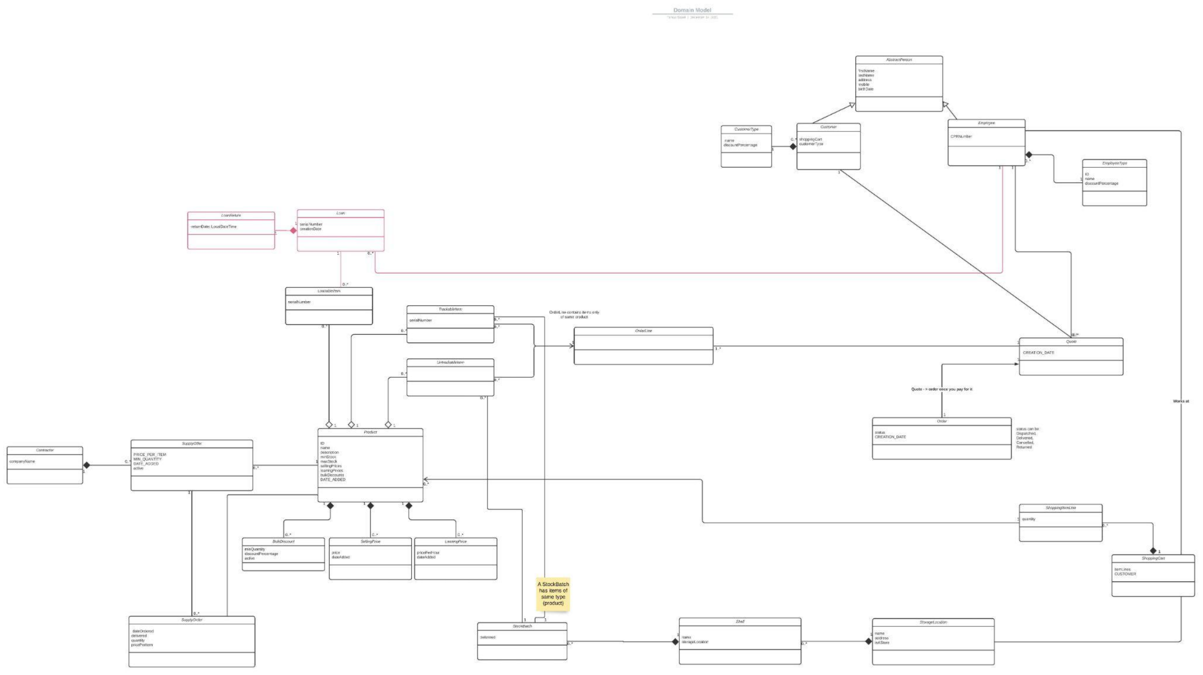
We then looked over the case and identified nouns that could represent the conceptual classes in this project.

Noun/Candidates	**In/Out**	**Evaluation**
Employee	In	Employees are the main users

		of the system
Customer	In	Customers are an essential part of the system
Product	In	Products are needed as the whole system revolves around them
Order	In	Orders are needed to fulfil the main use case - the purchase of products
Offers (Quotes)	In	An order can be a quote until it's paid.
Item	In	An Item is of a specific product
Stock	In	We need to know which items are in stock.
Contractor	In	Contractors are essential for company to restock products
Manager	Out	Manager is a type of Employee
Employee type	In	There could be multiple types of employees, including managers
Discount	In	It is important to keep track of the discounts for the statistics, so it's a class
Department (product group e.g. Kitchen)	In	There is an option to buy a group of products
Loan	In	We need to keep track of loans, and all of them might have their own attributes e.g. loan time, so it's a class
Price - for loans	In	It is important to keep track of the prices for the statistics
Price - for purchases	In	It is important to keep track of the prices for the statistics
Department (DIY/Timber)	In	There are multiple departments

Domain model

To create the domain model, we used the nouns table as inspiration. When creating it, we identified a lot more classes to make it all work e.g. orderline, trackable/untrackable items (e.g. fridges with serial numbers, and pencils) etc.
The scope for the domain model is all of the use cases we had identified.



4 5

Use Case Analysis

System Sequence Diagram

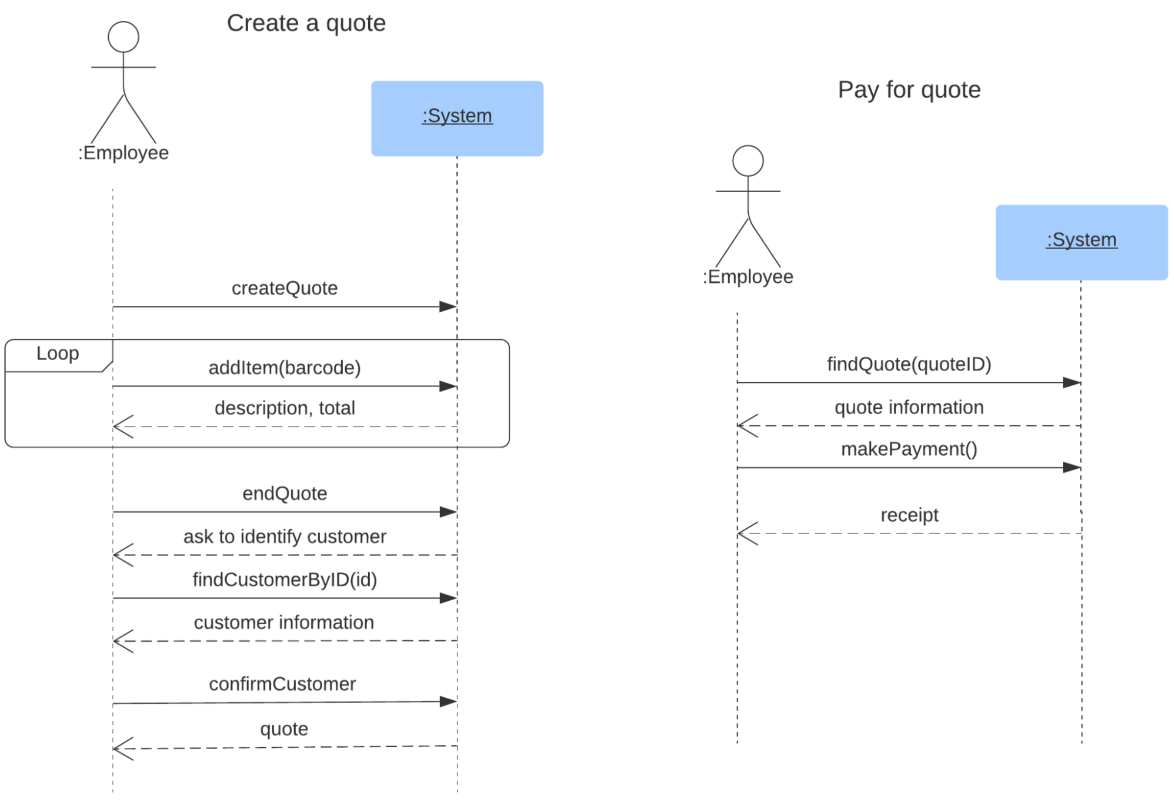
Then, after creating the fully dressed descriptions, we decided to create system sequence diagrams for each, where a system is treated as black box and the emphasis is put on the operations that happen between actors and the system. In these diagrams we are showing only the main success scenario.

Operation Contract

We believe that some of the operations in the system sequence diagram might be too abstract and difficult to understand, and so, we have chosen to describe some of the operation contracts and show the state before and changes after their execution.

[4] Classes and connections in red are not implemented yet, but will be implemented in the future
[5] Full size image of the domain model can be found at the end of the report or within the zip file in directory "report"

SSD: Sell Product(s)



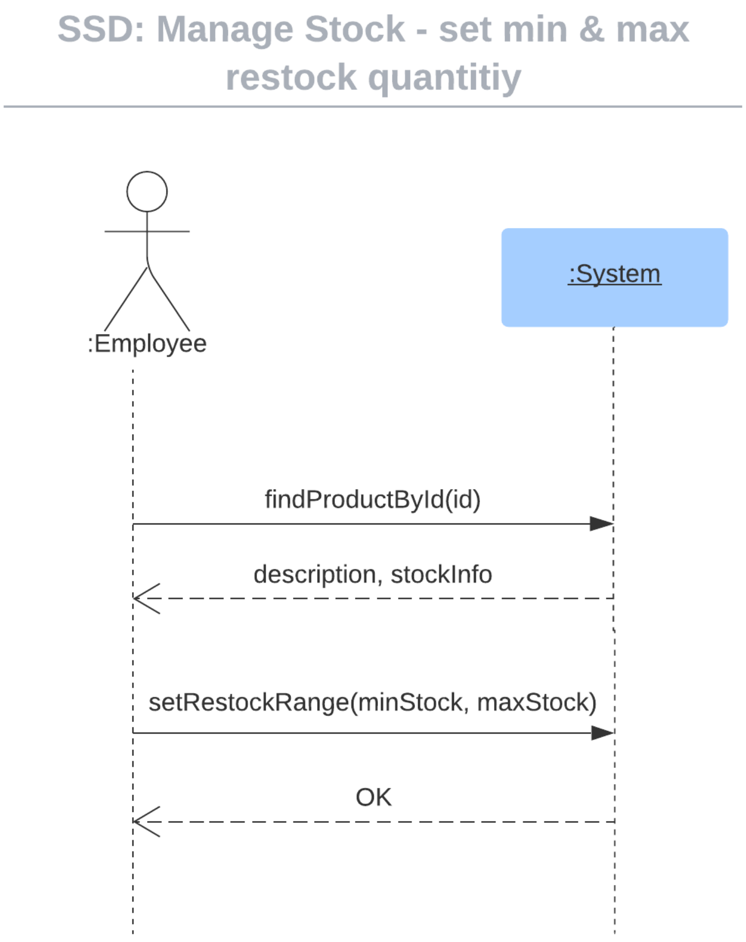
Operation: createQuote
Use Case: Sell Product(s)
Pre-Condition:
Post-Condition: - order instance o was created - o.status is set to offered

Operation: addItem(barcode)
Use Case: Sell Product(s)
Pre-Condition: - Item instance *i* already exists - Order instance *o* has been created
Post-Condition: -Item instance *i* was associated with Order *o*

Operation: findCustomerbyID(id)

Use Case: Sell Product(s)
Pre-Condition: - Order o exists with associated items. - Customer *c* exists in system - Order instance *o* was created
Post-Condition: - Customer c was associated with Order *o*

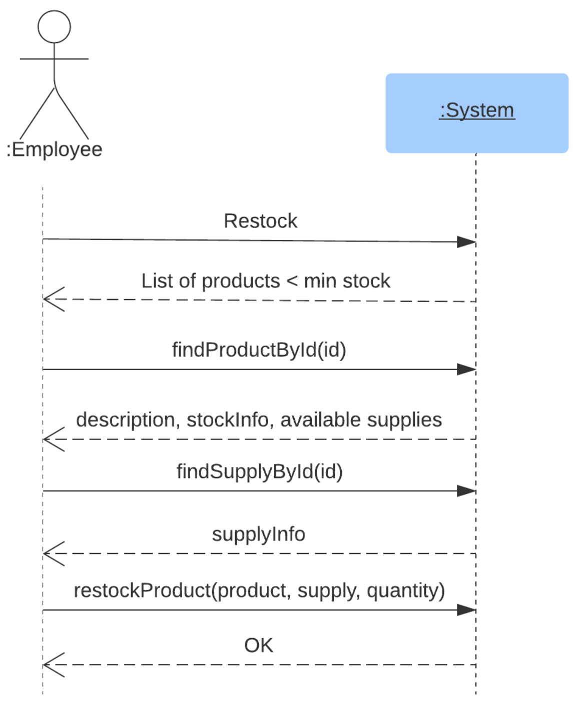
Operation: makePayment()
Use Case: Sell Product(s)
Pre-Condition: - Order *o* exists - Customer *c* was associated with Order *o* - Order *o* contains items.
Post-Condition: o.status is set to Paid


SSD: Manage Stock - set min & max restock quantitiy
:Employee
:System
findProductById(id)
description, stockInfo
setRestockRange(minStock, maxStock)
OK

Operation: setRestockRange

Use Case: Manage Stock: set min & max restock quantity
Pre-Condition: - Product *p* exists
Post-Condition: -p.minStock became minStock -p.maxStock became maxStock

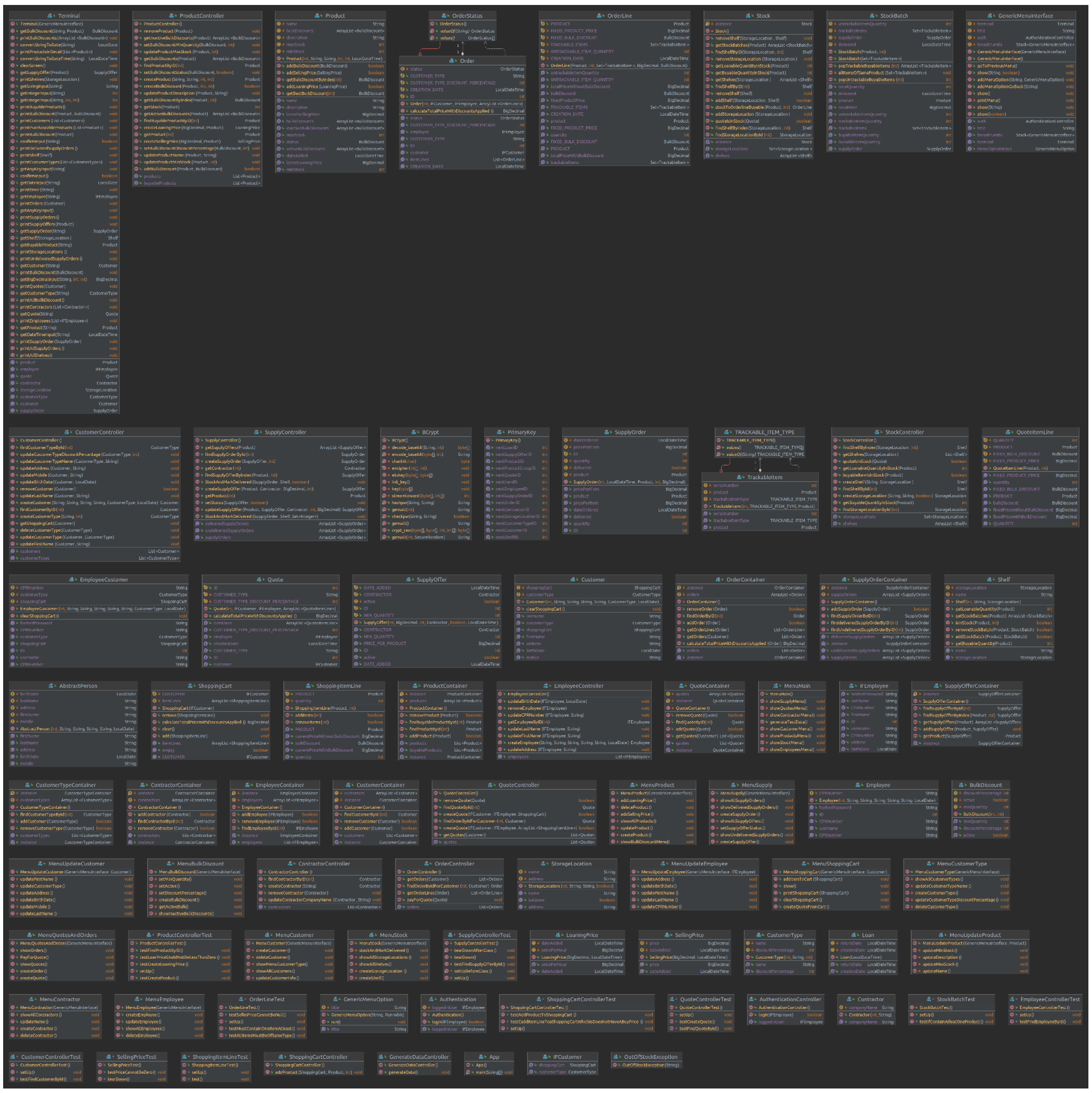
SSD: Manage Stock - restock


:Employee
:System
Restock
List of products < min stock
findProductById(id)
description, stockInfo, available supplies
findSupplyById(id)
supplyInfo
restockProduct(product, supply, quantity)
OK

Operation: restockProduct
Use Case: Manage Stock - restock
Pre-Condition: - Product *p* exists - Supply *s* exists
Post-Condition: - *Items items* (*i1, i2, i3...i*quantity*) are generated with a unique barcode - *items.costPrice* is set to *s.cost* - *Items are* added to p.items

Design

Design Class Diagram

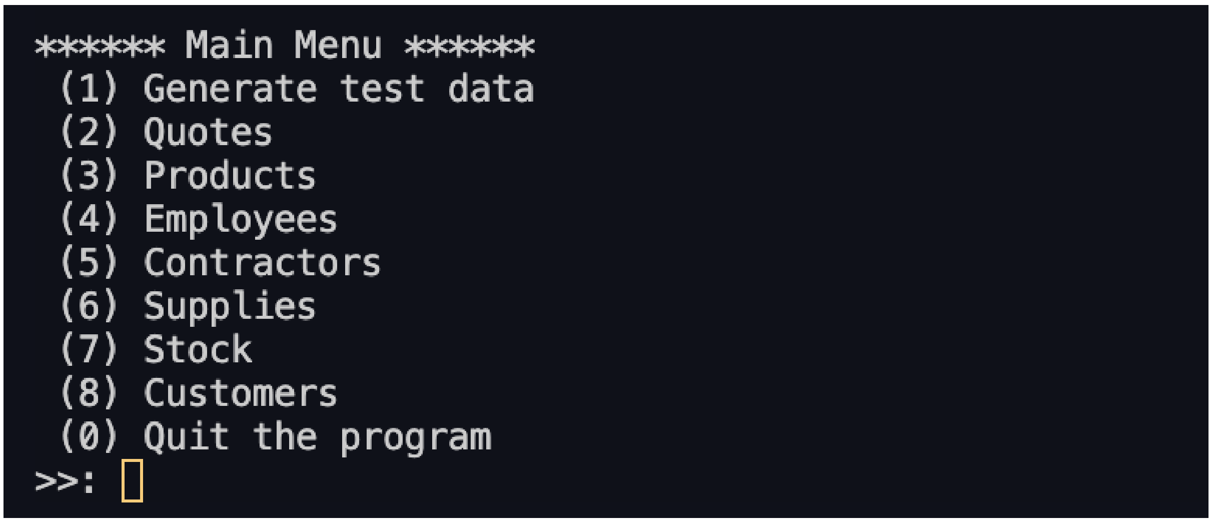


6 7

Mockups

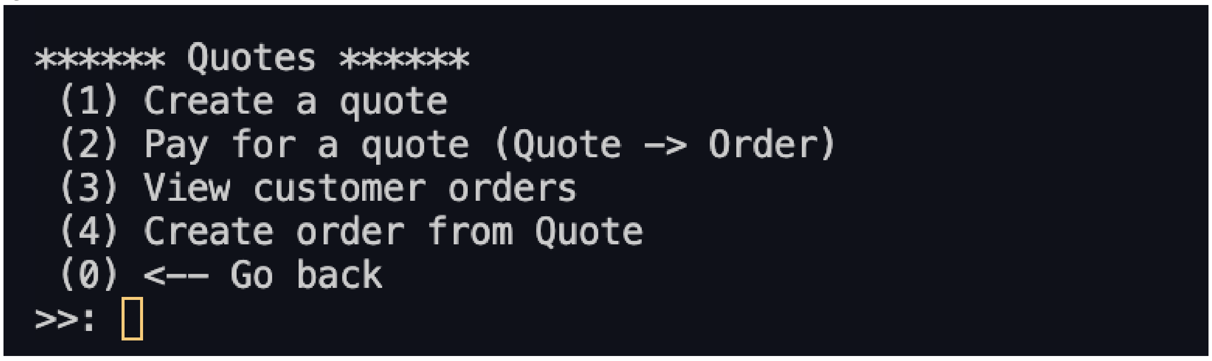
To help us visualise how the TUI for this project will look like, we decided to mockup the interface with the use of Java's print statements.
Main Menu:

6 Full size image of the domain model can be found at the end of the report or within the zip file in directory "report"
7 Class diagram does not include set nor get methods. They are only showed in view layer



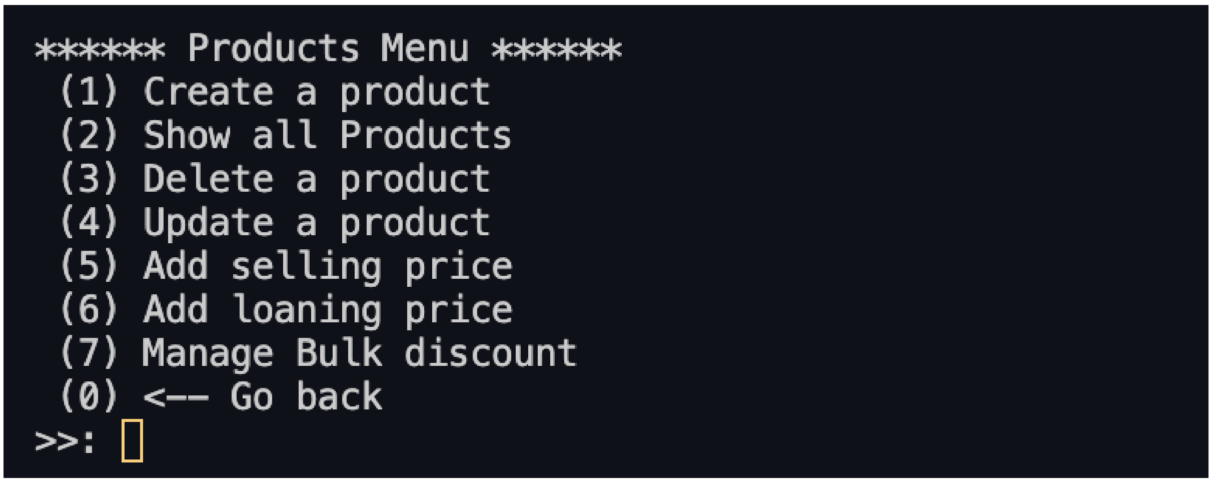
```
****** Main Menu ******
 (1) Generate test data
 (2) Quotes
 (3) Products
 (4) Employees
 (5) Contractors
 (6) Supplies
 (7) Stock
 (8) Customers
 (0) Quit the program
>>:
```

Quotes Menu



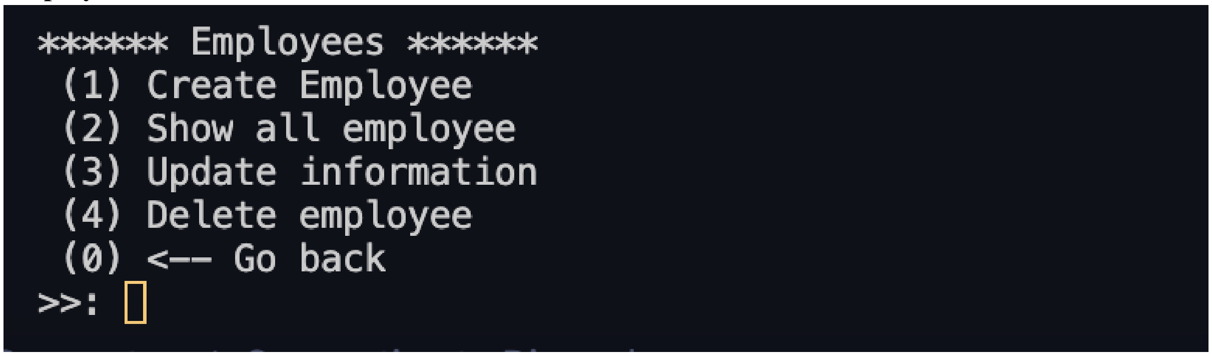
```
****** Quotes ******
 (1) Create a quote
 (2) Pay for a quote (Quote -> Order)
 (3) View customer orders
 (4) Create order from Quote
 (0) <-- Go back
>>:
```

Products Menu



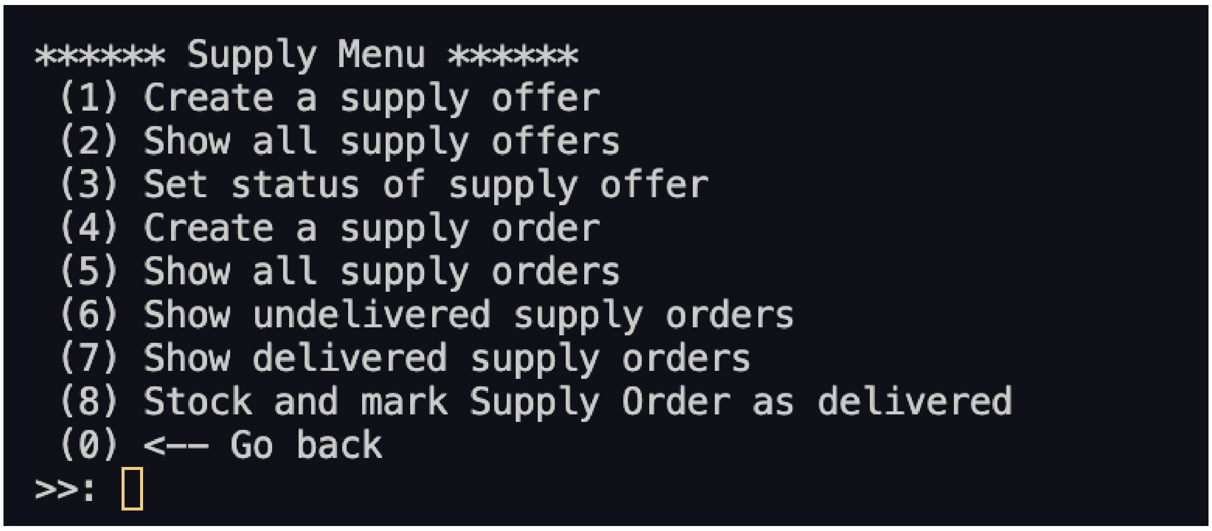
```
****** Products Menu ******
 (1) Create a product
 (2) Show all Products
 (3) Delete a product
 (4) Update a product
 (5) Add selling price
 (6) Add loaning price
 (7) Manage Bulk discount
 (0) <-- Go back
>>:
```

Employees Menu



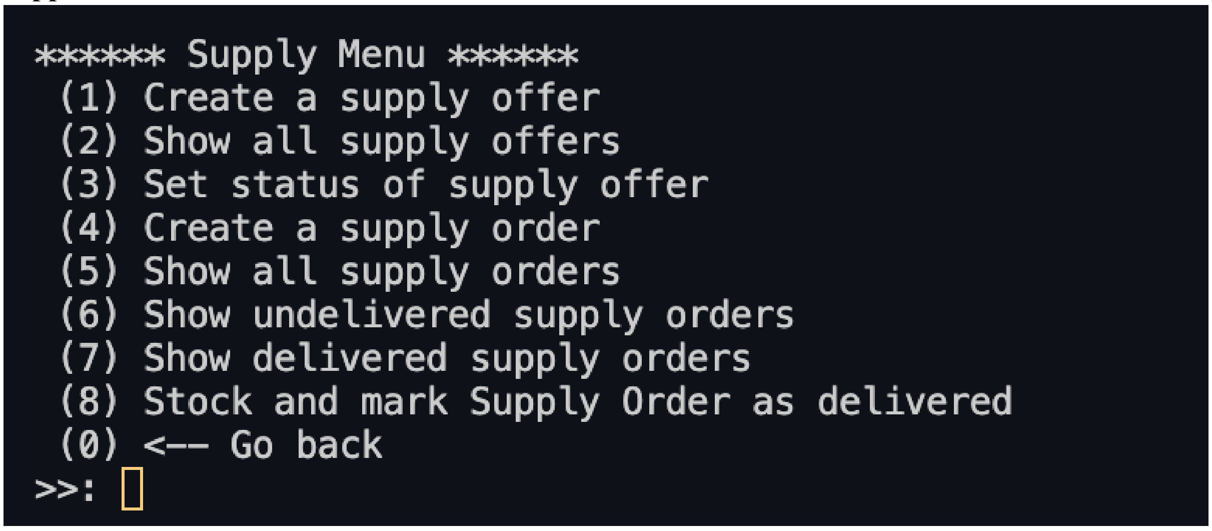
```
****** Employees ******
 (1) Create Employee
 (2) Show all employee
 (3) Update information
 (4) Delete employee
 (0) <-- Go back
>>: 
```

Contractors Menu



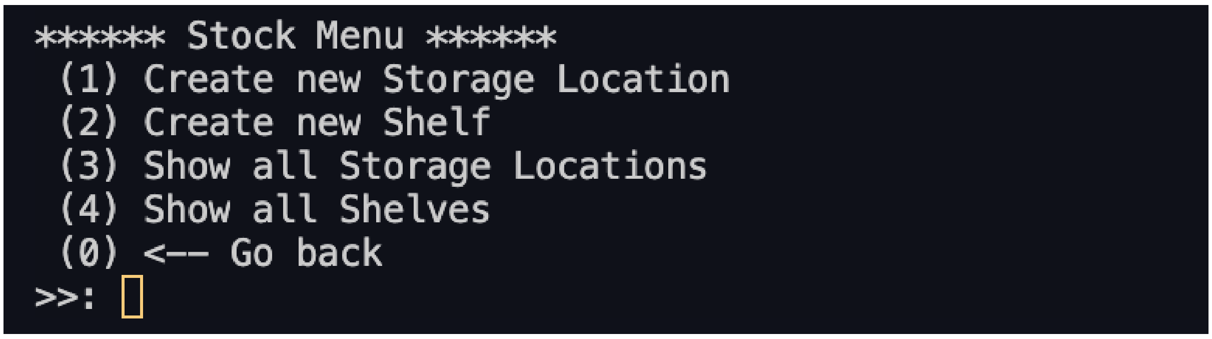
```
****** Supply Menu ******
 (1) Create a supply offer
 (2) Show all supply offers
 (3) Set status of supply offer
 (4) Create a supply order
 (5) Show all supply orders
 (6) Show undelivered supply orders
 (7) Show delivered supply orders
 (8) Stock and mark Supply Order as delivered
 (0) <-- Go back
>>: 
```

Supplies Menu



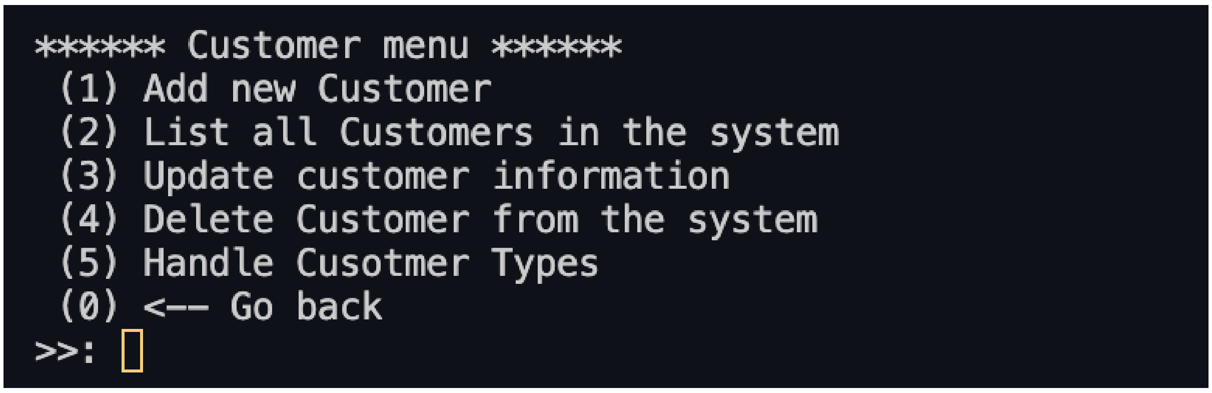
```
****** Supply Menu ******
 (1) Create a supply offer
 (2) Show all supply offers
 (3) Set status of supply offer
 (4) Create a supply order
 (5) Show all supply orders
 (6) Show undelivered supply orders
 (7) Show delivered supply orders
 (8) Stock and mark Supply Order as delivered
 (0) <-- Go back
>>: 
```

Stock Menu



```
****** Stock Menu ******
 (1) Create new Storage Location
 (2) Create new Shelf
 (3) Show all Storage Locations
 (4) Show all Shelves
 (0) <-- Go back
>>: 
```

Customer Menu

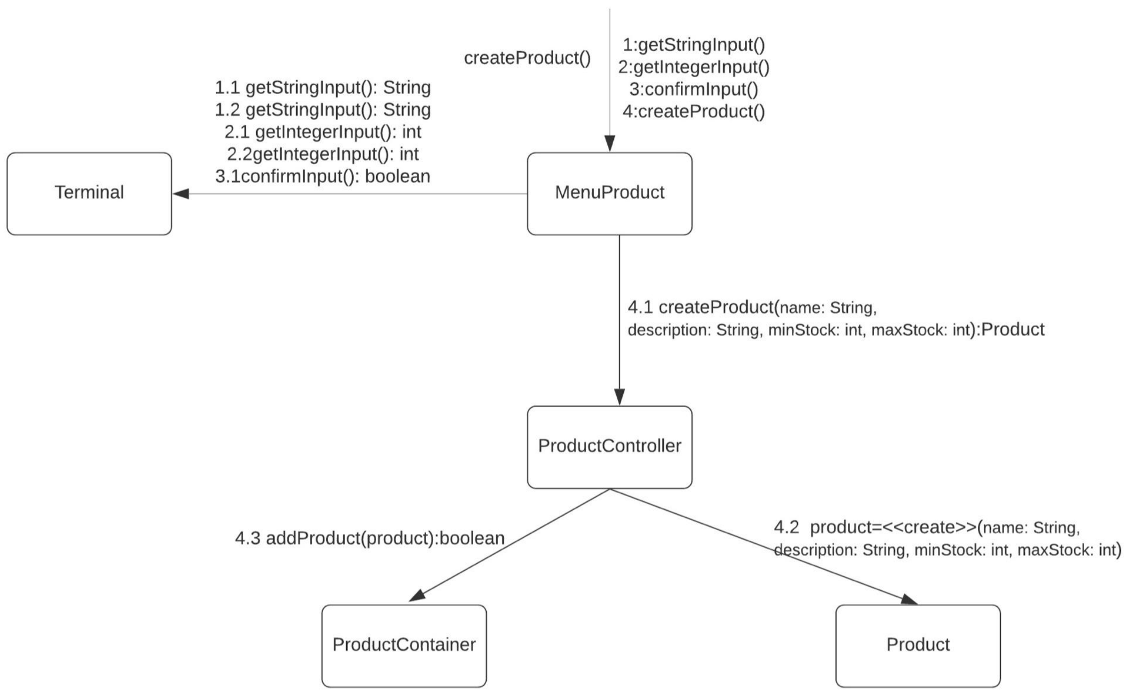


```
****** Customer menu ******
 (1) Add new Customer
 (2) List all Customers in the system
 (3) Update customer information
 (4) Delete Customer from the system
 (5) Handle Cusotmer Types
 (0) <-- Go back
>>: 
```

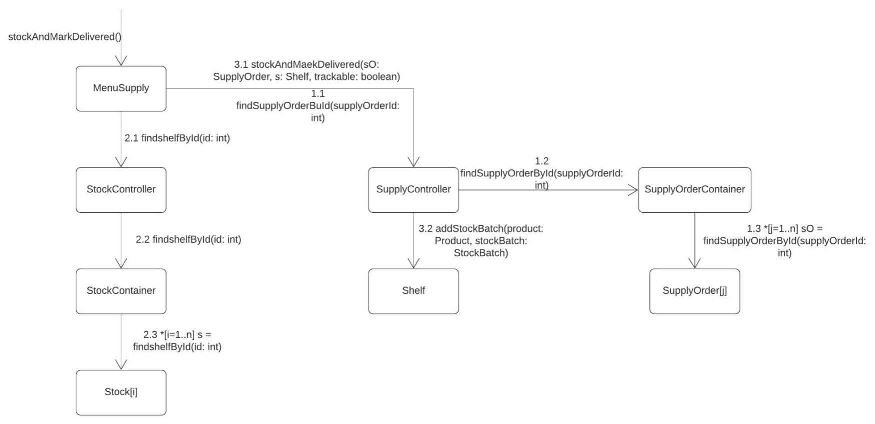
Communication Diagram

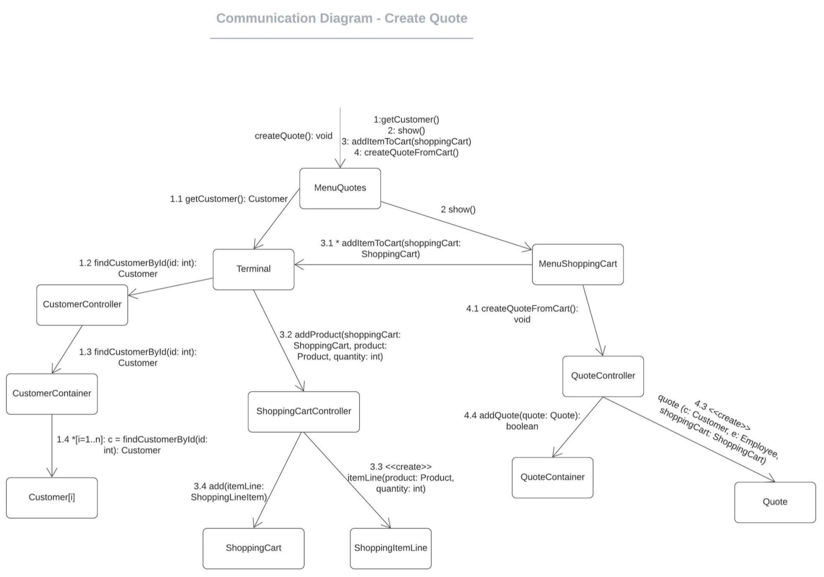
In the pictures below you can have a better understanding of how classes communicate and interact with each other.

Communication Diagram - Create Product


createProduct()
1:getStringInput()
2:getIntegerInput()
3:confirmInput()
4:createProduct()
1.1 getStringInput(): String
1.2 getStringInput(): String
2.1 getIntegerInput(): int
2.2getIntegerInput(): int
3.1confirmInput(): boolean
Terminal
MenuProduct
4.1 createProduct(name: String, description: String, minStock: int, maxStock: int):Product
ProductController
4.3 addProduct(product):boolean
4.2 product=<<create>>(name: String, description: String, minStock: int, maxStock: int)
ProductContainer
Product

Communication Diagram - Restock


stockAndMarkDelivered()
MenuSupply
3.1 stockAndMaekDelivered(sO: SupplyOrder, s: Shelf, trackable: boolean)
1.1 findSupplyOrderBuId(supplyOrderId: int)
2.1 findshelfById(id: int)
StockController
SupplyController
1.2 findSupplyOrderById(supplyOrderId: int)
SupplyOrderContainer
2.2 findshelfById(id: int)
3.2 addStockBatch(product: Product, stockBatch: StockBatch)
1.3 *[j=1..n] sO = findSupplyOrderById(supplyOrderId: int)
StockContainer
Shelf
SupplyOrder[j]
2.3 *[i=1..n] s = findshelfById(id: int)
Stock[i]


Communication Diagram - Create Quote
createQuote(): void
1:getCustomer()
2: show()
3: addItemToCart(shoppingCart)
4: createQuoteFromCart()
MenuQuotes
1.1 getCustomer(): Customer
2 show()
3.1 * addItemToCart(shoppingCart: ShoppingCart)
Terminal
MenuShoppingCart
1.2 findCustomerById(id: int): Customer
CustomerController
1.3 findCustomerById(id: int): Customer
CustomerContainer
1.4 *[i=1..n]: c = findCustomerById(id: int): Customer
Customer[i]
3.2 addProduct(shoppingCart: ShoppingCart, product: Product, quantity: int)
ShoppingCartController
3.4 add(itemLine: ShoppingLineItem)
3.3 <<create>> itemLine(product: Product, quantity: int)
ShoppingCart
ShoppingItemLine
4.1 createQuoteFromCart(): void
QuoteController
4.4 addQuote(quote: Quote): boolean
4.3 <<create>> quote (c: Customer, e: Employee, shoppingCart: ShoppingCart)
QuoteContainer
Quote

Implementation

In this phase we started designing and implementing our use cases. Unfortunately we did not have enough time to finish all of them.

These are the implementations:

Code Structure

We structured our code in four packages: model, view, controller and tests. When designing interaction between these packages, we tried to follow coding standards: low coupling and high cohesion. We also used some more advanced concepts like inheritance, to ensure that we have low code duplication.

User Interface Guide

Our system is based on TUI, therefore you can use the application only by text input. Our menus navigation is based on numbers. Pressing "0" and Enter key will get you back to the previous menu, or will end the application in the main menu. You can enter the command "-back" and press enter to go back, if you get stuck on the prompt.

Model layer

Model package contains data-type files and container files following our design class diagram.
Most of the container files communicate with a specific controller file in the controller layer, however some controller files communicate with multiple containers (for example

CustomerController communicates with CustomerContainer and CustomerTypeContainer). All of the containers are singletons.

Controller layer

Controller package contains 11 controller files. Each of those communicates with corresponding files in the model layer.
We decided to merge some of them, for example CustomerController and CustomerTypeController, because CustomerType is related to Customer and also in order to not have way too many files.
Controller GenerateDataController is the file we use to generate data for our application. It could be placed in the Model or the Viewlayer as well, but we decided to place it into a controller layer.

View layer

View package contains the main file: App.java, which contains the main method and is therefore an entry file to our application. Next we created GenericMenuInterface and GenericMenuOptions. Each menu in our application extends/inherits the GenericMenu and uses its methods and functionality.
Class Terminal is the largest class of our application. It contains methods to get the user input (for example getIntegerInput(), getStringInput()), methods that return a specific object from the specific container, based on for example ID from user input. It also includes print methods, that print out the object information about all objects of that class (e.g.: printCustomers() prints all customers and their information).

Test layer

Because this system is responsible for managing selling, loaning price, discounts and calculating prices for Orders and Quotes, we created multiple tests to secure reliability of our system. This layer contains 10 JUnit 5 test files that test some controller files as well as files from the model layer. Unfortunately we were unable to make more tests due to the lack of time and we rather prioritized finishing the implementation of the most important use cases.

Could be implemented in the future

Unfortunately we couldn’t implement everything we wanted, due to the short time frame we had to create our project. Because of that we know that our code is not perfect, and it could be improved. For example we currently can’t create statistics about any employee, and we can’t differentiate between a normal employee and a manager. These were low priority use-cases, so we wanted to leave them for last, but we did not have time to finish them. One more main function we could not implement is the loaning. We already created the basic need for that, in the form of loanable products, but we did not have time to make an interface for that.

(Possibly the create employee won't be implemented, max discount is a bit more than 20% (We have them separately that the bulk and customer discount cant be more than 20%), loaning is not implemented, but you can create products that can be loaned. Order is kinda fucked up so it could be improved.)

Conclusion

We have created quite a complex TUI system capable of managing Customers, Employees, employee and customer types, products, Selling/loaning prices, discounts, supplies, contractors, and storage locations. We will continue working on the system to implement the Loan use cases and we will remake the UI into the GUI. We believe that given a few more days or even a week, we could have the final system (with TUI) fully working with all of the use cases implemented and tested.

Even though not all of the use cases are finished, we believe that the solution is more than satisfactory as we have laid the groundwork and made it very extensible in terms of expansion.

Coding and naming conventions

Indentation

Use indentation of tabs (4 spaces wide), to make the workspace cleaner and less messy overall.

Bracket placement

Each curly bracket pair's opening bracket should be placed on the same line as the command that requires the brackets.

General

Like other open source projects, the code base for the Eclipse project should avoid using names that reference other companies or their commercial products.

Java Packages

All packages should be nouns, consisting of one word, using only lowercase ASCII letters. Avoid using underscore (_) or dollar sign ($) characters in the name.

Tests

For tests we will be using Junit 5. All tests shall go in a package called 'tests'. Each layer shall have its own tests subpackage (e.g. model layer shall have the package 'tests.model'). Each tests class shall be named following the standard approach of what's being tested followed by the word 'Test' (e.g. PersonTest) using CamelCase'ing. Each method in these classes shall start with 'test' and end with what's being tested using CamelCase'ing.

Classes and Interfaces

Class names should be nouns. The first letter and all other internal words should start with a capital letter.
Use meaningful, descriptive words, avoid the usage of acronyms and abbreviations for easier understanding. Commonly used abbreviations are allowed if everybody in the team agrees.
Example: Class name: LP
LP is an abbreviation for Long Play, but LP is more commonly used and easier to understand for many.

Interface names should be capitalized like class names.
All interfaces shall be prefixed with the letter “I”.

Methods

All methods should be verbs. Naming should use *camelCase-ing method.
Follow common practice for naming getters, setters and predicates.
Example: getName(), setName(), hasName(), isName()

Variables

All variables should be short and meaningful, making it obvious to the reader what their function is. Naming should use *camelCase-ing method.
Avoid the usage of one letter variable names, except for “throwaway” variables. Common names for temporary variables are:

Integer: i , j , k , m , n	Char: c , d , e	String: s

**camelCase* - *The first letter should be lowercase and all other internal words should start with a capital letter.*

**PascalCase* - *All words start with a capital letter..*

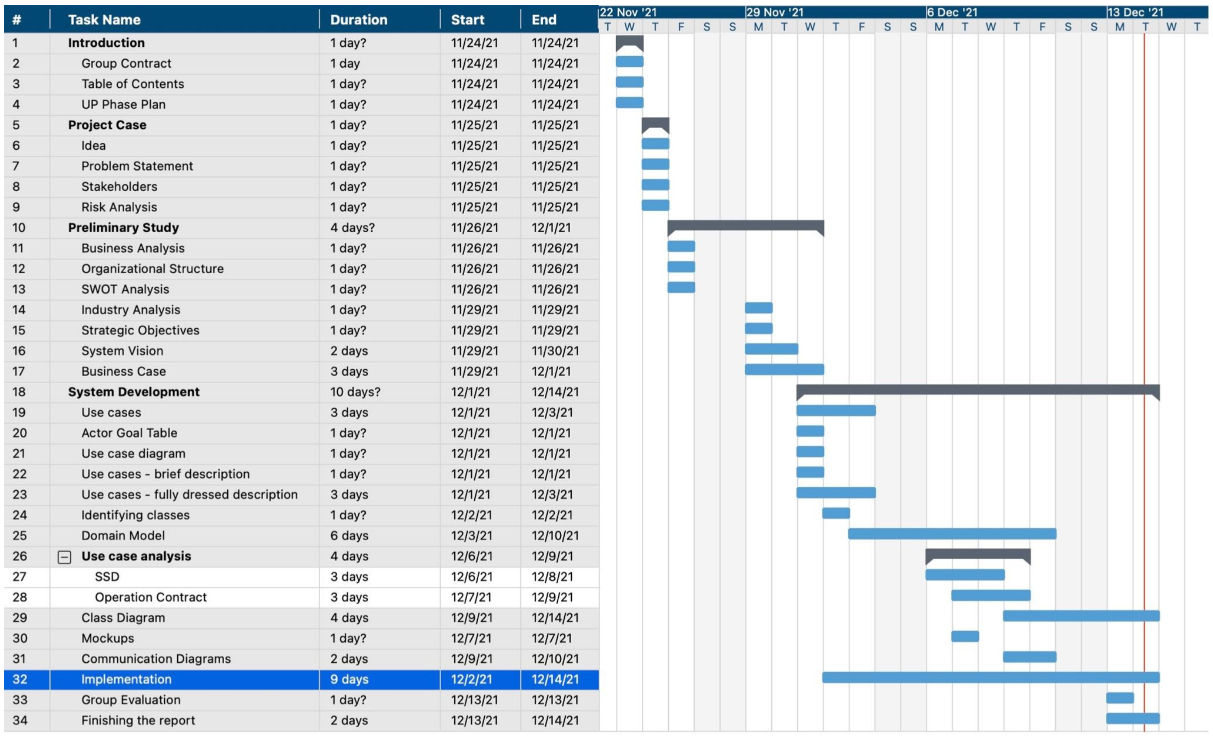
Documentation

Document everything - in report, and in code.

Project reflection

Schedule

Things never go as planned and that was also the case with our project. Some tasks we were able to complete relatively quickly, some took way more time than we expected. Here you can see how much time we spent with each task.



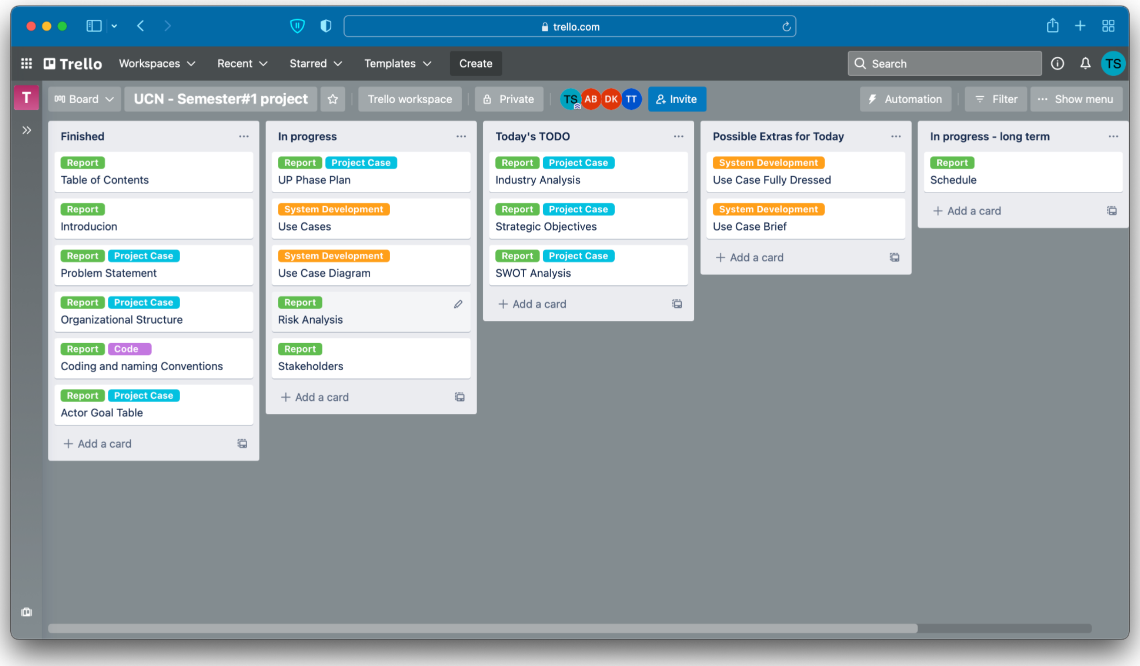
#	Task Name	Duration	Start	End
1	**Introduction**	1 day?	11/24/21	11/24/21
2	Group Contract	1 day	11/24/21	11/24/21
3	Table of Contents	1 day?	11/24/21	11/24/21
4	UP Phase Plan	1 day?	11/24/21	11/24/21
5	**Project Case**	1 day?	11/25/21	11/25/21
6	Idea	1 day?	11/25/21	11/25/21
7	Problem Statement	1 day?	11/25/21	11/25/21
8	Stakeholders	1 day?	11/25/21	11/25/21
9	Risk Analysis	1 day?	11/25/21	11/25/21
10	**Preliminary Study**	4 days?	11/26/21	12/1/21
11	Business Analysis	1 day?	11/26/21	11/26/21
12	Organizational Structure	1 day?	11/26/21	11/26/21
13	SWOT Analysis	1 day?	11/26/21	11/26/21
14	Industry Analysis	1 day?	11/29/21	11/29/21
15	Strategic Objectives	1 day?	11/29/21	11/29/21
16	System Vision	2 days	11/29/21	11/30/21
17	Business Case	3 days	11/29/21	12/1/21
18	**System Development**	10 days?	12/1/21	12/14/21
19	Use cases	3 days	12/1/21	12/3/21
20	Actor Goal Table	1 day?	12/1/21	12/1/21
21	Use case diagram	1 day?	12/1/21	12/1/21
22	Use cases - brief description	1 day?	12/1/21	12/1/21
23	Use cases - fully dressed description	3 days	12/1/21	12/3/21
24	Identifying classes	1 day?	12/2/21	12/2/21
25	Domain Model	6 days	12/3/21	12/10/21
26	**Use case analysis**	4 days	12/6/21	12/9/21
27	SSD	3 days	12/6/21	12/8/21
28	Operation Contract	3 days	12/7/21	12/9/21
29	Class Diagram	4 days	12/9/21	12/14/21
30	Mockups	1 day?	12/7/21	12/7/21
31	Communication Diagrams	2 days	12/9/21	12/10/21
32	Implementation	9 days	12/2/21	12/14/21
33	Group Evaluation	1 day?	12/13/21	12/13/21
34	Finishing the report	2 days	12/13/21	12/14/21

Group Evaluation

To manage our group work we used Trello. We put up daily tasks, and also long term tasks, where everybody could choose what they wanted to do. At the end of each day, we discussed what we have done on that day, what is tomorrow's plan, what tasks we need to do, when and where we will meet next time.

Most of the time we met at the campus Sofiendalsvej. We also held a few online meetings due to the snow storm and sickness, as well as met in person in a place of one of the group members. Once we even met at the campus Hobrovej.
We worked really hard on this project. Usually we met at 9AM and worked until 4PM, even on Saturdays. We believe that we had great teamwork and communication throughout the project, even though there were times when we did not agree with everything, we managed to compromise and create the current system.

All in all, we were satisfied with what we accomplished.



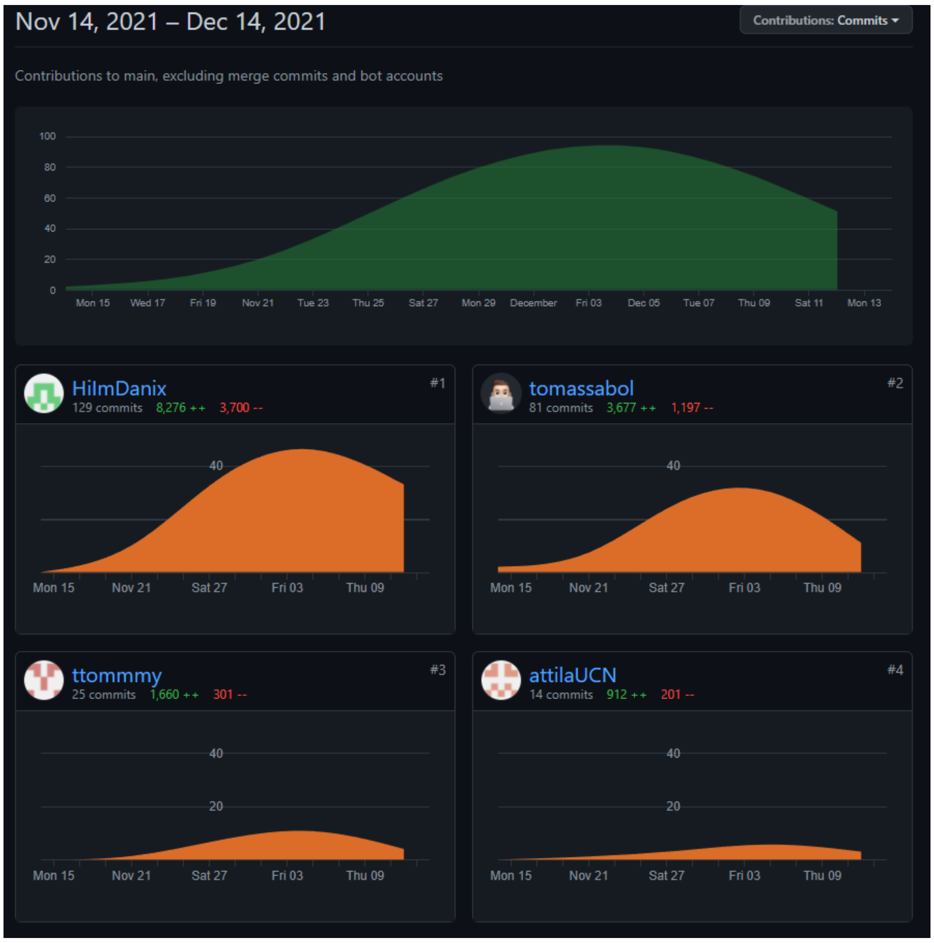
Finished
- Report — Table of Contents
- Report — Introducion
- Report, Project Case — Problem Statement
- Report, Project Case — Organizational Structure
- Report, Code — Coding and naming Conventions
- Report, Project Case — Actor Goal Table
- + Add a card

In progress
- Report, Project Case — UP Phase Plan
- System Development — Use Cases
- System Development — Use Case Diagram
- Report — Risk Analysis
- Report — Stakeholders
- + Add a card

Today's TODO
- Report, Project Case — Industry Analysis
- Report, Project Case — Strategic Objectives
- Report, Project Case — SWOT Analysis
- + Add a card

Possible Extras for Today
- System Development — Use Case Fully Dressed
- System Development — Use Case Brief
- + Add a card

In progress - long term
- Report — Schedule
- + Add a card



Nov 14, 2021 – Dec 14, 2021

Contributions to main, excluding merge commits and bot accounts

HilmDanix #1
129 commits 8,276 ++ 3,700 --

tomassabol #2
81 commits 3,677 ++ 1,197 --

ttommmy #3
25 commits 1,660 ++ 301 --

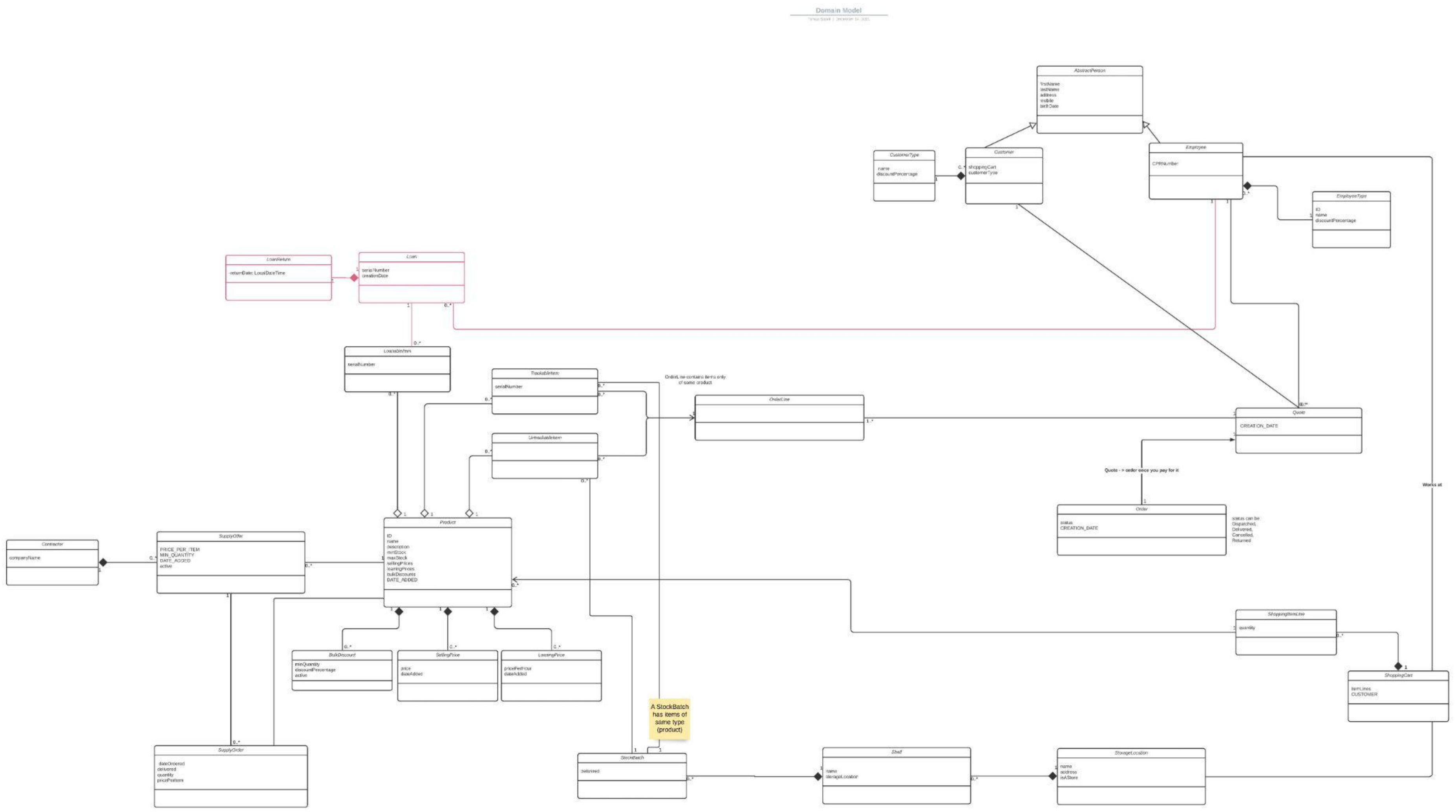
attilaUCN #4
14 commits 912 ++ 201 --

Appendices

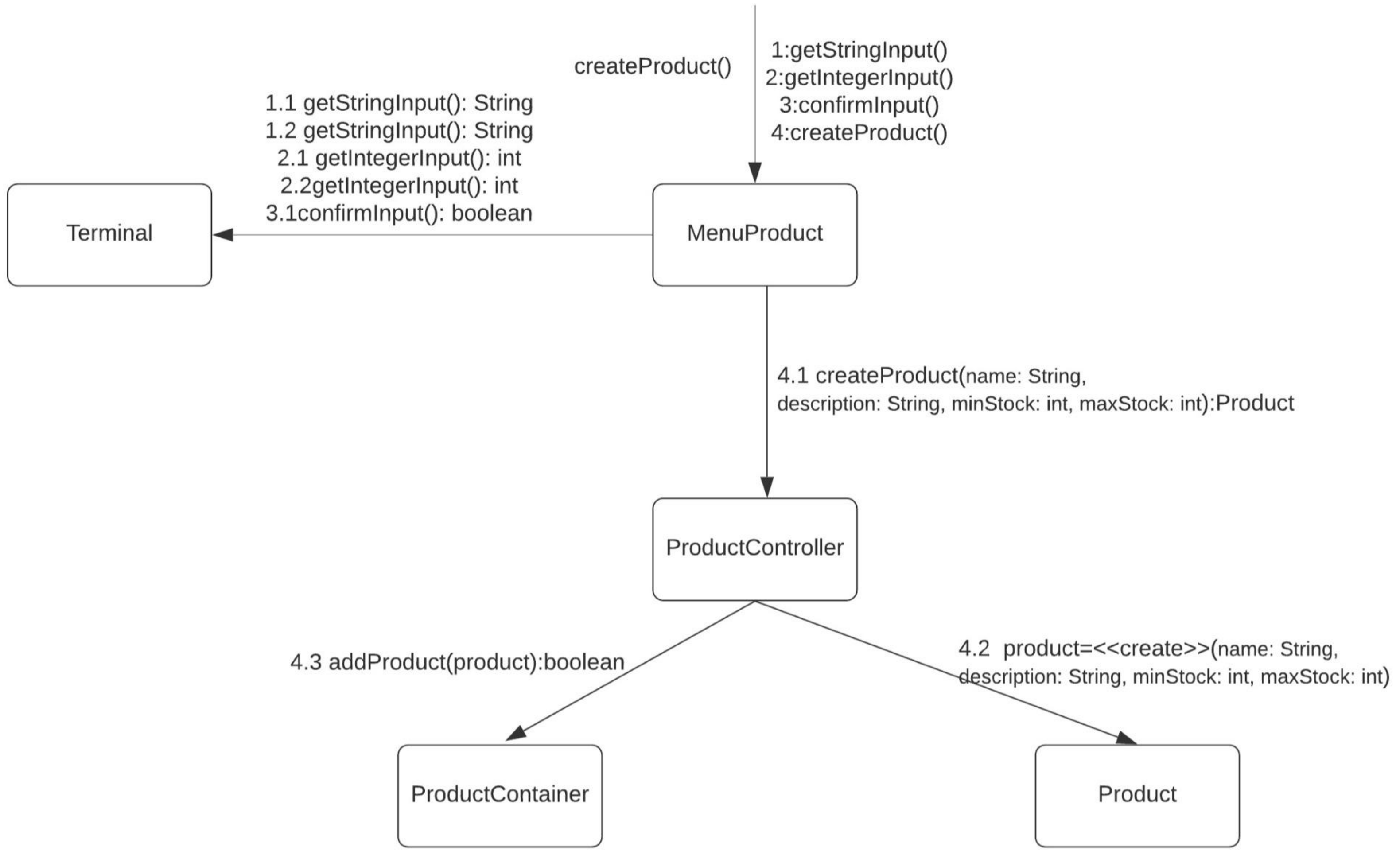
References

[1] LARMAN CRAIG SHAMKANT., *Larman: Applying UML / Elmasri*. S.l.: PEARSON EDUCATION LIMITED, Third edition.

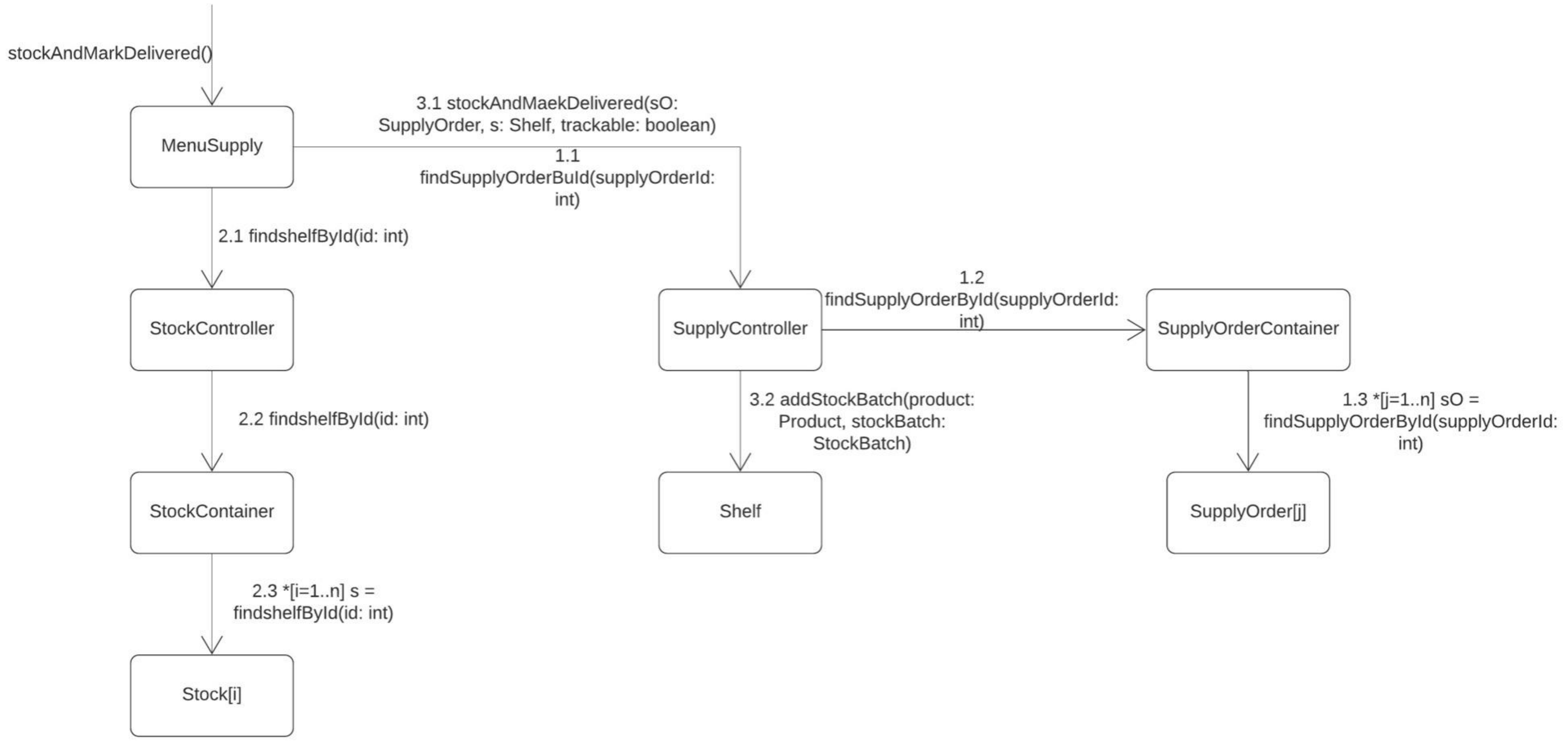
[2] D. J. Barnes and M. Kölling, *Objects first with java: A practical introduction using bluej*. Harlow: Pearson Education, 2017.


Domain Model
AbstractPerson
firstName
lastName
address
mobile
birthDate
CustomerType
name
discountPercentage
Customer
shoppingCart
customerType
Employee
CPRNumber
EmployeeType
ID
name
discountPercentage
LoanReturn
-returnDate: LocalDateTime
Loan
serialNumber
creationDate
LoanableItem
serialNumber
TrackableItem
serialNumber
UntrackableItem
OrderLine contains items only of same product
OrderLine
Quote
CREATION_DATE
Quote - > order once you pay for it
Order
status
CREATION_DATE
status can be Dispatched, Delivered, Cancelled, Returned
Works at
Contractor
companyName
SupplyOffer
PRICE_PER_ITEM
MIN_QUANTITY
DATE_ADDED
active
Product
ID
name
description
minStock
maxStock
sellingPrices
loaningPrices
bulkDiscounts
DATE_ADDED
BulkDiscount
minQuantity
discountPercentage
active
SellingPrice
price
dateAdded
LoaningPrice
pricePerHour
dateAdded
ShoppingItemLine
quantity
ShoppingCart
itemLines
CUSTOMER
SupplyOrder
dateOrdered
delivered
quantity
pricePerItem
A StockBatch has items of same type (product)
StockBatch
delivered
Shelf
name
storageLocation
StorageLocation
name
address
isAStore
1
0..*
1..*

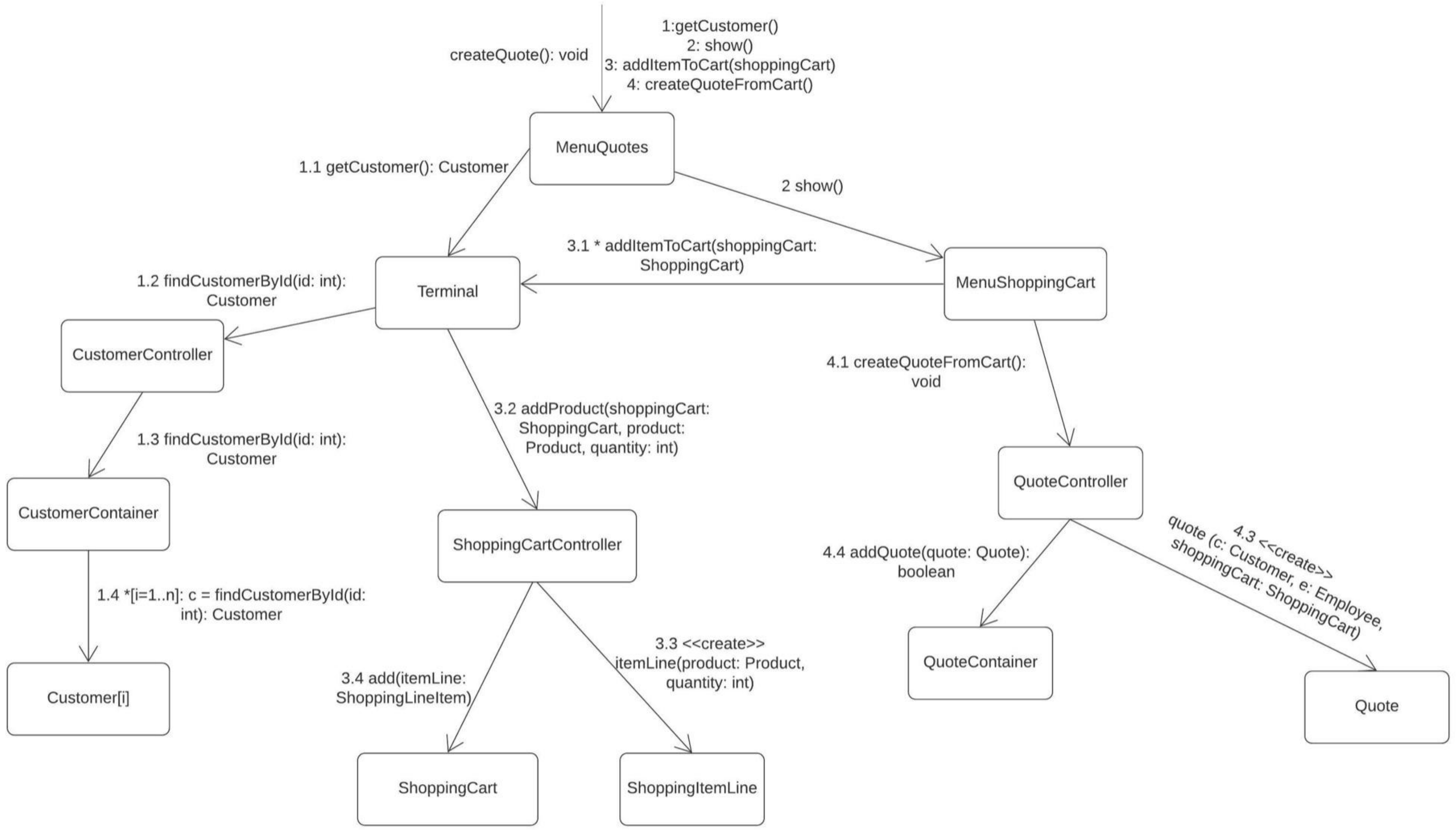
Communication Diagram - Create Product

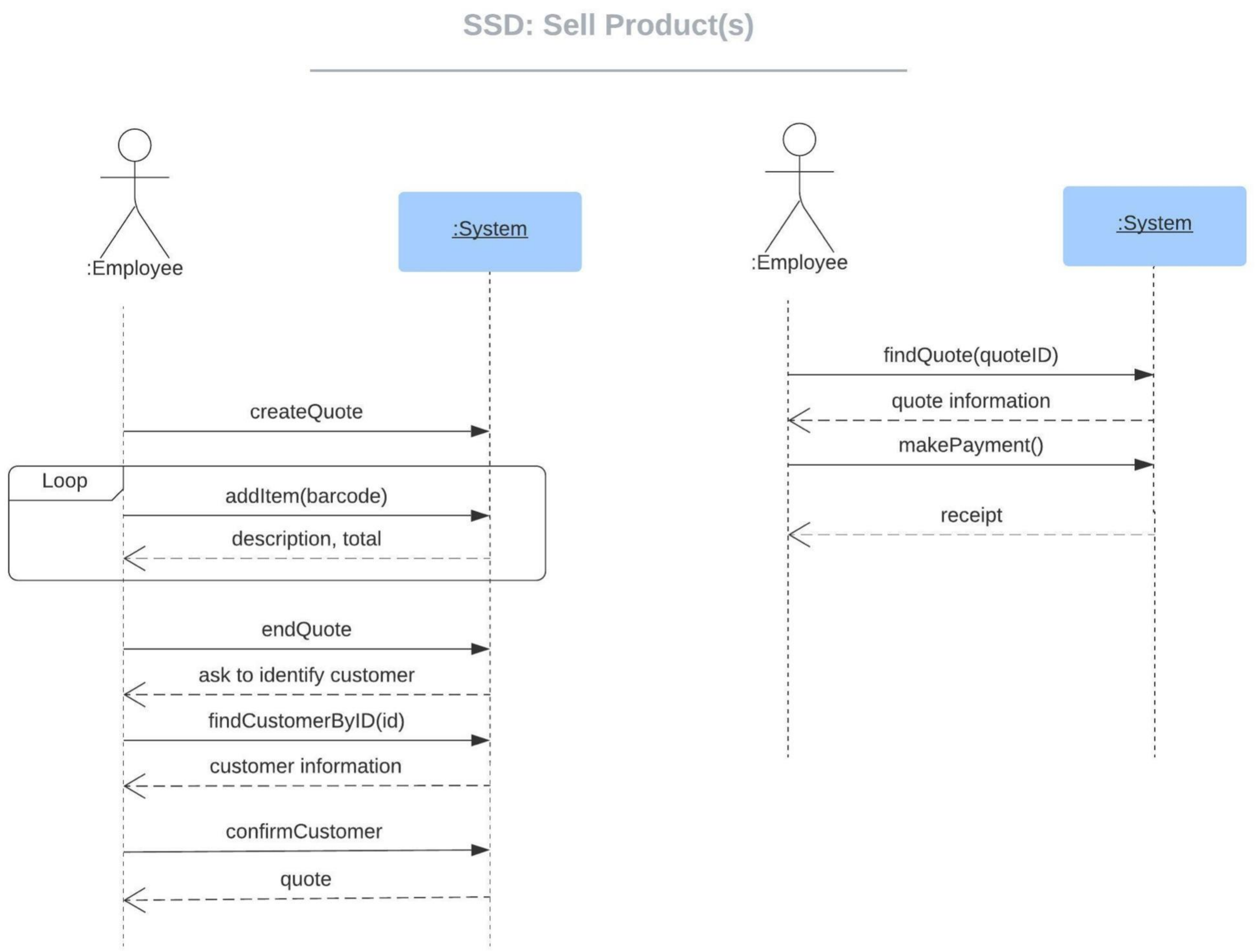

createProduct()
1:getStringInput()
2:getIntegerInput()
3:confirmInput()
4:createProduct()
1.1 getStringInput(): String
1.2 getStringInput(): String
2.1 getIntegerInput(): int
2.2getIntegerInput(): int
3.1confirmInput(): boolean
Terminal
MenuProduct
4.1 createProduct(name: String, description: String, minStock: int, maxStock: int):Product
ProductController
4.3 addProduct(product):boolean
4.2 product=<<create>>(name: String, description: String, minStock: int, maxStock: int)
ProductContainer
Product

Communication Diagram - Restock


stockAndMarkDelivered()
MenuSupply
3.1 stockAndMaekDelivered(sO: SupplyOrder, s: Shelf, trackable: boolean)
1.1 findSupplyOrderBuId(supplyOrderId: int)
2.1 findshelfById(id: int)
StockController
SupplyController
1.2 findSupplyOrderById(supplyOrderId: int)
SupplyOrderContainer
2.2 findshelfById(id: int)
3.2 addStockBatch(product: Product, stockBatch: StockBatch)
1.3 *[j=1..n] sO = findSupplyOrderById(supplyOrderId: int)
StockContainer
Shelf
SupplyOrder[j]
2.3 *[i=1..n] s = findshelfById(id: int)
Stock[i]

Communication Diagram - Create Quote


createQuote(): void
1:getCustomer()
2: show()
3: addItemToCart(shoppingCart)
4: createQuoteFromCart()
MenuQuotes
1.1 getCustomer(): Customer
2 show()
3.1 * addItemToCart(shoppingCart: ShoppingCart)
Terminal
MenuShoppingCart
1.2 findCustomerById(id: int): Customer
CustomerController
4.1 createQuoteFromCart(): void
3.2 addProduct(shoppingCart: ShoppingCart, product: Product, quantity: int)
1.3 findCustomerById(id: int): Customer
QuoteController
CustomerContainer
ShoppingCartController
4.4 addQuote(quote: Quote): boolean
4.3 <<create>> quote (c: Customer, e: Employee, shoppingCart: ShoppingCart)
1.4 *[i=1..n]: c = findCustomerById(id: int): Customer
3.3 <<create>> itemLine(product: Product, quantity: int)
3.4 add(itemLine: ShoppingLineItem)
QuoteContainer
Quote
Customer[i]
ShoppingCart
ShoppingItemLine


SSD: Sell Product(s)
:Employee
:System
createQuote
Loop
addItem(barcode)
description, total
endQuote
ask to identify customer
findCustomerByID(id)
customer information
confirmCustomer
quote
:Employee
:System
findQuote(quoteID)
quote information
makePayment()
receipt