UCN, University College

of Northern Denmark

IT-programme

AP Degree in Computer

Science

Class: dmaj0920

Title: 2nd-semester project

– bio&bio

Participants:  
Antonio Kovacevic  
Lucas Inaschwili-Hongre  
Marko Veljkovic  
Mihail Gerginov  
Teo Stanic

Supervisor: Mogens Holm Iversen

Repository link: <https://github.com/TheRandomTroll/ucn-2nd-semester-project>

Table of Contents

[Introduction 4](#_Toc73064451)

[Section 1: Business 5](#_Toc73064452)

[Gantt Chart 5](#_Toc73064453)

[Company description 5](#_Toc73064454)

[Cost-benefit analysis 7](#_Toc73064455)

[Supply chain management (SCM) 7](#_Toc73064456)

[Value chain analysis 8](#_Toc73064457)

[ERP & CRM 9](#_Toc73064458)

[E-commerce and online marketing 9](#_Toc73064459)

[SWOT analysis & TOWS matrix 10](#_Toc73064460)

[Section 2: System Development 12](#_Toc73064461)

[Unified Process 12](#_Toc73064462)

[Inception 12](#_Toc73064463)

[Elaboration 12](#_Toc73064464)

[Construction 13](#_Toc73064465)

[Transition 13](#_Toc73064466)

[Domain Model 14](#_Toc73064467)

[Use Cases 14](#_Toc73064468)

[Fully Dressed Use Case 14](#_Toc73064469)

[Order Processing – In Store: Fully Dressed Use Case 15](#_Toc73064470)

[Order Processing – Online: Fully Dressed Use Case 16](#_Toc73064471)

[Product CRUD: Fully Dressed Use Case 18](#_Toc73064472)

[Courier CRUD: Fully Dressed Use Case 19](#_Toc73064473)

[Customer CRU: Fully Dressed Use Case 20](#_Toc73064474)

[Use Case Diagram 21](#_Toc73064475)

[System Sequence Diagram 21](#_Toc73064476)

[Order Processing SSD 22](#_Toc73064477)

[Product CRUD 23](#_Toc73064478)

[Courier CRUD SSD 23](#_Toc73064479)

[Customer CRU SSD 24](#_Toc73064480)

[Communication Diagrams 25](#_Toc73064481)

[In Store Order Processing Communication Diagram 25](#_Toc73064482)

[Online Order Processing Communication Diagram 26](#_Toc73064483)

[Product CRUD Communication Diagram 26](#_Toc73064484)

[Courier CRUD Communication Diagram 27](#_Toc73064485)

[Customer CRU Communication Diagram 27](#_Toc73064486)

[Design Class Diagram 28](#_Toc73064487)

[Section 3: Programming 29](#_Toc73064488)

[Mapping the domain model to a relational model 29](#_Toc73064489)

[Writing the SQL scripts 29](#_Toc73064490)

[Database isolation levels 30](#_Toc73064491)

[Creating SQL tables 31](#_Toc73064492)

[SQL Keys 31](#_Toc73064493)

[Programming the solution (Back-end) 31](#_Toc73064494)

[Project structure 31](#_Toc73064495)

[Models 32](#_Toc73064496)

[DBConnection.java 33](#_Toc73064497)

[The singleton pattern 34](#_Toc73064498)

[Implementing DAO classes 34](#_Toc73064499)

[Implementing controllers 37](#_Toc73064500)

[Unit testing the back-end 37](#_Toc73064501)

[Programming the solution (GUI) 39](#_Toc73064502)

[UI Layout 39](#_Toc73064503)

[DBConnectivityUI.java 40](#_Toc73064504)

[Generating table content 41](#_Toc73064505)

[Updating data from the table 41](#_Toc73064506)

[The Observer pattern 42](#_Toc73064507)

[User Manual (Other GUI panels) 43](#_Toc73064508)

[For customers 43](#_Toc73064509)

[Creating a new account 44](#_Toc73064510)

[Logging into the system 46](#_Toc73064511)

[Updating your information 46](#_Toc73064512)

[Creating a new order 47](#_Toc73064513)

[For employees 50](#_Toc73064514)

[Product/courier management 50](#_Toc73064515)

[Conclusion 52](#_Toc73064516)

[Group process 52](#_Toc73064517)

[Future expansion of the project 52](#_Toc73064518)

[References 53](#_Toc73064519)

# Introduction

The 2nd semester project is the final project in the first year of our Computer Science degree at UCN. The goal of this project was similar to the 1st-semester project in the sense that the objective of both was to develop a system that could be used in a company or a store to be more precise, to make the employees’ work easier than before, but the main difference between the first and the second project is that the requirements for the second were higher. To begin with, we had to create a program that works 100% in GUI and doesn't use TUI anywhere. Furthermore, this time we had to use SQL Database for storing all the information that we needed, and the Database server was provided by UCN (UCN's Hildur server). And lastly, we had to create test cases so that we could test all kinds of different scenarios to see how the program will react. For that purpose, we have used JUnit which provides a framework for the easy creation of automated tests. This project is meant to prepare us for future teamwork projects both in our studies and in our work experience.

Our company of choice was bio&bio – a Croatian company specialized in selling gluten-free, organic, and vegan products. We chose this company because they are relatively young, with aspirations to break into a large market niche and establish dominance on Croatia’s bio goods market.

# Section 1: Business

Gantt chart

Before we even began with interviews and a business case, we had to arrange our tasks in advance. We used a Gantt chart to aid us with that. A Gantt chart is a type of bar chart (Klein, 1999) (Richman, 2002) that illustrates a project schedule. (Kumar, 2005) This chart lists the tasks to be performed on the vertical axis, and time intervals on the horizontal axis. (Klein, 1999) (Richman, 2002)The width of the horizontal bars in the graph shows the duration of each activity (Richman, 2002) (Selig, 2008) Gantt charts illustrate the start and finish dates of the terminal elements and summary elements of a project. Terminal elements and summary elements constitute the work breakdown structure of the project. Modern Gantt charts also show the dependency (i.e., precedence network) relationships between activities. Gantt charts can be used to show current schedule status using percent-complete shadings and a vertical "TODAY" line. The Gantt chart we created is in the appendix files, under ***docs/Gantt Chart.xlsx.***

Company description

Bio&bio is a daughter company of Biovega. Biovega, a leader in the Croatian market of organic products which employs 170 people as of today, is a business concept driven by the life passion of its founders – Zlatko Pejic and Jadranka Boban Pejic. The motivation behind the establishment of the company is the two’s desire to share their ecology-based lifestyle with others and encourage positive change. The assortment started getting bigger and this is how bio&bio came to life – a retail chain of organic product stores that nowadays consists of 16 physical and one online store. As a further improvement of their business, in 2010 Biovega started to manufacture their products at the oldest eco field in Croatia - Zrno in Habjanovcu, and has combined the knowledge and experience of traditional agriculture with the ecological standards in food production. Together with Bio Zrna, they run the organic-based bistro Zrno as well, in which they use field-grown food. The whole field is dedicated to a zero-waste mission and is often cited as an example for future organic food projects in Croatia.

The company was founded in 1990 with an initial share capital of 2116 Croatian Kuna (HRK). Throughout the years, their capital has gradually increased, and today, their share capital is estimated at around 2.87 million HRK. In 2019, they reported an increase in sales of 6.3% (from 112 million to 119 million HRK) with earnings before interest, taxes, depreciation, and amortization, amounting to 4.1 million HRK.

Biovega works for the distribution and marketing of the best-quality certified organic products and collaborates with 53 international sellers and 160 domestic sellers.

* In 2019, Biovega has shown that most of the organic products were grown and produced in Croatia, which displays their support of the local producers;
* The quality of work and products is confirmed by inspections, and on average they have passed about 70 inspections with no mistakes;
* After monitoring the system and the quality of the food by the SGS company, there were no mistakes or improvement suggestions.

They have 3 fundamental values for their work culture and wellbeing:

1. ***Ethics*** - they treat employees, partners, and customers in compliance with law enforcement and the ethics of business. They invest in the education of their employees and work on improving the quality of business, life, and development. The average age of the employees is 37 years, and 65% of them are female. The average salary in 2019 was 6525 HRK and they have invested around 400 thousand HRK in the education of their employees.
2. ***Ecology*** - they encourage ecological production, recycling, and energy and resource-saving.
3. ***Economy*** – they invest in the development of the business (complying with ISO and HACCP quality standards), making their business available to both customers and employees.

According to the company, the goals they have achieved in 2019 are:

1. Improving their leading position on the market;
2. Growing the number of customers and improving their opinions;
3. Improving their employees' opinions.

*„We have reached these goals by keeping our authenticity, improving our market activities, investing in education, continuing to work on the reduction of control and expenses, and thanks to this, we have finished the plan and improved our finances.“*

* ***Annual report of BioVega, 2019***

These are some of the main risks the company has mentioned:

* Market risks: currency risk, interest rate risk, and market liquidity risk
  1. Currency risk: The company is exposed to the risk of changes in foreign exchange rates. Their primary currency is EUR. The business risk in foreign currencies arises from recognized assets and liabilities. Receivables whose source of finances is denominated in EUR are secured against the risk of a sudden change in foreign exchange rates through contracts containing a currency clause. That's how the risk of changes in the exchange rate is minimized.
  2. Interest rate risk: Interest rate risk is the potential for investment losses that result from a change in interest rates. If interest rates rise, for instance, the value of a bond or other fixed-income investment will decline. The change in a bond's price given a change in interest rates is known as its duration. Interest rate risk can be reduced by holding bonds of different durations, and investors may also allay interest rate risk by hedging fixed-income investments with interest rate swaps, options, or other interest rate derivatives. (Chen, 2021)
  3. Market liquidity risk: Market or asset liquidity risk is asset illiquidity. This is the inability to easily exit a position. For example, we may own real estate but, owing to bad market conditions, it can only be sold imminently at a fire sale price. The asset surely has value, but as buyers have temporarily evaporated, the value cannot be realized. (Harper, 2020)
* Credit risk: The company has made deals with local banks that improve the need for liquidity.
* Liquidity risk: Liquidity risk is the risk of uncollected receivables from customers, which can be reflected in the quality of serving their obligations. Given the contractual terms and internal procedures, they require continuous partner analysis, business monitoring, and insurance through payment instruments, so that the liquidity risk is reduced.
* Operational risk: The company's operating risk represents the risk of loss that may arise as a consequence of inappropriate or ineffective internal procedures, the human factor, or external events. As part of operational risk management, there is a system for monitoring potential and recognizing operational risks.
* Personnel risk: The personnel risk is the fluctuation of employees in the company. The goal stabilization teams are continuously working to improve working conditions through education and remuneration.

## Cost-benefit analysis

A cost-benefit analysis is a systematic process that businesses use to analyze which decisions to make and which to forgo. The cost-benefit analyst sums the potential rewards expected from a situation or action and then subtracts the total costs associated with taking that action. Some consultants or analysts also build models to assign a dollar value on intangible items, such as the benefits and costs associated with living in a certain town. (Mindtools, 2016)

* Given an estimated hourly rate of **1000 Danish Krones** (DKK) (approx. 1010 HRK) and time for programming the solution about **120-130 hours**, we estimate that this project may cost around **121-131 thousand HRK**. It is a good idea for the company to perform a cost-benefit analysis with the inclusion of the costs for developing company software by outside programmers hired for this task, so that they have a more precise estimate of their new expenses regarding the development itself, and the revenue the company is expecting to gain after the improved software. The company expects the following benefits from the new system:
* Improved employee experience 🡪 increased productivity;
* Increased customer interest towards the products offered by the company 🡪 increased market share;
* Better customer experience brought about by the better and easier to use software à increased popularity and, respectively, profit.

A further step of the cost-benefit analysis would be to assign a monetary value to the company’s costs, as calculated above, as well as to its benefits. These are trickier to calculate and, in the company’s word, they’re unable to provide us with an estimate of the monetary value of the benefits they’re planning to gain through the new software. Some of the intangible (soft) benefits would be:

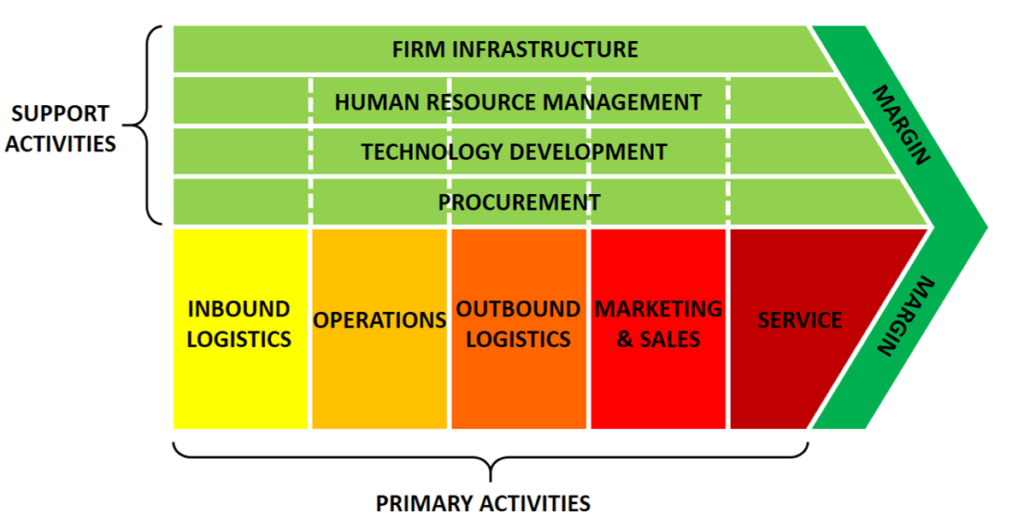
* Smaller environmental footprint – positive impact on the environment in the long term;
* Supporting local producers/farmers;
* Increased employee satisfaction – thanks to the new software, customer interests would be easier to track and it would also enable improved data analysis. This could potentially lead to reduced working hours and thus, an elevation in employee satisfaction.

## Supply chain management (SCM)

**Supply Chain Management (SCM)** is the coordination/management of all supply activities (production, services, information, etc.) of an organization from its suppliers and partners to its customers. (Kondylis & Mertz, 2021) In a company, the supply chain manager should take care of planning and strategies, the sources of raw materials, deliveries, the efficiency of manufacturing, and the return system (e.g., for defective materials). Things to keep in mind when organizing the supply chain management would be, for instance, ways to minimize costs along the way and reduce the food waste when supplying the company with products, in bio&bio’s case. The process components of Supply Chain Management are:

1. Plan – contacting local producers or providers that could supply raw materials;
2. Source – organic farms and certified Croatian organic producers;
3. Make – the actual manufacturing process during which the organic products are brought to life;
4. Deliver – sending them across stores or warehousing them and preparing for delivery;
5. Return – monitoring the produce for defective products and having a return mechanism at the ready.

## Value chain analysis



*“The idea of the value chain is based on the process view of organizations, the idea of seeing a manufacturing (or service) organization as a system, made up of subsystems each with inputs, transformation processes, and outputs. Inputs, transformation processes, and outputs involve the acquisition and consumption of resources – money, labor, materials, equipment, buildings, land, administration, and management. How value chain activities are carried out determines costs and affects profits.”*

***-*** (Porter, 1985; Rowe, 1994)

* **The inbound logistics** of the company refer to company-appointed buyers who select the organic crops which are to be used in the production process.
* **Operations –** the company operates on the Croatian market through company-operated stores.
* **Outbound logistics** are not a key part of the business as the company doesn’t focus on selling to retail stores or big supermarket chains.
* In terms of **marketing and sales,** the company invests more in a quality product but also relies on online marketing to distribute its products and popularize itself.
* **Service –** the high-quality customer service and loyalty obtained in the physical stores, as well as online delivery, is of high value to the company.
* **Infrastructure** - this includes departments like management, finance, legal, etc., which are required to keep the company’s stores operational. The company has its management department in its main office, while on-site it employs friendly employees with broad knowledge in the field of organic nutrition.
* **Human resource management –** the company keeps a low turnover of employees and organizes many pieces of training and seminars to keep their motivation and skill levels high.
* **Technology development –** the company implements only high-functioning technology when manufacturing the organic products it sells and it’s currently investing in better, customer-and-employee-friendlier software and better website interface.
* **Procurement –** the company is entirely responsible for the supply of raw materials.

## ERP & CRM

**Enterprise resource planning (ERP)** is a business process management software that allows an organization to use a system of integrated applications to manage the business and automate many back-office functions related to technology, services, and human resources. (Kondylis & Mertz, 2021) The benefits of an ERP system come from having a single, shared database for all financial and operational data. This greatly impacts reporting — both static monthly reports and ad hoc reports requested by leadership. A single source of financial and operational data also means employees can drill down into reports to uncover financial insights without the need for IT or finance teams to conduct the analysis and reporting. This allows businesses to make faster, data-backed decisions that can impact everything from profitability to new growth opportunities to creating efficiency across the organization. (Beal, 2020)

**Customer relationship management** (CRM) is a process in which a business or other organization administers its interactions with customers, typically using data analysis to study large amounts of information. (Bardicchia, 2020) The central promise of CRM is to give the business a central repository of all customer data, tracking all customer interactions. Armed with this information and using analytics, businesses can make more informed decisions on which customers to pursue added revenue, how sales teams are performing, how to service customers efficiently and appropriately, and more. (Beal, 2020)

A CRM is more fitting for this type of company because it's not a big enterprise, but a business that relies heavily on its direct contact with customers. Therefore, the “value for money” method would be the most effective in this case as part of the CRM system. It consists of convincing the customer that they're being offered the best possible quality on the market for the amount they're willing to pay. Furthermore, this system allows tracking down customer analytics and identify potential loyal customers who would generate revenue, and use the methods of “customer binding” to attract them.

*“However, our main asset is the bio&bio employees who approach each customer individually, passing on their knowledge of healthy and natural nutrition, which is the result of continuous education and personal experience.”*

* ***bio&bio.hr***

## E-commerce and online marketing

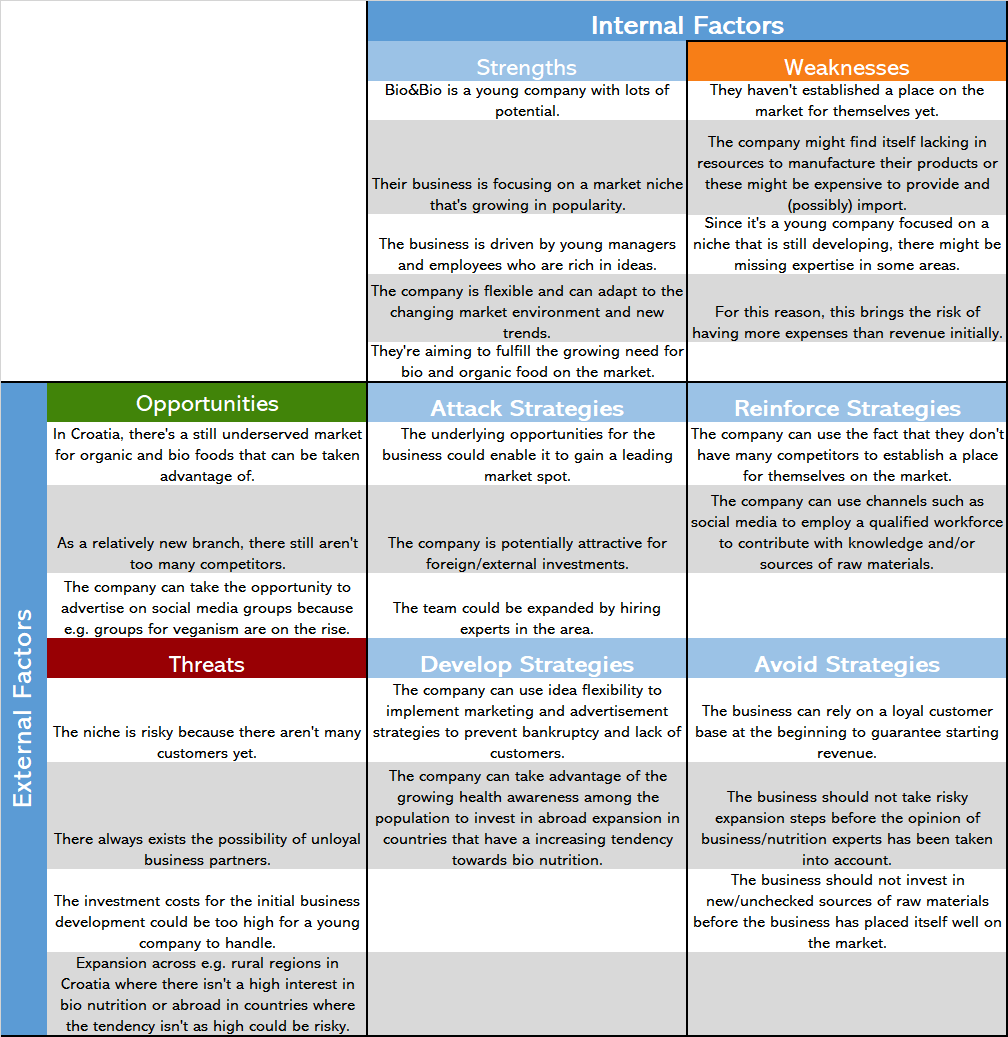
eCommerce refers to any form of business transaction conducted online. The most popular example of eCommerce is online shopping, which is defined as buying and selling goods via the internet on any device. However, eCommerce can also entail other types of activities, such as online auctions, payment gateways, online ticketing, and internet banking. (Oberlo, n.d.)

The company we’re working with has chosen to rely on online marketing for the most part, especially given the COVID situation around the world, opting for a safer and faster approach to reach their clients. We recommended them the following types of online marketing:

1. **Social media advertising –** through relevant ads on social media such as Facebook and Instagram, potential customers could get an impression of the products offered; the company could benefit from this type of advertising because the products are mostly aimed at a health-conscious target group which is expected to use social media and Internet sources on healthy eating and nutrition. Another way to use this method would be working with affiliates and influencers by paying them to have their product advertised on the respective persons’ social media pages.
2. **Cookie tracking** – by tracking the search history, ads could be shown to potential customers who could be interested in the company’s products. By following customer analytics, businesses could build a loyal customer network and specify their ads to be more relevant to users.
3. **SEO & SEM** - SEO is a method of ranking a website on organic search results, which requires a high advertising budget and relatively big website popularity. Whenever clients are looking for similar products to the ones that the company sells, they use keywords on search engines such as Google, which brings out a certain number of search results, but SEM focuses on a pay-per-click approach and displays the advertisement of the company *above* the search results, which could catch the attention of interested customers.
4. **Affiliate marketing -** Affiliate marketing is the process by which an affiliate earns a commission for marketing another person’s or company’s products. (Enfroy, n.d.) In the case of the company we’re working with, they could use this method to advertise the products of the umbrella company they originated from (Biovega) and make a profit.

## SWOT analysis & TOWS matrix

A SWOT analysis is a compilation of your company's strengths, weaknesses, opportunities, and threats. The primary objective of a SWOT analysis is to help organizations develop a full awareness of all the factors involved in making a business decision. (Schooley, 2019) The table shown on the next page displays a SWOT analysis of the company, as well as a TOWS matrix, used to match the environmental threats and opportunities with the company's weaknesses and especially its strengths.



# Section 2: System Development

**Note:** If the diagrams here are unclear, they can be found as .uxf files in the project appendices, under the **diagrams** folder. The prefix notations are as follows:

* SSD – System Sequence Diagram
* CD – Communication Diagram
* DCD – Design Class Diagram

## Unified Process

Unified process (UP) is an architecture-centric, use-case-driven, iterative, and incremental development process that leverages unified modeling language and is compliant with the system process engineering metamodel. The unified process can be applied to different software systems with different levels of technical and managerial complexity across various domains and organizational cultures.

A unified process is a refinement of a rational unified process. It is an extensible framework that can be customized for specific projects.

This process divides the development process into four phases:

### Inception

Inception is the smallest phase in the project, and ideally, it should be quite short. If the Inception Phase is long then it may be an indication of the excessive up-front specification, which is contrary to the spirit of the Unified Process. (Wikipedia, n.d.)

The following are typical goals for the Inception phase:

* Establish;
* Prepare a preliminary project schedule and cost estimate;
* Feasibility;
* Development of a business case.

### Elaboration

During the Elaboration phase, the project team is expected to capture a healthy majority of the system requirements. However, the primary goals of Elaboration are to address known risk factors and to establish and validate the system architecture. Common processes undertaken in this phase include the creation of use case diagrams, conceptual diagrams (class with only basic notation), and package diagrams (architectural diagrams). (Wikipedia, n.d.)

The architecture is validated primarily through the implementation of an Executable Architecture Baseline. This is a partial implementation of the system which includes the core most architecturally significant components. It is built in a series of small time-boxed iterations. By the end of the Elaboration phase, the system architecture must have stabilized, and the executable architecture baseline must demonstrate that the architecture will support the key system functionality and exhibit the right behavior in terms of performance, scalability, and cost. (Wikipedia, n.d.)

The final Elaboration phase deliverable is a plan (including cost and schedule estimates) for the construction phase. At this point the plan should be accurate and credible since it should be based on the Elaboration phase experience and since significant risk factors should have been addressed during the Elaboration phase. (Wikipedia, n.d.)

### Construction

Construction is the largest phase of the project. In this phase, the remainder of the system is built on the foundation laid in Elaboration. System features are implemented in a series of short, time-boxed iterations. Each iteration results in an executable release of the software. It is customary to write full-text use cases during the construction phase and each one becomes the start of a new iteration. Common Unified Modeling Language (UML) diagrams used during this phase include activity diagrams, sequence diagrams, collaboration diagrams, State Transition diagrams, and interaction overview diagrams. Iterative implementation for the lower risks and easier elements are done. The final Construction phase deliverable is software ready to be deployed in the Transition phase. (Wikipedia, n.d.)

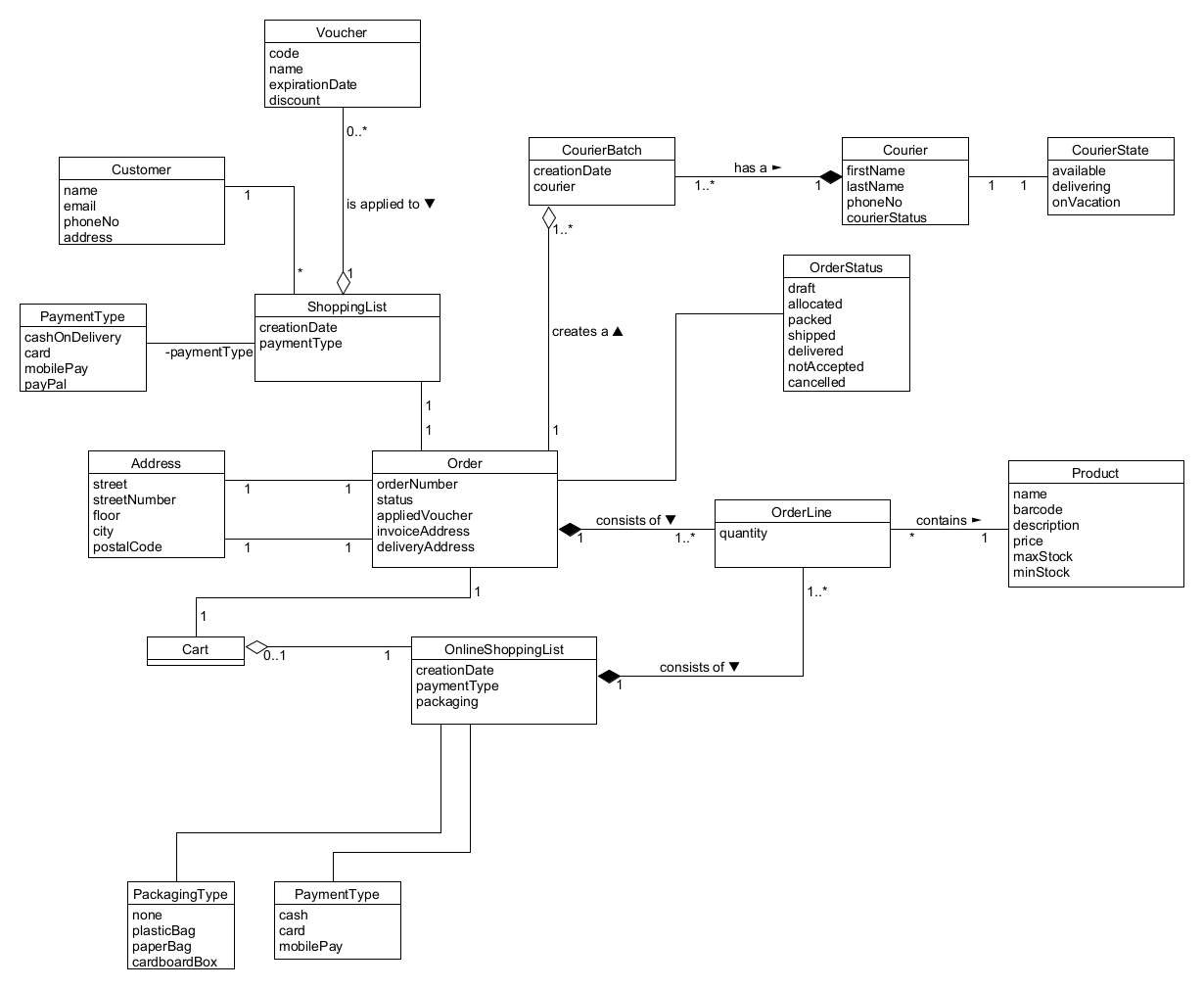
### Transition

The final project phase is Transition. In this phase, the system is deployed to the target users. Feedback received from an initial release (or initial releases) may result in further refinements to be incorporated over the course of several Transition phase iterations. The Transition phase also includes system conversions and user training. (Wikipedia, n.d.)

## Domain Model

A domain model is the most important and classic model of analysis. It illustrates noteworthy concepts in a domain. It can act as a source of inspiration for designing some software objects.

Our domain model shows the core of the whole system in a simplified manner, from the creation of Customer to Order fulfillment, to the whole delivery process and payments.



## Use Cases

A *use case* is a list of actions or event steps typically defining the interactions between an actor and a system to achieve a goal. We can have a written Use Case - an example would be Fully Dressed Use Case, or we can have a graphical Use Case made using UML Standard called Use Case Diagram.

### Fully Dressed Use Case

A fully Dressed Use Case is a carefully structured and detailed description enabling a deep understanding of the goals, tasks, and requirements. It goes through every step of the use case and shows all the possibilities and situations for specific use cases. A fully Dressed Use Case is reserved for the most important Use Cases only since there is no need to make every Use Case as detailed as possible.

#### Order Processing – In-Store: Fully Dressed Use Case

|  |  |  |
| --- | --- | --- |
| Use Case name | **Order Processing** | |
| Actors | Customer, Clerk | |
| Pre-conditions | Products have been put on the conveyor; the Customer has proceeded to checkout | |
| Post-conditions | A receipt has been printed; Transaction has been approved | |
| Frequency | Approx. 50 orders/day | |
| Flow of events | **Actor** | **System** | |
| 1. The customer presents a product. |  | |
| 2. The clerk scans the product. | 3. The system updates the current order list and subtotal price. | |
| *Steps 1-3 are repeated until the customer is satisfied* | | |
| 4. The customer proceeds to checkout. | 5. The system displays the total amount to pay. | |
| 6. The customer pays the displayed amount. |  | |
| 7. The clerk prints an invoice with all the purchased products, total price, etc. | 8. The system resets and is ready for a new order. | |
| Alternative flows | **1a. The customer decides to return an item after it has been scanned.** | | |
| 1. The clerk removes the product from the order. | 2. The system removes the product from the customer’s shopping cart and updates the subtotal. | |
| **4a. The customer applies a voucher.** | | |
| 1. The customer presents a voucher. |  | |
| 2. The clerk scans the voucher. | 3. The system reads the code, finds the corresponding discount percentage, and subtracts the calculated amount from the total price. | |
| **4b. The voucher is expired, invalid, or has been already used.** | | |
|  | 1. The system displays an error message and the price is not updated. | |
| **6a. The customer’s payment method is rejected.** | | |
| 1. The clerk prompts the customer for another attempt. |  | |
| 2. The customer attempts to pay again. | 3. After 3 unsuccessful attempts, the entire order is canceled. | |
|  | 4. The system resets and is ready for a new order. | |
| Special Requirements | · There must be at least 1 product before the customer can proceed to checkout.  · The customer may use only 1 voucher per order. | |

#### Order Processing – Online: Fully Dressed Use Case

|  |  |  |
| --- | --- | --- |
| Use Case name | **Order Processing** | |
| Actors | Customer, Clerk, Courier | |
| Pre-conditions | Products have been selected in the shopping cart; the Customer has proceeded to checkout | |
| Post-conditions | A receipt has been printed; Transaction has been approved | |
| Frequency | Approx. 20 orders/day | |
| Flow of events | **Actor** | **System** | |
| 1. The customer adds a product to the cart. | 2. The system updates the current order list and subtotal price, including delivery fees (if any). | |
| *Steps 1-2 are repeated until the customer is satisfied.* | | |
| 3. The customer proceeds to checkout. | 4. The system prompts the user to input data like phone number, address, etc. | |
| 5. The customer inputs the requested data. | 6. The system prompts the user to pay the displayed amount. | |
| 7. The customer pays the displayed amount. | 8. The system notifies the clerk that there is a new online order. | |
| 9. The clerk dispatches an available courier. | 10. The system resets and is ready for a new order. | |
| 11. The courier notifies the clerk that the order has been delivered. | 12. The system sends a message to the customer that their package has been delivered. | |
|  |  | 13. The courier is marked as available for new deliveries. |
| Alternative flows | **1a. The customer decides to remove an item from the shopping cart.** | | |
|  | 1. The system removes the product from the customer’s shopping cart and updates the subtotal. | |
| **1b. The customer’s order exceeds 299HRK.** | | |
|  | 1. The system notifies the user that delivery fees are lifted. | |
| **3a. The customer applies a voucher.** | | |
| 1. The customer inputs a voucher code. | 2. The system reads the code, finds the corresponding discount percentage, and subtracts the calculated amount from the total price. | |
| **3b. The voucher is expired, invalid, or has been already used.** | | |
|  | 1. The system displays an error message to the user and the price is not updated. | |
|  | **4a. The customer orders for the first time.** | |
|  |  | 1. The user’s information is persisted for further orders. |
|  | **11a. The delivery address is not available or invalid.** | |
|  | 1. The courier sends the new delivery address (a nearby kiosk, for example). | 2. The system sends an email to the user, informing them that the package has been delivered to the new address. |
| Special Requirements | · There must be at least 1 product before the customer can proceed to checkout.  · The customer may use only 1 voucher per order. | |

#### Product CRUD: Fully Dressed Use Case

|  |  |  |
| --- | --- | --- |
| Use Case name | **Product CRUD** | |
| Actors | Clerk | |
| Pre-conditions | The clerk has logged in to the system; Clerk has inputted all required data | |
| Post-conditions | A product has been created; Products have been listed; Product has been edited; Product has been deleted. | |
| Frequency | Approx. 100 requests/day | |
| Flow of events | **Actor** | **System** |
| **Create:** | |
| 1. The clerk chooses the option to create a new product. | 2. The system presents a form for product information. |
| 3. The clerk inputs all required product information. |  |
| 4. The clerk confirms product creation. | 5. The system reads information from the form and persists a new product. |
| **Retrieve:** | |
| 1. The clerk chooses the option to view a list of products. | 2. The system presents a list of all products. |
| **Update:** | |
| 1. The clerk chooses the option to edit the product. | 2. The system presents a form for product editing. All the information about the product is taken from the persistence. |
| 3. The clerk edits product information. |  |
| 4. The clerk confirms product editing. | 5. The system reads information from the form and looks up the product in the persistence, after which the respective values are updated. |
| **Delete:** | |
|  | 1. The clerk selects the product and chooses the option to delete the product. | 2. The system displays a prompt to the user. |
|  | 3. The clerk confirms the deletion. | 4. The product is looked up in the persistence, after which it is deleted. |

#### Courier CRUD: Fully Dressed Use Case

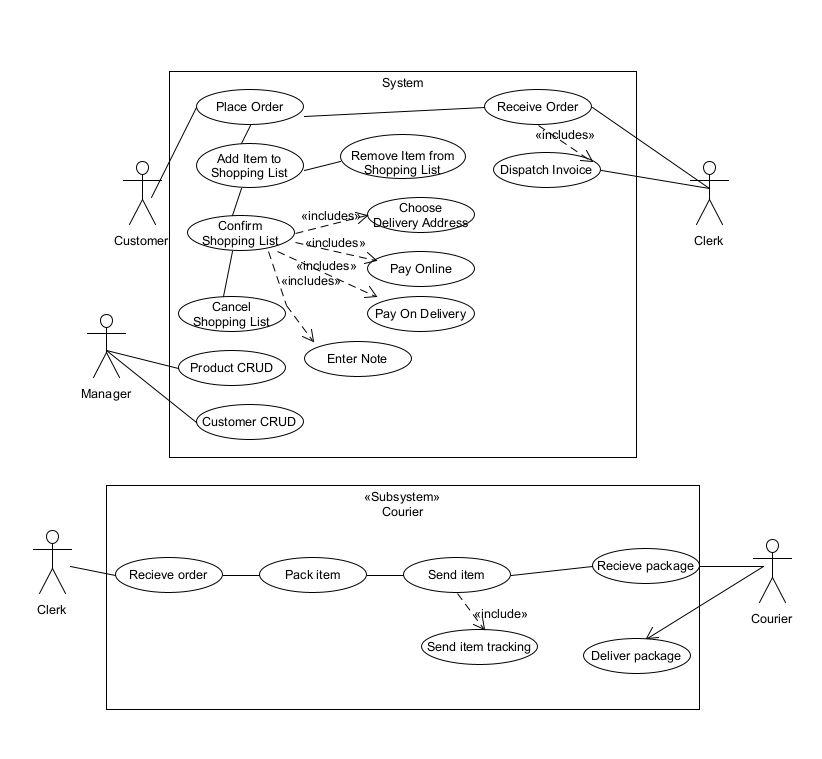
|  |  |  |
| --- | --- | --- |
| Use Case name | **Courier CRUD** | |
| Actors | Clerk | |
| Pre-conditions | The clerk has logged in to the system; Clerk has inputted all required data | |
| Post-conditions | A courier has been created; Couriers have been listed; Courier has been edited; Courier has been deleted. | |
| Frequency | Approx. 30 requests/week | |
| Flow of events | **Actor** | **System** |
| **Create:** | |
| 1. The clerk chooses the option to create a new courier. | 2. The system presents a form for courier information. |
| 3. The clerk inputs all required information about the courier. |  |
| 4. The clerk confirms courier creation. | 5. The system reads information from the form and persists a new courier. |
| **Retrieve:** | |
| 1. The clerk chooses the option to view a list of couriers. | 2. The system presents a list of all couriers. |
| **Update:** | |
| 1. The clerk chooses the option to edit the courier. | 2. The system presents a form for courier editing. All the information about the courier is taken from the persistence. |
| 3. The clerk edits courier information. |  |
| 4. The clerk confirms courier editing. | 5. The system reads information from the form and looks up the courier in the persistence, after which the respective values are updated. |
| **Delete:** | |
|  | 1. The clerk selects the courier and chooses the option to delete the courier. | 2. The system displays a prompt to the user. |
|  | 3. The clerk confirms the deletion. | 4. The courier is looked up in the persistence, after which it is deleted. |

#### Customer CRU: Fully Dressed Use Case

|  |  |  |
| --- | --- | --- |
| Use Case name | **Customer CRU** | |
| Actors | User | |
| Pre-conditions | User has inputted all required data | |
| Post-conditions | A customer has been created; Customers have been listed; User has been edited; User has been deleted. | |
| Frequency | Approx. 30 requests/week | |
| Flow of events | **Actor** | **System** |
| **Create:** | |
| 1. The user chooses the option to register as a new customer. | 2. The system presents a form for customer information. |
| 3. The user inputs all required customer information. |  |
| 4. The user confirms customer creation. | 5. The system reads information from the form and persists a new customer. |
| **Retrieve:** | |
| 1. The user chooses the option to log in to the system. | 2. The system presents a form with a field for the user to input his phone number. |
| 3. The customer inputs his phone number. | 4. The system validates the number and displays the customer’s data. |
| **Update:** | |
| 1. The user chooses the option to edit the customer. | 2. The system presents a form for customer editing. All the information about the customer is taken from the persistence. |
| 3. The user edits customer information. |  |
| 4. The user confirms customer editing. | 5. The system reads information from the form and looks up the customer in the persistence, after which the respective values are updated. |

### Use Case Diagram

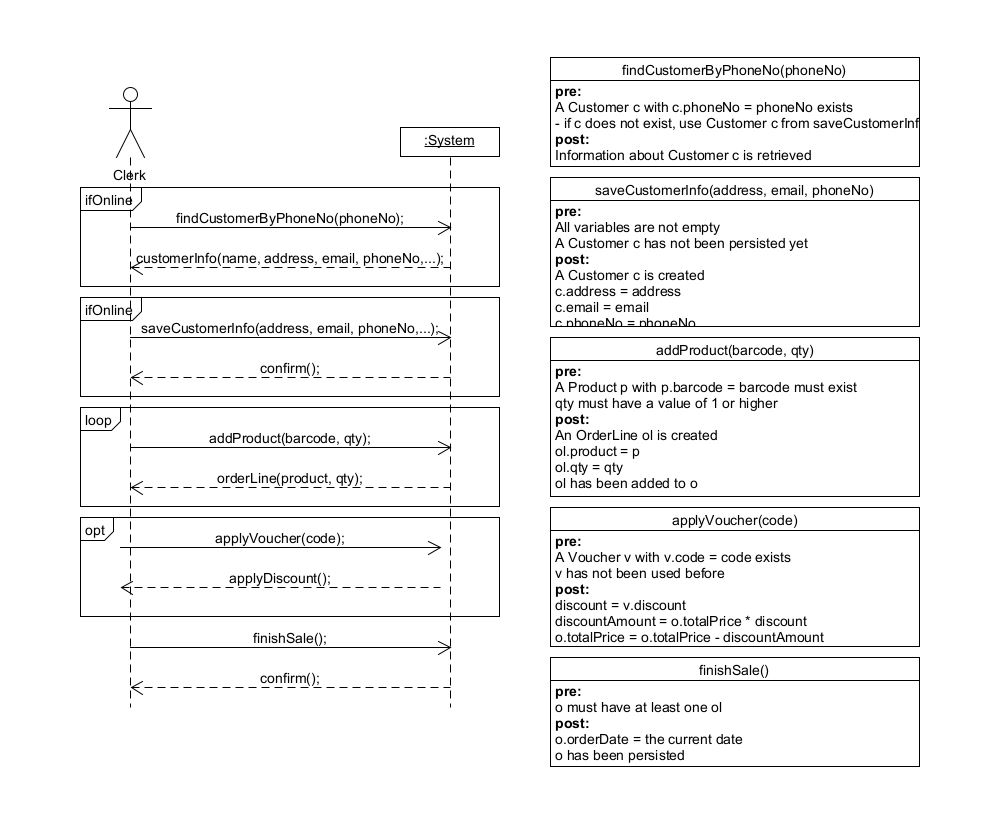
Use case diagrams are usually referred to as behavior diagrams used to describe a set of actions (use cases) that some system or systems (subject) should or can perform in collaboration with one or more external users of the system (actors). Each use case should provide some observable and valuable result to the actors or other stakeholders of the system. As visible in our diagram, the Customer can place an order, add, and remove the items from the shopping list, confirm the shopping list which includes the delivery address, online payment, and the delivery payment along with note entering. The manager has the option of the product CRUD and the customer CRUD. The clerk is there to receive the order and to dispatch the invoice.



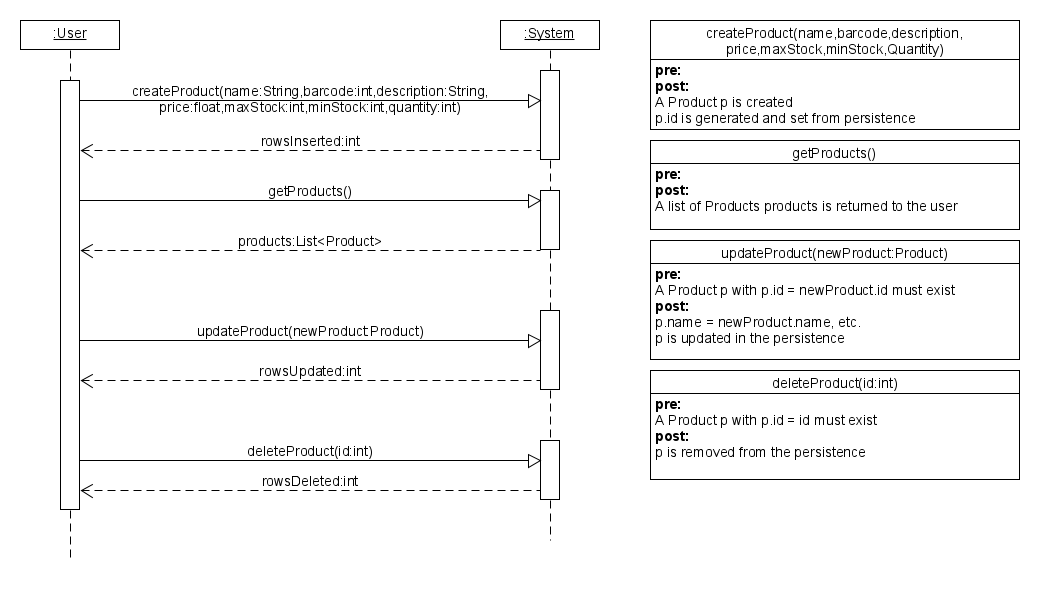
# System Sequence Diagram

A sequence diagram is a type of interaction diagram because it describes how—and in what order—a group of objects works together. These diagrams are used by software developers and business professionals to understand requirements for a new system or to document an existing process. Sequence diagrams are sometimes known as event diagrams or event scenarios. This is our System Sequence Diagram and as it can be seen below, it thoroughly explains the activities between a clerk and the system. The clerk can find the customer by phone number, save the customer’s info, add products, apply vouchers, and finish sales.

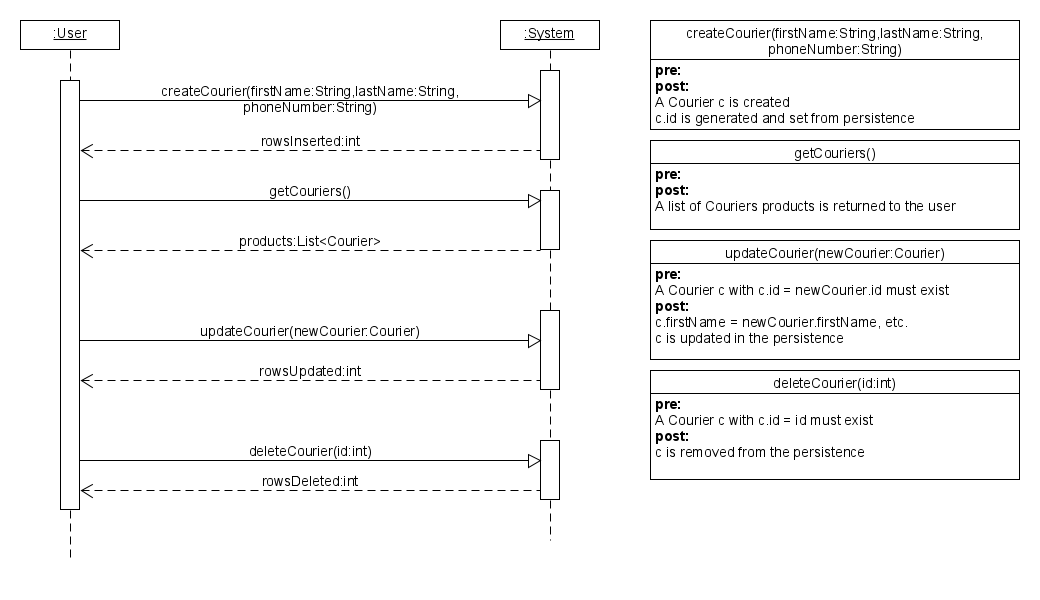
### Order Processing SSD



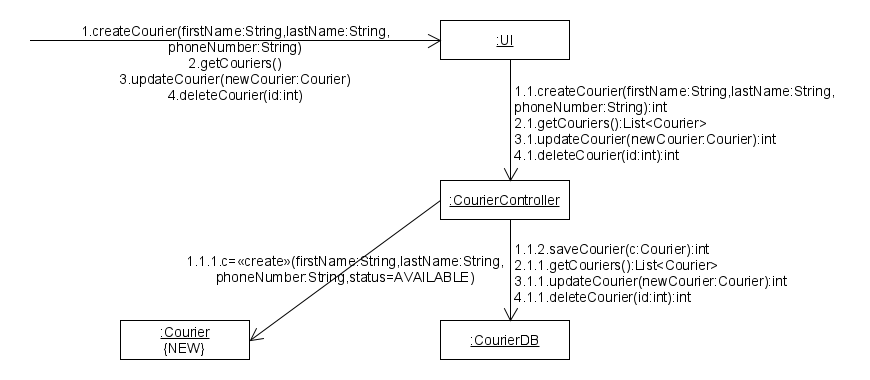
### Product CRUD



### Courier CRUD SSD



### Customer CRU SSD

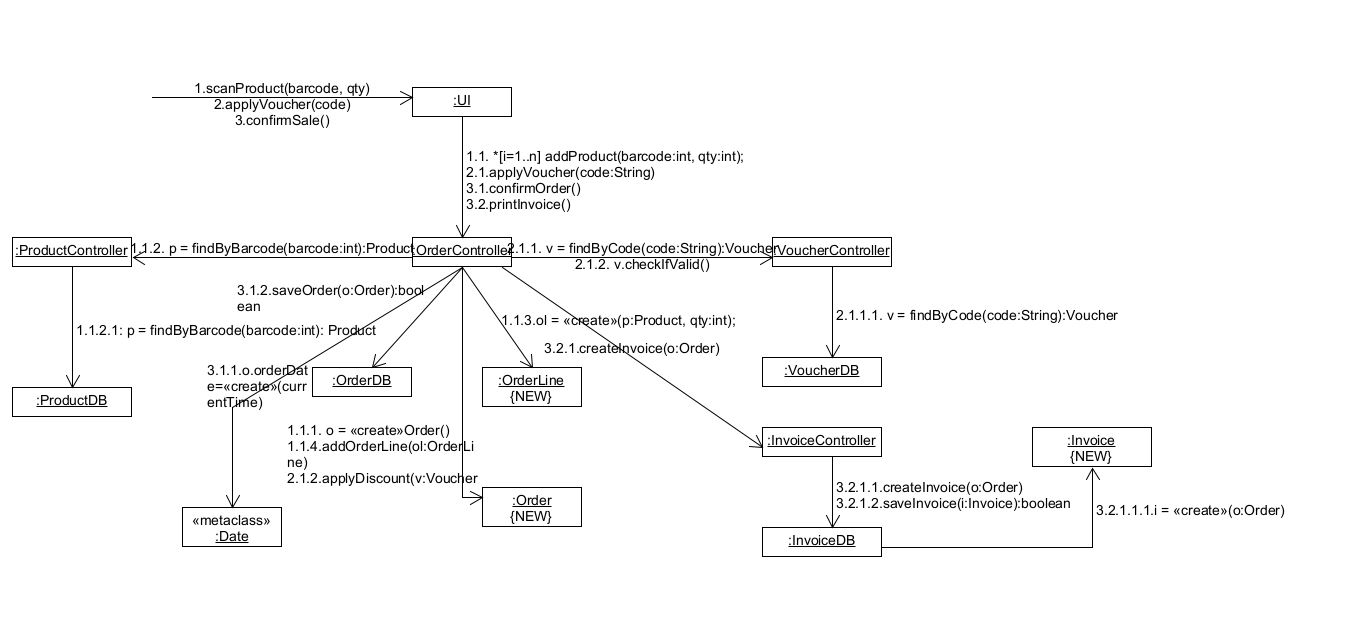


## Communication Diagrams

UML Communication Diagrams, previously known as collaboration diagrams, are a type of behavioral diagram that shows the interactions that take place between objects in a piece of software or system. This type of diagram emphasizes the messages exchanged between objects. Communication diagrams are best used when one use case has multiple scenarios that need depiction.

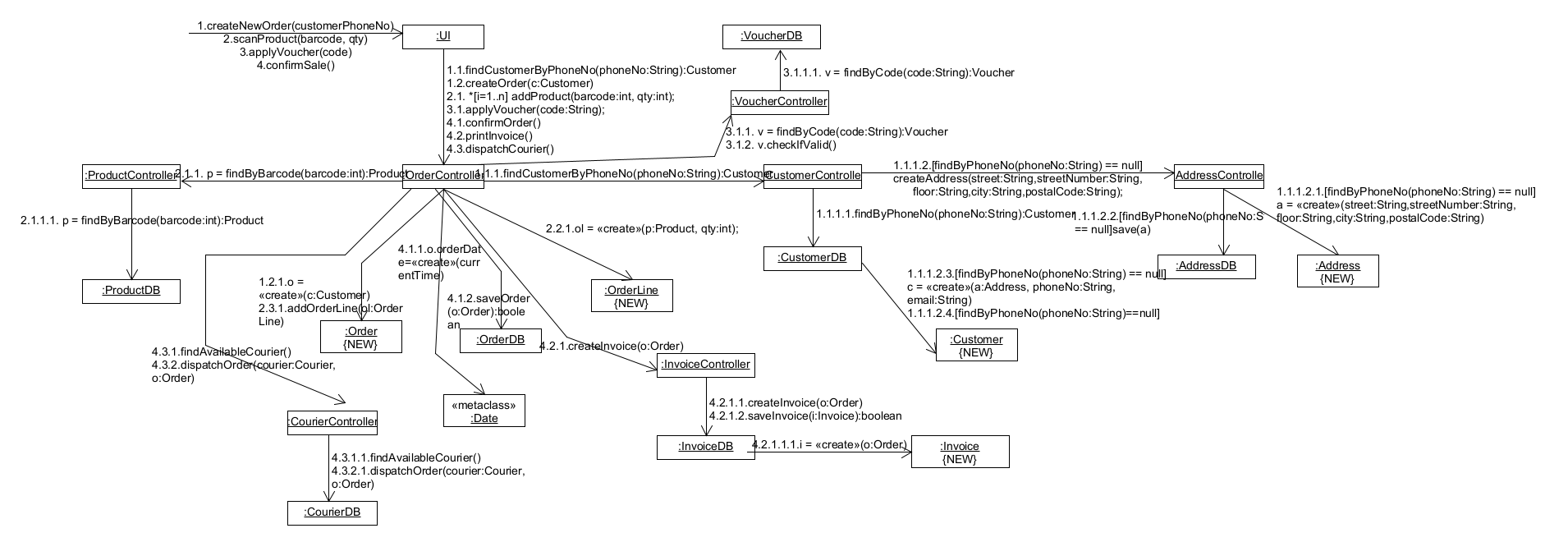
### In-Store Order Processing Communication Diagram

Our communication diagram shows the whole process starting with the UI, communicating with the OrderController, which can then communicate with the VoucherController, ProductController, InvoiceController, OrderDB, OrderLine, Order, and the metaclass Date. Through the ProductController it can communicate with the ProductDB; through the VoucherController it communicates with the VoucherDB, and the InvoiceController communicates with the InvoiceDB in which we can create new Invoices.

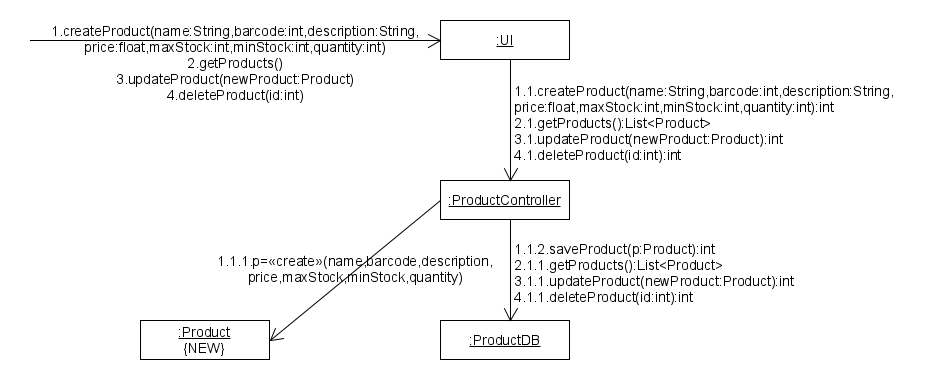


### Online Order Processing Communication Diagram

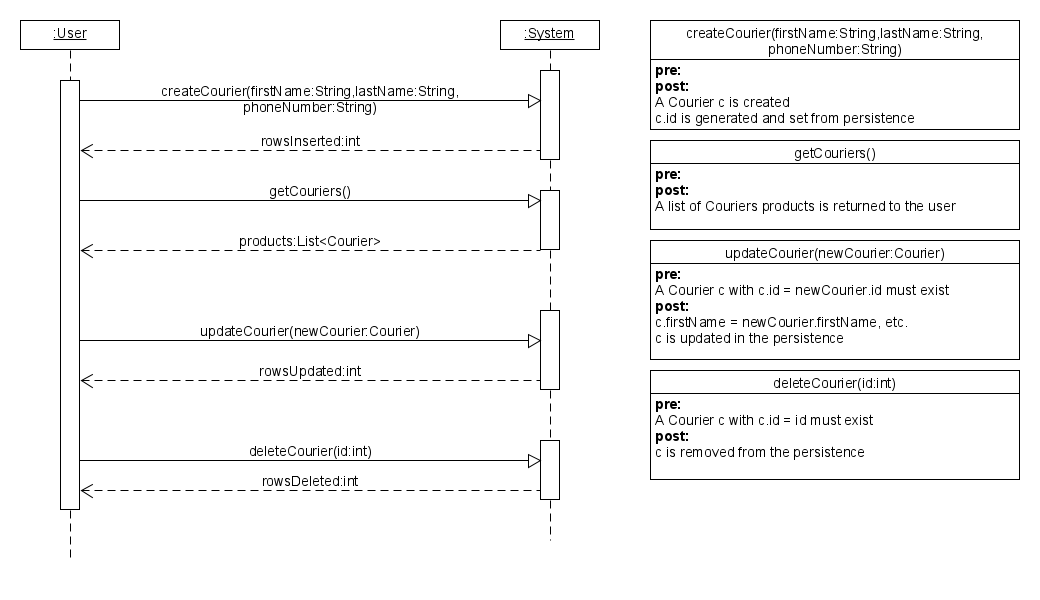
This is our online order processing communication diagram; it shows similar communications to the in-store one, but online. It also begins with the UI having communications with the Controllers. The difference is that here we have the CourierController along with the CourierDB, as we need to find couriers and dispatch orders, and there is also a CustomerController, which contacts the CustomerDB and creates new customers, also it contains the AddressController with the AddressDB and the ability to create a new address.



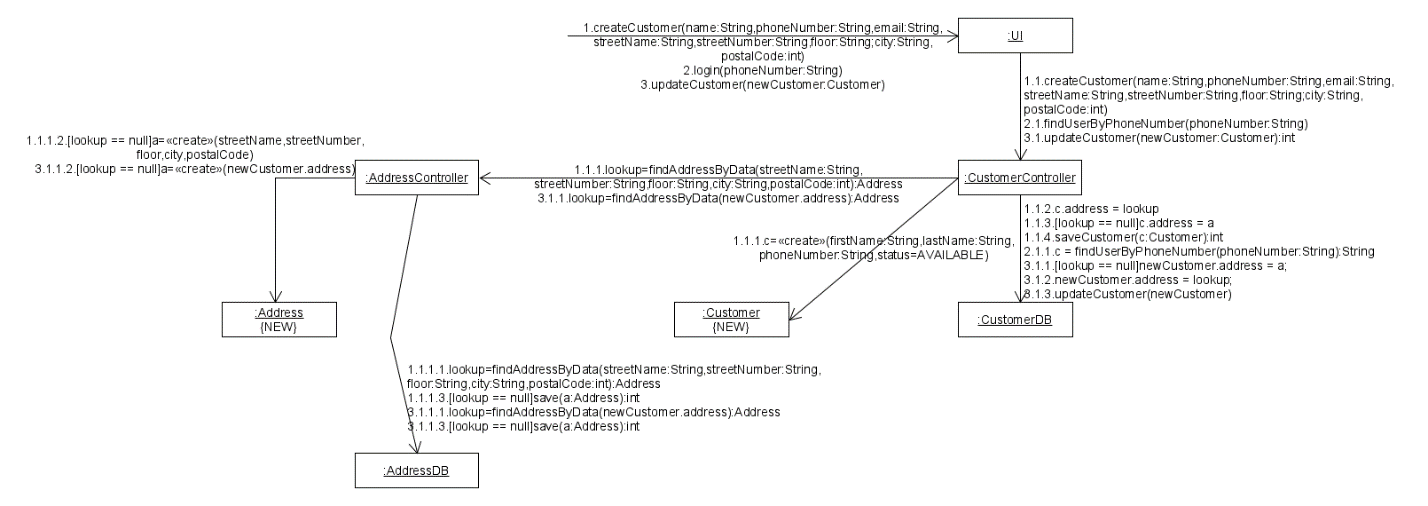
### Product CRUD Communication Diagram



### Courier CRUD Communication Diagram



### Customer CRU Communication Diagram



## Design Class Diagram

From a conceptual perspective, the class diagram can be used to visualize a domain model. For discussion, we also need a unique term to clarify when the class diagram is used from a software or design perspective. A common modeling term for this purpose is the design class diagram. In our words, the Design Class Diagram is an expanded Domain Model because it shows all the packages with all classes, with their functions and interaction between other classes in the system. It’s much more detailed and as such, it requires more technical knowledge and is not part of what the customer gets to see.

Our Design Class Diagram shows how we have utilized a 3-layer structure where we have Controller layer, Models layer, and Database layer. What all that means is that the Controller has to go through the Database to access the model or for example, a user of the system can’t just access the Model layer from the uppermost layer because the top layer isn’t even aware of its existence.

# Section 3: Programming

## Mapping the domain model to a relational model

The first step in creating the solution is to map the domain model into a relational model which will be used to create the tables necessary for the solution. When it comes to mapping the domain model, there are several rules which need to be followed:

* Multivalued attributes are not allowed;
  + There must not be a single cell representing multiple values (emails, educations, etc.)
* Composite attributes are best represented by their components;
  + For example – a name should be split into first and last name, and address should be broken into its basic attributes (number, floor, etc.), etc.
* Minimizing null values.

The general rule of thumb is that each conceptual class is mapped to a table, and its attributes are the columns of the table. A primary key must be identified (most commonly – an ID) for uniqueness, and all relations are indicated with foreign keys. If there are multi-valued attributes, they are converted to a separate table, where the primary key of the original table is turned into a foreign key (Fig. 2).

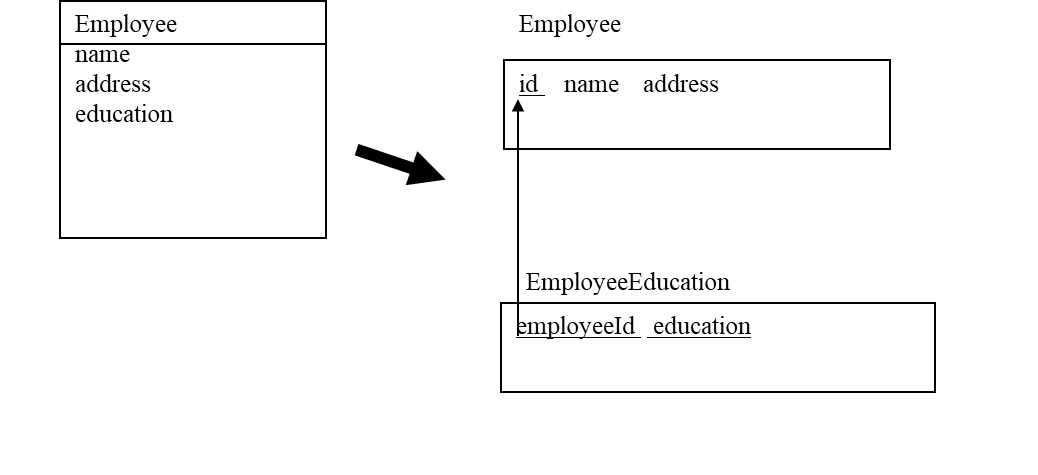


Fig. 2. Mapping a multi-valued attribute

There are different rules for the different associations between the domain model conceptual classes:

* **One-to-one (1-1):** We include the primary key from one of the sides on the other side as a foreign key;
* **One-to-many (1-\*):** We include the primary key from the one-side on the many-side as a foreign key;
* **Many-to-many (\*-\*):** We create a new table with the primary keys from both sides as foreign keys. The combination of the two foreign keys becomes a primary key in the new table.

## Writing the SQL scripts

The technology of choice for keeping the application persistent is the **Microsoft SQL Server**. The language for writing queries for said technology is **Structured Query Language (SQL)**. The database is located at **hildur.ucn.dk** under the name ***dmaj0920\_1086341***. The SQL queries are in the Eclipse project, under the *src/sql* folder.

### Database isolation levels

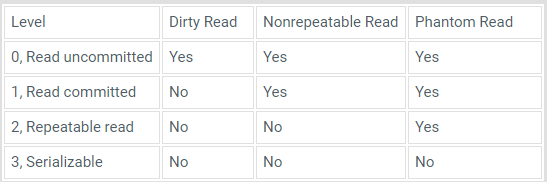
An isolation level represents a particular locking strategy employed in the database system to improve data consistency. The higher the isolation level, the more complex the locking strategy behind it. The isolation level provided by the database determines whether a transaction will encounter the following behaviors in data consistency:

* Dirty reads
  + User 1 modifies a row. User 2 reads the same row before User 1 commits. User 1 performs a rollback. User 2 has read a row that has never really existed in the database. User 2 may base decisions on false data.
* Non-repeatable reads
  + User 1 reads a row but does not commit. User 2 modifies or deletes the same row and then commits. User 1 rereads the row and finds it has changed (or has been deleted).
* Phantom reads.
  + User 1 uses a search condition to read a set of rows but does not commit. User 2 inserts one or more rows that satisfy this search condition, then commits. User 1 rereads the rows using the search condition and discovers rows that were not present before.

Isolation levels represent the database system’s ability to prevent these behaviors. The American National Standards Institute (ANSI) defines four isolation levels:

* Read uncommitted (0);
* Read committed (1);
* Repeatable read (2);
* Serializable (3). (DataDirect, 2020)

The following table shows what data consistency behaviors can occur at each isolation level. (DataDirect, 2020)



Since the system will only be used by a handful of workers at any given time, a decision has been made that the database will work on the Serializable isolation level.

### Creating SQL tables

SQL Tables are created with the **CREATE TABLE** statement, followed by all the columns of the defined table. The code below shows an example of a table being created for this project.

CREATE TABLE Products(

Id INT PRIMARY KEY IDENTITY,

Name VARCHAR(100),

Barcode INT UNIQUE,

Description VARCHAR(255),

Price FLOAT,

MaxStock INT,

MinStock INT CHECK(MinStock > 1),

Quantity INT

);

#### SQL Keys

An SQL key is either a single column (or attribute) or a group of columns that can uniquely identify rows (or tuples) in a table. This project uses several different types of keys:

1. Candidate keys - A candidate key is a single key or a group of multiple keys that uniquely identify rows in a table. In MSSQL a candidate key is defined using the **UNIQUE** keyword.
2. Primary keys – A primary key is the candidate key selected by the database administrator to uniquely identify tuples in a table. In MSSQL a candidate key is defined using the **PRIMARY KEY** keyword.
3. Foreign keys – A foreign key is an attribute that is a primary key in its parent table but is included as an attribute in another host table. In MSSQL a foreign key is defined in the following way:

**FOREIGN KEY (column\_name) REFERENCES PARENT\_TABLE(pk\_parent\_table)**

## Programming the solution (Back-end)

### Project structure

For the development of this project, the three-layered architecture was followed, meaning that there is a presentation layer for visualizing the data, a business layer for processing commands, and a persistence layer for storing information. This project also uses the Data Access Object (DAO) pattern, the Singleton pattern, and the Observer pattern. The package structure of this project is as follows:

|  |
| --- |
| src ├───controllers ├───db  │ └───interfaces ├───exceptions ├───interfaces  ├───models  │ └───enums  ├───test  └───ui  └───img |

* ***models –*** contains the models of the project, which are a direct representation of database tables and are used both to parse database responses from queries into classes and to use the fields of the classes to create entries in the database (Persistence layer)
  + ***models.enums*** – contains enumerations that make certain columns from the tables more developer-readable, meaning that, for example – instead of treating an order status as an integer, we can treat it as a named enumeral. (DRAFT, ALLOCATED, etc.)
* ***db –*** contains classes that execute queries, communicating with the database. An especially important class from this package is ***DBConnection***, which establishes a connection with the server. (Persistence layer)
  + ***db.interfaces –*** contains interfaces that are to be implemented by the classes written in the ***db*** package.
* ***controllers –*** contains classes that communicate with their respective ***db*** classes and/or with other controller classes. Their purpose is to execute commands sent by the user interface and to return the desired information (Business layer)
* ***ui –*** contains the user interface, which is to be used by the end-user to create responses and to receiver responses from the control layer. (Presentation layer)
  + ***img*** – contain images used by the user interfaces
* ***interfaces –*** in this project, this package only contains 2 interfaces – **Observer** and **Observable**, which are used to implement the Observer pattern, which will be mentioned more in-depth later.
* ***test –*** contains the unit tests for the control layer of the application.
* ***exceptions*** – contains custom exceptions used in various parts of the project.

### Models

Below is an implementation of a database table being transformed to a model (examples show both a class and an enumeration).

public class Product {  
 private int id;  
 private String name;  
 private int barcode;  
 private String description;  
 private float price;  
 private int maxStock;  
 private int minStock;  
 private int quantity;  
  
 // Constructors

// Getters and setters  
}

  public enum OrderStatus {

DRAFT,

ALLOCATED,

PACKED,

SHIPPED,

DELIVERED,

NOT\_ACCEPTED,

CANCELLED;

public int getValue() {

return ordinal() + 1;

}

}

The foreign keys are specified in the code by keeping a field inside the class of the same type as the parent table from the database. An example of this is shown below.

public class Customer {  
 private int id;  
 private String name;  
 private String phoneNumber;  
 private String email;  
 **private Address;**  
  
 // Constructors  
 // Getters and setters  
}

### DBConnection.java

***DBConnection.java*** is responsible for starting and maintaining a connection to the database and acts as a source point for sending requests and receiving responses. The way this connection is done in this project is via the **Java Database Connectivity** (JDBC) application program interface (API). The way JDBC works is by providing it with a hostname, a database name, and credentials for authentication. Below is the realization of the ***DBConnection*** class. One important line in this

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;

public class DBConnection {

private static final String SERVER\_NAME = "hildur.ucn.dk";

private static final int PORT\_NUMBER = 1433;

private static final String DATABASE\_NAME = "dmaj0920\_1086341";

private static final String USERNAME = "dmaj0920\_1086341";

private static final String PASSWORD = "\*\*\*\*\*\*\*\*";

private static Connection = null;

private static DBConnection dbConnection = null;

private DBConnection() throws SQLException {

String urlString = String.format("jdbc:sqlserver://%s:%d;databaseName=%s", SERVER\_NAME, PORT\_NUMBER, DATABASE\_NAME);

try {

DriverManager.registerDriver(new com.microsoft.sqlserver.jdbc.SQLServerDriver());

connection = DriverManager.getConnection(urlString, USERNAME, PASSWORD);

} catch (SQLException e) {

System.out.println("Cannot access database. Exception: " + e.getMessage());

}

connection.setAutoCommit(false);

}

public Connection getConnection() {

return connection;

}

public static DBConnection getInstance() throws SQLException {

if (dbConnection == null) {

dbConnection = new DBConnection();

}

return dbConnection;

}

public static void closeConnection() {

try {

connection.close();

dbConnection = null;

} catch (Exception e) {

System.out.println("Error trying to close the database " + e.getMessage());

}

}

public void startTransaction() throws SQLException {

connection.setTransactionIsolation(Connection.TRANSACTION\_SERIALIZABLE);

connection.setAutoCommit(false);

}

public void commitTransaction() throws SQLException {

connection.commit();

connection.setAutoCommit(true);

}

public void rollbackTransaction() throws SQLException {

connection.rollback();

connection.setAutoCommit(true);

}

}

#### The singleton pattern

The singleton pattern is a software design pattern that restricts the instantiation of a class to one "single" instance. This is useful when exactly one object is needed to coordinate actions across the system. The term comes from the mathematical concept of a singleton. When deciding whether a class should follow the singleton pattern, there are three conditions that the class needs to satisfy:

1. It controls concurrent access to a shared resource;
2. Access to the resource will be requested from multiple, disparate parts of the system;
3. There can be only one object.

***DBConnection.java*** fits these conditions, meaning that it was implemented following the singleton pattern. Furthermore, establishing a JDBC connection with a DBMS can be very slow. If your application requires database connections that are repeatedly opened and closed, this can become a significant performance issue. (Oracle, n.d.)

### Implementing DAO classes

The Data Access Object (DAO) pattern is a structural pattern that allows us to isolate the application/business layer from the persistence layer (usually a relational database, but it could be any other persistence mechanism) using an abstract API. (baeldung, 2020)

The functionality of this API is to hide from the application all the complexities involved in performing CRUD operations in the underlying storage mechanism. This allows both layers to evolve separately without knowing anything about each other. (baeldung, 2020)

The code below is displaying an example of a DAO-pattern interface. ***DataAccessException*** is an exception class used to notify that an error has occurred while trying to access the data via JDBC. We specify with the **throws** keyword that the exception is not handled by the method itself, and that another part of the code should handle it.

public interface ProductDBIF {

int createProduct(Product p) throws DataAccessException;

Product findByBarcode(int barcode) throws DataAccessException;

}

The way DAO classes are implemented in Java is by a class called ***PreparedStatement***. A Java JDBC **PreparedStatement** is a special kind of Java JDBC Statement object with some useful additional features, such as:

* Easy parameter insertion;
* Reusability;
* Increased performance;
* Easier batch updates. (Jenkov, 2019)

The code below shows the implementation of a DAO interface using prepared statements and transactions.

public class ProductDB implements ProductDBIF {

private static final String CREATE\_PRODUCT\_Q = "INSERT INTO Products (Name, Barcode, Description, Price, MaxStock, MinStock, Quantity) VALUES(?, ?, ?, ?, ?, ?, ?)";

private static final String FIND\_BY\_BARCODE\_Q = "SELECT Id, Name, Barcode, Description, Price, MaxStock, MinStock, Quantity FROM Products WHERE Barcode = ?";

private final PreparedStatement createProductPS;

private final PreparedStatement findByBarcodePS;

public ProductDB() throws DataAccessException {

try {

this.createProductPS = DBConnection.getInstance().getConnection().prepareStatement(CREATE\_PRODUCT\_Q, Statement.RETURN\_GENERATED\_KEYS);

this.findByBarcodePS = DBConnection.getInstance().getConnection().prepareStatement(FIND\_BY\_BARCODE\_Q);

} catch (SQLException e) {

throw new DataAccessException("Could not prepare statement", e);

}

}

@Override

public int createProduct(Product p) throws DataAccessException {

try {

DBConnection.getInstance().startTransaction();

this.createProductPS.setString(1, p.getName());

this.createProductPS.setInt(2, p.getBarcode());

this.createProductPS.setString(3, p.getDescription());

this.createProductPS.setDouble(4, p.getPrice());

this.createProductPS.setInt(5, p.getMaxStock());

this.createProductPS.setInt(6, p.getMinStock());

this.createProductPS.setInt(7, p.getQuantity());

int rows = this.createProductPS.executeUpdate();

DBConnection.getInstance().commitTransaction();

int productId;

ResultSet generatedKeys = this.createProductPS.getGeneratedKeys();

if (generatedKeys.next()) {

productId = generatedKeys.getInt(1);

} else {

throw new SQLException("Could not insert product, no ID obtained.");

}

p.setId(productId);

return rows;

} catch (SQLException e) {

try {

DBConnection.getInstance().rollbackTransaction();

} catch (SQLException e1) {

throw new DataAccessException("Could not rollback transaction", e1);

}

throw new DataAccessException("Could not insert data", e);

}

}

@Override

public Product findByBarcode(int barcode) throws DataAccessException {

try {

DBConnection.getInstance().startTransaction();

this.findByBarcodePS.setInt(1, barcode);

ResultSet rs = this.findByBarcodePS.executeQuery();

DBConnection.getInstance().commitTransaction();

Product p = null;

if(rs.next()) {

p = buildObject(rs);

}

return p;

} catch (SQLException e) {

try {

DBConnection.getInstance().rollbackTransaction();

} catch (SQLException e1) {

throw new DataAccessException("Could not rollback transaction", e1);

}

throw new DataAccessException("Could not fetch data", e);

}

}

private List<Product> buildObjects(ResultSet rs) throws DataAccessException {

List<Product> list = new ArrayList<>();

while(true) {

try {

if (!rs.next()) break;

list.add(buildObject(rs));

} catch (SQLException e) {

throw new DataAccessException("Could not parse data", e);

}

}

return list;

}

private Product buildObject(ResultSet rs) throws DataAccessException {

try {

return new Product(rs.getInt("Id"), rs.getString("Name"), rs.getInt("Barcode"), rs.getString("Description"), rs.getFloat("Price"), rs.getInt("MaxStock"), rs.getInt("MinStock"), rs.getInt("Quantity"));

} catch (SQLException e) {

throw new DataAccessException("Could not parse data", e);

}

}

}

### Implementing controllers

The controllers work as a man-in-the-middle between the user interface and the database. Controllers can also communicate with themselves. This is used to maintain the “one-controller-per-use-case” pattern and to avoid the scenario where a single piece of UI communicates with more than one controller, thus reducing coupling in the system. Below is a sample implementation of a controller, which communicates not only with the database, but with another controller as well.

public class CustomerController {

private final CustomerDBIF customerDB;

private final AddressController;

public CustomerController() throws DataAccessException {

this.customerDB = new CustomerDB();

this.addressController = new AddressController();

}

public Customer findByPhoneNo(String phoneNo) throws DataAccessException {

return this.customerDB.findByPhoneNo(phoneNo);

}

public int createCustomer(Customer c) throws DataAccessException {

return this.customerDB.createCustomer(c);

}

public int updateCustomer(Customer c) throws DataAccessException {

return this.customerDB.updateCustomer(c);

}

public int deleteCustomer(Customer c) throws DataAccessException {

return this.customerDB.deleteCustomer(c);

}

public Address findAddressByData(Address a) throws DataAccessException {

return this.addressController.findByData(a);

}

public Address findAddressById(int id) throws DataAccessException {

return this.addressController.findById(id);

}

public void createAddress(Address a) throws DataAccessException {

this.addressController.createAddress(a);

}

}

## Unit testing the back-end

Unit testing is a type of software testing where individual units or components of the software are tested. The purpose is to validate that each unit of the software code performs as expected. Unit Testing is done during the development (coding phase) of an application by the developers. Unit Tests isolate a section of code and verify its correctness. A unit may be an individual function, method, procedure, module, or object. (Guru99, n.d.)

**JUnit** is a unit testing framework for the Java programming language. JUnit has been important in the development of test-driven development and is one of a family of unit testing frameworks which is collectively known as xUnit that originated with SUnit. (Wikipedia, n.d.) A JUnit test fixture is a Java object. Test methods must be annotated by the **@Test** annotation. If the situation requires it, it is also possible to define a method to execute before (or after) each (or all) of the test methods with the **@BeforeEach** (or **@AfterEach**) and **@BeforeAll** (or **@AfterAll**) annotations. (Bechtold, et al., n.d.)

The code below shows an example of unit tests for the back-end we developed.

public class ProductControllerTests {

private ProductController productController;

@Before

public void setUp() {

try {

this.productController = new ProductController();

} catch (DataAccessException e) {

e.printStackTrace();

}

}

@Test

public void testFindProductByBarcode() {

Product p = null;

try {

p = this.productController.findByBarcode(93766442);

} catch (DataAccessException e) {

e.printStackTrace();

}

assertEquals("Dried Apple", p.getName());

assertEquals(16, p.getId());

}

@Test

public void testFindProductByInvalidBarcode() {

Product p = null;

try {

p = this.productController.findByBarcode(0);

} catch (DataAccessException e) {

e.printStackTrace();

}

assertNull(p);

}

@Test

public void testCreateProduct() {

try {

Product p = new Product("test", 12345678, "test description", 7.99f, 5, 50, 50);

int rows = this.productController.addProduct(p);

assertEquals(1, rows);

p = this.productController.findByBarcode(12345678);

assertEquals("test", p.getName());

} catch (DataAccessException e) {

e.printStackTrace();

}

}

@Test

public void testUpdateProduct() {

try {

Product p = this.productController.findByBarcode(12345678);

p.setName("updated test");

int rows = this.productController.updateProduct(p);

assertEquals(1, rows);

p = this.productController.findByBarcode(12345678);

assertEquals("updated test", p.getName());

} catch (DataAccessException e) {

e.printStackTrace();

}

}

@AfterClass

public static void cleanUpWhenFinish() {

try {

ProductController pc = new ProductController();

Product p = pc.findByBarcode(12345678);

int numDeleted = pc.deleteProduct(p.getId());

// Assert

assertEquals("One row deleted", 1, numDeleted);

} catch(Exception ex) {

System.out.println("Error: " + ex.getMessage());

} finally {

DBConnection.closeConnection();

}

}

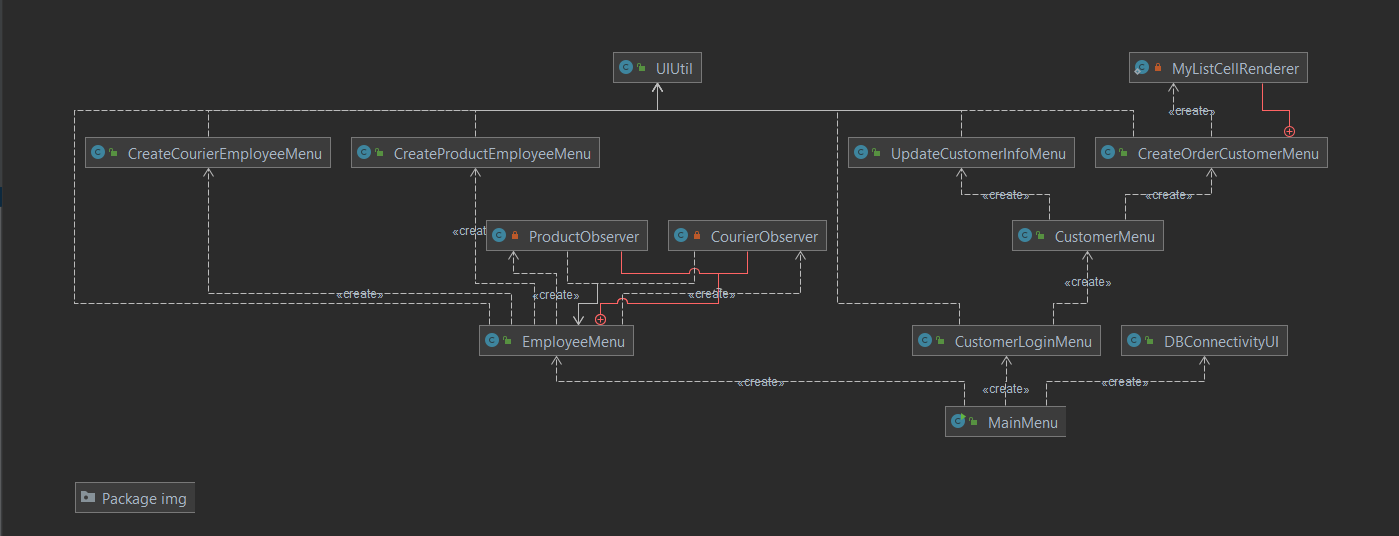
}

## Programming the solution (GUI)

The framework for GUI development by requirement is Swing. Swing is a GUI widget toolkit for Java. Swing was developed to provide a more sophisticated set of GUI components than the earlier Abstract Window Toolkit (AWT). Swing provides a look and feel that emulates the look and feel of several platforms and supports a pluggable look and feel that allows applications to have a look and feel unrelated to the underlying platform. It has more powerful and flexible components than AWT. In addition to familiar components such as buttons, checkboxes, and labels, Swing provides several advanced components such as tabbed panels, scroll panes, trees, tables, and lists.

### UI Layout

The image below displays the layout and interactions of all UI classes used in this project.



### DBConnectivityUI.java

One of the project’s requirements was to find a way to implement some sort of multi-threading in the solution of the project. One of the simplest ways to execute this is with a parallel database connectivity check. The way we check if the database is by performing a simple SQL query to the database once every number of seconds. The following two code snippets show the implementation of the above-mentioned ideas.

// Inside DBConnection.java

public boolean isDbConnected() {

try (PreparedStatement ps = connection.prepareStatement("SELECT 1")) {

return ps.execute();

} catch (SQLException e) {

return false;

}

}

public class DBConnectivityUI {  
 private JFrame frmDatabaseStatus;

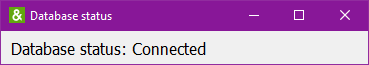
public DBConnectivityUI() {  
 initialize();  
 }  
  
 private void initialize() {

JLabel lblStatus = new JLabel();

// Labels and panel initialization  
  
class DBConnectivityWorker implements Runnable {

**@Override**

**public void run() {  
 while(true) {  
 boolean success = false;  
 try {  
 success = DBConnection.getInstance().isDbConnected();  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 lblStatus.setText("Database status: " + (success ? "Connected" : "Error connecting."));  
 try {  
 Thread.sleep(500);  
 } catch (InterruptedException e) {  
 break;  
 }  
 }  
 }  
 }  
  
 Thread t1 = new Thread(new DBConnectivityWorker());  
 t1.start();**  
 }  
  
 public void showWindow() {  
 EventQueue.invokeLater(() -> {  
 try {  
 frmDatabaseStatus.setVisible(true);  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 });  
 }  
}



### Generating table content

The data of some of the classes (for example – products) is visualized using a JTable. JTables accept an Object matrix for the rows and an Object array for the column names. The code below shows how the data is extracted for both the rows and the columns. Rows are converted into a string array, and the column names are extracted from a given class using reflection, which gets all the fields of the class and gets their names.

// EmployeeMenu.java

List<Product> products = null;

try {

products = this.productController.getProducts();

} catch (DataAccessException e) {

UIUtil.displayDBErrorMsg(e.getMessage());

}

assert products != null;

String[][] productRows = new String[products.size()][];

for (int i = 0; i < products.size(); i++) {

productRows[i] = products.get(i).toStringArray();

}

String[] productColumns = UIUtil.getClassFields(Product.class);

JTable tableProducts = new JTable(productRows, productColumns);

// Product.java  
public String[] toStringArray() {  
 return new String[]{String.valueOf(id), name, String.valueOf(barcode), ... };  
}

// UIUtil.java

public static String[] getClassFields(Class c) {

Field[] fields = c.getDeclaredFields();

String[] names = new String[fields.length];

for (int i = 0; i < fields.length; i++) {

names[i] = fields[i].getName();

}

return names;

}

### Updating data from the table

The way updating works in our system is that the user can double click a cell to edit its content. Upon pressing the Enter key, a query is sent to the database with the new data. However, some columns should not be edited (for example – the ID of a product). The way to prevent a person from editing a specific cell is by overriding the default table model of the specified table and specifying the conditions in which the cell is editable.

// EmployeeMenu.java

tableProducts = new JTable(new DefaultTableModel(productRows, productColumns) {  
 @Override

public boolean isCellEditable(int row, int column) {  
 return column != 0;  
 }  
});

We can run the query when the cell is updated by attaching a table model listener to the table:

tableProducts.getModel().addTableModelListener(e -> {  
 if (e.getType() == TableModelEvent.UPDATE) {  
 // Update product data  
 }  
});

### The Observer pattern

**The Observer pattern** is a behavioral design pattern. It specifies the communication between objects: observable and observers. **An observable is an object which notifies observers about the changes in its state.**

In this application, a place where this pattern is used is, for example, Product creation. The creation itself happens in a different dialog from the employee menu, and we want to update the table containing all products once the product has been inserted into the database. Creating a new query would be too slow, so an alternative is to use the already generated table and append a new row to it once the product is created.

However, if we have other analogous situations (for example, creating a courier), we cannot make the dialog itself implement the Observer interface. A solution to this issue is to create inner classes inside the dialog, which themselves implement the Observer interface for their respective classes.

public class EmployeeMenu {

private class ProductObserver implements Observer<Product> {

public void notifyUpdate(Product item) {

((DefaultTableModel) tableProducts.getModel()).addRow(item.toStringArray());

tableProducts.revalidate();

}

public ProductObserver getThis() {

return ProductObserver.this;

}

}

private class CourierObserver implements Observer<Courier> {

public void notifyUpdate(Courier item) {

((DefaultTableModel) tableCouriers.getModel()).addRow(item.toStringArray());

tableCouriers.revalidate();

}

public CourierObserver getThis() {

return CourierObserver.this;

}

}

.

.

.

JButton btnNewProduct = new JButton();

@Override

public void mouseClicked(MouseEvent e) {

**CreateProductEmployeeMenu cpem = new CreateProductEmployeeMenu();**

**cpem.addObserver(new ProductObserver().getThis());**

**cpem.showWindow();**

}

});

}

public class CreateProductEmployeeMenu implements Observable<Product> {  
 private JDialog dialogCreateProduct;  
  
 **private Observer<Product> observer;**  
  
 public CreateProductEmployeeMenu() {  
 initialize();  
 }  
  
 private void initialize() {  
 JButton btnCreateProduct = new JButton("Create Product");  
 btnCreateProduct.addMouseListener(new MouseAdapter() {  
 @Override

public void mouseClicked(MouseEvent e) {

.

.

.

// Logic for creating a new product  
 **observer.notifyUpdate(p);**  
 UIUtil.displayMessage("Product with id " + p.getId() + " created successfully.", "Success", JOptionPane.INFORMATION\_MESSAGE);  
 });  
 }  
 **@Override**

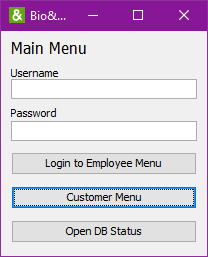
**public void addObserver(Observer<Product> observer) {  
 this.observer = observer;  
 }**  
}Graphical user interface, application

Description automatically generated

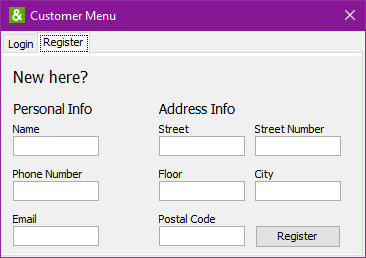
### User Manual (Other GUI panels)

#### For customers

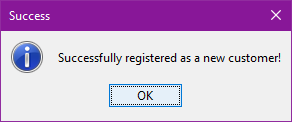
The customer menu can be accessed by pressing the “Customer Menu” button in the main menu.



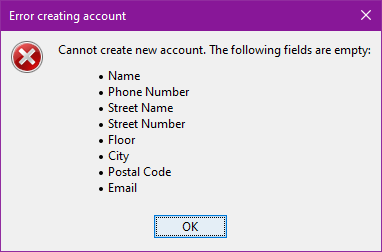
##### Creating a new account



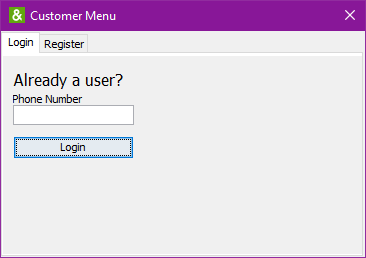
A new account can be created by clicking the “Register” tab on the top. Then you can input all your data and, once you are done, you can press the “Register” button. If everything is okay, you will be met with the following message:



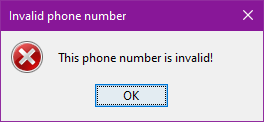
If any input fields are empty, the program will produce the following message:



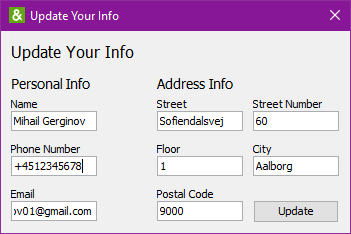
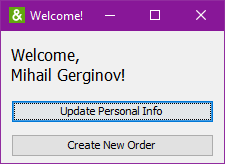
##### Logging into the system



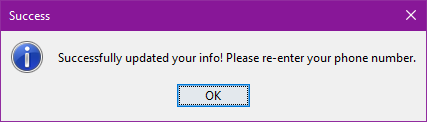
You can log into the system by clicking the “Login” tab on the top. Then you can input your phone number and, once you are done, you can press the “Register” button. If the phone number is valid, you will be redirected to the customer menu. Otherwise, the following error message will appear:



##### Updating your information

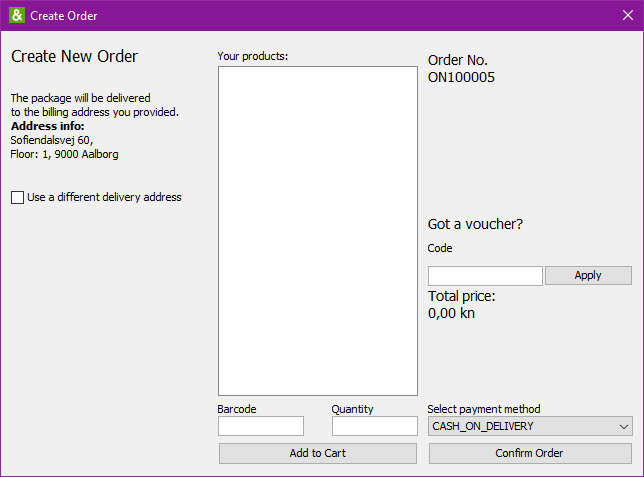


Once logged in, you can update (if needed) your personal info. By clicking the “Update Personal Menu” button, you will be redirected to a new window, where you can edit your data. Once you are satisfied, press the “Update” button. If everything is okay with the fields, the following message will appear, after which you will be logged out and you must log in with your new phone number (if you have updated it):

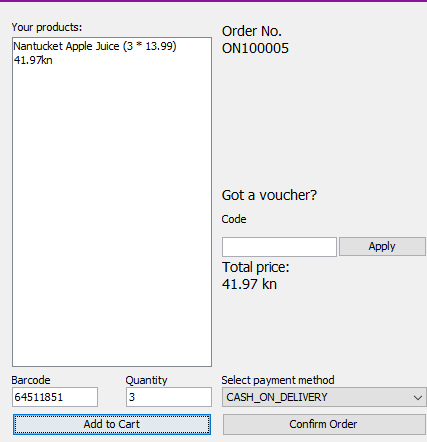


Just like the registration, an error message will appear if you have missed to fill in any fields.

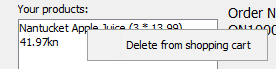
##### Creating a new order

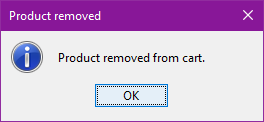


You can create a new order by pressing the “Create New Order” button. Upon clicking it, you will be presented with a shopping cart-like interface. You can insert products by typing their barcode and how much of them you want in the fields below the “Your Products” list and pressing the “Add to Cart” button.

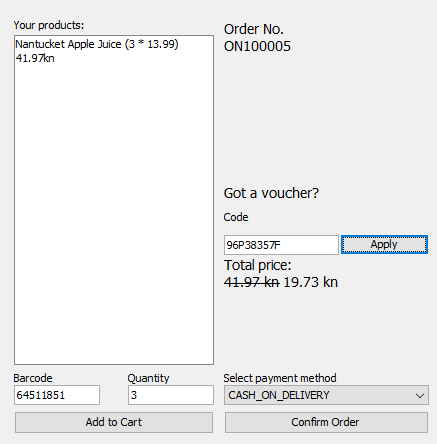


You can delete a product from the shopping cart by right-clicking on the product and pressing “Delete from shopping cart…”.

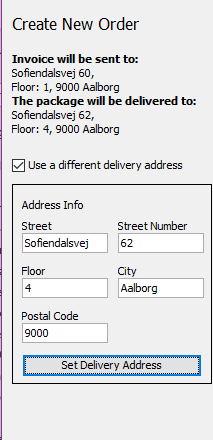




You can apply a voucher to your current order. If the voucher is valid (not expired and the code is valid), a confirmation message will appear, and the total price will be discounted.



If you want the delivery address to be different from the invoice address (the one you entered upon registration), you can check the box “Use a different delivery address”. There you can input all the information about the delivery address of choice.

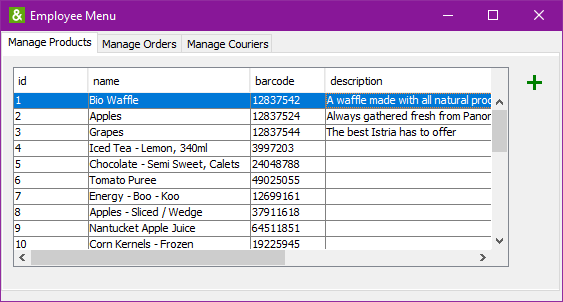


#### For employees

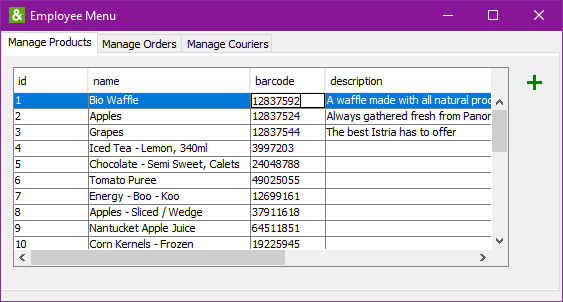
The employees’ menu can be accessed by entering the correct username and password (here they are ***admin*** and ***admin***, respectively), and pressing “Login to Employee Menu”.

##### Product/courier management

We will skip courier management since it is analogous to product management.



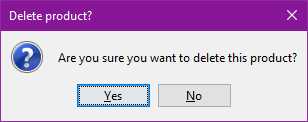
Here you can view all products. You can edit a product by double-clicking its cell.



Upon pressing the Enter key, the product will be updated, and the new information will be stored in the database.

You can delete a product by right-clicking it, and selecting “Delete product…”.





# Conclusion

## Group process

Group process refers to how an organization’s members work together to get something done. Our group process was to make the final project as a group. Our group members were Marko, Antonio, Lucas, Teo, and Mihail. The first thing we did was to establish the collaboration. We knew that if we wanted to work as a group, we needed all five of us and we also needed to do our best to succeed. A big part of that was to find our communication. We sat down in a room and discussed how we can communicate and contribute to our common goal. Of course, now that everything is online, we were stopped from seeing each other in person, but we didn't give up and we found the solution. Once every second day we would meet on Discord and discuss the project, what we have done, and what needs to be done. The second thing we went through was coordination. We established the goal and how to balance between learning outcome and result. The next thing we did was Information. We knew that we needed to make the project, but to start with the project, we needed to gather the necessary information about the company and how they want the application to look. We found out that if we wanted to get the information we need, we also needed to contact the firm. The last thing we did in the group process was establishing the rules. We have established some rules that all members needed to follow.

## Future expansion of the project

For the future of the project we envision the following milestones and goals:

* Business
  + Conduct more interviews with the company for feedback, etc
* System Development
* Programming

# References

baeldung, 2020. *The DAO Pattern in Java.* [Online]   
Available at: https://www.baeldung.com/java-dao-pattern

Bardicchia, M., 2020. *Digital CRM: Strategies and Emerging Trends: Building Customer Relationship in the Digital Era.* s.l.:s.n.

Beal, B., 2020. *CRM vs ERP: What’s the Difference?.* [Online]   
Available at: https://www.netsuite.com/portal/resource/articles/erp/erp-vs-crm.shtml

Bechtold, S., Brannen, S., Link, J. & etc., n.d. *JUnit Documentation.* [Online]   
Available at: https://junit.org/junit5/docs/current/user-guide/#writing-tests

Chen, J., 2021. *Interest Rate Risk.* [Online]   
Available at: https://www.investopedia.com/terms/i/interestraterisk.asp

DataDirect, P., 2020. *Isolation Levels.* [Online]   
Available at: https://media.datadirect.com/download/docs/odbc/allodbc/index.html#page/odbc/isolation-levels.html

Enfroy, A., n.d. *Affiliate Marketing in 2021: What It Is and How You Can Get Started.* [Online]   
Available at: https://www.bigcommerce.com/blog/affiliate-marketing

Gantt, H., 1974. Work, Wages and Profit. In: *Engineering Magazine.* Easton, Pennsylvania: Hive Publishing Company.

Guru99, n.d. *Guru99.* [Online]   
Available at: https://www.guru99.com/unit-testing-guide.html

Harper, D. R., 2020. *Understanding Liquidity Risk.* [Online]   
Available at: https://www.investopedia.com/articles/trading/11/understanding-liquidity-risk.asp

Jenkov, J., 2019. *Jenkov.* [Online]   
Available at: http://tutorials.jenkov.com/jdbc/preparedstatement.html

Klein, R., 1999. Scheduling of Resource-Constrained Projects. In: *Operations Research/Computer Science Interfaces Series.* s.l.:Springer US.

Kondylis, D. & Mertz, A. C. H. R., 2021. *E-business ERP and CRM.* Aalborg(Nordjylland): UCN.

Kondylis, D. & Mertz, A. C. H. R., 2021. *Supply Chain Management.* Aalborg(Nordjylland): UCN.

Kumar, P. P., 2005. Effective Use of Gantt Chart for Managing Large Scale Projects. In: *Cost Engineering.* Morgantown, WV: American Association of Cost Engineers, pp. 13-21.

Mindtools, 2016. *Cost-Benefit Analysis: Deciding, Quantitatively, Whether to Go Ahead.* [Online]   
Available at: https://www.mindtools.com/pages/article/newTED\_08.htm

Morris, P. W., 1994. The Management of Projects. In: s.l.:s.n.

Oberlo, n.d. *What is eCommerce?.* [Online]   
Available at: https://www.oberlo.com/ecommerce-wiki/ecommerce

Oracle, n.d. *Performance Tuning Your JDBC Application.* [Online]   
Available at: https://docs.oracle.com/cd/E13222\_01/wls/docs81/jdbc/performance.html

Porter, M. E., 1985. *Competitive Advantage.* New York: The Free Press.

Richman, L., 2002. In: *Project Management Step-by-step.* s.l.:s.n., p. 117.

Rowe, M. D. M. M., 1994. *Strategic Management: a methodological approach.* s.l.:Addison-Wesley.

Schooley, S., 2019. *SWOT Analysis: What It Is and When to Use It.* [Online]   
Available at: https://www.businessnewsdaily.com/4245-swot-analysis.html

Selig, G. J., 2008. In: *Implementing IT Governance: A Practical Guide to Global Best Practices in IT Management.* s.l.:Van Haren Publishing.

Wikipedia, n.d. *Unified Process.* [Online]   
Available at: https://en.wikipedia.org/wiki/Unified\_Process

Wikipedia, n.d. *Wikipedia.* [Online]   
Available at: https://en.wikipedia.org/wiki/JUnit