*UCN, University College of Northern Denmark*
*IT-Programme*
*AP Degree in Computer Science*
*CSC-CSD-S212*

# Second Semester Project

*Table reservation system*

Martin Glogolovský,  Petra Pastierová, Matej Rolko, Viktor Tindula
25-05-2022

UCN, University College of Northern Denmark

# AP Degree in Computer Science

*Class: CSC-CSD-S212*

Participants:

Matej Rolko

Viktor Tindula

Martin Glogolovský

Petra Pastierová

Supervisors:

Gianna Belle

Lars Nysom

Jakob Farian Krarup

Normal pages/characters: 20,53Pages/49 288 characters

Repository path: https://github.com/PoustniGaara/SecondSemseterProject (72 commits)

## Table of Contents

# Introduction

In this report, we document the process of implementing and developing a table reservation system for a firm called Café Peace, a popular restaurant in the centre of Aalborg. The café offers a wide range of meals, from breakfast and brunch to lunch and dinner. Additionally, people come here for coffee and dessert. During the weekends and holidays, the restaurant is remarkably busy especially around lunch and dinnertime. The café serves customers with reservations, but also takes walk-in customers if there is a table available. During rush hours, the employees rely on table reservations made by the customers in advance, helping them to prepare the restaurant and figure out the number of walk-in parties they can take. Our task is to create a system that unifies and simplifies this reservation process.

## Problem area

To reserve a table in the restaurant, the customers can make a booking either in person, on the website or by phone. The staff is responsible for managing and grouping all table reservations, using a paper spreadsheet, which causes chaos when multiple employees participate. Even though the café offers table reservations through their website, problems occur regularly, since an employee is needed to manually confirm all reservations via email. Because of this, the reservations are made through phone calls. Mistakes are made often, resulting in several wrong or unconfirmed reservations, causing difficulty for the staff and frustration within the customers.

## Methodology

This project is to be carried out using the *Unified Process,* an iterative and incremental software development process framework. The plan is to be constructed in *ProjectLibre* using *Gantt chart*, and for work division and progress tracking, *Trello* is to be used. The diagrams are to be created using Unified Modeling Language in *UMLet,* in order to visualize the design of the system. The system is to be programmed in *Java* programming language with the code written in *Eclipse* IDE. The GUI is to be made in Java UX library *Swing*. For version control, *Github* using *Git* is to be used. Data is to be stored in *Microsoft SQL* relation database. To manage the database, *Microsoft SQL Server Management Studio* will be used. All project work is to be documented in this report.

# 1. Feasibility study

## 1.1. Vison/Mission

The fundamental purpose of Café Peace is to provide its customers with the best service and experience while dining in a cosy, yet luxurious establishment, what they call "*an atmospheric oasis in the heart of Aalborg*." It is a family business that puts an emphasis on quality ingredients, community, and dining experience. (Café Peace, n.d.)

The café is trying to achieve a friendly, family-like environment. According to their website, the team works together with a collective goal to improve customer experience with collaboration and mutual success. They also state that they offer flexibility and consideration, in hope that it creates a positive work culture with committed employees and high spirit. The café's motto is "*Little things mean a lot,*" on which they put a lot of attention when modelling the little aspects of the experience they provide.

The vision of the company centers around providing a high-quality dining experience, while expanding their reach and customer base and enhancing the services they provide. One example of enhancing services is the optimalization of their table reservation system, the focus of this project.

## 1.2. Organisational Structure

Café Peace ApS is a limited liability company founded in 2009 by Anne Marie Køning as a family-owned restaurant (proof.dk, n.d.). It is a subsidiary of KØNIG HOLDING AALBORG ApS, which is also owned by Anne Marie Køning (proff.dk, n.d.). The café has a functional structure, it is structured vertically in departments, with managers supervising their staff (Gray, 2003). The café has a bar and a kitchen department. Each is managed by a head employee, the main waitress, and head chef. Anne Marie Koning, the owner, is at the top, with her children also being involved in managing the establishment. Because of the scale of the company,

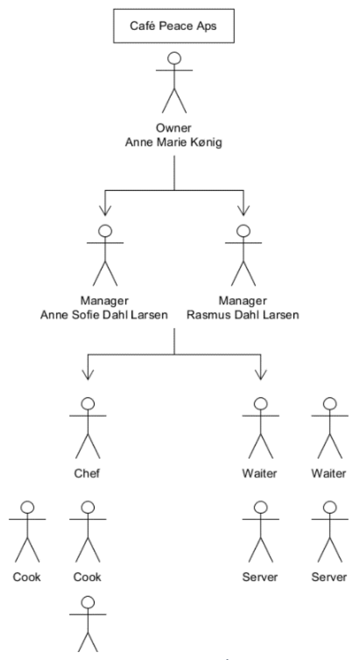


*Figure 1: Organisational structure*

regular employees communicate with whoever is available, but the main decision are always made by the owner. Other managing employees are the chefs in the kitchen, who manage cooks and dishwashers during their shifts. In terms of Mintzberg's generic description, the café has a simple structure (Brooks, 2009). Strategic apex is the owner, middle management is made of managers and main waitress and head chef, while the operating core is most of the employees, such as waiters, servers, cooks, and dishwashers.

The workforce is made of fully employed and part-time workers. Most of the staff is of Danish nationality, while the lower-level employees in the kitchen are part-time international student workers.

### 1.3. Culture and management

Organisational culture is the distribution and appointment of the command and duties within the organization. Every organization has certain values and follows some policies and guidelines which differentiate it from others. The principles and beliefs of an organization form its culture. This is learned by recruits through socialisation, training, and interventions from management. According to Charles Handy, a well-known philosopher from the 20th century who specialized in organisational structure, there are four types of culture which the organisations follow (Management Study Guide, n.d.). Café Peace, like many small businesses, has a power culture. This means it focuses on key decision makers who further delegate responsibilities to the other employees. Subordinates have no option but to follow their superior's instructions.

### 1.4. Stakeholder analysis

Stakeholder Analysis is the first step in Stakeholder Management, an important process that helps identify interested parties and determine how to best involve and communicate with them throughout the project (Mind Tools, n.d.). Stakeholders can be both external and internal. The table below shows an overview of the different stakeholders we have identified and their goals, behaviour, ideas, and salience. Stakeholder salience is determined by three variables – power, urgency, and legitimacy. Power is the ability to influence the project and its objectives. Legitimacy means authority and level of involvement in the project. Urgency is the degree to which stakeholder requirements call for immediate attention, depending on time-sensitivity and criticality. Stakeholders that rank highest in all three categories are the most important players in the project's success. This is shown in the Venn diagram directly under the table.

| Stakeholder | Goals | Expected behaviour | Ideas for action | Stakeholder salience |
|---|---|---|---|---|
| Management | Company's success | Embracing and propagating the change | Mandatory employee training in performing system related tasks | Definitive stakeholder |
| Workers | Having a more efficient way to do their job | Positive attitude towards new IT system | Participation in IT training | Dangerous stakeholder |
| Customers | Obtaining goods and services | none | none | Demanding stakeholder |

*Table 1: Stakeholders*

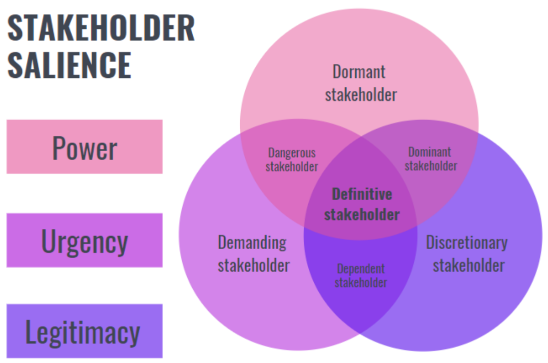


*Figure 2: Stakeholder salience*

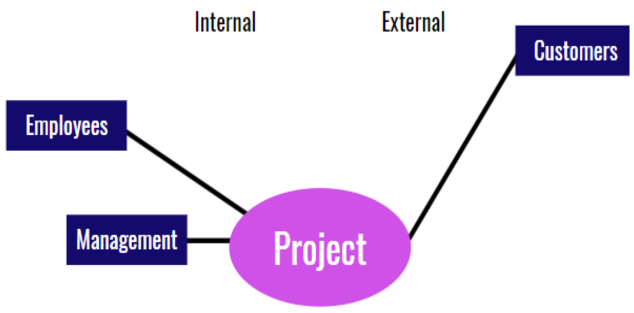


*Figure 3: Stakeholder Map*

Stakeholder mapping is a process of visualizing interested parties of a project on a map, based on the impact they impose. The stakeholders are split into two categories – external and internal. The closer the stakeholder is to the project on the map, the bigger the influence they have (Danda, 2020). This is useful when figuring out who and how to involve in the process of creating and launching a project.

As our project is relatively small-scale, we have identified three stakeholders – the management, employees, and customers. The management represents a definitive/core stakeholder, as they combine high power, high urgency, and high legitimacy. It is crucial to manage them closely. The employees are a dangerous stakeholder, as they have high power and urgency, but low legitimacy, meaning they can create trouble for the project. It is important to manage them cautiously. The customers have low power and legitimacy, but high urgency. They require attention to a certain degree, in a sense of having their needs met, but do not need to be involved in the project like the other stakeholders.

## 1.5. Porter's five forces

Porter's five forces analysis, named after Harvard Business school professor Michael E. Porter, is a model used to determine the industry's weaknesses and strengths and develop corporate strategy (Porter, 1979). It can be applied to any segment of economy, to understand the level of competition within the industry and enhance the company's long-term profitability. We do this by analysing the competitive environment and the threats that it poses, to gain a better understanding of the market, later turning this information to the company's advantage.

### Competitive rivalry

There are approximately 200 food service places in Aalborg (Dun & bradstreet, n.d.). However, when thinking about the café's largest competitors, we can exclude most fast food and takeout establishments, as they do not pose a direct threat to the services provided by Café Peace. Presently, the café works only as a restaurant, without delivery, which means their biggest competition is the other dining places in proximity. And as there is a noteworthy number of alternatives in the city centre, in order to stand out, the focus should be on differentiating the dining experience from the competitors.

### Threat of substitutions

When comparing Café Peace to the other cafés, one concern arises. Specifically, the fact that the food and drink selection is extremely similar. Almost every restaurant offers the same – burgers, sandwiches, salads, steak, and pasta. And while the items on the menu are not the exact same, they are similar enough for the customers to turn their eyes to the competition.

### Buyer bargain power

The existence of similarities between Café Peace and the competition, paired with the fact that the prices are on the higher end compared to some of the competitors, does pose a potential threat to the business. And while we feel that the prices are reasonable for the level of food quality, there is still the factor of eating out on a budget that certainly contributes to the decision-making process of potential customers.

Supplier bargain power

Café Peace partners with AB Catering, which is one of Denmark's largest and leading foodservice providers (AB Catering, n.d.). This comes with several benefits, including access to trained consultants, free delivery, and availability of a local department, which warrants a quick solution in case of missing goods or an accident. However, it is also important to note that having only one supplier can result in increased vulnerability of supply and risk of stock interruption if the supplier's operations are disrupted. Also, being contractually bound to one supplier in the time of rising inflation can be challenging when the supplier decides to increase prices or drop their standard. This is called a lopsided dependency, when the buying business becomes more dependent on the supplier than the other way around (Invest Northern Ireland, n.d.).

### Threat of new entrants into the industry

In theory, we could say that the threat of new entrants is relatively high, since the minimum share capital for starting a private limited company (otherwise known as ApS) is only 40,000DKK (Lawyers Denmark, 2021). However, it is important to note that a start-up most likely would not pose a significant threat for Café Peace, with its established name and loyal customer base. To confirm this, just one look at the restaurant's social media pages suffices – Café Peace currently has a following of over 24 thousand people (Facebook, 2022).

## 1.6. SWOT analysis

After talking to the café's former and current employees and analysing the company from both the internal and the external point of view, we were able to define the favourable and unfavourable areas and conduct a SWOT analysis – a useful tool in the company's strategic planning process (Bloisi, 2006).

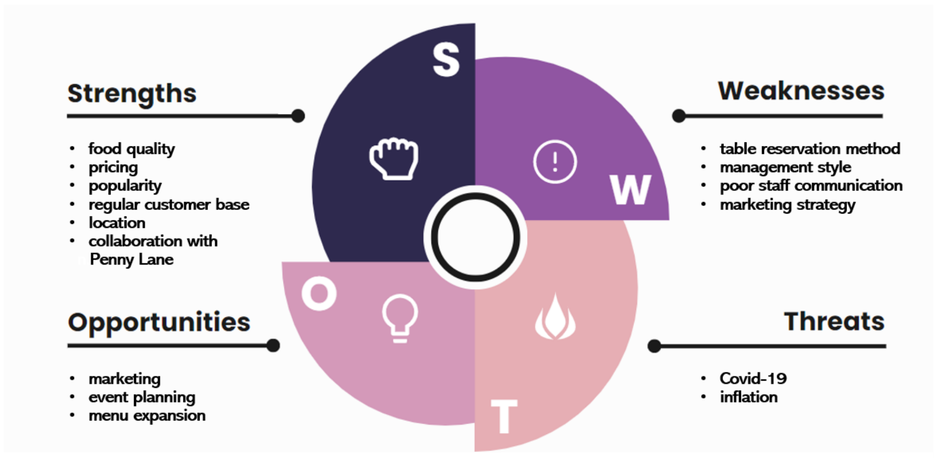


*Figure 4: SWOT*

### Strengths

Café Peace is quite popular amongst the citizens of Aalborg, which can be partially credited to its excellent location in the city centre. It is well-known for the outstanding food quality with reasonable pricing, which has certainly helped in building a regular customer base. The café is right across the street from Penny Lane, which is another popular café in Aalborg, but rather than getting intimidated by the competition, Café Peace has chosen to collaborate with Penny Lane, which is helping both businesses strive.

### Weaknesses

One of the biggest problems for the café is the lack of a table reservation system. Employees have also voiced their concern regarding the café's management style and inconsistencies in staff communication. Up until recently, the café did not have a social media presence and even though this has changed, there is still a lot to improve in their marketing strategy.

### Opportunities

One way to increase profitability is to bring focus to marketing. This could be done by expanding the vague menu selection by adding more options to the already offered food categories. The restaurant is fairly spacious, which presents the opportunity for organising various events on the premises.

### Threats

The main elements which could potentially endanger the business are the rising inflation and the possibility of regulations associated with the spread of the Covid-19 virus.

## 1.7. Strategic Goals

Strategic thinking is the process of envisioning and planning to form a match between organisational competencies and external opportunities with the goal of better serving customers. When setting strategic goals, the management must take into consideration the limitations and weaknesses of their business, which are determined by Porter's five forces and SWOT analysis.

The goals are up to the management, depending on what they want to focus on in the future of their company. This can be things like customer base enlargement, improving marketing strategy, menu expansion, employee training and such.

The Café is planning to expand their restaurant in Aalborg in autumn of this year. With even more tables to manage, we feel that investing in a new table management system is an important strategic goal to consider.

## 1.8. Change management

Change management is a systematic approach to dealing with the transition of an organisation's goals, processes, or technologies. Implementing a table reservation system can be categorized as a transitional change, as falls into the area of automizing tasks and processes. There are several ways of managing a change like this, this is done by implementing practice models, such as Kotter's 8-step Process for leading change, ADKAR, ITIL and such (Duffy, 2018). We have chosen Lewin's Change Management model as our guide. This three-step framework focuses on the Unfreeze-Change-Refreeze idea, using the analogy of changing the shape of a block of ice.

First, we must melt the ice to make it amenable to change (unfreeze). Any successful change process must start by understanding why the transition must take place. As Lewin put it, "Motivation for change must be generated before change can occur." In this stage we must prepare the organization to accept the necessity of this change. This can potentially be the most stressful part, because of the breaking down of status quo and the "way things are done", which can be frustrating for the parties involved. This is the time to focus on gaining support of the key stakeholders we have previously identified. We found ourselves at an advantage, as the management and the employees already exhibit a positive attitude towards this change.

Then, we must mold the iced water into the shape we want to achieve. This is when IT training comes into play. We propose a mandatory instructional meeting that would involve all 34 employees (Dun & Bradstreet, n.d.), to help them gain understanding of the system's functionalities and how to work with it. Since all employees at Café Peace are young people used to working with technologies, and our system is very simple and straight-forward, we believe that a single meeting would suffice.

Finally, the new shape must be solidified (freeze). It is up to the management to make sure the change is being communicated and supported. This can be done by giving feedback when mistakes occur and establishing a reward system for when employees adhere to the new normal.

## 1.9. Business case

This section describes the Café Peace business case, in relation to the implementation of a new IT system. We use this space to examine the available options, conduct a cost-benefit analysis and evaluate the risks and benefits of the switch with the goal of clarifying the purpose behind the implementation and what to expect following this change.

After conducting a business analysis, we were able to single out two available options for further action, first one being to take no action at all and continue with the presently attainable reservation method. Café Peace is currently using manual workforce for grouping and manging all reservations, which has led to several inconveniences surfacing when running the restaurant, as it is interfering with the performance of other tasks.

With errors occurring on a day-to-day basis, we recommend the option of implementing a unified reservation system as it is more beneficial and even substantial for the business to continue to strive. Especially considering the fact that the restaurant is planning on expanding their premises soon, having a more efficient way to manage table bookings is essential. Our product is a simple, clear-cut table reservation system made to fit the software available to Café Peace, with a fully customizable table layout, as to give the ability to make alterations to the table setup to suit the workday ahead. Moreover, the system contains an added feature which allows customers to place menu orders ahead of time, saving time for the kitchen staff.

It is important to note that an implementation of any new IT system comes with several risks that must be considered. The company will become dependent on the use of the system, which can lead to drawbacks in case of system failure or server shutdown. Missing functionalities, presence of bugs or poor usability can have a negative impact on the business. Therefore, testing and debugging need to be prioritized in the development process.

## Cost-benefit analysis

Cost-benefit analysis is a systematic process used to analyse and measure the benefits of an action minus the costs associated with taking that action. This is done to determine whether a project should be carried out before committing resources to it. The table below shows that there are more benefits than costs to carrying out the project, and that the benefits will impact the company in a positive way.

| Costs | | Benefits | |
|---|---|---|---|
| Tangible | Intangible | Tangible | Intangible |
| Learning period (time)<br>Software costs (money) | Tech dependency | Wage savings (money)<br>Efficiency (time) | Increased productivity<br>Clearer overview<br>Employee satisfaction<br>Error prevention<br>Better planning |

*Table 2: Cost-Benefit Analysis*

In addition to the cost-benefit analysis, as a justification behind investing in a new IT system, we have completed a five-year payback projection to determine the investment required for project completion and the estimated savings for the company in the following years.

Looking at the table below, we can see the balance between the investment and the savings and the timeframe in which the project becomes profitable, which is in this case expected already in the first year since the implementation. Staff savings have been determined based on the hourly wage (120kr) for the employee that is called in daily during rush hours (3-hour lunch shift, 2-hour dinner shift). We predict this shift opening will not need to be filled after the system is put into use, which means the company will be saving about 35 hours in employee wages per week.

| Item | Year 1 | Year 2 | Year 3 | Year 4 | Year 5 |
|---|---|---|---|---|---|
| Software purchase | 120 000 | | | | |
| Software maintenance | 30 000 | 15 000 | 15 000 | | |
| IT training | 10 000 | | | | |
| Cumulative total costs | 160 000 | 175 000 | 190 000 | 190 000 | 190 000 |
| Staff savings per year | 200 000 | 200 000 | 200 000 | 200 000 | 200 000 |
| Cumulative savings | 200 000 | 400 000 | 600 000 | 800 000 | 1 000 000 |
| Yearly savings | 40 000 | 185 000 | 185 000 | 200 000 | 200 000 |
| Cumulative total savings | 40 000 | 225 000 | 410 000 | 610 000 | 810 000 |

*Table 3: 5-year Payback Projection*

Based on this table we can conclude that the project is worth doing, as the savings outweigh the costs in the long run substantially.

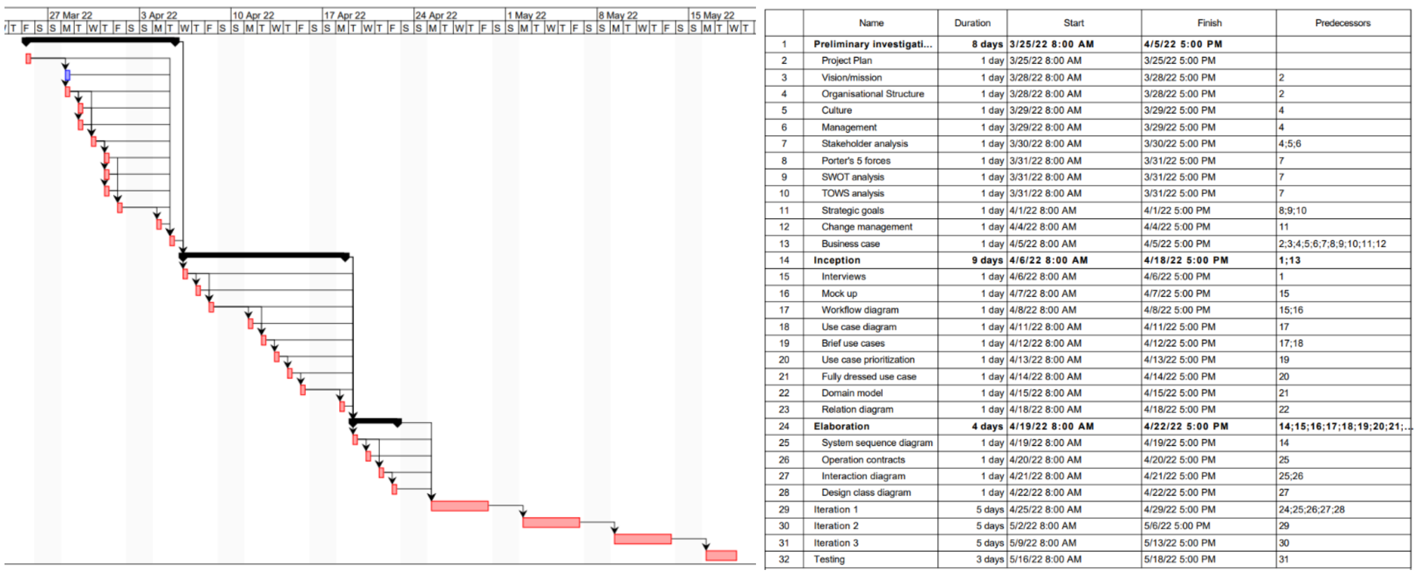
## 2. Project plan

This project is planned with Unified Process framework, using iterative and incremental development process. Unified Process iterative development is split into four phases – Inception, Elaboration, Construction and Transition.

In the first phase of the process, we focused on the Business itself. This phase started with summarization of information about business, its structure, management, culture and ended with identified workflow and use cases. In the business case we concluded if the project should be executed. The Requirements for the project, such as use case elaboration and creation of Domain model, are started.

The Elaboration phase focuses mainly on Requirements, Analysis and Design of the system. This phase usually has two iterations. First, it elaborates on the use cases and the diagram. Domain model is constructed, which is later mapped to Relation model. In the analysis part of this phase, System sequence diagrams with operation contracts are established. In the Design discipline, Interaction diagram and Design class diagram are formed. After the second iteration of the elaboration phase, Construction phase follows. In the construction phase, the system is built iteratively by increments. The implementation of system features, and testing are conducted at this stage. This phase has the most iterations in this project, three iterations of construction phase were conducted.

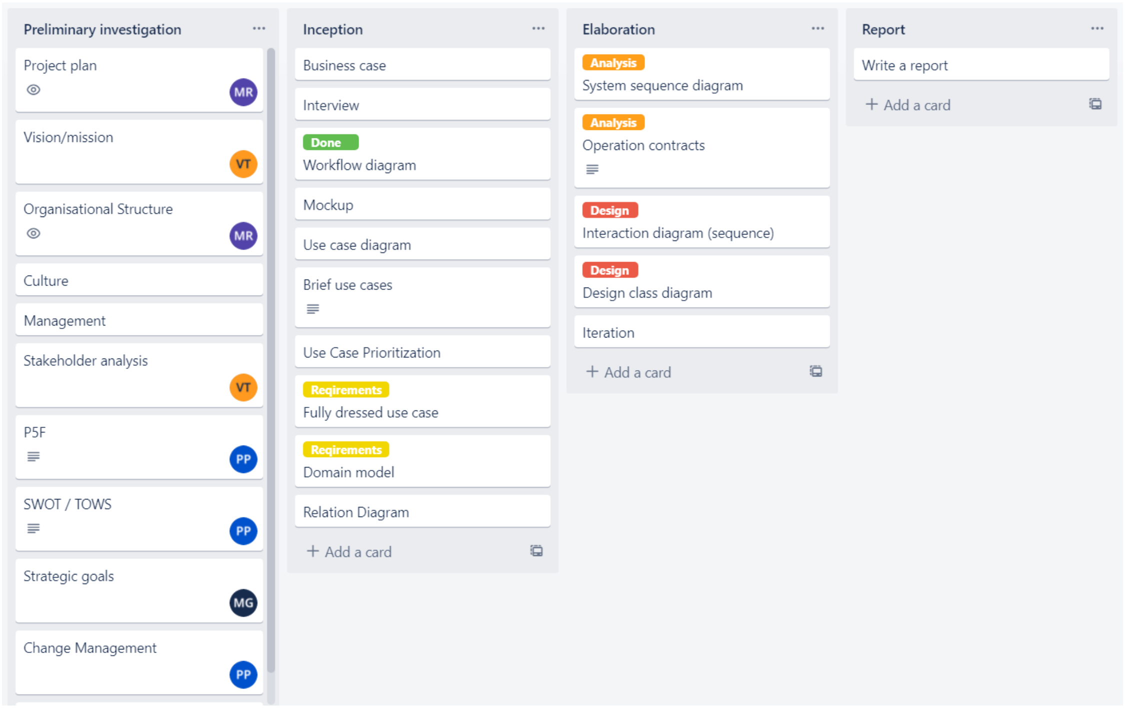
The project's schedule – Gantt chart – was constructed in Project Libre. In the figures below is the spreadsheet of the individual tasks with their duration, dates, and predecessors, which is visualised in the flow chart.



| | Name | Duration | Start | Finish | Predecessors |
|---|---|---|---|---|---|
| 1 | **Preliminary investigati...** | **8 days** | **3/25/22 8:00 AM** | **4/5/22 5:00 PM** | |
| 2 | Project Plan | 1 day | 3/25/22 8:00 AM | 3/25/22 5:00 PM | |
| 3 | Vision/mission | 1 day | 3/28/22 8:00 AM | 3/28/22 5:00 PM | 2 |
| 4 | Organisational Structure | 1 day | 3/28/22 8:00 AM | 3/28/22 5:00 PM | 2 |
| 5 | Culture | 1 day | 3/29/22 8:00 AM | 3/29/22 5:00 PM | 4 |
| 6 | Management | 1 day | 3/29/22 8:00 AM | 3/29/22 5:00 PM | 4 |
| 7 | Stakeholder analysis | 1 day | 3/30/22 8:00 AM | 3/30/22 5:00 PM | 4;5;6 |
| 8 | Porter's 5 forces | 1 day | 3/31/22 8:00 AM | 3/31/22 5:00 PM | 7 |
| 9 | SWOT analysis | 1 day | 3/31/22 8:00 AM | 3/31/22 5:00 PM | 7 |
| 10 | TOWS analysis | 1 day | 3/31/22 8:00 AM | 3/31/22 5:00 PM | 7 |
| 11 | Strategic goals | 1 day | 4/1/22 8:00 AM | 4/1/22 5:00 PM | 8;9;10 |
| 12 | Change management | 1 day | 4/4/22 8:00 AM | 4/4/22 5:00 PM | 11 |
| 13 | Business case | 1 day | 4/5/22 8:00 AM | 4/5/22 5:00 PM | 2;3;4;5;6;7;8;9;10;11;12 |
| 14 | **Inception** | **9 days** | **4/6/22 8:00 AM** | **4/18/22 5:00 PM** | **1;13** |
| 15 | Interviews | 1 day | 4/6/22 8:00 AM | 4/6/22 5:00 PM | 1 |
| 16 | Mock up | 1 day | 4/7/22 8:00 AM | 4/7/22 5:00 PM | 15 |
| 17 | Workflow diagram | 1 day | 4/8/22 8:00 AM | 4/8/22 5:00 PM | 15;16 |
| 18 | Use case diagram | 1 day | 4/11/22 8:00 AM | 4/11/22 5:00 PM | 17 |
| 19 | Brief use cases | 1 day | 4/12/22 8:00 AM | 4/12/22 5:00 PM | 17;18 |
| 20 | Use case prioritization | 1 day | 4/13/22 8:00 AM | 4/13/22 5:00 PM | 19 |
| 21 | Fully dressed use case | 1 day | 4/14/22 8:00 AM | 4/14/22 5:00 PM | 20 |
| 22 | Domain model | 1 day | 4/15/22 8:00 AM | 4/15/22 5:00 PM | 21 |
| 23 | Relation diagram | 1 day | 4/18/22 8:00 AM | 4/18/22 5:00 PM | 22 |
| 24 | **Elaboration** | **4 days** | **4/19/22 8:00 AM** | **4/22/22 5:00 PM** | **14;15;16;17;18;19;20;21;...** |
| 25 | System sequence diagram | 1 day | 4/19/22 8:00 AM | 4/19/22 5:00 PM | 14 |
| 26 | Operation contracts | 1 day | 4/20/22 8:00 AM | 4/20/22 5:00 PM | 25 |
| 27 | Interaction diagram | 1 day | 4/21/22 8:00 AM | 4/21/22 5:00 PM | 25;26 |
| 28 | Design class diagram | 1 day | 4/22/22 8:00 AM | 4/22/22 5:00 PM | 27 |
| 29 | Iteration 1 | 5 days | 4/25/22 8:00 AM | 4/29/22 5:00 PM | 24;25;26;27;28 |
| 30 | Iteration 2 | 5 days | 5/2/22 8:00 AM | 5/6/22 5:00 PM | 29 |
| 31 | Iteration 3 | 5 days | 5/9/22 8:00 AM | 5/13/22 5:00 PM | 30 |
| 32 | Testing | 3 days | 5/16/22 8:00 AM | 5/18/22 5:00 PM | 31 |

*Figure 5: Gantt chart*

For this project, project management application Trello was used, which features the use of static boards. This was utilized to create cards and lists of tasks to be done by group members. When a task was finished, the person who was assigned the task marked it with Done button. In the figure below is the early version of the Trello board for this project.



Preliminary investigation

- Project plan — MR
- Vision/mission — VT
- Organisational Structure — MR
- Culture
- Management
- Stakeholder analysis — VT
- P5F — PP
- SWOT / TOWS — PP
- Strategic goals — MG
- Change Management — PP

Inception

- Business case
- Interview
- Done — Workflow diagram
- Mockup
- Use case diagram
- Brief use cases
- Use Case Prioritization
- Reqirements — Fully dressed use case
- Reqirements — Domain model
- Relation Diagram
- + Add a card

Elaboration

- Analysis — System sequence diagram
- Analysis — Operation contracts
- Design — Interaction diagram (sequence)
- Design — Design class diagram
- Iteration
- + Add a card

Report

- Write a report
- + Add a card

*Figure 6: Trello board*

# 3. Requirements

## 3.1. Workflow diagram

To figure out everything that needed to be included in the new system, we first had to analyse all tasks the employees must perform to complete an order. Afterwards we have constructed an activity diagram, mapping out the workflow.

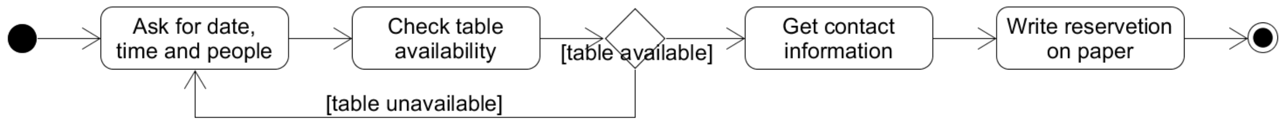


*Figure 7: Workflow diagram*

From the workflow diagram, we have determined the main problem of reserving a table at the café. An employee must be available and spend time with the person making the reservation. They have to write down all of the information on a sheet of paper. If they are busy, they can overlook the fact that they have overbooked the restaurant for a specific day or time.

## 3.2. Mock ups

When creating the mock-ups for our system, we strived to make the system as clear cut and user-friendly as possible. Our goal was to make it so the employees would be able to interact with the system without spending much time on how to use it. In the figure below is an example of one of our mock-ups, specifically the overview of the restaurant layout. For other mock-ups see Appendix C.

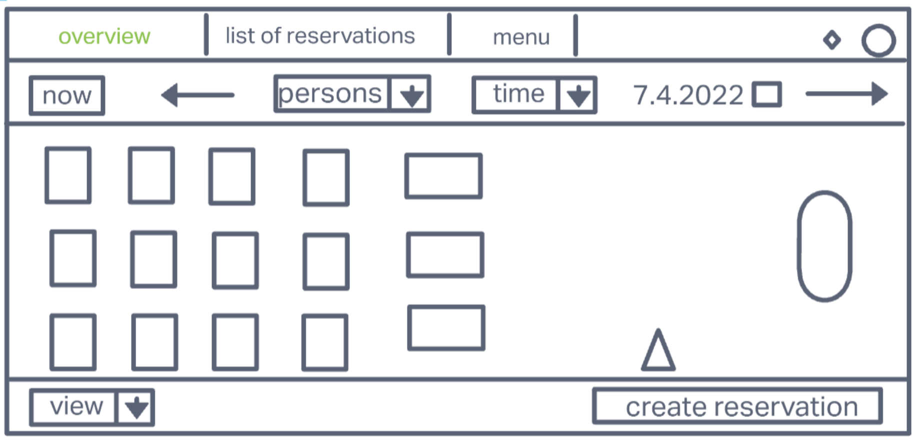


*Figure 8: Restaurant layout overview Mock-up*

## 3.3. Use case diagram

From the use case diagram, we have obtained a visual overview of the actions of our only actor, an employee, and their relations with the use cases. All use cases are within the table reservation system. The first two use cases are in connection with Reservations, one to make a new reservation and the second one is to change or delete a reservation. The other three use cases use CRUD in their naming, which is an abbreviation for Create, Read, Update and Delete.

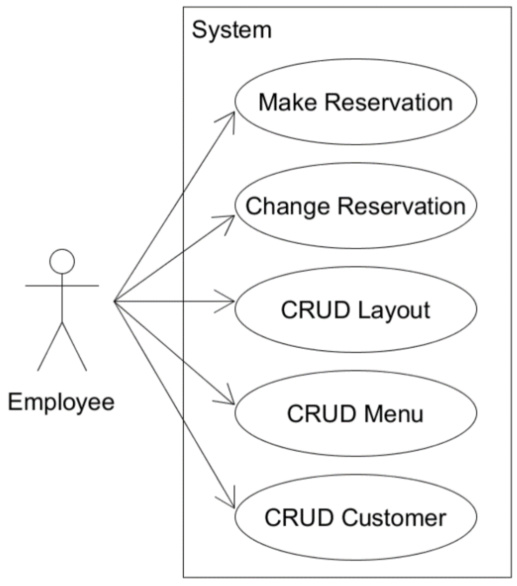


*Figure 9: Use case diagram*

## 3.4. Brief use cases

### *Use case: Make Reservation*

A customer wants to make a reservation. An employee uses the system to check if a table is available within the desired date and time for the number of people. The system shows available tables, and the employee chooses one of them and creates a reservation with name of the customer and his contact information. Optionally, the customer can choose a menu in advance.

### *Use case: Change Reservation*

When a customer wants to change some details in their reservation or to cancel the reservation entirely, an employee can easily carry out these changes in the system.

### *Use case: Create Layout*

This use case is used when an employee wants to create a new layout of the restaurant. For example, the restaurant has an indoor and outdoor space, but in the system, there is only the indoor area. The employee creates the digital layout based the actual layout of the outdoor space of the restaurant.

An employee wants to create a new layout of the restaurant's space. They use the system to create the desired view, with its name and tables.

If the restaurant changes the layout of tables, an employee can edit or even delete a view from the system.

#### *Use case: CRUD Menu*

This use case is used in connection to the restaurant's menus.

If an employee wants to create a new menu option, they choose names and ingredients of different meals in the menu and save it in the system. If they want to change something on the menu or delete it, they have an option to do so.

#### *Use case: CRUD Customer*

This use case is used when a customer wants to have a profile created in the system, for faster creation of reservations and other bonuses.

The customer asks an employee for registration in the restaurants system. The employee inputs customers data into the system and saves it there. If the customer wants to, they can have their information changed or deleted from the system.

### 3.5. Use case prioritization

After establishing use cases for this system, we had to prioritize them. We based the prioritization on criticality and risk of implementing individual use cases. The score we assigned to each use case was established on their complexity and business value. The highest prioritized and most complex use case in this case is the Make Reservation use case. All use case scores and priorities are shown in the table below.

| Use case | Criticality | Risk | Total score | Priority |
|---|---|---|---|---|
| Make Reservation | 5 | 5 | 25 | Highest |
| Change Reservation | 5 | 4 | 20 | High |
| Create Layout | 4 | 5 | 20 | High |
| CRUD Menu | 4 | 3 | 12 | High |
| CRUD Customer | 3 | 2 | 6 | Medium |

*Table 4: Use case prioritization*

## 3.6. Fully dressed use cases

### 3.6.1. Make reservation

Based on the use case prioritization we wrote the Make reservation use case in the fully dressed format. After defining the actors and conditions, we wrote the main success scenario or the main flow of events from the initiation of the reservation creation until the typical, unconditional successful path to a confirmed and saved reservation.

| Use Case | Make reservation | |
|---|---|---|
| Primary Actor | Employee | |
| Preconditions | Employee is logged in | |
| Postconditions | Reservation is created | |
| Flow of events | Actor | System |
| | 1. Employee selects the desired date and time in the system | 2. System shows available tables in the desired time |
| | 3. Employee selects number of people for which the reservation is for | 4. Updates the available tables to comply with the customer's request |
| | 6. Employee selects available table | |
| | 7. Employee asks the customer for additional information and inputs them into the system | |
| | 9. Employee asks the customer if they want to also reserve a specific menu with the reservation if so, they add it to the reservation | |
| | 9. Employee confirms the reservation | 10. Reservation is saved, and the employee is notified |
| | | 11. Table availability is refreshed in the layout |
| Alternative flow of events | *a. At any time, System fails:<br>1. Employee restarts system<br>2. Employee logs in<br>3. Employee start to create reservation again | |
| | 6a. Employee chooses unavailable table:<br>1. System notifies employee<br>2. System denies continuation<br>3. Employee choose different date and time. | |
| | 9a. Employee try to confirm the reservation, but in meantime the table was reserved by another employee:<br>1. System will deny confirmation and notify employee<br>2. Employee choose different date and time. | |

*Table 5: Fully dressed use case: Make reservation*

### 3.6.2. Change reservation

The second fully dressed use case – Change reservation – is described in the table below.

| Use Case | Change reservation | |
|---|---|---|
| Primary Actor | Employee | |
| Preconditions | Employee is logged in | |
| Postconditions | Reservation is changed | |
| Flow of events | Actor | System |
| | 1. Employee finds the reservation | 2. System shows the reservation |
| | 3. Employee changes information | 4. Updates reservation information |
| Alternative flow of events | *a. At any time, System fails:<br>1. Employee restarts system<br>2. Employee logs in<br>3. Employee starts the main scenario | |
| | 3a. Employee chooses unavailable table:<br>1. System notifies employee<br>2. System denies continuation<br>3. Employee choose different date and time | |
| | 4a. Employee try to confirm the reservation, but in meantime the table was reserved by another employee:<br>1. System will deny confirmation and notify employee<br>2. Employee choose different date and time. | |

*Table 6: Fully dressed use case: Change reservation*

### 3.6.3. Create Layout

The third use case – Create Layout – is described in the fully dressed format in the table below.

| Use Case | Create Layout | |
|---|---|---|
| Primary Actor | Employee | |
| Preconditions | Employee is logged in | |
| Postconditions | A new restaurant layout is created | |
| Flow of events | Actor | System |
| | 1. Employee opens layout editor | |
| | 2. Employee sets a size of the layout | |
| | 3. Employee adds layout items to the layout | |
| | 4. Employee set a name for the layout | 5. System checks if the name is available |
| | 6. Employee saves layout | 7. System saves layout |
| Alternative flow of events | *a. At any time, System fails:<br>1. Employee restarts system<br>2. Employee logs in<br>3. Employee start to create reservation again | |

| | |
|---|---|
| | 4a. Employee inputs already existing name for the layout:<br>1. System notifies employee<br>2. System blocks a button for saving the layout<br>3. Employee chooses different name |
| | 6a. Employee tries to save the layout but, in the meantime, there was created the layout with a same name.<br>System cancels saving and notifies employee<br>Employee changes the name of layout |

*Table 7: Fully dressed use case: Create Layout*

## 3.7. System Sequence Diagrams and Operation Contracts

SSD shows system events for one scenario of the use case. We designed it for the following use cases.

### Create reservation

Figure below shows the System Sequence diagram for the use case Create Reservation.

New reservation is created after an employee selects the date, time and table number. Next, customer information is inputted into the system. The system checks if the customer already exists in the database, if so, the customer information is retrieved and automatically filled in the GUI. After confirming the reservation by an employee, it is saved in the database.

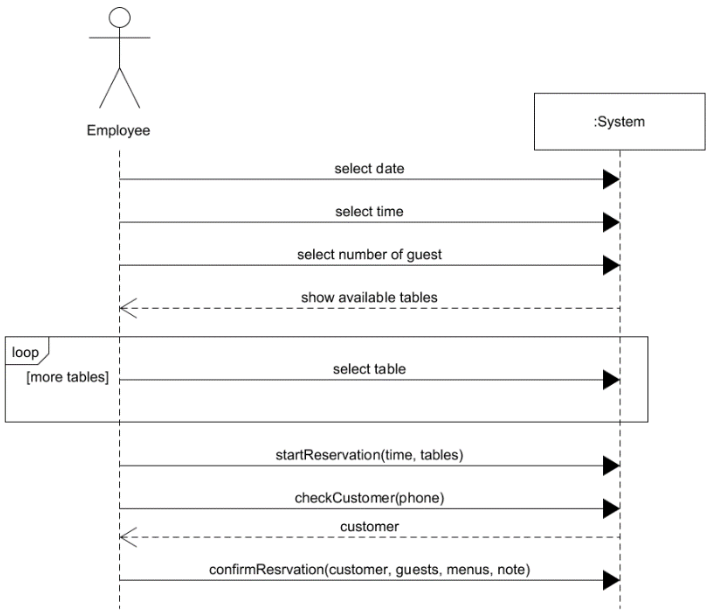


*Figure 10: Create Reservation System Sequence Diagram*

Tables below show Operation Contracts for every system operation in the System Sequence diagram above.

| Operation | startReservation(time, tables) |
|---|---|
| Use case | Create Reservation |
| Precondition | Date, time, and table(s) must be selected<br>Selected tables must exist in the system |
| Postcondition | Reservation object is created |

*Table 8: startReservation() operation contract*

| Operation | checkCustomer(phone) |
|---|---|
| Use case | Create Reservation |
| Precondition | Phone number is inputted |
| Postcondition | Customer object is returned if it exists in the database, if not null is returned |

*Table 9: checkCustomer() operation contract*

| Operation | confirmReservation(customer, guests, menus, note) |
|---|---|
| Use case | Create Reservation |
| Precondition | Reservation object with time and tables is initialized |
| Postcondition | Reservation attributes are set<br>Reservation is created and inserted into database |

*Table 10: confirmReservation() operation contract*

## Change reservation

After the reservation is created and customer want to change any specific details (duration, number of guests, notes, tables). We can get the reservation from the database and through GUI layer type and edit all the necessary information's and call the updateReservation() method to apply changes.

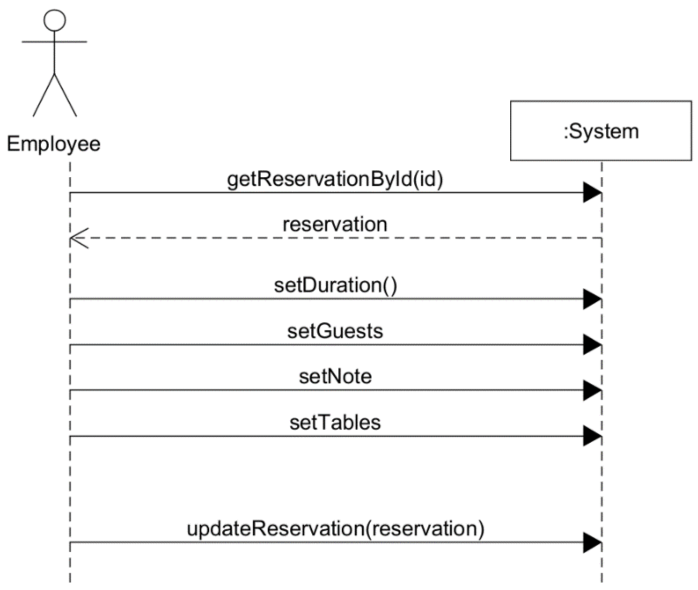


*Figure 11: Change Reservation System Sequence Diagram*

| Operation | getReservationById(id) |
|---|---|
| Use case | Change Reservation |
| Precondition | Id of reservation to be updated is known. |
| Postcondition | Reservation is retrieved from database. |

*Figure 12: getReservationById operation contract*

| Operation | updateReservation(reservation) |
|---|---|
| Use case | Change Reservation |
| Precondition | Employee sets new values for attributes they want to change in the reservation. |
| Postcondition | Reservation is updated in the database. |

*Figure 13: updateReservation operatiion contract*

## Create Layout

An employee starts with setting up the size of the layout so, the system can generate editor panel with requested size. Next, employee adds items into the panel. After he is done with adding the layout items, he sets the name for the layout. When he is finished and satisfied with everything, he saves the layout.

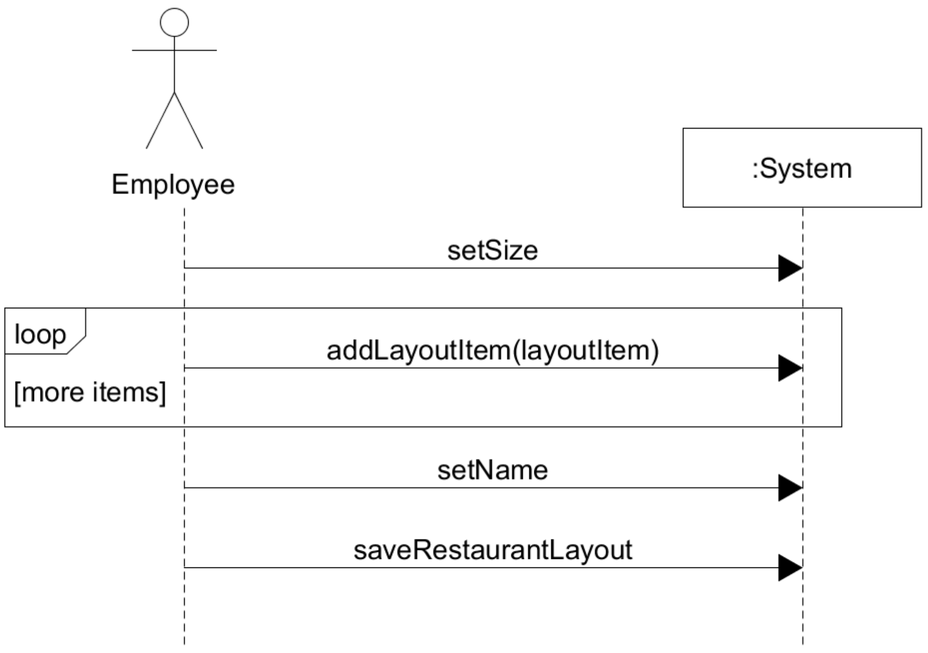


*Figure 14: Create layout system sequence diagram*

| Operation | addLayoutItem(layoutItem) |
|---|---|
| Use case | Create Layout |
| Precondition | Employee set correct non-repeating values for a layout item before it is added to the layout. |
| Postcondition | Layout item is added to the restaurant layout |

*Table 11: addLayoutItem operation contract*

| Operation | saveRestaurantLayout() |
|---|---|
| Use case | Create Layout |
| Precondition | All the layout details were set correctly and without duplicated values. |
| Postcondition | Layout is saved to the database. |

*Table 12: saveRestaurantLayout operation contract*

### 3.8. Domain model

The figure below depicts the domain model for this project. In the diagram, we have conceptualized the main classes, their attributes, and relationships. It illustrates the relationship between the Layout Item as a superclass and the Table as its subclass, along with a composite relationship between the Restaurant Layout and Layout Item classes.

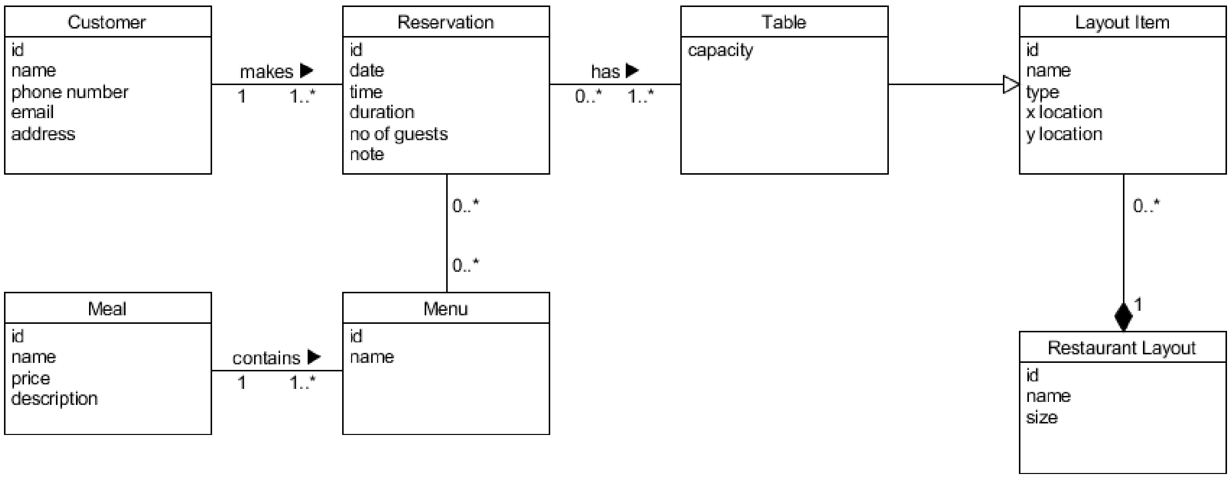


*Figure 15: Domain model*

# 4. Design

## 4.1. Relation model

The diagram below depicts the Relation model for this project, which represents the database and its way of storing data. During the process of creating this relation model, we followed the database design process.

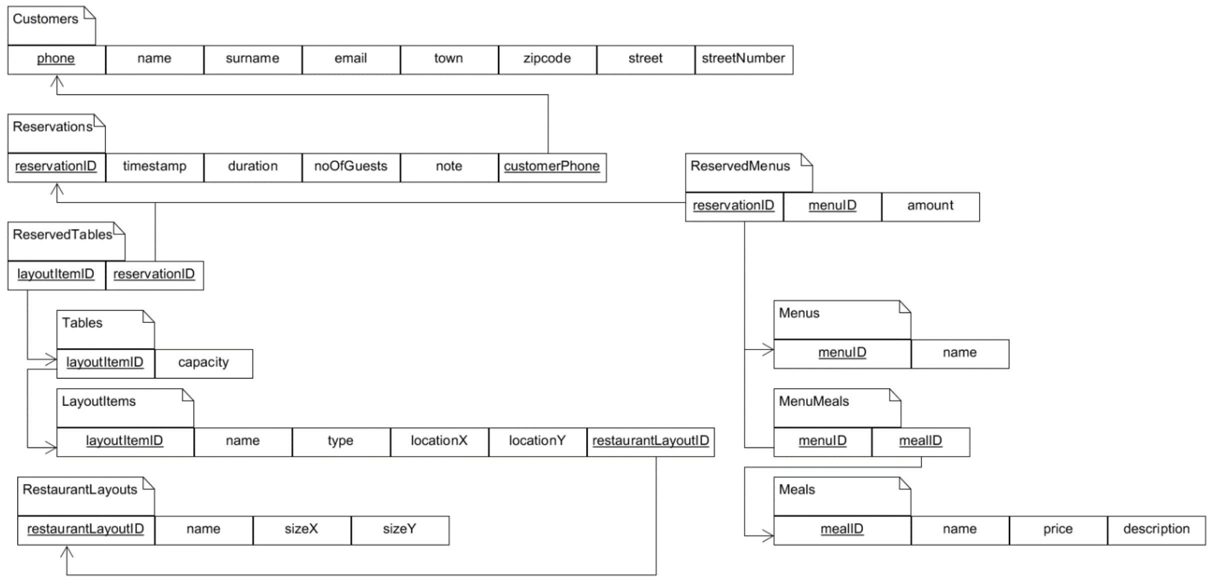


*Figure 16: Relation model*

### Database design process

We mapped the conceptual design from the high-level data model into the data model of the chosen RDBMS. This includes the database schemas, and the mapping strategies between object and non-object representations. The domain model was used as a representation of the data that needed to be persisted.

Conceptual classes from the domain model were mapped into tables in the relation model. We decided to use the plural of the class names as names for our tables, e.g., Reservations, Tables, Customers.

To assure uniqueness, an ID of an object was set as primary key. We looked for the presence of multivalued attributes in the domain model, but since that was not the case, we had nothing to eliminate. Then we looked for composite attributes, which we have divided, for example name attribute was divided into a name and a surname.

In the cases of one-to-many associations, we took a key from one-side and added it as an attribute to the many-side, which is defined with a foreign key constraint. The example of this is the attribute customer's phone from the relation customers which, is a foreign key in the relation reservations.

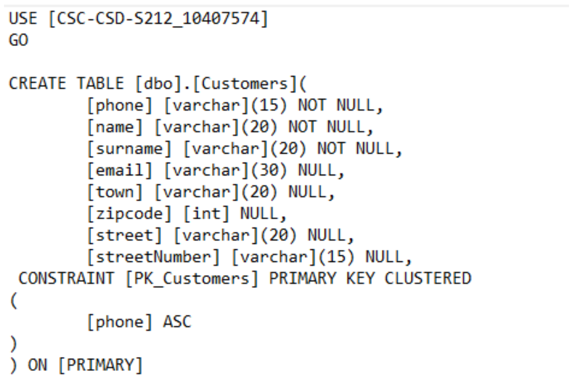
In the cases of many-to-many associations, we added a new table which includes two sets of foreign key attributes, one for each side of the association. The public key of this table is both foreign keys combined. This is done in the ReservedTables and ReservedMenus relations.

To transform generalization association into a relation model we chose "table each class" approach, where each class is mapped into a table. The general and specific parts of an object are bonded together with keys. We chose this approach since the Pull-down method was not possible, because of the creation of the instance of super class LayoutItem. Pull-up was not convenient as it would have created columns with null values.

After mapping the domain model to the relation model, quality, which guarantees absence of defects, was checked via normalization forms. After the 1st normalization of relation, we got rid of multivalued attributes, and all the attributes were of only one type. The 2nd normalization of relation made all non-key attributes fully functionally dependent on a master key. After the 3rd normalization of relation, no non-key attribute was transitively functionally dependent on a candidate key, thus the 3rd normal form finalized the normalization process of our relation model.

For this system, relation database is used. The database provided for this project is a MS SQL version 15.00.2080 database.

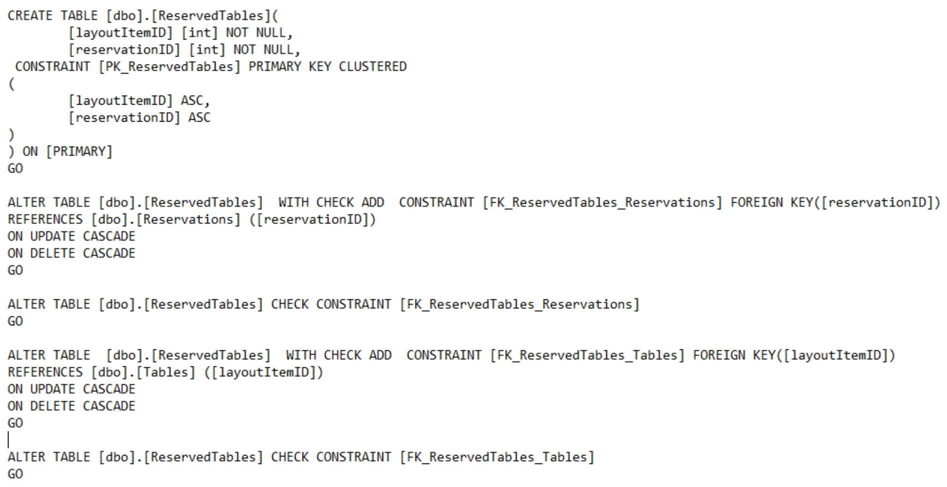
The figures below showcase an example of the SQL scripts for creation of the tables. We used identity specifications with automatic increment by one for ID columns. The tables' on delete and on update constraint was set to cascade. This was done, so transactions don't have to be used when deleting or updating information in the database.



```
USE [CSC-CSD-S212_10407574]
GO

CREATE TABLE [dbo].[Customers](
        [phone] [varchar](15) NOT NULL,
        [name] [varchar](20) NOT NULL,
        [surname] [varchar](20) NOT NULL,
        [email] [varchar](30) NULL,
        [town] [varchar](20) NULL,
        [zipcode] [int] NULL,
        [street] [varchar](20) NULL,
        [streetNumber] [varchar](15) NULL,
 CONSTRAINT [PK_Customers] PRIMARY KEY CLUSTERED
(
        [phone] ASC
)
) ON [PRIMARY]
```

*Figure 17: SQL script*



```sql
CREATE TABLE [dbo].[ReservedTables](
	[layoutItemID] [int] NOT NULL,
	[reservationID] [int] NOT NULL,
 CONSTRAINT [PK_ReservedTables] PRIMARY KEY CLUSTERED
(
	[layoutItemID] ASC,
	[reservationID] ASC
)
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[ReservedTables]  WITH CHECK ADD  CONSTRAINT [FK_ReservedTables_Reservations] FOREIGN KEY([reservationID])
REFERENCES [dbo].[Reservations] ([reservationID])
ON UPDATE CASCADE
ON DELETE CASCADE
GO

ALTER TABLE [dbo].[ReservedTables] CHECK CONSTRAINT [FK_ReservedTables_Reservations]
GO

ALTER TABLE  [dbo].[ReservedTables]  WITH CHECK ADD  CONSTRAINT [FK_ReservedTables_Tables] FOREIGN KEY([layoutItemID])
REFERENCES [dbo].[Tables] ([layoutItemID])
ON UPDATE CASCADE
ON DELETE CASCADE
GO

ALTER TABLE [dbo].[ReservedTables] CHECK CONSTRAINT [FK_ReservedTables_Tables]
GO
```

*Figure 18: SQL scripts*

## 4.2. GRASP Patterns

General Responsibility Assignment Software Patterns, abbreviated as GRASP, is a set of fundamental principles of object-oriented design. The first five basic patterns used in GRASP in the first iteration of elaboration phase are creator, controller, information expert, low coupling, and high cohesion. (Larman, 2004, p. 220)

The creator pattern assigns the responsibility of a class to create instance of another class. In our design, for example, the creator pattern is used when the Reservation object is initiated and used in the ReservationController class, which brings us to the next pattern – Controller.

Controller pattern coordinates requests from the UI-layer and is responsible for receiving or handling a system event. In this system design, we have three main controllers for the three main use cases – Reservation, Customer and Restaurant Layout controllers.

Information expert pattern is used in nearly all our classes, mainly in the controller and data access layer, by placing the responsibility on the class with the most information required to fulfil it.

Coupling is a measure of how strongly one element is connected to, has knowledge of, or relies on other elements. An element with low or weak coupling is not dependent on too many other elements. In our design, low coupling was reached by maintaining low dependency of classes in the system, where it was possible. The DAO pattern, on the other hand, creates high coupling between the interface classes and the implementation classes.

High cohesion was accomplished by keeping our classes and methods appropriately focused on a specific topic, easily manageable and understandable. For example, when accessing data in a database, we use the DAO class for every table in the database.

For more information and examples about coupling and cohesion see chapter Coding Standards.

## 4.3. Interaction Sequence Diagrams

### Create Reservation

The figure below shows the interaction sequence diagram for the create reservation use case. Creation of a new reservation starts in the GUI layer, specifically ReservationPanel class. From there the method createReservationObjects() is called and creates new Reservation object in the ReservationController with date and tables. After the phone number is inputted, the method findCustomer() in CustomerController checks if Customer with that number exists and returns it. When the confirmReservation method is called, the system sets the remaining attributes of Reservation and inserts the object into the database via ReservationConcreteDAO class.

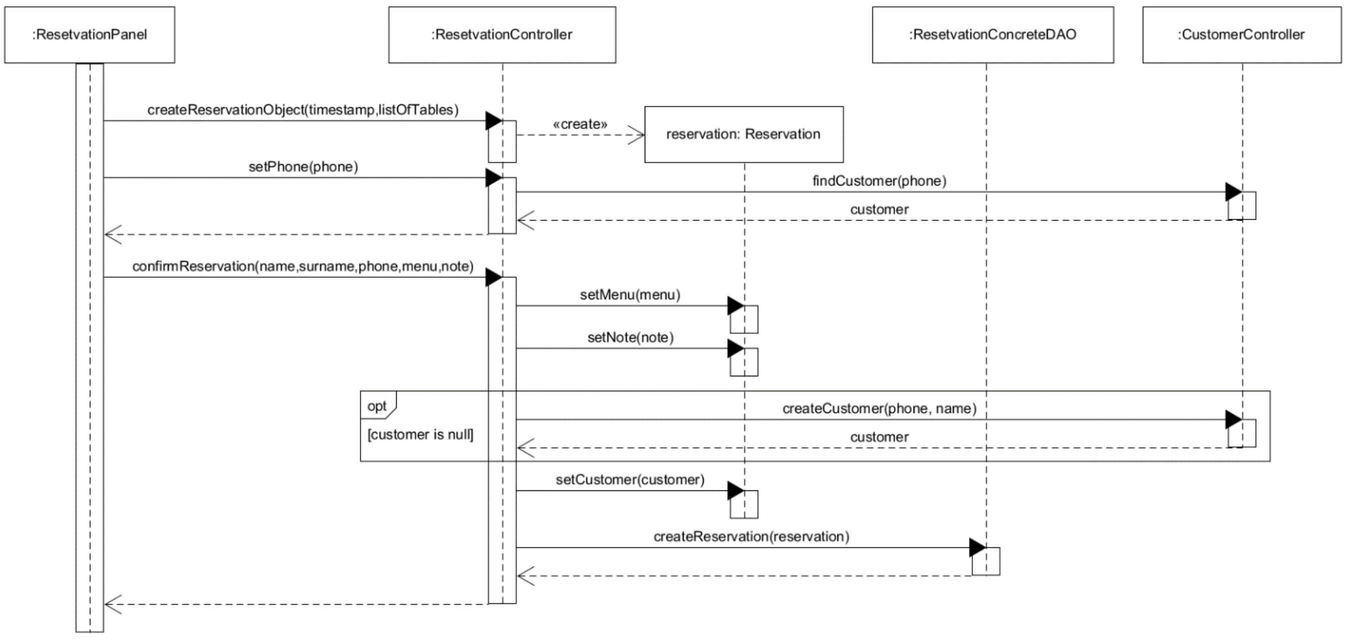


*Figure 19: Interaction Sequence Diagram Create reservation*

### Change reservation

In the use case Change reservation, we first need to retrieve reservation from the database with getCompleteReservationById() method which returns the reservation with the specified

ID. Then in the GUI panel we can set attributes we would like to change (duration, guests, note, tables). We apply changes with a setter method and call method update() which sends the reservation to the database through the DAO class and stores it there.

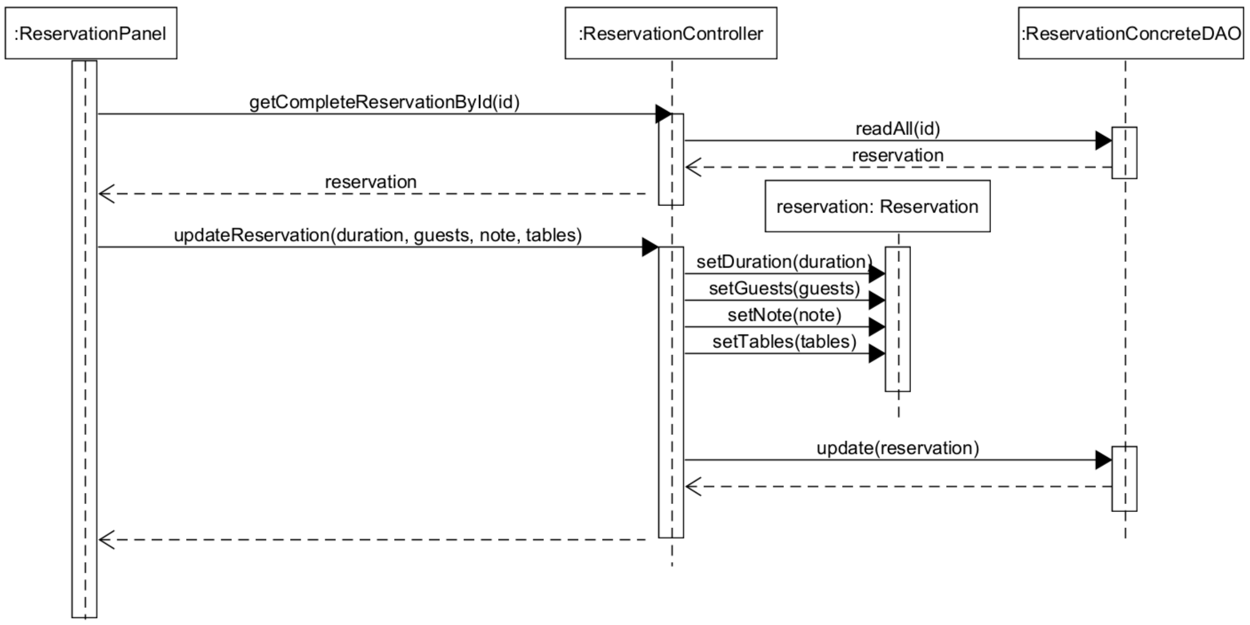


*Figure 20: Interaction Sequence Diagrams Change reservation*

## Create Layout

After an employee is done with editing the layout in the GUI class layoutEditorPanel, and saves the layout with saveRestaurantLayout(name,sizeX,sizeY,itemMap) method. Method from SaveRestaurantLayout(name,sizeX,sizY, itemMap) is then executed from the RestaurantLayoutController and this method starts a transaction which is holding create methods.

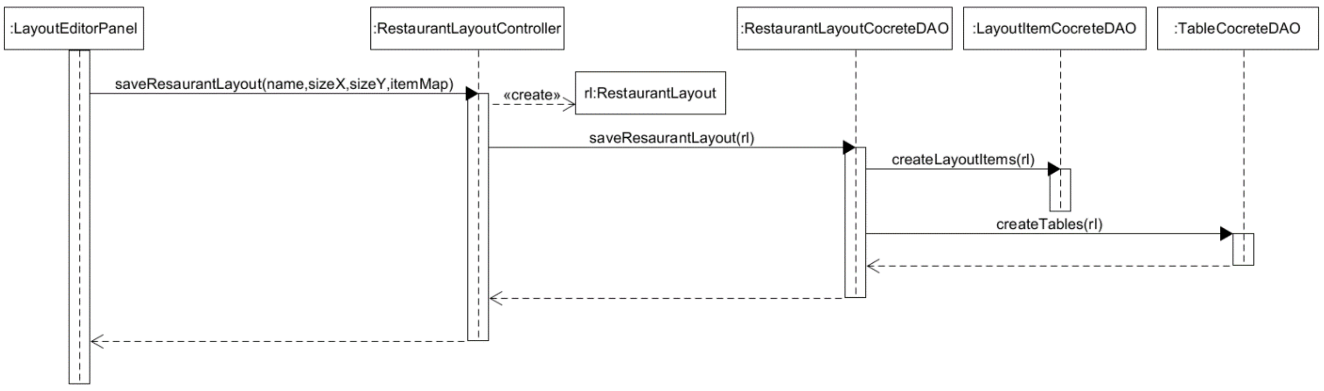


*Figure 21: Interaction Sequence Diagrams Create layout*

## 4.4. Design class diagram

In the figure below is the Design class diagram, which visualizes the architecture of our system – its software classes with attributes and methods and the relationships between them. This diagram is the most complex one, as it is the last part of the system development before the implementation phase. Due to the extensive amount of DAO classes, the complexity and size of the diagram increased over the development period. Because of that, we omitted setters and getters in the model layer and the user interface classes as well, to make it easier to navigate. Relationships between classes, if the class is using the other class only as a return type, are not shown. For larger full-page version of the diagram see appendix.

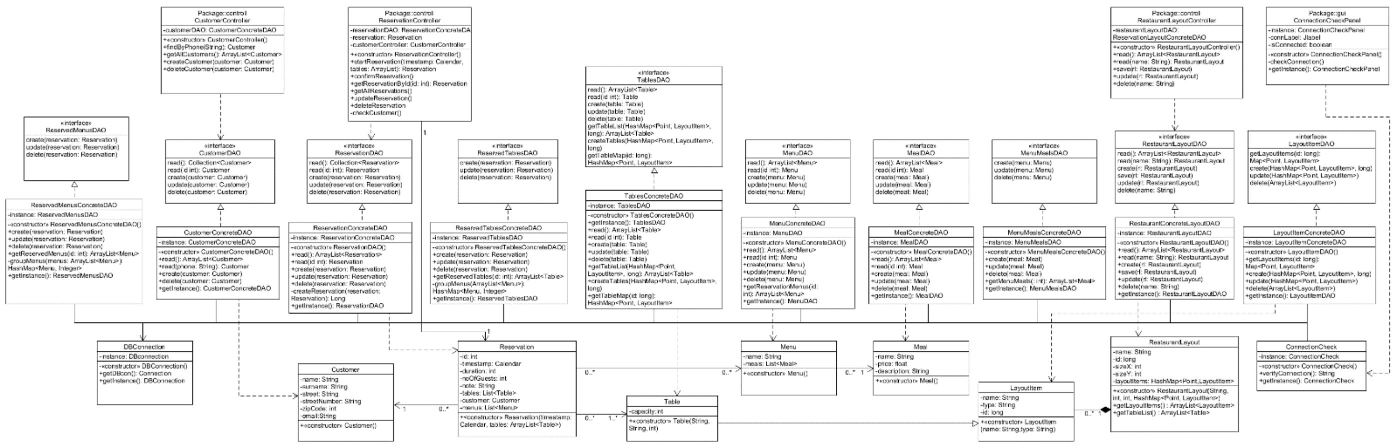


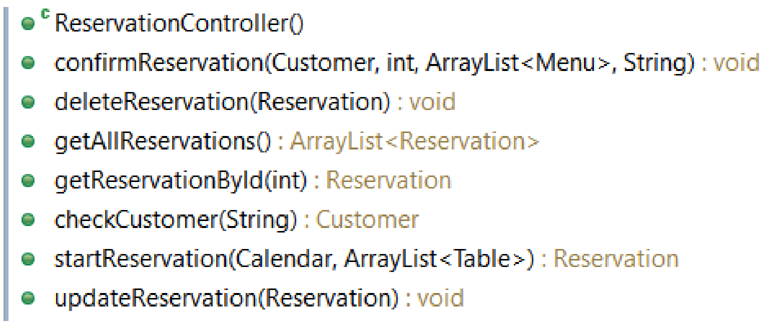
*Figure 22: Design class diagram*

# 5. Implementation

## 5.1. Code standards

During the implementation phase we strived to keep relative elements of classes together by binding all our related code as close as possible to ensure our cohesion would be high.

On the other hand, we tried to isolate our classes to be as independent as possible to ensure our coupling would be low.



```
ReservationController()
confirmReservation(Customer, int, ArrayList<Menu>, String) : void
deleteReservation(Reservation) : void
getAllReservations() : ArrayList<Reservation>
getReservationById(int) : Reservation
checkCustomer(String) : Customer
startReservation(Calendar, ArrayList<Table>) : Reservation
updateReservation(Reservation) : void
```

*Figure 23: Method names*

In the code we used descriptive and appropriate names for variables, methods, interfaces, and classes, in accordance with Java naming convention. An example of method names of ReservationController class are in the figure above. Names of packages, variables and methods start with lower-case letters. Names of classes and interfaces are descriptive and start with an upper-case letter. If the name of an element is made up of multiple words, the name is usually internally capitalized.

## 5.2. Architecture

In this project, three-layer open architecture is used, as it provides a lot of freedom to the developers who can separately update or replace only specific parts of the application without influencing the product as whole. Each of us can work on different sections of the application, improving our efficiency and speed. Having an open architecture means that a given layer can make use of any of the layers beneath it.

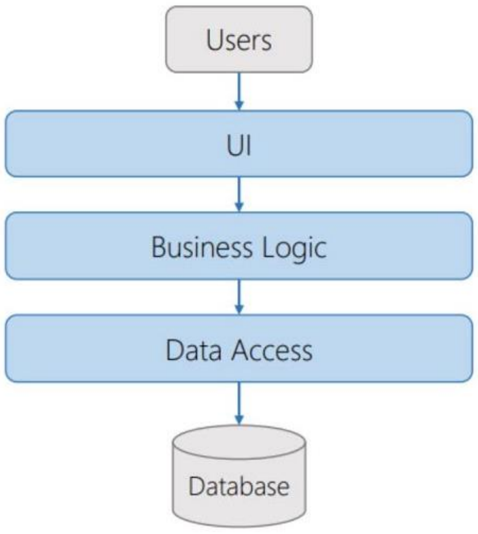


*Figure 24: 3-layer architecture*

In the picture we can see that this architecture logically separates code into three parts.

UI or presentation layer is the layer with which the user directly interacts. This layer is a user interface, the mechanism for getting input from the user. At the time of drafting this report we have not had time to implement GUI, but we are planning to use Java Swing to build our application.

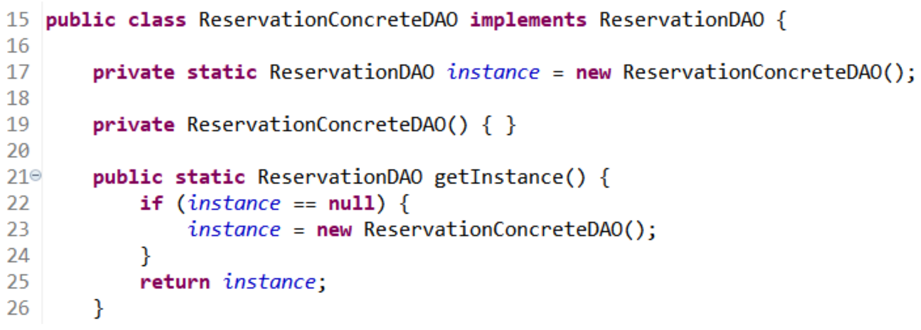
Business logic layer contains a set of components that are responsible for processing the information received from the UI layer. It implements all the necessary application logic, all the calculations. It interacts with the database and passes the result of processing to the UI layer. It contains controllers and view models such as Reservation Controller, Customer Controller, etc.

In Data Access layer we have used data access object (DAO)  a pattern that provides an abstract interface to our database. In this layer we have various methods that are working with the database. We can update delete insert or read row of fields stored in there and if necessary convert them to Java object which we can later work with.

## 5.3. Code Examples

### Singleton

All of the DAO implementation classes use singleton. We use the singleton pattern to instantiate only one instance of the DAO classes. The pattern consists of a private constructor of the class and the *instance* of the class, which is accessible via public static getInstance() method. The figure on the right is an example of one of the DAO classes in our code, specifically the ReservationConcreteDAO.



```
public class ReservationConcreteDAO implements ReservationDAO {

    private static ReservationDAO instance = new ReservationConcreteDAO();

    private ReservationConcreteDAO() { }

    public static ReservationDAO getInstance() {
        if (instance == null) {
            instance = new ReservationConcreteDAO();
        }
        return instance;
    }
```

*Figure 25: Singleton example*

## Lambda and Thread

We implemented Lambda expression in our code to create Thread via Java Runnable. The task in the thread checks the connection to the database every five seconds. Status of connection is presented to the user in the GUI via JLabel with text and green or red colour of the text, depending on the connection.

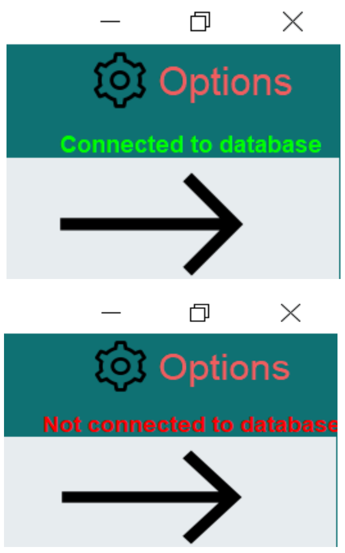
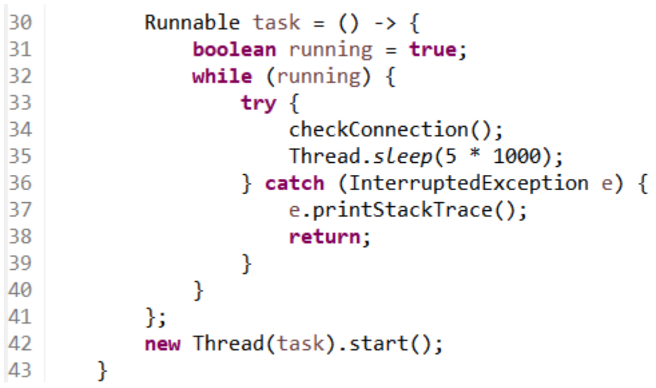


Figure 26: Connection to database



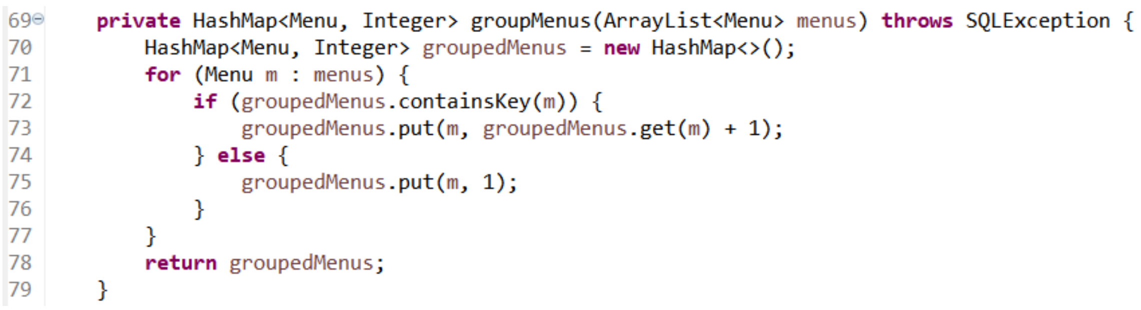
```java
        Runnable task = () -> {
            boolean running = true;
            while (running) {
                try {
                    checkConnection();
                    Thread.sleep(5 * 1000);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                    return;
                }
            }
        };
        new Thread(task).start();
    }
```

Figure 27: Thread and Lambda expression

We also considered a fact that after implementation of another thread we must challenge multithreading problems such as deadlocks, livelocks and starvations. We approached this problem by finding critical resources which is shared in between more threads.

## Collections

Throughout the code we used mostly ArrayLists, but when inserting Reserved Menus into the database, we needed to convert ArrayList of Menus to HashMap. The Key of the HashMap



```java
    private HashMap<Menu, Integer> groupMenus(ArrayList<Menu> menus) throws SQLException {
        HashMap<Menu, Integer> groupedMenus = new HashMap<>();
        for (Menu m : menus) {
            if (groupedMenus.containsKey(m)) {
                groupedMenus.put(m, groupedMenus.get(m) + 1);
            } else {
                groupedMenus.put(m, 1);
            }
        }
        return groupedMenus;
    }
```

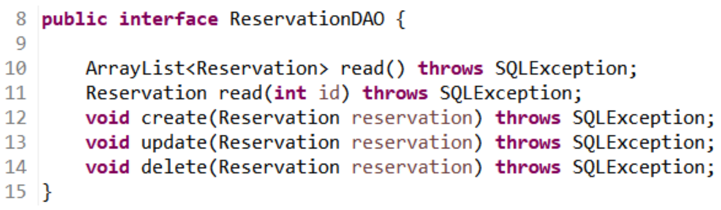
Figure 28: Collection

was Menu object and the value of that key was the amount of reservations of the menu. This was done, because the ReservedMenus table stores the amount of reserved Menus.

## DAO

When accessing data in the database, we used Data Access Object pattern. This pattern provides an abstract interface. Every relation in the database has its own DAO interface and class that implements that interface. Each of the classes that implement the DAO interface has ConcreteDAO in their name.

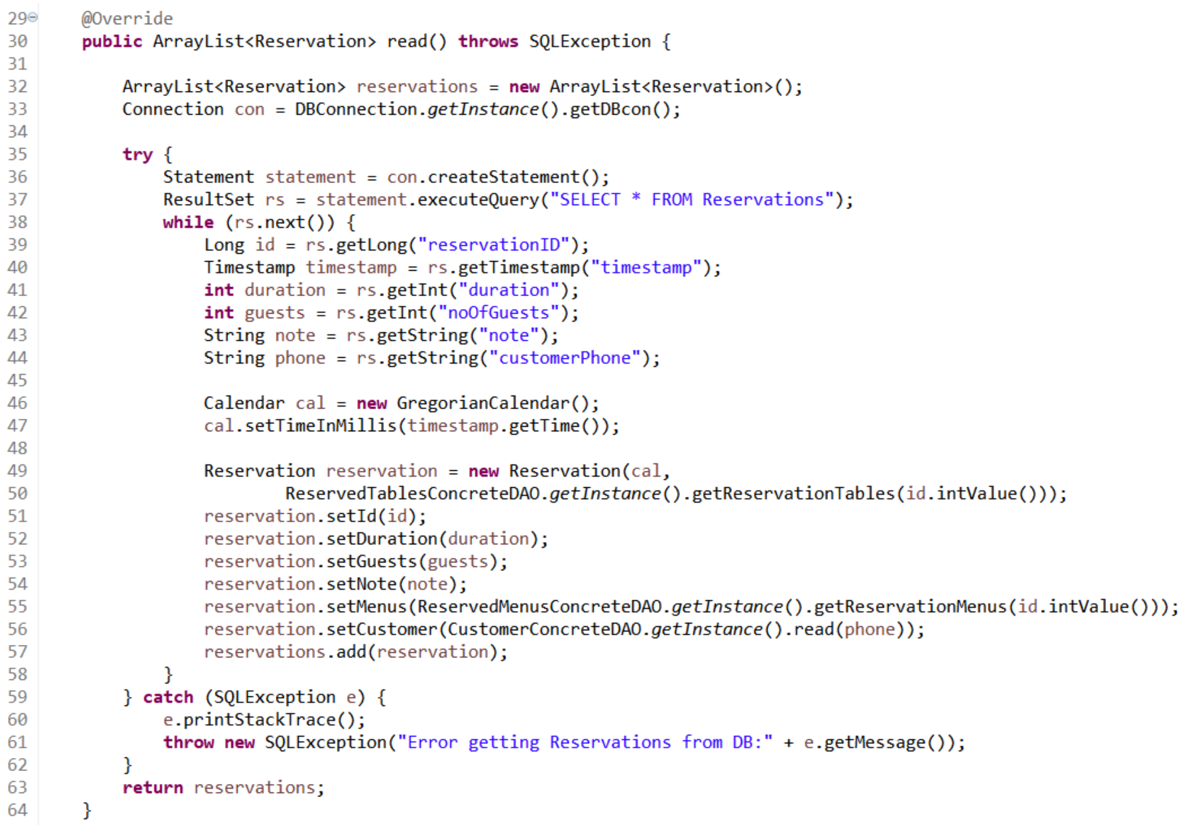
In the figures below is ReservationDAO interface and the method which implements this interface – ReservationConcreteDAO.



```java
public interface ReservationDAO {

    ArrayList<Reservation> read() throws SQLException;
    Reservation read(int id) throws SQLException;
    void create(Reservation reservation) throws SQLException;
    void update(Reservation reservation) throws SQLException;
    void delete(Reservation reservation) throws SQLException;
}
```

*Figure 29: DAO interface*



```java
    @Override
    public ArrayList<Reservation> read() throws SQLException {

        ArrayList<Reservation> reservations = new ArrayList<Reservation>();
        Connection con = DBConnection.getInstance().getDBcon();

        try {
            Statement statement = con.createStatement();
            ResultSet rs = statement.executeQuery("SELECT * FROM Reservations");
            while (rs.next()) {
                Long id = rs.getLong("reservationID");
                Timestamp timestamp = rs.getTimestamp("timestamp");
                int duration = rs.getInt("duration");
                int guests = rs.getInt("noOfGuests");
                String note = rs.getString("note");
                String phone = rs.getString("customerPhone");

                Calendar cal = new GregorianCalendar();
                cal.setTimeInMillis(timestamp.getTime());

                Reservation reservation = new Reservation(cal,
                        ReservedTablesConcreteDAO.getInstance().getReservationTables(id.intValue()));
                reservation.setId(id);
                reservation.setDuration(duration);
                reservation.setGuests(guests);
                reservation.setNote(note);
                reservation.setMenus(ReservedMenusConcreteDAO.getInstance().getReservationMenus(id.intValue()));
                reservation.setCustomer(CustomerConcreteDAO.getInstance().read(phone));
                reservations.add(reservation);
            }
        } catch (SQLException e) {
            e.printStackTrace();
            throw new SQLException("Error getting Reservations from DB:" + e.getMessage());
        }
        return reservations;
    }
```

*Figure 30: Dao implementation*

## Transaction

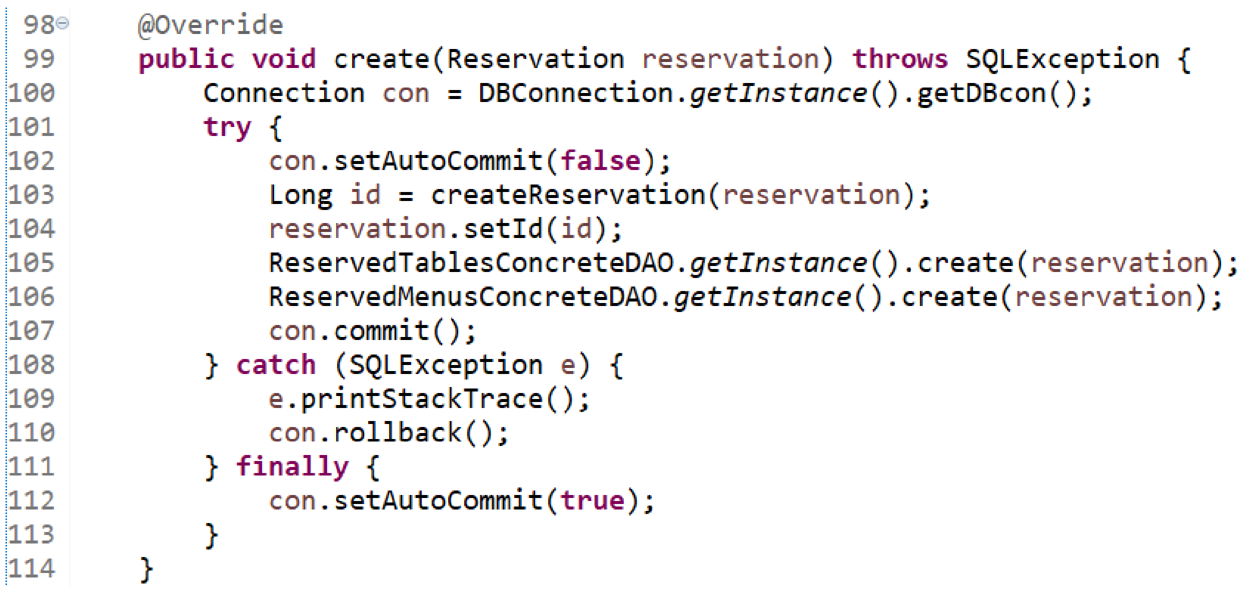
The next picture shows the implementation of a transaction in our code. The method is in the ReservationConcreteDAO class.

We implemented transaction because we needed to execute more SQL statements at the same time. The transaction assures that every statement which is part of it will be executed or none of the statements will be executed.

In the line 102 we set auto-commit to false, and after all sub-methods are successfully loaded, the transaction is committed (line 107). If some of the sub-methods of the transaction are not successfully executed, and throw an exception, the exception is caught, and the rollback method is called to restore the previous changes.

In *finally* we set the auto-commit back to true, so it does not mess with the rest of our code. All the code in the lines that we have referred to is a guaranty for the right transaction execution.

In our project implementation we use more than one transaction. You can find other transactions in the update method of the same class and in the class RestaurantLayoutConcreteDAO in the methods "save" and "update".



```
 98    @Override
 99    public void create(Reservation reservation) throws SQLException {
100        Connection con = DBConnection.getInstance().getDBcon();
101        try {
102            con.setAutoCommit(false);
103            Long id = createReservation(reservation);
104            reservation.setId(id);
105            ReservedTablesConcreteDAO.getInstance().create(reservation);
106            ReservedMenusConcreteDAO.getInstance().create(reservation);
107            con.commit();
108        } catch (SQLException e) {
109            e.printStackTrace();
110            con.rollback();
111        } finally {
112            con.setAutoCommit(true);
113        }
114    }
```

*Figure 31: Transaction*

## Batch processing

With the batch processing we can group related SQL statements into a batch and submit them with one call to the database.

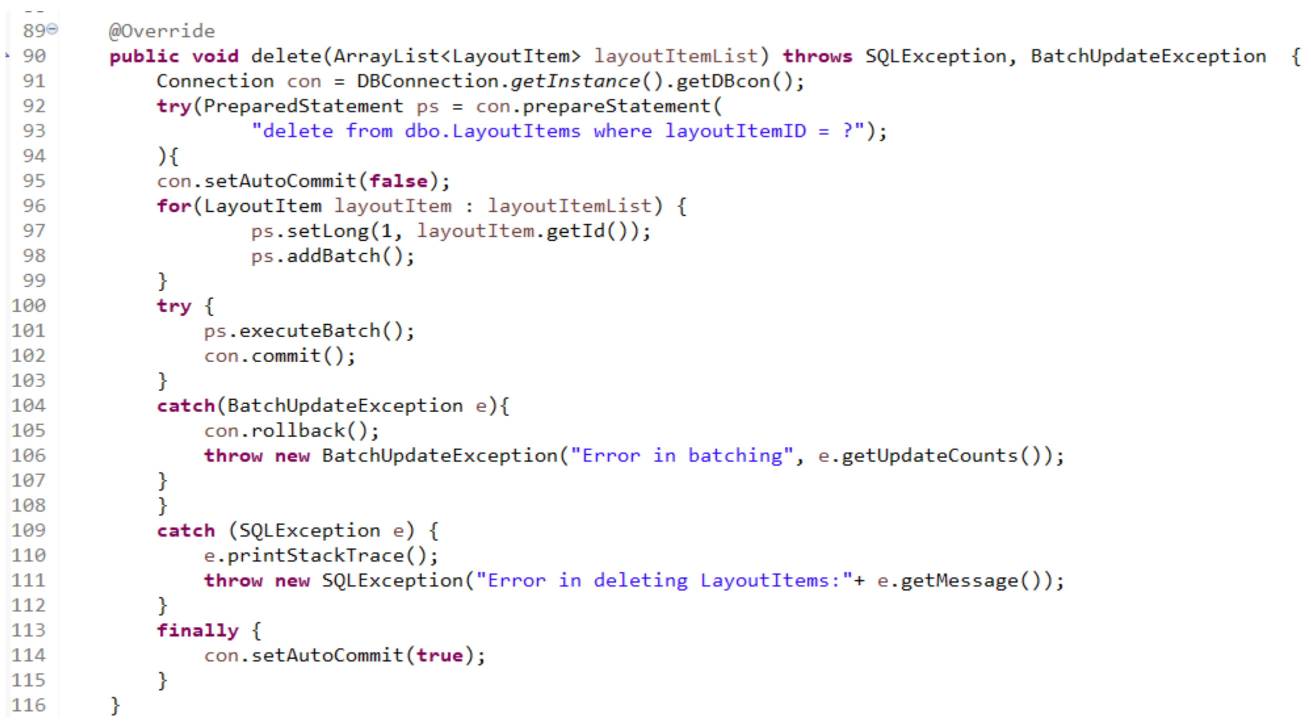
We find this solution convenient because instead of multiple requests to the database, we call just one request containing all the statements at once, so we reduce the amount of communication overhead, thereby improving performance.

Normally, if a one or more of the statements inside the batch are not for some reason successfully executed by ps.executeBatch() (line 101), they are left out from the operation, and not executed in the database. Only the statements without exceptions and errors will be executed.

As we did not find this solution sufficient, so we decided to use transaction-like approach, unless all of the statements in the batch are executed, nothing will be executed.

We achieved this by turning auto commit off (line 95), and then by committing the batch inside a try block (line 102). If any of the statements inside the batch cannot be committed, then the BatchUpdateException is thrown, caught and the whole batch is rolled back (line 105). Finally, the auto commit mode is set back to true value.

Method used in this example is in the LayoutItemConcreteDAO class, but we use batches widely throughout the project, mostly in ConcreteDAO classes.



```
 89     @Override
 90     public void delete(ArrayList<LayoutItem> layoutItemList) throws SQLException, BatchUpdateException  {
 91         Connection con = DBConnection.getInstance().getDBcon();
 92         try(PreparedStatement ps = con.prepareStatement(
 93                 "delete from dbo.LayoutItems where layoutItemID = ?");
 94         ){
 95         con.setAutoCommit(false);
 96         for(LayoutItem layoutItem : layoutItemList) {
 97                 ps.setLong(1, layoutItem.getId());
 98                 ps.addBatch();
 99         }
100         try {
101             ps.executeBatch();
102             con.commit();
103         }
104         catch(BatchUpdateException e){
105             con.rollback();
106             throw new BatchUpdateException("Error in batching", e.getUpdateCounts());
107         }
108         }
109         catch (SQLException e) {
110             e.printStackTrace();
111             throw new SQLException("Error in deleting LayoutItems:"+ e.getMessage());
112         }
113         finally {
114             con.setAutoCommit(true);
115         }
116     }
```

*Figure 32: Batch*

# 6. Testing

## 6.1. Create Layout

This test is testing whether newly created layout via SaveRestaurantLayout() method is persisted into the database. The test case with ID 01 tests method with proper parameters. The result was as we excepted and the test passed. In the test case with ID 02 we tested whether system throws an exception if someone tries to create a layout with already existing name. The result of the test was as we excepted, so both of our test cases for this use case passed.

| Test Case ID | Date | Test Case Description | Test Data | Expected Result | Actual Result | Pass/Fail |
|---|---|---|---|---|---|---|
| 01 | 23.05.2022 | Check method saveRestaurantLayout() with valid layout name | Layout name = "TestEclipse" | New layout is inserted to the database | As expected | PASS |
| 02 | 23.05.2022 | Check method saveRestaurantLayout)with invalid layout name | Layout name = "TestEclipse 2" | The Exception is thrown because there already exists layout with this name | As expected | PASS |

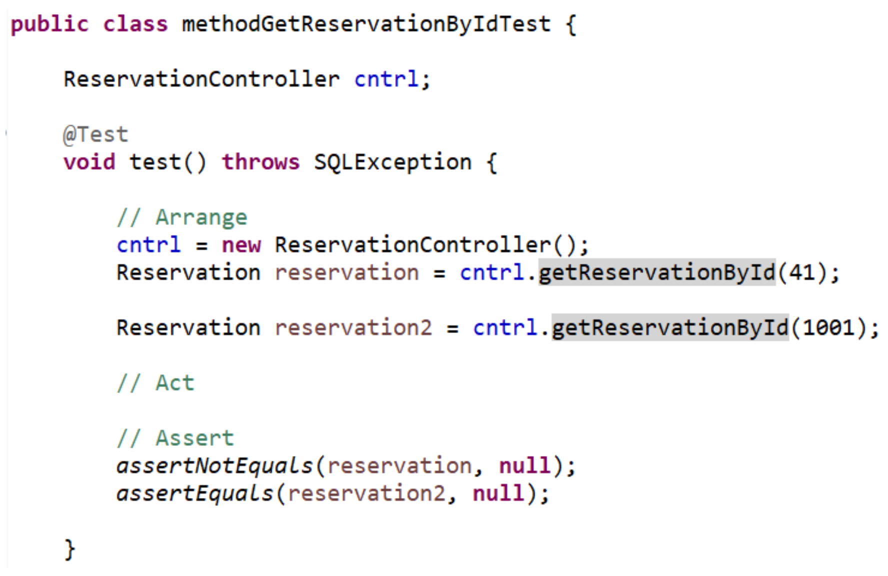
*Figure 33: Test case Create Layout*

## 6.2. Change reservation

In method getReservationById() we were testing valid and invalid id numbers. If the method is called with a valid id, the corresponding reservation should be retrieved from the database. If the wrong id is inserted, the reservation should not be returned and should stay as a null object. All tests were completed successfully, and all have passed.

| Test Case ID | Date | Test Case Description | Test Data | Expected Result | Actual Result | Pass/Fail |
|---|---|---|---|---|---|---|
| 01 | 21.05.2022 | Check method getReservationById() with valid id | id = 41 | Reservation with specified id is pulled from database | As expected | PASS |
| 02 | 21.05.2022 | Check method getReservationById() with invalid id | id = 1001 | No reservation is pulled from database | As expected | PASS |

*Figure 34: Test case change reservation*



```
public class methodGetReservationByIdTest {

    ReservationController cntrl;

    @Test
    void test() throws SQLException {

        // Arrange
        cntrl = new ReservationController();
        Reservation reservation = cntrl.getReservationById(41);

        Reservation reservation2 = cntrl.getReservationById(1001);

        // Act

        // Assert
        assertNotEquals(reservation, null);
        assertEquals(reservation2, null);

    }
```

*Figure 35: Test method*

## 6.3. Create Reservation

In this method we are testing if createCustomer() method is executed properly every time we call it. As the table below shows, if we try to create new customer with a phone number that already exists in the database, the exception is thrown and no customer is created.

| Test Case ID | Date | Test Case Description | Test Data | Expected Result | Actual Result | Pass/Fail |
|---|---|---|---|---|---|---|
| 01 | 21.05.2022 | Check method createCustomer() with valid phone number | phone = 387654359 | New customer in database is created. | As expected | PASS |
| 02 | 21.05.2022 | Check method createCustomer() with invalid phone number | Phone = 322114512 | Exception is thrown | As expected | PASS |

*Figure 36: Test case create reservation*

# 7. Conclusion

This section focuses on evaluating the results of the project. Below, we answer the question we have asked when drafting the problem statement at the beginning of our project work.

*How can we unify the reservations and make it easier and faster for customers and employees at the same time?*

Our proposal to create a unified table reservation system solves the issue of grouping and managing the reservations in a simple way, speeding up the entire process. Employees do not have to spend time and brainpower on figuring out the availability in order to plan ahead, the system does it for them, which frees up time to focus on other tasks at hand. The menu function helps the employees prepare for the customer's arrival. When using a paper spreadsheet, the margin for error is high, along with the risk of it getting misplaced or damaged. Our system stores this data safely, and keeps log of all previous reservations, so the information can be accessed whenever necessary. The reservation process can also be sped up if a customer registers within the system.

The result of this project was a successful construction and implementation of the proposed system, with the most important features being realized before the deadline of the planned period.

## 8. Group evaluation

In this section, we evaluate the group process during the work on the 2nd semester project at the Computer Science course at UCN in Aalborg. The appendix features the group contract, and here we conclude a description of the key aspects of our work, including the problems we have run into.

It was our goal to use the time we had as adequately as we could, however we found ourselves struggling to utilize it efficiently. In the finishing stages of our project, we have agreed that we should have started our project work sooner and made more effort to adhere to the plan. We feel that we have not always contributed to the best of our abilities. This was brought on by the fact that all group members have spent a certain amount of time travelling, which meant it was harder to align the times in which we were all available and arrange our meetings.

For the most part, we have communicated successfully and have not run into any problems regarding miscommunication. Everyone had an equal voice when it came to decision making, we made sure that everyone's opinions were heard and considered. When it came to task division, we have tried to divide the work in a way that everyone got to work on the part they wanted and agreed to help each other if one of us was struggling with the task at hand.

In our opinion, we worked well together as a group, despite the presence of minor complications, as they are an indisputable part of the learning experience.

# Appendices

## Appendix A. List of references

Café Peace. (n.d.) *OM CAFE PEACE* [About Café Peace]. Retrieved April 7, 2022, from https://cafepeace.dk/om-cafe-peace/

proff.dk. (n.d.) *CAFÉ PEACE ApS*. Retrieved March 30, 2022, from https://www.proff.dk/firma/caf%C3%A9-peace-aps/aalborg/barer-og-puber/0JBBWNI01X8/

proff.dk. (n.d.) *KØNIG HOLDING AALBORG ApS*. Retrieved March 30, 2022, from https://www.proff.dk/firma/k%C3%B8nig-holding-aalborg-aps/n%C3%B8rresundby/anden-forretningsservice/0I6ENZI10LQ/

Gray, C. F. (2003). Project Management: The Managerial Process (5th ed.). McGraw-Hill.

Brooks, I. (2009). Organisational Behaviour: Individuals, Groups and Organisations. (4th ed.). Pearson Education Limited.

Management Study Guide. (n.d.) *Understanding Organization and Organization Culture.* Retrieved April 8, 2022, from https://www.managementstudyguide.com/organization-culture.htm

Mind Tools. (n.d.) *Stakeholder Analysis.* Retrieved April 9, 2022, from https://www.mindtools.com/pages/article/newPPM_07.htm

Danda, D. (2020). *Analyze your Project's Stakeholders with the help of the Salience Model.* Retrieved April 8, 2022, from https://www.greycampus.com/blog/project-management/analyze-your-project-s-stakeholders-with-the-help-of-the-salience-model

Porter, M. E. (1979) *How Competitive Forces Shape Strategy.* Retrieved April 9, 2022 from Harvard Business Review: https://hbr.org/1979/03/how-competitive-forces-shape-strategy

Dun & bradstreet (n.d.) *Café Peace ApS*. Retrieved April 9, 2022, from https://www.dnb.com/business-directory/company-profiles.caf%C3%A9_peace_aps.4b993b6f3bb48c08160b6552138ad263.html

Dun & bradstreet (n.d.) *Restaurants And Other Eating Places Companies In Aalborg, Nordjylland, Denmark*. Retrieved April 9, 2022, from https://www.dnb.com/business-directory/company-information.restaurants_and_other_eating_places.dk.nordjylland.aalborg.html

Invest Northern Ireland (n.d.) *Choosing suppliers for your business*. Retrieved April 9, 2022, from https://www.nibusinessinfo.co.uk/content/choose-between-single-or-multiple-supplier-strategy

AB Catering (n.d.) *About AB Catering*. Retrieved April 9, 2022, from https://abcatering.dk/om-ab-catering/

Lawyers Denmark. (2021) *Costs for Starting a Business in Denmark*. Retrieved April 9, 2022, from https://www.lawyersdenmark.com/business-start-up-costs-in-denmark

Facebook. (2022) *Café Peace*. Retrieved May 24, 2022, from https://www.facebook.com/Cafe-Peace-383422368467049/?_rdr

Bloisi, W., Cook, C. W., Hunsaker, P. (2007). *Management and Organisational Behaviour* (2nd ed.). McGraw-Hill Education.

Duffy, M. (2018) A NEW MODEL FOR CAPTURING THE KEY ATTRIBUTES OF ORGANISATIONS AND DRIVING CHANGE. University of Sunderland.

Larman, C. (2004). Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development (3rd ed.). Prentice Hall.

Oracle (n.d.) Performance Extensions. Retrieved April 22, 2022, from https://docs.oracle.com/cd/E11882_01/java.112/e16548/oraperf.htm#JJDBC28752

Oracle. (1997). *Java Code Conventions*. Retrieved May 23, 2022, from https://www.oracle.com/technetwork/java/codeconventions-150003.pdf

## Appendix B. Problem statement

| **Problem statement – Table reservation system** | |
|---|---|
| **Student names** | Martin Glogovský, Petra Pastierová, Matej Rolko, Viktor Tindula |
| **Title (initial)** | 2nd Semester Project – Table reservation system |
| **Subject** | Café Peace is a popular restaurant situated in the centre of Aalborg. The café offers a wide range of meals, from breakfast and brunch to lunch and dinner. Additionally, people come here for coffee and dessert. During the weekends and holidays, the restaurant is remarkably busy especially around lunch and dinnertime. The café serves customers with reservations, but also takes walk-in customers if there is a table available. During the rush hours, the employees rely on table reservations made by the customers in advance, helping them to prepare the restaurant and figure out the number of walk-in parties they can take. |
| **Problem / Problem area** | In order to reserve a table in the restaurant, the customers can make a booking either in person, on the website or by phone. The staff is responsible for managing and grouping all table reservations, using a paper spreadsheet, which sometimes causes chaos when multiple employees participate. Even though the café offers table reservations through their website, problems occur regularly, since an employee is needed to manually confirm all reservations via email. Because of this, the reservations are made mainly through phone calls. Mistakes are made often, resulting in several wrong or unconfirmed reservations, causing difficulty for the staff and frustration within the customers. |
| **Problem statement** | How can we unify the reservations and make it easier and faster for customers and employees at the same time? |
| **Method / procedure** | This project is to be carried out using the *Unified Process*. The diagrams are to be created in *UMLet.* The system is to be programmed in *Java* programming language. The code is to be written in *Eclipse* IDE. The GUI is to be made in Java UX library *Swing*. For version control *Github* using *Git* is to be used. Data is to be stored in *Microsoft SQL* relation database. To manage the database *Microsoft SQL Server Management Studio* will be used. |

## Appendix C. Mock-ups

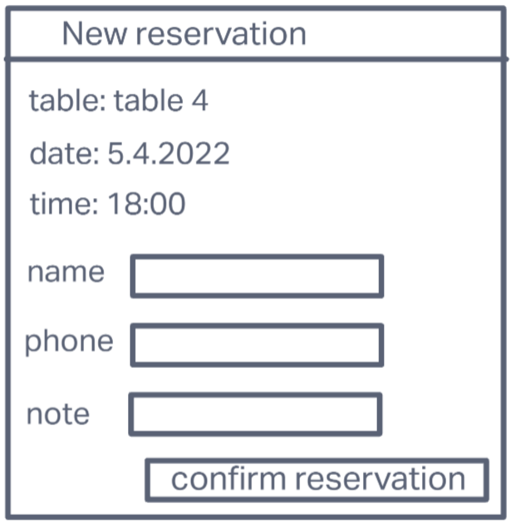
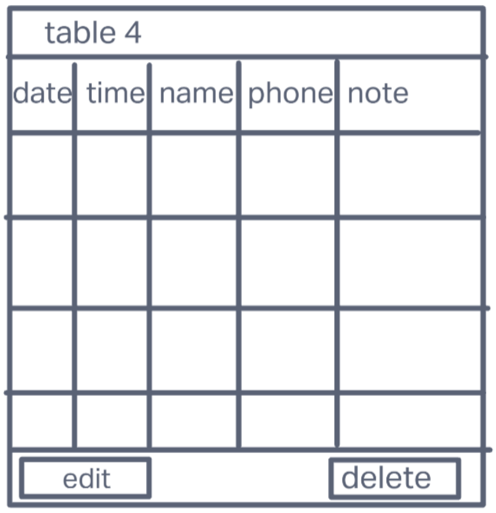


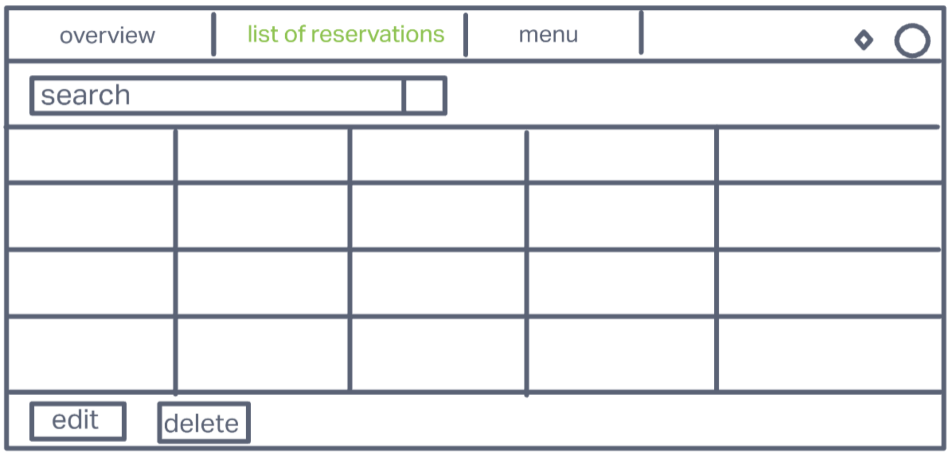
table: table 4

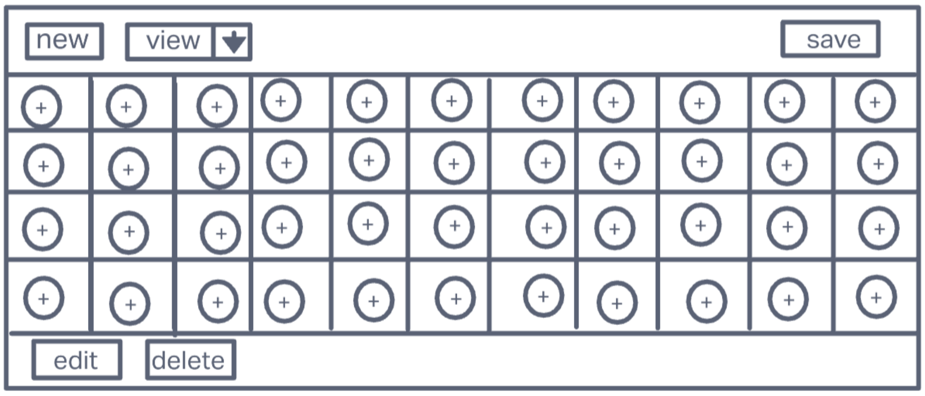
date: 5.4.2022

time: 18:00

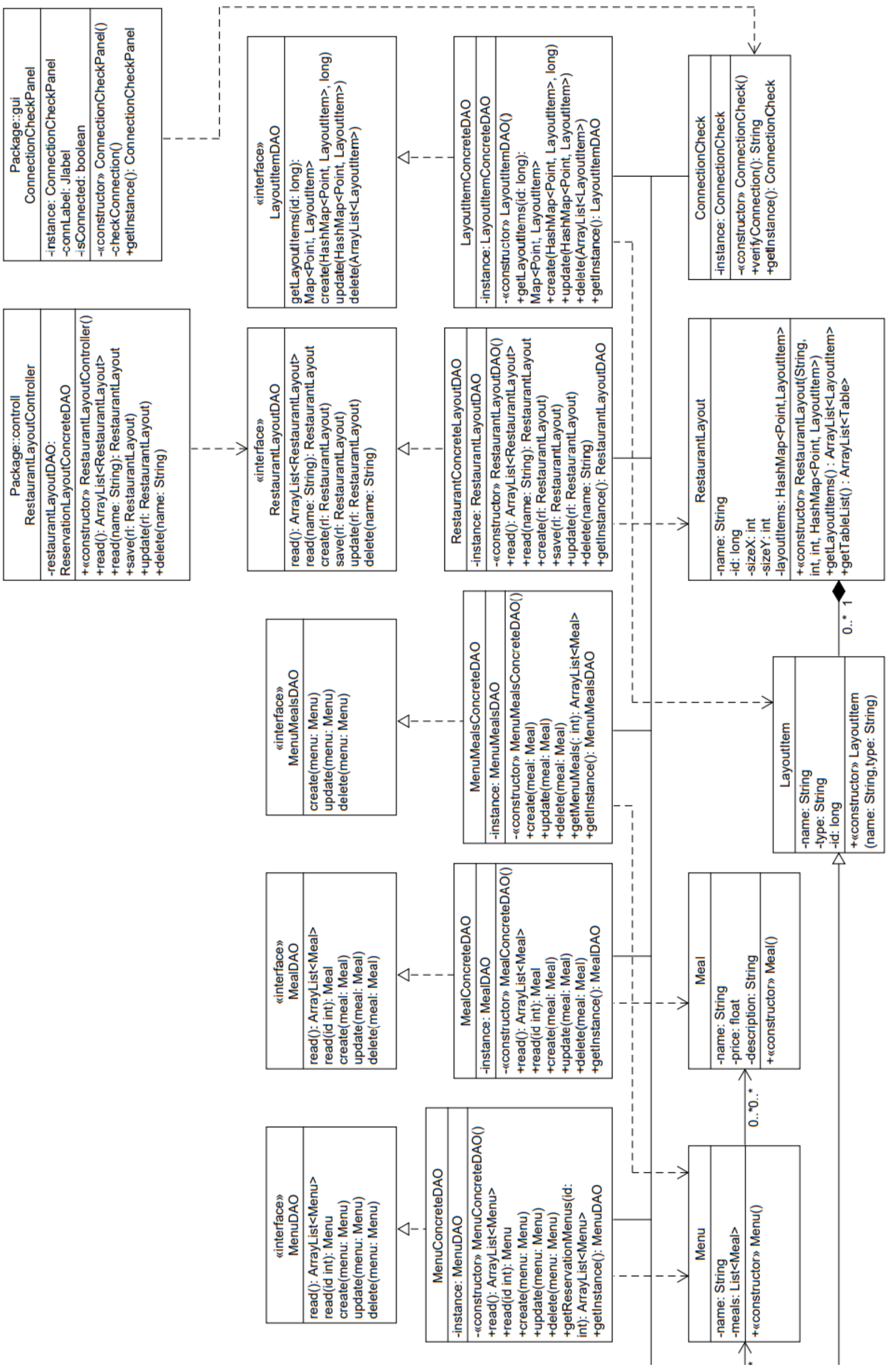


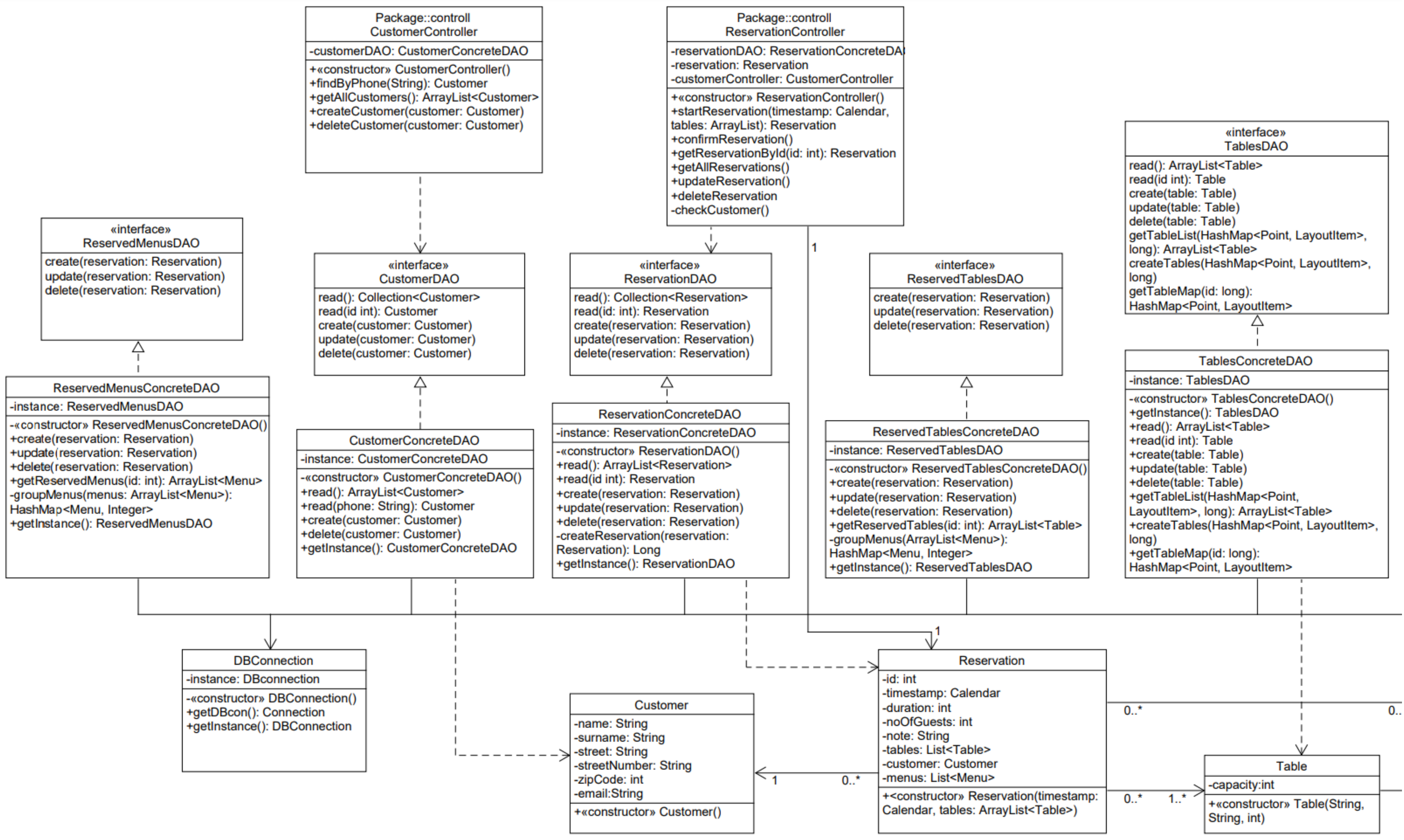
|  |  |  |  |  |
|---|---|---|---|---|
| date | time | name | phone | note |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |





# Appendix D. Design class diagram


Package::gui
ConnectionCheckPanel
-instance: ConnectionCheckPanel
-connLabel: Jlabel
-isConnected: boolean
-«constructor» ConnectionCheckPanel()
-checkConnection()
+getInstance(): ConnectionCheckPanel
Package::controll
RestaurantLayoutController
-restaurantLayoutDAO: ReservationLayoutConcreteDAO
+«constructor» RestaurantLayoutController()
+read(): ArrayList<RestaurantLayout>
+read(name: String): RestaurantLayout
+save(rl: RestaurantLayout)
+update(rl: RestaurantLayout)
+delete(name: String)
«interface»
LayoutItemDAO
getLayoutItems(id: long): Map<Point, LayoutItem>
create(HashMap<Point, LayoutItem>, long)
update(HashMap<Point, LayoutItem>)
delete(ArrayList<LayoutItem>)
LayoutItemConcreteDAO
-instance: LayoutItemConcreteDAO
-«constructor» LayoutItemDAO()
+getLayoutItems(id: long): Map<Point, LayoutItem>
+create(HashMap<Point, LayoutItem>, long)
+update(HashMap<Point, LayoutItem>)
+delete(ArrayList<LayoutItem>)
+getInstance(): LayoutItemDAO
ConnectionCheck
-instance: ConnectionCheck
-«constructor» ConnectionCheck()
+verifyConnection(): String
+getInstance(): ConnectionCheck
«interface»
RestaurantLayoutDAO
read(): ArrayList<RestaurantLayout>
read(name: String): RestaurantLayout
create(rl: RestaurantLayout)
save(rl: RestaurantLayout)
update(rl: RestaurantLayout)
delete(name: String)
RestaurantConcreteLayoutDAO
-instance: RestaurantLayoutDAO
-«constructor» RestaurantLayoutDAO()
+read(): ArrayList<RestaurantLayout>
+read(name: String): RestaurantLayout
+create(rl: RestaurantLayout)
+save(rl: RestaurantLayout)
+update(rl: RestaurantLayout)
+delete(name: String)
+getInstance(): RestaurantLayoutDAO
RestaurantLayout
-name: String
-id: long
-sizeX: int
-sizeY: int
-layoutItems: HashMap<Point,LayoutItem>
+«constructor» RestaurantLayout(String, int, int, HashMap<Point, LayoutItem>)
+getLayoutItems() : ArrayList<LayoutItem>
+getTableList() : ArrayList<Table>
0..*
1
«interface»
MenuMealsDAO
create(menu: Menu)
update(menu: Menu)
delete(menu: Menu)
MenuMealsConcreteDAO
-instance: MenuMealsDAO
-«constructor» MenuMealsConcreteDAO()
+create(meal: Meal)
+update(meal: Meal)
+delete(meal: Meal)
+getMenuMeals(: int): ArrayList<Meal>
+getInstance(): MenuMealsDAO
LayoutItem
-name: String
-type: String
-id: long
+«constructor» LayoutItem (name: String,type: String)
«interface»
MealDAO
read(): ArrayList<Meal>
read(id int): Meal
create(meal: Meal)
update(meal: Meal)
delete(meal: Meal)
MealConcreteDAO
-instance: MealDAO
-«constructor» MealConcreteDAO()
+read(): ArrayList<Meal>
+read(id int): Meal
+create(meal: Meal)
+update(meal: Meal)
+delete(meal: Meal)
+getInstance(): MealDAO
Meal
-name: String
-price: float
-description: String
+«constructor» Meal()
0..*0..*
«interface»
MenuDAO
read(): ArrayList<Menu>
read(id int): Menu
create(menu: Menu)
update(menu: Menu)
delete(menu: Menu)
MenuConcreteDAO
-instance: MenuDAO
-«constructor» MenuConcreteDAO()
+read(): ArrayList<Menu>
+read(id int): Menu
+create(menu: Menu)
+update(menu: Menu)
+delete(menu: Menu)
+getReservationMenus(id: int): ArrayList<Menu>
+getInstance(): MenuDAO
Menu
-name: String
-meals: List<Meal>
+«constructor» Menu()
*


Package::controll
CustomerController
-customerDAO: CustomerConcreteDAO
+«constructor» CustomerController()
+findByPhone(String): Customer
+getAllCustomers(): ArrayList<Customer>
+createCustomer(customer: Customer)
+deleteCustomer(customer: Customer)
Package::controll
ReservationController
-reservationDAO: ReservationConcreteDA
-reservation: Reservation
-customerController: CustomerController
+«constructor» ReservationController()
+startReservation(timestamp: Calendar, tables: ArrayList): Reservation
+confirmReservation()
+getReservationById(id: int): Reservation
+getAllReservations()
+updateReservation()
+deleteReservation
-checkCustomer()
«interface»
TablesDAO
read(): ArrayList<Table>
read(id int): Table
create(table: Table)
update(table: Table)
delete(table: Table)
getTableList(HashMap<Point, LayoutItem>, long): ArrayList<Table>
createTables(HashMap<Point, LayoutItem>, long)
getTableMap(id: long): HashMap<Point, LayoutItem>
«interface»
ReservedMenusDAO
create(reservation: Reservation)
update(reservation: Reservation)
delete(reservation: Reservation)
«interface»
CustomerDAO
read(): Collection<Customer>
read(id int): Customer
create(customer: Customer)
update(customer: Customer)
delete(customer: Customer)
«interface»
ReservationDAO
read(): Collection<Reservation>
read(id: int): Reservation
create(reservation: Reservation)
update(reservation: Reservation)
delete(reservation: Reservation)
«interface»
ReservedTablesDAO
create(reservation: Reservation)
update(reservation: Reservation)
delete(reservation: Reservation)
1
ReservedMenusConcreteDAO
-instance: ReservedMenusDAO
-«constructor» ReservedMenusConcreteDAO()
+create(reservation: Reservation)
+update(reservation: Reservation)
+delete(reservation: Reservation)
+getReservedMenus(id: int): ArrayList<Menu>
-groupMenus(menus: ArrayList<Menu>): HashMap<Menu, Integer>
+getInstance(): ReservedMenusDAO
CustomerConcreteDAO
-instance: CustomerConcreteDAO
-«constructor» CustomerConcreteDAO()
+read(): ArrayList<Customer>
+read(phone: String): Customer
+create(customer: Customer)
+delete(customer: Customer)
+getInstance(): CustomerConcreteDAO
ReservationConcreteDAO
-instance: ReservationConcreteDAO
-«constructor» ReservationDAO()
+read(): ArrayList<Reservation>
+read(id int): Reservation
+create(reservation: Reservation)
+update(reservation: Reservation)
+delete(reservation: Reservation)
-createReservation(reservation: Reservation): Long
+getInstance(): ReservationDAO
ReservedTablesConcreteDAO
-instance: ReservedTablesDAO
-«constructor» ReservedTablesConcreteDAO()
+create(reservation: Reservation)
+update(reservation: Reservation)
+delete(reservation: Reservation)
+getReservedTables(id: int): ArrayList<Table>
-groupMenus(ArrayList<Menu>): HashMap<Menu, Integer>
+getInstance(): ReservedTablesDAO
TablesConcreteDAO
-instance: TablesDAO
-«constructor» TablesConcreteDAO()
+getInstance(): TablesDAO
+read(): ArrayList<Table>
+read(id int): Table
+create(table: Table)
+update(table: Table)
+delete(table: Table)
+getTableList(HashMap<Point, LayoutItem>, long): ArrayList<Table>
+createTables(HashMap<Point, LayoutItem>, long)
+getTableMap(id: long): HashMap<Point, LayoutItem>
1
DBConnection
-instance: DBconnection
-«constructor» DBConnection()
+getDBcon(): Connection
+getInstance(): DBConnection
Reservation
-id: int
-timestamp: Calendar
-duration: int
-noOfGuests: int
-note: String
-tables: List<Table>
-customer: Customer
-menus: List<Menu>
+<constructor» Reservation(timestamp: Calendar, tables: ArrayList<Table>)
0..*
0..
Customer
-name: String
-surname: String
-street: String
-streetNumber: String
-zipCode: int
-email:String
+«constructor» Customer()
1
0..*
0..*
1..*
Table
-capacity:int
+«constructor» Table(String, String, int)

## Appendix E. Group Contract

University College of Northern Denmark

CSC-CSD-S212

May 25, 2022

2nd Semester Project

Group Cooperation Agreement

This 2nd Semester Project Group Cooperation Agreement dated May 25, 2022, is by and between the group members of Group 5 of CSC-CSD-S212. The agreement is designed to ensure a smooth cooperation during the Second Semester Project. It should increase the chances of success of the project. For this purpose, a number of rules were appointed which must be respected by all group members. Therefore, the group members agreed as follows:

1. Group members

(a) The Group consists of 4 members; Martin Glogovský, Petra Pastierová, Matej Rolko and Viktor Tindula.

2. Assignment details

(a) The deadline for this project and the date of hand-in is May 25, 2022 at 14:00.

(b) The supervisors of the assignment are Gianna Belle, Henrik Munk Hvarregaard and Jakob Farian Krarup.

3. Communication

(a) For sharing files and documents file-sharing platforms, such as GitHub and Microsoft Teams are used.

(b) For communication purposes online platforms, such as Discord and Facebook Messenger are used.

(c) Working time for major tasks is every workday between 8:30 a.m. to 4:00 p.m.

(d) Within collaboration hours, members are obligated to respond to messages within 15 minutes, unless given an appropriate reason not to.

(e) Members shall physically meet and collaborate on each day of the project unless given an appropriate reason not to.

(f) If members must complete their tasks outside collaboration hours, they shall update the rest of the group about the completion of said task. If they are certain they are unable to finish before the next collaboration session, they must inform the other members immediately.

4. Division of Work

(a) The members agree to split work between them in accordance with the project plan.

5. Absence from physical meetings and shortcoming of delegated tasks

(a) If a member does not show up to an agreed meeting, the supervisor is contacted unless their reason is decided to be sufficient by other group members present.

(b) If a member fails to complete their delegated task no later than 12 hours after its expected time, the next course of action is decided on by the other members and may be elevated to the supervisor.

(c) If a member's work is impeded by illness, or other sufficient reason, they are obligated to inform the others as soon as they are able to and shall make an effort to do partial work remotely, if at all possible. Their current tasks may then be delegated between all able-to-work group members.

(d) If a member's dedication to their tasks, work and collaboration efforts are not found satisfactory by the rest of the group, they may receive a strike. After receiving the previously agreed upon number of strikes (2), the situation may be elevated to the supervisor and further actions may be taken in collaboration with said supervisor.

6. Confirmation

(a) Submission of this form to the supervisor means that all members have agreed to the contents of this document and consider it binding.