REPORT
Computer Science 1st Year Project Exam - RoSys

LILLEBAELT ACADEMY
UNIVERSITY OF APPLIED SCIENCES

Group 1

Roxana Ion

Hedviga Arta

Jonas Laursen

Matthew Peterson

Table of Contents

Preface

The report is made as a part of participating in a project on the courses, System Development and Programming, in the first year of the Computer Science education at Erhvervsakademiet Lillebælt.

This document contains software development artifacts that were created during the sprints with descriptions, a description of our process and the way the group has been working with scrum.

The purpose of the report is to inform readers of the development of our project and the methods and tools we used to accomplish it.

Reading Guide

The report is based around American spelling and grammar. The project is split into two main parts, the report, which is described in the preface, and the appendix, containing extra materials referred to in the report.

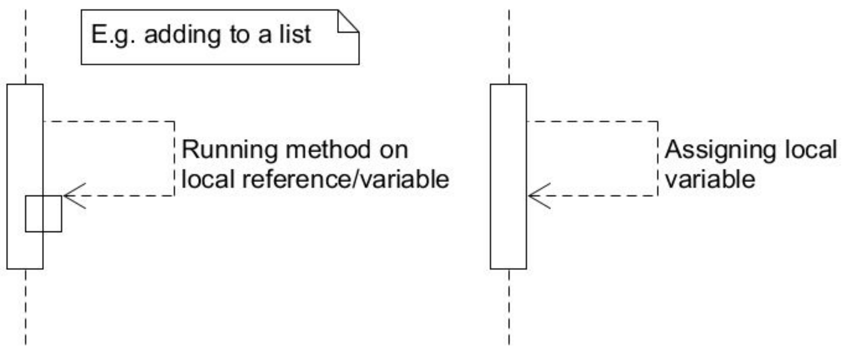
Words such as: us, we, group, etc. refer to the group as a whole.
Scrum Master refers to the person who was the Scrum Master for that week.

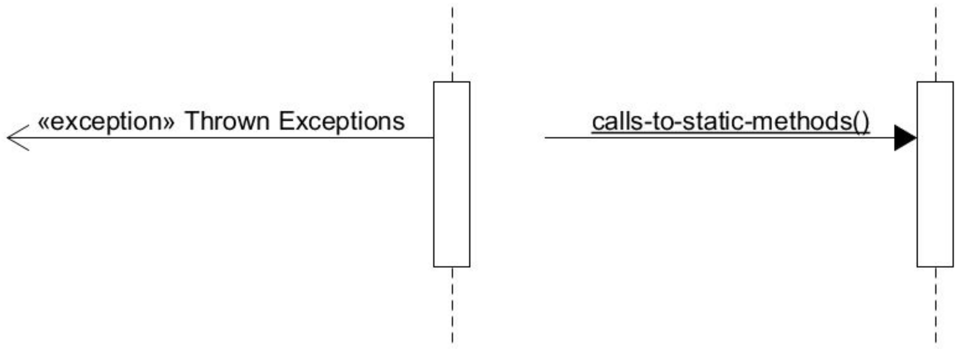
The word "Client" refers to both students and teachers.
The word "Users" refers to admins, students and teachers.

The program itself is named: Room Reservation System, and for short we named it RoSys (pronounced as roses).

Through this report, we have attempted to stick to a single use-case, which we have inserted the various diagrams for.

For our diagrams we used some non UML standard ways. You can see examples of these below:





Introduction

Erhvervsakademiet Lillebælt is an academy of higher education, which currently spans 2 campuses, one in Odense and one in Vejle.

EAL promotes project work highly, and hence has plenty of project rooms available for student use. Unfortunately, as we explain in our problem description, there are some issues with the assignment of project rooms.

The software we are developing is aimed to serve as a demonstration of how to better handle room reservations, and will, hopefully, be picked up for further development and integration into the school's system.

Ethics

Roles

Each member was assigned areas of main responsibility, to help facilitate task assignment. These areas also reflect which sections of the report was done by whom. Mainly, each member worked on Code, GUI, Tests, Database, Quality Assurance, Diagrams, Use cases, Design Class Diagram, Normalization, SSD, SD and Report.

Matthew Peterson
Implemented tests
Business Case
Operation Contracts

Hedviga Arta
Implemented GUI
Business Model
Personas

Roxana Ion
Implemented GUI
BPMN
Object Model

Jonas Laursen
Implemented tests
Scenarios
Domain Model

Tools and Methods

This section covers the basic tools and methodologies we have used in our project, as well as a description of why this tool or method was selected.

Microsoft Visual Studio

We chose to use Visual Studio because it is the tool we are most familiar with and it works well for the programming language C# and the markup language XAML.

Microsoft SQL Server

We decided to use a database, rather than a flat file system, to allow better integration with other systems. We are using SQL because a free Microsoft database was provided to us by the school, and is what we have been taught in our programming course, so we feel most confident with this.
The database will serve as our persistent storage. To interface with the database, we will be using Microsoft's SQL Server Management Studio, as this is the program we also used in class.

MeisterTask

We used MeisterTask to facilitate the organization of the development of our project. MeisterTask also served as our SCRUM Board. We chose to use this tool because it contains lots of useful features such as time tracking, task assignments and because it is user-friendly.

GitHub

To help ensure proper backup and logging of our activity, we decided to use Git for version control, and GitHub for remote storage.

GitHub is a very popular tool among developers, and is widely supported, which is also why we decided to use it.

More information about how we used Git and Github can be found in the appendix Group Contract.

Google Drive

We chose to use Google Drive for storing our report and some other shared files, because the cooperation features of Google Drive are superior to other options, such as OneDrive.

Facebook

For the communication part of our project, we chose Facebook chat because it is more practical and everyone has easy access to it.

P.O. Communication

The communication between us and our Project Owner happened on a weekly basis, in the form of face-to-face and Skype meetings. Due to his busy schedule, we did not have a specific day for the meetings, which meant that we decided on our next meeting date at the end of each meeting.

The meetings were for updating our progress, finalizing the next week's sprint, and ensure our visions align.

SCRUM

Our sprints will be 1 week in length, every Wednesday we will end one sprint and begin a new one. We will have a retrospective at the end of every sprint, and we will begin each day by having a sit-down meeting. We will switch scrum masters every sprint.

SOLID

To ensure our program can be developed in high quality, we employ the SOLID principles.

TDD

Another way of ensuring that we wrote quality software from the beginning is by using TDD practices. By using TDD we assured that our code and the business requirements parallel each other, also TDD helps us to adhere to the SOLID Principles.

Business Analysis

Business Model

In order to understand our product owner's company better, we created a business model that represents a stable description of the company. We acquired all the information from Lillebælt Academy's website and from the various documents found on the website. Also, in collaboration with our product owner, we were able to receive valuable information regarding the company.

Key Partners	***Key Activities***	***Value Proposition***	***Customer Relationships***	***Customer Segments***
- Other universities (for exchange programs, international experience) - Businesses (for lectures, events, internships) - Students - Teachers - Staff - Agents - FabLab - Odense and Vejle Municipalities - Government	- Teaching students - Organizing events and guest lectures. - Erasmus and other exchange programs. - Internships	- Improving knowledge by teaching students and offering them a nice study environment.	- Academy provides students with counseling. - Customers and all who are interested can call or get to information to get personal assistance. - Customers can also get offers by webpage or web shop. -Teacher assistance - Jobportal for offering student jobs.	- People who want to learn and take studies. - Students from other countries or schools. - Business owners to get new workers. - Other schools or educational places to get students.
	Key Resource		***Channels***	
	- Campuses - Equipment - Teachers knowledge - Staff - Government		- Lillebaelt Academy University of Applied Sciences creates awareness of its services by agents or other educational institutions, webpage and by working together with local businesses. - Lillebaelt Academy University of Applied Sciences assesses their quality by making questionnaires and having surveys from government's side and also analyzes.	

Cost Structures	***Revenue***
- Equipment - Salaries - Events - Clothing line	- Municipality/Governments funds - Lillebaelt Online Shop - Non-EU International students

Business Case

In our business case we looked at the various effects our system would have on the school and discussed the pros and cons of these effects. (Our full business case can be found in the Business Case section of the appendix.)

Problem Description

Project rooms are often reserved multiple weeks ahead (especially during the project weeks), yet rooms are often left empty. People reserve rooms and do not use them, or only use them for part of the time.

This is quite an annoyance to students, as when they request a room from the reception, their request cannot be fulfilled, and the students then must walk around the school, searching for rooms not being used.

Students have also experienced that while attempting to reserve a room via email, another group of students went to the reception and were assigned the room, even though the email had already been sent and received. Hence, the students who had sent the email had to give up the room to those students who went to the reception.

The problem occurs on an almost daily basis, and is most prominent during project weeks and close to end of semesters, due to exam work.

Main Solution

We were initially introduced to our problem during a previous project, named "Project EAL Campus", where we had to investigate our school and find something we considered to be an issue.

During the exhibition of the previous project, we found many other students had also picked up on the project room issues, therefore, we decided to investigate this issue further.

Our main solution is to develop a program to manage the reservations, rather than having people sit and slot people into calendars. It will have CRUD operations for reservations, users and rooms. It will also send notifications to users about their reservation and require them to confirm the reservation when they get to the room. We would also like it to have a queueing system.

This was the best solution to the problem because it would not be expensive for the school to implement, and it has a very quick response time compared to our other solutions. The only possible cost to the school would be paying employees to integrate our system into their existing system.

FURPS+

Before meeting with our product owner, we looked at the materials we had gathered already and set up what we believed to be the optimal FURPS+. During our meeting, we showed him, and he agreed. Our FURPS can be seen below.

We used these FURPS requirements when taking tasks, and to help align our product owners expectations with what we would be able to deliver.

Performance is a main criterion for our program. RoSys should work quickly and efficiently. The focus also was aimed towards reliability. The core part of the program is meant to run on a server, thus, should never be shut down. In case of unexpected shutdowns, for example, server crash or windows update, we wanted the program to return to its previous state. The program should also be easy to integrate into other systems, hence we attempted to follow SOLID for improved supportability. However, since we will be handing over the full source code of our project, it was not highest priority.

To allow others to more easily read and change our program, we wrote into our group contract, various rules, to help adhere to a more linear naming scheme.

Functionality – The system must require the user to check into their room when it is time for them to use it, otherwise, the reservation will be canceled. Users should be able to reserve rooms, cancel reservations and view their reservations. The system should alert users when they have a reservation soon. There must be different user types that have different privileges.

Usability – The user interface is not the priority for the project because we plan on integrating it into other systems that already have user interfaces. We will, however, create a user interface to demo it. The program should be intuitive and easy to use. Users should be able to do what they need to do in a very short amount of time.

Reliability – The system should never have problems reserving rooms unless the room is not in the system, then it will need to be added by an admin user.

Performance – The program is not very complex, so it will not require much processing power. There should be little wait time while using the system.

Supportability – The system should be created using TDD. The system will need to be updated if room numbers change, rooms are added or removed or information about a room changes. These things will need to be updated by an admin user.

Design Constraints – The system must be able to integrate with other systems. The system must use a relational database, WPF user interface, C# and TDD.

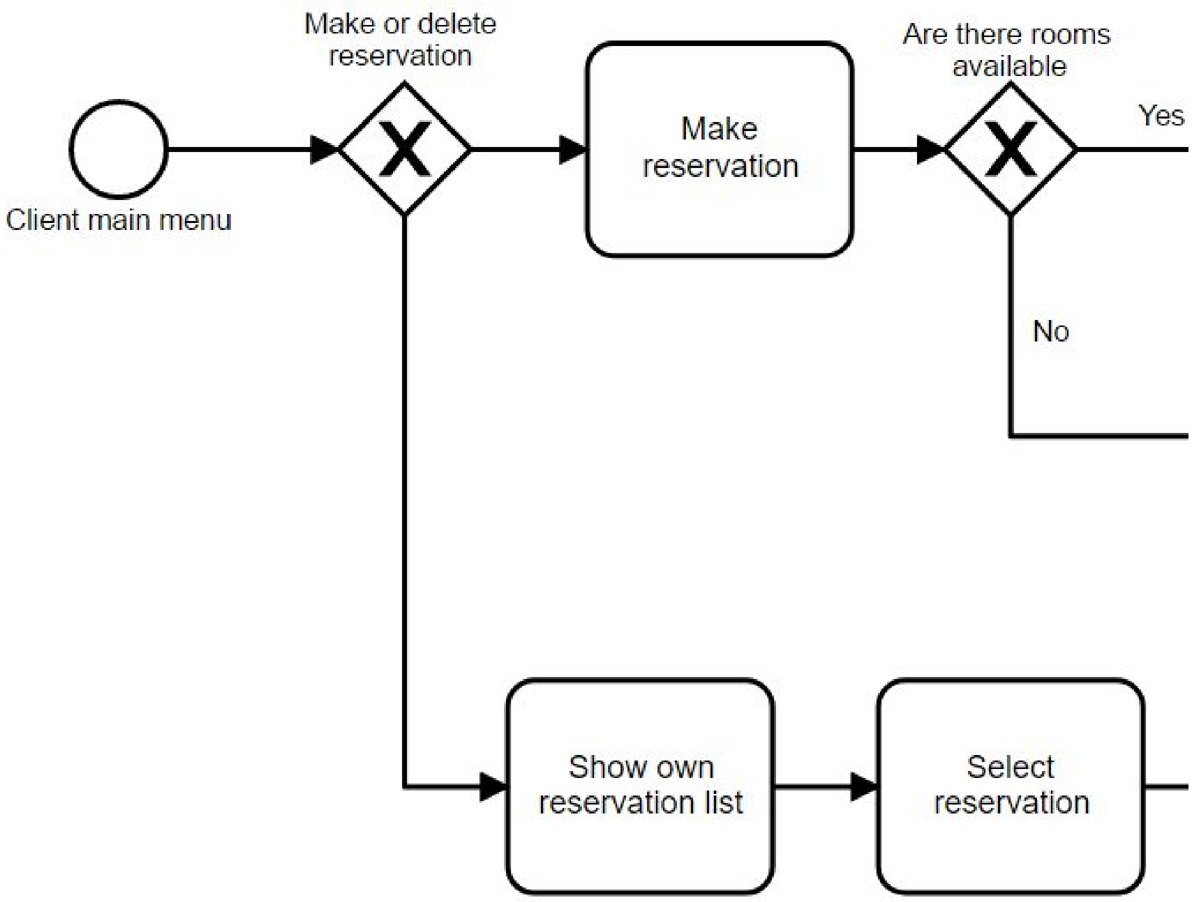
Implementation Requirements – The programmers must adhere to the standards set out in the group contract. TDD must be used to guarantee that everything works correctly.

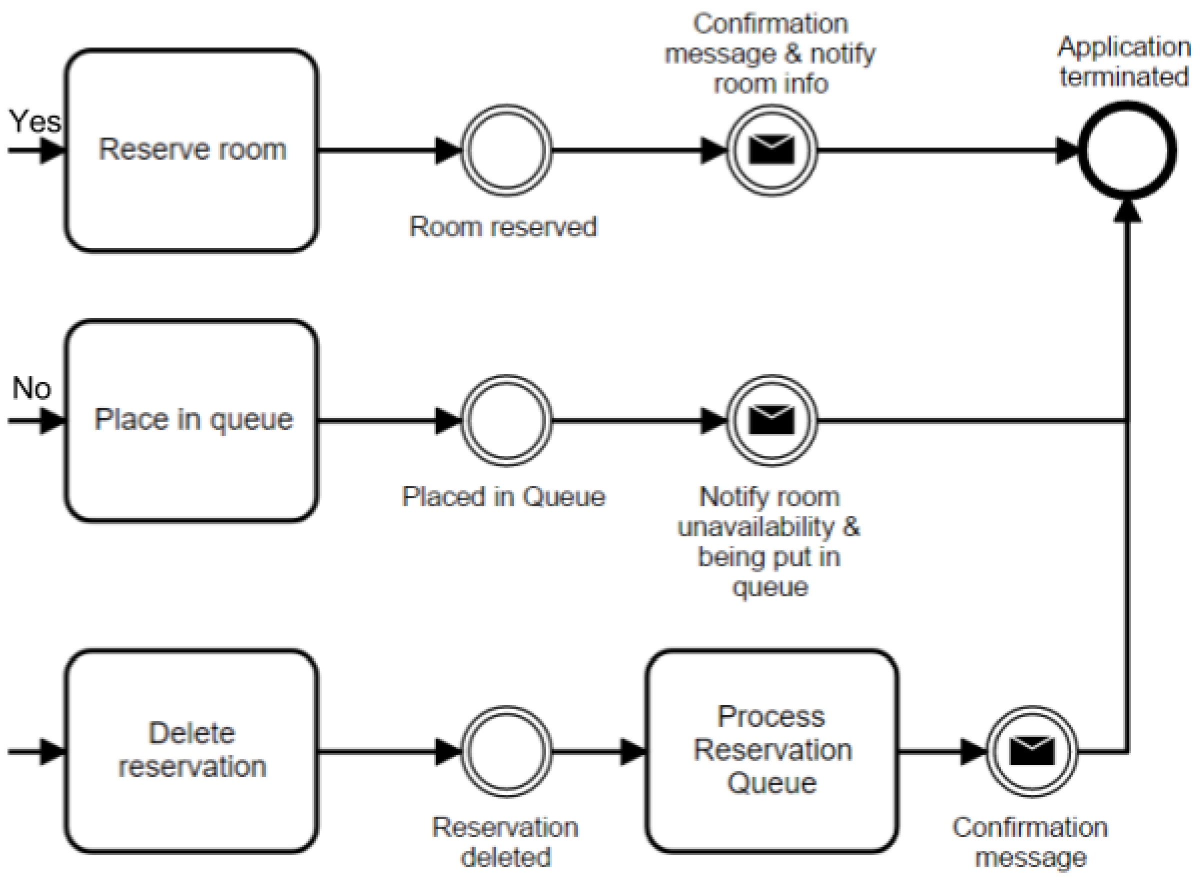
Interface Requirements – It must interact with any system the project owner wants to use it with.

Physical Requirements – It should be able to run on any PC.

BPMNs

In order to illustrate and get a better understanding of our program, the flow and the multiple processes it's going through, we have created BPMNs. Below is our BPMN for Clients, which we have split in two parts (due to readability). We also have created BPMNs for Admin and Queue System, which can be found in Appendix ???.


Make or delete reservation
Client main menu
Make reservation
Are there rooms available
Yes
No
Show own reservation list
Select reservation



Systems Artifacts

Use Cases

In this project, we have created five use cases that reflect the interaction between our actors and the system, as well as the requirements needed, the alternate flows and the outcome.

These are our five use cases:

1. Client reserves a room
2. Client deletes own reservation
3. Admin reserves a room
4. Admin deletes a reservation
5. Notification system

While working with SCRUM, our work was based on iterative cycles (sprints), with a length of one week. For each sprint, we have decided that we were going to set and fulfill one Use Case. That specific use case would serve as a guide meant to help us on splitting and prioritizing tasks in order to fulfill the requirements needed.

As mentioned before in our Reading Guide, for this report, we are going to focus mainly on only one use case, which is going to be the "Client reserves a room" use case. This way we make sure that our report follows a logical progression and also covers all the topics we learned about throughout the year.

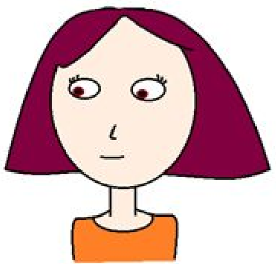
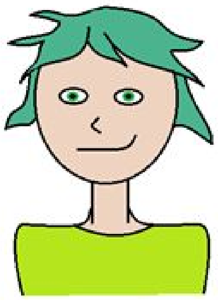
As the name suggests, the use case “Client reserves a room” is responsible for the process of students and teachers reserving

(Our other use cases can be found in Appendix ???)

Use case name	Client reserves a room
Use case description	A client wishes to reserve a room from the system
Actor	Client (Students & Teachers)
Pre-conditions	None
Basic Flow (Success Scenario)	1. Client enters reservation system 2. Client selects date, time frame and amount of people 3. System assigns the client a room 4. Notify client which room he’s been assigned
Post-conditions	Reservation has been created in repository, in rooms and users lists and in the database
Alternate Flow	AF1: No rooms available 3. System cannot assign a room 4. Put reservation in que 5.System throws exception NoRoomsAvailableException 6. Notify user that no rooms are available and is placed in que

Personas

To understand the way RoSys works and how it would be used we had to know who would be using it. We made student personas, admin persona and teacher persona but in the report, we have included only student personas for “Booking a project OR meeting room” scenario uses them. Also as students will be using our system more than admins or teachers, we decided to make two student personas. Other personas can be found in Appendix ****** .

 Sarah	Sarah is 22 years old. She is Filipino. Her mother is half Danish and they both moved to Denmark 2 years ago. Sarah is very interested in cultures and languages and learns Danish easily. She studies service, hospitality and tourism management at Erhvervsakademiet Lillebælt.	As a student, she needs to do project work with her group. In order to concentrate, she and her group need to find a quiet place to work.
 Cameron	Cameron is a very energetic and imaginative person. He is 25 and likes math and physics. He is studying energy technology. To study hard he needs peace and quiet.	Cameron needs project rooms to work with his group.

Scenarios

This scenario has two flows: if there is free room or if there isn't. We needed to know how would system interact with user and what kind of messages we would need to make for user to understand everything. Other scenarios can be found in Appendix *****.

Booking a room scenario:

A room is available:

Sarah comes to school, tuesday morning at around 8 am. She meets up with her group, ready to work for today.

Sarah opens her phone and enters the EAL App. She navigates to the section for project rooms.

Sarah's group has 4 members, including herself. She selects the option to book a new room, and fills in the form; number of people, time period, and date. She finishes off by clicking "Reserve room". After a few seconds, another message appears, explaining which room she has been assigned to.

No room is available:

Cameron is sitting in school for normal class. Their teacher presents them a new project and assigns everyone a group. Cameron's group meets together, and they decide they would like to start their project work tomorrow and therefore need a room.

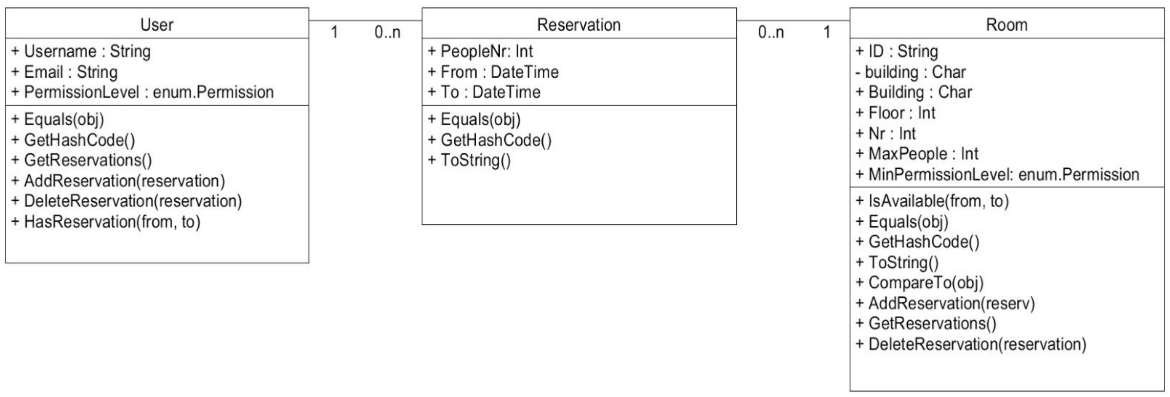
Cameron opens the EAL App, and navigates to the section for project rooms, selecting the option to book a new one.

Camerons group has 14 members, so he fills in 14 in the option for group members, and selects the time-frame: 8:00 to 16:00 the next day. He finishes off by clicking "Reserve room".

A few seconds later, another message explaining that no rooms are available in the selected time frame. A message appears on his screen, informing him that the request is being processed.

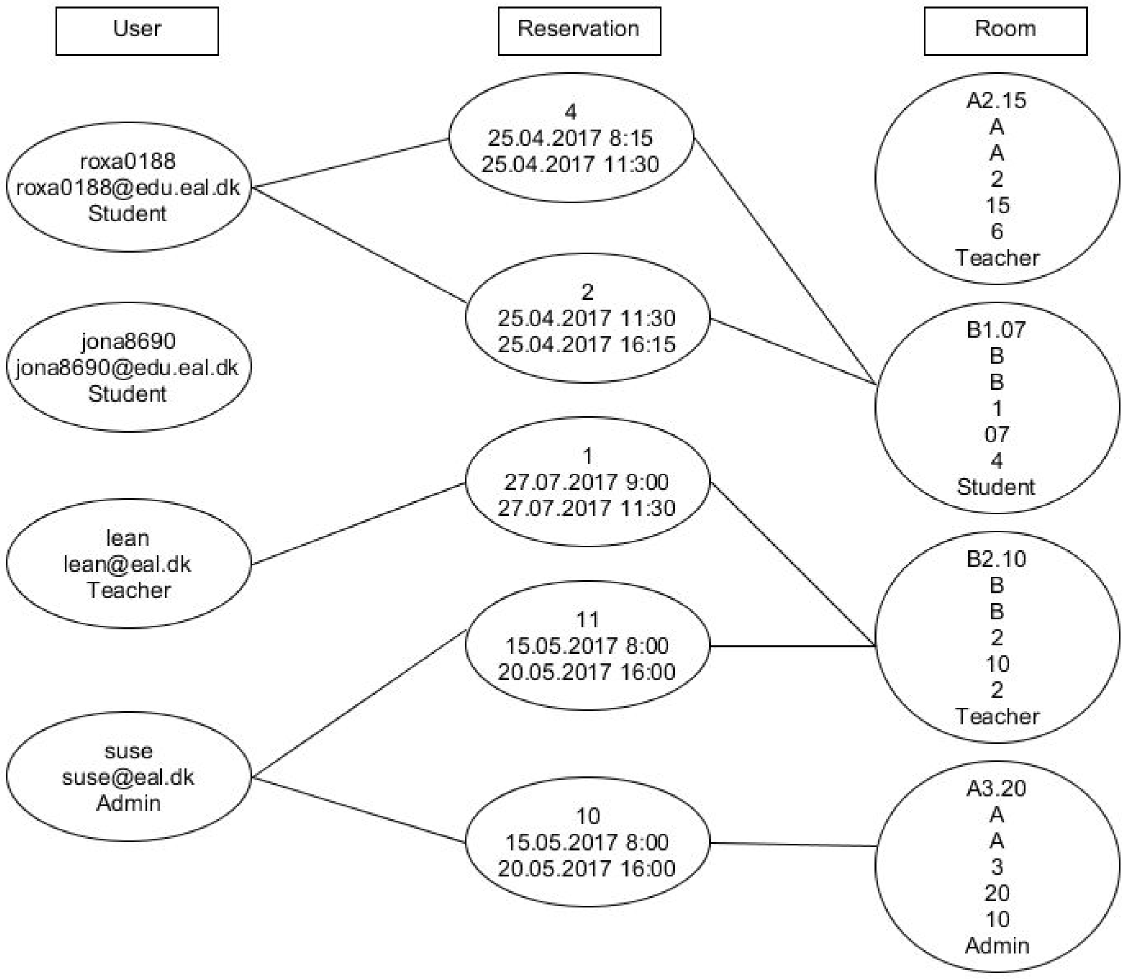
Domain Model

Our domain is pretty small. At the beginning of the project we were afraid that it would be too small. We have three conceptual classes: User, Reservation and Room. Users represent all possible people that can use our system, they will be divided by permission level, which is enum: Student, Teacher, Admin. Rooms also have permission, to reserve room user has to have the same level permission or higher. The Reservation is linking both Room and User together, and also it has datetimes that shows period of time in which reservation will be active as well as number of people that are going to use the room. Reservation can have only one room and only one user, but rooms can have multiple reservations at different times. Users can have many reservations at the same time, if admin has reserved it, for example. The example data will be shoved in object model and database later.



Object Model

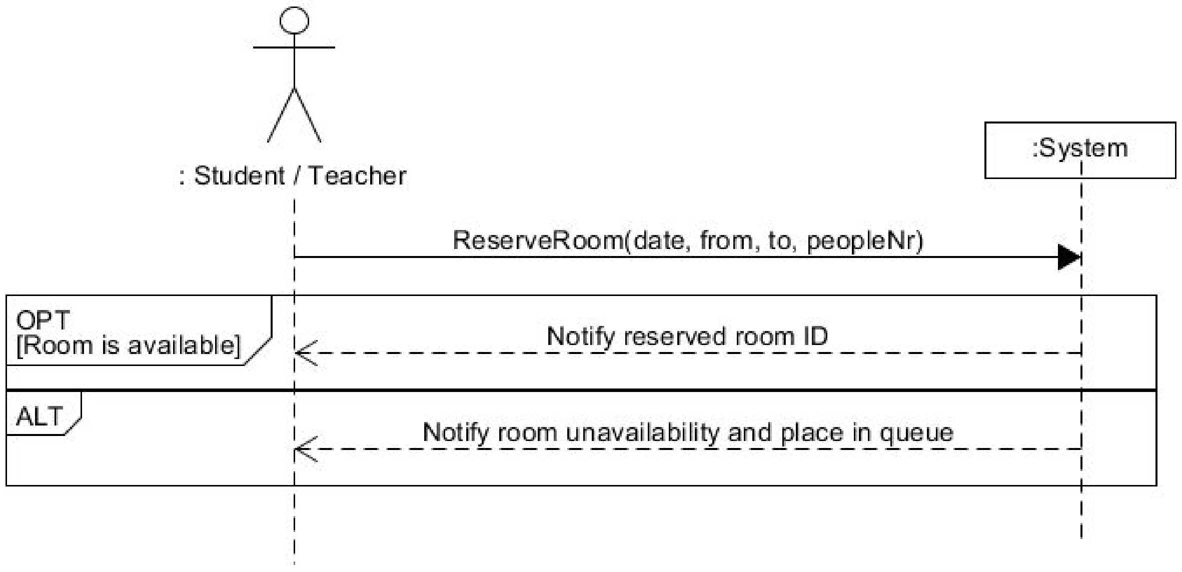
To make sure that our domain is correct and that we do understand with what kind of data we will need to deal with we made an object model. By using object model, it becomes easier for us to follow the connections and also to check whether those connections make sense between our domain classes.



Design Class Diagram

System Sequence Diagrams

To see what user would need to input and what he would receive we made multiple system sequence diagrams. As we have graphical user interface and MVVM pattern the user interacts with View part of our coude.


: Student / Teacher
:System
ReserveRoom(date, from, to, peopleNr)
OPT
[Room is available]
Notify reserved room ID
ALT
Notify room unavailability and place in queue

Operation Contracts

This is the operation contract for when an admin reserves a room in our system. We also have an operation contract for when a client reserves a room, but it is very similar to this. The rest of our operation contracts can be found in the Operation Contract section of the appendix.

Contract C01:

Operation:
 - ReserveRoom(date : string, from : string, to : string, room : IRoom, username : string)

Cross References:
 - SSD: SSD_Reserve
 - UC: Admin reserves a room

Preconditions:
 - No preconditions

Postconditions:
 - A Reservation was created and stored in the repository,
 - A Reservation was associated with a room and user

We made these operation contracts in order to clarify exactly what changes need to be made to the system after this operation has been completed. We also made it so that we had a reference to see if the operation was working the way we intended it to.

Sequence Diagrams

Tests

Implementation

GUI

Database

When we started working on our persistent storage it was clear that using a Database would be an optimal solution. Having a database allows other programs to very easily read data from our program, without having any relation to it. The school has provided us with Microsoft SQL Servers, which we used, since this is what we are familiar with from our class work.

Processing speed was the main concern of ours, so we did not wish to query the database for everything, and keep our objects in our repositories, so our program could easily get these from ram. Hence our decision to use the database as a "back-up" incase the program or server would crash, or otherwise need to be shut down.

Changes in the program are put into the database via stored procedures stored on the SQL server.

Furthermore, as integration is a main concern to us, we thought it would be great if our program could also process changes that happens to our database. We discussed this idea with our project owner, which he seemed to like.

The Change table

Tracking changes to the database was a fairly easy and straightforward process, after we discovered the power of triggers. For each table, we created three triggers, one for insert, update and delete. The pattern for each are the same. Our full SQL for the triggers can be found in our Appendix under "Database Triggers".

```
1.  CREATE TRIGGER  trgInsertReservations ON Reservations
2.  FOR INSERT AS
3.        declare @command   int;
4.        declare @table      NVarChar(max);
5.        declare @PK        NVarChar(max);
6.
7.        declare @ID        NVarChar(max);
8.        declare @DateTo    NVarChar(max);
9.        declare @DateFrom   NVarChar(max);
10.       declare @Username   NVarChar(max);
11.
12.       set @command =  0;
13.       set @table = 'Reservations';
14.       select @ID = i.ID from inserted i;
15.       select @DateTo = i.DateTo from inserted i;
16.       select @DateFrom = i.DateFrom from inserted i;
17.       select @Username = i.Username from inserted i;
18.       select @PK = @ID + ';' + @DateTo + ';' + @DateFrom + ';' +
    @Username;
19.
20.       INSERT INTO Change(Command, TableName, PrimaryKey) VALUES
21.       (@command, @table, @PK);
22.
23.       PRINT 'Trigger: Insert Reservation'
24. GO
```

The way all work, is simple. Each type of query are given an integer representation. We chose to put insert as 0, update as 1, and delete as 2. Each time a query is run on the tables: Rooms, Users or Reservations, a trigger is also executed, storing information about which command it was (line 11), which table (line 12), and the primary key of the row that was "changed".

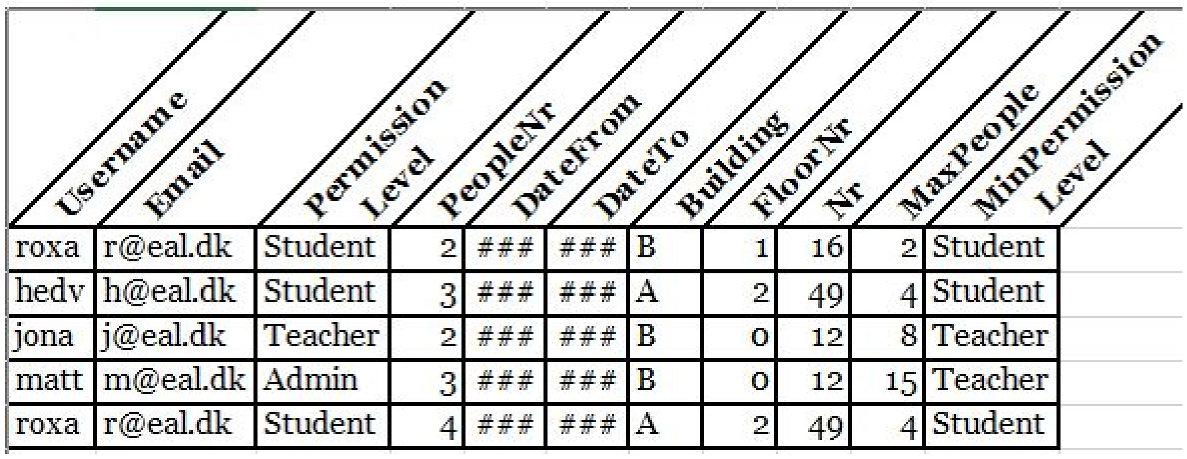
In our program, we then have a thread running, that reads this change table, and processes these changes into our program.

We started encountering various issues with this approach, mainly the issue of our program saving changes to the database. Shortly after having implemented the change table, we started seeing objects being duplicated in our program. This was because our own queries were also executing the triggers, which gave us quite a headache to deal with.

Initially, we tried to put an Identifier key in the change table that would be set, depending on if the query was by us, or external. This proved to be just as big a challenge, as we could not always properly update this identifier key, and after almost dropping the feature, we found out we could enable / disable the triggers on the go. We updated our stored procedures to simply disable the respective triggers before the query, and re-enable them again afterwards. This approach was very successful, and we managed to finish off this feature.

Normalization

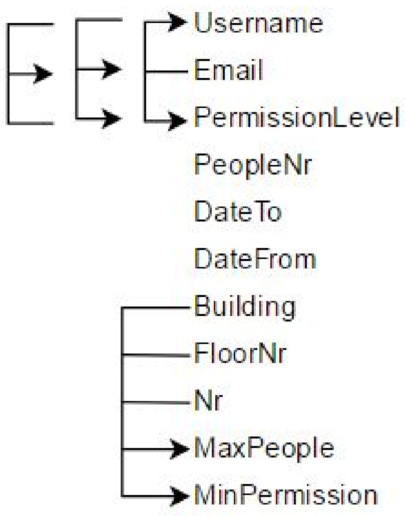
This is our reservations relation RESERVATION(Username, Email, PermissionLevel, PeopleNr, DateFrom, DateTo, Building, FloorNr, Nr, MaxPeople, MinPermissionLevel).
In an unnormalized form, our data structure looks like this:



Username	Email	Permission Level	PeopleNr	DateFrom	DateTo	Building	FloorNr	Nr	MaxPeople	MinPermission Level
roxa	r@eal.dk	Student	2	###	###	B	1	16	2	Student
hedv	h@eal.dk	Student	3	###	###	A	2	49	4	Student
jona	j@eal.dk	Teacher	2	###	###	B	0	12	8	Teacher
matt	m@eal.dk	Admin	3	###	###	B	0	12	15	Teacher
roxa	r@eal.dk	Student	4	###	###	A	2	49	4	Student

From & To being date-time stamps, e.g "07/04/2017 09:30:00", but we could not show this here, due to size constraints.

First we determined the candidate keys from the unnormalized Reservation table.
As for one room at specific time can be only one reservation we have this candidate key: (Building, FloorNr, Nr, DateTo, DateFrom).



Then we determined the functional dependencies:

Email -> (Username, PermissionLevel);
Username -> (Email, PermissionLevel);
(Username, PermissionLevel) ->Email;
(Building, FloorNr, Nr) -> (MaxPeople, MinPermissionLevel).

Since we have determinants that are not the candidate key, we needed to put them into a separate tables:
USERS(Username, Email, PermissionLevel), where Username is primary key,
ROOMS(Building, FloorNr, Nr, MaxPeople, MinPermissionLevel), where Building, FloorNr, Nr are the composite key.

We decided that it makes no sense for us to put PremissionLevel and Email in different table, we are not doing that, because it would unnecessarily complicate the user part in our database.

Final Database Schema

* Primary Key
- Foreign Key

Users	
Colum	**Type**
* Username	NVarChar (Primary)
Email	NVarChar
PermissionLevel	int

Rooms	
Column	**Type**
* Building	Char (Primary)
* FloorNr	Int (Primary)
* Nr	Int (Primary)
MinPermissionLevel	int
MaxPeople	int

Reservations	
Column	**Type**
* ID	Int (Primary / Surrogate)
- Username	NVarChar (Foreign / Users)
- Building	Char
- FloorNr	int
- Nr	int
From	DateTime2
To	DateTime2

People	int

Change	
Column	**Type**
* ID	int
Command	int
TableName	NVarChar
PrimaryKey	NVarCar

Abstracting the Connection

When we started working with the Database, one of the things we realised was how often we had to open and close the connection. We wished to create multiple files for the various domain resources (users, rooms & reservations), to keep them separate.

This initially meant we had some duplicate code for the opening and closing of the database. We decided to put these in an abstract class, that the individual classes would then implement. The decision to have the Database class abstract, was because it would have no use on its own, and did not want this to be confused.

```
using System.Data.SqlClient;
using System.Threading;

namespace DAL
{
        public abstract class Database
        {
        private static string _connInfo = DatabaseConn.ConnString;
        private static SqlConnection _conn;
        protected static object locked = new object();

        protected SqlConnection OpenConnection()
        {
                Monitor.Enter(locked);
                if (_conn == null)
                {
                _conn = new SqlConnection(_connInfo);
                }

                _conn.Open();
                return _conn;
        }

        protected void CloseConnection()
        {
                _conn.Close();
                Monitor.Exit(locked);
        }
        }
}
```

Later in the project, we also implemented a factory for the connection string, which we describe more on later. Having the open and close methods in this abstract class made implementing a variating connection string much easier.

Design Patterns

Design patterns can be seen as blueprints, on how to achieve certain niche features for your code. They are also used to assist programmers, across platforms and languages, in communicating how aspects of their code works. One design pattern can be implemented in many different ways depending on the language, but as long as it fulfills the description, it can be named so.

In our program, we implemented 2 common design patterns, namely singletons and a factory.

Singletons - Just a single instance

Our project mainly uses repositories to keep our objects in memory. This meant that all parts of the program needed the same instances of the repositories.

A common way this could have been done, would be via dependency injection, where you would simply instantiate a the class, and by constructors or other methods inject the references into the other classes.
We felt this could potentially cause issues, as it would require a lot of work to go into making sure the reference is kept passed around.

Hence, we used singletons. Singleton is basicly a pattern, that when followed only allows a single instance of an object to be created. It requires you to close down the constructor, so it can only be instantiated by the class itself. Then via a static property, you can fetch the instance, or create it if not yet created.

A full copy of the reservation repository can be found in the Appendix under Code > Core > ReservationRepository. A snippet is inserted below, to showcase the singleton.

```
private static ReservationRepository _instance = new ReservationRepository();
public static ReservationRepository Instance { get { return _instance; } }
private ReservationRepository() { }
```

Singleton Pattern used in Reservation Repository

These 3 lines combined, only allows a single instance of the repository to be used. This ensures that no other copies of the repository can exist.

As the _instance is static, the object is created as the program is launched, and is then fetched via the property Instance. The private constructor enforces that no other instances can be created, other than by the class itself.

Factory - Switching on the fly

When we started running tests on our database, we quickly encountered an issue, that when testing was complete, our databases would have been cleared, and we had to re-create rooms in the program to showcase the features.

We agreed that running the tests against it's own database would solve this issue. At this point, the connection string was hard-coded into the abstract Database class. Changing the string was easy, to connect to a different database, but having the program doing in runtime, would require a factory.

We implemented a factory, by having multiple classes implement an interface, and then instantiating the different classes depending on a system-set variable. Our factory can be seen below.

```
using System;

namespace DAL
{
        internal interface ConnInfo { string ConnString { get; } }

        public static class DatabaseConn
        {
        public static int systemEnviroment = 0;
        private static ConnInfo conn;
        public static string ConnString
        {
                get
                {
                switch (systemEnviroment)
                {
                        default: throw new IndexOutOfRangeException();
                        case 0: conn = new ConnProd(); break;
                        case 1: conn = new ConnTest(); break;
                }

                return conn.ConnString;
                }
        }
        }

        internal class ConnProd : ConnInfo
        {
        private const string CONNSTRING = @"Server=ealdb1.eal.local; Database=ejl73_db; User Id=ejl73_usr; Password=Baz1nga73";
        public string ConnString { get { return CONNSTRING; } }
        }

        internal class ConnTest : ConnInfo
        {
        private const string CONNSTRING = @"Server=ealdb1.eal.local; Database=ejl73_db; User Id=ejl73_usr; Password=Baz1nga73";
        public string ConnString { get { return CONNSTRING; } }
        }
}

```

We decided to put the whole factory in one file, as it was because of how closely related the classes where, and since they were all so small, having it together did not cause confusion.

Now, to execute our tests, on another database we would simply have to change the systemEnviroment int. In the future, other connection strings can easily be added to support other states of the program. For example, if a seperate database should be used for development.

Conclusion

Perspectivation

Process Evaluation

Appendix

Table of Contents

Program License

We decided upon using the MIT License, since it seems to have the best option in case someone wishes to continue working upon our project, they can do so, even for commercial purposes.

The MIT license is an “AS IS” license, so we cannot be held responsible for how our software is used by others, and how bugs in our software might affect them.

Use Cases

Use case name	Client deletes own reservation
Use case description	A client wishes to delete a reservation from the system
Actor	Client (Students & Teachers)
Pre-conditions	Client has reservation
Basic Flow (Success Scenario)	1. Client enters reservation system 2. Client requests the system to display his reservations list 3. Client selects which reservation he wants to delete 4. Client requests for reservation to be deleted 5. System checks reservation queue to see if any of the reservations can have a room 6. Client receives notifications that cancellation was successful
Post-conditions	Reservation(s) are removed from repository, Rooms and Users lists and database.

Alternate Flow	None significant

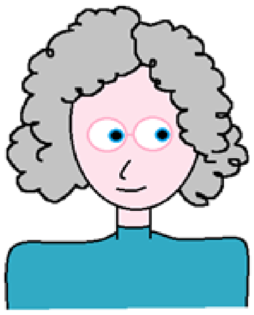
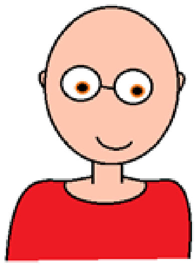
Use case name	Admin reserves a room
Use case description	Admin needs to reserve a room. Admins may need specific rooms for classes and such.
Actor	Admin (Reception, administration, e.t.c)
Pre-conditions	Available rooms must be displayed in the list box
Basic Flow (Success Scenario)	1. Admin enters the reservation management system 2. Admin requests to reserve a room 3. Admin selects the date and time for reservation 4. System updates the list of available rooms 5. Admin types username for the reservation 6. Admin selects room for reservation 7. Admin requests reservation 8. System assigns a room 9. Receives confirmation notification that room is reserved
Post-conditions	Reservation has been created
Alternate Flow	None significant

Use case name	Admin deletes a reservation
Use case description	An Admin needs to delete a reservation

Actor	Admin (Reception, administration, etc.)
Pre-conditions	Reservation exists
Basic Flow (Success Scenario)	1. Admin enters reservation management system 2. OPT: Inputs criteria (from date to date, username) to filter list 3. Admin selects which reservation he wants to delete 4. Admin requests deleting the reservation 5. Receives confirmation notification that room is deleted
Post-conditions	Reservation has been deleted
Alternate Flow	None significant

Use case name	Notification System
Use case description	System is sending a reminder 15 minutes before the client's reservation is beginning.
Actor	System
Pre-conditions	System must be running
Basic Flow (Success Scenario)	1. System gets upcoming reservations 2. System sends reminder for upcoming reservations 3. System checks again in one minute
Post-conditions	Reminder notification was sent
Alternate Flow	AF1: No upcoming reservations 1.System checks again in one minute

Personas

 Moira	Moira is 58 years old and works for Erhvervsakademiet Lillebælt. She is good with computers. She likes to help people, and she works in information at the Odense campus. Moira has a degree in economics.	Moira, as admin of the system, can book any room in the school. Her job is also to cancel unused reservations and book classrooms for lessons and lectures.
 Leo	47-year-old Leo is a teacher in Lillebaelt Academy. He likes news and politics. He has a goldfish named Eko. 10 years ago, he had a nose operation but he doesn't like to talk about it.	As a teacher, he can reserve teacher rooms, such as meeting rooms, project rooms and others.

Scenarios

Admin books a room:

Moria has been tasked with booking a meeting room for some teachers.

She enters the booking system, types in when the booking needs to be, and selects which room the booking should be valid for.

She hits the button to book the room, and a message appears that confirms the booking.

Admin cancel a reservation:

Moria is setting at the front information desk when a student approaches.
Despite the new booking system having launched, someone has booked a project room and accepted the booking by the room, yet has still left it empty.

Moria enters the admin side of the reservation system. She finds the room which is empty and marks it. She then clicks the "Delete Reservation" button.

Moria informs the student that the standing reservation has been cancelled, and asks the student to perform the new reservation themselves.

Canceling own reservation:

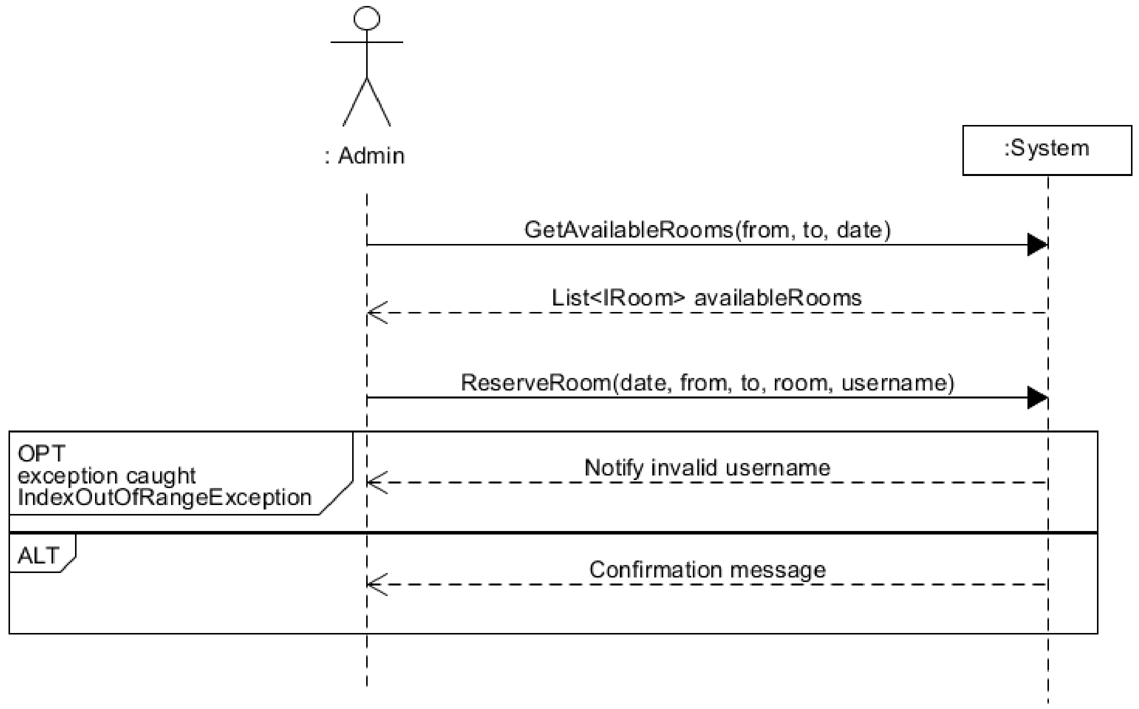
Leo was arranging a meeting between some teachers, but something has happened and he no longer needs his booked meeting room.

Leo opens up the booking system and enters the section to list own reservations. He marks the reservation for the room he no longer needs, and clicks "Delete Reservation".

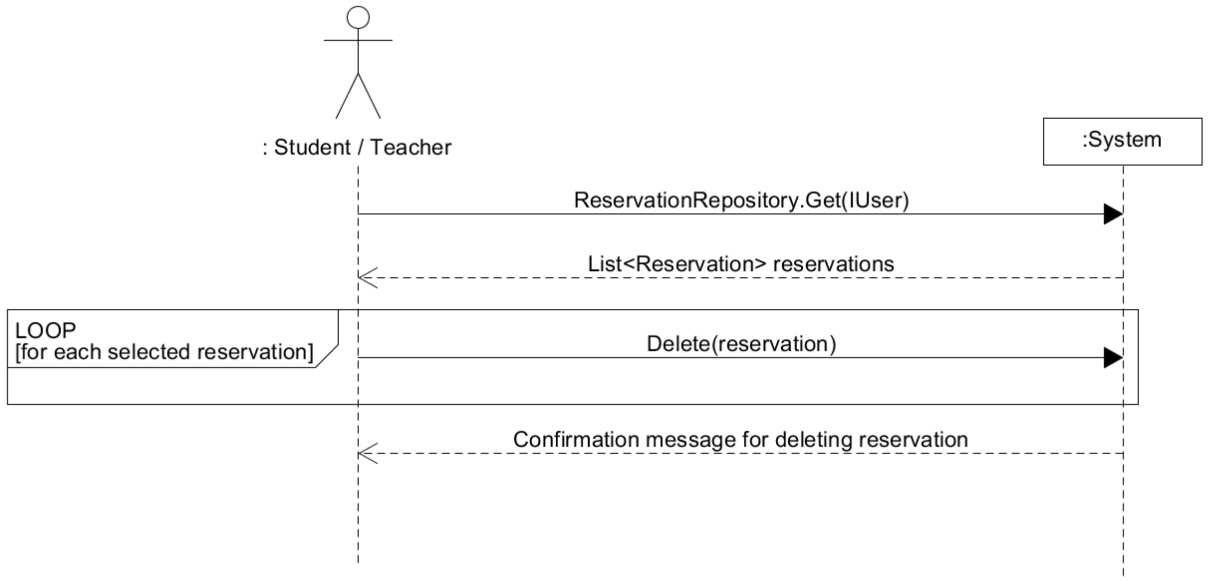
A message ensures Leo that his reservation has been cancelled.

System Sequence Diagrams

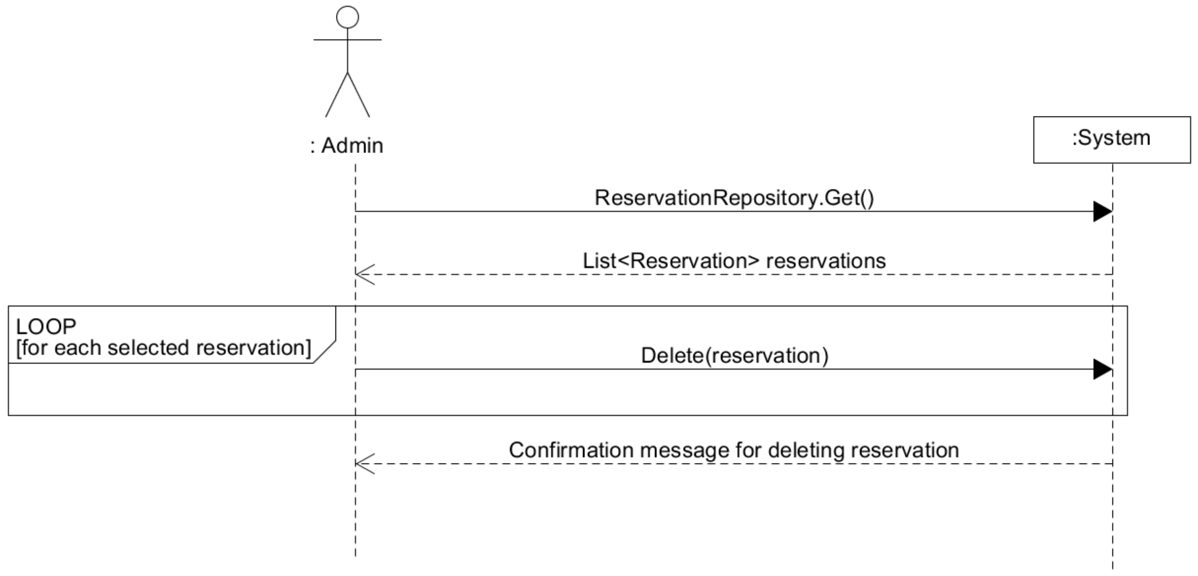
Reserve room by Admin



Delete reservation by Client


: Student / Teacher
:System
ReservationRepository.Get(IUser)
List<Reservation> reservations
LOOP
[for each selected reservation]
Delete(reservation)
Confirmation message for deleting reservation

Delete reservation by Admin


: Admin
:System
ReservationRepository.Get()
List<Reservation> reservations
LOOP
[for each selected reservation]
Delete(reservation)
Confirmation message for deleting reservation

Group Contract

General

This is a team project, not an I project. If in doubt, ask. No stupid ideas. Be honest with each other.
Grammar
Project spelling/grammar is to be American based.
No use of contractions such as: isn’t, wasn’t, can’t and others.

Conflict Handling

Should issues arise professionally in the group, e.g. disagreements over a procedure, a majority win vote should decide it.
Should someone break the contract, a penalty jar is set up. The penalty will be 5 kr.
Matthew will keep the money until the group decides to use it.
A log shall be kept over penalties, in the GitHub folder. The group will vote on whether the penalty will be applied or not, a majority vote is needed to decide.
In case of disagreement, a die is rolled to decide what we do.

Attendance and Meetings.

You are expected to show up every day. If unable to attend, inform the group via the common text chat.

If unable to attend, you are still expected to attempt to be in the daily scrum meeting via skype or another similar platform.
We will use a Slack group and events to facilitate organization. If you cannot attend an event, please be honest about why not. Always respond to events ASAP. DO NOT RESPOND MAYBE.
We meet at 10:00 am, Monday-Friday; 01:30 pm Sunday[1].
Starting with a stand-up meeting. If late to the stand-up meeting, penalty applies.

Scrum Usage
The scrum master will change every sprint, as with working pairs.
At the end of each sprint we will do a retrospective to reflect on our work
We will begin each day with a sit-down meeting
Sprints will be 1 week in length and will begin and end on wednesday.

Developer Tools
Organization of time, place, and specials meetings: Facebook and Events
Organization of project files, code, and version control: Git w/ GitHub and Google Drive.
22 Organization of tasks, project backlog etc: Meister Task

Naming Conventions
Methods must describe what they do
Global variables are named with an underscore at the beginning, unless they are constant.
Constant variables are all upper case.
Variables always start with lowercase and methods always start with uppercase, and then camelcase is used after.
Parameters of methods are all lower case.
Variables should describe what they are.
Properties named like methods.

Architecture
User interface should be MVVM.
Main class library should be called Core.
Unit and integration tests will be used, TDD.
File and Database management will be done by Data Access Layer.
The code should follow the SOLID principles.

Quality Assurance
Completing tasks in the QA section of the scrum board is the top priority
You cannot check tasks that you created (all grammar is checked by Matthew)
You must follow the given criteria when checking tasks
The scrum master will assign group members to check tasks

How you are expected to use Git / GitHub
Only change one method per commit.
Expected to make decent comments.

Sync whenever you leave your computer and whenever you feel it is necessary.
Each task should be dealt with in separate branches
After QA, a pull request to master should be made, then the scrum master reviews, merges, and closes the branch

Logging

There should be an entry in the log every day that work is done.
It is the common responsibility to ensure the log is written.
During meetings, a secretary is assigned to write a summary of the meetings.
The scrum master will choose the secretary

Voting

All votes are equal
You cannot abstain

Amending the contract

To amend the contract, a 75% vote is needed to approve of the amendment.

Team Pac-Man members:

Hedviga Arta Geriņa; hedv0149@edu.eal.dk
Matthew Peterson; matt2694@edu.eal.dk
Roxana Ion; roxa0188@edu.eal.dk
Jonas Laursen; jona8690@edu.eal.dk

[1] Jonas except from meeting Sunday due to work

Business Case

Executive Summary

A better way to have rooms available for reservation is needed. We will create a better reservation system at little to no cost to Erhvervsakademiet Lillebælt. There is little risk, but because we are second semester students we do not have the knowledge or expertise most professionals do, but the cost benefits are worth the risk.

Financial Appraisal

The project will have little to no cost for the school because we are completing this project as a school project and thus will not be requiring monetary compensation for the work we do on this specific project. If the school would like to have our group do extra work for them beyond what is in the scope of this project, we may seek monetary compensation. The school may need to pay to have its employees trained to use our solution and to implement the solution.

The purpose of the project is not focused on monetary gain because public schools are not trying to make a profit in the same way that a business is. The school gets money from the government, not the students. The school does, however, need students to continue operating.

Our system will help to make the school a more attractive option for incoming students because it will give it a better reputation as a place to work. Also, the students will be more able to work using the project rooms and this could have a positive effect, albeit small, on the performance of students.

Problem Description

Project rooms are often reserved multiple weeks ahead (especially during the project weeks), yet rooms are often left empty. People reserve rooms and do not use them, or only use them for part of the time.

This is quite an annoyance to students, as when they request a room from the reception, their request cannot be fulfilled, and the students then must walk around the school, searching for rooms not being used.

Students have also experienced that while attempting to reserve a room via email, another group of students went to the reception and were assigned the room, even though the email had already been sent and received. Hence, the students who had sent the email had to give up the room to those students who went to the reception.

The problem occurs on an almost daily basis, and is most prominent during project weeks and close to end of semesters, due to exam work.

Business Objective

Our goal is to have a better way to reserve rooms in the school. We want it to be very simple for anyone to reserve a room and to get the room they need. We also want to solve the problem of rooms being reserved by people who are not using them.
We will need access to various school resources such as: users, system for displaying information on the tablets and information about rooms. This will allow students to find the rooms they need to work. This is good for the school because it will improve their reputation and bring more students.

Benefits and Limitations

The benefits of this project include: happier students, more efficient students, remain competitive and improve customer service (with staff and students). We will not be focusing on working with the meeting rooms or administration rooms, our focus will be on the project rooms. Since we are second semester students we may not have the knowledge to complete every part that would be required to make a completed project. Also, we may not have time to complete everything.

Option Identification and Selection

<u>Problem: Too many unused but reserved rooms</u>
Solution 1: We will make a new reservation system for the school to use. It will have CRUD operations for reservations, users and rooms. It will also have students get notifications about their reservation and have them confirm the reservation when they get to the room.
Solution 2: Have a staff member walk around to all the different rooms to see if the rooms are in use, and if a room is not in use but is reserved, then the staff member will delete that reservation.
Solution 3: We do nothing.

Solution 1 is a relatively cheap solution to the problem. Also, it will cancel the reservations much faster than solution 2 because it does not require a person to walk around to the room. Solution 2 will require an employee to be paid to walk around the school, and this is just a waste of time and money when an IT solution would be much cheaper in the long run. Solution 3 is the cheapest, but it doesn’t solve the problem at all. Therefore, based on this, solution 1 is the best solution for the problem.

Main Solution: We will make a new reservation system for the school to use. It will have CRUD operations for reservations, users and rooms. It will also have students get notifications about their reservation and have them confirm the reservation when they get to the room.

Scope, Impact and Interdependencies

The project will affect all the users that reserve rooms and the reservation system for the school. It has interdependencies with the database of users and rooms. It is possible that it will have other dependencies beyond how we create it.

Outline Plan

We will be working in one week sprints using the scrum methodology. We will continue doing this until the end of the project on June 2nd.

Risk Assessment

Incident	Impact	Vulnerability	Threat

Unauthorized access to admin user	High	High	This would be very bad because it would basically make the system useless. It is highly vulnerable as well because there are many computer science students in the school who may be able to do this.
Memory leak, exposing reservations	Low	High	This would not be very bad because it will not have any effect on any reservations. It is highly vulnerable though because this is a school with many computer science students who may be able to do this.
Natural Disasters	High	Low	The chance of any sort of natural disaster happening in Denmark and damaging our system is extremely low, but if it did happen it would be very bad.

An admin user abuses their power to damage the system	Medium	Medium/Low	Since the admin users have a lot of power they could easily do it, but the chance that an admin would want to do this is low. It could cause an annoyance to many users, but it would not have dire consequences.
An admin accidentally damages the system	Medium	Low	Since the admin users have a lot of power they could easily do it, but the chance that they could accidentally do it is very low. It could cause an annoyance to many users, but it would not have dire consequences.
Power to school cut off	Medium	Low	This would not have any lasting effects, but the system would not work while the power is out. The chance of this happening is low.

Project Approach

We will work using the scrum methodology. Our sprints are one week in length and will begin and end on Wednesday. We will begin each day with an informal sit-down meeting where we discuss what we need help with and what we have done and will do. Also, at the end of each sprint the group will have a retrospective to discuss how things worked for that sprint and if there is any way we could improve the sprint for next time. We will rotate the scrum master for each sprint so that each group member can get experience with that position. Since we are working on this for our school project we will not be having anyone else doing any of the work for us. We will be getting guidance and assistance from teachers and other students.

Project Governance

Scrum Master:
Rotating between group members

Group Members:
Matthew Anthony Peterson
Roxana Ion
Hedviga Arta Gerina
Jonas Amstrup Laursen

Stakeholders:
Jens Mejer Pedersen – Rector
Asger Rabølle Nielsen – Pro-rector/Director of Education
Mette Greisen – Director of Education
Leif Bojesen – Director of Education
Ole Hjort Willatzen – Director of Finance and Resources
Torben Lindegaard Hansen – Director of Research and Development
Rikke Kjærsgaard Budden – Head of Study Administration
Marie Falk Nyboe – Head of Study Guidance and Career Centre
Peter Topp Jensen – IT Manager
All students and staff at EAL
All future students and staff
Future employers of students

Progress Reporting

KPI	Less Empty Project Rooms
Why measure?	This will show if the system is fixing the problem we set out to solve by doing this project.

How measure?	Count the amount of project rooms that are reserved and being used before we implement our system and the amount of project rooms that are reserved and being used after we implement our system. Then, take that value and divide it by the total number of project rooms available for reservation.
Who is responsible for the measurement?	Group 1
Expected date for measuring	During the last week of use for the old system around 13:00, and again one month after the implementation of our system around 13:00.
Expected values measured	The percentage reserved and used project rooms should be higher after we implement our system
Measure	
Plan of action in case the measure is outside the range of the expected measure	Remake or eliminate the system.
Responsible for action	Group or IT at school.

Operation Contracts

Contract C02: Client ReserveRoom

Operation:
- ReserveRoom(date : string, from : string, to : string, peopleNr : string)

Cross References:
- SSD: SSD_Reserve
- UC: Client reserves room

Preconditions:
- No preconditions

Postconditions:

- A Reservation was created and stored in the repository,
- A Reservation was associated with a room and user

Contract C03: Delete

Operation:
- ReservationRepository.Delete(reservation : Reservation)

Cross Reference:
- SSD: SSD_Delete
- UC: Client cancel own reservation; Admin cancels a reservation

Preconditions:
- Reservation Exists

Postconditions:
- A Reservation was deleted

Database Triggers

Reservations Table

```sql
CREATE TRIGGER  trgInsertReservations ON Reservations
FOR INSERT AS
      declare @command  int;
      declare @table      NVarChar(max);
      declare @PK       NVarChar(max);

      declare @ID       NVarChar(max);
      declare @DateTo   NVarChar(max);
      declare @DateFrom   NVarChar(max);
      declare @Username   NVarChar(max);

      set @command =  0;
      set @table = 'Reservations';
      select @ID = i.ID from inserted i;
      select @DateTo = i.DateTo from inserted i;
      select @DateFrom = i.DateFrom from inserted i;
      select @Username = i.Username from inserted i;
      select @PK = @ID + ';' + @DateTo + ';' + @DateFrom + ';' +
@Username;

      INSERT INTO Change(Command, TableName, PrimaryKey) VALUES
      (@command, @table, @PK);

      PRINT 'Trigger: Insert Reservation'
GO

CREATE TRIGGER trgUpdateReservations ON Reservations
FOR UPDATE AS
      declare @command  int;
      declare @table      NVarChar(max);
      declare @PK       NVarChar(max);

      declare @ID       NVarChar(max);
      declare @DateTo   NVarChar(max);
      declare @DateFrom   NVarChar(max);
      declare @Username   NVarChar(max);

      set @command =  1;
      set @table = 'Reservations';
      select @ID = i.ID from inserted i;
      select @DateTo = i.DateTo from inserted i;
      select @DateFrom = i.DateFrom from inserted i;
      select @Username = i.Username from inserted i;
      select @PK = @ID + ';' + @DateTo + ';' + @DateFrom + ';' +
@Username;
```

```sql

        INSERT INTO Change(Command, TableName, PrimaryKey) VALUES
        (@command, @table, @PK);

        PRINT 'Trigger: Update Reservation'
GO

CREATE TRIGGER trgDeleteReservations on Reservations
FOR DELETE AS
        declare @command  int;
        declare @table      NVarChar(max);
        declare @PK        NVarChar(max);

        declare @ID        NVarChar(max);
        declare @DateTo    NVarChar(max);
        declare @DateFrom   NVarChar(max);
        declare @Username   NVarChar(max);

        set @command =  2;
        set @table = 'Reservations';
        select @ID = i.ID from deleted i;
        select @DateTo = i.DateTo from deleted i;
        select @DateFrom = i.DateFrom from deleted i;
        select @Username = i.Username from deleted i;
        select @PK = @ID + ';' + @DateTo + ';' + @DateFrom + ';' + @Username;

        INSERT INTO Change(Command, TableName, PrimaryKey) VALUES
        (@command, @table, @PK);

        PRINT 'Trigger: Delete Reservation'
GO
```

Rooms Table

```
CREATE TRIGGER  trgInsertRooms ON Rooms
FOR INSERT AS
      declare @command  int;
      declare @table      NVarChar(max);
      declare @PK       NVarChar(max);

      declare @Building   NVarChar(max);
      declare @FloorNr  NVarChar(max);
      declare @Nr       NVarChar(max);

      set @command =  0;
      set @table = 'Rooms';
      select @Building = i.Building from inserted i;
      select @FloorNr = i.FloorNr from inserted i;
      select @Nr = i.Nr from inserted i;
    select @PK = @Building +';' +@FloorNr + ';' + @Nr;

      INSERT INTO Change(Command, TableName, PrimaryKey) VALUES
      (@command, @table, @PK);

      PRINT 'Trigger: Insert Room'
GO

CREATE TRIGGER trgUpdateRooms ON Rooms
FOR UPDATE AS
      declare @command  int;
      declare @table      NVarChar(max);
      declare @PK       NVarChar(max);

      declare @Building   Char;
      declare @FloorNr  int;
      declare @Nr       int;

      set @command =  1;
      set @table = 'Rooms';
      select @Building = i.Building from inserted i;
      select @FloorNr = i.FloorNr from inserted i;
      select @Nr = i.Nr from inserted i;
      select @PK = @Building +';' +@FloorNr + ';' + @Nr;

      INSERT INTO Change(Command, TableName, PrimaryKey) VALUES
      (@command, @table, @PK);

      PRINT 'Trigger: Update Rooms'
GO

CREATE TRIGGER trgDeleteRooms on Rooms
FOR DELETE AS
      declare @command  int;
      declare @table      NVarChar(max);
      declare @PK       NVarChar(max);
```

```

        declare @Building    Char;
        declare @FloorNr  NVarChar(max);
        declare @Nr       NVarChar(max);

        set @command =  2;
        set @table = 'Rooms';
        select @Building = i.Building from deleted i;
        select @FloorNr = i.FloorNr from deleted i;
        select @Nr = i.Nr from deleted i;
        set @PK = @Building + ';' + @FloorNr + ';' + @Nr;

        INSERT INTO Change(Command, TableName, PrimaryKey) VALUES
        (@command, @table, @PK);

        PRINT 'Trigger: Delete ' + @table
GO
```

Users Table

```sql
CREATE TRIGGER  trgInsertUser ON Users
FOR INSERT AS
      declare @command  int;
      declare @table      NVarChar(max);
      declare @PK       NVarChar(max);

      set @command =  0;
      set @table = 'Users';
      select @PK = i.Username from inserted i;

      INSERT INTO Change(Command, TableName, PrimaryKey) VALUES
      (@command, @table, @PK);

      PRINT 'Trigger: Insert User'
GO

CREATE TRIGGER trgUpdateUser ON Users
FOR UPDATE AS
      declare @command  int;
      declare @table      NVarChar(max);
      declare @PK       NVarChar(max);

      set @command =  1;
      set @table = 'Users';
      select @PK = i.Username from inserted i;

      INSERT INTO Change(Command, TableName, PrimaryKey) VALUES
      (@command, @table, @PK);

      PRINT 'Trigger: Update User'
GO

CREATE TRIGGER trgDeleteUser on Users
FOR DELETE AS
      declare @command  int;
      declare @table      NVarChar(max);
      declare @PK       NVarChar(max);

      set @command =  2;
      set @table = 'Users';
      select @PK = i.Username from deleted i;

      INSERT INTO Change(Command, TableName, PrimaryKey) VALUES
      (@command, @table, @PK);

      PRINT 'Trigger: Delete User'
GO
```

Code

Core

ReservationRepository

```
using Core.Exceptions;
using Core.Interfaces;
using System;
using System.Collections.Generic;

namespace Core
{
        public class ReservationRepository
        {
        private DALFacade _dalFacade = new DALFacade();
        private List<Reservation> _reservationRepository = new List<Reservation>();
        private List<Reservation> _queue = new List<Reservation>();
        private  RoomRepository _roomRepo = RoomRepository.Instance;

                private static ReservationRepository _instance = new ReservationRepository();
        public static ReservationRepository Instance { get { return _instance; } }

                private ReservationRepository() { }

        public IRoom RequestReservation(DateTime from, DateTime to, int peoplenr, IUser user)
        {

                if (user.HasReservation(from.AddSeconds(1), to.AddSeconds(-1)))
                {
                throw new UserAlreadyHasRoomException();
                }

                List<IRoom> rooms = _roomRepo.GetPossible(user.PermissionLevel, peoplenr);
                List<IRoom> availableRooms = RemoveUnavailableRooms(rooms, from, to);

                if (availableRooms.Count == 0)
                {
                _queue.Add(new Reservation(user, null, peoplenr, from, to));
                throw new NoRoomsAvailableException();
                }
                else
                {
                Reservation reservation = new Reservation(user, availableRooms[0], peoplenr, from, to);
                this.Add(reservation);
                return availableRooms[0];
                }

        }

        public List<IRoom> GetAvailableRooms(DateTime from, DateTime to, IUser
user)
        {
                List<IRoom> rooms = _roomRepo.GetPossible(user.PermissionLevel);
                List<IRoom> availableRooms = RemoveUnavailableRooms(rooms, from,
to);
              return availableRooms;
        }

        private List<IRoom> RemoveUnavailableRooms(List<IRoom> rooms, DateTime
from, DateTime to)
        {
                List<IRoom> availableRooms = new List<IRoom>();

                foreach (IRoom room in rooms)
                {
                bool roomAvailable = room.IsAvailable(from, to);
                if (roomAvailable == true)
                {
                        availableRooms.Add(room);
                }
                }
                return availableRooms;
         }

        internal void DeleteFromQueue(Reservation res)
        {
                _queue.Remove(res);
        }

        internal List<Reservation> GetQueue()
        {
                return _queue;
        }

        internal void LoadFromDatabase(Reservation reservation)
        {
              _reservationRepository.Add(reservation);
              reservation.Room.AddReservation(reservation);
              reservation.User.AddReservation(reservation);
        }

        internal void DeleteFromRepository(Reservation reservation)
        {
              _reservationRepository.Remove(reservation);
              reservation.Room.DeleteReservation(reservation);
```

```
            reservation.User.DeleteReservation(reservation);
        }

        public void Clear()
        {
                foreach (Reservation reservation in _reservationRepository)
                {
                  _dalFacade.DeleteReservation(reservation);
                }
                _reservationRepository.Clear();
        }

        public void Add(Reservation reservation)
        {
              _reservationRepository.Add(reservation);
              reservation.Room.AddReservation(reservation);
              reservation.User.AddReservation(reservation);
                _dalFacade.PassReservationToDAL(reservation);
        }

        public void Delete(Reservation reservation)
        {
              _reservationRepository.Remove(reservation);
              reservation.Room.DeleteReservation(reservation);
                reservation.User.DeleteReservation(reservation);
              _dalFacade.DeleteReservation(reservation);

                CheckReservationQueue();
        }

        private void CheckReservationQueue()
        {
                // Check the queue, see if anyone fits the criterias...
                List<Reservation> newRegistrations = new List<Reservation>();
                foreach (Reservation res in _queue)
                {
                List<IRoom> rooms =
_roomRepo.GetPossible(res.User.PermissionLevel, res.PeopleNr);
                List<IRoom> availableRooms = RemoveUnavailableRooms(rooms,
res.From, res.To);

                if (availableRooms.Count > 0)
                {
                        res.Room = availableRooms[0];
                        this.Add(res);
                        newRegistrations.Add(res);
                }
                }

                // remove the new registrations from the queue, and send
notifications
```

```
            foreach (Reservation res in newRegistrations)
            {
            ReservationsObserver.Instance.Message = "Dear " + res.User.Username + "\nYou have recived a reservation in room: " + res.Room.ID + "\nSee your reservations for more info.";
            _queue.Remove(res);
            }
        }

        public void Add(IUser user, IRoom room, int peoplenr, DateTime datefrom, DateTime dateto)
        {
            Reservation reservation = new Reservation(user, room, peoplenr, datefrom, dateto);
            this.Add(reservation);
        }

        public List<Reservation> Get()
        {
           return _reservationRepository;
        }

        public List<Reservation> Get(IUser user)
        {
            List<Reservation> reservationsByUser = new List<Reservation>();

            foreach (Reservation reservation in _reservationRepository)
            {
            if (reservation.User.Equals(user))
            {
                   reservationsByUser.Add(reservation);
            }
            }
            return reservationsByUser;
        }

        public List<Reservation> Get(IRoom room)
        {
            List<Reservation> reservationsByRoom = new List<Reservation>();

            foreach (Reservation reservation in _reservationRepository)
            {
            if (reservation.Room.Equals(room))
              {
                   reservationsByRoom.Add(reservation);
            }
            }
            return reservationsByRoom;
        }

        public Reservation Get(Reservation checkreservation)
        {
            Reservation result = null;

          foreach (Reservation reservation in _reservationRepository)
            {
            if (reservation.Equals(checkreservation))
            {
                    result = reservation;
            }
            }
            return result;
        }

        public List<Reservation> Get(DateTime? from, DateTime? to, IUser user)
        {
            List<Reservation> reservations = new List<Reservation>();
            List<Reservation> allReservations = this.Get();

            if (from != null)
            {
            foreach (Reservation reservation in allReservations)
            {
                    if (from < reservation.From)
                    {
                        reservations.Add(reservation);
                    }
            }
            allReservations = reservations;
            reservations = new List<Reservation>();
            }

            if (to != null)
            {
            foreach (Reservation reservation in allReservations)
            {
                    if (to > reservation.To)
                    {
                        reservations.Add(reservation);
                    }
            }
            allReservations = reservations;
            reservations = new List<Reservation>();
            }

            if (user != null)
            {
            foreach (Reservation reservation in allReservations)
            {
                    if (user.Equals(reservation.User))
                {
```

```
                                reservations.Add(reservation);
                          }
                   }
                   allReservations = reservations;
                   reservations = new List<Reservation>();
                   }
                   return allReservations;
            }
            }
}


```

DAL

Database

```
using System.Data.SqlClient;
using System.Threading;

namespace DAL
{
        public abstract class Database
        {
        private static string _connInfo = DatabaseConn.ConnString;
        private static SqlConnection _conn;
        protected static object locked = new object();

        protected SqlConnection OpenConnection()
        {
                Monitor.Enter(locked);
                if (_conn == null)
                {
                _conn = new SqlConnection(_connInfo);
                }

                _conn.Open();
                return _conn;
        }

        protected void CloseConnection()
        {
                _conn.Close();
                Monitor.Exit(locked);
        }
        }
}
```